

# Oracle8*i*

チューニング

リリース 8.1

---

## Oracle8i チューニング、リリース 8.1

部品番号 A62765-1

第 1 版 1999 年 5 月 (第 1 刷)

原本名: Oracle8i Tuning, Release 8.1.5

原本部品番号: A67775-01

原著者: Mark Bauer.

原本協力者: Benoit Dageville, Lilian Hobbs, Paul Lane, Lefty Leverenz, Rita Moran, Juan Tellez, Graham Wood, and Mohamed Zait.

Copyright © 1999, Oracle Corporation. All rights reserved.

Printed in Japan.

### 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。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当ソフトウェア (プログラム) のリバース・エンジニアリングは禁止されています。

このドキュメントの情報は、予告なしに変更されることがあります。オラクル社は本ドキュメントの無謬性を保証しません。

\* オラクル社とは、Oracle Corporation (米国オラクル) または日本オラクル株式会社 (日本オラクル) を指します。

### 危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル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り行ってください。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、日本オラクル株式会社と開発元である Oracle Corporation (米国オラクル) およびその関連会社は一切責任を負いかねます。当プログラムを米国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Legend が適用されます。

### Restricted Rights Legend

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication and disclosure of the Programs shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-14, Rights in Data -- General, including Alternate III (June 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

このドキュメントに記載されているその他の会社名および製品名は、あくまでその製品および会社を識別する目的にのみ使用されており、それぞれの所有者の商標または登録商標です。

### ドラフトのアルファ版およびベータ版ドキュメント

ドラフトのアルファ版およびベータ版ドキュメントはプレリリース状態のものです。これらのドキュメントは、オラクル社の機密かつ所有のドキュメントであり、デモおよび暫定使用のみを目的としたものです。タイプミスからデータの不正確さに至るまでのいくつかの誤りが存在することが考えられます。このドキュメントは予告なく変更する場合がありますが、当ソフトウェアを使用するハードウェアに限定するものではありません。オラクル社はプレリリースのドキュメントに対して、無謬性を保証しません。またそのドキュメントを使用したことによって損失および損害が発生した場合も一切責任を負いかねますのでご了承ください。

---

# 目次

はじめに .....	xxiii
------------	-------

## 第 I 部 チューニングの概要

### 1 Oracle パフォーマンス・チューニングの概要

パフォーマンス・チューニングについて .....	1-2
応答時間とスループットのトレードオフ .....	1-2
重要なリソース .....	1-4
過度な需要の影響 .....	1-5
問題を軽減するための調整 .....	1-6
チューニング担当者 .....	1-7
パフォーマンス目標の設定 .....	1-9
ユーザー期待値の設定 .....	1-9
パフォーマンスの評価 .....	1-9

### 2 パフォーマンス・チューニングの方法

チューニングが最も効果的なとき .....	2-1
システムの設計中および開発中の事前チューニング .....	2-1
本番システムを改善するための事後チューニング .....	2-3
優先順位付けされたチューニングのステップ .....	2-4
ステップ 1: 業務規則のチューニング .....	2-7
ステップ 2: データ設計のチューニング .....	2-7
ステップ 3: アプリケーション設計のチューニング .....	2-8
ステップ 4: データベースの論理構造のチューニング .....	2-9

ステップ 5: データベース操作のチューニング.....	2-9
ステップ 6: アクセス・パスのチューニング.....	2-9
ステップ 7: メモリー割当てのチューニング.....	2-10
ステップ 8: I/O および物理構造のチューニング.....	2-10
ステップ 9: リソースの競合のチューニング.....	2-11
ステップ 10: 基礎を形成するプラットフォームのチューニング.....	2-11
<b>チューニング方法の適用</b> .....	2-12
チューニングの明確な目標の設定.....	2-12
最小反復テストの作成.....	2-12
仮説のテスト.....	2-13
記録と自動テスト.....	2-13
一般的な過ちの回避.....	2-13
目標を達成したときのチューニングの停止.....	2-14
目標達成の証明.....	2-14

## 第 II 部 設計者およびプログラマのためのアプリケーション設計のチューニング

### 3 アプリケーションおよびシステムのパフォーマンス特性

<b>アプリケーションのタイプ</b> .....	3-1
オンライン・トランザクション処理 ( OLTP ).....	3-1
データ・ウェアハウス.....	3-3
多目的アプリケーション.....	3-5
<b>Oracle の構成</b> .....	3-6
分散システム.....	3-6
Oracle Parallel Server.....	3-8
クライアント / サーバー構成.....	3-9

### 4 データベース操作のチューニング

<b>チューニングの目標</b> .....	4-2
シリアル SQL 文のチューニング.....	4-2
パラレル実行のチューニング.....	4-3
OLTP アプリケーションのチューニング.....	4-3
<b>データベース操作のチューニング方法</b> .....	4-5
ステップ 1: リソースを最も多く消費している文の検索.....	4-5

ステップ 2: 使用するリソースの減少を目的とした文のチューニング.....	4-5
<b>SQL 文のチューニングのアプローチ</b> .....	4-6
索引の再構成.....	4-7
文の再構成.....	4-7
トリガーの変更または使用禁止.....	4-16
データの再構成.....	4-16
最新統計の収集およびプラン・スタビリティの使用による実行計画の保持.....	4-17

## 5 アプリケーションの登録

DBMS_APPLICATION_INFO パッケージ .....	5-2
権限.....	5-2
<b>モジュール名の設定</b> .....	5-2
例.....	5-2
構文.....	5-3
<b>アクション名の設定</b> .....	5-3
例.....	5-3
構文.....	5-4
<b>クライアント情報の設定</b> .....	5-4
構文.....	5-4
<b>アプリケーション情報の検索</b> .....	5-5
VSQLAREA の問合せ .....	5-5
READ_MODULE の構文 .....	5-5
READ_CLIENT_INFO の構文.....	5-6

## 6 データ・アクセス方法

<b>索引の使用方法</b> .....	6-2
索引作成のタイミング.....	6-2
論理構造のチューニング.....	6-3
索引を付ける列と式の選択.....	6-4
複合索引の選択.....	6-5
索引を使用する文の記述.....	6-6
索引を使用しない文の記述.....	6-7
索引の有効性の調査.....	6-7
高速全索引走査の使用.....	6-8
索引の再作成.....	6-8

索引の圧縮.....	6-9
一意性を施行するための一意でない索引の使用方法.....	6-10
未検査使用可能制約の使用方法.....	6-10
<b>関数ベース索引の使用方法.....</b>	<b>6-11</b>
関数ベース索引と索引構成表.....	6-12
<b>ビットマップ索引の使用方法.....</b>	<b>6-12</b>
ビットマップ索引の使用のタイミング.....	6-13
ビットマップ索引の作成.....	6-15
ビットマップ索引のための初期化パラメータ.....	6-17
B* ツリー索引に対するビットマップ・アクセス計画の使用方法.....	6-18
ビットマップ索引のサイズの推定.....	6-19
ビットマップ索引の制約事項.....	6-22
<b>ドメイン・インデックスの使用方法.....</b>	<b>6-22</b>
<b>クラスタの使用方法.....</b>	<b>6-23</b>
<b>ハッシュ・クラスタの使用方法.....</b>	<b>6-24</b>
ハッシュ・クラスタの用途.....	6-24
ハッシュ・クラスタの作成.....	6-25

## 7 オプティマイザ・モード、プラン・スタビリティおよびヒント

<b>コストベースの最適化.....</b>	<b>7-2</b>
コストベース・アプローチの適用時期.....	7-2
コストベース・アプローチの使用方法.....	7-3
コストベース・アプローチの目標の選択.....	7-3
分散が均一でないデータのヒストグラムの使用方法.....	7-4
<b>統計情報の生成.....</b>	<b>7-7</b>
DBMS_STATS パッケージでの統計収集.....	7-8
新しいオプティマイザ統計の収集.....	7-16
<b>自動統計収集.....</b>	<b>7-17</b>
コストベースの最適化計画に影響するパラメータ.....	7-19
オプティマイザの索引使用方法に影響するパラメータ.....	7-20
コストベース・アプローチの使用上のヒント.....	7-21
<b>ルールベースの最適化.....</b>	<b>7-22</b>
<b>実行計画を保存するプラン・スタビリティの使用.....</b>	<b>7-22</b>
プラン・スタビリティでのヒントおよび完全テキスト一致の使用.....	7-23
SQL 文とアウトラインの突合せ.....	7-24

Oracle によるアウトラインの格納方法 .....	7-24
プラン・スタビリティを使用可能にするパラメータ設定 .....	7-25
<b>アウトラインの作成</b> .....	7-25
カテゴリの作成とストアド・アウトラインへの割当て .....	7-25
ストアド・アウトラインの使用 .....	7-26
アウトライン・データの照会 .....	7-26
<b>OUTLN_PKG パッケージでのストアド・アウトラインの管理</b> .....	7-27
アウトライン表の移動 .....	7-28
<b>コストベース・オブティマイザのプラン・スタビリティ・プロシージャ</b> .....	7-29
コストベース・オブティマイザへの移行に対するアウトラインの使用 .....	7-29
RDBMS アップグレードとコストベース・オブティマイザ .....	7-30
<b>ヒントの使用</b> .....	7-32
ヒントの指定方法 .....	7-32
最適化アプローチと目標のヒント .....	7-34
アクセス方法のヒント .....	7-37
結合順序のヒント .....	7-46
結合操作のヒント .....	7-47
パラレル実行のヒント .....	7-51
その他のヒント .....	7-57
ビューでのヒントの使用方法 .....	7-62

## 8 分散問合せのチューニング

<b>リモート問合せおよび分散問合せ</b> .....	8-1
リモート・データ・ディクショナリ情報 .....	8-2
リモート SQL 文 .....	8-2
分散 SQL 文 .....	8-3
EXPLAIN PLAN と SQL の分解 .....	8-5
パーティション・ビュー .....	8-6
<b>分散問合せの制限事項</b> .....	8-10
<b>Transparent Gateway</b> .....	8-10
<b>まとめ：分散問合せのパフォーマンスの最適化</b> .....	8-11

## 9 トランザクション・モード

<b>ディスクリット・トランザクションの使用</b> .....	9-1
ディスクリット・トランザクションの使用時期の決定 .....	9-1

ディスクリット・トランザクションの処理方法.....	9-2
ディスクリット・トランザクションの処理中に発生するエラー .....	9-2
使用上の注意.....	9-3
例.....	9-3
直列可能トランザクションの使用方法.....	9-5

## 10 SQL および共有 PL/SQL 領域の管理

SQL 文と PL/SQL ブロックの比較 .....	10-2
同一の SQL 文かどうかのテスト .....	10-2
標準化された SQL フォーマットについて .....	10-3
共有プールでの共有 SQL と PL/SQL の維持 .....	10-3
大きな割当て用の領域の予約.....	10-3
オブジェクト除去の防止.....	10-3

## 11 データ・ウェアハウス・アプリケーションの最適化

データ・ウェアハウス・アプリケーションの特徴.....	11-1
データ・ウェアハウスの作成.....	11-2
マテリアライズド・ビューとディメンション .....	11-2
パラレル CREATE TABLE ... AS SELECT .....	11-3
索引のパラレル作成.....	11-3
高速全索引走査.....	11-3
パーティション表.....	11-3
ANALYZE 文 .....	11-4
パラレル・ロード .....	11-4
制約.....	11-4
データ・ウェアハウスの問合せ.....	11-5
Oracle Parallel Server .....	11-5
並列を考慮するオプティマイザ.....	11-6
パラレル実行.....	11-6
ビットマップ索引.....	11-7
ドメイン索引.....	11-7
スター問合せ .....	11-7
クエリー・リライト.....	11-8
データ・ウェアハウス・アプリケーションのチューニング .....	11-8
データ・ウェアハウスのバックアップとリカバリ.....	11-8



ファースト・スタート・パラレル・リカバリのチューニング.....	11-9
----------------------------------	------

## 第 III 部 設計者および DBA 用のアプリケーション設計ツール

### 12 診断ツールの概要

チューニング用のデータ・ソース.....	12-1
データ・ボリューム.....	12-2
オンライン・データ・ディクショナリ.....	12-2
オペレーティング・システム・ツール.....	12-2
動的パフォーマンス表.....	12-3
Oracle Trace と Oracle Trace Data Viewer .....	12-3
SQL トレース機能.....	12-3
アラート・ログ.....	12-3
アプリケーション・プログラムの出力.....	12-3
ユーザー .....	12-4
初期化パラメータ・ファイル.....	12-4
プログラム・テキスト.....	12-4
設計（分析）ディクショナリ.....	12-4
比較データ.....	12-4
動的パフォーマンス・ビュー.....	12-5
Oracle および SNMP サポート.....	12-5
EXPLAIN PLAN .....	12-5
Oracle Trace と Oracle Trace Data Viewer .....	12-6
SQL トレース機能と TKPROF.....	12-6
サポートされるスクリプト.....	12-6
アプリケーションの登録.....	12-7
Oracle Enterprise Manager、パックおよびアプリケーション.....	12-7
Oracle Enterprise Manager の概要.....	12-8
Oracle Diagnostics Pack .....	12-9
Oracle Tuning Pack.....	12-10
Oracle Parallel Server Management.....	12-11
ユーザー開発ツール.....	12-12

### 13 EXPLAIN PLAN の使用方法

EXPLAIN PLAN の概要 .....	13-1
------------------------	------

出力表の作成.....	13-2
PLAN_TABLE 出力の表示.....	13-3
出力表の列.....	13-3
ビットマップ索引と EXPLAIN PLAN.....	13-12
EXPLAIN PLAN とパーティション・オブジェクト .....	13-12
EXPLAIN PLAN によるレンジおよびハッシュ・パーティション化の表示.....	13-13
コンポジット・パーティション・オブジェクトでの情報のプルニング.....	13-15
パーシャル・パーティション・ワイズ・ジョイン.....	13-17
フル・パーティション・ワイズ・ジョイン.....	13-19
INLIST ITERATOR と EXPLAIN PLAN .....	13-20
ドメイン索引と EXPLAIN PLAN.....	13-21
EXPLAIN PLAN 出力のフォーマット .....	13-22
EXPLAIN PLAN 文の使用.....	13-22
表フォーマットでの PLAN_TABLE 出力の選択.....	13-23
ネストされたフォーマットでの PLAN_TABLE 出力の選択 .....	13-24
EXPLAIN PLAN の制限事項.....	13-25

## 14 SQL トレース機能と TKPROF

SQL トレース機能と TKPROF の概要.....	14-1
SQL トレース機能について .....	14-2
TKPROF について .....	14-2
SQL トレース機能と TKPROF の使用 .....	14-2
ステップ 1: トレース・ファイル管理用の初期化パラメータの設定.....	14-3
ステップ 2: SQL トレース機能の使用可能化 .....	14-4
現行セッションの SQL トレース機能の使用可能化.....	14-4
インスタンスに対する SQL トレース機能の使用可能化 .....	14-5
ステップ 3: TKPROF によるトレース・ファイルのフォーマット .....	14-5
TKPROF の出力例.....	14-6
TKPROF の構文.....	14-7
TKPROF 文の例.....	14-10
ステップ 4: TKPROF 出力の解釈.....	14-11
表形式の統計.....	14-11
ライブラリ・キャッシュ・ミス.....	14-13
文の切捨て.....	14-13
SQL 文を発行するユーザー .....	14-13

実行計画 .....	14-13
どの文をチューニングするか判断 .....	14-14
<b>ステップ 5: SQL トレース機能統計の格納</b> .....	14-15
TKPROF による出力 SQL スクリプトの生成 .....	14-15
TKPROF による出力 SQL スクリプトの編集 .....	14-15
出力表の問合せ .....	14-16
<b>TKPROF の解釈における落とし穴の回避</b> .....	14-18
負荷の大半を占める文の検出 .....	14-18
引数トラップ .....	14-18
読み込み一貫性トラップ .....	14-19
スキーマ・トラップ .....	14-19
時間トラップ .....	14-20
トリガー・トラップ .....	14-20
"正しい" バージョン .....	14-21
<b>TKPROF の出力例</b> .....	14-21
ヘッダー .....	14-22
本体 .....	14-22
サマリー .....	14-28

## 15 Oracle Trace の使用方法

<b>Oracle Trace について</b> .....	15-1
Oracle Trace データの使用方法 .....	15-1
<b>Oracle Trace Manager の使用方法</b> .....	15-4
収集の管理 .....	15-4
イベント・データの収集 .....	15-4
収集したデータへのアクセス .....	15-5
<b>Oracle Trace Data Viewer の使用方法</b> .....	15-6
Oracle Trace 事前定義済みデータ・ビュー .....	15-6
Oracle Trace データの表示 .....	15-11
「SQL Statement」プロパティ・ページ .....	15-13
「Details」プロパティ・ページ .....	15-13
「Details」プロパティ・ページの例 .....	15-13
選択された問合せの追加情報の取得 .....	15-15
<b>Oracle Trace データの手動収集</b> .....	15-18
Oracle Trace コマンド行インタフェースの使用 .....	15-18

Oracle Trace の制御のための初期化パラメータの使用 .....	15-20
Oracle Trace の制御のためのストアド・プロシージャの使用 .....	15-22
Oracle Trace の収集結果 .....	15-23
Oracle Trace データの Oracle 表へのフォーマット設定 .....	15-24
Oracle Trace 統計レポート作成ユーティリティ .....	15-24

## 第 IV 部 インスタンスのパフォーマンスの最適化

### 16 動的パフォーマンス・ビュー

チューニングのためのインスタンスレベルのビュー .....	16-2
チューニングのためのセッションレベルのビューまたは一時的なビュー .....	16-3
現行の統計値および変更率 .....	16-4
統計の現在の設定値の検索 .....	16-4
統計の変更率の検索 .....	16-5

### 17 システムのパフォーマンス問題の診断

適切に設計された既存のシステムの要因のチューニング .....	17-1
不十分な CPU .....	17-4
不十分なメモリー .....	17-4
不十分な I/O .....	17-5
ネットワークの制約 .....	17-5
ソフトウェアの制約 .....	17-6

### 18 CPU リソースのチューニング

CPU の問題について .....	18-1
CPU 問題の検出と解決 .....	18-3
システムの CPU 使用率 .....	18-3
Oracle の CPU 使用率 .....	18-5
システム・アーキテクチャの変更による CPU の問題の解決 .....	18-9
単層から 2 層へ .....	18-10
複数層：より小さなクライアント・マシンの使用 .....	18-11
2 層から 3 層へ：トランザクション処理モニターの使用 .....	18-11
3 層：複数の TP モニターの使用 .....	18-12
Oracle Parallel Server .....	18-12

## 19 メモリー割当てのチューニング

メモリー割当ての問題について.....	19-1
メモリー割当ての問題の検出方法.....	19-3
メモリー割当ての問題の解決方法.....	19-3
オペレーティング・システムのメモリー所要量のチューニング.....	19-4
ページングとスワッピングの低減.....	19-4
主メモリーへのシステム・グローバル領域 (SGA) の格納.....	19-5
個々のユーザーへの十分なメモリーの割当て.....	19-6
REDO ログ・バッファのチューニング.....	19-6
プライベート SQL と PL/SQL 領域のチューニング.....	19-7
不要な解析コールの識別.....	19-8
不要な解析コールの低減.....	19-9
共有プールのチューニング.....	19-10
ライブラリ・キャッシュのチューニング.....	19-12
データ・ディクショナリ・キャッシュのチューニング.....	19-18
マルチスレッド・サーバー・アーキテクチャでの大規模プールと共有プールのチューニング.....	19-20
3 層の接続でのメモリー使用の低減.....	19-20
VSSESSTAT ビュー.....	19-21
VSSESSTAT ビューの問合せ.....	19-21
共有プールの予約領域のチューニング.....	19-22
予約リストのチューニング・パラメータ.....	19-22
共有プールのスペース再利用の制御.....	19-22
初期化パラメータの値.....	19-23
SHARED_POOL_RESERVED_SIZE が小さすぎる場合.....	19-23
SHARED_POOL_RESERVED_SIZE が大きすぎる場合.....	19-24
SHARED_POOL_SIZE が小さすぎる場合.....	19-24
バッファ・キャッシュのチューニング.....	19-24
キャッシュ・ヒット率によるバッファ・キャッシュのアクティビティの評価.....	19-24
バッファ・キャッシュ・ミスの低減によるキャッシュ・ヒット率の増加.....	19-28
キャッシュ・ヒット率が高い場合の不必要なバッファの削除.....	19-28
バッファ・キャッシュでの LOB の調整.....	19-28
一時 LOB.....	19-28
複数バッファ・プールのチューニング.....	19-29
複数バッファ・プールの機能の概要.....	19-29
複数バッファ・プールの用途.....	19-30
複数バッファ・プールを使ったバッファ・キャッシュのチューニング.....	19-31

複数バッファ・プールの使用可能化.....	19-32
複数バッファ・プールの使用方法.....	19-33
デフォルトのバッファ・プールを表示するディクショナリ・ビュー.....	19-34
各バッファ・プールのサイズ設定.....	19-35
LRU ラッチ競合の識別と排除 .....	19-37
<b>ソート領域のチューニング</b> .....	19-38
<b>メモリーの再割当て</b> .....	19-39
<b>合計メモリー使用量の低減</b> .....	19-39

## 20 I/O のチューニング

<b>I/O の問題について</b> .....	20-1
I/O のチューニング: トップ・ダウンとボトム・アップ.....	20-2
I/O 要件の分析.....	20-2
ファイル格納の計画.....	20-5
データ・ブロック・サイズの選択.....	20-15
デバイス帯域幅の評価.....	20-16
<b>I/O の問題の検出</b> .....	20-17
システムの I/O 使用率の検査.....	20-17
Oracle の I/O 使用率の検査 .....	20-17
<b>I/O 問題の解決</b> .....	20-20
<b>I/O の分散によるディスク競合の低減</b> .....	20-20
ディスク競合とは.....	20-20
データ・ファイルと REDO ログ・ファイルの分離 .....	20-21
表データのストライプ化.....	20-21
表と索引の分離.....	20-22
Oracle と関係のないディスク I/O の低減 .....	20-22
<b>ディスクのストライプ化</b> .....	20-22
ストライプ化の目的.....	20-22
I/O のバランス化とストライプ化.....	20-22
ディスクを手動でストライプ化する方法.....	20-23
オペレーティング・システム・ソフトウェアでディスクをストライプ化する方法.....	20-25
RAID でハードウェア・ストライプ化を行う方法 .....	20-25
<b>動的な領域管理の回避</b> .....	20-26
動的拡張の検出.....	20-26
エクステンツの割当て.....	20-27

無制限のエクステントの評価.....	20-28
複数のエクステントの評価.....	20-29
ロールバック・セグメント内の動的領域管理の回避.....	20-29
移行行と連鎖行の削減.....	20-30
SQL.BSQ ファイルの修正 .....	20-33
<b>ソートのチューニング</b> .....	20-33
メモリーでのソート.....	20-34
ディスクでのソート.....	20-35
一時表領域の使用によるソート・パフォーマンスの最適化.....	20-36
一時表領域のストライプ化によるソート・パフォーマンスの改善.....	20-36
SORT_MULTIBLOCK_READ_COUNT を使用するソート・パフォーマンスの改善 .....	20-36
NOSORT の使用によるソートを行わない索引作成.....	20-37
GROUP BY NOSORT .....	20-37
<b>チェックポイントのチューニング</b> .....	20-38
チェックポイントがパフォーマンスに与える影響.....	20-38
チェックポイントの頻度の選択.....	20-38
ファースト・スタート・チェックポイント.....	20-39
<b>LGWR および DBWn の I/O のチューニング</b> .....	20-40
LGWR の I/O のチューニング.....	20-40
DBWn の I/O のチューニング .....	20-41
<b>バックアップおよび復元操作のチューニング</b> .....	20-45
ボトルネックの原因の発見.....	20-45
バックアップ操作を監視するための固定ビューの使用.....	20-46
バックアップのスループットの改善.....	20-49
<b>大規模プールの構成</b> .....	20-53

## 21 リソースの競合のチューニング

競合の問題の理解.....	21-1
競合の問題の検出.....	21-2
競合の問題の解決.....	21-3
<b>ロールバック・セグメントの競合の低減</b> .....	21-3
ロールバック・セグメントの競合の識別.....	21-3
ロールバック・セグメントの作成.....	21-4
<b>マルチスレッド・サーバー・プロセスの競合の低減</b> .....	21-5
ディスパッチャ固有のビューを使用する競合の識別.....	21-5

ディスパッチャ・プロセスの競合の低減.....	21-6
共有サーバー・プロセスの競合の低減.....	21-9
<b>パラレル実行サーバーの競合の低減.....</b>	<b>21-13</b>
パラレル実行サーバーの競合の識別.....	21-13
パラレル実行サーバーの競合の低減.....	21-13
<b>REDO ログ・バッファ・ラッチの競合の低減.....</b>	<b>21-14</b>
REDO ログ・バッファ内の領域の競合の検出 .....	21-14
REDO ログ・バッファ・ラッチの競合の検出 .....	21-15
REDO ログのアクティビティの検査 .....	21-16
<b>LRU ラッチの競合の低減.....</b>	<b>21-17</b>
<b>空きリスト競合の低減.....</b>	<b>21-19</b>
空きリストの競合の識別.....	21-19
空きリストの追加.....	21-20

## 22 ネットワークのチューニング

ネットワークの問題の検出.....	22-1
ネットワークの問題の解決.....	22-2
配列インタフェースの使用方法.....	22-2
事前開始プロセスの使用方法.....	22-2
セッション・データ単位バッファ・サイズの調整.....	22-2
リスナー・キューのサイズの増大.....	22-3
TCP.NODELAY の使用方法.....	22-3
専用サーバー・プロセスではなく共有サーバー・プロセスを使用する方法.....	22-3
Connection Manager の使用.....	22-4

## 23 マルチスレッド・サーバー・アーキテクチャのチューニング

MTS の設定.....	23-1
MTS の恩恵を受けるアプリケーションの種類 .....	23-1
MTS によるユーザー・スケーラビリティの改善.....	23-2
ディスパッチャの構成.....	23-3
接続プーリングと接続多重化.....	23-3
MTS によるスループットの最大化と応答時間の最短化.....	23-4
共有サーバー数の構成と管理.....	23-4
SDU サイズのチューニング .....	23-5
接続負荷の調整.....	23-5



MTS によるメモリー使用状況のチューニング.....	23-5
MTS での大規模プールと共有プールの構成 .....	23-5
PRIVATE_SGA の設定によるユーザー・セッションごとのメモリー使用量の制限 .....	23-6
接続、負荷および統計データを含む MTS 関連ビュー.....	23-7
MTS 機能のパフォーマンス問題.....	23-8
 <b>24 オペレーティング・システムのチューニング</b>	
オペレーティング・システムのパフォーマンスの問題の理解.....	24-1
オペレーティング・システムおよびハードウェア・キャッシュ.....	24-2
ロー・デバイス.....	24-2
プロセス・スケジューラ.....	24-2
オペレーティング・システムの問題の検出.....	24-3
オペレーティング・システムの問題の解決.....	24-3
UNIX ベース・システムでのパフォーマンス .....	24-4
NT システムでのパフォーマンス.....	24-4
メインフレーム・コンピュータでのパフォーマンス.....	24-4
 <b>25 インスタンス回復パフォーマンスのチューニング</b>	
インスタンス回復について.....	25-1
REDO ログ情報の適用方法 .....	25-2
回復時間最短化のトレードオフ.....	25-2
インスタンスおよびクラッシュの回復時間のチューニング.....	25-2
インスタンスとクラッシュの回復時間に影響を与える初期化パラメータの使用.....	25-3
チェックポイントの頻度に影響を与える REDO ログ・サイズの使用 .....	25-5
チェックポイントを開始するための SQL 文の使用 .....	25-5
インスタンス回復の監視.....	25-6
インスタンス回復のフェーズのチューニング.....	25-13
ロール・フォワード・フェーズのチューニング.....	25-13
ロールバック・フェーズのチューニング.....	25-14
透過的アプリケーション・フェイルオーバー.....	25-16
透過的アプリケーション・フェイルオーバーとは.....	25-16
透過的アプリケーション・フェイルオーバーの仕組み.....	25-16
透過的アプリケーション・フェイルオーバーの実現例.....	25-18
透過的アプリケーション・フェイルオーバーについて ( DBA 向け ).....	25-19
透過的アプリケーション・フェイルオーバーについて ( アプリケーション開発者向け ).....	25-22

透過的アプリケーション・フェイルオーバーの制限事項.....	25-23
--------------------------------	-------

## 第 V 部 パラレル実行

### 26 パラレル実行のチューニング

パラレル実行のチューニングの概要.....	26-2
パラレル実行をインプリメントするタイミング.....	26-3
フェーズ 1: パラレル実行のためのパラメータの初期化とチューニング.....	26-4
ステップ 1: パラレル実行の自動または手動チューニングの選択.....	26-5
完全に自動化されたパラレル実行環境で自動的に導出されるパラメータ設定.....	26-5
ステップ 2: 並列度の設定とマルチユーザー問合せ調整の使用可能化.....	26-7
並列度とマルチユーザー問合せ調整、およびその相互作用.....	26-7
表と問合せでの並列性の使用可能化.....	26-8
PARALLEL_THREADS_PER_CPU を使ったパフォーマンスの制御.....	26-8
ステップ 3: 一般パラメータのチューニング.....	26-9
パラレル操作のリソース制限を設定するパラメータ.....	26-9
リソース消費に影響するパラメータ.....	26-18
I/O に関連したパラメータ.....	26-26
パラレル実行でのパラメータ設定シナリオの例.....	26-29
例 1: 小規模なデータ・マート.....	26-29
例 2: 中規模サイズのデータ・ウェアハウス.....	26-30
例 3: 大規模なデータ・ウェアハウス.....	26-31
例 4: 非常に大規模なデータ・ウェアハウス.....	26-32
フェーズ 2: パラレル実行の物理データベース・レイアウトのチューニング.....	26-34
並列性のタイプ.....	26-34
データのパーティション化.....	26-43
パーティションのプルニング.....	26-48
パーティション・ワイズ・ジョイン.....	26-50
フェーズ 3: データベースの作成、移入およびリフレッシュ.....	26-59
パラレル・ロードによるデータベースへの移入.....	26-59
パラレル・ソートおよびハッシュ結合のための一時表領域の作成.....	26-66
パラレルでの索引の作成.....	26-67
パラレル SQL 文の実行.....	26-68
EXPLAIN PLAN を使ったパラレル操作計画の表示.....	26-69
パラレル DML のその他の考慮事項.....	26-69

<b>フェーズ 4: パラレル実行のパフォーマンスの監視</b> .....	26-73
動的パフォーマンス・ビューを使ったパラレル実行のパフォーマンスの監視.....	26-73
セッション統計の監視.....	26-75
オペレーティング・システム統計の監視.....	26-78

## 27 パラレル実行のパフォーマンス上の問題について

<b>パラレル実行のパフォーマンス上の問題について</b> .....	27-1
メモリー、ユーザーおよびパラレル実行サーバー・プロセスの計算式.....	27-2
パラレル操作用バッファ・プール・サイズの設定.....	27-4
計算式のバランス化.....	27-5
例：メモリー、ユーザーおよびパラレル実行サーバーのバランス化.....	27-8
パラレル実行の領域管理問題.....	27-11
Oracle Parallel Server でのパラレル実行のチューニング.....	27-12
<b>パラレル実行のチューニングのヒント</b> .....	27-16
デフォルトの並列度の上書き.....	27-16
SQL 文の書直し.....	27-17
パラレルでの表の作成と移入.....	27-17
パラレルでの索引の作成.....	27-19
パラレル DML のヒント.....	27-20
パラレルでの表のリフレッシュ.....	27-23
コストベースの最適化でのヒントの使用.....	27-25
<b>問題の診断</b> .....	27-26
リグレッションの有無.....	27-27
計画の変更.....	27-28
パラレル計画.....	27-28
シリアル計画.....	27-28
パラレル実行.....	27-29
作業負荷の均一な分散.....	27-30

## 第 VI 部   マテリアライズド・ビュー

### 28 マテリアライズド・ビューを使用したデータ・ウェアハウス

<b>マテリアライズド・ビューを使用したデータ・ウェアハウスの概要</b> .....	28-1
データ・ウェアハウスのマテリアライズド・ビュー.....	28-2

分散コンピューティングでのマテリアライズド・ビュー.....	28-2
モバイル・コンピューティングでのマテリアライズド・ビュー.....	28-2
サマリー管理のコンポーネント.....	28-3
用語.....	28-5
<b>マテリアライズド・ビュー</b> .....	28-6
マテリアライズド・ビューのスキーマ・デザイン・ガイド.....	28-7
<b>データ・ウェアハウスのための Oracle Tools</b> .....	28-8
<b>スタート・ガイド</b> .....	28-9

## 29 マテリアライズド・ビュー

マテリアライズド・ビューの必要性.....	29-2
マテリアライズド・ビューの作成.....	29-3
命名.....	29-4
記憶領域特性.....	29-5
作成方法.....	29-5
クエリー・リライトでの使用.....	29-5
クエリー・リライトの制限事項.....	29-6
リフレッシュ・オプション.....	29-7
マテリアライズド・ビューのデータの定義.....	29-10
<b>既存のマテリアライズド・ビューの登録</b> .....	29-15
マテリアライズド・ビューのパーティション化.....	29-17
マテリアライズド・ビューのパーティション化.....	29-18
事前作成表のパーティション化.....	29-19
マテリアライズド・ビューに対する索引作成の選択.....	29-20
マテリアライズド・ビューの無効化.....	29-20
セキュリティの問題.....	29-21
データ・ウェアハウスにおけるマテリアライズド・ビューの使用のガイドライン.....	29-21
マテリアライズド・ビューの変更.....	29-22
マテリアライズド・ビューの削除.....	29-22

## 30 ディメンション

データ・ウェアハウスのディメンション.....	30-1
ディメンションの作成.....	30-3
複数の階層.....	30-5
正規化ディメンション表の使用方法.....	30-7

ディメンションの表示.....	30-8
ディメンションと制約.....	30-9
<b>ディメンションの検証.....</b>	<b>30-9</b>
ディメンションの変更.....	30-10
ディメンションの削除.....	30-11

## 31 クエリー・リライト

クエリー・リライトの概要.....	31-1
コストベースのリライト.....	31-2
クエリー・リライトの使用可能化.....	31-3
クエリー・リライトの初期化パラメータ.....	31-4
クエリー・リライトの使用可能化の権限.....	31-4
Oracle が問合せをリライトするとき.....	31-4
クエリー・リライトの方法.....	31-6
SQL テキスト照合リライト手法.....	31-6
一般クエリー・リライト手法.....	31-7
制約とディメンションが必要になるとき.....	31-17
クエリー・リライトの正確性.....	31-17
クエリー・リライトが行われたかどうか.....	31-18
Explain Plan.....	31-18
クエリー・リライトの制御.....	31-19
クエリー・リライトの使用のガイドライン.....	31-20
制約.....	31-20
ディメンション.....	31-20
外部結合.....	31-21
SQL テキスト照合.....	31-21
集計.....	31-21
グループ化条件.....	31-21
統計.....	31-22

## 32 マテリアライズド・ビューの管理

マテリアライズド・ビュー管理の概要.....	32-1
ウェアハウス・リフレッシュ.....	32-3
完全リフレッシュ.....	32-4
高速リフレッシュ.....	32-5

ウェアハウス・リフレッシュを使用したリフレッシュのヒント.....	32-8
並列性のための推奨初期化パラメータ.....	32-13
リフレッシュの監視.....	32-13
マテリアライズド・ビューをリフレッシュした後のヒント.....	32-13
<b>サマリー・アドバイザ</b> .....	32-14
構造統計の収集.....	32-15
動的作業負荷統計の収集.....	32-15
マテリアライズド・ビューの推奨.....	32-17
マテリアライズド・ビューのサイズの見積り.....	32-19
<b>マテリアライズド・ビューが使用されているかどうか</b> .....	32-20

## 索引

---

# はじめに

データベース・アプリケーション、データベースおよびオペレーティング・システムを調整することによって、Oracle のパフォーマンスを改善できます。このような調整を行うことを、"チューニング"と呼びます。Oracle を適切にチューニングすることにより、特定のアプリケーションやハードウェア構成でデータベースの最高のパフォーマンスを実現できます。

**注意：**『Oracle8i チューニング』では、Oracle8i および Oracle8i Enterprise Edition の各製品について、特徴および機能を説明しています。Oracle8i と Oracle8i Enterprise Edition の基本機能は同じです。ただし、いくつかの拡張機能は Enterprise Edition でのみ使用可能であり、それらの一部はオプションです。たとえば、アプリケーション・フェイルオーバーを使用するには、Enterprise Edition および Oracle Parallel Server が必要です。

## 対象読者

このマニュアルは、Oracle の運用、メンテナンスおよびパフォーマンスの担当者を対象としています。このマニュアルの対象読者は、データベース管理者、アプリケーション設計者またはプログラマなどです。このマニュアルを読むには、Oracle8i、オペレーティング・システムおよびアプリケーションの設計について理解しておく必要があります。

## このマニュアルの構成

このマニュアルは 6 部構成です。第 I 部では、チューニングについて解説し、チューニングの方法を説明します。第 II 部では、システム設計者およびプログラマがパフォーマンスについて計画する方法を説明します。第 III 部では、設計者および DBA 用の設計ツールについて説明します。第 IV 部では、本番稼働時にパフォーマンスを最適化する方法について説明します。第 V 部では、パラレル実行のチューニングおよび処理について説明します。第 VI 部では、マテリアライズド・ビューの使用法と最適化方法を説明します。6 つの部それぞれの内容は次のとおりです。

## 第 I 部：チューニングの概要

---

第 1 章「Oracle パフォーマンス・チューニングの概要」	この章では、チューニングの概要について説明します。パフォーマンス・チューニングおよびそのプロセスに関わる担当者の役割を定義します。
第 2 章「パフォーマンス・チューニングの方法」	この章では、推奨するチューニング方法の優先順のステップを示します。

---

## 第 II 部：設計者およびプログラマが行うアプリケーション設計のチューニング

---

第 3 章「アプリケーションおよびシステムのパフォーマンス特性」	この章では、Oracle データベースを使う各種アプリケーションと、各アプリケーションを設計するときに使用可能な方法と機能について説明します。
第 4 章「データベース操作のチューニング」	この章では、データベース操作のチューニングに関する基本事項を説明します。
第 5 章「アプリケーションの登録」	この章では、データベースにアプリケーションを登録する方法と、各登録モジュールまたはコード・セグメントについての統計を検索する方法について説明します。
第 6 章「データ・アクセス方法」	この章では、パフォーマンスを改善できるデータ・アクセス方法の概要を説明し、避ける必要のある状態を示します。
第 7 章「オプティマイザ・モード、プラン・スタビリティおよびヒント」	この章では、使用可能な最適化モードをいつ使うか、およびヒントをどのように使って Oracle のパフォーマンスを改善するかを説明します。
第 8 章「分散問合せのチューニング」	この章では、分散問合せのチューニングに関するガイドラインを説明します。
第 9 章「トランザクション・モード」	この章では、読込み一貫性を保つためのさまざまな方法を説明します。

---



### 第 III 部 : 設計者および DBA 用のアプリケーション設計ツール

---

第 10 章「SQL および共有 PL/SQL 領域の管理」	この章では、共有 SQL を使ってパフォーマンスを改善する方法を説明します。
第 11 章「データ・ウェアハウス・アプリケーションの最適化」	この章では、企業規模のデータ・ウェアハウスをチューニングするための統合された Oracle8i 機能について説明します。
第 12 章「診断ツールの概要」	この章では、本番システムの監視とパフォーマンス問題の判断に利用できる多様な診断ツールを紹介します。
第 13 章「EXPLAIN PLAN の使用方法」	この章では、SQL コマンド EXPLAIN PLAN の使用方法およびその出力のフォーマット方法を説明します。
第 14 章「SQL トレース機能と TKPROF」	この章では、Oracle Server で実行するアプリケーションの監視およびチューニングに役立つ、基本的な 2 つのパフォーマンス診断ツール、SQL トレース機能と TKPROF の使用方法を説明します。
第 15 章「Oracle Trace の使用方法」	この章では、Oracle Trace の使用方法の概要を示し、Oracle Trace 初期化パラメータについて説明します。

---

### 第 IV 部 : Oracle インスタンス・パフォーマンスの最適化

---

第 16 章「動的パフォーマンス・ビュー」	この章では、パフォーマンス・チューニングおよび非定型調査の両方に非常に役立つビューについて説明します。
第 17 章「システムのパフォーマンス問題の診断」	この章では、正しく設計されている既存のシステムでのパフォーマンス要因の概要を説明します。
第 18 章「CPU リソースのチューニング」	この章では、CPU リソースの問題を識別および解決する方法を説明します。
第 19 章「メモリー割当てのチューニング」	この章では、データベース構造体にメモリーを割り当てる方法について説明します。これらの構造体のサイズを適切に設定すると、データベースのパフォーマンスが大幅に向上します。
第 20 章「I/O のチューニング」	この章では、Oracle の機能が最大限に発揮されない原因となる I/O のボトルネックを回避する方法を説明します。
第 21 章「リソースの競合のチューニング」	この章では、パフォーマンスに影響を与える競合の検出方法および低減方法を説明します。
第 22 章「ネットワークのチューニング」	この章では、チューニングに影響するネットワークの問題を示し、配列インタフェースおよびバンド外ブレイク、その他のチューニング・テクニックの使用を指摘します。

---

第 23 章「マルチスレッド・サーバー・アーキテクチャのチューニング」	この章では、マルチスレッド・サーバー・アーキテクチャのコンポーネントをチューニングする方法を説明します。
第 24 章「オペレーティング・システムのチューニング」	この章では、Oracle のパフォーマンスを最適化するためにオペレーティング・システムをチューニングする方法を説明します。
第 25 章「インスタンス回復パフォーマンスのチューニング」	この章では、回復のパフォーマンスをチューニングする方法を説明します。

---

## 第 V 部: パラレル実行

---

第 26 章「パラレル実行のチューニング」	この章では、パフォーマンスを改善するためのパラレル実行機能の使用方法およびチューニング方法について説明します。パーティションの最適化方法についても説明します。
第 27 章「パラレル実行のパフォーマンス上の問題について」	この章では、パラレル実行のパフォーマンスに関する概念を説明し、パラレル実行のパフォーマンスに関する問題の解決方法を示します。

---

## 第 VI 部: マテリアライズド・ビュー

---

第 28 章「マテリアライズド・ビューを使用したデータ・ウェアハウス」	この章では、データ・ウェアハウスについて解説し、マテリアライズド・ビューを使用してデータ・ウェアハウスの操作を最適化する方法を説明します。
第 29 章「マテリアライズド・ビュー」	この章では、マテリアライズド・ビューの概要を説明します。
第 30 章「ディメンション」	この章では、マテリアライズド・ビューのディメンションの最適化について説明します。
第 31 章「クエリー・リライト」	この章では、クエリー・リライトによってマテリアライズド・ビューの使用を最適化する方法を説明します。
第 32 章「マテリアライズド・ビューの管理」	この章では、マテリアライズド・ビューの管理方法を説明します。

---

## 関連資料

このマニュアルでは、読者が『Oracle8i 概要』、『Oracle8i アプリケーション開発者ガイド 基礎編』および『Oracle8i 管理者ガイド』をすでに読んでいることを想定しています。

Oracle Enterprise Manager とそのオプションのアプリケーションの詳細は、次に示すマニュアルを参照してください。

『Oracle Enterprise Manager 概説』

『Oracle Enterprise Manager 管理者ガイド』

『Oracle Enterprise Manager Application Developer's Guide』

『Oracle Enterprise Manager: Introducing Oracle Expert』

『Oracle Enterprise Manager: Oracle Expert ユーザーズ・ガイド』

『Oracle Enterprise Manager パフォーマンス・モニタリングおよびプランニング・ガイド』。

このマニュアルでは、Oracle TopSessions および Oracle Monitor、Oracle Tablespace Manager の使用方法が説明されています。

## 表記法

このマニュアルで使用する次の表記上の規則について説明します。

- 本文
- 構文図とその表記法
- コード例

### 本文

本文で使用している規則について説明します。

### 大文字

アルファベットの大文字は、コマンドのキーワード、オブジェクト名、パラメータ、ファイル名などを示します。

たとえば、" プライベート・ロールバック・セグメントを作成する場合、その名前はパラメータ・ファイルの ROLLBACK\_SEGMENTS パラメータで指定する必要があります。" というように使用します。

### 構文図とその表記法

このマニュアルの構文図と表記記号は、SQL 文、関数、ヒントおよびその他の要素の構文を示します。この項では、構文図と例の参照方法と、それらに基づいて SQL 文を記述する方法を説明します。

## キーワード

キーワードとは、SQL 言語で特別な意味を持つ用語です。このマニュアルの構文図では、キーワードは大文字で表記されています。キーワードは、構文図に表記されているとおりに SQL 文に記述してください。ただし、キーワードは大文字でも小文字でもかまいません。たとえば、CREATE TABLE 文の CREATE キーワードは、CREATE TABLE の構文図に示されているとおりに文の先頭に記述してください。

## パラメータ

パラメータは、構文図の中のプレースホルダとして機能します。パラメータは小文字で表記されています。パラメータは通常、データベース・オブジェクトの名前または Oracle のデータ型名、式です。構文図にパラメータがある場合は、SQL 文では適切な型のオブジェクトまたは式を代入してください。たとえば、CREATE TABLE 文を記述する場合は、構文図の *table* パラメータの位置に、作成する表の名前 (EMP など) を使用します (本文では、パラメータ名はイタリック体で示されることに注意してください)。

次のリストに、このマニュアルで構文図に表記されているパラメータと、実際に記述する文でこれらのパラメータに代入する値の例を示します。

パラメータ	説明	例
<i>table</i>	代入値は、パラメータにより指定されているタイプのオブジェクトの名前です。	emp
<i>'text'</i>	代入値は、引用符で囲った文字リテラルです。	'Employee Records'
<i>condition</i>	代入値は、TRUE または FALSE に評価される条件です。	ename > 'A'
<i>date</i>	代入値は、日付定数または日付データ型の式です。	TO_DATE ('01-Jan-1996', DD-MON-YYYY')
<i>expr</i>	代入値は、任意のデータ型の式です。	sal + 1000
<i>integer</i>	代入値は整数値です。	72
<i>rowid</i>	代入値はデータ型が ROWID の式です。	00000462.0001.0001
<i>subquery</i>	代入値は、別の SQL 文に含まれている SELECT 文です。	SELECT ename FROM emp
<i>statement_name</i>	代入値は、SQL 文または PL/SQL ブロックの識別子です。	s1
<i>block_name</i>		b1

## コード例

SQL および SQL\*Plus のコマンドまたは文は、本文の段落とは別にモノスペース・フォントで示します。次に例を示します。

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

例の文には、カンマや引用符などの句読点が含まれていることがあります。例にあるすべての句読点は必須です。すべての例の文はセミコロン (;) で終了しています。アプリケーションによって異なりますが、セミコロンまたは他の終了記号が文の末尾に必要なときと必要でないときがあります。

例の文での大文字の語は Oracle SQL のキーワードを示します。ただし、文を発行するときは、キーワードには大文字と小文字の区別はありません。

例の文の小文字の語は、その例の場合にのみ指定する語を示します。たとえば、小文字は表または列、ファイルの名前を示す場合があります。



# 第 I 部

## チューニングの概要

第 I 部では、Oracle Server のチューニングの概要を説明します。第 I 部には次の章が含まれます。

- 第 1 章「Oracle パフォーマンス・チューニングの概要」
- 第 2 章「パフォーマンス・チューニングの方法」





---

# Oracle パフォーマンス・チューニングの概要

Oracle Server は高度なチューニングができる高性能ソフトウェア製品です。柔軟性が高く、データベースのパフォーマンスに影響する細かい調整ができます。システムをチューニングすることで、最もニーズに合うようにパフォーマンスを調整できます。

チューニングは、システムの計画と設計の段階から始まり、システムのライフ・サイクル全体を通して継続します。計画段階でパフォーマンスの問題を慎重に考慮しておけば、本番稼働中のシステム調整が容易になります。

このマニュアルでは、最初にチューニングの概要を示し、チューニングの方法について説明します。第 II 部では、システムの設計者やプログラマーを対象に、最適なパフォーマンスを計画する方法を説明します。第 III 部では、設計者やデータベース管理者向けの設計ツールについて説明します。第 IV 部では、本番稼働中にパフォーマンスを最適化する方法を説明します。第 V 部と第 VI 部では、それぞれパラレル実行とマテリアライズド・ビューについて説明します。

トピックは次のとおりです。

- [パフォーマンス・チューニングについて](#)
- [チューニング担当者](#)
- [パフォーマンス目標の設定](#)
- [ユーザー期待値の設定](#)
- [パフォーマンスの評価](#)

## パフォーマンス・チューニングについて

パフォーマンスを考慮する際は、この項で説明するいくつかの基本概念を理解している必要があります。

- 応答時間とスループットのトレードオフ
- 重要なリソース
- 過度な需要の影響
- 問題を軽減するための調整

### 応答時間とスループットのトレードオフ

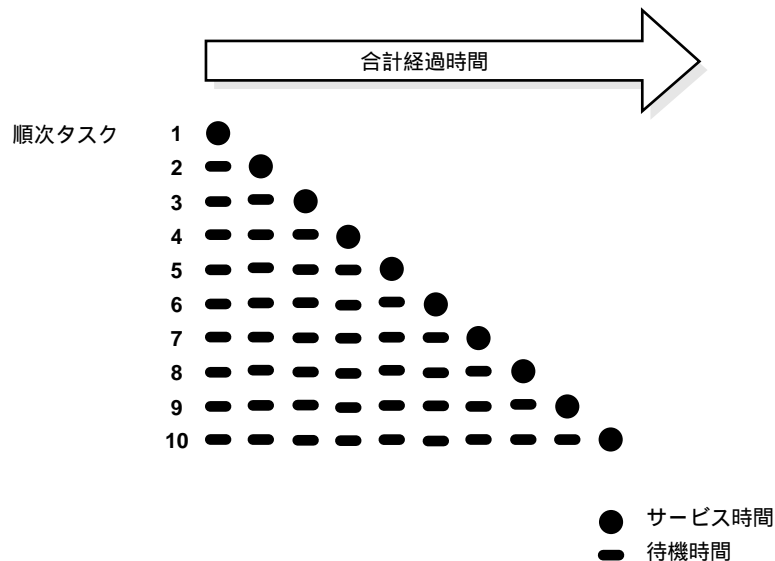
チューニングの目標はアプリケーションのニーズによって異なります。オンライン・トランザクション処理（OLTP）アプリケーションは、スループットを基準にパフォーマンスを定義します。OLTP アプリケーションは、非常に小さなトランザクションを 1 日に数千または数百万も処理する必要があります。これに対し、意思決定支援システム（DSS アプリケーション）は、応答時間を基準にパフォーマンスを定義します。DSS アプリケーションのユーザーは、非常に多様な種類の要求をデータベースに対して行います。ユーザーは、ある瞬間わずか数個のレコードをフェッチする問合せを入力し、次の瞬間には数 10 万個のレコードをさまざまな表からフェッチしてソートする大規模なパラレル問合せを入力することもあります。DSS 問合せを実行する多数のユーザーをアプリケーションがサポートする必要があるときは、スループットがより大きな問題となります。

#### 応答時間

応答時間はサービス時間に待機時間を加えたものと等しいので、サービス時間の短縮と待機時間の短縮の 2 つの方法でパフォーマンスを向上させることができます。

図 1-1 では、1 つのリソースに対して競合する 10 個の独立タスクがあります。

図 1-1 複数の独立タスクの順次処理



この例では、タスク 1 だけが待機することなく実行されています。タスク 2 は、タスク 1 が完了するまで待機しなければなりません。タスク 3 は、タスク 1 および 2 が完了するまで待機しなければなりません。以下同様です（この図では独立タスクを同サイズで示してありますが、実際のタスクのサイズはさまざまです）。

---

**注意：** パラレル処理では、複数のリソースがある場合は、より多くのリソースがタスクに割り当てられます。個々の独立タスクは、所有するリソースを使用して即時に実行されます。待機時間は発生しません。

---

## システム・スループット

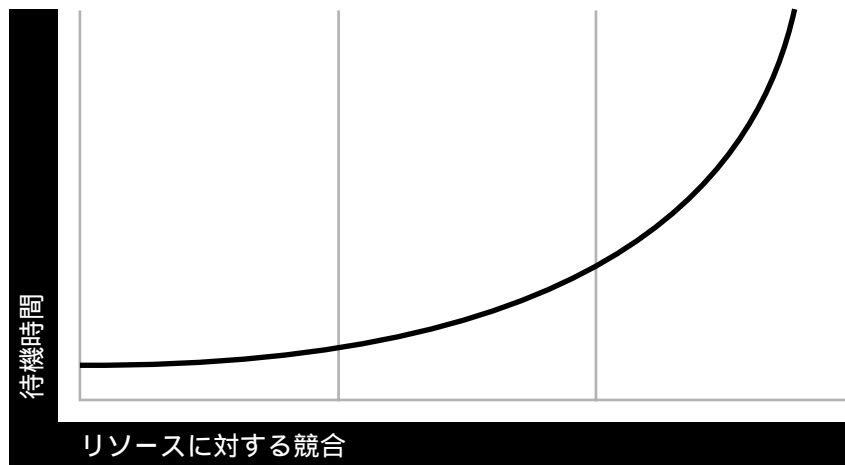
システム・スループットは、所定の時間内に達成される作業量を意味します。スループットを高める手法は 2 つあります。

- 同じリソースでより多くの作業を実行させる（サービス時間の短縮）
- 全体的な応答時間を短縮することによって、作業をより迅速に実行させる。これを行うために、待機時間を観察します。すべてのユーザーが待機しているリソースを複製できるかもしれません。たとえばシステムが CPU バウンドである場合には、CPU をさらに追加できます。

## 待機時間

タスクに対するサービス時間は変わらなくても、競合が増加すると待機時間が長くなります。1秒を要するサービスを多くのユーザーが待機している場合は、10番目のユーザーは1秒を要するサービスを9秒間待機しなければなりません。

図 1-2 リソースの競合の増加によって長くなる待機時間



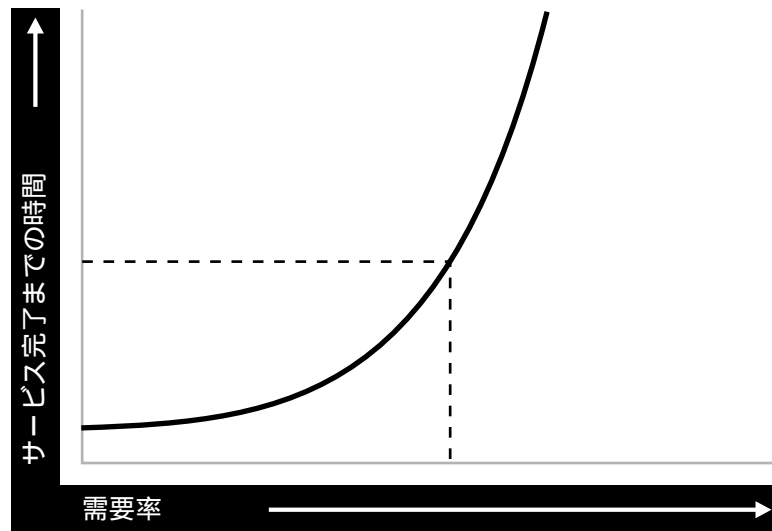
## 重要なリソース

CPU、メモリー、I/O 容量、ネットワーク帯域幅などのリソースは、サービス時間短縮の鍵です。リソースを追加すると、より高いスループットと迅速な応答時間を実現できます。パフォーマンスは次の項目に依存します。

- 使用可能なリソースの数
- リソースを必要とするクライアントの数
- クライアントがリソースを待機しなければならない時間の長さ
- クライアントがリソースを保持する時間の長さ

図 1-3 は、要求される単位数が増えるとサービス完了までの時間も長くなることを示しています。

図 1-3 サービス完了までの時間と需要率



この状態を処理するための方法は 2 つあります。

- 需要率を制限して、許容できる応答時間を維持する。
- 別の CPU やディスクなどの複数のリソースを追加する。

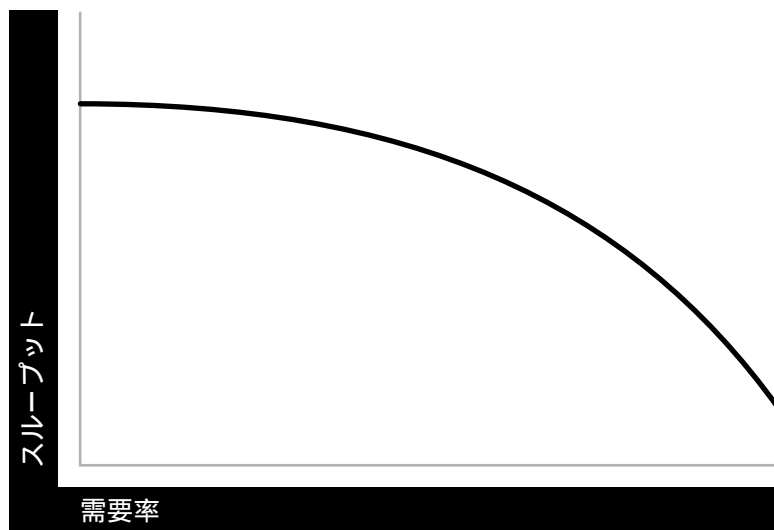
## 過度な需要の影響

過度の需要によって、次の問題が発生します。

- 応答時間の大幅な増加
- スループットの低下

需要率が達成可能なスループットを超える可能性がある場合は、需要リミッタが不可欠です。

図 1-4 長くなる応答時間 / 低下するスループット



## 問題を軽減するための調整

パフォーマンス問題は、次の調整を行うことで軽減できます。

### 単位消費量の調整

トランザクションあたりのリソース使用量を減らすこと、またはサービス時間を短縮することによって軽減できる問題もあります。または、トランザクションあたりの I/O を削減するなどの他のアプローチをとることもできます。

### 機能需要の調整

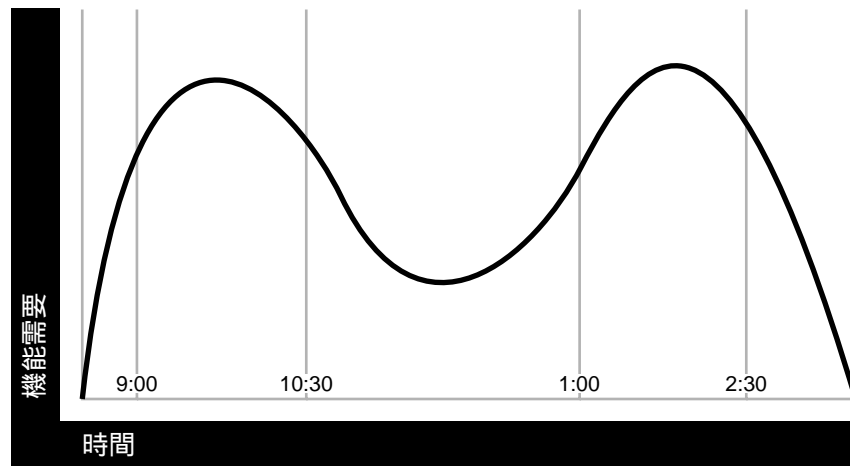
その他の問題は、作業の再スケジューリングまたは再配分によって解決できます。

### 容量の調整

リソースの追加または再割当てによっても問題を軽減できます。単一の CPU から対称型マルチプロセッサに移行することで複数の CPU の使用を開始すると、複数のリソースを使用できるようになります。

たとえば、システムが最もビジーな時間が午前 9:00 ~ 10:30 と午後 1:00 ~ 2:30 である場合は、より多くの容量を利用できる午後 2:30 以降に、バッチ・ジョブをバックグラウンドで実行できます。それによって、需要をより均一に分散できます。または、ピーク時の遅延を見込んでおくこともできます。

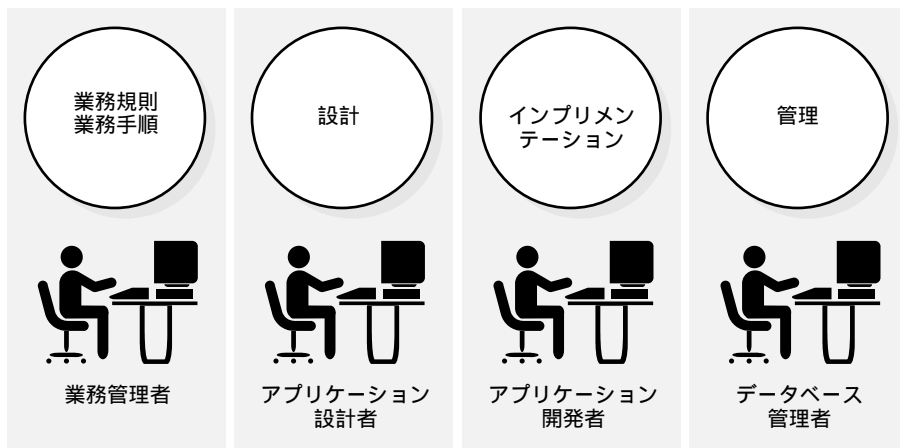
図 1-5 容量と機能需要の調整



## チューニング担当者

チューニング作業では、システムに携わるすべての人がなんらかの役割を受け持ちます。担当者がシステムの特性を説明し、それを文書化すれば、チューニング作業は非常に簡単になり、作業にかかる時間も大幅に短縮されます。

図 1-6 チューニング担当者



- 業務管理者は業務規則と手順を定義して再検証し、アプリケーション設計のために明確で適切なモデルを提供します。業務管理者は、システム全体のパフォーマンスに影響する可能性のある特定の種類の規則および手順を識別する必要があります。
- アプリケーションの設計者は、パフォーマンスの潜在的なボトルネックを考慮して設計する必要があります。アプリケーションの設計者はシステム的设计について説明し、担当者全員がアプリケーションでのデータの流れを理解できるようにします。
- アプリケーションの開発者は選択したインプリメンテーションの方針について説明し、モジュールや SQL 文を文のチューニング時に迅速かつ容易に識別できるようにします。
- データベース管理者 (DBA) はシステム・アクティビティを慎重に監視して文書化し、システムの異常なパフォーマンスを識別および修正できるようにします。
- ハードウェアとソフトウェアの管理者はシステムの構成を文書化して説明し、担当者全員がシステムを効率よく設計および管理できるようにします。

アプリケーションの開発と設計で行われた決定は、パフォーマンスに最も大きな影響を与えます。アプリケーションの本格稼働後は、データベース管理者がチューニングの主な責任を負うのが一般的です。

**関連項目：** パフォーマンス問題の識別と解決に役立つ問題解決方法については、[第 17 章「システムのパフォーマンス問題の診断」](#)を参照してください。



## パフォーマンス目標の設定

システムを設計またはメンテナンスする場合は、パフォーマンスについて特定の目標を設定し、チューニングを行う時期を認識するようにしてください。目標を設定しないで初期化パラメータや SQL 文を変更すると、場合によってはシステムのチューニングに無駄な時間を費やすことになります。

