Oracle8 Server

概要

Vol.1

リリース 8.0

1998 年 2 月

部品番号 A56808-1

ORACLE®

Enabling the Information Age™

Oracle8 Server 概要 : Vol.1

部品番号 A56808-1

リリース 8.0

第 1 版 : 1998 年 2 月

原本名 : Oracle8 Concepts, Volume 1

原本部品番号 : A58384-01

Copyright© Oracle Corporation, 1997, 1998.

All rights reserved.
Printed in Japan.

原本著者 : Lefty Leverenz

制限付権利の説明

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

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

危険な用途への使用について
当社製品は、原子力、航空産業、大量輸送、又は医療の分野など、本質的に危険が伴うアプリケーションを用途として特に開発されておりません。当社製品を上述のようなアプリケーションに使用することについての安全確保は顧客各位の責任と費用によって行っていただきたく、万一かかる用途での使用によりクレームや損害が発生いたしましても、当社および開発元である米国 Oracle Corporation （その関連会社も含みます）は一切責任を負いかねます。

ORACLE は、 Oracle Corporation の登録商標です。

本文中の他社の商品名は、それぞれ各社の商標または登録商標です。

目　次

はじめに xxv

Vol.1

第 I 部　Oracle の概要

1　Oracle Server **の基礎知識**

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

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

第 II 部 データベースの構造

2 データ・ブロック、エクステント、セグメント

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

エクステントの数とサイズの決定 2-11
エクステントはどのように割り当てられるか 2-11
エクステントはいつ割当てを解除されるか 2-12
セグメント 2-14
データ・セグメント 2-14
索引セグメント 2-15
一時セグメント 2-15
ロールバック・セグメント 2-16

3 **表領域とデータ・ファイル**

表領域とデータ・ファイルの基礎知識 3-2
表領域 3-3
SYSTEM 表領域 3-4
データベースにより多くの領域を割り当てる 3-4
表領域をオンライン / オフラインにする 3-7
読込み専用表領域 3-9
一時表領域 3-10
データ・ファイル 3-11
データ・ファイルの内容 3-11
データ・ファイルのサイズ 3-12
オフライン・データ・ファイル 3-12

4 **データ・ディクショナリ**

データ・ディクショナリの基礎知識 4-2
データ・ディクショナリの構造 4-2
SYS **、データ・ディクショナリの所有者** 4-3
データ・ディクショナリが使用される方法 4-3
Oracle はどのようにデータ・ディクショナリを使うか 4-3
Oracle ユーザーはどのようにデータ・ディクショナリを利用できるか 4-5
動的パフォーマンス表 4-6

第 III 部　Oracle インスタンス

5　データベースとインスタンスの起動と停止

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

6　メモリー構造

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

7 プロセスの構造

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

第 IV 部 オブジェクト・リレーショナル DBMS

8 スキーマ・オブジェクト

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

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

9 **パーティション表とパーティション索引**

パーティション化の基礎知識 9-2
パーティション化とは何か 9-2
パーティション化の利点 9-4
大規模データベース（ VLDB ） 9-4

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

10 ビルトイン・データ型

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

ROWID データ型 10-11
MLSLABEL データ型 10-15
Oracle データ型についての情報のまとめ 10-15
ANSI データ型、 DB2 データ型、 SQL/DS データ型 10-17
データ変換 10-18

11 ユーザー定義データ型（オブジェクト・オプション）

基礎知識 11-2
複合データ・モデル 11-2
マルチメディア・データ型 11-3
ユーザー定義データ型 11-3
オブジェクト型 11-3
コレクション型 11-9
アプリケーション・インタフェース 11-11
SQL 11-11
PL/SQL 11-12
Pro*C/C++ 11-12
OCI 11-12
OTT 11-13

12 ユーザー定義データ型の使用方法

参照とネーム変換 12-2
表の別名 12-2
引数なしのメソッド・コール 12-3
ユーザー定義型の記憶領域 12-4
リーフ・レベル属性 12-4
行オブジェクト 12-4
列オブジェクト 12-4
参照（ REF ） 12-5
ネストした表 12-5
VARRAY 12-5
オブジェクト属性のプロパティ 12-5
NULL 12-6

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

13 **オブジェクト・ビュー**

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

Vol.2

第 V 部 データ・アクセス

14 SQL **と** PL/SQL

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

解析 14-7
SQL **の処理** 14-8
SQL 文の実行の概要 14-8
DML 文の処理 14-10
DDL 文の処理 14-13
トランザクションの制御 14-13
PL/SQL 14-14
PL/SQL が実行される方法 14-14
PL/SQL の言語要素 14-16
ストアド・プロシージャ 14-17
外部プロシージャ 14-18

15 **トランザクションの管理**

トランザクションの基礎知識 15-2
文の実行とトランザクションの制御 15-3
文レベルのロールバック 15-4
Oracle **とトランザクションの管理** 15-4
トランザクションをコミットする 15-5
トランザクションをロールバックする 15-6
セーブポイント 15-7
2 フェーズ・コミット・メカニズム 15-7
離散トランザクションの管理 15-8

16 **アドバンスト・キューイング**

メッセージ・キューイングの基礎知識 16-2
同期通信 16-2
非同期通信 16-2
Oracle Advanced Queuing 16-2
キューイング・エンティティ 16-3
アドバンスト・キューイングの機能 16-4

17 プロシージャとパッケージ

ストアド・プロシージャとパッケージの基礎知識 17-2
ストアド・プロシージャとファンクション 17-2
パッケージ 17-4
プロシージャとファンクション 17-6
プロシージャのガイドライン 17-7
プロシージャの利点 17-7
無名 PL/SQL ブロックとストアド・プロシージャ 17-8
スタンドアロン・プロシージャ 17-9
ストアド・プロシージャの依存性の追跡 17-9
外部プロシージャ 17-9
パッケージ 17-9
パッケージの利点 17-14
パッケージの依存性の追跡 17-15
Oracle **がプロシージャとパッケージを格納する方法** 17-15
プロシージャとパッケージのコンパイル 17-15
コンパイル済みコードのメモリーへの格納 17-15
プロシージャとパッケージのデータベースへの格納 17-15
Oracle **がプロシージャとパッケージを実行する方法** 17-16
ユーザー・アクセスの検査 17-16
プロシージャの有効性の検査 17-16
プロシージャの実行 17-17

18 データベース・トリガー

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

トリガーのタイプ 18-7
行トリガーと文トリガー 18-7
BEFORE トリガーと AFTER トリガー 18-8
トリガーの組合せ 18-8
INSTEAD OF トリガー 18-11
トリガーの実行 18-13
トリガーの実行モデルと整合性制約のチェック 18-14
トリガーのデータ・アクセス 18-15
トリガーの記憶領域 18-17
トリガーの実行 18-17
トリガーの依存性のメンテナンス 18-17

19 Oracle の依存性の管理

依存性の問題の基礎知識 19-2
スキーマ・オブジェクトの依存性の解決 19-4
ビューと PL/SQL プログラム・ユニットのコンパイル 19-5
依存性の管理と存在しないスキーマ・オブジェクト 19-7
共有 SQL の依存性管理 19-8
ローカルとリモートの依存性の管理 19-8
ローカル依存性の管理 19-9
リモート依存性の管理 19-9

20 オプティマイザ

最適化とは？ 20-2
実行計画 20-2
実行の順序 20-5
コストベース最適化とルールベース最適化 20-6
コストベースのアプローチ 20-6
オプティマイザの操作の概要 20-11
オプティマイザの操作 20-11
SQL 文のタイプ 20-11
式と条件の評価 20-12

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

スター型変換 20-72

第 VI 部 パラレル SQL とダイレクト・ロード・インサート

21 ダイレクト・ロード・インサート

ダイレクト・ロード・インサートの基礎知識 21-2
ダイレクト・ロード・インサートの利点 21-2
INSERT ... SELECT 文 21-3
ダイレクト・ロード・インサート文の種類 21-3
シリアル INSERT とパラレル INSERT 21-3
ロギング・モード 21-5
ダイレクト・ロード・インサートについてのその他の考慮事項 21-8
索引のメンテナンス 21-8
領域についての考慮事項 21-8
ロックについての考慮事項 21-9
ダイレクト・ロード・インサートの制限事項 21-9

22 パラレル実行

パラレル実行の概要 22-2
パラレル化できる操作 22-2
Oracle が操作をパラレル化する方法 22-3
パラレル実行のプロセス・アーキテクチャ 22-5
パラレル・サーバー・プール 22-6
SQL 文のパラレル化 22-8
並行度の設定 22-12
操作の並行度の決定 22-12
作業負荷のバランス調整 22-14
SQL 文のパラレル化ルール 22-15
パラレル DDL 22-21
パラレル化できる DDL 文 22-22
パラレルの CREATE TABLE ... AS SELECT 22-22
回復可能性とパラレル DDL 22-23
パラレル DDL の領域管理 22-24

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

第 VII 部 データの保護

23 データの同時実行性と一貫性

マルチユーザー環境におけるデータの同時実行性と一貫性 23-2
回避可能な現象とトランザクション分離レベル 23-2
ロックのメカニズム 23-3
データの同時実行性と一貫性の管理方法 23-3
マルチバージョン一貫性制御 23-4
文レベルの読込み一貫性 23-5
トランザクション・レベルの読込み一貫性 23-6
Oracle の分離レベル 23-6
分離レベルの設定 23-6
コミット読込みと直列可能分離の比較 23-9
分離レベルの選択 23-11
データをロックする方法 23-14
トランザクションとデータ同時実行性 23-14
デッドロック 23-16
ロックの種類 23-18
DML （データ）ロック 23-19
DDL ロック（ディクショナリ・ロック） 23-26

ラッチと内部ロック 23-28
明示的（手動）データ・ロック 23-29
Oracle のロック管理サービス 23-38

24 **データの整合性**

データ整合性の定義 24-2
データ整合性のタイプ 24-2
Oracle がデータの整合性を施行する方法 24-3
整合性制約の基礎知識 24-4
整合性制約の利点 24-5
整合性制約のパフォーマンス・コスト 24-6
整合性制約のタイプ 24-6
NOT NULL 整合性制約 24-6
UNIQUE キー整合性制約 24-7
PRIMARY KEY 整合性制約 24-10
FOREIGN KEY （参照）整合性制約 24-11
CHECK 整合性制約 24-16
制約チェックのメカニズム 24-17
デフォルト列値と整合性制約チェック 24-18
遅延制約チェック 24-19
制約の属性 24-19
SET CONSTRAINTS モード 24-19
一意制約と一意索引 24-20
制約の使用可能および使用禁止、施行 24-20

25 **データベース・アクセスの制御**

データベース・セキュリティ 25-2
スキーマおよびデータベース・ユーザー、セキュリティ・ドメイン 25-2
ユーザーの認証 25-3
オペレーティング・システムによる認証 25-3
ネットワークによる認証 25-4
Oracle データベースによる認証 25-4

データベース管理者の認証 25-6
ユーザー表領域の設定と割当て制限 25-7
デフォルト表領域 25-7
一時表領域 25-7
表領域のアクセスと割当て制限 25-7
ユーザー・グループ PUBLIC 25-8
ユーザー・リソースの制限とプロファイル 25-9
システム・リソースのタイプと制限 25-9
プロファイル 25-11
ライセンス 25-12
同時使用ライセンス 25-13
名前付きユーザー・ライセンス 25-14

26 権限とロール

権限 26-2
システム権限 26-2
スキーマ・オブジェクト権限 26-3
ロール 26-10
ロールの一般的な使用方法 26-11
ロールのメカニズム 26-11
ロールの付与と取消し 26-12
ロールの付与と取消しを実行できるユーザー 26-12
ロールの命名 26-13
ロールとユーザーのセキュリティ・ドメイン 26-13
名前付き PL/SQL ブロックとロール 26-13
データ定義言語の文とロール 26-13
事前定義済みのロール 26-14
オペレーティング・システムとロール 26-14
分散環境におけるロール 26-15

27 監査

監査の基礎知識 27-2

監査機能 27-2
監査のメカニズム 27-4
文監査 27-6
権限監査 27-7
スキーマ・オブジェクト監査 27-7
ビューとプロシージャのスキーマ・オブジェクト監査オプション 27-8
文および権限、スキーマ・オブジェクトの監査の対象を限定する 27-8
成功した文の実行と失敗した文の実行の監査 27-9
BY SESSION 監査と BY ACCESS 監査 27-9
ユーザー別の監査 27-11

28 データベースの回復

データベース回復の基礎知識 28-2
エラーと障害 28-2
データベースの回復で使われる構造 28-6
データベースのバックアップ 28-7
REDO ログ 28-7
ロールバック・セグメント 28-7
制御ファイル 28-8
ロールフォワードとロールバック 28-8
REDO ログとロールフォワード 28-8
ロールバック・セグメントとロールバック 28-9
Recovery Manager 28-10
リカバリ・カタログ 28-10
パラレル化 28-11
レポートの生成 28-12
回復のパラレル実行 28-12
パラレル回復を活用できる状況 28-13
回復プロセス 28-13
データベースのアーカイブ・モード 28-14
NOARCHIVELOG モード（メディア回復が使用不可能） 28-15
ARCHIVELOG モード（メディア回復が使用可能） 28-15

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

第 VIII 部 分散処理と分散データベース

29 分散処理

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

30 分散データベース

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

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

31 データベースのレプリケーション

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

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

第 IX 部 付録

A **オペレーティング・システム固有の情報**

索　引

はじめに

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

Oracle8 および Oracle8 Enterprise Edition

『Oracle8 Server概要』では、Oracle8およびOracle8 Enterprise Edition製品の機能を説明しています。 Oracle8 および Oracle8 Enterprise Edition には、同じ基本機能がいくつかあります。ただし、Enterprise Edition だけで使用可能な拡張機能もいくつかあり、これらの一部はオプションです。たとえば、オブジェクト機能を使うには、 Enterprise Edition とオブジェクト・オプションが必要です。

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

対象読者

このマニュアルは、データベース管理者、システム管理者、アプリケーションの開発者を対象にしています。

前提条件

読者には、リレーショナル・データベースの概念と、Oracle を実行しているオペレーティング・システムの環境についての知識が必要です。最初に、第 1 章「Oracle Server の基礎知識」を必ず読んでください。第 1 章では、このマニュアル全体で使われている概念と用語について包括的に説明されています。

インストレーションや移行について

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

データベース管理について

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

アプリケーションの設計について

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

このマニュアルの構成

このマニュアルは 2 巻構成で、次のような部に分かれています。

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

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

Vol.1

第 I 部 : Oracle の概要

第 1 章 : Oracle Server の基礎知識

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

第 II 部 : データベースの構造

第 2 章 : データ・ブロック、エクステント、セグメント

Oracle データベース内の各種オブジェクトにデータがどのように格納され、記憶領域がどのように割り当てられるかについて説明します。この章で取り上げる領域管理に関する背景知識は、次の章と第 8 章「スキーマ・オブジェクト」の内容を補足するものです。

第 3 章 : 表領域とデータ・ファイル

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

第 4 章 : データ・ディクショナリ

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

第 III 部 : Oracle インスタンス

第 5 章 : データベースとインスタンスの起動と停止

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

第 6 章 : メモリー構造

Oracle データベース・システムで使うメモリー構造について説明します。

第 7 章 : プロセスの構造

Oracle インスタンスのプロセス構造と、 Oracle で利用できる各種のプロセス構成について説明します。

第 IV 部: オブジェクト・リレーショナル DBMS

第 8 章 : スキーマ・オブジェクト

表、ビュー、名前付き順序、シノニムなど、特定のユーザーのドメイン（スキーマ）内に作成できるデータベース・オブジェクトについて説明します。さらに、データを効率的に検索するための索引やクラスタなどのオプション構造についても説明します。

第 9 章 : パーティション表とパーティション索引

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

第 10 章 : ビルトイン・データ型

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

第 11 章 : ユーザー定義データ型（オブジェクト・オプション）

Oracle オブジェクト・リレーショナル・データベース管理システム（ ORDBMS ）によって提供されるオブジェクト拡張機能について概説します。

第 12 章 : ユーザー定義データ型の使用方法

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

第 13 章 : オブジェクト・ビュー

Oracle ORDBMS によってビューに提供される拡張機能について説明します。

Vol.2

第 V 部: データ・アクセス

第 14 章 : SQL と PL/SQL

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

第 15 章 : トランザクションの管理

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

第 16 章 : アドバンスト・キューイング

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

第 17 章 : プロシージャとパッケージ

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

第 18 章 : データベース・トリガー

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

第 19 章 : Oracle の依存性の管理

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

第 20 章 : オプティマイザ

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

第 VI 部 : パラレル SQL とダイレクト・ロード・インサート

第 21 章 : ダイレクト・ロード・インサート

シリアルまたはパラレルに実行できるダイレクト・ロード・インサート・パスについて説明します。

第 22 章 : パラレル実行

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

第 VII 部 : データの保護

第 23 章 : データの同時実行性と一貫性

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

第 24 章 : データの整合性

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

第 25 章 : データベース・アクセスの制御

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

第 26 章 : 権限とロール

システム・レベルとオブジェクト・レベルでのセキュリティについて説明します。

第 27 章 : 監査

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

第 28 章 : データベースの回復

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

第 VIII 部 : 分散処理と分散データベース

第 29 章 : 分散処理

Oracle Server を実行できる分散処理環境について説明します。

第 30 章 : 分散データベース

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

第 31 章 : データベースのレプリケーション

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

第 IX 部 : 付録

付録 A: オペレーティング・システム固有の情報

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

このマニュアルの使用方法

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

このマニュアルの各部は、前述の対象読者の中でも、さらに特定の読者を対象にしています。たとえば、第 1 章を読んだ後、主にセキュリティの管理について関心のある管理者は、第 VII 部「データの保護」の中の、特に第25章「データベース・アクセスの制御」、第26章「権限とロール」、第 27 章「監査」の部分をよく読む必要があります。

このマニュアルで使用する表記規則

このマニュアルでは、情報の種類に応じていくつかの表記を使います。

マニュアルの本文

このマニュアルの本文では、次のような表記規則が使われています。

英大文字

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

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

かぎかっこ

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

コード例

SQL および Oracle Enterprise Manager の行モード（ Server Manager ）、 SQL*Plus のコマンドや文は、モノスペース・フォントで記載します。たとえば、次のとおりです。

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

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

コード例の中の英大文字

例文の中の大文字の語は、 Oracle SQL のキーワードを示します。ただし、実際に SQL 文を発行するときには、キーワードの大 / 小文字は区別されません。

コード例の中の英小文字

例文の中の小文字の語は、単なる例として使われている語を示します。たとえば、小文字の語は、表、列、またはファイルの名前を示します。

第 I 部

Oracle の概要

第I部では、Oracle Serverの概念と用語について概説します。次の1つの章が含まれています。

- 第 1 章「 Oracle Server の基礎知識」

このマニュアルの残りの部分では、第1章に要約されている概念をもっと詳しく説明します。

1

Oracle Server の基礎知識

I am Sir Oracle,

And when I ope my lips, let no dog bark!

Shakespeare: *The Merchant of Venice*

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

- データベースと情報管理
- データベースの構造と領域管理
- メモリー構造とプロセス
- データの同時実行性と一貫性
- 分散処理と分散データベース
- 起動操作と停止操作
- データベース・セキュリティ
- データベースのバックアップとリカバリ
- データベース管理のオブジェクト・リレーショナル・モデル
- データ・アクセス

注意: この章には、 Oracle8 と Oracle8 Enterprise Edition の両方に関連する情報が記載されています。この章に記載されている機能とオプションの中には、Oracle8 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。Oracle8とOracle8 Enterprise Editionの違いについては、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

データベースと情報管理

データベース・サーバーは、情報管理の問題を解決する鍵となります。一般にサーバーでは、多数のユーザーが同時に同じデータをアクセスできるように、マルチユーザー環境において大量のデータを確実に管理できなければなりません。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なければならず、 権限のないアクセスに対して保護機能を備えながら、障害の回復についても効率の良い解決方法を提供しなければなりません。

Oracle Server は、次に示す機能について、効率の良い効果的な解決方法を提供します。

クライアント/サーバー（分散処理）環境	コンピュータ・システムまたはネットワークを最大限に活用するために、 Oracle では処理をデータベース・サーバーとクライアント・アプリケーション・プログラムに分割して実行します。データベース管理システムが稼働しているコンピュータはデータベース・サーバーのすべての役割を果たし、データベース・アプリケーションが稼働しているワークステーションは主にデータの解釈と表示を行います。
大規模データベースと領域管理	Oracle はサイズが数千ギガバイト（テラバイト）に及ぶような大規模データベースをサポートします。高価なハードウェア装置を効率良く利用するために、領域の利用形態を完全に管理します。
数多くの同時実行データベース・ユーザー	Oracle は、同じデータを操作する複数のデータベース・アプリケーションを数多くのユーザーが同時に実行する機能をサポートします。 Oracle はデータの競合を最小限に抑え、データの同時実行性を保証します。
高いトランザクション処理パフォーマンス	Oracle は、システム全体の高いパフォーマンスを保ちつつ、上記の各機能を実行します。データベース・ユーザーにとって、処理パフォーマンスの低さが問題になることはありません。
高い可用性	サイトによっては、 Oracle が、データベースの処理能力を低下させる時間帯を設けることなく、 1 日 24 時間稼働することもあります。データベースのバックアップやコンピュータ・システムの部分的な障害など、日常的なシステムの運用によってデータベースの使用を中断させることはありません。
制御可能な可用性	Oracle は、データベース・レベルと下位データベース・レベルで、データベースの可用性を選択的に制御できます。たとえば、他のアプリケーションに影響を及ぼすことなく、管理者が、特定のアプリケーションを使用禁止にして、そのアプリケーションのデータを再ロードすることができます。

オープンかつ業界標準	Oracle は、データ・アクセス言語およびオペレーティング・システム、ユーザー・インタフェース、ネットワーク通信プロトコルについて、業界で受け入れられている規格に準拠しています。 Oracle は、顧客の投資を守る「オープン」システムです。 Oracle Server のリリース 8.0 は、米国国家標準技術局によってANSI/ISO SQL92（構造化問合せ言語）規格のエントリ・レベルに100%準拠していることが確認されています。 Oracle は米国連邦政府の FIPS127-2 規格の要件を完全に満たし、規格に準拠していない SQL の使用を強調表示する「フラガー」を組み込んでいます。 Oracle は、システム管理の規格としてシンプル・ネットワーク管理プロトコル（ SNMP ）規格もサポートしています。このプロトコルを使うと、管理者は 1 つの管理インタフェースで異機種システムを管理できます。
扱いやすいセキュリティ機能	許可されていないデータベース・アクセスとデータベース使用からデータベースを保護するために、 Oracle では、データ・アクセスの制限と監視を可能にするフェイルセーフ・セキュリティ機能が提供されます。データ・アクセスの設計がかなり複雑な場合でも、これらの機能によって管理は容易になります。
データベースに施行される整合性	Oracle は、データの整合性、すなわち許容可能なデータについての規格を示す「ビジネス・ルール」を施行します。その結果、数多くのデータベース・アプリケーションで、チェック機能をコーディングしたり管理したりするコストを節約できます。
分散システム	ネットワーク分散環境では、物理的には複数のコンピュータ上にあるデータが Oracle によって結合されて 1 つの論理データベースとなり、すべてのネットワーク・ユーザーがその論理データベースにアクセスできます。分散システムは、非分散型のシステムと同水準のユーザーの透過性とデータの一貫性を維持しているだけでなく、ローカル・データベース管理システムの持つ利点も備えています。 Oracle には、 Oracle 以外のデータベースにあるデータへの透過的なアクセスを実現する異機種データベース・オプションも用意されています。

移植性	Oracle ソフトウェアは、さまざまなオペレーティング・システムのもとで稼働するように移植されています。 Oracle 向けに開発されたアプリケーションは、わずかな修正を加えるだけで、あるいはまったく修正することなく、任意のオペレーティング・システムに移植することができます。
互換性	Oracle ソフトウェアは、ほとんどの業界標準オペレーティング・システムを含む業界規格と互換性があります。したがって、 Oracle 向けに開発されたアプリケーションは、わずかな修正を加えるだけで、あるいはまったく修正することなく、事実上どのシステムでも利用できます。
接続性	Oracle ソフトウェアでは、いろいろなタイプのコンピュータとオペレーティング・システムを接続して、ネットワーク全体で情報を共有できます。
レプリケート環境	Oracle ソフトウェアでは、表のグループおよびそれらの表をサポートするオブジェクトを、複数のサイトにレプリケートできます。 Oracle では、これらのサイトのデータの変更について、データ・レベルのレプリケーションとスキーマ・レベルのレプリケーションの両方をサポートします。 Oracle の柔軟なレプリケーション・テクノロジにより、基本的なプライマリ・サイトのレプリケーションだけでなく、高度な動的モデルおよび共用オーナーシップ・モデルもサポートされます。

この後の項では、 Oracle アーキテクチャの概要について幅広く説明します。アーキテクチャ全体を構成している各部分ごとに説明していきます。

Oracle Server

Oracle Serverは、情報管理におけるオープンかつ広範囲で統合的なアプローチを実現する、オブジェクト・リレーショナル・データベース管理システムです。Oracle Serverは、OracleデータベースとOracle Serverインスタンスで構成されます。この後、データベースとインスタンスの関係について説明します。

構造化問合せ言語（SQL）

SQL は、データベースを定義して操作するためのプログラミング言語です。 SQL データベースはリレーショナル・データベースです。これは、データが一連の単純なリレーションに基づいて格納されるということです。 1 つのデータベースは、 1 つ以上の表を持ちます。さらに、それぞれの表には列と行があります。たとえば、従業員データベースにある表には、従業員番号という列があり、各行のその列には従業員番号が格納されます。

表の中のデータは、 SQL コマンドを使って定義したり、操作することができます。データをセットアップするには、データ定義言語（DDL）コマンドを使います。DDLコマンドには、データベースや表を作成および変更するコマンドが含まれます。

また、表の中のデータの更新、削除、取出しには、データ操作コマンド（DML）を使います。DML コマンドには、データを変更したりフェッチしたりするコマンドが含まれます。最もよく使われるSQLコマンドは、SELECTコマンドです。このコマンドは、データベースからデータを取り出すときに使います。

SQL コマンドの他に、 Oracle ServerにはPL/SQL と呼ばれるプロシージャ型言語も用意されています。 PL/SQL によって、プログラマは SQL 文のプログラムを書くことができます。 PL/SQL を使うと、 SQL プログラムの流れを制御したり、変数を使ったりできるほか、エラー処理プロシージャを書くこともできます。

データベース構造

Oracle データベースには、論理構造と物理構造の両方が存在します。物理的なサーバー構造と論理的なサーバー構造が分離されているため、論理的な記憶構造へのアクセスに影響を与えずに、データの物理的な記憶領域を管理できます。

物理データベース構造

Oracle データベースの物理構造は、データベースを構成するオペレーティング・システム・ファイルによって決定されます。各 Oracle データベースは、 3 つのタイプのファイル、つまり、 1 つ以上のデータ・ファイル、 2 つ以上の REDO ログ・ファイル、および 1 つ以上の制御ファイルから構成されます。 Oracle データベースのファイルが、データベース情報のための実際の物理的な記憶領域を提供します。

論理データベース構造

Oracle データベースの論理構造は、次の 2 つによって決定されます。

- 1 つ以上の表領域

  「表領域」とは、この章で後述する論理記憶領域のことです。

- データベースのスキーマ・オブジェクト。

  「スキーマ」とは、オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタ、データベース・リンクなどの構造体が含まれます。

表領域、セグメント、エクステントなどの論理構造によって、データベースの物理領域がどのように使われるかが決まります。スキーマ・オブジェクトと、それらのオブジェクト間の関連が、データベースのリレーショナル設計を形成します。

データベース構造の詳細は、1-8ページの「データベースの構造と領域管理」を参照してください。

Oracle インスタンス

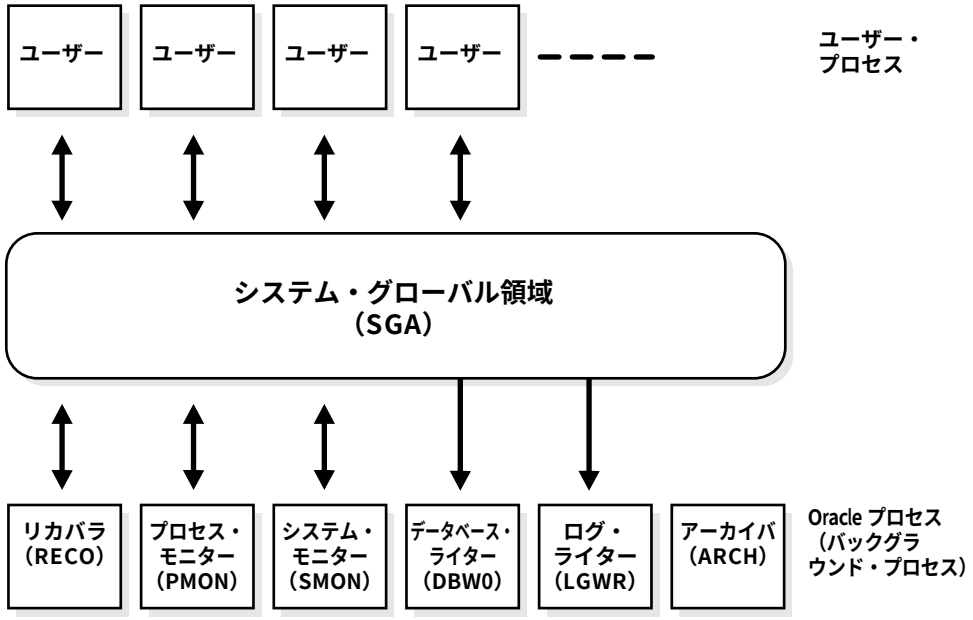
データベースを起動するたびに、システム・グローバル領域（SGA）が割り当てられ、Oracle バックグラウンド・プロセスが起動されます。システム・グローバル領域とは、各データベース・ユーザーが共有するデータベース情報のために使われるメモリー領域のことです。バックグラウンド・プロセスとメモリー・バッファの組合せのことを、「Oracle インスタンス」と呼びます。

Oracle インスタンスには、ユーザー・プロセスと Oracle プロセスという、2 つのタイプのプロセスがあります。

- 「ユーザー・プロセス」は、アプリケーション・プログラム（Oracle Forms アプリケーションなど）や Oracle Tool （Oracle Enterprise Manager など）のコードを実行します。
- 「Oracle プロセス」には、ユーザー・プロセスのための処理を実行するサーバー・プロセスと、Oracle Serverのためのメンテナンス処理を実行するバックグラウンド・プロセスがあります。

図 1-1 に、マルチ・プロセスの Oracle インスタンスを示します。

図 1-1 Oracle **インスタンス**



通信ソフトウェアと Net8

ユーザー・プロセスとサーバー・プロセスがネットワーク内の異なるコンピュータ上にある場合や、ユーザー・プロセスがディスパッチャ・プロセスを介して共有サーバー・プロセスに接続している場合、ユーザー・プロセスとサーバー・プロセスは Net8 を使って通信します。「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、マルチスレッド・サーバー構成を使っている場合だけです。「 Net8 」は、標準通信プロトコルへの Oracle インタフェースであり、コンピュータ間の正常なデータ転送を実現します。 1-26 ページの「 Oracle と Net8 」を参照してください。

Oracle Parallel Server による複数インスタンス・システム

注意: Oracle Parallel Serverオプションは、 Oracle8 Enterprise Editionを購入した場合にだけ使用可能です。Oracle8 Enterprise Editionで使用可能な機能とオプションの詳細は、『Oracle8 と Oracle8 Enterprise Editionの解説』を参照してください。

ハードウェア・アーキテクチャによっては（たとえば、共有ディスク・システム）、複数のコンピュータがデータ、ソフトウェア、または周辺装置へのアクセスを共有することができます。 Parallel Server オプション付きの Oracle は、 1 つの物理データベースを「共有する」複数のデータベース・インスタンスを実行することによって、このようなアーキテクチャを活用できます。適切なアプリケーションでは、Oracle Parallel Serverによって、複数のマシン上のユーザーが、優れたパフォーマンスで 1 つのデータベースにアクセスできます。

追加情報: Oracle Parallel Serverの詳細は、『Oracle8 Parallel Server概要および管理』を参照してください。

Oracle データベース

「 Oracle データベース」とは、 1 つの単位として扱われるデータの集合です。データベースの一般的な目的は、関連する情報を格納したり、取り出したりすることです。

データベースには、「論理構造」と「物理構造」があります。Oracle データベースの論理構造および物理構造の概要については、後述の「データベースの構造と領域管理」を参照してください。

データベースのオープンとクローズ

Oracle データベースは、「オープン」（アクセス可能）または「クローズ」（アクセス不能）のどちらかの状態です。通常、データベースはオープンで、使用可能な状態になっています。しかし、データベースのデータをユーザーから切り離す必要があるような特定の管理機能を実行するために、データベースをクローズする場合もあります。

データベースの構造と領域管理

この項では、データベースを構成する物理構造と論理構造を含む、Oracleデータベースのアーキテクチャについて説明します。制御可能なデータの可用性、論理データ構造と物理データ構造の分離、ディスク領域管理の細かい制御についての Oracle の解決方法を理解できます。

「 Oracle データベース」とは、 1 つの単位として扱われるデータの集合です。データベースの一般的な目的は、関連する情報を格納したり、取り出したりすることです。データベースには、「論理構造」と「物理構造」があります。

論理データベース構造

この後の項では、表領域およびスキーマ・オブジェクト、データ・ブロック、エクステント、セグメントをはじめとするデータベースの論理構造について説明します。

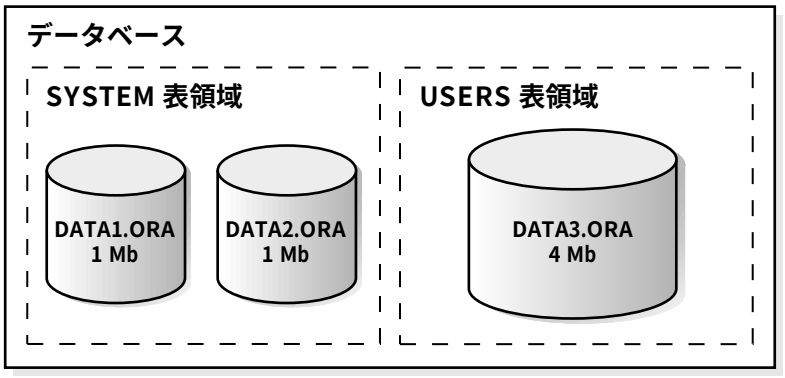
表領域

データベースは、「表領域」と呼ばれる論理的な記憶単位に分けられます。表領域は、関連する論理構造をまとめてグループ化するために使われます。たとえば、通常、表領域は 1 つのアプリケーションのすべてのオブジェクトをグループにまとめ、管理的な操作を簡単にします。

データベース、表領域、データ・ファイル

データベース、表領域、データ・ファイルの間の関連を図 1-2 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 1-2 **データベース、表領域、データ・ファイル**



この図から次のようなことがわかります。

- 各データベースは、論理的に 1 つ以上の表領域に分けられる。

- 表領域内のすべての論理構造のデータを物理的に格納するために、1 つ以上のデータ・ファイルが各表領域に対して明示的に作成される。
- 表領域のデータ・ファイルのサイズを合わせると、表領域全体の記憶容量になる（SYSTEM 表領域は 2MB 、 USERS 表領域は 4MB の記憶容量を持つ）。
- データベースの表領域の記憶容量を合わせると、データベース全体の記憶容量になる（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え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、多くの管理業務を簡単に実行できます。

スキーマとスキーマ・オブジェクト

「スキーマ」とは、データベース・オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタ、データベース・リンクなどの構造体が含まれます。（表領域とスキーマには何の関連もありません。同じスキーマ内のオブジェクトが別々の表領域に存在したり、異なるスキーマにあるオブジェクトが 1 つの表領域に保持されたりすることがあります。）スキーマ・オブジェクトの詳細は、1-41ページの「スキーマとスキーマ・オブジェクト」を参照してください。

データ・ブロック、エクステント、セグメント

Oracle では、データ・ブロック、エクステント、セグメントなどの論理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。これらの詳細は、第 2 章「データ・ブロック、エクステント、セグメント」を参照してください。

Oracle データ・ブロック

最も細かいレベルとして、 Oracle データベースのデータは「データ・ブロック」に格納されます。 1 つのデータ・ブロックは、ディスク上の特定のバイト数の物理データベース領域に対応します。データ・ブロックのサイズは、データベースの作成時に Oracle データベースに対して指定します。データベースは、 Oracle データ・ブロック内の空きデータベース領域を使って領域を割り当てます。

エクステント

論理的なデータベース領域の次のレベルは、エクステントと呼ばれます。「エクステント」は、特定数の連続したデータ・ブロックであり、1 回の割当てで取得され、特定のタイプの情報を格納するために使われます。

セグメント

エクステントの上位に位置する論理的なデータベース記憶領域のレベルは、セグメントと呼ばれます。「セグメント」は、特定の論理構造に割り当てられるエクステントの集合です。たとえば、次のような異なるタイプのセグメントがあります。

データ・セグメント	クラスタ化されていない表は、それぞれ 1 つのデータ・セグメントを持っています。表のデータはすべて、そのデータ・セグメントのエクステントに格納されます。各クラスタは、1つのデータ・セグメントを持っています。クラスタ内のあらゆる表のデータが、そのクラスタのデータ・セグメントに格納されます。
索引セグメント	各索引は、そのデータをすべて格納する索引セグメントを 1 つ持っています。
ロールバック・セグメント	データベース管理者は、「アンドゥ（取消し）」情報を一時的に格納するために、データベースごとに 1 つ以上のロールバック・セグメントを作成します。取消し情報は、次のような目的で使われます。 • 読込み一貫性を備えたデータベース情報を生成するため（ 1-21 ページの「読込み一貫性」を参照） • データベースの回復のとき（ 1-33 ページの「データベースのバックアップとリカバリ」を参照） • ユーザーの必要に応じて、コミットされていないトランザクションをロールバックするため
一時セグメント	一時セグメントは、 SQL 文の実行を完了するために一時的な作業領域が必要なときに、 Oracle によって作成されます。その文の実行が終了すると、一時セグメントのエクステントは後で利用できるようにシステムに戻されます。

Oracle は、 1 つのセグメントの既存のエクステントが満杯になった時点で、領域を動的に割り当てます。したがって、セグメントの既存のエクステントが満杯である場合は、必要に応じてそのセグメントに別のエクステントを割り当てます。エクステントは必要に応じて割り当てられるため、 1 つのセグメントの各エクステントはディスク上で連続している場合と連続していない場合があります。

物理データベース構造

この後の項では、データ・ファイル、 REDO ログ・ファイル、制御ファイルをはじめとする Oracle データベースの物理構造について説明します。

データ・ファイル

すべてのOracleデータベースは、1つ以上の物理「データ・ファイル」を持っています。データベースのデータ・ファイルには、すべてのデータベース・データが格納されます。表や索引などの論理データベース構造のデータは、物理的にはデータベースに割り当てられたデータ・ファイルに格納されます。

データ・ファイルの特性は次のとおりです。

- 1 つのデータ・ファイルは、 1 つのデータベースだけに対応付けられる。
- データベース・ファイルには、データベースの領域を使いきってしまった場合にファイルが自動的に拡張されるように、特定の特性を設定できる。
- 前述のように、 1 つ以上のデータ・ファイルによって、表領域と呼ばれるデータベース記憶の論理単位が形成される。

データ・ファイルの使用

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

修正されたデータや新規のデータは、必ずしも即時にデータ・ファイルに書き込まれるわけではありません。ディスクへのアクセス回数を減らし、パフォーマンスを向上させるために、データはいったんメモリー内に蓄積されて、適切なデータ・ファイルに一度に書き込まれます。これらの操作は、 Oracle の DBWn バックグラウンド・プロセスによって決定されます。（ Oracle のメモリーおよびプロセスの構造、データベースのデータをデータ・ファイルに書き込むときのアルゴリズムの詳細は、 1-12 ページの「メモリー構造とプロセス」を参照してください。）

REDO ログ・ファイル

すべての Oracle データベースは、 2 つ以上の「 REDO ログ・ファイル」の集まりを持っています。データベースの REDO ログ・ファイルの集まりは、データベースの「 REDO ログ」と呼ばれます。REDOログの主な目的は、データに加えられた変更をすべて記録することです。万一障害が発生して、修正済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・ファイルは、障害からデータベースを保護する上で重要です。 REDO ログ自体にかかわる障害から保護するために、 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、「多重化された REDO ログ」を使えるようにしています。

REDO ログ・ファイルの使用

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のを妨げるようなシステム障害や媒体障害が起きた場合に、データベースを回復するためにだけ使われます。

たとえば、予期しない電源異常によってデータベース操作が突然停止するような場合、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。しかし、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回復されます。 Oracle は、最新の REDO ログ・ファイル内にある情報をデータベースのデータ・ファイルに反映させることにより、電源障害が発生した時点までデータベースを復元します。

回復操作中に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。1-33ページの「データベースのバックアップとリカバリ」を参照してください。

制御ファイル

すべてのOracleデータベースには、「制御ファイル」があります。データベースの制御ファイルには、データベースの物理構造を指定するエントリが入っています。たとえば、制御ファイルには次のようなタイプの情報が入ります。

- データベース名
- データベースのデータ・ファイルと REDO ログ・ファイルの名前と位置
- 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

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しなければならないデータベース・ファイルと REDO ログ・ファイルを、制御ファイルを使って識別します。データベースの物理的な構造が変更される（たとえば、新しいデータベース・ファイルまたは REDO ログ・ファイルが作成される）と、データベースの制御ファイルは、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。

データベースの制御ファイルは、データベースの回復が必要な場合にも使われます。詳細は、1-33 ページの「データベースのバックアップとリカバリ」および第 28 章「データベースの回復」を参照してください。

メモリー構造とプロセス

この項では、データベースを管理するためにOracle Serverが使うメモリーとプロセスの構造について説明します。この項で説明するアーキテクチャの特徴によって、特に次のような処理をサポートする Oracle Server の機能を理解することができます。

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

Oracle Serverは、メモリー構造とプロセスを使ってデータベースを管理したりアクセスしたりします。すべてのメモリー構造は、データベース・システムを構成するコンピュータのメイン・メモリー内に存在します。

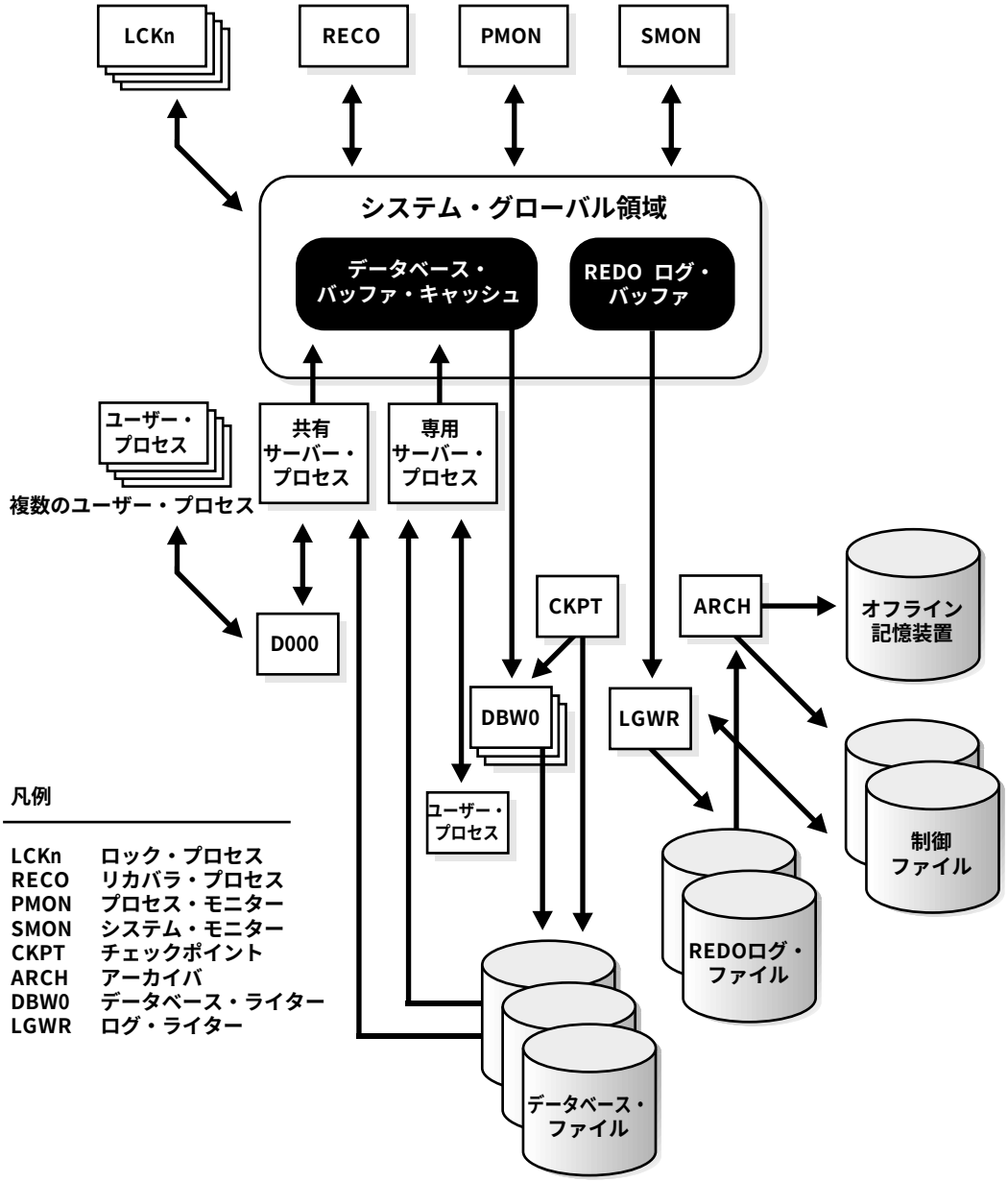
「プロセス」とは、これらのコンピュータのメモリー内で作業を行うジョブまたはタスクです。

1-14 ページの図 1-3 「Oracle のメモリー構造とプロセス」に、 Oracle Server のメモリーとプロセスの構造の代表的な例を示します。

メモリー構造

Oracle はメモリー構造を作成して使うことによって、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ために、メモリーが使われます。基本的なメモリー構造、つまりシステム・グローバル領域（データベース・バッファ、 REDO ログ・バッファ、共有プールが含まれる）とプログラム・グローバル領域が、 Oracle に関連付けられます。この後、それぞれのメモリー領域について詳しく説明します。

図 1-3 Oracle のメモリー構造とプロセス



システム・グローバル領域（SGA）

「システム・グローバル領域（SGA）」は、1つのOracleインスタンスのためのデータと制御情報が入る共有メモリー領域です。 SGA と Oracle バックグラウンド・プロセスによって、 1つの Oracle インスタンスが 構成されます。詳細は、 1-6 ページの「 Oracle インスタンス」および 1-17 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。

システム・グローバル領域はインスタンスの起動時に割り当てられ、そのインスタンスの停止時に割当てが解除されます。各インスタンスは、それぞれ専用のシステム・グローバル領域を持っています。

システム・グローバル領域内のデータは、Oracle Serverに現在接続しているユーザー間で共有されます。最適なパフォーマンスを得るには、システム・グローバル領域全体を可能な限り大きくして（ただし、実メモリーの範囲内)、メモリーに可能な限り多くのデータを格納し、ディスク I/O を最小限にとどめるようにします。

システム・グローバル領域内に格納される情報は、データベース・バッファ、 REDO ログ・バッファ、共有プールなどのいくつかのタイプのメモリー構造に分けられます。これらの領域のサイズは決まっていて、インスタンスの起動時に作成されます。

データベース・バッファ・キャッシュ

システム・グローバル領域の「データベース・バッファ」は、最後に使われたデータベース・データのブロックを格納します。 1つのインスタンス内のデータベース・バッファの集まりが、「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。最後に使われれたデータ（これは、多くの場合、最も頻繁に使われるデータ）をメモリー内に保持することによって、必要なディスク I/O が少なくなり、パフォーマンスが向上します。

REDO ログ・バッファ

システム・グローバル領域の「REDOログ・バッファ」は、「REDOエントリ」を格納します。 REDO エントリは、データベースに対して加えられた変更のログです。 REDO ログ・バッファ内に格納された REDO ログ・エントリは、オンライン REDO ログ・ファイルに書き込まれます。 REDO ログ・ファイルは、データベースの回復が必要になったときに使われます。バッファのサイズは固定されています。

共有プール

共有プールは、システム・グローバル領域の中で共有 SQL 領域などの共有メモリーが含まれている部分です。データベースに送られる一意の SQL 文それぞれを処理するために、共有 SQL 領域が必要になります（ 1-47 ページの「 SQL 文」を参照してください)。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入ります。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、他に利用できる共有メモリーがより多く残ります。

文ハンドルまたはカーソル

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

たとえば、プリコンパイラのアプリケーション開発において、カーソルはプログラムから利用できる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 文を解析するために使えます。アプリケーション開発者は、 SQL 文の実行フェーズを制御し、そのパフォーマンスが改善されるように、アプリケーションをコーディングできます。

プログラム・グローバル領域（ PGA ）

「プログラム・グローバル領域（PGA）」とは、 1つのサーバー・プロセスのためのデータと制御情報が入るメモリー・バッファです。サーバー・プロセスの起動時に、 Oracle によって作成されます。 PGA 内の情報は Oracle の構成によって異なります。

プロセスのアーキテクチャ

「プロセス」は「制御のスレッド」、つまり一連のステップを実行できるオペレーティング・システムのメカニズムです。オペレーティング・システムによっては、「ジョブ」または「タスク」という用語を使います。プロセスには、通常、プロセスそのものを実行するための専用のプライベート・メモリー領域があります。

Oracle Server には、ユーザー・プロセスと Oracle プロセスという 2 つの一般的なタイプのプロセスがあります。

ユーザー（クライアント）プロセス

「ユーザー・プロセス」は、Pro*C/C++プログラムなどのアプリケーション・プログラムや、Oracle Enterprise Manager などの Oracle Tool のソフトウェア・コードを実行するために作成され、維持されます。また、ユーザー・プロセスは、サーバー・プロセスとの通信を管理します。

後述するように、ユーザー・プロセスは、プログラム・インタフェースを介してサーバー・プロセスと通信します。

Oracle プロセス・アーキテクチャ

「 Oracle プロセス」は、他のプロセスによって呼び出され、呼び出したプロセスにかわって機能を実行します。各 Oracle プロセスとその特有の機能について、次に説明します。それらのプロセスには、サーバー・プロセスとバックグラウンド・プロセスがあります。

サーバー・プロセス

Oracle は、接続されたユーザー・プロセスの要求を処理するために、「サーバー・プロセス」を作ります。サーバー・プロセスは、対応付けられたユーザー・プロセスの要求を実行するために、ユーザー・プロセスと通信し、 Oracle と対話します。たとえば、ユーザーがシステム・グローバル領域のデータベース・バッファ内に存在しないデータを問い合せた場合、対応するサーバー・プロセスが、データ・ファイルからシステム・グローバル領域に適切なデータ・ブロックを読み込みます。

Oracle では、サーバー・プロセスあたりのユーザー・プロセス数が変動してもよいように構成できます。「専用サーバー構成」では、 1 つのサーバー・プロセスが 1 つのユーザー・プロセスの要求を処理します。「マルチスレッド・サーバー構成」では、多数のユーザー・プロセスが少数のサーバー・プロセスを共有できます。こうして、サーバー・プロセスの数を最低限に抑え、使用可能なシステム・リソースの利用効率を最大化できます。

ユーザー・プロセスとサーバー・プロセスが別個のプロセスになっているシステムと、それらが 1 つのプロセスにまとめられているシステムがあります。システムがマルチスレッド・サーバーを使っている場合、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なければなりません。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一式の「バックグラウンド・プロセス」を作成します。バックグラウンド・プロセスは、各ユーザー・プロセスごとに実行されている複数の Oracle プログラムが処理する機能を 1 つにまとめます。バックグラウンド・プロセスは、 I/O を非同期的に実行し、他の Oracle プロセスを監視することによって、並列性を強化してパフォーマンスおよび信頼性を向上させます。

SGA と一式の Oracle バックグラウンド・プロセスによって、 1 つの Oracle インスタンスが構成されます。（SGA の詳細は、 1-6 ページの「Oracle インスタンス」および 1-15 ページの「システム・グローバル領域（ SGA ）」を参照してください。） Oracle インスタンスは、それぞれ複数のバックグラウンド・プロセスを使います。これらのプロセスの名前は、 DBWn および LGWR 、 CKPT 、 SMON 、 PMON 、 ARCH 、 RECO 、 Dnnn 、 LCKn 、 SNPn 、 QMNn です。

データベース・ライター（ DBWn ）

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

Oracle は事前ロギングを行うので、 DBWn によって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（1-48ページの「トランザクション」を参照）。そのかわりに、 DBWn 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。通常の場合、 DBWn が書込みを行うのは、システム・グローバル領域に読み込む必要のあるデータが増加し、空き状態のデータベース・バッファがほとんどなくなった場合だけです。最低使用頻度のデータから先に、データ・ファイルに書き込まれます。 DBWn 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。

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。REDOログのデータは、システム・グローバル領域の REDO ログ・バッファ内に生成されます。トランザクションがコミットされたり、ログ・バッファが満杯になったりすると、 LGWR は 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。

チェックポイント（CKPT）

特定のタイミングで、システム・グローバル領域内のすべての修正されたデータベース・バッファが、 DBWn によってデータ・ファイルに書き込まれます。このイベントのことをチェックポイントといいます。「チェックポイント」プロセスは、チェックポイント時に DBWn に信号を送り、データベースのすべてのデータ・ファイルと制御ファイルを最新のチェックポイントを反映す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、インスタンスの起動時にインスタンス回復を実行します。複数インスタンス・システム（Oracle Parallel Server を使うシステム）では、あるインスタンスの SMON が、障害の発生した他のインスタンスのためにインスタンス回復を実行することもできます。また、 SMON は、使われなくなった一時セグメントをクリーン・アップし、クラッシュ時とインスタンス回復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回復します。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 によって回復されます。また、SMON はデータベース内の空きエクステントを合わせて連続した空き領域を作成し、割当てを容易にします。

プロセス・モニター（PMON）

ユーザー・プロセスが障害を起こすと、「プロセス・モニター」がプロセスの回復を実行します。 PMON は、キャッシュをクリーン・アップしたり、プロセスが使っていたリソースを解放したりします。また、ディスパッチャ・プロセス（後述）とサーバー・プロセスをチェックして、障害が発生している場合にはそれらを再起動します。

アーカイバ（ARCH）

アーカイバは、オンライン REDO ログ・ファイルが満杯になったときに、それらのログ・ファイルをアーカイブ記憶域にコピーします。 ARCH がアクティブになるのは、データベースの REDO ログを ARCHIVELOG モードで使っている場合だけです。（1-36 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、分散データベースにおける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われます。ローカル RECO は、ある決まった間隔でリモート・データベースに接続し、保留中の分散トランザクションがあれば、そのローカル部分を自動的にコミットまたはロールバックします。

ディスパッチャ（Dnnn）

「ディスパッチャ」は、オプションのバックグラウンド・プロセスです。これが存在するのは、マルチスレッド・サーバー構成を使っている場合だけです。使用中の通信プロトコルごとに、少なくとも 1 つのディスパッチャ・プロセス（D000 、 . . . 、 Dnnn）が作成されます。各ディスパッチャ・プロセスは、接続されたユーザー・プロセスから利用できる共有サーバー・プロセスへの要求の経路を指示し、適切なユーザー・プロセスにその応答を戻します。

ロック（LCKn）

Oracle Parallel Serverでは、インスタンス間のロックのために「ロック」プロセス（LCK0、. . .、LCK9）が使われます。この構成の詳細は、1-7ページの「Oracle Parallel Serverによる複数インスタンス・システム」を参照してください。

ジョブ・キュー（SNPn）

分散データベース構成では、最大36個の「ジョブ・キュー・プロセス」（SNP0、...、SNP9、SNPA 、 ... 、 SNPZ ）が自動的に表スナップショットをリフレッシュできます。これらのプロセスは定期的に起動され、自動リフレッシュがスケジュールされているスナップショットをリフレッシュします。複数のジョブ・キュー・プロセスを使うと、プロセス間でスナップショットのリフレッシュ・タスクが共有されます。これらのプロセスは、 DBMS_JOB パッケージによって作成されるジョブ要求も実行し、キューイングされたメッセージを他のデータベースのキューに波及させます。

キュー・モニター（QMNn）

「キュー・モニター」は、 Oracle Advanced Queuing（Oracle AQ）のメッセージ・キューをモニターするオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。

プログラム・インタフェース

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

- 通信メカニズムとしての機能。データ要求の形式設定、データの引渡し、エラーの検出と返送を行います。
- データ変換の機能。特に異機種コンピュータ間での変換、または外部ユーザー・プログラムのデータ型への変換を行います。

Oracle の動作例

次に、ユーザー・プロセスとそれに対応するサーバー・プロセスが別々のマシン（ネットワークを介して接続されている）上に存在するOracleの構成について、具体的に説明します。

1. 現在、インスタンスは、 Oracle が稼動しているコンピュータ（「ホスト」または「データベース・サーバー」と呼ばれることが多い）上で実行されています。
2. アプリケーションを実行しているコンピュータ（「ローカル・マシン」または「クライアント・ワークステーション」）は、ユーザー・プロセスでアプリケーションを実行します。クライアント・アプリケーションは、適切な Net8 ドライバを使って、サーバーへの接続を確立しようとします。
3. サーバーは、適切な Net8 ドライバを実行しています。サーバーはアプリケーションからの接続要求を検出すると、ユーザー・プロセスのために（専用の）サーバー・プロセスを作成します。

4. ユーザーは SQL 文を実行して、トランザクションをコミットします。たとえば、ユーザーは表の行に入っている名前を変更します。
5. サーバー・プロセスは SQL 文を受け取り、同一の SQL 文を含む共有 SQL 領域がないかどうか共有プールをチェックします。共有 SQL 領域が見つかった場合、サーバー・プロセスは要求されたデータに対するユーザーのアクセス権限をチェックし、既存の共有 SQL 領域を使って SQL 文を処理します。共有 SQL 領域が見つからなかった場合には、解析と処理を実行するための新しい共有 SQL 領域をその文に割り当てます。
6. サーバー・プロセスは、実際のデータ・ファイル（表）から必要なデータ値を取得するか、またはシステム・グローバル領域内に格納されている必要なデータ値を取得します。
7. サーバー・プロセスは、システム・グローバル領域内のデータを修正します。 DBWn プロセスは、書込みが効率よく行われる時期を判断して、修正したブロックをディスクに書き込みます。トランザクションがコミットされると、 LGWR プロセスがただちにそのトランザクションをオンライン REDO ログ・ファイルに記録します。
8. トランザクションが成功すると、サーバー・プロセスは、ネットワークを介してアプリケーションにメッセージを送信します。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体を通じて、他のバックグラウンド・プロセスも実行されていて、介入が必要かどうかを監視しています。さらに、データベース・サーバーは、他のユーザーのトランザクションを管理し、同一のデータを要求する異なるトランザクション間の競合を防止します。

これらのステップは、 Oracle が実行する操作の最も基本的なレベルにすぎません。（第 7 章「プロセス構造」を参照してください。）

データの同時実行性と一貫性

この項では、情報管理システムの次のような重要な要件を満たすために Oracle が使う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、修正する。
- マルチユーザー・システムのデータの同時実行性を最大化する。
- データベース・システムの数多くのユーザーの生産性を最大にするために、高パフォーマンスを実現する。

同時実行性

マルチユーザー・データベース管理システムの最大の関心は、「同時実行性」をどのように制御するか、すなわち多数のユーザーによる同一データへの同時アクセスをどのように制御するかということです。十分な同時実行制御機能がないと、データが不当に更新されたり変更されたりして、データの整合性が損なわれる可能性があります。

数多くのユーザーが同一データにアクセスしている場合、データの同時実行性を管理する 1 つの方法は、各ユーザーに順番待ちをしてもらうことです。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、あるいは無視できる程度まで短縮することです。すべてのデータ操作（DML）文を可能なかぎり干渉がなくなるように処理して、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しなければなりません。破壊的な相互作用とは、不正確なデータの更新または基礎となるデータ構造の不正な変更を引き起こす相互作用です。パフォーマンスとデータの整合性は、どちらも無視できません。

Oracle では、さまざま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うことで、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。 1-50 ページの「トランザクションを使ったデータの一貫性」で説明されているように、同時実行性と一貫性に関するこれらの機能がトランザクションで十分に活かされるようにするのは、アプリケーションの設計者の責任です。

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ない（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ない。
- 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ない。
- 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だけ、データを書き込むユーザーは他の書込みユーザーを待機する。

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、ロールバック・セグメント、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しなければなりません。更新が発生すると、更新によって変更されたデータの元の値が、データベースのロールバック・セグメント内に記録されます。この更新がコミットされていないトランザクションの一部である限り、修正されたデータを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ロールバック・セグメント内の情報を使って、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」開始された文だけが、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す。
- データベースの他のトランザクションが、修正されたデータを読込みまたは更新のために参照できる時期を制御する。

読込み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、このほかにトランザクション・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、更新をしない場合は、「読込み専用トランザクション」を使うとよいでしょう。トランザクションが読込み専用であることを指定した後は、それぞれの問合せの結果が同一時点について読込み一貫性を持つことがわかっているので、任意の表に対して必要な数だけの問合せを実行できます。

ロックのメカニズム

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

ロックは、次の 2 つの重要なデータベースの目標を達成するために使われます。

一貫性	ユーザーがデータを使い終わるまで、参照中または変更中のデータが（他のユーザーによって）変更されないことを保証する。
整合性	データベースのデータと構造が、すべての変更を正しい順序で反映していることを保証する。

ロックは、データの整合性を保証すると同時に、無制限の数のユーザーがデータに同時実行アクセスできるようにします。

自動ロック

Oracle のロックは自動的に実行されるため、ユーザーのアクションはまったく必要ありません。要求されるアクションによっては、必要に応じて暗黙のロックが SQL 文に対して発生します。

Oracle の洗練されたロック・マネージャ（ロック管理機能）は、行レベルで表データを自動的にロックします。行レベルで表データをロックすることによって、同一データへの競合が最小限に抑えられます。

Oracle のロック・マネージャは、ロックを設定する操作のタイプに応じて、異なるタイプの行ロックを保持します。ロックには一般に、「排他ロック」と「共有ロック」の 2 つのタイプがあります。排他ロックは、1つのリソース（行や表など）に対して1つだけ取得できます。一方、共有ロックは、 1 つのリソースに対して数多く取得できます。排他ロックと共有ロックでは、ロックされたリソースに対する問合せはいつでも許可されますが、そのリソースに対する他のアクティビティ（更新や削除など）は禁止されます。

手動ロック

状況によっては、ユーザーは、デフォルト・ロックを変更することができます。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ことによる）と表レベルの両方で手動変更できます。

分散処理と分散データベース

現代のコンピュータ環境においてコンピュータ・ネットワーキングが普及するにつれて、データベース管理システムには、処理機能と格納機能を分散する機能が必要になっています。この項では、これらの要件に応える Oracle のアーキテクチャ上の特徴について説明します。

クライアント / サーバー・アーキテクチャ： 分散処理

「分散処理」では、関連するジョブの集合に対する処理を分割するために、複数のプロセッサを使います。分散処理では、異なるプロセッサが関連するタスクのサブセットを集中処理することにより、 1 つのプロセッサの処理負荷が低減され、システム全体のパフォーマンスと能力が向上します。

Oracle データベース・システムでは、「クライアント / サーバー・アーキテクチャ」を使うことにより、分散処理を簡単に利用できます。このアーキテクチャでは、データベース・システムがフロントエンドの「クライアント」部と、バックエンドの「サーバー」部という 2 つに部分に分割されます。

クライアント

クライアント部は、フロントエンドのデータベース・アプリケーションであり、キーボード、ディスプレイ、およびマウスなどのポインティング・デバイスを使ってユーザーと対話します。クライアント部では、データ・アクセスはあまり重要な処理ではなく、サーバー部によって管理されるデータの要求、処理、表示を集中的に実行します。クライアント・ワークステーションは、このようなジョブに合わせて最適化できます。たとえば、必要なディスク容量を少なくしたり、グラフィック機能を活用したりできます。

サーバー

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

分散データベース

「分散データベース」は、ユーザーからは単一の論理データベースのように見えますが、実際には複数のデータベース・サーバーによって管理されるデータベースのネットワークです。分散データベース内のすべてのデータベースのデータは、同時にアクセスしたり変更したりできます。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入っているデータを論理的に結合し、ネットワーク上のすべてのユーザーに対してアクセス可能にできることです。

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

分散データベースでは、ネットワークを介した大量のデータに対するアクセスを増加させることができますが、その一方で、データの位置とネットワークをまたがるアクセスの複雑さを隠す機能も提供しなければなりません。分散データベース管理システムでは、さらに各ローカル・データベースを非分散データベースと同様に管理できるという特長も維持していなければなりません。

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済む（透過である）とき、「位置の透過性」が実現されます。ビュー、プロシージャ、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

「サイトの自律性」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ながら動作しますが、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、および削除を含む、すべてのDML操作をサポートしています。リモート・データをアクセスするには、リモート・オブジェクトのグローバル・オブジェクト名を含めた参照を作成します。リモート・データにアクセスするための特別なコーディングや複雑な構文は必要ありません。

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

```
SELECT * FROM emp@sales;
```

2 フェーズ・コミット

Oracle は、分散環境でも、非分散環境と同じようにデータの一貫性を保証できます。 Oracle では、トランザクション・モデルと「2 フェーズ・コミット・メカニズム」を使ってこの一貫性を保証します。

分散システム以外のシステムと同様にトランザクションを慎重に計画し、論理的な SQL 文の集合を1つの単位としてトランザクションに含めることにより、それらがすべて成功するか、さもなければすべて失敗するというようにできます。 Oracle の 2 フェーズ・コミット・メカニズムによって、発生したシステム障害やネットワーク障害のタイプに関係なく、分散トランザクションがすべての関連ノード上でコミットまたはロールバックされ、グローバルな分散データベースでのデータの一貫性を維持できることが保証されます。

データベース・ユーザーに対する完全な透過性

Oracle の 2 フェーズ・コミット・メカニズムは、分散トランザクションを発行するユーザーに対して完全に透過的です。トランザクションの終了を示す単純なCOMMIT文が、トランザクションをコミットするために 2 フェーズ・コミット・メカニズムを自動的に起動します。つまり、データベース・アプリケーションの本体に、分散トランザクションを含めるための特別なコーディングや複雑な文の構文は必要ありません。

システム障害やネットワーク障害からの自動回復

RECO（リカバラ）バックグラウンド・プロセスが、「インダウト分散トランザクション」（システム障害やネットワーク障害によってコミットが中断された分散トランザクション）の結果を自動的に解決します。障害が修復され、通信が再確立された後、各ローカルOracle Server の RECO が、関係するすべてのノード上でインダウト分散トランザクションを自動的にコミットするか、またはロールバックします。

手動による問題解決の機能

長時間にわたる障害が発生した場合、 Oracle では、各ローカル管理者が、障害によって発生したインダウト分散トランザクションを手動でコミットするか、またはロールバックできます。このオプションによって、ローカル・データベースの管理者は、長時間にわたる障害の結果として無期限に拘束される可能性のあるロックされたリソースを解放できます。

分散回復のための機能

あるデータベースを過去のある時点まで回復しなければならない場合、 Oracle の回復機能を使うことによって、他のサイトのデータベース管理者は自分たちのデータベースをその過去の時点に戻すことができます。これによって、グローバル・データベースは一貫した状態を保つことができます。

表のレプリケーション

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をレプリケートすることがよくあります。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のに役立ちます。

データは、スナップショット、つまりレプリケートされたマスター表を使ってレプリケートできます。詳細は、第 31 章「データベースのレプリケーション」を参照してください。

追加情報: 『Oracle8 Server レプリケーション』には、データベースのレプリケーションに関する詳細情報が記載されています。

Oracle と Net8

「Net8」は、ネットワークで使わ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、ネットワークでのデータの送受信方法が定義されます。ネットワーク環境では、 Oracle データベース・サーバーは、 Oracle ソフトウェアである Net8 を使って、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の広い範囲で、 Net8 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ます。

Net8 を使えば、アプリケーション開発者はデータベース・アプリケーション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う場合も、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まったく修正しなくても機能し続けます。

起動操作と停止操作

最初に Oracle Server を起動し、データベースをオープンしておかないと、ユーザーは Oracle データベースを利用できません。この操作は、データベース管理者が実行します。データベースを起動してシステム全体で利用できるようにするには、次の3つのステップを実行します。

1. Oracle Server のインスタンスを起動する。
2. データベースをマウントする。
3. データベースをオープンする。

これらのステップの説明は、5-5ページの「インスタンスとデータベースの起動」を参照してください。

Oracle Server が起動すると、初期化パラメータが含まれるパラメータ・ファイルが使われます。これらのパラメータは、データベース名、割り当てるメモリー量、制御ファイルの名前、および種々の制限や他のシステム・パラメータを指定します。パラメータ・ファイルのサンプルは、 5-3 ページの「パラメータ・ファイル」を参照してください。

追加情報: 初期化パラメータの詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

インスタンス、およびインスタンスが接続されているデータベースを停止するには、次の 3 つのステップを実行します。

1. データベースをクローズする。

2. データベースをディスマウントする。
3. Oracle Server のインスタンスを停止する。

インスタンスを停止すると、この 3 つのステップがすべて自動的に実行されます。詳細は、5-7 ページの「データベースとインスタンスの停止」を参照してください。

データベース・セキュリティ

Oracleのようなマルチユーザー・データベース・システムには、データベースをアクセスし、使用する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・アクセスを防止する。
- 権限のないスキーマ・オブジェクトへのアクセスを防止する。
- ディスクの使用を制御する。
- システム・リソース（ CPU 時間など）の使用を制御する。
- ユーザーのアクションを監査する。

「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。スキーマは、データベース・オブジェクト（表、ビュー、順序、シノニム、索引、クラスタ、プロシージャ、ファンクション、パッケージ、データベース・リンク）の論理的な集まりです。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・メカニズムは、次の事項をチェックします。

- ユーザーがデータベースへの接続を許可されているかどうか
- データベース監査がアクティブであるかどうか
- ユーザーはどのシステム操作を実行できるか

「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。データベース・セキュリティには、たとえば次のものが含まれます。

- 特定のスキーマ・オブジェクトにアクセスできるユーザーと、そのスキーマ・オブジェクトに対して各ユーザーが許可されている特定のタイプのアクション（たとえば、ユーザー SCOTT は EMP 表に対して SELECT 文と INSERT 文を発行できるが、 DELETE 文は発行できない）。
- 各スキーマ・オブジェクトに対して監査されるアクション。

セキュリティのメカニズム

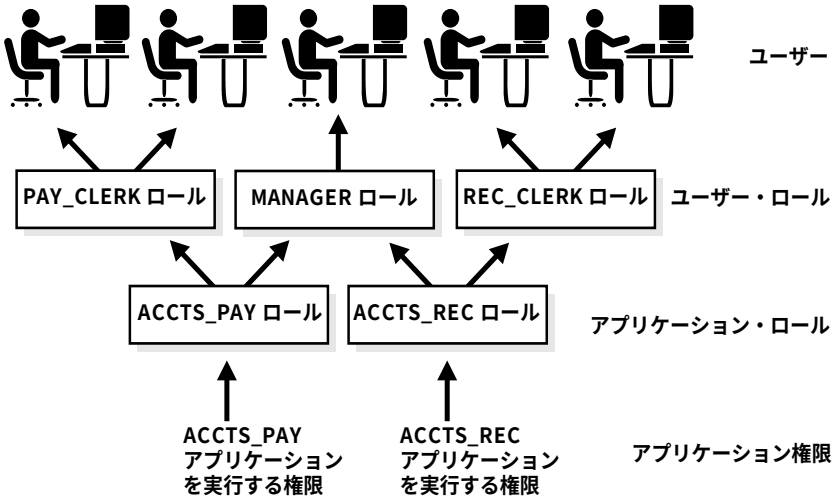
Oracle Server では、「任意アクセス制御」が可能です。任意アクセス制御とは、権限に基づいて情報へのアクセスを制限する方法です。ユーザーがスキーマ・オブジェクトにアクセスするには、そのユーザーは適切な権限を割り当てられていなければなりません。適切な権限を付与されているユーザーは、他のユーザーに自分で任意に権限を付与することができます。このため、このタイプのセキュリティを「任意」セキュリティと呼びます。

Oracle は、いくつかの異なる機能を使ってデータベース・セキュリティを管理します。次のとおりです。

- データベース・ユーザーとスキーマ
- 権限
- ロール
- 記憶領域の設定と割当て制限
- リソース制限
- 監査

図 1-4に、Oracleの各種セキュリティ機能の関連を示します。この後の項では、ユーザー、権限、ロールの概要について説明します。

図 1-4 Oracle のセキュリティ機能



データベース・ユーザーとスキーマ

それぞれの Oracle データベースは、ユーザー名のリストを持っています。データベースにアクセスするには、ユーザーはデータベース・アプリケーションを使ってデータベースの有効なユーザー名で接続しなければなりません。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属性を持っています。

- ユーザーが利用できるアクション（権限とロール）
- ユーザーに対する表領域の割当て制限（利用可能なディスク領域）
- ユーザーのためのシステム・リソース制限（ CPU 処理時間など）

ユーザーのセキュリティ・ドメインに寄与する各属性については、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、次のような権利が権限といえます。

- データベースに接続する権利（セッションを作成する権利）
- スキーマ内に表を作成する権利

- 他のユーザーの表から行を選択する権利
- 他のユーザーのストアド・プロシージャを実行する権利

Oracle データベースの権限は、システム権限とスキーマ・オブジェクト権限の 2 つに分類できます。

システム権限

「システム権限」によって、ユーザーは特定のシステム全体にわたるアクションを実行したり、特定タイプのスキーマ・オブジェクトに対して特定のアクションを実行したりすることができます。たとえば、表領域を作成する権限やデータベース内の任意の表の行を削除する権限はシステム権限です。多くのシステム権限は非常に強力であるため、それを使えるのは管理者とアプリケーション開発者だけです。

スキーマ・オブジェクト権限

「スキーマ・オブジェクト権限」によって、ユーザーは特定のスキーマ・オブジェクトに対して特定のアクションを実行できます。たとえば、特定の表の行を削除する権限は、オブジェクト権限です。エンドユーザーは、オブジェクト権限を付与される（割り当てられる）と、データベース・アプリケーションを使っ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をアクセスしたり修正したりできます。ユーザーは、次のような 2 通りの形態で権限を受け取ることができます。

- 権限は明示的にユーザーに対して付与できる。たとえば、 EMP 表にレコードを挿入するための権限を、明示的にユーザー SCOTT に付与できます。
- 権限は「ロール」（名前付き権限グループ）に対して付与できる。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る。たとえば、 EMP 表にレコードを挿入するための権限を、 CLERK という名前のロールに付与できます。さらに、このロールをユーザー SCOTT や BRIAN に付与できます。

ロールによって、より簡単でより優れた権限管理が実現できるため、権限は通常、ロールに対して付与し、特定のユーザーには付与しないようにします。次の項では、ロールとその使用について詳しく説明します。

ロール

Oracle では、ロールを使うことにより、簡単かつ制御された権限管理を実現できます。「ロール」は、関連する権限のグループに名前を付けたもので、ユーザーまたは他のロールに付与されます。ロールの次のような特徴によって、一層簡便な権限管理を実現できます。

- 権限を付与する件数を減らせる。データベース管理者は、多数のユーザーに同じ権限の集合を明示的に付与するのではなく、関連するユーザーのグループのための権限をロールに付与してから、グループの各メンバーにそのロールを付与できます。
- 動的な権限管理。グループの権限を変更しなければならない場合には、ロールの権限を修正するだけで済みます。グループのロールを付与した全ユーザーのセキュリティ・ドメインは、そのロールに対して加えられる変更を自動的に反映します。

- 権限の可用性を選択できる。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。
- アプリケーションを感知できる。ユーザーがアプリケーションを使おうとしたときに、選択したロールが自動的に使用可能または使用禁止にされるように、データベース・アプリケーションを設計できます。

データベース・アプリケーションについてのロールは、通常、データベース管理者が作成します。DBA は、アプリケーションを実行するのに必要なすべての権限をアプリケーション・ロールに付与します。その後、 DBA は、アプリケーション・ロールを別のロールや特定のユーザーに対して付与できます。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に入れた、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て、そのロールに付与された権限が無許可で使われるのを防止できます。通常、アプリケーションはその開始時に適当なロールが使用可能になるように設計されます。その結果、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っている必要がありません。

記憶領域の設定と割当て制限

Oracle は、データベースに対して割り当てられるユーザー当たりのディスク領域の使用を管理また制御できます。これには、デフォルト表領域、一時表領域、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、表または索引、クラスタを作成するときにそのスキーマ・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われます。デフォルト表領域の機能によって、スキーマ・オブジェクトの位置が指定されていない状況で、領域使用を指示する情報が Oracle に与えられます。

一時表領域

各ユーザーは、「一時表領域」を持っています。ユーザーが一時セグメントの作成を必要とするような 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われます。すべてのユーザーの一時セグメントを別々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。

表領域の割当て制限

Oracle は、スキーマ内のオブジェクトに対して利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、ユーザーが利用できる表領域ごとに設定できます。表領域の割当て制限によるセキュリティ機能によって、特定のスキーマのオブジェクトが消費するディスク領域の容量を選択的に制御できます。

プロファイルとリソース制限

各ユーザーには、そのユーザーが利用できるいくつかのシステム・リソース制限を指定した「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実行セッションの数
- CPU 処理時間
  - ユーザーのセッションで使用可能な時間
  - SQL 文による Oracle への 1 回のコールで使用可能な時間
- 論理 I/O の総量
  - ユーザーのセッションで利用可能な数
  - SQL 文による Oracle への 1 回のコールで使用可能な数
- ユーザーのセッションで許されるアイドル時間の総量
- ユーザーのセッションに許される接続時間の総量
- パスワード制限
  - 複数のログインが失敗したときにアカウントをロックする機能
  - パスワードの時間切れと猶予期間
  - パスワードの再使用と複雑さに関する制限

データベースの各ユーザーに対して、個別に異なるプロファイルを作成し、割り当てることができます。明示的にプロファイルを割り当てられなかった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・リソースが過度に消費されることを防止します。

監査

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

文監査

文監査は、特定のスキーマ・オブジェクトとは無関係の、特定の SQL 文の監査です。（さらに、データベース・トリガーを使うことによって、データベース管理者は 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に対する接続と接続の切離しを監査できます。

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の、強力なシステム権限の使用の監査です。権限監査では、全ユーザーの広範な監査、または特定ユーザーの集中的な監査ができます。
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・オブジェクトに対する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 文のような、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。これによって、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利用できます。

Trusted Oracle

「 Trusted Oracle 」は、オラクル社のマルチレベルのセキュリティ機能を装備したデータベース管理システム製品です。この製品は、取扱いに注意が必要な情報、または機密情報を扱う企業で必要とされる、高水準のセキュリティ機能を使ってデータを管理できるように設計されています。 Trusted Oracle は、 Oracle の基本製品やアプリケーションと互換性があり、標準の Oracle の機能をすべてサポートします。

さらに、 Trusted Oracle は広範囲にわたるマルチレベル・セキュリティのオペレーティング・システム環境で「必須アクセス制御」（MAC）を施行します。必須アクセス制御は、「ラベル」に基づいて情報へのアクセス制限を制御する手法です。ユーザーのラベルは、ユーザーがアクセスを許可されている情報、およびそのユーザーに許可されているアクセスのタイプ（読込みまたは書込み）を示します。スキーマ・オブジェクトのラベルは、そのオブジェクトに入っている情報の重要度を示します。ラベル付きスキーマ・オブジェクトへのアクセスが許可されるためには、ユーザーのラベルがMACの方針によって決定される一定の基準と合致していなければなりません。このタイプのアクセス制御は、ユーザーがインプリメントする任意制御よりも前に必ず施行されるため、このタイプのセキュリティを「必須」セキュリティと呼びます。

追加情報：詳細は、 Trusted Oracle のマニュアルを参照してください。

データベースのバックアップとリカバリ

この項では、次のことを実現するために Oracle で使われている構造とソフトウェアのメカニズムについて説明します。

- さまざま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回復（リカバリ）
- どのような状況にも対応する柔軟な回復操作

- バックアップとリカバリの処理中のデータの可用性（システムのユーザーが作業を継続できるようにする）

なぜ回復は重要か

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、データベースに影響を及ぼした場合には、データベースを回復しなければなりません。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、回復するデータベースに確実に反映させ、その障害によって引き起こされた問題からユーザーを隔離しながら、できるかぎり迅速に通常の運用状態に復帰させることです。

障害のタイプ

状況によっては、 Oracle データベースの操作が中断されます。最も一般的な障害のタイプについて、次に説明します。

ユーザー・エラー

ユーザー・エラーが発生した場合、エラーが発生する前の状態までデータベースを回復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としましょう。ユーザー・エラーからの回復を可能にし、その他の独特の回復要件を満たすために、 Oracle は厳密な時間管理による回復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回復できます。

文障害とプロセス障害

Oracle プログラム内の文の処理において論理的な障害（たとえば、文が有効な SQL 構造になっていない場合）があるときに文障害が発生します。文障害が発生すると、文による影響（それがある場合）は Oracle によって自動的に取り消され、ユーザーに制御が戻されます。

「プロセス障害」とは、異常な接続の切離し、または異常なプロセス終了など、 Oracle にアクセスしているユーザー・プロセスでの障害です。 Oracle と他のユーザー・プロセスは作業を継続できますが、障害のあったユーザー・プロセスは作業を継続することができません。 Oracle バックグラウンド・プロセス PMON は、障害の発生したユーザー・プロセスを自動的に検出するか、または Net8 からそのような報告を受け取ります。 PMON は、ユーザー・プロセスのコミットされていないトランザクションをロールバックし、このプロセスが使っていたリソースを解放することによって問題を解決します。

SQL 文構造の誤りやユーザー・プロセスの中断など、日常の一般的な問題が原因でデータベース・システムが全体として中断させらるような事態は決して起こりません。さらに、 Oracle は、システムや他のユーザーに与える影響を最小限に抑えながら、コミットされていないトランザクションの変更とロックされているリソースに対して、必要な回復処理を自動的に実行します。

インスタンス障害

インスタンス（システム・グローバル領域とバックグラウンド・プロセス）で作業を継続できないような問題が発生すると、インスタンス障害が発生します。インスタンス障害は、停電などのハードウェア問題、またはオペレーティング・システムのクラッシュなどのソフトウェア問題が原因で発生することがあります。インスタンス障害が発生した場合、システム・グローバル領域のバッファ内のデータは、データ・ファイルに書き込まれません。

インスタンス障害は、「インスタンス回復」を必要とします。インスタンスが再起動されると、インスタンスの回復が Oracle によって自動的に実行されます。インスタンス障害のために失われた、 SGA のデータベース・バッファ内のコミット済みデータを回復するために、 REDO ログが使われます。

媒体（ディスク）障害

データベースを運用する上で必要なファイルに対して読み書きを実行しているときに、エラーが発生することがあります。ディスク上の物理ファイルの読み書きにかかわる物理的な問題が原因となるため、このような障害をディスク障害と呼びます。よく起きる例は、ディスク・ヘッドのクラッシュです。これが発生すると、ディスク・ドライブ上のファイルをすべて失うことになります。データ・ファイル、 REDO ログ・ファイル、および制御ファイルを含む各種ファイルが、このタイプのディスク障害によって影響を受ける可能性があります。また、データベース・インスタンスは正常に機能し続けることができないため、システム・グローバル領域のデータベース・バッファ内のデータをデータ・ファイルに書き込むことができません。

ディスク障害では「媒体回復」が必要です。媒体回復は、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、データベースのデータ・ファイル内の情報がディスク障害の起こる直前の状態になるようにデータ・ファイルを復元します。ディスク障害からの回復を完了するには、データベースのデータ・ファイルのバックアップ、すべてのオンライン REDO ログ・ファイル、および必要なアーカイブ済みの REDO ログ・ファイルが必要です。

Oracle では、ディスク・クラッシュを含み、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、完全で迅速な回復を行うことができます。データベースを完全に回復するか、または部分的に特定の時点まで回復できるように、オプションが用意されています。

ディスク障害のためにいくつかのデータ・ファイルが破損しても、データベースの大部分は損なわれておらず、そのまま運用できるという場合に限って、必要な表領域を個別に回復する間、データベースをオープンしたままの状態にしておくことができます。したがって、データベースの損害を受けていない部分は、破損した部分の回復中も日常業務で使用できます。

回復のために使われる構造

Oracle は、いくつかの構造を使って、インスタンス障害やディスク障害から完全に回復します。すなわち、 REDO ログおよびロールバック・セグメント、制御ファイル、必要なデータベースのバックアップを使います。

REDO ログ

1-11 ページの「 REDO ログ・ファイル」で説明されているとおり、「 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 ログとアーカイブ済み REDO ログの 2 つの部分から構成されます。

オンライン REDO ログ

「オンライン REDO ログ」は、 2 つ以上の「オンライン REDO ログ・ファイル」の集合であり、データベースに対して行われたすべてのコミット済み変更を記録します。トランザクションがコミットされるたびに、システム・グローバル領域の REDO ログ・バッファ内に一時的に格納されている REDO エントリが、対応するオンライン REDO ログ・ファイルにバックグラウンド・プロセス LGWR によって書き込まれます。

オンライン REDO ログ・ファイルは循環方式で使われます。たとえば、 2 つのファイルによってオンライン REDO ログが構成されている場合には、最初のファイルが満杯かつ 2 番目のファイルが満杯になったら最初のファイルを再利用し、そのファイルが満杯になったら 2 番目のファイルを再利用する、というような流れで使われます。ファイルが満杯になるたびに、REDO エントリの集合を識別するために「ログ順序番号」がそのファイルに割り当てられます。

1 つの地点での障害によってデータベースが失われることがないように、 Oracle では複数のオンライン REDO ログ・ファイルのセットを維持できます。「多重化されたオンライン REDO ログ」は、物理的に別々のディスク上に置かれたオンライン REDO ログ・ファイルのいくつかのコピーから構成され、そのグループ内の各メンバーに加えられた変更は、すべてのメンバーに反映されます。

オンライン REDO ログ・ファイルを収録しているディスクで障害が生じても、他のコピーが破壊されることはなく、 Oracle から利用できます。システム操作を中断せずに、破壊されていないコピーを使って、失われたオンライン REDO ログ・ファイルを簡単に回復することができます。

アーカイブ済み REDO ログ

満杯になったオンライン REDO ログ・ファイルは、「アーカイブ済み REDO ログ」を作成して、再利用の前にアーカイブすることもできます。アーカイブ済みREDOログは、「アーカイブ済み（オフライン） REDO ログ・ファイル」によって構成されます。

アーカイブ済み REDO ログの存在の有無は、 REDO ログが使っているモードによって決まります。

ARCHIVELOG	満杯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	満杯になったオンライン REDO ログ・ファイルはアーカイブされません。

ARCHIVELOG モードでは、インスタンス障害とディスク障害の両方から、データベースを完全に回復できます。また、データベースがオープンされており、使用可能な状態になっている時でも、データベースをバックアップできます。ただし、アーカイブ済み REDO ログのメンテナンスのために、余分な管理作業が必要になります。

データベースの REDO ログを NOARCHIVELOG モードで運用している場合、データベースをインスタンス障害から完全に回復することはできますが、ディスク障害からは回復できません。さらに、データベースが完全にクローズされている間だけ、データベースをバックアップできます。アーカイブ済み REDO ログがまったく作成されないため、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。

制御ファイル

データベースの「制御ファイル」は、主に、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現行のログ順序番号についての情報を保管します。通常の回復手順では、回復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います。

多重制御ファイル

これは、多重化された REDO ログの機能と同様の機能です。 Oracle によって同じ制御ファイルが複数個維持され、そのすべてが同時に更新されます。

ロールバック・セグメント

1-9ページの「データ・ブロック、エクステント、セグメント」で説明されているように、ロールバック・セグメントには、 Oracle のいくつかの機能で使われるロールバック情報が記録されます。データベースの回復では、 REDO ログに記録された変更がすべて適用された後、Oracle はロールバック・セグメントの情報を使って、コミットされていないトランザクションを取り消します。ロールバック・セグメントはデータベース・バッファ内に格納されるため、この重要な回復情報は REDO ログによって自動的に保護されます。

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、媒体回復では、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復元する必要があります。データベースのファイルをバックアップする方法は次の 2 つに分類されます。

全体データベース・バックアップ

全体データベース・バックアップは、 Oracle データベースを構成する、すべてのデータ・ファイル、オンライン REDO ログ・ファイル、および制御ファイルのオペレーティング・システム・バックアップです。データベースをクローズして使用できない状態にしてあるときに、全体バックアップを実行します。

部分バックアップ

部分バックアップは、データベースの一部のオペレーティング・システム・バックアップです。部分バックアップには、個々の表領域のデータ・ファイルのバックアップや、制御ファイルのバックアップなどがあります。データベースの REDO ログが ARCHIVELOG モードで運用されている場合に限り、部分バックアップが有効です。

バックアップ計画に応じて、さまざまな部分バックアップを作成できます。データベースがオープンされているとき、またはクローズされているとき、特定の表領域がオンラインになっているとき、またはオフラインになっているとき、いろいろな組合せでデータ・ファイルと制御ファイルをバックアップできます。 REDO ログが ARCHIVELOG モードで運用されている場合、 REDO ログの追加バックアップは必要ありません。つまり、アーカイブ済み REDO ログは、満杯になったオンライン REDO ログ・ファイルのバックアップです。

基本的な回復手順

DBWn がデータ・ファイルにデータベース・バッファを書き込む方法のために、ある時点では、コミットされていないトランザクションで仮に変更されたデータ・ブロックのいくつかがデータ・ファイルに含まれている可能性があります。また、コミットされたトランザクションにより変更されたいくつかのブロックが、データ・ファイルに含まれていない可能性もあります。したがって、障害の発生後には次のような 2 つの状態が考えられます。

- コミット済みの変更を含んでいるブロックが、データ・ファイルに書き込まれておらず、その変更は REDO ログだけに存在する。したがって、 REDO ログには、データ・ファイルに適用が必要なコミット済みデータが含まれています。

- REDO ログには、コミットされていないデータが含まれている可能性があるため、REDO ログが回復中に適用した変更のうちコミットされていないトランザクションの変更は、データ・ファイルから消去しなければならない。

このような状態を解決するために、インスタンス障害や媒体障害からの回復では、ロールフォワードとロールバックという 2 つのステップが必ず使われます。

ロールフォワード

回復の最初のステップは「ロールフォワード」です。これによって、 REDO ログに記録されたすべての変更がデータ・ファイルに再適用されます。ロールフォワードは、必要なすべての REDO ログ・ファイルを使って、必要な時点までデータ・ファイルの内容を戻します。

Oracle は、必要な REDO 情報がすべてオンラインであれば、データベースの起動時にこの回復ステップを自動的に実行します。ロールフォワードが行われた後、データ・ファイルには、すべてのコミット済みの変更と REDO ログに記録されたコミットされていない変更が含まれています。

ロールバック

ロールフォワードは、回復処理の半分にすぎません。ロールフォワードの実行後、コミットされていない変更をすべて取り消す必要があります。 REDO ログ・ファイルが適用された後、コミットされなかったのに REDO ログに記録されていたトランザクションを識別して取り消すために、ロールバック・セグメントを使います。このプロセスを「ロールバック」といいます。 Oracle はこのステップを自動的に完了します。

Recovery Manager

Recovery Manager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・ファイルのバックアップを作成したり、バックアップを使ってデータベースを復元（リストア）または回復（リカバリ）したりします。

RecoveryManager は、「リカバリ・カタログ」というリポジトリを維持しています。リカバリ・カタログには、バックアップ・ファイルとアーカイブ済みログ・ファイルに関する情報が入ります。Recovery Managerは、このリカバリ・カタログを使って、復元操作と媒体回復の両方を自動化します。

リカバリ・カタログには、次の情報が入っています。

- データ・ファイルとアーカイブ・ログのバックアップに関する情報
- データ・ファイル・コピーに関する情報
- アーカイブ済み REDO ログとそれらのログのコピーに関する情報
- ターゲット・データベースの物理スキーマに関する情報
- 「ストアド・スクリプト」と呼ばれる名前付きのコマンド・シーケンス

追加情報: Recovery Managerの詳細は、『Oracle8 Serverバックアップおよびリカバリ』を参照してください。

データベース管理のオブジェクト・リレーショナル・モデル

データベース管理システムは、階層型からネットワークへ、そしてネットワークからリレーショナル・モデルへと発展してきました。主流になっているデータベース・モデルは、「リレーショナル・モデル」です。 Oracle は、リレーショナル・モデルをさらに「オブジェクト・リレーショナル・モデル」へと拡張しています。オブジェクト・リレーショナル・モデルでは、複雑な業務モデルをリレーショナル・データベースに格納できます。

リレーショナル・モデル

リレーショナル・モデルには、次のような 3 つの主要な側面があります。

構造体
: 「構造体」とは、データベースのデータを格納またはアクセスするための、正しく定義されたオブジェクト（表、ビュー、索引など）のことです。構造体とそこに含まれるデータは、「操作」により処理されます。

操作
: 「操作」とは、ユーザーがデータベースのデータや構造を処理するための、明確に定義されたアクションのことです。データベースの操作は、事前に定義された「整合性規則」に従っていなければなりません。

整合性規則
: 「整合性規則」とは、データベースのデータや構造に対してどの操作を実行できるかを決める規則です。この規則により、データベースのデータや構造は保護されます。

リレーショナル・データベース管理システムには、次のような利点があります。

- 物理データ記憶域と論理データベース構造が独立している
- すべてのデータに柔軟に、また簡単にアクセスできる
- データベース設計に柔軟性を持たせることができる
- データの記憶域と冗長性を軽減できる

オブジェクト・リレーショナル・モデル

オブジェクト・リレーショナル・モデルでは、ユーザーは「オブジェクト型」を定義できます。オブジェクト型には、データの構造と、データを操作するメソッドの両方を指定します。そして、そのようなデータ型をリレーショナル・モデルの中で使うことができます。

注意：ユーザー定義オブジェクト型は、オブジェクト・オプション付きの Oracle8 Enterprise Edition を購入した場合にのみ利用できます。 Oracle8 Enterprise Edition で使用可能な機能とオプションの詳細は、『 Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

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（たとえば、注文）を抽象化したものです。オブジェクト型は、次の 3 種類の要素から構成されます。

- 「名前」。これは、オブジェクト型を一意に識別するものです。
- 「属性」。これは、ビルトイン・データ型、またはその他のユーザー定義型です。属性は、実社会のエンティティ（実体）の構造をモデル化します。
- 「メソッド」。これは、 PL/SQL で作成されてデータベースに格納されたファンクションまたはプロシージャ、もしくは、 C などの言語で作成されて外部に格納されたファンクション（関数）またはプロシージャのことです。メソッドは、アプリケーションがデータに対して実行できる特定の操作をインプリメントします。すべてのオブジェクト型には、データ型の仕様に基づいて新しいオブジェクトを作成するための「コンストラクタ・メソッド」があります。

スキーマとスキーマ・オブジェクト

「スキーマ」は、ユーザーが使用可能なデータベース・オブジェクトの集合です。「スキーマ・オブジェクト」は、データベースのデータを直接参照する論理構造です。スキーマ・オブジェクトには、表、ビュー、順序、ストアド・プロシージャ、シノニム、索引、クラスタ、データベース・リンクなどの構造体が含まれます。（表領域とスキーマには何の関連もありません。同じスキーマ内のオブジェクトが別々の表領域に存在したり、異なるスキーマにあるオブジェクトが 1 つの表領域に保持されたりすることがあります。）

表

「表」は、Oracleデータベースにおけるデータ格納の基本単位です。データベースの表には、ユーザーがアクセスできるすべてのデータが保持されています。

表データは「行」と「列」に格納されます。すべての表は、「表名」と列の集合によって定義します。各列には、「列名」および「データ型」（CHAR、 DATE、 NUMBERなど）、「幅」（DATEのように、データ型によって事前に定義されていることもある）または「位取り」と「精度」（ NUMBER データ型の場合のみ）を指定します。表を作成したら、その中に有効なデータ行を挿入していくことができます。表の行に対しては、問合せまたは削除、更新を実行できます。

Oracle8では、表を「パーティション化」できます。詳細は、第9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。

表のデータに対して定義されたビジネス・ルール（業務規則）を施行するために、整合性制約とトリガーを表に対して定義できます。詳細は、 1-52 ページの「データの整合性」を参照してください。

ビュー

「ビュー」とは、1つ以上の表のデータをユーザーの必要に合せて調整したデータ表現です。ビューは、「格納された問合せ」とみなすこともできます。

ビューの中に実際にデータが格納されているわけではありません。データは、ビューの基礎となっている表から導出されます。これらの表をビューの「実表」といいます。実表は表のときもあれば、それ自体がビューのときもあります。

表の場合と同じように、ビューに対しても、いくつかの制限付きで問合せおよび更新、挿入、削除を実行できます。ビューに対して実行する操作は、すべて実表にも影響を与えます。

ビューは、次のような目的でよく使われます。

- 事前に定義された行と列の集合だけにアクセスを限定することにより、表のセキュリティ・レベルを強化する。たとえば、表のビューを作成するときに、機密性の高いデータ（給与情報など）が格納されている列はビューに含めないように定義できます。
- データの複雑さを隠す。たとえば、 12 個ある月別売上表を組み合せた 1 つのビューを作成して、年間データを分析し報告できます。また、複数の表から関連する列や行を「結合」して表示するビューを作成することもできます。しかも、データが実際にはいくつもの表から取られていることを感じさせずに表示できます。
- ユーザーのコマンドを単純化する。たとえば、ユーザーが相関副問合せについて知らなくても、複数の表から情報を選択できるようにします。
- 実表とは異なる視点からデータを提示する。たとえば、実表の列名を変更せずに、ビューで列名を改名できます。
- 複雑な問合せを保存する。たとえば、ある問合せで表の情報に対して膨大な計算を実行したとします。この問合せをビューとして保存しておけば、そのビューに問合せを行うときだけ同じ計算が実行されます。

2 つ以上の表の結合（複数の表からデータを選択する SELECT 文）を実行するビューは、特定の条件の下でしか更新できません。詳細は、 8-13 ページの「更新可能な結合ビュー」を参照してください。

順序

「順序」は、データベース表の数値列について、連続した一意の数値のリストを生成します。順序を使うと、1つの表または複数の表の行に対して一意の数値が自動的に生成されるので、アプリケーションのプログラミングが簡単になります。

たとえば、 2 人のユーザーが EMP 表に新しい従業員の行を同時に挿入しているとします。順序を使ってEMPNO列に一意の従業員番号を生成すると、一方のユーザーは、もう一方のユーザーが従業員番号を入力するのを待つ必要はありません。どちらのユーザーにも、順序により、有効な値が自動的に生成されるからです。

順序番号は表とは独立しているので、同じ順序を 1 つ以上の表に対して指定できます。作成された順序は、さまざまなユーザーがアクセスしたとき、実際の順序番号を生成します。

プログラム・ユニット

このマニュアルに出てくる「プログラム・ユニット」という用語は、ストアド・プロシージャ、ファンクション、パッケージ、トリガー、無名ブロックを指します。

「プロシージャ」または「ファンクション」は、 SQL および PL/SQL （ SQL に対する Oracle のプロシージャ型言語拡張機能）文を、特定の作業を行う実行単位としてグループ化したものです。SQLとPL/SQLの詳細は、1-47ページの「データ・アクセス」を参照してください。

プロシージャとファンクションを使うと、 SQL が持つ平易さや柔軟性と、構造型プログラミング言語が持つプロシージャ的な機能性を合体できます。PL/SQLを使って、そのようなプロシージャやファンクションを定義し、継続して使えるようにデータベースに格納します。プロシージャとファンクションはほとんど同じですが、唯一異なる点として、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 つの値を戻します。

「パッケージ」は、関連したプロシージャおよびファンクション、その他のパッケージ構成体をカプセル化し、データベースの単位としてまとめて格納するための手段です。パッケージにより、データベース管理者やアプリケーション開発者が物事を組織しやすくなるだけでなく、データベースの機能性やパフォーマンスも向上します。

シノニム

「シノニム」とは、表またはビュー、順序、プログラム・ユニットの別名のことです。シノニムはスキーマ・オブジェクトそのものではなく、スキーマ・オブジェクトへの直接の参照です。次のような目的で使います。

- スキーマ・オブジェクトの実名と所有者を隠す
- スキーマ・オブジェクトに対するパブリックなアクセスを実現する
- リモート・データベースの表、ビュー、またはプログラム・ユニットについて位置の透過性を実現する
- データベース・ユーザー側の SQL 文を単純にする

シノニムは、パブリックまたはプライベートのどちらにもできます。個々のユーザーは、当人だけが使える「プライベート・シノニム」を作成できます。「パブリック・シノニム」は、多くの場合、データベース管理者によって作成されます。パブリック・シノニムにより、基礎となるスキーマ・オブジェクトをすべてのデータベース・ユーザーが一般的な方法で、またシステム全体で使えるようになります。

索引、クラスタ、ハッシュ・クラスタ

索引、クラスタ、ハッシュ・クラスタは、表と対応付けられるオプションの構造体です。データ検索のパフォーマンスを向上させるために作成します。

「索引」は、データ検索のパフォーマンスを向上させるために作成します。このマニュアルでも、索引を使う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 の索引を使うと表データに高速にアクセスできます。 1 つの要求を処理するとき、Oracle は、要求された行を効率よく見つけるために使用可能な索引をいくつか（あるいはすべて）使います。索引が有効なのは、アプリケーションが表内のある範囲の行（たとえば、給与が 1000 ドルを超えるすべての従業員）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、行の削除など）は、関係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（意識させません）。

索引は、論理的にも物理的にもデータから独立しています。表や他の索引に影響を及ぼすことなく、いつでも削除したり作成したりできます。索引を削除してもアプリケーションはすべて機能しつづけますが、それまで索引が設定されていたデータに対するアクセスは遅くなります。

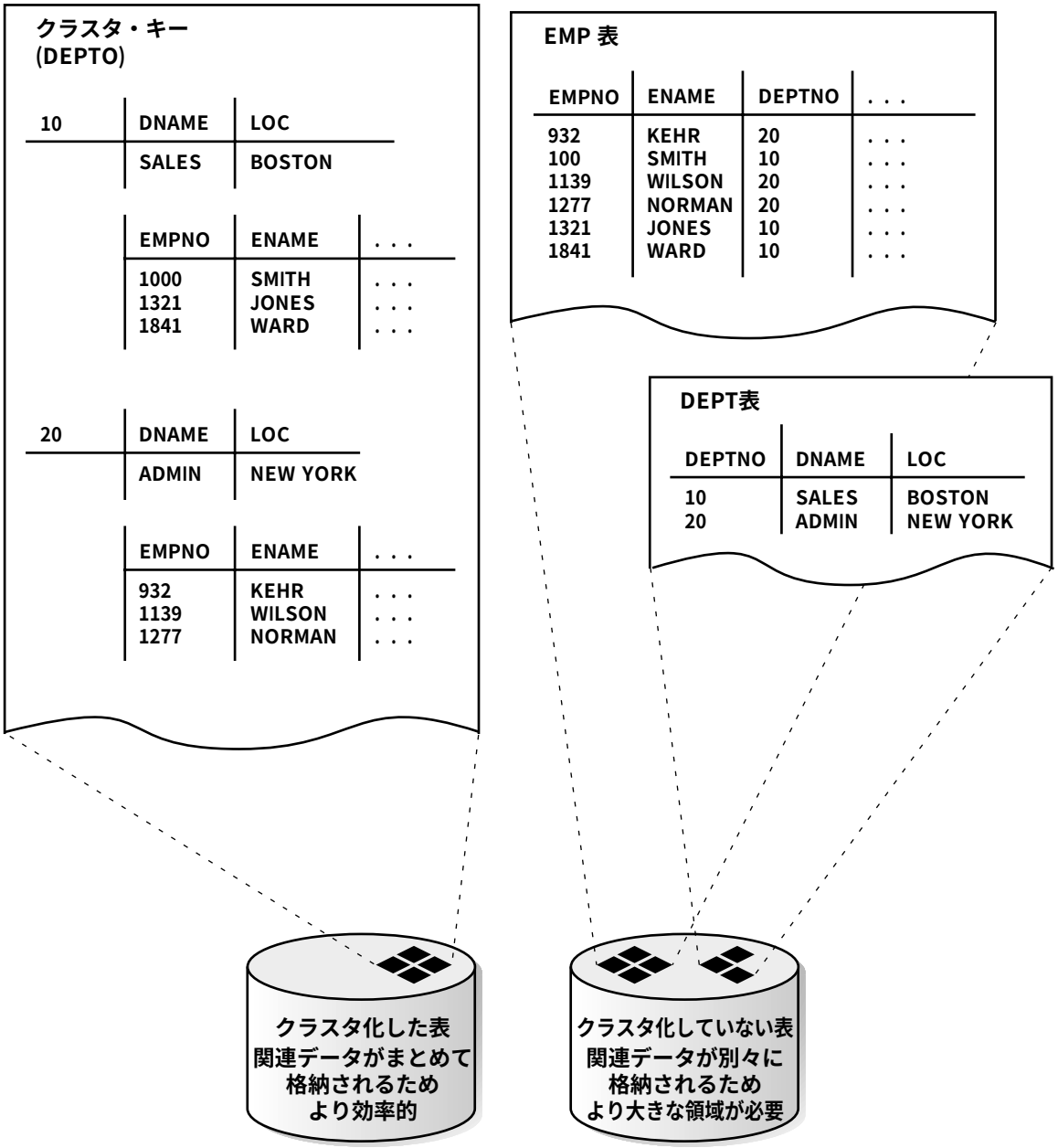
Oracle8 では、索引の「パーティション化」を行うことができます。詳細は、第 9 章「パーティション表とパーティション索引」を参照してください。

「クラスタ」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いるか、しばしば一緒に使われるため、まとめて格納されます。関連する行が物理的にまとめて格納されているため、ディスク・アクセス時間が短縮されます。

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

図 1-5 に、クラスタ化されたデータとクラスタ化されてないデータがどのように物理的に格納されるかを示します。

図 1-5 クラスタ化された表とクラスタ化されていない表



データの分散状況や、データに対して最も頻繁に実行される SQL 操作にもよりますが、クラスタを使うとデータ検索のパフォーマンスも向上します。特に、クラスタ化された表を結合して問い合せる場合、結合された表に共通の行は同じI/O操作で検索されるので、クラスタを使うことに利点があります。

索引と同じように、クラスタがアプリケーションの設計に影響することはありません。表がクラスタの一部になっているかどうかは、ユーザーとアプリケーションに対して透過的です。クラスタ化された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 を使ってアクセスします。

「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）の場合と同じような方法で表データをクラスタ化します。ただし、行は、行のクラスタ・キー値に「ハッシュ関数」を適用した結果に基づいてハッシュ・クラスタに格納されます。キーの値が同じすべての行は、ディスク上にまとめて格納されます。

等式を条件にした問合せを使って表を問い合せる（たとえば、部門が 10 のすべての行を戻す）ことが多い場合、索引表または索引クラスタよりも、ハッシュ・クラスタの方が適切です。ハッシュ・クラスタを使う問合せでは、指定されたクラスタ・キー値がハッシュ化されます。結果として生成されるハッシュ・キー値は、指定された行が格納されているディスク領域を直接指します。

データベース・リンク

「データベース・リンク」とは、あるデータベースから別のデータベースへの「パス」を記述する名前付きスキーマ・オブジェクトです。分散データベースで「グローバル・オブジェクト名」が参照されると、データベース・リンクが暗黙に使われます。詳細は、1-24ページの「分散データベース」および第 30 章「分散データベース」を参照してください。

データ・ディクショナリ

どの Oracle データベースにも、「データ・ディクショナリ」が 1 つずつあります。 Oracle データ・ディクショナリは、データベースの「読込み専用」の参照として使われる、表およびビューのセットです。たとえば、データ・ディクショナリには、データベースの論理構造と物理構造の両方に関する情報が格納されます。このような重要な情報の他にも、データ・ディクショナリには次のような情報が格納されます。

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

データ・ディクショナリは、データベースの作成時に作成されます。常にデータベースの正確な状態が反映されるよう、データ・ディクショナリは、特定のアクション（データベース構造の変更など）が実行されるたびに、 Oracle により自動的に更新されます。データベースを運用するうえで、データ・ディクショナリは重要な役割を果たします。進行する作業の記録および検証、指示を行うからです。たとえば、データベースの操作中、 Oracle は、スキーマ・オブジェクトが存在していること、またユーザーが適切なアクセス権を持っていること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。

データ・アクセス

この項では、次のような DBMS の一般要件に Oracle がどのように応えているかを説明します。

- データ・アクセス言語として業界で受け入れられている規格を遵守する。
- データの操作中にデータベース情報の一貫性を制御し、維持する。
- データベース情報の整合性を維持する規則を定義、施行するためのシステムを提供する。
- 高パフォーマンスを実現する。

SQL － 構造化問合せ言語

「 SQL 」は、簡単かつ強力なデータベース・アクセス言語であり、 RDBMS （リレーショナル・データベース管理システム）の標準言語です。オラクル社が Oracle 向けにインプリメントした SQL は、 ANSI/ISO 標準規格の SQL データ言語に 100 パーセント準拠しています。

SQL 文

Oracle データベース内の情報の操作は、すべて SQL 文を使って実行されます。 SQL 文とは、 Oracle に与えられ、 Oracle によって実行される SQL テキスト文字列です。次に示すように、文は、完全な SQL センテンスと等価でなければなりません。

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

実行できるのは完全な SQL 文だけです。次のような「センテンスの断片」を実行しようとすると、 SQL 文を実行するにはテキストが不足していることを示すエラーが発生します。

```
SELECT ename
```

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

データ定義文（DDL）

「DDL 文」は、スキーマ・オブジェクトを定義およびメンテナンスしたり、必要でなくなったスキーマ・オブジェクトを削除したりします。 DDL 文には、データベースやデータベース内の特定のオブジェクトにアクセスする「権限」、つまり権利を、あるユーザーから他のユーザーに付与するための文も含まれます。（1-27ページの「データベース・セキュリティ」を参照してください。）

データ操作文（DML）

「DML 文」は、データベースのデータを操作します。たとえば、表の行の問合せ、挿入、更新、削除はすべて DML 操作です。また、表やビューのロック、 SQL 文の実行計画の検査も DML 操作です。

トランザクション制御文

「トランザクション制御」文は、DML文によって行われる変更を管理します。これによって、ユーザーやアプリケーション開発者はいくつかの変更をグループ化して、論理トランザクションを作成できます。（1-48 ページの「トランザクション」を参照してください。）この文に含まれるのは、 COMMIT 、 ROLLBACK 、 SAVEPOINT などです。

セッション制御文

「セッション制御」文によって、ユーザーは自分の現行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、言語設定を変更したりできます。セッション制御文には ALTER SESSION と SET ROLE の 2 つがあります。

システム制御文

システム制御コマンドは、Oracle Serverインスタンスのプロパティを変更します。システム制御コマンドはALTER SYSTEMだけです。このコマンドは、共有サーバーの最小数などの設定の変更、およびセッションの停止、その他の作業を実行します。

埋込み SQL 文

埋込み SQL 文は、 DDL 文および DML 文、トランザクション制御文をプロシージャ型言語プログラム（Oracleプリコンパイラとともに使われるプログラムなど）に組み込んだものです。たとえば、 OPEN および CLOSE 、 FETCH 、 EXECUTE などが含まれます。

トランザクション

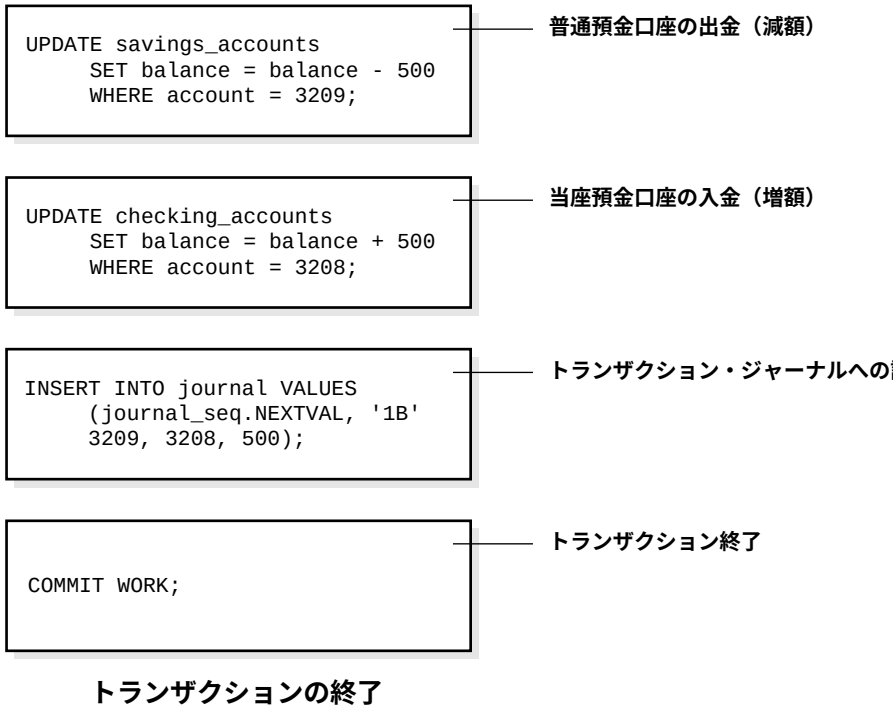
「トランザクション」は、単一のユーザーによって実行される 1 つ以上の SQL 文を含む論理作業単位です。 Oracleが互換性を持つANSI/ISO SQL規格によれば、トランザクションはユーザーの最初の実行可能 SQL 文で始まります。トランザクションは、ユーザーによって明示的にコミットまたはロールバックされるときに終了します（コミットとロールバックについては後述します）。

銀行のデータベースを考えてみます。顧客が普通預金口座から当座預金口座へ預金を移動するとき、トランザクションは、 3 つの別々の操作、つまり普通預金口座の減額および当座預金口座の増額、トランザクション・ジャーナルへのトランザクション記録を実行します。

Oracle は、 3 つの SQL 文をすべて実行することによって口座の差引勘定が正しく維持されることを保証しなければなりません。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、トランザクションの他の文の実行を取り消さなければなりません。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 1-6 に銀行のトランザクションの例を示します。

図 1-6 銀行のトランザクション



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 文を実行した結果の変更が確定します。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だけです。

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 文を実行した結果の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た場合、トランザクション内の SQL 文が実行されなかったかのように、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

多数の SQL 文を含む大規模なトランザクションでは、中間の目印、すなわち「セーブポイント」を宣言できます。「セーブポイント」は、トランザクションをより小さな部分に分割するために使います。

セーブポイントを使って、大規模なトランザクション内の任意の位置で、作業を記録できます。これにより、トランザクションの現在の位置から、トランザクション内で宣言したある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することができます。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ったデータの一貫性

トランザクション内の SQL 文が論理的にグループ化される限り、トランザクションを使って、データベース・ユーザーやアプリケーション開発者はデータへの一貫した変更を保証できます。トランザクションには、 1 つの論理作業単位に必要な部分をすべて含める必要があります。余分も不足もあってはなりません。トランザクションの開始から終了まで、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。トランザクションは、データに対して 1 つの一貫性のある変更を行う SQL 文だけで構成するべきです。

たとえば、銀行の例を思い出してください。2つの口座間の送金（トランザクション）には、一方の口座の預金高を増やすアクション（ 1 つの SQL 文）ともう一方の口座の預金高を減らすアクション（1つの SQL 文)、トランザクション・ジャーナルに記録するアクション（1つの SQL 文）が必要です。これらのアクションは、すべて失敗するか、すべて成功するかのどちらかでなければなりません。つまり、借方が存在しないのに貸方が成立することはあり得ません。一方の口座に新しく預金するといった関係のないアクションは、この送金トランザクションの中で実行するべきではありません。そのような文は、別のトランザクションに組み込みます。

PL/SQL

「PL/SQL 」は、 SQL に対する Oracle のプロシージャ型言語拡張機能です。 PL/SQL は、 SQL の持つ簡易性と柔軟性を、 IF...THEN 、 WHILE 、 LOOP などの構造化プログラミング言語のプロシージャ的な機能と一体化します。

データベース・アプリケーションを設計するときに、開発者はストアドPL/SQLを使うことの次のような利点を検討するとよいでしょう。

- PL/SQL コードはデータベースに集中的に格納できるため、アプリケーションとデータベースとの間のネットワーク通信量が低減され、アプリケーションとシステムのパフォーマンスが向上する。

- データ・アクセスをストアド PL/SQL コードによって制御できる。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の開発者が意図したとおりにだけ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る。

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は減少します。

この後の項では、データベース内に定義し、一括して格納できるさまざまなプログラム・ユニットについて説明します。

プロシージャとファンクション

「プロシージャ」と「ファンクション」を構成する一連の SQL 文および PL/SQL 文は、特定の問題を解決したり一連の関連タスクを実行したりすることを目的として、 1 つの単位としてグループ化されています。プロシージャは、作成後にコンパイル済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。プロシージャはコール元に値を戻さないのに対して、ファンクションはコール元に必ず 1 つの値を戻します。それ以外の点は同じです。

パッケージ

「パッケージ」によって、関連したプロシージャおよびファンクション、変数、その他のパッケージ構成体をカプセル化し、データベースの単位としてまとめて格納できます。パッケージによって、管理者やアプリケーション開発者がこのようなルーチンを組織化できる一方で、高機能（たとえば、グローバル・パッケージ変数を宣言し、パッケージ内のプロシージャで使用できる）と高パフォーマンス（たとえば、パッケージの全オブジェクトを一度に解析、コンパイルし、メモリーにロードする）が実現されます。

データベース・トリガー

Oracle では、表に対する挿入または更新、削除の結果として、自動的に実行されるプロシージャを作成できます。これらのプロシージャを「データベース・トリガー」と呼びます。

データベースの情報管理のために、トリガーをいろいろな方法で使えます。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施行、複雑なセキュリティ許可のカスタマイズのために使えます。

メソッド

「メソッド」とは、ユーザー定義データ型（オブジェクト型またはネストした表、変数配列）の一部であるプロシージャまたはファンクションのことです。

注意: ユーザー定義データ型は、オブジェクト・オプション付きの Oracle8 Enterprise Edition を購入した場合にのみ利用できます。 Oracle8 Enterprise Edition の詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

メソッドは、次の 2 つの点でストアド・プロシージャと異なります。

- メソッドは、対応付けられた型のオブジェクトを参照することによって起動する。
- メソッドは、対応付けられたオブジェクトが持つ属性と、その型についての情報に完全にアクセスできる。

すべてのユーザー定義データ型には、データ型の仕様に基づいて新しいオブジェクトを作成するためのシステム定義の「コンストラクタ・メソッド」があります。コンストラクタ・メソッドの名前は、ユーザー定義の型の名前と同じです。オブジェクト型の場合、コンストラクタ・メソッドのパラメータの名前と型は、オブジェクト型の属性の名前と型と同じです。コンストラクタ・メソッドは、新しいオブジェクトを値として戻すファンクションです。ネストした表および配列にも、コンストラクタ・メソッドがあります。

「比較メソッド」は、特定のオブジェクト型に属するオブジェクト間の順序関係を定義します。「マップ・メソッド」では、ビルトイン型を比較するOracleの機能を使います。たとえば、「四角形」という名のオブジェクト型に「高さ」と「幅」の属性があり、その四角形の「高さ」属性と「幅」属性の積を示す数値を戻すマップ・メソッド「面積」が定義されていれば、Oracle は 2 つの四角形の面積を比較することによって、それら 2 つの四角形を比較できます。「順序メソッド」は、独自の内部論理を使用して、特定のオブジェクト型に属する 2 つのオブジェクトを比較します。それは、順序関係を符号化した値を戻します。たとえば、最初のオブジェクトのほうが小さければ -1 を戻し、どちらも同じ大きさであれば 0 を戻し、最初のオブジェクトのほうが大きければ 1 を戻す、などです。

データの整合性

データがデータベース管理者やアプリケーション開発者によって決められたとおり、あるビジネス・ルールを遵守することを保証するのは非常に重要です。たとえば、ビジネス・ルールによって、 INVENTORY 表の SALE_DISCOUNT 列には 9 よりも大きな数値を入れないように規定されているとします。 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、データベースのデータ整合性規則を管理するための解決策として、整合性制約とトリガーを提供します。

整合性制約

「整合性制約」は、表の列に対してビジネス・ルールを定義する宣言の方法です。整合性制約は表のデータに関する文であり、常に次のように機能します。

- 整合性制約を表に対して作成した場合に、いくつかの既存の表データがその制約を満たしていないと、その制約は施行できない。
- 制約を定義された後、 DML 文の結果が整合性制約に違反する場合、その文はロールバックされ、エラーが戻される。

整合性制約は表に定義され、表の定義の一部として集中的にデータベースのデータ・ディクショナリに格納されるため、すべてのデータベース・アプリケーションは同じ一連の規則を遵守しなければなりません。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ので、アプリケーションごとに何度も変更する必要はありません。

Oracle がサポートする整合性制約は次のとおりです。

NOT NULL	表の列に NULL （空の入力）を許可しない。
UNIQUE	列（または列の集合）に重複値を許可しない。
PRIMARY KEY	列（または列の集合）に重複値と NULL を許可しない。
FOREIGN KEY	列（または列の集合）の値が、関連する表の UNIQUE またはPRIMARYKEYの値と一致している必要がある。（また、 FOREIGN KEY 整合性制約は、参照データが変更された場合に依存データを処理する方法を Oracle に指示する、参照整合性アクションも定義します。）
CHECK	制約の論理式を満たさない値を許可しない。

キー

「キー」という語は、いくつかのタイプの整合性制約の定義で使われます。「キー」は、特定のタイプの整合性制約の定義に含まれる列または列の集合です。キーは、リレーショナル・データベースの別々の表と列の間の関連を記述するもので、 次のようなタイプがあります。

主キー	表の PRIMARY KEY 制約の定義に含まれる列（または列の集合）です。主キーの値は、表内の各行を一意に識別します。主キーは、表あたり 1 つだけ定義できます。
一意キー	UNIQUE 制約の定義に含まれる列（または列の集合）です。
外部キー	参照整合性制約の定義に含まれる列（または列の集合）です。
参照キー	同じ表または別の表の一意キーまたは主キーで、外部キーによって参照されるキー。

キーの中の個々の値を、「キー値」と呼びます。

データベース・トリガー

集中的なアクションは、データベース・トリガーでPL/SQLコードを使って定義できます（整合性制約のように宣言するわけではありません)。「データベース・トリガー」は、対応付けられた表に対して INSERT または UPDATE 、 DELETE 文が発行されたときに起動される（暗黙的に実行される）ストアド・プロシージャです。データベース・トリガーは、値ベースの監査、および複雑なセキュリティ・チェックと整合性規則の施行などの機能とともに使って、データベース管理システムをカスタマイズできます。たとえば、通常の就業時間内に限って表を修正できるように、データベース・トリガーを作成できます。

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施行することはできますが、データベース・トリガーが整合性制約と同じ機能を持つわけではありません。特に、整合性規則を施行するために定義されたデータベース・トリガーは、すでに表にロードされているデータをチェックしません。したがって、整合性制約によって整合性規則を施行できない場合にだけ、データベース・トリガーを使うことを強くお薦めします。

第 II 部

データベースの構造

第 II 部では、 Oracle Server 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、物理的また論理的な記憶構造を取り上げます。第 II 部には、次の章が含まれています。

- 第 2 章「データ・ブロック、エクステント、セグメント」
- 第 3 章「表領域とデータ・ファイル」
- 第 4 章「データ・ディクショナリ」

注意: Oracle Server の構造に関するその他の情報は、次の章を参照してください。

- 第 6 章「メモリー構造」
- 第 7 章「プロセスの構造」
- 第 8 章「スキーマ・オブジェクト」
- 第 9 章「パーティション表とパーティション索引」

2

データ・ブロック、エクステント、セグメント

He was not merely a chip of the old block, but the old block itself.

Edmund Burke: *On Pitt's first speech*

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

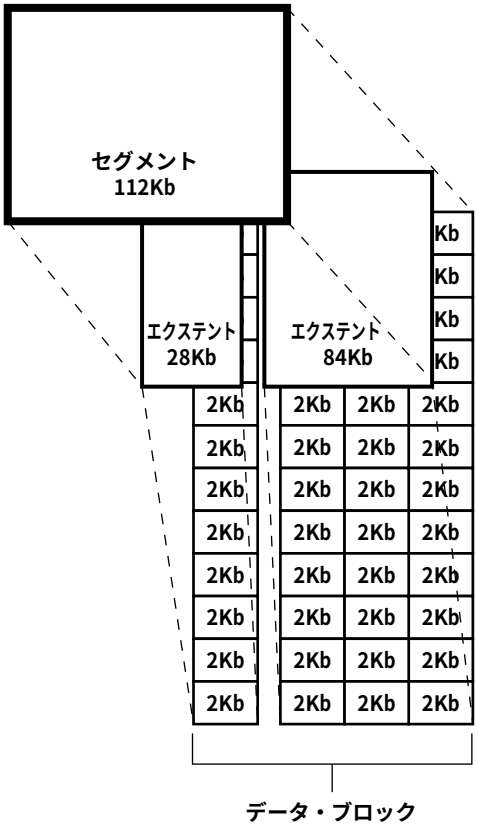
- データ・ブロック、エクステント、セグメントの関係
- データ・ブロック
- エクステント
- セグメント

追加情報: Trusted Oracle を使っている環境での記憶領域の詳細は、 Trusted Oracle のマニュアルを参照してください。

データ・ブロック、エクステント、セグメントの関係

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・ブロックおよびエクステント、セグメントです。次の図は、これらのデータ構造の間の関係を示しています。

図 2-1 **セグメント、エクステント、データ・ブロックの関係**



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

論理的なデータベース領域の次のレベルは、「エクステント」と呼ばれます。エクステントは、特定のタイプの情報を格納するために割り当てられる、ある決まった数の連続したデータ・ブロックです。

エクステントの上位に位置する論理的なデータベース記憶領域のレベルは、「セグメント」と呼ばれます。セグメントは、特定の型のデータ構造に割り当てられるエクステントの集合です。たとえば、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満杯であれば、そのセグメントには別のエクステントが割り当てられます。エクステントは必要に応じて割り当てられるため、 1 つのセグメントの各エクステントはディスク上で連続している場合と連続していない場合があります。

1つのセグメント（および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。表領域内で、1つのセグメントは複数のデータ・ファイルにまたがる（複数のファイルから、データを含むエクステントを持つ）ことがあります。しかし、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・ファイルからのものだけです。

データ・ブロック

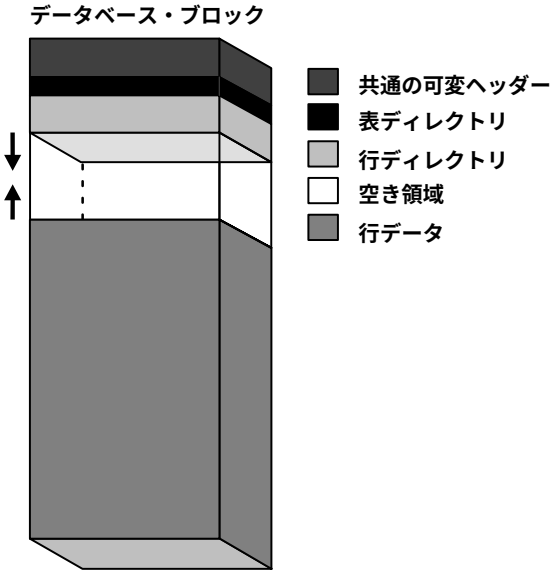
Oracle では、データベースのデータ・ファイル内の記憶領域は、「データ・ブロック」と呼ばれる単位で管理されます。データ・ブロックは、データベースで使われる I/O の最小単位です。これとは対照的に、物理的なオペレーティング・システム・レベルでは、すべてのデータはバイト単位で格納されます。各オペレーティング・システムには、「ブロック・サイズ」と呼ばれる単位があります。 Oracle では、データはオペレーティング・システム・ブロックではなく、 Oracle データ・ブロックの倍数を単位としていなければなりません。

どの Oracle データベースについても、その作成時にデータ・ブロックのサイズを設定します。このデータ・ブロック・サイズは、不必要なI/Oを避けるための最大値（ポートに固有の値）の範囲内で、オペレーティング・システムのブロック・サイズの倍数でなければなりません。 Oracle データ・ブロックは、 Oracle が使ったり割り当てたりできる最小の格納単位です。

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・ブロックに表または索引、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 に、データ・ブロックの形式を示します。

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、索引、ロールバック）などの、一般的なブロック情報が収録されます。

表ディレクトリ

データ・ブロックのこの部分には、このブロック内に行が入っている表についての情報が収録されます。

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。

いったんデータ・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、行が削除されてもこの領域は再利用されません。したがって、現在は空き状態でも、ある時点では最大 50 行が入っていたブロックでは、ヘッダー内の行ディレクトリに 100 バイトが割り当てられたままになっています。この領域は、新しい行がブロックに挿入されるときに限り再利用されます。

オーバーヘッド

データ・ブロック・ヘッダーおよび表ディレクトリ、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可能性があります。 2-9 ページの「行連鎖と移行」を参照してください。

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（たとえば、後続 NULL が NULL 以外の値に更新される場合）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。次の項「 PCTFREE 、 PCTUSED 、行連鎖の基礎知識」で、領域管理パラメータの詳細を説明します。

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あります。「トランザクション・エントリ」は、ブロック内の 1 つ以上の行をアクセスする INSERT および UPDATE 、 DELETE 、 SELECT...FOR UPDATE の各文について、ブロック内に必要です。トランザクション・エントリに必要な領域は、オペレーティング・システムによって異なります。しかし、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。

PCTFREE 、 PCTUSED 、行連鎖の基礎知識

PCTFREE と PCTUSED という 2 つの領域管理パラメータにより、特定のセグメントのすべてのデータ・ブロック内での行の挿入および更新における空き領域の使用を制御できます。これらのパラメータは、表またはクラスタ（専用の「データ」セグメントを持つ）を作成または変更するときに指定します。記憶領域パラメータ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およびCLOB、NCLOB、BFILE）には適用されません。それらのデータ型では、PCTFREE記憶領域パラメータや空きリストは使われません。詳細は、 10-9 ページの「LOB データ型」を参照してください。

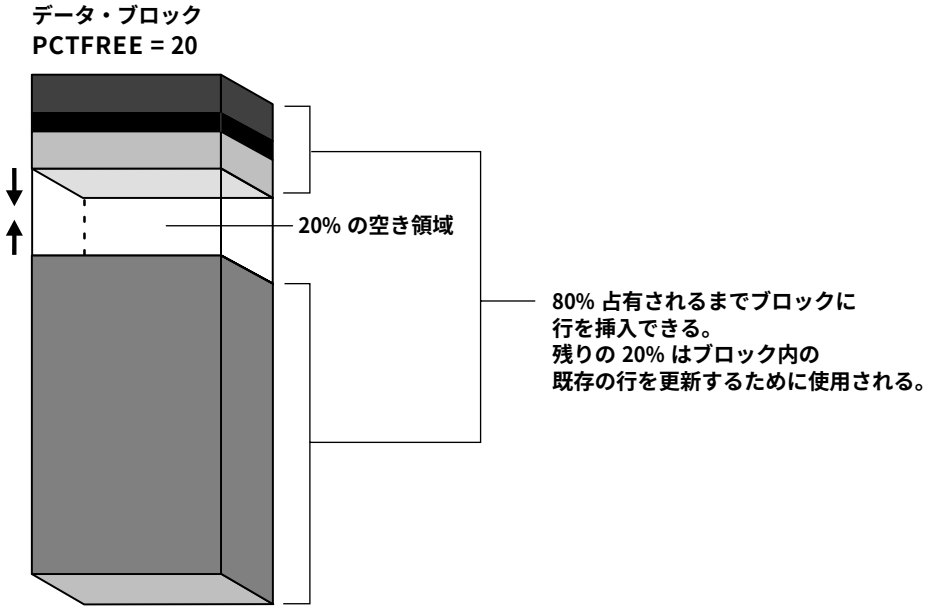
PCTFREE パラメータ

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・ブロック中で空き領域として「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場合を考えてみます。

```
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われます。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することができます。図 2-3 に、 PCTFREE の具体例を示します。

図 2-3 PCTFREE



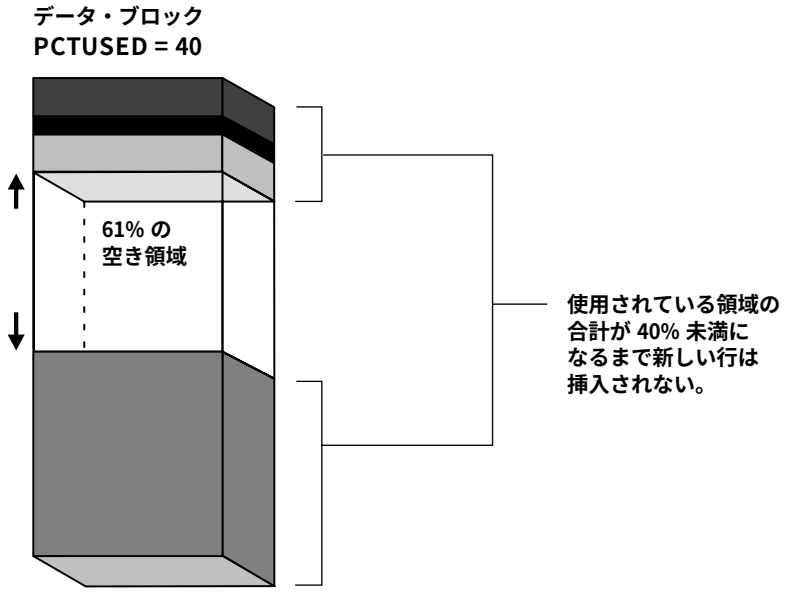
PCTUSED **パラメータ**

PCTUSED パラメータは、新しい行をブロックに追加するとき、行データとオーバーヘッドのために「使う」 ことが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杯になると、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え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のためだけに使われます。たとえば、 CREATE TABLE 文で次のようにパラメータを指定する場合を考えてみます。

```
PCTUSED 40
```

この場合、この表のデータ・セグメントとして使われるデータ・ブロックは、ブロックで使われている領域の総量が 39% 以下になるまでは、新しい行の挿入には使えないものとみなされます（ブロックの使われている領域が以前にPCTFREEに達したと想定します）。図 2-4に、この様子を示します。

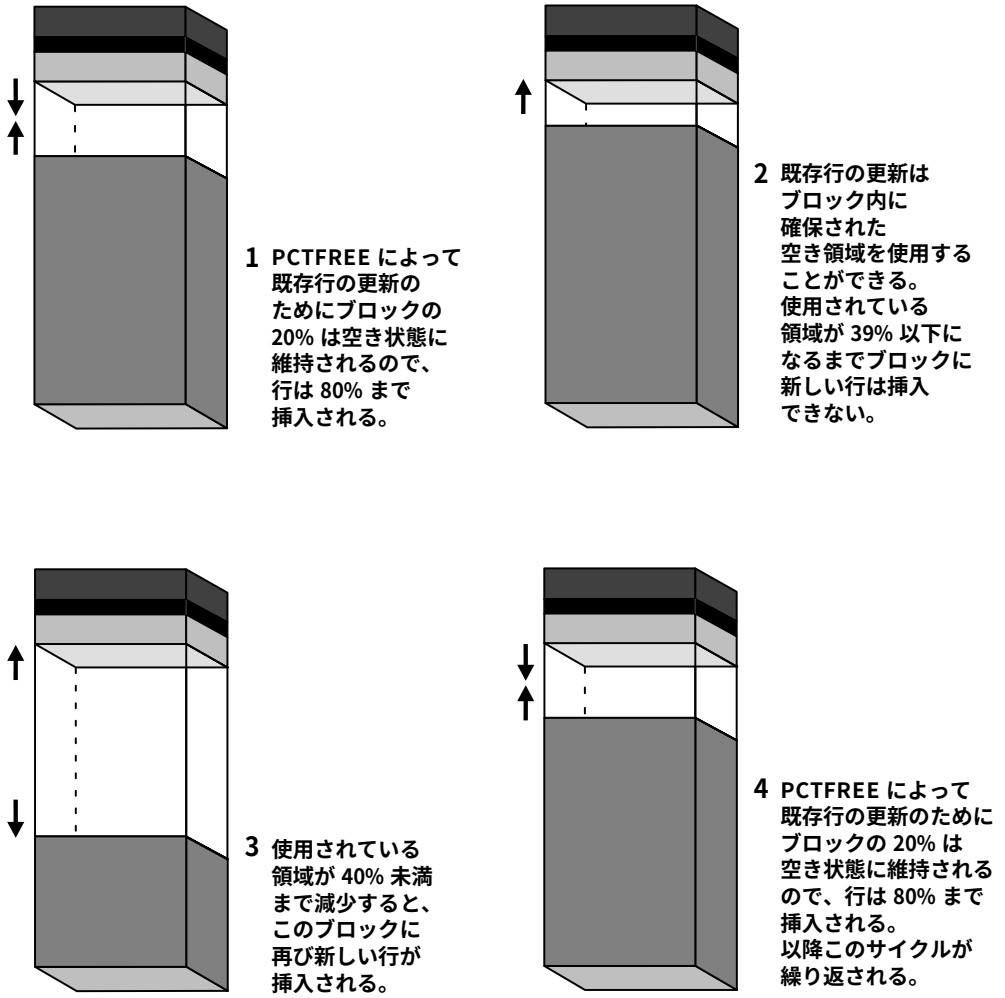
図 2-4 PCTUSED



PCTFREE と PCTUSED はどのように機能するか

PCTFREE と PCTUSED が一体となって機能することにより、データ・セグメント内にあるエクステントのデータ・ブロック内の領域の使用が最適化されます。図 2-5 に、これら 2 つのパラメータの相互作用を示します。

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



新しく割り当てられたデータ・ブロックでは、挿入に使える領域は、ブロック・サイズからブロック・オーバーヘッドと空き領域（PCTFREE）の合計を引いた大きさです。既存データの更新では、そのブロック内の使用可能な領域がすべて使われます。したがって、更新によって、ブロックの使用可能な領域がPCTFREE（更新用に確保され、挿入には使えない領域）よりも小さくなる可能性があります。

各データ・セグメントと索引セグメントについて、 Oracle は 1 つ以上の「空きリスト」を維持します。空きリストとは、セグメントのエクステン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。これらのデータ・ブロックは、挿入可能です。 INSERT 文を発行すると、表の空きリストから最初の使用可能データ・ブロックがチェックされ、可能であればそのデータ・ブロックが使われます。そのブロックの空き領域が INSERT 文に割り当てるのに十分な大きさでなく、かつ PCTUSED 以上である場合、そのブロックは空きリストから取り除かれます。 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わ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が使えるようになります。

データ・ブロック内の空き領域の可用性と圧縮

1 つ以上のデータ・ブロックの空き領域を増加させることのある文は、 DELETE 文と、既存の値を小さい値に更新する UPDATE 文の 2 つです。これらの文によって解放された領域は、次の条件を満たす後続の INSERT 文で使うことができます。

- 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える。
- 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、その領域が必要とされる場合に限り、使用可能になった領域を使える。

解放された領域は、データ・ブロック内の空き領域の主な領域と連続しているとは限りません。データ・ブロックの空き領域が連続した領域にまとめられるのは、 (1) INSERT 文または UPDATE 文が使おうとしているブロックに新しい行断片を入れるのに十分の大きさの空き領域がある場合で、(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ときに限ります。上記の場合にのみ圧縮が行われるのは、そうしないと、データ・ブロックの空き領域を絶えず圧縮しようとすることによりデータベース・システムのパフォーマンスが低下するためです。

行連鎖と移行

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

注意: 行の形式と行断片については、8-5ページの「行の形式とサイズ」で説明します。

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ものの、ブロックの空き領域がすでに満杯になっている場合です。この場合、Oracleはその行全体のデータを新しいデータ・ブロックに「移行」します（その行全体が新しいブロックに収まる場合)。移行された行の元の行断片は、行の移行先の新しいブロックを指すように保たれます。そのため、移行された行のROWIDは変わりません。

注意: ROWIDの詳細は、10-11ページの「ROWIDデータ型」を参照してください。

行が連鎖または移行されると、その行についての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・ブロックを走査しなければならないためです。

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

エクステント

エクステントは、複数の連続したデータ・ブロックで構成される、データベース記憶領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い切ると、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。

エクステントはいつ割り当てられるか

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・ブロックが満杯になり、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、そのセグメント内に直前に割り当てられたエクステントと同一またはそれ以上のサイズの後続エクステントです。(次の項では、増分エクステントのサイズを制御する要素について説明します。)

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

ロールバック・セグメントには必ず 2 つ以上のエクステントがあります。詳細は、 2-18 ページの「エクステントはどのように使われ、ロールバック・セグメントに割り当てられるか」を参照してください。

注意： この章は、 1 つのサーバー・プロセスによって SQL 文が解析されて実行され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。詳細は、 22-24 ページの「空き領域とパラレル DDL 」を参照してください。

エクステントの数とサイズの決定

エクステントに関して指定される「記憶領域パラメータ」によって、すべてのセグメントが定義されます。記憶領域パラメータは、すべてのタイプのセグメントに適用されます。記憶領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を制御します。たとえば、 CREATE TABLE 文の STORAGE 句で記憶領域パラメータを指定することによって、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したりすることができます。

エクステントはどのように割り当てられるか

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）を検索して、増分エクステントのサイズ以上のサイズを持つ最初の連続した空きデータ・ブロックを探す。

   a. 内部的な断片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データ・ブロックの集まりを検索する。（必要であれば、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る。）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、ちょうど 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まりが見つからないときは、必要量より大きい連続データ・ブロックの集まりを検索する。必要なエクステントのサイズよりも 5 ブロック以上大きい連続したブロックのグループが検出されると、そのブロックのグループは分割され、必要なサイズのエクステントが作成されると同時に空き領域とします。必要なエクステントのサイズよりも大きいブロックのグループが検出された場合でも、その差が 5 ブロック未満であれば、連続したブロックすべてが新しいエクステントに割り当てられます。

   先の例で考えると、ちょうど 20 個の連続したデータ・ブロックの集まりが見つからない場合は、 20 よりも大きい連続データ・ブロックの集まりを検索します。見つかった最初の集まりに 25 個以上のブロックがある場合は、ブロックを分割して、 20 ブロックを新しいエクステントに割り当て、残りの 5 個以上のブロックを空き領域と

します。見つかった最初の集まりに 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・ブロックの集まり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・ブロックの集まりを形成する。(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る。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る。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（空き領域は 5 つの連続したブロックより小さくはなりません)。

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え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する。

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い始めるときに、そのエクステントのブロックをフォーマットします。しかし、これはその必要がある場合だけです（セグメントの空きリストにあるブロックから始める)。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 オプション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、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ることもあります。

エクステントはいつ割当てを解除されるか

一般に、セグメントにデータが格納されているオブジェクトを（ 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 文を使って表やクラスタを切り捨てることができる。
- 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る。
- データベース管理者（ DBA ）は、次の SQL 構文を使って未使用のエクステントの割当てを解除できる。

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

エクステントが解放されると、 Oracle はデータ・ディクショナリを更新し、使用可能な領域としてエクステントが戻されたことを反映します。解放されたエクステントのブロックにあるデータにはアクセスできなくなり、そのブロックが他のエクステント用に再利用される時点でそのデータが消去されます。

追加情報: エクステントの割当て解除の詳細は、『Oracle8 Server 管理者ガイド』および『Oracle8 Server SQL リファレンス』を参照してください。

クラスタ化されていない表

クラスタ化されていない表が存在している限り、あるいはその表を切り捨てない限り、そのデータ・セグメントに割り当てられたデータ・ブロックは表に割り当てられたままになります。十分な領域があれば、新しい行はブロックに挿入されます。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え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には、他のエクステントが空き領域を必要とするとき、この領域を再生することができます。

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は、再生されて、その表領域内の他のオブジェクトが使えるようになります。その後、他のセグメントのために大きなエクステントが必要なとき、 Oracle は連続した再生済みエクステントを識別して結合し、要求された大きなエクステントを形成します。

クラスタ化表

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えるように残ります。エクステントが割当て解除されることはありません。エクステントを解放するため、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。

スナップショットとスナップショット・ログ

スナップショットとスナップショット・ログのエクステントは、クラスタ化されてない表やクラスタ化表の場合と同じ方法で割当て解除されます。

追加情報: スナップショットとスナップショット・ログの詳細は、『Oracle8 Server レプリケーション』を参照してください。

索引

索引セグメントに割り当てられたエクステントはすべて、その索引が存在する限り、割り当てられたままになります。索引や、それに対応する表またはクラスタを削除すると、エクステントは再生されて、表領域内で他の目的に使えるようになります。

ロールバック・セグメント

Oracle では、データベースのロールバック・セグメントが最適サイズを超えていないかどうかが定期的にチェックされます。ロールバック・セグメントが最適サイズを超えている（つまり、エクステントが多すぎる）場合、ロールバック・セグメントから 1 つ以上のエクステントの割当てが自動的に解除されます。詳細は、 2-22 ページの「エクステントはどのようにロールバック・セグメントから割当て解除されるか」を参照してください。

一時セグメント

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ので、他の複数ソートでそのセグメントを使えます。詳細は、 2-15 ページの「一時セグメント」を参照してください。

セグメント

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

Oracle データベースでは、次の 4 つのタイプのセグメントが使われます。

- データ・セグメント
- 索引セグメント
- 一時セグメント
- ロールバック・セグメント

この後の項では、各タイプのセグメントについて説明します。

データ・セグメント

クラスタ化されていない表またはパーティション、および Oracle データベースのクラスタにはそれぞれ、すべてのデータを保持するデータ・セグメントが 1 つあります。このデータ・セグメントは、 CREATE コマンドを使ってクラスタ化されていない表またはクラスタを作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、クラスタ化されていない表またはクラスタの記憶領域パラメータによって決定されます。これらの記憶領域パラメータは、適切な CREATE または ALTER コマンドによって直接設定できます。これらの記憶領域パラメータは、オブジェクトに対応するデータ・セグメントのデータ検索と格納の効率に影響を与えます。

注意: Oracleは、スナップショットとスナップショット・ログのセグメントを、クラスタ化されていない表やクラスタ化表の場合と同じ方法で作成します。

追加情報: スナップショットとスナップショット・ログの詳細は『Oracle8 Server レプリケーション』、 CREATE および ALTER コマンドの詳細は『Oracle8 Server SQL リファレンス』を参照してください。

索引セグメント

Oracle データベース内の各索引には、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。索引の索引セグメントは、 CREATE INDEX コマンドを発行した時点で作成されます。このコマンドでは、索引セグメントのエクステントの記憶領域パラメータと、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構いません。） 記憶領域パラメータを直接設定すると、データ検索と格納の効率に影響があります。

一時セグメント

Oracle では、問合せの処理中に、 SQL 文の解析と実行の中間段階で、一時的な作業領域が必要になることがよくあります。Oracleは、「一時セグメント」と呼ばれるこのディスク領域を自動的に割り当てます。通常、一時セグメントは、ソート操作の作業領域として必要です。ソート操作がメモリー内で実行できる場合、または Oracle が索引を使って操作を実行す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

次のコマンドでは、一時セグメントを使うことが必要になる場合があります。

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

一部の索引なしの結合および相関副問合せでも、一時セグメントを使う必要が生じることがあります。たとえば、問合せにDISTINCT句およびGROUP BY句、 ORDER BY 句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上記のコマンド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するとよいでしょう。

追加情報: SORT_AREA_SIZE およびその他の初期化パラメータの詳細は、『Oracle8 Server リファレンス・マニュアル』を参照してください。

一時セグメントはどのように割り当てられるか

一時セグメントは、ユーザー・セッション中に必要に応じて、文を発行したユーザーの一時表領域内に割り当てられます。一時表領域は、TEMPORARY TABLESPACEオプションを付けたCREATE USERまたはALTER USERコマンドで指定します。ユーザーに定義されている一時表領域がない場合、SYSTEM表領域がデフォルトの一時表領域として使われます。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ら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するとよいでしょう。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。

ユーザーの一時セグメント表領域を割り当てる方法は、第25章「データベース・アクセスの制御」を参照してください。

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含まれません。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含まれます。

ロールバック・セグメント

各データベースには1つ以上のロールバック・セグメントが入っています。ロールバック・セグメントには、各トランザクションで（コミットされたかどうかに関係なく）変更されたデータの古い値が記録されます。ロールバック・セグメントは、読込み一貫性を実現したり、トランザクションをロールバックしたり、データベースを回復したりするために使われます。ロールバック・セグメントがさまざまな状況下でどのように機能するかは、次に示す項を参照してください。

トピック	項
読込み一貫性	23-4ページの「マルチバージョン一貫性制御」
トランザクションのロールバック	15-6ページの「トランザクションをロールバックする」
データベースの回復	28-9ページの「ロールバック・セグメントとロールバック」

ロールバック・セグメントの内容

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、ロールバック・エントリは容易に見つかります。

データベース・ユーザーもデータベース管理者も、ロールバック・セグメントに対してアクセスや読込みを実行できません。Oracleだけが書込みや読込みを実行できます。（ロールバック・セグメントは、実際に作ったユーザーとは関係なく、ユーザー SYS によって所有されます。）

ロールバック・エントリのロギング

ロールバック・エントリはロールバック・セグメント内のデータ・ブロックを変更します。そして 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、インスタンス回復またはメディア回復の一部として自動的に復元されます。回復が完了すると、 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。

ロールバック情報はいつ必要になるか

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・セグメントを使う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ったりするのに使われます。

ロールバック・セグメントには、 1 つのトランザクションごとに、変更前のデータが記録されます。各トランザクションについて、新しい変更がそれぞれ前の変更とリンクされます。トランザクションをロールバックする必要がある場合は、一連の変更が、データを直前の状態に復元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、 Oracle は、ロールバック・セグメント内の情報によって、ある 1 時点に対応する一貫したデータの集合を作ることができます。

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る。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、読込み専用トランザクション（問合せだけ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ることができます。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 で使われる特定のロールバック・セグメントを指定できます。これによって、アプリケーションの開発者

やユーザーは、それぞれのトランザクションに合わせて、サイズの異なるロールバック・セグメントを選択できます。

トランザクションの存続期間において、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・セグメントだけ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込み一貫ビューを作成できるようにするため、その情報はロールバック・セグメントに残っています。そのようなビューのためにロールバック・データを可能な限り長く使用可能にするため、 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満杯になると、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することにより、ロールバック・データの書込みが継続されます。長時間実行トランザクション（アイドル状態またはアクティブ状態）では、新しいエクステントをロールバック・セグメントに割り当てる必要が生じることがあります。トランザクションでロールバック・セグメントのエクステントが使われる方法の詳細は、図 2-6 および図 2-7 、図 2-8 を参照してください。

各ロールバック・セグメントは、 1 つのインスタンスからの一定の数のトランザクションを処理できます。トランザクションを特定のロールバック・セグメントに明示的に割り当てない限り、使用可能なロールバック・セグメントへのアクティブ・トランザクションの配分は、すべてのロールバック・セグメントにほぼ同じ数のアクティブ・トランザクションが割り当てられるように行われます。この配分は使用可能なロールバック・セグメントのサイズには依存しません。したがって、すべてのトランザクションで同じ量のロールバック情報が生成される場合は、ロールバック・セグメントをすべて同じサイズにすることができます。

> **追加情報**： 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、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・システム別の Oracle のマニュアルを参照してください。

エクステントはどのように使われ、ロールバック・セグメントに割り当てられるか

ロールバック・セグメントの作成時には、記憶領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なければなりません。

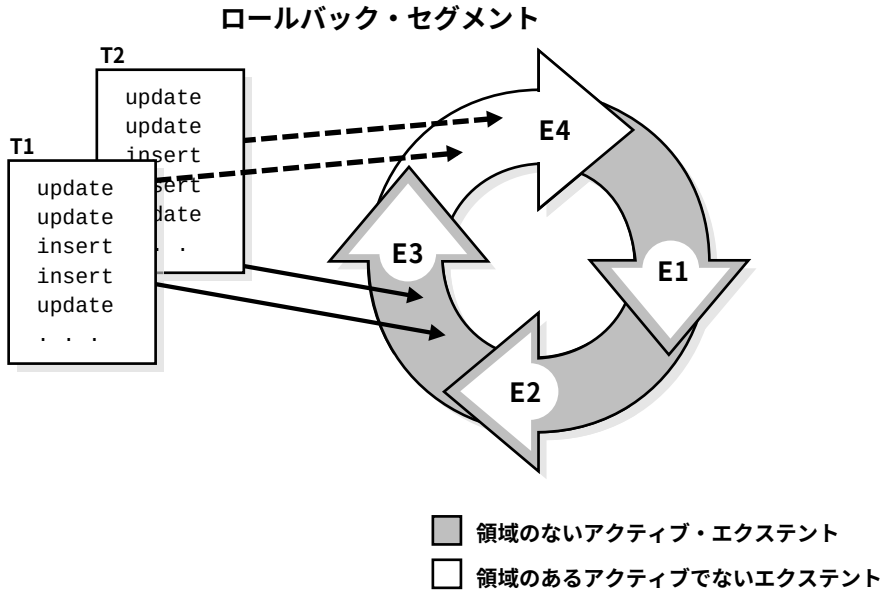
1 つのトランザクションは、 1 つのロールバック・セグメントに順次方式で書込みを行います。各トランザクションは、どの時点でもロールバック・セグメントの 1 つのエクステントだけに書き込みます。多数の「アクティブ」トランザクションは、 1 つのロールバック・セグメントに同時に書き込むことができ、しかもロールバック・セグメントの同一のエクステントに同時に書き込むこともできます。ただし、ロールバック・セグメントのエクステント内にある各データ・ブロックには、 1 つのトランザクションについての情報しか格納できません。

トランザクションが現行のエクステント領域をすべて使い尽くしてしまい、なお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する。
- 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する（そして割り当てる）。

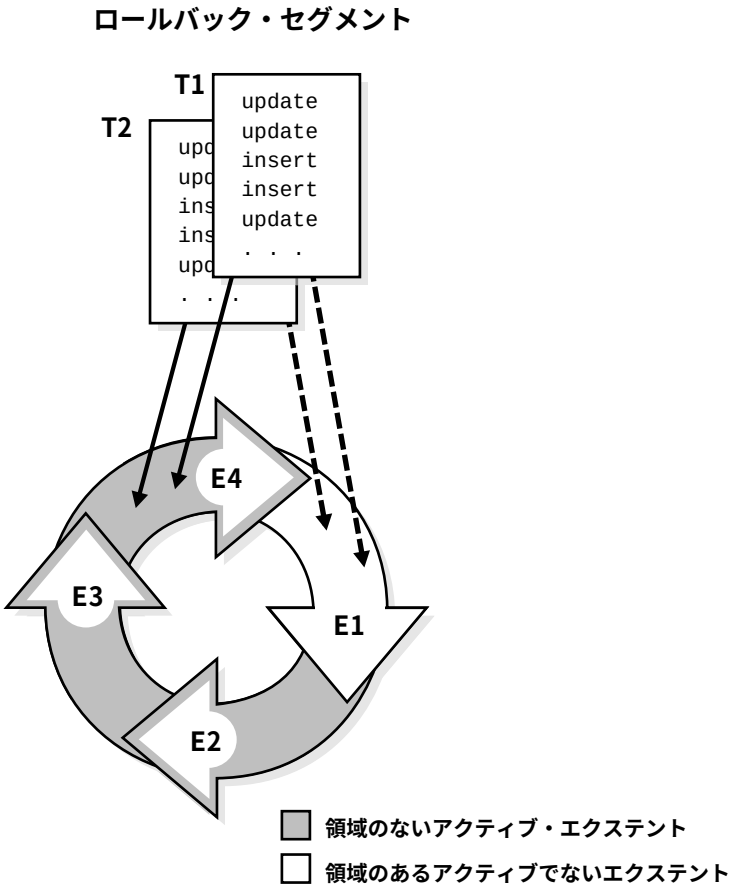
ロールバック領域をさらに取得する必要のあるトランザクションは、ロールバック・セグメントの次のエクステントを調べます。ロールバック・セグメントの次のエクステントにアクティブ・トランザクションからの情報が含まれていない場合は、そのエクステントが現行エクステントになります。それ以後にさらに領域が必要になったすべてのトランザクションは、ロールバック情報をその新しい現行エクステントに書き込むことができます。図 2-6に、ロールバック・セグメントの第 3 エクステント（ E3 ）から書込みを開始して、第 4 エクステント（ E4 ）まで書込みを続ける 2 つのトランザクション、 T1 と T2 を示します。

図 2-6 **ロールバック・セグメントに割り当てられたエクステントの使用**



トランザクションが書込みを継続し、現行のエクステントが満杯になると、 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 2-7 で、 E4 が完全に満杯であるときに 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をします。この図では、 E1がそのエクステントに相当します。この図は、ロールバック・セグメント内のエクステントが循環的に使われる様子を示しています。

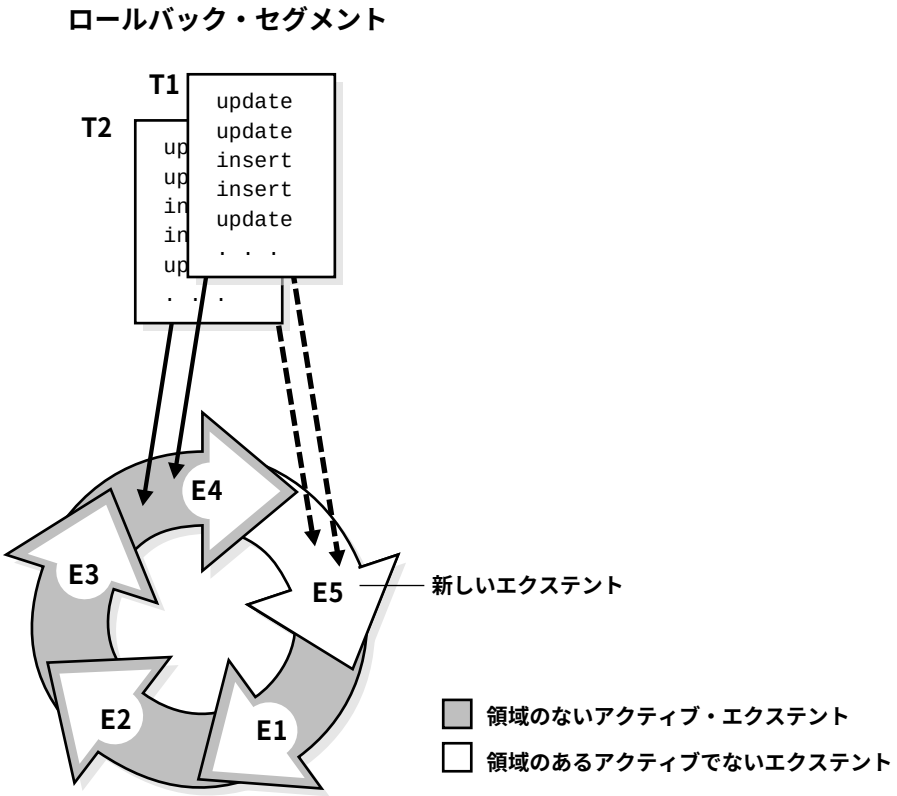
図 2-7 ロールバック・セグメントに割り当てられたエクステントの循環的な使用



トランザクションのロールバック情報の書込みを継続するときに、 Oracle は常にリング内の次のエクステントを最初に再利用しようとします。しかし、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・セグメントの記憶領域パラメータ 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 2-8 に、ロールバック・セグメントに割り当てられた新しいエクステントを示します。コミットされていないトランザクション（アイドル状態またはアクティブ状態のトランザクション、または継続中のインダウト分散トランザクション）が長時間実行されています。この時点で、トランザクションはロールバック・セグメントの 4 番目のエクステント（E4）に書き込んでいます。しかし、ロールバック・セグメントに割り当てられている次のエクステント（E1）にはアクティブなロールバック・エントリが含まれているため、 E4 が完全に満杯になると、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に書込みを継続します。

図 2-8 ロールバック・セグメントへの新しいエクステントの割当て



エクステントはどのようにロールバック・セグメントから割当て解除されるか

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えるようになります。

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、記憶領域パラメータ OPTIMAL（ロールバック・セグメントだけに適用される）を使って、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・セグメントの現在の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、かつ満杯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、一貫した読込みで使われる可能性が最も低いので、 Oracle はいつもそのような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。

`(INITIAL + NEXT + NEXT + ... MINEXTENTS` まで `)` バイト

SYSTEM ロールバック・セグメント

データベースの作成時にはいつでも、 SYSTEM という名前の初期ロールバック・セグメントが作られます。この初期ロールバック・セグメントは SYSTEM 表領域内に作られ、 SYSTEM 表領域のデフォルト記憶領域パラメータを使います。 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・セグメントをいつも取得します。

ロールバック・セグメントが複数ある場合、 Oracle は特別のシステム・トランザクションだけに SYSTEM ロールバック・セグメントを使い、ユーザー・トランザクションは他のロールバック・セグメントに割り当てようとします。 SYSTEM ロールバック・セグメント以外のロールバック・セグメントに対するトランザクションが多すぎる場合、 Oracle は必要に応じて SYSTEM ロールバック・セグメントを使います。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 インスタンスがデータベースをオープンするとき、そのインスタンスは、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・セグメントを取得しなければなりません。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利用できる、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構いません。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・セグメントを取得しなければならない。そのインスタンスが、データベースにアクセスしている唯一のインスタンスである場合は、 SYSTEM セグメントを取得します。そのインスタンスが、 Oracle Parallel Server 内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から求める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する。

   ```
   CEIL(TRANSACTIONS/TRANSACTIONS_PER_ROLLBACK_SEGMENT)
   ```

   CEIL は、入力引数の数値以上の最小の整数を戻す SQL 関数です。上記の例で、 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。（しかし、インスタンスは、上の計算によって算出される数のロールバック・セグメントを取得できなくても、データベースをオープンすることはできます。）

   > **注意** : TRANSACTIONS_PER_ROLLBACK_SEGMENT パラメータによって、ロールバック・セグメントを使え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決まるのは、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. SYSTEM ロールバック・セグメントを取得した後、インスタンスは次に、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する。 Oracle Parallel Server 内のインスタンスがデータベースをオープンして、別のインスタンスがすでに要求しているプライベート・ロールバック・セグメントを取得しようとした場合、ロールバック・セグメントを取得しようとしている 2 番目のインスタンスは起動時にエラーを受け取ります。また、存在しないプライベート・ロールバック・セグメントを取得しようとするインスタンスも、エラーを受け取ります。

4. インスタンスが、ステップ 3 で十分な数のロールバック・セグメントを取得できた場合は、それ以上のアクションは必要ありません。しかし、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にされるか、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えません。

データベース内に十分な数のセグメントがあり、データベースをオープンする各インスタンスが、１つの SYSTEM ロールバック・セグメントを含む 2 つ以上のロールバック・セグメントを取得できる限り、Oracle Parallel Serverで使われるデータベースはパブリック・セグメントだけを持ち、プライベート・セグメントは持たないということがあります。ただし、 Oracle Parallel Server を使っている場合に、プライベート・ロールバック・セグメントを使う必要が生じることもあります。

追加情報：Oracle Parallel Server でのロールバック・セグメントの使用方法は、『Oracle8 Parallel Server 概要および管理』を参照してください。

ロールバック・セグメントの状態

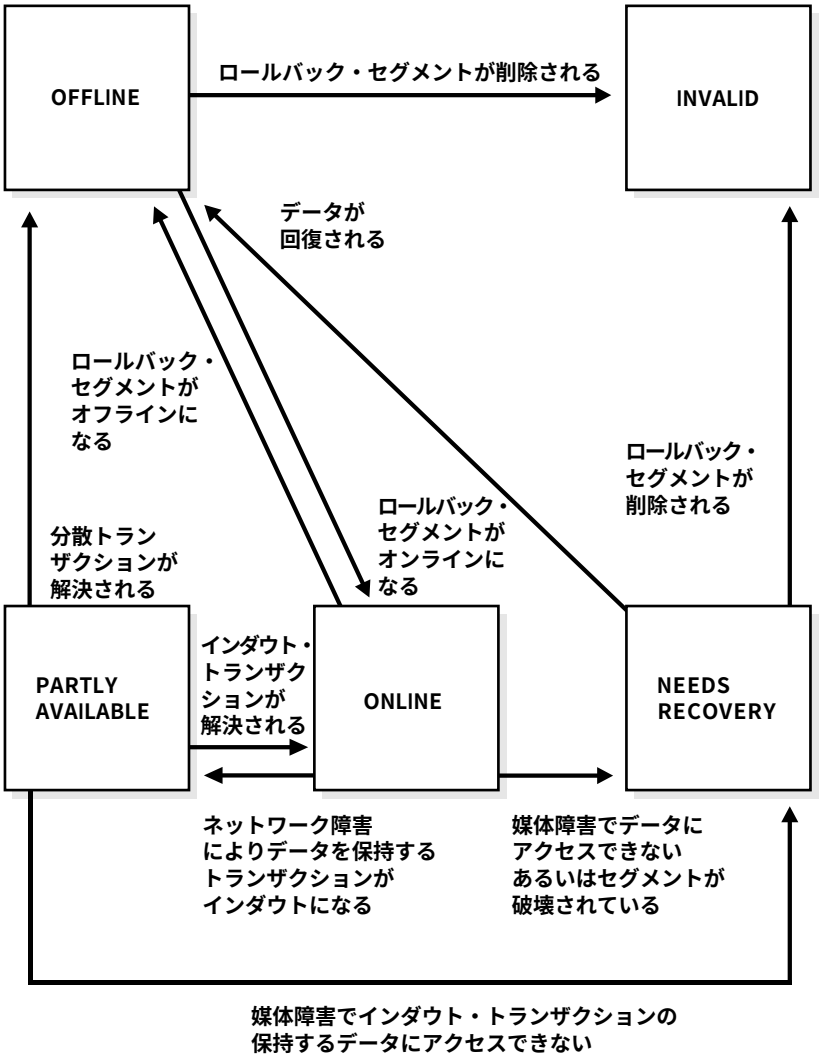
ロールバック・セグメントの状態は、常にいくつかの状態のうちの 1 つに相当します。その状態は、ロールバック・セグメントがオフラインになっているか、インスタンスによって取得されているか、未解決のトランザクションにかかわっているか、回復を必要としているか、削除されたかといった状況によって決まります。その状態に応じて、そのロールバック・セグメントをトランザクションで使えるかどうか、および DBA がどんな管理処理を実行できるかが決まります。

ロールバック・セグメントの状態には、次のものがあります。

OFFLINE	インスタンスによって取得されていない（オンラインになっていない）。
ONLINE	インスタンスによって取得されている（オンラインになっている）。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、データが壊れている。
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）のデータが入っている。
INVALID	削除された（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、新しいロールバック・セグメントの作成時に使われる。）

データ・ディクショナリ表 DBA_ROLLBACK_SEGS には、各ロールバック・セグメントの状態が、その他のロールバック・セグメント情報と一緒にリストされます。図 2-9 に、ロールバック・セグメントがある状態から別の状態に移行する様子を示します。

図 2-9 ロールバック・セグメントの状態間の移行



PARTLY AVAILABLE および NEEDS RECOVERY Rollback セグメント

PARTLY AVAILABLEおよびNEEDS RECOVERYの状態は、どちらもよく似ています。どちらの状態のロールバック・セグメントにも、ほとんどの場合、解決されていないトランザクションのデータが含まれています。

- PARTLY AVAILABLEのロールバック・セグメントは、ネットワーク障害が原因で解決できないインダウト分散トランザクションで使われています。 NEEDS RECOVERY のロールバック・セグメントは、データ・ファイルの欠損や破損などのローカルな媒体障害、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）で使われています。
- Oracle または DBA は、 PARTLY AVAILABLE のロールバック・セグメントをオンラインにできます。それとは対照的に、 NEEDS RECOVERY のロールバック・セグメントをオンラインにするには、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。（データベースを回復してトランザクションを解決すると、 NEEDS RECOVERY のロールバック・セグメントの状態は自動的に OFFLINE に変更されます。）
- DBA は、 NEEDS RECOVERY のロールバック・セグメントを削除できます。（これによって、DBAは破損したセグメントを削除できます。）PARTLY AVAILABLEのセグメントは削除できません。最初にインダウト・トランザクションを RECO プロセスで自動的に解決するか、手動で解決しておく必要があります。

  追加情報：分散トランザクションでの障害の詳細は、『Oracle8 Server 分散システム』を参照してください。

PARTLY AVAILABLEのロールバック・セグメントをオンラインにする（コマンドを使うか、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えるようになります。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依然として保持するため、そのロールバック・セグメントを使える新しいトランザクションの数は制限されます。（トランザクション表の詳細は、2-17ページの「ロールバック情報はいつ必要になるか」を参照してください。）

また、インダウト・トランザクションが解決されるまでは、そのトランザクションがロールバック・セグメント内に取得したエクステントは保持され続けるため、他のトランザクションはそれらのエクステントを使うことができません。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 AVAILABLE のセグメントをオンラインにするよりも、トランザクション用に新しくロールバック・セグメントを作るとよいでしょう。

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回復されると、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られます。

3

表領域とデータ・ファイル

Space — the final frontier . . .

Gene Roddenberry: *Star Trek*

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。

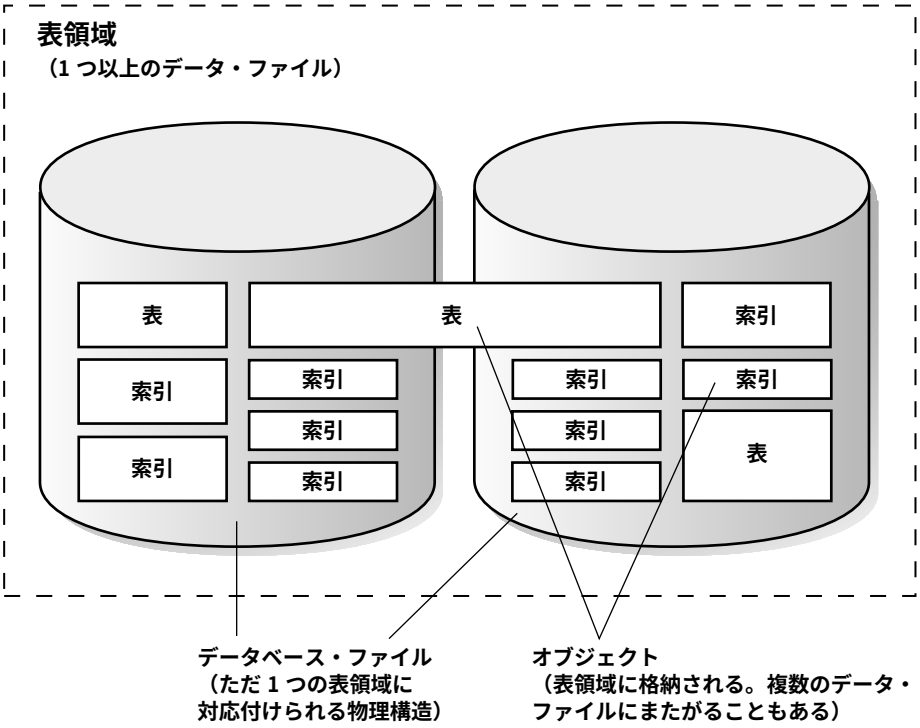
- 表領域とデータ・ファイルの基礎知識
- 表領域
- データ・ファイル

追加情報: Trusted Oracleを使っている環境での表領域とデータ・ファイルの詳細は、 Trusted Oracle のマニュアルを参照してください。

表領域とデータ・ファイルの基礎知識

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、この関連を示します。

図 3-1 **データ・ファイルと表領域**



データベース、表領域、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異があります。

データベースと表領域　　Oracle データベースは、 1 つ以上の、表領域と呼ばれる論理記憶単位から構成されます。データベースのデータは、データベースの表領域にまとめて格納されます。

表領域とデータ・ファイル	Oracleデータベース内の各表領域は、1つ以上の、データ・ファイルと呼ばれるファイルから構成されます。データ・ファイルは、 Oracle が実行されているオペレーティング・システムに適合した物理構造です。
データベースとデータ・ファイル	データベースのデータは、データベースの各表領域を構成するデータ・ファイル内にまとめて格納されます。たとえば、最も単純な Oracle データベースには、1つのデータ・ファイルで構成される表領域が1つあります。より複雑なデータベースには、それぞれ 2 つのデータ・ファイルから構成される表領域が 3 つあります（合計で 6 つのデータ・ファイルになります)。

この後の項では、表領域とデータ・ファイルについて、さらに詳しく説明します。

表領域

データベースは、「表領域」と呼ばれる 1 つ以上の論理的な記憶単位に分けられます。データベース管理者（ DBA ）は、表領域を使って次の事柄を行います。

- 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する。
- データベース・ユーザーに対する固有の領域割当て制限を設定する。
- 各表領域を個別にオンライン、オフラインにすることによって、データの可用性を制御する。
- 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する。
- 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る。

データベース管理者は、表領域の新規作成、表領域へのデータ・ファイルの追加、表領域内に作られるセグメントに対するデフォルトのセグメント記憶設定値の設定および変更、表領域の読込み専用または書込み可能への変更、一時的または永続的表領域の作成、および表領域の削除を行うことができます。

この項で取り上げるトピックは次のとおりです。

- SYSTEM 表領域
- データベースにより多くの領域を割り当てる
- 表領域をオンライン / オフラインにする
- 一時表領域
- 読込み専用表領域

表領域は、「セグメント」と呼ばれる記憶の論理単位に分割されます。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、セグメント」に説明があります。

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られる SYSTEM という表領域が含まれています。 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。

小規模なデータベースでは、 SYSTEM 以外の表領域は必要ないことがあります。しかし、オラクル社では、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 つ作るようお薦めしています。これによって、種々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・オブジェクトを同一のデータ・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。

注意 : SYSTEM 表領域は、データベースのオープン中は常にオンラインにしておく必要があります。 3-7 ページの「表領域をオンライン / オフラインにする」を参照してください。

ストアドPL/SQLプログラム・ユニットのために格納されるデータ（プロシージャ、ファンクション、パッケージ、トリガー）は、すべて SYSTEM 表領域に置かれます。データベースにこれらのプログラム・ユニットが多数格納されることになる場合、データベース管理者はこれらのオブジェクトが SYSTEM 表領域内で使用する領域を見込んでおく必要があります。これらのオブジェクトと、オブジェクトが必要とする領域の詳細は、第 17 章「プロシージャとパッケージ」および第 18 章「データベース・トリガー」を参照してください。

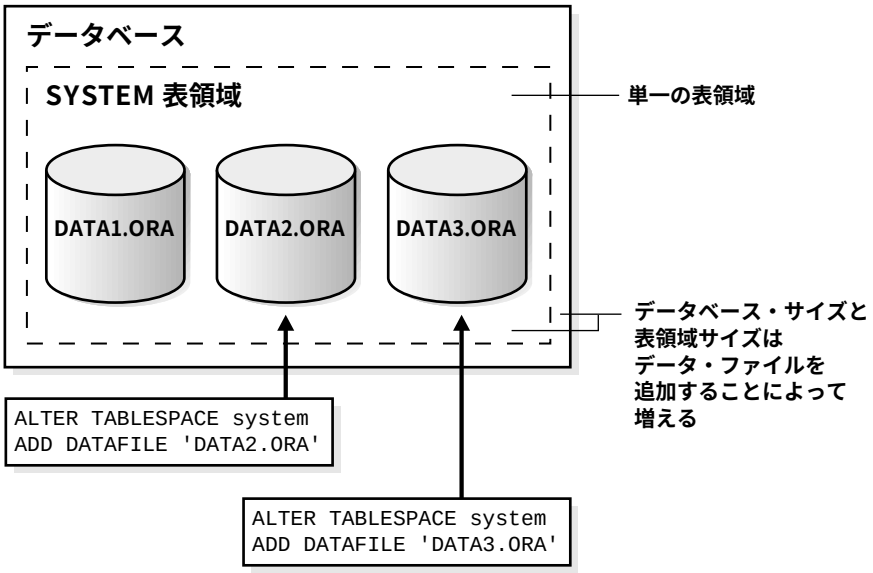
データベースにより多くの領域を割り当てる

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する
- 新しい表領域を追加する
- データ・ファイルのサイズを大きくする

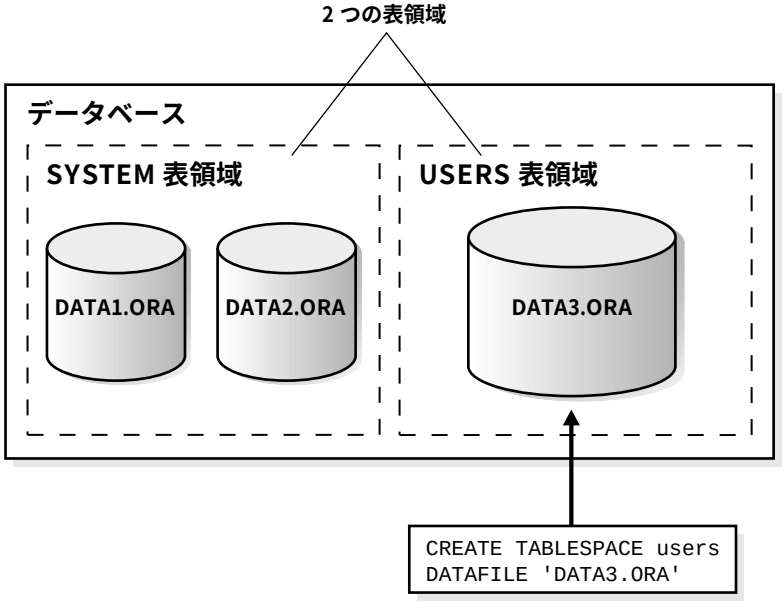
既存の表領域にデータ・ファイルをもう 1 つ追加すると、その表領域に割り当てられているディスク領域の容量を増やすことになります。図 3-2 は、このようにして領域を拡大する方法を示しています。

図 3-2 表領域にデータ・ファイルを追加してデータベースを拡大する



別の方法として、新しい表領域を作り（少なくとも 1 つのデータ・ファイルを作成することになる）、データベースのサイズを増やすこともできます。図 3-3 に、この方法を示します。

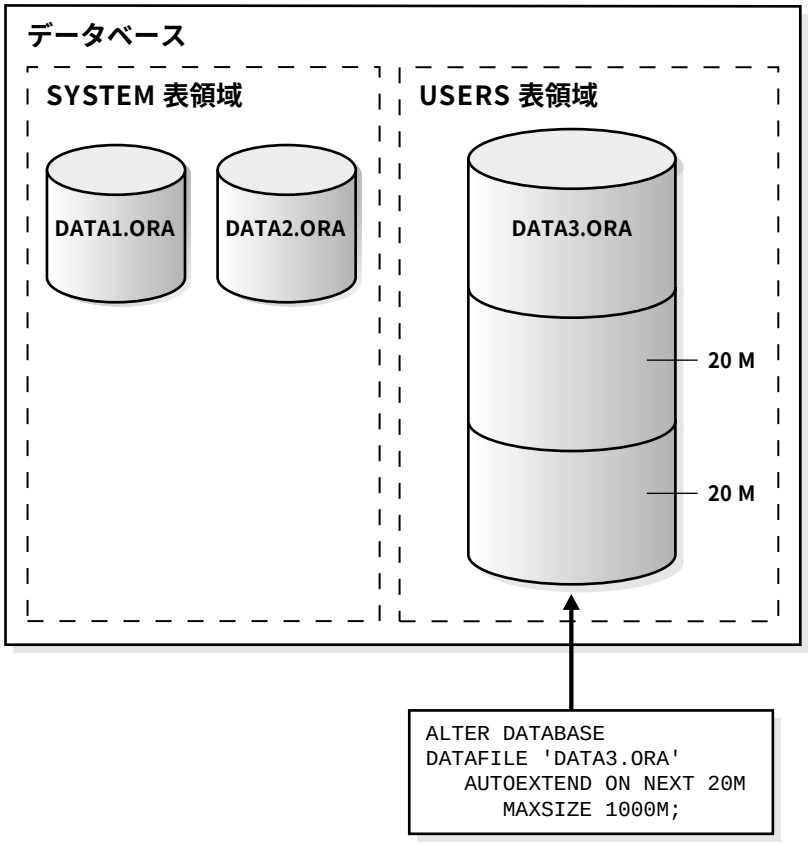
図 3-3 新しい表領域を追加してデータベースを拡大する



表領域のサイズは、表領域を構成する1つ以上のデータ・ファイルのサイズです。データベースのサイズは、データベースを構成する表領域の合計サイズです。

3 番目の方法として、データ・ファイルのサイズを変更する、あるいは、さらに領域が必要になるの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することができます。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の付加されたデータ・ファイルを追加します。図 3-4 に、この方法を示します。

図 3-4 データ・ファイルを動的にサイズ変更してデータベースを拡大する



追加情報: データベース内の領域を拡張する方法の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

表領域をオンライン / オフラインにする

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、そのデータベース内の表領域（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。

注意 : SYSTEM 表領域はデータベースのオープン中は常にオンラインになっています。 Oracle がデータ・ディクショナリを常に使えるようになっている必要があるからです。

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に入っているデータを使えます。しかし、データベース管理者は、次のような理由で表領域をオフラインにすることがあります。

- データベースの一部を使えないようにする。一方で、データベースの残りの部分は通常どおりアクセスできるようにする。
- 表領域のオフライン・バックアップを実行する（ただし、オンラインで使用中の表領域をバックアップすることもできます）。
- アプリケーションの更新やメンテナンスをする間、アプリケーションとその表のグループを一時的に使えないようにする。

使用中のロールバック・セグメントを含む表領域をオフラインにすることはできません。詳細は、 2-17 ページの「ロールバック・セグメント」を参照してください。

表領域がオフラインになった場合

表領域がオフラインになると、 Oracle では後続の SQL 文でその表領域内に含まれるオブジェクトを参照できなくなります。表領域内のデータを参照する完了済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、遅延ロールバック・セグメント（ SYSTEM 表領域にある）に保存されます。そして、表領域が再びオンラインになったときに、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りしたときには、 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だけに限られます。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からです。オフラインの表領域は、 Oracle 以外のユーティリティで読み込んだり、編集したりできません。したがって、表領域をデータベースからデータベースに転送することはできません。

追加情報: Oracleデータの転送は、『Oracle8 Serverユーティリティ』で説明されているツールによって実行できます。

ある種のエラーが発生すると（たとえば、データベース・ライター・プロセス、 DBWn が、表領域のデータ・ファイルに書き込もうとして、何度か失敗したとき）、Oracleは表領域をオンラインからオフラインへ自動的に変更します。オフラインの表領域にある表にアクセスしようとしユーザーは、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媒体障害である場合は、ハードウェアの問題を解決してから、表領域を回復しなければなりません。

特殊なプロシージャに対して表領域を使う

複数の表領域を作成して異なる種類のデータを分離する場合、さまざまなプロシージャに合わせて特定の表領域をオフラインにします。他の表領域はオンラインのままなので、その中の情報は使える状態のままです。しかし、表領域をオフラインにすると、特別な状況が発生することがあります。たとえば、 2 つの表領域を使って表データを索引データと分離する場合、次のようなことに注意してください。

- 索引を含む表領域がオフラインになっていても、問合せはその表データをアクセスできる。問合せは、表データをアクセスするときに索引を必要としないからです。
- 表データを含む表領域がオフラインになっていると、データベース内のその表データはアクセスできない。データをアクセスするには表が必要だからです。

要約すれば、ある文を実行するのに十分な情報がオンライン表領域内にあるとき、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。

読込み専用表領域

読込み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（回復）したりする必要をなくすことです。読込み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ので、 CD-ROM や WORM ドライブのような読込み専用媒体に置くことができます。

注意： 表領域は、表領域が作られたデータベースにおいてしかオンラインにできないため、読込み専用表領域はアーカイブ要件やデータ発行要件を満たすとは限りません。

新規の表領域は、必ず読み書き可能なものとして作られます。ALTER TABLESPACEコマンドの READ ONLY オプションにより、表領域を読込み専用に変更するとともに、この表領域に関連したデータ・ファイルをすべて読込み専用にすることができます。その後、 READ WRITE オプションを使って、読込み専用表領域を再び書込み可能にすることができます。

追加情報: ALTER TABLESPACEコマンドの詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

表領域を読込み専用にしても、その表領域のオフライン/オンライン状態が変化することはありません。オフラインのデータ・ファイルにはアクセスできません。読込み専用表領域に含まれるデータ・ファイルをオンラインにすると、読込みだけが可能になります。このデータ・ファイルは、対応付けられている表領域が読み書き状態に戻るまで、書込みができません。読込み専用表領域のデータ・ファイルは、 ALTER DATABASE コマンドの DATAFILE オプションを使えば個別にオンラインまたはオフラインにできます。

読込み専用表領域は修正できません。読込み専用表領域を更新するには、最初にその表領域を書込み可能にします。表領域の更新が終わったら、その表領域を読込み専用に再設定します。

読込み専用表領域は変更されないため、バックアップを繰り返す必要はありません。また、データベースを回復する必要が生じたときにも、読込み専用表領域は修正されていないので、回復する必要がありません。しかし、インスタンス回復またはメディア回復のときには、読込み専用表領域が書込み可能になっていたことがあるかどうかと、書込み可能になっていた時期に応じて、その表領域に注意を向けなければならないことがあります。

> **追加情報**: 回復の詳細は、『Oracle8 Server バックアップおよびリカバリ』を参照してください。

オフライン表領域から表や索引などの項目を削除できるのと同様、読込み専用表領域からもそれらの項目を削除できます。しかし、読込み専用表領域内でオブジェクトを作成したり変更したりすることはできません。

読込み専用表領域へのデータ・ファイルの追加は、その表領域をオフラインにしても実行できません。データ・ファイルを追加すると、 Oracle はファイル・ヘッダーを更新する必要がありますが、読込み専用表領域ではその書込み操作が禁止されているからです。

一時表領域

ソートのための領域は、ソート専用に指定した「一時表領域」を使えば、さらに効率的に管理できます。そうすることによって、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直列化を効率的に解消することができます。結合、索引作成、順序づけ（ ORDER BY）、集計の計算（GROUP BY）、ANALYZEコマンドを使ったオプティマイザ統計の収集など、ソートを使うすべての操作には、一時表領域の機能が役立ちます。これによるパフォーマンスの向上は、 Oracle Parallel Server 環境で特に顕著です。

「一時表領域」は、ソート・セグメントにだけ使えます。（一時表領域は、ユーザーが一時セグメントのために指定する表領域とは異なります。一時セグメントのために指定する表領域としては、ユーザーが使用できる任意の表領域を使えます。） 永続オブジェクトを一時表領域に置くことはできません。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われます。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。

メモリー内に入りきらない大規模なソート操作が複数ある場合には、一時表領域を使うとパフォーマンスが改善されます。最初のソート操作が実行されるときに、所定の一時表領域のソート・セグメントが作られます。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。

一時表領域を作るには、次のような SQL 構文を使います。