システムの設計時に特定の目標（たとえば、注文入力の実答時間を 3 秒未満にするなど）を設定してください。アプリケーションがこの目標に達しない場合は、妨げとなっているボトルネック（I/O の競合など）を識別し、その原因を判断し、対処措置を実行します。開発段階では、アプリケーションをテストして、設定したパフォーマンスの目標を達成しているかどうかを本番稼働の前に確認してください。

チューニングは常にトレードオフの連続です。ボトルネックを確認した後、希望の目標を実現するために他のシステム・リソースを犠牲にしなければならないことがあります。たとえば、I/O が問題となる場合には、メモリとディスクを増やさなければならないことがあります。増やすことができない場合には、目標のパフォーマンスを実現するためにシステムの並行性を制限しなければならないことがあります。ただし、パフォーマンスの目標を明確に定義すると、どの領域が最も重要であるかが明らかになるため、パフォーマンスの改善のために何を犠牲にしなければならないかを簡単に決定できます。

---

**注意：** パフォーマンス目標のためにデータの回復機能を犠牲にすることは決してしないでください。パフォーマンスも大切ですが、データの回復機能のほうがきわめて重要です。

---

## ユーザー期待値の設定

アプリケーションの開発者とデータベース管理者は、ユーザーの適切なパフォーマンス期待値を慎重に設定する必要があります。システムが特に複雑な操作を実行するときは、単純な操作を実行するときよりも実答時間が長くなります。このような場合には、実答時間が長くても不当ではありません。

DBA が 1 秒の実答時間を保証する場合は、これがどのように解釈される可能性があるかを検討してください。DBA はデータベース内での操作に 1 秒かかるという意味で、この目標を達成できると考えるかもしれませんが、しかし、ネットワークを介して問合せを行うユーザーに対してはネットワーク・トラフィックによる数秒の遅延が発生する可能性があり、ユーザーは期待する 1 秒以内に実答を受信できません。

## パフォーマンスの評価

パフォーマンス目標を明確に定義することで、パフォーマンス・チューニングが成功したことを簡単に判別できます。成功するかどうかは、ユーザー・コミュニティといっしょに設定した機能目標、および基準が達成されたかどうかを客観的に測定する担当者の能力、例外的な対処措置をとる担当者の能力に依存します。このチューニング・マニュアルの残りの章で

は、診断ツールおよび実行できる対処措置のタイプに関する情報とともに、チューニングの方法論を詳細に説明します。

パフォーマンス問題の解決を担当する DBA は、応答時間の決め手となるすべての要因を考慮する必要があります。パフォーマンス問題の認められた領域が実際の問題の要因でないことはよくあります。前述の例のユーザーはデータベースに問題があると結論づけるかもしれませんが、実際の問題はネットワークにあります。DBA は、すべてのパフォーマンスの問題がデータベースから生じていると単純に推測するのではなく、ネットワーク、ディスク、CPUなどを監視して問題の実際の要因を識別する必要があります。

本格稼働中のパフォーマンスを監視することで、適切にチューニングされたシステムを維持できます。アプリケーションのパフォーマンスの履歴を長期にわたって保持することで、有用な比較を行うことができます。広範囲の負荷レベルに対するリソースの使用状況を示すデータがあれば、拡張性を客観的に調査し、その詳細なパフォーマンス履歴に基づいて、将来に予想される負荷レベルのリソース要件を推測できます。

**関連項目：** [第 12 章「診断ツールの概要」](#)

---

## パフォーマンス・チューニングの方法

パフォーマンス・チューニングを成功させる鍵は、方法論を十分に計画することです。チューニング方針が異なれば、その効果も違ってきます。さらに、さまざまな用途に使用するシステム（オンライン・トランザクション処理システムや意思決定支援システム）には、さまざまなチューニング方法が必要です。

トピックは次のとおりです。

- [チューニングが最も効果的なとき](#)
- [優先順位付けされたチューニングのステップ](#)
- [チューニング方法の適用](#)

**関連項目：**[「Oracle Expert」](#) Oracle Expert は、データの収集と分析のプロセスを自動化し、データベース・チューニングの推奨事項、インプリメンテーション・スクリプト、パフォーマンス・レポートを提供します。

### チューニングが最も効果的なとき

最良の結果を得るために、システムのインプリメントを待ってからチューニングするのではなく、設計段階でチューニングしてください。

- [システムの設計中および開発中の事前チューニング](#)
- [本番システムを改善するための事後チューニング](#)

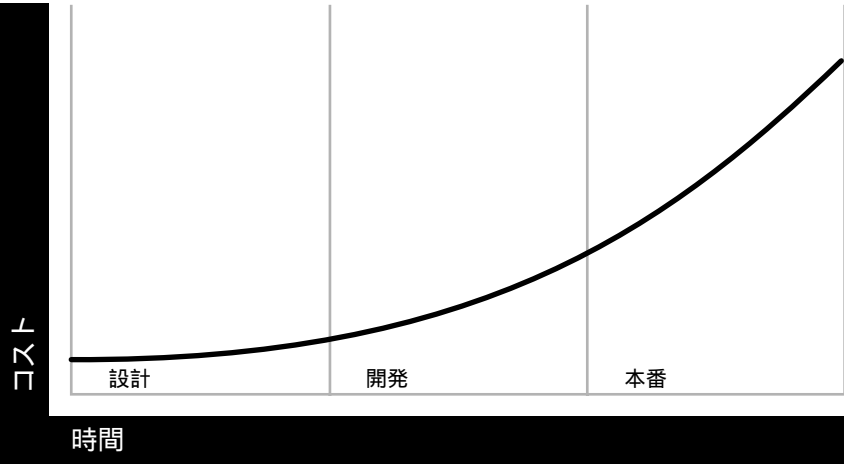
### システムの設計中および開発中の事前チューニング

チューニングの最も効果的なアプローチは、事前アプローチです。この章の 2-4 ページの[「優先順位付けされたチューニングのステップ」](#)で説明している方法を実行してください。

業務管理者は、アプリケーションの開発者と協力してパフォーマンス目標を確立し、パフォーマンスの現実的な期待値を設定しておく必要があります。これが設定されていると、

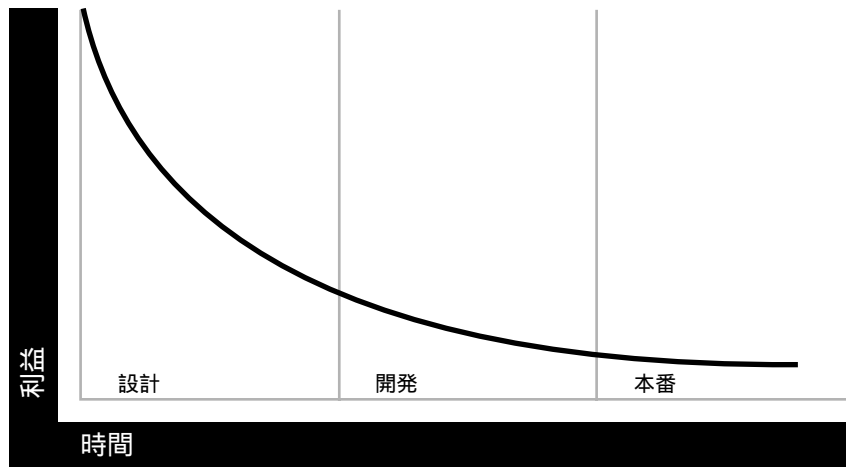
アプリケーションの開発者は、システム・リソースと使用可能な Oracle の機能をどのように組み合わせるとこれらのニーズを最もよく満たせるかを設計および開発中に判断できます。システムを適切に設計することで、アプリケーションのインプリメンテーションや本番使用時の管理費を最小限にできます。図 2-1 は、アプリケーションのライフ・サイクルで発生する相対的なチューニング・コストを示しています。

図 2-1 アプリケーションのライフ・サイクルでのチューニング・コスト



この図を補足するために、図 2-2 に、ライフ・サイクルを通したアプリケーションのチューニングによる相対的な利益は、かけたコストと反比例することを示します。

図 2-2 アプリケーションのライフ・サイクルでのチューニングの利益



最も効果的にチューニングを行えるのは設計段階です。設計時にチューニングを行うと、最小のコストで最大の利益が得られます。

## 本番システムを改善するための事後チューニング

チューニング作業は、応答時間が長いという苦情がユーザーから出た時点で始めるものではありません。一般的に、最も効果的なチューニング方法を採用するには、応答時間が長いという苦情が出た時点では遅すぎます。この時点でアプリケーションを完全に再設計したくない場合には、メモリーを割り当て直して I/O をチューニングすることで、パフォーマンスをわずかながら改善することができます。

たとえば、1 名の出納係と 1 名の支配人を雇用している銀行があるとします。この銀行には、20 ドルを超える預金の引き出しは支配人が承認しなければならないという業務規則があります。調査の結果、顧客の待ち行列が長いことがわかり、出納係を増やす必要があるという結論に達しました。出納係を 10 人増やすことはできますが、そうするとボトルネックが支配人の職務に移動することがわかります。しかし、この銀行は他にも支配人を雇うのは費用がかかりすぎると判断するかもしれません。この例では、既存の業務規則を使用してシステムをどれほど入念にチューニングしても、パフォーマンスを改善するには非常に費用がかかります。

あるいは、システムの拡張性を高めるために業務規則の変更が必要な場合もあります。150 ドルを超える預金の引き出しのみが管理者の承認を必要とするというように規則を変更すれば、拡張性のある解決策となります。この状態では、効果的なチューニングは、プロセスの末端ではなく最上位設計レベルで行えません。

問題に対応して既存の本番システムをチューニングすることは可能です。このアプローチをとるには、一番下の方法から上に向かって順番に作業を進め、ボトルネックを検出して修正

します。共通の目標は、所定のプラットフォーム上で Oracle をより高速にすることです。しかし、Oracle Server とオペレーティング・システムの両方が適切に稼働していることもあります。この場合は、パフォーマンスをさらに向上させるには、アプリケーションのチューニングまたはリソースの追加が必要です。その後でないと、Oracle が提供している多くの機能を十分に利用できません。これらの機能が、適切に設計されたシステムにおいて正しく使われると、パフォーマンスを大幅に改善できます。

使用方法によっては、適切に設計されたシステムのパフォーマンスが低下する可能性があります。したがって、本番でのチューニングは正しいシステム・メンテナンスの重要な部分を占めます。

**関連項目：** 第 IV 部の「Oracle インスタンス・パフォーマンスの最適化」では、CPU、メモリー、I/O、ネットワーク、競合およびオペレーティング・システムのチューニング方法を説明します。また、『Oracle8i 概要』では、パフォーマンスのボトルネックを迅速かつ容易に検出し、対処措置を判断できるよう、Oracle Server のアーキテクチャと機能に関する背景情報を説明します。

## 優先順位付けされたチューニングのステップ

推奨される Oracle データベースのチューニング方法を次のステップに示します。ステップには成果が大きい順に優先順位が付いています。つまり、パフォーマンスに最も大きな影響を与えるステップが最初にリストされています。したがって、最適な結果を得るには、設計と開発段階のチューニングからインスタンスのチューニングまで、リストされている順序でチューニングの問題を解決します。

ステップ 1: 業務規則のチューニング

ステップ 2: データ設計のチューニング

ステップ 3: アプリケーション設計のチューニング

ステップ 4: データベースの論理構造のチューニング

ステップ 5: データベース操作のチューニング

ステップ 6: アクセス・パスのチューニング

ステップ 7: メモリー割当てのチューニング

ステップ 8: I/O および物理構造のチューニング

ステップ 9: リソースの競合のチューニング

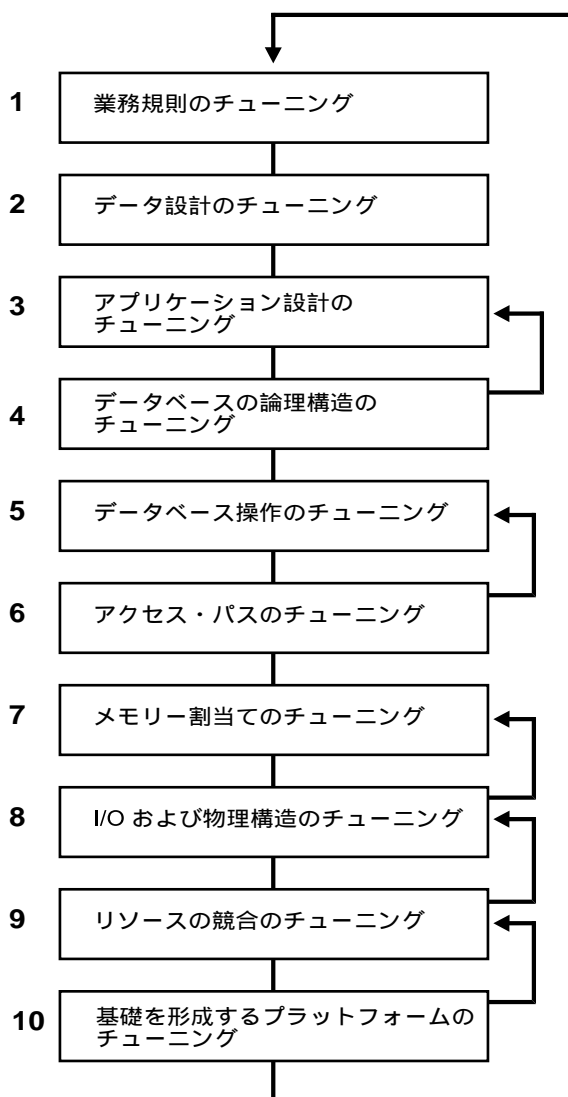
ステップ 10: 基礎を形成するプラットフォームのチューニング

これらのステップを完了した後で、データベースのパフォーマンスを再評価し、さらにチューニングが必要かどうかを判断します。

チューニングは反復的なプロセスです。後続のステップで得られたパフォーマンスの向上によって、前のステップでさらにパフォーマンスが改善されることがあるので、一連のチューニング・プロセスを再度実行すると有効な場合があります。

図 2-3 に、チューニング方法を示します。

図 2-3 チューニング方法



あるステップで行う決定は、後続のステップに影響する可能性があります。たとえば、ステップ5では、SQL 文をいくつか記述し直すことがあります。これらのSQL 文は、ステップ7で扱う解析とキャッシングの問題に大きく影響します。また、ステップ8でチューニングされるディスク I/O は、ステップ7でチューニングされるバッファ・キャッシュのサイズ



に依存します。図で示されている戻り先はステップ 1 ですが、どのステップからもそれより前の任意のステップに戻ることができます。

## ステップ 1: 業務規則のチューニング

パフォーマンスを最適化するには、業務規則を適応させなければならない場合があります。業務規則は、システム全体の上位レベルの分析と設計に関係します。マルチスレッド・サーバーをシステム全体で使用するかどうかなどの構成の問題は、このレベルで検討します。このように、計画担当者は、システムのパフォーマンス要件が具体的な業務ニーズに直接対応していることを確認します。

DBA が直面するパフォーマンス問題は、設計とインプリメンテーションの問題、すなわち不適切な業務規則が実際の原因となっていることがあります。設計者は、アプリケーションの業務機能を記述する際、必要以上に詳述することがあります。設計者は、単に実行する必要がある機能を文書化するのではなく、インプリメンテーションを文書化します。業務管理者がインプリメンテーションから業務の機能または要件を効果的に要約すれば、設計者は適切なインプリメンテーションをより自由に選択できます。

たとえば、小切手印刷の業務機能を考えます。実際の要件は人々にお金を払うことであり、紙の印刷は必ずしも要件ではありません。1 日に 100 万枚もの小切手を印刷することは非常に困難ですが、多くの口座振込みをテープに記録し、それを銀行に送付して処理してもらうことは比較的容易です。

業務規則は、システムがサポートできる同時ユーザー数およびトランザクション応答時間、オンラインに格納されるレコード数に関する現実的な見積もりと矛盾してはいけません。たとえば、低速の広域ネットワーク回線上で高度な対話型アプリケーションを実行することは無意味です。

同様に、インターネット・サービスのユーザーを勧誘する企業が、すべての新規加入者に 1 か月あたり 10 時間の無料時間を広告しているとします。1 日あたり 50,000 人のユーザーがこのサービスにサインアップすると、要求はクライアント / サーバー構成の容量をはるかに超えます。この企業は、複数層の構成を検討する必要があります。さらに、サインアップ・プロセスは単純でなければなりません。このプロセスでは、マルチスレッド・サーバーまたはトランザクション・モニター・アプローチを利用し、ユーザーからデータベースへの 1 つの接続、または専用接続を使用しない複数のデータベースへの接続のみが必要です。

## ステップ 2: データ設計のチューニング

データ設計の段階では、アプリケーションがどのデータを必要としているかを判断する必要があります。どのリレーションが重要であるか、その属性は何かを検討する必要があります。最後に、パフォーマンス目標を最もよく満たすように情報を構造化する必要があります。

一般に、データベース設計プロセスは、冗長なデータが存在しないようにするためにデータを分析する正規化段階を通ります。主キー以外では、1 つのデータ要素はデータベースの 1 箇所のみに記述する必要があります。ただし、データを正規化した後で、パフォーマンス上の理由からそれを冗長化することが必要になる場合があります。頻繁に使用されるサマリー

値をデータベースに保持する必要があると判断する場合などが考えられます。たとえば、アクセスされるたびにすべての行の合計金額を与えられた順序でアプリケーションに再計算させるかわりに、各オーダーの合計値を表す数字をデータベースに常に保持することを決定することがあります。この情報に迅速にアクセスするために、主キーと外部キーの索引を設定できます。

もう1つのデータ設計の考慮点は、データ競合の回避です。1,000人のユーザーがデータの0.5%にしかアクセスしない1テラバイトサイズのデータベースを考えてください。このデータの"ホット・スポット"は、パフォーマンス問題の原因となる可能性があります。

データへのアクセスのローカライズ、すなわちパーティション・レベル、プロセスおよびインスタンスレベルへのアクセスのローカライズも試してください。つまり、特定の値セット内でデータを要求するプロセスが特定のインスタンスに制限されるように、データへのアクセスをローカライズします。競合は、複数のリモート・プロセスが、ある特定のデータ・セットに同時にアクセスしようとしたときに始まります。

Oracle Parallel Server では同期点を探してください。同期点とは、一度に1つのプロセスを、順次に行う必要がある時点、またはアプリケーションの一部分です。たとえば、連番の注文番号が必要になるのは、欠陥のある設計から生じる同期点です。

競合を回避するのに役立つ Oracle8i の2つの機能強化のインプリメントも検討してください。

- データをパーティション化するかどうかの検討。
- ローカルまたはグローバル索引を使用するかどうかの検討。

**関連項目：** 第2章「パフォーマンス・チューニングの方法」、26-59ページの「フェーズ3: データベースの作成、移入およびリフレッシュ」、および『Oracle8i 概要』のパーティション化と索引の説明

## ステップ3: アプリケーション設計のチューニング

業務管理者とアプリケーションの設計者は、業務目標を効果的なシステム設計に変換する必要があります。業務処理は、システム内の特定のアプリケーション、またはアプリケーションの特定の部分に関係します。

高機能プロセス設計の例として、効果を十分に配慮したデータのキャッシングがあります。たとえば、流通アプリケーションでは、1日の初めに1回だけ税率を選択し、それをアプリケーション内にキャッシングできます。こうすることにより、1日の間に同じ情報を何度も検索する必要がなくなります。

このレベルでも、個々のプロセスの構成を検討できます。たとえば、モバイル・エージェントを使用して中央システムにアクセスするPCユーザーもいれば、中央システムに直接接続されているユーザーもいます。これらのユーザーは同じシステムを利用していますが、ユーザーのタイプごとにアーキテクチャは異なります。ユーザーは、異なるメール・サーバーと異なるバージョンのアプリケーションを必要とすることもあります。

## ステップ 4: データベースの論理構造のチューニング

アプリケーションとシステムを設計した後で、データベースの論理構造を計画できます。これは主に、データの索引付けに過不足がないことを確認するための索引設計の微調整に関係します。データ設計の段階（ステップ 2）では、主キーおよび外部キーの索引を決定します。論理構造設計の段階では、アプリケーションをサポートするその他の索引を作成できます。

競合によるパフォーマンス上の問題は、同じブロックへの挿入および不正な順序番号の使用に関係することがよくあります。索引の設計、方法および位置、また、順序ジェネレータとクラスタの使用には、特に注意してください。

**関連項目：** 6-2 ページの「[索引の使用方法](#)」

## ステップ 5: データベース操作のチューニング

Oracle Server 自体をチューニングする前に、SQL 言語、およびアプリケーションの処理速度を高めるために設計された Oracle の機能をアプリケーションが十分に活用していることを確認してください。アプリケーションのニーズに基づいて、次のような機能と手法を利用します。

- 配列処理
- Oracle オプティマイザ
- 行レベルのロック・マネージャ
- PL/SQL

SQL 文を効率的に記述するために Oracle の問合せ処理メカニズムについても理解する必要があります。第 7 章「[オプティマイザ・モード、プラン・スタビリティおよびヒント](#)」では、Oracle の問合せオプティマイザと、最適なパフォーマンスを実現する文の記述方法について説明します。また、この章では、オプティマイザの統計管理を概説し、プラン・スタビリティ機能での実行計画の保護について説明します。

**関連項目：** 第 IV 部「[インスタンスのパフォーマンスの最適化](#)」

## ステップ 6: アクセス・パスのチューニング

データに効率的にアクセスできるようにします。クラスタ、ハッシュ・クラスタ、B\* ツリー索引およびビットマップ索引の使用を検討してください。

効果的なアクセスの保証とは、索引の追加、すなわち特定のアプリケーション用の索引を追加してそれらを再度削除することを意味する場合があります。つまり、データベースの作成後に再び設計を分析しなければならないこともあります。そのとき、さらにデータを正規化したり代替索引を作成したりできます。アプリケーションをテストすると、要求される応答時間を達成していないことが明らかになる場合があります。その場合は、設計をさらに改善する方法を探してください。

**関連項目：** [第 6 章「データ・アクセス方法」](#)

## ステップ 7: メモリー割当てのチューニング

Oracle メモリー構造にメモリー・リソースを適切に割り当てることで、パフォーマンスに良い影響が与られます。

Oracle8i の共有メモリーは、次の構造に動的に割り当てられます。これらの構造はすべて共有ブールの一部です。共有ブールで使用可能なメモリーの合計量は明示的に設定しますが、それに含まれる各構造のサイズはシステムが動的に設定します。

- データ・ディクショナリ・キャッシュ
- ライブラリ・キャッシュ
- コンテキスト領域（マルチスレッド・サーバーを実行している場合）

次の構造には、メモリー割当てを明示的に設定できます。

- バッファ・キャッシュ
- ログ・バッファ
- 順序キャッシュ

メモリー・リソースを適切に割り当てると、キャッシュ・パフォーマンスの改善、SQL 文の解析の削減およびページングとスワッピングの削減を達成できます。

プロセスのローカル領域には次のものがあります。

- コンテキスト領域（マルチスレッド・サーバーを実行していないシステム）
- ソート領域
- ハッシュ領域

ページングまたはスワッピングの原因となるので、システム・グローバル領域（SGA）にはマシンの物理メモリーを大量に割り当てないように注意してください。

**関連項目：** [第 19 章「メモリー割当てのチューニング」](#) および『Oracle8i 概要』のメモリー構造とプロセスに関する情報

## ステップ 8: I/O および物理構造のチューニング

多くのソフトウェア・アプリケーションのパフォーマンスは、ディスク I/O によって低下する傾向があります。ただし、Oracle Server は、I/O によってパフォーマンスが必ずしも制限されることなく設計されています。I/O と物理構造のチューニングには、次の手順が関係します。

- I/O を分散し、ディスクの競合を回避するためにデータを分散する。
- 最もアクセスしやすいようにデータをデータ・ブロックに格納する。空きリストの正しい数、および PCTFREE と PCTUSED に適切な値を設定する。
- 大容量 OLTP アプリケーションのパフォーマンスに悪影響を及ぼす表の動的拡張を避けるために、データに対して十分な大きさを持つエクステントを作成する。
- ロー・デバイスの使用方法を評価する。

関連項目： [第 20 章「I/O のチューニング」](#)

## ステップ 9: リソースの競合のチューニング

複数の Oracle ユーザーが同時実行処理を行うと、Oracle リソースの競合が発生することがあります。競合が発生すると、プロセスはリソースが使用可能になるまで待機しなければなりません。次の種類の競合を削減するように注意してください。

- ブロックの競合
- 共有プールの競合
- ロックの競合
- ping (パラレル・サーバー環境)
- ラッチの競合

関連項目： [第 21 章「リソースの競合のチューニング」](#)

## ステップ 10: 基礎を形成するプラットフォームのチューニング

基礎を形成するシステムのチューニング方法の詳細は、プラットフォーム固有の Oracle マニュアルを参照してください。たとえば、UNIX ベースのシステムでは、次のものをチューニングする必要があります。

- UNIX バッファ・キャッシュのサイズ
- 論理ボリューム・マネージャ
- 各プロセスのメモリーとサイズ

関連項目： [第 24 章「オペレーティング・システムのチューニング」](#)

## チューニング方法の適用

この項では、チューニング方法の適用方法を説明します。

- [チューニングの明確な目標の設定](#)
- [最小反復テストの作成](#)
- [仮説のテスト](#)
- [記録と自動テスト](#)
- [一般的な過ちの回避](#)
- [目標を達成したときのチューニングの停止](#)
- [目標達成の証明](#)

### チューニングの明確な目標の設定

最初に明確な目標を設定するまではチューニングを開始しないでください。何が " 成功 " かを定義しておかないと、成功することはできません。

" できるだけ処理速度を高める " というのも目標のように思えますが、これを達成したかどうかを判断するのは非常に困難です。チューニングの結果が、基礎になる業務要件を満たしているかどうかを判断するのは、さらに困難です。よりわかりやすい目標の表現は、次のような文です。"20 人のオペレータが 1 時間当たり 20 個のオーダーを入力する必要があり、シフトの最後の 30 分間に梱包一覧表を作成する必要がある。"

個々のチューニングの測定単位を検討するときには、目標を念頭に置いてください。パフォーマンス上の利点を目標に照らして検討してください。

各目標は対立する可能性があることも忘れないでください。たとえば、特定の SQL 文のパフォーマンスを最適化するために、データベース内で同時に実行されている他の SQL 文を犠牲にしなければならないことがあります。

### 最小反復テストの作成

一連の最小反復可能テストを作成します。たとえば、パフォーマンス問題の原因となっている単一の SQL 文を識別する場合には、パフォーマンスの差異を統計的に見極められるように、( SQL トレース機能または Oracle Trace が使用可能になっている ) SQL\*Plus 内でその SQL 文のオリジナル・バージョンと改訂バージョンの両方を実行してください。多くの場合、パフォーマンス問題の原因であった 1 つの SQL 文を識別するだけでチューニングが成功します。

たとえば、4 時間の実行を 2 時間に短縮する必要があるとします。その場合は、本番環境に似たテスト環境で試します。たとえば、500 の部門すべてを処理するかわりに 1 つの部門を処理するなど、追加制限条件を加えられます。テスト・ケースは、改善されたことを直感的に判断できるように、1 分より長く実行するのが理想的です。ただし、5 分は超さない方がいいでしょう。また、タイミング機能を使用してテスト実行を測定する必要があります。

## 仮説のテスト

最小反復テストが設定されており、テストの実行および結果の要約とレポート作成を行うスクリプトがあると、さまざまな仮説をテストして効果を調べることができます。

Oracle のキャッシング・アルゴリズムでは、最初にデータがキャッシュされるときには、次にメモリーの同じデータにアクセスするときよりもオーバーヘッドが大きいことに注意してください。したがって、2 つのテストを順番に実行する場合は、2 番目のテストの方が 1 番目よりも速く実行されます。本来ならディスクから読み込む必要のあるデータがキャッシュから取り出されることで、時間が短縮されるからです。

## 記録と自動テスト

テストのスクリプトに記録処理を組み込み、各変更の効果を記録してください。テストも自動化する必要があります。自動化には、次の利点があります。

- チューニング担当者がテストを迅速に実施できるようにしてコスト効率を高めるため
- テストする各仮説について、テストが同じ方法で同じ機器を使用して実行されるようにするため

また、システム・パフォーマンスの観察から得たテスト結果は、目標データに照らして念入りに調べたうえで受け入れる必要があります。

## 一般的な過ちの回避

経験の少ないチューニング担当者に多い過ちは、問題の原因に関する予想にいつまでも執着することです。次に多い過ちは、さまざまな解決策を手当たりしだいに試行することです。

解決のプロセスを吟味する良い方法は、自分が問題と考えることを論理的に文章化してみることです。そのようにして自分の考えを表現するだけで、間違いを見つけられることがよくあります。最もよい結果を得るには、チームに相談してパフォーマンス問題を解決するようにしてください。パフォーマンスのチューニング担当者はアプリケーションを詳細に知らなくても SQL 文をチューニングできますが、チームには、アプリケーションを理解している人、および SQL のチューニング担当者が考案した解決策を検証できる人が含まれている必要があります。

### 性急な解決策の回避

推測でシステムに変更を加えないように注意してください。また、仮説を立てたら、それをグローバルにインプリメントする前に十分検討する必要があります。性急な行動をとると、システムのパフォーマンスが大幅に低下し、そのために環境の一部をバックアップから再作成しなければならなくなるおそれがあります。

## 先入観の回避

チューニングの問題に取り組むときは、先入観を持たないようにしてください。ユーザーにパフォーマンスの問題を知らせてもらうようにしてください。ただし、問題が存在する理由をユーザーが知っているとは期待しないでください。

たとえば、あるユーザーは、システム・メモリーの深刻な問題を長い間抱えていました。午前中はシステムが調子よく稼働しますが、午後になるとパフォーマンスが急速に低下します。システムをチューニングするコンサルタントは、PL/SQL メモリーのリークが原因であると告げられました。結局のところ、これはまったく問題ではありませんでした。

ユーザーは、メモリーが 64MB でユーザー数が 20 人のマシンで、SORT\_AREA\_SIZE を 10MB に設定していました。ユーザーがシステムにログオンし、最初にソートを実行したときに、ユーザーのセッションはソート領域に割り当てられていました。ソート領域は、各セッションが継続している間中保持されていました。そのため、システムには 200MB の仮想メモリーの負荷がかかり、しかたなくスワッピングとページングが行われていました。

## 目標を達成したときのチューニングの停止

チューニングの目標を立てることの最大の利点の 1 つは、成功を定義できることです。ある点を超えると、システムのチューニングを継続してもコスト効率が悪くなります。

## 目標達成の証明

チューニング担当者は、パフォーマンス目標を達成したことに自信を持っているかもしれませんが。それでもなお、チューニング担当者はこれを 2 つのコミュニティに証明する必要があります。

- 問題の影響を受けるユーザー
- アプリケーションの成功に関する責任者

第 II 部では、設計者とプログラマを対象に、アプリケーション設計のチューニングについて説明します。



# 第 II 部

---

## 設計者およびプログラマのための アプリケーション設計のチューニング

第 II 部では、最適なパフォーマンスを得るためのアプリケーションの設計とチューニングに関する背景情報を説明します。第 II 部には次の章が含まれます。

- [第 3 章「アプリケーションおよびシステムのパフォーマンス特性」](#)
- [第 4 章「データベース操作のチューニング」](#)
- [第 5 章「アプリケーションの登録」](#)
- [第 6 章「データ・アクセス方法」](#)
- [第 7 章「オプティマイザ・モード、プラン・スタビリティおよびヒント」](#)
- [第 8 章「分散問合せのチューニング」](#)
- [第 9 章「トランザクション・モード」](#)
- [第 10 章「SQL および共有 PL/SQL 領域の管理」](#)
- [第 11 章「データ・ウェアハウス・アプリケーションの最適化」](#)



---

# アプリケーションおよびシステムの パフォーマンス特性

この章では、Oracle データベースを使用するさまざまなアプリケーションおよびシステムと、各アプリケーションを設計するときに使用可能な方法と機能について説明します。トピックは次のとおりです。

- [アプリケーションのタイプ](#)
- [Oracle の構成](#)

## アプリケーションのタイプ

Oracle Server 上では、数千種類におよぶアプリケーションを作成できます。この項では、最も一般的なタイプを分類し、各アプリケーションの設計時の考慮事項を説明します。各カテゴリでは、該当するシステムにとって重要なパフォーマンス問題を記述しています。

- [オンライン・トランザクション処理 \(OLTP\)](#)
- [データ・ウェアハウス](#)
- [多目的アプリケーション](#)

**関連項目：** これらのトピックの詳細と、その機能をインプリメントする方法は、『Oracle8i 概要』、『Oracle8i アプリケーション開発者ガイド 基礎編』および『Oracle8i 管理者ガイド』を参照してください。

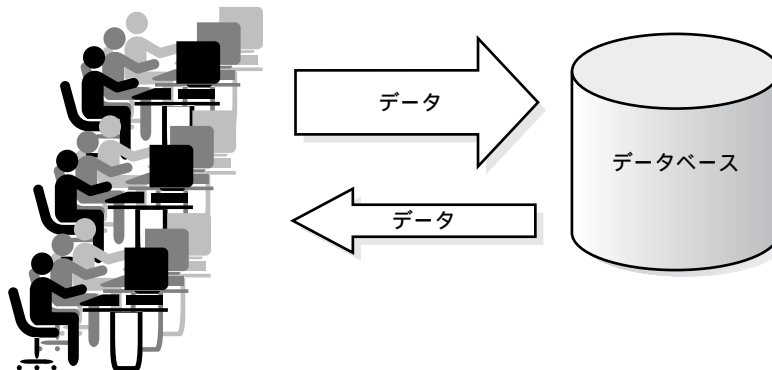
## オンライン・トランザクション処理 (OLTP)

オンライン・トランザクション処理 (OLTP) アプリケーションは、高スループットの挿入 / 更新処理集中型のシステムです。これらのシステムの特徴は、数百のユーザーが同時にアクセスするデータの量が増えることです。OLTP アプリケーションの例としては、航空券予約システム、大型の注文入力システム、銀行のシステムなどがあります。OLTP システムは、

可用性（ときには週 7 日、1 日 24 時間の可用性）、処理速度（スループット）、並行性および回復可能性に重点を置いています。

図 3-1 に、OLTP アプリケーションと Oracle Server との相互関係を示します。

図 3-1 オンライン・トランザクション処理システム



OLTP システムの設計時には、同時実行ユーザーが多数いることが原因でシステムのパフォーマンスが低下することがないようにしてください。また、索引やクラスタは挿入および更新アクティビティの速度が低下するので、索引やクラスタを過度に使用しないでください。

OLTP システムのチューニングで重要な要素を次に示します。

- ロールバック・セグメント
- 索引、クラスタ、ハッシング
- ディスクリット・トランザクション
- データ・ブロック・サイズ
- 表およびロールバック・セグメントへの領域の動的割当て
- トランザクション処理モニターおよびマルチスレッド・サーバー
- 共有プール
- 適切にチューニングされた SQL 文
- 整合性制約
- クライアント / サーバー・アーキテクチャ
- 動的に変更可能な初期化パラメータ

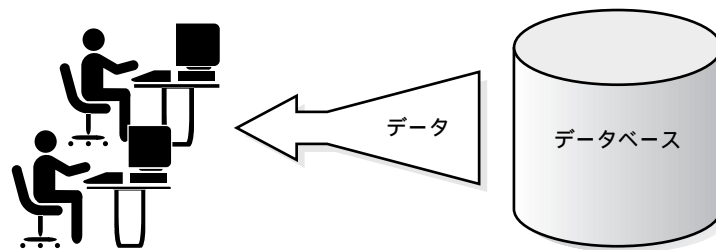
- プロシージャ、パッケージおよびファンクション

**関連項目：** これらのトピックの詳細は、『Oracle8i 概要』および『Oracle8i 管理者ガイド』を参照してください。システムを設計する前にこれらのトピックについてよく読み、特定の状況ではどの機能が有用であるかを判断してください。

## データ・ウェアハウス

データ・ウェアハウス・アプリケーションは、通常、大量の情報をユーザー定義のレポートに変換します。意思決定支援アプリケーションは、OLTP アプリケーションで収集された大量のデータに対して問合せを実行します。意思決定担当者は、組織が採用する方針を決定する際に、これらのアプリケーションを使用します。図 3-2 に、意思決定支援アプリケーションと Oracle Server との相互関係を示します。

図 3-2 データ・ウェアハウス・システム



意思決定支援システムの例としては、統計調査によって収集した情報を基に、消費者の消費パターンを判断するマーケティング・ツールがあります。統計データは収集されてシステムに入力され、市場調査員がこのデータを問い合せて、どの地域でどの商品が一番売れているかを判断します。このレポートは、各地域でどの商品を仕入れて売り出すかを判断する際に役立ちます。

データ・ウェアハウス・システムの重要な目標は、応答時間および正確性、可用性です。意思決定支援システムの設計時には、大量のデータに対する問合せが適切な時間枠内で実行されるようにしてください。意思決定担当者は、その日ごとのレポートを必要とすることがよくあるので、レポートを一晩で完成させる必要があります。

意思決定支援システムでパフォーマンスにとって重要なことは、問合せを適切にチューニングすることと、索引およびクラスタ、ハッシングを適切に使用することです。意思決定支援システムのインプリメントとチューニングで重要なトピックを次に示します。

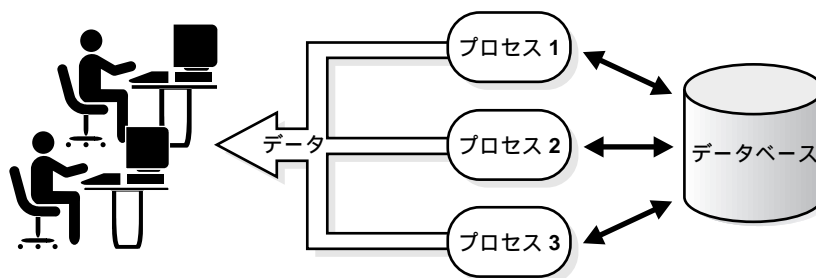
- マテリアライズド・ビュー
- 索引（B\* ツリーおよびビットマップ）
- クラスタ、ハッシング

- データ・ブロック・サイズ
- パラレル実行
- スター問合せ
- オプティマイザ
- 問合せでのヒントの使用方法
- SQL 文の PL/SQL 関数

データ・ウェアハウス・システムでの応答時間を改善する 1 つの方法は、パラレル実行を使用することです。この機能により、単一の SQL 文を処理するために、複数のプロセスを同時に実行できます。Oracle では、処理を多数のプロセスに分割することによって、1 つのサーバーだけで処理する場合よりも高速に、複雑な文を実行できます。

図 3-3 に、パラレル実行を示します。

図 3-3 パラレル実行処理



パラレル実行を使うと、意思決定支援アプリケーションまたは超大規模データベース環境に関連するデータ中心の操作でパフォーマンスを大幅に改善できます。また、これは OLTP 処理に役立つ場合もあります。

対称型マルチプロセッシング (SMP) システム、クラスタ化システムまたは大規模パラレル・システムでは、パラレル実行によって最高のパフォーマンスが達成されます。これは、1 つのシステム上の複数の CPU に操作を効率的に分割できるためです。

パラレル実行は、ハードウェア・リソースを追加したときにシステムのパフォーマンスを向上させる際に役立ちます。システムの CPU とディスク・コントローラにすでに大きな負荷がかかっている場合は、パラレル実行によってパフォーマンスを向上させる前にシステムの負荷を軽減する必要があります。

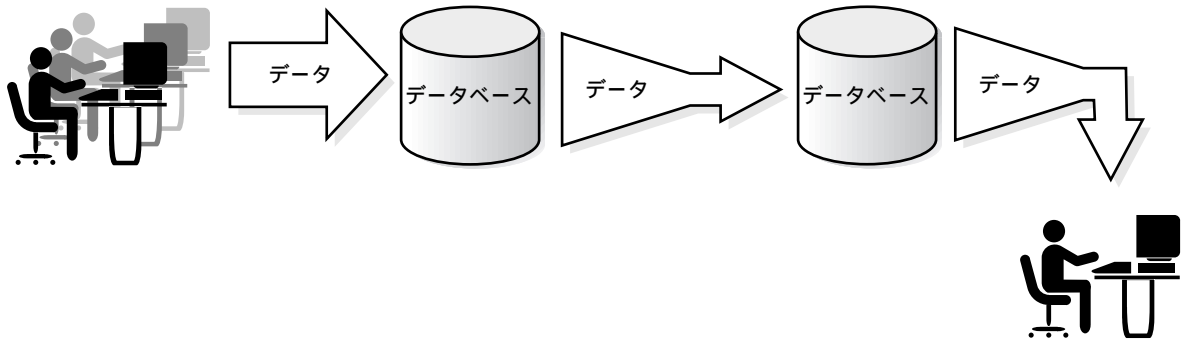
**関連項目：** Oracle のデータ・ウェアハウス機能については、[VI 項の「マテリアライズド・ビュー」](#)と第 11 章「[データ・ウェアハウス・アプリケーションの最適化](#)」を参照してください。パラレル実行のパフォーマンスについては、[第 26 章「パラレル実行のチューニング」](#)で説明します。パラレル実行の一般的な情報は、『Oracle8i 概要』に記載されています。

## 多目的アプリケーション

多くのアプリケーションは、いくつかの構成と Oracle オプションに依存しています。チューニング担当者は、アプリケーションが実行するアクティビティのタイプを決定して、そのアクティビティに最も適した機能を判断する必要があります。多目的構成の代表的な例としては、OLTP とデータ・ウェアハウス・システムを組み合わせたものがあります。OLTP アプリケーションによって収集されたデータがデータ・ウェアハウス・システムの入力データになることがよくあります。

[図 3-4](#) に、複数の構成とアプリケーションから Oracle Server にアクセスする様子を示します。

図 3-4 OLTP/ データ・ウェアハウスを組み合わせたシステム



OLTP システムとデータ・ウェアハウス・システムを組み合わせたシステムの一例として、小売店から収集した情報に基づいて、消費者の消費パターンを判断するマーケティング・ツールがあります。小売店は毎日の販売記録からデータを収集し、市場調査員はこのデータを問い合せて、どの地域でどの商品が一番売れているかを判断します。このレポートは、各小売店が特定の品目の在庫量を判断するのに使います。

この例では OLTP システムとデータ・ウェアハウス・システムの両方が同じデータベースを使用しますが、この 2 つのシステムが競合すると、パフォーマンス上の問題が発生する可能性があります。これを解決するには、OLTP データベースに小売店から収集したデータを格納し、次にそのデータのイメージをもう 1 つのデータベースにコピーします。データ・ウェアハウス・アプリケーションは、このもう 1 つのデータベースに問い合わせます。この構成で

は、データ・ウェアハウス・アプリケーションの正確性が少し低下する可能性があります（データが 1 日に 1 回しかコピーされないという点で）。そのかわり、どちらのシステムでもパフォーマンスが大幅に向上します。

複数のシステムを組み合わせたシステムでは、最も重要な目標を判断する必要があります。システム全体で適切なパフォーマンスを実現するには、優先度の低い目標の達成について妥協する必要がある場合があります。

## Oracle の構成

使用可能なハードウェアおよびソフトウェアに応じてシステムを構成できます。基本構成は次のとおりです。

- [分散システム](#)
- [Oracle Parallel Server](#)
- [クライアント / サーバー構成](#)

使用しているアプリケーションとオペレーティング・システムによって異なりますが、これらのいずれか、またはいくつかの組合せを使用することによって、チューニング担当者のニーズに対応できます。

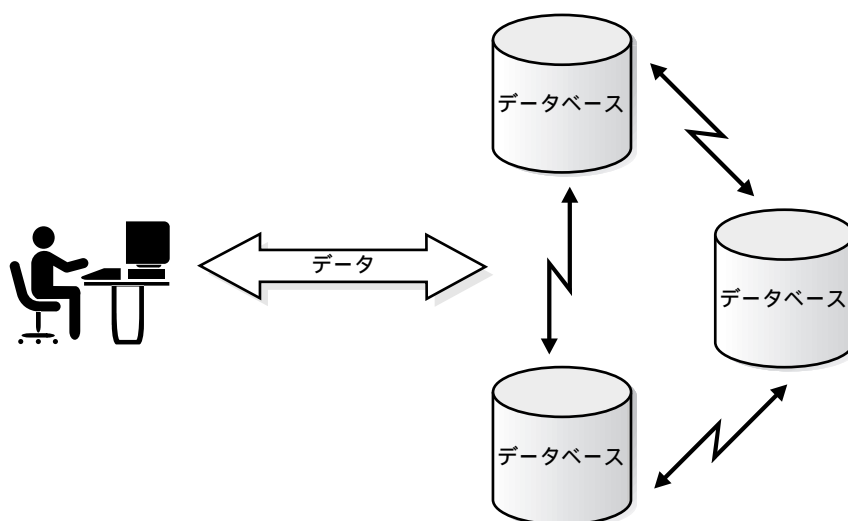
## 分散システム

分散アプリケーションでは、複数のマシン上にある複数のデータベースにデータを分散します。複数の小型サーバー・マシンを使用すると、大型の中央サーバーを 1 台使用するよりも安価であり、かつ高度な柔軟性を実現できます。分散データベース構成は、小型で高性能なサーバー・マシンと安価な接続オプションを利用します。分散システムを使用すると、データを複数のサイトに格納でき、さらにそのような配置を意識しないで、それぞれのサイトからすべてのデータにアクセスできるようになります。

[図 3-5](#) に、Oracle Server の分散データベース構成を示します。



図 3-5 分散データベース・システム



分散データベース・システムの例としては、国内の複数の地域に注文入力オペレータがいる通信販売アプリケーションがあります。入力オペレータは中央在庫データベースのコピーにアクセスしますが、入力オペレータはローカルの注文入力システム上で、操作をローカルに実行します。ローカルで入力された注文は、中央の出荷部門に毎日転送されます。ローカルの注文入力システムは、その同じ地域の顧客と対応する入力オペレータには便利です。会社全体の在庫データベースは中央にあります。これは通信販売機能の処理に便利です。

分散データベース・システムは、可用性、正確性、並行性および回復性に重点を置いています。分散システムの設計時には、データの位置が最も重要になります。ローカル・クライアントが最も頻繁に使用するデータに迅速にアクセスできるようにし、リモート操作は頻繁に発生しないようにしてください。レプリケーションは、データ位置の問題を扱う 1 つの方法です。分散データベース・システムの設計で重要なトピックを次に示します。

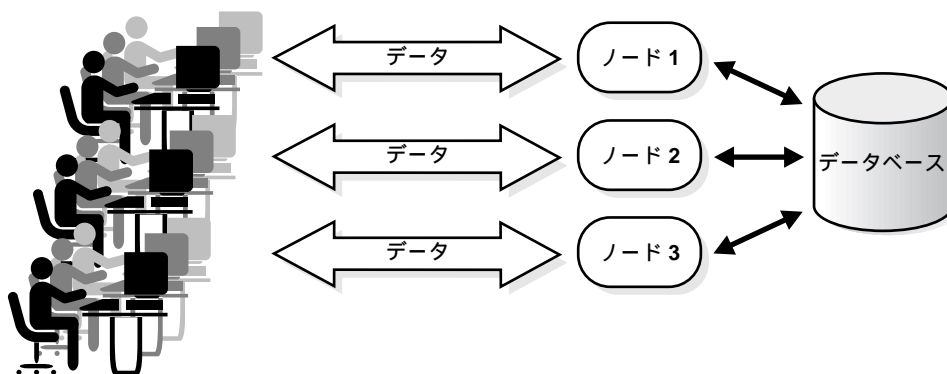
- ネットワーク構成
- 分散データベースの設計
- 対称型レプリケーション
- 表のスナップショットおよびスナップショット・ログ
- プロシージャ、パッケージおよびファンクション

**関連項目：**『Oracle8i 分散システム』、『Oracle8i レプリケーション・ガイド』および第 8 章「分散問合せのチューニング」

## Oracle Parallel Server

Oracle Parallel Server (OPS) は、クラスタ化システムまたは大規模パラレル・システムで使用可能です。Parallel Server を使用すると、複数のマシン上の個別インスタンスが、すべて同一のデータベースにアクセスできるようになります。この構成では、データのスループットが大幅に向上します。図 3-6 に、Oracle Parallel Server のオプションを示します。

図 3-6 Oracle Parallel Server



OPS を構成する場合に最も考慮すべき点は、複数ノード間でのデータの競合を防ぐことです。OPS のキャッシュ・フュージョン機能を使用すると、データ競合の起こるノード間でのブロックの ping を最小限にできますが、それでもデータの適切なパーティション化に努める必要があります。これは、データの整合性をとるために各ノードがまずそのデータのロックを取得しなければならない write/write の競合の場合、特に重要です。

複数のノードが DML 操作の同一のデータにアクセスする場合、データは最初に必ずディスクに書き込まれる必要があり、その後で次のノードがロックを取得できるようになります。このような競合によってパフォーマンスが著しく低下します。このようなシステムでは、パフォーマンスの最適化のために、複数のノード間でデータを効率よくパーティション化する必要があります。Oracle は非ロック問合せロジックを使用しているため、読み込み専用データは、ロックの競合の問題が発生することなく OPS 構成内のすべてのインスタンスで効率よく共有できます。

**関連項目：**『Oracle8i Parallel Server 概要および管理』

## クライアント / サーバー構成

クライアント / サーバー・アーキテクチャでは、システムの作業がクライアント（アプリケーション）マシンとサーバー（ここでは Oracle Server）に分散されます。通常、クライアント・マシンとは、Oracle Server を収容している大型のサーバー・マシンに接続してグラフィカル・ユーザー・インタフェース（GUI）アプリケーションを実行するワークステーションです。

**関連項目：** 複数層システムの詳細は、18-9 ページの「[システム・アーキテクチャの変更による CPU の問題の解決](#)」を参照してください。



---

## データベース操作のチューニング

いくつかの Oracle Tools とアプリケーションでは、構造化問合せ言語 (SQL) の使用を単純化したり、見えないようにしたりしていますが、すべてのデータベース操作は SQL を使用して遂行されます。この章では、データベース操作のチューニングに関する問題を、SQL の観点から概説します。

- [チューニングの目標](#)
- [データベース操作のチューニング方法](#)
- [SQL 文のチューニングのアプローチ](#)

**関連項目：** PL/SQL 文のチューニングの詳細は、『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。

# チューニングの目標

この項では、次のトピックについて説明します。

- シリアル SQL 文のチューニング
- パラレル実行のチューニング
- OLTP アプリケーションのチューニング

データベース操作のチューニングには、常にアプリケーションの特定の達成目標を考慮して取り組みます。チューニングするのはシリアル SQL 文とパラレル操作のどちらですか？オンライン・トランザクション処理（OLTP）アプリケーションとデータ・ウェアハウス・アプリケーションのどちらになりますか？

- データ・ウェアハウス操作は大量のデータを処理し、パラレル操作の目標との間に高い相関関係があります。
- OLTP アプリケーションには大量トランザクションがあり、シリアル操作との間に高い相関関係があります。

したがって、表 4-1 に示すように、これら 2 つの異なるアプリケーションのチューニング目標は対照的です。

表 4-1 チューニングの目標の対比

チューニングの状況	目標
シリアル SQL 文	操作によるリソースの使用率を最小化します。
パラレル操作	ハードウェアのスループットを最大化します。

## シリアル SQL 文のチューニング

1 つの SQL 文に関する限り、チューニングの目標は次のように述べることができます。

実行中の操作によるリソースの使用率を最小化すること

実際にアプリケーションを修正しなくても、別の SQL 構文を試すことができます。そのためには、検討している別の文で EXPLAIN PLAN コマンドを使用して、その実行計画とコストを既存のものと比較します。SQL 文のコストは、EXPLAIN PLAN によって生成される最初の行の POSITION 列に示されます。ただし、実際により速く実行される文を判断するには、アプリケーションを実行しなければなりません。

関連項目： 4-6 ページの「SQL 文のチューニングのアプローチ」

## パラレル実行のチューニング

パラレル実行のチューニングの目標は、次のように述べることができます。

ハードウェアのスループットを最大化します。

高性能のシステムがあり、優先順位の高い SQL 文を大量に実行する場合は、使用可能なリソースをすべて利用できるように SQL 文をパラレル化してもよいでしょう。

Oracle は、次の操作をパラレルで実行できます。

- パラレル問合せ
- パラレル DML (INSERT、UPDATE、DELETE、APPEND ヒント、パラレル索引走査を含む)
- パラレル DDL
- パラレル回復
- パラレル・ロード
- パラレル伝播 (レプリケーションの場合)

次のような状況では、パラレル化できる操作を検討してください。

- 経過時間が長い場合  
問合せかバッチ・ジョブかにかかわらず、データベースで実行している操作の経過時間が長くなる場合は、パラレル操作を使用することで経過時間を短縮できます。
- 処理される行数が多い場合  
複数の行全体が単一のプロセスでアクセスされないように、分割できます。

**関連項目：** パラレル実行の基本原理は、[第 26 章「パラレル実行のチューニング」](#)および『Oracle8i 概要』を参照してください。

## OLTP アプリケーションのチューニング

OLTP アプリケーションのチューニングには、主にシリアル SQL 文のチューニングが関係します。設計の問題を 2 つ考慮する必要があります。1 つは SQL と共有 PL/SQL の使用、もう 1 つはさまざまなトランザクション・モードの使用です。

**関連項目：** データ・ウェアハウス・アプリケーションのチューニングの詳細は、[第 11 章「データ・ウェアハウス・アプリケーションの最適化」](#)を参照してください。

## SQL および共有 PL/SQL

解析時間を最小限にするために、OLTP アプリケーションの SQL 文ではバインド変数を使用してください。こうすると、すべてのユーザーが同じ SQL 文を共有できるようになり、解析に必要なリソースを節約できます。

## トランザクション・モード

上級のユーザーはディスクリット・トランザクションを使用できます。これは、パフォーマンスが最も重要な場合で、そのようにアプリケーションを設計できる場合に限ります。

アプリケーションが ANSI 互換でなければならない場合は、直列可能トランザクションを使用できます。直列可能トランザクションには本質的なオーバーヘッドがあるので、オラクル社ではかわりに読み込みコミット・トランザクションを使用することをお勧めします。

**関連項目：** [第9章「トランザクション・モード」](#)

## トリガー

トリガーの過度な使用によってシステムのパフォーマンスが低下する場合は、CREATE TRIGGER コマンドまたは REPLACE TRIGGER コマンドを実行して、トリガーを起動する条件を変更してください。ALTER TRIGGER コマンドを使用してトリガーをオフにすることもできます。

---

---

**注意：** ログオン、ログオフおよびエラーなどのイベントはすべてのユーザーに影響するので、これらの頻繁なイベントでトリガーを過度に使用すると、パフォーマンスが低下することがあります。

---

---



# データベース操作のチューニング方法

新規の SQL 文を作成するか、既存のアプリケーションで問題のある文をチューニングするかにかかわらず、データベース操作のチューニングの方法は、基本的に CPU とディスク I/O のリソースに関係します。

- [ステップ 1: リソースを最も多く消費している文の検索](#)
- [ステップ 2: 使用するリソースの減少を目的とした文のチューニング](#)

## ステップ 1: リソースを最も多く消費している文の検索

チューニング作業では、利益がチューニングのコストを明らかに上回る SQL 文に注目します。TKPROF、SQL トレース機能、Oracle Trace などのツールを使用して、問題のある文とストアド・プロシージャを検索します。あるいは、一時セグメントに対応付けられたセッションと SQL 文を表示する V\$SORT\_USAGE ビューを問い合わせることもできます。

チューニングによってパフォーマンスを改善できる可能性が最も高い文として、次のものがあります。

- 全体として最も多くリソースを消費している文
- 行単位で最も多くリソースを消費している文
- 最も頻繁に実行される文

V\$SQLAREA ビューでは、まだキャッシュ内にある文のなかから大量のディスク I/O とバッファ取得を行った文を検索できます（バッファ取得は、使われているおおよその CPU リソース量を示します）。

**関連項目：** 動的パフォーマンス・ビューの詳細は、[第 14 章「SQL トレース機能と TKPROF」](#)、[第 15 章「Oracle Trace の使用方法」](#)、および『Oracle8i リファレンス・マニュアル』を参照してください。

## ステップ 2: 使用するリソースの減少を目的とした文のチューニング

アプリケーションの設計は、パフォーマンスの根源をなすことを忘れないでください。いくら SQL 文をチューニングしても、非効率なアプリケーション設計を完全に補うことはできません。SQL 文のチューニングで問題が生じた場合は、おそらくアプリケーションの設計を変更する必要があります。

特定の SQL 文で消費されるリソースを削減するために採用できる方針は 2 つあります。

- 文が使用するリソースを少なくする。
- 文の使用頻度を減らす。

文がほとんどのリソースを消費する理由は、文が作業のほとんどを行っているということ、または文の動作が非効率的であるということ、あるいはその両方です。ただし、作業単位あたり（処理される行あたり）で使われているリソースが少ないほど、アプリケーション自体

を変更することでしか、使われるリソースを大幅に削減できない可能性が高くなります。すなわち、SQL を変更するよりも、アプリケーションが処理する行の数を少なくするか、同じ行の処理頻度を減らす方が効果的である可能性があります。

これら 2 つのアプローチは、相互に排他的ではありません。前者では、(索引構造の変更による) プログラムの変更を行わずに、または関連するロジックではなく SQL 文そのもののみを変更することでこれを達成できるので、コストは明らかに低くなります。

**関連項目：** 第 18 章「CPU リソースのチューニング」および第 20 章「I/O のチューニング」

## SQL 文のチューニングのアプローチ

この項では、SQL 文の効率を高める 5 つの方法を説明します。

- [索引の再構成](#)
- [文の再構成](#)
- [トリガーの変更または使用禁止](#)
- [データの再構成](#)
- [最新統計の収集およびプラン・スタビリティの使用による実行計画の保持](#)

---

---

**注意：** この項で説明するガイドラインは、頻繁に実行される本番 SQL に関するものです。ここで推奨されていない技法の大半は、パフォーマンスが重要でない非定型文または頻繁に実行されないアプリケーションでは使用してもかまいません。

---

---

## 索引の再構成

索引の再構成は、文やデータの再構成よりもアプリケーションに与える影響が大きいため、最初に行うことが妥当です。

- 選択不可能な索引を削除して DML の速度をあげる。
- パフォーマンスが重要なアクセス・パスの索引を作成する。
- ハッシュ・クラスタは考慮するが、一意性を監視する。
- クラスタ・キーが同じようにサイズ設定されている場合にだけ索引クラスタを考慮する。

索引を万能策として使用しないでください。アプリケーションの開発者は、十分な索引を作成するだけでパフォーマンスが改善されると考えることがあります。1 人のプログラマーが適切に索引を作成すれば、アプリケーションのパフォーマンスは十分に改善される可能性があります。ただし、50 名のプログラマーがそれぞれ索引を作成すると、アプリケーションのパフォーマンス改善は望めないでしょう。

## 文の再構成

索引を再構成した後で、文の再構成を試してみることができます。非効率的な SQL 文は、修正するよりも書き直す方が簡単なことがよくあります。指定の文の用途を理解すれば、要件を満たす新しい文を迅速かつ容易に作成できるでしょう。

### かわりの SQL 構文の検討

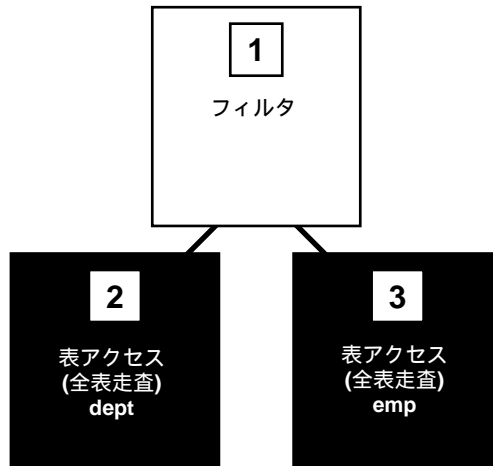
SQL は柔軟性がある言語なので、アプリケーションのニーズに応える SQL 文が複数ある可能性があります。2 つの SQL 文は同じ結果を生成しますが、Oracle はもう一方よりも速く処理します。EXPLAIN PLAN 文の結果を使って、2 つの文の実行計画とコストを比較し、どちらの文がより効率的か判断することができます。

次の例は、同じ機能を実行する 2 つの SQL 文の実行計画を示します。どちらの文も、EMP 表に従業員を持たない DEPT 表のすべての部門を戻します。それぞれの文は、副問合せによって EMP 表を検索します。EMP 表の DEPTNO 列には索引 DEPTNO\_INDEX が存在するものとします。

最初の文とその実行計画を次に示します。

```
SELECT dname, deptno
FROM dept
WHERE deptno NOT IN
    (SELECT deptno FROM emp);
```

図 4-1 2 つの全表走査を行う実行計画

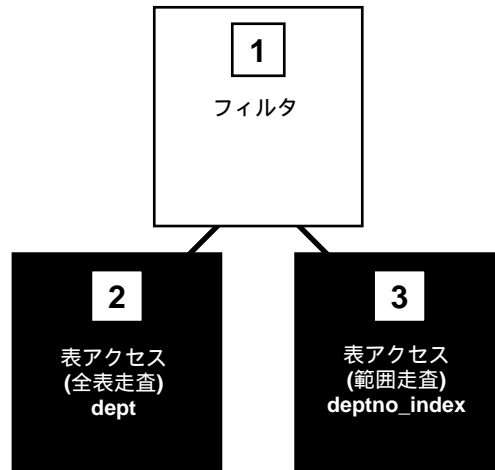


上図のステップ 3 の出力は、DEPTNO 列の索引にもかかわらず、Oracle が EMP 表の全表走査を行うことによってこの文を実行することを示します。この全表走査は、時間のかかる処理となる可能性があります。EMP 表を探索する副問合せには索引を使用可能にする WHERE 句がないため、Oracle は索引を使いません。

しかし、次の SQL 文は、索引にアクセスすることによって同じ行を選択します。

```
SELECT dname, deptno
FROM dept
WHERE NOT EXISTS
  (SELECT deptno
   FROM emp
  WHERE dept.deptno = emp.deptno);
```

図 4-2 全表走査と索引走査を行う実行計画



**関連項目：** 実行計画の解釈の詳細は、『Oracle8i 概要』のオプティマイザの章を参照してください。

副問合せの WHERE 句が EMP 表の DEPTNO 列を参照するため、索引 DEPTNO\_INDEX が使用されます。索引は、実行計画のステップ 3 で使用されます。DEPTNO\_INDEX の索引範囲走査によって、最初の文の EMP 表の全表走査よりも処理時間は短くなります。さらに、最初の問合せは、DEPT 表のあらゆる DEPTNO について EMP 表の全表走査を 1 回実行します。このため、2 番目の SQL 文は最初の SQL 文よりも高速です。

この例の最初の問合せのように、アプリケーションに NOT IN 演算子を使用する文がある場合、NOT EXISTS 演算子を使用するように、その文を書き直すことを検討すべきです。このようにすることで、索引があれば文は索引を使用できます。