```
CREATE TABLESPACE tablespace TEMPORARY | PERMANENT;
```

また、次の構文を使うと、表領域を PERMANENT から TEMPORARY に、または TEMPORARY から PERMANENT に変更することができます。

```
ALTER TABLESPACE tablespace TEMPORARY;
```

追加情報: CREATE TABLESPACEコマンドとALTER TABLESPACEコマンドの詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

データ・ファイル

Oracle データベース内の表領域は、1 つ以上の物理的な「データ・ファイル」から構成されます。1 つのデータ・ファイルは、1 つの表領域および 1 つのデータベースだけ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ることにより、表領域のデータ・ファイルを作ります。データ・ファイルが作成されるとき、そのファイルが Oracle に割り当てられる前に、Oracle が実行されているオペレーティング・システムによってそのファイルから古い情報や許可が消去されます。ファイルが大きい場合には、この処理にかなりの時間がかかることがあります。

追加情報: 使用するオペレーティング・システムにおいてデータ・ファイルのファイル・ヘッダーに必要な領域の容量は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

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。

データ・ファイルの内容

データ・ファイルが最初に作成されるとき、割り当てられるディスク領域はフォーマットされますが、ユーザー・データは含まれていません。しかしその領域は、対応する表領域の将来のセグメントのデータを保持するために確保されます。その領域は、Oracle だけが使うことになります。表領域内のデータが増加すると、Oracle は対応付けられたデータ・ファイル内の空き領域を使ってセグメントにエクステントを割り当てます。詳細は、第 2 章「データ・ブロック、エクステント、セグメント」を参照してください。

表領域内のスキーマ・オブジェクトに対応付けられたデータは、物理的には、表領域を構成する１つ以上のデータ・ファイルに格納されます。スキーマ・オブジェクトは、特定のデータ・ファイルには対応しません。データ・ファイルは、ある決まった表領域内のオブジェクト・データを保持するためのリポジトリです。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）を使わない限り、データベース管理者およびエンド・ユーザーは、どのデータ・ファイルにオブジェクトが格納されるかを制御できません。

データ・ファイルのサイズ

データ・ファイルを作った後でそのデータ・ファイルのサイズを変更したり、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したりできます。この機能によって、表領域当たりのデータ・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。

追加情報：データ・ファイルのサイズ変更の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

オフライン・データ・ファイル

SYSTEM以外の表領域は、いつでも「オフライン」（使用不能）にしたり、「オンライン」（使用可能）にしたりできます。表領域をオフラインまたはオンラインにすると、その表領域を構成するすべてのデータ・ファイルは１つの単位としてオフラインまたはオンラインになります。個々のデータ・ファイルをオフラインにすることもできます。しかし、それを行うのは通常、一部のデータベース回復手順を実行するときだけです。

4

データ・ディクショナリ

LEXICOGRAPHER — A writer of dictionaries, a harmless drudge.

Samuel Johnson: *Dictionary*

この章では、「データ・ディクショナリ」と呼ばれる、各Oracleデータベースの読込み専用の参照表とビューの中枢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・ディクショナリの基礎知識
- データ・ディクショナリの構造
- SYS 、データ・ディクショナリの所有者
- データ・ディクショナリが使用される方法
- 動的パフォーマンス表

データ・ディクショナリの基礎知識

「データ・ディクショナリ」は Oracle データベースで最も重要な部分の 1 つであり、対応付けられているデータベースに関する情報を提供する読込み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、現在使われている領域の容量
- 列のデフォルト値
- 整合性制約に関する情報
- Oracle ユーザーの名前
- それぞれのユーザーに付与されている権限とロール
- 監査情報。たとえば、各種スキーマ・オブジェクトをだれがアクセスしたり更新したりしたか。
- Trusted Oracleでは、すべてのスキーマ・オブジェクトとユーザーのラベル（Trusted Oracle のマニュアルを参照）
- 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 文を使います。データ・ディクショナリは読込み専用なので、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）だけを発行できます。

データ・ディクショナリの構造

データベースのデータ・ディクショナリは、次の要素で構成されています。

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。これらの表に読み書きできるのは Oracle だけです。これらの表は標準化され、データのほとんどは、暗号形式で格納されているので、ユーザーが直接アクセスすることはめったにありません。

ユーザー・アクセス可能ビュー	データ・ディクショナリの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 句を使っ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（更新または削除、挿入）しないでください。そのような操作により、データの整合性が損なわれることがあります。セキュリティ管理者は、このアカウントを厳しく管理する必要があります。

警告: 基礎となるデータ・ディクショナリ表のデータを変更したり操作したりすると、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・ディクショナリが使用される方法

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、ユーザーおよびスキーマ・オブジェクト、記憶構造に関する情報を検索するためにデータ・ディクショナリにアクセスする。
- 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する。
- Oracle ユーザーは、データベースについての情報の読込み専用リファレンスとしてデータ・ディクショナリを使うことができる。

Oracle はどのようにデータ・ディクショナリを使うか

データ・ディクショナリの実表に入っているデータは、 Oracle が機能するために必要な情報です。したがって、データ・ディクショナリの情報を書き換えたり変更したりする作業は、Oracle だけが行うべきです。

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

たとえば、ユーザー KATHY が PARTS という表を作成すると、新しい表および列、セグメント、エクステント、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、システム全体で使うスキーマ・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・シノニムに使わ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にデータ・ディクショナリをキャッシュに格納する

ユーザー・アクセスの妥当性検査やスキーマ・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ので、データ・ディクショナリ情報の大部分は SGA （「ディクショナリ・キャッシュ」）内に格納されます。すべての情報は、 LRU （「最低使用頻度」）アルゴリズムを使ってメモリーに格納されます。

通常、キャッシュに保持されるのは、解析処理に必要な情報です。表とそれらの列について記述している COMMENTS 列は、頻繁にアクセスされない限りキャッシュには保持されません。

他のプログラムとデータ・ディクショナリ

他の Oracle 製品は、既存のビューを参照したり、独自の表またはビューを作成してデータ・ディクショナリ表に追加したりできます。ソフトウェアがリリース・アップしても、シノニムが変更される可能性は低いので、データ・ディクショナリを参照するプログラムを作成するアプリケーションの開発者は、基礎となる表ではなくパブリック・シノニムを参照するべきです。

新しいデータ・ディクショナリ項目を追加する

新しい表またはビューをデータ・ディクショナリに追加できます。新しいオブジェクトをデータ・ディクショナリに追加する場合、その新しいオブジェクトの所有者はユーザー SYSTEM か、第三の Oracle ユーザーである必要があります。

注意： ユーザー SYS に属する新しいオブジェクトは絶対に作成しないでください。ただし、オラクル社が提供するスクリプトを実行して、データ・ディクショナリのオブジェクトを作成する場合は例外です。

データ・ディクショナリの項目を削除する

データ・ディクショナリに対するすべての変更は、 DDL 文への応答として Oracle が実行するので、どのユーザーも、データ・ディクショナリの表にあるデータを絶対に削除したり変更したりしないでください。

この規則の唯一の例外は、表 SYS.AUD$ です。監査が使用可能になっていると、この表は際限なく肥大化する可能性があります。 AUDIT_TRAIL 表は削除すべきではありませんが、この表の行は情報を提供するだけのもので、 Oracle の実行に必要ではないので、セキュリティ管理者はこの表のデータを削除できます。

Oracle ユーザーはどのようにデータ・ディクショナリを利用できるか

データ・ディクショナリのビューは、すべてのデータベース・ユーザーのためのリファレンスとしての役目を果たします。データ・ディクショナリ・ビューには、 SQL 言語を介してアクセスします。すべての Oracle ユーザーがアクセスできるビューもいくつかありますが、その他のビューは管理者だけが使う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、それぞれは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（すべてのユーザーがアクセスできる）

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 は含まれない。 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る。

追加情報: 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、『Oracle8 Server リファレンス・マニュアル』を参照してください。

接頭辞が USER のビュー

通常のデータベース・ユーザーにとって最も関心のあるのは、接頭辞が USER のビューでしょう。これらのビューには次のような特徴があります。

- ユーザーが作成したスキーマ・オブジェクトやユーザーによる権限付与に関する情報など、ユーザー独自のプライベートなデータベース環境を参照する。
- ユーザーに関係する行だけを表示する。
- 列 OWNER が暗黙的に想定される（現ユーザー）ことを除いて、他のビューと同一の列を持っている。

- ALL_ ビューにある情報のサブセットを戻す。
- 使いやすいように短縮した PUBLIC シノニムを持つことができる。

たとえば、次の問合せは、自分のスキーマに入っているすべてのオブジェクトを戻します。

```
SELECT object_name, object_type FROM user_objects;
```

接頭辞が ALL のビュー

接頭辞が ALL のビューは、ユーザーのデータベース全体の概要を参照します。これらのビューは、ユーザーが所有しているスキーマ・オブジェクトに加えて、権限とロールの PUBLIC への付与や明示的な付与によってそのユーザーがアクセスできるようになったスキーマ・オブジェクトに関する情報を戻します。たとえば次の問合せは、アクセス権を持っているすべてのオブジェクトに関する情報を戻します。

```
SELECT owner, object_name, object_type FROM all_objects;
```

接頭辞が DBA のビュー

接頭辞が DBA のビューには、データベース全体のグローバル・ビューが示されます。したがって、これらのビューに問合せを発行できるのはデータベース管理者だけです。システム権限 SELECT ANY TABLE を付与されているユーザーは、データ・ディクショナリの接頭辞が DBA のビューに問合せを発行できます。

DBA ビューへの問合せは管理者によってのみ発行されるべきなので、これらのビューのシノニムは作成されません。このため、DBA ビューに問合せを発行するには、管理者は、次のようにして所有者 SYS をビューの名前の接頭辞として指定する必要があります。

```
SELECT owner, object_name, object_type FROM sys.dba_objects;
```

SELECT ANY TABLE システム権限がある管理者は、スクリプト・ファイル DBA_SYNONYMS.SQL を使って、自分のアカウントに DBA ビューのプライベート・シノニムを作成できます。このスクリプトを実行して作成されるのは、現ユーザーだけに対するシノニムです。

DUAL

表 DUAL は、既知の結果を保証するために、 Oracle とユーザー作成のプログラムによって参照される小さな表です。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。

> **追加情報** : DUAL 表の詳細は、『 Oracle8 Server SQL リファレンス』の SELECT コマンドの説明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、現在のデータベース・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」といいます。

動的パフォーマンス表は実際の表ではないので、ほとんどのユーザーはこれにアクセスするべきではありません。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したりすることができます。

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ています。これらのシノニム名は、 V$ で始まります。たとえば、 V$DATAFILE にはデータベースのデータ・ファイルに関する情報が格納され、 V$FIXED_TABLE 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。

追加情報: 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、『 Oracle8 Server リファレンス・マニュアル』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 Server インスタンスを構成しているメモリー構造とプロセスについて説明し、 Oracle インスタンスを起動したり停止したりする方法を説明します。

第 III 部には、次の章が含まれています。

- 第 5 章「データベースとインスタンスの起動と停止」
- 第 6 章「メモリー構造」
- 第 7 章「プロセスの構造」

5

データベースとインスタンスの起動と停止

Greetings, Prophet;
The Great Work begins:
The Messenger has arrived.

Tony Kushner: *Angels in America*, Part I

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

- Oracle インスタンスの概要
  - 管理者権限での接続
  - パラメータ・ファイル
- インスタンスとデータベースの起動
- データベースとインスタンスの停止

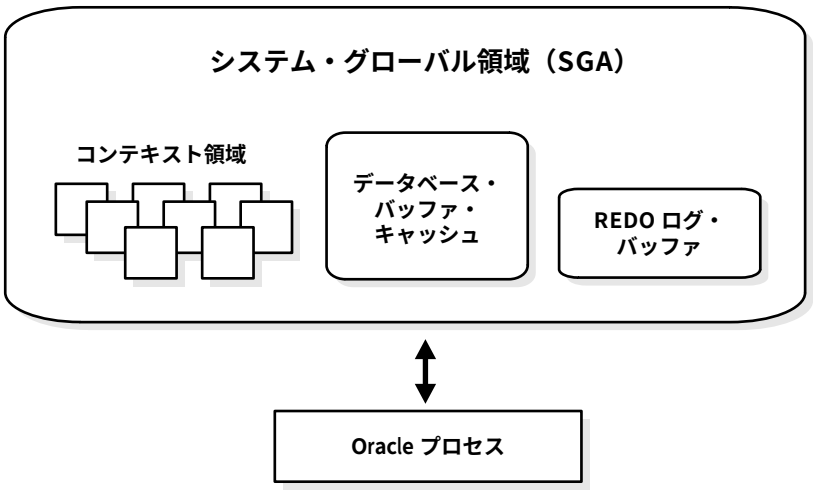
追加情報: Trusted Oracleを使っている環境での起動と停止の詳細は、Trusted Oracle のマニュアルを参照してください。

Oracle インスタンスの概要

稼動中のすべての Oracle データベースは、 Oracle インスタンスに対応付けられます。データベース・サーバー上のデータベースが起動すると、コンピュータの種類にかかわらず、 Oracle はシステム・グローバル領域（ SGA ）というメモリー領域を割り当て、 1 つ以上の Oracle プロセスを起動します。この SGA と Oracle プロセスの組合せのことを、「 Oracle インスタンス」といいます。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、データベースを使う 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。 SGA と Oracle プロセスの詳細は、第 6 章「メモリー構造」および第 7 章「プロセスの構造」を参照してください。

図 5-1 Oracle **インスタンス**



インスタンスとデータベース

インスタンスを起動した後、 Oracle は指定されたデータベースにインスタンスを関連付けます。これをデータベースの「マウント」といいます。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させて、それぞれ自分の物理データベースを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な並列処理システム（ MPP ）では、 Oracle Parallel Server により、 1 つのデータベースに複数のインスタンスをマウントできるようになります。

追加情報: Oracle Parallel Serverの詳細は、『Oracle8 Parallel Server概要および管理』を参照してください。

Trusted Oracle を使っている環境でのインスタンスの詳細は、 Trusted Oracle のマニュアルを参照してください。

データベース管理者だけが、インスタンスを起動してデータベースをオープンすることができます。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することができます。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、管理者権限を使って 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 データベースの状態を制御できません。

管理者権限での接続

データベースの起動と停止は強力な管理オプションであり、管理者権限を使って Oracle に接続するユーザーだけがこれを行うことができます。オペレーティング・システムに応じて、ユーザーの管理権限を確立するには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って接続できる。
- 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・ファイルを使ってデータベース管理者を認証している。
- データベースに INTERNAL ログインのパスワードがあり、ユーザーがそのパスワードを知っている。

追加のセキュリティとして、ユーザーが管理者権限で接続できるのは専用サーバーだけです（共有サーバーには接続できません）。

管理者権限で接続すると、そのユーザーは SYS が所有するスキーマ内に置かれます。これによって、 SYS スキーマ内のすべてのオブジェクトにアクセスできるようになります。

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 25 章「データベース・アクセスの制御」を参照してください。

追加情報: 各オペレーティング・システムで管理者権限がどのような働きをするかの詳細は、オペレーティング・システム別の Oracle マニュアルを参照してください。

パラメータ・ファイル

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

- 名前（ファイルなどの）を指定するパラメータ

- 制限（最大数など）を設定するパラメータ
- SGA のサイズなどの容量に影響するパラメータ。これを「可変パラメータ」といいます

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うメモリーの容量
- 満杯になったオンライン REDO ログ・ファイルの処理方法
- データベースの制御ファイルの名前と位置
- データベースのプライベート・ロールバック・セグメントの名前

パラメータ・ファイルの例

パラメータ・ファイルの例を次に示します。

```
db_block_buffers = 550
db_name = ORA8PROD
db_domain = US.ACME.COM
#
license_max_users = 64
#
control_files = filename1, filename2
#
log_archive_dest = c:¥logarch
log_archive_format = arch%S.ora
log_archive_start = TRUE
log_buffer = 64512
log_checkpoint_interval = 256000
# rollback_segments = rs_one, rs_two
```

Oracle は、このファイルの各国語サポート（ NLS ）パラメータに定義されている文字列リテラルを、データベースのキャラクタ・セットとして取り扱います。

パラメータ値の変更

データベース管理者は、データベース・システムのパフォーマンスを改善するために、可変パラメータを調整できます。どのパラメータがシステムに最も強い影響を与えるかは、データベースの数多くの特性や変数によって異なります。

変更されたパラメータ値は、インスタンスを起動してパラメータ・ファイルを読み込んだ後にのみ有効になります。いくつかのパラメータは、インスタンスの実行中にALTER SESSION コマンドまたは ALTER SYSTEM コマンドを使って「動的に」変更することもできます。

追加情報: すべての初期化パラメータの説明は、『Oracle8 Server リファレンス・マニュアル』を参照してください。 SGA に影響するパラメータの詳細は、 6-11 ページの「 SGA のサイズ」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、システム全体で使えるようにするには、次の 3 つのステップを実行します。

1. インスタンスを起動する。
2. データベースをマウントする。
3. データベースをオープンする。

データベース管理者は、Oracle Enterprise Managerを使ってこれらのステップを実行できます。

追加情報: 『Oracle Enterprise Manager 管理者ガイド』を参照してください。

インスタンスを起動する

Oracle は、インスタンスを起動するときに、まずパラメータ・ファイルを読み込んで初期化パラメータの値を判別し、次に SGA （データベース情報のために使わ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。

SGA の詳細は第 6 章「メモリー構造」を、バックグラウンド・プロセスの詳細は第 7 章「プロセスの構造」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したりできます。こうすると、接続はRESTRICTED SESSIONシステム権限を付与されているユーザーだけに制限されます。

異常な状況でのインスタンスの強制起動

異常な状況では、たとえばインスタンスのプロセスの 1 つが正常に終了していないなど、直前のインスタンスが「きれいに」停止されていないことがあります。そのような状況では、次回インスタンスを通常起動するときにデータベースからエラーが戻されます。この問題を解決するには、直前のインスタンスが残した Oracle プロセスの残骸をすべて終了してから、新しいインスタンスを起動しなければなりません。

データベースをマウントする

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ったパラメータ・ファイルの CONTROL_FILES 初期化パラメータに指定されています。）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ので、データベース管理者だけ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。

Parallel Server によるデータベースのマウントのモード

注意：この章で説明している機能は、 Parallel Server オプション付きの Oracle8 Enterprise Edition を購入した場合にのみ利用できます。詳細は、『Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

Oracle で複数のインスタンスに同じデータベースを同時にマウントできる場合、データベース管理者は、排他モードと共有モードのどちらでデータベースを実行するかを選択できます。

排他モード

データベースを最初にマウントしたインスタンスが排他モードの場合、データベースをマウントできるのはそのインスタンスだけです。 Parallel Server オプションをサポートしていないバージョンの Oracle では、インスタンスは排他モードでしかデータベースをマウントできません。

共有モード

データベースを最初にマウントしたインスタンスが共有モード（「パラレル・モード」ともいう）で起動されている場合、共有モードで起動された他のインスタンスもその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、事前に決められた最大数によって決まります。これは、データベースの作成時に指定できます。

追加情報：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、『Oracle8 Parallel Server 概要および管理』を参照してください。

データベースをオープンする

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。オープンされたデータベースに有効なユーザーが接続して、データベースの情報にアクセスできるようになります。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的な使用方法で使える状態にします。

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。3-7ページの「表領域をオンライン/オフラインにする」を参照してください。

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データベース・ファイルのバックアップから回復を実行する必要があります。

インスタンス回復

データベース管理者がインスタンスを異常終了させたため、または電源障害が発生したためにデータベースが異常にクローズされた場合、 Oracle はデータベースの再オープン時にインスタンス回復を自動的に実行します。 28-3 ページの「データベース・インスタンス障害」を参照してください。

ロールバック・セグメントの取得

データベースをオープンするときに、インスタンスは 1 つ以上のロールバック・セグメントを取得しようとします。2-22ページの「SYSTEMロールバック・セグメント」および「Oracle インスタンスとロールバック・セグメントのタイプ」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

ときには、データベースが異常にクローズし、 1 つ以上の分散トランザクションが「インダウト」（コミットもロールバックもされない状態）になることがあります。データベースを再オープンし、インスタンス回復が完了すると、 RECO バックグラウンド・プロセスがすぐ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。詳細は、第 30 章「分散データベース」を参照してください。

> **追加情報**: 分散トランザクションの障害からの回復の詳細は、『 Oracle8 Server 分散システム』を参照してください。

データベースとインスタンスの停止

データベースとそれに対応付けられたインスタンスを停止するには、次の 3 つのステップを実行します。

1. データベースをクローズする。
2. データベースをディスマウントする。
3. インスタンスを停止する。

データベース管理者は、Oracle Enterprise Managerを使ってこれらのステップを実行できます。インスタンスを停止するたびに、Oracleはこの3つのステップをすべて自動的に実行します。

> **追加情報**: 『 Oracle Enterprise Manager 管理者ガイド』を参照してください。

データベースをクローズする

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・データと回復のためのデータが、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。このように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ので、通常の操作は実行できません。データベースがクローズされても、まだマウントされていれば、制御ファイルはオープンされたままになります。

インスタンスの異常終了によるデータベースのクローズ

万一の非常時には、オープンされているデータベースのインスタンスを異常終了（中断）させて、データベースをクローズし、すぐ完全に停止させることができます。 SGA のバッファにあるすべてのデータをデータ・ファイルと REDO ログ・ファイルに書き込む操作が省略されるので、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 がインスタンス回復を自動的に実行します。

注意： データベースがオープンされているときにシステム・クラッシュや電源障害が発生すると、インスタンスは事実上「異常終了する」ため、データベースを再オープンした時点でインスタンス回復が実行されます。

データベースをディスマウントする

データベースがクローズされると、 Oracle はデータベースをディスマウントしてインスタンスから切り離します。この時点でインスタンスはコンピュータのメモリーに残留しています。

データベースをディスマウントすると、データベースの制御ファイルはクローズされます。

インスタンスを停止する

データベース停止の最後のステップは、インスタンスの停止です。インスタンスを停止すると、 SGA がメモリーから削除され、バックグラウンド・プロセスが停止します。

インスタンスの異常な停止

異常な状況として、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、インスタンスがきれいに停止されないことがあります。前のインスタンスの残骸が残っていると、その後にインスタンスを起動しようとしたとき、ほとんどの場合に失敗します。このような状況では、データベース管理者は、まず前のインスタンスの残骸を削除してから新しいインスタンスを起動するか、 Oracle Enterprise Manager で SHUTDOWN ABORT コマンドを発行して、新しいインスタンスの起動を強制実行することができます。

追加情報: インスタンスとデータベースの起動と停止の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

6

メモリー構造

Yea, from the table of my memory
I'll wipe away all trivial fondrecords.

Shakespeare: *Hamlet*

この章では、 Oracle データベース・システムにおけるメモリー構造とプロセスについて説明します。この章の内容は次のとおりです。

- Oracle メモリー構造の概要
- システム・グローバル領域（ SGA ）
- プログラム・グローバル領域（ PGA ）
- ソート領域
- 仮想メモリー
- ソフトウェア・コード領域

Oracle メモリー構造の概要

Oracle は、メモリーを使ってさまざまな情報を格納します。

- 実行中のプログラム・コード
- 接続されているセッションについての情報（セッションが現在アクティブでない場合も）
- プログラムの実行時に必要な情報（たとえば、行をフェッチしている問合せの現在の状態）
- Oracle プロセス間で共有され、やり取りされる情報（たとえば、ロック情報）
- 周辺メモリーにも永続的に格納されるキャッシュ・データ（たとえば、データ・ブロックや REDO ログ・エントリ）

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

- ソフトウェア・コード領域
- システム・グローバル領域（ SGA ）
  - データベース・バッファ・キャッシュ
  - REDO ログ・バッファ
  - 共有プール
- プログラム・グローバル領域（ PGA ）
  - スタック領域
  - データ領域
- ソート領域

システム・グローバル領域（ SGA ）

システム・グローバル領域（ SGA ）とは、共有メモリー構造の 1 つのグループで、 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入れられます。複数のユーザーが同じインスタンスに同時に接続している場合、そのインスタンスの SGA 内のデータは複数のユーザー間で「共有」されます。このため、 SGA は「共有グローバル領域」と呼ばれることもあります。

5-2 ページの「 Oracle インスタンスの概要」で説明したように、 SGA と Oracle プロセスが Oracle インスタンスを構成します。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、それ専用の SGA があります。

SGA は読取りと書込みが可能です。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 に入っている情報を読み込むことができます。また、Oracleの稼動中に、複数のプロセスがSGAに書込みを行うことができます。

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
- 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収められています。この部分を「固定 SGA 」と呼びます。ここには、ユーザー・データは格納されません。

SGA には、ロック情報などのプロセス間でやりとりされる情報も格納されます。

システムがマルチスレッド・サーバー・アーキテクチャを使っている場合、要求キューと応答キュー、およびプログラム・グローバル領域の内容の一部は SGA に格納されます。（ 6-12 ページの「プログラム・グローバル領域（PGA）」および7-20ページの「ディスパッチャの要求と応答キュー」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・ブロックのコピーを保持しておく領域です。インスタンスに同時接続しているすべてのユーザー・プロセスは、データベース・バッファ・キャッシュへのアクセスを共有します。

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

データベース・バッファ・キャッシュの編成

キャッシュ内のバッファは、2つのリスト（使用済みリストと最低使用頻度（LRU）リスト）に編成されます。「使用済みリスト」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が入ったバッファのことです。「最低使用頻度（ LRU ）リスト」は、空きバッファと使用中バッファ、そして使用済みリストへ移動していない使用済みバッファを保持します。空きバッファは、修正されていないバッファで、使うことができるバッファです。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をアクセスするとき、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。引き続き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はじめて必要になったとき、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・ブロックをコピーしなければなりません。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、まず空きバッファを見つける必要があります。プロセスは、 LRU リストの最低使用頻度側から検索を開始します。空きバッファが見つかるか、制限しきい値の数のバッファを検索するまで、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使用済みリストに移動してから検索を続けます。空き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。

Oracle ユーザー・プロセスが空きバッファを見つけられずに、制限しきい値の数のバッファを検索した場合、プロセスは LRU リストの検索を停止し、使用済みバッファのいくらか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。 DBW0 プロセス（または複数の DBWn プロセス）の詳細は、 7-8 ページの「データベース・ライター（ DBWn ）」を参照してください。

LRU アルゴリズムと全表走査

ユーザー・プロセスは、全表走査を実行するときに、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、全表走査された表は一時的に必要なものなので、もっと頻繁に使われるブロックをキャッシュ内に残しておくために全表走査された表のブロックをすぐにキャッシュから取り除く必要があるからです。

表走査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。全表走査時に表のブロックをリストのMRU側に置く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 句を使います。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。

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

データベース・バッファ・キャッシュのサイズ

初期化パラメータ DB_BLOCK_BUFFERS によって、データベース・バッファ・キャッシュ内のバッファ数を指定します。キャッシュ内の各バッファは、（初期化パラメータ DB_BLOCK_SIZE で指定される） 1 つの Oracle データ・ブロックと同じサイズです。そのため、キャッシュ内の各データベース・バッファには、データ・ファイルから読み込まれたデータ・ブロックを 1 つ保持できます。

キャッシュのサイズには制限があるため、ディスク上のすべてのデータをキャッシュに入れられるとは限りません。キャッシュが満杯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取り分けるために、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れば、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。

> **追加情報**：バッファ・キャッシュの詳細は、『Oracle8 Server チューニング』を参照してください。

複数のバッファ・プール

異なるバッファ・プールを持つデータベース・バッファ・キャッシュを構成して、バッファ・キャッシュ内にデータを保持するか、またはデータ・ブロックの使用直後に新しいデータがバッファを使えるようにすることができます。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、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する。
- 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する。
- 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・オブジェクトが入る。

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

> **追加情報**：バッファ・プールの詳細は、『Oracle8 Server チューニング』を参照してください。 STORAGE 句の BUFFER_POOL オプションの構文は、『Oracle8 Server 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が入っています。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 または UPDATE 、 DELETE 、 CREATE 、 ALTER 、 DROP の各操作によってデータベースに加えられた変更を再構築、または再実行するのに必要な情報が入ります。 REDO エントリは、必要に応じてデータベースの回復時に使われます。

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

> **追加情報**: 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、7-9ページの「ログ・ライター・プロセス(LGWR)」を参照してください。オンライン・ REDO ログ・ファイルおよびグループの詳細は、『 Oracle8 Server バックアップおよびリカバリ』を参照してください。

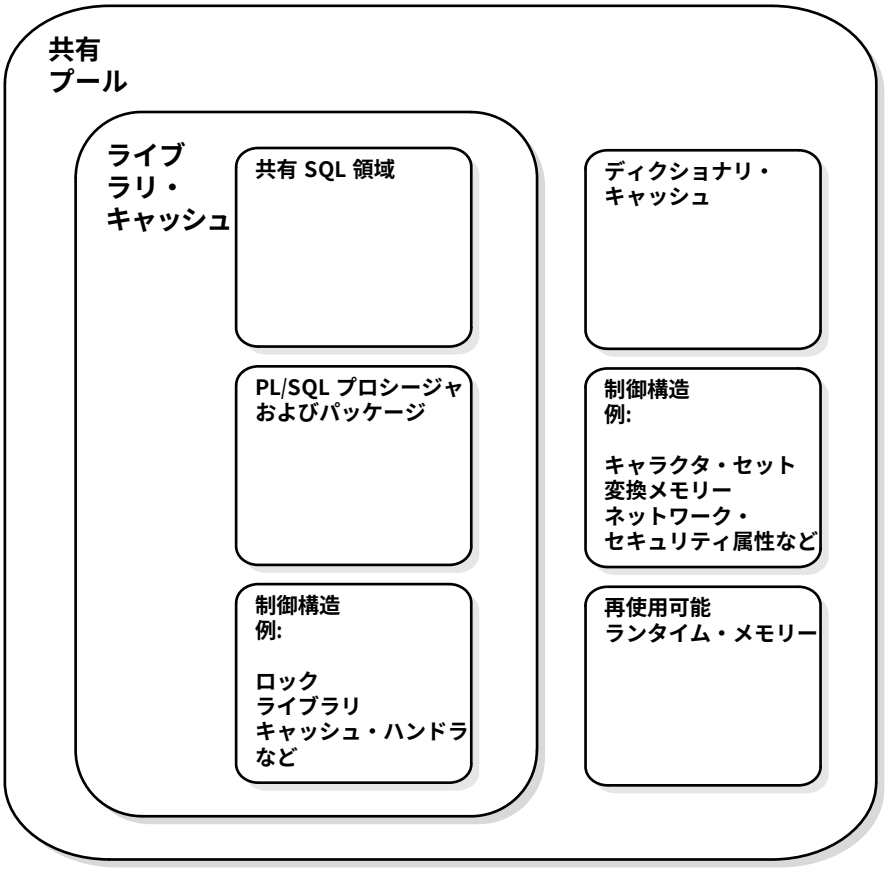
初期化パラメータ LOG_BUFFER は、 REDO ログ・バッファのサイズをバイト単位で設定します。一般的には、トランザクションが長いか、数が多い場合には特に、設定する値を大きくすればするほどログ・ファイルのI/Oは少なくなります。デフォルト設定は、ホスト・オペレーティング・システムの最大データ・ブロック・サイズの 4 倍です。

共有プール

SGA の共有プール部分には、 3 つの主な領域（ライブラリ・キャッシュ、ディクショナリ・キャッシュ、制御構造）が含まれています。図 6-1 に、共有プールの内容を示します。

共有プールの合計サイズは、初期化パラメータ SHARED_POOL_SIZE によって決まります。このパラメータのデフォルト値は 3,500,000 バイトです。このパラメータの値を大きくすると、共有プール用に確保されるメモリーの量が増加し、したがって共有 SQL 領域用に確保される領域が増加します。

図 6-1 共有プールの内容



ライブラリ・キャッシュ

ライブラリ・キャッシュには、共有 SQL 領域、プライベート SQL 領域、 PL/SQL プロシージャおよびパッケージのほか、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入ります。

複数のユーザーが共有 SQL 領域を使えるようにしなければならないため、ライブラリ・キャッシュは SGA 内の共有プールに含まれます。ライブラリ・キャッシュのサイズは、データ・ディクショナリ・キャッシュと共に、共有プールのサイズによって制限を受けます。

共有 SQL 領域とプライベート SQL 領域

Oracle は、実行するそれぞれの SQL 文を、「共有 SQL 領域」と「プライベート SQL 領域」によって表現します。Oracleは、2人のユーザーが同じSQL文を実行する場合にそれを認識し、これらのユーザーのために共有 SQL 領域を再利用します。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っていなければなりません。

共有 SQL 領域

共有 SQL 領域には、 1 つの SQL 文または同一の SQL 文に関する解析ツリーと実行計画が入ります。 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、複数の同一の DML 文のために 1 つの共有 SQL 領域を使うことによってメモリーを節約します。共有 SQL 領域は、常に共有プール内にあります。

> **追加情報**: 同一のSQL文を判別するための基準については、『Oracle8 Server チューニング』を参照してください。

SQL 文を解析するときに、 Oracle は共有プールからメモリーを割り当てます。このメモリーのサイズは、文の複雑さの程度によって異なります。 SQL 文が新しい共有 SQL 領域を必要とし、共有プール全体が割り当て済みの場合、 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、修正した最低使用頻度アルゴリズムを使ってその共有プールから項目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。

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実行時バッファを格納します。 SQL 文を発行するそれぞれのセッションには、プライベート SQL 領域があります。同一の SQL 文を送る各ユーザーは、 1 つの共有 SQL 領域を使う自分のプライベート SQL 領域を持っています。つまり、多くのプライベート SQL 領域を、同じ共有 SQL 領域に対応付けることができます。（セッションの詳細は、 7-4 ページの「接続とセッション」を参照してください。）

プライベート SQL 領域は、さらに持続領域と実行時領域に分けられます。

- 「持続領域」には、複数の実行にまたがって持続するバインド情報、データ型変換のコード（定義したデータ型が、選択した列のデータ型と異なる場合）、その他の状況情報（再帰カーソルまたはリモート・カーソルの番号や、パラレル問合せの状況）が格納されます。持続領域のサイズは、文で指定されているバインドおよび列の数によって異なります。たとえば、問合せで多くの列が指定されると、持続領域が大きくなります。
- 「実行時領域」には、 SQL 文の実行時に使う情報が入ります。実行時領域のサイズは、実行する SQL 文のタイプと複雑さの程度、そしてその文で処理する行のサイズによって異なります。一般には、 INSERT 文および UPDATE 文、 DELETE 文の実行時領域は、SELECT 文の実行時領域よりも多少小さくなります。

Oracleは、実行要求の最初のステップで実行領域を作ります。INSERT文およびUPDATE文、DELETE文では、Oracleは文の実行後に実行時領域を解放します。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だけ、実行時領域を解放します。

プライベート SQL 領域の位置は、セッションのために確立される接続のタイプによって異なります。セッションが専用サーバーを介して接続されている場合、プライベート SQL 領域はユーザーの PGA 内に位置します。しかし、セッションがマルチスレッド・サーバーを介して接続されている場合、持続領域、および実行時領域（SELECT 文の場合）は SGA 内に位置します。

カーソルと SQL 領域

Oracle プリコンパイラ・プログラムまたは 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えます。 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 領域を使います。詳細は、 14-6 ページの「カーソル」を参照してください。

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、どのアプリケーション・ツールを使っているかに大きく依存します。このパラメータのデフォルト値は 50 です。

プライベート SQL 領域は、対応するカーソルがクローズされるか、文ハンドルが解放されるまで存在します。 Oracle は文の実行が完了した後に実行時領域を解放しますが、持続領域は待機し続けます。持続領域を解放して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び使うことがないものをすべてアプリケーションの開発者側でクローズする必要があります。

ソートを必要とする大量のデータ処理を行う問合せでは、部分的な 1 回のフェッチの結果でクライアントの要求が満たされるのであれば、アプリケーションの開発者側でその問合せを取り消さなければなりません。たとえば、Oracle Officeアプリケーションでは、メール・メッセージを作るときに 60 個以上のテンプレートのリストからユーザーが選択できます。 Oracle Office が最初の 10 個のテンプレート名を表示し、ユーザーがテンプレートの 1 つを選択した場合、アプリケーションはさらに他のテンプレート名を表示するのではなく、残りの問合せの処理を取り消すべきです。

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、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された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、そしてパッケージ変数（パッケージ・インスタンシエーションとも呼ばれる）、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われますが、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い、その文を実行するセッションごとにプライベート SQL 領域を使います。

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、そしてそのユーザーについての参照情報を収録した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。詳細は、第 4 章「データ・ディクショナリ」を参照してください。

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・データを入れておくため、メモリー内に 2 箇所の特別な場所が指定されています。 1 つの領域は、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ので、「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・データを入れておくメモリー内のもう 1 つの領域は、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。

共有プール内のメモリーの割当てと再使用

一般に、共有プール内のあらゆる項目（共有SQL領域やディクショナリ行）は、修正LRUアルゴリズムに基づいてフラッシュされるまでは、そのまま存在します。新しい項目に領域を割り当てるために共有プール内にさらに領域が必要になると、定期的には使われていない項目のメモリーが解放されます。修正 LRU アルゴリズムを使うことにより、多数のセッションが使っている共有プール項目は、元々その項目を作成したプロセスが終了したとしても、その項目が使われている限りメモリー内に残ります。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は、最小限に抑えられます。

SQL 文が実行のために Oracle に送られると、 Oracle は次のメモリー割当てステップを自動的に実行します。

1. 同一の文について共有 SQL 領域がすでに存在しているかどうかを確認するため、共有プールをチェックします。その文のための共有 SQL 領域がすでに存在する場合、その領域は、その文の後続の新しいインスタンスを実行するために使われます。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るとしても、それがしばらく使われていないのであれば、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われる場合、その文は再解析されて新しい共有 SQL 領域が共有プール内に割り当てられます。

2. セッションのためにプライベート SQL 領域を割り当てます。プライベート SQL 領域の正確な位置は、セッションのために確立された接続によって異なります（ 6-8 ページの「共有 SQL 領域とプライベート SQL 領域」を参照してください）。

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。

- 表またはクラスタ、索引の統計を更新または削除するために ANALYZE コマンドを使う場合、分析されたスキーマ・オブジェクトを参照する文が入っているすべての共有 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、そのオブジェクトが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。 SQL 文の無効化と依存性の問題の詳細は、第 19 章「 Oracle の依存性の管理」を参照してください。
- データベースのグローバル・データベース名を変更すると、すべての情報が共有プールからフラッシュされます。
- 管理者はインスタンス起動後のパフォーマンス（データ・バッファ・キャッシュではなく、共有プールに関するパフォーマンス）を査定するために、共有プール内のすべての情報を手動でフラッシュできます。こうすれば、現行のインスタンスを停止する必要がありません。

SGA のサイズ

SGA のサイズは、インスタンスの起動時に決定されます。多くのシステムで最適なパフォーマンスを実現するには、 SGA 全体が実メモリーに入らなければなりません。 SGA 全体が実メモリーに収まらず、その一部を格納するのに仮想メモリーが使われる場合（ 6-16 ページの「仮想メモリー」を参照）、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（ディスクに対する読み書き）を実行するため、データベース・システム全体のパフォーマンスがかなり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を与えます。詳細は、『 Oracle8 Server チューニング』を参照してください。

SGA のサイズは、いくつかの初期化パラメータによって決定されます。次のパラメータは、SGA サイズに最も影響を与えます。

DB_BLOCK_SIZE	1 つのデータ・ブロックおよびデータベース・バッファのサイズ（単位はバイト）。
DB_BLOCK_BUFFERS	SGA に割り当てられるデータベース・バッファの数。それぞれのバッファのサイズは、 DB_BLOCK_SIZE で指定したサイズです。（SGA内のデータベース・バッファ・キャッシュに割り当てられる領域の総量は、「 DB_BLOCK_SIZE × DB_BLOCK_BUFFERS 」になります。）
LOG_BUFFER	REDO ログ・バッファに割り当てられるバイト数。
SHARED_POOL_SIZE	共有 SQL 文と共有 PL/SQL 文に割り当てられる領域のサイズ（単位はバイト）。

Oracle Enterprise Manager（またはServer Manger）を使っているときには、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。Server Managerコマンドの SHOW を SGA オプションとともに使うことにより、現行のインスタンスの SGA サイズを表示することもできます。

> **追加情報**: Oracle Enterprise Manager（またはServer Manager）でのSGAサイズの表示の詳細は、『Oracle Enterprise Manager管理者ガイド』を参照してください。
>
> 上記の初期化パラメータとそれらのパラメータがSGA に与える影響の詳細は、『 Oracle8 Server チューニング』を参照してください。使用しているオペレーティング・システム別の情報は、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・ガイドも参照してください。

SGA のメモリーの使用方法の制御

いくつかの初期化パラメータを使って、 SGA がどのようにメモリーを使うかを制御できます。これらのパラメータの詳細は、『 Oracle8 Server リファレンス・マニュアル』を参照してください。

物理メモリー

LOCK_SGA パラメータと LOCK_SGA_AREAS パラメータは、 SGA 全体または特定の SGA 領域を物理メモリーにロックします。

SGA 開始アドレス

SHARED_MEMORY_ADDRESS パラメータと HI_SHARED_MEMORY_ADDRESS パラメータは、実行時のSGAの開始アドレスを指定します。これらのパラメータは、リンク時にSGAの開始アドレスを指定しないプラットフォーム上でのみ使います。 64 ビットのプラットフォームでは、 HI_SHARED_MEMORY_ADDRESS は 64 ビット・アドレスの上位 32 ビットを指定します。

拡張バッファ・キャッシュ・メカニズム

USE_INDIRECT_DATA_BUFFERSパラメータは、 4 GBを超える物理メモリーをサポートできる 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。

プログラム・グローバル領域（ PGA ）

プログラム・グローバル領域（ PGA ）とは、 1 つのプロセス（サーバーまたはバックグラウンド）のデータと制御情報が含まれるメモリー領域のことです。そのため、 PGA は「プロセス・グローバル領域」と呼ばれることもあります。

PGA は、プロセスが書き込める、非共有のメモリー領域です。サーバー・プロセスごとに 1 つのPGAが割り当てられます。このPGAはそのサーバー・プロセスに対して排他的であり、そのプロセスのために活動している Oracle コードによってだけ読み書きされます。

オペレーティング・システムと構成によって異なりますが、ユーザーが Oracle データベースに接続して、セッションが確立されると、 Oracle によって PGA が割り当てられます。（セッションの詳細は、 7-4 ページの「接続とセッション」を参照してください。）

PGA の内容

PGA の内容は、対応するインスタンスがマルチスレッド・サーバーを実行しているかどうかによって異なります。（マルチスレッド・サーバーの詳細は、7-19ページの「マルチスレッド・サーバー」を参照してください。

スタック領域

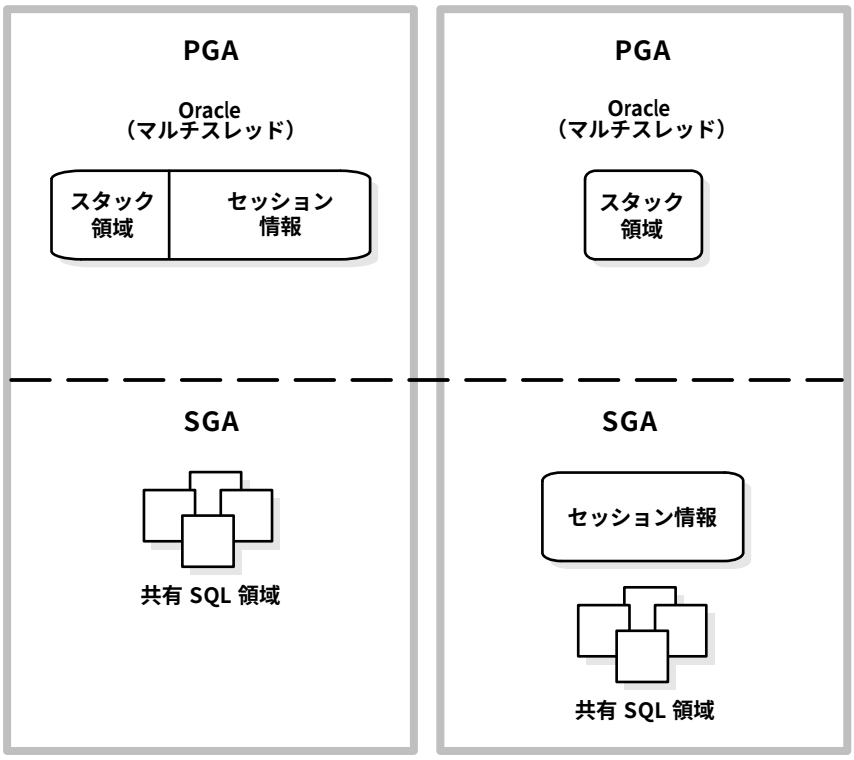
PGAには、「スタック領域」が必ず含まれます。スタック領域とは、セッションの変数、配列、その他の情報を保持しておくために割り当てられるメモリーです。

セッション情報

インスタンスがマルチスレッド・サーバーなしで実行されている場合、 PGA にはプライベート SQL 領域など、ユーザーのセッションに関する情報も格納されます。そのインスタンスがマルチスレッド・サーバー構成で実行されている場合、このセッション情報は PGA 内には存在しませんが、そのかわりに SGA 内に割り当てられます。

図 6-2 に、さまざまな構成でセッション情報が格納される場所を示します。

図 6-2 マルチスレッド・サーバーがある場合とない場合のセッション情報の位置



PGA のサイズ

PGA のサイズは固定であり、オペレーティング・システムによって異なります。クライアントとサーバーが異なるマシン上に存在するとき、 PGA は接続時にデータベース・サーバー上に割り当てられます。接続時に十分なメモリーが使えなければ、そのオペレーティング・システム・エラーを示す範囲のエラー番号の Oracle エラーが発生します。いったん接続されると、 PGA 領域が使い果たされることはありません。つまり、接続時に十分なメモリーが存在しているか、メモリーが不足しているかのどちらかです。

次の初期化パラメータが PGA のサイズに影響を与えます。

- OPEN_LINKS
- DB_FILES
- LOG_FILES

Oracle バックグラウンド・プロセス（ DBW0 や LGWR など）のために作られる各 PGA 内のスタック領域のサイズは、いくつかの追加パラメータの影響を受けます。

追加情報：PGA の詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

ソート領域

ソートを実行するには、メモリー内に領域が必要です。 Oracle がデータのソートを実行するメモリー内の部分を「ソート領域」といいます。ソート領域は、ソートを要求したOracleユーザー・プロセスのメモリー内にあります。

ソート領域は、ソートされるデータの量を収容するために大きくなることがありますが、初期化パラメータSORT_AREA_SIZEの値によって制限を受けます。バイト数で表現されるデフォルト値は、オペレーティング・システムによって異なります。

ソート処理中に、 Oracle はソート領域内のデータを参照する必要のないタスクをいくつか実行することがあります。そのような場合、 Oracle はデータの一部をディスク上の一時セグメントに書き込み、そのデータを含むソート領域の一部を割当て解除することにより、ソート領域のサイズを小さくします。たとえば、 Oracle がアプリケーションに制御を戻す場合に、このような割当て解除が生じる可能性があります。

初期化パラメータ SORT_AREA_RETAINED_SIZE で指定されているサイズまで、ソート領域のサイズは小さくなります。このパラメータの値は、バイト単位で指定します。最小値は、データベース・ブロック 1 つ分のサイズです。最大（デフォルト）値は、 SORT_AREA_SIZE 初期化パラメータで指定した値です。

ソート時に解放されたメモリーは、同一の Oracle プロセスで使えますが、オペレーティング・システムに対して解放されるわけではなりません。

ソートするデータの量がソート領域に収まらない場合、収めることができるように、データはより小さな断片に分けられます。続いて、各断片は個々にソートされます。個々にソートされた断片のことを、「ラン（run）」といいます。すべてのランがソートされた後に、それらがマージされて最終結果が生成されます。

ソート・ダイレクト書込み

ソート・ダイレクト書込みは、ソート領域のサイズに基づいて、ダイレクト書込みバッファのサイズと個数を導き出す自動チューニングの手段を提供します。このバッファ用のメモリーは、ソート領域から取得されるので、必要なチューニング・パラメータは1つだけです。また、オプティマイザ・コスト・モデルも提供されます。

メモリーおよび一時領域がシステム上に十分にあるときにディスクへの大規模なソートを多数実行する場合、初期化パラメータ SORT_DIRECT_WRITES の設定値によってソートのパフォーマンスを向上させられます。

追加情報：ソート領域と SORT_DIRECT_WRITES パラメータの詳細は、『 Oracle8 Server チューニング』を参照してください。

仮想メモリー

多くのオペレーティング・システム上で、 Oracle は「仮想メモリー」を利用します。仮想メモリーとは、オペレーティング・システムの機能の 1 つであり、実メモリー単独によって提供されるメモリーよりも多くの見かけ上のメモリーを提供することにより、主メモリーをより柔軟に使えるようにする機能です。

仮想メモリーは、実（主）メモリーと2次記憶領域（通常はディスク領域）の組合せを使って、メモリーをシミュレートします。オペレーティング・システムは、アプリケーション・プログラムに対して 2 次記憶領域を主メモリーのように見せることによって、仮想メモリーにアクセスします。

提案: 通常、実メモリー内に SGA 全体を置くと効率が最も良くなります。多くのプラットフォームでは、 LOCK_SGA パラメータと LOCK_SGA_AREAS パラメータを使って、 SGA またはその一部を実メモリーにロックできます。

ソフトウェア・コード領域

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

ソフトウェア領域のサイズは通常固定されており、ソフトウェアの更新時か再インストール時に限り変化します。これらの領域に必要なサイズは、オペレーティング・システムによって異なります。

ソフトウェア領域は読込み専用であり、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・プログラムは共有することも共有しないこともできます。 SQL*Forms や SQL*Plus などの Oracle Tools およびユーティリティによっては、共有でインストールできるものもありますが、共有にできないものもあります。 Oracle の複数のインスタンスが同じコンピュータ上で実行されている場合、それらのインスタンスは異なるデータベースにおいても同じ Oracle コード領域を使えます。

追加情報: 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・システムで使えるわけではありません（たとえば、MS-DOS が稼働している PC では使えません）。詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

7

プロセスの構造

If the good people, in their wisdom, shall see fit to keep me in thebackground, I have been too familiar with disappointments to be verymuch chagrined.

Abraham Lincoln, *Address at New Salem (1832)*

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
- シングル・プロセスの Oracle
- マルチ・プロセスの Oracle
- Oracle の構成におけるバリエーション
- Oracle の動作例
- 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・モジュールを実行しなければなりません。

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

これらのコード・モジュールは、プロセスによって実行されます。「プロセス」は、「制御のスレッド」、つまり一連のステップを実行できるオペレーティング・システムのメカニズムです。（オペレーティング・システムによっては、「ジョブ」または「タスク」という用語を使います。）プロセスには、通常、プロセスそのものを実行するための専用のプライベート・メモリー領域があります。

プロセス構造は、 Oracle 構成およびオペレーティング・システム、選択する Oracle オプションによって異なります。

シングル・プロセスの Oracle

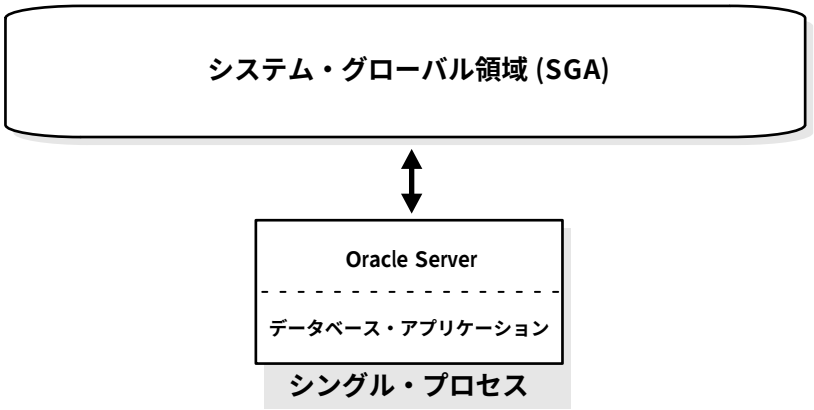
「シングル・プロセス Oracle 」（「シングル・ユーザー Oracle 」とも呼ばれる）は、 Oracle Server のコードと 1 人のユーザーのアプリケーション・プログラムのすべてが 1 つのプロセスによって実行されるデータベース・システムです。 Oracle インスタンスの部分とクライアント・アプリケーション・プログラムの部分の実行を分けるために別のプロセスが使われることはありません。

図 7-1 に、シングル・プロセスの Oracle インスタンスを示します。 1つのプロセスが、データベース・アプリケーションおよび Oracle に関連したすべてのコードを実行します。

シングル・プロセス環境における Oracle インスタンスは、 1 人のユーザーだけがアクセスできます。つまり、複数のユーザーが同時にデータベースにアクセスすることはできません。たとえば、 MS-DOS オペレーティング・システムでは複数のプロセスを実行できないので、 PC 上の MS-DOS で稼動する Oracle にアクセスできるのは、 1 人のユーザーだけです。

この構成は、通常はデータベース操作に関連付けられる分散処理を利用しないので、マルチ・プロセスの Oracle ほど一般的ではありません。

図 7-1 シングル・プロセスの Oracle インスタンス



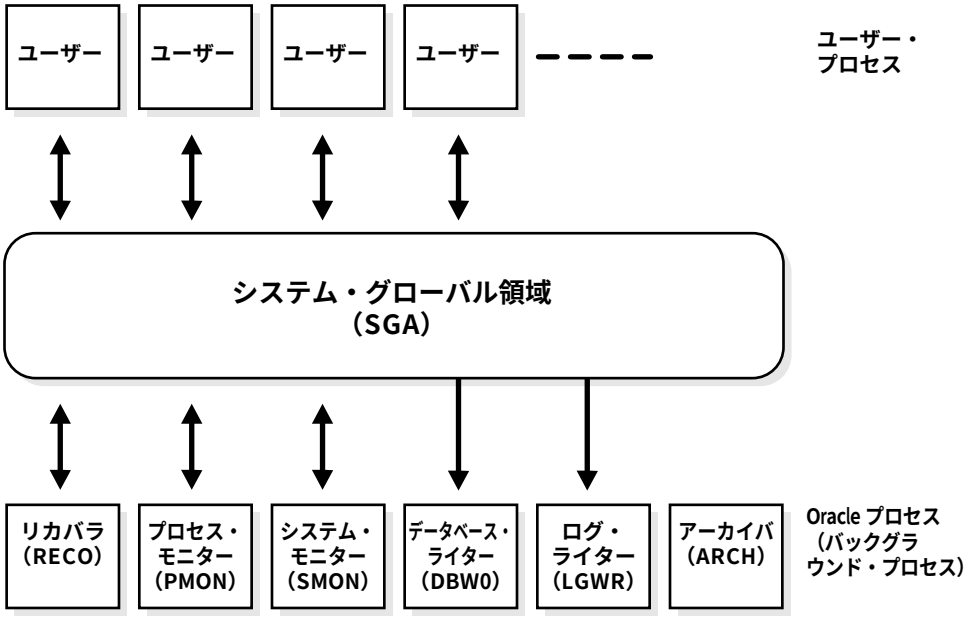
マルチ・プロセスの Oracle

「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、接続されているユーザーごとの個別のプロセスを実行するために、複数のプロセスを使います。マルチ・プロセスの 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も、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。

データベースの主な利点の 1 つは、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ので、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。

図 7-2に、マルチ・プロセスのOracleインスタンスを示します。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。この図は、 Oracle システムとして、同じマシン上でアプリケーションを実行している複数の同時実行ユーザーを表しています。このような構成は通常、メインフレームやミニコンピュータ上で稼動します。

図 7-2 マルチ・プロセスの Oracle インスタンス



マルチ・プロセス・システムでは、プロセスを 2 つのグループ（ユーザー・プロセスと Oracle プロセス）に分類することができます。ユーザー・プロセスは、アプリケーションまたは Oracle Tool のコードを実行し、 Oracle プロセスは Oracle Server のコードを実行します。

ユーザー・プロセス

ユーザーがアプリケーション・プログラム（ Pro*C プログラムなど）または Oracle Tool（Oracle Enterprise Manager や SQL*Plus など）を実行すると、 Oracle はユーザーのアプリケーションを実行するために「ユーザー・プロセス」を作成します。

接続とセッション

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

「接続」とは、ユーザー・プロセスと Oracle インスタンスの間の通信路のことです。通信路は、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）を使うことにより確立されます。

「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 インスタンスへ接続するときの、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しなければなりません。そうすると、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、それらのユーザーを同じユーザー名で同時に存在させることができます。たとえば、 SCOTT/TIGER というユーザー名 / パスワードを持つユーザーは、同じ Oracle インスタンスに何度も接続できます。

マルチスレッド・サーバーを使っていない構成では、 Oracle はユーザー・セッションごとにサーバー・プロセスを 1 つずつ作ります。しかし、マルチスレッド・サーバーを使っている場合は、1つのサーバー・プロセスを多数のユーザー・セッションで共有できます。詳細は、7-19 ページの「マルチスレッド・サーバー」を参照してください。

Oracle プロセス

マルチ・プロセス・システムにおいては、 Oracle は、 2 つのタイプのプロセス（サーバー・プロセスとバックグラウンド・プロセス）によって制御されます。

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・プロセスが作られます。アプリケーションと Oracle が同一のマシン上で稼動しているときには、システムのオーバーヘッドを少なくするために、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。しかし、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、ユーザー・プロセスは独立したサーバー・プロセスを通じて Oracle と通信します。詳細は、7-15 ページの「 Oracle の構成におけるバリエーション」を参照してください。

サーバー・プロセス

各ユーザー・アプリケーションのために作られた「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、実行する。
- 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・バッファに読み込む。
- 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、かつ多くのユーザーが使えるようにするため、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 プロセスを使います。

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られます。その他のオペレーティング・システムでは、サーバー・プロセスが、 Oracle のインストールの一部として作られます。

追加情報: プロセスが作られる方法の詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

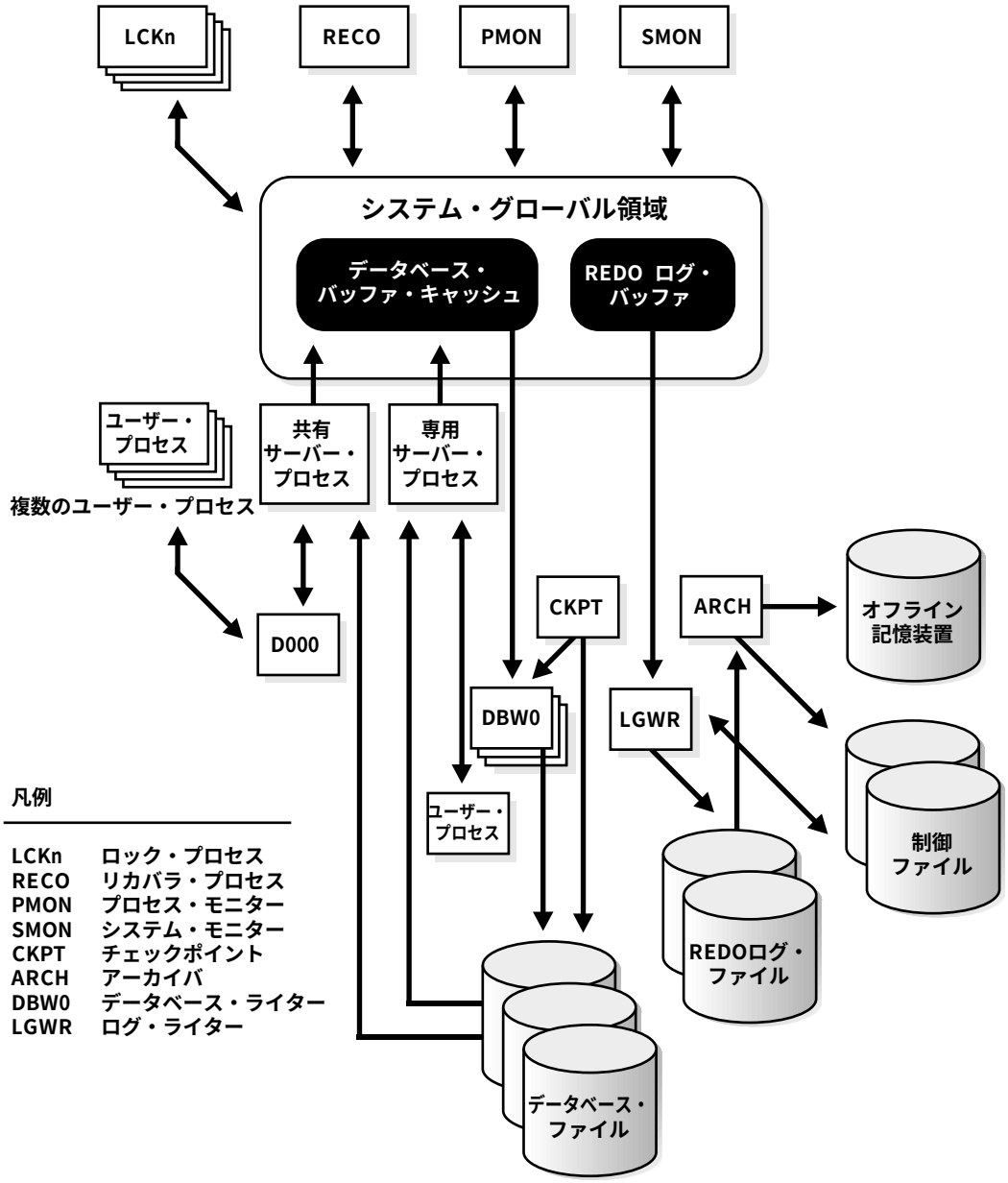
1 つの Oracle インスタンスは、複数のバックグラウンド・プロセスを持つことができます。しかし、常にすべてが存在するわけではありません。 Oracle インスタンスのバックグラウンド・プロセスには、次のものがあります。

- データベース・ライター（ DBW0 または DBWn ）
- 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（ ARCH ）
- リカバラ（ RECO ）
- ロック（ LCKn ）
- ジョブ・キュー（ SNPn ）
- キュー・モニター（ QMNn ）
- ディスパッチャ（ Dnnn ）
- サーバー（ Snnn ）

図 7-3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。その後の項で、各プロセスについて説明します。

追加情報: 図 7-3には、 Oracle Parallel Serverは示されていません。詳細は、『 Oracle8 Parallel Server 概要および管理』を参照してください。

図 7-3 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（ DBWn ）

「データベース・ライター・プロセス（DBWn）」は、バッファの内容をデータ・ファイルに書き込みます。DBWn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。（ 6-3 ページの「データベース・バッファ・キャッシュ」を参照してください。）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 ）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 DBWn プロセスは、単一プロセッサ・システムでは効果がありません。

データベース・バッファ・キャッシュ内のバッファが変更されると、そのバッファには「使用済み」のマークが付きます。 DBWn プロセスの主な機能は、使用済みバッファをディスクに書き込むことによって、バッファ・キャッシュを「クリーン」な状態に保つことです。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、空きバッファの数が減少します。空き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ユーザー・プロセスは、空きバッファを見つけることができません。ユーザー・プロセスが空きバッファを常に見つけることができるように、 DBWn はバッファ・キャッシュを管理します。

DBWn プロセスは、最低使用頻度（ LRU ）のバッファをディスクに書き込みます。最低使用頻度の使用済みバッファをディスクに書き込むことにより、 DBWn は、使用頻度の高いバッファをメモリー内に保ちながら、空きバッファを見つけ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、すぐに使わ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。

DBWn プロセスの個数は、初期化パラメータ DB_WRITER_PROCESSES で指定します。システムで複数の DBWn プロセスを使っている場合は、 DB_BLOCK_LRU_LATCHES パラメータの値を調整して、各DBWn プロセスが同じ個数のラッチ（LRUバッファ・リスト）を持つようにする必要があります。

> **追加情報**：DB_WRITER_PROCESSES と DB_BLOCK_LRU_LATCHES の設定に関するアドバイスは、『 Oracle8 Server チューニング』を参照してください。

DBWn 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走査してもクリーンな再使用可能バッファを見つけられない場合、 DBWn に書込み信号が送られる。 DBWn は、 1 回の「マルチ・ブロック書込み」によって、使用済みバッファをディスクに書き込みます。（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。）
- 「チェックポイント」が発生する場合、ログ・ライター・プロセス（ LGWR ）は DBWn に信号を送り、 DBWn はチェックポイントを完了するために書き込む必要のあるバッファをすべて書き込む。 LGWR が 1 つのオンライン REDO ログ・ファイルへの書込みを停

止して別のオンライン REDO ログ・ファイルへの書込みを開始すると、チェックポイントは初期化パラメータ LOG_CHECKPOINT_INTERVAL と LOG_CHECKPOINT_TIMEOUT で指定された間隔で各ログ・スイッチごとに発生します。

- DBWn は、定期的にバッファを書き込んで、 REDO スレッド（ログ）内のクラッシュまたはインスタンス回復を開始する必要のある位置（「増分チェックポイント」）まで進める。このログ位置は、バッファ・キャッシュ内の最も古い使用済みバッファによって決まります。増分チェックポイント機能は、キャッシュ内の使用済みバッファの個数が初期化パラメータ DB_BLOCK_MAX_DIRTY_TARGET で指定されたしきい値よりも大きくなったときに発生します。詳細は、 28-4 ページの「増分チェックポイント機能」を参照してください。

どの場合でも、効率を向上させるために DBWn はバッチ（マルチ・ブロック）書込みを実行します。

プラットフォームによっては、 DBW0 プロセスが複数の I/O サーバー・プロセスを持つことができます。これらのプロセスの 1 つがあるディスクへの書込みによりブロックしている場合に、他のプロセスは別のディスクへの書込みを続けることができます。複数の DBWn プロセスを持つシステム上では、これらのプロセスを使うことはできません。I/Oサーバー・プロセスの個数は、初期化パラメータ DBWR_IO_SLAVES で制御します。

> **追加情報**： 使用しているプラットフォームの複数 I/O サーバー・プロセスの詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。
>
> 単一の DBW0 プロセスまたは複数の DBWn プロセスのパフォーマンスの監視と調整の方法は、『Oracle8 Server チューニング』を参照してください。

ログ・ライター・プロセス (LGWR)

「ログ・ライター・プロセス（ LGWR ）」は、 REDO ログ・バッファ管理、つまりディスク上の REDO ログ・ファイルへの REDO ログ・バッファの書込みを行います（ 6-5 ページの「 REDO ログ・バッファ」を参照)。 LGWR は、最後の書込み以後にバッファにコピーされた REDO エントリすべてを、 REDO ログ・ファイルに書き込みます。

REDO ログ・バッファは循環バッファです。 LGWR が REDO ログ・バッファから REDO ログ・ファイルに REDO エントリを書き込んだ後、サーバー・プロセスはディスクに書き込まれた REDO ログ・バッファのエントリ上に新しいエントリをコピーできます。 REDO ログへのアクセスが頻繁なときにも、新しいエントリを書き込めるよう常にバッファ内の領域を空けておくため、 LGWR が行う書き込みは通常は非常に高速になります。

LGWR は、バッファの 1 つの連続した部分をディスクに書き込みます。 LGWR は、次のものを書き込みます。

- ユーザー・プロセスがトランザクションをコミットしたときのコミット・レコード
- REDO ログ・バッファ
  - 3 秒ごとに書き込む
  - 3 分の 1 が満杯になったときに書き込む

– DBWn 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、必要に応じて書き込む

注意 : DBWn 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なければなりません（「事前書込みプロトコル」）。 DBWn は、いくつかの REDO レコードが書き込まれていないことを発見した場合に、 REDO レコードをディスクに書き込むよう 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えない場合、LGWR はそのグループ内の他のファイルへ書き込みを続け、このエラーのログを LGWR トレース・ファイルやシステム ALERT ファイルに記録します（7-14ページの「トレース・ファイルと ALERT ファイル」を参照）。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えない場合、 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」メカニズムといいます。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、 Oracle Parallel Server 構成および分散データベースで回復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。

追加情報: SCN およびその使用方法の詳細は、『Oracle8 Parallel Server 概要および管理』と『 Oracle8 Server 管理者ガイド』を参照してください。

アクティビティが高いときには、 LGWR は「グループ・コミット」を使ってオンライン REDO ログ・ファイルに書き込むこともあります。たとえば、あるユーザー・トランザクションをコミットするときのことを考えてみます。 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まなければなりません。このときに、他のユーザーはCOMMIT 文を発行します。しかし、 LGWR は前の書込み操作を完了するまで、他のユーザーのオンライン REDO ログ・ファイルに書き込んでトランザクションを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ので、トランザクションのエントリを個々に処理するよりも、必要となる I/O を低減できます。そのため、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。

プラットフォームによっては、 LGWR プロセスが複数の I/O サーバー・プロセスを持つことができます。これらのプロセスの 1 つがある DBWR がディスクへの書込みによりブロックしている場合に、他のプロセスは別のディスクへの書込みを続けることができます。 I/O サーバー・プロセスの個数は、初期化パラメータ LGWR_IO_SLAVES で制御します。

> **追加情報**： 使用しているプラットフォームの複数 I/O サーバー・プロセスの詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。
>
> LGWR のパフォーマンスの監視と調整の詳細は、 Oracle8 Server チューニングを参照してください。

チェックポイント・プロセス（ CKPT ）

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して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。そのような処理は、常に DBWn が実行します。

Oracle Enterprise Manager の System_Statistics モニターによって表示される「DBWR チェックポイント」の統計には、完了したチェックポイント要求の数が示されます。

> **追加情報**： チェックポイント間隔を変更した場合の影響の詳細は、『 Oracle8 Server 管理者ガイド』を参照してください。
>
> Oracle Parallel ServerのCKPTの詳細は、『Oracle8 Parallel Server概要および管理』を参照してください。

システム・モニター（ SMON ）

「システム・モニター・プロセス（SMON）」は、インスタンスの起動時にインスタンス回復を実行します。 SMON には、使われなくなった一時セグメントをクリーン・アップする機能と連続した空きエクステントを 1 つに合せて、空き領域のより大きなブロックを使えるようにする機能もあります。 SMON は、処理が必要かどうかをチェックするために定期的に「起動」します。他のプロセスで SMON を起動する必要が検出された場合は、そのプロセスから SMON をコールできます。

Oracle Parallel Server 環境では、 SMON 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回復を実行します。

> **追加情報**: Oracle Parallel Server における SMON の詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

プロセス・モニター（ PMON ）

ユーザー・プロセスが失敗すると、「プロセス・モニター（ PMON ）」がプロセスを回復します。 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・プロセスが使っていたリソースを解放したりします。たとえば、アクティブ・トランザクション表の状態をリセットしてロックを解除し、アクティブ・プロセスのリストからプロセス ID を削除します。

PMON は、ディスパッチャ・プロセスとサーバー・プロセスの状態も定期的にチェックし、消滅したサーバー・プロセスがあれば再起動します（ただし、 Oracle が意図的に終了させたプロセスは除きます）。

SMONと同じように、PMONは、処理が必要かどうかをチェックするために定期的に「起動」し、別のプロセスで起動する必要が検出された場合にコールできます。

リカバラ・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われるバックグラウンド・プロセスであり、分散トランザクションに関連する障害を自動的に解決します。

ノードの RECO プロセスは、インダウト分散トランザクションにかかわる他のデータベースに自動的に接続されます。 RECO プロセスは、関係するデータベース・サーバー間の接続を再確立するときに、すべてのインダウト・トランザクションを自動的に解決し、解決されるインダウト・トランザクションに対応する行を各データベースの保留中のトランザクション表からすべて削除します。

RECO プロセスがリモート・サーバーとの接続に失敗した場合、 RECO は決められた時間間隔で自動的に再接続しようとします。ただし、 RECO が次に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）。

> **追加情報**: 分散トランザクションの回復の詳細は、『 Oracle8 Server 分散システム』を参照してください。

RECO プロセスは、インスタンスが分散トランザクションを許可している場合と DISTRIBUTED_TRANSACTIONS パラメータがゼロより大きい場合にだけ存在します。この初期化パラメータをゼロに設定すると、インスタンスの起動時に RECO は作られません。

アーカイバ・プロセス（ARCH）

「アーカイバ・プロセス（ARCH）」は、オンラインREDOログ・ファイルが満杯になると、指定された記憶デバイスにそれをコピーします。 ARCH は、 ARCHIVELOG モードで REDO ログを使っており、かつ自動アーカイブを使っているときにだけ存在します。

> **追加情報**: オンライン REDO ログのアーカイブの詳細は、 28-7 ページの「REDO ログ」と Oracle8 Server バックアップおよびリカバリを参照してください。
>
> ARCH プロセスの使用方法の詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

ロック・プロセス（LCKn）

Parallel Server オプションを使っている場合、最大 10 個のロック・プロセス（LCK0 、 . . . 、LCK9）によって内部インスタンスがロックされます。ほとんどのOracle Parallel Serverシステムでは、 1 つの LCK プロセス（LCK0）で十分です。

> **追加情報**: このバックグラウンド・プロセスの詳細は、『Oracle8 Parallel Server 概要および管理』を参照してください。

ジョブ・キュー・プロセス（SNPn）

分散データベース構成では、最大36個の「ジョブ・キュー・プロセス」（SNP0、...、SNP9、SNPA 、 ... 、 SNPZ）をが自動的に表スナップショットをリフレッシュできます。これらのプロセスは定期的に起動され、自動リフレッシュがスケジュールされているスナップショットをリフレッシュします。複数のジョブ・キュー・プロセスを使うと、プロセス間でスナップショットのリフレッシュ・タスクが共有されます。

他の Oracle バックグラウンド・プロセスとは異なり、 SNPn プロセスの障害は、インスタンス障害の原因にはなりません。 SNPn プロセスに障害が発生すると、 Oracle はそれを再起動します。

これらのプロセスは、 DBMS_JOB パッケージによって作成されるジョブ要求も実行し、キューイングされたメッセージを他のデータベースのキューに波及させます（16-2 ページの「Oracle Advanced Queuing」を参照してください）。

> **追加情報**: このバックグラウンド・プロセスとジョブ・キューの詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

キュー・モニター・プロセス（QMNn）

「キュー・モニター・プロセス」は、メッセージ・キューを監視するOracle Advanced Queuing（Oracle AQ）のオプションのバックグラウンド・プロセスです。最大 10 個のキュー・モニター・プロセスを構成できます。これらのプロセスは、SNPnプロセスと同様に、プロセスの障害がインスタンス障害の原因とならない点が他の Oracle バックグラウンド・プロセスと異なります。

メッセージ・キューとキュー・モニター・プロセスの詳細は、 16-2 ページの「Oracle Advanced Queuing」を参照してください。

ディスパッチャ・プロセス（Dnnn）

「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、マルチスレッド構成をサポートします。（7-19 ページの「マルチスレッド・サーバー」を参照してください。）マルチスレッド・サーバーでは、同じユーザー数に必要な共有サーバー・プロセスの数がより少なくて済みます。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、マルチスレッド・サーバーによってより多くのユーザーをサポートできます。

1 つのデータベース・インスタンスに対して複数のディスパッチャ・プロセスを作ることができます。 Oracle で使うネットワーク・プロトコルごとに、少なくとも 1 つのディスパッチャを作る必要があります。データベース管理者は、プロセスあたりの接続数についてのオペレーティング・システムの制限に応じて、ディスパッチャ・プロセスを適切な数だけ起動しなければなりません。また、インスタンスの実行中にディスパッチャ・プロセスを追加および削除できます。

注意： ディスパッチャに接続するそれぞれのユーザー・プロセスは、両方のプロセスが同一のマシン上で実行されている場合であっても、 Net8 または SQL*Net バージョン 2 を介して接続しなければなりません。

マルチスレッド・サーバー構成では、ネットワーク・リスナー・プロセスがクライアント・アプリケーションからの接続要求を待機し、それぞれのクライアント・アプリケーションをディスパッチャ・プロセスに経路指定します。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。

追加情報： ネットワーク・リスナーの詳細は、 7-19 ページの「マルチスレッド・サーバー」と『 Oracle Net8 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（Snnn）

各「共有サーバー・プロセス」は、マルチスレッド・サーバー構成では複数のクライアント要求を処理します。詳細は、7-23ページの「共有サーバー・プロセス」を参照してください。

トレース・ファイルと ALERT ファイル

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

追加情報： エラー・メッセージの詳細は、『 Oracle8 Server エラー・メッセージ』を参照してください。

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。これについての唯一の例外は、ジョブ待ち行列プロセス（ SNPn ）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調整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、該当する場合にいつもトレース・ファイルにこの情報を書き込みます。しかし、インスタンスまたはセッションの初期化パラメータ SQL_TRACE が TRUE に設定されている場合に限り、サーバー・プロセスはトレース・ファイルに調整情報を書き込みます。（内部エラーに関する情報は、常にトレース・ファイルに書き込まれます。）

それぞれのセッションは、SQL_TRACEパラメータを指定したSQLコマンドALTER SESSIONを使うことにより、対応付けられたサーバー・プロセスのためにトレースのロギングを使用可能または使用禁止にできます。たとえば、次の文により、セッションのトレース・ファイルに情報が書き込まれるようになります。

```
ALTER SESSION SET SQL_TRACE = TRUE;
```

データベースには、それぞれ「 ALERT ファイル」もあります。データベースの ALERT ファイルは、次のようなメッセージとエラーの履歴ログです。

- 発生したすべての内部エラー（ ORA-600 ）、ブロック障害エラー（ ORA-1578 ）、デッドロック・エラー（ ORA-60 ）
- 管理的な操作、たとえば CREATE/ALTER/DROP DATABASE/TABLESPACE/ROLLBACK SEGMENT の各 SQL 文と、 Oracle Enterprise Manager または Server Manager 文の STARTUP 、 SHUTDOWN 、 ARCHIVE LOG 、 RECOVER
- 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
- スナップショットの自動リフレッシュにおけるエラー

Oracle は、これらのイベントの記録を保管するのに、オペレータのコンソール上に情報を表示するかわりに ALERT ファイルを使います。（多くのシステムではコンソール上にもこの情報が表示されます。）管理操作が成功すると、タイムスタンプとともに「completed（完了）」というメッセージが ALERT ファイルに書き込まれます。

Oracle の構成におけるバリエーション

マルチプロセスの Oracle インスタンスでは、接続しているユーザー用のコードを、次の 3 つの方法のいずれかで構成できます。

シングル・タスクの Oracle	それぞれのユーザーについて、データベース・アプリケーションと Oracle Server コードの両方を結合して 1 つのユーザー・プロセスにする。

専用サーバー（2 タスクの Oracle）	それぞれのユーザーについて、データベース・アプリケーションを実行するプロセス（ユーザー・プロセス）と、Oracle Serverコードを実行するプロセス（専用サーバー・プロセス）を別々にする。
マルチスレッド・サーバー	データベース・アプリケーションを、Oracle Serverコードを実行するプロセスとは別のプロセス（ユーザー・プロセス）によって実行する。Oracle Serverコードを実行する各サーバー・プロセス（「共有サーバー・プロセス」）は、複数のユーザー・プロセスを処理できます。

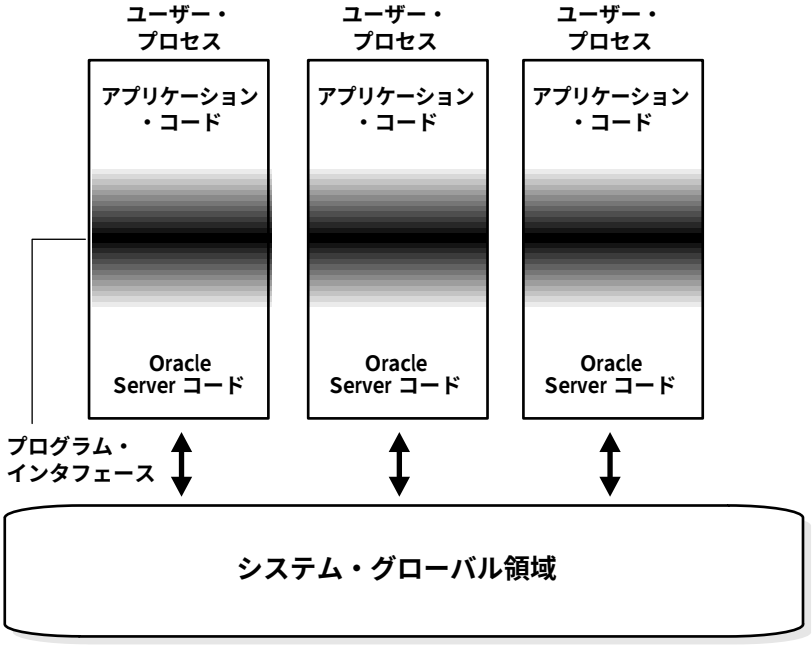
この後の項では、それぞれのバリエーションについて詳しく説明します。

追加情報：オペレーティング・システムによっては、構成を選択できるものもあります。このようなオプションの詳細は、オペレーティング・システム別の Oracle のマニュアルを参照してください。

シングル・タスク構成

図 7-4 に、シングル・タスクの Oracle 構成を示します。この構成で、データベース・アプリケーションとOracle Serverコードはすべて同じ「ユーザー・プロセス」というプロセス内で実行されます。

図 7-4 結合されたユーザー / サーバー・プロセスを使う Oracle



この Oracle の構成は、データベース・アプリケーション・コードと Oracle コードを 1 つのプロセス内で分離して維持できるオペレーティング・システム（VAX VMSオペレーティング・システムなど）でのみ実行できます。この分離は、データの整合性と機密のために不可欠です。 UNIX などのように、この分離を行うことができないオペレーティング・システムもあり、その場合はアプリケーション・コードとサーバー・コードを別々のプロセスで実行する必要があります。

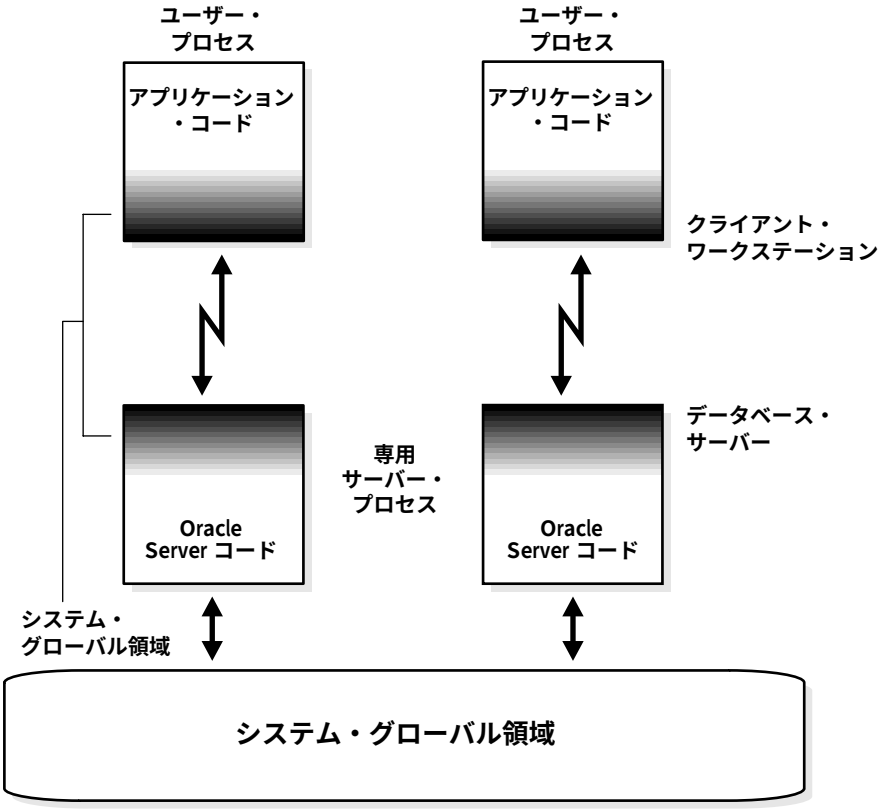
注意: Oracle Server コードの分離と保護、データベース・アプリケーションと Oracle ユーザー・プログラムとの間のデータの受渡しは、「プログラム・インタフェース」の役割です。7-26ページの「プログラム・インタフェース」を参照してください。

シングル・タスク構成を使っているプロセスは、いつでも 1 つの Oracle 接続だけを許可します。しかし、ユーザー作成のプログラムでは、ネットワーク（ Net8 ）インタフェースを使って Oracle に同時に接続しながら、このタイプの接続を維持できます。

専用サーバー（ 2 タスク）構成

図 7-5 は、専用サーバー・アーキテクチャを使い、 2 台のコンピュータ上で稼働している Oracle を示しています。この構成では、データベース・アプリケーションが 1 台のマシン上のユーザー・プロセスによって実行され、対応付けられたOracle Serverがもう1台のマシン上のサーバー・プロセスによって実行されます。

図 7-5 専用サーバー・プロセスを使う Oracle



ユーザー・プロセスとサーバー・プロセスは、お互いに分離した別のプロセスです。ユーザー・プロセスごとに作ら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・プロセスのためだけに活動するので、「専用サーバー・プロセス」（またはシャドウ・プロセス）といいます。

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であり、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る）。

図 7-5 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 Serverコードの両方が実行される場合にも使われますが、この 2 つのプログラムを 1 つのプロセスで実行しようとしても、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って通信します。

- 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ってジョブを実行する。
- ユーザー・プロセスと専用サーバー・プロセスが別々のコンピュータ上で実行される場合、プログラム・インタフェースはプログラム間の通信メカニズム（ネットワーク・ソフトウェアや Net8 など）を提供する。

  追加情報： このような通信リンクは、オペレーティング・システムとインストレーションに依存しています。詳細は、オペレーティング・システム別の Oracle のマニュアルと Net8 のマニュアルを参照してください。

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、マルチスレッド・サーバー・アーキテクチャを選ぶべきです。

専用サーバーの使用方法の詳細は、 7-24 ページの「専用サーバー・プロセスを使う Oracle の例」を参照してください。

マルチスレッド・サーバー

マルチスレッド・サーバー構成では、多くのユーザー・プロセスがごく少数のサーバー・プロセスを共有できます。ユーザー・プロセスはディスパッチャ・バックグラウンド・プロセスに接続し、ディスパッチャは次に利用できる共有サーバー・プロセスへクライアント要求を経路指定します。

マルチスレッド・サーバー構成の利点は、システムのオーバーヘッドが削減され、サポートできるユーザー数が増えることです。わずかな数の共有サーバー・プロセスで、多くの専用サーバー・プロセスが処理するのと同じ量の処理ができます。また、各ユーザーに必要なメモリーの容量も相対的に小さくなります。

マルチスレッド・サーバー・システムでは、さまざまなプロセスが多数必要です。

- ユーザー・プロセスをディスパッチャまたは専用サーバーに接続するネットワーク・リスナー・プロセス（リスナー・プロセスは、 Oracle ではなく Net8 の一部です）

- 1 つ以上のディスパッチャ・プロセス
- 1 つ以上の共有サーバー・プロセス

マルチスレッド・サーバーでは、 Net8 または SQL*Net バージョン 2 が必要です。

注意: 共有サーバーを使うには、ユーザー・プロセスは、Oracleインスタンスと同一のマシン上で実行されている場合でも、 Net8 または SQL*Net バージョン 2 を介して接続しなければなりません。

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・クライアントで使うネットワーク・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・プロセスを使えるかどうかを決定します。使える場合には、リスナー・プロセスは負荷が最も軽いディスパッチャ・プロセスのアドレスを戻し、ユーザー・プロセスはディスパッチャに直接接続します。

ユーザー・プロセスによっては、ディスパッチャと通信できない（たとえば、バージョン 2 より前の SQL*Net を使って接続した場合）こともあるので、ネットワーク・リスナー・プロセスは、このようなユーザーをディスパッチャに接続できません。この場合、つまりユーザー・プロセスが専用サーバーを要求する場合（ 7-24 ページの「マルチスレッド・サーバーでの操作の制限」を参照）、リスナーは専用サーバー・プロセスを作って適切な接続を確立します。

追加情報: ネットワーク・リスナーの詳細は、『 Oracle Net8 管理者ガイド』を参照してください。

ディスパッチャの要求と応答キュー

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。ユーザーがコールを行うと、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。

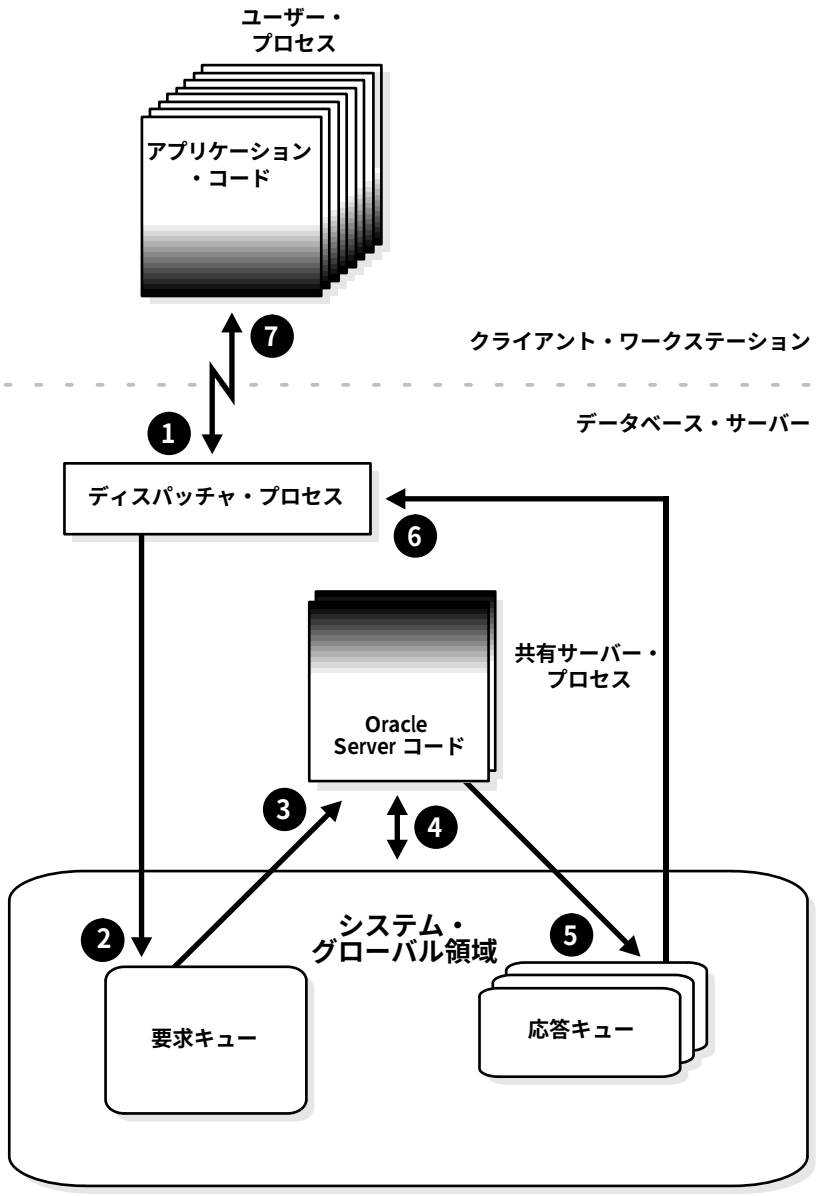
要求キューは SGA 内に存在し、インスタンスのディスパッチャ・プロセスすべてに共通です。共有サーバー・プロセスは、共通の要求キューをチェックして新しい要求がないかどうかを調べ、先入れ先出し方式に基づいて新しい要求をピックアップします。 1 つの共有サーバー・プロセスがキュー内の要求を 1 つピックアップし、その要求を完了するのに必要なデータベースに対するコールをすべて出します。

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、適切なユーザー・プロセスに完了した要求を戻します。

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。次に利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ので別の要求で使うことができます。 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・プロセスを使えます。

図 7-6 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、そしてディスパッチャがユーザー要求を共有サーバー・プロセスへ伝達する方法を示します。

図 7-6 マルチスレッド・サーバー構成と共有サーバー・プロセス



共有サーバー・プロセス

共有サーバー・プロセスと専用サーバー・プロセスには同じ機能性がありますが、共有サーバー・プロセスは特定のユーザー・プロセスと対応付けられていません。むしろ、共有サーバー・プロセスは、マルチスレッド・サーバー構成における任意の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、（すべての共有サーバー・プロセスがアクセスできなければならない）ユーザーに関係した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特有の変数だけが含まれます。 6-12 ページの「プログラム・グローバル領域（ PGA ）」には、さまざまなタイプのインスタンス構成における PGA の内容について詳しい説明があります。

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なければなりません。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ことにより、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。リソースの制限とプロファイルの詳細は、第 25 章「データベース・アクセスの制御」を参照してください。

Oracleは、要求キューの長さに基づいて、共有サーバー・プロセスの数を動的に調整します。作成できる共有サーバー・プロセスの数は、初期化パラメータ MTS_SERVERS と MTS_-MAX_SERVERS の値で指定した範囲です。

人工デッドロック

共有サーバー・プロセスの数が制限されている場合、「人工」デッドロックが生じる可能性があります。次のような状況では、人工デッドロックになることがあります。

1. あるユーザーが、 FOR UPDATE 句を指定した SELECT 文または LOCK TABLE 文を発行することによって、リソースに対する排他ロックを取得します。
2. 文の実行が完了すると、ロック要求を処理する共有サーバー・プロセスが解放されます。
3. 他のユーザーが、ロックされているリソースにアクセスしようとします。ロックされている必要なリソースが使えるようになるまで、各共有サーバー・プロセスはユーザー・プロセスにバインドされたままです。最終的には、ロックされたリソースを待機しているユーザー・プロセスにすべての共有サーバーがバインドされる可能性があります。
4. 最初のユーザーが新しい要求（ COMMIT 文または ROLLBACK 文など）を送って、以前に取得したロックを解放しようとしますが、共有サーバー・プロセスがすべて使用中であるために送ることができません。

Oracle が人工デッドロックを検出すると、ロックされているリソース（人工デッドロックを引き起こしているリソース）を解放するという要求を元のユーザーが送るまで、必要に応じて新しい共有サーバー・プロセスが自動的に作られます。最大数の共有サーバー・プロセス（ MTS_MAX_SERVERS パラメータで指定）がすでに起動されている場合、データベース管理者はユーザーの接続を切断することによってデッドロックを手動で解決しなければなりません。これにより、共有サーバー・プロセスが解放されるので、人工デッドロックが解決されます。

システム上で人工デッドロックが頻繁に発生する場合には、 MTS_MAX_SERVERS の値を増やしてください。

マルチスレッド・サーバーでの操作の制限

インスタンスの停止、インスタンスの起動、そして媒体回復を含む、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って接続しているときに実行されます。マルチスレッド・サーバーにより構成されたシステムで管理者権限を使って接続する場合は、ディスパッチャ・プロセスではなく専用サーバー・プロセスを使うことを接続文字列で明言しなければなりません（ SRVR=DEDICATED ）。

> **追加情報**： 適切な接続文字列の構文の詳細は、オペレーティング・システム別の Oracle のマニュアルか、『 Oracle Net8 管理者ガイド』を参照してください。

マルチスレッド・サーバー構成の使用方法の詳細は、 7-25 ページの「マルチスレッド・サーバーを使う Oracle の例」を参照してください。

Oracle の動作例

Oracle データベース・システムのメモリー構造およびプロセス、さまざまな構成についてこれまで説明してきましたが、それらのすべて部分が協調して作動する様子について理解しておくとよいでしょう。この後の項では、 2 タスクおよびマルチスレッド・サーバー Oracle 構成について具体的に説明し、比較します。

専用サーバー・プロセスを使う Oracle の例

次の例は、専用サーバー・アーキテクチャの簡単な具体例です。これらのステップは、 Oracle が実行する操作で最も基本的なレベルのものだけを示しています。

1. 現在、データベース・サーバー・マシンは、複数のバックグラウンド・プロセスを使って Oracle を実行しています。
2. クライアント・ワークステーション上のユーザー・プロセスは、 SQL*Plus などのデータベース・アプリケーションを実行します。クライアント・アプリケーションは、 Net8 ドライバを使って、サーバーへの接続を確立しようとします。

3. 現在、データベース・サーバーは、適切な Net8 ドライバを実行中です。データベース・サーバー上のネットワーク・リスナー・プロセスは、クライアント・データベース・アプリケーションからの接続要求を検出し、ユーザー・プロセスのためにデータベース・サーバー上に専用サーバー・プロセスを作ります。
4. ユーザーが SQL 文を 1 つ実行します。たとえば、表の中に行を挿入します。
5. 専用サーバー・プロセスはその文を受け取ります。この時点では、次の 2 つの方法で SQL 文の処理が継続されます。
   - 共有プールに同一の SQL 文のための共有 SQL 領域がある場合、サーバー・プロセスはクライアントの SQL 文を実行するために既存の共有 SQL 領域を使います。
   - 共有プールに同一の SQL 文のための共有 SQL 領域がない場合、その文のために共有プール内に新しい共有 SQL 領域が割り当てられます。

   どちらの場合も、プライベート SQL 領域はセッションの PGA 内に作られ、専用サーバー・プロセスは、要求されたデータへのユーザーのアクセス権限をチェックします。
6. サーバー・プロセスは、必要であれば、実際のデータ・ファイルからデータ・ブロックを検索します。または、すでにインスタンスの SGA にあるバッファ・キャッシュに格納されているデータ・ブロックを使います。
7. サーバー・プロセスは、共有 SQL 領域に格納されている SQL 文を実行します。データは最初に SGA 内で変更されます。 DBW0 プロセスによって最も効率的だと判断された時期に、データはディスクに永続的に書き込まれます。 LGWR プロセスは、ユーザーから次のコミット要求があった時点で、オンライン REDO ログ・ファイル内にトランザクションを記録します。
8. 要求が成功すると、サーバーはネットワークを介してその旨のメッセージを送ります。成功しなかった場合は、適切なエラー・メッセージを送信します。
9. この手順全体にわたって、他のバックグラウンド・プロセスが稼働しており、介入が必要となる条件がないかどうか監視しています。さらに Oracle は、他のトランザクションを管理し、同じデータを要求する別のトランザクション間で競合が起こらないようにしています。

マルチスレッド・サーバーを使う Oracle の例

次の例は、マルチスレッド・サーバー・アーキテクチャの簡単な具体例です。

1. 現在、データベース・サーバーは、マルチスレッド・サーバー構成を使って Oracle を実行しています。
2. クライアント・ワークステーション上のユーザー・プロセスは、 SQL*Forms などのデータベース・アプリケーションを実行します。クライアント・アプリケーションは、適切な Net8 ドライバを使って、データベース・サーバーへの接続を確立しようとします。
3. 現在、データベース・サーバー・マシンは、適切な Net8 ドライバを実行中です。データベース・サーバー上のネットワーク・リスナー・プロセスは、ユーザー・プロセスの接続要求を検出し、ユーザー・プロセスをどのように接続するかを決定します。ユーザー

が Net8 または SQL*Net バージョン 2 を使っている場合、リスナーは、利用可能なディスパッチャ・プロセスのアドレスを使って再接続するようにユーザー・プロセスに通知します。

注意：ユーザー・プロセスが SQL*Net バージョン 1 または 1.1 に接続している場合には、 SQL*Net のリスナーはユーザー・プロセスのために専用サーバー・プロセスを作ります。その後、前の例で説明したように作動します。（ユーザー・プロセスは、共有サーバー・プロセスを使うには Net8 または SQL*Net バージョン 2 に接続する必要があります。）

4. ユーザーが SQL 文を 1 つ発行します。たとえば、表の中の行を更新します。
5. ディスパッチャ・プロセスはユーザー・プロセスの要求を要求キューに入れます。要求キューは SGA 内にあり、すべてのディスパッチャ・プロセスによって共有されています。
6. 利用可能な共有サーバー・プロセスは、共通のディスパッチャ要求キューをチェックして、キューにある次のSQL文をピックアップします。その後、前の例のステップ5、6、7 で説明したとおりに SQL 文を処理します。（ステップ 5 では、セッションのプライベート SQL 領域の一部が SGA 内に作られます。）
7. 共有サーバー・プロセスは、 SQL 文の処理を完了すると、要求を送ったディスパッチャ・プロセスの応答キューにその結果を入れます。
8. ディスパッチャ・プロセスは、その応答キューをチェックし、要求を出したユーザー・プロセスに完了済みの要求を送り返します。

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ぐ。
- 通信メカニズムとして機能し、情報要求の形式化、データの受渡し、エラーの検出と通知を行う。
- データを特に異機種コンピュータ間で変換し解釈する、または外部ユーザー・プログラム・データ型に変換し解釈する。

「 Oracle コード」はサーバーとして機能し、データ・ブロックから行をフェッチするなど、「アプリケーション」（クライアント）のためにデータベース・タスクを実行します。 Oracle コードは、 Oracle ソフトウェアとオペレーティング・システム固有のソフトウェアの両方によって提供されるいくつかの部分で構成されます。

プログラム・インタフェースの構造

プログラム・インタフェースは、次の要素から構成されます。

- Oracle コール・インタフェース（ OCI ）または Oracle ランタイム・ライブラリ（ 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）
- 各種「 Net8 ドライバ」（プロトコル固有の通信ソフトウェア）
- オペレーティング・システムの通信ソフトウェア
- サーバー側または Oracle 側のプログラム・インタフェース（ OPI とも呼ばれる）

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。

Net8 は、プログラム・インタフェースの一部であり、これによってクライアント・アプリケーション・プログラムとOracle Serverが通信ネットワーク内の別々のコンピュータ上に常駐できるようになります。

プログラム・インタフェース・ドライバ

「ドライバ」とは、通常はネットワークを介してデータを転送するソフトウェアの一部です。ドライバは、接続、切断、エラーの通知、エラーのテストなどの操作を実行します。ドライバは通信プロトコルに固有のものです。デフォルト・ドライバが常に存在します。

複数のドライバ（非同期または DECnet ドライバなど）をインストールし、その 1 つをデフォルト・ドライバとして選択しますが、各ユーザーは接続の時点で必要なドライバを指定することにより、他のドライバを使えます。プロセスが異なれば、異なるドライバを使用できます。 1 つのプロセスにおいても、異なる Net8 ドライバを使うことにより、 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。

ドライバの選択とインストール、そしてインストール後の新規ドライバの追加の詳細は、使っているシステムのインストレーション・ガイドおよび構成ガイド、 Net8 のマニュアルに記載されています。 Net8 のマニュアルでは、 Oracle にアクセスしながら、実行時にドライバを選択する方法について説明しています。

追加情報 : Net8 の詳細は、『 Net8 管理者ガイド』を参照してください。

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。

追加情報: 通信ソフトウェアはオラクル社から提供されることもありますが、通常、ハードウェア・ベンダーまたはサード・パーティーのソフトウェア会社から別個に購入します。使用している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・システム別の Oracle のマニュアルを参照してください。

第 IV 部

オブジェクト・リレーショナル DBMS

第 IV 部では、データベースを管理するための Oracle リレーショナル・モデルと、そのモデルに対するオブジェクト拡張機能について説明します。

第 IV 部には、次の章が含まれています。

- 第 8 章「スキーマ・オブジェクト」
- 第 9 章「パーティション表とパーティション索引」
- 第 10 章「ビルトイン・データ型」
- 第 11 章「ユーザー定義データ型（オブジェクト・オプション）」
- 第 12 章「ユーザー定義データ型の使用方法」
- 第 13 章「オブジェクト・ビュー」

8

スキーマ・オブジェクト

My object all sublime
I shall achieve in time —
To let thepunishment fit the crime.

Sir William Schwenck Gilbert: *The Mikado*

この章では、ユーザー・スキーマに含まれるさまざまな種類のデータベース・オブジェクトについて説明します。この章の内容は次のとおりです。

- スキーマ・オブジェクトの概要
- 表
- ビュー
- 順序生成機能
- シノニム
- 索引
- 索引構成表
- クラスタ
- ハッシュ・クラスタ

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

スキーマ・オブジェクトの概要

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

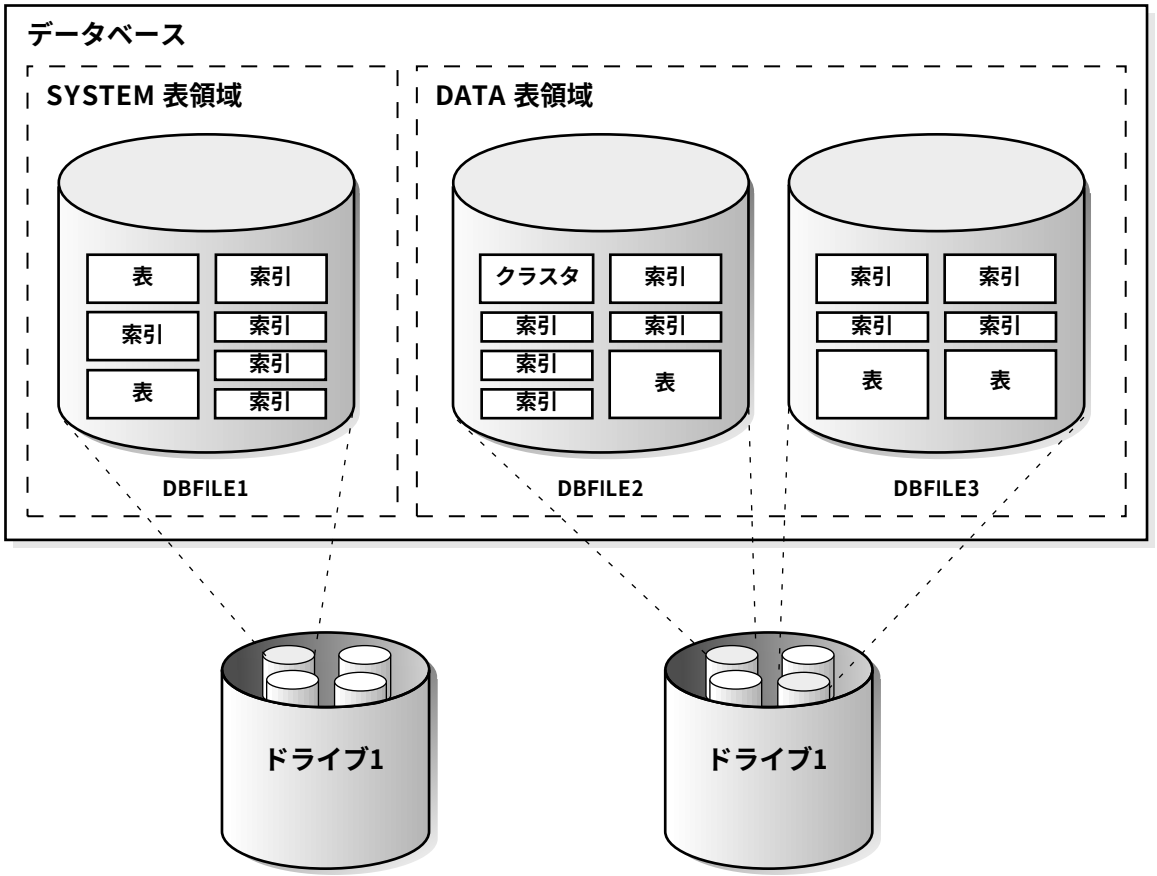
スキーマ・オブジェクトは、論理的なデータ記憶構造です。各スキーマ・オブジェクトは、その情報を格納するディスク上の物理ファイルと 1 対 1 では対応しませんが、 データベースの表領域内に論理的に格納されます。各オブジェクトのデータは、物理的には、表領域の 1 つ以上のデータ・ファイルに格納されます。表、索引、クラスタなど、いくつかのオブジェクトについては、表領域のデータ・ファイル内のオブジェクトに対して割り当てるディスク領域の容量を指定できます。

スキーマと表領域の間にはどんな関連もありません。異なるスキーマのオブジェクトが 1 つの表領域に含まれることもあれば、 1 つのスキーマのオブジェクトが異なる表領域に含まれることもあります。図 8-1 に、オブジェクトおよび表領域、データ・ファイルの関連を示します。

追加情報: この章では、表およびビュー、順序、シノニム、索引、クラスタについて説明します。その他の種類のスキーマ・オブジェクトについては、このマニュアルの別の章または他のマニュアルで説明しています。次のとおりです。

- データベース・リンクについては、第 30 章「分散データベース」に説明があります。
- オブジェクト・ビューについては、第 13 章「オブジェクト・ビュー」と『 Oracle8 Server アプリケーション開発者ガイド』に説明があります。
- プロシージャおよびファンクション、パッケージについては、第 17 章「プロシージャとパッケージ」に説明があります。
- トリガーについては、第 18 章「データベース・トリガー」に説明があります。
- スナップショットについては、『Oracle8 Serverレプリケーション』に説明があります。

図 8-1 スキーマ・オブジェクトおよび表領域、データ・ファイル



表

「表」は、Oracleデータベースにおけるデータ記憶の基本単位です。データは「行」と「列」に格納されます。すべての表は、「表名」（たとえば、 EMP ）と列の集合で定義されます。各列には、「列名」（EMPNO、ENAME、JOBなど）および「データ型」（VARCHAR2、DATE、NUMBER など）、「幅」（幅は DATE のようにデータ型によって事前定義されていることもある）や「精度」と「位取り」（ NUMBER データ型の列の場合だけ）を指定します。行は、 1 つのレコードに対応する列情報の集まりです。

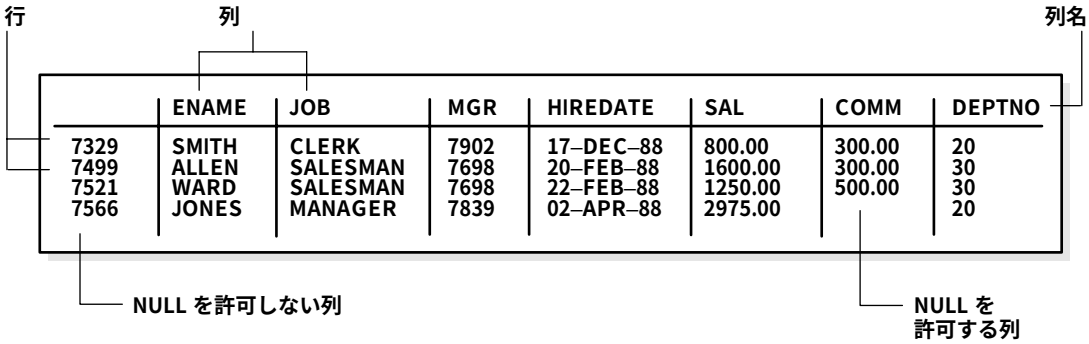
Oracle データ型の詳細は、第 10 章「ビルトイン・データ型」を参照してください。

表の各列に対してルールを任意に指定できます。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。整合性制約の詳細は、第 24 章「データの整合性」を参照してください。

表を作成すると、 SQL 文を使って、データ行をその表に挿入できます。挿入した表データに対して、 SQL を使って問合せ、削除、更新を実行できます。

図 8-2 に、 EMP という名前のサンプル表を示します。

図 8-2 EMP **表**



表データはどのように格納されるか

クラスタ化しない表を作成するとき、表の将来のデータを保持するために、データ・セグメントが表領域内に自動的に割り当てられます。表のデータ・セグメントに対する領域の割当て、およびこの確保した領域の使用は、次の方法で制御できます。

- データ・セグメントに対して記憶領域パラメータを設定することにより、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 パラメータを設定することにより、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。

クラスタ化した表のデータは、対応するクラスタに対して作成されたデータ・セグメント内に格納されます。クラスタ化した表を作成または変更するときに、記憶領域パラメータは指定できません。つまり、クラスタに対して設定された記憶領域パラメータが、クラスタ内のすべての表の記憶を制御します。

クラスタ化し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。25-7ページの「ユーザー表領域の設定と割当て制限」を参照してください。

行の形式とサイズ

Oracle では、データベース表の各行が 1 つ以上の行断片として格納されます。 1 つの行全体を単一のデータ・ブロックに挿入できる場合、その行は 1 つの行断片として格納されます。しかし、 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、複数の行断片を使って行が格納されます。 1 つのデータ・ブロックには、通常、行当たり 1 つの行断片だけが格納されます。1つの行を複数の行断片に格納する必要がある場合は、複数のブロックにわたって行が「連鎖」されます。連鎖行の各断片は、それぞれの断片のROWIDを使って連鎖されます。2-9 ページの「行連鎖と移行」を参照してください。

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、その結果として複数のデータ・ブロックにまたがることもあります。

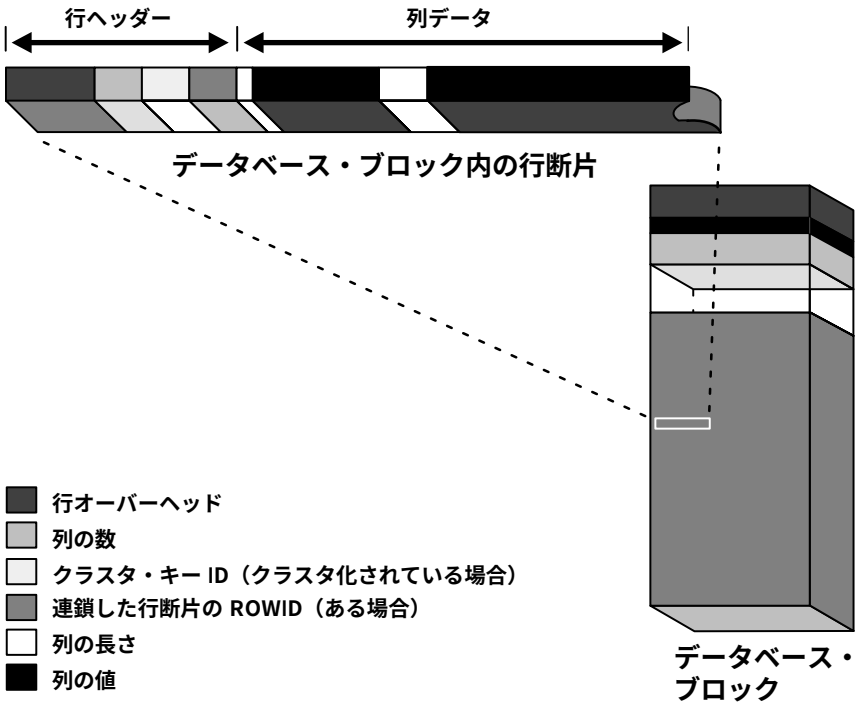
図 8-3 に、行断片の形式を示します。

「行ヘッダー」は、データの前に位置し、次のような情報を収録します。

- 行断片
- 連鎖（連鎖された行断片の場合）
- 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）

1つのブロックに完全に収まるクラスタ化されていない行は、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、データ型に依存します。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。

図 8-3 行断片の形式



領域を節約するために、 NULL を含む列は、列の長さ（ゼロ）だけを格納します。 NULL 列のデータは格納されません。また、末尾にある NULL 列については、行ヘッダーによって新しい行の始まりがわかるため、列の長さが格納されません（たとえば、表の最後の 3 列が NULL の場合、これらの列についての情報は何も格納されません。）

注意: それぞれの行は、データ・ブロック・ヘッダーの行ディレクトリで2バイトを使います。

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。 8-28 ページの「クラスタ」を参照してください。

列の順序

列の順序は、表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、最後に格納される列になります。

一般的には、行に対して必要となる領域を少なくするために、 NULL を頻繁に格納するような列を最後の列にします。ただし、作成しようとしている表に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。

行断片の ROWID

「ROWID」は、それぞれの行断片を位置またはアドレスで識別します。 ROWIDが行断片に割り当てられると、対応する行の削除や、 Export/Import ユーティリティによるエクスポートまたはインポートが実行されるまで、行断片はそのROWIDを保持します。ある行のクラスタ・キー値が変化した場合、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なので、 SELECT および UPDATE 、 DELETE などの SQL 文で ROWID を参照すると便利です。詳細は、 10-11 ページの「 ROWID データ型」を参照してください。

NULL

「 NULL 」は、ある行のある列に値が入っていないことを意味します。 NULL は、欠落しているデータ、不明なデータ、または適用できないデータを示します。他の値（たとえばゼロ）を暗示するためにはNULL を使わないでください。 NOT NULL またはPRIMARY KEY 制約が列に対して定義されていない場合に限り、列に NULL 値を入力できます。これらの制約が列に対して定義されているなら、その列に値を持たない行を挿入することはできません。

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 バイトだけが必要となります。行内の末尾にある NULL には、格納領域が必要ありません。新しい行の行ヘッダーが、前の行の残りの列が NULL であることを知らせるためです。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します。

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、不明となります。SQL文の中でNULLを識別するには、IS NULL述語を使います。NULLをNULL以外の値に変換するには、 SQL 関数の NVL を使います。

> **追加情報**: IS NULLおよびNVL関数を使った比較の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

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（ 8-24 ページの「ビットマップ索引と NULL 」を参照してください）。

列のデフォルト値

新しい行が挿入され、列に対する値が省略されたときに、デフォルト値が自動的に入力されるように、表の列にデフォルト値を割り当てることができます。列のデフォルト値は、実際に INSERT 文が値を指定しているかのように機能します。

正当なデフォルト値には、リテラルや、列または LEVEL 、 ROWNUM 、 PRIOR を参照しない式が含まれます。デフォルト値は、 SQL 関数 SYSDATE および USER 、 USERENV 、 UID を含むことができます。デフォルトのリテラルまたは式のデータ型は、その列のデータ型に一致しているか、あるいは変換可能でなければなりません。

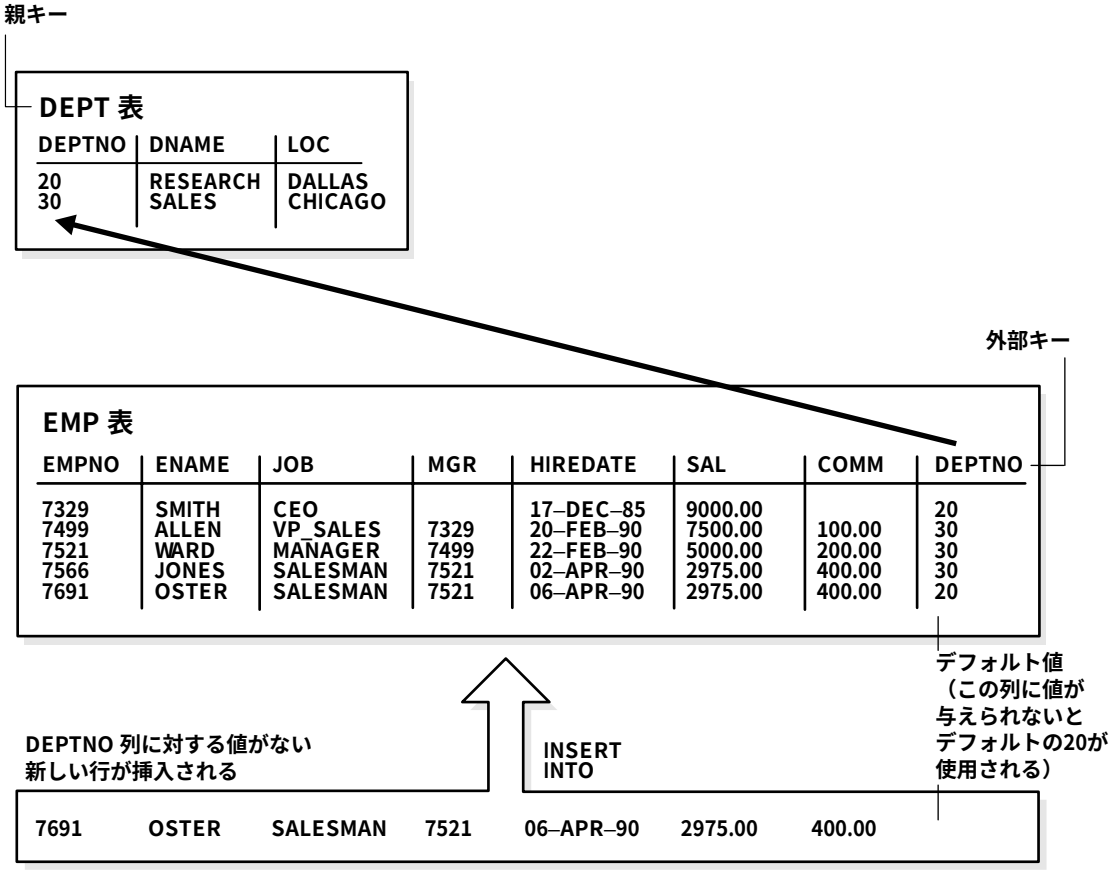
列に対してデフォルト値が明示的に定義されていない場合、その列のデフォルトは NULL に設定されます。

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。図 8-4 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。従業員の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 」が割り当てられます。デフォルト値が割り当てられた後で、 DEPTNO 列に定義されている FOREIGN KEY 整合性制約がチェックされます。

整合性制約の詳細は、第 24 章「データの整合性」を参照してください。

図 8-4 DEFAULT 列値



ネストした表

Oracle オブジェクト・リレーショナル・データベースでは、データ型として別の表を指定した表を作成できます。つまり、表の列値として別の表を「ネスト」できます。Oracle Serverは、親表の行とは「調和しない」ネストした表のデータを、ネストした表列に対応付けた「格納表」を使って格納します。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。

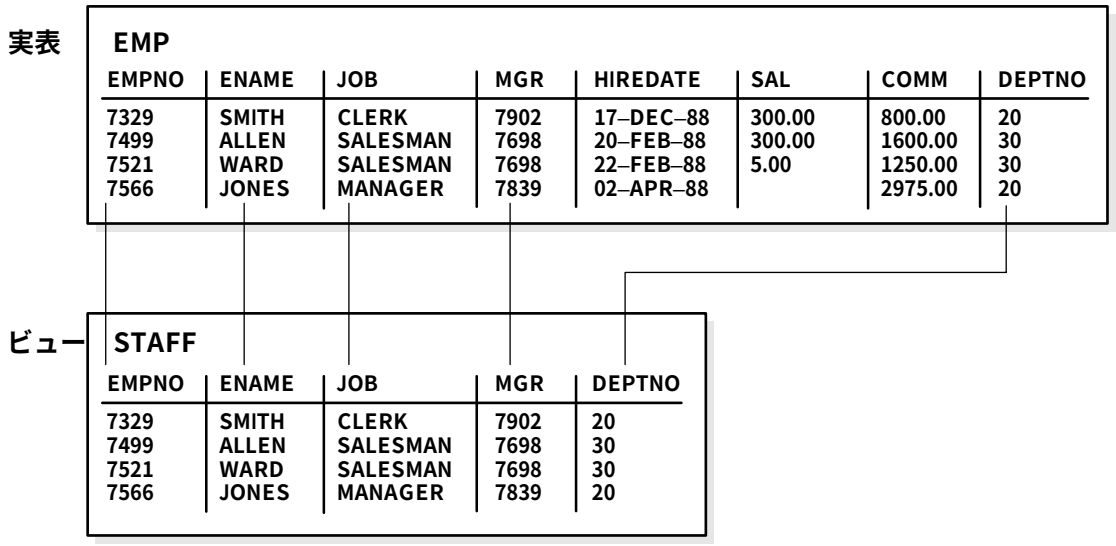
追加情報: 『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

ビュー

ビューとは、1 つ以上の表（または他のビュー）に含まれているデータを調整して表現したものです。ビューは問合せの結果を取り込んで、それを表として取り扱います。つまり、ビューは「格納された問合せ」または「仮想表」と考えることができます。多くの場合、ビューは表と同じように使えます。

たとえば、EMP 表には、いくつかの列と多数の情報行があります。それらのうちユーザーには 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、ユーザーがアクセスできる EMP 表のビューを作成できます。図 8-5 に、実表 EMP から導出された STAFF というビューの例を示します。ビューが、実表の 5 つの列だけを表示していることに注目してください。

図 8-5 **ビューの例**



ビューは表から導出されるので、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ほか、いくつかの制限付きでビューのデータの更新、挿入、削除を行うことができます。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作用し、実表の整合性制約とトリガーの対象になります。

追加情報：『Oracle8 Server SQL リファレンス』を参照してください。

注意：ビューでは整合性制約とトリガーを明示的に定義できませんが、ビューから参照される実表に対しては定義できます。

ビューの記憶領域

表とは異なり、ビューには記憶領域は割り当てられず、ビューが実際にデータを持つことはありません。ビューは、そのビューで参照する表からデータの抽出または取出しを行う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（スナップショットを含む）を使えます。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（格納された問合せ）以外には記憶領域が必要ありません。

ビューはどのように使用されるか

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提示する手段を提供します。様々なユーザーに合わせてデータの表現形式を変化させることができるため、ビューは非常に強力な機能です。

ビューは次の目的でよく使われます。

- 事前に定義された行と列の集合だけにアクセスを限定することにより、表のセキュリティ・レベルを強化する。

  たとえば、図 8-5 は、 STAFF ビューが実表 EMP の列 SAL と COMM をどのように隠すかを示しています。

- データの複雑さを隠す。

  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。ビューは、この情報が複数の表から抽出されたものであるという事実を隠します。

- ユーザー側のコマンドを単純化する。

  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。

- 実表とは異なる見方でデータを提示する。

  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する。

  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、ビューを使っ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する。

  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。

  追加情報: GROUP BYとUNIONの詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

- 複雑な問合せを保存する。

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けば、そのビューに問合せを行うときだけ同じ計算が実行されます。

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。ビューが SQL 文で参照されると、 Oracle はそのビューを参照する文を、ビューを定義する問合せとマージしてから、マージした文を共有 SQL 領域で解析し、実行します。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 領域に同一の文が含まれていない場合に限って解析されます。したがって、ビューを使うと、共有 SQL に対応付けられているメモリーの使用量を減らせるという利点があります。

NLS パラメータ

文字列リテラルが含まれているビューや、 NLS パラメータを引数として持つ SQL 関数（ TO_CHAR 、 TO_DATE 、 TO_NUMBER など）を評価する場合、 Oracle はセッションの NLS パラメータからこれらのパラメータのデフォルト値を取り出します。ビューの作成者は、ビューの定義で NLS パラメータを明示的に指定することにより、これらのデフォルト値を上書きできます。

索引の使用方法

Oracle がビューに対する問合せで索引を使うかどうかは、元の問合せがビュー定義の問合せとマージされるときにどのように変換されるかによって決まります。たとえば、次のビューについて考えてみます。

```
CREATE VIEW emp_view AS
  SELECT empno, ename, sal, loc
    FROM emp, dept
    WHERE emp.deptno = dept.deptno AND dept.deptno = 10;
```

ここで、ユーザーは次の問合せを発行します。

```
SELECT ename
  FROM emp_view
  WHERE empno = 9876;
```

Oracle によって作成される最終的な問合せは次のようになります。

```
SELECT ename
  FROM emp, dept
  WHERE emp.deptno = dept.deptno AND
        dept.deptno = 10 AND
        emp.empno = 9876;
```

可能であればいつでも、 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とマージします（基礎となるビューの問合せもマージされます）。マージされた問合せは、その問合せがビューを参照しないで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うことができます。

場合によっては、ユーザーの発行した問合せとビュー定義をマージできないこともあります。そのような場合には、参照された列の索引すべてを使えるとは限りません。

依存性とビュー

ビューは、他のオブジェクト（表、スナップショット、または他のビュー）を参照する問合せによって定義されるため、参照されるオブジェクトに依存します。 Oracle は、ビューの依存性を自動的に処理します。たとえば、ビューの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合致するかどうかを判断します。データベースの依存性の詳細は、第 19 章「 Oracle の依存性の管理」を参照してください。

更新可能な結合ビュー

「結合ビュー」とは、ビューの定義に FROM 句で複数の表またはビューが指定され、 DISTINCT 句および AGGREGATION 句、 GROUP BY 句、 START WITH 句、 CONNECT BY 句、ROWNUM句、集合演算（UNION ALL、INTERSECTなど）のいずれをも使っていないビューを指します。

「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、DELETE の各操作が実行可能である結合ビューを指します。ビューのどの列が更新可能であるかは、データ・ディクショナリ・ビュー ALL_UPDATABLE_COLUMNS 、DBA_UPDATABLE_COLUMNS 、 USER_UPDATABLE_COLUMNS に示されています。表 8-1 は、更新可能な結合ビューの規則を示しています。

表 8-1 結合ビューに対する INSERT 、 UPDATE 、 DELETE の規則

規則	説明
一般規則	結合ビューに対する INSERT または UPDATE 、 DELETE 操作では、そのビューの基礎となる表を一度に 1 つだけ変更できる。
UPDATE 規則	結合ビュー内の更新可能なすべての列は、キー保存表の各列に対応していなければならない。ビューを定義するときにWITH CHECK OPTION句を指定した場合は、すべての結合列と繰返し表のすべての列が更新できなくなる。
DELETE 規則	結合に含まれているキー保存表が厳密に 1 つであれば、結合ビューの行を削除できる。ビューを定義するときにWITH CHECK OPTION句を指定し、キー保存表を繰り返し指定した場合は、ビューから行を削除できない。
INSERT 規則	INSERT 文では、明示的にも暗黙的にもキー保存表以外の列を参照してはならない。結合ビューを定義するときにWITH CHECK OPTION句を指定した場合、 INSERT 文は使用できない。

更新可能でないビューは、INSTEAD OFトリガーを使って変更できます。詳細は、18-11ページの「 INSTEAD OF トリガー」を参照してください。

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」を使うと、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、削除できます。また、オブジェクト、 REF 、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。

追加情報: 第13章「オブジェクト・ビュー」と『Oracle8 Serverアプリケーション開発者ガイド』を参照してください。

順序生成機能

順序生成機能は、連続的な数値を生成します。順序生成機能は、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ので、マルチユーザー環境において特に有効です。したがって、この順序生成機能によって「直列化」（ 2 つのトランザクションの文が同時に連続する番号を生成する必要のある状態）が低減されます。この直列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かなり短縮されます。

順序番号とは、データベース内で定義される最大38桁の Oracle 整数です。順序の定義には、順序の名前、昇順あるいは降順、数の増分値などの一般的な情報を指定します。順序定義で 1 つ重要なことは、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ということです。

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えるようになっています。

順序番号は、順序を参照する SQL 文によって使われます。文を発行して、新しい順序を生成したり、現行の順序番号を使ったりできます。いったんユーザーのセッションで文によって順序番号が生成されると、そのセッションにかぎり固有の順序番号が使用可能になります。つまり、順序を参照する各ユーザーは、固有の現行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、同じ順序生成機能を 1 つの表または複数の表に対して使うことができます。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われた個々の順序番号はスキップされます。必要であれば、アプリケーションは、これらの順序番号を受け取って再利用するように作ることもできます。

追加情報: 順序を使ったときのパフォーマンスの詳細は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

シノニム

シノニムとは、表、ビュー、スナップショット、順序、プロシージャ、ファンクション、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領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われます。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する。
- データベース・ユーザー側の SQL 文を単純にする。

パブリック・シノニムとプライベート・シノニムの両方を作成できます。「パブリック」シノニムは、PUBLICという名前の特別なユーザー・グループが所有しており、データベースのすべてのユーザーがアクセスできます。「プライベート・シノニム」は、特定のユーザーのスキーマに含まれており、そのユーザーが他のユーザーに対するシノニムの使用許可を制御します。

シノニムは、分散システム内の位置を含めて、基礎になるオブジェクトの所在を隠すため、分散データベース環境でもそれ以外の環境でもたいへん便利です。基礎になるオブジェクトの名前が変更または移動されても、シノニムを再定義する必要があるだけで、シノニムを基礎とするアプリケーションは修正しなくても作動を続けます。

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。実表の所在を隠したり、 SQL 文の複雑さを低減するために、データベース管理者はしばしばパブリック・シノニムを作成します。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 という表がある。
- SALES_DATA 表に対する SELECT 権限が PUBLIC に付与されている。

ここで、データベースのユーザーが、次のような SQL 文を使って SALES_DATA 表に問い合せます。

```
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 表に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。索引を明示的に作成して、表に対する SQL 文の実行速度を向上させることができます。このマニュアルでも、索引を使うと特定の情報をすばやく見つけることができます。同じように Oracle の索引を使うと表データに高速にアクセスできます。適切に使えば、索引はディスクI/Oを低減する基本的な手段になります。

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。これには、B* ツリー索引（現在では最も一般的）および 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、ビットマップ索引があります。

索引の有無によって、 SQL 文の表現を変更する必要はありません。索引は、データへの高速なアクセス・パスにすぎないからです。つまり、索引は実行速度にだけ影響を与えます。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。索引が削除されてもすべてのアプリケーションは作動し続けますが、以前索引が付いていたデータにアクセスする場合、速度が遅くなる可能性があります。索引は独立した構造体であるため、記憶領域を必要とします。

作成された索引は、 Oracle によって自動的にメンテナンスされ、使われます。ユーザーは何もしないでも、新しい行の追加、行の更新、行の削除など、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ます。

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。しかし、表に対して数多くの索引が存在すると、更新、削除、挿入のパフォーマンスは低下します。これは、表に関連する索引も更新しなければならないためです。

オプティマイザは、既存の索引を使って別の索引を作成できます。この結果、索引作成がずっと高速になります。

一意索引と一意でない索引

索引には、一意索引と一意でない索引の 2 つがあります。一意索引によって、索引を定義された列に重複値を持つ行が複数存在しないことが保証されます。一意でない索引は、このような制限を列値に課しません。

表に対して一意索引を明示的に定義することはしないようにお薦めします。一意性は厳密に論理的な概念であり、表の定義に関連付けるべきだからです。そのかわりに、希望の列に対して UNIQUE 整合性制約を定義します。 Oracle は、一意キーに対して自動的に一意索引を定義することにより、 UNIQUE 整合性制約を施行します。

複合索引

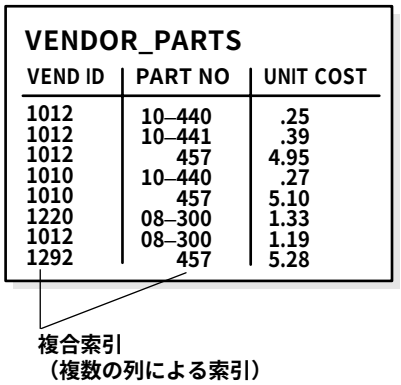
「複合索引」（連結索引とも呼ばれる）は、表の中の複数の列に対して作成される索引です。複合索引を構成する複数の列は、どんな順序で指定しても構いません。また、複合索引の列は表の中で隣接している必要もありません。

SELECT 文の WHERE 句で複合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、複合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。このため、定義の中で指定する列の順序が重要です。通常は、最も頻繁にアクセスされる列や選択される列を最初に定義します。

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

図 8-6 は、 VENDOR_ID および PART_NO 列に複合索引を設定した VENDOR_PARTS 表を示しています。

図 8-6 索引および主キー、一意キー、外部キー



通常の複合索引の最大列数は32であり、ビットマップ索引の最大列数は30です。またキー値は、データ・ブロックの利用可能なディスク領域のおよそ 2 分の 1 （オーバーヘッド分を差し引いたもの）を超えることができません。

索引とキー

「索引」と「キー」という 2 つの用語は同じ意味で使われることがありますが、これらの用語の区別を理解しておく必要があります。「索引」は、データベース内に実際に格納される構造体であり、ユーザーはSQL文を使って作成、変更、削除します。作成された索引により、表データへの高速なアクセス・パスが提供されます。一方、「キー」は厳密に論理的な概念です。キーは、整合性制約と呼ばれる Oracle の別の機能に対応します。

整合性制約は、データベースのビジネス・ルールを施行します（第 24 章「データの整合性」を参照）。いくつかのタイプの整合性制約は、索引によって施行されるため、キーと索引という用語が同じ意味で使われることもあります。しかし、これらの用語を混同しないようにしてください。

索引はどのように格納されるか

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、この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領域パラメータを設定することにより制御できる。
- 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ことによって制御できる。

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置く必要はありません。また、Oracle では索引と表のデータの両方を並行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。 25-7 ページの「ユーザー表領域の設定と割当て制限」を参照してください。

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・オーバーヘッドおよびエントリ・オーバーヘッド、 ROWID 、索引を付ける値 1 つあたり 1 バイト、の合計を引いたものです。索引ブロックのオーバーヘッドに対して必要となるバイト数は、オペレーティング・システムによって異なります。

> **追加情報**: 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・システム別の Oracle の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、索引がボトムからロードされます。たとえば、次の文を考えてみます。

```
CREATE INDEX emp_ename ON emp(ename);
```

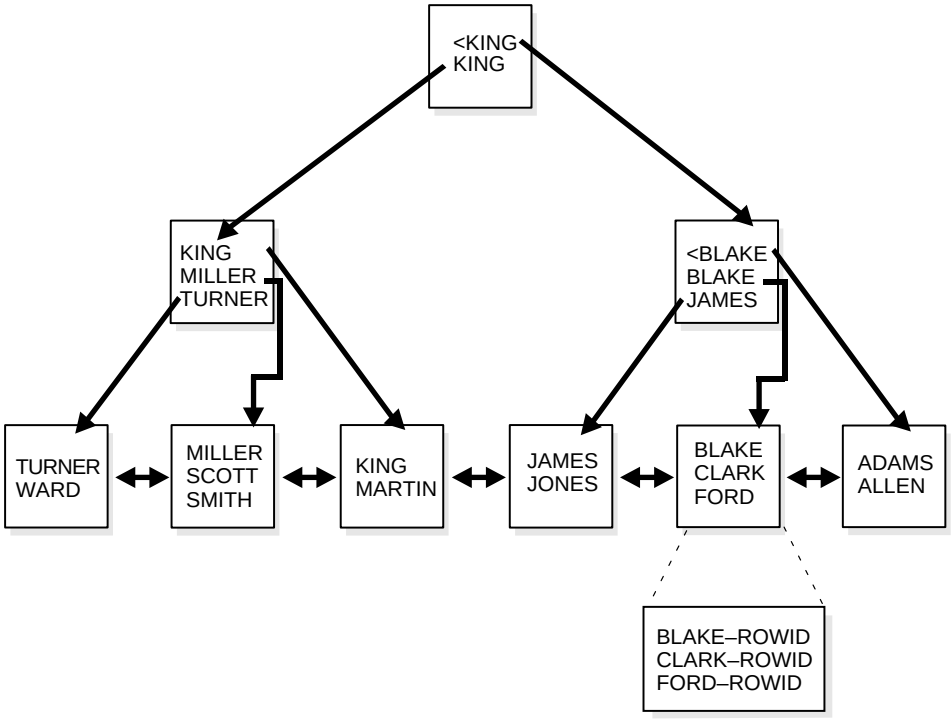
Oracle は、 ENAME 列に基づいて EMP 表をソートし、 次にこのソート順に、 ENAME とそれに対応する ROWID という形で索引をロードします。索引を使う場合、 Oracle は、ソートされた ENAME 値を迅速に検索し、その ENAME 値に対応する ROWID 値を使って、求める ENAME 列を持つ行を見つけます。

Oracle は CREATE INDEX コマンドのキーワードとして ASC および DESC 、 COMPRESS 、 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ってリーフ・ノードではなくブランチ・ノードに格納されている索引データには作用しません。

索引の内部構造

Oracle は、すべての行へのアクセス時間を均一化するために B* ツリー索引を使います。 B* ツリー索引の理論は、このマニュアルでは説明しません。詳細は、データ構造を扱ったコンピュータ・サイエンスの文献を参照してください。図 8-7 に、 B* ツリー索引の構造を示します。

図 8-7 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、実際の行を突き止めるためのROWIDが含まれます。リーフ・ブロックは、二重にリンクされます。文字データを持つ列の索引は、データベースのキャラクタ・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。一意でない索引の場合、ROWID はソートされた順序でキーに含まれるため、一意でない索引は索引キーと ROWID によってソートされます。クラスタ索引を除いて、 NULL だけ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 だけを含めた場合、一意索引に違反しません。

B* ツリー構造の利点

B* ツリーには次のような利点があります。

- 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、必要な時間はほぼ同じになる。
- B* ツリー索引は自動的にバランスが保たれる。
- B* ツリーのすべてのブロックの平均 4 分の 3 が満杯になる。
- 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する。
- 挿入、更新、削除は効率的で、高速検索のためにキー順序が維持される。
- 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ない。

逆キー索引

「逆キー索引」を作成する場合は、標準の索引とは違って、列の順序は保ちながら、索引が定義されている各列（ ROWID を除く）のバイトを逆にします。索引に対する変更が少数のリーフ・ブロックに集中するOracle Parallel Server環境では、この機能によりパフォーマンスの低下を防ぐことができます。索引のキーを逆にすることにより、挿入値は索引のリーフ・キー全体に分散されます。

逆キーを使うと、その索引に対する索引範囲走査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）走査しか実行できません。

状況によっては、逆キー索引を使った方がOLTP Oracle Parallel Serverアプリケーションのパフォーマンスが高速になることもあります。たとえば、Oracle Officeでメール・メッセージに対する索引を保持する場合、古いメッセージを保持するユーザーもいるので、索引は最新のメッセージだけでなく、それら古いメッセージに対するポインタもメンテナンスしなければなりません。

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 （再構築）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構築すると、逆キー索引が再構築されて生成されます。標準の索引を逆キー索引に再構築することはできません。その場合は、CREATE コマンドを使ってください。

ビットマップ索引

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

索引を使う目的は、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、その目的を達成するため、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います。

ビットマップ中の各ビットは、該当する可能性のあるROWIDに対応しています。あるビットがセットされているなら、それに対応する ROWID の行には該当するキー値が含まれています。マッピング機能がビット位置を実際のROWIDに変換するので、内部的には別の表現を使っているものの、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ければ、ビットマップ索引は領域を有効に利用することができます。

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。そのような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。

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

従来のB*ツリー索引を使っ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済みます。

ビットマップ索引は、数多くの並行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（ DSS ）に適しています。

ビットマップ索引は、 Oracle のコストベース最適化アプローチおよび実行エンジンと統合されます。また、他の Oracle 実行メソッドとシームレスに組み合わせて使用できます。たとえば、オプティマイザが 2 つの表をハッシュ結合することを決めたとします。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*ツリー索引を使います。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*ツリー索引や全表走査など）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、ローカル索引でなければなりません。詳細は、9-21ページの「索引のパーティション化」を参照してください。）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。

カーディナリティー

カーディナリティーの低い列ほど、ビットマップ索引を使ったときの効果は大きくなります。カーディナリティーの低い列とは、表全体の行数と比べて個別の値の数が少ない列のことです。ある列に同じ値が 100 個以上含まれているなら、その列はビットマップ索引の候補です。値の繰返しが少ない（つまりカーディナリテイーが高い）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なら、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、カーディナリティーの高いデータ、つまり個別の値が多いデータ（ CUSTOMER_NAME や PHONE_NUMBER など）に対して有効です。通常の B* ツリー索引は、索引を付けるデータよりも数倍大きくなることがあります。一方、ビットマップ索引は、適切に使えば B* ツリー索引より相当小さなサイズで済みます。

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う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ことによって解決速度を速くできます。結果の行数が少ないなら、全表走査を行う必要はないので、問合せはすぐに終わります。

ビットマップ索引の例

表 8-2 に、ある会社の顧客データの一部を示します。

表 8-2 ビットマップ索引の例

CUSTOMER #	MARITAL_ STATUS	REGION	GENDER	INCOME_ LEVEL
101	single	east	male	bracket_1
102	married	central	female	bracket_4
103	married	west	female	bracket_2
104	divorced	west	male	bracket_4
105	single	central	female	bracket_2
106	married	central	female	bracket_3

MARITAL_STATUS 、 REGION 、 GENDER 、 INCOME_LEVEL はすべてカーディナリティーの低い列（最初の 2 つには値が 3 つ、 3 番目には 2 つ、 4 番目には 4 つしかない）なので、これらの列にはビットマップ索引を作成できます。一方、 CUSTOMER# はカーディナリティーの高い列なので、ビットマップ索引は作成しない方がよいでしょう。この場合は、 B* ツリー索引を作成した方が、表示と検索が効率的になります。