## AND と = を使った条件の組立て

等価結合を使ってください。変換されない列値に対して等価結合を実行する文は、例外なく、最も容易にチューニングできます。

## 有利な結合順序の選択

結合順序は、パフォーマンスに大きな影響を与えることがあります。SQL のチューニングの主な目的は、行にアクセスする際に、結果に影響しない不要な作業の実行を回避することです。このことから 3 つの一般ルールが導かれます。

- インデックスを介して必要な行を取得する方が効率的な場合には、全表操作を避ける。
- 100 行をフェッチする別のインデックスを使える場合は、駆動表から 10,000 行をフェッチするインデックスの使用は避ける。
- 結合順序の後ろの表に対して結合する行が少なくなるような結合順序を選択する。

次の例は、結合順序を効果的にチューニングする方法を示しています。

```
SELECT info
FROM taba a, tabb b, tabc c
WHERE a.acol between :alow and :ahigh
      AND b.bcol between :blow and :bhigh
      AND c.ccol between :clow and :chigh
      AND a.key1 = b.key1
      AND a.key2 = c.key2;
```

1. 駆動表と駆動インデックス（存在する場合）を選択する。

上記の例における最初の 3 つの条件は、それぞれ 1 つの表にだけ適用されるフィルタ条件です。最後の 2 つの条件は結合条件です。

フィルタ条件は、駆動表と駆動インデックスの選択を左右します。一般に、表内の排除される部分の比率が最も高くなるフィルタ条件を含む表を、駆動表とする必要があります。したがって、:alow から :ahigh の範囲が acol の範囲より狭くても、:b\* と :c\* の範囲が相対的に広い場合は、taba を駆動表とし、その他をすべて等価とする必要があります。

2. 正しいインデックスを選択する。

駆動表を決めた後で、その表の駆動に使える最も選択性の高いインデックスを選択します。あるいは、全表走査の方が効果的な場合には、それを選択します。そこから、結合はすべて結合インデックスを介して行う必要があります。この結合インデックスは、主キーまたは外部キーに付けられているもので、その表を結合ツリー内のそれより前にある表に接続するために使います。駆動表を除いて、非結合条件にインデックスを使うことはめったにありません。そのため、taba が駆動表として選択された後は、b.key1 のインデックスを使って tabb を駆動し、c.key2 のインデックスを使って tabc を駆動する必要があります。

### 3. 未使用の最適なフィルタを最初に駆動する最適な結合順序を選択する。

その後の結合の作業は、最適な未使用フィルタを持つ表に先に結合することで削減できます。したがって、"bcol between ..." が "ccol between ..." よりも限定的な（表示される行をより高い比率で拒否する）場合は、tabb が tabc よりも前に結合されていれば最後の結合がより簡単に（より少ない行で）行われます。

## 変換されない列値の使用

変換されない列値を使用します。たとえば、次の例を使用します。

```
WHERE a.order_no = b.order_no
```

次の例は使用しません。

```
WHERE TO_NUMBER (substr(a.order_no, instr(b.order_no, '.') - 1)
= TO_NUMBER (substr(a.order_no, instr(b.order_no, '.') - 1)
```

述語句または WHERE 句では、SQL 関数を使わないでください。特に副問合せでは、集計関数を使用すると、導出された値がしばしばマスター・レコード上に保持されることがよくあります。

## タイプが混在する式の回避

モードが混在する式は避け、暗黙の型変換に注意してください。VARCHAR2 列 *charcol* の索引を使用するときに、WHERE 句が次のようであるとします。

```
AND charcol = <numexpr>
```

ここで、*numexpr* は数値型の式（たとえば、1、USERENV('SESSIONID')、numcol、numcol+0、...）であり、Oracle はこの式を次のように変換します。

```
AND to_number(charcol) = numexpr
```

これによって次のような結果が生じます。

- 列を引数として持つ関数など、列を使う式は、オプティマイザがその列に索引を使える可能性を無視する原因となる（一意索引も例外ではありません）。
- 数値には変換されない文字列として *charcol* を持つ行をシステムが 1 行でも処理すると、エラーが戻される。

この問題は、一番上の式を明示変換に置き換えることで回避できます。

```
AND charcol = to_char(<numexpr>)
```

あるいは、すべての型変換を明示的に行います。

```
numcol = charexpr
```

上記の文では、デフォルトでは常に文字を数値に変換するので、*numcol* に対して索引を使用できます。ただし、この動作は変更の対象となります。型変換を明示的に行うと、*charexpr* が常に数値に変換されることが明らかになります。

### 特定の値に対する別の SQL 文の作成

SQL は、プロシージャ型言語ではありません。1 つの SQL を多くの異なる用途に使用することは妥当ではありません。そのようにすると、通常はそれぞれのタスクで最適な結果が得られなくなります。SQL を使用してさまざまなタスクを行う場合は、1 つの文にパラメータを指定して異なるタスクを実行するかわりに、2 つの異なる文を作成してください。

最適化（実行計画の決定）は、どの値で問合せが置換されるかをデータベースが認識する前に行われます。したがって、実行計画はそれらの値が何であるかに依存してはなりません。次に例を示します。

```
SELECT info from tables
WHERE ...
      AND somecolumn BETWEEN decode(:loval, 'ALL', somecolumn, :loval)
      AND decode(:hival, 'ALL', somecolumn, :hival);
```

この例では、データベースは *somecolumn* 列に対して索引を使用できません。この列に関係する式は、BETWEEN の両側で同じ列を使用するからです。

このことは、他の選択性の高い、索引作成可能な条件を使って駆動表にアクセスできる場合には問題になりません。ただし、これにあてはまらない場合もよくあります。この例のような条件で索引を使うこともよくありますが、あらかじめ *:loval* などの値を知っておく必要があります。この情報があれば、索引を使用してはならない ALL のケースを除外できます。



:loval と :hival に実際の値が与えられている場合に常に索引を使用するときは（つまり、:loval が :hival と等しくなることがよくあり、狭い範囲を期待している場合は）、この例を論理的に同じ次の形式に書き直すことができます。

```
SELECT /* change this half of union all if other half changes */ info
FROM tables
WHERE ...
    AND somecolumn between :loval and :hival
    AND (:hival != 'ALL' and :loval != 'ALL')
UNION ALL
SELECT /* Change this half of union all if other half changes. */ info
FROM tables
WHERE ...
    AND (:hival = 'ALL' OR :loval = 'ALL');
```

この新規問合せで EXPLAIN PLAN を実行すると、望ましい実行計画と望ましくない実行計画の両方が得られたように見えます。しかし、データベースが UNION ALL のいずれか半分を評価する最初の条件は、:hival と :loval が ALL であるかどうかの複合条件となります。問合せのその部分に関する実行計画から実際に行を取得する前に、データベースがこの条件を評価します。UNION ALL 問合せの一部に関して条件が偽として戻されると、その部分はそれ以上評価されません。与えられている値に関して最適な実行計画の部分だけが実際に実行されます。:hival と :loval に関する最終条件は相互に排他的であることが保証されているので、UNION ALL の半分だけが実際に行を戻します（UNION ALL 内の ALL は、その排他性により論理的に有効です。コストの高いソートを実行せずに、問合せの両半分に関して重複行を除外できる計画の実行が可能になります）。

## アクセス・パスを制御するヒントの使用

/\*+ORDERED \*/ などのオプティマイザ・ヒントを使用してアクセス・パスを制御します。これは、従来の技法や、CUST\_NO + 0 などの "秘策" を使うよりも優れたアプローチです。たとえば、次の例を使用します。

```
SELECT /*+ FULL(EMP) */ E.ENAME
FROM EMP E
WHERE E.JOB = 'CLERK';
```

次の例は使用しません。

```
SELECT E.ENAME
FROM EMP E
WHERE E.JOB || ' ' = 'CLERK';
```

**関連項目：** ヒントの詳細は、[第7章「オプティマイザ・モード、プラン・スタビリティおよびヒント」](#)を参照してください。

## 副問合せでの IN および NOT IN 使用時の注意

WHERE (NOT) EXISTS が便利な代替方法であることを覚えておいてください。

## アプリケーションでの組み込みデータ値リスト使用時の注意

通常、データ値リストは、エンティティがないことを示します。次に例を示します。

```
WHERE TRANSPORT IN ('BMW', 'CITROEN', 'FORD', HONDA')
```

上記の WHERE 句の実際の目的は、transport のモードが automobile であるかどうかを判別することであり、特定の車種を識別することではありません。transport type='AUTOMOBILE' である、参照表が使用可能でなければなりません。

DISTINCT の使用は最低限にしてください。DISTINCT は、常に SORT を作成します。すべてのデータは、結果を戻す前にインスタンスエートする必要があります。

## データベースの呼出し回数の削減

適切な場合には、INSERT、UPDATE または DELETE ... RETURNING を使用して、1 回の呼出しでデータを選択および修正してください。この技法は、データベースの呼出し回数を減らすことでパフォーマンスを改善します。

**関連項目：** INSERT、UPDATE および DELETE コマンドの構文については、『Oracle8i SQL リファレンス』を参照してください。

## ビューを管理するときの注意

ビューを結合するとき、ビューに対して外部結合を実行するとき、およびビューのリサイクルを検討するときには注意してください。

**ビューを結合するときの注意** Oracle の共有 SQL 領域は、ビューを参照する問合せの解析コストを削減します。さらに、オブティマイザの改良によって、ビューに対する条件の処理が非常に効率的になっています。これらの要因が重なって、非定型問合せに対するビューの使用が可能になっています。ただし、ビューへの結合、特にある複合ビューから別の複合ビューへの結合はお勧めできません。

次の例は、GROUP BY の結果である列に関する問合せを示しています。ビュー全体が最初にインスタンスiertされ、次にビュー・データに対する問合せが実行されています。

```
CREATE VIEW DX(deptno, dname, totalsal)
AS SELECT D.deptno, D.dname, E.sum(sal)
   FROM emp E, dept D
   WHERE E.deptno = D.deptno
   GROUP BY deptno, dname
SELECT * FROM DX WHERE deptno=10;
```

**ビューへの外部結合を実行するときの注意** 複数表のビューに対する外部結合は、問題の原因となります。たとえば、e.empno および e.deptno、d.deptno に索引を持つ通常の emp 表と dept 表から開始して、次のビューを作成できます。

```
CREATE VIEW EMPDEPT (EMPNO, DEPTNO, ename, dname)
AS SELECT E.EMPNO, E.DEPTNO, e.ename, d.dname
   FROM DEPT D, EMP E
   WHERE E.DEPTNO = D.DEPTNO(+);
```

次に、可能な限り単純な問合せを組み立てて、ビューの基礎を形成する表の索引付き列 (e.deptno) にこのビューを外部結合できます。

```
SELECT e.ename, d.loc
   FROM dept d, empdept e
   WHERE d.deptno = e.deptno(+)
   AND d.deptno = 20;
```

結果として、次の実行計画が生成されます。

```
QUERY_PLAN
-----
MERGE JOIN OUTER
TABLE ACCESS BY ROWID DEPT
  INDEX UNIQUE SCAN DEPT_U1: DEPTNO
FILTER
VIEW EMPDEPT
  NESTED LOOPS OUTER
    TABLE ACCESS FULL EMP
    TABLE ACCESS BY ROWID DEPT
      INDEX UNIQUE SCAN DEPT_U1: DEPTNO
```

ビューの両方の表が結合されるまで、オブティマイザにはビューが一致する行を生成するかどうか分かりません。したがって、オブティマイザは、ビューのすべての行を生成し、残りの問合せから戻されたすべての行で MERGE JOIN OUTER を実行する必要があります。このアプローチは、少なくとも 1 つの大規模表と複数表のビューを結合したいときに、必要な行がごくわずかな場合にはきわめて非効率です。

前の例では、この問題を解決することは比較的容易です。dept に対する 2 番目の参照は不要なので、emp に対して直接外部結合を行えます。他の場合では、結合は必ずしも外部結合である必要はありません。ビューへの結合から (+) を取り除くだけで、依然としてビューを使えます。

**ビューのリサイクル禁止** ある用途のために作成したビューを、不適当かもしれない他の用途に使うことには注意してください。次の例を考えます。

```
SELECT dname from DX
7 WHERE deptno=10;
```

dname と deptno は、DEPT 表から直接取得できます。(この例の前方で宣言されている) DX ビューを問い合せてこの情報を取得することは非効率的です。問合せに応答するために、ユーザーが EMP 表のデータを必要としない場合でも、ビューは DEPT 表と EMP 表の結合を実行します。

## トリガーの変更または使用禁止

トリガーを使用すると、システムのリソースが消費されます。使用するトリガーが多すぎると、パフォーマンスに悪影響が及び、トリガーを変更または使用禁止にしなければならない場合があります。

## データの再構成

索引と文を再構成した後で、データの再構成について検討できます。

- 導出された値の導入。応答時間が重要なコードでは GROUP BY を避けます。
- 欠落したエンティティと交差表をインプリメントします。
- ネットワーク負荷の低減。データの移行、レプリケート、パーティション化を行います。

どのデータ分散戦略でも、その総合的な目的は、ネットワーク送信回数が最小限になるようなデータ属性の位置を見つけることです。現行の送信回数が過度である場合は、データの移動（移行）が本質的な解決策です。

しかし、ネットワーク負荷（または、メッセージ伝送の遅延）を許容できるレベルにまで低減するデータの位置が 1 つもないことがよくあります。この場合には、複数のコピーを保持すること（データのレプリケート）、またはデータの異なる部分を異なる場所に保持すること（データのパーティション化）を検討します。

分散問合せが必要な場所では、ストアード・プロシージャ内の PL/SQL またはユーザー・インタフェース・コード内で、必要な結合をプロシージャ型でコーディングすると効果的な場合があります。

ネットワーク間結合を検討するときは、リモート・ノードからデータを持ってきてローカルで結合を実行するか、結合をリモートで実行できます。どのオプションを選択するかは、さまざまなノード上の相対データ量によって決まります。

## 最新統計の収集およびプラン・スタビリティの使用による実行計画の保持

アプリケーションの SQL 文をチューニングしたら、DBMS\_STATS パッケージの便利なプロシージャを使用して統計をメンテナンスすることを考慮してください。また、システムが変更されてもアプリケーションのパフォーマンス特性はメンテナンスされるよう、プラン・スタビリティ機能をインプリメントすることを考慮してください。これらのトピックについては、[第 7 章「オブティマイザ・モード、プラン・スタビリティおよびヒント」](#)を参照してください。



---

## アプリケーションの登録

アプリケーションの開発者は、Oracle Trace および SQL トレース機能とともに DBMS\_APPLICATION\_INFO パッケージを使用して、後で各種モジュールのパフォーマンスの追跡に使用するために、データベースに格納されている実行モジュール名またはトランザクション名を記録しておくことができます。この章では、データベースにアプリケーションを登録する方法と、各登録モジュールまたはコード・セグメントについての統計を検索する方法について説明します。

Oracle では、アプリケーションの名前と、そのアプリケーションによって実行されるアクションを、アプリケーションがデータベースに登録する方法が提供されます。アプリケーションを登録すると、システム管理者とパフォーマンスのチューニングを行う担当者がモジュール別にパフォーマンスを追跡できるようになります。システム管理者はこの情報を使って、モジュール別のリソースの使用状況も追跡できます。アプリケーションがデータベースに登録されると、その名前とアクションが V\$SESSION ビューと V\$SQLAREA ビューに記録されます。

ユーザーがモジュールを入力する度に、アプリケーションがそのモジュールの名前とアクションの名前を自動的に設定するようにします。モジュール名は、Oracle Forms アプリケーションのフォームの名前、または Oracle プリコンパイラ・アプリケーションのコード・セグメントの名前にすることができます。通常、アクションの名前は、モジュール内の現行のトランザクションの名前またはこのトランザクションについての記述にします。

トピックは次のとおりです。

- [モジュール名の設定](#)
- [アクション名の設定](#)
- [クライアント情報の設定](#)
- [アプリケーション情報の検索](#)

## DBMS\_APPLICATION\_INFO パッケージ

アプリケーションをデータベースに登録するには、DBMS\_APPLICATION\_INFO パッケージのプロシージャを使用します。DBMS\_APPLICATION\_INFO では、次のプロシージャが提供されます。

表 5-1 DBMS\_APPLICATION\_INFO パッケージのプロシージャ

プロシージャ	説明
SET_MODULE	現在実行されているモジュールの名前を設定します。
SET_ACTION	現行モジュール内の現行アクションの名前を設定します。
SET_CLIENT_INFO	セッションのクライアント情報フィールドを設定します。
READ_MODULE	現行セッションのモジュール・フィールドとアクション・フィールドの値を読み込みます。
READ_CLIENT_INFO	現行セッションのクライアント情報フィールドを読み込みます。

## 権限

このパッケージを使用する前に、DBMSUTL.SQL スクリプトを実行して DBMS\_APPLICATION\_INFO パッケージを作成する必要があります。

**関連項目：** オラクル社が提供しているパッケージおよびストアド・プロシージャの実行方法の詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。

## モジュール名の設定

現行のアプリケーションまたはモジュールの名前を設定するには、DBMS\_APPLICATION\_INFO パッケージの SET\_MODULE プロシージャを使用してください。このモジュール名は、プロシージャの名前（ストアド・プロシージャを使用している場合）またはアプリケーションの名前にしてください。アクション名は、実行するアクションがわかるものにしなければなりません。

## 例

次の SQL 文で BEGIN キーワードから始まる PL/SQL ブロック例では、モジュール名とアクション名を設定します。

```
CREATE PROCEDURE add_employee(  
  name      VARCHAR2(20),  
  salary    NUMBER(7,2),  
  manager   NUMBER,  
  title     VARCHAR2(9),
```



```

        commission NUMBER(7,2),
        department NUMBER(2)) AS
BEGIN
    DBMS_APPLICATION_INFO.SET_MODULE(
        module_name => 'add_employee',
        action_name => 'insert into emp');
    INSERT INTO emp
        (ename, empno, sal, mgr, job, hiredate, comm, deptno)
        VALUES (name, next.emp_seq, manager, title, SYSDATE,
            commission, department);
    DBMS_APPLICATION_INFO.SET_MODULE('','');
END;

```

## 構文

SET\_MODULE プロシージャの構文とパラメータを次に示します。

```

DBMS_APPLICATION_INFO.SET_MODULE(
    module_name    IN VARCHAR2,
    action_name    IN VARCHAR2)

```

module_name	現在実行中のモジュールの名前。現行のモジュールが終了すると、新しいモジュールがある場合はこのプロシージャを新しいモジュールの名前でコールし、ない場合はこのパラメータは NULL になります。48 バイトを超える部分は切り捨てられます。
action_name	現行モジュール内の現行アクションの名前。アクションを指定しない場合、この値は NULL になります。32 バイトを超える部分は切り捨てられます。

## アクション名の設定

現行モジュール内の現行のアクションの名前を設定するには、DBMS\_APPLICATION\_INFO パッケージの SET\_ACTION コマンドを使用してください。アクションの名前は、実行中の現行アクションを記述するテキストにしてください。各トランザクションを開始する前に、アクション名を設定しなければならない場合があります。

## 例

次に、登録プロシージャを使用するトランザクションの例を示します。

```

CREATE OR REPLACE PROCEDURE bal_tran (amt IN NUMBER(7,2)) AS
BEGIN
    -- balance transfer transaction
    DBMS_APPLICATION_INFO.SET_ACTION(
        action_name => 'transfer from chk to sav');

```

```
UPDATE chk SET bal = bal + :amt
WHERE acct# = :acct;
UPDATE sav SET bal = bal - :amt
WHERE acct# = :acct;
COMMIT;
DBMS_APPLICATION_INFO.SET_ACTION('');
END;
```

トランザクションの終了後、後続のトランザクションが正しく記録されるように、トランザクション名を NULL に設定してください。トランザクション名を NULL に設定しないと、後続のトランザクションが前のトランザクションの名前で記録されます。

## 構文

SET\_ACTION プロシージャのパラメータをこの項で説明します。このプロシージャの構文は、次のとおりです。

```
DBMS_APPLICATION_INFO.SET_ACTION(action_name IN VARCHAR2)
```

action_name	現行モジュール内の現行アクションの名前。現行アクションが終了すると、次のアクションが存在する場合はその名前でこのプロシージャをコールし、ない場合はこのパラメータは NULL になります。32 バイトを超える部分は切り捨てられます。
-------------	---

## クライアント情報の設定

クライアント・アプリケーションについての追加情報を設定するには、DBMS\_APPLICATION\_INFO パッケージの SET\_CLIENT\_INFO プロシージャを使用します。

## 構文

SET\_CLIENT\_INFO プロシージャのパラメータをこの項で説明します。このプロシージャの構文は、次のとおりです。

```
DBMS_APPLICATION_INFO.SET_CLIENT_INFO(client_info IN VARCHAR2)
```

client_info	このパラメータを使用して、クライアント・アプリケーションについての追加情報を設定します。この情報は、V\$SESSIONS ビューに格納されます。64 バイトを超える部分は切り捨てられます。
-------------	---

# アプリケーション情報の検索

登録アプリケーションのモジュール名とアクション名を検索するには、V\$SQLAREA を問い合わせるか、または DBMS\_APPLICATION\_INFO パッケージの READ\_MODULE プロシージャをコールします。クライアント情報を検索するには、V\$SESSION ビューを問い合わせるか、または DBMS\_APPLICATION\_INFO パッケージの READ\_CLIENT\_INFO をコールします。

## V\$SQLAREA の問合せ

次の問合せの例では、V\$SQLAREA の MODULE 列と ACTION 列の使用方法を示します。

```
SELECT sql_text, disk_reads, module, action
FROM v$sqlarea
WHERE module = 'add_employee';
```

SQL_TEXT	DISK_READS	MODULE	ACTION
INSERT INTO emp (ename, empno, sal, mgr, job, hiredate, comm, deptno) VALUES (name, next.emp_seq, manager, title, SYSDATE, commission, department)	1	add_employee	insert into emp

1 row selected.

## READ\_MODULE の構文

READ\_MODULE プロシージャのパラメータをこの項で説明します。このプロシージャの構文は、次のとおりです。

```
DBMS_APPLICATION_INFO.READ_MODULE(  
    module_name OUT VARCHAR2,  
    action_name OUT VARCHAR2)
```

module_name	SET_MODULE をコールしてモジュール名に最後に設定された値。
action_name	SET_ACTION または SET_MODULE をコールしてアクション名に最後に設定された値。

## READ\_CLIENT\_INFO の構文

READ\_CLIENT\_INFO プロシージャのパラメータをこの項で説明します。このプロシージャの構文は、次のとおりです。

```
DBMS_APPLICATION_INFO.READ_CLIENT_INFO(client_info OUT VARCHAR2)
```

client_info	SET_CLIENT_INFO プロシージャに最後に設定されたクライアント情報の値。
-------------	--

---

## データ・アクセス方法

この章では、パフォーマンスを改善できるデータ・アクセス方法の概要を説明し、避ける必要のある状態を示します。また、ヒントを使用してさまざまなアプローチを強制する方法を説明します。トピックは次のとおりです。

- [索引の使用方法](#)
- [関数ベース索引の使用方法](#)
- [ビットマップ索引の使用方法](#)
- [ドメイン索引の使用方法](#)
- [クラスタの使用方法](#)
- [ハッシュ・クラスタの使用方法](#)

## 索引の使用方法

この項では、次のトピックについて説明します。

- [索引作成のタイミング](#)
- [論理構造のチューニング](#)
- [索引を付ける列と式の選択](#)
- [複合索引の選択](#)
- [索引を使用する文の記述](#)
- [索引を使用しない文の記述](#)
- [索引の有効性の調査](#)
- [索引の再作成](#)
- [一意性を施行するための一意でない索引の使用方法](#)
- [未検査使用可能制約の使用方法](#)

## 索引作成のタイミング

索引は表から小さな割合の行を選択する問合せのパフォーマンスを改善します。一般的なガイドラインとして、表全体の行の 2% または 4% 未満を問い合わせる場合は、その表に対して索引を作成します。すべてのデータを索引から検索できる状態、または索引付けされた列や式を他の表への結合に使用できる状態では、この値が高くなる可能性があります。

このガイドラインは次の仮定に基づいています。

- 問合せによって参照されるキーに同じ値を持つ行が、表に割り当てられたデータ・ブロック全体にわたって一様に分布している。
- 表の行は、問合せによって参照されるキーに対してランダムに並んでいる。
- 表の列数が比較的に少ない。
- 表に対するほとんどの問合せには、比較的単純な WHERE 句が使用される。
- キャッシュ・ヒット率が低く、オペレーティング・システムのキャッシュはない。

これらの仮定が、表のデータとそのデータにアクセスする問合せに当てはまらない場合は、通常、問合せによって表の少なくとも 25% の行にアクセスしない限り索引が有効にならないこともあります。

## 論理構造のチューニング

コストベースの最適化は、問合せ実行でのあまり有効でない索引の使用を避けるのに役立ちますが、SQL エンジンでは、表に対して定義されている索引が使用されているかどうかにかかわらず、継続的にすべての索引をメンテナンスする必要があります。索引のメンテナンスに、I/O 中心のアプリケーションでは CPU と I/O リソースが大量に必要となる場合があります。" 万々に備えて " 索引を作成することはお薦めしません。索引は必要になるまで作成しないでください。

索引に関する限り、最適なパフォーマンスを維持するには、アプリケーションで使用していない索引を削除する必要があります。EXPLAIN PLAN を使用してすべてのアプリケーションの SQL を処理し、結果の計画を獲得することによって、どの実行計画にも参照されない索引を検出できます。使用されない索引は、必ずというわけではありませんが、通常は有効ではありません。

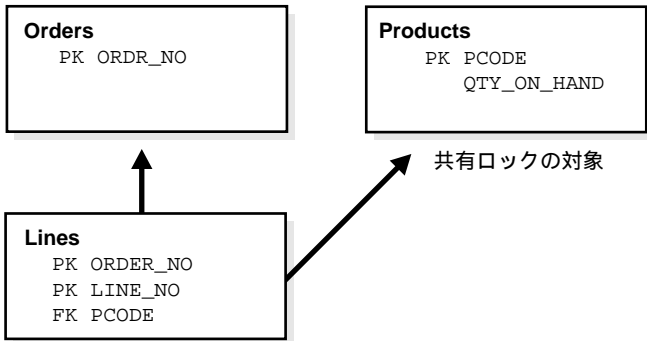
アプリケーション内では、文の実行計画の調査ですぐには明らかにならない索引の使用方法もあります。特に、Oracle は、外部キー制約を施行するときに子表での共用ロックの使用を回避するために、外部キー索引に " ピン " (非トランザクション・ロック) を使用します。

多くのアプリケーションでは、外部キー索引は、決して (または、めったに) 問合せをサポートしません。図 6-1 の例では、指定された製品に対してすべての注文の各行を探す必要がないこともあります。ただし、(「[複合索引の選択](#)」で説明するように) LINES(PCODE) の索引が先行部分として存在しないときは、PRODUCTS(PCODE) が更新または削除されるたびに LINES 表に共有ロックがかけられます。このような共有ロックが問題となるのは、PRODUCTS 表が頻繁に DML の対象となる場合だけです。

この競合が発生した場合は、それを取り除くためにアプリケーションは次のいずれかを行います。

- 索引メンテナンスのための追加負荷を受け入れる
- 制約を使用禁止にした状態で実行するリスクを受け入れる

図 6-1 外部キー制約



## 索引を付ける列と式の選択

キーは、索引を付ける列または式です。索引を付ける索引キーを選択するために、次のガイドラインに従ってください。

- WHERE 句で頻繁に使用されるキーに索引を付けることを検討します。
- SQL 文で表を結合するために頻繁に使用されるキーに索引を付けることを検討します。結合の最適化の詳細は、6-24 ページの「ハッシュ・クラスタの使用方法」を参照してください。
- 正確な選択性のキーだけに索引を付けます。索引の選択性は、索引を付けるキーについて同じ値を持つ行の表内での割合です。同じ値を持つ行がほとんどない場合、索引の選択性は最適です。

**注意：** Oracle では、整合性制約を使用して定義されるすべての一意キーおよび主キーのキーと式に、暗黙に索引を作成します。これらの索引は最も選択的であり、パフォーマンスを最適化する上で最も効果的です。

表の行数を索引付き列の値の種類で割ることによって索引の選択性を判断することができます。ANALYZE 文を使用して、これらの値を取得することができます。このようにして算出した選択性を百分率に直すとよいでしょう。

- 異なる値をほとんど持たないキーまたは式には標準の B\* ツリー索引を使用しません。通常そのようなキーや式は選択性に劣っているので、頻繁に選択されるキー値がその他のキー値に比べて少ない場合を除くと、パフォーマンスは最適化されません。このような場合には、並行性の高い OLTP アプリケーションが関係していない限り、ビットマップ索引の使用が効果的です。



- 頻繁に修正される列には索引を付けません。索引付きの列を修正する UPDATE 文および索引付きの表を修正する INSERT 文と DELETE 文では、索引がない場合よりも、処理に長い時間が必要となります。このような SQL 文は、表のデータだけでなく、索引のデータも修正しなければなりません。また、取消しと再実行に関する情報も追加的に生成されます。
- 関数や演算子といっしょに WHERE 句に指定されているだけのキーには索引を付けません。索引付きのキーに (MIN と MAX 以外の) 関数や演算子を使用する WHERE 句は、索引を使用するアクセス・パスを選択しません。
- 同時実行の数多くの INSERT 文および UPDATE 文、DELETE 文が親表と子表をアクセスする場合、参照整合性制約の外部キーに索引を付けることを検討します。このような索引を使用すると、子表を共有ロックせずに親表で UPDATE および DELETE を実行できます。
- 索引を付けるキーを選択するとき、問合せのパフォーマンス利得が、INSERT、UPDATE、DELETE のパフォーマンス損失および索引を格納するために必要となる領域の使用に見合う価値があるかどうか検討してください。SQL 文の処理時間を索引の有無によって実際に比較することをお勧めします。SQL トレース機能で処理時間を測定することができます。

**関連項目：** ロックに対する外部キーの効果は、『Oracle8i 概要』を参照してください。

## 複合索引の選択

複合索引には複数のキー列が含まれています。複合索引には、次のような単一列索引を上回る利点があります。

### 選択性の向上

場合によっては、それぞれの選択性が劣る複数の列や式を組み合わせて複合索引にすると、より正確な選択性を得ることができます。

### 追加のデータ記憶領域

問合せによって選択される列がすべて複合索引に含まれている場合、Oracle は、表にアクセスすることなく、索引からこれらの値を戻すことができます。

SQL 文に複合索引の先頭部分を利用する条件が含まれていれば、その文は複合索引に係するアクセス・パスを使えます。索引の先頭部分とは、その索引を作成した CREATE INDEX 文で列リストの先頭から連続的に指定された 1 つ以上の列の組合せのことです。次のような CREATE INDEX 文を考えます。

```
CREATE INDEX comp_ind
ON tabl(x, y, z);
```

この場合、X、XY および XYZ という列の組合せは、索引の先頭部分となります。ただし、YZ、Z および Y という列の組合せは、索引の先頭部分にはなりません。

複合索引を構成するキーを選択するために、次のガイドラインに従ってください。

- WHERE 句の条件で、頻繁に AND 演算子で結合して使用されるキー・セットに対して、複合索引を作成することを検討します。特に結合したときの選択性が個々のキーの選択性よりも優れている場合、複合索引を作成するとよいでしょう。
- 複数の問合せが 1 つ以上のキー値に基づいて同じキー・セットを選択する場合、これらのキーのすべてを含む複合索引を作成することを検討します。

もちろん、索引を作成するときの、一般的なパフォーマンスの利点およびトレードオフに関する前述のガイドラインを検討してください。複合索引内でのキーの順序を指定するために次のガイドラインに従ってください。

- WHERE 句で使用されるキーが先頭部分を構成するように、索引を作成してください。
- 一部のキーがより頻繁に WHERE 句で使用される場合には、頻繁に選択されるキーが先頭部分を構成するように索引を作成し、これらのキーだけを使用する文が索引を使用できるようにしてください。
- すべてのキーが同程度の頻度で WHERE 句で使用される場合には、CREATE INDEX 文の中で、選択性が最も高いキーから最も低いキーの順に指定すると、問合せのパフォーマンスが最も向上します。
- すべてのキーが同程度に頻繁に WHERE 句で使用されるが、そのキーの 1 つでデータが物理的に順序付けられている場合には、そのキーを複合索引の先頭にしてください。

## 索引を使用する文の記述

索引を作成しても、単に索引が存在するだけでは、オブティマイザはその索引を使うアクセス・パスを選択できません。オブティマイザは、SQL 文にアクセス・パスを使用可能にする構造が含まれている場合に限り、そのようなアクセス・パスを選択できます。

索引を使うアクセス・パスを SQL 文が確実に選択するようにするには、そのようなアクセス・パスを使用可能にする条件が文に含まれていることを確認してください。コストベースのアプローチを使用している場合は、索引の統計も生成します。いったん文に対してアクセス・パスが使用可能になると、オブティマイザは、他のアクセス・パスの可用性に基づいて、そのアクセス・パスを選択するかどうかを判断します。

また、新しい索引を作成して SQL 文をチューニングする場合、オブティマイザがアプリケーションの実行時にこれらの索引を使用するかどうか判断するために、EXPLAIN PLAN 文を使用することもできます。新しい索引を作成して現在解析中の文をチューニングする場合、Oracle はその文を無効にします。その文が次に実行されるとき、オブティマイザは新しい索引を使用する可能性のある新しい実行計画を自動的に選択します。新しい索引をリモート・データベース上に作成して分散 SQL 文をチューニングする場合、その文が次に解析されるとき、オブティマイザはこれらの索引について検討します。

また、ある SQL 文をチューニングする方法が、他の文の実行計画に対するオブティマイザの選択に影響を及ぼす場合があることも忘れないでください。たとえば、ある文によって使われる索引を作成した場合、オブティマイザは、アプリケーションの他の文に対しても、そ

の索引の使用を選択する場合があります。このため、最初に調整することに決めた文をチューニングした後、アプリケーションのパフォーマンスを再検査し、SQL トレース機能を利用すべきです。

## 索引を使用しない文の記述

場合によっては、既存の索引を使うアクセス・パスを SQL 文で使いたくないこともあります。索引の選択性があまり優れておらず、全表走査の方が効率的であることがわかっている場合、このようにすることがあります。そのような索引アクセス・パスを使用可能にする条件が文に含まれている場合、次に示す方法の 1 つを使って、オプティマイザが全表走査を行うように強制することができます。

- 意味を変えないように文を修正することによって、索引アクセス・パスを使用不能にできます。
- FULL ヒントを使って、オプティマイザが索引走査のかわりに全表走査を選択するように強制することができます。
- INDEX ヒント、INDEX\_COMBINE ヒント、または AND\_EQUAL ヒントを使用して、オプティマイザが、ある索引（またはリストされている索引セット）を別の索引（または索引セット）のかわりに使用するように強制することができます。

Oracle の将来のバージョンではオプティマイザの動作が変更される可能性があるため、最初に示したような方法を使ってアクセス・パスを選択するのは、長期的な計画には適していません。そのかわりに、オプティマイザに特定のアクセス・パスを提示するためにヒントを使用してください。

## 索引の有効性の調査

索引が優れているかどうかを判別する方法は、索引を作成して、それを分析し、問合せに EXPLAIN PLAN を使用して、オプティマイザがその索引を使用するかどうかを確認することです。オプティマイザで使用される場合は、メンテナンスにコストがかからない限り、その索引を保持してください。ただし、この方法では、非常に時間がかかりリソースが大量に消費されます。適切な方法としては、計画のオプティマイザ・コスト（EXPLAIN PLAN 出力の最初の行）を索引の有無で比較することができます。

パラレル実行は、索引を効果的に利用します。このオプションはパラレル索引範囲走査は実行しませんが、ネスト・ループ・ジョインを実行するためにパラレル索引参照を実行します。索引の選択性が高い（索引項目あたりの行数が少ない）場合は、パラレル表走査よりも順次索引参照を使うことをお勧めします。

## 高速全索引走査の使用

高速全索引走査は、問合せで必要なすべてのキーを含む索引がある場合に、全表走査の代替となります。高速全走査は表走査と同様に、マルチブロック I/O を使用し、パラレル化できるので通常の全索引走査より高速です。ただし、通常の索引走査とは異なり、キーは使用できず、行が必ずしもソート順で戻されるとは限りません。次の問合せと計画を使って、この機能を説明します。

```
SELECT COUNT(*) FROM t1, t2
WHERE t1.c1 > 50 and t1.c2 = t2.c1;
```

計画は次のようになります。

```
SELECT STATEMENT
  SORT AGGREGATE
    HASH JOIN
      TABLE ACCESS1FULL
        INDEXT2_C1_IDX FAST FULL SCAN
```

表 T2(C2) で必要なすべての列が索引 T2\_C1\_IDX に含まれるため、オプティマイザはこの索引に対して高速全索引走査を使用します。

高速全走査には次の制限があります。

- 表の索引付けされた列の少なくとも 1 つに NOT NULL 制約が必要です。
- 高速全索引走査をパラレルで実行したい場合は、索引に対する parallel 句が必要です。索引の並列度は個別に設定されます。索引は表の並列度を継承しません。
- 索引が分析済みであることを確認します。そうでない場合は、オプティマイザが索引を使用しないことがあります。

高速全走査には、特別な索引ヒント INDEX\_FFS があります。この形式と引数は、通常の INDEX ヒントと同じです。

**関連項目：** 7-43 ページの「[INDEX\\_FFS](#)」

## 索引の再作成

索引を縮小し分断化された領域を最小化するため、または索引の記憶特性を変更するために索引を再作成する場合があります。既存の索引のサブセットとなる新しい索引を作成するとき、または既存の索引を新しい記憶特性で再構築するとき、Oracle は、実表ではなく既存の索引を使用してパフォーマンスを向上させることがあります。

ただし、既存の索引よりも実表を使った方が効果的な場合もあります。多くの DML が実行された表の索引を考えてみてください。DML のために索引のサイズが大きくなり、各ブロックが 50% 以下しか満たされなくなる場合があります。索引が表のほとんどの列を参照している場合、索引は表よりも大きくなりかねません。その場合は、索引よりも実表を使っ

た方が、索引の再作成が速くできます。あるいは、元の索引の列のサブセット上に新しい索引を作成することもできます。

たとえば、NAME および CUSTID、PHONE、ADDR、BALANCE という列を持つ CUST という名前の表と、この表の列 NAME および CUSTID、BALANCE に関する I\_CUST\_CUSTINFO という名前の索引があるとします。列 CUSTID および NAME に関する I\_CUST\_CUSTNO という名前の新しい索引を作成する場合は、次のように入力します。

```
CREATE INDEX I_CUST_CUSTNO ON CUST(CUSTID,NAME);
```

Oracle は、表全体にアクセスするのではなく、自動的に既存の索引 (I\_CUST\_CUSTINFO) を使って新しい索引を作成します。使用する構文は、I\_CUST\_CUSTINFO が存在していないときと同じです。

同様に、EMP 表の列 EMPNO と MGR に関する索引があるときに、この複合索引の記憶特性を変更する場合、Oracle は、既存の索引を使用して新しい索引を作成します。

ALTER INDEX ... REBUILD 文は、既存の索引を再編成または圧縮するため、または既存の索引の記憶特性を変更するために使用します。REBUILD 文は、新しい索引の基礎として既存の索引を使用します。STORAGE (エクステンツの割当て用)、TABLESPACE (索引を新しい表領域に移動する) および INITRANS (エントリの最初の数を変更する) などのすべての索引記憶用の文がサポートされています。

ALTER INDEX ... REBUILD は、高速全走査機能を使用するので、通常は索引を削除して再作成するよりも高速です。この文は、マルチブロック I/O を使用して索引ブロックをすべて読み込んでから、ブランチ・ブロックを廃棄します。さらに、このアプローチには、再作成の実行中も問合せで旧索引を利用できるという利点があります (ただし、DML では利用できません)。

**関連項目：** CREATE INDEX 文と ALTER INDEX 文の詳細および索引の再作成の制限については、『Oracle8i SQL リファレンス』を参照してください。

## 索引の圧縮

COALESCE オプションを持つ ALTER INDEX 文を使用して索引のリーフ・ブロックを合わせることができます。それにより、索引のリーフ・レベルを結合して空きブロックにし、再利用できます。また、索引をオンラインで再作成することもできます。

この文の構文の詳細は、『Oracle8i SQL リファレンス』と『Oracle8i 管理者ガイド』を参照してください。

## 一意性を施行するための一意でない索引の使用法

UNIQUE 制約、または PRIMARY KEY 制約の一意性の側面に関して、一意性を施行するために、表の既存の一意でない索引を使用できます。このアプローチの利点は、制約が使用禁止にされているときでも索引が使用可能であり、有効であるということです。したがって、使用禁止にされている UNIQUE 制約または PRIMARY KEY 制約を使用可能にするために、その制約に対応付けられている UNIQUE 索引を再作成する必要はありません。これにより、大規模表で操作を使用可能にする際に時間を大幅に節約できます。

さらに、一意でない索引を使用して一意性を施行すると、索引の重複を排除できます。主キー列がすでに複合索引の同一キーとして組み込まれている場合、その列に対する UNIQUE 索引は不要です。既存の索引を使用して、制約を使用可能にして施行できるので、索引を複製しないことによってスペースを大幅に節約できます。

## 未検査使用可能制約の使用法

使用可能未検査制約は、使用可能検査済み制約と同じように動作します。制約を使用可能未検査状態にすることは、表に入力された新規データが制約に準拠しなければならないことを意味します。既存のデータはチェックされません。制約を使用可能未検査状態にすることで、表をロックしないで制約を使用可能にすることができます。

制約を使用禁止から使用可能に変更する場合、表をロックする必要があります。使用可能化操作の間に表の操作が制約に準拠していることを保証するメカニズムがないため、新たに DML、問合せまたは DDL を実行できません。使用可能未検査状態では、制約に違反する操作は表に対して実行されません。

表の並列一貫読み込み問合せによって使用可能未検査制約を検査済みにすることで、制約に違反するデータがあるかどうかを判断できます。ロックは実行されず、使用可能化操作は表に対する読み込み機能または書き込み機能をブロックしません。さらに、使用可能未検査制約は並列で検査済みにすることができます。複数の制約を同時に検査済みにし、並列問合せを使用して各制約の妥当性検査を実行できます。

制約と索引を持つ表を作成するアプローチは次のとおりです。

1. 制約を持つ表を作成します。NOT NULL 制約には名前を付けなくても構いませんが、検査済み使用可能で作成してください。その他の制約 (CHECK、UNIQUE、PRIMARY KEY および FOREIGN KEY) にはすべて名前を付け、" 使用禁止で作成 " します。

---

**注意：** デフォルトでは、制約は ENABLED 状態で作成されます。

---

2. 旧データを表にロードします。
3. 制約に必要な索引を始めとして、すべての索引を作成します。
4. 未検査の制約をすべて使用可能にします。これは、外部キーの前に主キーに対して行ってください。

5. ユーザーがデータを問合せおよび変更できるようにします。
6. 制約ごとに個別の ALTER TABLE 文を使用して、すべての制約を検査します。これは、外部キーの前に主キーに対して行ってください。

次に例を示します。

```
CREATE TABLE t (a NUMBER CONSTRAINT apk PRIMARY KEY DISABLE,
                b NUMBER NOT NULL);
CREATE TABLE x (c NUMBER CONSTRAINT afk REFERENCES t DISABLE);
```

ここでは、インポートまたは高速ローダーを使用してデータを t にロードします。

```
CREATE UNIQUE INDEX tai ON t (a);
CREATE INDEX tci ON x (c);
ALTER TABLE t MODIFY CONSTRAINT apk ENABLE NOVALIDATE;
ALTER TABLE x MODIFY CONSTRAINT afk ENABLE NOVALIDATE;
```

これで、ユーザーは t で挿入、更新、削除および選択を実行できるようになります。

```
ALTER TABLE t ENABLE CONSTRAINT apk;
ALTER TABLE x ENABLE CONSTRAINT afk;
```

これで、制約は検査済み使用可能になりました。

**関連項目：** 整合性制約の詳細は、『Oracle8i 概要』を参照してください。

## 関数ベース索引の使用方法

関数ベース索引は、式の索引です。オラクル社では、使用できる状況であれば必ず関数ベース索引を使用することをお薦めしています。関数ベース索引は、LOB または REF 以外の列で索引を使用する際に定義します。ネストした表の列およびオブジェクトのタイプにこれらの列を組み込むことはできません。

関数ベース索引は、反復可能な SQL 関数ならどれに対しても作成できます。オラクル社では、ORDER BY 句での範囲走査と関数に、関数ベースの索引を使用することをお薦めしています。

関数ベース索引を使用すると、WHERE 句に関数を持つ文を効率的に評価できます。関数ベース索引を作成して、索引で計算集中型の式を実現できます。これにより、Oracle は、SELECT 文および DELETE 文を処理する際に式の値の計算をバイパスできます。ただし、INSERT 文や UPDATE 文を処理する際は、文を処理する関数が Oracle によって評価されます。

たとえば、次の索引を作成します。

```
CREATE INDEX idx ON table_1 (a + b * (c - 1), a, b);
```

Oracle は、次のような問合せを処理する際にこれを使用できます。

```
SELECT a FROM table_1 WHERE a + b * (c - 1) < 100;
```

UPPER(column\_name) キーワードまたは LOWER(column\_name) キーワードで定義された関数ベース索引を使用すると、大文字と小文字を区別しない検索ができます。たとえば、次のような索引があります。

```
CREATE INDEX uppercase_idx ON emp (UPPER(empname));
```

これを使用すると、次のような問合せの処理が容易になります。

```
SELECT * FROM emp WHERE UPPER(empname) = 'MARK';
```

また、関数ベース索引は、SQL 文で効率的な言語照合を実現する NLS ソート索引にも使用できます。

Oracle では、DESC のマークのある列の索引は、関数ベース索引として処理されます。DESC のマークのある列は、降順でソートされます。

---

---

**注意：** セッション・パラメータ QUERY\_REWRITE\_ENABLED を TRUE に設定して、関数ベース索引を問合せで使用できるようにしてください。QUERY\_REWRITE\_ENABLED が FALSE だと、関数ベース索引内の式の値の取得に関数ベース索引を使用できません。ただし、実際の列の値の取得に関数ベース索引を使用することはできます。

---

---

## 関数ベース索引と索引構成表

IOT での 2 次索引に関数ベース索引にすることもできます。

## ビットマップ索引の使用方法

この項では、次のトピックについて説明します。

- [ビットマップ索引の使用のタイミング](#)
- [ビットマップ索引の作成](#)
- [ビットマップ索引のための初期化パラメータ](#)
- [B\\* ツリー索引に対するビットマップ・アクセス計画の使用方法](#)
- [ビットマップ索引のサイズの推定](#)
- [ビットマップ索引の制約事項](#)

**関連項目：** ビットマップ索引の概要は、『Oracle8i 概要』を参照してください。



## ビットマップ索引の使用のタイミング

この項では、特定の表に対するビットマップ索引を考慮するときに検証する必要がある、索引の3つの側面（パフォーマンス、記憶領域およびメンテナンス）について説明します。

### パフォーマンスに関する考慮事項

ビットマップ索引は、次の特性を持つ問合せのパフォーマンスを非常に向上させる可能性があります。

- カーディナリティが低いまたは中位の列に関する条件が WHERE 句に複数含まれている。
- カーディナリティが低いまたは中位の列に関する個々の条件が大量の行を選択する。
- カーディナリティが低いまたは中位の列の一部または全部に対してビットマップ索引が作成されている。
- 問合せの対象となる表に多数の行が含まれている。

複数のビットマップ索引を使って、単独の表に対する条件を評価できます。このため、ビットマップ索引は、長い WHERE 句を含む複合非定型問合せにとって非常に有効です。ビットマップ索引は、集合問合せでもスター・スキーマでの結合の最適化でも最適なパフォーマンスを提供します。

**関連項目：** 詳細は、『Oracle8i 概要』に記載されている逆結合と半結合の説明を参照してください。

### 記憶領域に関する考慮事項

ビットマップ索引は、複数列（または連結）B\* ツリー索引が使う記憶領域に比べると、はるかに記憶領域を節約できます。B\* ツリー索引だけを含むデータベースの場合は、1 回の問合せの中でいっしょにアクセスされることが多い列を予想し、それらの列に対して複合 B\* ツリー索引を作成する必要があります。

この B\* ツリー索引は、大量の記憶領域が必要になるだけでなく、順序付けの必要もあります。つまり、(MARITAL\_STATUS、REGION、GENDER) の B\* ツリー索引は、REGION と GENDER だけにアクセスする問合せでは役に立ちません。データベースに関する完全な索引を作成するには、これらの列の他の順列に対する索引を作成する必要があります。カーディナリティが低い列が3つある単純な事例では、6つの複合 B\* ツリー索引が考えられます。どの複合 B\* ツリー索引を作成するかを決定するときに、ディスク容量とパフォーマンスのどちらが重要であるかを考慮する必要があります。

ビットマップ索引を使うと、この問題が解決されます。ビットマップ索引は、問合せの実行中に効率よく結合されるため、小さな3つの単独列ビットマップ索引によって、6つの3列 B\* ツリー索引と同じ成果をあげることができます。

ビットマップ索引を使用すると、データ・ウェアハウジング環境では特に、B\* ツリー索引よりもずっと効率が上がります。ビットマップ索引を作成する目的は、スペースの効率的な使用だけではありません。さらに重要な目的として、実行の効率化があります。

ビットマップ索引は、一意キー列に対して作成されると、通常の B\* ツリー索引を上回る記憶領域を必要とします。ただし、同じ値が数百または数千もあるような列に対して作成された場合、一般にビットマップ索引のサイズは、通常の B\* ツリー索引の 25% 未満になります。ビットマップそのものは、圧縮形式で記憶されます。

ただし、B\* ツリー索引とビットマップ索引の相対サイズをただ比較するだけでは、有効性の正確な基準にはなりません。パフォーマンス上の特性がそれぞれ異なるため、カーディナリティが高いデータでは B\* ツリー索引を維持し、カーディナリティが低いデータではビットマップ索引を作成してください。

### メンテナンスに関する考慮事項

ビットマップ索引はデータ・ウェアハウス・アプリケーションでは有効ですが、挿入、更新および削除の同時実行が集中的に行われる OLTP アプリケーションには適しません。データ・ウェアハウス環境では、通常、データは大規模な挿入と更新によってメンテナンスされます。索引のメンテナンスは、DML 操作が終わるまで延期されます。たとえば 1000 行を挿入する場合、挿入される行はソート・バッファに置かれた後、1000 個の索引エントリのすべての更新が一括処理されます。(このため、ビットマップ索引に対する挿入および更新のパフォーマンスを向上するためには、SORT\_AREA\_SIZE を適切に設定する必要があります)。したがって、各ビットマップ・セグメントの更新は、セグメント内の複数の行が変更された場合でも DML 操作ごとに 1 回だけ更新されます。

---

**注意：** 上記のソートは通常のソートであり、SORT\_AREA\_SIZE によって決まる通常のソート領域を使用します。6-17 ページの「[ビットマップ索引のための初期化パラメータ](#)」で説明する BITMAP\_MERGE\_AREA\_SIZE パラメータおよび CREATE\_BITMAP\_AREA\_SIZE パラメータは、パラメータ名が示す特定の操作だけに影響します。

---

UPDATE、DELETE、DROP TABLE などの DML 文および DDL 文は、従来の索引に作用するようにビットマップ索引にも作用します。つまり、一貫性モデルは同じです。1 つのキー値に対する圧縮されたビットマップは、1 つ以上のビットマップ・セグメントで構成され、各セグメントのサイズは最大でも半ブロックです(ただしそれよりも小さくなる可能性があります)。ロック単位は、このようなビットマップ・セグメントです。このため、多数のトランザクションが同時に更新を行う環境では、パフォーマンスが影響を受ける可能性があります。大量の DML 操作によって索引のサイズが増大し、問合せのパフォーマンスが低下した場合は、ALTER INDEX ... REBUILD 文を使用して索引を圧縮することで、効率のよいパフォーマンスが回復できます。

B\* ツリー索引の各エントリには、1 つの ROWID が含まれています。したがって、この索引エントリがロックされると、単独の行がロックされます。ビットマップ索引では、1 つのエントリにある範囲の ROWID が含まれている可能性があります。ビットマップ索引のエント

リがロックされると、その範囲に含まれている ROWID がすべてロックされます。この範囲に含まれる ROWID の数が、同時実行性に影響を与えます。たとえば、一意の値を持つ列に対するビットマップ索引では、値ごとに 1 つの ROWID がロックされます。この場合、同時実行性は B\* ツリー索引と同じです。同時実行性は、1 つのビットマップ・セグメント内の ROWID が増加するにつれて低下します。

ロックの問題は DML 操作に影響するため、負荷が高い OLTP 環境にも影響を与える可能性があります。ただし、ロックの問題は、問合せのパフォーマンスには影響しません。他の種類の索引と同じように、ビットマップ索引の更新は、コストがかかる操作です。それにもかかわらず、単独の文で多数の行の挿入または多くの更新が実行される大量挿入または大量更新では、ビットマップ索引を使ったパフォーマンスの方が、通常の B\* ツリー索引よりも優れています。

## ビットマップ索引の作成

ビットマップ索引を作成するには、次に示す CREATE INDEX 文で BITMAP キーワードを使用します。

```
CREATE BITMAP INDEX ...
```

複数列（連結）ビットマップ索引もサポートされています。最大 32 列に対して定義できます。索引に対する DROP、ANALYZE、ALTER などの SQL 文でも、特別なキーワードを使わないでビットマップ索引を参照できます。ビットマップ索引の制限の詳細は、『Oracle8i SQL リファレンス』を参照してください。

---

**注意：** ビットマップ索引を使用するには、COMPATIBLE 初期化パラメータに 7.3.2 以上を設定する必要があります。

---

## 索引のタイプ

システム索引ビュー USER\_INDEXES、ALL\_INDEXES および DBA\_INDEXES は、TYPE 列に BITMAP という単語を表示してビットマップ索引を示します。ビットマップ索引は、UNIQUE としては宣言できません。

## ヒントの使用

INDEX ヒントは、従来の索引と同じ方法でビットマップ索引でも使えます。

INDEX\_COMBINE ヒントを使用して、最もコスト効率の高いヒントをオプティマイザに示します。オプティマイザは、WHERE 句の条件で与えられた、結合できる可能性のあるすべての索引を認識します。ただし、これらすべてを使うのはコスト効率がよくない可能性があります。INDEX\_COMBINE の方が INDEX よりも用途の広いヒントなので、ビットマップ索引には INDEX\_COMBINE を使用することをお勧めします。

使用する索引を決めるとき、オプティマイザは、ヒントで指定されている索引の他に、コスト効率がよいように見えるヒントなしの索引も組み込みます。ヒントの引数として特定の索

引が与えられた場合、オプティマイザは、指定されたビットマップ索引を組み合わせで使おうとします。

ヒントに索引が指定されていない場合は、すべての索引がヒントで指定されているとみなされます。したがって、オプティマイザは、コスト効率を考慮しないで、WHERE 句に与えられている可能なものの組合せをできるだけ多く行おうとします。オプティマイザは、コスト効率がよいか悪いかにかかわらず、計画内のヒントで指定された索引を常に使用しようとしています。

**関連項目：** 7-42 ページの「[INDEX\\_COMBINE](#)」

## パフォーマンスと記憶領域に関するヒント

ビットマップ索引を使って最適なパフォーマンスと記憶領域の使用状況を達成するには、次の考慮事項に注意してください。

- ブロック・サイズが大きいと、ビットマップ索引の記憶効率と検索効率が向上する
- ビットマップ索引をできる限り小さく圧縮するためには、NULL 値を持たないすべての列に対して NOT NULL 制約を宣言する必要がある
- 圧縮されたビットマップ表現には、可変長データ型よりも固定長データ型の方が適している

**関連項目：** ビットマップ出力の詳細は、第 13 章「[EXPLAIN PLAN の使用方法](#)」を参照してください。

## ROWID へのビットマップの効率的なマップ

ALTER TABLE 構文のある SQL 文を使用して、ROWID へのビットマップのマップを最適化します。MINIMIZE RECORDS\_PER\_BLOCK 句を使用するとこの最適化が使用可能になり、NOMINIMIZE RECORDS\_PER\_BLOCK 句を使用すると使用禁止になります。

使用可能にした場合、Oracle によって表が操作されてブロックのレコードの最大数が突き止められ、この表がこの最大数に制限されます。これにより、ビットマップ索引でブロックごとに割り当てられるビットが少なくなり、ビットマップ索引が小さくなります。この文が表に課すブロックおよびレコードの割当て制限は、ビットマップ索引にのみ役立ちます。したがって、ビットマップ索引が多くない表に対してこのマップを使用することはお薦めしません。

**関連項目：** 6-12 ページの「[ビットマップ索引の使用法](#)」。また、MINIMIZE 構文と NOMINIMIZE 構文の使用の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## NULL 値の索引付け

ビットマップ索引は NULL に対しても作成できますが、他のすべての種類の索引ではできません。たとえば、次の問合せを実行する、STATE 列と PARTY 列がある表を考えてみてください。

```
SELECT COUNT(*) FROM people WHERE state='CA' and party !='D';
```

NULL に索引付けすると、ビットマップ・マイナス計画が使用可能になります。つまり、'D' と NULL に等しい party のビットマップが 'CA' に等しい state ビットマップから引かれます。EXPLAIN PLAN の出力は次のようになります。

```
SELECT STATEMENT
SORT
  BITMAP CONVERSION
    BITMAP MINUS
      BITMAP MINUS
        BITMAP INDEX      SINGLE VALUE      STATE_BM
        BITMAP INDEX      SINGLE VALUE      PARTY_BM
        BITMAP INDEX      SINGLE VALUE      PARTY_BM
```

NOT NULL 制約が party に対して存在する場合、2 番目のマイナス (minus) 演算 (party が NULL の場合) は必要ではなくなり、除外されます。

## ビットマップ索引のための初期化パラメータ

次の 2 つの初期化パラメータはパフォーマンスに影響を与えます。

### CREATE\_BITMAP\_AREA\_SIZE

このパラメータは、ビットマップを作成するために割り当てるメモリー容量を決定します。デフォルト値は 8MB です。これより大きな値を設定すると、索引が短時間で作成される場合があります。カーディナリティが非常に低い場合、このパラメータには小さな値を設定できます。たとえば、カーディナリティが 2 しかない場合は、この値はメガバイト単位ではなくキロバイト単位にすることができます。一般的な規則として、カーディナリティが高くなるにつれて、最適のパフォーマンスを実現するために必要なメモリーが増加します。このパラメータは、システムまたはセッション・レベルで動的に変更できません。

### BITMAP\_MERGE\_AREA\_SIZE

このパラメータは、索引の範囲走査から検索したビットマップをマージするために使われるメモリー容量を決定します。デフォルト値は 1MB です。ビットマップ・セグメントをソートしてから単独のビットマップにマージする必要があるため、大きな値を設定するとパフォーマンスが向上します。このパラメータは、システムまたはセッション・レベルで動的に変更できません。

**関連項目：** ビットマップ索引の効率の向上については、6-16 ページの「[ROWID へのビットマップの効率的なマップ](#)」を参照してください。

## B\* ツリー索引に対するビットマップ・アクセス計画の使用方法

表に少なくとも 1 つのビットマップ索引が存在する場合、オプティマイザは、その表の標準の B\* ツリー索引を使用するビットマップ・アクセス・パスの使用を考慮します。このアクセス・パスには、B\* ツリー索引とビットマップ索引の組合せが関係する可能性があります。ただし、ヒントの中で特に指示されない限り、オプティマイザは単独の B\* ツリー索引を使用するビットマップ・アクセス・パスは生成しません。

B\* ツリー索引に対してビットマップ・アクセス・パスを使用するには、索引内に記憶されている ROWID をビットマップに変換する必要があります。このような変換の後で、ビットマップに対して使用できるさまざまなブール演算の使用が可能になります。例として、次の問合せを考慮します。この問合せでは、列 C1 にビットマップ索引が、列 C2 と C3 に通常の B\* ツリー索引があります。

```
EXPLAIN PLAN FOR
SELECT COUNT(*) FROM T
WHERE
  C1 = 2 AND C2 = 6
OR
  C3 BETWEEN 10 AND 20;
SELECT STATEMENT
  SORT AGGREGATE
    BITMAP CONVERSION COUNT
      BITMAP OR
        BITMAP AND
          BITMAP INDEX C1_IND SINGLE VALUE
          BITMAP CONVERSION FROM ROWIDS
            INDEX C2_IND RANGE SCAN
          BITMAP CONVERSION FROM ROWIDS
        SORT ORDER BY
          INDEX C3_IND RANGE SCAN
```

ここでは、BITMAP CONVERSION 行ソースの COUNT オプションによって、問合せに該当する行の数をカウントしています。また、計画の中には、B\* ツリー索引で検索された ROWID からビットマップを生成するための FROM ROWIDS 変換が含まれています。計画内に ORDER BY ソートがあるのは、列 C3 を対象とする条件によって B\* ツリー索引から戻される ROWID リストが複数あるという事実のためです。これらのリストは、ビットマップに変換される前にソートされます。

## ビットマップ索引のサイズの推定

ビットマップ索引のサイズの厳密な測定はできませんが、サイズの推定はできます。この項では、B\* ツリー索引のサイズを計算して、表のビットマップ索引のサイズを推測する方法を説明します。カーディナリティ、NOT NULL 制約および固有の値の数がビットマップのサイズにどのように影響するかも示します。

ある表に対するビットマップ索引のサイズを推定するには、表の B\* ツリー索引のサイズから推測することができます。次のアプローチに従ってください。

1. 『Oracle8i 概要』に記載されている標準計算式を使用して、表の B\* ツリー索引のサイズを計算します。
2. 表データのカーディナリティを判断します。
3. カーディナリティ値から、[図 6-2](#) または [図 6-3](#) のグラフに従って、ビットマップ索引のサイズを推測します。

100 万行の表に関して、さまざまな数の固有の値を持つ列の索引サイズを B\* ツリー索引とビットマップ索引について [図 6-2](#) に示します。[図 6-2](#) を使用して、その表の B\* ツリー索引のサイズに対して相対的にビットマップ索引のサイズを推定できます。サイズは正確ではありません。結果は表によってある程度異なります。

グラフの生成には、ランダムに分散されたデータが使用されていることに注意してください。ユーザーのデータで、特定の値がより緊密にクラスタ化される傾向がある場合は、このグラフで示すよりもはるかに小さなビットマップ索引を生成できるかもしれません。列に NOT NULL 制約が含まれている場合、ビットマップ索引はグラフのものよりもわずかに小さくなる可能性があります。

[図 6-3](#) は、500 万行の表について同様のデータを示しています。カーディナリティが 100,000 を超える場合、ビットマップ索引のサイズは [図 6-2](#) の場合ほど急激には増加しません。表の行数が多くなるほど、あるカーディナリティで繰り返される値が多くなります。