表8-3に、この例のREGION列に対するビットマップ索引を示します。これは、各地域に対応する 3 つのビットマップから成り立っています。

表 8-3 サンプル・ビットマップ

REGION='east'	REGION='central'	REGION='west'
1	0	0
0	1	0
0	0	1
0	0	1
0	1	0
0	1	0

ビットマップの各項目（または「ビット」）はCUSTOMER 表の 1 つの行に対応しており、 各ビットの値は対応する行の値に依存しています。たとえば、 REGION='east' というビットマップの最初のビットは 1 です。 CUSTOMER 表の最初の行で region が "east" だからです。 REGION の値が "east" の行は他にないので、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うと、図 8-8 に示すとおり、結果のビットマップから該当する値を数えるだけで、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って表にアクセスできます。

図 8-8 ビットマップ索引を使った問合せの実行

status = 'married'		region = 'central'		region = 'west'						
0		0		0		0		0		0
1		1		0		1		1		1
1	AND	0	OR	1	=	1	AND	1	=	1
0		0		1		0		1		0
0		1		0		0		1		0
1		1		0		1		1		1

ビットマップ索引と NULL

他のほとんどのタイプの索引とは異なり、ビットマップ索引には NULL 値を持つ行が含まれます。 NULL の索引作成は、グループ関数 COUNT を使った問合せなど、いくつかのタイプの SQL 文に有用です。

例 1:

```
SELECT COUNT(*) FROM EMP;
```

NULL データを持つ行を含め、すべての表の行の索引が作成されるので、この問合せには任意のビットマップ索引を使えます。 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だけを使うことができます。

例 2:

```
SELECT COUNT(*) FROM EMP WHERE COMM IS NULL;
```

この問合せは、 COMM のビットマップ索引によって最適化できます。

例 3:

```
SELECT COUNT(*) FROM CUSTOMER
    WHERE GENDER = 'M' AND STATE != 'CA';
```

この問合せは、GENDER = 'M'のビットマップを検索し、STATE = 'CA'のビットマップを除くことによって回答されます。 STATE に NULL 値が含まれる場合（つまり、 NOT NULL 制約がない場合）は、 STATE = 'NULL' のビットマップも結果から除く必要があります。

パーティション表に対するビットマップ索引

他の索引の場合と同じように、ビットマップ索引もパーティション表に対して作成できます。ただし、制限が 1 つだけあります。ビットマップ索引はパーティション表に対してローカルでなければなりません。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、パーティション化されていない表でだけサポートされます。）

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 9 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。

> **追加情報**: ビットマップ索引の詳細は、『 Oracle8 Server チューニング』に記載されています。

索引構成表

「索引構成表」が通常の表と違うのは、表のデータが、対応付けられた索引に保持される点です。行の追加、更新、削除など、表データを変更しても、更新されるのは索引だけです。

索引構成表は、 1 つ以上の列に対して 1 つの索引が定義された通常の表と似ています。しかし、データベース・システムは、表本体と B* ツリー索引のために 2 つの領域をメンテナンスするのではなく、コード化されたキー値と、それに対応する行の関連列値が含まれた 1 つの B* ツリー索引をメンテナンスします。索引項目の 2 番目の要素として ROWID を格納するかわりに、実際のデータ行を B* ツリー索引に格納します。データ行は表の主キーに対して作成され、各 B* ツリー索引項目には <primary_key_value, non_primary_key_column_values> が含まれます。

索引構成表は、主キーまたは主キーの有効な接頭辞である任意のキーを使ってデータをアクセスするときに適しています。キー値の重複はありません。非キー列値はキーと一緒に格納されるからです。

アプリケーションは、通常の表と同じように、 SQL 文を使っ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 8-4 に、索引構成表と通常の表の違いをまとめます。

表 8-4 索引構成表と通常の表の比較

通常の表	索引構成表
ROWIDが行を一意に識別する。任意で、主キーを指定できる。	主キーが行を一意に識別する。主キーは必ず指定しなければならない。
暗黙の ROWID 列がある。物理的な 2 次索引を作成できる。	暗黙の ROWID 列がない。物理的な 2 次索引を作成できない。
ROWID に基づいてアクセスする。	主キーに基づいてアクセスする。
順次走査はすべての行を戻す。	全索引走査は主キーの順序ですべての行を戻す。
UNIQUE 制約とトリガーを定義できる。	UNIQUE 制約は定義できないがトリガーは定義できる。
表は、他の表が含まれるクラスタに格納できる。	索引構成表は、クラスタには格納できない。
分散環境、レプリケーション、パーティション化をサポート。	分散環境、レプリケーション、パーティション化はサポートされていない。

追加情報: 索引構成表の作成方法とメンテナンス方法の詳細は、『Oracle8 Server 管理者ガイド』を参照してください。

索引構成表の利点

索引構成表では、行が索引に格納されるので、完全一致または範囲検索を使った問合せの場合、キーを使ったデータ・アクセスが高速です。表および索引でキー列が重複しないので、記憶領域要件は軽減されます。キーと共に格納されるデータ行には、非キー列値だけが含まれます。また、データ行をキーと一緒に格納するので、通常の表の索引の場合に必要になる、キー値と対応する行をリンクする ROWID 用の追加領域が不要になります。

行オーバーフロー領域付きの索引構成表

B* ツリー索引項目は、 <key, ROWID> の組だけで成り立っているので、サイズは小さくて済みました。ところが、索引構成表の場合、 B* ツリー索引項目は<key,non_key_column_values> の組から成り立っているので、サイズは大きくなります。索引項目が非常に大きくなると、リーフ・ノードは 1 行または行の断片を格納するだけで満杯になり、高い密度で格納されるという B* ツリー索引の特徴が失われてしまいます。

この問題に対処するため、 Oracle では行オーバーフロー領域（Row Overflow Area）句が提供されています。オーバーフロー表領域としきい値を指定できます。しきい値は、ブロック・サイズのパーセントで指定します。

指定されたしきい値よりも行サイズが大きくなると、その行の非キー列値は、指定されたオーバーフロー表領域に格納されます。その場合、索引項目には<key, rowhead>の組が含まれます。 rowhead は、残りの列の先頭部分を表します。通常の行断片と似ていますが、 rowhead は残りの列値が含まれているオーバーフロー行断片を指すところが異なります。

索引構成表に適したアプリケーション

索引構成表は、特に次のようなアプリケーションに適しています。

- 情報検索（ IR ）アプリケーション
- Spatial Data アプリケーション
- OLAP アプリケーション

情報検索アプリケーション

「情報検索 (IR) アプリケーション」は、ドキュメント・コレクションに対する内容ベースの検索をサポートします。検索をサポートするために、 IR アプリケーションは、ドキュメント・コレクションに対する索引をメンテナンスします。「逆索引」には、通常、ドキュメントに含まれる個別のワードごとに、 <token, document_id, occurrence_data> という形式の項目が含まれています。内容ベースの検索では、アプリケーションは、索引を走査して該当するトークン（検索する値）を探します。

逆索引をモデル化する通常の表を定義できます。表データの検索を高速化するために、トークンに対応する列に対して索引を定義することもできます。しかし、この方法には次のような問題点があります。

- 索引を使ってデータを取り出すと、行あたり 1 つの ROWID ベース・フェッチが余分に増える。通常、内容ベースの IR 問合せでは、指定された問合せ用語を探すためにすべての逆索引項目をフェッチする必要があります。 IR アプリケーションの場合、重複は例外ではなく標準であるため、 1 つの問合せ用語につき何千もの重複値が存在することがあります。このため、行オーバーヘッドあたり1つのROWIDベース・フェッチというのは非常に重大なことで、 IR の検索パフォーマンスに多大な影響を与えます。
- 表および索引にキー（トークン）列の重複があると、無駄な領域が生じる。索引は巨大になることがあるので、領域要件が許容範囲を越えることがあります。

場合によっては、元になる表の複数列に連結索引を定義することにより、検索パフォーマンスが向上することがあります。連結索引を定義した場合、オカレンス・データが必要でない（つまり、ブール問合せ）なら、索引構成検索が可能です。そのような場合、逆表レコードのROWIDフェッチは行われません。問合せに近接述語（たとえば、「Oracle€orporation」）が含まれているなら、連結索引アプローチを使ったとしても、元になる表にアクセスしなければなりません。さらに、連結索引の作成とメンテナンスには、トークンに単一列の索引を定義した場合と比べて、より多くの時間がかかります。また、表と索引でキー（トークン）の複数列が重複するため、格納オーバーヘッドは高くなります。

索引構成表を使って逆索引を生成すると、上記の問題を回避できます。次のとおりです。

- データ行がキーと一緒に格納されているため、索引のオカレンス・データを検索することには、索引を走査して、該当するリーフ・ノードからデータ行を取得することが含まれる。
- 索引には、非キー列値だけがキーと一緒に格納される。したがって、データの重複はありません。また、通常の表に対してメンテナンスされている索引に必要になる、余分な ROWID 領域のオーバーヘッドは回避されます。

また、索引構成表はアプリケーションから参照できるので、協調的索引作成に適しています。協調的索引作成では、アプリケーションとデータベースが協調しながら、アプリケーション固有の索引を管理するからです。

Spatial Data アプリケーション

Spatial Data アプリケーションは、アプリケーション固有の索引をメンテナンスするためにある種の索引を使うので、索引構成表に適しています。

Spatial Data アプリケーションは、 Spatial Data 問合せを処理するために索引をメンテナンスします。たとえば、グリッドの集合の中に存在するオブジェクトに対する Spatial Data 索引は、次の形式のエントリを持つ逆索引としてモデル化できます。

```
<grid_id, spatial_object_id, spatial_object_data>
```

索引構成表は、必要な検索パフォーマンスと低い領域コストを同時に実現するので、そのような逆索引をモデル化するのに適しています。

OLAP アプリケーション

オンライン分析処理（OLAP）アプリケーションは、普通、複数次元ブロックを操作します。複数次元ブロックの一部分を高速に検索できるように、次元値のセットをページのセットにマップする索引をメンテナンスします。逆索引のエントリの形式は次のとおりです。

```
<dimension_value, list_of_pages>
```

OLAP アプリケーションがメンテナンスする索引は、索引構成表として簡単にモデル化できます。

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、頻繁に一緒に使用されます。

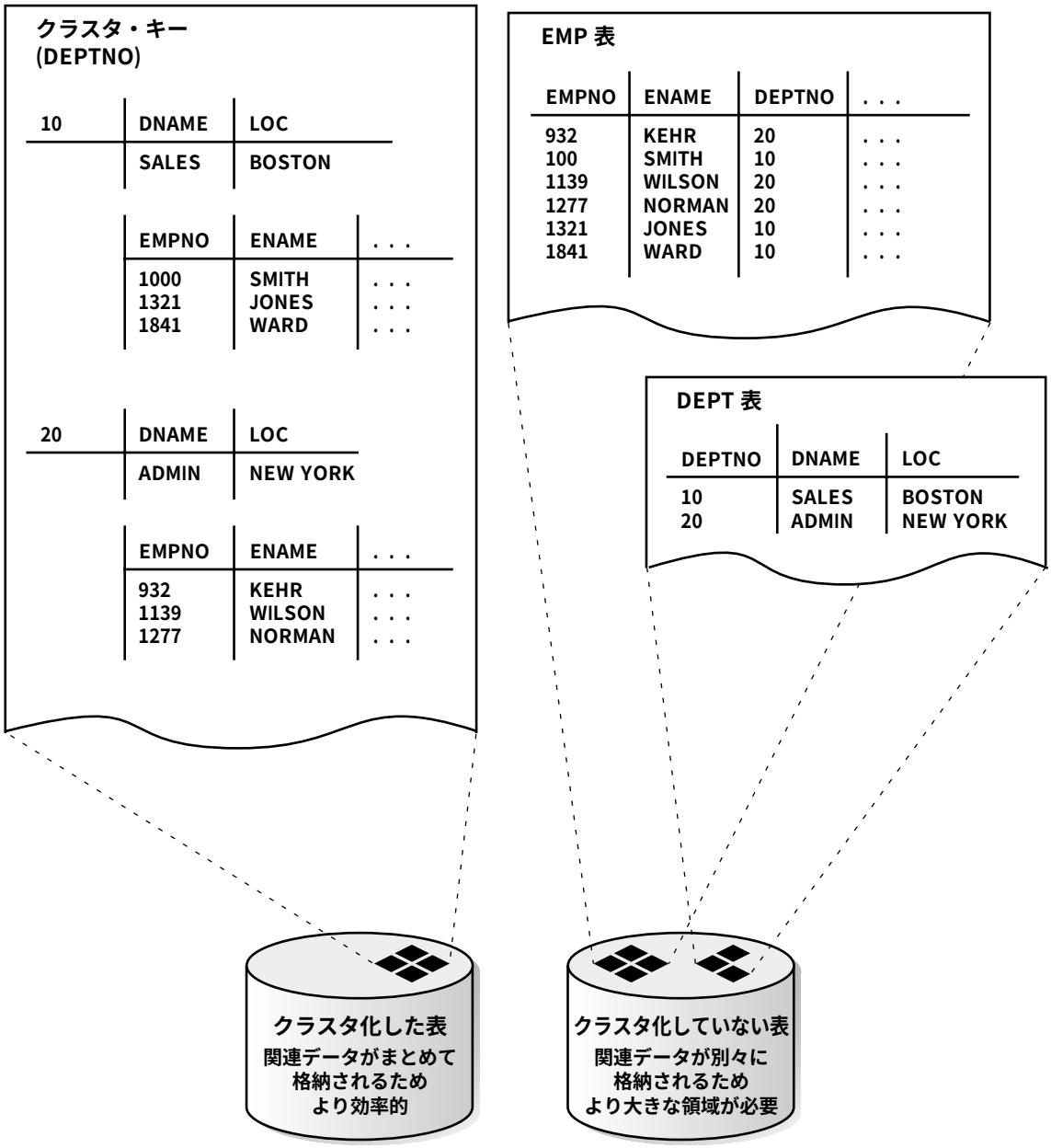
たとえば、 EMP 表と DEPT 表は DEPTNO 列を共有しています。 EMP 表と DEPT 表をクラスタ化すると（図 8-9 「クラスタ化表データ」）、 EMP 表と DEPT 表の両方に属する各部門のすべての行が物理的に同じデータ・ブロックに格納されます。

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うことには次の 2 つの基本的な利点があります。

- ディスク I/O が少なくなり、クラスタ化した表を結合するときのアクセス時間が改善される。
- 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る。

  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領域は、クラスタ化していない表で必要な記憶領域よりも少なくなる場合があります。たとえば、図 8-9 では、各クラスタ・キー（各 DEPTNO ）は、 EMP 表と DEPT 表の両方で同じ値を持つ多数の行に対して一度だけ格納されています。

図 8-9 クラスタ化表データ



パフォーマンスの考慮事項

それぞれに索引を付けて表に別々に格納した場合と比較すると、クラスタによって INSERT 文のパフォーマンスが低下することがあります。このような欠点は、領域の使用方法と、表を走査するために確認しなければならないブロック数に関係があります。複数の表が各ブロック内にデータを持っているため、クラスタ化した表にデータを格納するには、その同じ表をクラスタ化しない場合よりも多くのブロックを使わなければなりません。

クラスタ化して格納した方がよいデータを識別するには、参照整合性制約によって関連付けられている表と、JOIN を使って頻繁に一緒にアクセスされる表を見つけます。表データを結合するために使われる列に基づいて表をクラスタ化すると、問合せを処理するためにアクセスしなければならないデータ・ブロックの数が少なくなります。つまり、クラスタ・キーに基づく結合のために必要とされる行がすべて同じブロック内に存在します。そのため、結合のパフォーマンスが改善されます。同様に、個々の表をクラスタ化すると有効な場合もあります。たとえば、 EMP 表は、同じ部門の従業員に対する行をクラスタ化するために、DEPTNO 列に基づいてクラスタ化することもできます。アプリケーションがしばしば部門ごとに行を処理する場合に、このクラスタ化は有効です。

索引と同じように、クラスタがアプリケーションの設計に影響することはありません。クラスタの存在は、ユーザーとアプリケーションに対して透明です。クラスタ化された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に格納されているデータと同じように、SQL を使ってアクセスします。

追加情報: クラスタを使ったときのパフォーマンスへの影響の詳細は、『 Oracle8 Server チューニング』を参照してください。

クラスタ化されたデータ・ブロックの形式

一般に、クラスタ化されたデータ・ブロックは、クラスタ化されていないデータ・ブロックと同じ形式ですが、表ディレクトリに追加のデータがあります。ただし、同じクラスタ・キー値を共有するすべての行は、同一のデータ・ブロック内に格納されます。

クラスタの作成時に、 1 つのクラスタ・キー値に対するすべての行を格納するのに必要な領域の平均容量を、 CREATE CLUSTER コマンドの SIZE パラメータで指定します。 SIZE によって、データ・ブロックごとに格納できるクラスタ・キーの最大数が決まります。

たとえば、データ・ブロックごとに使用可能な領域が 1700 バイトで、指定されたクラスタ・キー・サイズが 500 バイトの場合、各データ・ブロックは 3 つのクラスタ・キーに対する行を保持できる可能性があります。 SIZE がデータ・ブロックで利用できる領域の容量よりも大きい場合、各データ・ブロックは 1 つのクラスタ・キー値に対応する行だけを保持します。

データ・ブロックあたりのクラスタ・キー値の最大数は SIZE によって固定されますが、Oracle は各クラスタ・キー値に対する領域を実際に確保するわけではありません。また、確実にその数のクラスタ・キーを各ブロックに割り当てるわけでもありません。たとえば、SIZE の指定によってデータ・ブロックあたり 3 つのクラスタ・キー値を格納できるという場合でも、 1 つのクラスタ・キー値の行がブロック内の使用可能な領域をすべて占有する可能性があります。特定のキーに対して、 1 つのブロックに収めることができる行数よりも多くの行が存在する場合は、必要に応じてブロックが連鎖されます。

クラスタ・キー値は、データ・ブロックに 1 度だけ格納されます。

クラスタ・キー

「クラスタ・キー」は、クラスタ化表が共通に持つ列、または列のグループです。クラスタを作成するときに、クラスタ・キーの列を指定します。その後、クラスタに追加する表を作成するたびに、その同じ列を指定できます。

クラスタ・キーの一部として指定される（クラスタの作成時）各列について、クラスタ内に作成するすべての表は、クラスタ・キー列のサイズとタイプに一致する列を持っていなければなりません。クラスタ・キーは最大 16 列で形成されます。またクラスタ・キー値はデータ・ブロック内の利用可能なディスク領域のおよそ 2 分の 1 （オーバーヘッド分を差し引いたもの）を超えることができません。クラスタ・キーに、LONG列またはLONG RAW列を含めることはできません。

表のクラスタ化列のデータ値は更新できます。しかし、データの配置はクラスタ・キーに依存しているため、行のクラスタ・キーを変更すると、 Oracle によって行の物理的な再配置が実行されることがあります。そのため、頻繁に更新される列は、クラスタ・キーには適していません。

クラスタ索引

クラスタを作成した後、クラスタ・キー列に対して索引を作成しなければなりません。「クラスタ索引」は、特にクラスタのために定義される索引です。この索引は、各クラスタ・キー値のエントリを含みます。

クラスタ内の行を突き止めるために、クラスタ索引を使ってクラスタ・キー値が検索されます。クラスタ・キー値は、そのキー値に対応付けられているデータ・ブロックへのポインタとして機能します。そのため、各行は、最低2回のI/Oによってアクセスされます（索引内を走査しなければならないレベルの数によっては、 2 回より多くなる場合もあります）。

クラスタ索引を作成しないと、そのクラスタ化表に対する DML 文（ INSERT 文と SELECT 文を含む）は実行できません。そのため、クラスタ化表のデータは、クラスタ索引が作成されるまでロードできません。

クラスタ索引は、表索引と同じように索引セグメントに格納されます。そのため、クラスタをある表領域に格納し、クラスタ索引を別の表領域に格納することができます。

クラスタ索引は、次の点で表索引と異なります。

- すべて NULL で構成されるキーが、クラスタ索引内にエントリを持つ。
- 索引エントリは、特定のクラスタ・キー値に対する連鎖の先頭ブロックを指す。
- クラスタ索引は、クラスタ行あたり 1 エントリではなく、クラスタ・キー値あたり 1 エントリを収録する。
- 表索引がなくても表データをアクセスできるが、クラスタ索引がないとクラスタ化データをアクセスできない。

クラスタ索引を削除すると、クラスタ内のデータはそのまま残りますが、新しいクラスタ索引を作成するまではデータを利用できません。別の表領域にクラスタ索引を移動したり、記憶特性を変更したりするために、クラスタ索引を削除する場合があります。そのような場合には、そのクラスタの索引を再作成しなければ、クラスタ内のデータをアクセスできません。

ハッシュ・クラスタ

ハッシングは、データ検索のパフォーマンスを改善するための、表データを格納するオプションの方法です。ハッシングを使うには、「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のに使います。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、単一の列または複合キー（複数列のキー）にすることができます（索引クラスタのキーと同じ）。ハッシュ・クラスタ内の行を検索または格納するために、Oracleはその行のクラスタ・キー値に対してハッシュ関数を適用します。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、発行された文による読み書きはそのデータ・ブロックに対して実行されます。

ハッシュ・クラスタは、索引を持つクラスタ化しない表、または索引クラスタに代わるものです。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位置付けられます。

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、最低2回のI/Oが実行されます（2回を超えることが多い）。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、表またはクラスタの行を読み書きするためにさらにもう1回のI/Oが実行されます。それとは対照的に、ハッシュ・クラスタでは、ハッシュ関数を使ってクラスタ内の行が位置付けられます（I/Oは不要）。その結果、ハッシュ・クラスタ内の行を読み書きする場合は、最低1回のI/O操作で済みます。

データはどのようにハッシュ・クラスタに格納されるか

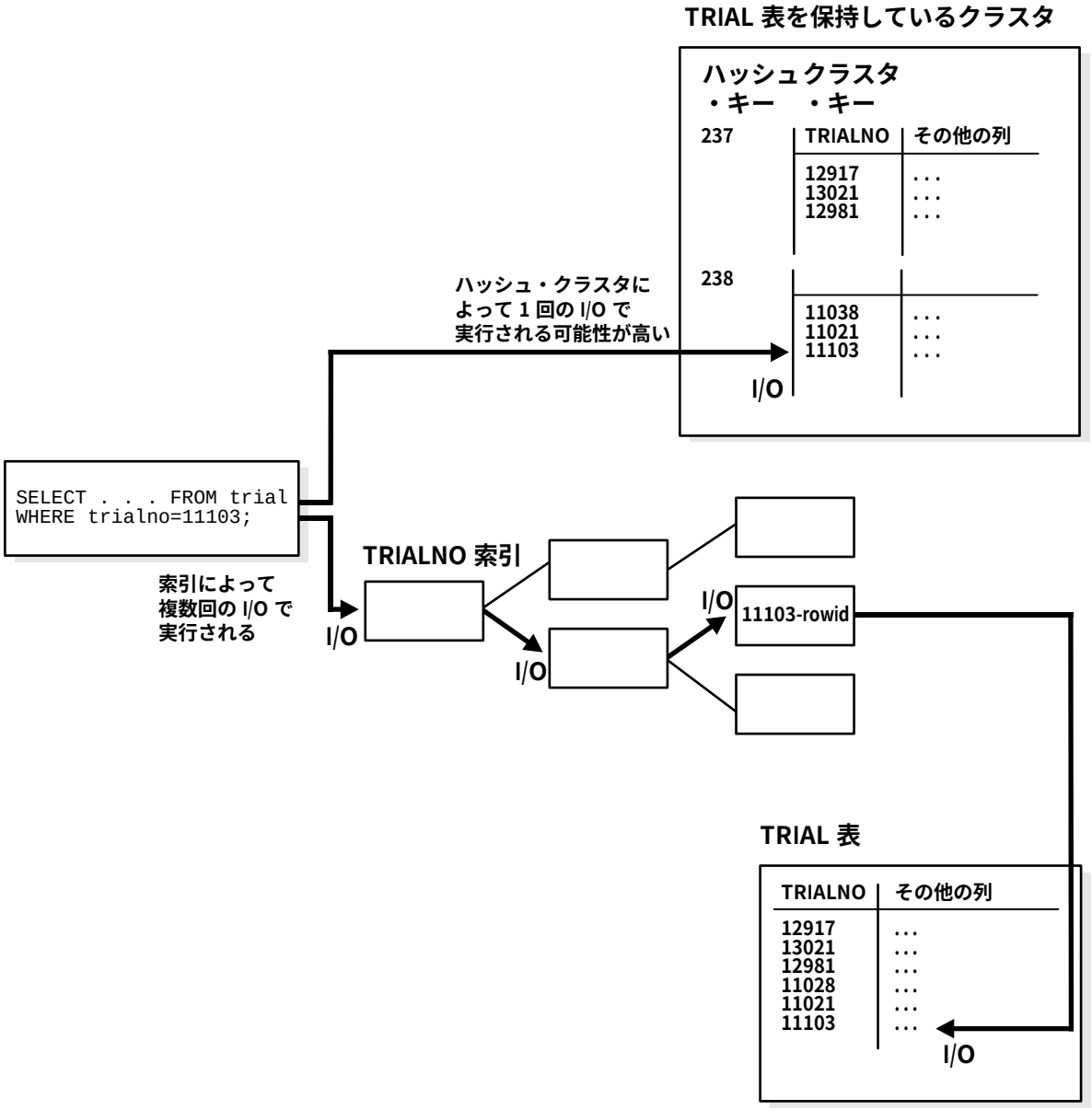
ハッシュ・クラスタは、関連する行をまとめて同じデータ・ブロック内に格納します。ハッシュ・クラスタ内の行は、そのハッシュ値に基づいてまとめて格納されます。

> **注意**：それとは対象的に、索引クラスタは、各行のクラスタ・キー値に基づいて、クラスタ化表の関連する行をまとめて格納します。

ハッシュ・クラスタを作成すると、そのクラスタのデータ・セグメントに対して記憶領域の初期容量が割り当てられます。Oracleによってハッシュ・クラスタに最初に割り当てられる記憶領域は、クラスタ内のハッシュ・キーの行数とサイズの予測値によって決められます。

図 8-10は、索引を持つ表とハッシュ・クラスタ内の表のデータ検索を示しています。この後の項では、ハッシュ・クラスタ記憶領域の内部操作について詳しく説明します。

図 8-10 ハッシングと索引付け： データ格納と情報



ハッシュ・キー値

ハッシュ・クラスタ内で行を検索または格納するために、 Oracle はハッシュ関数を行のクラスタ・キー値に適用します。結果として得られるハッシュ値は、クラスタ内の 1 つのデータ・ブロックに対応し、 Oracle は発行された文による読み書きをそのデータ・ブロックに対して実行します。ハッシュ・クラスタに対するハッシュ値の数は、作成時に固定されており、CREATE CLUSTER コマンドの HASHKEYS パラメータによって決まります。

HASHKEYSの値は、クラスタに対して使われるハッシュ関数によって生成できる一意のハッシュ値の数を制限します。HASHKEYSに指定した数値は、最も近い素数に丸められます。たとえば、 HASHKEYS を 100 に設定すると、任意のクラスタ・キー値に対しハッシュ関数は 0 から 100 までのハッシュ値を生成します（つまり、ハッシュ値は 101 個あります）。

そのため、ハッシュ・クラスタ内の行の分布は、HASHKEYSパラメータに対して設定する値によって直接的に制御されます。所定の行数に対してハッシュ・キー数を増やすと、「衝突」（2つのクラスタ・キー値が同一のハッシュ値を持っている状態）の可能性が低くなります。衝突したクラスタ・キー値を持つ複数の行を格納するためにオーバーフロー・ブロックが必要になる（つまり、余分なI/Oが必要になる）ことがあるので、衝突の数をなるべく少なくすることが重要です。

データ・ブロックごとに割り当てられるハッシュ・キーの最大数は、CREATE CLUSTERコマンドの SIZE パラメータで指定します。 SIZE は、各ハッシュ値に対応付けられる行の平均数を格納するために必要な領域の推定合計量（単位はバイト）です。たとえば、データ・ブロックあたりの使用可能な空き領域が 1700 バイトで、 SIZE が 500 バイトに設定される場合、データ・ブロックあたり 3 つのハッシュ・キーが割り当てられます。

注意: ハッシュ・クラスタの SIZE パラメータの重要性は、索引クラスタの SIZE パラメータの重要性と類似しています。しかし、索引クラスタでは、同じハッシュ値ではなく同じクラスタ・キー値を持つ行にSIZEが適用されます。

データ・ブロックあたりのハッシュ・キー値の最大数は SIZE によって指定されますが、Oracle はそれぞれのハッシュ・キー値に対する領域をブロック内に実際に確保するわけではありません。たとえば、 SIZE の指定によればブロックあたり 3 つのハッシュ・キー値を格納できるという場合でも、 1 つのハッシュ・キー値の行がブロック内の使用可能な領域をすべて占有する可能性があります。 1 つのブロックに収まる行数よりもハッシュ・キー値の行数のほうが多い場合、必要に応じてブロックが連鎖されます。

なお、各行のハッシュ値は行の一部としては格納されません。各行のクラスタ・キー値が格納されます。そのため、 SIZE の適切な値を決めるときには、格納するすべての行についてクラスタ・キー値を含めて考えなければなりません。

ハッシュ関数

ハッシュ関数は、クラスタ・キー値に適用されてハッシュ値を戻す関数です。その後、 Oracle はハッシュ値を使って、ハッシュ・クラスタの適切なデータ・ブロック内に行を位置付けます。ハッシュ関数の役割は、クラスタの使用可能なハッシュ値の間で行をなるべく分散させることです。この目標を達成するために、ハッシュ関数は衝突の数を最小化しなければなりません。

Oracle の内部ハッシュ関数を使用する

クラスタを作成するときには、 Oracle の内部ハッシュ関数を使うことも、その関数を使わないこともできます。内部ハッシュ関数では、クラスタ・キーは単一の列または複合キーです。

さらに、クラスタ・キーは、任意のデータ型（LONGとLONG RAWを除く）の列で構成できます。内部ハッシュ関数は、使用可能なハッシュ・キー間でクラスタ・キー値を十分に分散させ、どんなタイプのクラスタ・キーについても衝突数を最小にします。

クラスタ・キーをハッシュ関数として指定する

クラスタ・キーが、その範囲にわたって一様に分布している一意の識別子である場合、内部ハッシュ関数を使わないで、単にハッシュする列を指定することもできます。

Oracle は、内部ハッシュ関数を使ってハッシュ値を生成するかわりに、クラスタ・キー値をチェックします。クラスタ・キー値がHASHKEYSより小さい場合は、そのクラスタ・キー値をハッシュ値とします。しかし、クラスタ・キー値がHASHKEYS以上である場合は、そのクラスタ・キー値をHASHKEYSに指定された数値で割って、その余りをハッシュ値とします。つまり、「ハッシュ値」 = 「クラスタ・キー値」 MOD 「ハッシュ・キー数」となります。

クラスタ・キー値がクラスタ内で均等に分散している場合は、CREATE CLUSTERコマンドの HASH IS パラメータを使ってクラスタ・キー列を指定します。指定するクラスタ・キーは、位取りがゼロの数（整数）だけを含む単一列で構成されていなければなりません。内部ハッシュ関数を使わない場合に整数以外のクラスタ・キー値を指定すると、操作（ INSERT または UPDATE 文）はロールバックされ、エラーが戻されます。

ユーザー定義のハッシュ関数を指定する

ハッシュ・クラスタのためのハッシュ関数として SQL 式を指定することもできます。クラスタ・キー値がクラスタ内で均等に分散していない場合は、クラスタ行をハッシュ値にもっと効果的に分散させるユーザー固有のハッシュ関数を作成することを考えてください。

たとえば、従業員情報が入っているハッシュ・クラスタがあり、クラスタ・キーが従業員の自宅の市外局番である場合、多くの従業員が同一のハッシュ値にハッシュされる可能性があります。この問題を解決するには、CREATE CLUSTERコマンドのHASH IS句に次の式を記述します。

```
MOD((emp.home_area_code + emp.home_prefix + emp.home_suffix), 101)
```

この式は最初に市外局番の列を取り込み、これに電話番号の局番の列と番号の列を加算した値をハッシュ値の数（上の例では 101 ）で割り、その余りをハッシュ値として使います。その結果、クラスタ行はさまざまなハッシュ値にもっと均等に分散されます。

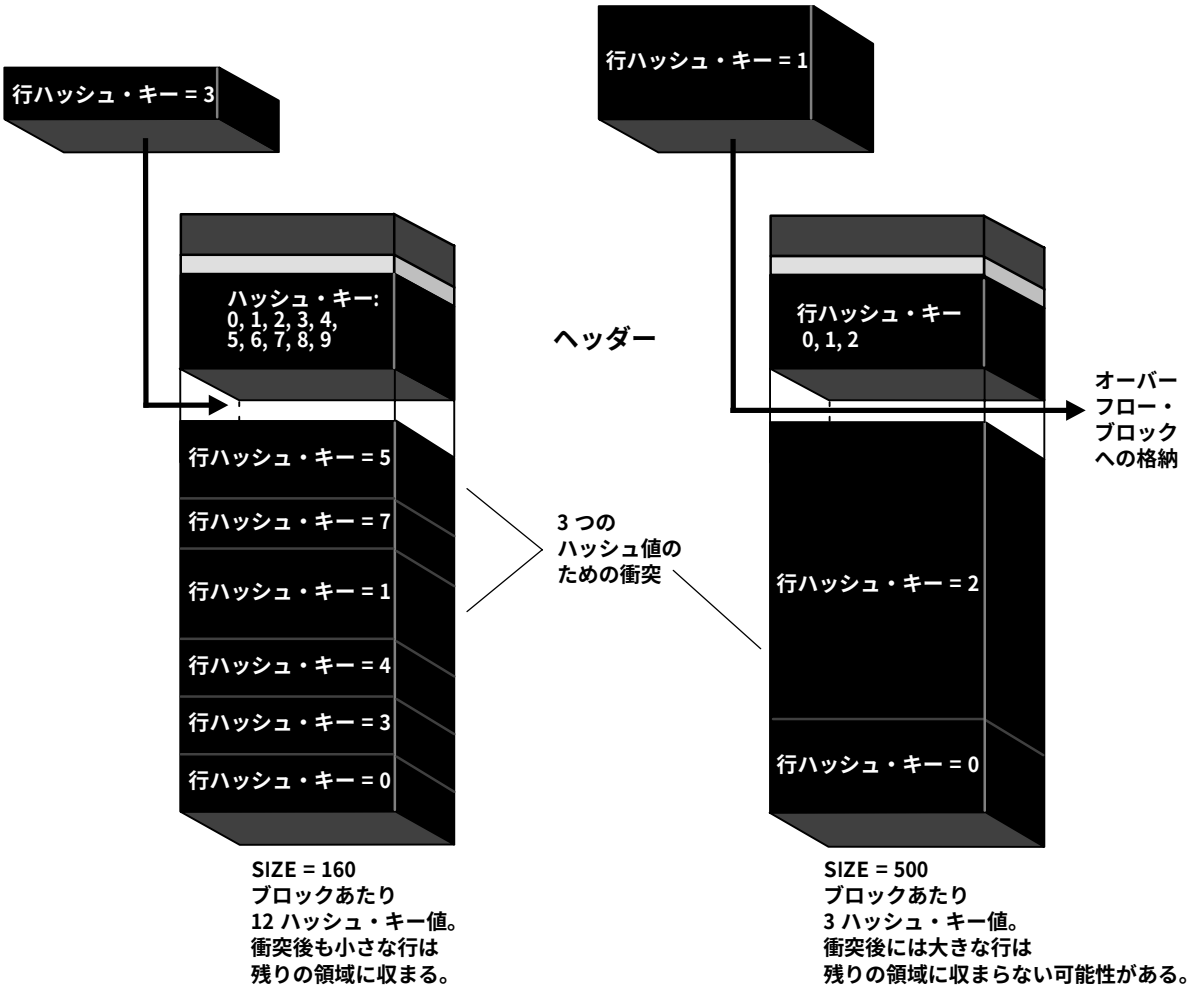
ハッシュ・クラスタに対する領域の割当て

他のタイプのセグメントと同じように、ハッシュ・クラスタの作成時のエクステントの割当ては、 STORAGE 句の INITIAL 、 NEXT 、 MINEXTENTS パラメータによって制御されます。ただし、ハッシュ・クラスタでは、「ハッシュ表」と呼ばれる領域の最初の部分が作成時に割り当てられ、クラスタのすべてのハッシュ・キーがマップできるようになります（領域全体は SIZE * HASHKEYS の大きさになる）。そのため、ハッシュ・クラスタに対する領域の初期割当ては、 SIZE と HASHKEYS の値にも依存しています。「 SIZE * HASHKEYS 」と、 STORAGE 句（ INITIAL 、 NEXT など）によって指定された値のうちどちらか大きい方が使われます。

その後にハッシュ・クラスタに割り当てられる領域は、満杯になったデータ・ブロックからオーバーフローした行を保持するために使われます。たとえば、特定のハッシュ・キーに対する最初のデータ・ブロックが満杯になっているとします。そのようなクラスタ化表にユーザーが行を挿入すると、行のクラスタ・キーは満杯のデータ・ブロックに格納されているハッシュ値にハッシュされます。そのため、そのハッシュ・キーに対して割り当てられた「ルート・ブロック」（元のブロック）に、行は挿入されません。そのかわりに、ハッシュ・キーのルート・ブロックに連鎖されたオーバーフロー・ブロックに行が挿入されます。

衝突が頻繁に発生すると、ハッシュ・クラスタ内のオーバーフロー・ブロックが増加する（したがってデータ検索のパフォーマンスが低下する）場合もあり、増加しない場合もあります。衝突が発生し、ハッシュ・キーに対して割り当てられた最初のブロック内に空き領域がない場合は、新しい行を保持するためにオーバーフロー・ブロックを割り当てる必要があります。図 8-11 に示されているように、このようなことが起こる可能性は、ハッシュ・クラスタの作成時に指定される、各ハッシュ・キー値とこれに対応するデータの平均サイズに大きく依存しています。

図 8-11 ハッシュ・クラスタ内の衝突とオーバーフロー・ブロック



平均サイズが小さく、各行が一意のハッシュ・キー値を持っている場合、データ・ブロックごとに多くのハッシュ・キー値を割り当てることができます。この場合、衝突している小さな行は、ハッシュ・キーに対するルート・ブロックの領域に収まる可能性が高くなります。しかし、ハッシュ・キー値の平均のサイズが大きいか、各ハッシュ・キー値が複数の行に対応する場合、データ・ブロックあたりに割当て可能なハッシュ・キー値はほんのわずかです。この場合、大きな行は、ハッシュ・キー値に対して割り当てられたルート・ブロックに収まらず、オーバーフロー・ブロックが割り当てられる可能性があります。

9

パーティション表とパーティション索引

Like to a double cherry, seeming parted,
But yet an union inpartition;
Two lovely berries molded on one stem.

Wm. Shakespeare: *A Midsummer-Night's Dream*

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、そしてパーティション化の管理における考慮事項について説明し、 次の内容を取り上げます。

- パーティション化とは何か
- パーティション化の利点
- パーティション化の基本的なモデル
- 表および索引をパーティション化するときのルール
- DML パーティション・ロック
- メンテナンス操作
- 索引の管理
- パーティション表およびパーティション索引についての権限
- パーティション表およびパーティション索引についての監査
- SQL の拡張 : 拡張パーティション表名

注意 : この章で説明している機能は、パーティション化オプション付きの Oracle8 Enterprise Edition を購入した場合にのみ利用できます。 Oracle8 Enterprise Edition で使用可能な機能とオプションの詳細は、『 Oracle8 と Oracle8 Enterprise Edition の解説』を参照してください。

パーティション化の基礎知識

この項では、 Oracle データベースで大規模な表や索引を管理するときにパーティション化がどのように役立つかについて説明します。

注意: Oracleは、表と索引のパーティション化だけをサポートしています。クラスタ化された表やその索引、そしてスナップショットのパーティション化はサポートしていません。

パーティション化とは何か

「パーティション化」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できるようにすることで、そのような表や索引をサポートするときの主な問題に対処します。

パーティションを定義すると、 SQL 文は、表または索引全体ではなくパーティションをアクセスしたり操作したりできるようになります。パーティションは、一般に大量の履歴データを格納および分析するデータ・ウェアハウス・アプリケーションに特に有用です。

表や索引のすべてのパーティションは、物理属性は異なっていることがありますが、論理属性はすべて同じです。たとえば、表内のすべてのパーティションは、同じ列定義と制約定義を共有します。また、索引内のすべてのパーティションは、同じ索引列を共有します。しかし、記憶領域の指定と、 PCTFREE 、 PCTUSED 、 INITRANS 、 MAXTRANS などのその他の物理属性は、同じ表または索引のパーティションごとに変えることができます。

各パーティションは、別個のセグメントに格納されます。任意で、各パーティションを別個の表領域に格納できます。そのことには、次のような利点があります。

- データが破壊される可能性を抑えることができる。
- 各パーティションを別々にバックアップしたり回復したりできる。
- パーティションを複数のディスク・ドライブに分配することにより、 I/O 負荷のバランスをとれる。

パーティション化の概念の詳細は、 9-10 ページの「パーティション化の基本的なモデル」の項に記載があります。

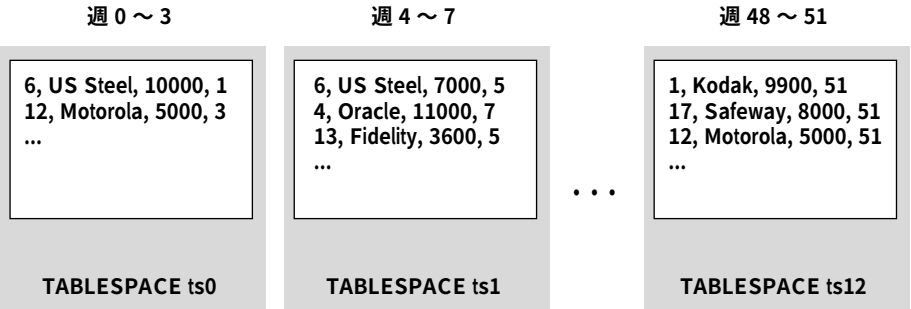
パーティション表の例

図 9-1 では、 sales 表の履歴データを週の番号に基づいて 13 個のパーティションに分けています。パーティション表を作成する SQL 文は次のとおりです。

```
CREATE TABLE sales ( acct_no         NUMBER(5),
                     acct_name       CHAR(30),
                     amount_of_sale  NUMBER(6),
                     week_no         INTEGER )
        PARTITION BY RANGE ( week_no ) ...
```

```
(PARTITION sales1 VALUES LESS THAN ( 4 ) TABLESPACE ts0,
 PARTITION sales2 VALUES LESS THAN ( 8 ) TABLESPACE ts1,
 ...
 PARTITION sales13 VALUES LESS THAN ( 52 ) TABLESPACE ts12 );
```

図 9-1 **週ごとにパーティション化した** SALES **表**



追加情報：パーティション表の詳細な例は『Oracle8 Server 管理者ガイド』を参照してください。

パーティションの切詰め

Oracle Serverには、パーティションを明示的に認識するためのインテリジェント機能が組み込まれています。この知識は、アクセスする必要のあるパーティションをマークし、不要なパーティションがこれらの SQL 文によるアクセスから排除（切詰め）されるようにするために、SQL 文の最適化で利用されます。

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が排除されます。たとえば、売上データQ1（第1四半期）だけについての問合せであれば、残りの3つに関するデータを取り出す必要はありません。このようなインテリジェント切詰め機能によって、データの量を大幅に減少させることができるので、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のすべての行が切詰めで使わ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。

パーティションの切詰めでは、基礎となる表のパーティションを排除できない場合であっても、索引と表が別々の列にパーティション化されていれば索引パーティションを排除できます。大規模な表に対する操作のパフォーマンスは、 SQL 文がアクセスまたは変更する必要のあるデータ量を削減するパーティション索引を作ることによって改善できます。

不要なパーティションを SQL 文から切り詰める機能によって、パーティション・レベルのロード、除去、バックアップ、復元、再編成、索引作成といった様々な操作のパフォーマンスと可用性が向上します。

パーティション化の利点

この項では、パーティションを使うことに利点があるデータベースのクラスを明らかにし、それに伴う問題点を説明します。

- 大規模データベース（ VLDB ）
- 定期メンテナンス操作による休止時間の短縮
- データ障害による休止時間の短縮
- DSS のパフォーマンス
- I/O のパフォーマンス
- ディスクのストライプ化 : パフォーマンスと可用性
- パーティションの透過性

大規模データベース（ VLDB ）

「大規模データベース（VLDB）」には、数百ギガバイトから数テラバイトのデータが含まれています。パーティション化は、構造化されていないデータではなく、大部分が構造化されたデータが入っている VLDB をサポートできます。 VLDB のサイズが大きいのは、一般に、非常に多くのデータ・オブジェクトがあるためではなく、数個の非常に大きなデータ・オブジェクト（表や索引）があるためです。

VLDB は、次の 2 つのカテゴリに大別できます。

- 「オンライン・トランザクション処理（ OLTP ）」データベースは、非常に多くの同時実行トランザクション向けに設計されています。各トランザクションは、少量のデータを処理する比較的簡単な操作です。
- 「意思決定支援システム（DSS）」は、大量のデータをアクセスして処理する必要のある、非常に複雑な問合せ向けに設計されています。

VLDB は、ほとんどの作業負荷が OLTP であれば、 OLTP データベースであるといえます。同様に、ほとんどの作業負荷が DSS 問合せであれば、 VLDB は DSS データベースであるといえます。

パーティション化は、 OLTP VLDB と DSS VLDB をどちらも効果的にサポートします。

履歴データベース

履歴データベースは、典型的な種類の DSS VLDB です。履歴データベースには、履歴データの表と全社データの表という 2 種類の表が含まれます。

- 「履歴」表には、最近の特定の期間（たとえば最近の 24 ヶ月間）における会社全体の業務取引きが記録される。履歴表には、次の 2 種類があります。

– 「実」表には、ベースライン情報（売上、伝票、注文など）が含まれる。

– 「ロールアップ」表には、 GROUP BY 、 AVERAGE 、 COUNT などの操作を使ってベース情報から導出したサマリー情報が含まれる。

履歴データの表に反映される期間は「巻上げ式」です。それで、データベース管理者（DBA）は、最も古いトランザクションを構成している一連の行を定期的に削除し、新しいトランザクションを構成する一連の行のために領域を割り当てます。たとえば、1997 年 4 月 30 日の業務が終了した時点で、 DBA は 1995 年 5 月の業務を記録した行（およびサポートしている索引項目すべて）を削除し、 1997 年 5 月の業務を記録するための領域を割り当てます。

履歴 VLDB のデータの大多数は、非常に大規模ないくつかの履歴データ表に格納されます。この表は、サイズが大きくなり、古いデータを削除して新しいデータを入力するという操作を円滑に行う必要があるため、特別な問題が提起されます。

- 「エンタープライズ」表には、会社全体のビジネス・エンティティ（部門、場所、製品など）が記録される。この情報は、時間の経過とともにゆっくりと変更が加えられます。定期的に修正されることはありません。全社データの表が大きくない場合でも、履歴データの表と全社データの表を結合する操作を含む、長時間実行される多くの DSS 問合せのパフォーマンスに影響を与えます。

パーティション化は、履歴データを時間に基づくパーティションに分割することにより、それらのパーティションを別々に管理したり自由に追加および削除したりできるようにして、大規模な履歴データ表やその索引をサポートする際の問題点に対処します。

ミッション・クリティカルなデータベース

ミッション・クリティカルな OLTP データベースには、非常に大きいものではなくても、可用性とパフォーマンスについて特別の問題があります。たとえば、非常に短時間（たいてい 1 時間未満）で、予定のメンテナンス操作を実行したり、 10 ギガバイトの表を回復したりする必要があります。さらに、表や索引を複数のドライブに分散させている場合にも、 DBA はデータの配置をある程度制御できる必要があります。

メンテナンス操作の時間枠、回復時間、障害の影響を減らす目的でクリティカルな表と索引がパーティションに分けられている場合、パーティション化により、ミッション・クリティカルなデータベースの可用性を高めることができます。パーティションごとにパフォーマンス・パラメータを制御することにより、クリティカルな表や索引へのアクセスのパフォーマンスを改善することもできます。

定期メンテナンス操作による休止時間の短縮

パーティションにより、データのロード、索引作成、データ・パージなどのデータ管理操作を表全体に対してではなくパーティション・レベルで実行できるので、これらの操作にかかる時間を大幅に軽減することができます。

パーティション化により、定期的なメンテナンス操作による休止時間の影響を大幅に軽減することができます。

- 表または索引全体ではなく、個々のパーティションごとに「パーティションのメンテナンス操作」を実行する。

- 別々のパーティションに対してメンテナンス操作を同時に実行できるように、「パーティションを独立」させる。

パーティションのメンテナンス操作

「パーティションのメンテナンス操作」は、表全体または索引全体のメンテナンス操作よりも高速です。次の比率でスピードアップを実現できます。

（表全体または索引全体のレコード数）/（パーティション内のレコード数）

これは、パーティションをまたがって格納される構成体（グローバル索引や参照整合性制約）がない場合の数字です。

休止時間をさらに短縮するために、パーティションのメンテナンス操作では、表レベルや索引レベルのメンテナンス操作で使えるパフォーマンス機能（ PARALLEL 、 NOLOGGING 、 DIRECT （または APPEND ）オプションなど）を活用できます。

パーティションの独立

パーティションのメンテナンス操作のために「パーティションを独立」させると、同じ表または索引の別々のパーティションに対してメンテナンス操作を同時に実行したり、メンテナンス操作の影響を受けないパーティションに対してSELECT操作やDML操作を同時に実行したりできます。

たとえば、パーティション PA と PB に同時にダイレクト・パス・ロードを実行し、アプリケーションではその他のパーティションに対して標準のSQL SELECT操作やDML操作を実行する、ということが可能です。

データの移動を含む操作では、パーティションの独立が特に重要です。そのような操作には非常に時間がかかります（数分、数時間、あるいは数日）。パーティション化すると、パーティションをまたいで格納される構成体（グローバル索引や参照整合性制約）がない場合に、データの移動を含む操作の際に他のパーティションが使えなくなる時間枠を短時間（数秒）に短縮できます。

時間のかからない操作（データの移動を伴わない操作）であれば短時間で完了するため、パーティションを独立させる必要はありません。

データ障害による休止時間の短縮

メンテナンス操作のなかには、予定外の出来事によるものがあります。たとえば、ハードウェア障害またはソフトウェア障害によってデータが消失または破損し、回復操作が必要になることがあります。データベース、表領域、データ・ファイルに対して RECOVER コマンドを実行することにより、ハードウェア障害や多くのシステム・ソフトウェア障害から回復できます。回復中の表領域またはデータ・ファイルにレコードが含まれている表や索引は、回復中には使えません。ミッション・クリティカルな OLTP データベースの場合には、可用性を高めることが特に重要です。

パーティションは互いに独立しているので、ある部分（またはある部分のサブセット）が使えなくても、それによってデータの他の部分が使えなくなることはありません。

パーティションを別々の表領域に格納することには、次のような利点があります。

- 回復の単位（表領域）が小さくなるため、 RECOVER コマンドの実行による休止時間が短くなる。
- 回復の単位が小さくなるため、オフライン表領域の回復に必要なディスク・リソース（遅延ロールバック・セグメント）が少なくなる。
- 回復する表領域に格納されているパーティションだけをオフラインにすればよいので、使えないデータの量が少なくなる。ユーザー・アプリケーションやメンテナンス操作で、別のパーティションにはアクセスできます。これは、パーティションの独立が役立つもう 1 つの例です。

DSS のパフォーマンス

非常に大きな表に対して DSS 問合せを実行すると、特別なパフォーマンス上の問題が生じます。表の走査が必要な非定型の問合せは、表内の各行を検査しなければならないため、時間がかかります。関係のない行のサブセットを識別してスキップする手段はありません。履歴データの表の場合は、多くの問合せが、最近生成された行を集中してアクセスするため、この問題は特に重要です。

パーティションを作れば、 DSS のパフォーマンス上の問題を解決する助けになります。 1 つのパーティション（または特定の範囲のパーティション）に対応する行だけを必要とする非定型問合せは、表の走査ではなくパーティションの走査を使って実行できます。

たとえば、 1997 年 10 月に生成されたデータを要求する問合せでは、何年にもわたるアクティビティによって生成された行ではなく、1997年10月に格納された行だけを走査できます。これにより応答時間が速くなり、ソートが必要な問合せのための一時ディスク領域を実質的に減らすこともできます。

I/O のパフォーマンス

パーティション化により、データを複数の物理デバイス間にどのように分散させるかを制御できます。I/Oの使用のバランスをとるため、表や索引の各パーティションを格納する場所を指定できます。

このレベルの位置制御があれば、ディスクの競合を減らし、より速いデバイスを使うことにより、高速な応答時間が求められるアプリケーションの特別な必要を満たすことができます。一方、あまり頻繁にはアクセスされないデータ（古い履歴データなど）は、遅いディスクに移したり、階層的な記憶領域をサポートするサブシステムに格納したりできます。

ディスクのストライプ化： パフォーマンスと可用性

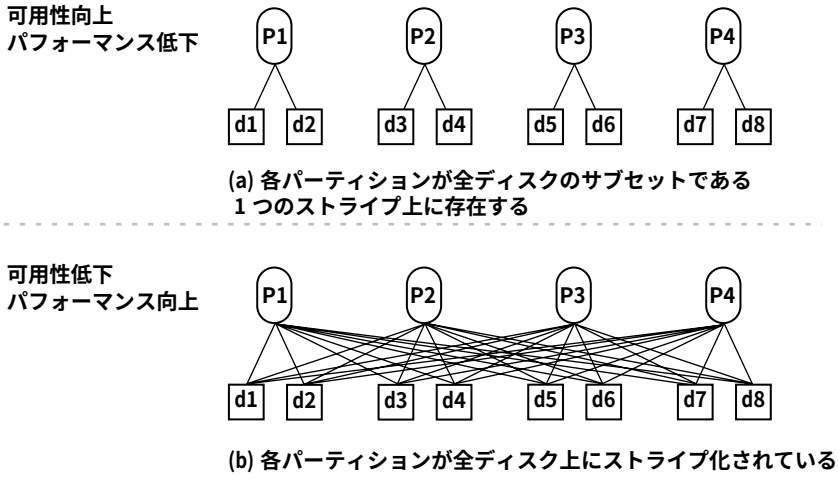
ディスクのストライプ化とパーティション化は、どちらもディスク・アームの競合を減らすことによりパフォーマンスを改善できるようにする手段です。どちらの手段を使うか、またどの範囲についてそれらの手段を使うかは、データベースを物理的に設計する段階で考慮する必要がある重要な問題です。これらの問題は、パフォーマンスの観点からだけではなく、可用性とパーティションの独立の観点からも考慮しなければなりません。

図 9-2は、パーティション化とストライプ化を結合した2つの極端な例を示しています。（a）と（ｂ）は、どちらも４つのパーティションを８台のディスクに分散させています。（ａ）は各パーティションを専用の1組のディスクにストライプ化しているのに対して（b）は各パーティションを８台すべてのディスクにストライプ化しています。

- （ｂ）はパフォーマンス特性に優れていますが、１台でもディスクが障害が起こすと、すべてのパーティションが悪影響を受けます。
- （ａ）は、１台のディスクが障害を起こしても１つのパーティションにしか影響がないため、可用性の特性に優れています。

この２つの中間的な構成も可能であり、パーティションのサブセットをディスクのサブセットにストライプ化することができます。

図 9-2 **パーティションとディスクのストライプ化**



表と索引をどのようにパーティション化するか、そして各パーティションを格納するディスクをどのようにストライプ化するかを決定するときには、パフォーマンスと可用性を比較考量する必要があります。

ミッション・クリティカルなデータベースの場合は、パーティションの独立と可用性を優先することをお薦めします。したがって、複数のディスクにストライプ化するパーティションは、それぞれ専用のディスク・ドライブ・セットにストライプ化するべきです。ディスクのセットには十分な数のドライブを含めるようにして、特定のパーティションへのアクセスに必要な I/O の並列性を達成します。

パーティションの透過性

大多数のアプリケーション・プログラムには「パーティションの透過性」が必要です。つまり、そのプログラムでアクセスするデータがパーティション化されているかどうか、またどのようにパーティション化されているかを、プログラムからは意識しないで済むようでなければなりません。

しかし、一部のアプリケーション・プログラムでは、表全体ではなくて個々のパーティションにアクセスするよう明示的に要求することにより、パーティションを活用できます。たとえば、非常に大規模な表に対して実行する長いバッチ・ジョブを、個々のパーティションに対して 1 晩ずつ短時間で実行するシーケンスに分けることがあります。

パーティション・ビューを使った手動のパーティション化

パーティション表を使うかわりに、同一のテンプレートを使って表をいくつか作成し、それらの表を結合（UNION）するビューを定義できます。この操作を「手動パーティション化」といい、このビューを「パーティション・ビュー」といいます。

パーティション・ビューは、 Oracle7 、リリース 7.3 で使えるパーティション化の唯一の形態です。 Oracle8 の新しいアプリケーションに使うことはお薦めしません。 Oracle7 データベースで作成されたパーティション・ビューは、 ALTER TABLE コマンドの EXCHANGE PARTITION オプションを使ってパーティション表に変換できます。

注意: Oracle8 は、 Oracle7 、リリース 7.3 との下位互換性のためだけにパーティション・ビューをサポートしています。

追加情報: パーティション・ビューをパーティション表に変換する手順は、『Oracle8 Server移行ガイド』と『Oracle8 Server管理者ガイド』を参照してください。

パーティション・ビューの基本概念は、パーティション基準（ WHERE 句または CHECK 制約）を使って大きな表を複数の物理的な表に分け、UNION ALLビューを使って小さな表をまとめて表示することです。その後、実表に対して同一のキー指定による「実索引」の集合を定義し、UNION ALLビューを使うときに索引の機能を提供します。パーティション・ビューが正しく動作するためには、索引を作成しなければなりません。

キー範囲を使ってパーティション・ビューから選択を行う問合せがアクセスできるのは、そのキー範囲内にある実表だけです。オプティマイザは、パーティション・ビューの表に対して別々の実行計画を使えます。（対照的に、オプティマイザはパーティション表内のすべてのパーティションに対して 1 つの実行計画を使います。）

パーティション・ビューを使った手動パーティション化には、不利な点が多くあります。

- 構成が複雑である。

  データベース管理者は、パーティションに対応する実表と実索引を正しく定義しなければならず、これらの定義をメンテナンスする必要があります。パーティションをまたい

でデータを移動する DDL 操作（ split 、 move など）と等価の機能は、 Export/Import ユーティリティまたは SQL スクリプトによってインプリメントしなければなりません。

- パーティションの透過性が不足している。

  一部の SQL 操作は、 UNION ALL ビューではなく実表を使って実行しなければなりません。たとえば、 INSERT では実表を参照するので、 INSERT 文に指定する表名を知るためにユーザー・コードが必要になります。

- パフォーマンスが低い。

  UNION ALLビューに対するSQL操作の中には、オプティマイザが既存の実索引すべてを使わないために、パフォーマンスが低いものがあります。

- メモリーの利用効率が悪い。

  UNION ALLビューに対して実行されるSQLコンパイル済みの問合せは、そのビューをサポートするすべての表のための記述情報を内部的にレプリケートします。

- DDL の制限。

  グローバル索引と参照整合性制約は、 UNION ALL ビューに対しては定義できません。

- ロードの制限

  UNION ALL ビューに対してダイレクト・ロードは実行できません。

パーティション化の基本的なモデル

パーティション化は、CREATE TABLE文とCREATE INDEX文のオプションを使って指定します。パーティション表またはパーティション索引を作成した後、そのパーティション化の属性を修正するには、 ALTER TABLE または ALTER INDEX 文を使います。 CREATE TABLE 文と CREATE INDEX 文のパーティション化の構文はよく似ています。

CREATE TABLE 文では、次の事項を指定します。

1. 列や制約の定義などの、表の論理属性。
2. 表の物理属性。
   - 表が非パーティション表であれば、この属性はその表に対応付けられたセグメントの実際の物理属性である。
   - 表がパーティション表であれば、この表レベルの属性により、その表の個々のパーティションのデフォルトが指定される。
3. パーティション表の場合、「パーティションの指定」。次のものを指定できます。
   - 行をパーティションにマップするときに使う表レベルのアルゴリズム
   - 表内の各パーティションにつき 1 つの、パーティション記述のリスト

各パーティション記述には、行をパーティションにマップするときのアルゴリズムについて補足的なパーティション・レベルの情報を定義する句が含まれます。この句では、そのパーティションのパーティション名と物理属性も指定できます。

データ型の制限

パーティション表の場合、論理属性には追加の制限事項があります。パーティション表には、データ型が LONG データ型、 LONG RAW データ型、 LOB データ型（ BLOB 、 CLOB 、 NCLOB 、 BFILE ）、またはオブジェクト型である列を含めることはできません。

表（または索引）が、DATE データ型を持つ1つの列上でパーティション化されている場合、そのパーティションの記述ではTO_DATE書式マスクを使う必要があります。そうしないと、パーティションの切詰めを行うことはできません。 9-15 ページの「 TO_DATE 書式マスク」を参照してください。

ビットマップの制限

パーティション表に対してビットマップ索引を作成できますが、ビットマップ索引はパーティション表に対してローカルでなければならないという制限があります。グローバル索引にすることはできません。（ 9-21 ページの「索引のパーティション化」を参照してください。）

コストベースの最適化

コストベースのオプティマイザは、 SQL 文がパーティション表またはパーティション索引にアクセスする場合に使います。ルールベースの最適化は、パーティションでは使えません。パーティション表のすべてのパーティションに対して単一の実行計画を使います。

ANALYZE コマンドを使って、パーティションごとに統計を収集できます。特に、パーティション表内のデータの特性に大きな変化があったときは、統計を収集することが重要になります。統計は、次のデータ・ディクショナリ・ビューで見ることができます。

- ALL_TAB_PARTITIONS 、 DBA_TAB_PARTITIONS 、 USER_TAB_PARTITIONS
- ALL_IND_PARTITIONS 、 DBA_IND_PARTITIONS 、 USER_IND_PARTITIONS
- ALL_PART_COL_STATISTICS 、 DBA_PART_COL_STATISTICS 、 USER_PART_COL_-STATISTICS

レンジ・パーティション化

「レンジ・パーティション化」は、列値の範囲に基づいて行をパーティションにマップします。レンジ・パーティション化は、表または索引についてパーティション化の指定をすることによって定義します。

```
PARTITION BY RANGE ( column_list )
```

さらに、個々のパーティションごとにパーティション化の指定をします。

```
VALUES LESS THAN ( value_list )
```

各値の意味は次のとおりです。

- column_list は、列の順序付きリストであり、行または索引エントリがどのパーティションに属するかを指定する。
  - これらの列を「パーティション化列」という。
  - 特定の行のパーティション化列の値は、その行の「パーティション化キー」を構成する。
- value_list は、 column_list に指定した列の値の順序付きリストである。
  - value_list の各値は、リテラルか、定数を引数に持つ TO_DATE() または RPAD() 関数のいずれかでなければならない。（9-15ページの「TO_DATE書式マスク」を参照。）
  - i 番目のパーティションのパーティション化を指定する value_list は、そのパーティションの上限（その値は含まない）を定義する。この上限を「パーティション・バウンド」といいます。
  - i番目のパーティションのパーティション・バウンドは、パーティションの上限（その値は含まない）を定義する。i番目のパーティションのパーティション・バウンドは、 (i+1) 番目のパーティションのパーティション・バウンドより小さくなければなりません。

i番目のパーティションでは、すべての行（または索引エントリが指している行）の「パーティション・キー」は、そのパーティションの「パーティション・バウンド」未満になっています。 i 番目のパーティションが表または索引の先頭のパーティションでない場合、 i 番目のパーティションにあるパーティション化キーすべては、 (i-1) 番目のパーティションのパーティション・バウンド以上になっています。（パーティション化キーとパーティション・バウンドが比較される方法や、複数列からなるパーティション化キーが処理される方法の詳細は、 9-13 ページの「パーティション・バウンドとパーティション化キー」を参照してください。）

たとえば、 4 つのパーティション（四半期の売上ごとに 1 つ）がある次の表では、「sale_year=1997、sale_month=7、sale_day=18」の行は、パーティション化キー（1997、7 、 18 ）であり、 3 番目のパーティションに属しているので、表領域 tsc に格納されます。「sale_year=1997、 sale_month=7、 sale_day=1」の行は、パーティション化キー（1997、7、1）であり、これも3番目のパーティションに属しているので、表領域tscに格納されます。

```
CREATE TABLE sales
    ( invoice_no NUMBER,
      sale_year INT NOT NULL,
      sale_month INT NOT NULL,
      sale_day INT NOT NULL )
  PARTITION BY RANGE( sale_year, sale_month, sale_day)
    ( PARTITION sales_q1 VALUES LESS THAN( 1994, 04, 01)
        TABLESPACE tsa,
      PARTITION sales_q2 VALUES LESS THAN( 1994, 07, 01)
        TABLESPACE tsb,
```

```
PARTITION sales_q3 VALUES LESS THAN( 1994, 10, 01)
  TABLESPACE tsc,
PARTITION sales_q4 VALUES LESS THAN( 1995, 01, 01)
  TABLESPACE tsd );
```

パーティション名

各パーティションには名前があります。パーティション名は、スキーマ・オブジェクトとその部分を命名するときの通常のルールに適合していなければなりません。特に、次の点に注意してください。

- 表パーティションの名前は、同じ親表に属するすべてのパーティションの中で一意でなければならない。
- 索引パーティションの名前は、同じ親索引に属するすべてのパーティションの中で一意でなければならない。

パーティションは改名できますが、パーティション名のシノニムを作ることはできません。

> **追加情報**: スキーマ・オブジェクトの命名のルールの詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

パーティションの参照

パーティション名は、 DDL 文や DML 文において、さらには Import/Export や SQL*Loader などのユーティリティ文において参照することができます。パーティション名の指定は、その親表または親索引の名前を指定するコンテキストでのみ可能であり、スキーマ名で修飾することはできません。（親オブジェクトを修飾するためのスキーマ名は使えます。）

たとえば、次のとおりです。

```
ALTER TABLE admin.patient_visits DROP PARTITION pv_dec92
```

SQL 文でパーティションを参照する方法の詳細は、9-39 ページの「SQL の拡張: 拡張パーティション表名」を参照してください。

パーティション・バウンドとパーティション化キー

この項では、ある行がどのパーティションに属するかを判別するため、その行のパーティション化キーが一連の上限および下限とどのように比較されるかについて説明します。

パーティション・バウンド

表と索引のすべてのパーティションには、VALUES LESS THAN句によって指定された「上限値」（その値未満）があります。最初のパーティションを除くすべてのパーティションには、1つ前のパーティションのVALUES LESS THANによって指定された「下限値」（その値以上）があります。

パーティション・バウンド全体で、表または索引内のパーティションの順序を定義します。「最初」のパーティションは、 VALUES LESS THAN 句の値が最も低いパーティションであり、「最後」もしくは「最高」のパーティションは、 VALUES LESS THAN 句の値が最も高いパーティションです。

表に挿入しようとした行のパーティション化キーが、表内で最も高いパーティションのパーティション・バウンド以上であれば、挿入は失敗します。

パーティション化キー

1 つの「パーティション化キー」は、最大 16 列までの順序付きリストで構成されます。ある行のパーティション化キーは、パーティション化列の値を順番に並べたリストです。

パーティション化キーには、 LEVEL 、 ROWID 、 MLSLABEL 疑似列または ROWID 型の列を含めません。

パーティション化キーとパーティション・バウンドの文字値の比較では、文字はそのバイナリ値に基づいて比較されます。ただし、 1 文字が複数バイトからなる場合は、 Oracle は文字ではなく各バイトのバイナリ値を比較します。

この比較では、列のデータ型に対応する比較のルールも適用されます（たとえば、 ANSI CHAR データ型の比較では、空白埋めによる比較が行われます）。 NLS パラメータ、特に初期化パラメータ NLS_SORT および NLS_LANGUAGE と環境変数 NLS_LANG は、比較には影響を与えません。

MAXVALUE

パーティション・バウンド value_list のどの値についても、キーワード MAXVALUE を指定できます。このキーワードは、仮想の「無限大」値を示しており、そのデータ型の他のどんな値（ NULL 値を含む）よりも高い値としてソートされます。

たとえば、 office 表を state 列（ CHAR(10) 列）に基づいてパーティション化するには、次のようなパーティション・バウンドを使って 3 つのパーティションに分けることができます。

- VALUES LESS THAN (’I’): A から H までの文字で始まる州が入る。
- VALUES LESS THAN (’S’): I から R までの文字で始まる州が入る。
- VALUES LESS THAN (MAXVALUE): SからZまでの文字で始まる州と、米国以外の特殊コードで始まる地域が入る。

NULL

パーティション・バウンド value_list に NULL は指定できません。パーティション・バウンド value_list の値として空の文字列を指定することもできません。データベース・サーバーでは空の文字列が NULL として処理されるからです。

行をパーティションに割り当てる目的で、 Oracle は NULL 値を MAXVALUE 以外のすべての値より大きな値としてソートします。 NULL は、 MAXVALUE よりは小さい値としてソートされます。

したがって、 NULL を受入れ可能な列に基づいて表をパーティション化する場合、その列に NULL 値が含まれるのであれば、その列の最も高いパーティションのパーティション・バウンドには MAXVALUE を指定するべきです。そうしないと、 NULL 値が含まれる行は表の最も高いパーティションよりも上にマップされるため、挿入は失敗します。

TO_DATE 書式マスク

パーティション・キーに、 DATE データ型を持つ列が含まれる場合は、 TO_DATE() 書式マスクを使ってパーティション・バウンドを指定しなければなりません。そうしないと、パーティションの排除（「切詰め」）を行うことはできません。

たとえば、 DATE 列を使って sales 表を作成するとします。

```
CREATE TABLE sales
    ( invoice_no NUMBER,
      sale_date DATE NOT NULL )
  PARTITION BY RANGE (sale_date)
    ( PARTITION sales_q1
        VALUES LESS THAN (TO_DATE('94-04-01','YY-MM-DD'))
        TABLESPACE tsa,
      PARTITION sales_q2
        VALUES LESS THAN (TO_DATE('94-07-01','YY-MM-DD'))
        TABLESPACE tsb,
      PARTITION sales_q3
        VALUES LESS THAN (TO_DATE('94-10-01','YY-MM-DD'))
        TABLESPACE tsc,
      PARTITION sales_q4
        VALUES LESS THAN (TO_DATE('95-01-01','YY-MM-DD'))
        TABLESPACE tsd );
```

たとえば、 sales 表のデータを問合せまたは変更するときにも、 TO_DATE() 書式マスクを使う必要があります。

```
SELECT * FROM sales
  WHERE sale_date < TO_DATE('94-06-15','YY-MM-DD');
```

複数列からなるパーティション化キー

複数の列に基づいて表や索引をパーティション化すると、各パーティション・バウンドとパーティション化キーは、値のリスト（ベクトル）になります。この場合、キーは ANSI SQL2 のベクトル比較ルールに基づいて順序付けられます（これは、 Oracle で複数列の索引キーが順序付けられるときと同じ方法です。）

ベクトル V1 と V2 が同数の値を含んでおり、「 Vx[i] 」は、 Vx にある i 番目の値を示すとします。さらに、 V1[i] と V2[i] のデータ型には互換性があるとします。このとき、比較の結果は次のようになります。

- 「 V1 = V2 」になるのは、すべての i について V1[i] = V2[i] である場合。
- 「 V1 < V2 」になるのは、ある n について V1[n] < V2[n] であり、 n より小さいすべての i について V1[i] = V2[i] である場合。
- 「 V1 > V2 」になるのは、ある n について V1[n] > V2[n] であり、 n より小さいすべての i について V1[i] = V2[i] である場合。

つまり、パーティション化キー PK がパーティション・バウンド PB 以下であるかどうかを知りたい場合は、 PK と PB の対応する値を比較していき、値が等しくない組合せが見つかった時点で、その組合せの大小関係によって PK と PB の大小関係が決まります。

たとえば、パーティションPのパーティション・バウンドが（7, 5, 10）であり、次に低いパーティションのパーティション・バウンドが（ 6, 7, 3 ）であれば、次のようになります。

- 次の理由で、キー（ 6, 9, 11 ）と（ 7, 3, 15 ）はパーティション P に入る。
  - キー（ 6, x, x ）は、（ 7, x, x ）より小さく、（ 6, 9, x ）は（ 6, 7, x ）より大きい。
  - キー（ 7, 3, x ）は、（ 7, 5, x ）より小さく、（ 6, 7, x ）より大きい。
- キー（ 6, 5, 0 ）と（ 7, 5, 11 ）は、 P 以外のパーティションに入る。

パーティション・バウンド value_list の 1 つの要素に MAXVALUE がある場合は、それ以降の要素の値はすべて無関係になります。たとえば、パーティション・バウンド（ 10, MAXVALUE, 5）は、パーティション・バウンド（10, MAXVALUE, 6）またはパーティション・バウンド（ 10, MAXVALUE, MAXVALUE ）と等価です。

複数列のパーティション化キーが役立つのは、表の主キーに複数の列が含まれているが、キーの中で最も重要な列に基づいて行を分配しても均等に分けることができないような場合です。たとえば、supplier_parts表に、どの業者がどの部品を納入しているかの情報を記録しており、その表の主キーは(suppnum, partnum)であるとします。一部の業者は数百～数千種類の部品を納入しているものの、その他の業者は数種類の特殊な部品しか納入していないため、 suppnum （業者番号）に基づいてパーティションを作るだけでは不十分です。その代わり、 (suppnum, partnum) に基づいて表をパーティション化できます。

複数列のパーティション化キーは、日付を DATE 列ではなく、 3 つの CHAR 列で表現している場合にも役立ちます。

パーティション・バウンドによって課される暗黙的な制約

表内の最も高いパーティションとして MAXVALUE 以外のパーティション・バウンドを指定した場合、その表には暗黙のCHECK制約が課されます。この制約は、データ・ディクショナリ内には記録されません（しかし、パーティション・バウンドそのものは記録されます）。

同一レベル・パーティション化

2 つの表または索引に同一の論理パーティション化属性がある場合のことを、「同一レベル・パーティション化」といいます。それらは同じ型のオブジェクトである必要はありません。たとえば、表と索引でも同一レベル・パーティション化を実施できます。

A と B というパーティション表について、 A[i] が A における i 番目のパーティションであり、B[i] が B における i 番目のパーティションである場合、次のすべての条件が満たされていれば、 A と B には同一レベル・パーティション化が実施されています。

- パーティション数が同数（ N ）である。
- パーティション化列が同数（ M ）である。
- 1 <= i <= N のすべての i について、 A[i] と B[i] には同じパーティション・バウンドが指定されている。

Apcol[i] を A の i 番目のパーティション化列、 Bpcol[i] を B の i 番目のパーティション化列とした場合、さらに次の条件も満たしている必要があります。

- 1 <= i <= Mのすべてのiについて、Apcol[i]とBpcol[i]のデータ型が、長さ、精度、位取りを含めて同じになっている。

A[i] と B[i] は、物理的な属性の点で異なっていても構いません。特に、それらが同じ表領域内に存在する必要はありません。

同一レベル・パーティション化は、データベースの設計時に考慮することが重要です。

- パーティションのメンテナンス操作および表領域の回復操作の際に、休止時間と使えないデータの量を減らす。たとえば、表とその索引に同一レベル・パーティション化を実施すると、パーティションを分けることの影響を、 1 つの表パーティションとそれに対応する索引パーティションに限定できます。ただし、同一レベル・パーティション化を実施されていない索引が表にある場合は、表の 1 つのパーティションを分割するときに索引全体を再編成する必要があります。
- 表間の同一レベル・パーティション化は、実行する必要のある大規模なソートと結合の数を減らすことで実行計画を改善する。（この改善は、 SQL 文の順次実行にのみ適用されます。）
- データの関連サブセットにおける表領域の不完全回復（ Point-in-Time 回復）を容易にする。たとえば、表とその主キー索引、または親表と子表に同一レベル・パーティション化を実施できます。そうしておくと、対応するパーティションを特定の時点まで回復できます。

同一レベル・パーティション化の例

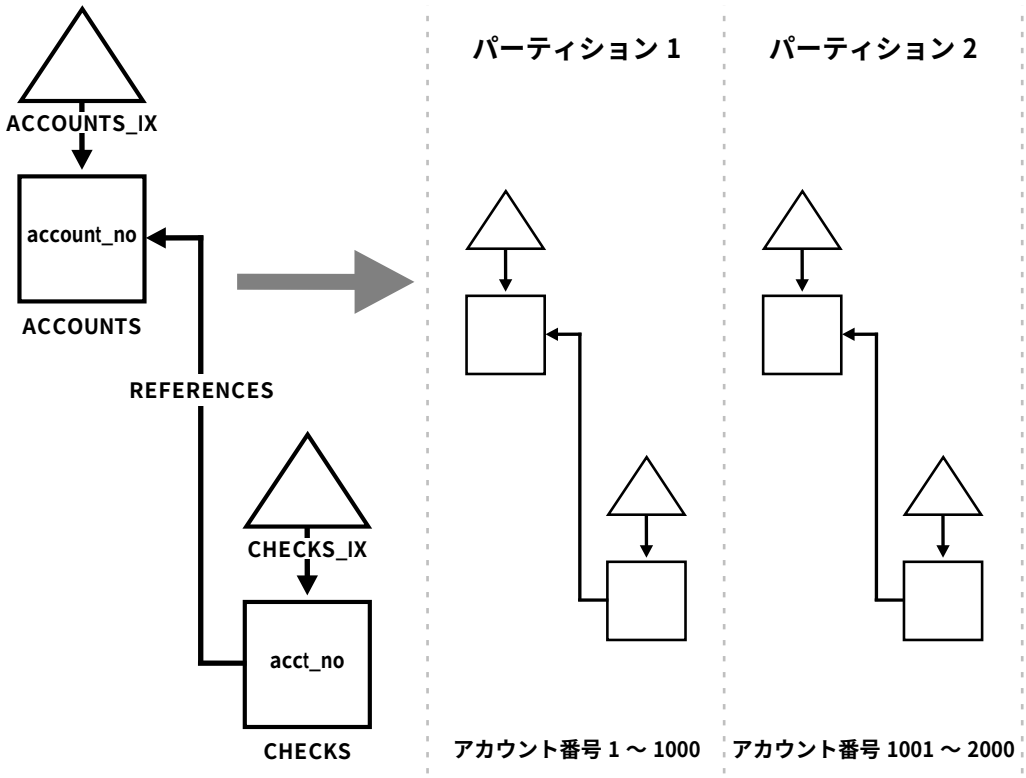
図 9-3は、同一レベル・パーティション化を実施した4つの論理的に関連のあるスキーマ・オブジェクトを示しています。

- ACCOUNTS には、列 ACCOUNT_NO に基づいてレンジ・パーティション化を実施し、パーティションが 2 つあります。最初のパーティションには、アカウント番号 1000 までが入ります。 2 番目のパーティションには、アカウント番号 2000 までが入ります。

- ACCOUNTS_IX は、 ACCOUNTS 表の列 ACCOUNT_NO に対する索引です。表と同様、索引にも ACCOUNT_NO に基づくレンジ・パーティション化を実施し、パーティションが 2 つあります。そして、 ACCOUNTS のパーティションと同じパーティション・バウンドを指定します。
- CHECKSは、列ACCT_NOに基づいてレンジ・パーティション化を実施した、2つのパーティションがある表です。このパーティションには、 ACCOUNTS 表のパーティションと同じパーティション・バウンドを指定します。 ACCT_NO は、 ACCOUNTS にある ACCOUNT_NO を参照する外部キーです。
- CHECKS_IX は、 CHECKS の列（ ACCT_NO, CHECK_NO ）に対する索引です。この索引には、 ACCT_NO に基づいてレンジ・パーティション化を実施し、 2 つのパーティションがあります。 ACCOUNTS のパーティションと同じパーティション・バウンドを指定します。

これら 4 つのスキーマ・オブジェクト間の論理関係を、図 9-3 の左側に示します。物理的なパーティション化については、図の右側に示します。（三角形は索引を、長方形は表を表します。）

図 9-3 同一レベルでパーティション化された表と索引



表および索引をパーティション化するときのルール

この項では、パーティション表とパーティション索引を作るときのルール、そしてパーティションの物理的な属性を説明します。

表のパーティション化

表をパーティション化するときのルールは、次のように簡単なものです。

- 次のような場合に、表にレンジ・パーティション化を実施できる。
  - 表がクラスタの一部になっていない。
  - 表が LOB 、 LONG または LONG RAW データ型、オブジェクト型を含んでいない。

  - 表が索引構成表でない。
- パーティション索引および非パーティション索引を、パーティション表および非パーティション表と混在させることができる。
  - パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。
  - 非パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。（パーティション化されていない表で作成できるのはグローバル索引だけです。 9-23 ページの「グローバル索引」を参照してください。）

表パーティションの物理属性

デフォルトの物理属性は、 CREATE TABLE文によってパーティション表を作る時点でまず指定します。パーティション表そのものに対応するセグメントはないため、この属性はメンバー・パーティションの物理属性を導出するときに限り使われます。デフォルトの物理属性は、 ALTER TABLE MODIFY DEFAULT ATTRIBUTES を使って修正できます。

CREATE TABLEとALTER TABLE ADD PARTITIONによって作成される表パーティションの物理属性は、次のようにして決まります。

- パーティションの属性が指定されていない場合は、実表に（明示的に、またはデフォルトで）指定された対応する物理属性の値が使われる。

既存の表パーティションの物理属性は、 ALTER TABLE MOVE PARTITION および ALTER TABLE MODIFY PARTITIONによって修正できます。その後の属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、文を発行する前のパーティションの属性値が使われる。

> **注意**: ALTER TABLE MOVE PARTITIONは、パーティションが存在する表領域を変更するためにも使えます。

ALTER TABLE SPLIT PARTITIONによって作成される表パーティションの物理属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、分割するパーティションの属性値が使われる。（これは、グローバル索引の分割にも当てはまります。欠落している属性は、分割する索引パーティションから継承されます。）

表のすべてのパーティションの物理属性は、 ALTER TABLE を使って変更できます。たとえば、ALTER TABLE tablename NOLOGGINGは、tablenameのすべてのパーティションのロギング・モードを NOLOGGING に変更します。

索引のパーティション化

索引をパーティション化するときのルールは、表をパーティション化するときのルールと似ています。

- 次の条件を満たしていれば、索引にレンジ・パーティション化を実施できる。
  - クラスタ索引ではない。
  - クラスタ化表に対して定義された索引ではない。
  - 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なければならない。
- パーティション索引および非パーティション索引を、パーティション表および非パーティション表と混在させることができる。
  - パーティション表には、パーティション索引または非パーティション索引（あるいはその両方）を含めることができる。
  - 非パーティション表には、パーティション索引または非パーティション B* ツリー索引（あるいはその両方）を含めることができる。
  - 非パーティション表に対するビットマップ索引は、レンジ・パーティション化できない。

しかし、パーティション索引はパーティション表よりも複雑で、レンジ・パーティション索引にはローカル同一キー索引、ローカル非同一キー索引、グローバル同一キー索引、グローバル非同一キー索引の 4 種類があります。これらの種類については、この後説明します。Oracle は、この 4 種類のうち 3 種類をサポートしています（グローバル非同一索引は、実際のアプリケーションでは役に立ちません）。

ローカル索引

「ローカル索引」では、特定の索引パーティションにあるすべてのキーが、基礎を形成する 1 つの表パーティションに格納された行だけを参照します。ローカル索引は、 LOCAL 属性を指定することにより作成されます。

Oracleは、ローカル索引とその基礎になる表が同一レベルでパーティション化されるように、ローカル索引を構築します。 Oracle は、基礎になる表と同じ列に基づいて索引にレンジ・パーティション化を実施し、基礎になる表と同数のパーティションを作成し、対応するパーティションと同じパーティション・バウンドを指定します。また、基礎になる表のパーティションを追加、削除、分割しても、索引のパーティション化は Oracle により自動的に維持されます。これによって、索引と表は同一レベルでパーティション化された状態を保ちます。

表と索引を同一レベルでパーティション化することには、次のような利点があります。

- 基礎になる表パーティションに対してメンテナンス操作を実行するときに、 1 つの索引パーティションだけが影響を受ける。
  - パーティション表の索引がすべてローカルであれば、パーティションのメンテナンス操作の所要時間は、パーティション・サイズに比例した時間で済む。
  - ローカル索引はパーティションの独立をサポートしている。

– ローカル索引は、履歴データの表における古いデータのロールアウトと、新しいデータのロールインをスムーズにサポートする。

- Oracle は、ローカル索引と基礎になる表が同一レベルでパーティション化されているという状況を活用して、より効率的な問合せアクセス計画を生成できる。
- ローカル索引により、表領域の不完全回復の作業を簡略化できる。表のパーティションをある特定の時点まで回復するには、対応する索引エントリも同じ時点まで回復する必要があります。このことを実現するための唯一の方法は、ローカル索引を使うことです。これを使えば、対応する表と索引パーティションを同時に回復できます。

ローカル同一キー索引

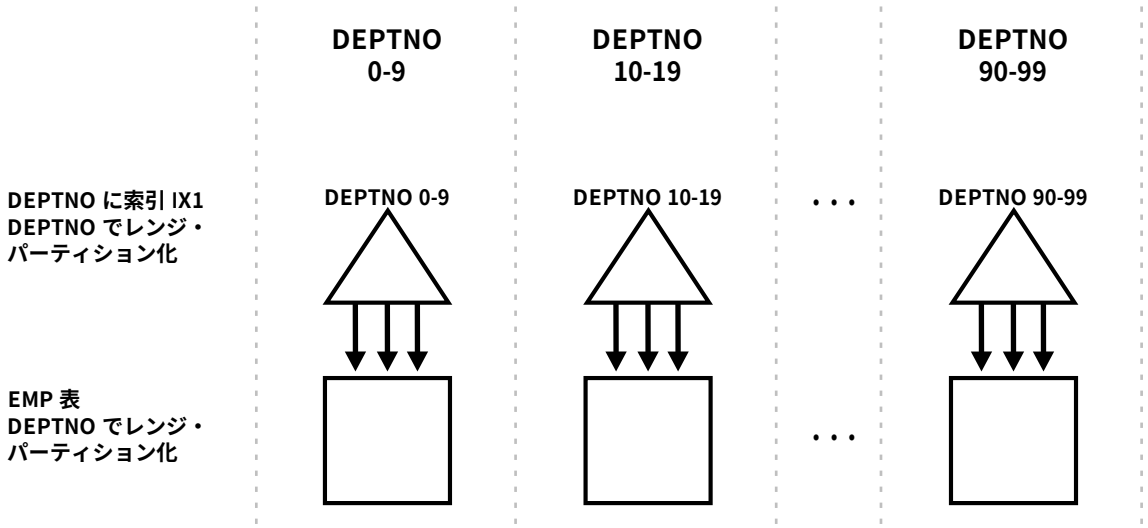
索引列の左側のキー（prefix）に基づいてパーティション化されている場合、その索引はローカル「同一キー」(local prefixed) 索引になります。

たとえば、sales表とそのローカル索引sales_ixがweek_num列に基づいてパーティション化されている場合、索引sales_ixが列(week_num,xaction_num)に対して定義されているのであれば、その索引は「ローカル同一キー」索引です。他方、索引 sales_ix が列 product_num に対して定義されているのであれば、それは同一キー索引ではありません。

図 9-4 では、ローカル同一キー索引の別の例を示します。

ローカル同一キー索引は、一意の索引にも、一意でない索引にもできます。

図 9-4 ローカル同一キー索引



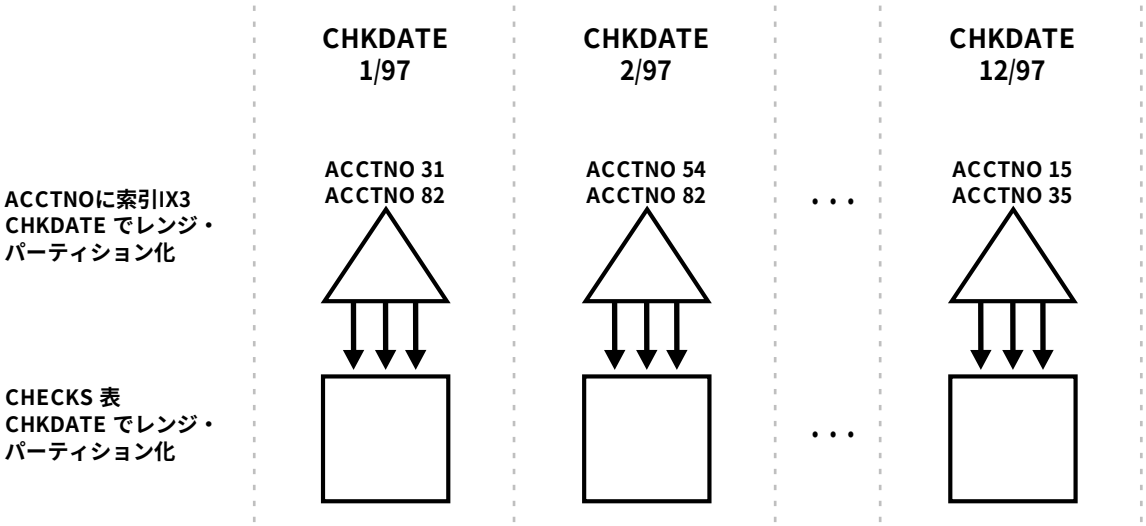
ローカル非同一キー索引

索引列の左側のキー（prefix）以外の列に基づいてパーティション化されている場合、その索引はローカル「非同一キー」（ local non-prefixed ）索引になります。

一意のローカル非同一キー索引を作成するには、索引キーがパーティション化キーのサブセットでなければなりません。

図 9-5 に、ローカル非同一キー索引の別の例を示します。

図 9-5 ローカル非同一キー索引



グローバル索引

「グローバル索引」では、特定の索引パーティションにあるすべてのキーが、基礎になる複数の表パーティションに格納されている行を参照できます。グローバル索引は、 GLOBAL 属性を指定する（これがデフォルト）ことにより作成されます。作成時にグローバル索引の初期のパーティション化を定義する作業と、時間の経過の中でパーティション化を維持していく作業は、ユーザーの責任で行わなければなりません。索引パーティションは、必要に応じて削除したり分割したりできます。

通常、グローバル索引は、基礎になる表と同一レベルでパーティション化されることはありません。グローバル索引を基礎になる表と同一レベルでパーティション化することを妨げる理由は何もありませんが、 Oracle は、問合せ計画の生成時やパーティションのメンテナンス操作の実行時に同一レベル・パーティション化の利点を活用しません。それで、基礎になる表と同一レベルでパーティション化する索引は、 LOCAL として作成するべきです。

グローバル索引には、すべてのパーティション内のすべての行のエントリを持つ1つの B* ツリーが（概念的に）含まれています。各索引パーティションには、表内の多くの異なったパーティションを参照するキーが入っています。

グローバル索引の最も高位のパーティションのパーティション・バウンドには、すべての値に MAXVALUE を指定しなければなりません。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。

索引列の左側のキーでパーティション化されている場合、そのグローバル索引はグローバル「同一キー」索引です。（図 9-6 に例を示します。）索引列の左側のキー以外の列でパーティション化されている場合、そのグローバル索引はグローバル「非同一キー」索引になります。Oracle は、グローバルな非同一キー索引をサポートしていません。

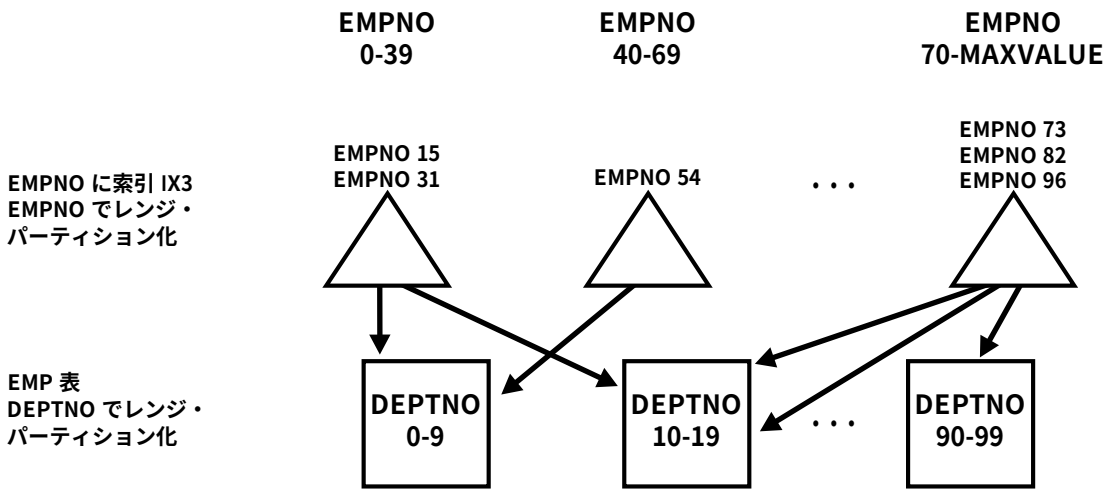
グローバル同一キー索引は、一意の索引にも、一意でない索引にもできます。

グローバル索引は、ローカル索引よりも次の点で管理が煩雑です。

- 基礎になる表パーティションのデータを移動したり削除したりすると（ SPLIT 、 MOVE 、 DROP 、または TRUNCATE ）、グローバル索引のすべてのパーティションが影響を受けます。その結果、グローバル索引では、パーティションのメンテナンス（グローバル索引や索引パーティションの再作成など）にパーティション・サイズではなく表サイズに比例した時間がかかり、パーティションの独立はサポートされません。
- 基礎になる表パーティションをある特定の時点まで回復する場合、グローバル索引内の対応するエントリもすべて同じ時点まで回復しなければなりません。このエントリは索引のすべてのパーティションにわたって散らばっている（回復の対象ではない別のパーティションのエントリと混在している）ことがあるため、グローバル索引全体を作り直すしか方法がありません。

非パーティション索引は、グローバル同一キー索引として処理されます。

図 9-6 グローバル同一キー索引



パーティション索引の種類のまとめ

表 9-1 に、 Oracle がサポートする 3 種類のパーティション索引をまとめます。

- 索引が「ローカル」であれば、基礎になる表と同一レベルでパーティション化されます。それ以外の場合、索引は「グローバル」です。
- 「同一キー」索引は、索引列の左側のキーに基づいてパーティション化されます。それ以外の場合、索引は「非同一キー」索引です。

表 9-1 パーティション索引の種類

索引の種類	索引が表と同一レベルでパーティション化される	索引列の左側のキーに基づいて索引がパーティション化される	UNIQUE 属性を指定できる	例		
				表のパーティション化列	索引列	索引のパーティション化列
ローカル同一キー	はい	はい	はい	A	A 、 B	A
ローカル非同一キー	はい	いいえ	はい [1]	A	B	A
グローバルな同一キー	いいえ [2]	はい	はい	A	B	B
グローバル非同一キー [3]	—	—	—	—	—	—

[1] 一意のローカル非同一キー索引を作成するには、索引キーがパーティション化キーのサブセットでなければなりません。

[2] グローバル・パーティション索引は基礎になる表と同一レベルでパーティション化できますが、DROPやSPLIT PARTITIONなどのパーティション・メンテナンス操作を実行した後では、パーティション化の利点を活用したり、同一レベル・パーティション化を維持したりすることはできません。

[3] この種類の索引はサポートされていません。

非同一キー索引の重要性

非同一キー索引は、履歴データベースの場合に特に便利です。履歴データを含む表では、高速にアクセスする必要をサポートするために1つの列に対して索引を定義するのが普通ですが、古いデータをロールアウトして新しいデータにロールインする期間をサポートするために別の列（基礎になる表と同じ列）に基づいて索引をパーティション化します。

図 9-1 （9-3ページの「週ごとにパーティション化したSALES表」）にあるsales表を例にとって考えてみましょう。この表には、1年分の売上データが、13個のパーティションに分けて記録されています。week_noに基づいてレンジ・パーティション化が実施されているので、1つのパーティションに4週間分のデータが入ります。ユーザーは、sales上にローカル非同一索引キーsales_ixを作ります。アカウント番号によってデータに高速にアクセスする必要のある問合せがあるため、sales_ix索引はacct_noに対して定義します。しかし、sales表と一致させるため、パーティション化はweek_noに基づいて実施します。4週ごとにsalesとsales_ixのパーティションは古い順に削除され、新しいパーティションが追加されます。

同一キー索引と非同一キー索引のパフォーマンスへの影響

同一キー索引を走査するよりも、非同一キー索引を走査する方がコストが高くなります。

索引キーが同一（ローカルまたはグローバル）であり、Oracleに索引列を含む述語が提示された場合、パーティションの切詰めによってその述語の適用を索引パーティションのサブセットに限定できます。

たとえば、図 9-4 （9-22ページの「ローカル同一キー索引」）の場合、述語がDEPTNO=15であれば、オプティマイザはその述語がその索引の2番目のパーティションだけに当てはまることを認識します。（述語にバインド変数が含まれる場合、オプティマイザは正確にどのパーティションであるかは認識できませんが、関係するパーティションが1つだけであることは認識できます。その場合、実行時には1つの索引パーティションだけがアクセスされます。）

索引が非同一キー索引である場合、Oracleは索引列を含む述語をN個すべての索引パーティションに適用しなければならないことがよくあります。これには、1つのキーを調べるか、索引範囲の走査を実行する必要があります。範囲を指定して走査する場合、OracleはN個の索引パーティションからの情報を組み合せる処理も実行しなければなりません。たとえば、図 9-5 （9-23ページの「ローカル非同一キー索引」）の場合、ローカル索引はACCTNOの索引キーによってCHKDATEにパーティション化されています。述語がACCTNO=31であれば、Oracleは12個の索引パーティションすべてを調査します。

もちろん、パーティション化列に対する述語も含まれている場合には、複数の索引を調査する必要はなくなるかもしれません。Oracleは、ローカル索引と基礎になる表が同一レベルでパーティション化されているという状況を活用して、パーティション・キーに基づいてパーティションを切り詰めます。たとえば、図 9-5の述語がCHKDATE<3/97であれば、Oracleが調査する必要のあるパーティションは2つだけです。

非同一キー索引の場合、パーティション・キーが WHERE 句の一部である（ただし索引キーの一部ではない）場合、オプティマイザは、基礎になる表パーティションに基づいて、どの索引パーティションを調査するかを判断します。

ローカルな非同一キー索引のキーを使った多くの問合せやDML文がすべての索引パーティションを調査しなければならない場合、そのような索引によって提供されるパーティションの独立の程度は低いものです。

索引のパーティション化のためのガイドライン

表の索引をどのようにパーティション化するかを決定するときには、その表をアクセスする必要があるいくつかのアプリケーションの組合せを考慮に入れなければなりません。パフォーマンスという面と、可用性と管理しやすさという面の両面を比較検討する必要があります。

考慮するべきガイドラインは次のとおりです。

- OLTP アプリケーションの場合 :
  - グローバル索引とローカル同一キー索引を使うと、索引パーティション調査の数が最小になるため、ローカル非同一キー索引を使う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のメンテナンス操作を実行するときに、より高い可用性をサポートします。ローカル非同一キー索引は、履歴データベースの場合に非常に便利です。
- DSS アプリケーションの場合は、ローカル非同一キー索引を使うと、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走査できるため、パフォーマンスを改善できます。

  たとえば、図 9-5 （ 9-23 ページの「ローカル非同一キー索引」）の表 CHECKS に対して述語「ACCTNO between 40 and 45」を使った問合せを実行すると、非同一キー索引IX3のすべてのパーティションがパラレルで走査されます。一方、図 9-4（9-22ページの「ローカル同一キー索引」）の表DEPTNOに対して述語「DEPTNO between 40 and 45」を使った問合せは、同一キー索引 IX1 の 1 つのパーティションにしかアクセスしないため、パラレル化できません。
- 履歴データ表の場合、索引はできるだけローカルにします。これにより、通常の定期的なパーティションの削除操作の影響を小さくできます。
- パーティション化列でない列の UNIQUE 索引は、グローバルでなければなりません。キーにパーティション化キーが含まれていない一意のローカル非同一キー索引はサポートされていないからです。

索引パーティションの物理属性

デフォルトの物理属性は、 CREATE INDEX 文によってパーティション索引を作る時点で初めに指定します。パーティション索引そのものに対応するセグメントはないため、この属性はメンバー・パーティションの物理属性を導出するときに限り使われます。デフォルトの物理属性は、 ALTER INDEX を使って後で修正できます。

CREATE INDEX によって作成されるパーティションの物理属性は、次のようにして決まります。

- パーティションの属性が指定されていない場合は、索引に（明示的に、またはデフォルトで）指定された対応する物理属性の値が使われる。 LOCAL 索引のパーティションの TABLESPACE 属性を扱うときには、このルールに関して重要な例外があります。つまり、ユーザーが TABLESPACE 値を指定しなかった場合には、基礎になる表の対応するパーティションの値が使われます。

ALTER TABLE ADD PARTITION の処理過程で作成されたローカル索引のパーティションの物理属性（ TABLESPACE を除く）には、各索引のデフォルトの物理属性が設定されます。

ALTER TABLE SPLIT PARTITION によって作成される索引パーティションの物理属性（ TABLESPACE を除く）は、次のようにして決まります。

- 分割する索引パーティションの物理属性（ TABLESPACE を除く）が使われる。

既存の索引パーティションの物理属性は、ALTER INDEX MODIFY PARTITIONおよびALTER INDEX REBUILD PARTITION によって修正できます。その後の属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、文を発行する前のパーティションの属性値が使われる。ALTER INDEX REBUILD PARTITIONは、パーティションが存在する表領域を変更するためにも使えます。

ALTER INDEX SPLIT PARTITIONによって作成されるグローバル索引パーティションの物理属性は、次のようにして決まります。

- 新しい値を指定しなかった物理属性については、分割するパーティションの属性値が使われる。

表のすべてのパーティションの物理属性は、 ALTER INDEX を使って変更できます。たとえば、 ALTER INDEX indexname NOLOGGING は、 indexname のすべてのパーティションのロギング・モードを NOLOGGING に変更します。

DML パーティション・ロック

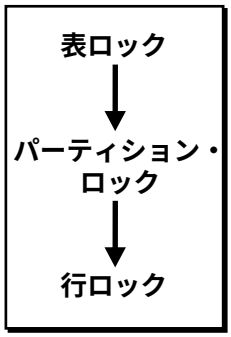
DML 表ロックは、 DML 文（ INSERT および UPDATE 、 DELETE ）と、 DDL 文および LOCK TABLE 文とを同期化します。さらに、 DML 表ロックは、 DDL 文および LOCK TABLE 文どうしも同期化します。

DDL 操作とユーティリティ操作で「パーティションを独立させる」ために、Oracleは「DML パーティション・ロック」をサポートしています。パーティションを独立させることにより、別のパーティション上のアクティビティを休止させずに、選択したパーティションに対して DDL 操作とユーティリティ操作を実行できます。

パーティション・ロックの目的は、個々のパーティション内のデータを保護すると同時に、複数のユーザーが表内のパーティションに同時にアクセスできるようにすることです。

図 9-7 に示されているとおり、 DML ロックの階層においてパーティション・ロックは表ロックと行ロックの中間に位置します。

図 9-7 DML **ロックの階層**



パーティション・ロックは、表ロックと同じモードで取得できます。共有（S)、排他（X)、行共有（ SS ）、行排他（ SX ）、そして行共有排他（ SSX ）モードがあります。

Oracle Parallel Server のパフォーマンスの考慮事項

DML ロックの余分なレベルを実行すると、余分のメッセージが分散ロック・マネージャに送られるので、Oracle Parallel Server環境において短いトランザクションのパフォーマンスが影響を受けることがあります。

Oracle Parallel Server環境でパフォーマンスを改善するために、ALTER TABLE DISABLE TABLE LOCK 文を使って、選択した表について DML ロックをオフにできます。これにより、表およびパーティションの DML ロックがどちらも使用禁止になります。 DML ロックが使用禁止のときには、 DDL 文は使えません。

追加情報:『Oracle8 Parallel Server 概要および管理』を参照してください。

メンテナンス操作

この項では、次の内容を取り上げます。

- パーティションのメンテナンス操作
- 索引の管理
- パーティション表およびパーティション索引についての権限
- パーティション表およびパーティション索引についての監査

この章の中で、「メンテナンス操作」とは、表や索引の定義を変更したり、大量のデータのロードやアンロードを実行したりする DDL 文またはユーティリティ（ Export 、 Import 、 SQL*Loader など）のことです。

非パーティション表と非パーティション索引に対する既存のメンテナンス操作は、すべてパーティション表とパーティション索引に対して実行できます。たとえば、 DROP TABLE でパーティション表を削除したり、 Export でパーティション表をエクスポートしたりできます。しかし、メンテナンス操作の中には、パーティション表またはパーティション索引全体に対してではなく、個々のパーティションに対して実行しなければならないものもあります。たとえば、ALTER TABLE ALLOCATE EXTENTは、パーティション表に対して実行できません。そのため、パーティションに新規エクステントを追加するときは、そのパーティションに対して ALTER TABLE MODIFY PARTITION ALLOCATE EXTENT を使います。

メンテナンス操作は、所要時間が操作対象のオブジェクトのサイズ（レコード数）の影響を受けなければ、高速であるとみなされます。高速なメンテナンス操作は、ディクショナリとセグメント・ヘッダーを変更するだけで、データを走査したり更新したりはしません。その操作は、短時間（数秒）で完了することが期待されています。たとえば、RENAMEは高速な操作ですが、 CREATE INDEX は高速な操作ではありません。

パーティションのメンテナンス操作

「パーティション・メンテナンス操作」は、パーティション表またはパーティション索引のパーティションの 1 つを修正します。たとえば、既存の表に新しいパーティションを追加したり、I/O負荷のバランスを改善するためにパーティションを別の表領域に移動したり、パーティションをロードしたりする操作があります。

パーティションのメンテナンス操作の中には、計画されたイベントであるものもあります。たとえば、履歴データベースでは、データベース管理者（DBA）はデータベースから最も古いパーティションを定期的に削除し、一連の新しいパーティションを追加します。この削除操作と追加操作は、定期的な予定に基づいて実行されます。 計画されたメンテナンス操作の別の例は、データを再クラスタ化して断片化を減らす目的で行う、定期的なエクスポート / インポートです。

パーティションのその他のメンテナンス操作は、計画されていないイベントであり、アプリケーションやシステムの問題から回復するために必要になるものです。たとえば、予期していなかったトランザクション・アクティビティが発生すると、 DBA はもう一度 I/O 負荷のバランスをとるためにパーティションを分割したり、 1 つ以上のパーティションを再作成したりする必要が生じるかもしれません。

パーティションのメンテナンス操作には、次のような操作があります。

- パーティションを既存の表に追加する。
- パーティションの修正 ― パーティションの物理属性を変更する。
- パーティションの移動 ― パーティションを、別の表領域に移動したり、再クラスタ化したり、パラメータ（作成時のパラメータを含む）を変更したりする。
- パーティションを改名する。
- パーティションを削除する。
- 表パーティションを切り捨てる（領域の再生を要求する、または要求しない）。
- 既存のパーティションを 2 つのパーティションに分ける。

- 1 つの表パーティションにデータをロードする。
- 1 つの表パーティションからデータをエクスポートする。
- 表パーティションをインポートする。
- 索引パーティションを再作成する。

メンテナンス操作の同時実行モデル

この項で説明する同時実行モデルは、複数の DDL 操作とユーティリティ操作を同じスキーマ・オブジェクトに対して同時に実行するための条件を定義します。また、どの問合せおよび DML 操作を DDL 操作やユーティリティ操作と同時に実行できるかについても定義しています。

このモデルは、すべての DDL 文に適用されます。また、 SQL*Loader のようなユーティリティにも適用されます。

1 ステップ操作と 3 ステップ操作

メンテナンス操作には、 1 ステップ操作と 3 ステップ操作の 2 種類があります。

1 ステップ操作は、次のような操作です。

- この操作は、関係する表を排他（ X ）モードで DML ロックする。索引操作は、基礎になる表をロックする。さらに、操作の実行中は、排他ディクショナリ・ロックが保持される。
- この操作には、高速で実行されるもの（たとえば、ALTER TABLE ADD PARTITION）と、他の操作を同時に実行できないもの（たとえば、 ALTER TABLE ADD column）がある。
- CREATE INDEXおよびALTER INDEX REBUILD（また、削除または分割しているグローバル・パーティションが USABLE な場合の、 ALTER INDEX DROP/SPLIT PARTITION ）を除く、すべての索引操作は 1 ステップ操作である。
- CREATE INDEX 、および MOVE 、 SPLIT 、 REBUILD 、または EXCHANGE PARTITION を除く、すべての Oracle DDL 文は 1 ステップ操作である。

3 ステップ操作は、次のような操作です。

- この操作は、関係する表に対して制限の弱い DML ロックを取得する。排他（ X ）モードで 1 つのパーティションだけをロックするか、表全体をロックする場合でも、 S モード、 SS モード、 SX モードのどれかでロックします。
- この操作には次の 3 つのステップがある。
  - ステップ 1: 共有ディクショナリ・ロックを保持した状態で、ディクショナリを読み込む。ステップ 1 は、短時間（数秒）で実行されます。このステップの終了時に、適切な DML ロックが取得され、ディクショナリ・ロックが解放されます。
  - ステップ 2: 表または索引レコードを走査あるいは更新する。ステップ 2 には、長い時間がかかることがあります（数分、数時間）。

- ステップ3: 排他ディクショナリ・ロックを保持した状態で、ディクショナリを更新する。ステップ 3 は、短時間（数秒）で実行されます。

- この操作は長時間実行されるが、その他の操作を同時に実行できる。実際にどの操作を同時に実行できるかは、次に説明するとおり、文によって取得される特定のDML ロックによって異なります。
- 次の操作は 3 ステップ操作である。
  - ALTER TABLE MOVE PARTITION、ALTER TABLE SPLIT PARTITION、ALTER TABLE EXCHANGE PARTITION 、表パーティションのダイレクト・パス・ロード。これらの文は、実表を行排他（SX）モードでロックし、パーティションを排他（X）モードでロックします。
  - CREATE INDEX および ALTER INDEX REBUILD PARTITION （グローバル索引の場合)。これらの文は、基礎になる表を共有（ S ）モードでロックします。
  - ALTER INDEX REBUILD PARTITION（ローカル索引の場合)。この文は、基礎になる表を行共有（SS）モードでロックし、基礎になる表パーティションを共有（S）モードでロックします。
  - ALTER TABLE MODIFY PARTITION REBUILD UNUSABLE LOCAL INDEXES 。この文は、基礎になる表を行共有（SS）モードでロックし、基礎になる表パーティションを共有（ S ）モードでロックします。

最後に、操作によっては、状況に応じて 1 ステップ・プロトコルと 3 ステップ・プロトコルのどちらかに従うものがあります。

- ALTER TABLE DROP PARTITION,ALTER TABLE TRUNCATE PARTITION

  変更する表にグローバル索引が定義されていない場合、または使用可能（ ENABLED ）参照制約によって表を参照する場合、このグループの文は 1 ステップ・プロトコルを使って実行されるため、高速になります。それ以外の場合は、 3 ステップ・プロトコルを使って実行されます。後者の場合、実表は行排他（SX）モードでロックされ、パーティションは排他（ X ）モードでロックされます。

- ALTER INDEX SPLIT PARTITION （グローバル索引の場合のみ)。

  分割するパーティションが USABLE であれば、文は 3 ステップ・プロトコルで実行され、 SPLIT の結果として作成されるパーティションは USABLE になります。他方、分割するパーティションが UNUSABLE であれば、操作は 1 ステップ・プロトコルで実行され、その結果として作成されるパーティションも UNUSABLE になります。

- ALTER INDEX DROP PARTITION （グローバル索引の場合のみ)。

  削除するパーティションが USABLE であれば、文は 3 ステップ・プロトコルで実行されます。それ以外の場合は、 1 ステップ・プロトコルで実行されます。

従来型パスの SQL*Loader と Import では SQL INSERT が使われるため、このモデルにおいては DML 操作として分類されます。 Export では SQL SELECT が使われるため、問合せ操作として分類されます。

同時に実行できる操作

この項のルールは、 1 ステップ操作と 3 ステップ操作の定義から導き出せるものです。

1 ステップ操作が進行中の場合 :

- 表に対する問合せを実行できる。
- その他の操作（ DDL 、ユーティリティ、または DML ）は実行できない。

問合せ（ READ 操作）には DML ロックが必要ないため、 SPLIT または MOVE するパーティションに対する問合せは、 SPLIT や MOVE の処理中でも許可されます。しかし、操作の終了時に現行のセグメントが削除され、その領域が再使用されます。領域が再使用されると、エラーが通知されます。

ALTER TABLE MOVE PARTITION 、 ALTER TABLE SPLIT PARTITION 、 ALTER TABLE EXCHANGE PARTITION 、または表パーティションのダイレクト・パス・ロードがパーティションに対して進行中の場合 :

- 同じ表内の別のパーティションを移動または分割、ダイレクト・パス・ロードできる。
- 表に対する問合せを実行できる。
- パーティションに書込みを実行しない DML 操作であれば、表に対する DML 操作を実行できる。
- パーティションに対するローカル索引パーティション以外のローカル索引パーティションであれば、再作成できる。
- 上記以外のメンテナンス操作を表やその索引に対して実行することはできない。

USABLE パーティションに対する CREATE INDEX 、 ALTER INDEX REBUILD PARTITION 、または ALTER INDEX DROP/SPLIT PARTITION （グローバル索引の場合）が進行中の場合 :

- 基礎になる表に対する問合せを実行できる。
- 表に対して他の索引を作成したり、既存の索引にパーティションを再作成したり、既存の索引内の USABLE パーティションを削除 / 分割したりできる。
- 表に対する DML 操作は実行できない。また、上記以外のメンテナンス操作を表やその索引に対して実行することはできない。

ALTER INDEX REBUILD PARTITION （ローカル索引の場合）が、基礎になる表パーティションに対応するパーティション上で進行中の場合 :

- 基礎になる表のパーティション以外のパーティションを移動または分割、ダイレクト・パス・ロードできる。
- 表に対する問合せを実行できる。
- 基礎になる表パーティションに書込みを実行しない DML 操作であれば、表に対する DML 操作を実行できる。

- 索引内の他のパーティションを再作成できる。また、表に対して別の索引を作成したり、別の索引内のパーティションを再作成したりできる。
- 上記以外のメンテナンス操作を表やその索引に対して実行することはできない。

表のパーティションに対するメンテナンス操作の中には、表パーティションまたは索引パーティションのグローバル索引を UNUSABLE にするものもあります。 1 つの例は、 ALTER TABLE MOVE PARTITION です。 DBA は、パーティションのメンテナンス操作に加えて、グローバル索引の再作成操作を含むスクリプトを実行する必要があります。結果的にユーザー側から見ると、この操作は表全体へのシリアル・アクセスになります。 ALTER TABLE MOVE/SPLIT PARTITION のような操作は、パーティション・グローバル索引のすべてのパーティションだけではなく、非パーティション・グローバル索引も UNUSABLE にします。

グローバル索引のすべてのパーティションをマークする表パーティション操作は、ローカル索引のうち1つのパーティション（操作される表パーティションに対応するパーティション）も UNUSABLE としてマークします。

同様に、パーティションのメンテナンス操作によっては、操作の前に参照整合性制約を使用禁止にして、操作後に再び使用可能にしなければならないものがあります。1つの例として、空ではないパーティションの ALTER TABLE DROP PARTITION があります。 DBA は、パーティションのメンテナンス操作に加えて、制約を再び使用可能にするスクリプトを実行する必要があります。結果的にユーザー側から見ると、この操作は表全体へのシリアル・アクセスになります。

問合せとパーティションのメンテナンス操作

パーティションのメンテナンス操作を開始する前、またはパーティションのメンテナンス操作でディクショナリを更新する前に実行を開始した問合せは、一貫性のある読込みによって、問合せのスナップショット時に存在していたデータに正しくアクセスします。ディクショナリの更新が完了した後のそのような問合せの動作は、スナップショット時に存在していたデータが正しく検索される場合とエラーが戻される場合があるという意味で、予測できません。

パーティション索引を使う問合せで、Index Unusableとマークされた索引パーティションのいくつかを使って開始する問合せは、これらのパーティションの 1 つに実際に初めてアクセスするときにエラーを戻します。このことは、問合せの開始後にパーティションが USABLE にされたとしても発生します。

カーソルの無効化

新しい DDL 文の多くがパーティション・ベースであるとしても、カーソルの無効化は表ベースのままです。つまり、 DDL 文が表 T の 1 つのパーティション P にしか影響を与えず、かつカーソルがパーティション P にアクセスしない場合でも、表 T を修正する DDL 文は、表 T に依存しているすべてのカーソルも無効化するということです。

RECOVERABLE 操作と UNRECOVERABLE 操作

すべてのパーティション・メンテナンス操作は、回復可能（ LOGGING ）モードで実行できます。しかし、次に挙げる一部の操作は、 NOLOGGING オプションをサポートします。

- パラレルの CREATE TABLE ... AS SELECT

- CREATE INDEX
- ダイレクト・パスの SQL*Loader
- ダイレクト・ロード・インサート

LOGGING がデフォルトです。しかし、データベースが NOARCHIVELOG モードで 実行されている場合は、 NOLOGGING がデフォルトです。 LOGGING/NOLOGGING オプションをサポートしていない DDL やユーティリティ文は、必ず回復可能モード（ LOGGING ）で実行されます。

注意 : [NO]LOGGING は操作の属性ではなく、物理オブジェクトの属性です。そのため、 INSERT 操作では [NO]LOGGING を指定できません。INSERT に関係する表や索引のロギング・モードを変更するときは、INSERT文を実行する前にALTER TABLE/INDEX [NO]LOGGINGを実行する必要があります。詳細は、 21-5 ページの「ロギング・モード」を参照してください。

索引の管理

ローカル索引またはグローバル索引の改名、物理的な記憶領域の属性の変更、パーティションの再作成は、いつでも実行できます。索引をパーティション化する方法を変更するときには、索引がローカルであるかグローバルであるかに応じて異なった処理が必要です。

ローカル索引

Oracle では、ローカル索引のパーティション化が、基礎になる表のパーティション化と必ず一致することが保証されます。基礎になる表が変更されると、Oracle8は必要に応じて自動的に索引パーティションを作成したり削除することにより、 2 つのパーティション化を一致させます。ローカル索引のパーティションは、明示的に追加、削除、分割することはできません。

ローカル索引ごとに、次のようになります。

- 基礎になる表にパーティションを追加すると、その新しい表パーティションと同じパーティション・バウンドを使って新しい索引パーティションが自動的に作成される。
- 基礎になる表のパーティションを削除すると、対応する索引パーティションが自動的に削除される。
- 基礎になる表でパーティションを分割すると、対応する索引パーティションが自動的に分割される。この 2 つの新しい索引パーティションには、新しい表パーティションと同じパーティション・バウンドがあります。

  親の表パーティションを分割した結果として作成されたローカル索引パーティションは、対応する表パーティションが空でないかぎり、UNUSABLEとマークされますので注意してください。

Oracleが（ADDまたはSPLITにより）新しいローカル索引パーティションを作成する場合、Oracleは次のように操作します。

- 索引パーティションに、対応する表パーティションと同じ名前を割り当てようとする。その名前を割り当てられない場合は、SYS_Pnnnという形式の名前を生成します（9-13ページの「パーティション名」を参照してください）。パーティションの名前は後で改名できます。
- ADD PARTITIONについては、Oracleは基になる索引のデフォルトの物理記憶領域属性を使ってセグメントを作成する。親索引に表領域（DEFAULTを除く。DEFAULT表領域の場合、ローカル索引パーティションは、対応する実表パーティションと同じ位置に配置される）が指定されている場合、索引パーティションはその表領域内に配置されます。親索引に表領域が指定されていない場合は、新しい索引パーティションが存在する表領域が、基礎になる表の対応するパーティションの表領域になります。この属性は後で修正できます。
- SPLIT PARTITIONの場合、分割する索引パーティションの属性が、分割した後の索引パーティションにも使われる。ただし、パーティション名は例外です（Oracleは、可能なときだけそれらの表パーティション名を再使用します）。たとえば、TPという名前の表パーティションとIPという名前のローカル索引がある場合、TPをTPとTP1に分割すると、ローカル索引パーティションの名前はIPとTP1（TP1がすでに使われているなら、システムが生成する別の名前）になります。TPをTP1とTP2に分割すると、ローカル索引パーティションの名前はTP1とTP2になります。つまり、Oracleが表パーティション名を再使用するときは、ローカル索引パーティション名も再使用するということです。なお、その他の属性は、分割する索引パーティションから継承されます。

グローバル索引

グローバル索引のパーティション化を維持する作業は、DBAの責任で行います。DBAは、グローバル索引のパーティションを削除したり分割したりできます。しかし、グローバル索引の最高位のパーティションのパーティション・バウンドは必ずMAXVALUEに設定されているため、グローバル索引にパーティションを追加することはできません。

索引パーティションの再作成

ローカル・パーティション索引またはグローバル・パーティション索引の1つのパーティションを再作成するには、ALTER INDEX REBUILD PARTITION文を使えます。これにより、DROP INDEXを実行してからCREATE INDEXを実行する（この操作は索引内のすべてのパーティションに影響を与える）という必要がなくなります。

ALTER INDEX REBUILD PARTITIONには、4つの重要な役割があります。

- 索引パーティションを再クラスタ化して、領域を回復し、パフォーマンスを改善する。
- 索引パーティションが存在するボリューム上でメディア障害が発生した場合、または索引パーティションを含むセグメントがソフトウェアによって破損された場合に、索引パーティションを修復する。
- ImportまたはSQL*Loaderを使って基礎になる表パーティションをロードした後、ローカル索引パーティションを再生成する。これらのユーティリティは、索引のメンテナンスを回避するためのパフォーマンス・オプションを提供し、関係するパーティションを

Index Unusable としてマークし、後で DBA が再作成できるようになります。(Index Unusable は、次の項で説明します。) つまり、「索引を削除してから索引を再作成する」という方針は、「索引パーティションを UNUSABLE としてマークしてから索引パーティションを再作成する」という方針に変更されています。

- 基礎になる表に対するパーティション・メンテナンス操作で Usable とマークされた索引パーティションを再作成する。

Index Unusable （索引使用禁止状態）属性

一部のメンテナンス操作は、索引をIndex Unusable（IU）とマークします。Index Unusable（索引使用禁止状態）は、非パーティション索引の属性であり、パーティション索引内のパーティションの属性でもあります。索引または索引パーティションが IU とマークされている場合、その索引（またはパーティション）を必要とする SELECT 文または DML 文を実行しようとすると、エラーになります。

1 つの索引パーティションが IU とマークされた場合は、そのパーティションを使う前に、そのパーティションを再作成して再び有効にする必要があります。しかし、 1 つのパーティションが IU とマークされても、索引のその他のパーティションは有効であり、 IU とマークされたパーティションにアクセスしない限り、その索引を必要とする SELECT 文または DML 文を実行できます。

IU パーティションを再作成する前に IU パーティションを分割または改名したり、 GLOBAL 索引の IU パーティションを削除したりできます。

非パーティション索引が IU とマークされる場合は、その索引を削除して再作成できます。また GLOBAL index.x の IU パーティションも削除して再作成できます。非パーティション索引を再作成するときには、 ALTER INDEX REBUILD も使用できます。

次の 6 種類のメンテナンス操作により、索引パーティションが Index Unusable とマークされます。どの場合にも、操作が完了した後、その索引パーティションを再作成しなければなりません。

- ローカル索引のメンテナンスを実行しないことにするオプションがある、パーティションのインポートまたは従来型パスの SQL*Loader などの操作。インポートが完了すると、関係するローカル索引パーティションは IU とマークされます。
- ダイレクト・パスの SQL*Loader が索引指定するデータの索引が期限切れの場合、関係するローカル索引パーティションとグローバル索引は IU 状態のままになる。(索引使用禁止状態（ Index Unusable ）は、以前はダイレクト・ロード状態と呼ばれていました。) 索引は次の 2 つの理由により期限切れになることがあります。
  - 領域管理でエラーが発生したため（たとえば、エクステント不足)、ロード操作で索引をメンテナンスできない。
  - ユーザーが SKIP_INDEX_MAINTENANCE オプションを要求した。
- ROWIDを変更する、ALTER TABLE MOVE PARTITIONなどのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティションとすべてのグローバル索引パーティションを IU とマークします。

- 表から行を削除するALTER TABLE TRUNCATE PARTITIONまたはDROP PARTITIONなどのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティションとすべてのグローバル索引パーティションを IU とマークします。
- ローカル索引のパーティション定義を修正するが、新しい定義に一致するように索引データを自動的に再作成することはしない、 ALTER TABLE SPLIT PARTITION などのパーティション・メンテナンス操作。この種の操作は、関係するローカル索引パーティション（複数のこともある）を IU とマークします。(ALTER TABLE SPLIT PARTITION は、 ROWID に変更を加えるため、すべてのグローバル索引パーティションも IU とマークします。）
- 索引のパーティション化の定義を修正するが、関係するパーティションを自動的に再作成することはしない、ALTER INDEX SPLIT PARTITIONなどの索引メンテナンス操作。この種の操作は、関係する索引パーティション（複数のこともある）をIUとマークします。しかし、グローバル索引の USABLE パーティションを分割すると、その結果として作成されるパーティションも USABLE になります。 IU とマークされたパーティションを分割すると、その結果として作成されるパーティションも IU になります。(IU であるか、空でないグローバル索引のパーティションを削除すると、索引の次のパーティションは IU になりますので注意してください。）

パーティション表およびパーティション索引についての権限

パーティションについての権限は、個々のパーティションごとにではなく、親表または親索引に対して付与されます。

ユーザーまたはロールが、非パーティションや非パーティション索引に対して Oracle の操作を実行するのに必要な権限（必要なリソース権限を含む）を持っている場合、その同じ Oracle の操作はパーティション表やパーティション索引に対しても実行できます。たとえば、次のとおりです。

- 非パーティション表を作成できる場合は、パーティション表も作成できる。
- 非パーティション索引を削除できる場合は、パーティション索引も削除できる。
- ALTER を使って非パーティション表に列を追加できる場合は、 ALTER を使ってパーティション表にも列を追加できる。

ユーザーまたはロールが表または索引に対してALTER操作を実行するための権限を持っている場合、表または索引のパーティションに対して、新しいALTER操作を起動できます。ただし、次の場合を除きます。

- 次の操作を実行するとき、表所有者ではないユーザーは、DROP ANY TABLE権限を必要とします（ ALTER 権限に加えて）。
  - ALTER TABLE DROP PARTITION
  - ALTER TABLE TRUNCATE PARTITION
- 次の操作の場合、 ALTER 権限に加えて、領域を取得しようとしている表領域での領域割当て制限が必要です。

- ALTER INDEX MODIFY PARTITION
- ALTER INDEX REBUILD PARTITION
- ALTER INDEX SPLIT PARTITION
- ALTER TABLE ADD PARTITION
- ALTER TABLE MODIFY PARTITION
- ALTER TABLE MOVE PARTITION
- ALTER TABLE SPLIT PARTITION

パーティション表およびパーティション索引についての監査

すべてのAll ALTER TABLE PARTITION操作は、 ALTER TABLE操作と同様に監査されます。パーティションに対しては、新しい監査属性は使われません。

SQL の拡張： 拡張パーティション表名

パーティション・レベルの一括操作は、特定のパーティションの行に限定されます。たとえば、すべてのグローバル索引をUNUSABLEにせずにパーティションを削除するユーザーは、その特定のパーティションにあるすべての行を削除することが必要な場合があります。

そのような操作は、拡張パーティション表名の構文を使うと、自然に表現できます。WHERE 句述語を使ってその同じ操作を表現しようとすると、特にレンジ・パーティション化キーが複数の列を使っているときなどには、非常に厄介になります。

次のDML文の表指定構文には、非リモート・パーティション表を指定するためのパーティション・オプションが含まれています。

- INSERT
- UPDATE
- DELETE
- LOCK TABLE
- SELECT

たとえば、次のとおりです。

```
SELECT * FROM schema.table PARTITION part_name;
```

この構文は、個々のパーティションを表として見るための簡単な方法になります。拡張パーティション表名を使うと、 1つのパーティションだけから選択するビューを作成することができ、そのビューを表のかわりに使うことができます。

そのようなビューを使うと、このビューについての権限を他のユーザーやロールに付与したり取り消したりすることにより、パーティション・レベルのアクセス制御メカニズムを作成することもできます。アプリケーションの移植性と ANSI 構文への準拠を考慮に入れるならば、ビューを使って、 Oracle のこの独自の拡張機能からアプリケーションを隔離するようお薦めします。

拡張パーティション表名の使用には、次のような制限があります。

1. 拡張パーティション表名はリモート・スキーマ・オブジェクトを参照できない。

   拡張パーティション表名には、 dblink や、変換すると dblink 付きの表になるシノニムを含めることができません。リモート・パーティションを使う必要がある場合は、拡張パーティション表名構文を使うビューをリモート・サイトで作成し、そのリモート・ビューを参照できます。

2. 拡張パーティション表名構文は、 PL/SQL ではサポートされていない。

   拡張パーティション表名構文を使用した SQL 文は、 DBMS_SQL パッケージの動的 SQL では使えますが、 PL/SQL ブロックでは使えません。また、 PL/SQL ブロック内のパーティションを参照する必要がある場合は、そうするかわりに、拡張パーティション表名構文を使うビューを使用できます。

3. 実表にだけ指定できる。

   パーティションの拡張は、実表に指定しなければなりません。シノニム、ビュー、その他のスキーマ・オブジェクトには指定できません。

拡張パーティション表名の例

次の文には、有効な拡張パーティション表名が含まれています。

```
SELECT * FROM sales PARTITION (nov95) s
  WHERE s.amount_of_sale > 1000;

UPDATE sales PARTITION (feb96) s
  SET s.account_name = UPPER(s.account_name);

DELETE FROM sales PARTITION (nov95)
  WHERE amount_of_sale != 0;

INSERT INTO sales PARTITION (oct95)
  SELECT * FROM lastest_data;

INSERT INTO sales PARTITION (oct95) VALUES (...);

INSERT INTO sales PARTITION (oct95) (acct_no, ..., week_no) VALUES (...);
```

```
LOCK TABLE sales PARTITION (jun95) IN EXCLUSIVE MODE;

CREATE VIEW sales_feb96 AS
  SELECT * FROM sales PARTITION (feb96);
```

10

ビルトイン・データ型

I am the voice of today, the herald of tomorrow. ... I am the leadenarmy that conquers the world ― I am TYPE.

Frederic William Goudy: *The Type Speaks*

この章では、 Oracle ビルトイン・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。トピックは次のとおりです。

- Oracle データ型
  - 文字データ型
  - NUMBER データ型
  - DATE データ型
  - LOB データ型
  - RAW および LONG RAW データ型
  - ROWID データ型
  - MLSLABEL データ型
- ANSI データ型、 DB2 データ型、 SQL/DS データ型
- データ変換

Oracle データ型

列定義では、次のビルトイン・データ型を使えます。

- 文字データ型
  - CHAR データ型
  - VARCHAR2 データ型
  - 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 データ型
- MLSLABEL データ型

追加情報: PL/SQL には、その他のデータ型として、 BOOLEAN、参照型、複合型（コレクションおよびレコード）、そしてユーザー定義サブタイプがあります。 PL/SQL のデータ型の詳細は、『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。

文字データ型

文字データ型は、文字コード体系（通常、キャラクタ・セットまたはコード・ページと呼ばれる）に対応するバイト値によって、文字（英数字）データを文字列に格納します。

データベースのキャラクタ・セットは、データベースの作成時に設定され、その後は変更されません。キャラクタ・セットには、7ビットASCII（情報交換用米国標準コード）、EBCDIC（拡張 2 進化 10 進コード）、コード・ページ 500 、および日本語拡張 UNIX コードなどがあります。 Oracle は、シングルバイトとマルチバイトの両方の文字コード体系をサポートしています。

追加情報: 文字データ型を選択する方法は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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 までの値で指定しなければなりません（単位は文字数ではなくバイト数)。(デフォルトは 1 です。） Oracle によって、次のことが保証されます。

- 表に行を挿入したり更新したりするとき、 CHAR 列の値の長さは固定長になる。
- 短い値を与えると、固定長に合わせて空白が埋め込まれる。
- 末尾に空白が続く長い値を与えると、固定長に合わせて値から空白が切り捨てられる。
- 値が大きすぎると、 Oracle からエラーが戻される。

Oracle は、空白埋め比較方法を使って、 CHAR 値を比較します。

> **追加情報**: 比較方法の詳細は、『Oracle8 Server SQL リファレンス』を参照してください。

VARCHAR2 データ型

VARCHAR2 データ型には、可変長の文字列が格納されます。 VARCHAR2 列を含む表を作成するときは、 VARCHAR2 列の文字列の最大長を 1 から 4000 までの値で指定します（単位は文字数ではなくバイト数)。各行の値は、可変長フィールドとして列に格納されます（値が列の最大長を超えている場合は、 Oracle からエラーが戻される)。

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）だけが格納されます。

Oracle は非空白埋め比較方法によって VARCHAR2 値を比較します。

> **追加情報** : 『 Oracle8 Server SQL リファレンス』を参照してください。

VARCHAR データ型

現在のところ、 VARCHAR データ型は、 VARCHAR2 データ型と同義です。しかし、 Oracle の将来のバージョンでは、 VARCHAR データ型にはさまざまな比較方法によって比較される可変長の文字列が格納されるようになる可能性があります。このため、可変長文字列を格納する場合は、 VARCHAR2 データ型を使ってください。

文字データ型と NLS キャラクタ・セットの列の長さ

Oracle の各国語サポート（ NLS ）機能は、文字データ型にさまざまなキャラクタ・セットを使えるようにします。各国語サポートを使うと、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・セットの間で変換したりできます。クライアント・セッションでは、データベース・キャラクタ・セットとは異なる各国文字キャラクタ・セットを使うことができます。

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、文字データを入れる列のある表の領域を推定する場合に考慮する必要があります。

追加情報: Oracle の NLS 機能の詳細は、『Oracle8 Server リファレンス・マニュアル』と『Oracle8 Server ユーティリティ』を参照してください。

NCHAR および NVARCHAR2 データ型

NCHAR データ型と NVARCHAR2 データ型は NLS 文字データを格納します。 NCHAR データ型は、固定長または可変長の各国文字キャラクタ・セットに対応する固定長の文字列を格納するのに対し、 NVARCHAR2 データ型は可変長の文字列を格納します。

NCHAR または NVARCHAR2 の列を含む表を作成するときは、最大サイズとして文字数（固定長の各国文字キャラクタ・セットの場合）またはバイト数（可変長の各国文字キャラクタ・セットの場合）のいずれかを指定します。

- NCHAR 列の最大長は 2000 バイト、または 2000 バイトに入れられる文字数です。
- NVARCHAR2 列の最大長は 4000 バイト、または 4000 バイトに入れられる文字数です。

追加情報: NCHARおよびNVARCHAR2データ型の詳細は、『Oracle8 Server リファレンス・マニュアル』と『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。

LOB 文字データ型

文字データのLOBデータ型はCLOBとNCLOBで、 最大4ギガバイトの文字データ（CLOB）または各国文字キャラクタ・セット・データ（NCLOB）を格納できます。これらのデータ型は、 10-9 ページの「LOB データ型」に説明があります。

LONG データ型

LONG と定義されている列には、可変長の文字データを格納することができ、最大 2 ギガバイトまでの情報を入れることができます。 LONG データは、異なるシステム間で移動しても適切に変換されるテキスト・データです。

LONG データ型の列は、ビュー定義のテキストを格納するために、データ・ディクショナリで使われます。 LONG 列は、 SELECT リストおよび UPDATE 文の SET 句、 INSERT 文の VALUES 句で使えます。

注意: LONGデータ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションでは、大量の文字データを格納するのに CLOB および NCLOB データ型を使ってください。

追加情報: LONG データ型には多くの制限があります。『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

LONG RAW データ型の詳細は、 10-10 ページの「RAW および LONG RAW データ型」も参照してください。

NUMBER データ型

NUMBER データ型には、固定長および浮動小数点の数値が格納されます。事実上どんな大きさの数値でも格納可能です。 Oracle が稼働するシステムであれば、最大 38 桁の精度で、システム間の移植性が保証されています。

NUMBER 列には、次の数値を格納できます。

- $1 \times 10^{-130} \sim 9.99..9 \times 10^{125}$ の範囲の正の数（最大有効桁数は 38 ）。
- $-1 \times 10^{-130} \sim 9.99..99 \times 10^{125}$ の範囲の負の数（最大有効桁数は 38 ）。
- 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に限り生成される）。

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「位取り」（小数点未満の桁数）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。位取りを指定しないと、位取りはゼロになります。

Oracle では、 38 桁以下の精度で数値の移植性が保証されています。次のように、位取りだけを指定して、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、指定した位取りが保持されます。

数値フィールドを指定するときは、精度と位取りを指定するのがよい方法です。こうしておくと、入力時の整合性チェックがさらに強化されます。

表 10-1 に、さまざま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 10-1 スケール変更係数が数値データの格納に与える影響

入力データ	指定	格納方法
7,456,123.89	NUMBER	7456123.89
7,456,123.89	NUMBER(*,1)	7456123.9
7,456,123.89	NUMBER(9)	7456124
7,456,123.89	NUMBER(9,2)	7456123.89
7,456,123.89	NUMBER(9,1)	7456123.9
7,456,123.89	NUMBER(6)	（精度を超えているので、受け入れられない）
7,456,123.89	NUMBER(7,-2)	7456100

負の位取りを指定すると、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 10-1 に示されているとおり、 Oracle によって 100 の位で丸められることを意味しています。

数値の入力と出力において、標準Oracleのデフォルト小数点文字はピリオドです。たとえば、1234.56 というようになります。（小数点文字は、数値の整数部と小数部を分離する文字です。） デフォルトの小数点文字は、パラメータ NLS_NUMERIC_CHARACTERS を使って変更できます。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。現在のデフォルト小数点文字を使っていない数値を入力するには、 TO_NUMBER 関数を使います。

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 バイトが使われます。（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 x 10^2 という形式で格納され、指数（ 2 ）を格納するのに 1 バイト、仮数の 3 桁のの有効数字（ 4 、 1 、 2 ）を格納するのに 2 バイトが使われます。

このことを考慮に入れると、値の精度をpとすると（位取りは無関係）、数値データNUMBER (p) の列データ・サイズは、次の式を使って計算できます。

1 バイト	（指数部）
+ FLOOR(p/2)+1 バイト	（仮数部）
+ 1 バイト	（有効数字が 38 桁未満の負数の場合のみ）

データのバイト数

ゼロおよび正と負の無限大（ Oracle V5 データベースからのインポート時に限って生成される）は、特別な表現で格納されます。ゼロと負の無限大にはそれぞれ 1 バイト、正の無限大には 2 バイトが必要です。

DATE データ型

DATEデータ型には、ある時点の値（日付と時刻）が表に格納されます。DATEデータ型には、年（世紀を含む）および月、日、時、分、秒（真夜中から数える）が格納されます。

Oracle では、ユリウス暦の西暦前 4712 年 1 月 1 日から西暦 4712 年 12 月 31 日までの日付を格納できます。 BCE （書式マスクでは 'BC' ）が明確に指定されない限り、デフォルトとして日付は西暦として扱われます。

Oracleは、日付を格納するために固有の内部形式を使います。日付データは、それぞれ世紀、年、月、日、時、分、および秒に対応する 7 バイトの固定長フィールドに格納されます。

日付の入出力に対する標準的な Oracle のデフォルト日付書式は、 DD-MON-YY です。次に例を示します。

```
'13-NOV-92'
```

このデフォルト日付書式は、インスタンスごとにパラメータ NLS_DATE_FORMAT を使って変更できます。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 関数を使ってください。

```
TO_DATE ('November 13, 1992', 'MONTH DD, YYYY')
```

注意： 標準的な日付書式 DD-MON-YY を使った場合、 YY は 20 世紀における年を表します（たとえば、 31-DEC-92 は 1992 年 12 月 31 日を表します）。 20 世紀以外の世紀年を表すには、前述のように異なる書式マスクを使ってください。

Oracle は、時刻を 24 時間形式（ HH:MI:SS ）で格納します。デフォルトでは、時刻部分に何も指定しない場合、日付フィールドの時刻部分は 00:00:00 A.M. （真夜中）になります。時刻だけを入力した場合、日付部分のデフォルトは現在の月の 1 日になります。日付の時刻部分を入力するには、次のように時刻部分を表す書式マスクを指定してTO_DATE関数を使ってください。

```
INSERT INTO birthdays (bname, bday) VALUES
    ('ANDY',TO_DATE('13-AUG-66 12:56 A.M.','DD-MON-YY HH:MI A.M.'));
```

ユリウス歴を使う

ユリウス暦を使うと、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）が基準になるので、現在の日付は 2,400,000 辺りになります。）ユリウス暦は、通常は整数ではなく、小数部が日付部分になります。 Oracle は簡略化された方法を使っているので、結果は整数値になります。ユリウス暦は、さまざまな計算方法と解釈があります。 Oracle で使っている計算方法では、 7 桁の数値（最も一般的に使われる日付の場合）が生成されます。たとえば、 08-APR-93 は 2449086 になります。

注意: Oracleのユリウス暦日付は、他の日付アルゴリズムで生成されるユリウス暦日付と互換性がない場合があります。

日付データをユリウス暦に変換するために、日付関数（ TO_DATE または TO_CHAR ）に書式マスク「J」を指定できます。たとえば、次の問合せはユリウス歴の日付書式ですべての日付を戻します。

```
SELECT TO_CHAR (hiredate, 'J') FROM emp;
```

ユリウス歴を計算で使う場合は、 TO_NUMBER 関数を使う必要があります。ユリウス歴を入力する場合は、 TO_DATE を使えます。

```
INSERT INTO emp (hiredate) VALUES (TO_DATE(2448921, 'J'));
```

日付算術

Oracle日付算術では、過去採用されてきたさまざまな暦の変則を考慮しています。たとえば、ユリウス歴からグレゴリウス歴への切替え（ 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することができます。しかし、そのような日付は、日付算術では無視され、次の「実在する」日付として処理されます。たとえば、 04-10-1582 の次の日は 15-10-1582 で、 05-10-1582 の次の日も 15-10-1582 になります。

注意: 日付算術に関するこの説明は、すべての国（アジア圏など）の日付規格に適用できるとは限りません。

世紀と西暦 2000 年

Oracle は、データを世紀情報と一緒に格納します。たとえば Oracle データベースは、単に 96 または 01 ではなく、 1996 または 2001 として格納します。 DATE データ型は、常に 4 桁の年を内部的に格納しており、データベースに内部的に格納されている他のすべての日付は、 4 桁の年になっています。インポートおよびエクスポート、回復などの Oracle ユーティリティでも、年表示を 4 桁で正しく処理しています。

しかし、中には、年を仮定して（年は必ず 19xx のはず、など）作成されているアプリケーションもあります。このため、アプリケーション・プログラマは西暦 2000 年についてのコードを再検討し、テストする必要があります。

追加情報: 世紀と日付書式マスクの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。日付書式コードの一般情報は、『Oracle8 Server 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 および CLOB 、 NCLOB 、 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、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 4 ギガバイトです。このデータ型を使えば、個々のデータに対する効率的なランダム・アクセスが可能です。

LOB データ型は、いくつかの点で LONG および LONG RAW データ型と異なります。たとえば、 LOB データ型（ NCLOB を除く）にはユーザー定義オブジェクト型の属性を指定できますが、LONGデータ型には指定できません。LOBの最大サイズは4ギガバイトですが、LONG の最大サイズは 2 ギガバイトです。また、 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 データ型は、インライン（表の中）、ライン外（表領域の中。LOB ロケータを使う）、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。

追加情報: LOB の格納方法と LOB ロケータの詳細は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

SQL 文を使って、表に LOB 列を定義したり、ユーザー定義オブジェクト型に LOB 属性を定義したりできます。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 属性 LOGGING および NOLOGGING の詳細は、 21-7 ページの「デフォルトのロギング・モード」を参照してください。

BLOB データ型

BLOB データ型は、構造化されていないバイナリ・データをデータベースに格納します。格納できるバイナリ・データの最大サイズは、 4 ギガバイトです。

BLOB はトランザクションに全面的に参加します。 DBMS_LOB パッケージ、 PL/SQL 、または OCI によって BLOB 値に加えられる変更は、コミット / 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 / 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、最大で 4 ギガバイトの文字データをデータベースに格納します。 CLOB はシングルバイト・キャラクタ・セットのデータを格納し、 NCLOB は固定長のマルチバイト各国文字キャラクタ・セットのデータ（ NCHAR データ）を格納します。

CLOB と NCLOB はトランザクションに全面的に参加します。 DBMS_LOB パッケージ、 PL/SQL 、または OCI によって CLOB または NCLOB 値に加えられる変更は、コミット / 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 / セッションにまたがることはできません。

NCLOBの属性を持ったオブジェクト型を作成することはできませんが、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。

BFILE データ型

BFILE データ型は、構造化されていないバイナリ・データを、データベースの外にあるオペレーティング・システム・ファイルに格納します。 BFILE 列または BFILE 属性は、データが含まれる外部ファイルを参照するファイル・ロケータを格納します。格納できるデータの最大サイズは、 4 ギガバイトです。

BFILE は読込み専用ですので、変更はできません。ランダム読込みだけをサポートしており（順次読込みはサポートしていない）、トランザクションには参加しません。BFILEのファイルの整合性と耐久性を管理するのは、基礎になるオペレーティング・システムです。データベース管理者は、参照するファイルが存在していること、またそのファイルに対するオペレーティング・システムの読込み許可が Oracle プロセスに与えられていることを確認する必要があります。

RAW および LONG RAW データ型

注意: RAWおよびLONG RAWデータ型は、既存のアプリケーションとの下位互換性を保つために用意されています。新しいアプリケーションの場合は、大量のバイナリ・データを格納するのに BLOB および BFILE データ型を使ってください。

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データを移動するときに変換されない）データに使います。これらのデータ型は、バイナリ・データやバイト文字列用に用意されています。たとえば、図形データまたは音声データ、文書、バイナリ・データの配列を格納するのにLONG RAWを使えます。解釈は、用途によって異なります。

RAW は、 VARCHAR2 文字データ型と同じく可変長のデータ型です。ただし、 RAW または LONG RAWデータの送信時に、Net8（ユーザー・セッションをインスタンスに接続する）および Import/Export ユーティリティによって文字変換は実行されません。それとは対照的に、Net8 と Import/Export ユーティリティは、データベースのキャラクタ・セットとユーザー・セッションのキャラクタ・セットが異なる場合に、これらのキャラクタ・セット間で CHAR および VARCHAR2 、 LONG データを自動変換します。

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。

追加情報: LONG RAWデータ型のその他の制限は、『Oracle8 Serverアプリケーション開発者ガイド』を参照してください。

ROWID データ型

拡張 ROWID データ型には、データベース内の各行のアドレスが格納されます。拡張 ROWID は、パーティション化されていない表や索引だけでなく、パーティション表やパーティション索引の行も識別します。また、表領域に対する相対的なデータ・ブロック・アドレスもサポートしています。既存の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 データ型も使用可能です。

ROWID と ROWID データ型

クラスタ化されてない表のすべての行には、行の行断片（行が複数の行断片にわたって連鎖している場合は、最初の行断片）の物理アドレスに対応する、一意のROWIDが割り当てられています。クラスタ化された表の場合、同じデータ・ブロックにある異なる表の行の ROWID が同じである場合もあります。

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 .. . 文を実行することによっ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。しかし、それぞれの行のアドレスは、予約語ROWIDを列名として使うことにより、SQL問合せで取得できます。たとえば、次のようにします。

```
SELECT ROWID, ename FROM emp;
```

行に割り当てられている ROWID は、その行がエクスポートされたりインポートされたり（ Import および Export ユーティリティを使う）しない限り、変更されません。表から行を削除する（そしてその後、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。

疑似列 ROWID の値は、 INSERT または UPDATE 文で設定することはできません。また、ROWID の値は削除できません。 Oracle は、疑似列 ROWID の ROWID を内部的に使って、 10-14 ページの「 ROWID の使用方法」で説明するさまざまな操作を実行します。

疑似列 ROWID の ROWID は他の表の列と同じように参照できますが（ SELECT リストや WHERE 句で使う）、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あるわけでもありません。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。

拡張 ROWID

拡張 ROWID は、それぞれの選択された行の物理アドレスを基本 64 コード化で表現したものです。次に例を示します。

```
SELECT ROWID, ename FROM emp
```

```
    WHERE deptno = 20;
```

上記の問合せでは、次のような行情報が戻されます。

```
ROWID              ENAME
------------------ ----------
AAAAaoAATAAABrXAAA BORTINS
AAAAaoAATAAABrXAAE RUGGLES
AAAAaoAATAAABrXAAG CHEN
AAAAaoAATAAABrXAAN BLUMBERG
```

拡張 ROWID の書式は、 OOOOOOFFFBBBBBBRRR という 4 つの部分から成っています。

- 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 ）を表します。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: その行が入っているデータ・ファイル（例ではファイル AAT ）。ファイル番号は、データベース内で一意です。
- BBBBBB: その行が入っているデータ・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 および DBA_OBJECTS 、 ALL_OBJECTS から取得できます。たとえば、次の問合せは、スキーマ SCOTT の EMP 表のデータ・オブジェクト番号を戻します。

```
SELECT DATA_OBJECT_ID FROM DBA_OBJECTS
    WHERE OWNER = 'SCOTT' AND OBJECT_NAME = 'EMP';
```

DBMS_ROWID パッケージを使って、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したり（またはその逆も）できます。

> **追加情報** : DBMS_ROWID パッケージの詳細は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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 を使って問合せを実行すると、このバイナリ表現が VARCHAR2/16 進数表現に変換されます。たとえば、次の問合せでは、

```
SELECT ROWID, ename FROM emp
    WHERE deptno = 30;
```

次のような行情報が戻されます。

```
ROWID              ENAME
------------------ ----------
00000DD5.0000.0001 KRISHNAN
00000DD5.0001.0001 ARBUCKLE
00000DD5.0002.0001 NGUYEN
```

上記のように、制限付き ROWID の VARCHAR2/16 進数表現は、 block.row.file という 3 つの部分から成ります。

- その行が入っているデータ・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。
- その行が入っているブロックの中の行（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。
- その行が入っているデータ・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 を使って、 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 を使って拡張 ROWID を 4 つのコンポーネント（データベース・オブジェクトおよびファイル、ブロック、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 を使って制限付き ROWID を 3 つのコンポーネント（ブロックおよび行、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知りたい場合（表のストライプ化の場合など)、次のように拡張ROWIDを問い合せることによって、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
```

追加情報 : ROWID の使用方法の詳細は、『 Oracle8 Server SQL リファレンス』および『 PL/SQL ユーザーズ・ガイドおよびリファレンス』、『 Oracle8 Server チューニング』、 Oracle Tools とユーティリティについて扱ったその他のマニュアルを参照してください。

ROWID と Oracle 以外のデータベース

Oracle データベース・アプリケーションは、 SQL*Connect または Oracle Open Gateway を使って、 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。さらに、 VARCHAR2/16 進形式への標準変換も使えません。プログラムで ROWID データ型を使うこともできますが、最大 256 バイトまでの 16 進形式への非標準変換を使う必要があります。

追加情報 : Oracle 以外のシステムで ROWID を処理する場合の詳細は、 OCI またはプリコンパイラの関連マニュアルを参照してください。

ROWID の使用方法

Oracleは、 ROWIDを内部的に使っ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の開発者は、次のような重要な用途のために ROWID を使うこともできます。

- ROWID は特定の行にアクセスする最も高速な方法である。
- ROWID は表の編成を把握するのに使える。
- ROWID は表の中の行の一意識別子である。

DML 文の中で ROWID を使うときに、その前に、 ROWID が変更されないことを検証し保証する必要があります。目的の行をロックして、削除されないようにしてください。無効な ROWID を使ってデータを要求すると、状況によっては、文が失敗する原因になります。

また、ROWIDデータ型を使っ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 を格納することができます。 ROWID データ型を使っ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にあるそれぞれの値を、適切な列データとして格納するには 6 バイト必要です。

MLSLABEL データ型

Trusted Oracleは、 MLSLABELデータ型を用意しています。このデータ型には、マルチレベル・セキュリティを備えたオペレーティング・システムで生成されたラベルの、 Trusted Oracle 内部表現が格納されます。 Trusted Oracle は、ラベルを使ってデータベース・アクセスを制御します。

Oracle8 でも、 Trusted Oracle アプリケーションとの互換性を保つために MLSLABEL データ型を使って列を定義することができますが、その場合、 Oracle8 でその列に指定できる値は NULL だけです。

Trusted Oracle で表を作成すると、 ROWLABEL という列が自動的に表に追加されます。この列には、その表のすべての行の MLSLABEL データ型のラベルが入れられます。

> **追加情報**: MLSLABEL データ型および ROWLABEL 列、 Trusted Oracle の詳細は、 Trusted Oracle のマニュアルを参照してください。

Oracle データ型についての情報のまとめ

表 10-2 に、 Oracle データ型に関する重要な情報をまとめます。

表 10-2 Oracle ビルトイン・データ型の要約

データ型	説明	列の長さとデフォルト
CHAR (size)	長さが size バイトの固定長文字データ。	表のどの行でも固定（後続空白あり）。最大サイズは行あたり 2000 バイト、デフォルト・サイズは行あたり 1 バイトです。sizeを設定する前に、キャラクタ・セットについて検討してください（1バイトまたはマルチバイトのどちらか）。
VARCHAR2 (size)	可変長文字データ。最大サイズ size を指定する必要があります。	行ごとに可変。最大サイズは 4000 バイトです。 size を設定する前に、キャラクタ・セットについて検討してください。

表 10-2 Oracle ビルトイン・データ型の要約

データ型	説明	列の長さとデフォルト
NCHAR(size)	固定長文字データ。長さは、size 文字または size バイト（各国文字キャラクタ・セットによって変わる）。	表のどの行でも固定（後続空白あり）。列の「サイズ」は、文字数（固定幅の各国文字キャラクタ・セットの場合）またはバイト数（可変幅の各国文字キャラクタ・セットの場合）のいずれかで指定します。最大「サイズ」は、1つの文字を格納するのに必要なバイト数によって決まります。行あたりの上限は 2000 バイトです。デフォルトは 1 文字または 1 バイト（キャラクタ・セットによって変わる）です。
NVARCHAR2 (size)	長さが size 文字またはバイト（各国文字キャラクタ・セットによって変わる）の固定長文字データ。最大サイズ size を指定する必要があります。	行ごとに可変。列の「サイズ」は、文字数（固定幅の各国文字キャラクタ・セットの場合）またはバイト数（可変幅の各国文字キャラクタ・セットの場合）のいずれかで指定します。最大「サイズ」は、1つの文字を格納するのに必要なバイト数によって決まります。行あたりの上限は 4000 バイトです。デフォルトは 1 文字または1バイト（キャラクタ・セットによって変わる）です。
LONG	可変長文字データ。	表の行ごとに可変。行あたりの最大サイズは 2^31 - 1 バイト、つまり 2 ギガバイトです。
NUMBER (p, s)	可変長数値データ。精度 p やスケール s の最大値は 38 です。	行ごとに可変。 1 つの列に必要な最大領域は行あたり 21 バイト。
DATE	固定長の日時データ。範囲は西暦前 4712 年 1 月 1 日から西暦 4712 年 12 月 31 日までです。	表のそれぞれの行の固定値は7バイト。デフォルト形式は、 NLS_DATE_FORMAT パラメータで指定されている文字列（たとえば、 DD-MON-YY ）です。
RAW (size)	可変長ロー・バイナリ・データ。最大サイズ size を指定する必要があります。	表の行ごとに可変。行あたりの最大サイズは 2000 バイトです。
LONG RAW	可変長ロー・バイナリ・データ。	表の行ごとに可変。行あたりの最大サイズは 2^31 - 1 バイト、つまり 2 ギガバイトです。
BLOB	バイナリ・データ。	最大サイズは 2^{32} - 1 バイト、つまり 4 ギガバイトです。
CLOB	シングルバイト文字データ。	最大サイズは 2^{32} - 1 バイト、つまり 4 ギガバイトです。
NCLOB	シングルバイトまたは固定長マルチバイトの各国文字キャラクタ・セット（ NCHAR ）データ。	最大サイズは 2^{32} - 1 バイト、つまり 4 ギガバイトです。

表 10-2 Oracle ビルトイン・データ型の要約

データ型	説明	列の長さとデフォルト
BFILE	外部ファイルに格納されるバイナリ・データ。	最大サイズは 2^{32} - 1 バイト、つまり 4 ギガバイトです。
ROWID	行アドレスを表すバイナリ・データ。	表のどの行でも 10 バイト（拡張 ROWID ）または 6 バイト（制限付き ROWID ）に固定。
MLSLABEL	Trusted Oracle データ型。	Trusted Oracle のマニュアルを参照してください。

ANSI データ型、 DB2 データ型、 SQL/DS データ型

表 10-3 に、 ANSI データ型から Oracle データ型への変換を示します。 ANSI/ISO データ型 NUMERIC および DECIMAL 、 DEC では、固定小数点の数値だけを指定できます。これらのデータ型については、 s （スケール）はデフォルトの 0 に設定されます。

表 10-3 ANSI データ型から Oracle データ型への変換

ANSI SQL データ型	Oracle データ型
CHARACTER (n) 、 CHAR(n)	CHAR(n)
NUMERIC (p,s) 、 DECIMAL (p,s) 、 DEC (p,s)	NUMBER (p,s)
INTEGER 、 INT 、 SMALLINT	NUMBER (38)
FLOAT (p)	FLOAT (p)
REAL	FLOAT (63)
DOUBLE PRECISION	FLOAT (126)
CHARACTER VARYING(n) 、 CHAR VARYING(n)	VARCHAR2 (n)

IBM 製品 SQL/DS と DB2 のデータ型 TIME および TIMESTAMP、 GRAPHIC、 VARGRAPHIC、 LONG VARGRAPHIC に対応する Oracle データ型は存在しないので使用できません。 TIME データ型と TIMESTAMP データ型は、 Oracle データ型 DATE の部分構成要素です。

表 10-4 に、 DB2 と SQL/DS の変換を示します。

表 10-4 SQL/DS 、 DB2 データ型から Oracle データ型への変換

DB2 または SQL/DS データ型	Oracle データ型
CHARACTER (n)	CHAR(n)
VARCHAR(n)	VARCHAR2 (n)

表 10-4 SQL/DS 、 DB2 データ型から Oracle データ型への変換

DB2 または SQL/DS データ型	Oracle データ型
LONG VARCHAR	LONG
DECIMAL (p,s)	NUMBER (p,s)
INTEGER, SMALLINT	NUMBER (38)
FLOAT (p)	FLOAT (p)
DATE	DATE

データ変換

場合によっては、予期しているのとは異なるデータ型が与えられることがあります。そのような値は、 Oracle が次の関数の 1 つを使って、データを必要なデータ型に変換できるとき、データ変換が自動的に行われます。

- TO_NUMBER()
- TO_CHAR()
- TO_DATE()
- CHARTOROWID()
- ROWIDTOCHAR()
- HEXTORAW()
- RAWTOHEX()

追加情報: データ型の暗黙変換の規則は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

Trusted Oracleを使っている場合、データ変換およびMLSLABELデータ変換の詳細は、 Trusted Oracle のマニュアルを参照してください。

11

ユーザー定義データ型（オブジェクト・オプション）

We must learn to explore all the options and possibilities thatconfront us in a complex and rapidly changing world.

James William Fulbright, *Speech in the Senate (1964)*

オブジェクト・オプションを使うと、アプリケーションのデータの構造と動作をモデル化するデータ型を定義できます。

この章では、次の内容を取り上げます。

- 基礎知識
- ユーザー定義データ型
- アプリケーション・インタフェース

注意：この章で説明している機能は、オブジェクト・オプション付きのOracle8 Enterprise Editionを購入した場合にのみ利用できます。この章で使われている Oracle Server という用語は、オブジェクト・オプション付きのOracle8 Enterprise Editionを指します。Oracle8 Enterprise Editionで使用可能な機能とオプションの詳細は、『Oracle8 と Oracle8 Enterprise Editionの解説』を参照してください。

基礎知識

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。これを使うことにより、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を、速く、効率的に、そして十分な信頼性をもってアクセスできます。

オブジェクト・オプションが追加されると、 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）になります。これは、ユーザーがいろいろな種類のデータを定義し（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えるようになった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・オプション付きの Oracle 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。この理由で、オブジェクト指向のプログラミング技法を使っ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。

Oracle がユーザー定義データ型をサポートしたことにより、アプリケーション開発者は、イメージおよびオーディオ、ビデオなどの複合データをより簡単に処理できるようになりました。

複合データ・モデル

Oracle Server では、複合ビジネス・モデルを SQL で定義し、それをデータベース・スキーマの一部にすることができます。データを管理し共有するアプリケーションに入っている必要があるのは、データ・ロジックではなく、アプリケーション・ロジックだけです。

例

たとえば、発注情報を使って、購買、買掛管理、出荷、売掛管理の機能を編成しているとします。

1 件の発注情報には、関連する供給業者または顧客と、不特定数の品目が含まれています。さらに、アプリケーションでは、発注に関して状況情報を動的に計算する機能がしばしば必要になります。たとえば、出荷済みまたは未出荷の品目の現在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うことになる、「オブジェクト型」と呼ばれるスキーマ・オブジェクトを定義する方法は、この章の後半で扱います。オブジェクト型では、発注などの構造化されたデータ単位を形成する個々の要素を指定します。この要素のことを「属性」といいます。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（たとえば、品目リスト）。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」といいます。

テンプレートと合致する発注情報を作成し、数値や日付とまったく同じように表の列に格納することができます。

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ので、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って、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、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、 RDBMS はかなりの程度まで効率が良くなります。 RDBMS には、値を比較したり、値の分散状況を判別したり、能率的な索引を作成したり、その他の最適化を実行したりするための機能が備えられています。

これらの基本的なデータ型にうまく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、空間データなどがあります。オブジェクト・オプション付きの Oracle では、そのような複合データ型のモデル化と実現がサポートされます。

ユーザー定義データ型

Oracle のビルトイン・データ型については、第 10 章「ビルトイン・データ型」で説明しています。オブジェクト・オプションでは、ユーザー定義データ型に、次の 2 つのカテゴリが追加されています。

- オブジェクト型
- コレクション型

ユーザー定義データ型では、ビルトイン・データ型と他のユーザー定義データ型を、アプリケーション・データの構造と動作をモデル化するデータ型の構築ブロックとして使います。

ユーザー定義データ型は、スキーマ・オブジェクトです。これらのオブジェクトの使用は、他のスキーマ・オブジェクトと同様の管理制御を受けます（第 12 章「ユーザー定義データ型の使用方法」を参照）。

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（たとえば、注文）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、ビルトイン型か他のユーザー定義型です。

- 「メソッド」。これは、 PL/SQL で作成されてデータベースに格納されたファンクションまたはプロシージャ、もしくは、 C などの言語で作成されて外部に格納されたファンクション（関数）または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インプリメントします。

オブジェクト型はテンプレートです。テンプレートに合わせて構造化されたデータ単位を「オブジェクト」といいます。

発注情報の例

ここで説明する例では、 external_person および lineitem 、 purchase_order というオブジェクト型を定義する方法を示します。

external_person と lineitem のオブジェクト型は、ビルトイン型の属性を持ちます。オブジェクト型purchase_orderの構造はもっと複雑で、実際の発注情報の構造と厳密に一致するものになっています。

purchase_orderの属性は、 id、 contact、およびlineitemsです。属性contactはオブジェクトで、属性 lineitems はネストした表です（ 11-10 ページの「ネストした表」を参照）。

```
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 ) ;
```

上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ありません。また、実際の発注情報の複雑さも完全には反映していません。

> **追加情報**：発注情報の詳細な例は、『Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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、記憶領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のほとんどについて、 lineitem または external_person 、 purchase_order を使うことができます。

たとえば、次のようなリレーショナル表を定義して、発注先の担当者（ contact ）を追跡し記録できます。

```
CREATE TABLE contacts (
  contact external_person
  date DATE ) ;
```

contact 表は、列の 1 つがオブジェクト型で定義されているリレーショナル表です。リレーショナル表の列を占めるオブジェクトを「列オブジェクト」といいます（ 11-7 ページの「行オブジェクトと列オブジェクト」を参照）。

メソッド

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、それ専用の get_value メソッドがあります。たとえば、 x と y が発注情報オブジェクトを保持する PL/SQL 変数で、 w と z が数値を保持する変数である場合、次の 2 つの文による w と z の結果値は異なる可能性があります。

```
w = x.get_value() ;
z = y.get_value() ;
```

これらの文を実行すると、 w には変数 x で表される発注情報の値、 z には変数 y で表される発注情報の値が代入されます。

「 x.get_value () 」という用法により、 get_value メソッドが起動されます。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からです。つまり、 1 番目のサンプル文では発注情報 x の属性を使って値を計算し、 2 番目のサンプル文では発注情報 y の属性を使って値を計算します。これを「自己参照的なスタイル」のメソッドの起動といいます。

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す。コンストラクタ・メソッドのパラメータの名前は、オブジェクト型の属性の名前です。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返します。

たとえば、次のような式は、