図 6-2 ビットマップ索引のサイズの推定 : 100 万行の表

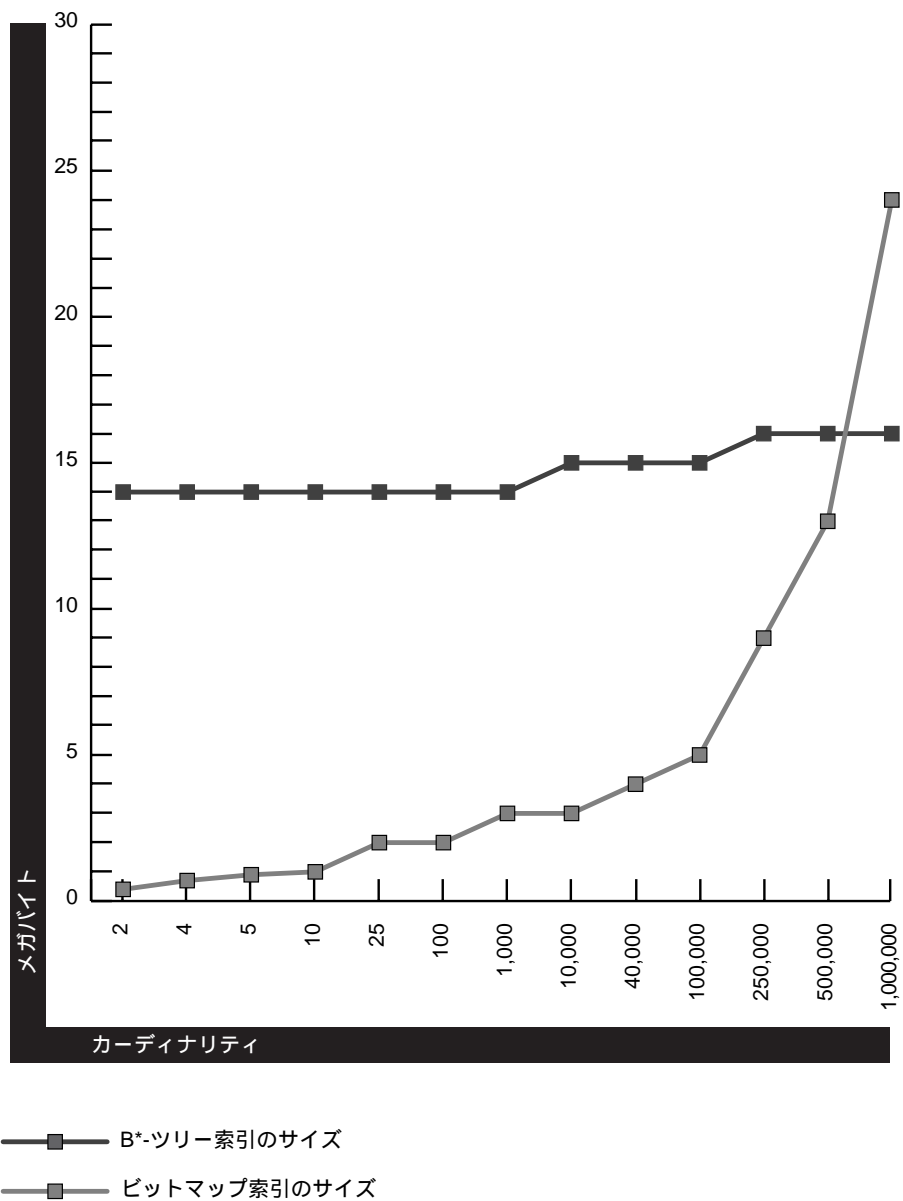
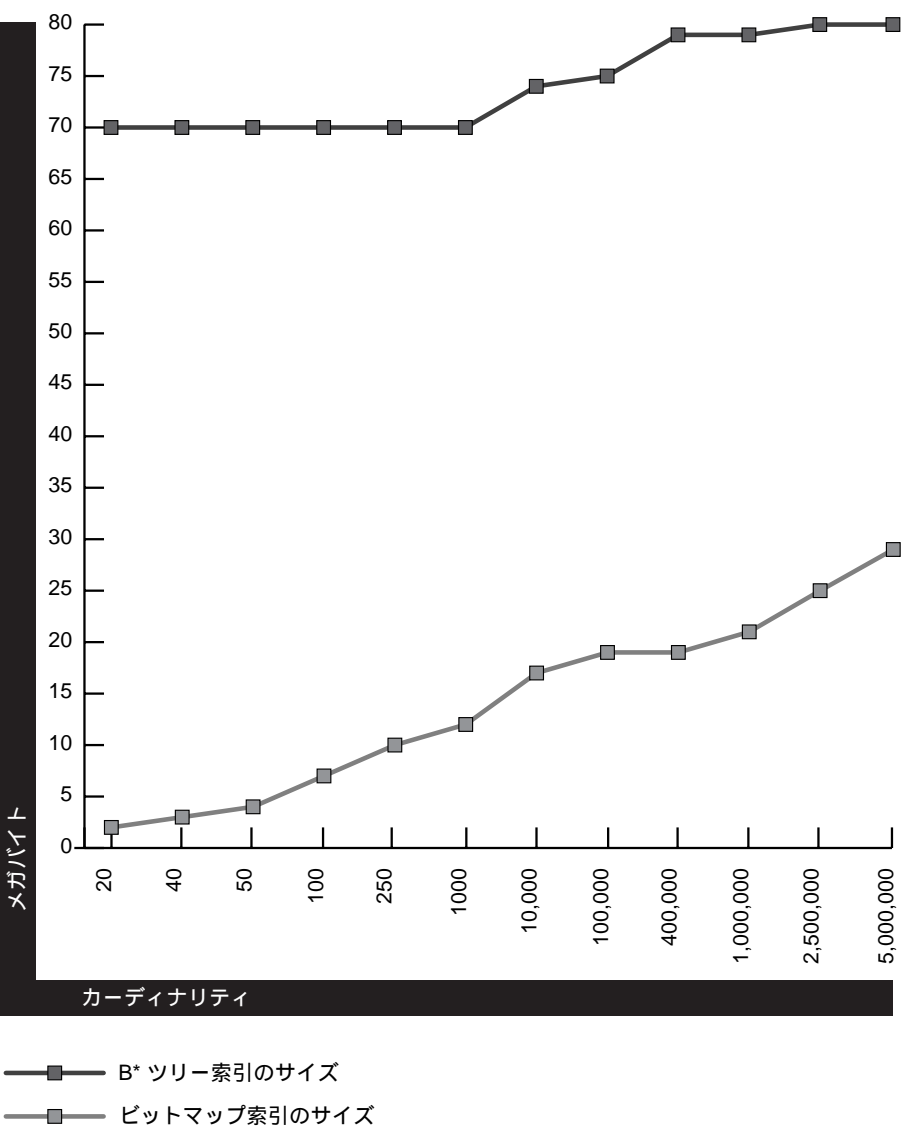




図 6-3 ビットマップ索引のサイズの推定：500 万行の表



## ビットマップ索引の制約事項

ビットマップ索引には、次の制約事項があります。

- ダイレクト・ロードを伴うビットマップ索引には、SORTED\_INDEX フラグが適用されません。
- ビットマップ索引は、ルールベース・オプティマイザによる考慮の対象になりません。
- ビットマップ索引は、参照の整合性をチェックするためには使用できません。

## ドメイン索引の使用方法

ドメイン索引は、ユーザー定義の索引タイプで指定された索引作成論理で作成されます。通常、ユーザー定義の索引タイプはデータ・カートリッジに含まれます。たとえば、Spatial カートリッジには、Spatial データに索引付けする SpatialIndextype があります。

索引タイプを使用すると、特定の演算子の述部に合うデータに効率よくアクセスできます。たとえば、SpatialIndextype を使用すると、ある枠ボックスにオーバーラップする Spatial データを効率よく検索できます。

ドメイン索引の作成と保守で指定できるパラメータは、カートリッジによって異なります。同様に、ドメイン索引のパフォーマンスと記憶域の特性は、カートリッジ固有のマニュアルを参照してください。

次の点については、適切なカートリッジ・マニュアルを参照してください。

- 索引付けできるデータ型
- 使用できる索引タイプ
- 索引タイプで使用できる演算子
- ドメイン索引を作成およびメンテナンスする方法
- 問合せで演算子を効率よく使用する方法
- パフォーマンス特性の内容

---

**注意：** 索引タイプは CREATE INDEXTYPE SQL 文でも作成できます。

---

**関連項目：** SpatialIndextype の詳細は、『Oracle8i Spatial 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。

## クラスタの使用法

表をクラスタ化するかどうかを判断するために、次のガイドラインに従ってください。

- アプリケーションが結合処理で頻繁にアクセスする表をクラスタ化することを検討します。
- アプリケーションが表の結合を頻繁には行わない場合、または共通の列の値を頻繁に修正する場合、それらの表はクラスタ化しません。表をクラスタ化した場合、Oracle は修正された行を別のブロックへ移行して、クラスタをメンテナンスしなければならない場合もあります。このため、クラスタ化していない表の値を修正する場合よりも、行のクラスタ・キー値を修正する場合の処理時間は長くなる可能性があります。
- アプリケーションが複数の表の 1 つについてだけ頻繁に全表走査を行う場合、それらの表はクラスタ化しません。クラスタ化した表の全表走査に要する処理時間は、クラスタ化していない表の全表走査よりも長くなる可能性があります。複数の表がいっしょに格納されているため、Oracle が読み込む必要のあるブロックは多くなります。
- マスター・レコードを選択してからディテール・レコードを選択することがよくある場合、マスター / ディテール表をクラスタ化することを検討します。ディテール・レコードはマスター・レコードと同じデータ・ブロックに格納されるため、選択時にそれらがメモリー内に存在している可能性が高くなり、Oracle による I/O 処理が少なくなる場合があります。
- 同じマスターについて多くのディテール・レコードを選択することがよくある場合、ディテール表だけをクラスタに格納することを検討します。このようにすれば、同じマスターについて多くのディテール・レコードを選択する問合せのパフォーマンスが改善され、マスター表に対する全表走査のパフォーマンスも低下させずに済みます。
- 同じクラスタ・キー値を持つすべての表からのデータが、1 つまたは 2 つの Oracle ブロックを超える場合、それらの表はクラスタ化しません。クラスタ化した表の行をアクセスする場合、Oracle はその値を持つ行を含んでいるブロックをすべて読み込みます。これらの行が複数のブロックを使っている場合、単一行をアクセスすることによって、クラスタ化していない表で同じ行をアクセスする場合よりも多くの読み込み処理が必要になる場合があります。

アプリケーションのニーズに関して、クラスタの利点と欠点を検討してください。たとえば、結合文のパフォーマンス向上が、クラスタ・キー値を修正する文のパフォーマンス低下に優先する場合もあるでしょう。表をクラスタ化した場合と別々に格納した場合について実験して、処理時間を比較するとよいでしょう。クラスタを作成するには、CREATE CLUSTER コマンドを使用します。

**関連項目：** クラスタの作成の詳細は、『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。

## ハッシュ・クラスタの使用方法

ハッシュ・クラスタは、ハッシュ関数をそれぞれの行のクラスタ・キー値に適用することによって、表データをグループ分けします。同じクラスタ・キー値を持つすべての行が、ディスク上に格納されます。アプリケーションのニーズに関して、ハッシュ・クラスタの利点および欠点を検討してください。特定の表をハッシュ・クラスタに格納する場合と、索引付きで単独に格納する場合を実験して、処理時間を比較するとよいでしょう。この項では、次のトピックについて説明します。

- [ハッシュ・クラスタの用途](#)
- [ハッシュ・クラスタの使用方法](#)

## ハッシュ・クラスタの用途

ハッシュ・クラスタを使う場合を判断するために、次のガイドラインに従ってください。

- 同じ列または列の組合せを使用する等価条件を WHERE 句が含む場合に、WHERE 句を持つ SQL 文がよくアクセスする表を格納するために、ハッシュ・クラスタを使用することを検討します。この列または列の組合せをクラスタ・キーとして指定します。
- 将来挿入する行およびすぐに挿入する行を含めて、特定のクラスタ・キー値を持つすべての行を保持するために必要な領域を決定できる場合、ハッシュ・クラスタに表を格納します。
- データベースの領域が不十分で、将来挿入する行に対して追加の領域を割り当てる余裕がない場合、ハッシュ・クラスタは使用しない。
- 常に成長する表を保持するために、時々新たに、より大きなハッシュ・クラスタを作成するプロセスが非実用的である場合、その表を格納するためにハッシュ・クラスタは使用しない。
- アプリケーションが全表走査をしばしば実行し、表が拡大することを見越してハッシュ・クラスタに大きな領域を割り当てる必要がある場合は、その表をハッシュ・クラスタに格納しません。そのような全表走査では、いくつかのブロックにはほとんど行が含まれていなくても、ハッシュ・クラスタに割り当てられているブロックをすべて読み込まなければなりません。表を単独で格納することによって、全表走査によって読み込まれるブロック数が減少する場合があります。
- アプリケーションがクラスタ・キー値を頻繁に修正する場合、ハッシュ・クラスタに表は格納しません。表をクラスタ化した場合、Oracle は修正された行を別のブロックへ移行して、クラスタをメンテナンスしなければならない場合があります。このため、クラスタ化していない表の値を修正する場合よりも、行のクラスタ・キー値を修正する場合の処理時間は長くなる可能性があります。
- このリストの先の項目に基づいてハッシングが表に対して適切な場合、表が他の表と結合されることがよくあるかどうかとは無関係に、単一の表をハッシュ・クラスタに格納すると有効な場合があります。

## ハッシュ・クラスタの作成

ハッシュ・クラスタを作成するには、HASHKEYS パラメータを指定した CREATE CLUSTER 文を使用します。

ハッシュ・クラスタを作成するとき、CREATE CLUSTER 文の HASHKEYS パラメータを使用して、ハッシュ・クラスタに対するハッシュ値の数を指定しなければなりません。ハッシュ走査について最高のパフォーマンスを得るために、少なくともクラスタ・キー値の数と同じ大きさの HASHKEYS 値を選択してください。そのような値によって、衝突の可能性や複数のクラスタ・キー値によって同じハッシュ値が生成される可能性が低減されます。衝突が発生すると、Oracle は、ハッシュ走査の実行後に正しいクラスタ・キー値に対して各ブロック内の行をテストしなければなりません。衝突によってハッシュ走査のパフォーマンスが低下します。

Oracle は指定された HASHKEYS 値を常に最も近い素数に切り上げ、実際のハッシュ値の数を取得します。この切り上げは、衝突を低減することが目的です。

**関連項目：** ハッシュ・クラスタの作成の詳細は、『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。



---

## オプティマイザ・モード、プラン・スタビリティおよびヒント

この章では、コストベースの最適化とルールベースの最適化について説明します。また、プラン・スタビリティを使用してパフォーマンス特性を保つ方法と、コストベース・オプティマイザに移行する方法を説明します。さらに、Oracle パフォーマンスを向上させるヒントの使用方法も説明します。トピックは次のとおりです。

- [コストベースの最適化](#)
- [統計情報の生成](#)
- [自動統計収集](#)
- [ルールベースの最適化](#)
- [実行計画を保存するプラン・スタビリティの使用](#)
- [アウトラインの作成](#)
- [OUTLN\\_PKG パッケージでのストアド・アウトラインの管理](#)
- [コストベース・オプティマイザのプラン・スタビリティ・プロシージャ](#)
- [ヒントの使用](#)

**関連項目：** オプティマイザ、アクセス方法、結合操作およびパラレル実行については、『Oracle8i 概要』を参照してください。

## コストベースの最適化

この項では、次のことについて説明します。

- [コストベース・アプローチの適用時期](#)
- [コストベース・アプローチの使用方法](#)
- [コストベース・アプローチの目標の選択](#)
- [分散が均一でないデータのヒストグラムの使用方法](#)
- [統計情報の生成](#)
- [自動統計収集](#)
- [コストベースの最適化計画に影響するパラメータ](#)
- [オブティマイザの索引使用方法に影響するパラメータ](#)
- [コストベース・アプローチの使用上のヒント](#)

また、次のことについても簡単に説明します。

- [ルールベースの最適化](#)

## コストベース・アプローチの適用時期

一般には、コストベースの最適化アプローチはいつでも使用できます。ルールベースのアプローチは既存のアプリケーションでも利用できますが、新しいオブティマイザ機能はコストベースのアプローチを使用します。

次の機能はコストベースの最適化でのみ使用できます。

- パーティション表
- 索引構成表
- 逆索引
- パラレル実行
- スター変換
- スター結合

---

---

**注意：** 正確な実行計画を完成させるには、各表の統計を集める必要があります。

---

---

一般に、コストベース・アプローチは、ルールベース・アプローチで選択される実行計画と同等またはそれ以上に優れた実行計画を選択します。特に複数の結合または複数の索引を持



つ大規模な問合せで、優れた実行計画を選択します。コストベース・アプローチでは、担当者自身による SQL 文のチューニングが不要になるので、生産性も向上します。

スター問合せのパフォーマンスを効率的にするには、コストベースの最適化を使用します。同様に、ハッシュ結合やヒストグラムにもこれを使用してください。コストベースの最適化は、常にパラレル実行およびパーティション表とともに使用します。コストベース・オブティマイザの有効性を保つには、統計を最新に保つ必要があります。

**関連項目：** ルールベース・オブティマイザからコストベース・オブティマイザへの移行については、7-29 ページの「[コストベース・オブティマイザへの移行に対するアウトラインの使用](#)」を参照してください。

## コストベース・アプローチの使用方法

SQL 文に対してコストベースの最適化を使用するには、SQL 文がアクセスする表の統計を収集し、次のいずれかの方法でコストベースの最適化を使用可能にします。

- OPTIMIZER\_MODE 初期化パラメータが、デフォルト値である CHOOSE に設定されていることを確認します。
- 使用中のセッションだけでコストベースの最適化を使用可能にするには、ALL\_ROWS または FIRST\_ROWS オプションを指定して ALTER SESSION SET OPTIMIZER\_MODE 文を発行します。
- SQL 文ごとにコストベースの最適化を使用可能にするには、RULE 以外のヒントを使用します。

コストベースのオブティマイザが生成する計画は、表のサイズによって決まります。アプリケーションのプロトタイプをテストするために、少量のデータでコストベース・オブティマイザを使用した場合は、完全なサイズのデータベースのために選択される計画とプロトタイプのために選択された計画とが同じになるとは限りません。

**関連項目：** 最新バージョンのコストベース・オブティマイザへのアップグレードについては、7-30 ページの「[RDBMS アップグレードとコストベース・オブティマイザ](#)」を参照してください。

## コストベース・アプローチの目標の選択

オブティマイザで生成された実行計画は、オブティマイザの目標によって変わる可能性があります。スループットを最高にするための最適化では、索引走査よりも全表走査、ネスト・ループ・ジョインよりもソート / マージ結合となる場合が多いでしょう。しかし、応答時間を最短にするための最適化は、索引走査またはネスト・ループ・ジョインとなる場合が多いでしょう。

たとえば、ネストしたループ操作またはソート / マージ操作で実行できる結合文を考えます。ネストしたループ操作では、最初の行がより速く戻されるのに対して、ソート / マージ操作では全体の問合せ結果がより速く戻されるでしょう。目標がスループットの改善である

場合、オブティマイザは多くの場合にソート / マージ結合を選択します。目標が応答時間の改善である場合、オブティマイザは多くの場合ネスト・ループ・ジョインを選択します。

アプリケーションのニーズに基づいて、オブティマイザの目標を選択してください。

- Oracle Reports アプリケーションのように、バッチで実行されるアプリケーションの場合、最高のスループットを目標に最適化してください。アプリケーションを起動するユーザーの関心は、アプリケーションを完了するために必要な時間のみに向けられているため、通常バッチ・アプリケーションではスループットの重要度が高くなります。アプリケーションの実行中にユーザーが個々の文の結果を調べることはないため、応答時間はそれほど重要ではありません。
- Oracle Forms アプリケーションや SQL\*Plus の問合せのような対話形式のアプリケーションの場合、最短の応答時間を目標に最適化してください。対話形式のユーザーは、文がアクセスする最初の行を参照するために待機しています。このため、対話形式のアプリケーションでは応答時間が重要となります。
- ROWNUM を使用して行数を制限する問合せの場合には、応答時間を最短にするために最適化してください。ROWNUM 問合せの方法によって、応答時間のための最適化を行うことによって、最高の結果が生じます。

特に何も指定しない場合、コストベース・アプローチは、最高のスループットを目標に最適化を行います。次の方法でコストベース・アプローチの目標を変更できます。

- セッションのすべての SQL 文に対するコストベース・アプローチの目標を変更するには、ALL\_ROWS または FIRST\_ROWS オプションを指定して、ALTER SESSION SET OPTIMIZER\_MODE 文を発行します。
- 個々の SQL 文に対するコストベースのアプローチの目標を指定するには、ALL\_ROWS ヒントまたは FIRST\_ROWS ヒントを使用します。

**例:** 次の文は、セッションに対するコストベースのアプローチの目標を最短の応答時間に変更します。

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS;
```

## 分散が均一でないデータのヒストグラムの使用方法

データが均一に分散している場合は、コストベース・アプローチで特定の SQL 文の実行コストをかなり正確に突き止められます。したがって、問合せのコストを評価するためのヒストグラムはオブティマイザには必要ありません。

ただし、データの分散が均一でない場合は、特定の列のデータ分散パターンを示すヒストグラムを作成してください。これらのヒストグラムは、コストベース・オブティマイザが使用できるよう、Oracle によってデータ・ディクショナリに格納されます。

ヒストグラムは永続オブジェクトなので、使用する場合はメンテナンスとスペースのコストがかかります。ヒストグラムの計算は、データの分布が非常に片寄っている列に対してのみ行ってください。Oracle がヒストグラムの作成に使用する統計は、すべてのオブティマイザの統計と同様に、静的です。列のデータ分布が頻繁に変わる場合は、その列について新しい

ヒストグラム統計を集めます。これは、明示的に行うか、7-17 ページで説明している [自動統計収集機能](#) を使用して行います。

次の特性を持つ列では、ヒストグラムは有用ではありません。

- 列のすべての述語にバインド変数が使われている
- 列データが均一に分散している
- 問合せの WHERE 句で列が使用されていない
- 列が一意であり、等価述語でしか使用しない

## ヒストグラムの作成

問合せの WHERE 句で頻繁に使われる列で、データの分布が非常に片寄っている列に対してヒストグラムを作成します。それには、DBMS\_STATS パッケージの GATHER\_TABLE\_STATS プロシージャを使用します。たとえば、EMP 表の SAL 列に対して 10 バケットのヒストグラムを作成する場合は、次の文を発行します。

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS  
( 'scott', 'emp', METHOD_OPT => 'FOR COLUMNS SIZE 10 sal' );
```

SIZE キーワードは、ヒストグラムの最大バケット数を示します。SAL 列についてのヒストグラムは、給与額が同じ社員の数が極端に多く、それ以外の給与額の社員がほとんどいない場合に作成します。また、表の単一パーティションのヒストグラムを収集することもできます。

列統計情報が、データ・ディクショナリ・ビューの USER\_TAB\_COLUMNS、ALL\_TAB\_COLUMNS、および DBA\_TAB\_COLUMNS に表示されます。

ヒストグラムが、データ・ディクショナリ・ビューの USER\_HISTOGRAMS、DBA\_HISTOGRAMS、および ALL\_HISTOGRAMS に表示されます。

**関連項目：** DBMS\_STATS パッケージの詳細は、7-8 ページの [「DBMS\\_STATS パッケージでの統計収集」](#) を参照してください。  
ANALYZE 文およびそのオプションの詳細は、『Oracle8i SQL リファレンス』を参照してください。

## ヒストグラムのバケット数の選択

ヒストグラムのバケット数のデフォルト値は 75 です。ほとんどのデータ分布は、この値によって適切なレベルの詳細情報が得られます。ただし、ヒストグラム内のバケット数 (' サンプルング率 ' と呼ばれる) およびデータ分布は、すべてがヒストグラムの有用性に作用するため、最良の結果を得るためにはさまざまなバケット数で試してみる必要があります。

列内に発生頻度が高いものが比較的少ない場合は、バケット数の目盛を発生頻度の目盛より多く設定してください。

## ヒストグラムの表示

データベース内の既存のヒストグラムについての情報は、次のデータ・ディクショナリ・ビューを使用して検索できます。

- USER\_HISTOGRAMS
- ALL\_HISTOGRAMS
- DBA\_HISTOGRAMS

各列のヒストグラムのバケット数は、次の列にあります。

- USER\_TAB\_COLUMNS
- ALL\_TAB\_COLUMNS
- DBA\_TAB\_COLUMNS

**関連項目：** データ・ディクショナリ・ビューの列の説明、およびヒストグラムの使用方法と制限事項の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## 統計情報の生成

コストベース・アプローチは統計情報に依存するので、コストベース・アプローチを使用する前に、SQL 文がアクセスするすべての表、クラスタおよび索引の全タイプの統計を生成する必要があります。これらの表のサイズとデータ分布が頻繁に変化する場合、これらの統計を定期的に生成して、表のデータが正確に反映されるようにしてください。

Oracle は次の技法を使用して統計情報を生成します。

- ランダムなデータ・サンプリングに基づく見積り
- 正確な計算
- ユーザー定義の統計収集メソッドの使用

表統計の計算には多くの時間と領域がかかるので、正確な値が必要でなければ、計算ではなく表とクラスタの見積りを使用してください。その理由は次のとおりです。

- 正確な計算が行われると、常に正確な値が取得できます。ただし、計算は見積りよりも処理時間が長くなります。表の統計を計算するために必要な時間は、表の全表走査と表の行のソートを実行するために必要な時間とほぼ同じになります。
- 多くの場合、見積りは計算よりもかなり高速に処理されます。これは、見積りでは表全体を走査しないためで、大きな表では特に速くなります。

正確な計算を実行する場合、Oracle では、表の走査とソートを行うための十分な領域を必要とします。メモリーに十分な領域がない場合には、一時領域が必要になることがあります。見積りの場合、Oracle では、指定されたサンプル表の行をすべて走査およびソートするための十分な領域を必要とします。索引の場合、計算にそれほど時間も領域も必要ではないので、完全な計算を実行することが最適な方法です。

表、列または索引の統計を生成するとき、分析したオブジェクトの統計がすでにデータ・ディクショナリ内に収録されている場合、Oracle は既存の統計を更新します。そして、分析したオブジェクトにアクセスする解析済みの SQL 文を無効にします。

文が次に実行されるときに、オプティマイザは、新しい統計に基づいて新しい実行計画を自動的に選択します。リモート・データベースに対して発行される分散 SQL 文が分析したオブジェクトにアクセスする場合、その文が次に解析されるときに新しい統計が使用されません。

計算や見積りの指定を行った場合でも、統計の見積りによって節約される時間が取るに足らないものであれば、Oracle は統計を計算します。

統計タイプを列またはドメイン・インデックスと関連付けると、列またはドメイン・インデックスを分析したときに、Oracle によって統計タイプで統計コレクション・メソッドが呼び出されます。

**関連項目：** ユーザー定義の統計の詳細は、『Oracle8i Data Cartridge Developer's Guide』を参照してください。ANALYZE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

次の見出しで説明するように、DBMS\_STATS パッケージで統計を収集します。

## DBMS\_STATS パッケージでの統計収集

DBMS\_STATS にある統計収集プロシージャは次のとおりです。

表 7-1 DBMS\_STATS パッケージの統計収集プロシージャ

プロシージャ	説明
GATHER_INDEX_STATS	索引統計を収集します。
GATHER_TABLE_STATS	表、列、および索引の統計を収集します。
GATHER_SCHEMA_STATS	スキーマ内のすべてのオブジェクトの統計を収集します。
GATHER_DATABASE_STATS	データベース内のすべてのオブジェクトの統計を収集します。

### 索引統計の収集

索引を作成または再作成する際に索引統計を収集するには、ANALYZE SQL 文の COMPUTE STATISTICS 句を使用します。COMPUTE STATISTICS 句を使用しない場合や、大幅な DML 変更をした場合は、GATHER\_INDEX\_STATS プロシージャを使用して索引統計を収集します。GATHER\_INDEX\_STATS プロシージャは、パラレルでは実行されません。このプロシージャを使用すると、次の文を実行した場合と同じことになります。

```
ANALYZE INDEX [ownname.]indname [PARTITION partname] COMPUTE STATISTICS |
ESTIMATE STATISTICS SAMPLE estimate_percent PERCENT
```

**関連項目：** COMPUTE STATISTICS 句の詳細は、『Oracle8i SQL リファレンス』を参照してください。

### 例

この PL/SQL 例では、索引統計を収集します。

```
EXECUTE DBMS_STATS.GATHER_INDEX_STATS(
    'scott','emp_idx');
```

### 構文

GATHER\_INDEX\_STATS プロシージャの構文とパラメータを次に示します。

```
PROCEDURE GATHER_INDEX_STATS(
    ownname VARCHAR2,
    indname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER DEFAULT NULL,
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
```

```
statown VARCHAR2 DEFAULT NULL);
```

ownname	分析する索引のスキーマ。
indname	索引の名前。
partname	パーティションの名前。
estimate_percent	見積もる行の割合 (NULL の場合は " 計算 " します)。有効範囲は (0.000001,100) です。
stattab	新しい統計を収集する前に Oracle が元の統計のバックアップをとる、ユーザー統計表の名前。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。
statid	統計のバックアップをとるための、stattab の第 2 識別子。
statown	ownname パラメータで識別された位置と違う場合に、Oracle が stattab を格納するスキーマ。

### 表統計の収集

表の作成後、次の例に示すように、GATHER\_TABLE\_STATS プロシージャを使用して表、列および索引の統計を収集します。GATHER\_TABLE\_STATS プロシージャでは、できるだけ多くのプロセスでパラレル処理を使用します。

### 例

次の PL/SQL 例では、表統計を収集します。

```
EXECUTE DBMS_STATS.GATHER_TABLE_STATS (
    'scott', 'emp');
```

### 構文

GATHER\_TABLE\_STATS プロシージャの構文とパラメータを次に示します。

```
PROCEDURE GATHER_TABLE_STATS
    ownname VARCHAR2,
    tabname VARCHAR2,
    partname VARCHAR2 DEFAULT NULL,
    estimate_percent NUMBER DEFAULT NULL,
    block_sample BOOLEAN DEFAULT FALSE,
    method_opt VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',
    degree NUMBER DEFAULT NULL,
    granularity VARCHAR2 DEFAULT 'DEFAULT',
    cascade BOOLEAN DEFAULT FALSE,
    stattab VARCHAR2 DEFAULT NULL,
    statid VARCHAR2 DEFAULT NULL,
    statown VARCHAR2 DEFAULT NULL);
```

ownname	分析する表のスキーマ名。
tabname	表の名前。
partname	パーティションの名前。
estimate_percent	見積もる行の割合 (NULL は " 計算 " を意味します)。有効範囲は (0.000001,100) です。
block_sample	この値は、デフォルトのランダム行サンプリングではなくランダム・ブロック・サンプリングを使用するかどうかを示します。ランダム・ブロック・サンプリングの方がより効率的ですが、データがディスク上にランダムに分散されない場合、サンプル値が相関して統計が正確でなくなるおそれがあります。この値が適用されるのは、統計を見積もる場合のみです。BLOCK_SAMPLE の詳細は、『Oracle8i 概要』を参照してください。



method_opt	<p>この値には、次の形式のオプションがあります。</p> <p>(統計を確実にパラレルで収集できるようにするには、句 'SIZE 1' (すなわち、ヒストグラムなし) が必要です)</p> <p>FOR ALL [INDEXED] COLUMNS [SIZE integer]</p> <p>FOR COLUMNS [SIZE integer]column   attribute[column   attribute...]</p> <p>オブティマイザ関連の表統計は、常に収集されます。</p>
degree	<p>この値は、並行度を指定します (NULL の場合は " 表のデフォルト値を使用 " します)。</p>
granularity	<p>この値は、収集する統計の単位を指定します (この値の影響を受けるのはパーティション表のみです)。</p> <p>PARTITION - パーティション・レベルの統計を収集します。</p> <p>GLOBAL - グローバルな統計を収集します。</p> <p>ALL - すべての統計を収集します。</p> <p>SUBPARTITION - サブパーティション・レベルの統計を収集します。</p> <p>DEFAULT - パーティションおよびグローバルな統計を収集します。</p>
cascade	<p>このパラメータを TRUE に設定すると、この表の索引について統計が収集されます。索引統計は、パラレルでは収集されません。このオプションを使用すると、それぞれの表の索引で GATHER_INDEX_STATS プロシージャを実行した場合と同じことになります。索引統計の最新の収集は、文構文 'CREATE INDEX... COMPUTE STATISTICS' を使用して行われたと想定しているので、デフォルト値は FALSE です。</p>
stattab	<p>新しい統計を収集する前に Oracle が元の統計のバックアップを入れる、ユーザー統計表の名前。パラメータ stattab、statid、および statown の詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。</p>
statid	<p>統計のバックアップをとるための、stattab の第 2 識別子。</p>
statown	<p>ownname パラメータで識別された位置と違う場合に、Oracle が stattab を格納するスキーマ。</p>

## スキーマ統計の収集

次の例に示すように、GATHER\_SCHEMA\_STATS プロシージャを実行して、スキーマ内の全オブジェクトの統計を収集します。

### 例

この PL/SQL 例では、スキーマ統計を収集します。

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('scott');
```

### 構文

GATHER\_SCHEMA\_STATS プロシージャの構文とパラメータを次に示します。Oracle は、次に示す全パラメータの表すべてに値を渡します。

```
PROCEDURE GATHER_SCHEMA_STATS(  
    ownname VARCHAR2,  
    estimate_percent NUMBER DEFAULT NULL,  
    block_sample BOOLEAN DEFAULT FALSE,  
    method_opt VARCHAR2 DEFAULT 'FOR ALL COLUMNS SIZE 1',  
    degree NUMBER DEFAULT NULL, INSTANCES NUMBER DEFAULT NULL,  
    granularity VARCHAR2 DEFAULT 'ALL',  
    cascade BOOLEAN DEFAULT FALSE,  
    stattab VARCHAR2 DEFAULT NULL,  
    statid VARCHAR2 DEFAULT NULL,  
    options VARCHAR2 DEFAULT 'GATHER',  
    objlist OUT OBJECTTAB,  
    statown VARCHAR2 DEFAULT NULL);
```

ownname	分析するスキーマの名前 ( NULL の場合は " 現ユーザーのスキーマ " になります )。
estimate_percent	見積もる行の割合 ( NULL の場合は " 計算 " します )。有効範囲は (0.000001,100) です。
block_sample	デフォルトのランダム行サンプリングではなくランダム・ブロック・サンプリングを使用するかどうかを示します。ランダム・ブロック・サンプリングの方がより効率的ですが、データがディスク上にランダムに分散されない場合、サンプル値が相関して統計が正確でなくなるおそれがあります。この値が適用されるのは、統計を見積もる場合のみです。
method_opt	次の形式の方法オプションを示します ( 統計を確実にパラレルで収集できるようにするには、句 'SIZE 1' ( すなわち、ヒストグラムなし ) が必要です )。  FOR ALL [INDEXED] COLUMNS [SIZE integer]

degree	並行度を示します ( NULL の場合は " 表のデフォルト値を使用 " します )。
granularity	<p>収集する統計の単位を示します。これが適用されるのは、パーティション表のみです。</p> <p>PARTITION - パーティション・レベルの統計を収集します。</p> <p>GLOBAL - グローバルな統計を収集します。</p> <p>ALL - すべての統計を収集します。</p> <p>SUBPARTITION - サブパーティション・レベルの統計を収集します。</p> <p>DEFAULT - パーティションおよびグローバルな統計を収集します。</p>
cascade	索引についても統計が収集されることを示します。索引統計収集は、並列化されません。このオプションを使用すると、スキーマ内のそれぞれの索引で GATHER_INDEX_STATS プロシージャを実行し、さらに表と列の統計を収集した場合と同じことになります。
stattab	新しい統計を収集する前に Oracle が元の統計のバックアップをとる、ユーザー統計表の名前。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。
statid	統計のバックアップをとるための、stattab の第 2 識別子。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。
options	7-17 ページの次の項 <a href="#">自動統計収集</a> を参照してください。
objlist	7-17 ページの次の項 <a href="#">自動統計収集</a> を参照してください。
statown	ownname パラメータで識別された位置と違う場合に、Oracle が stattab を格納するスキーマ。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。

## データベース統計の収集

データベースの作成後、GATHER\_DATABASE\_STATS プロシージャを実行してデータベース統計を収集します。

### 例

この PL/SQL 例では、データベース統計を収集します。

```
EXECUTE DBMS_STATS.GATHER_DATABASE_STATS;
```

### 構文

GATHER\_DATABASE\_STATS プロシージャの構文とパラメータを次に示します。

```
PROCEDURE GATHER_DATABASE_STATS(  
  estimate_percent NUMBER DEFAULT NULL,  
  block_sample BOOLEAN DEFAULT FALSE,  
  method_opt VARCHAR2 DEFAULT `FOR ALL COLUMNS SIZE 1`,  
  degree NUMBER DEFAULT NULL,  
  granularity VARCHAR2 DEFAULT,  
  cascade BOOLEAN DEFAULT FALSE,  
  stattab VARCHAR2 DEFAULT NULL,  
  statid VARCHAR2 DEFAULT NULL,  
  options VARCHAR2 DEFAULT 'GATHER',  
  objlist OUT OBJECTTAB,  
  statown VARCHAR2 DEFAULT NULL);
```

estimate_percent	見積もる行の割合を示します ( NULL の場合は " 計算 " します )。有効範囲は (0.000001,100) です。
block_sample	デフォルトのランダム行サンプリングではなくランダム・ブロック・サンプリングを使用するかどうかを示します。ランダム・ブロック・サンプリングの方がより効率的ですが、データがディスク上にランダムに分散されない場合、サンプル値が相関して統計が正確でなくなるおそれがあります。この値が適用されるのは、統計を見積もる場合のみです。
method_opt	次の形式の方法オプションを示します ( 統計を確実にパラレルで収集できるようにするには、句 'SIZE 1' ( すなわち、ヒストグラムなし ) が必要です )。  FOR ALL [INDEXED] COLUMNS [SIZE integer]
degree	並行度を示します ( NULL の場合は " 表のデフォルト値を使用 " します )。

granularity	<p>収集する統計の単位を示します（これが適用されるのはパーティション表のみです）。</p> <p>PARTITION - パーティション・レベルの統計を収集します。</p> <p>GLOBAL - グローバルな統計を収集します。</p> <p>ALL - すべての統計を収集します。</p> <p>SUBPARTITION - サブパーティション・レベルの統計を収集します。</p> <p>DEFAULT - パーティションおよびグローバルな統計を収集します。</p>
cascade	<p>この値を TRUE に設定すると、索引についても統計が収集されます。索引統計収集は、並列化されません。このオプションを使用すると、データベース内のすべての索引で GATHER_INDEX_STATS プロシージャを実行し、さらに表と列の統計を収集した場合と同じことになります。</p>
stattab	<p>新しい統計を収集する前に Oracle が元の統計のバックアップを入れる、ユーザー統計表の名前。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。</p>
statid	<p>統計のバックアップをとるための、stattab の第 2 識別子。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。</p>
options	<p>7-17 ページの次の項「<a href="#">自動統計収集</a>」を参照してください。</p>
objlist	<p>7-17 ページの次の項「<a href="#">自動統計収集</a>」を参照してください。</p>
statown	<p>ownname パラメータで識別された位置と違う場合に、Oracle が stattab を格納するスキーマ。詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。</p>

## 新しいオプティマイザ統計の収集

特定のスキーマに対する新しいオプティマイザの統計収集には、次のプロシージャをお勧めします。

新しい統計を収集する前に、DBMS\_STATS.EXPORT\_SCHEMA\_STATS を使用して既存の統計を抽出および保存します。次に DBMS\_STATS.GATHER\_SCHEMA\_STATS を使用して新しい統計を収集します。GATHER\_SCHEMA\_STATS プロシージャの 1 回のコールで、この両方をインプリメントすることもできます。

主要な SQL 文でパフォーマンスが著しく低下した場合は、大きめのサンプル・サイズを使用してもう一度統計を収集するか、または次のステップを実行します。

1. DBMS\_STATS.EXPORT\_SCHEMA\_STATS プロシージャを使用して新しい統計を保存します。
2. DBMS\_STATS.IMPORT\_SCHEMA\_STATS を使用して古い統計を復元します。これで、アプリケーションを再度実行する準備ができました。

新しい統計を使用した場合に大半の SQL 文のパフォーマンスが向上し、問題のある SQL 文が少数であれば、新しい統計を使用できます。その場合は、次のようにします。

1. 古い統計を使用して問題のある各 SQL 文のストアド・アウトラインを作成します。

---

---

**注意：** ストアド・アウトラインは、事前コンパイルされた実行計画であり、Oracle はこれを使用して検証済みのアプリケーションのパフォーマンス特性を模倣します。

---

---

2. DBMS\_STATS.IMPORT\_SCHEMA\_STATS を使用して新しい統計を復元します。
3. これで、アプリケーションを新しい統計で実行する準備ができました。ただし、問題のある SQL 文については前のパフォーマンス・レベルが引き続き達成されます。

**関連項目：** ストアド・アウトラインの詳細は、7-25 ページの「[アウトラインの作成](#)」を参照してください。

## 自動統計収集

この機能を使用すると、統計を自動的に収集できます。また、統計が古くなった表のリストや統計のない表のリストの作成にも使用できます。

この機能は、GATHER\_SCHEMA\_STATS プロシージャと GATHER\_DATABASE\_STATS プロシージャで *options* パラメータと *objlist* パラメータを指定して実行します。*options* パラメータには次の値を使用します。

- GATHER STALE - 統計が古くなった表で統計を収集します。
- GATHER - (デフォルト) すべての表で統計を収集します。
- GATHER EMPTY - 統計のない表だけで統計を収集します。
- LIST STALE - 統計が古くなった表のリストを作成します。
- LIST EMPTY - 統計のない表のリストを作成します。

*objlist* パラメータは、LIST STALE オプションと LIST EMPTY オプションのアウトプット・パラメータを示します。*objlist* パラメータは、タイプ DBMS\_STATS.OBJECTTAB に属します。

### 自動統計収集を使用可能にする方法

GATHER STALE オプションを使用すると、統計が古く、また MONITORING 属性が使用可能になっている表の統計だけが収集されます。表の監視を使用可能にするには、この章の 7-18 ページにある「[監視と自動統計収集のために表を指定する方法](#)」で説明するように、CREATE TABLE 文と ALTER TABLE 文の MONITORING キーワードを使用します。

GATHER STALE オプションを使用すると、コストベース・オブティマイザの統計が最新の状態に保たれます。また、このオプションを定期的に使用すると、一度にすべての表で ANALYZE 文を使用した場合に起こるオーバーヘッドを避けることもできます。GATHER オプションを使用すると、GATHER STALE よりも多くの表について統計が収集されるので、オーバーヘッドが著しく大きくなるおそれがあります。

アプリケーションに合った統計収集の頻度を設定するには、GATHER\_SCHEMA\_STATS プロシージャと GATHER\_DATABASE\_STATS プロシージャのスクリプトまたはジョブのスケジューリング・ツールを使用します。収集の頻度によって、統計収集プロセスで起こるオーバーヘッドの処理に対し、オブティマイザの正確な統計を出すタスクのバランスをとります。

### 統計が古くなった表または統計のない表のリストの作成方法

統計が古くなった表のリストを作成するには、GATHER\_SCHEMA\_STATS プロシージャと GATHER\_DATABASE\_STATS プロシージャを使用します。これらのリストを使用して、手動で統計を収集する表を指定します。

これらのプロシージャは、統計のない表のリスト作成にも使用できます。これらのリストを使用して、自動または手動のどちらかで統計を収集する表を指定します。

## 監視と自動統計収集のために表を指定する方法

特定の表の統計を自動的に収集するには、MONITORING キーワードを使用して監視属性を使用可能にします。このキーワードは、CREATE TABLE および ALTER TABLE 文構文に属します。

使用可能にすると、Oracle によって表の DML アクティビティが監視されます。これには、最後に統計が収集されてから行われたその表の挿入、更新および削除の概数が含まれます。Oracle はこのデータを使用して、統計が古くなった表を識別します。

USER\_TAB\_MODIFICATIONS ビューを照会して、Oracle がこれらの表の監視から取得したデータを参照します。

---

---

**注意：** Oracle がこのビューに情報を伝える間、数時間の遅れが出る場合もあります。

---

---

表の監視を使用禁止にするには、NOMONITORING キーワードを使用します。

**関連項目：** CREATE TABLE 構文および ALTER TABLE 構文、MONITORING キーワードおよび NOMONITORING キーワードの詳細は、『Oracle8i SQL リファレンス』を参照してください。

## 複数バージョンの統計の保存方法

*stattab* パラメータ、*statid* パラメータおよび *statown* パラメータを指定すると、表の複数バージョンの統計を保存できます。前のバージョンの統計をアーカイブするために宛先の表を識別するには、*stattab* を使用します。また、たとえばそのバージョンが作成された日付と時間を示すには、*statid* を使用してこれらのバージョンを識別します。宛先のスキーマが実際の表のスキーマと異なる場合は、*statown* を使用して宛先のスキーマを識別します。

**関連項目：** パラメータ *stattab*、*statid* および *statown* については、この章の前の方にある GATHER\_SCHEMA\_STATS および GATHER\_DATABASE\_STATS プロシージャのパラメータ説明リストで説明しています。



## コストベースの最適化計画に影響するパラメータ

次のパラメータは、コストベースの最適化計画に影響を与えます。

OPTIMIZER_FEATURES_ENABLED	B_TREE_BITMAP_PLANS や FAST_FULL_SCAN_ENABLED などいくつかのオプティマイザ機能をオンにします。
OPTIMIZER_MODE	初期化パラメータとして、インスタンス起動時にオプティマイザ・モード（ルールベースの最適化、スループットに関するコストベースの最適化、応答時間に関するコストベースの最適化、または統計の存在に基づく選択）を設定します。セッション中に値を動的に変更するには、ALTER SESSION 文の OPTIMIZER_MODE パラメータを設定します。
OPTIMIZER_PERCENT_PARALLEL	オプティマイザがコスト・ファンクションで使用する並列度を定義します。
HASH_AREA_SIZE	値が大きいほどハッシュ結合のコストが低くなり、より多くのハッシュ結合が行われます。
SORT_AREA_SIZE	値が大きいほどソートのコストが低くなり、より多くのソート / マージ結合が行われます。
DB_FILE_MULTIBLOCK_READ_COUNT	値が大きいほど表走査のコストが低くなり、索引よりも表走査が優先されます。

データ・ウェアハウス・アプリケーションでは、次のパラメータの設定が必要になります。

ALWAYS_ANTI_JOIN	Oracle が使用する逆結合タイプ ( NESTED_LOOPS/MERGE/HASH ) を設定します。
HASH_JOIN_ENABLED	ハッシュ結合機能の使用可能と使用不可を切り替えます。 データ・ウェアハウス・アプリケーションでは必ず TRUE に設定してください。

次のパラメータは、変更する必要はほとんどありません。

HASH_MULTIBLOCK_IO_COUNT	値が大きいほどハッシュ結合のコストが低くなり、より多くのハッシュ結合が行われます。
OPTIMIZER_SEARCH_LIMIT	すべての可能な結合の順列が考慮される、FROM 句内の表の最大数。
BITMAP_MERGE_AREA_SIZE	範囲述語に一致する異なるビットマップをマージするために使用する領域のサイズ。サイズを大きくすると、範囲述語に対するビットマップ索引が優先されます。

**関連項目：** 各パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## オブティマイザの索引使用方法に影響するパラメータ

OLTP アプリケーションと DSS アプリケーションの両方において、特にネスト・ループ・ジョイン文をはじめとする広範な文について、オブティマイザの索引使用を扱うパラメータが 2 つあります。

- OPTIMIZER\_INDEX\_COST\_ADJ
- OPTIMIZER\_INDEX\_CACHING

### OPTIMIZER\_INDEX\_COST\_ADJ の使用

索引の使用を拡張するには、OPTIMIZER\_INDEX\_COST\_ADJ パラメータを使用します。このパラメータは、索引の選択性に関係なく、すべての索引の使用を促進します。また、これは、ネスト・ループ・ジョイン・プロープの索引キャッシングの単なるモデル化ではなく、索引使用一般に適用されます。

### OPTIMIZER\_INDEX\_CACHING の使用

次の 2 つの条件が重なった場合は、OPTIMIZER\_INDEX\_CACHING を使用します。

- ネスト・ループ・ジョインに使用される索引が、環境内で頻繁にキャッシュされる場合
- オブティマイザがネスト・ループ・ジョインを使用する程度が不十分な場合

こうした環境でこのパラメータを使用すると、OPTIMIZER\_INDEX\_COST\_ADJ を使用した場合に比べて 2 つの利点があります。まず、OPTIMIZER\_INDEX\_CACHING では、選択性の高い索引が優先的に使用されます。つまり、このパラメータで比較的低い値を使用すると、オブティマイザが非リーフ索引ブロックすべてのキャッシュを効率よくモデル化します。その場合、オブティマイザはこの索引の使用コストを主に選択性に基づいて計算しま

す。したがって、OPTIMIZER\_INDEX\_CACHING を低い値に設定すると、選択性が低く望ましくない索引を使いすぎずに、索引のキャッシングを希望どおりモデル化できます。

次に、OPTIMIZER\_INDEX\_CACHING を使用した場合、その影響はネスト・ループ・ジョイン・プローブのキャッシュされた索引使用のモデル化に限定されます。したがって、副次作用が少なく済みます。

## コストベース・アプローチの使用上のヒント

この項では、コストベース・アプローチの使用について追加情報を示します。

### 大きなキャッシュを持つシステムの間合せ計画

コストベースの最適化は、バッファ・キャッシュのヒット率が非常に低いマルチユーザー・システム上で間合せが実行されることを前提としています。したがって、大きなバッファ・キャッシュを持つシングル・ユーザー・システムでは、コストベース・オブティマイザによって選択される計画が、最良の計画とは限らない場合があります。また、大きなキャッシュを持つシングル・ユーザー・システム上で間合せ計画の応答時間を計測しても、同じ間合せを通信量の多いマルチユーザー・システム上で実行するときのパフォーマンスの予測としては適切な手段でない場合があります。

### 表の前の索引の分析

表の分析は、索引の分析よりも多くのシステム・リソースを使います。表の索引を個別に分析するか、サンプリング率を高くして索引の作成中に統計を収集する方がよい場合もあります。

### 統計の生成（可能であれば随時）

アクセス・パスと結合方法のヒントを使うと、コストベースの最適化が起動されます。コストベースの最適化は統計に依存するので、システムのデフォルトとしてルールベースの最適化が選択されている場合でも、ヒントを持つ間合せで参照されているすべての表の統計を収集することが重要です。

### ユーザー定義の構造による統計の収集、選択性およびコストのファンクションの指定

列、スタンドアロン・ファンクション、タイプ、パッケージ、索引、索引タイプなどのユーザー定義の構造は、通常、オブティマイザには "不透明" です。つまり、オブティマイザはこれらの構造に関する統計を持たないので、それらを使用する間合せで正確な選択性やコストを計算することはできません。

そのため、ユーザー定義の構造で統計の収集、選択性、およびコストのファンクションを指定することをお勧めします。これは、オブティマイザのデフォルト値が不正確なために実行計画の費用が高くなる場合もあるからです。

## ルールベースの最適化

ルールベースの最適化もサポートされていますが、新規のアプリケーションはコストベースの最適化を使用して作成することをお勧めします。コストベース・オブティマイザは DSS の新しい拡張機能をサポートしているので、データ・ウェアハウス・アプリケーションにはコストベースの最適化を使用してください。ハッシュ結合、改良されたスター問合せ処理、ヒストグラムなど、最新のパフォーマンス拡張大部分は、コストベースの最適化を介してのみ使用可能です。

Oracle バージョン 6 を使用して OLTP アプリケーションを開発し、オブティマイザのルールに基づいて慎重に SQL 文をチューニングした場合は、これらのアプリケーションを新しいバージョンの Oracle にアップグレードするときに、ルールベースの最適化を引き続き使ってもよいでしょう。

統計の収集も SQL 文へのヒントの追加も行わない場合、その文はルールベースの最適化を使用します。ただし、Oracle サーバーではルールベースのアプローチは利用できないため、最終的にはコストベースのアプローチを使用するように既存のアプリケーションを移行してください。

サード・パーティ・ベンダーが提供しているアプリケーションを使用している場合は、そのアプリケーションにはどのタイプの最適化が最も適しているかを判断するために、ベンダーに問い合せてください。

単純に統計を収集することによって、試験的にコストベースの最適化を使用可能にできます。その後、これらの統計を削除するか、OPTIMIZER\_MODE 初期化パラメータの値または ALTER SESSION 文の OPTIMIZER\_MODE オプションに RULE を設定することで、ルールベースの最適化に戻すことができます。また、コストベースのアプローチを使用せずに、データに対する統計情報を収集および検査する場合に、この値を使うこともできます。

**関連項目：** ルールベース・オブティマイザからコストベース・オブティマイザへ移行するプロシージャについては、7-29 ページの「[コストベース・オブティマイザのプラン・スタビリティ・プロシージャ](#)」を参照してください。統計の収集方法については、7-8 ページの「[DBMS\\_STATS パッケージでの統計収集](#)」を参照してください。

## 実行計画を保存するプラン・スタビリティの使用

プラン・スタビリティを使用すると、データベース環境を変更した場合にアプリケーションのパフォーマンス特性に影響が及ぶのを防ぐことができます。こうした変更には、オブティマイザ・モード設定の変更や、SORT\_AREA\_SIZE や BITMAP\_MERGE\_AREA\_SIZE などメモリー構造のサイズに影響するパラメータの変更が含まれます。プラン・スタビリティは、アプリケーションでパフォーマンス変更のリスクを冒すことができない場合に特に役立ちます。

プラン・スタビリティは、実行計画を "ストアド・アウトライン" に保存します。Oracle では、1 つまたはすべての SQL 文についてストアド・アウトラインを作成できます。ストア

ド・アウトラインを使用可能にすると、オブティマイザによってアウトラインから等価の実行計画が生成されます。

Oracle がストアド・アウトラインに保持する計画は、システム構成または統計の変更に問わず一貫しています。また、ストアド・アウトラインを使用すると、以降の Oracle リリースでオブティマイザが変更されても、生成した実行プランの安定性は保たれます。アウトラインをカテゴリにまとめ、Oracle がアウトラインの管理と配置を単純化するために使用するアウトラインのカテゴリを指定することもできます。

プラン・スタビリティは、Oracle の新しいバージョンへアップグレードする際に、ルールベース・オブティマイザからコストベース・オブティマイザへ移行するときにも役立ちます。

---

**注意：** 大規模分散型のアプリケーションを開発する場合は、ストアド・アウトラインを使用すると、すべての顧客が確実に同じ実行計画にアクセスするようにできます。詳細は、7-17 ページの「[自動統計収集](#)」を参照してください。

---

## プラン・スタビリティでのヒントおよび完全テキスト一致の使用

Oracle ではヒントを使用してストアド計画を記録するので、プラン・スタビリティが実行計画を制御する程度は、Oracle のヒント・メカニズムがアクセス・パスを制御する範囲によって決まります。また、プラン・スタビリティは、問合せにストアド・アウトラインがあるかどうかを判別する際に、問合せの "完全テキスト一致" に依存します。

似通った SQL 文はストアド・アウトラインを共有できる可能性があります。SQL テキストとそのストアド・アウトラインは 1 対 1 で対応しています。異なるリテラルを述語に指定すると、異なるアウトラインが適用されます。これを避けるには、アプリケーションのリテラルをバインド変数に置き換えてください。それにより、SQL 文のテキストが完全に一致し、アウトラインを共有できるようになります。

**関連項目：** Oracle による SQL 文とアウトラインの突合せの方法については、7-24 ページの「[SQL 文とアウトラインの突合せ](#)」を参照してください。

プラン・スタビリティは、パフォーマンスに問題がない場合、実行計画の保存に依存します。しかし、多くの環境では、"dates" や "order numbers" などのデータタイプの属性はすぐ変わる可能性があります。そのような場合に実行計画を永続的に使用すると、データ特性が変更されたときにパフォーマンスが低下することがあります。

つまり、動的な環境では、計画の保存に依存する技法はコストベースの最適化の使用目的に反することになります。コストベースの最適化では、データの状態を正確に反映した統計に基づいて実行計画の生成が試みられます。したがって、プラン・スタビリティを制御する必要性と、データ特性の変更に対するオブティマイザの適能力から得られる利益とを評価する必要があります。

## アウトラインでのヒントの使用方法

アウトラインは主に、特定の SQL 文の実行計画生成に対するオプティマイザの結果に相当するヒントのセットからなります。Oracle によってアウトラインが作成されると、プラン・スタビリティは実行計画の生成に使用したのと同じデータを使用して最適化の結果を調べます。つまり、Oracle は実行計画への入力を使用して、実行計画そのものではなくアウトラインを生成します。

---

**注意：** アウトラインは変更できません。SQL 文にヒントを埋め込むことはできますが、Oracle はヒントで修正された SQL 文をアウトラインに保管されている元の SQL 文とは別のものと見なすので、このヒントの埋込みは Oracle によるアウトラインの使用方法には影響しません。

---

## SQL 文とアウトラインの突合せ

Oracle は、SQL 文をコンパイルしてアウトラインとマッチングする際に、2 つのシナリオのいずれかを使用します。最初のシナリオでは、システム / セッションのパラメータ `USE_STORED_OUTLINES` を `FALSE` に設定してアウトラインを使用禁止にした場合、Oracle は SQL テキストとアウトラインのマッチングを試みません。2 番目のシナリオでは、次の 2 つの " マッチング " ステップがあります。

まず、Oracle が特定のアウトライン・カテゴリを使用するよう指定した場合は、そのカテゴリ内のアウトラインだけがマッチングの候補になります。次に、入ってくる文の SQL テキストがそのカテゴリのアウトラインの SQL テキストと完全に一致する場合、Oracle は両方のテキストを同一のものとみなし、アウトラインを使用します。少しでも違いがあれば、不一致とみなされます。

この違いには、スペースの違い、キャリッジ・リターンの違い、埋込みヒント、またはコメント・テキストの違いも含まれます。これらのルールは、カーソルのマッチングのルールと同じです。

## Oracle によるアウトラインの格納方法

Oracle は、アウトライン・データを `OLS` 表に、ヒント・データを `OLSHINTS` 表に格納します。データを削除しない限り、アウトラインはいつまでも保持されつづけます。Oracle は実行計画をキャッシュに保持し、それらが無効になった場合やキャッシュの大きさが不十分でそれらすべてを保持できない場合のみ再作成します。

アウトラインが実行計画のキャッシングに及ぼす影響は、計画がキャッシュ内にあるどうかを示すためにアウトラインのカテゴリ名が SQL テキストとともに使用されることのみです。これにより、Oracle があるカテゴリでコンパイルされた実行計画を使って、別のカテゴリでコンパイルすべき SQL 文を実行するような事態を防ぐことができます。

## プラン・スタビリティを使用可能にするパラメータ設定

アウトラインを適切に機能させるために、接尾辞 "\_ENABLED" で終わるパラメータをはじめとするいくつかのパラメータの設定は、実行環境全体にわたって一貫したものでなければなりません。該当するパラメータは次のとおりです。

- QUERY\_REWRITE\_ENABLED
- STAR\_TRANSFORMATION\_ENABLED
- OPTIMIZER\_FEATURES\_ENABLE

## アウトラインの作成

すべての SQL 文に対して自動的にアウトラインを作成することもできますし、特定の SQL 文に対して自分でアウトラインを作成することもできます。どちらの場合も、アウトラインはルールベースまたはコストベースのオプティマイザから入力データを導出します。

パラメータ CREATE\_STORED\_OUTLINES を TRUE に設定すると、ストアド・アウトラインが自動的に作成されます。Oracle は、アクティブにされることにより、実行された SQL 文すべてに対してアウトラインを作成します。CREATE OUTLINE 文を使用して、特定の文に対するストアド・アウトラインを作成することもできます。

**関連項目：** CREATE OUTLINE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。ルールベース・オプティマイザからコストベース・オプティマイザへの移行については、7-29 ページの「[コストベース・オプティマイザへの移行に対するアウトラインの使用](#)」を参照してください。

## カテゴリの作成とストアド・アウトラインへの割当て

アウトライン・カテゴリ名を作成して、アウトラインを割り当てることができます。前述のように、これによりカテゴリ内のすべてのアウトラインを一度に操作できるので、アウトラインの管理が簡単になります。

CREATE\_STORED\_OUTLINES パラメータも CREATE OUTLINE 文もカテゴリ名を受け入れます。どちらの場合も、カテゴリ名を指定すると、それ以降に作成したアウトラインがすべて Oracle によってそのカテゴリに割り当てられます。これは、カテゴリ名を再設定するか CREATE\_STORED\_OUTLINES パラメータを FALSE に設定してアウトラインの生成を中断するまで行われます。

CREATE\_STORED\_OUTLINES を TRUE に設定するか、カテゴリ名なしで CREATE OUTLINE 文を使用すると、アウトラインはカテゴリ名 DEFAULT に割り当てられます。

**関連項目：** CREATE\_- パラメータおよび USE\_STORED\_OUTLINES パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## ストアド・アウトラインの使用

Oracle が SQL 文をコンパイルする際にストアド・アウトラインを使用するには、システム・パラメータ `USE_STORED_OUTLINES` を `TRUE` またはカテゴリ名に設定します。`USE_STORED_OUTLINES` を `TRUE` に設定すると、Oracle は `DEFAULT` カテゴリでアウトラインを使用します。`USE_STORED_OUTLINES` パラメータでカテゴリを指定すると、`USE_STORED_OUTLINES` パラメータを別のカテゴリに再設定するか、`USE_STORED_OUTLINES` を `FALSE` に設定してアウトラインの使用を中断するまで、Oracle はそのカテゴリでアウトラインを使用します。カテゴリ名を指定したがそのカテゴリ内に SQL 文と一致するアウトラインが見つからない場合、Oracle は `DEFAULT` カテゴリでアウトラインを検索します。

指定されたアウトラインは、アウトラインを持つ SQL 文のコンパイルだけを制御します。`USE_STORED_OUTLINES` を `FALSE` に設定すると、Oracle はアウトラインを使用しません。`USE_STORED_OUTLINES` を `FALSE` に設定し、`CREATE_STORED_OUTLINES` を `TRUE` に設定した場合、Oracle はアウトラインを作成しますが、使用はしません。

ストアド・アウトラインの使用をアクティブにした場合、Oracle は常にコストベースのオプティマイザを使用します。これは、アウトラインがヒントに依存し、そのヒントのほとんどが効率性のためコストベース・オプティマイザを必要とするからです。

## アウトライン・データの照会

次のビューから、データ・ディクショナリに格納されているアウトラインおよびそれに関連するヒント・データの情報にアクセスできます。

- `USER_OUTLINES`
- `USER_OUTLINE_HINTS`

たとえば、アウトライン・カテゴリが `'MYCAT'` である `USER_OUTLINES` ビューからアウトライン情報を獲得するには、次の構文を使用します。

```
SELECT NAME,SQL_TEXT FROM USER_OUTLINES WHERE CATEGORY='mycat';
```

すると、カテゴリ `"MYCAT"` 内の全アウトラインの名前とテキストが表示されます。アウトライン `"NAME1"` について生成されているヒントをすべて参照するには、次の構文を使用します。

```
SELECT HINT FROM USER_OUTLINE_HINTS WHERE NAME='name1';
```

**関連項目：** 必要に応じてアウトライン表をある表領域から別の表領域に移動するプロシージャについては、7-28 ページの「[ある表領域から別の表領域へアウトライン表を移動するプロシージャ](#)」を参照してください。



# OUTLN\_PKG パッケージでのストアド・アウトラインの管理

ストアド・アウトラインとそのアウトライン・カテゴリを管理するには、OUTLN\_PKG パッケージのプロシージャを使用します。OUTLN\_PKG には、次のプロシージャがあります。

表 7-2 OUTLN\_PKG のアウトライン管理プロシージャ

プロシージャ	説明
DROP_UNUSED	作成されてから Oracle によって使用されていないアウトラインを削除します。
DROP_BY_CAT	指定したカテゴリ名に割り当てられているアウトラインを削除します。
UPDATE_BY_CAT	あるカテゴリから別のカテゴリにアウトラインを再割当てします。

## 使用されていないアウトラインの削除

必要ないアウトラインは OUTLN\_PKG の DROP\_UNUSED プロシージャを使用して削除できます。Oracle が使用しないアウトラインが多すぎて表領域がいっぱいになっている場合は、このプロシージャを使用するとパフォーマンスが改善されます。

## 構文

DROP\_UNUSED プロシージャの構文は次のとおりです。

```
OUTLN_PKG.DROP_UNUSED;
```

## カテゴリ内のアウトラインの削除

特定のカテゴリ内のアウトラインを削除するには、DROP\_BY\_CAT プロシージャを実行します。

## 構文

DROP\_BY\_CAT プロシージャの構文とパラメータは次のとおりです。

```
OUTLN_PKG.DROP_BY_CAT(  
    category_name);
```

category\_name            削除するカテゴリの名前。

## 別のカテゴリへのアウトラインの再割当て

あるカテゴリから別のカテゴリにアウトラインを再割当てするには、UPDATE\_BY\_CAT プロシージャを使用します。

### 構文

UPDATE\_BY\_CAT プロシージャの構文とパラメータは次のとおりです。

```
OUTLN_PKG.UPDATE_BY_CAT(  
    old_category_name,  
    new_category_name);
```

old_category_name	新しいカテゴリに再割当てするアウトライン・カテゴリの名前を指定します。
new_category_name	アウトラインの割当て先となるカテゴリの名前を指定します。

---

**注意：** 特定のアウトラインを操作するには、DDL 文 CREATE、DROP および ALTER を使用します。たとえば、アウトラインの改名やそのカテゴリの変更には ALTER OUTLINE 文を使用します。

---

**関連項目：** CREATE、DROP および ALTER 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## アウトライン表の移動

Oracle は、USER\_OUTLINES ビューと USER\_OUTLINE\_HINTS ビューを、それぞれ OLS 表と OLSHINTS 表のデータに基づいて作成します。これらの表は、"OUTLN" と呼ばれるスキーマを使用して SYS 表領域で作成されます。アウトラインが SYS 表領域で領域を使いすぎている場合は、アウトラインを移動できます。そのためには、次のプロシージャを使用し、別の表領域を作成してそこにアウトライン表を移動します。

### ある表領域から別の表領域へアウトライン表を移動するプロシージャ

アウトライン表を移動するには、次のプロシージャを使用します。

1. 次の構文で OLS 表と OLSHINTS 表をエクスポートします。

```
EXP OUTLN/OUTLN FILE = exp_file TABLES = 'OL$' 'OL$HINTS' SILENT=y
```

2. 次の構文で前の OLS 表と OLSHINTS 表を削除します。

```
CONNECT OUTLN/outln_password;  
DROP TABLE OL$;  
CONNECT OUTLN/outln_password;  
DROP TABLE OL$HINTS;
```

3. 次の構文でこれらの表の新しい表領域を作成します。

```
CREATE TABLESPACE outln_ts  
DATAFILE 'tspace.dat' SIZE 2MB  
DEFAULT STORAGE (INITIAL 10KB NEXT 20KB  
MINEXTENTS 1 MAXEXTENTS 999 PCTINCREASE 10) ONLINE;
```

4. 次の構文で OLS 表と OLSHINTS 表をインポートします。

```
IMPORT OUTLN/outln_password  
FILE=exp_file TABLES = 'OLS' 'OLSHINTS' IGNORE=y SILENT=y
```

IMPORT 文によって、OUTLN というスキーマに OLS 表と OLSHINTS 表が再作成されますが、このスキーマは "OUTLN\_TS" という新しい表領域にあります。

## コストベース・オブティマイザのプラン・スタビリティ・プロシージャ

この項では、コストベース・オブティマイザの機能を利用してパフォーマンスを大幅に改善するプロシージャを説明します。プラン・スタビリティでは、システムが目標としている実行計画でパフォーマンスに問題がないものは保存し、一方、残りの SQL 文では新しいコストベース・オブティマイザ機能を利用する手段を提供します。

この項で説明するトピックは次のとおりです。

- [コストベース・オブティマイザへの移行に対するアウトラインの使用](#)
- [RDBMS アップグレードとコストベース・オブティマイザ](#)

## コストベース・オブティマイザへの移行に対するアウトラインの使用

ルールベース・オブティマイザを使用して開発したアプリケーションの場合、パフォーマンスを最適化するため SQL 文の手動チューニングに多大な作業量を投入していることがあります。プラン・スタビリティを使用すると、ルールベースの最適化からコストベースの最適化へアップグレードする際にアプリケーションの動作を保存することによって、すでにパフォーマンス・チューニングに投入した作業を生かすことができます。コストベースの最適化に切り替える前にアプリケーションのアウトラインを作成すると、ルールベース・オブティマイザで生成した計画を使用しながら、切替え後に新しく作成されたアプリケーションで生成した文で通常のコストベース計画を使用できます。アプリケーションのアウトラインを作成および使用するには、次の手順を実行します。

---

---

**注意：** この手順をよく読んで、意味を十分理解してから実行してください。

---

---

1. 次のような構文を実行してアウトライン・カテゴリを指定します。ここでは、例として RBOCAT アウトライン・カテゴリを指定します。

```
ALTER SESSION SET CREATE_STORED_OUTLINES = rbocat;
```

2. 重要な SQL 文すべてにストアド・アウトラインを獲得できるよう十分に時間をとってアプリケーションを実行します。
3. 次の構文でアウトラインの生成を中断します。

```
ALTER SESSION SET CREATE_STORED_OUTLINES = FALSE;
```

4. DBMS\_STATS パッケージで統計を収集します。
5. パラメータ OPTIMIZER\_MODE を CHOOSE に変更します。
6. 次の構文を入力して、Oracle がカテゴリ RBOCAT のアウトラインを使用するようにします。

```
ALTER SESSION SET USE_STORED_OUTLINES = rbocat;
```

7. アプリケーションを実行します。

プラン・スタビリティの制約上の理由から、このアプリケーションの SQL 文のアクセス・パスは変更しないようにしてください。

---

---

**注意：** ステップ 2 で問合せが実行されなかった場合は、コストベースの最適化に切り替えた後も、古い動作の問合せを獲得する可能性があります。その場合は、オブティマイザ・モードを RULE に変更し、問合せのアウトラインを作成してから、オブティマイザ・モードを再び CHOOSE に戻します。

---

---

## RDBMS アップグレードとコストベース・オブティマイザ

コストベースの最適化で Oracle の新しいバージョンにアップグレードする場合は、いくつかの SQL 文に、オブティマイザの変更に伴って変更された実行計画がある可能性が常にあります。多くの場合、このような変更はパフォーマンスにより影響を与えますが、アプリケーションによってはすでに十分に機能しており、どのような動作の変化も不必要なリスクと思われることもあります。このようなアプリケーションに対しては、次の手順により、アップグレード前にアウトラインを作成します。

---

---

**注意：** この手順をよく読んで、内容を十分理解してから実行してください。

---

---

1. 次の構文を入力して、アウトラインを作成できるようにします。

```
ALTER SESSION SET CREATE_STORED_OUTLINES = ALL_QUERIES;
```

2. 重要な SQL 文すべてにストアド・アウトラインを獲得できるよう十分に時間をとってアプリケーションを実行します。
3. 次の構文を入力して、アウトラインの生成を中断します。

```
ALTER SESSION SET CREATE_STORED_OUTLINES = FALSE;
```

4. 新しいバージョンの RDBMS に本番システムをアップグレードします。
5. アプリケーションを実行します。

アップグレード後は、ストアド・アウトラインを使用可能にするか、あるいは、アップグレード後にパフォーマンスの低下を示す文があればバックアップとして格納されていたアウトラインを使用します。

バックアップを使用する場合は、次のようにして、問題のある文にストアド・アウトラインを使用することもできます。

1. 問題のある SQL 文それぞれについて、関連するストアド・アウトラインの CATEGORY を次のようなカテゴリ名に変更します。

```
ALTER OUTLINE outline_name CHANGE CATEGORY TO problemcat;
```

2. 次の構文を入力して、Oracle がカテゴリ "PROBLEMCAT" のアウトラインを使用するようにします。

```
ALTER SESSION SET USE_STORED_OUTLINES = problemcat;
```

## テスト・システムでのアップグレード

本番システムとは別に、テスト・システムはアップグレードと合わせてオプティマイザの動作を試すのに役立ちます。インポートとエクスポートを使用して、本番システムからテスト・システムへ統計を移行できます。それにより、テスト・システムの表をデータで満たす必要が少なくなります。

アウトラインはカテゴリごとにシステム間で移動できます。たとえば、PROBLEMCAT カテゴリでアウトラインを作成したら、問合せベースのエクスポート・オプションを使用してカテゴリごとにエクスポートします。これは、ソース・データベース内のアウトラインをすべてエクスポートするのではなく、選択したアウトラインだけがあるデータベースから別のデータベースにエクスポートするうえで、便利で効率的な方法です。これを行うには、次の文を発行します。

```
EXP OUTLN/outln_password FILE=<exp-file> TABLES= 'OL$' 'OL$HINTS'  
QUERY='WHERE CATEGORY="problemcat"'
```

## ヒントの使用

アプリケーションの設計者は、オブティマイザが知らない、データに関する情報を把握していることがあるでしょう。たとえば、特定の問合せに対して、特定の索引を選択する方がよいことを知っている場合もあります。この情報に基づいて、オブティマイザよりも効率的に実行計画を選択することもできます。その場合は、ヒントを使用して、オブティマイザが最適な実行計画を使用するように強制することができます。

ヒントを使用することによって、通常オブティマイザによって行われる意思決定を行うことができます。ヒントを使用して、次の項目を指定できます。

- SQL 文に対する最適化アプローチ
- SQL 文に対するコストベースのアプローチの目標
- 文によってアクセスされる表のアクセス・パス
- 結合文の結合順序
- 結合文の結合操作

---

---

**注意：** ヒントを使用すると、管理、チェックおよび制御する必要のあるコードが増えます。

---

---

## ヒントの指定方法

ヒントは、それらが含まれる SQL 文ブロックの最適化だけに適用されます。文ブロックは、次のいずれかの文または文の一部です。

- 単純な SELECT 文、UPDATE 文または DELETE 文
- 複合文の親文または副問合せ
- 複合問合せの一部

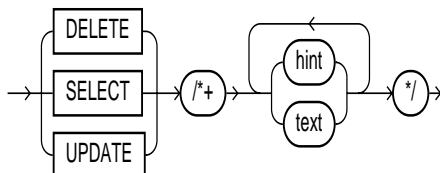
たとえば、UNION 演算子で結合した 2 つの問合せから構成されている複合問合せには、各構成要素の問合せに対して 1 つずつ、合計 2 つの文ブロックがあります。このため、この最初の構成要素の問合せにおけるヒントはその最適化だけに適用され、2 番目の構成要素の問合せの最適化には適用されません。

SQL 文に対するヒントは、文内でそれをコメントで囲むことによってオブティマイザに送ります。

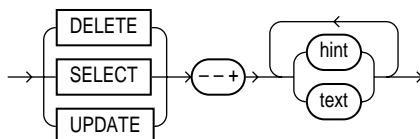
**関連項目：** コメントの詳細は、『Oracle8i SQL リファレンス』を参照してください。

文ブロックは、ヒントを含むコメントを 1 つだけを持つことができます。このコメントは、SELECT、UPDATE または DELETE キーワードの後にのみ続きます。次の構文図は、Oracle

が文ブロックでサポートするコメントの 2 つのスタイルに含まれるヒントの構文を示しています。



または



各項目は次のとおりです。

DELETE SELECT UPDATE	DELETE、SELECT または UPDATE キーワードで文ブロックは開始します。ヒントを含むコメントは、これらのキーワードの直後に限り指定できます。
+	プラス記号によって、Oracle はヒントのリストとしてコメントを解釈します。プラス記号は、コメント区切り記号の直後に続く必要があります（空白は許されません）。
<i>hint</i>	この項で説明するヒントの 1 つです。コメントが複数のヒントを含む場合、ヒントの各ペアは最低 1 つの空白で区切らなければなりません。
<i>text</i>	ヒントとともに指定できる、ヒント以外のコメントのテキストです。

次のように、ヒントの指定が間違っている場合、Oracle はそれらを見捨てるだけで、エラーを戻しません。

- ヒントを含んでいるコメントが DELETE、SELECT または UPDATE キーワードに続いていない場合、Oracle はそのヒントを見捨てます。
- Oracle は構文エラーを含むヒントを見捨てますが、同じコメント内に正しく指定されている他のヒントは考慮に入れます。
- Oracle は矛盾するヒントの組合せを見捨てますが、同じコメント内の他のヒントは考慮に入れます。

- SQL\*Forms バージョン 3 のトリガー、Oracle Forms 4.5、Oracle Reports 2.5 など、PL/SQL バージョン 1 を使う環境では、Oracle はすべての SQL 文でヒントを無視します。

索引タイプに固有のその他の条件については、この章の後半で説明します。

オブティマイザは、コストベースのアプローチを使用しているときにのみヒントを認識します。文ブロックに (RULE ヒントを除く) ヒントが含まれている場合、オブティマイザはコストベースのアプローチを自動的に使用します。

以降の項では、それぞれのヒントの構文について説明します。

## 最適化アプローチと目標のヒント

この項で説明するヒントを使用することによって、コストベースまたはルールベースの最適化アプローチを選択できます。さらに、コストベースのアプローチでは、最高のスループットまたは最高の応答時間を目標として選択できます。

- [ALL\\_ROWS](#)
- [FIRST\\_ROWS](#)
- [CHOOSE](#)
- [RULE](#)

SQL 文に最適化アプローチと目標を指定するヒントが含まれている場合、オブティマイザは、統計の有無、OPTIMIZER\_MODE 初期化パラメータの値および ALTER SESSION 文の OPTIMIZER\_MODE パラメータにかかわらず、指定されたアプローチを使います。

---

**注意：** オブティマイザの目標は直接実行される問合せにのみ適用されます。ヒントを使用して、PL/SQL の内部から実行される SQL 文のためのアクセス・パスを判断してください。ALTER SESSION... SET OPTIMIZER\_MODE 文は、PL/SQL の内部から実行される SQL には作用しません。

---

### ALL\_ROWS

ALL\_ROWS ヒントは、最高のスループットを目標として (つまり合計のリソース使用率を最小限にして) コストベースのアプローチを選択して、文ブロックを最適化します。



このヒントの構文は次のとおりです。



たとえば、オブティマイザは、次の文を最適化するために、最高のスループットを目標としたコストベースのアプローチを使用します。

```

SELECT /*+ ALL_ROWS */ empno, ename, sal, job
  FROM emp
 WHERE empno = 7566;
  
```

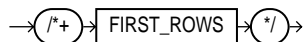
## FIRST\_ROWS

FIRST\_ROWS ヒントは、最短の応答時間を目標として（最初の行を戻すためにリソース使用率を最小限にして）コストベースのアプローチを選択して、文ブロックを最適化します。

このヒントによって、オブティマイザは次の選択を行います。

- 索引走査が使用可能な場合、オブティマイザは全表走査よりも索引走査を選択することがあります。
- 索引走査が使用可能な場合、関連する表がネストしたループの内部表であるときはいつでも、オブティマイザはソート / マージ結合よりもネスト・ループ・ジョインを選択することがあります。
- 索引走査が ORDER BY 句によって使用可能になっている場合、オブティマイザはソート処理を避けるために索引走査を選択することがあります。

このヒントの構文は次のとおりです。



たとえば、オブティマイザは、次の文を最適化するために、最短の応答時間を目標としたコストベースのアプローチを使います。

```

SELECT /*+ FIRST_ROWS */ empno, ename, sal, job
  FROM emp
 WHERE empno = 7566;
  
```

オブティマイザは、DELETE 文ブロックと UPDATE 文ブロック、および次の構文のいずれかを含む SELECT 文ブロックでこのヒントを無視します。

- 集合演算子（UNION、INTERSECT、MINUS、UNION ALL）
- GROUP BY 句
- FOR UPDATE 句

- 集計ファンクション
- DISTINCT 演算子

Oracle は最初の行を戻す前に文によってアクセスされた行をすべて検索しなければならないため、これらの文は最高の応答時間を目標に最適化することができません。これらの文にこのヒントを指定しても、オブティマイザはコストベースのアプローチを使用して、最高のスループットを目標に最適化を行います。

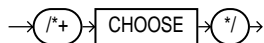
SQL 文に ALL ROWS ヒントまたは FIRST ROWS ヒントを指定した場合、データ・ディクショナリ内に文がアクセスする表に関する統計情報が作成されていないと、オブティマイザは、デフォルトの統計値（そのような表に割り当てられている記憶領域など）を使用して、欠けている統計を見積もってから、実行計画を選択します。

これらの見積りは ANALYZE 文によって生成した見積りほど正確ではありません。そのため、コストベースの最適化を使用する文によってアクセスされる表に対しては必ず ANALYZE 文を使用して統計を生成してください。ALL ROWS ヒントまたは FIRST ROWS ヒントとともに、アクセス・パスまたは結合操作を指定した場合、オブティマイザはヒントによって指定されたアクセス・パスと結合操作を優先します。

### CHOOSE

CHOOSE ヒントを使用すると、オブティマイザは、SQL 文からアクセスされる表の統計が存在するかどうかに基づいて、SQL 文のルールベースのアプローチとコストベースのアプローチのいずれかを選択します。データ・ディクショナリに、これらの表のうち 1 つ以上の統計が含まれている場合、オブティマイザは、コストベースのアプローチを使用し、最高のスループットを目標にして最適化します。データ・ディクショナリにこれらの表の統計が含まれていない場合、オブティマイザは、ルールベースのアプローチを使用します。

このヒントの構文は次のとおりです。



次に例を示します。

```
SELECT /*+ CHOOSE */ empno, ename, sal, job
FROM emp
WHERE empno = 7566;
```

### RULE

RULE ヒントは、文ブロックに対してルールベースの最適化を選択します。また、このヒントによって、オブティマイザは文ブロックに指定された他のヒントをすべて無視します。

このヒントの構文は次のとおりです。



たとえば、オプティマイザは、次の文に対してルールベースのアプローチを使用します。

```
SELECT --+ RULE
       empno, ename, sal, job
FROM   emp
WHERE  empno = 7566;
```

Oracle の将来バージョンでは、ルールベースのアプローチとともに、RULE ヒントは利用できなくなります。

## アクセス方法のヒント

この項では、表のアクセス方法を指示するいくつかのヒントについて説明します。

- FULL
- ROWID
- CLUSTER
- HASH
- HASH\_AJ
- HASH\_SJ
- INDEX
- INDEX\_ASC
- INDEX\_COMBINE
- INDEX\_JOIN
- INDEX\_DESC
- INDEX\_FFS
- NO\_INDEX
- MERGE\_AJ
- MERGE\_SJ
- AND\_EQUAL
- USE\_CONCAT

- **NO\_EXPAND**
- **REWRITE**
- **NOREWRITE**

これらのヒントの 1 つを指定すると、指定されたアクセス・パスが索引やクラスタの存在および SQL 文の構文構成体に基づいて使用できる場合だけ、オプティマイザはそのアクセス・パスを選択します。ヒントが使えないアクセス・パスを指定すると、オプティマイザはその指定を無視します。

アクセスする表は、文に指定する場合と同じように正確に指定しなければなりません。文が表の別名を使用している場合、表の名前ではなく、表の別名をヒントで使用する必要があります。スキーマ名が文中にある場合は、ヒント内の表名にそのスキーマ名を入れないください。

---

---

**注意：** アクセス・パスのヒントの場合、SELECT 文の FROM 句で SAMPLE オプションを指定すると、Oracle はヒントを無視します。SAMPLE オプションの詳細は、『Oracle8i 概要』および『Oracle8i リファレンス・マニュアル』を参照してください。

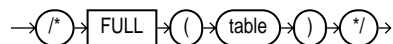
---

---

### FULL

FULL ヒントは、指定された表に対して全表走査を選択します。

このヒントの構文は次のとおりです。



*table* は、全表走査が行われる表の名前または別名です。

たとえば、Oracle では、WHERE 句の条件によって使用可能になる ACCNO 列に索引が付けられていても、ACCOUNTS 表で全表走査を実行して、この文を実行します。

```
SELECT /*+ FULL(A) DON'T USE THE INDEX ON ACCNO */ ACCNO, BAL
FROM ACCOUNTS A
WHERE ACCNO = 7086854;
```

---

---

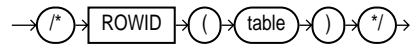
**注意：** ACCOUNTS 表は別名 "A" を持っているので、ヒントでは、名前ではなく別名によって表を参照しなければなりません。また、スキーマ名が FROM 句に指定されていても、ヒントにその名前を指定しないでください。

---

---

## ROWID

ROWID ヒントは、指定された表に対して ROWID による表走査を明示的に選択します。ROWID ヒントの構文は次のとおりです。



*table* は、ROWID によってアクセスされる表の名前または別名です。

## CLUSTER

CLUSTER ヒントは、指定された表をアクセスするためにクラスタ走査を選択します。これはクラスタ化オブジェクトにだけ適用されます。CLUSTER ヒントの構文は次のとおりです。



*table* は、クラスタ走査によってアクセスされる表の名前または別名です。

次の例に、CLUSTER ヒントの使用方法を示します。

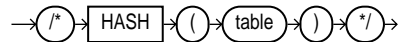
```

SELECT --+ CLUSTER
       EMP ENAME, DEPTNO
FROM   EMP, DEPT
WHERE  DEPTNO = 10 AND
       EMP.DEPTNO = DEPT.DEPTNO;

```

## HASH

HASH ヒントは、指定された表をアクセスするためにハッシュ走査を選択します。これは、クラスタに格納されている表にのみ適用されます。HASH ヒントの構文は次のとおりです。



*table* は、ハッシュ走査によってアクセスされる表の名前または別名です。

## HASH\_AJ

HASH\_AJ ヒントは、指定された表にアクセスするために、NOT IN 副問合せをハッシュ逆結合に変換します。HASH\_AJ ヒントの構文は次のとおりです。



## HASH\_SJ

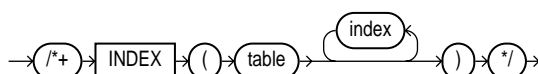
HASH\_SJ ヒントは、指定された表にアクセスするために、相互に関連する EXISTS 副問合せをハッシュ・セミ・ジョインに変換します。HASH\_SJ ヒントの構文は次のとおりです。



## INDEX

INDEX ヒントは、指定された表に対して索引走査を選択します。INDEX ヒントはドメイン・インデックス、B\* ツリー索引およびビットマップ索引に使用できます。ただし、INDEX\_COMBINEの方が INDEX よりも用途の広いヒントなので、ビットマップ索引には INDEX\_COMBINE の使用をお勧めします。

INDEX ヒントの構文は次のとおりです。



各項目は次のとおりです。

*table* 走査される索引に対応付けられている表の名前または別名です。

*index* 索引走査が行われる索引です。

このヒントでは、次のように 1 つまたは複数の索引を指定することができます。

- ヒントが使用可能な索引を 1 つ指定している場合、オプティマイザはその索引に対して走査を行います。オプティマイザは、表に対して全表走査や別の索引に対する走査を検討しません。
- ヒントが使用可能な索引のリストを指定している場合、オプティマイザはリスト内の各索引について走査コストを検討し、コストの最も低い索引走査を行います。また、このリストから複数の索引を走査し、その結果をマージするようなアクセス・パスのコストが最も低い場合、オプティマイザはそのようなアクセス・パスを選択します。ただし、オプティマイザは、全表走査や、ヒントでリストされていない索引に対する走査については検討しません。
- ヒントが索引を指定していない場合、オプティマイザは表に対して使用可能な各索引の走査コストを検討し、コストの最も低い索引走査を行います。また、複数の索引を走査し、その結果をマージするようなアクセス・パスのコストが最も低い場合、オプティマイザはそのようなアクセス・パスを選択します。オプティマイザは、全表走査については検討しません。

たとえば、入院中のすべての男性患者の名前、身長および体重を選択する次の問合せについて考えます。

```
SELECT name, height, weight
FROM patients
WHERE sex = 'm';
```

SEX 列に索引が存在し、この列には値 M と F が含まれているとします。入院中の男性患者と女性患者の数が等しい場合、問合せは表の行を比較的大きな割合で戻すため、全表走査の方が索引走査よりも高速で処理される可能性があります。しかし、入院中の男性患者の割合が非常に小さい場合、問合せは、表の行を比較的小さな割合で戻します。この場合は、索引走査の方が全表走査よりも高速で処理される可能性があります。

それぞれの列値の発生数が異なっても、オブティマイザはそれに対応できません。コストベースのアプローチでは、それぞれの値が各行に現れる可能性は等しいものと想定します。異なる値を 2 つだけ持つ列の場合、オブティマイザはそれぞれの値が行の 50% に現れるものと想定し、コストベースのアプローチは、索引走査ではなく、全表走査を選択する場合があります。

問合せの WHERE 句の値がごく一部の行にしか現れない場合、INDEX ヒントを使って、オブティマイザが索引走査を選択するように強制することができます。次の文で、INDEX ヒントは、SEX 列の索引 SEX\_INDEX に対して索引走査を明示的に選択します。

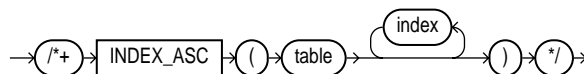
```
SELECT /*+ INDEX(PATIENTS SEX_INDEX) USE SEX_INDEX, SINCE THERE ARE FEW
        MALE PATIENTS */
        NAME, HEIGHT, WEIGHT
FROM PATIENTS
WHERE SEX = 'M';
```

INDEX ヒントは、インリスト述語に適用されます。このヒントは、インリスト述語に関して、可能な場合にはヒントで指定した索引をオブティマイザに強制的に使用させます。複数列のインリストは索引を使用しません。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## INDEX\_ASC

INDEX\_ASC ヒントは、指定された表に対して索引走査を選択します。文が索引範囲走査を使う場合、Oracle は索引付きの値について昇順に索引エントリを走査します。INDEX\_ASC ヒントの構文は次のとおりです。



各パラメータは、INDEX ヒントと同じ目的で指定します。

Oracle の範囲走査のデフォルト動作は、索引付けされた値の昇順に索引項目を走査する作業なので、このヒントでは、INDEX ヒント以外に何も指定しません。ただし、デフォルトの

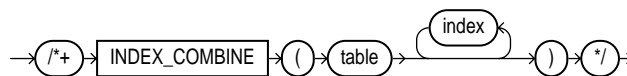
動作が変更された場合は、INDEX\_ASC ヒントを使って昇順の範囲走査を明示的に指定するとよいでしょう。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。

**関連項目：** INDEX\_ASC ヒントの詳細は、『Oracle8i 分散システム』を参照してください。

## INDEX\_COMBINE

INDEX\_COMBINE ヒントは、表のビットマップ・アクセス・パスを明示的に選択します。INDEX\_COMBINE ヒントに引数として索引が与えられなかった場合、オプティマイザは、最良のコストを見積もることができるビットマップ索引のブール値の組合せを表に対して使用します。引数として特定の索引が与えられた場合、オプティマイザは、指定されたビットマップ索引のブール値の組合せの使用を試みます。INDEX\_COMBINE の構文は次のとおりです。



このヒントは、分散問合せの最適化を使用する場合やビットマップ索引に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## INDEX\_JOIN

INDEX\_JOIN ヒントは、オプティマイザが索引結合をアクセス・パスとして使用することを明示的に指示します。このヒントがよい影響を及ぼすには、問合せの解決に必要な列をすべて含む索引がいくらか必要です。



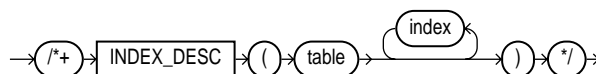
各項目は次のとおりです。

<i>table</i>	走査される索引に対応付けられている表の名前または別名です。
<i>index</i>	索引走査が行われる索引です。



## INDEX\_DESC

INDEX\_DESC ヒントは、指定された表に対して索引走査を選択します。文が索引範囲走査を使用する場合、Oracle は索引エントリを索引付きの値について降順に走査します。INDEX\_DESC ヒントの構文は次のとおりです。

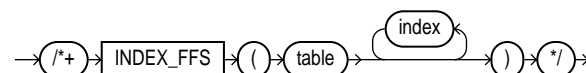


各パラメータは、INDEX ヒントと同じ目的で指定します。このヒントは、複数の表にアクセスする SQL 文には効果がありません。そのような文は、常に索引付きの値について昇順で範囲走査を行います。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## INDEX\_FFS

このヒントによって、全表走査ではなく高速全索引走査が実行されます。INDEX\_FFS の構文は次のとおりです。

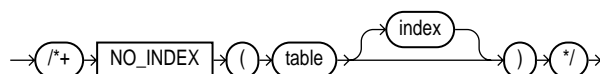


**関連項目：** 6-8 ページの「[高速全索引走査の使用](#)」

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## NO\_INDEX

NO\_INDEX ヒントは、指定された表の索引を明示的に禁止します。NO\_INDEX ヒントの構文は次のとおりです。



このヒントを使用すると、次のように 1 つまたは複数の索引を指定することができます。

- ヒントが使用可能な索引を 1 つ指定している場合、オプティマイザはその索引に対して走査を検討しません。指定されていないその他の索引はひきつづき検討されます。

- ヒントが使用可能な索引のリストを指定している場合、オブティマイザはその索引のどれに対しても走査を検討しません。リストで指定されていないその他の索引は検討されます。
- このヒントで索引を指定しない場合、オブティマイザは表のどの索引に対しても走査を検討しません。この動作は、NO\_INDEX ヒントで使用可能な表の索引すべてのリストを指定した場合と同じです。

NO\_INDEX ヒントは、関数ベース索引、B\* ツリー索引、ビットマップ索引、クラスタ索引またはドメイン・インデックスに適用されます。

NO\_INDEX ヒントと索引ヒント (INDEX、INDEX\_ASC、INDEX\_DESC、INDEX\_COMBINE または INDEX\_FFS) の両方で同じ索引を指定すると、指定した索引の NO\_INDEX ヒントも索引ヒントも無視され、オブティマイザはその索引を検討します。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## MERGE\_AJ

MERGE\_AJ ヒントは、指定された表にアクセスするために、NOT IN 副問合せをマージ逆結合に変換します。MERGE\_AJ ヒントの構文は次のとおりです。



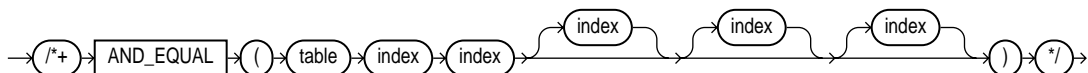
## MERGE\_SJ

MERGE\_SJ ヒントは、指定された表にアクセスするために、相互に関連する EXISTS 副問合せをマージ・セミ・ジョインに変換します。MERGE\_SJ ヒントの構文は次のとおりです。



## AND\_EQUAL

AND\_EQUAL ヒントは、複数の単一列索引の走査をマージするアクセス・パスを使う実行計画を選択します。AND\_EQUAL ヒントの構文は次のとおりです。



各項目は次のとおりです。

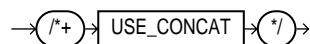
<i>table</i>	マージされる索引に対応付けられている表の名前または別名。
<i>index</i>	索引走査が行われる索引。索引は最低 2 つ指定しなければなりません。5 つを超える索引を指定することはできません。

## USE\_CONCAT

USE\_CONCAT ヒントを使用すると、問合せの WHERE 句の OR 条件の組合せが、UNION ALL 集合演算子を使用する複合問合せに強制的に変換されます。通常、この変換は、連結を使用する問合せのコストが使用しないコストよりも低い場合に限り実行します。

USE\_CONCAT ヒントは、インリストの処理をオフにし、インリストに含まれるすべてを分離して OR に変換します。

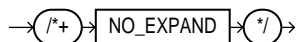
このヒントの構文は次のとおりです。



## NO\_EXPAND

NO\_EXPAND ヒントは、コストベース・オブティマイザが OR 条件または WHERE 句の INLISTS を持つ問合せの OR 拡張を検討するのを防ぎます。通常、オブティマイザは OR 拡張の使用を検討し、使用しない場合よりもコストが低いと判断すると、このメソッドを使用します。

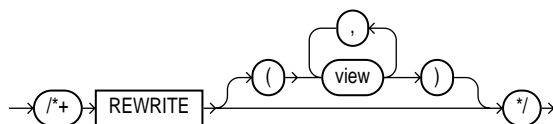
このヒントの構文は次のとおりです。



## REWRITE

REWRITE ヒントは、ビュー・リストとともにまたはビュー・リストなしで使用します。REWRITE ヒントをビュー・リストとともに使用する場合、リストに適当なマテリアライズド・ビューがあると、Oracle はコストに関わらずそのビューを使用します。リスト外のビューは検討されません。ビュー・リストを指定しない場合は、Oracle によって適当なマテリアライズド・ビューが検索され、コストに関わらずそれが常に使用されます。

このヒントの構文は次のとおりです。



**関連項目：** マテリアライズド・ビューの詳細は、『Oracle8i 概要』および『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。

### NOREWRITE

NOREWRITE ヒントは、要求の問合せブロックで使用します。このヒントは、パラメータ QUERY\_REWRITE\_ENABLED の設定を上書きして、問合せブロックのクエリー・リライトを使用禁止にします。

このヒントの構文は次のとおりです。



## 結合順序のヒント

この項で説明するヒントは、結合順序を指示します。

- ORDERED
- STAR

### ORDERED

ORDERED ヒントによって、Oracle は FROM 句に指定された順序で表を結合します。

このヒントの構文は次のとおりです。



たとえば、次の文は表 TAB1 を表 TAB2 と結合し、その結果を表 TAB3 と結合します。

```
SELECT /*+ ORDERED */ TAB1.COL1, TAB2.COL2, TAB3.COL3
FROM TAB1, TAB2, TAB3
WHERE TAB1.COL1 = TAB2.COL1
AND TAB2.COL1 = TAB3.COL1;
```

結合を行う SQL 文に ORDERED ヒントを指定しない場合は、表を結合する順序をオプティマイザが選択します。オプティマイザにはわからないような、各表から選択される行数に関する情報を把握している場合、ORDERED ヒントを使って結合順序を指定するとよいでしょう。そのような情報によって、オプティマイザよりも適切に内部表と外部表を選択することができます。

## STAR

STAR ヒントは、可能な場合にはスター問合せ計画の使用を強制します。スター計画は、結合順序の最後の問合せに最大の表を持ち、それを連結索引上のネスト・ループ・ジョインに結合します。STAR ヒントが適用されるのは、少なくとも 3 つの表があり、最大の表の連結索引が少なくとも 3 つの列を持ち、アクセスまたは結合方法のヒントが矛盾しない場合です。オプティマイザは、小規模表の別の並換えも考慮します。

このヒントの構文は次のとおりです。



表を分析する場合、通常はオプティマイザが効率のよいスター計画を選択します。また、ヒントを使用して、その計画を改良できます。最も精密な方法は、FROM 句内の表の順序を、索引内のキーの順序と同じにし、大規模表を最後に指定することです。その後、次のヒントを使用します。

```
/*+ ORDERED USE_NL(FACTS) INDEX(FACTS FACT_CONCAT) */
```

ここで、"facts" は表、"fact\_concat" は索引です。より一般的な方法は、STAR ヒントを使うことです。

**関連項目：** スター計画の詳細は、『Oracle8i 概要』を参照してください。

## 結合操作のヒント

この項では、表の結合処理を指示するヒントについて説明します。

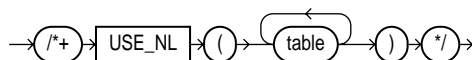
- [USE\\_NL](#)
- [USE\\_MERGE](#)
- [USE\\_HASH](#)
- [DRIVING\\_SITE](#)
- [HASH\\_AJ と MERGE\\_AJ](#)
- [HASH\\_SJ と MERGE\\_SJ](#)

結合する表は、文に指定する場合と同じように正確に指定しなければなりません。文が表の別名を使用している場合、表の名前ではなく、表の別名をヒントで使用する必要があります。スキーマ名が文中にある場合は、ヒント内の表名にそのスキーマ名を入れないでください。

USE\_NL ヒントと USE\_MERGE ヒントは、ORDERED ヒントとともに使用することをお勧めします。Oracle では、参照表が結合の内部表になったときにこれらのヒントを使用し、参照表が外部表のときにはこれらのヒントを無視します。

## USE\_NL

USE\_NL ヒントによって、内部表として指定した表を使っているネスト・ループ・ジョインによって、別の行ソースに指定したそれぞれの表を結合します。USE\_NL ヒントの構文は次のとおりです。



*table* は、ネスト・ループ・ジョインの内部表として使用される表の名前または別名です。

たとえば、ACCOUNTS 表と CUSTOMERS 表を結合する次のような文があるとします。これらの表は、クラスタにはともに格納されていないものと想定します。

```

SELECT ACCOUNTS.BALANCE, CUSTOMERS.LAST_NAME, CUSTOMERS.FIRST_NAME
FROM ACCOUNTS, CUSTOMERS
WHERE ACCOUNTS.CUSTNO = CUSTOMERS.CUSTNO;

```

コストベースのアプローチでは最高のスループットがデフォルトの目標であるため、オプティマイザは、ネストしたループ操作またはソート / マージ操作を選択してこれらの表を結合します。この場合、問合せによって選択されるすべての行をより速く戻す操作が選択されます。

ただし、スループットを最高にするよりも、応答時間を最高にする、つまり問合せによって選択される最初の行を戻すために必要な経過時間を最小にするために、文を最適化することもできます。その場合、USE\_NL ヒントを使うことによって、オプティマイザにネスト・ループ・ジョインを選択するように強制できます。次の文で、USE\_NL ヒントは、CUSTOMERS 表を内部表とするネスト・ループ・ジョインを選択します。

```

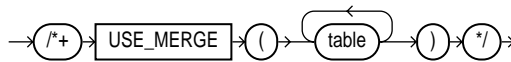
SELECT /*+ ORDERED USE_NL(CUSTOMERS) USE N-L TO GET FIRST ROW FASTER */
ACCOUNTS.BALANCE, CUSTOMERS.LAST_NAME, CUSTOMERS.FIRST_NAME
FROM ACCOUNTS, CUSTOMERS
WHERE ACCOUNTS.CUSTNO = CUSTOMERS.CUSTNO;

```

多くの場合、ネスト・ループ・ジョインは、ソート / マージ結合よりも速く最初の行を戻します。ソート / マージ結合では、両方の表のすべての選択行を読み込んでソートし、ソート済みの行ソースの最初の行の結合が終わってからでないと、最初の行を戻せません。これに対して、ネスト・ループ・ジョインは、一方の表から最初に選択した行を読み込み、もう一方の表から最初の一致行を読み込んで、それらを結合した後に、最初の行を戻すことができます。

## USE\_MERGE

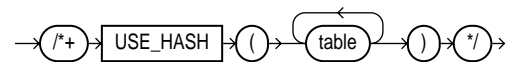
USE\_MERGE ヒントによって、Oracle はソート / マージ結合で指定されたそれぞれの表を別の行ソースと結合します。USE\_MERGE ヒントの構文は次のとおりです。



*table* は、ソート / マージ結合を使用して結合順序の前の表を結合した結果の行ソースと結合する表です。

## USE\_HASH

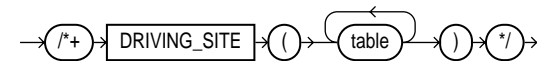
USE\_HASH ヒントによって、Oracle は、指定されたそれぞれの表と別の行ソースとをハッシュ結合を使って結合します。USE\_HASH ヒントの構文は次のとおりです。



*table* は、結合順序の中で前の表を結合した結果発生する行ソースとハッシュ結合を使用して結合する表です。

## DRIVING\_SITE

DRIVING\_SITE ヒントを使用すると、Oracle が選択したサイトとは別のサイトで問合せ実行を行うことができます。このヒントは、ルールベースの最適化とコストベースの最適化のどちらでも使用できます。このヒントの構文は次のとおりです。



*table* は、実行が行われるサイトにある表の名前または別名です。

例：

```
SELECT /*+DRIVING_SITE(DEPT)*/ * FROM EMP, DEPT@RSITE
WHERE EMP.DEPTNO = DEPT.DEPTNO;
```

この問合せがヒントなしで実行されると、DEPT の行はローカル・サイトに送られ、結合はそこで実行されます。ヒント付きで実行されると、EMP の行はリモート・サイトに送られ、問合せはそこで実行されて、結果がローカル・サイトに戻されます。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## HASH\_AJ と MERGE\_AJ

図 7-1 パラレル・ハッシュ逆結合

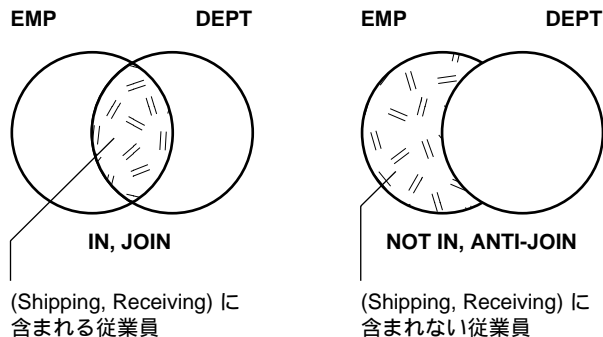


図 7-1 で示すように、SQL の IN 述語は、2 つの集合にまたがる結合を使用して評価されます。このため、emp.deptno は dept.deptno と結合され、部門 (DEPT) の集合内の従業員 (EMP) のリストが生成されます。

または、SQL の NOT IN 述語は、2 つの集合を減算する逆結合を使って評価されます。emp.deptno は dept.deptno と逆結合され、部門の集合に含まれないすべての従業員が選択されます。このため、Shipping または Receiving 部門に所属しないすべての従業員のリストが得られます。

特定の問合せに対しては、MERGE\_AJ または HASH\_AJ ヒントを NOT IN 副問合せの中に入れます。MERGE\_AJ はソート / マージ逆結合を、HASH\_AJ はハッシュ逆結合を使います。

次に例を示します。

```
SELECT * FROM EMP
WHERE ENAME LIKE 'J%' AND
DEPTNO IS NOT NULL AND
DEPTNO NOT IN (SELECT /*+ HASH_AJ */ DEPTNO FROM DEPT
WHERE DEPTNO IS NOT NULL AND
LOC = 'DALLAS');
```

前述の条件が満たされた場合に常に逆結合への変形が発生するように希望するときは、ALWAYS\_ANTI\_JOIN 初期化パラメータに MERGE または HASH を設定します。実行可能なときは、該当する種類の逆結合が常に発生します。



## HASH\_SJ と MERGE\_SJ

特定の問合せに対しては、HASH\_SJ または MERGE\_SJ ヒントを EXISTS 副問合せの中を含めます。HASH\_SJ はハッシュ・セミ・ジョインを使い、MERGE\_SJ はソート・マージ・セミ・ジョインを使用します。次に例を示します。

```
SELECT * FROM T1
WHERE EXISTS (SELECT /*+ HASH_SJ */ * FROM T
WHERE T1.C1 = T2.C1
AND T2.C3 > 5);
```

この場合は、副問合せが t1 と t2 の特殊な種類の結合に変換されます。このとき、この副問合せの意味は保たれます。つまり、t1 の 1 つの行に対して t2 に一致する行が複数ある場合でも、t1 の行は一度しか戻されません。

副問合せがセミ・ジョインとして評価されるのは、次の制限がある場合のみです。

- 副問合せに 1 つしか表がない場合。
- 外部問合せブロックそのものは副問合せではない場合。
- 副問合せが等価述語と相互に関係している場合。
- 副問合せが GROUP BY、CONNECT BY または ROWNUM の参照を含まない場合。

前述の条件が満たされた場合に常にセミ・ジョインへの変形が発生するようにしたいときは、ALWAYS\_SEMI\_JOIN 初期化パラメータに HASH または MERGE を設定します。実行可能なときは、該当する種類のセミ・ジョインが常に発生します。

## パラレル実行のヒント

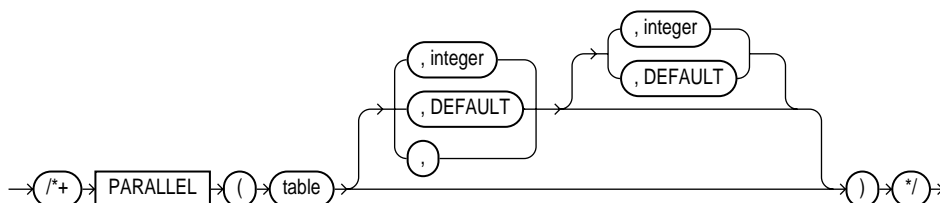
この項では、パラレル処理を行ったときに、文がどのように並列化されるか、または並列化されないかを指示するヒントについて説明します。

- [PARALLEL](#)
- [NOPARALLEL](#)
- [PQ\\_DISTRIBUTE](#)
- [APPEND](#)
- [NOAPPEND](#)
- [PARALLEL\\_INDEX](#)
- [NOPARALLEL\\_INDEX](#)

**関連項目：** [第 26 章「パラレル実行のチューニング」](#)

## PARALLEL

PARALLEL ヒントを使って、パラレル操作のために使える同時サーバーの数を自由に指定します。このヒントは、SQL 文の INSERT、UPDATE および DELETE 部分と表走査部分に適用されます。パラレル制限のいずれかに違反すると、ヒントは無視されます。構文は次のとおりです。



PARALLEL ヒントでは、別名が問合せに指定されている場合、表別名を使用しなければなりません。このヒントでは、表名の後に 2 つの値を取ることができ、これらカンマで区切ります。最初の値では、指定の表の並列度を指定し、2 番目の値では、パラレル・サーバーの各インスタンスでどのように表を分割するかを指定します。DEFAULT を指定するか、または値を指定しないと、問合せコーディネータが初期化パラメータ（後の項で説明）の設定を調べ、デフォルトの並列度を識別します。

次の例では、PARALLEL ヒントによって、EMP 表定義に指定されている並列度が上書きされます。

```

SELECT /*+ FULL(SCOTT_EMP) PARALLEL(SCOTT_EMP, 5) */ ENAME
FROM SCOTT.EMP SCOTT_EMP;
  
```

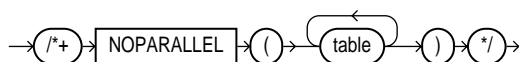
次の例では、PARALLEL ヒントによって、EMP 表定義に指定されている並列度が上書きされ、オプティマイザが、初期化パラメータに指定されているデフォルトの並列度を使うように命令されます。また、このヒントでは、表をすべての使用可能なインスタンスに分割し、各インスタンスでデフォルトの並列度を使用するように指定します。

```

SELECT /*+ FULL(SCOTT_EMP) PARALLEL(SCOTT_EMP, DEFAULT,DEFAULT) */ ENAME
FROM SCOTT.EMP SCOTT_EMP;
  
```

## NOPARALLEL

NOPARALLEL ヒントを使用すると、表に設定されている PARALLEL 指定が上書きされません。一般に、ヒントは表句よりも優先されます。このヒントの構文は次のとおりです。



次の例に、NOPARALLEL ヒントを示します。

```
SELECT /*+ NOPARALLEL(scott_emp) */ ename
FROM scott.emp scott_emp;
```

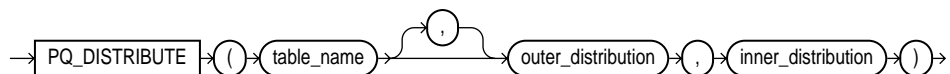
NOPARALLEL ヒントは、次のようなヒントを指定するのと同じです。

## PQ\_DISTRIBUTE

PQ\_DISTRIBUTE ヒントを使用すると、パラレル結合操作のパフォーマンスが改善されます。これを行うには、結合した表の列を作成者と顧客の問合せサーバー間で配布する方法を指定します。このヒントを使用すると、オブティマイザが通常行う決定は上書きされます。

オブティマイザによって選択された配布を識別するには、EXPLAIN PLAN 文を使用します。表が両方ともシリアルの場合、オブティマイザは配布ヒントを無視します。Oracle が結合操作をパラレル化する方法の詳細は、『Oracle8i 概要』を参照してください。

配布ヒントの構文は次のとおりです。



各項目は次のとおりです。

<i>table_name</i>	結合の内部表として使用される表の名前または別名。
<i>outer_distribution</i>	外部表の配布です。
<i>inner_distribution</i>	内部表の配布です。

表の配布には次の 6 つの組合せがあります。

- ハッシュ対ハッシュ
- ブロードキャスト対なし
- なし対ブロードキャスト
- パーティション対なし
- なし対パーティション
- なし対なし

表 7-3 で説明しているように、結合表の配布方法の組合せはサブセットのみが有効です。次に例を示します。

表 7-3 配布ヒントの組合せ

配布	説明
ハッシュ対ハッシュ	結合キーでハッシュ関数を使用して、各表の行を顧客問合せサーバーにマップします。マップが完了すると、各問合せサーバーは結果パーティションのペア間で結合を実行します。このヒントは、表のサイズが同じくらいで、結合操作がハッシュ結合またはソート / マージ結合でインプリメントされている場合にお勧めします。
ブロードキャスト対なし	外部表のすべての行が各問合せサーバーにブロードキャストされます。内部表の行はランダムにパーティション化されます。このヒントは、外部表が内部表に比べて非常に小さい場合にお勧めします。大まかな指針としては、ブロードキャストとなしのヒントは、内部表のサイズ * 問合せサーバーの数 < 外部表のサイズである場合に使用します。
なし対ブロードキャスト	内部表のすべての行が各顧客問合せサーバーにブロードキャストされます。外部表の行はランダムにパーティション化されます。このヒントは、内部表が外部表に比べて非常に小さい場合にお勧めします。大まかな指針として、なしとブロードキャストのヒントは、内部表のサイズ * 問合せサーバーの数 < 外部表のサイズである場合に使用します。
パーティション対なし	内部表のパーティション化を使用して、外部表の行をマップします。内部表は、結合キーでパーティション化する必要があります。このヒントは、外部表のパーティションの数が問合せサーバーの数と等しいか、その倍数に近い場合にお勧めします。たとえば、パーティションが 14 で問合せサーバーが 15 の場合などです。 <b>注意:</b> 内部表がパーティション化されていない場合やパーティション化キーで等価結合されていない場合、オプティマイザはこのヒントを無視します。
なし対パーティション	外部表のパーティション化を使用して、内部表の行をマップします。外部表は、結合キーでパーティション化する必要があります。このヒントは、外部表のパーティションの数が問合せサーバーの数と等しいか、その倍数に近い場合にお勧めします。たとえば、パーティションが 14 で問合せサーバーが 15 の場合などです。 <b>注意:</b> 外部表がパーティション化されていない場合やパーティション化キーで等価結合されていない場合、オプティマイザはこのヒントを無視します。
なし対なし	各問合せサーバーは、各テーブルから 1 つずつ、一致するパーティションのペア間で結合操作を実行します。どちらの表も、結合キーで等価パーティション化する必要があります。

例 :R と S の 2 つの表がハッシュ結合を使用して結合された場合は、ハッシュ配布を使用するヒントが次の問合せに組み込まれます。

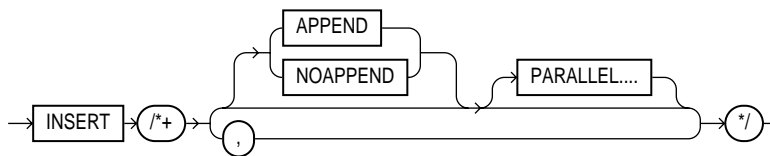
```
SELECT <column_list>
/*+ORDERED PQ_DISTRIBUTE(S HASH HASH) USE_HASH (S)*/
FROM R,S
WHERE R.C=S.C;
```

外部表 R をブロードキャストするには、問合せを次のようにします。

```
SELECT <column list>
/*+ORDERED PQ_DISTRIBUTE(S BROADCAST NONE) USE_HASH (S) */
FROM R,S
WHERE R.C=S.C;
```

## APPEND

INSERT に APPEND ヒントを使用すると、データが表に単純に追加されます。ブロック内の既存の部分空き領域は使用されません。このヒントの構文は次のとおりです。



INSERT が PARALLEL ヒントまたは句を使用して並列化されている場合、デフォルトでは APPEND モードが使用されます。NOAPPEND を使用すると APPEND モードを上書きできます。APPEND ヒントはシリアル挿入とパラレル挿入の両方に適用されます。

追加操作は、対象となる表に対して [NO]LOGGING オプションが設定されているかどうかに応じて、LOGGING または NOLOGGING モードで実行されます。ALTER TABLE... [NO]LOGGING 文を使用して適切な値を設定してください。

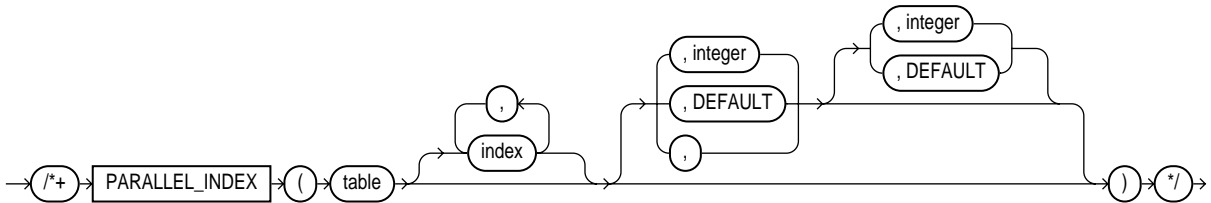
APPEND ヒントには特定の制限が適用されます。これらの制限は、『Oracle8i 概要』に記載されています。これらの制限のいずれかに違反すると、ヒントは無視されます。

## NOAPPEND

NOAPPEND を使用すると、追加モードが上書きされます。

## PARALLEL\_INDEX

PARALLEL\_INDEX ヒントを使用して、パーティション索引の索引範囲走査をパラレル化するために同時に使用するサーバー数を指定します。PARALLEL\_INDEX ヒントの構文は次のとおりです。



各項目は次のとおりです。

*table* 走査される索引に対応付けられている表の名前または別名。

*index* 索引走査が行われる索引（オプション）。

このヒントでは、表名の後に 2 つの値を取ることができ、これらをカンマで区切ります。最初の値では、指定の表の並列度を指定します。2 番目の値では、パラレル・サーバーのインスタンスでどのように表を分割するかを指定します。DEFAULT を指定するか、または値を指定しないと、問合せコーディネータが初期化パラメータ（後の項で説明）の設定を調べ、デフォルトの並列度を識別します。

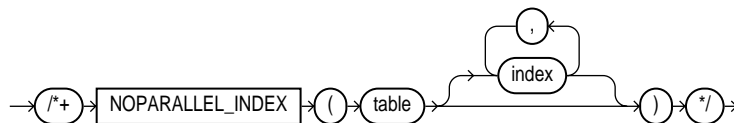
次に例を示します。

```
SELECT /*+ PARALLEL_INDEX(TABLE1,INDEX1, 3, 2) */;
```

この例では、それぞれ 2 つのインスタンスで 3 つのパラレル・サーバー・プロセスが使用されます。

## NOPARALLEL\_INDEX

NOPARALLEL\_INDEX ヒントを使用すると、索引に設定されている PARALLEL 属性が上書きされます。このようにして、パラレル索引走査を回避できます。このヒントの構文は次のとおりです。



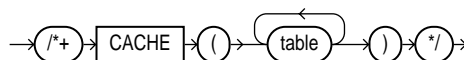
## その他のヒント

この項ではその他のいくつかのヒントを説明します。

- [CACHE](#)
- [NOCACHE](#)
- [MERGE](#)
- [NO\\_MERGE](#)
- [PUSH\\_JOIN\\_PRED](#)
- [NO\\_PUSH\\_JOIN\\_PRED](#)
- [PUSH\\_SUBQ](#)
- [STAR\\_TRANSFORMATION](#)
- [ORDERED\\_PREDICATES](#)

## CACHE

CACHE ヒントでは、全表走査が実行され、取り出されたブロックが、バッファ・キャッシュ内で LRU リストの最後に使用されたものの位置に配置されるように指定します。このオプションは、小さい参照表の場合に役立ちます。このヒントの構文は次のとおりです。



次の例では、CACHE ヒントによって、表のデフォルトのキャッシュ仕様が上書きされます。

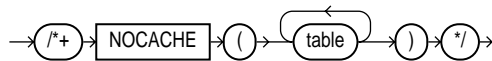
```

SELECT /*+ FULL (SCOTT_EMP) CACHE(SCOTT_EMP) */ ENAME
FROM SCOTT.EMP SCOTT_EMP;

```

## NOCACHE

NOCACHE ヒントでは、全表走査が実行され、取り出されたブロックが、バッファ・キャッシュ内で最低使用頻度の LRU リストの終わりに配置されるように指定します。これは、バッファ・キャッシュ内のブロックの通常の動作です。このヒントの構文は次のとおりです。



次の例に、NOCACHE ヒントを示します。

```

SELECT /*+ FULL(SCOTT_EMP) NOCACHE(SCOTT_EMP) */ ENAME
FROM SCOTT.EMP SCOTT_EMP;

```

## MERGE

MERGE ヒントを使用することによって、問合せごとにビューをマージするようにできます。このヒントの構文は次のとおりです。



次に例を示します。

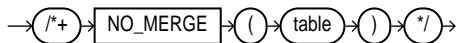
```

SELECT /*+ MERGE(V) */ T1.X, V.AVG_Y
FROM T1
     (SELECT X, AVG(Y) AS AVG_Y
      FROM T2
      GROUP BY X) V
WHERE T1.X = V.X AND T1.Y = 1;

```

## NO\_MERGE

NO\_MERGE ヒントを使用すると、Oracle はマージ可能なビューをマージしません。NO\_MERGE ヒントの構文は次のとおりです。



このヒントを使用すると、ビューにアクセスする方法に、ユーザーがより多くの影響を及ぼすことができます。次に例を示します。

```

SELECT /*+ NO_MERGE(V) */ T1.X, V.AVG_Y
FROM T1

```



```
(SELECT X, AVG(Y) AS AVG_Y
FROM T2
GROUP BY X) V
WHERE T1.X = V.X AND T1.Y = 1;
```

これにより、ビュー *v* はマージされません。

引数を指定せずに NO\_MERGE ヒントを使用するときは、ビュー問合せブロック内に入れてください。ビュー名を引数として指定して NO\_MERGE を使用するときは、周囲の問合せに入れてください。

このヒントは、分散問合せの最適化を使用する場合に役立ちます。詳細は、『Oracle8i 分散システム』を参照してください。

## PUSH\_JOIN\_PRED

PUSH\_JOIN\_PRED ヒントを使用すると、述語結合が強制的にビューへ送信されます。

このヒントの構文は次のとおりです。

→ (/\*+ → PUSH\_JOIN\_PRED → ( → table → ) → \*/ →

次に例を示します。