```
purchase_order(
```

```
1000376,
external_person ('John Smith','1-800-555-1212'),
NULL )
```

次のような属性をも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。

NULL オブジェクトと NULL 属性については、12-6ページの「NULL」を参照してください。

比較メソッド

メソッドは、オブジェクトの比較でも役割を果たします。 Oracle には、特定のビルトイン型の 2 つのデータ項目（たとえば 2 つの数値）を比較し、一方が他方より大きい、等しい、または小さいかどうかを判別する機能があります。しかし、 Oracle では、定義者がさらに指示を与えないかぎり、任意のユーザー定義型の 2 つの項目を比較することができません。 Oracle は、特定のオブジェクト型のオブジェクト間の順序関係を定義するための 2 つの方法として、「マップ」メソッドと「順序」メソッドを提供しています。

マップ・メソッドでは、ビルトイン型を比較する Oracle の機能を使います。たとえば、「rectangle （四角形）」というオブジェクト型に「height （高さ）」と「width （幅）」という属性を定義したとします。「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。

順序メソッドは、もっと一般的な方法です。順序メソッドは、独自の内部論理を使用して、特定のオブジェクト型に属する 2 つのオブジェクトを比較します。それは、順序関係を符号化した値を戻します。たとえば、最初のオブジェクトのほうが小さければ -1 を戻し、どちらも同じ大きさであれば 0 を戻し、最初のオブジェクトのほうが大きければ 1 を戻す、などです。

たとえば、「address（住所）」というオブジェクト型に、「street（番地）」、「city（市）」、「state（州）」「zip （郵便番号）」という属性を定義するとします。アプリケーションで住所を比較して「より大きい」か「より小さい」かを判別するのは無意味ですが、 2 つの住所が等しいかどうかを判別するには複雑な計算をする必要があるかもしれません。

オブジェクト型を定義する際、マップ・メソッドか順序メソッドのどちらかを指定できますが、両方は指定できません。オブジェクト型に比較メソッドがない場合、 Oracle はその型の 2 つのオブジェクト間の大小関係を判別できません。ただし、その型の 2 つのオブジェクトが等しいかどうかの判別は試行できます。

比較メソッドを持たない型の 2 つのオブジェクトを比較する場合、 Oracle は、対応する属性を比較します。

- すべての属性が NULL 以外であり、等しい場合、 Oracle はその 2 つのオブジェクトが等しいと報告します。
- 2 つのオブジェクトに等しくない NULL 以外の値がある場合、 Oracle はその 2 つのオブジェクトが等しくないと報告します。
- その他の場合、 Oracle は、比較できないことを報告します（ NULL ）。

追加情報: 比較メソッドの指定方法と使用方法の例は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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、それらの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_t OF external_person ;
```

Oracle では、次の 2 つの方法でこの表を表示できます。

- 単一列の表。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。
- 複数列の表。オブジェクト型 external_person のそれぞれの属性、つまり名前と電話番号が列を 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_t VALUES (
       'John Smith'
       '1-800-555-1212' ) ;

SELECT VALUE(p) FROM external_person_t p
        WHERE p.name = 'John Smith' ;
```

最初の命令は、複数列の表としてのexternal_person_tに発注情報オブジェクトを挿入します。2 番目の命令は、単一列の表としての external_person_t 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」といいます。表の列だけ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だけ表示されるオブジェクトのことを「列オブジェクト」といいます。

参照（REF）

リレーショナル・モデルの場合、外部キーは多対 1 の関係を表します。オブジェクト・オプション付きの Oracle では、多対 1 の関係で「1」の側が行オブジェクトである場合に、この関係を表すための効率的な方法が提供されています。Oracleは、すべての行オブジェクトに、「オブジェクト識別子」という一意かつ不変の識別子を付与します。 Oracle は、オブジェクト識別子の内部構造の説明や、内部構造へのアクセス手段を提供していません。内部構造は、随時変更されることがあります。

オブジェクト識別子を使うと、それに対応する行オブジェクトを、他のオブジェクトやリレーショナル表から参照できます。 REF というビルトイン・データ型は、そのような参照を表します。REFは、指定したオブジェクト型の行オブジェクトへの参照をカプセル化します。

オブジェクト・ビュー（第 13 章「オブジェクト・ビュー」を参照）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から具象化します。 Oracle は、これらの識別子を使って、オブジェクト・ビューの行オブジェクトへの REF を組み立てます。

REF を使って、参照先のオブジェクトを検査したり更新したりすることができます。また、参照先のオブジェクトのコピーを取得するためにも使えます。 REF に対して実行できる変更は、同じオブジェクト型の異なるオブジェクトへの参照に内容を置き換えるか、 NULL 値を割り当てるという変更だけです。

有効範囲付き REF

列タイプまたはコレクション要素、オブジェクト型の属性を REF として宣言する際、 REF に制約をかけて、指定したオブジェクト表への参照だけを含めるようにすることができます。このような REF を「有効範囲付き REF」といいます。有効範囲付き REF は、必要な記憶領域が少なくて済み、有効範囲なしの REF よりも効率よくアクセスできます。

ぶら下がり 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たりして、そのオブジェクトを使えなくなる可能性があります。このような REF を「ぶら下がり REF」といいます。Oracle の SQL は、この条件が当てはまるかどうか REF をテストする述語（IS DANGLING）を提供しています。

REF の参照解除

REF が参照するオブジェクトにアクセスすることを、その REF への「参照を解除する」といいます。 Oracle は、この操作のために DEREF 演算子を提供しています。ぶら下がり 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「暗黙的な参照解除」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とすると、次の式は、

```
x.manager.name
```

x の manager 属性が参照する person オブジェクトの name 属性を含む文字列を表します。つまり、上記の式は、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 演算子を適用することによって取得します。 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 ;
```

オブジェクトと REF の記憶領域の詳細は、 12-4 ページの「ユーザー定義型の記憶領域」を参照してください。

> **追加情報** : REF の使用方法の例は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

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」といいます。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 とは異なります。

VARRAY

「配列」とは、データ「要素」を順番に並べた集合のことです。特定の配列のすべての要素は、同じデータ型で構成されます。各要素には「索引」があり、これは配列内におけるその要素の位置に対応する番号です。

配列内の要素の数が、配列のサイズになります。 Oracle の配列のサイズは可変（ variable ）なので、 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、次の用途に使えます。

- 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（ 12-15 ページの「ユーザー定義型の Import/Export 」を参照）。

> **追加情報**: VARRAY の使用方法は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

ネストした表

「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、この列の型はビルトイン型かオブジェクト型です。オブジェクト型の場合は、この表を、オブジェクト型の各属性に対応する列のある、複数列の表とみなすこともできます。

たとえば、発注情報の例では、次の文により、品目を入れるネストした表のために使う表型を宣言します。

```
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、次の用途に使えます。

- 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われた場合、 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（ 12-5 ページの「ネストした表」を参照）。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 lineitem 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うことです。

> **追加情報**: ネストしたカーソルの使用方法の詳細は、『 Oracle8 Server リファレンス・マニュアル』を参照してください。また、ネストした表の使用方法の詳細は、『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

アプリケーション・インタフェース

Oracle には、アプリケーション・プログラムでユーザー定義データ型を使うための機能が多数あります。

- SQL
- PL/SQL
- Pro*C/C++
- OCI
- OTT

SQL

Oracle SQL DDL は、ユーザー定義データ型のために次のようなサポートを提供しています。

- オブジェクト型およびネストした表、配列の定義。
- 権限の指定。
- ユーザー定義型の表の列の指定。
- オブジェクト表の作成。

Oracle SQL DML は、ユーザー定義データ型のために次のようなサポートを提供しています。

- オブジェクトとコレクションの問合せおよび更新。
- REF の操作。

追加情報: Oracle SQL構文の完全な説明は、『Oracle8 Server SQLリファレンス』を参照してください。

PL/SQL

PL/SQL は、 SQL を拡張するプロシージャ型言語です。パッケージおよびデータのカプセル化、情報の隠ぺい、オーバーロード、例外処理など、最新のソフトウェア・エンジニアリング機能が提供されています。 PL/SQL は、ストアド・プロシージャに使う言語です。

PL/SQL では、ファンクションおよびプロシージャ内から、ユーザー定義型をサポートする SQL 機能を使うことができます。

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インプリメント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。

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

Pro*C/C++

Oracle Pro*C/C++プリコンパイラを使うと、プログラマはCおよびC++プログラムでユーザー定義データ型を使えます。

Pro*C を使う開発者は、 Object Type Translator を使って、 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うことができます。

Pro*C は、オブジェクト型とコレクションのコンパイル時型チェック機能と、データベース型から C データ型への自動型変換を提供しています。

Pro*Cは、オブジェクトを作成したり破棄したりするためのEXEC SQL構文と、サーバーにあるオブジェクトにアクセスする 2 つの方法を提供しています。

- Pro*C プログラムに埋め込まれている SQL 文と PL/SQL ファンクションまたはプロシージャ。
- オブジェクト・キャッシュへの単純なインタフェース（OCIの項で説明します）。網羅用の（ traversing ）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。

追加情報: Pro*C プリコンパイラの詳細は、『 Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

OCI

Oracle コール・インタフェース（ OCI ）は、 Oracle Server への C 言語インタフェースのセットです。 OCI を使うと、プログラマはサーバーの機能を使うときに融通を利かせることができます。

OCI の重要な要素は、オブジェクト・キャッシュと呼ばれる作業領域をアプリケーション・プログラムから使えるようにするための一式のコールです。「オブジェクト・キャッシュ」は、クライアント側に置かれるメモリー・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・キャッシュを使うアプリケーション・プログラムが完全に制御および管理します。Oracle Serverは、オブジェクト・キャッシュにアクセスできません。オブジェクト・キャッシュを使うアプリケーション・プログラムは、サーバーとのデータの一貫性を維持しなければならず、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。

OCI は、次の機能を持つ関数を提供しています。

- SQL を使ってサーバー上のオブジェクトにアクセスする。
- 網羅用の（ traversing ）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、管理する。
- Oracle の日付および文字列、数値を C データ型に変換する。
- オブジェクト・キャッシュのメモリーのサイズを管理する。

OCI の同時実行性が向上し、個々のオブジェクトにロックをかけることができるようになりました。また、複合オブジェクト検索をサポートすることにより、パフォーマンスが向上しています。

OCI を使う開発者は、 Object Type Translator を使って、 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。

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

OTT

Oracle 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・パッケージを使うときの手助けをします。

> **追加情報** : OTT の詳細は、『 Oracle コール・インタフェース・プログラマーズ・ガイド』および『 Pro*C/C++ プリコンパイラ・プログラマーズ・ガイド』を参照してください。

12

ユーザー定義データ型の使用方法

It is not enough to have a good mind. The main thing is to use itwell.

René; Descartes, *Le Discours de la Méthode*

この章では、ユーザー定義データ型を使うにあたって理解しておく必要のある主要な概念を説明します。次の内容を取り上げます。

- 参照とネーム変換
- ユーザー定義型の記憶領域
- オブジェクト属性のプロパティ
- ユーザー定義型の権限とそのメソッド
- 依存性と不完全な型
- ユーザー定義型の Import/Export

注意: この章で説明している機能は、オブジェクト・オプション付きの Oracle8 Enterprise Edition を購入した場合にのみ利用できます。

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

参照とネーム変換

Oracle SQL は、使いやすさを目指して設計されています。たとえば、 projects という表に assignment という列があり、 depts という表には assignment という列がない場合、次のように記述できます。

```
SELECT *
FROM projects
WHERE EXISTS
  (SELECT * FROM depts
           WHERE assignment = task);
```

Oracle は、それぞれの列がどの表に属するかを判断します。次のように列名を表名で修飾することは可能ですが、そうする必要はありません。

```
SELECT *
FROM projects
WHERE EXISTS
  (SELECT * FROM depts
           WHERE projects.assignment = depts.task);
```

次のように列名を表の別名で修飾することは可能ですが、そうする必要はありません。

```
SELECT *
FROM projects pj
WHERE EXISTS
  (SELECT * FROM depts dp
           WHERE pj.assignment = dp.task);
```

表の別名

上記の最初の SELECT 文の形式は最も記述しやすくて理解しやすい形式ですが、後で depts 表に assignment 列を追加して、問合せを変更し忘れると、予期しない結果になる可能性があります。 Oracle は自動的にその問合せを再コンパイルし、新しいバージョンでは depts 表の assignment 列が使われます。このような状況のことを「内側獲得」といいます。

内側獲得や、それに類似した SQL 文の解釈間違いを回避するには、表の別名を使って、オブジェクトのメソッドや属性への参照を修飾する必要があります。このことは、 REF による属性参照にも適用されます。このような要件のことを「獲得回避規則」といいます。

たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (ssno VARCHAR(20));
CREATE TABLE ptab1 OF person;
CREATE TABLE ptab2 (c1 person);
```

これらの文は、オブジェクト型 person と、 2 つの表を定義します。最初の表は、オブジェクト型 person のためのオブジェクト表です。 2 番目の表には、型 person の列が 1 つあります。

このとき、次の文を考えてみます。

```
SELECT      ssno FROM ptab1 ;  -- 正しい
SELECT   c1.ssno FROM ptab2 ;  -- 間違い
SELECT p.c1.ssno FROM ptab2 p ;-- 正しい
```

- 最初の SELECT 文で、 ssno は ptab1 の列の名前です。それ以上の修飾は不要です。
- 2 番目の SELECT 文で、 ssno は、 c1 という列に入っている person オブジェクトの属性の名前です。この参照には、表の別名が必要です。
- 3 番目の SELECT 文は 2 番目と同じですが、必要な表の別名 p が指定されています。

オブジェクトの属性への参照を、表の別名ではなく表の名前で修飾することは、その表名をさらにスキーマ名で修飾したとしても、この要件を満たすことにはなりません。

たとえば、問合せで次のような文を使っても、

```
scott.projects.assignment.duedate
```

scott スキーマの projects 表の assignment 列の dudate 属性を参照することはできません。

表の別名は、問合せの中で一意になっている必要があり、問合せの中に指定できるスキーマ名と同じにするべきではありません。

注意: オラクル社では、すべての UPDATE 文、 DELETE 文、 SELECT 文、および副問合せの中で表の別名を定義し、列にオブジェクト型が入っているかどうかに関係なく、列の参照を表の別名で修飾することをお薦めしています。

引数なしのメソッド・コール

メソッドは、ファンクションまたはサブルーチンです。メソッドを起動する正しい構文では、メソッド名の後にカッコを記述し、コール引数を囲みます。あいまいさを避けるため、 Oracle では、引数がないメソッド・コールにも空のカッコが必要です。

たとえば、オブジェクト型 t の列 c を持つ表 tb があり、 t に引数を取らないメソッド m がある場合、次のような問合せは正しい構文の一例です。

```
SELECT p.c.m() FROM tb p;
```

これは、 PL/SQL の規則とは異なっています。 PL/SQL のファンクションおよびプロシージャでは、引数がないコールの場合はカッコを省略できます。

ユーザー定義型の記憶領域

Oracle は、ユーザー定義型のデータを表に格納して管理します。ユーザー定義型の複雑な構造が、自動的に、そして人の目には見えない形で、表のシンプルな四角形構造にマップされます。

リーフ・レベル属性

オブジェクト型の構造は、ツリー構造に似ています。幹から伸びているブランチ（枝）が属性を表します。属性がオブジェクト型である場合は、このブランチから、新しいオブジェクト型の属性としてサブブランチが分岐します。

それぞれのブランチは、最終的には、ビルトイン型またはコレクション型の属性で終わります。これらの属性を、元のオブジェクト型の「リーフ・レベル属性」といいます。Oracleは、それぞれのリーフ・レベル属性用の列を表の中に用意します。

コレクション型でないリーフ・レベル属性を、オブジェクト型の「リーフ・レベル・スカラー属性」といいます。

行オブジェクト

オブジェクト表には、すべてのリーフ・レベル・スカラー属性または REF 属性について、Oracle が実際のデータを格納する列があります。これは、 VARRAY （非常に大きい場合を除く）の場合も同じです（ 12-5 ページの「 VARRAY 」を参照）。 Oracle は、表型のリーフ・レベル属性を、オブジェクト表に対応付けられた別々の表に格納します。これらの表を、オブジェクト表の宣言の一部として宣言しなければなりません（ 12-5 ページの「ネストした表」を参照）。

オブジェクト表にあるオブジェクトの個々の属性にアクセスするには、単に表の列にアクセスします。オブジェクトの値そのものにアクセスすると、オブジェクト表の列を引数として、その型のデフォルト・コンストラクタが起動されます。つまり、 Oracle はそのオブジェクトのコピーを提供します。

Oracleは、システム生成のオブジェクト識別子を非表示列に格納します。 Oracleは、このオブジェクト識別子を使って、オブジェクトへの REF を組み立てます。

列オブジェクト

表にオブジェクト型の列が定義されている場合、 Oracle は、そのオブジェクト型のリーフ・レベル属性のための非表示列を表に追加します。その追加された列には、そのオブジェクトの NULL 情報（つまり、トップレベルのオブジェクトとネストされたオブジェクトのアトミック NULL ）が格納されます。

参照（REF）

Oracleは、行オブジェクトに対してビルトイン・ファンクションREFを起動することにより、行オブジェクトへのREFを組み立てます。組み立てられたREFは、オブジェクト識別子およびオブジェクト表のメタデータ、ROWID（オプション）で構成されています。オブジェクト表にあるオブジェクトへの、ROWID付きで有効範囲なしのREFのサイズは、46バイトです。オブジェクト・ビューへのREFおよびROWIDなしのREF、有効範囲付きREFのサイズは、もっと小さくなります。

REFのROWIDは、効率的なアクセスのためのヒントとして使われます。 REF項目の参照を解除する際、 OracleはROWIDを使って行を選択します。識別された行のオブジェクト識別子がREFの識別子と一致した場合、アクセスは成功します。一致しない場合、 Oracleはオブジェクト識別子の索引を使って正しい行を識別します。

REF型の列のREFのサイズは、その列と対応付けられている記憶プロパティに応じて異なります。たとえば、列がREF WITH ROWIDとして宣言されている場合、 OracleはROWIDをREF列に格納し、そうでない場合はROWIDを廃棄します。

列をSCOPE句を指定したREFとして宣言すると、 Oracleは、オブジェクト表のメタデータとROWIDを列に格納しません。有効範囲付きREFの長さは16バイトです。

ネストした表

ネストした表の行は、別々の格納表に格納されます。ネストした表を含む表を定義するときは、格納表の名前を指定する必要があります。表定義（列、または列定義に含まれるオブジェクト型）に2つ以上の表型がある場合は、それぞれ別々の格納表を指定する必要があります。

表定義の中にあるそれぞれのネストした表に対応する格納表の各行には、親表の行に入っているそのネストした表のすべてのインスタンスが入ります。

VARRAY

VARRAYのすべての要素は、 1つの列に格納されます。配列のサイズが4000バイト未満の場合、 Oracleはその配列をひとまとめにして格納します。 4000バイトより大きい場合は、BLOBに格納します。

オブジェクト属性のプロパティ

Oracleでは、オブジェクト属性のいくつかのプロパティを指定できます。

- NULL
- デフォルト
- 制約
- 索引
- トリガー

NULL

表の列またはオブジェクト、オブジェクト属性、コレクション、コレクション要素の特性の 1 つは、それを NULL にすることができるという点です。つまり、その項目を NULL に初期化したり、未初期化状態のままにしたりできます。通常、そのことは、その項目の値は今は不明だが、後で使用可能になるということを意味します。

値が NULL であるオブジェクトのことを「アトミック NULL 」といいます。さらに、オブジェクトの属性をNULLにすることもできます。これら2つのNULLの使用方法は異なります。

たとえば、次のように定義される contacts 表を考えてみます。

```
CREATE TYPE external_person AS OBJECT (
  name        VARCHAR2(30),
  phone       VARCHAR2(20) ) ;

CREATE TABLE contacts (
  contact     external_person
  date        DATE ) ;
```

このとき、次の文は、

```
INSERT INTO contacts VALUES (
  external_person (NULL, NULL),
 '24 Jun 1997' );
```

この文とは異なる結果になります。

```
INSERT INTO contacts VALUES (
  NULL,
 '24 Jun 1997' );
```

どちらの場合も、 Oracle は contacts 表に新しい行のためのスペースを割り当てて、 date 列に指定された値を設定します。最初の文の場合、 Oracle は external_person 列にオブジェクトためのスペースを割り当てて、その各属性に NULL を設定します。 2 番目の文の場合は、 external_person 列に NULL を設定し、オブジェクトのためのスペースは割り当てません。

表の行は NULL にすることができません。このため、 Oracle は行オブジェクトを NULL にすることを許可しません。同様に、オブジェクトのネストした表にも、 NULL 値の要素を入れることはできません。

ネストした表または配列は、 NULL にすることができます。 NULL コレクションは、空のコレクション、つまり要素が入っていないコレクションとは異なります。

デフォルト

表の列をオブジェクト型またはコレクション型として宣言する場合は、DEFAULT句を含めることができます。DEFAULT句には、その列の値が明示的に指定されなかった場合に使われる値を指定します。DEFAULT句には、そのオブジェクトまたはコレクションのコンストラクタ・メソッドの「リテラル起動」を含める必要があります。

コンストラクタ・メソッドの「リテラル起動」は、リテラル、またはコンストラクタ・メソッドのリテラル起動のどちらかを引数とするコンストラクタ・メソッドの起動として再帰的に定義します。

たとえば、次の文を考えてみます。

```
CREATE TYPE person AS OBJECT (
  id        NUMBER
  name      VARCHAR2(30),
  address   VARCHAR2(30) ) ;

CREATE TYPE people AS TABLE OF person ;
```

このとき、次の文は、ネストした表型 people のコンストラクタ・メソッドのリテラル起動です。

```
people ( person(1, 'John Smith', '5 Cherry Lane'),
         person(2, 'Diane Smith', NULL) )
```

次の例は、コンストラクタ・メソッドのリテラル起動を使ってデフォルトを指定する方法を示しています。

```
CREATE TABLE department (
  d_no    CHAR(5) PRIMARY KEY,
  d_name  CHAR(20),
  d_mgr   person DEFAULT person(1,'John Doe',NULL),
  d_emps  people DEFAULT people() )
  NESTED TABLE d_emps STORE AS d_emps_tab ;
```

people( )の項が、空のpeople表を作るためのコンストラクタ・メソッドのリテラル起動になっていることに注意してください。

制約

他の表に対するのと同様、オブジェクト表に対しても制約を定義できます。

列オブジェクトのリーフ・レベル・スカラー属性に制約を定義できます。ただし、有効範囲なしの REF には定義できません（ 11-8 ページの「有効範囲付き REF 」を参照）。

次に、いくつかの例を示します。

最初の例では、オブジェクト表 person_extent の ssno 列に主キー制約を定義します。

```
CREATE TYPE location (
  building_no NUMBER,
  city        VARCHAR2(40) ) ;

CREATE TYPE person (
  ssno      NUMBER,
  name      VARCHAR2(100),
  address   VARCHAR2(100),
  office    location ) ;

CREATE TABLE person_extent OF person (
  ssno        PRIMARY KEY ) ;
```

次の例の department 表には、前の例で定義したオブジェクト型 location を型とする列があります。この例では、表の列 dept_loc に入れる location オブジェクトのスカラー属性に制約を定義します。

```
CREATE TABLE department (
  deptno      CHAR(5) PRIMARY KEY,
  dept_name   CHAR(20),
  dept_mgr    person,
  dept_loc    location,
  CONSTRAINT  dept_loc_cons1
      UNIQUE (dept_loc.building_no, dept_loc.city),
  CONSTRAINT dept_loc_cons2
       CHECK (dept_loc.city IS NOT NULL) ) ;
```

索引

他の表と同様、オブジェクト表や、ネストした表の列または属性の格納表にも、索引を定義できます。

索引は、列オブジェクトのリーフ・レベル・スカラー属性に対して定義できます。ただし、REF に対しては、REF が有効範囲付きの場合に限って、REF 属性または列に索引を定義できます（11-8 ページの「有効範囲付き REF 」を参照）。

次の例では、オブジェクト列の属性に索引を定義します。

```
CREATE TABLE department (
  deptno     CHAR(5) PRIMARY KEY,
```

```
  dept_name  CHAR(20),
  dept_addr  address ) ;

CREATE INDEX i_dept_addr1
          ON department (dept_addr.city);
```

このコーディングにより、 department の address の city 属性に索引が作成されます。

索引の定義において列名を指定できる位置には、オブジェクト列のスカラー属性を指定することもできます。

トリガー

他の表と同様、オブジェクト表にもトリガーを定義できます。ネストした表の列または属性の格納表にはトリガーを定義できません。

トリガー・アクションを定義するコード内では、コレクション（または LOB ）の値を修正できません。ユーザー定義型に対してトリガーを使う場合の制限事項は、その他には特にありません。

次の例では、前の項で定義した person_extent 表にトリガーを定義します。

```
CREATE TABLE movement (
     ssno        NUMBER,
     old_office  location,
     new_office  location ) ;

CREATE TRIGGER trig1
  BEFORE UPDATE
            OF office
            ON person_extent
   FOR EACH ROW
           WHEN new.office.city = 'REDWOOD SHORES'
   BEGIN
     IF :new.office.building_no = 600 THEN
      INSERT INTO movement (ssno, old_office, new_office)
       VALUES (:old.ssno, :old.office, :new.office) ;
     END IF;
   END;
```

ユーザー定義型の権限とそのメソッド

ユーザー定義データ型の権限は、システム・レベルとスキーマ・オブジェクト・レベルに存在します。

システム権限

Oracle は、ユーザー定義型に対して次のシステム権限を定義しています。

- CREATE TYPE 。自分のスキーマ内にユーザー定義型を作成できます。
- CREATE ANY TYPE 。任意のスキーマ内にユーザー定義型を作成できます。
- ALTER ANY TYPE 。任意のスキーマ内のユーザー定義型を変更できます。
- DROP ANY TYPE 。任意のスキーマ内の、名前を指定した型を削除できます。
- EXECUTE ANY TYPE。任意のスキーマ内の、名前を指定した型を使用したり参照したりできます。

CONNECT ロールおよび 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、上記の権限すべてが含まれています。

スキーマ・オブジェクト権限

ユーザー定義型に適用されるスキーマ・オブジェクト権限は、 EXECUTE だけです。

ユーザー定義型に対するEXECUTE権限があると、その型を次の目的で使うことができます。

- 表を定義する。
- リレーショナル表に列を定義する。
- 名前を指定した型の変数またはパラメータを宣言する。

EXECUTE 権限があれば、コンストラクタなど、その型のメソッドを起動できます。

メソッドの実行とそれに関連する許可は、 PL/SQL ストアド・プロシージャの場合と同じです。

新しい型または表で型を使う

これまで説明した許可の他にも、次の場合には特定の権限が必要です。

- 他のユーザーが作成した型を使って、型または表を作成する。
- 自分が作成した新しい型または表の使用権限を他のユーザーに付与する。

新しい型または表を定義する場合は、EXECUTE ANY TYPEシステム権限か、その定義で使う型に対する EXECUTE オブジェクト権限が必要です。これらの権限は、ロールを通してではなく、明示的に受け取らなくてはなりません。

新しい型または表へのアクセス権限を他のユーザーに付与する場合は、必要な型に対するGRANTオプション付きのEXECUTEオブジェクト権限か、オプションWITH ADMIN OPTION付きのEXECUTE ANY TYPEシステム権限が必要です。これらの権限は、ロールを通してではなく、明示的に受け取らなくてはなりません。

例

CONNECT ロールと RESOURCE ロールに、 user1 および user2 、 user3 という 3 つのユーザーが存在するとします。

user1 は、 user1 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、 user2 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 オプション付きの EXECUTE 権限を持っているからです。

```
GRANT EXECUTE ON type3 TO user3 ;
GRANT SELECT on tab2 TO user3 ;
```

しかし、次の権限付与は正しく実行されません。 user2 は、 user1.type1 に対して GRANT オプション付きの EXECUTE を持っていないからです。

```
GRANT SELECT ON tab1 TO user3;
```

ユーザー 3 は、次のアクションを正しく実行できます。

```
CREATE TYPE type4 AS OBJECT (attr4 user2.type3);
CREATE TABLE tab3 OF type4;
```

型アクセスとオブジェクト・アクセスについての権限

表の使用を規定する権限は、オブジェクト表にも等しく適用されます。

- SELECT 権限があれば、その表からオブジェクトとその属性にアクセスできる。
- UPDATE 権限があれば、その表にあるオブジェクトの属性を変更できる。
- INSERT 権限があれば、その表に新しいオブジェクトを追加できる。
- DELETE 権限があれば、その表からオブジェクトを削除できる。

同様の表および列の権限により、ユーザー定義型の表の列の使用方法が規定されます。

ユーザー定義型のデータを取り出す場合、型情報は必要ありません。しかし、データを解釈する場合は、型情報が必要です。 Oracle は、型情報の要求を受け取ると、要求側にその型に対する EXECUTE 権限があるかどうかを検証してから、要求された情報を提供します。

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_type;
```

さらに、次の 2 つの問合せについて考えます。

```
SELECT VALUE(e) FROM emp e;
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。

しかし、 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ので、 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って、 user3 は、次の問合せを実行できます。

```
SELECT tab1.col1.attr2 from user2.tab1 tab1;
SELECT t.attr4.attr3.attr2 FROM tab3 t;
```

基礎となる型に対する権限を user3 は持っていないものの、 user3 によるどちらの SELECT 文も成功します。それらの型および表の所有者が GRANT オプション付きの必要な権限を持っているからです。

Oracle は、次の要求に対する権限をチェックし、要求側がそのアクションに必要な権限を持っていないとエラーを戻します。

- オブジェクトの REF 値を使ってオブジェクト・キャッシュ内でオブジェクトをピンしようとすると、 Oracle は、そのオブジェクトを収容しているオブジェクト表に対する SELECT 権限をチェックする。
- 既存のオブジェクトを修正したり、オブジェクト・キャッシュからオブジェクトをフラッシュしたりしようとすると、 Oracle は目的のオブジェクト表に対する UPDATE 権限をチェックする。新しいオブジェクトをフラッシュしようとすると、 Oracle は目的のオブジェクト表に対する INSERT 権限をチェックする。

- オブジェクトを削除しようとすると、 Oracle は目的の表に対する DELETE 権限をチェックする。名前付きの型のオブジェクトをピンしようとすると、 Oracle はそのオブジェクト型に対する EXECUTE 権限をチェックする。
- メソッドを起動すると、 Oracle は対応するオブジェクト型に対する EXECUTE 権限をチェックする。

Oracle は、オブジェクト表に対する列レベルの権限は提供していません。

依存性と不完全な型

型定義は、相互に依存し合うことがあります。たとえば、オブジェクト型employee（従業員）および department （部門）を定義するときに、 employee の 1 つの属性をその従業員が所属する部門にし、 department の 1 つの属性をその部門を管理する従業員にする場合を考えます。

このように、直接であれ中間の型を介してであれ、互いに依存し合っている型のことを「相互に依存する型」といいいます。相互に依存する型のダイアグラムでは、依存性が矢印によって表され、矢印のパスは必ずどちらかの型のところで開始し終了しています。

Oracle で、このように循環する依存性が許可されるのは、そのサイクルの少なくとも 1 つのブランチで REF が使われている場合だけです。

たとえば、次のような型を定義できます。

```
CREATE TYPE department;

CREATE TYPE employee AS OBJECT (
  name VARCHAR2(30),
  dept REF department,
  supv REF employee ) ;

CREATE TYPE emp_list AS TABLE OF employee;

CREATE TYPE department AS OBJECT (
  name    VARCHAR2(30),
  mgr     REF employee,
  staff   emp_list ) ;
```

上記の定義は、相互に依存する型の適正な組であり、SQL DDL 文の順序も適正です。これらの定義文は、 Oracle によりエラーなしでコンパイルされます。先頭の文は次のようになっています。

```
CREATE TYPE department;
```

この文は、任意指定です。この文により、コンパイル処理がエラーなしで進行します。この文は、 department を「不完全なオブジェクト型」として設定します。不完全なオブジェクト型への REF はエラーなしでコンパイルされるので、 employee のコンパイルもエラーなしで進行します。

Oracle が最後の文まで到達すると、 department の定義が完了し、 department のすべてのコンポーネントが正しくコンパイルされたので、コンパイル処理はエラーなしで終了します。

もし department を不完全な型として宣言しなかったとすると、 employee のコンパイル時にエラーが起きます。その後、 Oracle は、 employee を不完全なオブジェクト型としてスキーマ・オブジェクトのライブラリに自動追加します。この処理により、emp_listおよびdepartmentの宣言はエラーなしでコンパイルされます。 emp_list と employee が完了した後、 employee が再コンパイルされると、それはエラーなしでコンパイルされ、完全なオブジェクト型になります。

不完全な型を完全にする

不完全なオブジェクト型を宣言したら、その型をオブジェクト型として完全なものにする必要があります。たとえば、その型を表型や配列型に宣言することはできません。代替処置としては、その型を削除するしかありません。

このことは、Oracleによってその型が不完全なオブジェクト型にされた場合（前の項の例で、employee のコンパイルが失敗した場合など）も同じです。

この制限は、不完全なオブジェクト型への REF がスキーマ内のどこにも存在しない場合でも適用されます。 Oracle が従う規則は、ある型に REF ターゲット候補というフラグを立てた後は、その型が削除されるまで、その型は REF ターゲット候補のままであるというものです。

Oracle は、オブジェクト型だけを、 REF ターゲット候補として認識します。

表の型の依存性

アクセス時に型定義に依存するデータが表に入っている場合、その型を変更すると、その表のデータにアクセスできなくなります。その型に必要な権限が取り消されたり、型またはその型が依存する型が削除されたりすると、そのような状況になります。その場合、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。

依存する型が削除されたために無効になった表へのアクセスは、二度と回復できません。許可される唯一のアクションは、その表の削除だけです。

SQL コマンド REVOKE および DROP TYPE に指定した型が、その型に依存する表またはその他の型を持つものであると、エラーが出て異常終了します。

どちらのコマンドの場合も、FORCEオプションを付ければ、この動作が上書きされます。コマンドは正しく実行され、影響を受けた表または型は無効になります。

ユーザー定義型の Import/Export

Export ユーティリティと Import ユーティリティは、 Oracle データベースと外部との間でデータを移動します。 また、データのバックアップをとったりアーカイブしたりし、 Oracle RDBMS の異なるリリースへの移行を支援することもできます。

Export と Import では、ユーザー定義型がサポートされています。 Export は、ユーザー定義型の定義および外部ファンクション・ライブラリの定義、ディレクトリ別名の定義、関連するすべてのデータを、ダンプ・ファイルに書き込みます。 Import は、ダンプ・ファイルからそれらの項目を再作成します。

追加情報 : Export と Import の詳細は、『 Oracle8 Server ユーティリティ』を参照してください。

13

オブジェクト・ビュー

The choice of a point of view is the initial act of a culture.

José; Ortega y Gasset, *The Modern Theme*

この章では、オブジェクト・ビューについて説明します。次の内容を取り上げます。

- 基礎知識
- オブジェクト・ビューの定義
- オブジェクト・ビューの使用方法
- オブジェクト・ビューの更新

注意：この章で説明している機能は、オブジェクト・オプション付きのOracle8 Enterprise Editionを購入した場合にのみ利用できます。この章で使われている Oracle Server という用語は、オブジェクト・オプション付きのOracle8 Enterprise Editionを指します。Oracle8 Enterprise Editionで使用可能な機能とオプションの詳細は、『Oracle8 と Oracle8 Enterprise Editionの解説』を参照してください。

基礎知識

ビューが仮想表であるのと同じように、「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・ビューを使うと、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（ビルトイン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に対する専用化された、または制限された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・ビューを使って、機密データの入っている属性がなく、削除メソッドがないバージョンの従業員オブジェクト表を提供できます。

オブジェクト・ビューにより、オブジェクト指向アプリケーションでリレーショナル・データを使えるようになります。ユーザーは、オブジェクト・ビューを使っ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る。
- リレーショナル表からオブジェクト指向表へデータを漸進的かつ透過的に変換する。
- 既存のオブジェクト指向アプリケーションで従来の RDBMS データを使う。

オブジェクト・ビューの利点

オブジェクト・ビューを使うと、パフォーマンスが改善されることがあります。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行き来するので、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することができ、 C または C++ の構造体にマップできるので、 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、従来のデータを徐々に移行していくための手段になります。

オブジェクト・ビューを使うと、リレーショナル・アプリケーションとオブジェクト指向アプリケーションを共存させることができます。それで、あるパラダイムを別のパラダイムに徹底的に切替えなくても、既存のリレーショナル・データにオブジェクト指向アプリケーションを簡単に適用できます。

オブジェクト・ビューには、同じリレーショナル・データまたはオブジェクト・データを複数の方法で表示できるという柔軟性もあります。それで、データベースにデータを格納する方法を変えないでも、異なるアプリケーションのために異なるメモリー内オブジェクト表現を使うことができます。

オブジェクト・ビューの定義

概念としては、オブジェクト・ビューを定義する手順は簡単です。

1. オブジェクト・ビューの行によって表現するオブジェクト型を定義する。
2. どのリレーショナル表のどのデータに、その型のオブジェクトの属性が入っているかを指定する問合せを作成する。
3. 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、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する。

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生成する一意のオブジェクト識別子と対応しています。しかし、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何か一意の値（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います。

複合オブジェクト・ビューを更新できるようにするには、さらに次のステップを実行する必要があります。

4. INSTEAD OFトリガー・プロシージャ（13-4ページの「オブジェクト・ビューの更新」を参照）を作成して、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 がそのプロシージャを実行するようにする。

上記のステップを実行すれば、オブジェクト・ビューをオブジェクト表と同じように使えるようになります。

たとえば、次の SQL 文は、オブジェクト・ビューを定義します。

```
CREATE TABLE emp_table (
    empnum   NUMBER (5),
    ename    VARCHAR2 (20),
    salary   NUMBER (9, 2),
    job      VARCHAR2 (20) ) ;

CREATE TYPE employee_t (
    empno    NUMBER (5),
    ename    VARCHAR2 (20),
    salary   NUMBER (9, 2),
    job      VARCHAR2 (20) ) ;

CREATE VIEW emp_view1 OF employee_t
    WITH OBJECT OID (empno) AS
        SELECT   e.empnum, e.ename, e.salary, e.job
        FROM     emp_table e
        WHERE    job = 'Developer' ;
```

このオブジェクト・ビューは、ユーザーには、基礎を形成する型が employee_t であるオブジェクト表のように見えます。それぞれの行には type employee_t 型のオブジェクトが入っており、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。しかし、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、その結合に含まれるすべての表からキーを提供し、オブジェクト・ビューの行を一意に識別できるようにする必要があります。

注意: WITH OBJECT OID 句の列（例では empno ）は、基礎を形成するオブジェクト型（例では employee_t ）の属性にもなっている必要があります。こうすることにより、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。

オブジェクト・ビューの使用方法

オブジェクト・ビューの行データが 2 つ以上の表から取られている場合もありますが、そのオブジェクトは 1 回の操作でネットワークを通過します。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、プログラマにとってそのインスタンスは C または C++ の構造体、あるいは PL/SQL のオブジェクト変数のように見えます。このインスタンスは、他のネイティブな構造体と同じように処理できます。

SQL 文の中で、オブジェクト・ビューは、オブジェクト表を参照するときと同じ方法で参照できます。たとえば、オブジェクト・ビューは、 SELECT リストまたは UPDATE-SET 句、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うときと同じ OCI コールを使って、クライアント側でオブジェクト・ビューのデータをアクセスできます。たとえば、 REF をピンするときに OCIObjectPin() を使い、オブジェクトをサーバーにフラッシュするときに OCIObjectFlush() を使えます。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。

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

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うのと同じSQL DMLを使って更新および挿入、削除できます。あいまいさがなければ、 Oracle はオブジェクト・ビューの実表を更新します。

ビューの問合せに、結合、集合演算子、グループ・ファンクション、 GROUP BY 、 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、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」（第18章「データベース・トリガー」を参照）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。

追加情報 : INSTEAD OF トリガーを使った発注情報 / 品目の実例は、
『 Oracle8 Server アプリケーション開発者ガイド』を参照してください。

索　引

数字

2 タスク・モード 7-16, 7-17
　説明 7-17
　ネットワーク通信 7-19
　プログラム・インタフェース 7-19
　リスナー・プロセス 7-14
2 フェーズ・コミット
　手動による問題解決 1-25
　説明 1-25, 30-12
　トランザクションの管理 15-7
　トリガー 18-14
　パラレル DML 22-30
3 値論理（ TRUE 、 FALSE 、不明）
　NULL のために生じる 8-7

A

ADMIN OPTION
　EXECUTE ANY TYPE 12-10
　システム権限 26-3
　ロール 26-12
ADT 、オブジェクト型を参照
Advanced Networking Option 30-16
　データの暗号化 30-17
AFTER トリガー 18-8
　起動されるタイミング 18-14
　定義 18-8
ALERT ファイル 7-14
　REDO ログ 7-10
ALL 20-14
ALL_ROWS ヒント 20-39
ALL_UPDATABLE_COLUMNS ビュー 8-13
ALL_ ビュー 4-6
ALTER ANY TYPE 権限 12-10
　権限も参照
ALTER INDEX コマンド
　REBUILD PARTITION 9-36
ALTER SESSION コマンド 14-5
　ENABLE PARALLEL DML 22-28
　HASH_JOIN_ENABLED 20-64
　OPTIMIZER_GOAL 20-39
　SET CONSTRAINTS DEFERRED 24-20
　動的パラメータ 5-4
　トランザクション分離レベル 23-7, 23-30
ALTER SYSTEM コマンド 14-5
　動的パラメータ 5-4
ALTER TABLE コマンド
　CACHE 句 6-4
　DEALLOCATE UNUSED 2-12
　EXCHANGE PARTITION 9-9
　SPLITPARTITIONのためのロギングなしモード21-7
　監査 27-7
　制約を施行する 24-20
　制約を使用禁止または使用可能にする 24-20
ALTER USER コマンド
　一時セグメント 2-16
ALTER コマンド 14-4
　パーティションの監査 9-39
ALWAYS_ANTI_JOIN パラメータ 20-70
ALWAYS_SEMI_JOIN パラメータ 20-70
ANALYZE コマンド 14-4
　COMPUTE STATISTICS 句 20-38

ESTIMATE STATISTICS 句 20-38
共有プール 6-10
パーティションの統計 9-11
ヒストグラムの作成 20-7
ANSI SQL 規格
Oracle が準拠 1-3
データ型 10-17
ANSI/ISO SQL 規格 1-3
データ同時実行性 23-2
複合外部キー 24-14
分離レベル 23-10
ANY 20-14
AQ
キュー・モニター・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キュー表 16-4
キュー表のエクスポート 16-8
メッセージ・キューイング 16-2
リモート・データベース 16-7
AQ_ADMINISTRATOR ロール 16-5
AQ_TM_PROCESS パラメータ 16-4, 16-5
ARCHIVELOG モード
アーカイバ・プロセス（ ARCH ） 7-13, 28-15
概要 1-37
全体データベース・バックアップ 28-20
定義 28-15
部分データベース・バックアップ 1-38, 28-21
ARCH バックグラウンド・プロセス 1-18, 7-13
アーカイバ・プロセスも参照
AUDIT コマンド 14-4
ロック 23-27

B

B* ツリー索引 8-19
索引構成表 8-25
ビットマップ索引との対比 8-21, 8-22
BEFORE トリガー 18-8
起動されるタイミング 18-14
定義 18-8
BETWEEN 20-15
BFILE データ型 10-10
BLOB 10-9
BOOLEAN データ型 10-2
BUFFER_POOL_KEEP パラメータ 6-5
BUFFER_POOL_RECYCLE パラメータ 6-5

C

CACHE 句 6-4
CASCADE アクション
DELETE 文 24-15
CHAR データ型 10-3
空白埋め比較方法 10-3
CHECK 制約 24-16
1 つの列に対する複数の制約 24-17
一部が NULL の外部キー 24-14
チェックのメカニズム 24-18
定義 24-16
パーティション・ビュー 9-9
副問合せは指定禁止 24-16
CHOOSE ヒント 20-39
CKPT バックグラウンド・プロセス 1-18, 7-11
CLEANUP_ROLLBACK_ENTRIES パラメータ 22-30
CLOB データ型 10-9
COMMENT コマンド 14-4
COMMIT コマンド 14-5
2 フェーズ・コミット 15-7, 30-12
DDL による暗黙 15-2, 15-4
高速コミット 7-10
トランザクションの終了 15-2, 15-4
パラレル DML での 2 フェーズ・コミット 22-30
COMPLEX_VIEW_MERGING パラメータ 20-23
COMPUTE STATISTICS 句 20-38
CONNECT BY 句
ビュー問合せの最適化 20-23
CONNECT INTERNAL 5-3
CONNECT ロール 26-14
ユーザー定義型 12-10, 12-11
CPU 時間の制限 25-10
CREATE ANY TYPE 権限 12-10
権限も参照
CREATE FUNCTION コマンド 17-15
CREATE INDEX コマンド
一時セグメント 2-15
オブジェクト型 12-9
パラレル化のルール 22-19
ロギングなしモード 21-7
CREATE PACKAGE BODY コマンド 17-10, 17-15
CREATE PACKAGE コマンド
パッケージ名 17-15

例 17-10, 18-9
ロック 23-27
CREATE PROCEDURE コマンド
プロシージャ名 17-15
例 17-6
ロック 23-27
CREATE SYNONYM コマンド
ロック 23-27
CREATE TABLE AS SELECT
ダイレクト・ロード・インサートとの比較 21-2
パラレル化のルール 22-20
領域の断片化 22-24
ロギングなしモード 21-7
CREATE TABLE コマンド
CACHE 句 6-4
監査 27-6, 27-7, 27-9
制約を使用可能または使用禁止にする 24-20
パラレル化 22-22
例
オブジェクト表 11-7, 11-11, 12-2, 12-8
ネストした表 11-11
列オブジェクト 11-5, 12-2
ロック 23-27
CREATE TRIGGER コマンド
コンパイルと格納 18-17
例 18-9, 18-12, 18-16
オブジェクト表 12-9
ロック 23-27
CREATE TYPE 権限 12-10
権限も参照
CREATE TYPE コマンド
VARRAY 11-10
オブジェクト・ビュー 13-3
オブジェクト型 11-4, 12-2, 12-6, 12-7
ネストした表 11-4, 11-10, 12-7
不完全な型 12-13
CREATE USER コマンド
一時セグメント 2-16
CREATE VIEW コマンド
例 18-12
オブジェクト・ビュー 13-3
ロック 23-27
CREATE コマンド 14-4

D

DATE データ型 10-7
デフォルト書式の変更 10-7
パーティション化 9-11, 9-15
日付算術 10-8
ユリウス暦 10-8
DB_BLOCK_BUFFERS パラメータ
システム・グローバル領域のサイズ 6-11
バッファ・キャッシュ 6-4
DB_BLOCK_LRU_LATCHES パラメータ 7-8
DB_BLOCK_MAX_DIRTY_TARGET パラメータ 7-9, 28-5
DB_BLOCK_SIZE パラメータ
システム・グローバル領域のサイズ 6-11
バッファ・キャッシュ 6-4
DB_FILE_MULTIBLOCK_READ_COUNT パラメータ 20-56
コストベース最適化 20-66
DB_FILES パラメータ 6-14
DB_NAME パラメータ 28-18
DB_WRITER_PROCESSES パラメータ 1-17, 7-8
DBA_QUEUE_SCHEDULES ビュー 16-7
DBA_SYNONYMS.SQL スクリプト
使用 4-6
DBA_UPDATABLE_COLUMNS ビュー 8-13
DBA_ ビュー 4-6
DBA ロール 26-14
ユーザー定義型 12-10
DBMS 1-2
一般要件 1-47
オブジェクト・リレーショナル DBMS 11-2
DBMS_AQADM パッケージ 16-4, 16-5
DBMS_AQ パッケージ 16-3
DBMS_JOB パッケージ 7-13
DBMS_LOCK パッケージ 23-38
DBMS_SQL パッケージ 14-18
DDL 文の解析 14-18
DBW*n* バックグラウンド・プロセス 7-8
データベース・ライター・プロセスも参照
DBWR_IO_SLAVES パラメータ 7-9
DDL 1-48, 14-4
データ定義言語も参照
DEFERROR ビュー
競合 31-25
DELETE 権限、オブジェクト表 12-11, 12-13

DELETE コマンド 14-3
外部キー参照 24-15
データ・ブロック内の領域の解放 2-9
トリガー 18-2, 18-6
INSTEAD OF トリガー 18-11
パラレル DELETE 22-16
Digital POLYCENTER Manager on NetView 30-18
DISTINCT 演算子
ビューの最適化 20-23
DISTRIBUTED_TRANSACTIONS パラメータ 7-12
DML 1-48, 14-3
データ操作言語も参照
D*nnn* バックグラウンド・プロセス 1-18, 7-14
ディスパッチャ・プロセスも参照
DROP ANY TYPE 権限 12-10
権限も参照
DROP TABLE コマンド
監査 27-6, 27-7
DROP TYPE コマンド
FORCE オプション 12-14
依存性 12-14
DROP コマンド 14-4
DSS データベース
索引のパーティション化 9-27
スコア表 22-28
ディスクのストライプ化 22-36
パーティション 9-4
パフォーマンス 9-7
パラレル DML 22-27
DUAL 表 4-6

E

Enterprise Manager
ALERT ファイル 7-15
PL/SQL 14-16, 14-17
SGA のサイズの表示 6-12
SQL 文 14-2
アドバンスト・キューイング 16-8
起動 5-5
システム権限の付与 26-3
スキーマ・オブジェクト権限 26-4
チェックポイント統計 7-11
停止 5-7, 5-8
統計モニター 25-12
パッケージの実行 17-6
パラレル回復 28-13
プロシージャの実行 17-4
分散データベース 30-17
ロールの付与 26-12
ロックとラッチのモニター 23-28
ESTIMATE STATISTICS 句 20-38
EXCHANGE PARTITION 9-9
EXECUTE ANY TYPE 権限 12-10
権限も参照
EXECUTE 権限
権限も参照
ユーザー・アクセスの検査 17-16
ユーザー定義型 12-10, 12-12, 12-13
EXECUTE 権限の GRANT オプション 12-10
EXECUTE ユーザー定義型 12-10
EXP_FULL_DATABASE ロール 26-14
EXPLAIN PLAN コマンド 14-3
アクセス・パス 20-44, 20-45, 20-46, 20-47, 20-48, 20-49, 20-50, 20-51, 20-52, 20-53, 20-54, 20-55
スター型変換 20-74
スター問合せ 20-73
Export ユーティリティ
パーティションのメンテナンス操作 9-29
バックアップでの使用方法 28-22
ユーザー定義型 12-15

F

FAST_FULL_SCAN_ENABLED パラメータ 20-41
FIPS 規格 1-3, 14-6
FIRST_ROWS ヒント 20-39
FORCE オプション
オブジェクト型の依存性 12-14
FOREIGN KEY 制約
NULL 24-14
親キー値の変更 24-15
親キー表の更新 24-15
親表の行の削除 24-15
制約チェック 24-18
列の最大数 24-12

G

GRANT ANY PRIVILEGE システム権限 26-3
GRANT コマンド 14-4
ロック 23-27

GROUP BY 句
　ビューの最適化 20-23

H

HASH_AJ ヒント 20-70
HASH_AREA_SIZE パラメータ 20-65
HASH_JOIN_ENABLED パラメータ 20-64
HASH_MULTIBLOCK_IO_COUNT パラメータ 20-65
HASH_SJ ヒント 20-70
HASHKEYS パラメータ 8-35
HI_SHARED_MEMORY_ADDRESS パラメータ 6-12
HIGH_VALUE 列
　USER_TAB_COLUMNS ビュー 20-57
HP OpenView 30-18

I

IBM NetView/6000 30-18
ID 列
　競合の検出 31-25
IMP_FULL_DATABASE ロール 26-14
Import ユーティリティ
　回復での使用方法 28-22
　パーティションのメンテナンス操作 9-29
　ユーザー定義型 12-15
INDEX_FFS ヒント 20-41
INIT.ORA ファイル 5-3, 5-5
INSERT 権限、オブジェクト表の 12-11, 12-12
INSERT コマンド 14-3
　INSERT ... SELECT のパラレル化 22-18
　空きリスト 2-9
　ダイレクト・ロード・インサート 21-2
　トリガー 18-2, 18-6
　　BEFORE トリガー 18-8
　　INSTEAD OF トリガー 18-11, 18-13
　パラレル INSERT の記憶領域 21-8
INSTEAD OF トリガー 18-11
　オブジェクト・ビュー 13-5
INTERNAL 接続 5-3
　監査レコードを生成しない 27-4
INTERSECT 演算子
　ビュー問合せの最適化 20-23
　複合問合せ 20-12
　例 20-36
INVALID 状態 19-3
IN 演算子 20-13
　ビューのマージ 20-24
IN 副問合せ 20-23
IS NULL 述語 8-7
ISO SQL 規格 1-3, 10-17
　複合外部キー 24-14

J

JOB_QUEUE_PROCESSES パラメータ 16-7

L

LCK*n* バックグラウンド・プロセス 1-19, 7-13
　ロック・プロセスも参照
LGWR_IO_SLAVES パラメータ 7-11
LGWR バックグラウンド・プロセス 1-17, 7-9
　ログ・ライター・プロセスも参照
LICENSE_MAX_SESSIONS パラメータ 25-13
LICENSE_SESSIONS_WARNING パラメータ 25-13
LIKE 20-13
LOB データ型 10-9
　BFILE 10-10
　BLOB 10-9
　CLOB および NCLOB 10-9
　デフォルトのロギング・モード 21-7
LOCK TABLE コマンド 14-3
LOCK_SGA_AREAS パラメータ 6-12, 6-16
LOCK_SGA パラメータ 6-12, 6-16
LOG_ARCHIVE_START パラメータ 28-16
LOG_BUFFER パラメータ 6-6
　システム・グローバル領域のサイズ 6-11
LOG_CHECKPOINT_INTERVAL パラメータ 7-8
LOG_CHECKPOINT_TIMEOUT パラメータ 7-8
LOG_FILES パラメータ 6-14
LONG RAW データ型 10-10
　LONG データ型との類似点 10-10
　索引の設定は禁止 10-11
　パーティション化の制限事項 9-11
LONG データ型
　記憶領域 8-7
　自動的に最後の列になる 8-7
　定義 10-4
　パーティション化の制限事項 9-11
LOW_VALUE 列
　USER_TAB_COLUMNS ビュー 20-57
LRU 6-3, 6-4, 7-8
　共有 SQL プール 6-8, 6-10

ディクショナリ・キャッシュ 4-4
ラッチ 7-8

M

MAC 1-33
MAXVALUE
パーティション表とパーティション索引 9-14
MERGE_AJ ヒント 20-70
MERGE_SJ ヒント 20-70
MERGE ヒント 20-23
MINIMUM EXTENT パラメータ 22-24
MINUS 演算子
ビュー問合せの最適化 20-23
複合問合せ 20-12
MLSLABEL データ型 10-15
MOVE PARTITION コマンド
パラレル化のルール 22-19
ロギングなしモード 21-7
MPP
大規模並列処理を参照
MTS_MAX_SERVERS パラメータ 7-23
人工デッドロック 7-24
MTS_SERVERS パラメータ 7-23

N

NCHAR データ型 10-4
NCLOB データ型 10-9
29-5, 29-4
Net8 29-4, 30-4
Advanced Networking Option 30-16
概要 29-4
マルチスレッド・サーバーの要件 7-14, 7-20
NEXT 記憶領域パラメータ
パラレル DML 21-8
NLS
各国語サポートを参照
NLS_DATE_FORMAT パラメータ 10-7
NLS_LANGUAGE パラメータ 9-14
NLS_LANG 環境変数 9-14
NLS_NUMERIC_CHARACTERS パラメータ 10-6
NLS_SORT パラメータ
ORDER BY アクセス・パス 20-53
パーティション化キーに影響を与えない 9-14
NOARCHIVELOG モード 28-15
LOGGING モードとの関係 21-5
回復用のデータベース・バックアップ 28-20
概要 1-37
定義 28-15
NOAUDIT コマンド 14-4
ロック 23-27
NOLOGGING モード
影響を受ける SQL 操作 21-7
ダイレクト・ロード・インサート 21-5
パーティション 9-34
パラレル DDL 22-22, 22-23
NOREVERSE オプション、索引 8-20
NOT 20-15
NOT IN 副問合せ 20-70
NOT NULL 制約
PRIMARY KEY による暗黙 24-11
UNIQUE キー 24-10
制約チェック 24-18
定義 24-6
Novell NetWare Management System 30-18
NULL
NULL 以外の値 8-7, 20-68
UNIQUE キー制約 24-10
UNIQUE キーでの不一致 24-10
アトミック 12-6
オブジェクト型 12-6
外部キー 24-14, 24-15
格納方法 8-7
禁止 24-6
索引 8-7, 8-24
主キーでは使用禁止 24-10
値に変換する 8-7
最適化 20-68
定義 8-7
デフォルト値 8-8
パーティション表とパーティション索引 9-14
比較では不明扱い 8-7
列の順序 8-7
NUM_DISTINCT 列
USER_TAB_COLUMNS ビュー 20-56
NUM_ROWS 列
USER_TABLES ビュー 20-56
NUMBER データ型 10-5
内部形式 10-6
丸め 10-6
NVARCHAR2 データ型 10-4

NVL 関数 8-7

O

OCI 7-27
OCIObjectFlush 13-4
OCIObjectPin 13-4
オブジェクト・キャッシュ 11-12
ストアド・プロシージャ 14-17
バインド変数 14-12
無名ブロック 14-16
OID 11-8, 12-4, 13-3, 13-4
WITH OBJECT OID 句 13-3, 13-4
OLTP データベース 9-4
索引のパーティション化 9-27
パーティション 9-5
バッチ・ジョブ 22-28
パラレル DML 22-27
OPEN_CURSORS パラメータ 14-6
プライベート SQL 領域の管理 6-9
OPEN_LINKS パラメータ 6-14
OPTIMAL 記憶領域パラメータ 2-22
OPTIMIZER_GOAL オプション 20-39
OPTIMIZER_MODE 20-38
影響を与えるヒント 20-39
Oracle
Oracle Server 1-4
Parallel Server オプション 1-7
Parallel Server も参照
SQL 処理 14-8
Trusted Oracle 1-33
アーキテクチャ 1-8, 1-12
移植性 1-4
インスタンス 1-6, 1-15, 5-2
拡張性 29-4
機能 1-2
クライアント / サーバー・アーキテクチャ 29-2
構成 7-2, 7-15
シングル・プロセス Oracle 7-2
マルチ・プロセス Oracle 7-3, 7-15
互換性 1-4
準拠する規格 1-3
整合性制約 24-4
接続性 1-4
異なる Oracle バージョン 30-7
データ・アクセス 1-47
動作例 1-19, 7-24
ネットワークでの使用 1-4, 1-26
プロセス 1-16, 7-5
ライセンス 25-12
Oracle AQ 16-1
キュー・モニター・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キュー表 16-4
キュー表のエクスポート 16-8
メッセージ・キューイング 16-2
リモート・データベース 16-7
Oracle Enterprise Manager
Enterprise Manager を参照
Oracle Forms
PL/SQL 14-15
オブジェクト依存性 19-11
Oracle Names
グローバルなディレクトリ・サービス 30-4
Oracle Parallel Server 1-7
Parallel Server も参照
Oracle Replication Manager 31-17
Oracle Security Server 30-16
Oracle Server 1-4
Oracle も参照
Oracle Type Translator （ OTT ） 11-13
Oracle オープン・ゲートウェイ 30-8
Oracle コード 7-2, 7-26
Oracle コール・インタフェース（ OCI ） 7-27
OCIObjectFlush 13-4
OCIObjectPin 13-4
オブジェクト・キャッシュ 11-12
ストアド・プロシージャ 14-17
バインド変数 14-12
無名ブロック 14-16
Oracle プリコンパイラ
FIPS フラガー 14-6
埋込み SQL 14-5
カーソル 14-10
ストアド・プロシージャ 14-17
バインド変数 14-12
無名ブロック 14-16
Oracle プログラム・インタフェース（ OPI ） 7-27
Oracle ブロック 1-9, 2-2
データ・ブロックも参照

ORDBMS 1-40, 11-2
ORDERED ヒント 20-66
OTT 11-13

P

Parallel Server 1-7
DML ロックとパフォーマンス 9-29
PCM ロック 23-18
インスタンス・グループ 22-14
逆キー索引 8-20
共有モード 5-6
ロールバック・セグメント 2-24
システム・モニター・プロセス 7-12
システム変更番号 7-10
ディスクの親和性 22-36
データベースとインスタンス 5-2
データベースのマウント 5-6
同時実行の制限 25-13
名前付きユーザー・ライセンス 25-14
排他モード 5-6
ロールバック・セグメント 2-24
パラレル SQL 22-1
ファイルとログの管理ロック 23-29
分散ロック 23-18
分離レベル 23-11
ロック・プロセス 1-19, 7-13
PARALLEL_DEFAULT_MAX_SCANS パラメータ（廃止） 22-14
PARALLEL_DEFAULT_SCANSIZE パラメータ（廃止） 22-14
PARALLEL_INDEX ヒント 22-12
PARALLEL_MAX_SERVERS パラメータ 22-7
PARALLEL_MIN_PERCENT パラメータ 22-14
PARALLEL_MIN_SERVERS パラメータ 22-6, 22-8
PARALLEL_SERVER_IDLE_TIME パラメータ 22-8
PARALLEL 句
パラレル化ルール 22-15
PARALLEL ヒント 22-12
UPDATE と DELETE 22-16
パラレル化ルール 22-15
PCTFREE 記憶領域パラメータ
PCTUSED 2-7
その仕組み 2-5
PCTINCREASE 記憶領域パラメータ
パラレル DML 21-8
PCTUSED 記憶領域パラメータ
PCTFREE 2-7
その仕組み 2-6
PGA 1-16, 6-12
マルチスレッド・サーバー 7-23
PL/SQL 14-14
DDL 文の解析 14-18
PL/SQL エンジン 14-14, 17-2
コンパイラ 17-15
製品 14-15
プロシージャの実行 17-17
オブジェクト・ビュー 13-4
オプティマイザの目標 20-39
解析ロック 23-27
外部プロシージャ 14-18, 17-9
概要 1-50, 14-14
言語要素 14-16
実行 14-14, 17-16, 17-17
ストアド・プロシージャ 1-42, 14-14, 17-2, 17-6
データ型 10-2
データベース・トリガー 1-53, 18-1
動的 SQL 14-18
名前付き PL/SQL ブロックで使用禁止にされたロール 26-13
バインド変数
ユーザー定義型 11-12
パッケージ 17-4, 17-9
プログラム・ユニット 1-42, 6-9, 14-14, 17-2
共有 SQL 領域 6-9
コンパイル済み 14-15, 17-8, 17-15
文の監査 27-4
無名ブロック 14-14, 17-9
ユーザー・ロック 23-38
ユーザー定義データ型 11-12
例外処理 14-17
PMON バックグラウンド・プロセス 1-18, 7-12
プロセス・モニター・プロセスも参照
PRIMARY KEY 制約 24-10
暗黙の NOT NULL 制約 24-11
施行に索引が使われる 24-11
名前 24-11
制約チェック 24-18
説明 24-10
列の最大数 24-11
Pro*C/C++

SQL 文の処理 14-10
ユーザー定義データ型 11-12
PUBLIC ユーザー・グループ 25-8, 26-13
プロシージャの有効性 17-17
PUSH_JOIN_PREDICATE パラメータ 20-69
PUSH_JOIN_PRED ヒント 20-69
P コード 17-16

Q

QMN*n* バックグラウンド・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5

R

RAW データ型 10-10
RDBMS 1-40
Oracle も参照
オブジェクト・リレーショナル DBMS 1-40, 11-2
REBUILD INDEX PARTITION コマンド 9-36
パラレル化のルール 22-19
ロギングなしモード 21-7
REBUILD INDEX コマンド
パラレル化のルール 22-19
ロギングなしモード 21-7
Recovery Manager 1-39, 28-10
カタログを使わない操作 28-11
パラレル操作 28-11
リカバリ・カタログ 28-10
レポートの生成 28-12
REDO ログ・バッファ 1-15, 6-5
書込み 7-9
サイズ 6-6
循環 7-9
トランザクションのコミット 7-10
ログ・ライター・プロセス 6-6
REDO ログ・ファイル 1-11, 28-7
アーカイバ・プロセス（ ARCH ） 7-13
アーカイブ済み 1-37, 28-15
アーカイブ時のエラー 28-17
自動 28-16
手動 28-17
一時セグメントが関係する場合 2-16
オンラインまたはオフライン 1-36, 1-37, 28-7
回復 28-7
概要 1-11, 1-36
制御ファイルに名前がある 28-18
多重化 1-37
用途 1-11
トランザクションのコミット前の書込み 7-10
バッファ管理 7-9
パラレル回復 28-12
物理データベース構造 1-5
モード 1-37
ロールフォワード 28-8
ログ・ライター・プロセス 7-9
ログ順序番号 1-36
制御ファイルに記録される 28-18
REF 11-8
REF ターゲット 12-14
REF のターゲット候補 12-14
暗黙的な解除 11-8
オブジェクト・ビューの行 13-3
オブジェクト識別子から組み立てる 12-4, 12-5
サイズ 12-5
索引 12-8
参照解除 11-8
相互に依存する型 12-13
表の別名の使用 12-2
ピン 12-12, 13-4
ぶら下がり 11-8
有効範囲付き 11-8, 12-5
REFERENCES 権限
ロールを介して付与された場合 26-14
REF ターゲット 12-14
REF も参照
REMOTE_DEPENDENCIES_MODE パラメータ 19-9
RENAME コマンド 14-4
Replication Manager 31-17
RESOURCE ロール 26-14
ユーザー定義型 12-10, 12-11
RESTRICTED SESSION 権限 25-13
REVERSE オプション、索引 8-20
REVOKE コマンド 14-4
FORCE オプション 12-14
オブジェクト型と依存性 12-14
ロック 23-27
ROLLBACK コマンド 14-5
ROWID 8-7
Oracle 以外のデータベース 10-14
REF 12-5

アクセス 10-11
クラスタ化された行 8-7
索引のソートでの使用 8-20
内部使用 10-14
表アクセス 20-40
変更 10-11
ROWID スナップショット 31-11
ROWID データ型 10-11
拡張 ROWID 形式 10-11
制限付き ROWID 形式 10-12
ROWLABEL 列 10-15
ROWNUM 疑似列
索引を使えない 20-54
ビュー問合せの最適化 20-23, 20-32
RPC 30-11, 31-19
RULE ヒント
OPTIMIZER_MODE 20-39

S

SAVEPOINT コマンド 14-5
SCN 15-5
システム変更番号も参照
SELECT 権限、オブジェクト表の 12-11, 12-12
SELECT コマンド 14-3
問合せも参照
副問合せ 14-11
Server Manager
ALERT ファイル 7-15
PL/SQL 14-16, 14-17
SGA のサイズの表示 6-12
SQL 文 14-2
セッション変数 14-16
統計モニター 25-12
パッケージの実行 17-6
プロシージャの実行 17-4
ロックとラッチのモニター 23-28
SESSION_ROLES ビュー
PL/SQL ブロックからの問合せ 26-13
SET CONSTRAINTS コマンド
DEFERRABLE または IMMEDIATE 24-19
SET ROLE コマンド 14-5
SET TRANSACTION コマンド 14-5
ISOLATION LEVEL 23-7, 23-30
READ ONLY 2-17
SGA
システム・グローバル領域を参照
SHARED_MEMORY_ADDRESS パラメータ 6-12
SHARED_POOL_SIZE パラメータ 6-6
システム・グローバル領域のサイズ 6-11
SHUTDOWN ABORT コマンド 5-8
SMON バックグラウンド・プロセス 1-18, 7-12
システム・モニター・プロセスも参照
SMP アーキテクチャ
ディスクの親和性 22-37
S*nnn* バックグラウンド・プロセス 7-14
SNP*n* バックグラウンド・プロセス 1-19, 7-13
自動スナップショット・リフレッシュ 31-9, 31-10
メッセージの波及 16-7
SOME 20-14
SORT_AREA_RETAINED_SIZE パラメータ 6-15
SORT_AREA_SIZE パラメータ 2-15, 6-15
コストベース最適化 20-66
SORT_DIRECT_WRITES パラメータ 6-15
Spatial Data アプリケーション
索引構成表 8-28
SPLIT PARTITION コマンド
パラレル化のルール 22-19
ロギングなしモード 21-7
SQL 14-2
PL/SQL 1-50, 14-14
埋込み 1-48, 14-5
ユーザー定義データ型 11-12
カーソル 14-6
解析 14-7
概要 1-47, 14-2
関数
COUNT 8-24
NVL 8-7
ビュー問合せの最適化 20-23, 20-30
列のデフォルト値 8-8
共有 SQL 14-7
再帰 14-6
カーソル 14-6
システム制御文 14-5
セッション制御文 14-5
データ操作言語（DML） 14-3
データ定義言語（DDL） 14-4
動的 SQL 14-18
トランザクション 1-48, 15-2, 15-5
トランザクション制御文 14-5

パラレル実行 22-2
ファンクション 14-2
CHECK 制約 24-16
文のタイプ 1-47, 14-3
最適化 20-11
文レベルのロールバック 15-4
メモリー割当て 6-10
ユーザー定義データ型 11-11, 12-2
OCI 11-13
埋込み SQL 11-12
予約語 14-2
SQL*Loader
ダイレクト・ローダー 21-2
パーティション操作 9-29, 9-31
SQL*Menu
PL/SQL 14-15
SQL*Module
FIPS フラガー 14-6
ストアド・プロシージャ 14-17
SQL*Net

SQL*Plus
SQL 文 14-2
ストアド・プロシージャ 14-17
接続 25-3
無名ブロック 14-16
SQL_TRACE パラメータ 7-15
SQL92 23-2
SQL 文 1-47, 14-2, 14-8
1 つの SQL 文で起動されるトリガーの数 18-14
依存オブジェクトの参照 19-4
埋込み 14-5
カーソルの作成 14-10
解析 14-11
解析ロック 23-27
概要 1-47
監査 27-6, 27-9
概要 1-32
レコードが生成される場合 27-4
再帰
OPTIMIZER_GOAL は影響を与えない 20-39
最適化
複合文 20-20
文のタイプ 20-11
実行 14-8, 14-13
実行計画 20-2
障害 28-2
正常な実行 15-3
タイプ 1-47, 14-3, 20-11
単純 20-11
ディクショナリ・キャッシュ・ロック 23-28
トランザクション 14-13
トリガー 18-2, 18-8
トリガー・イベント 18-6
配列処理 14-13
パラレル化 22-2, 22-8
パラレル問合せ 22-2
ハンドル 1-16
必要な権限 26-3
複合 20-12, 20-20
最適化 20-20
分散
最適化 20-37
定義 20-12
ノードへのルーティング 14-11
分散データベース 30-10
変換
例 20-17
リソース制限 25-10
SQL 文のハンドル 1-16, 6-9
SQL 領域
共有 1-15, 6-8, 14-7
プライベート 6-8
持続 6-8
実行時 6-8
STAR_TRANSFORMATION_ENABLEDパラメータ 20-75
STAR_TRANSFORMATION ヒント 20-75
STAR ヒント 20-72
STORAGE 句
使用 2-11
パラレル問合せ 22-24
SunSoft
SunNet Manager 30-18
SYS.AUD$ ビュー
削除 4-5
SYSDBA 権限 5-3
SYSOPER 権限 5-3
SYSTEM 表領域 3-4
いつもオンラインでなければならない 3-7

格納されているデータ・ディクショナリ 4-5, 4-2
格納されるプロシージャ 17-16
メディア障害 28-6
SYSTEM ユーザー名
セキュリティ・ドメイン 25-3
SYSTEM ロールバック・セグメント 2-22
SYS ユーザー名
監査レコードを生成しない 27-4
セキュリティ・ドメイン 25-3
データ・ディクショナリ表を所有する 4-3

T

TO_DATE 関数 10-7
パーティションの切詰め 9-11, 9-15
TRANSACTIONS_PER_ROLLBACK_SEGMENT パラメータ 2-23
TRANSACTIONS パラメータ 2-23
TRUNCATE コマンド 14-4
5-3
Trusted Oracle
MLSLABEL データ型 10-15
説明 1-33
必須アクセス制御 1-33

U

UNION ALL 演算子
OR からの変換 20-17
ビュー問合せの最適化 20-23
例 20-18, 20-20, 20-34
UNION ALL ビュー 9-9
UNION 演算子
ビュー問合せの最適化 20-23
複合問合せ 20-12
例 20-25, 20-35
UNIQUE キー制約 24-7
NOT NULL 制約 24-10
NULL 24-10
サイズの制限 24-9
施行に索引が使われる 24-9
制約チェック 24-18
複合キー 24-8, 24-10
列の最大数 24-9
UPDATE No Action 制約 24-15
UPDATE 権限、オブジェクト表の 12-11, 12-12
UPDATE コマンド 14-3
外部キー参照 24-15
データ・ブロック内の領域の解放 2-9
トリガー 18-2, 18-6
BEFORE トリガー 18-8
INSTEAD OF トリガー 18-11
パラレル UPDATE 22-16
USE_INDIRECT_DATA_BUFFERS パラメータ 6-12
USER_TAB_COLUMNS ビュー 20-56, 20-57
USER_TABLES ビュー 20-56
USER_UPDATABLE_COLUMNS ビュー 8-13
USER_ ビュー 4-5
USER 疑似列 26-6

V

V_$ ビューと V$ ビュー 4-6
V$LICENSE 25-13
VARCHAR2 データ型 10-3
RAW データ型との類似点 10-10
非空白埋め比較方法 10-3
VARCHAR データ型 10-3
VARRAY 11-9
VLDB
パーティション 9-4
パラレル SQL 22-2

W

WITH OBJECT OID 句 13-3, 13-4

あ

アーカイバ・プロセス（ARCH）
自動アーカイブ 28-16
説明 7-13
定義 1-18
例 28-15
アーカイブ済み REDO ログ 1-37
自動アーカイブ 28-16
手動アーカイブ 28-17
使用可能にする 28-15
アーキテクチャ
MPP 22-37
Oracle 1-12
SMP 22-37
クライアント / サーバー 1-23
空きリスト 2-9

空き領域（データ・ブロックのセクション） 2-5
アクセス・パス
ROWID による単一行アクセス 20-44
一意キーまたは主キーによる単一行アクセス 20-46
クラスタ結合 20-46
クラスタ結合による単一行アクセス 20-44
最適化 20-40
索引クラスタ・キー 20-47
定義 20-4
ハッシュ・クラスタ・キー 20-47
ハッシュ・クラスタ・キー（一意キーの指定付き）による単一行アクセス 20-45
複合索引 20-48
リスト 20-42
アクセス制御 26-2
権限 26-2
任意 1-28
パスワード暗号化 25-4
必須 1-33
ロール 26-10
アクセスを逐次化できません 23-10
アドバンスト・レプリケーション 31-3
RPC 31-19
概要 31-11
行レベルのレプリケーション 31-19
更新可能スナップショット 31-14
混合構成 31-15
順序 31-24
使用方法 31-12
ジョブ・キュー 31-20
遅延トランザクション 31-20
同期伝播 31-27
非同期伝播 31-19
プロシージャ・レプリケーション 31-26
マルチマスター構成 31-13
アドバンスト・キューイング（ Oracle AQ ） 16-1
キュー・モニター・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キュー表 16-4
キュー表のエクスポート 16-8
メッセージ・キューイング 16-2
リモート・データベース 16-7
アトミック NULL 12-6
アプリケーション
Spatial Data アプリケーション 8-28
アプリケーション・トリガーとデータベース・トリガー 18-4
意思決定支援システム（ DSS ） 8-22
パラレル SQL 22-2, 22-22
依存性 19-9
オブジェクト依存性 19-11
オンライン・トランザクション処理（ OLTP ）
逆キー索引 8-20
オンライン分析処理（ OLAP ） 8-28
コードの共有 6-16
索引構成表 8-27
情報検索（ IR ） 8-27
制約の違反を検出できる 24-6
セキュリティの強化 1-31, 24-5
ダイレクト・ロード・インサート 22-28
データ・ウェアハウス 8-21, 20-70
データ・ディクショナリの参照 4-4
データベース・アクセス 7-2
トランザクションの終了 15-5
ネットワーク通信 29-5
パラレル DML 22-27
プログラム・インタフェース 7-26
プロセス 7-2, 7-4
離散トランザクション 15-8
ロール 26-11
暗号化 30-17
暗黙的な参照解除 11-8

い

意思決定支援システム（ DSS ） 9-4
スコア表 22-28
ディスクのストライプ化 22-36
パーティション 9-4
パフォーマンス 9-7, 22-27
パラレル DML 22-27
パラレル SQL 22-2, 22-22, 22-27
ビットマップ索引 8-22
異種間サービス 30-9
異種間分散データベース 30-8
移植性 1-4
依存性
Oracle Forms トリガー 19-11
オブジェクト型の定義 12-13, 12-14

共有プール 19-8
権限 19-6
スキーマ・オブジェクト 19-2
存在しない参照オブジェクト 19-7
存在しない他のオブジェクト 19-7
リモート・オブジェクト 19-8
ローカル 19-9
一意キー 1-52, 1-53, 24-8
検索 20-46
最適化 20-20
複合 24-8, 24-10
一意索引 8-16
一意性競合 31-22
一意でない索引 8-16
一時セグメント 2-14, 2-16
REDO ログに記録されない場合 2-16
エクステントの割当て解除 2-14
削除 2-14
操作 2-15
パラレル DDL 22-24
パラレル INSERT 21-8
表領域 2-14, 2-16
割当て 2-16
割当て制限は無視される 25-8
一時表領域 3-10
位置の透過性 1-24
一貫性、データ
マルチバージョンの一貫性モデル 1-21, 1-50
読込み一貫性も参照
インスタンス 1-6
インスタンス・グループ 22-14
回復 28-4
SMON プロセス 7-12
増分チェックポイント 28-4
データベースのオープン 5-7
概要 1-6
仮想メモリー 6-16
起動 5-5
障害 1-35, 28-3
シングル・プロセス 7-2
図 7-6
制限モード 5-5
説明 5-2
定義 1-15
停止 5-7, 5-8
データベースに対応付ける 5-2, 5-6
データベースの共有 1-7
プロセスの構造 7-2
マルチ・プロセス 7-3, 7-15
メモリー構造 6-2
ロールバック・セグメントの取得 2-23
インスタンス・グループ、パラレル操作 22-14
インターオペレータ並行性 22-10
インダウト・トランザクション 2-21, 5-7
イントラオペレータ並行性 22-10

う

ウェアハウス
データ・ウェアハウスも参照
表データのリフレッシュ 22-27
内側獲得 12-2
埋込み SQL 文 1-48, 14-5

え

永続キューイング 16-2
エクステント
概要 2-10
増分 2-10
定義 2-2
データ・ブロックの集合 2-10
データ・ブロックの割当て 2-11
パラレル DDL 22-24
ロールバック・セグメント内
現行の変更 2-19
ロールバック・セグメントの削除 2-22
ロールバック・セグメントへの割当て
セグメント作成後 2-21
セグメント作成時 2-18
割当て解除
いつ実行されるか 2-12
ロールバック・セグメント 2-22
割当てが行われる方法 2-11
エクステントの割当て解除 2-12
エラー
埋込み SQL 14-5
トレース・ファイルに記録される 7-14

お

応答キュー 7-20

応答時間 20-6
コストベースのアプローチ 20-38
オブジェクト・キャッシュ
OCI 11-13
Pro*C 11-12
オブジェクト・ビュー 13-4
権限 12-12
オブジェクト・ビュー 8-14, 13-1
INSTEAD OF トリガーの使用方法 13-5
オブジェクト識別子 13-3, 13-4
行オブジェクト 11-8
更新 13-4
定義 13-2
変更の問題 18-11
利点 13-2
オブジェクト・リレーショナル DBMS （ORDBMS）
1-40, 11-2
オブジェクト型 1-40, 11-2, 11-3
Oracle Type Translator 11-13
オブジェクト・ビュー 8-14
キャッシュでのロック 11-13
行オブジェクト 11-7
コンストラクタ・メソッド 1-52, 11-5, 12-4
相互に依存する 12-13
属性 11-2, 11-3, 11-4
発注の例 11-2, 11-4
比較メソッド 11-6
表の別名の使用 12-2
不完全 12-14
メソッド 1-51, 11-4
PL/SQL 11-12
発注の例 11-2, 11-5
メソッド・コール 12-3
メッセージ・キューイング 16-5
列オブジェクト 11-7
索引 12-8
オブジェクト型のメソッド 1-51, 11-4
PL/SQL 11-12
空のカッコの使用 12-3
コンストラクタ・メソッド 1-52, 12-4
自己参照的なスタイルのメソッドの起動 11-5
実行権限 12-10
順序メソッド 1-52, 11-6
発注の例 11-2, 11-5
マップ・メソッド 1-52, 11-6
オブジェクト権限 26-3
スキーマ・オブジェクトも参照
オブジェクト識別子 11-8, 13-3
WITH OBJECT OID 句 13-3, 13-4
オブジェクト・ビュー 13-3, 13-4
オブジェクト型 12-4
索引 12-5
行オブジェクト 11-8
オブジェクト表 11-3, 11-7
仮想的なオブジェクト表 13-2
行オブジェクト 11-7
索引 12-8
制約 12-7
トリガー 12-9
オフライン・バックアップ
全体データベース・バックアップ 28-20
オフライン REDO ログ・ファイル 1-37, 28-7
オペレーティング・システム
管理者の権限 5-3
通信ソフトウェア 7-27
認証 25-3
ブロック・サイズ 2-3
ロール 26-14
オンライン・トランザクション処理（OLTP） 9-4
逆キー索引 8-20
オンライン REDO ログ 1-36, 28-7
アーカイブ 28-15, 28-16
制御ファイルに記録される 28-18
多重化 28-6
チェックポイント 28-19
メディア障害 28-6
オンライン分析処理（OLAP）
索引構成表 8-28

か

カーソル
埋込み SQL 14-5
オープン 6-9, 14-6
オブジェクト依存性 19-8
概要 1-16
再帰 14-6
再帰 SQL 14-6
最大数 14-6
作成 14-10
ストアド・プロシージャ 14-16