```
SELECT /*+ PUSH_JOIN_PRED(V) */ T1.X, V.Y
FROM T1
      (SELECT T2.X, T3.Y
FROM T2, T3
      WHERE T2.X = T3.X) V
where t1.x = v.x and t1.y = 1;
```

## NO\_PUSH\_JOIN\_PRED

NO\_PUSH\_JOIN\_PRED ヒントを使用すると、述語結合がビューへ送信されません。このヒントの構文は次のとおりです。

→ (/\*+ → NO\_PUSH\_JOIN\_PRED → ( → table → ) → \*/) →

## PUSH\_SUBQ

PUSH\_SUBQ ヒントを使用すると、マージされていない副問合せは、実行計画で最初に可能なステップで評価されます。通常、マージされていない副問合せは、実行計画の最後のステップとして実行されます。副問合せのコストが相対的に低く、副問合せが行数を大幅に減少させる場合には、パフォーマンスが向上し、副問合せの評価が速くなります。

このヒントは、副問合せがリモート表に適用されている場合、またはマージ結合を使って結合されたリモート表に適用されている場合は、効果がありません。このヒントの構文は次のとおりです。

→ (/\*+ → PUSH\_SUBQ → \*/) →

## STAR\_TRANSFORMATION

STAR\_TRANSFORMATION ヒントによって、オブティマイザは、変換 (TRANSFORMATION) が使われている最適な計画を使います。このヒントを使用しない場合、オブティマイザは、変換された問合せのための最適な計画のかわりに、変換なしで生成された最善の計画を使うというコストベースの決定を行うことがあります。

ヒントが与えられている場合でも、変換が行われる保証はありません。オブティマイザは、合理的と思われる場合にのみ副問合せを生成します。副問合せが生成されない場合は、変換された問合せがないので、変換されていない問合せの最善の計画がヒントとは無関係に使用されます。

このヒントの構文は次のとおりです。

→ (/\*+ → STAR\_TRANSFORMATION → \*/) →

**関連項目：** スター変換の詳細は、『Oracle8i 概要』を参照してください。  
また、『Oracle8i リファレンス・マニュアル』で  
STAR\_TRANSFORMATION\_ENABLED について説明します。このパラ  
メータによって、オブティマイザがスター変換の実行を考慮するようにし  
ます。

## ORDERED\_PREDICATES

ORDERED\_PREDICATES ヒントを使用すると、索引キーとして使用されている術語を除く  
述語評価の順序をオブティマイザに保存させることができます。このヒントは、SELECT 文  
の WHERE 句で使用します。

ORDERED\_PREDICATES ヒントを使用しない場合は、Oracle によってすべての述語が次の  
規則に従って、指定した順に評価されます。

- ユーザー定義のファンクション、タイプ・メソッド、または副問合せのない述語は、  
WHERE 句で指定された順序。
- ユーザーが計算したコストを持つユーザー定義ファンクションとタイプ・メソッドのあ  
る述語は、コストの昇順。
- ユーザーが計算したコストを持たないユーザー定義のファンクションとタイプ・メソ  
ッドのある述語は、WHERE 句で指定された順序。
- WHERE 句で指定されていない述語（たとえば、オブティマイザが過渡的に生成した述  
語）は、ここで評価されます。
- 副問合せを持つ述語は、WHERE 句で指定された順序で最後に評価されます。

---

**注意：** 前述したように、ORDERED\_PREDICATES ヒントを使用して索  
引キーの述語評価の順序を保存することはできません。

---

このヒントの構文は次のとおりです。

→ (/\*+ → ORDERED\_PREDICATES → \*/) →

**関連項目：** 述語の順序付けの詳細は、『Oracle8i 概要』を参照してくださ  
い。

## ビューでのヒントの使用方法

ビュー（または副問合せ）の内側やそれに対するヒントの使用は、お薦めしません。あるコンテキストで定義したビューを別のコンテキストで使うことができるため、そのようなヒントが予期しない結果を招くことがあるからです。特に、ビュー内のヒントまたはビューに対するヒントは、そのビューが最上位の問合せにマージ可能かどうかによって処理方法が異なります。

それでも、ビューでヒントを使用するようにする場合は、この後の項で状況ごとの動作についての説明があります。

- [ヒントとマージ可能ビュー](#)
- [ヒントとマージ不可能ビュー](#)

### ヒントとマージ可能ビュー

この項では、マージ可能なビューでのヒントの動作について説明します。

**最適化アプローチと目標のヒント** 最適化アプローチと目標のヒントは、最上位の問合せに、またはビューの内側に指定できます。

- そのようなヒントが最上位の問合せにある場合は、ビューの内側にあるヒントにかかわらずそのヒントが使用されます。
- 最上位のオプティマイザ・モード・ヒントがない場合は、参照されているビューのすべてのモード・ヒントに一貫性がある限り、それらのモード・ヒントが使用されます。
- 参照されているビューの2つまたは3つのモード・ヒントが矛盾する場合は、そのビューのすべてのモード・ヒントが廃棄されて、セッション・モードが使用されます（デフォルトがユーザー指定かには関係しません）。

**ビューに対するアクセス方法と結合ヒント** 参照されるビューに対するアクセス方法と結合ヒントは、そのビューが単一の表を含んでいない限り（または単一の表を持つ他のビューを参照していない限り）無視されます。そのような単一表ビューでは、ビューに対するアクセス方法や結合ヒントは、そのビューの中の表に対して適用されます。

**ビューの内側のアクセス方法と結合ヒント** アクセス方法と結合ヒントは、ビュー定義に含めることができます。

- ビューが副問合せの場合（つまり、ビューが SELECT 文の FROM 句にある場合）、そのビューの内側のすべてのアクセス方法と結合ヒントは、そのビューが最上位の問合せにマージされるときに保存されます。
- 副問合せでないビューでは、そのビューの中のアクセス方法と結合ヒントが保存されるのは、最上位の問合せが他の表やビューを参照していない場合（つまり、SELECT 文の FROM 句にそのビューしか含まれていない場合）のみ。

**ビューに対するパラレル実行ヒント** ビューに対する PARALLEL、NOPARALLEL、PARALLEL\_INDEX、および NOPARALLEL\_INDEX ヒントは、参照されるビュー内のすべての表に常に繰り返し適用されます。最上位の問合せのパラレル実行ヒントは、参照されるビューの内側のそのようなヒントを上書きします。

## ヒントとマージ不可能ビュー

マージ不可能なビューでは、ビューの内側の最適化アプローチと目標のヒントは無視されません。つまり、最上位の問合せによって最適化モードが決定されます。

マージ不可能なビューは最上位の問合せとは別に最適化されるので、そのビューの内側のアクセス方法と結合ヒントは常に保存されます。同じ理由から、最上位の問合せ内のビューに対するアクセス方法も無視されます。

ただし、最上位の問合せ内のビューに対する結合ヒントは保存されます。この場合、マージ不可能なビューは表と同様であるためです。



---

## 分散問合せのチューニング

Oracle は、複数のデータベースのデータにアクセスするための透過的分散問合せをサポートしています。また、透過的分散トランザクションや透過的全自動 2 フェーズ・コミットなど、他にも多くの分散機能を提供しています。この章では、Oracle8i オプティマイザが SQL 文を分解する方法と、それが分散問合せにどのように影響するかを説明します。この章では、オプティマイザを制御する方法と、パフォーマンスのボトルネックを避ける方法のガイドラインを示します。

トピックは次のとおりです。

- [リモート問合せおよび分散問合せ](#)
- [分散問合せの制限事項](#)
- [Transparent Gateway](#)
- [まとめ : 分散問合せのパフォーマンスの最適化](#)

### リモート問合せおよび分散問合せ

SQL 文が 1 つ以上のリモート表を参照する場合に、オプティマイザはすべてのリモート表が同じサイトにあるかどうかを最初に判断します。すべての表が同じリモート・サイトにある場合、Oracle は問合せを実行するために問合せ全体をリモート・サイトに送信します。リモート・サイトは、結果の行をローカル・サイトに戻します。これは、リモート SQL 文と呼ばれます。表が複数のサイトにある場合、オプティマイザは各リモート表にアクセスするために、問合せを別々の SQL 文に分解します。これは、分散 SQL 文と呼ばれます。問合せが実行されるサイトは "駆動サイト" と呼ばれます。通常、これはローカル・サイトです。

この項では、次のトピックについて説明します。

- [リモート・データ・ディクショナリ情報](#)
- [リモート SQL 文](#)
- [分散 SQL 文](#)
- [EXPLAIN PLAN と SQL の分解](#)

## ■ パーティション・ビュー

# リモート・データ・ディクショナリ情報

SQL 文が複数の表を参照する場合、オブティマイザは、SQL 文を分解する前にどの列がどの表に属するかを判断する必要があります。たとえば、次のような問合せがあるとします。

```
SELECT DNAME, ENAME
FROM DEPT, EMP@REMOTE
WHERE DEPT.DEPTNO = EMP.DEPTNO
```

上記の例では、オブティマイザはまず最初に DNAME 列が DEPT 表に属し、ENAME 列が EMP 表に属することを判断する必要があります。オブティマイザは、すべてのリモート表のデータ・ディクショナリ情報を取得した後で、分解された SQL 文を作成できます。

分解された SQL 文では、列名と表名が二重引用符で囲まれます。特殊文字または予約語、空白を含む列名と表名は、二重引用符で囲む必要があります。

また、この機構は、選択リスト内のアスタリスク (\*) を実際の列名で置き換えます。次に例を示します。

```
SELECT * FROM DEPT@REMOTE;
```

上記の例は、次のように分解された SQL 文となります。

```
SELECT A1."DEPTNO", A1."DNAME", A1."LOC" FROM "DEPT" A1;
```

---

---

**注意：** 単純化するために、この章の残りの部分では二重引用符を使いません。

---

---

# リモート SQL 文

SQL 文全体がリモート・データベースに送信される場合、オブティマイザは、名前の競合を避けるために、問合せ内のすべての表と列に関して表の別名 A1、A2 などを使用します。次に例を示します。

```
SELECT DNAME, ENAME
FROM DEPT@REMOTE, EMP@REMOTE
WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

上記の例がリモート・データベースに送信されるときは、次のようになります。

```
SELECT A2.DNAME, A1.ENAME
FROM DEPT A2, EMP A1
WHERE A1.DEPTNO = A2.DEPTNO;
```



## 分散 SQL 文

問合せが 1 つ以上のデータベース上のデータにアクセスするときは、1 つのサイトが問合せの実行を " 駆動 " します。このサイトは " 駆動サイト " と呼ばれます。データはこのサイトで結合およびグループ化、順序付けされます。デフォルトでは、ローカルの Oracle サーバーが駆動サイトになります。DRIVING\_SITE というヒントを使用すると、手動で駆動サイトを指定できます。

SQL 文の分解は重要です。これによってレコード数が決まり、ネットワーク上で送信する必要のある表までも決まるからです。オプティマイザが SQL 文を分解する方法を知ておくと、分散問合せのパフォーマンスを最適化するのに役立ちます。

SQL 文が 1 つ以上のリモート表を参照する場合に、オプティマイザは、異なるデータベース上で実行される別々の問合せに SQL 文を分解する必要があります。次に例を示します。

```
SELECT DNAME, ENAME
FROM DEPT, EMP@REMOTE
WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

上記の例は、次のように分解されます。

```
SELECT DEPTNO, DNAME FROM DEPT;
```

上記の例はローカルに実行されます。

```
SELECT DEPTNO, ENAME FROM EMP;
```

上記の例は、リモート・データベースに送信されます。両方の表のデータはローカルで結合されます。これはすべて自動的、かつユーザーまたはアプリケーションに対して透過的に行われます。

ただし、ローカル表をリモート・データベースに送信し、2 つの表をリモート・データベース上で結合する方がよい場合もあります。これを行うには、ビューを作成するか、DRIVING\_SITE ヒントを使用します。リモート・データベース上にビューを作成することにした場合は、リモート・データベースからローカル・データベースへのデータベース・リンクも必要です。

次に例を示します (リモート・データベース上)。

```
CREATE VIEW DEPT_EMP AS
SELECT DNAME, ENAME
FROM DEPT@LOCAL, EMP
WHERE DEPT.DEPTNO = EMP.DEPTNO;
```

次に、ローカル表とリモート表のかわりにリモート・ビューから選択します。

```
SELECT * FROM DEPT_EMP@REMOTE;
```

これで、ローカル DEPT 表がネットワークを介してリモート・データベースに送信され、リモート・データベース上で EMP 表と結合されて、その結果がローカル・データベースに戻されます。

**関連項目：** このヒントの詳細は、7-49 ページの「[DRIVING\\_SITE](#)」を参照してください。

### ルールベースの最適化

ルールベースの最適化は、リモート表の索引に関する情報を持っていません。したがって、ルールベースの最適化が、ローカル表を結合の外部表とするネスト・ループ・ジョインをローカル表とリモート表の間に生成することはありません。ルールベースの最適化は、ローカル表で利用できる索引に応じて、リモート表を外部表とするネスト・ループ・ジョイン、またはソート / マージ結合を使用します。

### コストベースの最適化

コストベースの最適化は、ルールベースの最適化の場合よりも多くの実行計画を考慮できます。コストベースの最適化は、リモート表の索引が使用可能であるかどうか、およびそれを使う意味があるのはどのような場合かを認識しています。コストベースの最適化は、全表走査だけでなくリモート表の索引アクセスも考慮しますが、ルールベースの最適化は全表走査だけを考慮します。

コストベースの最適化で選択される特定の実行計画と表アクセスは、表と索引の統計によって異なります。次に例を示します。

```
SELECT DNAME, ENAME
FROM DEPT, EMP@REMOTE
WHERE DEPT.DEPTNO = EMP.DEPTNO
```

上記の例では、オブティマイザはローカル DEPT 表を駆動表として選択し、索引を使用してリモート EMP 表にアクセスできます。この場合に、分解された SQL 文は次のようになります。

```
SELECT ENAME FROM EMP WHERE DEPTNO = :1
```

この分解された SQL 文は、ネスト・ループ操作に使われます。

### ビューの使用法

表が複数のリモート・サイト上にある場合は、DRIVING\_SITE ヒントを使うよりもビューを作成する方が効果的です。すべての表が同じリモート・データベース上にあるとは限らない場合、オブティマイザは各リモート表に個別にアクセスします。次に例を示します。

```
SELECT D.DNAME, E1.ENAME, E2.JOB
FROM DEPT D, EMP@REMOTE E1, EMP@REMOTE E2
WHERE D.DEPTNO = E1.DEPTNO
AND E1.MGR = E2.EMPNO;
```

上記の例は、次のように分解された SQL 文となります。

```
SELECT EMPNO, ENAME FROM EMP;
```

および

```
SELECT ENAME, MGR, DEPTNO FROM EMP;
```

2 つの EMP 表をリモートで結合する場合は、ビューを作成してこれを実行できます。リモート・データベース上でリモート表を結合したビューを作成します。次に例を示します（リモート・データベース上）。

```
CREATE VIEW EMPS AS
SELECT E1.DEPTNO, E1.ENAME, E2.JOB
FROM EMP E1, EMP E2
WHERE E1.MGR = E2.EMPNO;
```

次に、リモート表のかわりにリモート・ビューから選択します。

```
SELECT D.DNAME, E.ENAME, E.JOB
FROM DEPT D, EMPS@REMOTE E
WHERE D.DEPTNO = E.DEPTNO;
```

上記の例は、次のように分解された SQL 文となります。

```
SELECT DEPTNO, ENAME, JOB FROM EMPS;
```

## ヒントの使用

分散問合せでは、すべてのヒントがローカル表でサポートされます。しかし、リモート表では、結合順序ヒントと結合操作ヒントのみを使用できます（アクセス方法、パラレル・ヒントなどのヒントは効果がありません）。リモートで行われる問合せでは、すべてのヒントがサポートされます。

**関連項目：** 7-46 ページの「[結合順序のヒント](#)」および 7-47 ページの「[結合操作のヒント](#)」

## EXPLAIN PLAN と SQL の分解

EXPLAIN PLAN は、SQL 文の実行計画全体に関する情報だけでなく、オプティマイザが SQL 文を分解する方法に関する情報も提供します。EXPLAIN PLAN は、PLAN\_TABLE 表に情報を保存します。SQL 文でリモート表が使用されている場合は、リモート表が参照されていることを示す値 REMOTE が OPERATION 列に格納され、リモート・データベースに送信される分解された SQL 文が OTHER 列に格納されます。次に例を示します。

```
EXPLAIN PLAN FOR SELECT DNAME FROM DEPT@REMOTE
SELECT OPERATION, OTHER FROM PLAN_TABLE
```

---

```
OPERATION OTHER
```

---

```
REMOTE      SELECT A1."DNAME" FROM "DEPT" A1
```

表の別名、および列名と表名を囲む二重引用符に注意してください。

**関連項目：** [第 13 章「EXPLAIN PLAN の使用方法」](#)。

## パーティション・ビュー

パーティション・ビューを使用すると、構造が同じで、異なるパーティションのデータを含む表を連結できます。これは、分散データベースで非常に有効です。分散データベースでは、各パーティションが 1 つのデータベースにあり、各パーティションのデータは共通の位置のプロパティを持っています。

そのようなパーティション・ビューに対して問合せが実行され、ビューのパーティションのサブセットとなる結果セットがその問合せに含まれるとき、問合せで必要ではないパーティションをスキップする計画をオブティマイザが選択します。このパーティション絞込みは、実行計画がすべてのパーティションを参照する場合に、実行時に発生します。

### 使用方法のルール

この項では、UNION ALL ビューを使ってオブティマイザでパーティションをスキップするときの状況について説明します。パーティション・ビューを含む Oracle サーバーは、次のルールに準拠する必要があります。

- PARTITION\_VIEW\_ENABLED 初期化パラメータが TRUE に設定されている
- コストベース・オブティマイザが使われている

---

**注意：** コストベース・オブティマイザを使用するには、UNION ALL ビューで使用されるすべての表を分析する必要があります。または、ヒントを使用するか、パラメータ OPTIMIZER\_MODE を ALL\_ROWS または FIRST\_ROW に設定することもできます。OPTIMIZER\_MODE または PARTITION\_VIEW\_ENABLED を設定するには、ALTER SESSION 文を使用することもできます。

---

UNION ALL ビューには複数の SELECT 文が含まれており、それぞれが " ブランチ " と呼ばれます。UNION ALL ビューで定義する各 SELECT 文が次のルールに準拠する場合、UNION ALL ビューはパーティション・ビューになります。

- ブランチの FROM 句に 1 つだけ表がある。
- ブランチに含まれる WHERE 句で、ビューに含まれるパーティションからデータのサブセットが定義されている。

- 副問合せ、group by、集計関数、distinct、rownum、connect by/start with を含む WHERE 句がブランチで使われていない。
- 各ブランチの選択リストが "\*" (または "\*" を明示的に指定したもの) である。
- UNION ALL ビューのすべてのブランチの列名と列のデータ型が、まったく同じである。
- ブランチで使われるすべての表が、同一の列に索引を持ち (索引がある場合) 列数も同じでなければならない。

パーティション絞込みは、定数述語の列移行性に基づきます。パーティション・ビューにアクセスする問合せで使われる WHERE 句は、UNION ALL ビュー定義におけるそれぞれのブランチの WHERE 句になります。次の例を見てください。

```
SELECT * FROM EMP_VIEW WHERE deptno=30;
```

このとき、ビュー EMP\_VIEW は次のように定義されています。

```
SELECT * FROM EMP@d10 WHERE deptno=10
UNION ALL
SELECT * FROM EMP@d20 WHERE deptno=20
UNION ALL
SELECT * FROM EMP@d30 WHERE deptno=30
UNION ALL
SELECT * FROM EMP@d40 WHERE deptno=40
```

この問合せで使われている "WHERE deptno=30" という述語は、UNION ALL ビューのそれぞれの問合せになります。"WHERE deptno=10 and deptno=30" のような WHERE 句に対しては、オプティマイザが移行性ルールを適用し、"10=30" という別の述語が生成されます。この新たな述語は常に偽であるため、その表 (EMP@d10) にアクセスする必要はありません。

移行性は、次のルールに準拠する述語に適用されます。

- 各ブランチの WHERE 句の述語が次の形式である。

```
RELATION AND RELATION ...
```

ここで、relation の形式は次のとおりです。

```
COLUMN_NAME RELOP CONSTANT_EXPRESSION
```

relop は、=、!=、>、>=、<、<= のいずれかです。

---

**注意：** BETWEEN ... AND はこれらのルールによって許可されていますが、IN は許可されていません。

---

- ビューを参照する問合せの少なくとも 1 つの述語が同一の形式で存在する。

## EXPLAIN PLAN の出力

システムがパーティション・ビューを認識していることを確認するためには、EXPLAIN PLAN の出力を調べます。問合せがパーティション・ビューに対して実行された場合、次の操作が EXPLAIN PLAN 出力の OPERATIONS 列に入ります。

VIEW	このエントリには、COST 列のオプティマイザ・コストも含まれます。
UNION-ALL	このエントリは、OPTION 列に PARTITION を指定します。
FILTER	操作が UNION-ALL 操作の子である場合、常に偽になる定数述語が生成されたことが FILTER によって示されます。パーティションは絞り込まれます。

UNION-ALL 操作の option 列に PARTITION がない場合、パーティション・ビューは認識されておらず、パーティションの絞込みは行われません。UNION ALL ビューが、8-6 ページの「[使用方法のルール](#)」で定義されているルールを遵守していることを確認してください。

## パーティション・ビューの例

次の例で、2 つのパーティションに分けられているパーティション・ビュー CUSTOMER を示します。EAST データベースには East Coast (東海岸) の顧客が含まれ、WEST データベースには West Coast (西海岸) の顧客が含まれます。

WEST データベースには次の表 CUSTOMER\_WEST が含まれます。

```
CREATE TABLE CUSTOMER_WEST
( cust_no    NUMBER CONSTRAINT CUSTOMER_WEST_PK PRIMARY KEY,
  cname      VARCHAR2(10),
  location   VARCHAR2(10)
);
```

EAST データベースにはデータベース CUSTOMER\_EAST が含まれます。

```
CREATE TABLE CUSTOMER_EAST
( cust_no    NUMBER CONSTRAINT CUSTOMER_EAST_PK PRIMARY KEY,
  cname      VARCHAR2(10),
  location   VARCHAR2(10)
);
```

EAST データベースでは次のパーティション・ビューが作成されます (WEST データベースでも同様のビューを作成できます)。

```
CREATE VIEW customer AS
SELECT * FROM CUSTOMER_EAST
```

```
WHERE location='EAST'
UNION ALL
SELECT * FROM CUSTOMER_WEST@WEST
WHERE location='WEST';
```

次の文を実行しても、WEST データベースの CUSTOMER\_WEST 表はアクセスされないことに注意してください。

```
EXPLAIN PLAN FOR SELECT * FROM CUSTOMER WHERE location='EAST';
```

---

**注意：** EAST データベースでは、依然として CUSTOMER\_WEST 表の列名と列のデータ型の情報が必要です。そのため、WEST データベースとの接続が依然として必要です。また、コストベース・オブティマイザを使用する必要があります。たとえば、ALTER SESSION SET OPTIMIZER\_MODE=ALL\_ROWS という文を発行することによってこれを実現できます。

---

EXPLAIN PLAN の出力で示したように、オブティマイザは CUSTOMER\_WEST パーティションがアクセスされる必要がないことを認識します。

```
SELECT LPAD(' ',LEVEL*3-3) || OPERATION OPERATION,COST,OPTIONS,
OBJECT_NODE, OTHER
FROM PLAN_TABLE
CONNECT BY PARENT_ID = PRIOR ID
START WITH PARENT_ID IS NULL
```

OPERATION	COST	OPTIONS	OBJECT_NOD	OTHER
-----	-----	-----	-----	-----
SELECT STATEMENT	1			
VIEW	1			
UNION-ALL		PARTITION		
TABLE ACCESS	1	FULL		
FILTER				
REMOTE	1		WEST.WORLD	SELECT "CUST_NO", "CNAME", "LOCATION" FROM "CUSTOMER _WEST" "CUSTOMER_WEST" WH ERE "LOCATION"='EAST' AND "LOCATION"='WEST'

## 分散問合せの制限事項

同じバージョンの Oracle 内で行う分散問合せには、次の制限事項があります。

- 分散問合せにはコストベースの最適化を使う必要があります。ルールベースの最適化は、表が等価結合で結合された場合には、リモート表とローカル表の間にネスト・ループ・ジョインを生成しません。
- コストベースの最適化では、問合せ計画を生成するときに、1 つのリモート表あたりで 20 を超える索引は考慮されません。索引の順序は場合によって異なります。索引が 20 の制限を超える場合は、問合せ計画内でランダムに変化します。
- リモート表の逆キー索引は、オブティマイザでは利用できません。そのため、逆キー索引しか持たない列を使う等価結合がある場合は、ネスト・ループ・ジョインをリモート表に使えません。
- コストベースの最適化は、リモートのオブジェクトがパーティション化されていることを認識できません。そのため、オブティマイザは、リモートのパーティション・オブジェクトに対して最適でない計画を生成する可能性があります。特にパーティション・ブルニングが可能な場合には、オブジェクトをローカルにしてください。
- リモート・ビューはマージされず、オブティマイザはリモート・ビューの統計を持ちません。優れた問合せ計画を得るためには、すべてのマージ可能なビューはすべてのサイトでレプリケートするのが最善です（次の例外を参照してください）。
- コストベースの最適化もルールベースの最適化も、結合をリモートで実行できません。すべての結合は、それを駆動したサイトで実行されます。このことは、選択リスト内のすべての表がリモートである場合に、CREATE TABLE ... AS SELECT のパフォーマンスに影響します。この場合は、リモートのサイトに SELECT 文のためのビューを作成する必要があります。

## Transparent Gateway

Transparent Gateway は、他のデータ・ソース（リレーショナル・データベース、階層データベース、ファイル・システムなど）のデータにアクセスするために使用します。

Transparent Gateway は、Oracle 以外のシステムのデータに、それがまるで別の Oracle データベースであるかのように透過的にアクセスする手段を提供します。

### 異機種間分散 SQL 文の最適化

SQL 文で Oracle 以外のシステムのデータにアクセスするとき、その SQL 文は異機種間分散 SQL 文と呼ばれます。異機種間分散 SQL 文を最適化するには、Oracle データベースのみにアクセスする分散 SQL 文の最適化の場合と同様のガイドラインに従います。ただし、通常、Oracle 以外のシステムでは、Oracle8i がサポートしている関数と演算子をすべてサポートしているわけではないことを考慮に入れる必要があります。したがって、Transparent Gateway は、これらのデータ・ソースがどの関数と演算子をサポートしているかを（接続時に）Oracle に示します。他のデータ・ソースが関数または演算子をサポートしていない場



合、Oracle によってその関数または演算子が実行されます。この場合に、Oracle は他のデータ・ソースからデータを取得して、関数または演算子をローカルに適用します。これは、SQL 文の分解方法に影響し、特に Oracle が他のデータ・ソースと同じマシン上にない場合にはパフォーマンスにも影響する可能性があります。

## ゲートウェイとパーティション・ビュー

Oracle Transparent Gateways バージョン 8 以降ではパーティション・ビューと一緒に使えます。8-6 ページの「[使用方法のルール](#)」で定義されているルールに準拠していることを確認してください。特に次の点に注意します。

- コストベース・オプティマイザを使う必要があります（ヒントを使うか、パラメータ OPTIMIZER\_MODE を ALL\_ROWS または FIRST\_ROWS に設定する）。
- 各パーティションで使われる索引が同一でなければなりません。使用しているゲートウェイ固有のインストレーションおよびユーザズ・ガイドを参照して、そのゲートウェイが Oracle 以外のシステムの索引情報を Oracle Server に送信するかどうかを調べてください。ゲートウェイが索引情報をオプティマイザに送信する場合は、各パーティションで使われる索引数が同一であること、および同一の列に索引を付けていることを確認してください。ゲートウェイが索引情報を送信しない場合は、Oracle オプティマイザはパーティションの索引を認識しません。そのため、Oracle 以外のシステムの各パーティションで使われる索引は、同一であるとみなされます。1 つのパーティションが Oracle サーバーにある場合は、そのパーティションで索引を定義することができます。
- UNION ALL ビューのすべてのブランチの列名と列のデータ型が、同一である必要があります。Oracle 以外のシステムのデータ型は、Oracle のデータ型にマップされます。Oracle 以外のさまざまなシステムにある各パーティションのデータ型がすべて、同一の Oracle のデータ型にマップされることを確認してください。データ型がどのように Oracle のデータ型にマップされるかを調べるには、SQL\*Plus または Server Manager で DESCRIBE コマンドを実行してください。

## まとめ：分散問合せのパフォーマンスの最適化

分散問合せのパフォーマンスを改善するにはいくつかの方法があります。

- 最適な SQL 文の選択。  
多くの場合、同じ結果を生成する SQL 文が複数あります。すべての表が同じデータベース上にある場合、これらの SQL 文の間のパフォーマンスの違いは最小限ですが、表がさまざまなデータベース上にある場合は、パフォーマンスの違いが大きくなる場合があります。
- コストベースの最適化の使用。  
コストベースの最適化は、リモート表で索引を使用し、ルールベースの最適化よりも多くの実行計画を考慮でき、一般によりよい結果をもたらします。コストベースの最適化

による分散問合せのパフォーマンスは一般に満足のいくものです。SQL 文の変更またはビューの作成、プロシージャ型コードの使用が必要になることはめったにありません。

- ビューを使用する。

状況によっては、ビューを使って分散問合せのパフォーマンスを改善できることもあります。たとえば、次のような場合です。

- リモート・データベース上で複数のリモート表を結合する場合
- ネットワークを介して異なる表を送信する場合
- プロシージャ・コードを使用する。

PL/SQL プロシージャやプリコンパイラ・プログラムなどのプロシージャ・コードで分散問合せを置き換える方が効率的なことがまれにあります。このオプションについてここで述べるのは、頻繁に必要なからではなく、すべての可能性について言及するためです。

---

## トランザクション・モード

この章では、読み込み一貫性を保つためのさまざまなモードを説明します。トピックは次のとおりです。

- ディスクリット・トランザクションの使用方法
- 直列可能トランザクションの使用方法

### ディスクリット・トランザクションの使用方法

BEGIN\_DISCRETE\_TRANSACTION プロシージャを使うことによって、短い非分散トランザクションのパフォーマンスを改善することができます。このプロシージャによってトランザクション処理を合理化するため、短いトランザクションがより高速に実行できるようになります。この項では、次のトピックについて説明します。

- ディスクリット・トランザクションの使用時期の決定
- ディスクリット・トランザクションの処理方法
- ディスクリット・トランザクションの処理中に発生するエラー
- 使用上の注意
- 例

### ディスクリット・トランザクションの使用時期の決定

ディスクリット・トランザクション処理は、次のようなトランザクションに有効です。

- ごく少数のデータベース・ブロックを修正する。
- 個々のデータベース・ブロックの変更をトランザクションあたり 2 回以上実行しない。
- 長時間実行問合せによって要求されるデータは修正しない。
- 修正した後のデータの新しい値を参照する必要がない。

- LONG 値を含んでいる表は修正しない。

ディスクリート・トランザクションの使用を判断する上で、次の要因を検討してください。

- 9-3 ページの「[使用上の注意](#)」で説明するように、トランザクションが、ディスクリート・トランザクションに課される制約の範囲内で動作するように設計できるか。
- 一般的な使用条件の下で、ディスクリート・トランザクションを使うことによって、大幅なパフォーマンスの改善が可能か。

ディスクリート・トランザクションは、標準のトランザクションと同時に使用できます。通常の手順の一部として、ディスクリート・トランザクションを使用するかどうかを選択する必要があります。高度なアプリケーション要件を持つ経験の豊富なユーザーにとっては、ディスクリート・トランザクションは全トランザクションのサブセットとして使用されるだけですが、速度が最も重大な要因である場合は、パフォーマンス改善によって設計の制約が正当化されます。

## ディスクリート・トランザクションの処理方法

ディスクリート・トランザクションでは、データに加えた変更はすべてトランザクションのコミットまで延期されます。REDO 情報は生成されますが、メモリー内の別々の位置に格納されます。

トランザクションがコミット要求を発行すると、REDO 情報が（他のグループ・コミットとともに）REDO ログ・ファイルに書き込まれ、データ・ブロックへの変更がブロックに直接適用されます。ブロックは、通常の方法でデータ・ファイルに書き込まれます。一度コミットが完了すると、制御はアプリケーションに戻されます。このように、トランザクションがコミットされるまではブロックは実際には修正されないの、ロールバック情報を生成する必要がなくなり、REDO 情報は REDO ログ・バッファに格納されます。

他のトランザクションと同じように、ディスクリート・トランザクションのコミットされていない変更は、同時実行のトランザクションから参照できません。通常のトランザクションの場合、データの一貫したビューを必要とする問合せでは、ロールバック情報を使って古いバージョンのデータを再作成します。ディスクリート・トランザクションではロールバック情報が生成されないの、長い問合せの間に開始し、終了するようなディスクリート・トランザクションを行った場合、その長い問合せがディスクリート・トランザクションによって変更されたデータを要求すると、"snapshot too old" というエラーを受け取る可能性があります。よって、ディスクリート・トランザクションによって頻繁に修正される表（の大部分）をアクセスする問合せは実行しないでください。

## ディスクリート・トランザクションの処理中に発生するエラー

ディスクリート・トランザクションの処理中に発生するエラーによって、事前定義された例外 `DISCRETE_TRANSACTION_FAILED` が発生します。これらのエラーには、下記の使用上の注意に示されているようなディスクリート・トランザクションの障害が含まれます（たとえば、トランザクションの開始後に `BEGIN_DISCRETE_TRANSACTION` を呼び出した場

合、またはトランザクションで同じデータベース・ブロックを複数回修正しようとした場合に、例外が発生します)。

## 使用上の注意

BEGIN\_DISCRETE\_TRANSACTION プロシージャは、トランザクションの最初の文よりも前にコールする必要があります。このプロシージャの呼出しは、トランザクションの存続期間中だけ効果があります(つまり、トランザクションがコミットまたはロールバックされると、その次のトランザクションは標準のトランザクションとして処理されます)。

このプロシージャを使用するトランザクションは、分散トランザクションに含むことはできません。

ディスクリット・トランザクションでは、それ自身が加えた変更を参照することはできません。しかし、値を更新する前に、SELECT 文の FOR UPDATE 句を使って古い値を取得し、行をロックすることはできます。

ディスクリット・トランザクションでは、変更を参照できないため、参照整合性制約にかかわる 2 つの表に対して挿入や更新を実行することはできません。

たとえば、EMP 表には、DEPT 表を参照する FOREIGN KEY 制約が DEPTNO 列にあります。ディスクリット・トランザクションでは、DEPT 表に部門を追加してから、その部門に所属する従業員を追加することはできません。トランザクションがコミットされるまで部門は表に追加されないのに、整合性制約では EMP 表への挿入が行われる前にその部門が存在していることが要求されるからです。これらの 2 つの操作は、別々のディスクリット・トランザクションで実行する必要があります。

ディスクリット・トランザクションは、各データ・ブロックを一度しか変更できないため、同一の表に対する DML 文の特定の組合せは他の組合せよりもディスクリット・トランザクションに適しています。ともに使用される 1 つの INSERT 文と 1 つの UPDATE 文の組合せが同じブロックを処理する可能性は最も低いでしょう。処理の対象となる表のサイズに依存しますが、複数の UPDATE 文も同じブロックを更新する可能性は低いでしょう。しかし、複数の INSERT 文(または値を指定するために問合せを使う INSERT)は、多くの場合同じデータ・ブロックに挿入します。表がクラスタ化されていない限り、別々の表に対して実行される複数の DML 操作が同じデータ・ブロックを処理することはありません。

## 例

図書館の蔵書貸出しのアプリケーションは、BEGIN\_DISCRETE\_TRANSACTION プロシージャを使うトランザクションの例です。次のプロシージャは、蔵書番号を引数とする図書館のアプリケーションによって呼び出されます。このプロシージャは、貸出しを許可する前にその蔵書が予約されているかどうか検査します。利用できる蔵書よりも多くが予約されている場合、必要に応じて、本を予約するために別のプロシージャを呼び出す図書館アプリケーションに状態 RES が戻されます。そうでなければ、その蔵書は貸し出され、利用できる蔵書目録が更新されます。

```
CREATE PROCEDURE checkout (bookno IN NUMBER (10)
                           status OUT VARCHAR(5))
AS
DECLARE
    tot_books    NUMBER(3);
    checked_out  NUMBER(3);
    res          NUMBER(3);
BEGIN
    dbms_transaction.begin_discrete_transaction;
    FOR i IN 1 .. 2 LOOP
        BEGIN
            SELECT total, num_out, num_res
            INTO tot_books, checked_out, res
            FROM books
            WHERE book_num = bookno
            FOR UPDATE;
            IF res >= (tot_books - checked_out)
            THEN
                status := 'RES';
            ELSE
                UPDATE books SET num_out = checked_out + 1
                WHERE book_num = bookno;
                status := 'AVAIL'
            ENDIF;
            COMMIT;
            EXIT;
        EXCEPTION
            WHEN dbms_transaction.discrete_transaction_failed THEN
                ROLLBACK;
            END;
    END LOOP;
END;
```

上のループ構造では、DISCRETE\_TRANSACTION\_FAILED 例外がトランザクションで発生すると、トランザクションはロールバックされ、ループによって再びトランザクションが実行されます。ROLLBACK 文がトランザクションを終了すると、次のトランザクションは標準のトランザクションとして処理を行うため、2 番めのループはディスクリート・トランザクションではありません。このループ構造は、ディスクリート・トランザクションで障害が発生すると同じトランザクションが再実行されることを保証します。

## 直列可能トランザクションの使用法

Oracle では、アプリケーションの開発者がトランザクション分離レベルを設定することができます。分離レベルによって、あるトランザクションが認識できる変更とその他のトランザクションが認識できる変更が決まります。ISO/ANSI SQL3 仕様では、次のトランザクション分離レベルが規定されています。

### SERIALIZABLE

トランザクションは必ず更新を認識し、リピータブル・リードが可能で、ファントムを経験することはありません。直列可能トランザクションで行われた変更は、そのトランザクションにしか見えません。

### READ COMMITTED

トランザクションではリピータブル・リードを行うことはできず、このトランザクションまたは他のトランザクションの中で行われた変更はすべてのトランザクションによって認識されます。これは、トランザクション分離レベルのデフォルト値です。

トランザクション分離レベルを設定する場合は、トランザクションを開始する前に行う必要があります。特定のトランザクションに対しては SET TRANSACTION ISOLATION LEVEL 文を、セッション内のすべての後続トランザクションに対しては ALTER SESSION SET ISOLATION\_LEVEL 文を使用します。

**関連項目：** SET TRANSACTION と ALTER SESSION の構文の詳細は、『Oracle8i SQL リファレンス』を参照してください。





---

## SQL および共有 PL/SQL 領域の管理

Oracle は、DDL 文が内部的に発行した再帰的 SQL 文と同様に、ユーザーおよびアプリケーションから直接発行された SQL 文や PL/SQL ブロックを比較します。同一の文が 2 つ発行されると、最初の文を処理するために使用された SQL 領域または PL/SQL 領域が共有されます。つまり、同じ文をもう一度実行する際に再利用されます。

共有 SQL 領域および共有 PL/SQL 領域は、共有メモリー領域です。Oracle プロセスはすべて共有 SQL 領域を使用できます。共有 SQL 領域を使用することによって、データベース・サーバー上のメモリーの使用量が減少し、それによって、システムのスループットが向上します。

共有 SQL 領域と共有 PL/SQL 領域は、古くなると "最低使用頻度" (LRU) アルゴリズムによって共有プールから除去されます (これはデータベース・バッファの場合と似ています)。パフォーマンスを改善したり、再解析が行われないようにするために、サイズの大きい SQL 領域または PL/SQL 領域が古くなって共有プールから除去されない処置を考えることがあるかもしれません。

この章では、共有 SQL を使ってパフォーマンスを改善する方法を説明します。トピックは次のとおりです。

- [SQL 文と PL/SQL ブロックの比較](#)
- [共有プールでの共有 SQL と PL/SQL の維持](#)

## SQL 文と PL/SQL ブロックの比較

この項では、次のトピックについて説明します。

- 同一の SQL 文かどうかのテスト
- 標準化された SQL フォーマットについて

### 同一の SQL 文かどうかのテスト

Oracle は、2 つ以上のアプリケーションが、データベースに同一の SQL 文または PL/SQL ブロックを送ってこないかチェックしています。ある文が現在共有プールに入っている別の文と同一である場合に、文を解析する必要はありません。Oracle は、次のステップに従って同一の文を識別します。

1. 発行された文のテキスト文字列は、ハッシュされます。ハッシュ値が、共有プール内の既存の SQL 文に対するハッシュ値と同じであれば、Oracle はステップ 2 へ進みます。
2. 発行された文のテキスト文字列は、大文字小文字の違い、空白およびコメントを含め、ステップ 1 で識別された既存のすべての SQL 文と比較されます。
3. 発行された文で参照されるオブジェクトは、ステップ 2 で識別された既存のすべての文の参照オブジェクトと比較されます。たとえば、2 人のユーザーが EMP 表を持っている場合、次の文を考えます。

```
SELECT * FROM emp;
```

この場合、文は各ユーザーに対して異なる表を参照するので、同一とはみなされません。

4. SQL 文で使われるバインド変数のバインドの種類は、一致する必要があります。

---

**注意：** ほとんどの Oracle 製品は、文をデータベースに渡す前に SQL を変換します。首尾一貫した SQL 文の集合が生成されるように、文字は大文字に統一して変換され、空白は圧縮され、バインド変数は改名されます。

---

## 標準化された SQL フォーマットについて

アプリケーションのすべてのユーザーに標準化された方法で SQL 文を記述させることは、必要でも有益でもありません。300 名のユーザーが標準化された SQL を使って非定型で動的な文を作成すると、同じ SQL 文が生成されることはないでしょう。300 名が、そのときどきで、まったく同じ表のまったく同じ列をまったく同じ順序で参照する可能性はほとんどありません。それとは対照的に、300 名のユーザーが同じアプリケーション（コマンド・ファイル）を実行しているときは、同一の SQL 文が生成されます。

アプリケーション内に、ほぼ同じ 2 つの文があり、300 名のユーザーがそれらを使う場合には、ごくわずかの利点しかありませんが、1 つの文を 600 名のユーザーが使う場合には大きな利点があります。

## 共有プールでの共有 SQL と PL/SQL の維持

この項では、共有 SQL および PL/SQL を共有プールに維持する 2 つの手法を説明します。

- [大きな割当て用の領域の予約](#)
- [オブジェクト除去の防止](#)

### 大きな割当て用の領域の予約

ユーザーが共有プールを一杯にした後で大きなパッケージが除去されると、問題が発生する可能性があります。その後、誰かがその大きなパッケージを呼び戻す必要がある場合は、その領域を共有プールに作成するために莫大な量のメンテナンスを行う必要があります。この問題は、SHARED\_POOL\_RESERVED\_SIZE 初期化パラメータによって大きな割当て用の領域を予約することで避けられます。このパラメータは、SHARED\_POOL\_RESERVED\_SIZE\_MIN\_ALLOC パラメータで指定された値よりも大きな空間の割当てを共有プールに予約します。

---

**注意：** Oracle は、大量の連続するメモリー領域の必要性を低減するためにセグメント化したコードを使用していますが、ラージ・オブジェクトをメモリー内に確保するとパフォーマンスが向上する場合があります。

---

### オブジェクト除去の防止

DBMS\_SHARED\_POOL パッケージを使用すると、オブジェクトが共有メモリー内に維持されるため、これらのオブジェクトは通常の LRU メカニズムによって除去されることはありません。DBMSPOOL.SQL および PRVTPPOOL.PLB プロシージャ・スクリプトは、DBMS\_SHARED\_POOL のパッケージ仕様部とパッケージ本体を作成します。

DBMS\_SHARED\_POOL パッケージを使用し、SQL 領域と PL/SQL 領域をメモリーの断片化が発生する前にロードすると、オブジェクトは通常の LRU メカニズムによって除去されることなく、メモリー内に維持されます。このプロシージャによって、メモリーが確実に使

用可能になり、SQL 領域と PL/SQL 領域の除去後にこれらの領域にアクセスする場合に、ユーザーの応答時間中に突然原因不明のスローダウンが発生するのを防ぐことができます。

## DBMS\_SHARED\_POOL の使用時期

DBMS\_SHARED\_POOL パッケージで提供されるプロシージャは、STANDARD パッケージや DIUTIL パッケージなどのサイズの大きな PL/SQL オブジェクトをロードする場合に役立ちます。

大きな PL/SQL オブジェクトがロードされると、領域を確保するために多数の小さなオブジェクトを共有プールから除去する必要があるため、ユーザーの応答時間に影響が現れます。これは、メモリー断片化が原因です。場合によっては、このような大きなオブジェクトをロードするには、メモリーが十分でないこともあります。

また、DBMS\_SHARED\_POOL は、頻繁に実行されるトリガーの場合にも便利です。頻繁に使う表のコンパイル済みトリガーを共有プール内に維持することができます。

さらに、DBMS\_SHARED\_POOL は順序もサポートしています。共有プールから順序が除去されると、順序番号が失われます。DBMS\_SHARED\_POOL は、共有プール内に順序を保持し、順序番号の消失を防ぐのに便利です。

## DBMS\_SHARED\_POOL の使用方法

DBMS\_SHARED\_POOL パッケージを使用して SQL 領域または PL/SQL 領域を確保するには、次のステップを実行してください。

1. メモリー内に確保しておくパッケージまたはカーソルを決定します。
2. データベースを起動します。
3. DBMS\_SHARED\_POOL.KEEP をコールして確保します。

この手順により、オブジェクトがロードされる前にシステムの共有メモリーが使い尽くされないことが保証されます。その結果、オブジェクトをインスタンスの早い時期に確保することにより、大きなメモリー領域を共有プールの中央に確保するために発生する可能性のある、メモリーの断片化を防ぐことができます。

DBMS\_SHARED\_POOL パッケージが提供するプロシージャについて次に説明します。

## DBMS\_SHARED\_POOL.SIZES

このプロシージャは、指定のサイズよりも大きい共有プール内のオブジェクトを示します。

```
DBMS_SHARED_POOL.SIZES(MINSIZE IN NUMBER)
```

### 入力パラメータ:

minsize	共有プール内にある、このサイズよりも大きなオブジェクトが表示されます。サイズの単位は KB (キロバイト) です。
---------	---

このプロシージャの実行結果を表示するには、プロシージャをコールする前に SQL\*Plus で次のコマンドを発行してください。

```
SET SERVEROUTPUT ON SIZE minsize
```

このコマンドの実行結果は、KEEP または UNKEEP プロシージャの引数として使用できます。

たとえば、共有プール内にある 2000 キロバイトを超えるオブジェクトを表示するには、次の SQL\*Plus コマンドを発行します。

```
SET SERVEROUTPUT ON SIZE 2000
EXECUTE DBMS_SHARED_POOL.SIZES(2000);
```

## DBMS\_SHARED\_POOL.KEEP

このプロシージャにより、共有プール内にオブジェクトを維持できます。

```
DBMS_SHARED_POOL.KEEP(OBJECT IN VARCHAR2,
[TYPE IN CHAR DEFAULT P])
```

### 入力パラメータ:

object	オブジェクトの名前、または共有プール内に維持するオブジェクトのカーソル・アドレスです。この値は、SIZES プロシージャをコールしたときに表示される値です。
type	共有プール内に維持するオブジェクトのタイプです。次のタイプがあります。
P	パッケージ / プロシージャ / ファンクション
C	カーソル
R	トリガー
Q	順序
T	タイプ
JS	Java ソース
JC	Java クラス
JR	Java リソース

ユーザー定義データ型を確保すると、LRU メカニズムでそれらが除去されないように、タイプの本体も仕様も共有メモリーに確保されます。タイプを共有メモリーに確保するには、タイプに対する実行権限が必要です。

たとえば、本体が存在しないタイプ 'TY' を確保しようとした場合は、タイプの仕様 TY だけが確保されます。TY のタイプ本体を作成したら、再度タイプを保持して 'TY' のタイプ本体も共有メモリーに確保する必要があります。

次の例では、ユーザー 'user2' が、共有メモリーに確保されている user1 のスキーマにタイプ 'TY' を保持する方法を示します。この例では、user1.TY に対する実行権限が user2 にあると想定しています。

```
BEGIN
  SYS.DBMS_SHARED_POOL.KEEP('user1.TY', 'T');
END;
```

---

**注意：** フラグは 'T' か 't' のどちらかです。

---

**DBMS\_SHARED\_POOL.UNKEEP**

このプロシージャを使用すると、共有プール内に確保したオブジェクトを共有プールから除去できます。