定義 14-6
プライベート SQL 領域 6-9, 14-6
解析 14-11
DBMS_SQL パッケージ 14-18
SQL 文 14-11, 14-18
埋込み SQL 14-5
解析コール 14-7
解析ロック 14-11, 23-27
実行 14-7
解析ツリー 17-16
共有 SQL 領域 6-8
構築 14-7
データベースへの格納 17-16
外部キー 1-52
一部が NULL 24-15
親キーを使うための権限 26-5
定義 1-53
外部キーのマッチング
完全一致、部分一致、不一致 24-15
回復
Recovery Manager 1-39, 28-10
インスタンス
SMON プロセス 7-12
インスタンスの回復 28-4
異常終了後に必要な回復 5-8
増分チェックポイント 28-5
データベースのオープン 5-7
パラレル DML 22-31
概要 1-33, 28-8
基本的な手順 1-38, 28-9
災害時回復 28-22
図 28-14
推奨事項 28-14
スタンバイ・データベース 28-23
全体データベース・バックアップ 28-20
使われる構造 1-36, 28-6
データベース・バッファ 28-8
パラレル DML 22-30
パラレル回復 28-12
パラレル復旧 28-11
プロセスの回復 7-12, 28-3
分散処理 7-12
分散トランザクション 5-7
文障害 28-3
メディア回復
使用可能または使用禁止 28-14
ディスパッチャ・プロセス 7-24
ロールバック 28-9
ロールフォワード 28-8
外部結合
NULL に対する NULL 以外の値 20-68
定義 20-12
外部プロシージャ 14-18, 17-9
書込みが読込みを阻止するか 23-10
拡張 ROWID 形式 10-11
拡張性
クライアント / サーバー・アーキテクチャ 29-4
バッチ・ジョブ 22-28
パラレル DML 22-27
パラレル SQL 実行 22-2
獲得回避規則 12-2
仮想表 1-41
仮想メモリー 6-16
カタログ、レプリケーション 31-18
カッコ、メソッド・コールでの使用 12-3
各国語サポート (NLS)
CHECK 制約 24-16
NCHAR および NVARCHAR2 データ型 10-4
NCLOB データ型 10-9
キャラクタ・セット 10-3
クライアントとサーバーで個別に選択可能 30-18
パラメータ 5-4
ビュー 8-12
仮読込み 23-3, 23-10
監査 1-32, 27-1
DDL 文 27-6
DML 文 27-6
アクセス別 27-10
必須 27-11
オプションはいつ有効になるか 27-5
監査オプション 27-2
監査証跡 27-3
オペレーティング・システム 27-4, 27-6
データベース 27-3
監査レコード 27-3
権限 27-2, 27-7
失敗した実行 27-9
スキーマ・オブジェクト 27-2, 27-7
成功した実行 27-9
セキュリティ 27-6

セッション別 27-9
禁止 27-11
説明 1-32, 27-2
対象範囲 27-2, 27-8
タイプ 27-2
使うデータ・ディクショナリ 4-5
データベースと OS のユーザー名 25-4
トランザクションに依存しない 27-4
パーティション表とパーティション索引 9-39
文 27-2, 27-6
分散データベース 27-5
ユーザー 27-11
監査証跡
ディクショナリ内のデータの削除 4-5
監視、ユーザー・アクション 1-32, 27-2
関数
SQL 14-2
COUNT 8-24
NVL 8-7
デフォルトの列値 8-8
ビューでの使用 8-12
ビュー問合せの最適化 20-23, 20-30
ハッシュ関数 8-36
完全リフレッシュ 31-8
管理者権限
監査されない 27-4
管理要求 31-18

き

キー
一意 24-7
複合 24-8, 24-10
親 24-12, 24-14
外部キー 24-11, 24-12
キー値 1-53
逆キー索引 8-20
クラスタ 1-44, 8-29
検索 20-45
索引 8-17, 8-20, 24-9, 24-11
参照 1-53, 24-12
主キー 24-10
制約における 1-53
値の最大記憶領域 8-17
定義 24-8
ハッシュ 8-35
記憶域
NULL 8-7
REF 12-5
オブジェクト表 12-4
索引 8-18
索引クラスタ 8-31
索引パーティション 9-27
データ・ファイル 3-11
トリガー 18-2, 18-17
ネストした表 12-5
ハッシュ・クラスタ 8-33
パラレル DDL の断片化 22-24
パラレル INSERT 21-8
ビュー 8-12
表パーティション 9-20
表領域の取消し 25-8
表領域の割当て制限 25-8
ユーザーごとの割当て制限 1-31
ユーザーに対する制限 25-7
論理構造 3-3, 8-2
記憶領域パラメータ
NEXT 21-8
OPTIMAL （ロールバック・セグメント） 2-22
PCTINCREASE 21-8
設定 2-11
規格 1-3
ANSI/ISO 1-3, 24-4, 24-14
分離レベル 23-2, 23-10
FIPS 1-3, 14-6
Oracle が準拠する 1-3
整合性制約 24-4, 24-14
疑似コード 17-16
トリガー 18-17
記述フェーズ、問合せの処理 14-12
疑似列
CHECK 制約が禁止する
LEVEL および ROWNUM 24-16
ROWID 10-11
ROWNUM
索引を使えない 20-54
ビュー問合せの最適化 20-23, 20-32
USER 26-6
ビューの変更 18-12
起動 5-2, 5-5
SGA の割当て 6-2

開始アドレス 6-12
回復 28-4
強制実行 5-5
共有モード 5-6
高速ウォームスタート 28-4
ステップ 5-5
制限モード 5-5
ディスパッチャ・プロセスでは禁止 7-24
排他モード 5-6
基本レプリケーション 31-2, 31-4
使用方法 31-4
逆キー索引 8-20
キャッシュ
オブジェクト・キャッシュ 11-12, 11-13, 12-12
オブジェクト・ビュー 13-4
キャッシュ・ヒット 6-4
キャッシュ・ミス 6-4
共有 SQL 領域 6-6, 6-8
データ・ディクショナリ 4-4, 6-10
位置 6-6
データベース・バッファ 1-15
バッファ・キャッシュ 6-3
複数のバッファ・プール 6-5
バッファの書込み 7-8
プライベート SQL 領域 6-8
ライブラリ・キャッシュ 6-6
キャラクタ・セット
CLOB および NCLOB データ型 10-9
NCHAR および NVARCHAR2 10-4
各国語 5-4
列の長さ 10-3
キュー・モニター・プロセス（QMN*n*） 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キューイング 16-2
キュー・モニター・プロセス 1-19, 7-13, 16-4
間隔統計 16-7
実行の時間枠 16-5
キュー表 16-4, 16-8
キュー表のエクスポート 16-8
リモート・データベース 16-7
キューイングのエージェント 16-4
行 1-41, 8-3
ROWID での表示 10-12, 10-13
ROWID はいつ変更されるか 10-11
ROWID を使う検索 20-40, 20-44
アドレス 8-7
格納形式 8-5
行オブジェクト 11-7
行ソース 20-3
クラスタ化 8-6
ROWID 8-7
サイズ 8-5
索引構成表での行オーバーフロー 8-26
説明 8-3
断片 8-5
定義 1-41
データ・ブロック内での形式 2-4
トリガー 18-7
フェッチ 14-11
ブロックにまたがる連鎖 2-9, 8-5
ヘッダー 8-5
ロック 9-28, 23-10, 23-19, 23-21
行オブジェクト 11-7
行キャッシュ 6-10
競合 31-22
一意性 31-22
解決 31-25
行レベルのレプリケーション 31-25
検出 31-25
更新 31-23
削除 31-23
データ
デッドロック 7-23, 23-16
ロックの段階的拡大が発生しない 23-16
データ・モデル 31-23
プロシージャ・レプリケーション 31-26
列グループ 31-25
レプリケーション 31-22
ロールバック・セグメント 2-18
競合の解決 31-25
行ソース 20-3
行断片 8-5
識別方法 8-7
ヘッダー 8-5
行ディレクトリ 2-4
行データ（データ・ブロックのセクション） 2-5
行トリガー 18-7
トリガーも参照

起動されるタイミング 18-14
行のロック 23-10, 23-19
直列可能トランザクション 23-7
業務ルール
制約を使って施行する 24-1
共有 SQL 領域 6-8, 14-7
ANALYZE コマンド 6-10
SQL のロード 14-11
依存性の管理 6-10
解析ロック 23-27
概要 1-15, 14-7
サイズ 6-8
説明 6-8
プロシージャ、パッケージ、トリガー 6-9
共有グローバル領域（ SGA ） 6-2
システム・グローバル領域も参照
共有サーバー 1-17
管理者権限でセツゾクできない 5-3
共有サーバー・プロセス（ S*nnn* ） 7-14, 7-23
説明 7-23
共有所有権 31-24
共有プール 6-6
ANALYZE コマンド 6-10
依存性の管理 6-10
オブジェクト依存性 19-8
概要 1-15
行キャッシュ 6-10
サイズ 6-6
説明 6-6
フラッシュ 6-11
プロシージャとパッケージ 17-15
割当て 6-10
共有モード 5-6
ロールバック・セグメント 2-24
共有ロック
共有表ロック（ S ） 23-23
行レベル・ロック 23-10, 23-19
行レベルのレプリケーション 31-19
競合の検出 31-25
切詰め、パーティション 9-3, 22-4, 22-36
TO_DATE 書式マスク 9-11, 9-15
索引 9-26
索引パーティション 9-3

く

クライアント / サーバー・アーキテクチャ 29-2
概要 1-23, 29-2
クライアント 1-23
図 29-3
直接接続と間接接続 30-2
プログラム・インタフェース 7-26
分散処理 29-3
分散データベース 30-2
クラスタ
ROWID 8-7
概要 8-28
格納形式 8-31
キー 1-44, 8-29, 8-32
NULL の索引付けに対する影響 8-7
記憶領域パラメータ 8-4
クラスタ化するデータの選択 8-31
結合 8-31, 20-44, 20-63, 20-46
索引 8-16, 8-32
走査 20-47
ハッシュとの対比 8-33
走査 6-4, 20-40, 20-44
結合 20-46
ハッシュ 20-45, 20-47
定義 1-44
ディクショナリ・ロック 23-28
ハッシュ 8-33
記憶領域 8-33
索引との対比 8-33
衝突の解決 8-35
走査 20-41, 20-45, 20-47
領域の割当て 8-37
ルート・ブロック 8-37
パフォーマンスの考慮事項 8-31
パラメータの設定 8-31
クラスタ・キー 1-44, 8-29
クラスタ化コンピュータ・システム
Oracle Parallel Server 5-2
クラスタ結合 20-63
グループ、インスタンス 22-14
グループ・コミット 7-11
グローバル・スキーマ・オブジェクト名 1-46, 30-6
グローバル・データベース名
共有プール 6-11
グローバル索引 9-23, 9-25

パーティションの管理 9-36
クロス結合 20-12

け

計画
OR 演算子 20-18
SQL の実行 14-3, 14-11
結合 20-59, 20-65
スター型変換 20-74
ビューのアクセス 20-26, 20-29, 20-30
ビューの結合 20-32
複合問合せ 20-34, 20-35, 20-36
複合文 20-21
ゲートウェイ 30-8
結合
外部 20-12
NULL に対する NULL 以外の値 20-68
クラスタ 8-31, 20-44, 20-63
検索 20-46
クロス 20-12
最適化 20-66
実行計画 20-59
選択表示結合ビュー 20-22
ソート / マージ 20-61
コストベース最適化 20-66
例 20-52
直積 20-12
定義 20-12
等価結合 20-12
ネストされたループ 20-60
コストベース最適化 20-66
ハッシュ結合 20-64
半結合 20-70
非結合 20-70
非同一レベル結合 20-12
ビュー 1-42, 8-13
ビューにカプセル化 1-42, 8-11
副問合せに変換する 20-20
結合ビュー 8-13
権限
RESTRICTED SESSION 25-13
解析時にチェックされる 14-11
概要 1-29, 26-2
監査の使用方法 1-33, 27-7
管理者
監査されない 27-4
システム 26-2
概要 1-30
付与と取消し 26-2
ユーザー定義型 12-10
スキーマ・オブジェクト 26-3
DML 操作と DDL 操作 26-4
概要 1-30
パッケージ 26-8
付与と取消し 26-4
プロシージャ 26-6
データベースの起動または停止 5-3
トリガー権限 26-7
取消し 26-2, 26-4
オブジェクト依存性 19-6
パーティション表とパーティション索引 9-38
ビュー 26-5
作成 26-5
使用 26-6
付与 1-30, 26-2, 26-4
例 26-8, 26-9
プロシージャ 26-6
作成と変更 26-7
実行 17-16, 26-6
パッケージ内 26-8
ユーザー定義型
ADMIN OPTION付きのEXECUTE ANY TYPE 12-10
ALTER ANY TYPE 12-10
CREATE ANY TYPE 12-10
CREATE TYPE 12-10
DELETE 12-11, 12-13
DROP ANY TYPE 12-10
EXECUTE 12-10, 12-12, 12-13
EXECUTE ANY TYPE 12-10
GRANT オプション付きの EXECUTE 12-10
INSERT 12-11, 12-12
SELECT 12-11, 12-12
UPDATE 12-11, 12-12
システム権限 12-10
使用 12-10, 12-14
ピン時のチェック 12-12
列レベル、オブジェクト表に対する 12-13
ロールによって獲得される 12-10
ロール 26-10

制限 26-13
ロールとしてグループ化 1-30

こ

更新
更新可能性、ビューの 8-13, 18-11
位置の透過性 30-14
オブジェクト・ビュー 13-4
オブジェクト・ビューの更新可能性 13-5
更新可能な結合ビュー 8-13
頻繁に更新が行われる環境 23-8
分散 30-10
リモート 30-10
レプリケートされたデータでの競合 31-23
更新可能スナップショット 31-14
構造化問合せ言語（SQL） 1-47, 14-2
SQL も参照
構造体
スキーマ・オブジェクト 8-2
物理 1-5, 1-10
データ・ファイル 3-11
ロック 23-26
論理 1-5, 1-8
表領域 3-3
高速ウォームスタート 28-4
高速コミット 7-10
高速全索引走査 20-41
高速トランザクション・ロールバック 28-10
高速リフレッシュ 31-8
コール
Oracle コール・インタフェース 7-27
リモート・プロシージャ 30-11
互換性 1-4
コストベース最適化 20-6
統計 20-38
ヒストグラム 20-7
コミット、トランザクション
概要 1-49
グループ・コミット 7-11
高速コミット 7-10
実現 7-10
定義 15-2
パラレル DML 22-30
コミット読込みの分離 23-7
コミット読込み分離 23-6
コレクション 11-9
可変配列（VARRAY） 11-9
ネストした表 11-10
コレクションのメソッド
コンストラクタ・メソッド 1-52
混合構成
アドバンスト・レプリケーション 31-15
コンストラクタ・メソッド 1-52, 11-5, 12-4
リテラル起動 12-7
コンパイル、オブジェクト型 12-13
コンパイル済み PL/SQL 17-15
疑似コード 17-16, 18-17
共有プール 14-15
再コンパイル 17-17
トリガー 18-17
プロシージャ 17-8
利点 17-8
コンパイル済みトリガー 18-17

さ

サーバー 1-23
共有 1-17
クライアント / サーバー・アーキテクチャ 29-2
専用 1-17, 7-17
マルチスレッド・サーバーとの比較 7-19
専用サーバー・アーキテクチャ 7-16
定義 1-23
プロセス 1-16
マルチスレッド 1-17
アーキテクチャ 7-16, 7-19
専用サーバーとの比較 7-19
プロセス 7-14, 7-19, 7-23
サーバー・プロセス 1-16, 7-5
災害時回復 28-22
再帰 SQL
カーソル 14-6
最低使用頻度アルゴリズム（LRU）
共有 SQL プール 6-8, 6-10
全表走査 6-4
ディクショナリ・キャッシュ 4-4
データベース・バッファ 6-3
ラッチ 7-8
最適化 20-2
DISTINCT 20-23
GROUP BY ビュー 20-23

NULL に対する NULL 以外の値 20-68
PL/SQL 20-39
SQL 文のタイプ 20-11
アプローチの選択 20-38
コストベース 20-6, 20-66
アクセス・パスの選択 20-55
ヒストグラム 20-7
リモート・データベース 20-37
例 20-56
索引作成 8-16
式と述語の変換 20-12
実行される操作 20-11
手動 20-39
推移律 20-15
説明 20-2
選択表示結合ビュー 20-22
問合せの選択性 20-56
統計 20-38
パーティション索引 9-26
パーティションの切詰め 9-3
索引 9-26
パーティションの実行計画 9-9, 9-11
パラレル SQL 22-8
半結合 20-70
ヒント 20-39, 20-41
複合ビューのマージ 20-23
分散型の SQL 文 20-37
文へのビューのマージ 20-22
マージなし 20-32
ルールベース 20-10, 20-66
アクセス・パスの選択 20-58
例 20-58
サイト自律性 1-24, 30-15
作業負荷の不整 22-14
索引 1-43, 8-16
B* ツリー構造 8-19
Index Unusable （IU） 9-37
LONG RAW データ型では禁止 10-11
NULL 8-7, 8-24
REF 12-8
ROWID 8-20
位置 8-18
一意 8-16
一意走査 20-41
一意でない索引 8-16
オブジェクト識別子 12-5
オブジェクト列の属性 12-8
概要 1-43, 8-16
格納形式 8-18
キー 8-17
UNIQUE キー制約 24-9
主キー制約 24-11
逆キー索引 8-20
クラスタ 8-32
削除 8-33
走査 20-47
表との対比 8-32
グローバル索引 9-23, 9-36
高速全走査 20-41
最適化 20-17
索引構成表 8-25
作成
既存の索引を使った 8-16
整合性制約の施行 24-9, 24-11
説明 1-43, 8-16
走査 20-41
MAX または MIN 20-52
ORDER BY 20-53
クラスタ・キー 20-47
制限 20-54
単一列 20-48
非有界範囲 20-51
複合 20-48
有界範囲 20-50
ダイレクト・ロード・インサートの後の再作成 21-8
内部構造 8-19
パーティション 9-2, 9-21
パーティション化のガイドライン 9-27
パーティションについての権限 9-38
パーティションの監査 9-39
パーティションの管理 9-35
パーティションの切詰め 9-3
パーティションの再作成 9-36
パーティション表 8-25
パフォーマンス 8-16
パラレル DDL 記憶領域 22-24
パラレルの索引走査 22-5
範囲走査 20-41
ビットマップ索引 8-21, 8-25

NULL 8-7
パラレル問合せおよび DML 8-22
ビューでの使用 8-12
複合 8-17
走査 20-48
ブランチ・ブロック 8-19
文の変換 20-17
ユーザー定義型 12-8
リーフ・ブロック 8-19
連結 8-17
ローカル索引 9-21, 9-35
ロギングなしモード 21-7
索引構成表 8-25
アプリケーション 8-27
キュー表 16-8
行オーバーフロー領域 8-26
利点 8-26
索引セグメント 1-10, 2-15
削除 No Action 制約 24-15
削除カスケード制約 24-15
削除競合 31-23
参照
オブジェクト 19-2
キー 1-53, 24-12
パーティション 9-13
参照解除 11-8
暗黙的 11-8
参照整合性 23-11, 24-11, 24-12
CASCADE 規則 24-3
PRIMARY KEY 制約 24-10
RESTRICT 規則 24-3
SET DEFAULT 規則 24-3
SET NULL 規則 24-3
一部が NULL の外部キー 24-15
自己参照型制約 24-14, 24-17
例 24-17

し

自己参照的なスタイルのメソッドの起動 11-5
システム・グローバル領域（ SGA ） 6-2
REDO ログ・バッファ 6-5, 15-5
いつ割り当てられるか 6-2
概要 1-15, 6-2
共有および書込み可能 6-2
共有プール 6-6
固定 6-3
サイズ 6-11
可変パラメータ 5-4
図解 5-2
データ・ディクショナリのキャッシュ 4-4
データベース・バッファ・キャッシュ 6-3
内容 6-3
マルチスレッド・サーバーでの使用を制限する 25-11
ロールバック・セグメント 15-5
割当て 5-5
システム・モニター・プロセス（ SMON ） 7-12
Parallel Server 7-12, 22-31
一時セグメントのクリーン・アップ 7-12
インスタンスの回復 28-4
定義 1-18, 7-12
パラレル DML インスタンス回復 22-31
パラレル DML システム回復 22-31
ロールバック、トランザクション 28-10
システム権限 26-2
ADMIN OPTION 12-10, 26-3
権限も参照
説明 26-2
付与と取消し 26-2
ユーザー定義型 12-10
システム制御文 1-48, 14-5
システム変更番号（ SCN ）
REDO ログ 7-10
いつ決定されるか 23-5
定義 15-5
トランザクションのコミット 15-5
読込み一貫性 23-5
事前書込み 7-10
持続領域 6-8
実行計画
EXPLAIN PLAN 14-3
OR 演算子 20-18
SQL の解析 14-11
位置 6-8
概要 20-2
結合 20-59, 20-65
参照する 20-4
実行順序 20-5
スター型変換 20-74
パーティションとパーティション・ビュー 9-9, 9-

11
ビューのアクセス 20-26, 20-29, 20-30
ビューの結合 20-32
複合問合せ 20-34, 20-35, 20-36
複合文 20-21
例 20-21
実行時領域 6-8
実表 1-42
ビューも参照
データ・ディクショナリ 4-2
自動リフレッシュ
リフレッシュ・グループ 31-9
リフレッシュ間隔 31-9
シノニム 19-7
オブジェクトから権限を継承する 26-3
概要 1-43
使用方法 8-15
制約が間接的に影響を与える 24-4
説明 8-15
データ・ディクショナリ・ビュー 4-4
パブリック 8-15
プライベート 8-15
シャドウ・プロセス 7-17
シャドウ列グループ 31-25
主キー 1-53, 24-10
検索 20-46
最適化 20-20
定義 24-3
利点 24-10
主キー・スナップショット 31-11
述語
パーティションの切詰め 9-3
索引 9-26
ビュー問合せの最適化 20-22
ビューへの追加 20-25, 20-30
例 20-25, 20-28
手動リフレッシュ 31-10
手動ロック 1-23, 23-29
順序 1-42, 8-14
CHECK 制約が禁止する 24-16
監査 27-7
数値の生成 8-14
数値の長さ 8-14
調整生成 31-24
表からの独立 8-14
順序メソッド 1-52, 11-6
障害 28-2
REDO ログファイルのアーカイブ 28-17
回復も参照
インスタンス 1-35, 28-3
回復 28-4
説明 1-34, 28-2
耐障害性 28-22
提供されている保護対策 28-6
データベース・バッファ 28-8
内部エラー
トレース・ファイルに記録される 7-14
ネットワーク 28-3
文障害とプロセス障害 1-34, 7-12, 28-2
メディア 1-35, 28-5
ユーザー・エラー 1-34, 28-2
使用済みバッファ 6-3
増分チェックポイント 7-9, 28-4
情報検索（ IR ）アプリケーション
索引構成表 8-27
情報の整理統合
アドバンスト・レプリケーション 31-14
初期化パラメータ
ALWAYS_ANTI_JOIN 20-70
ALWAYS_SEMI_JOIN 20-70
AQ_TM_PROCESS 16-4, 16-5
BUFFER_POOL_KEEP 6-5
BUFFER_POOL_RECYCLE 6-5
CLEANUP_ROLLBACK_ENTRIES 22-30
COMPLEX_VIEW_MERGING 20-23
DB_BLOCK_BUFFERS 6-4, 6-11
DB_BLOCK_LRU_LATCHES 7-8
DB_BLOCK_MAX_DIRTY_TARGET 7-9, 28-5
DB_BLOCK_SIZE 6-4, 6-11
DB_FILE_MULTIBLOCK_READ_COUNT 20-56, 20-66
DB_FILES 6-14
DB_NAME 28-18
DB_WRITER_PROCESSES 1-17, 7-8
DBWR_IO_SLAVES 7-9
DISTRIBUTED_TRANSACTIONS 7-12
FAST_FULL_SCAN_ENABLED 20-41
HASH_AREA_SIZE 20-65
HASH_JOIN_ENABLED 20-64
HASH_MULTIBLOCK_IO_COUNT 20-65

HI_SHARED_MEMORY_ADDRESS 6-12
JOB_QUEUE_PROCESSES 16-7
LGWR_IO_SLAVES 7-11
LICENSE_MAX_SESSIONS 25-13
LICENSE_SESSIONS_WARNING 25-13
LOCK_SGA 6-12, 6-16
LOCK_SGA_AREAS 6-12, 6-16
LOG_ARCHIVE_START 28-16
LOG_BUFFER 6-6, 6-11
LOG_CHECKPOINT_INTERVAL 7-8
LOG_CHECKPOINT_TIMEOUT 7-8
LOG_FILES 6-14
MTS_MAX_SERVERS 7-23, 7-24
MTS_SERVERS 7-23
NLS_LANGUAGE 9-14
NLS_NUMERIC_CHARACTERS 10-6
NLS_SORT 9-14
OPEN_CURSORS 6-9, 14-6
OPEN_LINKS 6-14
OPTIMIZER_MODE 20-38
PARALLEL_DEFAULT_MAX_SCANS 22-14
PARALLEL_DEFAULT_SCANSIZE 22-14
PARALLEL_MAX_SERVERS 22-7
PARALLEL_MIN_PERCENT 22-14
PARALLEL_MIN_SERVERS 22-6, 22-8
PARALLEL_SERVERS_IDLE_TIME 22-8
PUSH_JOIN_PREDICATE 20-69
REMOTE_DEPENDENCIES_MODE 19-9
ROLLBACK_SEGMENTS 2-23
SHARED_MEMORY_ADDRESS 6-12
SHARED_POOL_SIZE 6-6, 6-11
SORT_AREA_RETAINED_SIZE 6-15
SORT_AREA_SIZE 2-15, 6-15, 20-66
SORT_DIRECT_WRITES 6-15
SQL_TRACE 7-15
STAR_TRANSFORMATION_ENABLED 20-75
TRANSACTIONS 2-23
TRANSACTIONS_PER_ROLLBACK_SEGMENT 2-23
USE_INDIRECT_DATA_BUFFERS 6-12
初期即時制約 24-19
初期遅延制約 24-19
ジョブ 7-2
ジョブ・キュー 31-20
スナップショットのリフレッシュ 31-10
ジョブ・キュー・プロセス（ SNP*n* ）
自動スナップショット・リフレッシュ 31-9, 1-19, 31-10
メッセージの波及 16-7, 7-13
署名のチェック 19-9
処理
DDL 文 14-13
DML 文 14-10
概要 14-8
問合せ 14-11
パラレル SQL 22-2
分散 1-23
シリアル伝播 31-21
シングル・タスク・モード 7-15
シングル・プロセスのシステム（シングル・ユーザーのシステム） 7-2
シンプル・ネットワーク管理プロトコル（SNMP）のサポート
データベース管理 30-18
親和性
パーティション 22-36
パラレル DML 22-37

す

スキーマ 25-2
オブジェクト 8-2
定義 25-2
内容 8-2
表領域との対比 8-2
ユーザー定義データ型 11-12
ユーザーとの対応付け 1-27, 8-2
スキーマ・オブジェクト 8-1
INVALID 状態 19-3
依存性 19-2
存在しない他のオブジェクト 19-7
トリガー管理 18-14
ビュー 8-13
分散データベース 19-10
失われた権限に依存する 19-6
概要 1-9, 1-41, 8-2
監査 1-33, 27-7
グローバルな名前 30-6
権限 26-3
作成
表領域の割当て制限が必要 25-8

情報 4-2
定義 1-5
データ・ファイルとの関連 3-12, 8-2
デフォルトの表領域 25-7
トリガーの依存性 18-17
取り消した表領域内 25-8
分散データベースにおける名前 30-6
ユーザー定義型 11-3
スキーマ・オブジェクト権限 26-3
DML 操作と DDL 操作 26-4
概要 1-30
ビュー 26-5
付与と取消し 26-4
スキーマ名
データベース内で一意の 30-6
分散データベースにおける 30-6
列名の修飾 12-3
スター型変換 20-72
制限 20-75
例 20-73
スター問合せ 20-72
拡張スター・スキーマ 20-72
索引 20-71
チューニング 20-71
ヒント 20-72
スタック領域 6-13
スタンバイ・データベース 28-22
ストアド・ファンクション 1-43, 17-2, 17-6
ストアド・プロシージャ 1-43, 14-14, 17-2, 17-6
プロシージャも参照
コール 14-17
トリガーとの対比 18-2
変数と定数 14-16
無名ブロックと対比 17-8
スナップショット
ROWID 31-11
グループ 31-18
更新 31-8
更新可能 31-14
サイト 31-18
主キー 31-11
単純
構造 31-7
定義問合せ 31-7
複合 31-10
副問合せ 31-7
読込み専用 31-6
リフレッシュ 7-13, 31-8, 31-10
ログ 31-9
スナップショットの定義問合せ 31-7
スナップショットのリフレッシュ
完全メディア回復 31-8
グループ 31-9
リフレッシュ間隔 31-9
高速リフレッシュ 31-8
ジョブ・キュー・プロセス（ SNP*n* ） 1-19, 7-13
自動スナップショット・リフレッシュ 31-9
スナップショット・ログ 31-9
スナップショット読込み時期 23-10
スループット 20-6
コストベースのアプローチ 20-38
スレッド
マルチスレッド・サーバー 7-14, 7-19

せ

世紀 10-8
制御ファイル 1-12, 28-18
回復 1-37
概要 1-12, 28-18
記録される変更内容 28-18
指定方法 5-4
多重化 1-37, 28-19
チェックポイント 28-19
データベースのマウントでの使用 5-6
内容 28-18
バックアップ 28-21
物理データベース構造 1-5
制限
ダイレクト・ロード・インサート 21-9
パーティション
拡張パーティション表名 9-40
データ型 9-11, 9-15
ビットマップ索引 9-11
パーティション・ビュー 9-10
パラレル DML とダイレクト・ロード・インサート 22-33
制限付き ROWID 形式 10-12
制限モード
インスタンスの起動 5-5
整合性規則 1-40

パラレル DML の制限事項 22-34
整合性制約 24-2
制約も参照
デフォルトの列値 8-8
静的所有権 31-23
制約 1-52
CHECK 24-16
FOREIGN KEY 1-53, 24-11
NOT NULL 24-6, 24-10
PRIMARY KEY 1-53, 24-10
UNIQUE キー 1-53, 24-7
一部が NULL 24-10
アプリケーションが違反を検出できる 24-6
一時的に使用禁止にする 24-6
違反した場合どうなるか 24-4
オブジェクト表 12-7
概要 1-52
索引による施行 8-18
PRIMARY KEY 24-11
UNIQUE 24-9
参照制約
更新の効果 24-15
自己参照型 24-14
施行のメカニズム 24-17
制約を施行する 24-20
制約を使用可能または使用禁止にする 24-20
代替処置 24-5
タイプのリスト 1-53, 24-1
定義 8-4
デフォルト値 24-18
トリガーとの対比 18-5
トリガーは違反できない 18-14
パフォーマンスに対する効果 24-6
パラレル CREATE TABLE 22-19
ビューでは定義できない 8-10
評価されるタイミング 8-8
制約を施行する 24-20
制約を使用可能にする 24-20
制約を使用禁止にする 24-20
西暦 2000 年 10-8
セーブポイント 1-50, 15-7
暗黙的 15-4
概要 1-50
説明 15-7
ロールバック 15-6
セキュリティ 1-30, 25-2
アプリケーションを使った施行 1-31
監査 27-2, 27-6
監査データの削除 4-5
管理者権限 5-3
施行のメカニズム 1-28
システム 1-27, 4-3
説明 1-27
データ 1-27
データの暗号化 30-17
ドメイン 1-29, 25-2
任意アクセス制御 1-28, 25-2
パスワード 25-4
ビュー 8-11
ビューによる強化 26-6
プログラム・インタフェースでの実施 7-26
プロシージャによる強化 26-7
分散データベース 30-15
メッセージ・キュー 16-5
ユーザー・アクションの監査 1-32
セキュリティ・ドメイン 1-29, 25-2
使用可能なロール 26-12
表領域の割当て制限 25-7
セグメント 1-10, 2-14
一時 1-10, 2-15
SMON によるクリーン・アップ 7-12
削除 2-14
操作 2-15
パラレル INSERT 21-8
表領域 2-14, 2-16
割当て 2-15
割当て制限は無視される 25-8
エクステントの割当て解除 2-12
概要 1-10, 2-14
索引 2-15
定義 2-3
データ 2-14
ヘッダー・ブロック 2-10
ロールバック 2-16
セッション
PARALLEL DML を有効にする 22-28
PGA 内のスタック領域 6-13
監査オプションはいつ有効になるか 27-5
時間制限 25-11
情報が格納される場所 6-13

セッション別監査 27-9
接続との比較 7-4
定義 7-5, 27-9
同時セッションに対する制限 1-32
ライセンスによる 25-13
トランザクション分離レベル 23-30
パッケージの状態 19-6
ユーザー当たりの制限 25-11
リソース制限 25-9
セッション制御文 1-48, 14-5
接続
埋込み SQL 14-5
管理者権限での 5-3
制限 5-5
セッションとの比較 7-4
定義 7-4
ユーザー名 25-2
リスナー・プロセス 7-14
接続性 1-4
全索引走査 20-41
全体データベース・バックアップ 1-38, 28-20
選択性、問合せの 20-56
選択表示結合ビュー 20-22
全表走査 20-40, 20-54
LRU アルゴリズム 6-4
選択性 20-56
パラレル問合せ 22-5
マルチブロック読込み 20-56
ルールベースのオプティマイザ 20-58
専用サーバー 7-17
使用例 7-24
定義 1-17
マルチスレッド・サーバー 7-19

そ

走査 20-40
一意 20-41, 20-46, 20-47
クラスタ 20-44, 20-45, 20-46, 20-47
索引 20-47
高速全索引走査 20-41
索引 20-41
MAX または MIN 20-52
ORDER BY 20-53
クラスタ・キー 20-47
制限 20-54
選択性 20-56
単一列 20-48
ビットマップ 20-42
非有界範囲 20-51
複合 20-48
有界範囲 20-50
全表 20-40, 20-54
LRU アルゴリズム 6-4
パラレル問合せ 22-5
マルチブロック読込み 20-56
ルールベースのオプティマイザ 20-58
ハッシュ・クラスタ 20-45, 20-47
範囲 20-41, 20-48
MAX または MIN 20-52
ORDER BY 20-53
非有界 20-51
有界 20-50
表走査と CACHE 句 6-4
増分チェックポイント 7-9, 28-4
ソート / マージ結合 20-61
アクセス・パス 20-52
コストベース最適化 20-66
例 20-52
ソート・ダイレクト書込み機能 6-15
ソート領域 6-15
即時制約 24-19
属性、オブジェクト型の 11-2, 11-3, 11-4
阻止しているトランザクション 23-10
阻止しているトランザクションを待機するか 23-10
ソフトウェア・コード領域 6-16
プログラムとユーティリティによる共有 6-16

た

大規模データベース (VLDB) 9-4
パーティション 9-4
パラレル SQL 22-2
大規模並列処理（ MPP ）
複数の Oracle インスタンス 5-2
耐障害性 28-22
対称型レプリケーション 31-3, 31-12
概要 31-11
使用方法 31-12
代替キー
競合の検出 31-25
タイプ

データ型、オブジェクト型を参照
タイムスタンプのチェック 19-9
大量パラレル処理（MPP）
親和性 22-6, 22-36, 22-37
パラレル SQL 実行 22-2
ダイレクト・ロード・インサート 21-2
シリアル INSERT 21-3
制限 21-9, 22-33
パラレル・ロードとパラレル INSERT の比較 21-2
パラレル INSERT 21-3
領域管理 21-8
ロギング・モード 21-5
多重化
REDO ログ・ファイル 1-37
回復 28-6
制御ファイル 1-37, 28-19
タスク 7-2
単純スナップショット
構造 31-7
ダンプ・ファイル
Export と Import 12-15
断片化
パラレル DDL 22-24

ち

チェックポイント
DBW*n* プロセス 7-11
DBW*n* プロセスへの信号 7-8
制御ファイル 28-19
増分 7-9, 28-4
チェックポイント・プロセス（CKPT） 1-18, 7-11
統計 7-11
チェックポイント・プロセス（CKPT） 1-18, 7-11
遅延トランザクション 31-20
遅延制約
初期遅延または初期即時 24-19
遅延可能または遅延不可 24-19
調整順序生成 31-24
直積 20-12

つ

通信プロトコル 29-5

て

定義フェーズ、問合せの処理の 14-12
ディクショナリ
データ・ディクショナリを参照
ディクショナリ・キャッシュ・ロック 23-28
停止 5-7, 5-8
SGA の割当て解除 6-2
異常 5-5, 5-8
ステップ 5-7
ディスパッチャ・プロセスでは禁止 7-24
定数
計算されるとき 20-13
ストアド・プロシージャ内 14-16
比較 20-13
ディスク障害 1-35, 28-5
ディスクの親和性
パーティション 22-36
パラレル DML 22-37
ディスクのストライプ化
親和性 22-36
パーティション 9-7
ディスク領域
データ・ファイルへの割当て 3-11
表への割当ての制御 8-4
ディスパッチャ・プロセス（D*nnn*）
7-20
Net8 を介したユーザー・プロセスの接続 7-14
応答キュー 7-20
起動と停止の禁止 7-24
セッション当たりの SGA 領域の制限 25-11
説明 7-14
定義 1-18
ネットワーク・プロトコル 7-14
リスナー・プロセス 7-14
データ
アクセス 1-47
制御 25-2
セキュリティ・ドメイン 25-2
メッセージ・キュー 16-5
一貫性
基礎となる原理 23-14
手動ロック 23-29
定義 1-50
トランザクション・レベル 23-6
反復可能読込み 23-6

読込み一貫性 1-21
ロック 23-3
ロック動作の例 23-30
整合性 1-20, 8-4, 24-2
2 フェーズ・コミット 1-25
CHECK 制約 24-16
概要 1-52
参照制約 24-3
施行 24-3, 24-5
タイプ 24-2
パラレル DML の制限事項 22-34
同時アクセス 23-2
表への格納方法 8-4
分散操作 1-24
レプリケーション 1-25, 31-2
ロック 23-19
データ・ウェアハウス 20-70
スター問合せ 20-70
ビットマップ索引 8-21
表データのリフレッシュ 22-27
データ・オブジェクト番号
拡張 ROWID 10-12
データ・セグメント 1-10, 2-14, 8-4
データ・ディクショナリ
DUAL 表 4-6
アクセス 4-2
依存性の追跡 19-3
オブジェクトを追加する 4-4
監査証跡（ SYS.AUD$ ） 4-5
キャッシュ 6-10
位置 6-6
行キャッシュ 6-10
更新 4-4
構造 4-2
最適化で使われるビュー 20-7
使用方法 4-3
表と列の定義 14-11
所有者 4-3
接頭辞が ALL のビュー 4-6
接頭辞が DBA のビュー 4-6
接頭辞が USER のビュー 4-5
定義 1-46, 4-2
統計データ 20-38
パーティションの統計 9-11
動的パフォーマンス表 4-6
内容 4-2, 6-10
プロシージャ 17-16
バックアップ 28-21
パブリック・シノニム 4-4
ビューの接頭辞 4-5
プロシージャの有効性 17-16
ロック 23-26
データ・ディクショナリ・ビューの接頭辞 4-5
データ・ファイル
ROWID での表示 10-12, 10-13
オフラインにする 3-12
オンライン表領域またはオフライン表領域にある 3-12
回復不能 28-12
概要 1-8, 1-10, 3-11
制御ファイルに名前がある 28-18
データ・ファイル 1 内のディクショナリ 28-21
内容 3-11
バックアップ 28-21
パラレル回復 28-13
表領域との関連 3-2
物理データベース構造 1-5
読込み専用 3-9
回復 28-5
読込み専用表領域 3-10
データ・ブロック 1-9, 2-2
ROWID での表示 10-12, 10-13
空きデータ・ブロックの結合 2-12
空きリスト 2-9
空き領域の制御 2-5
エクステントへの割当て 2-11
概要 2-2
行ディレクトリ 8-6
行の格納方法 8-5
行を挿入できる領域 2-8
クラスタ化 8-31
クラスタによる共有 8-28
形式 2-3
ディスクへ書き込む 7-8
ハッシュ・キー 8-37
バッファ・キャッシュに格納される 6-3
メモリーにキャッシュされる 7-8
読込み専用トランザクション 23-30
データ・ブロック内の空き領域の圧縮 2-9
データ・モデル 1-40

データ・ロック
存続期間 23-14
段階的な拡大 23-15
変換 23-15
データ型 10-2, 10-15
ANSI 10-17
BOOLEAN 10-2
CHAR 10-3
DATE 10-7
DB2 10-17
LOB データ型 10-9
BFILE 10-10
BLOB 10-9
CLOB および NCLOB 10-9
デフォルトのロギング・モード 21-7
LONG 10-4
記憶領域 8-7
MLSLABEL 10-15
NCHAR および NVARCHAR2 10-4
NUMBER 10-5
PL/SQL 10-2
RAW および LONG RAW 10-10
REF 11-8
ROWID 10-11
SQL/DS 10-17
VARCHAR 10-3
VARCHAR2 10-3
オブジェクト型 1-40, 11-3
コレクション 11-9
使用可能なデータ型のリスト 10-2
ネストした表 8-9, 11-10
配列型 11-9
表との関連 8-3
変換
Oracle 以外の型 10-17
Oracleデータ型から別のOracleデータ型へ 10-18
プログラム・インタフェース 7-26
マルチメディア 11-3
文字 10-2, 10-9
ユーザー定義 11-1, 11-3
要約 10-15
列 1-41
データ所有権モデル 31-23
共有所有権 31-24
静的所有権 31-23
動的所有権 31-23
プライマリ所有権 31-23
データ操作言語（ DML ）
監査 27-6
権限制御 26-4
取得されるロック 23-24
説明 14-3
定義 1-48
トリガー 18-3, 18-15
パーティション・ロック 9-28
パラレル DML 22-3, 22-26
パラレル DML のためのトランザクション・モデル 22-29
副問合せの直列可能分離 23-13
分散トランザクション 30-10
文の処理 14-10
データ定義言語（ DDL ）
DBMS_SQL での解析 14-18
暗黙のコミット 15-4
監査 27-6
説明 14-4
定義 1-48
パラレル DDL 22-3
文の処理 14-13
ロールと権限 26-13
ロック 23-26
データベース
アーカイブのモード 28-14
アクセス制御
概要 1-47
セキュリティ・ドメイン 25-2
パスワード暗号化 25-4
オープン 5-6
ロールバック・セグメントの取得 2-23
オープンとクローズ 5-2
回復 1-33, 28-2
拡張性 22-2, 22-27, 29-4
起動 5-2
強制実行 5-8
クローズ 5-8
インスタンスの異常終了 5-8
グローバル・データベース名 30-4
構成 5-3
サイズ

判別する方法 3-6
使用の制限 25-9
スキーマが組み込まれている 25-2
スタンバイ 28-22
制御ファイルに格納される名前 28-18
定義 1-7, 1-8
停止 5-7
ディスマウント 5-8
バックアップ 1-38, 28-19
物理構造 1-5, 1-10, 2-2
ROWID を使って明らかにする 10-13
分散 1-24, 30-1
2 フェーズ・コミット 1-25
概要 1-23, 1-24, 30-1
グローバル・データベース名の変更 6-11
サイト自律性 30-15
ノード 1-24, 30-2
表のレプリケーション 1-25
文の最適化 20-37
マウント 5-6
論理構造 1-5
論理構造（オブジェクト） 1-8
データベース・スキーマのオブジェクト 1-5
スキーマ・オブジェクトも参照
データベース・トリガー 1-53, 18-1
トリガーも参照
データベース・バッファ
空き 6-3
書込み 7-8
キャッシュのサイズ 6-4
クリーン 7-8
コミット、トランザクションの 7-10
使用済み 6-3, 7-8
使用中 6-3
定義 1-15, 6-3
トランザクションをコミットした後 15-6
バッファ・キャッシュ 6-3, 7-8
複数のバッファ・プール 6-5
データベース・ライター・プロセス（ DBW*n* ） 7-8
アクティブになる時 7-8
概要 1-17
最低使用頻度アルゴリズム（ LRU ） 7-8
事前書込み 7-10
チェックポイント信号 7-8
チェックポイント時のディスクへの書込み 7-11
定義 7-8
トレース・ファイル 28-6
複数の DBW*n* プロセス 7-8
複数の I/O プロセス 7-9
メディア障害 28-6
データベース・リンク 1-46
概要 30-6
定義 1-46
データベース管理システム（ DBMS ） 1-2
Oracle Server 1-4
オブジェクト・リレーショナル DBMS 11-2
原理 1-40
データベース管理者（ DBA ）
DBA ロール 12-10, 26-14
データ・ディクショナリ・ビュー 4-6
認証 25-6
パスワード・ファイル 25-7
バックアップおよびリカバリに対する責任 28-2
データベースの構成
パラメータ・ファイル 5-3
プロセスの構造 7-2, 7-15
データ変換
ANSI データ型 10-17
SQL/DS および DB2 データ型 10-17
プログラム・インタフェース 7-26
デッドロック
回避 23-18
検出 23-17
人工 7-23
定義 23-16
分散トランザクション 23-17
デフォルト値 8-8
制約が与える影響 8-8, 24-18
ユーザー定義型 12-7
伝播
シリアル 31-21
パラレル 31-21

と

問合せ
DML 14-3
IN 副問合せの最適化 20-23
ROWID による表走査のパラレル化 22-3
一時セグメント 2-15, 14-12
位置の透過性 30-14

記述フェーズ 14-12
行のフェッチ 14-11
処理 14-11
スター問合せ 20-70
選択性 20-56
定義 20-11
定義フェーズ 14-12
デフォルト・ロック 23-25
トリガーの使用方法 18-15
パーティションによってパラレル化される索引走査 22-5
パラレル処理 22-2
非定型 22-22
ビュー問合せとのマージ 8-12
ビュー問合せの最適化 20-22
ビューとして格納 1-41, 8-10
フェーズ 23-5
複合
OR を複合問合せに変換する 20-17
最適化 20-34
定義 20-12
分散またはリモート 30-10
読込み一貫性 1-22, 23-5
同一キー索引 9-22, 9-25
同一行の書込みが書込みを阻止するか 23-10
同一レベル・パーティション化 9-17
等価結合
クラスタ結合 20-63
ソート / マージ 20-61
定義 20-12
ハッシュ結合 20-64
同期データ伝播 31-27
統計
ANALYZE コマンド 20-38
オプティマイザでの使用 20-6, 20-7
キューイング 16-7
チェックポイント 7-11
パーティション表とパーティション索引 9-11
同時実行性
制限 1-32, 21-9
データベース当たりの 25-13
ユーザー当たりの 25-11
説明 23-2
ダイレクト・ロード・インサート 21-9
定義 1-20
トランザクション 23-14
パーティションのメンテナンス 9-31
ロックを使って施行する 1-22
動的 SQL
DBMS_SQL パッケージ 14-18
動的所有権 31-23
動的パーティション化 22-6
動的パフォーマンス表（ V$ 表） 4-6
ドライバ 7-27
トランザクション 1-48, 15-1
アドバンスト・キューイング 16-2
アプリケーションの終了 15-5
インダウト
自動解決 1-25, 5-7, 15-8
手動による解決 1-25
部分的に使用可能なセグメントの使用 2-26
ロールバック・セグメント 2-21
ロールバック・セグメントのアクセスの制限 2-26
開始 15-4
概要 1-48
コミット 1-49, 7-10, 15-4, 15-5
グループ・コミット 7-11
ロールバック・セグメントの使用 2-18
コミット前に書き込まれる REDO ログ・ファイル 7-10
システム変更番号 7-10
システム変更番号を割り当てる 15-5
終了 15-4
一貫したデータ 14-13
手動ロック 23-30
セーブポイント 1-50, 15-7
説明 15-2
直列可能 23-6
定義と制御 14-13
データ・ブロック内で使われる領域 2-5
デッドロック 15-4, 23-16
同時実行性 23-14
トランザクション制御文 14-5
トランザクションの制御 14-13
トリガー 18-15
パラレル DML での 2 フェーズ・コミット 22-30
非同期処理 16-2
分散 1-22
2 フェーズ・コミット 1-25, 15-7, 30-12

自動解決 7-12
デッドロック 23-17
パラレル DML の制限事項 22-35
文レベルのロールバック 15-4
読込み一貫性 1-21, 23-6
読込み専用 1-22, 23-6
ロールバック・セグメントには割り当てられない 2-17
離散トランザクション 14-14, 15-8
ロールバック 1-49, 15-6
オフライン表領域 2-27
部分的な 15-6
ロールバック・セグメントの使用 2-17
ロールバック・セグメント 2-17
ロールバック・セグメントへの書込み 2-18
ロールバック・セグメントへの配分 2-18
ロールバック・セグメントへの割当て 2-17
トランザクション集合の一貫性 23-9, 23-10
トランザクション制御文 1-48, 14-5
トランザクション表 2-17
回復時にリセット 7-12
トリガー 1-53, 18-1, 19-7
AFTER トリガー 18-8
BEFORE トリガー 18-8
INSTEAD OF トリガー 18-11
オブジェクト・ビュー 13-5
INVALID 状態 19-3, 19-6
Oracle Forms トリガー 18-4
UNKNOWN では起動されない 18-7
アクション 18-7
タイミング 18-8
依存性の管理 18-17, 19-6
使用可能なトリガー 18-14
イベント 18-6
概要 1-53, 18-2
各部分 18-5
カスケード 18-3
監査 27-8
記憶領域 18-17
起動（実行） 18-2, 18-17
実行されるステップ 18-13
タイミング 18-14
必要な権限 18-17
行 18-7
共有 SQL 領域 6-9
実行する権限 26-7
使用可能または使用禁止 18-13
使用禁止にされたロール 26-13
使用方法 18-3
スキーマ・オブジェクトの依存性 18-14, 18-17
制限 18-7, 22-35
ダイレクト・ロード・インサート 21-9
パラレル DML 22-33
制約が適用される 18-14
制約との対比 18-5
タイプ 18-7
データ・アクセス 18-15
データ整合性の維持 1-53
データ整合性の施行 24-4
ビューでは定義できない 8-10
複数トリガーの起動順序 18-14
プログラム・ユニット 1-51
プロシージャとの対比 18-2
文 18-8
ユーザー定義型 12-9
例 18-9, 18-12, 18-15
取消し 1-10
ロールバックも参照
トレース・ファイル 7-14
DBW*n* 28-6
LGWR トレース・ファイル 7-10
トレース・ファイルに記録される内部エラー 7-14

な

内容を保証しない書込み 23-10
内容を保証しない読込み 23-2, 23-10
名前付きユーザー・ライセンス 25-14

に

任意アクセス制御 1-28, 25-2
認証
Oracle 25-4
オペレーティング・システム 25-3
説明 25-3
ネットワーク 25-4

ね

ネストされたループ結合 20-60
コストベース最適化 20-66

ネストした表 8-9, 11-10
索引 12-8
ネットワーク
2 タスク・モード 7-19
Net8 29-4, 30-4
Oracle Names 30-4
Oracle の使用 1-7, 1-26
クライアント / サーバー・アーキテクチャでの使用 29-2
障害 28-3
通信プロトコル 7-27, 29-5
ディスパッチャ・プロセス 7-14, 7-20
ドライバ 7-27
ネットワーク認証サービス 25-4
分散データベース 30-2, 30-4
リスナー・プロセス 7-14

の

ノード
パラレル・サーバーでのディスク親和性 22-36
分散データベース 1-24

は

パーティション 9-2, 9-10
DML パーティション・ロック 9-28
EXCHANGE PARTITION 9-9
LONG および LONG RAW の制限事項 9-11
OLTP データベース 9-5
VLDB 9-4
拡張パーティション表名 9-39
グローバル索引 9-23, 9-36
索引のパーティション化 9-21, 9-27
実行計画 9-9, 9-11
親和性 22-36
制限
拡張パーティション表名 9-40
データ型 9-11, 9-15
ビットマップ索引 9-11
同一キー索引 9-22
同一レベル・パーティション化 9-17
統計 9-11
同時実行のメンテナンス操作 9-31
動的パーティション化 22-6
パーティション・バウンド 9-13
パーティション化キー 9-12, 9-14
パーティション化の基本的なモデル 9-10
パーティションの切詰め 9-3
TO_DATE 書式マスク 9-11, 9-15
索引 9-26
ディスクのストライプ化 22-36
ブロック範囲によるパラレル化 22-4
パーティションの再作成 9-36
パーティションの参照 9-13
パーティションの透過性 9-9
パーティション名 9-13
パラレル DDL 22-22
パラレル化のルール 22-19, 22-20
パラレル問合せ 22-3
ビットマップ索引 8-25
表のパーティション化 9-19
物理属性 9-20, 9-27
メンテナンス操作 9-29
利点 9-4, 9-5
レンジ・パーティション化 9-11
ディスクのストライプ化 22-36
ローカル索引 9-21, 9-35
ロギングなしモード 21-7
パーティション・ビュー 9-9
パーティション化キー 9-12, 9-14
複数列からなるキー 9-15
排他モード 2-24, 5-6
排他ロック
RX ロック 23-22
行ロック（ TX ） 23-19
表ロック（ TM ） 23-20
バイナリ・データ
BFILE 10-10
BLOB 10-9
RAW および LONG RAW 10-10
配列
VARRAY のサイズ 11-10
可変（ VARRAY ） 11-9
配列処理 14-13
バインド変数
最適化 20-57
ユーザー定義型 11-12
パスワード
アカウントのロック 25-5
暗号化 25-4
管理者権限 5-3

時間切れ 25-5
データベース・ユーザーの認証 25-4
パスワード・ファイル 25-7
パスワードの再使用 25-5
パスワードを指定した接続 7-5
パスワードを指定しない接続 25-3
複雑さの検証 25-5
ロールでの使用 1-31
バックアップ
Export を使った補助 28-22
Recovery Manager 1-39, 28-10
概要 1-33, 28-19
制御ファイル 28-21
全体データベース・バックアップ 1-38, 28-20
タイプ 1-38
データ・ファイル 28-21
パラレル 28-11
部分 1-38, 28-21
読込み専用表領域 28-22
バックエンド 29-2
バックグラウンド・プロセス 1-17, 7-5, 7-6
プロセスも参照
概要 1-17
図 7-6
説明 7-5
トレース・ファイル 7-14
パッケージ 17-4, 17-9
概要 1-43
格納 17-15
監査 27-7
キューイング 16-3
共有 SQL 領域 6-9
権限
構成体ごとに分割 26-8
実行 26-6, 26-8
実行 14-15, 17-16
セッションの状態 19-6
動的 SQL 14-18
パブリック 17-14
プライベート 17-14
プログラム・ユニット 1-51
有効性 17-16
利点 17-14
例 17-10, 26-8, 26-9
ロック用 23-38
ハッシュ・クラスタ 1-46, 8-33
概要 1-46
走査 20-41, 20-45, 20-47
ハッシュ結合 20-64
HASH_AREA_SIZE パラメータ 20-65
HASH_MULTIBLOCK_IO_COUNTパラメータ 20-65
発注の例
オブジェクト型 11-2, 11-4
バッファ
REDO ログ・バッファ 1-15, 6-5
データベース・バッファ・キャッシュ 1-15, 6-3, 7-8
増分チェックポイント 7-9, 28-4
バッファ・キャッシュ 6-3, 7-8
拡張バッファ・キャッシュ（32 ビット） 6-12
複数のバッファ・プール 6-5
バッファ・プール 6-5
パフォーマンス
DSS データベース 9-7, 22-27
I/O 9-7
Oracle Parallel Server と DML ロック 9-29
SGA のサイズ 6-11
回復 28-5
クラスタ 8-31
グループ・コミット 7-11
向上させる構造体 1-43
索引作成 8-16
実行計画の表示 20-4
制約が与える影響 24-6
同一キー索引と非同一キー索引 9-26
動的パフォーマンス表（V$） 4-6
パーティション 9-7
パッケージ 17-14
パラレル回復 28-13
リソース制限 25-9
パブリック・ロールバック・セグメント 2-22
パラメータ
各国語サポート 5-4
記憶域 2-5, 2-11
初期化 5-3
初期化パラメータを参照
ロック動作 23-18
パラメータ・ファイル 5-3
起動時の使用 5-5

例 5-4
パラレル・コーディネータ・プロセス 22-5
パラレル・サーバー・プロセス 22-5
パラレル・バックアップ操作 28-11
パラレル・モード 5-6
パラレル DDL
エクステントの割当て 22-24
パーティション表とパーティション索引 22-22
パラレル化ルール 22-15
並行性のタイプ 22-3
パラレル DELETE 22-16
パラレル DML 22-26
PARALLEL DML を有効にする 22-28
アプリケーション 22-27
回復 22-30
制限 22-33
トランザクション・モデル 22-29
パラレル化ルール 22-15
ビットマップ索引 8-22
並行性のタイプ 22-3
並行度 22-15, 22-17
リソースのロックとエンキュー 22-31
パラレル SQL
Parallel Server 22-1
インスタンス・グループ 22-14
オプティマイザ 22-8
行をパラレル・サーバーに割り当てる 22-8
コーディネータ・プロセス 22-5
サマリー表またはロールアップ表 22-22
実行計画にある操作 22-8
パラレル・サーバー・プロセスの数 22-6
パラレル化ルール 22-15
並行度 22-12
マルチスレッド・サーバー 22-7
パラレル UPDATE 22-16
パラレル回復 28-11, 28-12
パラレル伝播 31-21
パラレル問合せ 22-2
インターオペレータ並行性 22-10
イントラオペレータ並行性 22-10
全表走査 22-5
パーティション表とパーティション索引 22-3
パラレル化ルール 22-15
ビットマップ索引 8-22
半結合 20-70
反復可能読込み 23-3
反復不能読込み 23-2, 23-10

ひ

比較メソッド 11-6
非結合 20-70
非コミット読込み 23-3
ビジネス・ルール
アプリケーション・コードで施行する 24-5
ストアド・プロシージャを使って施行する 24-5
制約を使って施行する 1-52
利点 24-5
トリガーを使って施行 1-53
ヒストグラム 20-7
必須アクセス制御 1-33
ビットマップ索引 8-21
NULL 8-7, 8-24
スター型変換 20-73
走査 20-42
パーティション表 9-11
パラレル問合せおよび DML 8-22
非同一キー索引 9-26
非同一レベル結合
定義 20-12
非同期 I/O
パラレル回復 28-13
非同期処理 16-2
非同期レプリケーション 31-19
ビュー 1-41, 8-10
INVALID 状態 19-3
NLS パラメータ 8-12
NULL に対する NULL 以外の値 20-68
SQL ファンクション 8-12
依存性の状態 19-5
オブジェクト・ビュー 8-14, 13-1
行オブジェクト 11-8
更新可能性 13-5
概要 1-41, 8-10
格納方法 8-11
監査 27-7, 27-8
疑似列 18-12
権限 26-5
更新可能性 8-13, 13-5, 18-11
コンパイルの前提条件 19-5
最適化 20-22

索引 8-12
式を含む 18-12
実表 1-42
実表の変更 19-5
使用方法 8-11
スキーマ・オブジェクトの依存性 8-13, 19-4, 19-7
制約が間接的に影響を与える 24-4
制約とトリガーは定義できない 8-10
セキュリティ・アプリケーション 26-6
選択表示結合ビュー 20-22
定義の展開 19-5
データ・ディクショナリ
更新可能な列 8-13
ユーザー・アクセス可能ビュー 4-3
パーティション・ビュー 9-9
パーティションの統計 9-11
ヒストグラム 20-9
複合ビューのマージ 20-23
変更 18-11
変更可能 18-11
本質的に変更可能 18-11
列の最大数 8-10
表
DUAL 4-6
依存ビューに対する影響 19-5
オブジェクト表 11-3, 11-7
仮想 13-2
索引 12-8
制約 12-7
トリガー 12-9
概要 1-41, 8-3
拡張パーティション表名 9-39
仮想またはビュー 1-42
監査 9-39, 27-7
キュー表 16-4, 16-8
クラスタ化 8-28
権限 26-4
索引 8-16
索引構成表 8-25
サマリー表またはロールアップ表 22-22
実表 1-42
データ・ディクショナリでの使用 4-2
ビューとの関連 8-11
整合性制約 24-2, 24-4
整合性制約を含む 1-52
制約を施行する 24-20
制約を使用可能または使用禁止にする 24-20
全表走査とバッファ・キャッシュ 6-4
使うトリガー 18-2
データ・ウェアハウスでのリフレッシュ 22-27
データの格納方法 8-4
動的パーティション化 22-6
ネストした表 8-9, 11-10
索引 12-8
パーティション 9-2, 9-19
パーティションについての権限 9-38
ハッシュ 8-37
パラレル DDL 記憶領域 22-24
パラレル作成 22-22
パラレル問合せでの STORAGE 句 22-24
パラレルの表走査 22-3
ビューでの提示 8-10
表の別名 12-2, 12-3
表名
列名の修飾 12-2, 12-3
表領域に含まれる 8-4
表領域の指定 8-4
領域割当ての制御 8-4, 21-8
履歴データ 22-28
列の最大数 8-10
レプリケーション 1-25, 31-2, 31-3
ロギングなしモード 21-7
ロック 9-28, 23-20, 23-21, 23-23
表ディレクトリ 2-4
表領域 3-3
一時 1-31, 3-10
ユーザーのデフォルト 25-7
一時セグメントに使われる 2-14, 2-16
オブジェクト作成のデフォルト 1-31, 25-7
オフライン 1-9, 3-7, 3-12
再マウント時にオフラインのまま 3-8
索引データとの関係 3-9
読込み専用にできない 3-10
オンライン 1-9, 3-7, 3-12
概要 1-8, 3-3
サイズ 3-6
スキーマとの対比 8-2
説明 3-3
データ・ファイルとの関連 3-2
表に対して指定する方法 8-4

ユーザーからのアクセスを取り消す 25-8
読込み専用 3-9
オブジェクトの削除 3-10
ロギングなしモード 21-7
ロック 23-29
割当て制限 1-31, 25-7
制限ありと制限なし 25-8
デフォルトはない 25-8
ヒント
INDEX 20-72
INDEX_FFS 20-41
MERGE 20-23
MERGE_AJ および HASH_AJ 20-70
MERGE_SJ および HASH_SJ 20-70
OPTIMIZER_MODE と OPTIMIZER_GOAL を上書き 20-39
ORDERED 20-66, 20-72
PARALLEL 22-12
PARALLEL_INDEX 22-12
PUSH_JOIN_PRED 20-69
STAR 20-72
USE_HASH 20-64

ふ

ファイル
ALERT とトレース・ファイル 7-10, 7-14
Oracle データベース 1-8, 1-10, 28-6
制御ファイル、データ・ファイル、REDO ログ・ファイルも参照
オペレーティング・システム 1-5
初期化パラメータ 5-3, 5-5
ダンプ・ファイルの Export と Import 12-15
パスワード 25-7
管理者権限 5-3
ファイル管理ロック 23-29
ファジー読込み 23-2
ファンクション
PL/SQL
権限 26-6, 17-2, 17-6
プロシージャも参照
使用禁止にされたロール 26-13
パラレル DML の制限事項 22-35
プロシージャと対比 1-51, 17-2
SQL
CHECK 制約 24-16
フェールオーバー・データベース 31-12
フェッチする、問合せの行を 14-13
埋込み SQL 14-5
不完全なオブジェクト型 12-14
複合索引 8-17
複合スナップショット 31-10
複合ビューのマージ 20-23
副問合せ 14-11
CHECK 制約が禁止する 24-16
DML 文
直列可能分離 23-13
IN 副問合せの最適化 20-23
NOT IN 20-70
問合せも参照
結合に変換する 20-20
リモート更新 30-10
副問合せスナップショット 31-7
不整なパラレル DML 作業負荷 22-14
物理データベース構造 1-5
部分バックアップ 28-21
付与
EXECUTE ユーザー定義型 12-10
権限とロール 26-2
プライベート・ロールバック・セグメント 2-22
プライベート SQL 領域
カーソル 6-9
持続領域 6-8
実行時領域 6-8
説明 6-8
どのように管理されるか 6-9
プライマリ所有権 31-23
ぶら下がり REF 11-8
フラッガ、規格化されていない機能 1-3, 14-6
ブランチ・ブロック 8-19
プリコンパイラ
FIPS フラガー 14-6
埋込み SQL 14-5
カーソル 14-10
ストアド・プロシージャ 14-17
バインド変数 14-12
無名ブロック 14-16
プログラム・インタフェース 7-26
2 タスク・モード 7-19
Oracle 側（ OPI ） 7-27
概要 1-19

構造 7-27
シングル・タスク・モード 7-17
ユーザー側（UPI） 7-27
プログラム・グローバル領域（PGA） 1-16, 6-12
サイズ 6-14
内容 6-13
非共有と書込み可能 6-12
マルチスレッド・サーバー 7-23
割当て 6-13
プログラム・ユニット 1-42, 14-14, 17-2
共有プール 6-9
コンパイルの前提条件 19-5
プロシージャ 14-14, 17-1, 17-6, 19-7
INVALID 状態 19-3, 19-6
依存性の追跡 19-6
カーソル 14-16
外部プロシージャ 14-18, 17-9
格納 17-15
監査 27-7, 27-8
共有 SQL 領域 6-9
権限
作成または変更 26-7
実行 26-6
パッケージ内での実行 26-8
コンパイルの前提条件 19-5
実行 14-15, 17-16
使用禁止にされたロール 26-13
ストアド・プロシージャ 14-14, 14-17, 17-2
セキュリティの強化 17-7, 26-7
トリガー 18-2
ファンクションと対比 1-51, 17-2
無名ブロックと対比 17-8
有効性 17-16
利点 17-7
リモート・プロシージャ・コール 30-11
例 17-6, 26-8, 26-9
プロシージャ・レプリケーション 31-26
競合の検出 31-26
ラッパー 31-26
プロセス 7-2
Oracle 1-16, 7-5
シングル・プロセス Oracle 7-2
アーカイバ（ARCH） 1-18, 7-13
回復時 28-13
概要 1-16
キュー・モニター（QMN*n*） 1-19, 7-13, 16-4
構造 7-2
サーバー 1-16, 1-23, 7-5
共有 7-14, 7-23
専用 7-17
システム・モニター（SMON） 1-18, 7-12
シャドウ 7-17
障害 28-3
ジョブ・キュー（SNP*n*）
スナップショットのリフレッシュ 31-10, 1-19, 7-13, 31-9
メッセージの波及 16-7
専用サーバー 7-23
チェックポイント 7-8
チェックポイント（CKPT） 1-18, 7-11
ディスパッチャ（D*nnn*） 1-18, 7-14
データベース・ライター（DBW*n*） 1-17, 7-8
トレース・ファイル 7-14
バックグラウンド 1-17, 7-5
図 7-6
パラレル・コーディネータ 22-5
パラレル・サーバー・プロセス 22-5
プロセス・モニター（PMON） 1-18, 7-12
分散トランザクションの解決 7-12
マルチ・プロセス Oracle 7-3
マルチスレッド・サーバー 7-19
クライアントの要求 7-20
人工デッドロック 7-23
ユーザー 1-16, 7-4
PGA の割当て 6-13
サーバー・プロセスの共有 7-14
障害からの回復 7-12
リカバラ（RECO） 1-18, 7-12
インダウト・トランザクション 1-25
リスナー 7-14
共有サーバー 7-20
ログ・ライター（LGWR） 1-17, 7-9
ロック（LCK*n*） 1-19, 7-13
プロセス・グローバル領域（PGA） 6-12
プログラム・グローバル領域を参照
プロセス・モニター・プロセス（PMON）
説明 1-18, 7-12
タイムアウト・セッションのクリーン・アップ 25-11
ネットワーク障害 28-3

パラレル DML プロセス回復 22-30
プロセス障害 28-3
ブロック
データベース 2-3
データ・ブロックも参照
無名 14-14, 17-8
プロファイル
いつ使うか 25-12
概要 1-32
パスワード管理 25-5
フロントエンド 29-2
文
SQL 文を参照
分散処理環境
クライアント/サーバー・アーキテクチャ 1-23, 29-3
説明 1-23, 29-2
データ操作言語 14-10
分散データベース 30-7
分散データベース 30-1
2 フェーズ・コミット 1-25, 30-12
異種間 30-8
依存スキーマ・オブジェクト 19-8
概要 1-24, 30-2
監査 27-5
管理ツール 30-17
クライアント / サーバー・アーキテクチャ 29-2
グローバル・スキーマ・オブジェクト名 30-6
異なる Oracle バージョン 30-7
サーバーをクライアントとしても使える 29-2
サイト自律性 30-15
ジョブ・キュー・プロセス（ SNP*n* ） 1-19, 7-13
図 30-3
データベース・リンク 30-6
デッドロック 23-17
透過性 30-13
ノード 30-2
表のレプリケーション 1-25, 31-2, 31-3
読込み専用 31-2, 31-4
読込みと更新 31-11
分散更新 30-10
分散問合せ 30-10
文の最適化 20-37
メッセージの波及 16-7
リカバラ・プロセス（ RECO ）と 7-12
リモート依存性 19-9
リモート問合せと更新 30-10
分散データベースでのネーム変換 30-6
分散トランザクション
2 フェーズ・コミット 1-25, 15-7
最適化 20-37
定義 30-11
ノードへの文のルーティング 14-11
パラレル DML の制限事項 22-35
文トリガー 18-7
トリガーも参照
起動されるタイミング 18-14
説明 18-8
文へのビューのマージ 20-22
分離レベル
コミット読込み 23-7
設定 23-6, 23-29
選択 23-11
文レベルの読込み一貫性 23-5

へ

並行度 22-15, 22-17
問合せ操作間 22-10
パラレル SQL 22-6, 22-12
ページ 2-2
ヘッダー
行断片 8-5
データ・ブロック 2-4
別の行の書込みが書込みを阻止するか 23-10
別名
列名の修飾 12-2, 12-3
変換、データの
ANSI データ型 10-17
SQL/DS および DB2 データ型 10-17
プログラム・インタフェース 7-26
変数
埋込み SQL 14-5
オブジェクト変数 13-4
ストアド・プロシージャ内 14-16
バインド変数
最適化 20-57
ユーザー定義型 11-12

ほ

保管後転送レプリケーション 31-19

ま

マスター・グループ 31-17
マスター・サイト 31-17
マスター定義サイト 31-17
マスター表
　スナップショット・ログ 31-9
マップ・メソッド 1-52, 11-6
マルチ・プロセス・システム（マルチ・ユーザー・システム） 7-3
マルチ・ブロック書込み 7-8
マルチ・ユーザー環境 7-3
マルチスレッド・サーバー 7-19
　7-20, 7-14
　共有サーバー・プロセス 7-14, 7-23
　サーバー・プロセス 1-17, 7-14, 7-23
　使用例 7-25
　人工デッドロック 7-23
　制限された操作 7-24
　説明 7-16, 7-19
　専用サーバーとの比較 7-19
　ディスパッチャ・プロセス 1-18, 7-14
　必要なプロセス 7-19
マルチバージョン同時実行性制御 23-5
マルチバージョンの一貫性モデル 1-21
マルチマスター・レプリケーション 31-13
マルチメディア・データ型 11-3
マルチユーザー環境 1-2

む

無名 PL/SQL ブロック 14-14, 17-8
　アプリケーション 14-16
　ストアド・プロシージャと対比 17-8
　ストアド・プロシージャのコール 14-17
　動的 SQL 14-18
　パフォーマンス 17-9

め

明示的ロック 23-29
メソッド
　コンストラクタ・メソッド 11-5
　　リテラル起動 12-7
　比較メソッド 11-6
メッセージ・キューイング 16-2
　キュー・モニター・プロセス 1-19, 7-13, 16-4
　　間隔統計 16-7
　　実行の時間枠 16-5
　キュー表 16-4
　キュー表のエクスポート 16-8
　メッセージ 16-3
　リモート・データベース 16-7
メッセージ「スナップショットが古すぎます」 23-5
メディア障害 1-35, 28-5
メモリー
　SQL 文のための割当て 6-10
　カーソル（文ハンドル） 1-16
　拡張バッファ・キャッシュ（ 32 ビット） 6-12
　仮想 6-16
　共有 SQL 領域 6-8
　構造 6-2
　構造の概要 1-12
　システム・グローバル領域（ SGA ）
　　SGA のサイズ 6-11
　　開始アドレス 6-12
　　初期化パラメータ 6-11, 6-12
　　物理メモリーへのロック 6-12, 6-16
　　割当て 6-2
　ストアド・プロシージャ 17-8, 17-15
　ソート領域 6-15
　ソフトウェア・コード領域 6-16
　内容 6-2
　プロセスでの使用 7-2

も

モード
　2 タスク 7-16, 7-17
　アーカイブ・ログ 28-14
　共有 5-6
　シングル・タスク 7-15
　排他 5-6
　表ロック 23-20

ゆ

有効範囲付き REF 11-8, 12-5
ユーザー 25-2
　PUBLIC ユーザー・グループ 25-8, 26-13
　アクセス権 25-2
　一時表領域 1-31, 2-16, 25-7
　監査 27-11
　権限 1-29

シングル・ユーザー Oracle 7-2
スキーマ 1-27, 25-2
スキーマに対応付けられる 8-2
セキュリティ・ドメイン 1-29, 25-2, 26-13
専用サーバー 7-17
データ・ディクショナリにリストされている 4-2
デフォルトの表領域 25-7
同時実行動作の調整 1-20
認証 25-3
パスワード暗号化 25-4
表領域 1-31
表領域の割当て制限 1-31, 25-7
プロセス 1-16, 7-4
プロファイル 1-32, 25-11
分散データベース 30-16
マルチ・ユーザー環境 7-3
マルチユーザー環境 1-2
ユーザーの数によるライセンス 25-14
ユーザー名 1-29, 25-2
セッションと接続 7-5
ライセンス 25-12
リソース使用に対する制限 1-31
リソース制限 25-9
ロール 26-10
ユーザーのタイプ 26-11
ユーザー・プログラム・インタフェース（UPI） 7-27
ユーザー・プロセス
PGA の割当て 6-13
共有サーバー・プロセス 7-23
セッション 7-5
接続 7-4
専用サーバー・プロセス 7-17
ユーザー・ロック 23-38
ユーザー定義データ型 11-1, 11-3, 12-1
Export と Import 12-15
オブジェクト・リレーショナル・モデル 1-40
オブジェクト型 11-2, 11-3
表の別名の使用 12-2
記憶域 12-4
権限 12-9
コレクション 11-9
可変配列（VARRAY） 11-9
ネストした表 11-10
不完全な型 12-13

よ

読込み
使用済み 23-2
データ・ブロック
制限 25-10
反復可能 23-6
読込み一貫性
DML の副問合せ 23-13
定義 1-21
問合せ 14-12
トランザクション 1-21, 23-6
トリガー 18-14, 18-15
マルチバージョンの一貫性モデル 1-21
メッセージ「スナップショットが古すぎます」 23-5
ロールバック・セグメント 2-17
読込みが書込みを阻止するか 23-10
読込み専用スナップショット 31-6
リフレッシュの種類 31-10
読込み専用トランザクション 1-22
読込み専用表領域
制限 3-10
説明 3-9
バックアップ 28-22
読込み専用レプリケーション 31-2, 31-4
使用方法 31-4
予約語 14-2

ら

ライセンス
現行の制限を表示する 25-13
同時使用 25-13
名前付きユーザー 25-14
ラッチ
LRU 7-8
説明 23-28
ラッパー
プロシージャ・レプリケーション 31-26
ラベル
MLSLABEL データ型 10-15

り

リアルタイム・レプリケーション 31-27
リーフ・ブロック 8-19

リカバラ・プロセス（ RECO ） 1-18, 7-12
インダウト・トランザクション 1-25, 5-7, 15-8
離散トランザクションの管理
要約 15-8
リスナー・プロセス 7-14
リソース制限
CPU 時間の制限 25-10
値の決定 25-12
概要 1-32
コール・レベル 25-10
セッション・レベル 25-9
セッション当たりのアイドル時間 25-11
セッション当たりの接続時間 25-11
セッション当たりのプライベート SGA 領域 25-11
ユーザー当たりのセッション数 25-11
論理読込みの制限 25-10
リテラル起動
コンストラクタ・メソッド 12-7
リフレッシュ
ジョブ・キュー・プロセス（ SNP*n* ） 1-19, 7-13
リフレッシュ・グループ 31-9
自動リフレッシュ 31-9
手動リフレッシュ 31-10
リフレッシュ間隔
スナップショット・リフレッシュ・グループ 31-9
リフレッシュの種類
読込み専用スナップショット 31-10
リモート・データベース 1-24, 30-2
分散データベースも参照
データベース・リンク 30-6
リモート・トランザクション 30-11
リモート・プロシージャ・コール 30-11, 31-19
リモート依存性 19-9
領域管理
MINIMUM EXTENT パラメータ 22-24
PCTFREE 2-5
PCTUSED 2-6
空き領域の圧縮 2-9
行連鎖 2-9
セグメント 2-14
ダイレクト・ロード・インサート 21-8
パラレル DDL 22-24
リレーショナル・データベースの操作 1-40
リレーショナル DBMS （ RDBMS ）
SQL 14-2
Oracle も参照
オブジェクト・リレーショナル DBMS 11-2
原理 1-40
リレーション 1-41
履歴データベース
パーティション 9-4
メンテナンス操作 9-30
リンク 30-6

る

ルート・ブロック 8-37
ルールベース最適化 20-10

れ

例外
ストアド・プロシージャ 14-17
トリガー実行中の 18-15
呼び出す 14-17
列
NULL の使用禁止 24-6
疑似列
ROWID 10-11
ROWNUM 20-23, 20-32, 20-54
USER 26-6
順序 8-7
整合性制約 8-4, 8-8, 24-3, 24-6
説明 8-3
定義 1-41
デフォルト値 8-8
ネストした表 8-9
ビューまたは表での最大数 8-10
列オブジェクト 11-7
索引 12-8
列名
問合せでの修飾 12-2
問合せの修飾 12-3
連結索引での最大数 8-17
列グループ 31-25
シャドウ 31-25
列のパーティション化 9-12
レプリケーション 31-1
オブジェクト 31-17
カタログ 31-18
管理者 31-22

基本 31-2, 31-4
競合 31-22
解決 31-25
解決メソッド 31-25
行レベルのレプリケーション 31-25
検出 31-25
データ・モデル 31-23
プロシージャ・レプリケーション 31-26
列グループ 31-25
グループ 31-17
サイト 31-17
受信者 31-22
制限
ダイレクト・ロード・インサート 21-9
パラレル DML 22-33
対称型レプリケーション 31-11, 31-12
対称型レプリケーシヨンの使用方法 31-12
定義 31-2
分散データベース 30-7
伝播担当者 31-22
プロシージャ 31-26
読込み専用レプリケーシヨンの使用方法 31-4
リアルタイム 31-27
レプリケーション管理 API 31-18
レプリケートされたデータの受信者 31-22
レプリケートされたデータの伝播担当者 31-22
連結索引 8-17
連鎖、行の 2-9, 8-5
レンジ・パーティション化 9-11

ろ

ローカル・データベース 1-24
ローカル索引 9-21, 9-25
パーティションの管理 9-35
ビットマップ索引
パーティション表 8-25
パラレル問合せおよび DML 8-22
ロール 1-30, 26-10
CONNECT ロール 12-10, 12-11, 26-14
DBA ロール 12-10, 26-14
DDL 文 26-13
EXP_FULL_DATABASE ロール 26-14
IMP_FULL_DATABASE ロール 26-14
RESOURCE ロール 12-10, 12-11, 26-14
アプリケーション 26-11
アプリケーションにおける 1-31
依存性管理 26-13
オペレーティング・システムを介した管理 26-14
概要 1-30
機能性 26-2
キュー管理者 16-5
グローバル認証サービス 30-16
権限に関する制限 26-13
事前定義済み 26-14
使用可能または使用禁止 26-12
使用方法 26-11
スキーマには含まれない 26-13
セキュリティ・ドメイン 26-13
取消し 26-12
名前付きPL/SQLブロックで使用禁止にされた 26-13
パスワードの使用 1-31
付与 26-2, 26-12
付与できるユーザー 26-12
分散データベース・アプリケーション 30-16
命名 26-13
ユーザー 26-11
ロールバック 2-16, 15-6
回復時 1-39, 28-9
高速トランザクション・ロールバック 28-10
セーブポイントまで 15-6
説明 15-6
定義 1-49
トランザクションの終了 15-2, 15-4, 15-6
文レベル 15-4
ロールバック、回復時の 28-9
ロールバック、トランザクションの 1-49, 15-2, 15-6
高速ウォームスタート 28-4
ロールバック・エントリ 2-16
ロールバック・セグメント 1-10, 2-16
1 つあたりのトランザクション数 2-18
SYSTEM ロールバック・セグメント 2-22
アクセス 2-17
いつ取得されるか 2-23
いつ使われるか 2-17
インダウト分散トランザクション 2-21
エクステントの割当て 2-18
新しいエクステント 2-21
エクステントの割当て解除 2-22
オフライン 2-24, 2-26

オフライン表領域 2-27
オンライン 2-24, 2-26
回復が必要な状態 2-24
回復での使用 1-38, 28-9
概要 2-16, 28-7
起動時に取得する 5-7
競合 2-18
削除 2-22
制限 2-26
取得時のクラッシュ 2-23
状態 2-24
遅延 2-27
次のエクステントへの移動 2-19
定義 1-10
トランザクション 2-17
トランザクションからどのように書き込まれるか 2-18
トランザクションのコミット 2-18
パブリック 2-22
パラレル回復 28-10
部分的に使用可能な状態 2-24, 28-4
プライベート 2-22
無効な状態 2-24
読込み一貫性 1-21, 2-17, 23-4
ロールバック・セグメント 2-18
ロック 23-29
ロールフォワード、回復時の 1-39, 28-8
ロギング・モード
NOARCHIVELOG モードとの関係 21-5
影響を受ける SQL 操作 21-7
ダイレクト・ロード・インサート 21-5
パーティション 9-34
パラレル DDL 22-22, 22-23
ログ・ライター・プロセス（LGWR） 1-17, 7-9
REDO ログ・バッファ 6-6
アーカイブ・モード 28-15
グループ・コミット 7-11
システム変更番号 15-5
事前書込み 7-10
手動アーカイブ 28-17
複数の I/O プロセス 7-11
ログ管理ロック 23-29
ログ順序番号 1-36
ロック 1-22, 23-3
DML が取得する 23-26
図 23-24
DML パーティション・ロック 9-28
Oracle での手順 23-14
Oracle のロック管理サービス 23-38
オブジェクト・レベルのロック 11-13
解析 14-11, 23-27
概要 1-22, 23-3
行（TX） 23-19
行共有表ロック（RS） 23-21
行排他ロック（RX） 23-22
共有行排他ロック（SRX） 23-23
共有表ロック（S） 23-23
共有副排他ロック（SSX） 23-23
自動 1-22, 23-14, 23-18
手動 1-23, 23-29
動作の例 23-30
タイプ 23-18
段階的拡大が発生しない 23-16
ディクショナリ 23-26
クラスタ 23-28
存続期間 23-28
ディクショナリ・キャッシュ 23-28
データ 23-19
存続期間 23-14
デッドロック 23-16, 23-17
回避 23-18
トランザクションをコミットした後 15-6
内部 23-28
排他表ロック（X） 23-24
パラレル・キャッシュ管理（PCM） 23-18
パラレル DML 22-31
表（TM） 23-20
表領域 23-29
表ロックのモード 23-20
ファイル管理ロック 23-29
副共有表ロック（SS） 23-21
副排他表ロック（SX） 23-22
変換 23-15
ラッチ 23-28
ロールバック・セグメント 23-29
ログ管理ロック 23-29
ロック・プロセス（LCK*n*） 1-19, 7-13
論理構造 1-8
論理データベース構造 1-5
論理ブロック 2-2

論理読込みの制限 25-10

わ

割当て制限
　　ゼロに設定 25-8
　　表領域 1-31, 25-7
　　一時セグメントでは無視される 25-8
　　表領域アクセスの取消し 25-8