```
DBMS_SHARED_POOL.UNKEEP(OBJECT IN VARCHAR2,
  [TYPE IN CHAR DEFAULT 'P'])
```

**入力パラメータ：**

object	オブジェクトの名前、または共有プール内に維持する必要のなくなったオブジェクトのカーソル・アドレスです。この値は、SIZES プロシージャをコールしたときに表示される値です。
type	共有プールから除去するオブジェクトのタイプです。次のタイプがあります。
P	パッケージ
C	カーソル
R	トリガー
Q	順序
T	タイプ
JS	Java ソース
JC	Java クラス
JR	Java リソース

---

# データ・ウェアハウス・アプリケーションの最適化

この章では、企業規模のデータ・ウェアハウスのチューニングに使用する統合された Oracle の機能について説明します。システムおよびデータ・レイアウト、アプリケーションをインテリジェントにチューニングすることによって、パフォーマンスの高いスケーラブルなデータ・ウェアハウスを作成できます。

トピックは次のとおりです。

- [データ・ウェアハウス・アプリケーションの特徴](#)
- [データ・ウェアハウスの作成](#)
- [データ・ウェアハウスの問合せ](#)
- [データ・ウェアハウス・アプリケーションのチューニング](#)
- [データ・ウェアハウスのバックアップとリカバリ](#)

## データ・ウェアハウス・アプリケーションの特徴

データ・ウェアハウス・アプリケーションは、次のような CPU バウンドかつ I/O バウンドのデータ集中型タスクを多数使用して、大容量のデータを処理します。

- 表のロード、索引付けおよびサマリー
- データの走査、結合、ソート、集計およびフェッチ

データ・ウェアハウス・アプリケーションは、何 GB ものデータに対するタスクを実行するためにリソースを必要とするという点で、他のタイプのデータ処理と区別されます。データの量と複雑さによって、複数の CPU の導入またはパラレル処理の適用検討、タスクに直接関連する特定のデータ処理機能の考慮の必要性が明らかに示されることがあります。

たとえば、典型的なデータ・ウェアハウス・アプリケーションでは、データ集中処理型のタスクが 100GB 以上のデータに対して実行されることがあります。1CPU あたり 1 時間に 2 ~

30GB のデータ処理速度では、1 つの CPU でこのサイズのタスクを実行するのに数時間かかります。

したがって、数 GB (10G 以上であれば必ず) のデータを処理する場合は、CPU の数を増やすことを検討する必要があります。

同様に、10GB のデータをコピーする必要がある場合は、エクスポート / インポートを使用すると 1 つの CPU で数時間を要することを考慮してください。対照的に、10 個以上の CPU で CREATE TABLE... AS SELECT を並列して使用する場合は 1 時間もかかりません。

実際の処理時間は、問合せの複雑さや使用するハードウェア構成をチューニングすることによって達成できる処理速度などの多くの要因に依存します。特定の操作に関するパフォーマンス特性を見きわめるには、使用しているシステム上で簡単なテストを実行してください。

## データ・ウェアハウスの作成

この項では、データ・ウェアハウスの作成に有用な機能について概説します。データ・ウェアハウス機能には次のものがあります。

- [マテリアライズド・ビューとディメンション](#)
- [パラレル CREATE TABLE... AS SELECT](#)
- [索引のパラレル作成](#)
- [高速全索引走査](#)
- [パーティション表](#)
- [ANALYZE 文](#)
- [パラレル・ロード](#)
- [制約](#)

**関連項目：**『Oracle8i 概要』および『Oracle8i SQL リファレンス』

## マテリアライズド・ビューとディメンション

マテリアライズド・ビューは、事前に計算された結果を含む、問合せのサマリーが格納されたものです。マテリアライズド・ビューを使用すると、データ・ウェアハウスの問合せ処理が大幅に改善されます。ディメンションは、データ間の関係を説明するもので、より複雑なタイプのクエリー・リライトの実行に必要です。

**関連項目：** マテリアライズド・ビューの詳細は、[VI 項の「マテリアライズド・ビュー」](#)、『Oracle8i 概要』、『Oracle8i 管理者ガイド』、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。



## パラレル CREATE TABLE ... AS SELECT

CREATE TABLE ... AS SELECT 操作を並列して実行する機能によって、非常に大規模な表を効率よく再編成できます。数十 GB または数 TB のデータを格納する表で領域を再編成または再生するときに、シリアルのアプローチをとれないことがあります。エクスポート / インポートを使用してこのようなタスクを実行すると、ダウン時間が許容できないほど長くなる可能性があります。使用可能な一時領域が大量にある場合は、CREATE TABLE ... AS SELECT を使用してこのタスクをパラレルで実行できます。このアプローチでは、整合性制約の再定義はオプションです。この機能はデータ・ウェアハウス環境で、事前に計算された結果を格納するサマリー表の作成に使用されることがよくあります。

**関連項目：** 27-17 ページの「[パラレルでの表の作成と移入](#)」と『Oracle8i 概要』

## 索引のパラレル作成

並列して索引を作成する機能は、データ・ウェアハウス・アプリケーションと OLTP アプリケーションの両方に有用です。非常に大規模な表では、索引の再作成に非常に長い時間がかかることがあります。DBA は、定期的に大量のデータをロードして索引を再作成することがあります。索引をパラレルで作成できる機能によって、新しいデータをロードする前に索引を削除し、その後で索引を再作成することができます。

**関連項目：** 26-67 ページの「[パラレルでの索引の作成](#)」

## 高速全索引走査

既存の索引が、問合せで必要となるすべての列をすでに含んでいる場合、索引に対する FAST FULL SCAN (高速全走査) は、全表走査の代替としてより短時間で実行されます。これは表走査と同様に、マルチブロック I/O を使い、並列化できます。ヒント INDEX\_FFS を使用すると、高速全索引走査が施行されます。

**関連項目：** 6-8 ページの「[高速全索引走査の使用](#)」および 7-43 ページの「[INDEX\\_FFS](#)」

## パーティション表

非常に大規模な表や重要な役割を持っている表のダウン時間の発生は、パーティションを使用することで避けられます。大規模な表は、パーティション化機能を使って複数の物理的パーティションに分割できます。データ・ウェアハウスでは、日付によってパーティション化することで、履歴データを管理できます。さらに、通常は表レベルで実行するバックアップおよびリストアなどの操作を、すべてパーティション・レベルで行うことができます。新しいパーティションを追加することによって新しいデータ用のスペースを追加し、既存のパーティションを削除することによって古いデータを削除できます。キーの範囲を使ってパーティション表から選択する問合せは、その範囲に含まれるパーティションだけを検索対

象とします。このようにしてパーティションは、可用性および管理性、表走査のパフォーマンスを大幅に改善します。

---

---

**注意：** パフォーマンス上の理由から、Oracle ではパーティション・ビューではなくパーティション表を使用してください。パーティション・ビューからパーティション表への移行の手順は、『Oracle8i 移行ガイド』を参照してください。

---

---

**関連項目：** パーティション表の詳細は、26-43 ページの「[データのパーティション化](#)」と『Oracle8i 概要』を参照してください。

## ANALYZE 文

ANALYZE 文を使用して、表、索引およびクラスタの記憶特性を分析し、統計を収集してデータ・ディクショナリに格納できます。オプティマイザは、コストベースのアプローチにおいてこれらの統計を使用し、発行された SQL 文に対して最も効果的な実行計画を判断します。この目的に対して許容するオーバーヘッドの量に応じて、統計を計算または見積もることができます。

**関連項目：** 『Oracle8i 管理者ガイド』

## パラレル・ロード

非常に大量のデータをロードする場合、対象となる表が使えなくなる時間の長さが許容できないほど長くなることがあります。データを並列にロードする機能を使うと、そのダウン時間を大幅に短縮できます。

**関連項目：** SQL\*Loader の従来型パス・ロードおよびダイレクト・パス・ロードの詳細は、[第 26 章「パラレル実行のチューニング](#)」、特に 26-59 ページの「[フェーズ 3: データベースの作成、移入およびリフレッシュ](#)」、および『Oracle8i ユーティリティ・ガイド』を参照してください。

## 制約

VALIDATE 状態の制約は、DML 文、ロードおよび索引メンテナンスのパフォーマンスを低下させます。しかし、問合せの最適化の中には、VALIDATE 状態の制約の存在に依存するものがあります。

一連の制約を DISABLE NOVALIDATED 状態から ENABLE VALIDATED に最も速く変更する方法は、まずそれらをすべて ENABLE NOVALIDATE することです。それから、それらを個々に検査します。検査には長い時間がかかりますが、ENABLE NOVALIDATE が完了すれば、すべての表に対して問合せや変更ができます。ダイレクト・ロードでは、このようにして自動的に制約が再び使用可能にされます。

データ・ウェアハウスでは、DISABLE VALIDATE 状態が効果的な場合があります。この状態にすると、オプティマイザが一意キーまたは主キーの妥当性を認識しますが、索引は必要ありません。DISABLE VALIDATE 状態では、キーに対する挿入、更新および削除は禁止されます。

**関連項目：** 詳細は、『Oracle8i 管理者ガイド』の「整合性制約の管理」に関する章を参照してください。また、『Oracle8i SQL リファレンス』にも DISABLE VALIDATE キーワードと NOVALIDATE キーワードの説明があります。

## データ・ウェアハウスの問合せ

この項では、データ・ウェアハウスの問合せに有用な Oracle の機能についてまとめます。データ・ウェアハウス機能には次のものがあります。

- [Oracle Parallel Server](#)
- [並列を考慮するオプティマイザ](#)
- [パラレル実行](#)
- [ビットマップ索引](#)
- [スター問合せ](#)
- [スター変換](#)
- [クエリー・リライト](#)

### Oracle Parallel Server

Oracle Parallel Server は、OLTP アプリケーションとデータ・ウェアハウス・アプリケーションの両方に次のような重要な利益をもたらします。

- アプリケーション・フェイルオーバー
- スケーラブルなパフォーマンス
- ロード・バランシング
- マルチユーザーでの拡張性

Oracle Parallel Server のフェイルオーバー機能（データベースへの接続が切断されたときに、アプリケーションが自動的に再接続する機能）を使用すると可用性が向上し、主に OLTP アプリケーションに利益をもたらします。同様に、データベースに接続できるユーザー数の拡張性は、OLTP 環境に大きな利益をもたらします。アプリケーションがアクセスするデータを各サーバーごとに分けている場合は、Oracle Parallel Server 上の OLTP のパフォーマンスも拡張できます。

データ・ウェアハウス・アプリケーションにとって、Oracle Parallel Server の主な利点はパフォーマンスの拡張性です。Oracle Parallel Server のアーキテクチャによって、パラレル実行時には、優れたロード・バランシングを行うことができます。Oracle Parallel Server のクラスタまたは MPP のノードが一時的にスローダウンする場合は、そのノード上のパラレル実行サーバーに最初に割り当てられた（ただしサーバーによってまだ開始されていない）作業は、他ノード上のサーバーによって実行できるので、そのノードが重大なボトルネックとなるのを防げます。Oracle Parallel Server はクラスタおよび MPP 上のパラレル実行に不可欠ですが、ほとんどの問合せ環境では、分散ロック・マネージャのオーバーヘッドは最小です。

**関連項目：**『Oracle8i Parallel Server 概要および管理』のテキストを参照してください。

## 並列を考慮するオプティマイザ

並列性の認識は、Oracle のコストベース・オプティマイザに組み込まれています。パラレル実行について考慮することは、問合せ実行計画の作成における基本構成要素です。さらに、計画を選択する際に応答時間とスループットに関するトレードオフを制御できます。

オプティマイザは、使用可能なプロセッサと、問合せがアクセスするデータを保管するディスク・ドライブの数に基づいて、並列度に関する高機能のデフォルトを選択します。アクセス・パスの選択（表走査と索引アクセスの選択など）では並列度が考慮されるので、パラレル実行するために最適化された計画をたてられます。実行計画はさらに拡張性が高く、オプティマイザのコストとパラレル実行の実行時間との相関関係が改善されています。

初期化パラメータ `OPTIMIZER_PERCENT_PARALLEL` は、オプティマイザがコスト・ファンクションの中で応答時間を最短化するために使う重み付けを定義します。

**関連項目：** 26-22 ページの「[OPTIMIZER\\_PERCENT\\_PARALLEL](#)」

## パラレル実行

パラレル実行を使用すると、単一の問合せまたは DML 文を複数のプロセスで同時に処理できます。複数のサーバー・プロセスに作業を分割することによって、Oracle は 1 つのサーバー・プロセスだけで処理する場合よりも高速で操作を実行できます。

パラレル実行は、データ処理集中型のデータ・ウェアハウス操作のパフォーマンスを大幅に改善します。パラレル問合せは、ハードウェア・リソースを追加した場合のパフォーマンス面の拡張性を提供します。問合せ処理で単一システムの多くの CPU 間で効果的に分散できる SMP（対称マルチプロセッシング）、クラスタ化システム、MPP（超並列プラットフォーム）を使用すると、最大のパフォーマンスを得られます。

**関連項目：** パラレル実行の詳細は、第 26 章「[パラレル実行のチューニング](#)」、第 27 章「[パラレル実行のパフォーマンス上の問題について](#)」、および『Oracle8i 概要』を参照してください。

## ビットマップ索引

通常の B\* ツリー索引は、従業員名などのように、各キー値またはキー値の範囲が数個のレコードのみを参照するときに最も効率よく動作します。対照的に、ビットマップ索引は、従業員の性別などのように各キー値が多くのレコードを参照するときに最も効率よく動作します。

ビットマップ索引は、通常の索引と同じ機能を提供しますが、索引付けを非常に高速にし、スペースを効率的に使用する別の内部表現を使用します。ビットマップ索引は、非定型問合せによるデータ量は多いものの、同時処理されるトランザクションは少ないようなデータ・ウェアハウス・アプリケーションにとって利益があります。

ビットマップ索引は、多くの種類の非定型問合せの応答時間を短縮します。また、他の索引方法と比較してスペース使用量を大幅に削減し、ローエンドなハードウェアにおいてもパフォーマンスを大きく向上させます。ビットマップ索引は並列で作成でき、コストベースの最適化と完全に統合されています。

## ドメイン索引

索引タイプのスキーマ・オブジェクトを使用すると、"ドメイン索引"と呼ばれるアプリケーション固有の索引を作成できます。ドメイン索引は、アプリケーション固有のドメインでデータの索引を作成するために使用します。ドメイン索引は、索引タイプが提供するルーチンによって作成、管理およびアクセスされる索引のインスタンスです。これは、データベースによって保守され、"索引"として参照される B\* ツリー索引とは対照的です。

**関連項目：** 6-22 ページの「[ドメイン索引の使用方法](#)」

## スター問合せ

あるタイプのデータ・ウェアハウス設計は、"スター"・スキーマと呼ばれています。一般にスター・スキーマは、大きな 1 つの "事実" 表と、かなり小さな複数の "ディメンション" 表（参照表）から構成されます。スター問合せは、通常は問合せの中の述語によって、複数のディメンション表をファクト表の 1 つに結合します。

Oracle のコストベースの最適化は、スター問合せを認識し、それらのための効率のよい実行計画を生成します。実際、スター問合せを効率よく実行するには、コストベースの最適化を使用する必要があります。コストベースの最適化を使用可能にするには、表に対して ANALYZE を実行し、OPTIMIZER\_MODE パラメータが RULE に設定されていないことを確認します。

**関連項目：** スター問合せの最適化の詳細は、『Oracle8i 概要』を参照してください。STAR ヒントの詳細は、「[STAR](#)」を参照してください。

## スター変換

スター変換は、スター問合せを効率よく実行するために設計されたコストベースの変換です。スター問合せは少数のディメンション表と密度の濃いファクト表で効率よく機能しますが、スター変換は大量のディメンション表とまばらなファクト表で効率よく機能します。

スター変換は、初期化パラメータ `STAR_TRANSFORMATION_ENABLED` を `TRUE` に設定することで使用可能になります。`STAR_TRANSFORMATION` ヒントを使用して、変換が使用される最適な計画をオプティマイザに使用させるようにすることができます。

**関連項目：** 7-60 ページの「[STAR\\_TRANSFORMATION](#)」では、このヒントの使用方法を説明します。また、スター変換の詳細は、『Oracle8i 概要』を参照してください。『Oracle8i リファレンス・マニュアル』では、`STAR_TRANSFORMATION_ENABLED` 初期化パラメータについて説明します。

## クエリー・リライト

マテリアライズド・ビューを定義した場合、Oracle は元表ではなくサマリー表を使用して問合せを透過的にリライトします。

## データ・ウェアハウス・アプリケーションのチューニング

データ・ウェアハウス・アプリケーションのチューニングには、シリアル SQL 文とパラレル SQL 文の両方のチューニングが関係します。

データ・ウェアハウス・アプリケーションでは共有 SQL をお勧めしません。これらの SQL 文では、バインド変数ではなくリテラル値を使用してください。バインド変数を使用すると、オプティマイザは列の選択性に関して総括的な仮説を立てます。対照的に、リテラル値を指定すると、オプティマイザは値のヒストグラムを使用するので、より優れたアクセス計画を提供できます。

**関連項目：** [第 10 章「SQL および共有 PL/SQL 領域の管理」](#)

## データ・ウェアハウスのバックアップとリカバリ

ディスク障害からの回復、人的エラーからの回復、ソフトウェア障害からの回復、火災からの回復など、回復にはさまざまなレベルがあり、それぞれ異なる手順が必要です。Oracle が提供するものは、解決策のほんの一部です。組織は、長い使用不能時間による業務コストを検討することによって、バックアップとリカバリにかかる費用を判断する必要があります。

`NOLOGGING` オプションを使うと、ログ生成のオーバーヘッドを伴うことなく特定の操作を実行できます。ログをとらない場合でも、ディスクのミラー化または RAID テクノロジを使っている場合はディスクの障害を避けられます。ウェアハウスを毎日または毎週テープからロードする場合は、いくつかのリモート・ロケーションにテープのコピーを保管し、問題

が発生したときにテープから再ロードすることによって、すべての障害から完全に回復できます。

見方を変えると、ディスクをミラー化してバックアップをとると同時にログのアーカイブも行い、リモート待機システムをメンテナンスすることもできます。ミラー化されたディスクによって、ディスク障害による可用性の損害を避けることができ、人的エラー（間違った表の削除など）やソフトウェア・エラー（ディスク・ブロックの破壊など）が発生した場合の総合的な損失からも保護できます。火災、停電またはその他の問題がプライマリ・サイトで発生した場合は、バックアップ・サイトによって故障状態が長時間となることが防げます。

**関連項目：** 回復および NOLOGGING オプションの詳細は、26-71 ページの「[\[NO\]LOGGING オプション](#)」、『Oracle8i 管理者ガイド』、および『Oracle8i SQL リファレンス』を参照してください。

## ファースト・スタート・パラレル・リカバリのチューニング

ファースト・スタート・パラレル・リカバリを使用すると、回復のスループットが改善され、複数の同時プロセスを使用するのでロールバック・セグメントの適用に必要な時間が少なくて済みます。初期化パラメータ `FAST_START_PARALLEL_ROLLBACK` は、使用するパラレル回復の上限を決定します。

ファースト・スタート・パラレル・リカバリでは、未回復のロールバック・セグメントそれぞれにつき少なくとも 1 つのプロセスが割り当てられるように十分な数の回復プロセスが自動的に開始されます。この機能では、可能であれば、複数のプロセスが 1 つのトランザクションをロールバックするように指定されます。ただし、アクティブな回復プロセスの数が、`FAST_START_PARALLEL_ROLLBACK` の値で設定された上限を超えることはありません。

### いつファースト・スタート・パラレル・リカバリを使用するか

一般に、DSS 環境では、OLTP 環境よりも大きいトランザクションが行われる場合が多くなります。そのため、ファースト・スタート・パラレル・リカバリは、多くの場合 DSS システムに適用されます。

インスタンスに対してトランザクション回復の進捗を監視することにより、ファースト・スタート・パラレル・リカバリを実行する程度を決めます。これを行うには、この項の後半で説明するように、2 つの回復関連 V\$ ビューの内容を調べます。

### ファースト・スタート・パラレル・リカバリに適切なパラレル化の決定

`FAST_START_PARALLEL_ROLLBACK` でデフォルト設定されている値の 20 は、ほとんどのシステムで適切です。ただし、次に説明するように、`V$FAST_START_SERVERS` ビューと `V$FAST_START_TRANSACTIONS` ビューの情報を利用すると、このパラメータをより正確にチューニングできます。また、全体の回復処理の目標も考慮してください。

**V\$FAST\_START\_SERVERS ビュー**

このビューは、ファースト・スタート・パラレル・リカバリを実行するサーバー・プロセスの進捗に関するデータを提供します。FAST\_START\_PARALLEL\_ROLLBACK の値を 1 またはそれ以上に設定すると、ファースト・スタート・パラレル・リカバリが使用可能になります。SMON を使用可能にするとトランザクションはシリアルに回復され、V\$FAST\_START\_SERVERS の行は 1 つのみになります。

ファースト・スタート・パラレル・リカバリを使用可能にすると、このビューの行の合計数は、アクティブなリカバリ・プロセスの数よりも 1 つ多くなります。その他のプロセスとして SMON があります。PARALLEL\_TRANSACTION\_RECOVERY\_DEGREE の値によって、このビューの行の数が制限されます。このビューの行はそれぞれリカバリ・プロセスに対応しています。V\$FAST\_START\_SERVERS 列とその定義は次のとおりです。

State	状態を示す値は、"IDLE" か "RECOVERING" のどちらかです。
Undoblocksdone	サーバーが完了した割当て作業の割合。
PID	Oracle PID。

**V\$FAST\_START\_TRANSACTIONS ビュー**

このビューは、各ロールバック・トランザクションの回復の進捗に関する情報を提供します。このビューには、ロールバック・トランザクションごとに行が 1 つあります。V\$FAST\_START\_TRANSACTIONS 列とその定義は次のとおりです。

Rollback segment number	トランザクションのロールバック・セグメント数。
Slot	ロールバック・セグメント内でトランザクションが占めるスロット。
Wrap	スロットのインカネーション番号。
State	トランザクションの状態は、"TO BE RECOVERED"、"RECOVERING"、または "RECOVERED" のいずれかです。
Work_done	このトランザクションで完了した回復の割合。
Oracle PID	このトランザクションの回復が割り当てられた現在のサーバー・プロセスの ID。
Time elapsed	トランザクションで回復が始まってから費やされた秒数。
Parent xid	親トランザクションのトランザクション ID。PDML トランザクションに対してだけ有効です。

Oracle は、ファースト・スタート・パラレル・リカバリ中に、両方の表を継続的に更新します。



## ファースト・スタート・パラレル・リカバリの並列度を決定

回復のパフォーマンスとシステムのリソース使用率を調整するには、パラメータ `FAST_START_PARALLEL_ROLLBACK` をチューニングします。ファースト・スタート・パラレル・リカバリ処理を過度に行うと、CPU とディスクの使用率の問題から、関連のないデータベース操作の応答時間が遅くなります。ファースト・スタート・パラレル・リカバリの処理を調整する必要がある場合は、サーバーの実行中であっても、`FAST_START_PARALLEL_ROLLBACK` の値を変更できます。回復中にこの値を変更すると、`SMON` によって新しい並列度で回復が再開されます。

さらに回復作業を行う必要がある場合は、`FAST_START_PARALLEL_ROLLBACK` の値を `HIGH` に変更します。CPU をパラレル回復で占有しないためには、`FALSE` に設定します。Oracle は 2 倍または 4 倍の数のリカバリ・プロセスを作成するので、`CPU_COUNT` パラメータは正しく設定されます。



# 第 III 部

---

## 設計者および DBA 用の アプリケーション設計ツール

第 III 部では、データベースのチューニング方法と、最適なデータベース・パフォーマンスを得るために使用するさまざまなデータ・アクセス方法について説明します。第 III 部には次の章が含まれます。

- [第 12 章「診断ツールの概要」](#)
- [第 13 章「EXPLAIN PLAN の使用方法」](#)
- [第 14 章「SQL トレース機能と TKPROF」](#)
- [第 15 章「Oracle Trace の使用方法」](#)



---

## 診断ツールの概要

この章では、本番システムの監視とパフォーマンス問題の判断に利用できる多様な診断ツールを紹介します。

トピックは次のとおりです。

- チューニング用のデータ・ソース
- 動的パフォーマンス・ビュー
- Oracle および SNMP サポート
- EXPLAIN PLAN
- Oracle Trace と Oracle Trace Data Viewer
- SQL トレース機能と TKPROF
- サポートされるスクリプト
- アプリケーションの登録
- Oracle Enterprise Manager、バックおよびアプリケーション
- Oracle Parallel Server Management
- ユーザー開発ツール

### チューニング用のデータ・ソース

この項では、チューニング用のデータのさまざまなソースを説明します。これらのソースの多くは一過性のものです。次のものがあります。

- データ・ボリューム
- オンライン・データ・ディクショナリ
- オペレーティング・システム・ツール

- 動的パフォーマンス表
- Oracle Trace と Oracle Trace Data Viewer
- SQL トレース機能
- アラート・ログ
- アプリケーション・プログラムの出力
- ユーザー
- 初期化パラメータ・ファイル
- プログラム・テキスト
- 設計（分析）ディクショナリ
- 比較データ

## データ・ボリューム

最も頻繁に見られるチューニング用のデータ・ソースはデータそのものです。データには、いくつかのトランザクションがいつ実行されたかを示す情報が含まれていることがあります。たとえば、監査表に追加された行数は、"スループット"と呼ばれる実行された有効作業量の最適な測定単位となります。このような行がタイム・スタンプを含んでいる場合は、表に問合せを行い、グラフィックス・パッケージを使用して、スループットを日時に対してプロットできます。このようなタイム・スタンプは、アプリケーションの残りの部分から見える必要はありません。

アプリケーションに監査表が含まれない場合は、追加するとパフォーマンスが低下する場合があるので、注意してください。情報取得の価値と、取得のためのパフォーマンス・コストのトレード・オフを検討してください。

## オンライン・データ・ディクショナリ

Oracle オンライン・データ・ディクショナリは、ANALYZE 文とともに使用されるときに、チューニング用データの貴重なソースとなります。この文は、主にコスト・ベースのオプティマイザで使用するために、クラスタおよび表、列および索引統計をディクショナリ内に格納します。ディクショナリは、パフォーマンスを改善する（低下させる可能性もある）ために索引も使用可能にします。

## オペレーティング・システム・ツール

オペレーティング・システム・レベルでデータを収集するツールは、主に拡張性を判断するのに便利ですが、任意のチューニング・アクティビティの初期段階でも参考にする必要があります。それによって、ハードウェア・プラットフォームに飽和状態となっている部分がないことを確認できます。容量以上に割り当てられているネットワーク・リソースがないことを主にチェックするために、分散システムではネットワーク・モニターも必要です。また、

UNIX コマンドの ping などの単純なメカニズムを使用して、メッセージのターンアラウンド・タイムを設定することも有効です。

**関連項目：** プラットフォーム固有のツールの詳細は、オペレーティング・システムのマニュアルを参照してください。

## 動的パフォーマンス表

システムのチューニングとパフォーマンス問題の調査に役立つ多くの V\$ 動的パフォーマンス・ビューが使用可能です。これらを使用すると、SGA 内のメモリー構造にアクセスできます。

**関連項目：** [第 16 章「動的パフォーマンス・ビュー」](#) および『Oracle8i 概要』には各ビューの詳細情報が記載されています。

## Oracle Trace と Oracle Trace Data Viewer

Oracle Trace は、特定のデータベース・ユーザーの SQL イベントと Wait イベントをすべて含む Oracle Server イベントのアクティビティを収集します。この情報を使用して、データベースとアプリケーションをチューニングできます。

**関連項目：** Oracle Trace イベントと Wait イベントの詳細は、[第 15 章「Oracle Trace の使用方法」](#)を参照してください。

## SQL トレース機能

SQL トレース・ファイルは、接続されているプロセスから発行される SQL 文と、これらの文で使用されるリソースを記録します。一般に、動的パフォーマンス・ビューはインスタンスのチューニングに使用し、SQL トレース・ファイルの出力はアプリケーションのチューニングに使用します。

**関連項目：** [第 14 章「SQL トレース機能と TKPROF」](#)

## アラート・ログ

Oracle 環境で予期しない事態が起きたときは、アラート・ログをチェックしてイベントの発生時刻の前後にエントリがあるかどうかを確認してください。

## アプリケーション・プログラムの出力

プロジェクトのなかには、すべてのアプリケーション・プロセス（クライアント側）自身がリソース使用方法を監査証跡に記録することを指示されているものもあります。データベース・コールがライブラリを介して行われている場合は、監査証跡メカニズムを使用して、クライアント / サーバー・メカニズムの応答時間をコールごとのレベルであまり費用をかけずに記録できます。このレベルの（構築または実行に費用がかからない）機能がなくても、

バッチ待ち行列マネージャがレポートしたリソース使用量を保存するだけで、チューニングに使用できる優れたデータ・ソースを用意できます。

## ユーザー

一般に、パフォーマンス問題が発生したときに一連の情報がユーザーから提供されます。

## 初期化パラメータ・ファイル

システムが何の実行を指示したか、およびそれをどのように行おうとしたかに関する正確なデータを持つことが重要です。このデータの一部は、Oracle パラメータ・ファイルから入手できます。

## プログラム・テキスト

アプリケーションが実行した内容に関するデータは、プログラムまたはプロシージャのコードからも入手できます。コードにはプログラム・ロジックと SQL 文の両方が含まれます。サーバー側のコード（ストアド・プロシージャ、制約、およびトリガー）は、この場合はクライアント側のコードと同じ種類のデータの一部と考えられます。チューニング担当者が作業する必要があるサイトでは、一時的な問題によって、またはアプリケーションがパッケージ化されていてソース・コードがリリースされていないために、プログラムのソース・コードを利用できないことがよくあります。このような場合は、チューニング担当者がプログラムからオブジェクトへのクロス・リファレンス情報を獲得することが依然として重要です。そのため、実行可能コードは使用を許可されたデータ・ソースとなっています。幸い、SQL は実行可能プログラム内にもテキストで保持されています。

## 設計（分析）ディクショナリ

設計または分析ディクショナリも、アプリケーションの目的のアクションとリソース使用率を追跡するために使用できます。ただし、アプリケーションが完全にコード生成プログラムによって作成されている場合のみ、データを設計ディクショナリが提供できます。本来はプログラムおよびプロシージャから抽出する必要のあるものです。

## 比較データ

比較データは、ほとんどのチューニング状況で非常に貴重です。チューニングは、各サイトでしばしばコールド・スタートから行われます。チューニング担当者は、専門知識と経験に加えて、データ抽出用のいくつかのツールを持ってサイトにやってきます。熟練したチューニング担当者は、特定の状態に類似性を認め、他の場合に有効だった解決策を適用しようとします。通常はこれらの診断は純粋に主観的なものです。

そのアプリケーションに対して実行した能力調査、または同じアプリケーションが許容できるパフォーマンスを達成している同じまたは別のサイトから得たデータのどちらかが、ベ



スラインとして存在する場合にはチューニングが容易です。次に行う作業は、問題のある環境を修正し、最適な環境に近づけることです。

直接に関連するデータが見つからない場合は、同じようなプラットフォームまたは同じようなアプリケーションのデータをチェックして、同じパフォーマンス特性があるかどうかを確認できます。同じ問題がいたるところに存在することが判明した場合は、特定の効果を期待してチューニングすることには意味がありません。

## 動的パフォーマンス・ビュー

Oracle のパフォーマンスを監視するための主なツールは、Oracle がシステムを監視するために提供している動的パフォーマンス・ビューのセットです。これらのビューの名前は "VS" で始まります。このマニュアルでは、パフォーマンス・チューニングのときの使用方法を説明しています。これらのビューはデータベース・ユーザー SYS が所有しており、管理者はこれらのビューへのアクセス権限をすべてのデータベース・ユーザーに付与できます。システムのチューニングには、一部の動的パフォーマンス・ビューだけが関連します。

**関連項目：** 第 16 章「動的パフォーマンス・ビュー」および『Oracle8i 概要』には各ビューの詳細情報が記載されています。

## Oracle および SNMP サポート

シンプル・ネットワーク管理プロトコル (SNMP) によって、ユーザーは独自のツールとアプリケーションを作成できます。SNMP は、異機種間管理アプリケーションの標準オープン・プロトコルとして認められているプロトコルです。Oracle SNMP サポートによって、SNMP ベースの管理アプリケーションがネットワーク上で Oracle データベースを検出し、識別して監視できるようになりました。Oracle は、いくつかのデータベース管理情報ベース (MIB) をサポートしています。すべてのデータベース管理システム用の標準の MIB (ベンダーに依存しない) および Oracle 固有の情報が入っている Oracle 固有の MIB があります。このマニュアルで説明する統計には、これらの MIB によってサポートされているものとそうでないものがあります。このマニュアルで説明する統計を SNMP を介して取得できる場合は、その点に言及しています。

**関連項目：** 『Oracle SNMP サポート・リファレンス・ガイド』

## EXPLAIN PLAN

EXPLAIN PLAN は、問合せオプティマイザによって使用されるアクセス・パスを表示する SQL 文です。EXPLAIN PLAN コマンドの計画出力にはそれぞれ文のタイプを示す行があります。

EXPLAIN PLAN の結果は、ある程度慎重に解釈する必要があります。計画が表面上は効率的に見えないからといって、必ずしも文の実行が遅いとはかぎりません。実行計画の主観的

な見地からではなく、実際のリソース使用方法に基づいてチューニング用の文を選択してください。

**関連項目：** [第 13 章「EXPLAIN PLAN の使用方法」](#) および『Oracle8i SQL リファレンス』

## Oracle Trace と Oracle Trace Data Viewer

Oracle Trace は、すべての SQL イベントと Wait イベントをはじめとする重要な Oracle Server イベント・データを収集します。SQL イベントには、解析操作、実行操作、フェッチ操作など、SQL 文アクティビティを完全に分類したものが含まれます。サーバー・イベント用に収集されるデータには、特定のイベントが消費する I/O や CPU などのリソース使用率メトリックが含まれます。

多くのリソースを消費する SQL 文は、Oracle Trace を使用すれば容易に識別できます。Oracle Trace Data Viewer は、平均経過時間、CPU 使用量、処理される行ごとのディスク読み込みなど、SQL 文メトリックをはじめとする Oracle Trace データを要約します。

Oracle Trace の収集は、Oracle Trace Manager を介して管理できます。Oracle Trace Data Viewer と Oracle Trace Manager は、Oracle Diagnostics Pack で入手できます。

**関連項目：** [第 15 章「Oracle Trace の使用方法」](#)。

## SQL トレース機能と TKPROF

SQL トレース機能は、どのセッションにも使用できます。SQL トレース機能は、セッションがサーバーに対して実行するすべての解析、実行、フェッチ、コミットまたはロールバック要求によるリソース使用方法を、オペレーティング・システムのテキスト・ファイルに記録します。

TKPROF は、SQL トレース機能が作成したトレース・ファイルを要約し、オプションとして EXPLAIN PLAN 出力を組み込みます。TKPROF は、実行した各文を、消費したリソース、コールした回数および処理した行数とともにレポートします。このため、リソースを最も多く使用している文を非常に簡単に検出できます。経験または使用可能なベースラインがあれば、使用されたリソースが妥当であるかどうかを評価できます。

## サポートされるスクリプト

多くの PL/SQL パッケージが提供されており、インスタンスのチューニングをサポートする多数の SQL\*Plus スクリプトが提供されています。UTLBSTAT.SQL と UTLESTAT.SQL、UTLCHAIN.SQL および UTLDTREE.SQL、UTLLOCKT.SQL がその例です。

これらの統計スクリプトはインスタンス管理をサポートし、パフォーマンス履歴の構築を可能にしています。これらのスクリプトは次の用途に使えます。

- 統計を収集するたびに DDL を発行する必要を取り除く。

- データの収集とレポートを分離することで、典型的なシステム操作の実行中に一定範囲の観測を行い、任意の開始点から任意の終了点の間の統計のレポート作成を可能にします。
- インスタンスが適切にチューニングされたかどうかを判別するために使用できるいくつかの比率をレポートとして作成します。
- バッファ・キャッシュ内の LRU 統計を使用しやすい形式で提示します。

## アプリケーションの登録

アプリケーションの名前と、そのアプリケーションによって実行されるアクションをデータベースに登録できます。アプリケーションを登録すると、システム管理者とパフォーマンスのチューニングを行う担当者がモジュール別にパフォーマンスを追跡できるようになります。システム管理者はこの情報を使用して、モジュール別のリソースの使用状況も追跡できます。アプリケーションがデータベースに登録されると、その名前とアクションが V\$SESSION ビューと V\$SQLAREA ビューに記録されます。

さらに、Oracle Trace ではアプリケーション登録データが収集されます。

**関連項目：** 詳細は、『Oracle Trace User's Guide』、『Oracle Trace Developer's Guide』、および第 5 章「[アプリケーションの登録](#)」を参照してください。

## Oracle Enterprise Manager、パックおよびアプリケーション

この項では、Oracle Enterprise Manager、そのパックおよびそれが提供する最も有用な診断ツールとチューニング・アプリケーションについて説明します。内容は次のとおりです。

- Oracle Enterprise Manager の概要
- Oracle Diagnostics Pack
  - Oracle Capacity Planner
  - Oracle Performance Manager
  - Oracle Advanced Event Tests
  - Oracle Trace
- Oracle Tuning Pack
  - Oracle Expert
  - Oracle Index Tuning Wizard
  - Oracle SQL Analyze
  - Oracle Auto-Analyze

- Oracle Tablespace Manager

## Oracle Enterprise Manager の概要

Oracle は、Oracle Enterprise Manager プラットフォームで、洗練されたデータベース、システム管理環境の必要性に対処しています。このツールを使用すると、Oracle 環境を包括的に管理できます。

Oracle Enterprise Manager を使用すると、各部門から全社レベルまで、レプリケーション構成、WEB サーバー、メディア・サーバーなどの広範な Oracle インプリメンテーションを管理できます。Oracle Enterprise Manager は、次のもので構成されます。

- 管理タスクやアプリケーションの実行元となる中央コンソール。
- WEB ブラウザ内から Oracle Enterprise Manager コンソールやデータベース管理アプリケーションを実行するためのサポート。
- 重要な管理サービスを常に利用できることを保証する、パラレル化されていない拡張性とフェイルオーバー機能を提供する軽量の 3 層アーキテクチャ。
- あらゆる環境の管理データを格納する中央リポジトリ。Oracle Enterprise Manager は、分散システムの協同管理に責任を負う管理者チームをサポートします。
- イベント管理、サービス検出、ジョブの作成 / 管理のための共通サービス。
- イベントのリモート監視、ジョブの実行、管理コンソールとの通信を行うためのサーバー側高機能エージェント。
- リアルタイムまたは履歴パフォーマンス・データの収集と管理に使用する低いオーバーヘッドの枠組み。
- セキュリティ、記憶、バックアップ、回復、インポートおよびソフトウェア配布について Oracle データベースを管理するアプリケーション
- レプリケーション、Oracle Parallel Server、その他の Oracle Server 構成を管理する階層化アプリケーション。
- Oracle Diagnostics Pack と呼ばれる、監視、診断および計画を行うオプション製品。
- Oracle Tuning Pack と呼ばれる、アプリケーション、データベースおよびシステムをチューニングするオプション製品。
- Oracle Change Management Pack と呼ばれる、Oracle メタデータ変更を管理するオプション製品。

Oracle Enterprise Manager パックは、Oracle Enterprise Manager システム管理テクノロジーの上に構築されたウィンドウ・ベースおよび Java ベースのアプリケーション・セットです。このマニュアルの性質上、Oracle Change Management Pack については割愛します。

## Oracle Diagnostics Pack

Oracle Diagnostics Pack は、データベース、オペレーティング・システムおよびアプリケーションの状態を監視、診断およびメンテナンスします。履歴分析とリアルタイム分析の両方を使用して、問題が発生する前に自動的にそれを回避します。このパックには、強力な容量計画機能があり、ユーザーは将来のシステム・リソース要件を容易に計画できます。

Oracle Diagnostics Pack のコンポーネントには、Oracle Capacity Planner、Oracle Performance Manager、Oracle Advanced Event Tests および Oracle Trace が含まれます。次の項では、各コンポーネントについて説明します。

### Oracle Capacity Planner

Oracle Capacity Planner を使用すると、Oracle データベースとオペレーティング・システムの履歴パフォーマンス・データを収集および分析できます。Oracle Capacity Planner では、収集したいパフォーマンス・データ、収集の間隔、ロードのスケジューリングおよびデータ管理方針を指定できます。また、Oracle Capacity Planner の詳細分析とレポートを使用すると、収集したデータの検討、使用しやすいグラフやレポートへのデータのフォーマットおよび将来のリソースのニーズを予測するための分析を行うことができます。

### Oracle Performance Manager

Oracle Performance Manager は、データベースとオペレーティング・システムのパフォーマンス・データを獲得、計算および提示します。これによって、メモリーの効果的な使用、ディスク I/O の最小化およびリソースの競合の低減に必要なキー・メトリックスをユーザーが監視できるようになります。Oracle Performance Manager は、パフォーマンス・メトリックスのグラフィカルなリアルタイム・ビューを提供し、パフォーマンス問題の解決に使用する明細データにすばやくアクセスするためにユーザーが監視ビューヘドリル・ダウンできます。パフォーマンス・データは、リアルタイム・モードで獲得および表示されます。また、再実行のためにデータを記録できます。

Oracle Performance Manager には、多くの事前定義チャートがあります。また、自分で独自のチャートを作成することもできます。グラフィカル・モニターは、カスタマイズと拡張が可能です。ユーザーは、表、折れ線グラフ、棒グラフ、立方体グラフ、円グラフなどの 2 次元または 3 次元のさまざまなグラフィカル・ビューで監視情報を表示できます。また、監視の割合をカスタマイズできます。

さらに、Oracle Performance Manager では、データベース・セッションごとにデータベース・アクティビティに焦点を絞ったビューがあります。Top Sessions チャートは、セッションごとに Oracle 動的パフォーマンス・データのサンプルを抽出して分析し、メモリー消費量、CPU 使用率、またはファイル I/O アクティビティなどの特定の選択基準に基づいて上位の Oracle ユーザーを自動的に判断します。

また、Oracle Performance Manager 内の Database Locks チャートには、ロックしているユーザー、ロック・タイプ、ロックされているオブジェクト、保持されているモードや要求されているモードなどの詳細情報を含むデータベース・ロックが表示されます。

## Oracle Advanced Event Tests

Oracle Diagnostics Pack には、Oracle Advanced Event Tests が含まれています。これは、Oracle Event Management System で実行中のホスト・イベントおよびデータベース・イベントで、エージェントが監視します。アドバンスド・イベント・テストをコンソールから開始して、管理されているサーバー上で問題を自動的に検出することができます。Oracle Advanced Event Tests には、データベース・サービスを監視する事前定義イベントとデータベース・パフォーマンスに影響するシステム・イベントがあります。

たとえば、パフォーマンス監視イベントには、I/O の監視、メモリー構造のパフォーマンス、ユーザー・プログラム応答時間が含まれます。I/O の監視には、ディスク I/O 率と SQL\*Net I/O 率があります。このツールを使用すると、I/O 率のしきい値を指定することもできます。このしきい値を超過すると、警告を受け取ります。

メモリー構造のパフォーマンス監視には、ライブラリ・キャッシュ、データ・ディクショナリ、データベース・バッファのヒット率が含まれます。さらに、動的パフォーマンス表 VSSYSSTAT によって獲得された統計を柔軟に監視できます。

Oracle Advanced Event Tests を使用すると、Oracle の記憶域構造の状態およびパフォーマンスの監視や、CPU の過度な使用、過度の CPU ロードまたはページング、ディスク容量問題などにもなる問題の検出ができます。

Oracle Advanced Event Tests は、問題イベントについて管理者に警告するとともに、それを自動的に修正するようにも構成できます。"Fixit Job" を使用すると、イベント警告レベルに達したときに、事前に決められたアクションが自動的に行われます。

## Oracle Tuning Pack

Oracle Tuning Pack は、効率の悪い SQL、劣ったデータ構造、システム・リソースの不適切な使用など、主要なデータベースおよびアプリケーションのボトルネックを識別しチューニングすることによって、システム・パフォーマンスを最適化します。このパックは、チューニングの機会を事前に見つけて、分析と、システムのチューニングに必要な変更を自動的に生成します。この製品には、作業中にチューニングする方法の訓練を DBA に対して行う強力な教育ツールが内在しています。

## Oracle Expert

Oracle Expert を使用すると、データベースのパフォーマンス・チューニングを自動化できます。Oracle Diagnostics Pack やその他の Oracle 監視アプリケーションで検出されたパフォーマンス問題は、Oracle Expert で分析され解決されます。Oracle Expert は、データの収集と分析のプロセスを自動化し、"エキスパート" データベース・チューニングの推奨事項、インプリメンテーション・スクリプトおよびレポートを提供するルールベースの推論エンジンを含みます。

## Oracle SQL Analyze

Oracle SQL Analyze は、問題のある SQL 文を識別し、そのチューニングを援助します。SQL Analyze を使用すると、リソースを集中的に使う SQL 文を検出し、SQL 文の実行計画を調べ、文のさまざまなオプティマイザ・モードとバージョンをベンチマークおよび比較し、別の SQL を生成してアプリケーションのパフォーマンスを改善することができます。

## Oracle Tablespace Manager

Oracle Tablespace Manager を使用すると、Oracle の領域管理問題を識別して修正できます。Oracle Tablespace Manager には、Tablespace Allocation グラフィックス、Tablespace Reorganization ツールおよび Tablespace Analyzer ツールの 3 つの主要な機能があります。

「Segments and Extents Information」ページの Tablespace Allocation グラフィックスは、特定の Oracle インスタンスと関連付けられたあらゆる表領域の特性（表領域のデータ・ファイルとセグメント、データ・ブロックの合計、空きデータ・ブロック、表領域の現在の記憶域割当てで使用可能な空きブロックの割合など）の完全図を提供します。

Reorganization ツールを使用すると、スペース使用の改善およびパフォーマンスの向上のために、特定のオブジェクトまたは完全な表領域を再作成できます。Analyzer ツールを使用すると、データベースの統計を自動的に最新に保つことができます。

## Oracle Index Tuning Wizard

Oracle Index Tuning Wizard は、索引変更が有利に働く表を識別し、それぞれの表について最良の索引方針を見極め、それを検証のために提示して、その推奨のものをインプリメントできるようにします。

## Oracle Auto-Analyze

Oracle Auto-Analyze を使用すると、Oracle データベース統計を保守できます。Auto-Analyze は、ユーザー指定のデータベース保守期間中に実行されるので、古い統計の更新でパフォーマンスに悪影響が及ぶことはあまりありません。この保守期間中、Auto-Analyze は更新に必要なオブジェクトの特定のスキーマを調べます。また、更新が必要なオブジェクトの順序を優先し、統計を更新します。保守期間中に統計更新が完了しない場合、Auto-Analyze は更新操作の状態を保持し、次の保守期間中に更新を再開します。

# Oracle Parallel Server Management

Oracle Parallel Server Management (OPSM) は、Oracle Parallel Server のための包括的かつ統合的なシステム管理ソリューションです。OPSM を使用すると、オープンなクライアント / サーバー・アーキテクチャを介して異機種環境で実行しているマルチインスタンス・データベースを管理できます。

OPSM を使用すると、パラレル・データベースの管理に加えて、ジョブのスケジューリング、イベント管理の実行、パフォーマンスの監視、パラレル・データベースをチューニングするための統計の獲得が行えます。

OPSM の詳細は、『Oracle Parallel Server Management for UNIX 構成ガイド』および『Oracle Parallel Server Management ユーザーズ・ガイド』を参照してください。インストールの手順は、使用しているプラットフォーム固有のインストレーション・ガイドを参照してください。

## ユーザー開発ツール

DBA が社内用のパフォーマンス・ツールを設計したサイトもあります。このようなツールとして、次のものがあります。

- 表を拡張するのに十分な領域があるかどうかを判断する空き領域モニター
- ロック監視ツール
- 表およびすべての関連索引を表示するスキーマ記述スクリプト
- ユーザーあたりのデフォルト表領域および一時表領域を示すツール

このようなプログラムは、自動的に実行されるように設定することにより Oracle と統合できます。



---

## EXPLAIN PLAN の使用方法

この章では、実行計画、EXPLAIN PLAN 文を紹介し、その出力の解釈方法を説明します。また、特定の SQL 文に対するチューニングの投資を活かすプラン・スタビリティ機能とストアド・アウトラインの使用方法も説明します。さらに、アプリケーションのパフォーマンス特性を制御するアウトラインを管理するプロシージャを示します。この章のトピックは次のとおりです。

- [EXPLAIN PLAN の概要](#)
- [出力表の作成](#)
- [PLAN\\_TABLE 出力の表示](#)
- [出力表の列](#)
- [EXPLAIN PLAN 出力のフォーマット](#)
- [EXPLAIN PLAN の制限事項](#)

**関連項目：** EXPLAIN PLAN の構文は、『Oracle8i SQL リファレンス』を参照してください。

### EXPLAIN PLAN の概要

EXPLAIN PLAN 文は、SELECT、UPDATE、INSERT および DELETE 文について Oracle オプティマイザが選択した実行計画を表示します。文の実行計画とは、Oracle がその文を実行するために行う一連の処理です。実行プランのコンポーネントには次のものが含まれます。

- 文によって参照される表の順序
- 文で言及される各表のアクセス方法
- 文の結合操作の影響を受ける表の結合方法

EXPLAIN PLAN 出力は、Oracle が SQL 文を実行する方法を示します。しかし、EXPLAIN PLAN の結果だけでは、よく調整された文とうまく機能しない文を区別できません。たとえば、文が索引を使用することが EXPLAIN PLAN 出力で示されたとしても、その文が効率的

に機能しているとはかぎりません。索引の使用は、非常に非効率的である場合もあります。したがって、EXPLAIN PLAN を使用する場合は、アクセス計画を判別し、後からテストによってそれが最適な計画であることを証明するのが最もよい方法でしょう。

計画を評価する際は、必ず文の正確なりソース消費を調べてください。最善の結果を得るには、Oracle Trace、または SQL トレース機能および TKPROF を使用して個々の SQL 文のパフォーマンスを調べてください。

**関連項目：** [第 14 章「SQL トレース機能と TKPROF」](#) および [第 15 章「Oracle Trace の使用方法」](#)

## 出力表の作成

EXPLAIN PLAN 文を発行する前に、出力を入れる表を作成します。次のいずれかの方法を使用します。

- SQL スクリプト UTLXPLAN.SQL を実行し、自分のスキーマ内に PLAN\_TABLE というサンプル出力表を作成してください。このスクリプトの正確な名前と位置は、ご使用のオペレーティング・システムによって異なります。PLAN\_TABLE は、EXPLAIN PLAN 文が実行計画の記述行を挿入するデフォルト表です。
- 選択した名前で出力表を作成するために、CREATE TABLE 文を発行します。EXPLAIN PLAN 文を発行するときは、その出力先をこの表にすることができます。

EXPLAIN PLAN 文の出力を格納するために使用する表は、PLAN\_TABLE と同じ列名とデータ型を持っていなければなりません。

```
CREATE TABLE plan_table
(statement_id    VARCHAR2(30),
timestamp       DATE,
remarks         VARCHAR2(80),
operation       VARCHAR2(30),
options         VARCHAR2(30),
object_node     VARCHAR2(128),
object_owner    VARCHAR2(30),
object_name     VARCHAR2(30),
object_instance NUMERIC,
object_type     VARCHAR2(30),
optimizer       VARCHAR2(255),
search_columns  NUMERIC,
id              NUMERIC,
parent_id       NUMERIC,
position        NUMERIC,
cost            NUMERIC,
cardinality     NUMERIC,
bytes           NUMERIC,
other_tag       VARCHAR2(255),
partition_start VARCHAR2(255),
```

```
partition_stop    VARCHAR2(255),
partition_id      VARCHAR2(255),
other             LONG,distribution VARCHAR2(30));
```

## PLAN\_TABLE 出力の表示

次のスクリプトを使用して、最新の計画表出力を表示します。

- UTLPLS.SQL - シリアル処理用に計画表出力を表示します。
- UTLXPLS.SQL - パラレル実行列で計画表出力を表示します。

EXPLAIN PLAN 出力で示される行ソース件数の値では、計画の各ステップで処理される行数を識別します。これは、問合せの効率が悪い部分、たとえば、効率の悪い操作を実行しているアクセス計画を持つ行ソースを識別するのに役立ちます。

**関連項目：** 13-24 ページの「[ネストされたフォーマットでの PLAN\\_TABLE 出力の選択](#)」

## 出力表の列

EXPLAIN PLAN 文が使用する PLAN\_TABLE には、次のような列があります。

表 13-1 PLAN\_TABLE 列

列	説明
STATEMENT_ID	EXPLAIN PLAN 文で指定した、オプションの STATEMENT_ID パラメータの値です。
TIMESTAMP	EXPLAIN PLAN 文が発行された日時です。
REMARKS	実行計画の各ステップに関連付けるコメント (最大 80 バイト) です。PLAN_TABLE の行に関するコメントを追加または変更する必要がある場合は、UPDATE 文を使って PLAN_TABLE の行を修正してください。
OPERATION	このステップで実行された内部処理の名前です。文に対して生成された最初の行の列には、次の値の 1 つが含まれます。 DELETE STATEMENT INSERT STATEMENT SELECT STATEMENT UPDATE STATEMENT この列の値の詳細は、 <a href="#">表 13-4</a> を参照してください。

表 13-1 PLAN\_TABLE 列

OPTIONS	OPERATION 列に記述されている処理に関するバリエーションです。 この列の値の詳細は、 <a href="#">表 13-4</a> を参照してください。
OBJECT_NODE	オブジェクト (表名またはビュー名) を参照するために使用されたデータベース・リンクの名前です。パラレル実行を指定したローカル問合せの場合は、各処理による出力の使用順序がこの列に記述されます。
OBJECT_OWNER	表または索引を含むスキーマを所有しているユーザーの名前です。
OBJECT_NAME	表または索引の名前です。
OBJECT_INSTANCE	元の文に指定されているオブジェクトの位置に対応する順番を示す数値です。この数値は元の文テキストに関して、左から右へ、外側から内側へ付番されています。ビューの展開があると、この数値は予測できません。
OBJECT_TYPE	たとえば、索引に対する NON-UNIQUE のような、オブジェクトに関する記述的な情報を与える修飾子です。
OPTIMIZER	オブティマイザの現行モードです。
SEARCH_COLUMNS	現在は使用されていません。
ID	実行計画の各ステップに割り当てられた数値です。
PARENT_ID	ID ステップの出力で処理する次の実行ステップの ID です。
POSITION	同じ PARENT_ID を持っているすべてのステップに対する処理順序です。
OTHER	ユーザーにとって有効な実行ステップに関するその他の情報です。
OTHER_TAG	OTHER 列の内容を示します。この列で可能な値の情報は、 <a href="#">表 13-2</a> を参照してください。
DISTRIBUTION	"生成側" 問合せサーバーから "受取側" 問合せサーバー間で行を分配する方法を格納します。作成者と顧客の問合せサーバーの詳細は、『Oracle8i 概要』を参照してください。

表 13-1 PLAN\_TABLE 列

PARTITION_START	<p>アクセスされるパーティション範囲の開始パーティションです。これは、次のいずれかの値をとります。</p> <p><math>n</math> は、先頭パーティションが SQL コンパイラで識別され、そのパーティション番号が <math>n</math> で示されることを意味します。</p> <p>KEY は、先頭パーティションが実行時にパーティション・キー値から識別されることを意味します。</p> <p>ROW LOCATION は、先頭パーティション（末尾のパーティションと同じになります）が実行時に、検索される各レコードの位置から計算されることを意味します。レコードの位置は、ユーザーまたはグローバル索引によって獲得されます。</p> <p>INVALID は、アクセスされるパーティションの範囲が空であることを意味します。</p>
PARTITION_STOP	<p>アクセスされるパーティション範囲の終了パーティションです。これは、次のいずれかの値をとります。</p> <p><math>n</math> は、末尾のパーティションが SQL コンパイラで識別され、そのパーティション番号が <math>n</math> で示されることを意味します。</p> <p>KEY は、末尾のパーティションが実行時にパーティション・キー値から識別されることを意味します。</p> <p>ROW LOCATION は、末尾のパーティション（先頭パーティションと同じになります）が実行時に、検索される各レコードの位置から計算されることを意味します。レコードの位置は、ユーザーまたはグローバル索引によって獲得されます。</p> <p>INVALID は、アクセスしたパーティションの範囲が空であることを意味します。</p>
PARTITION_ID	PARTITION_START 列と PARTITION_STOP 列の値の対を計算したステップです。
COST	<p>オブティマイザのコストベース・アプローチによって見積もられた操作コスト。ルールベース・アプローチを使用する文では、この列は NULL になります。表アクセス操作のためのコストは判断されません。この列の値には、特定の単位はなく、単に実行計画のコストを比較するために使用される重み値を示します。</p>
CARDINALITY	この操作によってアクセスされる行数のコストベースのアプローチによる見積りです。
BYTES	この操作によってアクセスされるバイト数のコストベースのアプローチによる見積りです。

表 13-2 で、OTHER\_TAG 列に使われる値を説明します。

表 13-2 PLAN\_TABLE の OTHER\_TAG 列の値

OTHER_TAG テキスト ( 例 )	意味	説明
ブランク		シリアル実行。
serial_from_remote (S R)	リモートからシリアル	リモート・サイトでシリアル実行されます。
serial_to_parallel (S P)	シリアルからパラレル	シリアル実行。ステップの出力は、パーティション化されるか、パラレル実行サーバーにブロードキャストされます。
パラレルからパラレル (P P)	パラレルからパラレル	パラレル実行。ステップの出力は、パラレル実行サーバーの 2 番目のセットに再パーティション化されます。
parallel_to_serial (P S)	パラレルからシリアル	パラレル実行。ステップの出力は、シリアル " 問合せコーディネータ " プロセスに戻されます。
parallel_combined_with_parent (PwP)	親と組み合せたパラレル	パラレル実行。ステップの出力は、同じパラレル・プロセスの次のステップに送られます。親へのプロセス間通信はありません。
parallel_combined_with_child (PwC)	子と組み合せたパラレル	パラレル実行。ステップの入力は、同じパラレル処理の前のステップから受け取ります。子からのプロセス間通信はありません。

表 13-3 で、DISTRIBUTION 列に使われる値を説明します。

**表 13-3 PLAN\_TABLE の DISTRIBUTION 列の値**

DISTRIBUTION テキスト	説明
PARTITION (ROWID)	UPDATE または DELETE を実行する行の rowid を使用し、表または索引のパーティション化に基づいて行を問合せサーバーにマップします。
PARTITION (KEY)	列のセットを使用し、表または索引のパーティション化に基づいて行を問合せサーバーにマップします。パーシャル・パーティション・ワイズ・ジョイン、パラレル INSERT、パーティション表の CREATE TABLE AS SELECT、CREATE INDEX によるパーティション化されたグローバル索引の作成に使用します。
HASH	結合キーでハッシュ関数を使用して、行を問合せサーバーにマップします。パラレル結合またはパラレル GROUP BY に使用します。
RANGE	ソート・キーの範囲を使用して、行を問合せサーバーにマップします。文に ORDER BY 句がある場合に使用します。
ROUND-ROBIN	行を問合せサーバーにランダムにマップします。
BROADCAST	表全体の行を各問合せサーバーにブロードキャストします。ある表がその他の表に比べて非常に小さい場合、パラレル結合に使用します。
QC (ORDER)	問合せコーディネータが、最初の問合せサーバーから最後の問合せサーバーまで順番に入力データを受け取ります。文に ORDER BY 句がある場合に使用します。
QC (RANDOM)	問合せコーディネータが、入力データをランダムに受け取ります。文に ORDER BY 句がない場合に使用します。

表 13-4 に、EXPLAIN PLAN 文によって生成される OPERATION と OPTION の組合せと、実行計画におけるそれぞれの意味を示します。

表 13-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値

OPERATION	OPTION	説明
AND-EQUAL		複数の rowid のセットを受け取り、重複をなくして、そのセットの共通部分を戻す処理。この処理は単一系列索引のアクセス・パスに対して使用されます。
	CONVERSION	TO ROWIDS は、ビットマップ表現を、表にアクセスするために使用できる実際の rowid に変換します。 FROM ROWIDS は、rowid をビットマップ表現に変換します。 COUNT は、実際の値が必要ない場合に rowid の数を戻します。
	INDEX	SINGLE VALUE は、索引内の単一のキー値のビットマップを参照します。 RANGE SCAN は、ある範囲のキー値のビットマップを検索します。 FULL SCAN は、開始キーまたは終了キーがない場合、ビットマップ索引全走査が実行されます。
	MERGE	範囲走査の結果の複数のビットマップを 1 つのビットマップにマージする。
	MINUS	片方のビットマップのビットを、もう一方のビットマップから減算します。行ソースは否定述語に対して使われます。減算が発生する可能性があるビットマップを作り出す非否定述語がある場合にだけ使用できます。「 <a href="#">ビットマップ索引と EXPLAIN PLAN</a> 」で例を示します。
	OR	2 つのビットマップのビット OR を計算します。
CONNECT BY		CONNECT BY 句を含んでいる問合せに対する、階層順での行の検索。
CONCATENATION		複数の行のセットを受け取り、そのセットの UNION-ALL を戻す処理。
COUNT		表から選択された行の数をカウントする処理。
	STOPKEY	戻される行数を WHERE 句の ROWNUM 式によって制限するカウント処理。
DOMAIN INDEX		ドメイン・インデックスからの 1 つ以上の rowid の検索。



表 13-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値

OPERATION	OPTION	説明
FILTER		行のセットを受け取り、そのいくつかを取り除き、残りを戻す処理。
FIRST ROW		問合せで選択される最初の行だけの検索。
FOR UPDATE		FOR UPDATE 句を含んでいる問合せによって選択される行を検索し、ロックする処理。
HASH JOIN		2 つのセットの行を結合し、結果を戻す操作。
(これらは結合操作です。)	ANTI	ハッシュ逆結合。
	SEMI	ハッシュ・セミ・ジョイン。
INDEX (これらの操作はアクセス方法です。)	UNIQUE SCAN	索引からの単一の rowid の検索。
	RANGE SCAN	索引からの 1 つ以上の rowid の検索。索引値は昇順で走査されます。
	RANGE SCAN DESCENDING	索引からの 1 つ以上の rowid の検索。索引値は降順で走査されます。
INLIST ITERATOR		IN リスト述語内の各値に対して、下の操作を反復します。
INTERSECTION		2 つの行のセットを受け取り、重複をなくして、そのセットの共通部分を戻す処理。
MERGE JOIN		2 つの行のセットを受け取り、それぞれを特定の値でソートし、一方のセットの各行を他方の行と突き合せて結合し、その結果を戻す処理。
(これらは結合操作です。)	OUTER	外部結合文を実行するマージ結合処理。
	ANTI	マージ逆結合。
	SEMI	マージ・セミ・ジョイン。
MINUS		2 つの行のセットを受け取り、最初のセットにあって 2 番目のセットにない行を戻して、重複をなくす処理。
NESTED LOOPS		外側のセットと内側のセット、2 つの行のセットを受け取る処理。Oracle は、外側のセットの各行を内側のセットの各行と比較し、条件を満足する行を戻します。
(これらは結合操作です。)	OUTER	外部結合文を実行するネスト・ループ処理。

表 13-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値

OPERATION	OPTION	説明
PARTITION		PARTITION_START 列および PARTITION_STOP 列によって指定された範囲の各パーティションに対して、下の操作を反復します。
		PARTITION は、単一のパーティション・オブジェクト（表または索引）やパーティション・オブジェクトのセット（パーティション表やそのローカル索引）に適用できるパーティションの境界を示します。パーティションの境界は、PARTITION の PARTITION_START や PARTITION_STOP の値で指定されます。PARTITION_START と PARTITION_STOP に有効な値については、表 13-1 を参照してください。
	SINGLE	1 つのパーティションへのアクセス。
	ITERATOR	多数のパーティション（サブセット）へのアクセス。
	ALL	すべてのパーティションへのアクセス。
	INLIST	ITERATOR に似ているが、インリスト述語に基づきます。
	INVALID	アクセスするように設定されているパーティションが空であることを示します。
PROJECTION		内部処理。
REMOTE		リモート・データベースからのデータの検索。
SEQUENCE		順序値のアクセスを伴う処理。
SORT	AGGREGATE	選択した行のグループにグループ関数を適用した結果として得られる単一行の検索。
	UNIQUE	行のセットをソートし、重複をなくす処理。
	GROUP BY	GROUP BY 句を持つ問合せで、行のセットをグループにソートする処理。
	JOIN	マージ結合の前に、一連の行をソートする操作。
	ORDER BY	ORDER BY 句を持つ問合せに対して行のセットをソートする処理。
TABLE ACCESS (これらの操作はアクセス方法です。)	FULL	表のすべての行の検索。
	CLUSTER	索引クラスタのキーの値に基づいた、表からの行の検索。
	HASH	ハッシュ・クラスタのキーの値に基づいた、表からの行の検索。
	BY ROWID	rowid に基づいた、表からの行の検索。

表 13-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値

OPERATION	OPTION	説明
	BY USER ROWID	ユーザー指定の rowid を使用して表の行が指定される場合。
	BY INDEX ROWID	表がパーティション化されておらず、索引を使用して行を見つけた場合。
	BY GLOBAL INDEX ROWID	表がパーティション化されており、グローバル索引だけを使用して行を見つけた場合。
	BY LOCAL INDEX ROWID	表がパーティション化されており、1 つ以上のローカル索引と場合によってはいくつかのグローバル索引を使用して、行を見つけた場合。
		パーティション境界：  パーティション境界は次のようにして計算されている可能性があります。  前の PARTITION ステップによって決定される場合。この場合、PARTITION_START 列の値と PARTITION_STOP 列の値は PARTITION ステップにある値をレプリケートし、PARTITION_ID には PARTITION ステップの ID が組み込まれます。PARTITION_START と PARTITION_STOP に使用できる値は、n、KEY、INVALID です。  TABLE ACCESS または INDEX ステップ自体で決定される場合。この場合、PARTITION_ID にはそのステップの ID が組み込まれます。PARTITION_START と PARTITION_STOP に使用できる値は、n、KEY、ROW LOCATION (TABLE ACCESS のみ)、INVALID です。
UNION		2 つの行のセットを受け取り、重複をなくして、そのセットの連結結果を戻す処理。
VIEW		ビューの問合せを実行し、結果の行を別の処理に戻す処理。

---

**注意：** アクセス方法と結合操作については、『Oracle8i 概要』で説明します。

---

## ビットマップ索引と EXPLAIN PLAN

EXPLAIN PLAN 出力で、タイプを示す BITMAP という語とともに索引行ソースが表示されます。次のサンプルの問合せと計画を考えてください。ここでは、表へのアクセスに必要な ROWID を生成するために TO ROWIDS オプションが使われています。

```
EXPLAIN PLAN FOR
SELECT * FROM T
WHERE
C1 = 2 AND C2 <> 6
OR
C3 BETWEEN 10 AND 20;

SELECT STATEMENT
TABLE ACCESS T BY ROWID
BITMAP CONVERSION TO ROWIDS
BITMAP OR
BITMAP MINUS
BITMAP MINUS
BITMAP INDEX C1_IND SINGLE VALUE
BITMAP INDEX C2_IND SINGLE VALUE
BITMAP INDEX C2_IND SINGLE VALUE
BITMAP MERGE
BITMAP INDEX C3_IND RANGE SCAN
```

この例では、述語 C1=2 によってビットマップが生成され、そこから減算が行われます。このビットマップから、C2=6 に対するビットマップ内のビットが減算されます。同様に、C2 IS NULL に対するビットマップ内のビットが減算され、この計画の中に 2 つの MINUS 行ソースがある理由が説明されます。NULL 減算は、列に NOT NULL 制約が付いていない限り、意味論上の正確さを保つために必要です。

## EXPLAIN PLAN とパーティション・オブジェクト

EXPLAIN PLAN を使用すると、特定の問合せのパーティション・オブジェクトに Oracle がアクセスする方法を参照できます。

プルニング後にアクセスされたパーティションは、PARTITION START 列と PARTITION STOP 列に表示されます。レンジ・パーティションの行ソース名は "PARTITION RANGE" です。ハッシュ・パーティションの場合、行ソース名は "PARTITION HASH" です。

結合される表のいずれかの計画表の DISTRIBUTION 列に "PARTITION(KEY)" がある場合、結合はパースャル・パーティション・ワイズ・ジョインを使用してインプリメントされます。パースャル・パーティション・ワイズ・ジョインが可能なのは、結合される表のいずれかが結合列でパーティション化されており、かつ、表が並列化されている場合です。

EXPLAIN PLAN 出力の結合行ソースの前にパーティション行ソースがある場合、結合はフル・パーティション・ワイズ・ジョインを使用してインプリメントされます。フル・パー

ティション・ワイズ・ジョインが可能なのは、両方の結合表がそれぞれの結合列でパーティション化されている場合だけです。次に、いくつかの種類のパーティションに対する実行計画の例を示します。

EXPLAIN PLAN によるレンジおよびハッシュ・パーティション化の表示

HIREDATE で範囲ごとにパーティション化されている次の EMP\_RANGE 表を検討し、プルニングの表示方法を例示します。標準 Oracle スキーマの表 EMP と DEPT が存在することを想定しています。

```
CREATE TABLE EMP_RANGE
PARTITION BY RANGE(HIREDATE)
(
PARTITION EMP_P1 VALUES LESS THAN (TO_DATE('1-JAN-1981','DD-MON-YYYY')),
PARTITION EMP_P2 VALUES LESS THAN (TO_DATE('1-JAN-1983','DD-MON-YYYY')),
PARTITION EMP_P3 VALUES LESS THAN (TO_DATE('1-JAN-1985','DD-MON-YYYY')),
PARTITION EMP_P4 VALUES LESS THAN (TO_DATE('1-JAN-1987','DD-MON-YYYY')),
PARTITION EMP_P5 VALUES LESS THAN (TO_DATE('1-JAN-1989','DD-MON-YYYY'))
)
AS SELECT * FROM EMP;
```

例 1:

```
EXPLAIN PLAN FOR SELECT * FROM EMP_RANGE;
```

EXPLAIN PLAN 出力を表示するために、次を入力します。

```
@?/RDBMS/ADMIN/UTLXPLS
```

次のようなものが表示されます。

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		105	8K	1		
PARTITION RANGE ALL					1	5
TABLE ACCESS FULL	EMP_RANGE	105	8K	1	1	5

6 rows selected.

パーティション行ソースは、表アクセス行ソースの上に作成されます。これが、アクセスされるパーティションのセット上を反復します。

例 1 では、述語がプルニングに使用されていないので、パーティション・イテレータはすべてのパーティション（オプション ALL）を対象とします。計画表の PARTITION\_START 列と PARTITION STOP 列は、1 から 5 までのすべてのパーティションへのアクセスを示します。

例 2:

```
EXPLAIN PLAN FOR SELECT * FROM EMP_RANGE
WHERE HIREDATE >= TO_DATE('1-JAN-1985','DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		3	54	1		
PARTITION RANGE ITERATOR					4	5
TABLE ACCESS FULL	EMP_RANGE	3	54	1	4	5

6 rows selected.

例 2 では、パーティション行ソースがパーティション 4 から 5 までを反復します。これは、HIREDATE で述語を使用してその他のパーティションをプルニングしたためです。

例 3:

```
EXPLAIN PLAN FOR SELECT * FROM EMP_RANGE
WHERE HIREDATE < TO_DATE('1-JAN-1981','DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		2	36	1		
TABLE ACCESS FULL	EMP_RANGE	2	36	1	1	1

5 rows selected.

例 3 では、パーティション 1 だけがアクセスされ、それがコンパイル時に認識されます。したがって、パーティション行ソースは必要ありません。

ハッシュ・パーティション化の計画

パーティション行ソース名が "PARTITION RANGE" ではなく "PARTITION HASH" であることを除き、Oracle はハッシュ・パーティション・オブジェクトに対して同じ情報を表示します。また、ハッシュ・パーティション化では、プルニングが可能なのは等価述語かインリスト述語を使用している場合だけです。

## コンポジット・パーティション・オブジェクトでの情報のプルニング

Oracle がコンポジット・パーティション・オブジェクトのプルニング情報を表示する方法を例示するために、HIREDATE でレンジ・パーティション化され、DEPTNO でハッシュごとにサブパーティション化された表 EMP\_COMP を検討します。

```
CREATE TABLE EMP_COMP PARTITION BY RANGE(HIREDATE) SUBPARTITION BY HASH(DEPTNO)
SUBPARTITIONS 3
(
  PARTITION EMP_P1 VALUES LESS THAN (TO_DATE('1-JAN-1981','DD-MON-YYYY')),
  PARTITION EMP_P2 VALUES LESS THAN (TO_DATE('1-JAN-1983','DD-MON-YYYY')),
  PARTITION EMP_P3 VALUES LESS THAN (TO_DATE('1-JAN-1985','DD-MON-YYYY')),
  PARTITION EMP_P4 VALUES LESS THAN (TO_DATE('1-JAN-1987','DD-MON-YYYY')),
  PARTITION EMP_P5 VALUES LESS THAN (TO_DATE('1-JAN-1989','DD-MON-YYYY'))
)
AS SELECT * FROM EMP;
```

**例 1:**

```
EXPLAIN PLAN FOR SELECT * FROM EMP_COMP;
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		105	8K	1		
PARTITION RANGE ALL					1	5
PARTITION HASH ALL					1	3
TABLE ACCESS FULL	EMP_COMP	105	8K	1	1	15

7 rows selected.

例 1 では、Oracle がコンポジット・パーティション・オブジェクトの全パーティションの全サブパーティションにアクセスする場合の実行計画を示しています。そのために、2 つのパーティション行ソースが使用されています。1 つはパーティションを反復するレンジ・パーティション行ソースで、もう 1 つはアクセスされる各パーティションのサブパーティションを反復するハッシュ・パーティション行ソースです。

この例では、プルニングが実行されていないので、レンジ・パーティション行ソースはパーティション 1 から 5 までは反復します。各パーティション内で、ハッシュ・パーティション行ソースは現在のパーティションのサブパーティション 1 から 3 までは反復します。その結果、表アクセス行ソースは、サブパーティション 1 から 15 まではアクセスします。つまり、コンポジット・パーティション・オブジェクトのすべてのサブパーティションにアクセスします。

例 2:

```
EXPLAIN PLAN FOR SELECT * FROM EMP_COMP WHERE HIREDATE =
TO_DATE('15-FEB-1987', 'DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		1	96	1		
PARTITION HASH ALL					1	3
TABLE ACCESS FULL	EMP_COMP	1	96	1	13	15

6 rows selected.

例 2 では、最後のパーティション 5 だけがアクセスされます。このパーティションはコンパイル時に認識されるので、計画内で表示する必要はありません。ハッシュ・パーティション行ソースは、そのパーティション内のすべてのサブパーティションのアクセスを表示します。つまりサブパーティション 1 から 3 ままで、これは EMP\_COMP 表のサブパーティション 13 から 15 までに変換されます。

例 3:

```
EXPLAIN PLAN FOR SELECT * FROM EMP_COMP WHERE DEPTNO = 20;
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		2	200	1		
PARTITION RANGE ALL					1	5
TABLE ACCESS FULL	EMP_COMP	2	200	1		

6 rows selected.

この例では、述語 "DEPTNO = 20" によって各パーティション内のハッシュ・ディメンションでブルニングが使用可能なので、Oracle は単一のサブパーティションにアクセスするだけで済みます。そのサブパーティションの番号はコンパイル時に認識されるので、ハッシュ・パーティション行ソースは必要ありません。



例 4:

```
VARIABLE DNO NUMBER;  
EXPLAIN PLAN FOR SELECT * FROM EMP_COMP WHERE DEPTNO = :DNO;
```

Plan Table

Operation	Name	Rows	Bytes	Cost	Pstart	Pstop
SELECT STATEMENT		2	200	1		
PARTITION RANGE ALL					1	5
PARTITION HASH SINGLE					KEY	KEY
TABLE ACCESS FULL	EMP_COMP	2	200	1		

7 rows selected.

例 4 は、"DEPTNO = 20" が "DEPTNO = :DNO" に置き換わったことを除き、例 3 と同じです。この場合、サブパーティションの番号はコンパイル時には不明であり、ハッシュ・パーティション行ソースが割り当てられます。Oracle が各パーティション内でアクセスするサブパーティションは 1 つだけなので、その行ソースのオプションは SINGLE です。PARTITION START と PARTITION STOP は "KEY" に設定されています。これは、Oracle がサブパーティションの番号を実行時に判別することを意味します。

パーシャル・パーティション・ワイズ・ジョイン

例 1:

この例では、EMP\_RANGE がパーティション化列で結合され、並列化されます。DEPT 表はパーティション化されていないので、これにより、パーシャル・パーティション・ワイズ・ジョインが使用可能になります。Oracle は結合前に DEPT 表を動的にパーティション化します。

```
ALTER TABLE EMP PARALLEL 2;  
STATEMENT PROCESSED.  
ALTER TABLE DEPT PARALLEL 2;  
STATEMENT PROCESSED.
```

問合せの計画を表示するには、次を入力します。

```
EXPLAIN PLAN FOR SELECT /*+ ORDERED USE_HASH(D) */ ENAME, DNAME  
FROM EMP_RANGE E, DEPT D  
WHERE E.DEPTNO = D.DEPTNO  
AND E.HIREDATE > TO_DATE('29-JUN-1986', 'DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	TQ	IN-OUT	PQ Distrib	Pstart	Pstop
SELECT STATEMENT		1	51	3					
HASH JOIN		1	51	3	2,02	P->S	QC (RANDOM)		
PARTITION RANGE ITERATOR					2,02	PCWP		4	5
TABLE ACCESS FULL	EMP_RANGE	3	87	1	2,00	PCWP		4	5
TABLE ACCESS FULL	DEPT	21	462	1	2,01	P->P	PART (KEY)		

8 rows selected.

この計画は、DIST 列にパーティション・キーを示すテキスト "PART (KEY)" があるので、オプティマイザがパーティション・ワイズ・ジョインを選択したことを示しています。

例 2:

例 2 では、EMP\_COMP がハッシュ・パーティション化列の DEPTNO で結合され、並列化されています。DEPT 表はパーティション化されていないので、これにより、パーシャル・パーティション・ワイズ・ジョインが使用可能になります。ここでも、Oracle は DEPT 表を動的にパーティション化します。

```
ALTER TABLE EMP_COMP PARALLEL 2;
STATEMENT PROCESSED.
EXPLAIN PLAN FOR SELECT /*+ ORDERED USE_HASH(D) */ ENAME, DNAME
FROM EMP_COMP E, DEPT D
WHERE E.DEPTNO = D.DEPTNO
AND E.HIREDATE > TO_DATE('13-MAR-1985','DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	TQ	IN-OUT	PQ Distrib	Pstart	Pstop
SELECT STATEMENT		1	51	3					
HASH JOIN		1	51	3	0,01	P->S	QC (RANDOM)		
PARTITION RANGE ITERATOR					0,01	PCWP		4	5
PARTITION HASH ALL					0,01	PCWP		1	3
TABLE ACCESS FULL	EMP_COMP	3	87	1	0,01	PCWP		10	15
TABLE ACCESS FULL	DEPT	21	462	1	0,00	P->P	PART (KEY)		

9 rows selected.

## フル・パーティション・ワイズ・ジョイン

次の例では、EMP\_COMP と DEPT\_HASH がハッシュ・パーティション化列で結合されます。これにより、フル・パーティション・ワイズ・ジョインが使用可能になります。  
"PARTITION HASH" 行ソースが、計画表出力の結合行ソースの上に表示されます。

表 DEPT\_HASH を作成するには、次を入力します。

```
CREATE TABLE DEPT_HASH
  PARTITION BY HASH(deptno)
  PARTITIONS 3
  PARALLEL
  AS SELECT * FROM DEPT;
```

問合せの計画を表示するには、次を入力します。

```
EXPLAIN PLAN FOR SELECT /*+ ORDERED USE_HASH(D) */ ENAME, DNAME
  FROM EMP_COMP E, DEPT_HASH D
  WHERE E.DEPTNO = D.DEPTNO
  AND E.HIREDATE > TO_DATE('29-JUN-1986','DD-MON-YYYY');
```

Plan Table

Operation	Name	Rows	Bytes	Cost	TQ	IN-OUT	PQ Distrib	Pstart	Pstop
SELECT STATEMENT		2	102	2					
PARTITION HASH ALL					4,00	PCWP		1	3
HASH JOIN		2	102	2	4,00	P->S	QC (RANDOM)		
PARTITION RANGE ITERATOR					4,00	PCWP		4	5
TABLE ACCESS FULL	EMP_COMP	3	87	1	4,00	PCWP		10	15
TABLE ACCESS FULL	DEPT_HASH	63	1K	1	4,00	PCWP		1	3

9 rows selected.

## INLIST ITERATOR と EXPLAIN PLAN

INLIST ITERATOR 操作は、索引が IN リスト述語をインプリメントする場合に、EXPLAIN PLAN 出力に表示されます。たとえば、次のような問合せがあるとします。

```
SELECT * FROM EMP WHERE EMPNO IN (7876, 7900, 7902);
```

EXPLAIN PLAN 出力は次のようになります。

OPERATION	OPTIONS	OBJECT_NAME
-----	-----	-----
SELECT STATEMENT		
INLIST ITERATOR		
TABLE ACCESS	BY ROWID	EMP
INDEX	RANGE SCAN	EMP_EMPNO

INLIST ITERATOR 操作は、IN リスト述語内の各値に対して、INLIST ITERATOR の下に表示された操作を反復します。パーティション表およびパーティション索引の場合は、IN リスト列には、次に説明する 3 種類の可能性があります。

### 索引列

IN リスト列 EMPNO は索引列で、パーティション列ではないため、計画は次のようになります (IN リスト演算子は表操作の上に表示されますが、パーティションの操作よりは下に表示されます)。

OPERATION	OPTIONS	OBJECT_NAME	PARTITION_START	PARTITION_STOP
-----	-----	-----	-----	-----
SELECT STATEMENT				
PARTITION	INLIST		KEY (INLIST)	KEY (INLIST)
INLIST ITERATOR				
TABLE ACCESS	BY ROWID	EMP	KEY (INLIST)	KEY (INLIST)
INDEX	RANGE SCAN	EMP_EMPNO	KEY (INLIST)	KEY (INLIST)

パーティションの開始キーと終了キーに対して KEY(INLIST) 指定があるため、索引の開始 / 終了キーに対して IN リスト述語が表示されます。

### 索引とパーティション列

EMPNO が索引付けされている列で、それがパーティション列でもある場合は、計画ではパーティション操作の上に INLIST ITERATOR 操作が含まれます。

OPERATION	OPTIONS	OBJECT_NAME	PARTITION_START	PARTITION_STOP
-----	-----	-----	-----	-----
SELECT STATEMENT				
INLIST ITERATOR				
PARTITION	ITERATOR		KEY (INLIST)	KEY (INLIST)
TABLE ACCESS	BY ROWID	EMP	KEY (INLIST)	KEY (INLIST)
INDEX	RANGE SCAN	EMP_EMPNO	KEY (INLIST)	KEY (INLIST)

### パーティション列

EMPNO がパーティション列で、索引がない場合は、INLIST ITERATOR 操作は割り当てられません。

OPERATION	OPTIONS	OBJECT_NAME	PARTITION_START	PARTITION_STOP
-----	-----	-----	-----	-----
SELECT STATEMENT				
PARTITION			KEY (INLIST)	KEY (INLIST)
TABLE ACCESS	FULL	EMP	KEY (INLIST)	KEY (INLIST)

EMP\_EMPNO がビットマップ索引である場合、計画は次のとおりです。

OPERATION	OPTIONS	OBJECT_NAME
-----	-----	-----
SELECT STATEMENT		
INLIST ITERATOR		
TABLE ACCESS	BY INDEX ROWID	EMP
BITMAP CONVERSION	TO ROWIDS	
BITMAP INDEX	SINGLE VALUE	EMP_EMPNO

### ドメイン索引と EXPLAIN PLAN

また、EXPLAIN PLAN を使用すると、ドメイン索引に対するユーザー定義の CPU および I/O コストを導出できます。EXPLAIN PLAN は、PLAN\_TABLE の "OTHER" 列にこれらの統計を表示します。

たとえば、RESUME 列にドメイン索引 EMP\_RESUME を持つユーザー定義オペレータ CONTAINS が表 EMP にあり、EMP\_RESUME の索引タイプがオペレータ CONTAINS をサポートしているとします。次の問合せをします。

```
SELECT * from EMP where Contains(resume, 'Oracle') = 1
```

すると、次のような計画が表示されます。

OPERATION	OPTIONS	OBJECT_NAME	OTHER
-----	-----	-----	-----
SELECT STATEMENT			
TABLE ACCESS	BY ROWID	EMP	
DOMAIN INDEX		EMP_RESUME	CPU: 300, I/O: 4

## EXPLAIN PLAN 出力のフォーマット

この項では、EXPLAIN PLAN 出力のフォーマットのオプションについて説明します。

- [EXPLAIN PLAN 文の使用](#)
- [表フォーマットでの PLAN\\_TABLE 出力の選択](#)
- [ネストされたフォーマットでの PLAN\\_TABLE 出力の選択](#)

**注意：** EXPLAIN PLAN 文の出力は、Oracle オプティマイザの動作を反映します。オプティマイザの動作は Oracle Server がリリースされるたびに拡張されるので、EXPLAIN PLAN 文の出力も拡張されます。

## EXPLAIN PLAN 文の使用

次の例は、SQL 文と、この文に対応して EXPLAIN PLAN で生成された実行計画を示します。このサンプルの問合せは、SALGRADE 表で示される範囲内に給与が存在しない従業員の名前と関連情報を検索します。

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

この EXPLAIN PLAN 文は実行計画を生成して、その出力を PLAN\_TABLE に格納します。

```
EXPLAIN PLAN
SET STATEMENT_ID = 'Emp_Sal'
FOR SELECT ename, job, sal, dname
FROM emp, dept
WHERE emp.deptno = dept.deptno
AND NOT EXISTS
  (SELECT *
   FROM salgrade
   WHERE emp.sal BETWEEN losal AND hisal);
```

## 表フォーマットでの PLAN\_TABLE 出力の選択

次の SELECT 文を使用します。

```
SELECT operation, options, object_name, id, parent_id, position, cost, cardinality,
other_tag, optimizer
FROM plan_table
WHERE statement_id = 'Emp_Sal'
ORDER BY id;
```

すると、次のような出力が生成されます。

OPERATION	OPTIONS	OBJECT_NAME	ID	PARENT_ID	POSITION	COST	CARDINALITY	BYTES	OTHER_TAG	OPTIMIZER
SELECT STATEMENT			0			2	2	1	62	CHOOSE
FILTER			1	0		1				
NESTED LOOPS			2	1		1	2	1	62	
TABLE ACCESS FULL		EMP	3	2		1	1	1	40	ANALYZED
TABLE ACCESS FULL		DEPT	4	2		2		4	88	ANALYZED
TABLE ACCESS FULL		SALGRADE	5	1		2	1	1	13	ANALYZED

ORDER\_BY 句は、ID 値によって順に実行計画のステップを戻します。ただし、これらのステップはこの順序では実行されません。PARENT\_ID は ID から情報を受け取るので、複数の ID ステップが 1 つの PARENT\_ID に対応します。

たとえば、ステップ 2 のネスト・ループ・ジョインとステップ 5 の表アクセスは、どちらもステップ 1 に対応しています。次の項では、処理順序をネストした表現で示します。

出力の最初の行に対する POSITION 列の値は、オプティマイザがこの実行計画でこの文を実行した場合のコストを 5 と見積もっていることを示します。他の行については、同じ親を持つ他の子への相対位置を示しています。

**注意：** CONNECT BY では、順序は保たれません。この例で行が正しい順序で出力されるようにするには、最初に表を切り捨てるか、またはビューを作成してそのビューから選択しなければなりません。次に例を示します。

```
CREATE VIEW test AS
SELECT id, parent_id,
lpad(' ', 2*(level-1))||operation||' '||options||' '||object_name||' '||
decode(id, 0, 'Cost = '||position) "Query Plan"
FROM plan_table
START WITH id = 0 and statement_id = 'TST'
CONNECT BY prior id = parent_id and statement_id = 'TST';
SELECT * FROM foo ORDER BY id, parent_id;
```

この結果、次のようになります。

```
ID  PAR Query Plan
---  -
0    Select Statement   Cost = 69602
1  0    Nested Loops
2  1      Nested Loops
3  2          Merge Join
4  3              Sort Join
5  4                  Table Access Full T3
6  3                      Sort Join
7  6                          Table Access Full T4
8  2                              Index Unique Scan T2
9  1                                  Table Access Full T1
10 rows selected.
```

## ネストされたフォーマットでの PLAN\_TABLE 出力の選択

このタイプの SELECT 文は、ネスト形式の出力を生成し、SQL 文の処理順序をよりわかりやすく表現します。

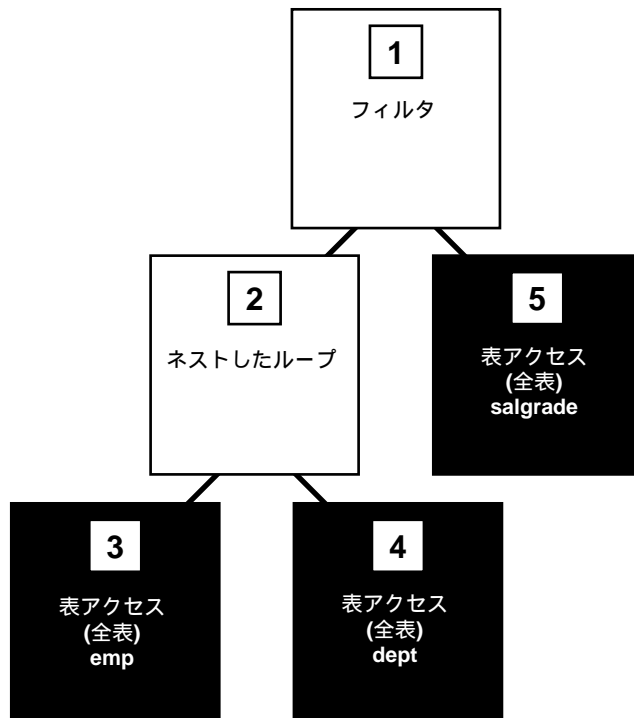
```
SELECT LPAD(' ',2*(LEVEL-1))||operation||' '||options
       ||' '||object_name
       ||' '||DECODE(id, 0, 'Cost = '||position) "Query Plan"
FROM plan_table
START WITH id = 0 AND statement_id = 'Emp_Sal'
CONNECT BY PRIOR id = parent_id AND statement_id = 'Emp_Sal';
```

```
Query Plan
-----
SELECT STATEMENT   Cost = 5
  FILTER
    NESTED LOOPS
      TABLE ACCESS FULL EMP
      TABLE ACCESS FULL DEPT
      TABLE ACCESS FULL SALGRADE
```



この順序は、図 13-1 に示すツリー構造に類似しています。

図 13-1 実行計画のツリー構造



ツリー構造は、SQL 文の実行中に発生する処理が別の処理にその結果を送る様子を示しています。実行計画の各ステップには PLAN\_TABLE の ID 列を示す数値が割り当てられます。各ステップは " ノード " と呼ばれます。各ノードの処理結果はその親ノードに渡され、入力として使用されます。

## EXPLAIN PLAN の制限事項

EXPLAIN PLAN は、日付バインド変数の型変換を暗黙のうちに実行する文をサポートしません。一般にバインド変数では、EXPLAIN PLAN 出力は実際の実行計画を表さないことがあります。

SQL 文のテキストから TKPROF がバインド変数のタイプを判断することはできません。タイプは CHARACTER であると想定され、これ以外の場合はエラー・メッセージが表示されます。この制限事項は、SQL 文に適切な型変換を入れることで対処できます。

**関連項目：** [第 14 章「SQL トレース機能と TKPROF」](#)

---

## SQL トレース機能と TKPROF

SQL トレース機能および TKPROF は、Oracle Server のもとで実行されるアプリケーションの監視と調整に役立つ 2 つの基本的なパフォーマンス診断ツールです。この章では次の問題を取り上げます。

- [SQL トレース機能と TKPROF の概要](#)
- [ステップ 1: トレース・ファイル管理用の初期化パラメータの設定](#)
- [ステップ 2: SQL トレース機能の使用可能化](#)
- [ステップ 3: TKPROF によるトレース・ファイルのフォーマット](#)
- [ステップ 4: TKPROF 出力の解釈](#)
- [ステップ 5: SQL トレース機能統計の格納](#)
- [TKPROF の解釈における落とし穴の回避](#)
- [TKPROF の出力例](#)

### SQL トレース機能と TKPROF の概要

SQL トレース機能および TKPROF を使うと、アプリケーションが実行する SQL 文の効率を正確に評価できます。最善の結果を得るには、EXPLAIN PLAN 単体ではなくこれらのツールとともに使用してください。この項では次の内容を取り上げます。

- [SQL トレース機能について](#)
- [TKPROF について](#)
- [SQL トレース機能と TKPROF の使用](#)

## SQL トレース機能について

SQL トレース機能は、個々の SQL 文に関するパフォーマンス情報を提供します。SQL トレース機能は、文単位に次の統計を生成します。

- 解析、実行、フェッチのカウント
- CPU 時間と経過時間
- 物理読み込みと論理読み込み
- 処理された行数
- ライブラリ・キャッシュでのミス
- それぞれの解析が行われるユーザー名
- 各コミットおよびロールバック

セッションまたはインスタンスに対して、SQL トレース機能を使用可能にできます。SQL トレース機能が使用可能にされると、ユーザー・セッションまたはインスタンスで実行されるすべての SQL 文のパフォーマンス統計がトレース・ファイルに格納されます。

パフォーマンスに問題のあるアプリケーションに対して SQL トレース機能を実行する際のオーバーヘッドは、アプリケーションの非効率性から発生するオーバーヘッドと比較すると通常わずかです。

## TKPROF について

TKPROF プログラムを実行すると、トレース・ファイルの内容をフォーマットし判読可能なファイルとして出力します。オプションとして、TKPROF は次のことも実行します。

- SQL 文の実行計画を判断します。
- 統計をデータベースに格納する SQL スクリプトを作成します。

TKPROF は、実行した各文を、消費したリソースおよびコールした回数、処理した行数とともにレポートします。この情報を使うと、リソースを最も多く使っている文を簡単に検出できます。経験または参考にできる基準をもとに、使われたリソースが実行された作業に対して妥当であるかどうかを評価できます。

## SQL トレース機能と TKPROF の使用

SQL トレース機能および TKPROF を使うには、次のステップに従います。

1. トレース・ファイル管理用の初期化パラメータを設定します。
2. 対象とするセッションに対して SQL トレース機能を使用可能にして、アプリケーションを実行します。このステップでは、アプリケーションによって発行された SQL 文に関する統計を含むトレース・ファイルが作成されます。

3. ステップ 2 で作成されるトレース・ファイルを判読可能な出力ファイルに変換するために、TKPROF を実行します。このステップではオプションとして、データベースに統計を保存する SQL スクリプトを作成できます。
4. ステップ 3 で作成した出力ファイルを解釈します。
5. 任意で、ステップ 3 で作成した SQL スクリプトを実行してデータベースに統計を格納します。

以降の項では、これらの各ステップについて詳しく説明します。

## ステップ 1: トレース・ファイル管理用の初期化パラメータの設定

セッションに対して SQL トレース機能が使用可能になると、Oracle はトレース・ファイルを生成します。このファイルには、そのセッションのトレースされた SQL 文に関する統計が記録されています。インスタンスに対して SQL トレース機能が使用可能になると、Oracle はプロセスごとに個別のトレース・ファイルを作成します。

SQL トレース機能を使用可能にする前に、次のことを行う必要があります。

1. TIMED\_STATISTICS、USER\_DUMP\_DEST、MAX\_DUMP\_FILE\_SIZE パラメータの設定をチェックします。

表 14-1 SQL トレース機能動的初期化パラメータ

パラメータ	説明
TIMED_STATISTICS	このパラメータは、SQL トレース機能による CPU 時間や経過時間などの時間統計の収集、および動的パフォーマンス表の中のさまざまな統計の収集を使用可能または使用禁止にします。デフォルト値は FALSE であり、時間計測は使用禁止になっています。値 TRUE によって時間計測が使用可能になります。時間計測を使用可能にすると、下位レベル操作に対する時間計測呼出しが余分に発生します。これはセッション・パラメータです。
MAX_DUMP_FILE_SIZE	SQL トレース機能がインスタンス・レベルで使用可能にされているときは、サーバーに対するすべてのコールはオペレーティング・システムのファイル形式でテキスト行を生成します。これらのファイルの最大サイズ（オペレーティング・システム・ブロック単位）は、初期化パラメータ MAX_DUMP_FILE_SIZE によって制限されます。デフォルト値は 500 です。トレース出力が切り捨てられている場合、別のトレース・ファイルを生成する前にこのパラメータの値を大きくしてください。これはセッション・パラメータです。
USER_DUMP_DEST	このパラメータで、オペレーティング・システムの規則に従って、トレース・ファイルの出力先をフルパスで指定する必要があります。このパラメータのデフォルト値は、オペレーティング・システムのシステム・ダンプのデフォルトの出力先です。この値は、ALTER SYSTEM SET USER_DUMP_DEST=newdir を使用して変更できます。これはシステム・パラメータです。

2. 結果のトレース・ファイルを認識する方法を考えます。

トレース・ファイルを名前で区別できるようにしてください。Oracle は、USER\_DUMP\_DEST で指定されたユーザー・ダンプ出力先にこれらを書き込みます。ただし、このディレクトリには、通常は生成された名前を持つ何百ものファイルがすぐに含まれます。トレース・ファイルを生成元のセッションやプロセスと合致させることは困難かもしれません。SELECT 'program name' FROM DUAL のような文をプログラムに組み込むことによって、トレース・ファイルにタグを付けることができます。これによって、各ファイルの生成元のプロセスを追跡できます。

3. オペレーティング・システムがファイルの複数のバージョンを保持している場合、SQL トレース機能が生成するトレース・ファイルの数に対して、バージョンの制限が十分高いことを確認してください。
4. 生成されたトレース・ファイルの所有者は、データベース管理者以外のオペレーティング・システム・ユーザーです。データベース管理者が TKPROF を実行してこれらのファイルをフォーマットする場合は、このユーザーが前もって、管理者がトレース・ファイルを利用できる状態にしておく必要があります。

## ステップ 2: SQL トレース機能の使用可能化

セッションまたはインスタンスに対して、SQL トレース機能を使用可能にできます。この項では次の内容を取り上げます。

- [現行セッションの SQL トレース機能の使用可能化](#)
- [インスタンスに対する SQL トレース機能の使用可能化](#)

---

**注意：** SQL トレース機能を実行するとシステムのオーバーヘッドが増加するので、この機能は SQL 文を調整するときのみ使用可能にし、調整が終了したら使用禁止にする必要があります。

---

### 現行セッションの SQL トレース機能の使用可能化

現行セッションに対して SQL トレース機能を使用可能にするには、次のように入力します。

```
ALTER SESSION SET SQL_TRACE = TRUE;
```

あるいは、DBMS\_SESSION.SET\_SQL\_TRACE プロシージャを使って、セッションに対して SQL トレース機能を使用可能にすることもできます。

SQL トレース機能を使用禁止にするには、次のように入力します。

```
ALTER SESSION SET SQL_TRACE = FALSE;
```

アプリケーションが Oracle との接続を切断すると、そのセッションの SQL トレース機能は自動的に使用禁止になります。

---

**注意：** ALTER SESSION 文を含めるには、アプリケーションを修正する必要があります。たとえば、Oracle Forms で ALTER SESSION 文を発行するには、-s オプションを指定して Oracle Forms を起動するか、または統計オプションを指定して Oracle Forms(Design) を起動してください。Oracle Forms の詳細は、『Oracle Forms Reference』を参照してください。

---

## インスタンスに対する SQL トレース機能の使用可能化

インスタンスに対して SQL トレース機能を使用可能にするには、SQL\_TRACE 初期化パラメータの値を TRUE に設定します。すべてのセッションに関する統計が収集されます。

インスタンスに対して SQL トレース機能を使用可能にした後で、次のように入力することで個々のセッションに対してトレース機能を使用禁止にできます。

```
ALTER SESSION SET SQL_TRACE = FALSE;
```

## ステップ 3: TKPROF によるトレース・ファイルのフォーマット

この項では次の内容を取り上げます。

- [TKPROF の出力例](#)
- [TKPROF の構文](#)
- [TKPROF 文の例](#)

TKPROF は、SQL トレース機能によって生成されたトレース・ファイルを入力として、フォーマットされた出力ファイルを生成します。TKPROF は、実行計画を生成するためにも使用できます。

SQL トレース機能によって多数のトレース・ファイルが生成されると、次の処理を実行できます。

- 各トレース・ファイルごとに TKPROF を実行して、フォーマットした出力ファイルを各セッションに 1 つずつ作成する。
- トレース・ファイルを連結し、その結果のファイルに対して TKPROF を実行して、インスタンス全体のフォーマットした出力ファイルを生成する。

TKPROF は、トレース・ファイルに記録されている COMMIT および ROLLBACK をレポートしません。

## TKPROF の出力例

TKPROF の出力例は次のようになります。

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

call	count	cpu	elapsed	disk	query current	rows
Parse	1	0.16	0.29	3	13	0
Execute	1	0.00	0.00	0	0	0
Fetch	1	0.03	0.26	2	2	4

Misses in library cache during parse: 1  
Parsing user id: (8) SCOTT

Rows	Execution Plan
14	MERGE JOIN
4	SORT JOIN
4	TABLE ACCESS (FULL) OF 'DEPT'
14	SORT JOIN
14	TABLE ACCESS (FULL) OF 'EMP'

この文では、TKPROF 出力は次の情報を含みます。

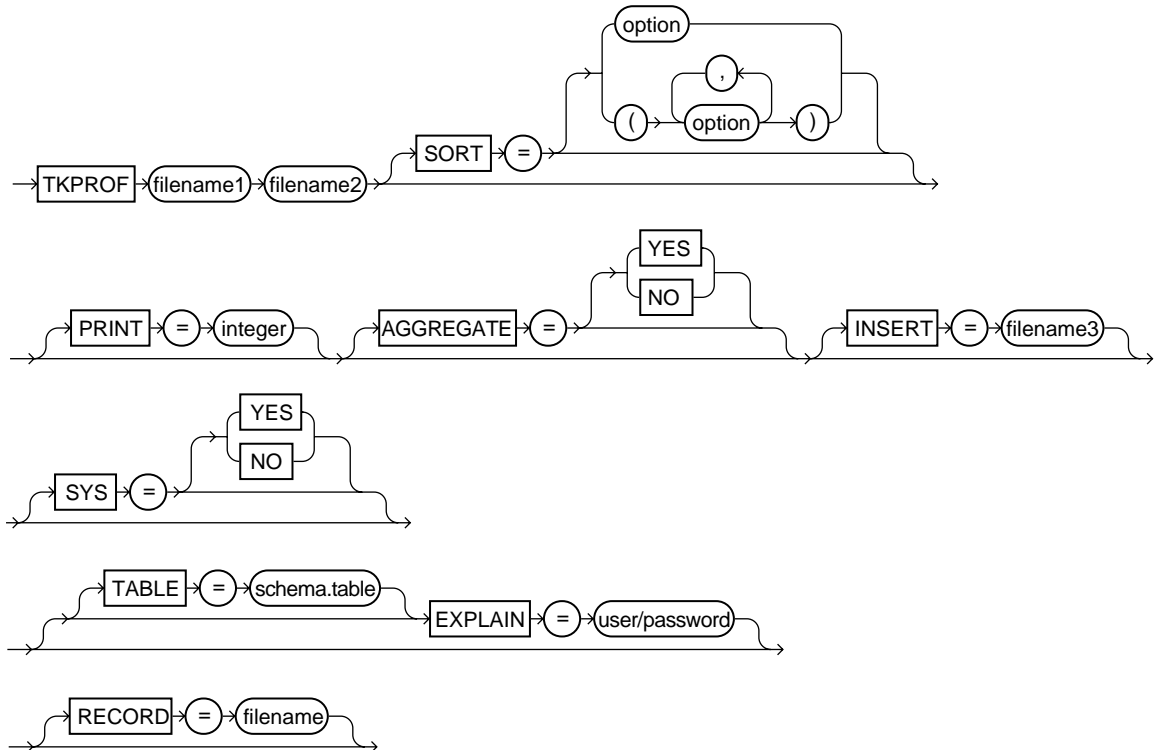
- SQL 文のテキスト
- 表形式で示された SQL トレース統計
- 文の解析と実行におけるライブラリ・キャッシュ・ミスの回数
- 文を最初に解析したユーザー
- EXPLAIN PLAN によって生成された実行計画

TKPROF は、トレース・ファイルのユーザー・レベル文と再帰的 SQL コールの要約も提供します。



## TKPROF の構文

TKPROF は次の構文を使用して呼び出します。



引数を指定しないで TKPROF を呼び出すと、オンライン・ヘルプが表示されます。

以下の引数は、TKPROF を実行するために使用されます。

**表 14-2 TKPROF の引数**

引数	意味
filename1	入力ファイル、つまり SQL トレース機能によって生成された統計を収録しているトレース・ファイルを指定します。このファイルは、単一のセッションに対して生成されたトレース・ファイル、または複数のセッションの個々のトレース・ファイルを結合して生成したファイルのどちらでもかまいません。
filename2	出力ファイルを指定します。TKPROF はフォーマットした出力を作成します。

表 14-2 TKPROF の引数

引数	意味
SORT	トレースした SQL 文を出力ファイルにリストする前に、指定したソート・オプションに基づいて降順にソートします。複数のオプションが指定されている場合、出力はソート・オプションに指定されている値の合計によって降順にソートされます。このパラメータを指定しないと、TKPROF は出力ファイルにそれぞれの文を使われた順にリストします。ソート・オプションは次のとおりです。
PRSCNT	解析回数
PRSCPU	解析に費やされた CPU 時間
PRSELA	解析に費やされた経過時間
PRSDSK	解析中のディスクに対する物理読み込みの回数
PRSMIS	解析中の一貫モード・ブロック読み込みの回数
PRSCU	解析中の現行モード・ブロック読み込みの回数
PRSMIS	解析中のライブラリ・キャッシュ・ミスの回数
EXECNT	実行回数
EXECPU	実行に費やされた CPU 時間
EXEELA	実行に費やされた経過時間
EXEDSK	実行中のディスクに対する物理読み込みの回数
EXEQRY	実行中の一貫モード・ブロック読み込みの回数
EXECU	実行中の現行モード・ブロック読み込みの回数
EXEROW	実行中に処理された行数
EXEMIS	実行中のライブラリ・キャッシュ・ミスの回数
FCHCNT	フェッチ回数
FCHCPU	フェッチに費やされた CPU 時間
FCHELA	フェッチに費やされた経過時間
FCHDSK	フェッチ中のディスクに対する物理読み込みの回数
FCHQRY	フェッチ中の一貫モード・ブロック読み込みの回数
FCHCU	フェッチ中の現行モード・ブロック読み込みの回数
FCHROW	フェッチされた行数

表 14-2 TKPROF の引数

引数	意味
PRINT	ソートされた SQL 文の先頭から、 <i>integer</i> に指定した数だけ出力ファイルにリストします。このパラメータを指定しないと、TKPROF はトレースした SQL 文をすべて出力ファイルにリストします。このパラメータは、INSERT オプションの SQL スクリプトには影響しません。SQL スクリプトは、常にトレースされたすべての SQL 文に関する統計を挿入します。
AGGREGATE	AGGREGATE = NO を指定すると、TKPROF は同じ SQL テキストについて複数ユーザーを集計しません。
INSERT	トレース・ファイルの統計をデータベースに格納する SQL スクリプトを作成します。TKPROF は、 <i>filename3</i> に指定される名前でこのスクリプトを作成します。このスクリプトは表を作成し、トレースされた各 SQL 文の統計が入っている行をこの表に挿入します。
SYS	ユーザー SYS が発行した SQL 文または再帰的 SQL 文の出力ファイルへのリストを使用可能または使用禁止にします。デフォルト値は YES で、TKPROF がこれらの SQL 文をリストします。値 NO が指定されると、TKPROF はこれらの SQL 文をリストしません。このパラメータは、INSERT オプションの SQL スクリプトには影響しません。SQL スクリプトは、常にトレースされたすべての SQL 文 (再帰的 SQL 文を含む) に関する統計を挿入します。
TABLE	実行計画が出力ファイルに書き込まれる前に、TKPROF が一時的にこれらの実行計画を格納しておく表のスキーマと名前を指定します。指定された表がすでに存在している場合は、TKPROF はその表の行をすべて削除して、(より多くの行を表に書き込む) EXPLAIN PLAN 文でその表を使用した後で、その表の行をすべて削除します。指定した表が存在しない場合は、TKPROF はこの表を作成して使用し、使用後にこの表を削除します。指定されるユーザーは、表に対して INSERT、SELECT、DELETE 文を発行できなければなりません。表がまだ存在しない場合は、この指定されるユーザーが前述の文に加えて CREATE TABLE 文と DROP TABLE 文も発行できなければなりません。これらの文を発行するための権限については、『Oracle8i SQL リファレンス』を参照してください。このオプションを指定すると、複数のユーザーが、EXPLAIN の値に指定されている同一のユーザーで同時に TKPROF を実行できます。これらの複数ユーザーが、TABLE に個別に異なる値を指定しておくことで、一時計画表の処理時にお互いのデータを破壊するような状況が発生することを防ぐことができます。EXPLAIN パラメータを指定して TABLE パラメータを指定しないと、TKPROF は、EXPLAIN パラメータで指定されたユーザーのスキーマ内の表 PROF\$PLAN_TABLE を使用します。TABLE パラメータを指定して EXPLAIN パラメータを指定しないと、TKPROF は TABLE パラメータを無視します。
RECORD	トレース・ファイル内の非再帰的 SQL 文をすべて収録した SQL スクリプトを、指定したファイル名で作成します。トレース・ファイルのユーザー・イベントを再実行する場合に、このオプションを指定できます。
EXPLAIN	トレース・ファイルの各 SQL 文の実行計画を決定して、これらの実行計画を出力ファイルに書き込みます。TKPROF は、このパラメータに指定されるユーザーとパスワードで Oracle に接続した後で、EXPLAIN PLAN 文を発行して実行計画を決定します。指定されるユーザーは、CREATE SESSION システム権限を持っている必要があります。EXPLAIN オプションが使われている場合は、TKPROF が大きなトレース・ファイルを処理するのに要する時間が長くなります。

## TKPROF 文の例

この項では、2 つの簡単な TKPROF の使用例を示します。TKPROF 出力の詳細な例は、14-21 ページの「[TKPROF の出力例](#)」を参照してください。

### 例 1

SORT パラメータと PRINT パラメータを組み合わせて大規模なトレース・ファイル进行处理する場合は、リソースを最も多く使う文だけを含む TKPROF 出力ファイルを生成できます。たとえば、次の文は、トレース・ファイルに格納されている、ほとんどの物理 I/O を生成した 10 個の文を印刷します。

```
TKPROF ora53269.trc ora 53269.prf
SORT = (PRSDSK, EXEDSK, FCHDSK)
PRINT = 10
```

### 例 2

この例では、TKPROF を実行して、"dlsun12\_jane\_fg\_svrngr\_007.trc" というトレース・ファイルを取り込み、"outputa.prf" というフォーマットされた出力ファイルに書き込みます。

```
TKPROF DLSUN12_JANE_FG_SVRNGR_007.TRC OUTPUTA.PRF
EXPLAIN=SCOTT/TIGER TABLE=SCOTT.TEMP_PLAN_TABLE_A INSERT=STOREA.SQL SYS=NO
SORT=(EXECPUR,FCHCPU)
```

スクリーン上では、この例は複数行にわたって表示される可能性があるのですが、使っているオペレーティング・システムによっては継続文字を使用しなければならない場合もあります。

この例で使用されている他のパラメータに注意してください。

- EXPLAIN の値によって、TKPROF はユーザー SCOTT として接続し、EXPLAIN PLAN 文を使用して、トレースされた各 SQL 文の実行計画を生成します。これを使ってアクセス・パスおよび行ソース件数を取得できます。
- TABLE の値によって、TKPROF はスキーマ SCOTT の表 TEMP\_PLAN\_TABLE\_A を一時計画表として使用します。
- INSERT の値によって、TKPROF はトレースされたすべての SQL 文に関する統計をデータベース内に格納する SQL スクリプト、STOREA.SQL を生成します。
- SYS パラメータに値 NO が指定されているため、TKPROF は出力ファイルに再帰的 SQL 文をリストしません。そのため、一時表操作などの内部 Oracle 文は無視できます。
- SORT の値によって、TKPROF は SQL 文を出力ファイルに書き込む前に、SQL 文の実行にかかった CPU 時間と行のフェッチにかかった CPU 時間の合計値の順に SQL 文をソートします。効率を最大化するために、SORT パラメータを常に使ってください。

## ステップ 4: TKPROF 出力の解釈

この項では、TKPROF 出力を解釈するためのヒントを示します。

- 表形式の統計
- ライブラリ・キャッシュ・ミス
- 文の切捨て
- SQL 文を発行するユーザー
- 実行計画
- どの文を調整するか判断

TKPROF は非常に有用な分析を提供しますが、効率の最も正確な尺度は対象アプリケーションの実際のパフォーマンスです。TKPROF 出力の最後に、トレース実行期間中にプロセスがデータベース・エンジンで実行した作業のサマリーがあります。

**関連項目：** V\$SYSSTAT および V\$SESSTAT に含まれる統計の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

### 表形式の統計

TKPROF は、SQL トレース機能によって戻される SQL 文の統計を行と列でリストします。各行は、SQL 文を処理する 3 つのステップの 1 つに対応します。各行はステップごとの統計を収録しており、ステップは次のように CALL 列の値によって識別されます。

PARSE	このステップでは、SQL 文を実行計画に変換します。このステップには、適切なセキュリティ認可のチェックと、表、列およびその他の参照オブジェクトの存在のチェックが含まれます。
EXECUTE	このステップでは、Oracle によって文が実際に実行されます。INSERT 文、UPDATE 文および DELETE 文の場合、このステップはデータを修正します。SELECT 文の場合、このステップは選択された行を識別します。
FETCH	このステップでは、問合せを満足する行を検索します。フェッチは SELECT 文についてのみ実行されます。

SQL トレース機能の出力におけるその他の列は、文の解析、実行、フェッチについて組み合わせた統計です。TIMED\_STATISTICS がオンでない場合は、これらの値は 0 (ゼロ) になります。QUERY と CURRENT の合計が、アクセスされたバッファの総数となります。

COUNT	文が解析、実行またはフェッチされた回数です。
CPU	文に対するすべての解析コール、実行コールまたはフェッチ・コールにかかった CPU 時間の合計 (単位は秒) です。

ELAPSED	文に対するすべての解析コール、実行コールまたはフェッチ・コールにかかった経過時間の合計（単位は秒）です。
DISK	すべての解析コール、実行コールまたはフェッチ・コールに対して、ディスク上のデータ・ファイルから物理的に読み込んだデータ・ブロックの総数です。
QUERY	すべての解析コール、実行コールまたはフェッチ・コールに対して、一貫モードで検索されたバッファの総数です。通常バッファは問合せに対して一貫モードで検索されます。
CURRENT	現行モードで検索されたバッファの総数です。INSERT、UPDATE、DELETE などの文では、バッファは現行モードで検索されます。

## Rows

処理された行に関する統計は、ROWS 列に表示されます。

ROWS	SQL 文によって処理された行の総数です。この値には、SQL 文の副問合せによって処理された行は含まれません。
------	---

SELECT 文の場合、戻された行数はフェッチ・ステップに表示されます。UPDATE 文、DELETE 文および INSERT 文の場合、処理された行数は実行ステップに表示されます。

---

---

**注意：** 行ソースの件数は、カーソルがクローズされたときに表示されます。SQL\*Plus では、ユーザー・カーソルは 1 つしかないので、文が実行されるたびに直前のカーソルがクローズされます。これにより、行ソースの件数が表示されます。PL/SQL には、独自のカーソル処理方法があり、親カーソルがクローズされても子カーソルはクローズされません。終了（または再接続）によって、件数が表示されます。

---

---

## 統計の分解能

タイミング統計の分解能は 100 分の 1 秒なので、100 分の 1 以下のカーソル操作は正確に計測できません。統計を解釈するときには、このことを覚えておいてください。非常に高速に実行する単純な問合せの結果を解釈するときには特に注意してください。

## 再帰的コール

ユーザーが発行した SQL 文を実行するために、Oracle は追加の文を発行しなければならないことがあります。このような文を再帰的コールまたは再帰的 SQL 文といいます。たとえば、十分な領域のない表に行を挿入しようすると、Oracle は再帰的コールを実行して動的に領域を割り当てます。データ・ディクショナリの情報がデータ・ディクショナリ・キャッシュで利用できないため、ディスクから検索しなければならない場合にも、再帰的コールが生成されます。

SQL トレース機能が使用可能になっているときに、再帰的コールが発生すると、TKPROF は再帰的コールの原因となった文に加えて再帰的 SQL 文の統計を表示します。コマンド行パラメータ SYS を NO に設定することによって、再帰的コールの出力ファイルへのリストを抑制できます。再帰的 SQL 文の統計は、再帰的コールの原因となった SQL 文のリストではなくその再帰的 SQL 文のリストに含まれます。したがって、SQL 文の処理に必要なリソースの合計を計算するときは、その文を原因とする再帰的コールの統計とともにその文自体の統計も考慮する必要があります。

## ライブラリ・キャッシュ・ミス

TKPROF では、各 SQL 文の解析ステップと実行ステップの結果として生じるライブラリ・キャッシュ・ミスの回数もリストします。これらの統計は、表形式の統計に続く別の行に表示されます。文でライブラリ・キャッシュ・ミスが発生しなかった場合、TKPROF はこの統計をリストしません。14-6 ページの「[TKPROF の出力例](#)」では、解析ステップではライブラリ・キャッシュ・ミスが 1 回発生し、実行ステップではライブラリ・キャッシュ・ミスが発生しませんでした。

## 文の切捨て

次の SQL 文は、SQL トレース・ファイルでは 25 文字に切り捨てられます。

```
SET ROLE  
GRANT  
ALTER USER  
ALTER ROLE  
CREATE USER  
CREATE ROLE
```

## SQL 文を発行するユーザー

TKPROF は、各 SQL 文を発行したユーザーのユーザー ID もリストします。SQL トレース入力ファイルが複数ユーザーからの統計を収録し、文が複数のユーザーによって発行された場合は、TKPROF は文を解析した最後のユーザーの ID をリストします。すべてのデータベース・ユーザーのユーザー ID が、ALL\_USERS.USER\_ID 例のデータ・ディクショナリに表示されます。

## 実行計画

TKPROF のコマンド行に EXPLAIN パラメータを指定すると、TKPROF は EXPLAIN PLAN 文を使用して、トレースされた SQL 文ごとに実行計画を生成します。TKPROF は実行計画の各ステップによって処理された行数も表示します。

**注意：** インスタンスの始動直後に生成されたトレース・ファイルは、スタートアップ・プロセスのアクティビティを反映するデータを含みます。特に、これらは、システム・グローバル領域（SGA）のキャッシュが満杯になったときの不均衡な I/O アクティビティを反映します。調整を行うときには、このようなトレース・ファイルは無視してください。

**関連項目：** 実行計画の解釈の詳細は、[第 13 章「EXPLAIN PLAN の使用方法」](#)を参照してください。

## どの文を調整するかの判断

次のリストは、ある SQL 文の TKPROF 出力を出力ファイルの表示どおりに示しています。

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

call	count	cpu	elapsed	disk	query current	rows
Parse	11	0.08	0.18	0	0	0
Execute	11	0.23	0.66	0	3	6
Fetch	35	6.70	6.83	100	12326	2
total	57	7.01	7.67	100	12329	8

Misses in library cache during parse: 0

```
10 user SQL statements in session.
0 internal SQL statements in session.
10 SQL statements in session.
```

2 行の挿入または更新、削除、および 824 行の検索に CPU 時間が 7.01 秒かかることを許容できる場合は、このトレース出力をさらに調査する必要はありません。実際に、チューニング作業での TKPROF レポートの主な使用方法は、詳細なチューニング段階のプロセスを排除することです。

このサマリーを見ると、（解析コールが 11 回なのに SQL 文は 10 個しかないために）不要な解析コールが 1 回行われていることと、配列フェッチ操作が実行されていることがわかります（これは、実行されたフェッチ回数よりも多くの行がフェッチされていることからわかります）。

最後に、物理 I/O がほとんど行われていません。これは、おそらく同じデータベース・ブロックが継続的にアクセスされていることを意味します。

プロセスがリソースを使いすぎていることを明確にしたら、次の手順は問題の原因となっている SQL 文を発見することです。通常は、どのプロセスでも調整する必要のある SQL 文の比率はわずかで、それらが最もリソースを使用している SQL 文です。



続く例は、すべて TIMED\_STATISTICS=TRUE で生成されました。ただし、ロックの問題と効率の悪い PL/SQL ループを除いて、問題の文を発見するためには CPU 時間と経過時間のどちらも必要ありません。重要なのは、問合せモード（すなわち、読み込み一貫性の対象）と現行モード（読み込み一貫性の対象ではない）の両方でアクセスするブロックの数です。セグメント・ヘッダーと更新されるブロックは常に現行モードで獲得されますが、すべての問合せ処理と副問合せ処理は問合せモードでデータを要求します。これらの測定単位は、インスタンス統計 CONSISTENT GETS および DB BLOCK GETS とまったく同じです。

SYS として解析された SQL は、ディクショナリ・キャッシュをメンテナンスするために使われる再帰的 SQL コールです。通常はあまり利点がありません。内部 SQL 文の数が多いと思える場合は、何が行われているかを確認するためにチェックする価値はあります（領域管理オーバーヘッドが過多の場合があります）。

## ステップ 5: SQL トレース機能統計の格納

この項では次の内容を取り上げます。

- [TKPROF による出力 SQL スクリプトの生成](#)
- [TKPROF による出力 SQL スクリプトの編集](#)
- [出力表の問合せ](#)

SQL トレース機能によって生成されたアプリケーションに関する統計の履歴を維持し、別の時点でこれらの統計を比較することがあります。TKPROF は、表を作成して、統計の入った行をその表に挿入する SQL スクリプトを生成します。このスクリプトには、次の文が記述されています。

- TKPROF\_TABLE という出力表を作成する CREATE TABLE 文
  - トレースした各 SQL 文ごとに統計行を 1 行ずつ TKPROF\_TABLE に追加する INSERT 文
- TKPROF の実行後にこのスクリプトを実行すると、統計をデータベースに格納できます。

### TKPROF による出力 SQL スクリプトの生成

TKPROF を実行する場合は、INSERT パラメータを使用して、生成される SQL スクリプトの名前を指定します。このパラメータを指定しないと、TKPROF はスクリプトを生成しません。

### TKPROF による出力 SQL スクリプトの編集

TKPROF によって SQL スクリプトが作成された後、実行する前にスクリプトを編集できます。

以前収集した統計の出力表をすでに作成しており、新しい統計をこの表に追加する場合は、スクリプトから CREATE TABLE 文を削除します。これによって、スクリプトは新しい行を既存の表に挿入します。

異なるデータベースの統計を別々の表に格納するために複数の出力表を作成している場合は、CREATE TABLE 文と INSERT 文を編集して、出力表の名前を変更してください。

## 出力表の問合せ

次の CREATE TABLE 文は TKPROF\_TABLE を作成します。

```
CREATE TABLE tkprof_table
  (date_of_insert    DATE,
   cursor_num        NUMBER,
   depth             NUMBER,
   user_id           NUMBER,
   parse_cnt         NUMBER,
   parse_cpu         NUMBER,
   parse_elap        NUMBER,
   parse_disk        NUMBER,
   parse_query       NUMBER,
   parse_current     NUMBER,
   parse_miss        NUMBER,
   exe_count         NUMBER,
   exe_cpu           NUMBER,
   exe_elap          NUMBER,
   exe_disk          NUMBER,
   exe_query         NUMBER,
   exe_current       NUMBER,
   exe_miss          NUMBER,
   exe_rows          NUMBER,
   fetch_count       NUMBER,
   fetch_cpu         NUMBER,
   fetch_elap        NUMBER,
   fetch_disk        NUMBER,
   fetch_query       NUMBER,
   fetch_current     NUMBER,
   fetch_rows        NUMBER,
   clock_ticks       NUMBER,
   sql_statement     LONG);
```

ただし、出力表のほとんどの列は、フォーマットされた出力ファイルに記録されている統計と直接対応しています。たとえば、PARSE\_CNT 列の値は出力ファイルの解析ステップに関するカウント統計に対応しています。

これらの列は、統計が入っている行を識別する際に役立ちます。

SQL\_STATEMENT      この列の値は、SQL トレース機能が収集した統計行の対象となる SQL 文です。この列のデータ型が LONG の場合は、式または WHERE 句条件ではこの列を使用できません。

DATE_OF_INSERT	この列の値は、行が表に挿入された日時です。この値は、SQL トレース機能によって統計が収集された時刻と完全には一致しません。
DEPTH	この列の値は、SQL 文が発行された再帰レベルを示します。たとえば、値 1 はユーザーがその文を発行したことを示します。値 2 は、Oracle が値 1 の文 (ユーザー発行の文) を処理する再帰的コールとして、その文を生成したことを示します。値 <i>n</i> は、Oracle がその文を値 <i>n</i> -1 の文を処理する再帰的コールとして生成したことを示します。
USER_ID	この列の値は、この文を発行するユーザーを識別します。この値はフォーマットした出力ファイルにも出力されます。
CURSOR_NUM	この列の値は、各 SQL 文が割り当てられているカーソルを追跡して記録するために Oracle が使います。文の実行計画は出力表に格納されません。

次の問合せは、出力表からの統計を返します。これらの統計は、14-6 ページの「TKPROF の出力例」で示したフォーマットされた出力に対応します。

```
SELECT * FROM tkprof_table;
```

Oracle は次のように応答します。

DATE_OF_INSERT	CURSOR_NUM	DEPTH	USER_ID	PARSE_CNT	PARSE_CPU	PARSE_ELAP
-----						
21-DEC-1998	1	0	8	1	16	22
-----						
PARSE_DISK	PARSE_QUERY	PARSE_CURRENT	PARSE_MISS	EXE_COUNT	EXE_CPU	
-----						
3	11	0	1	1	0	
-----						
EXE_ELAP	EXE_DISK	EXE_QUERY	EXE_CURRENT	EXE_MISS	EXE_ROWS	FETCH_COUNT
-----						
0	0	0	0	0	0	1
-----						
FETCH_CPU	FETCH_ELAP	FETCH_DISK	FETCH_QUERY	FETCH_CURRENT	FETCH_ROWS	
-----						
2	20	2	2	4	10	
-----						
SQL_STATEMENT						
-----						
SELECT * FROM EMP, DEPT WHERE EMP.DEPTNO = DEPT.DEPTNO						

## TKPROF の解釈における落とし穴の回避

この項では、TKPROF の解釈における細かなポイントをいくつか説明します。

- [負荷の大半を占める文の検出](#)
- [引数トラップ](#)
- [読み込み一貫性トラップ](#)
- [スキーマ・トラップ](#)
- [時間トラップ](#)
- [トリガー・トラップ](#)
- ["正しい" バージョン](#)

**関連項目：** TKPROF およびバインド変数の詳細は、13-25 ページの「[EXPLAIN PLAN の制限事項](#)」を参照してください。

### 負荷の大半を占める文の検出

合計を調べて、負荷の大半を占める文を識別してみてください。

多くの異なるジョブを 1 つの問合せで実行しないようにしてください。特定のオプション・パラメータが指定されているときや、提供されるパラメータがワイルド・カードを含んでいるときに使う必要のある異なる問合せは、分離するほうが効果的です。

特定のパラメータがレポート・ユーザーによって指定されていない場合、問合せは、"%" に設定されたバインド変数を使います。このため、問合せの LIKE 句は無視されます。これらの句が指定されていない問合せを実行するほうが効率的です。

---

---

**注意：** TKPROF は、SQL 文のテキストからバインド変数の TYPE を識別できません。TYPE は CHARACTER であると仮定されます。それ以外の場合は、SQL 文に適切な型変換を組み込む必要があります。

---

---

### 引数トラップ

実行時にバインドされる値を認識していない場合は、"引数トラップ" に陥る可能性があります。LIKE 演算子が使われている場合には特に、バインド変数の特定の値または値のタイプについて、問合せが著しく効率が悪くなることがあります。これは、オブティマイザが、値がわからずに選択内容について想定しなければならないことが原因です。

## 読み込み一貫性トラップ

次の例は、読み込み一貫性トラップを示しています。非コミット・トランザクションが NAME 列に一連の更新を行ったことを知らないと、多くのブロックがアクセスされる理由を判断することは非常に再現です。

通常、このようなケースは再現可能ではありません。そのプロセスが再度実行された場合に、別のトランザクションが同じようにそのプロセスに影響を及ぼすことはあまりありません。

```
select NAME_ID
from CQ_NAMES where NAME = 'FLOOR';
```

call	count	cpu	elapsed	disk	query current	rows
----	-----	----	-----	----	-----	----
Parse	1	0.10	0.18	0	0	0
Execute	1	0.00	0.00	0	0	0
Fetch	1	0.11	0.21	2	101	0

```
Misses in library cache during parse: 1
Parsing user id: 01 (USER1)
```

Rows	Execution Plan
----	-----
0	SELECT STATEMENT
1	TABLE ACCESS (BY ROWID) OF 'CQ_NAMES'
2	INDEX (RANGE SCAN) OF 'CQ_NAMES_NAME' (NON_UNIQUE)

## スキーマ・トラップ

この例は極端な場合を示しているため、スキーマ・トラップは容易に検出できます。最初は、明確で単純に索引付けされた問合せが多くのデータベース・ブロックを検索する必要がある理由、または現行モードでブロックにアクセスすることが必要な理由を理解することは困難です。

```
select NAME_ID
from CQ_NAMES where NAME = 'FLOOR';
```

call	count	cpu	elapsed	disk	query current	rows
-----	-----	-----	-----	-----	-----	-----
Parse	1	0.06	0.10	0	0	0
Execute	1	0.02	0.02	0	0	0
Fetch	1	0.23	0.30	31	51	3

```
Misses in library cache during parse: 0
Parsing user id: 02 (USER2)
```

Rows	Execution Plan
-----	-----
0	SELECT STATEMENT
2340	TABLE ACCESS (BY ROWID) OF 'CQ_NAMES'
0	INDEX (RANGE SCAN) OF 'CQ_NAMES_NAME' (NON-UNIQUE)

2 つの統計は、問合せが全表走査を介して実行された可能性があることを示しています。これらの統計は、現行モードでのブロック・アクセスと、実行計画の Table Access 行ソースに由来する行数です。これは、トレース・ファイルが生成された後、TKPROF が実行される前に必要な索引が構築されたことを示しています。

時間トラップ

次の例で示すように、特定の問合せに長時間かかる理由がわからないことがあります。

```
update CQ_NAMES set ATTRIBUTES = lower(ATTRIBUTES)
where ATTRIBUTES = :att
```

call	count	cpu	elapsed	disk	query current	rows
-----	-----	-----	-----	-----	-----	-----
Parse	1	0.06	0.24	0	0	0
Execute	1	0.62	19.62	22	526	12
Fetch	0	0.00	0.00	0	0	0

```
Misses in library cache during parse: 1
Parsing user id: 02 (USER2)
```

Rows	Execution Plan
-----	-----
0	UPDATE STATEMENT
2519	TABLE ACCESS (FULL) OF 'CQ_NAMES'

ここでも、別のトランザクションによる妨害というのが答えです。この場合は、別のトランザクションが更新を発行する前後の数秒間、表 CQ\_NAMES で共有ロックを保持しています。妨害の影響が発生していることを診断できるようになるにはかなりの経験が必要です。妨害によって発生する遅延が短時間である（または前の例のようにブロック・アクセスにおける増加がわずかである）場合は、比較用のデータが必要です。一方、妨害がわずかなオーバーヘッドの原因にしかならず、本質的に文の効率がよい場合は、統計を分析の対象にする必要はありません。

トリガー・トラップ

ある文に対してレポートされたリソースは、文が処理されていた間に発行されたすべての SQL 用のリソースを含みます。したがって、これらには、トリガーで使われるリソースと、他の再帰的 SQL で使われるリソース（領域割当てで使われるリソースなど）が含まれます。SQL トレース機能が使用可能にされている場合は、TKPROF はこれらのリソースを 2 回レ

ポートします。リソースが実際に低い再帰レベルで消費されている場合は、DML 文を調整しようとすることは避けてください。

ロー・トレース・ファイルを検査して、リソースが消費された場所を正確に知ることができます。再帰的 SQL のエントリは、ユーザーの文の PARSING IN CURSOR エントリの後です。トレース・ファイル内では、順序を定義することはやや困難です。

"正しい" バージョン

これらのトラップの 1 つから生じた出力と比較するために、競合の影響を受けず、索引が適切に配置された、索引付けされた問合せの TKPROF 出力を示します。

```
select NAME_ID
from CQ_NAMES where NAME = 'FLOOR';
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.02	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.00	0.00	0	2	0	1

Misses in library cache during parse: 0  
Parsing user id: 02 (USER2)

Rows	Execution Plan
0	SELECT STATEMENT
1	TABLE ACCESS (BY ROWID) OF 'CQ_NAMES'
2	INDEX (RANGE SCAN) OF 'CQ_NAMES_NAME' (NON-UNIQUE)

この正しいバージョンの注目すべき機能の 1 つは、解析コールには経過時間と CPU 時間ともに 10 ミリ秒を要する一方で、問合せとフェッチの実行にはまったく時間がかかっていないことです。実際は、この問合せはすでに解析済みの形式として共有 SQL 領域で使用可能なので、解析は行われていません。この不合理性は、10 ミリ秒単位で時間を刻んでいることが原因です。これは、単純で効率のよい問合せを確実に記録するには長すぎます。

TKPROF の出力例

この項では、TKPROF 出力の詳細な例を示します。簡潔化のために各部分を編集してあります。

ヘッダー

Copyright (c) Oracle Corporation 1979, 1998. All rights reserved.  
Trace file: v80\_ora\_2758.trc  
Sort options: default  
\*\*\*\*\*  
count = number of times OCI procedure was executed  
cpu = cpu time in seconds executing  
elapsed = elapsed time in seconds executing  
disk = number of physical reads of buffers from disk  
query = number of buffers gotten for consistent read  
current = number of buffers gotten in current mode (usually for update)  
rows = number of rows processed by the fetch or execute call  
\*\*\*\*\*  
The following statement encountered a error during parse:  
select deptno, avg(sal) from emp e group by deptno  
 having exists (select deptno from dept  
 where dept.deptno = e.deptno  
 and dept.budget > avg(e.sal)) order by 1  
Error encountered: ORA-00904  
\*\*\*\*\*

本体

alter session set sql\_trace = true

call	count	cpu	elapsed	disk	query	current	rows
Parse	0	0.00	0.00	0	0	0	0
Execute	1	0.00	0.10	0	0	0	0
Fetch	0	0.00	0.00	0	0	0	0
-----							
total	1	0.00	0.10	0	0	0	0

Misses in library cache during parse: 0  
Misses in library cache during execute: 1  
Optimizer goal: CHOOSE  
Parsing user id: 02 (USER02)  
\*\*\*\*\*  
select emp.ename, dept.dname from emp, dept  
 where emp.deptno = dept.deptno

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.11	0.13	2	0	1	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.00	0.00	2	2	4	14
-----							
total	3	0.11	0.13	4	2	5	14



Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 02 (USER02)

Rows Execution Plan

```

-----
      0  SELECT STATEMENT   GOAL: CHOOSE
     14  MERGE JOIN
      4    SORT (JOIN)
      4      TABLE ACCESS (FULL) OF 'DEPT'
     14    SORT (JOIN)
     14      TABLE ACCESS (FULL) OF 'EMP'

```

\*\*\*\*\*

select a.ename name, b.ename manager from emp a, emp b

where a.mgr = b.empno(+)

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.01	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.01	0.01	1	50	2	14
total	3	0.02	0.02	1	50	2	14

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 01 (USER01)

Rows Execution Plan

```

-----
      0  SELECT STATEMENT   GOAL: CHOOSE
     13  NESTED LOOPS (OUTER)
     14    TABLE ACCESS (FULL) OF 'EMP'
     13      TABLE ACCESS (BY ROWID) OF 'EMP'

     26    INDEX (RANGE SCAN) OF 'EMP_IND' (NON-UNIQUE)

```

\*\*\*\*\*

select ename,job,sal

from emp

where sal =  
 (select max(sal)  
 from emp)

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.00	0.00	0	12	4	1

TKPROF の出力例

total 3 0.00 0.00 0 12 4 1

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 01 (USER01)

Rows Execution Plan

```
-----
      0  SELECT STATEMENT    GOAL: CHOOSE
     14  FILTER
     14    TABLE ACCESS (FULL) OF 'EMP'
     14    SORT (AGGREGATE)
     14    TABLE ACCESS (FULL) OF 'EMP'
```

\*\*\*\*\*

```
select deptno
from emp
where job = 'clerk'
group by deptno
having count(*) >= 2
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.00	0.00	0	1	1	0
-----							
total	3	0.00	0.00	0	1	1	0

Misses in library cache during parse: 13

Optimizer goal: CHOOSE

Parsing user id: 01 (USER01)

Rows Execution Plan

```
-----
      0  SELECT STATEMENT    GOAL: CHOOSE
      0  FILTER
      0    SORT (GROUP BY)
     14  TABLE ACCESS (FULL) OF 'EMP'
```

\*\*\*\*\*

```
select dept.deptno,dname,job,ename
from dept,emp
where dept.deptno = emp.deptno(+)
order by dept.deptno
```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.00	0.00	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.00	0.00	0	3	3	10
-----							
total	3	0.00	0.00	0	3	3	10

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 01 (USER01)

Rows Execution Plan

```

-----
      0  SELECT STATEMENT   GOAL: CHOOSE
     14  MERGE JOIN (OUTER)
          4  SORT (JOIN)
             4  TABLE ACCESS (FULL) OF 'DEPT'
          14  SORT (JOIN)
             14  TABLE ACCESS (FULL) OF 'EMP'

```

\*\*\*\*\*

```

select grade,job,ename,sal
from   emp,salgrade
where  sal between losal and hisal
order by grade,job

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.04	0.06	2	16	1	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.01	0.01	1	10	12	10
total	3	0.05	0.07	3	26	13	10

Misses in library cache during parse: 1

Optimizer goal: CHOOSE

Parsing user id: 02 (USER02)

Rows Execution Plan

```

-----
      0  SELECT STATEMENT   GOAL: CHOOSE
     14  SORT (ORDER BY)
     14  NESTED LOOPS
          5  TABLE ACCESS (FULL) OF 'SALGRADE'
         70  TABLE ACCESS (FULL) OF 'EMP'

```

\*\*\*\*\*

```

select lpad(' ',level*2)||ename org_chart,level,empno,mgr,job,deptno
from   emp
connect by prior empno = mgr
start  with ename = 'clark'
       or ename = 'blake'
order  by deptno

```

call	count	cpu	elapsed	disk	query	current	rows
Parse	1	0.01	0.01	0	0	0	0
Execute	1	0.00	0.00	0	0	0	0
Fetch	1	0.01	0.01	0	1	2	0

TKPROF の出力例

```
-----
total          3      0.02      0.02          0          1          2          0
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 02   (USER02)
Rows      Execution Plan
-----
      0  SELECT STATEMENT      GOAL: CHOOSE
      0  SORT (ORDER BY)
      0  CONNECT BY
     14  TABLE ACCESS (FULL) OF 'EMP'
      0  TABLE ACCESS (BY ROWID) OF 'EMP'
      0  TABLE ACCESS (FULL) OF 'EMP'
*****
create table tkoptkp (a number, b number)

call      count      cpu    elapsed      disk      query      current      rows
-----
Parse      1      0.00      0.00          0          0          0          0
Execute    1      0.01      0.01          1          0          1          0
Fetch      0      0.00      0.00          0          0          0          0
-----
total      2      0.01      0.01          1          0          1          0
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 02   (USER02)
Rows      Execution Plan
-----
      0  CREATE TABLE STATEMENT  GOAL: CHOOSE
*****
insert into tkoptkp
values
(1,1)

call      count      cpu    elapsed      disk      query      current      rows
-----
Parse      1      0.07      0.09          0          0          0          0
Execute    1      0.01      0.20          2          2          3          1
Fetch      0      0.00      0.00          0          0          0          0
-----
total      2      0.08      0.29          2          2          3          1
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 02   (USER02)
Rows      Execution Plan
-----
```

```
0 INSERT STATEMENT GOAL: CHOOSE
.
*****
insert into tkoptkp select * from tkoptkp

call      count      cpu    elapsed      disk      query    current    rows
-----
Parse      1         0.00      0.00         0         0         0         0
Execute    1         0.02      0.02         0         2         3        11
Fetch      0         0.00      0.00         0         0         0         0
-----
total      2         0.02      0.02         0         2         3        11
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 02 (USER02)
Rows      Execution Plan
-----
0 INSERT STATEMENT GOAL: CHOOSE
12 TABLE ACCESS (FULL) OF 'TKOPTKP'
*****
select *
from
  tkoptkp where a > 2

call      count      cpu    elapsed      disk      query    current    rows
-----
Parse      1         0.01      0.01         0         0         0         0
Execute    1         0.00      0.00         0         0         0         0
Fetch      1         0.00      0.00         0         1         2        10
-----
total      3         0.01      0.01         0         1         2        10
Misses in library cache during parse: 1
Optimizer goal: CHOOSE
Parsing user id: 02 (USER02)
Rows      Execution Plan
-----
0 SELECT STATEMENT GOAL: CHOOSE
24 TABLE ACCESS (FULL) OF 'TKOPTKP'
*****
```

サマリー

OVERALL TOTALS FOR ALL NON-RECURSIVE STATEMENTS							
call	count	cpu	elapsed	disk	query	current	rows
Parse	18	0.40	0.53	30	182	3	0
Execute	19	0.05	0.41	3	7	10	16
Fetch	12	0.05	0.06	4	105	66	78
total	49	0.50	1.00	37	294	79	94
Misses in library cache during parse: 18							
Misses in library cache during execute: 1							
OVERALL TOTALS FOR ALL RECURSIVE STATEMENTS							
call	count	cpu	elapsed	disk	query	current	rows
Parse	69	0.49	0.60	9	12	8	0
Execute	103	0.13	0.54	0	0	0	0
Fetch	213	0.12	0.27	40	435	0	162
total	385	0.74	1.41	49	447	8	162
Misses in library parse cache: 13							
19 user SQL statements in session.							
69 internal SQL statements in session.							
88 SQL statements in session.							
17 statements EXPLAINed in this session.							
*****							
Trace file: v80_ora_2758.trc							
Trace file compatibility: 7.03.02							
Sort options: default							
1 session in tracefile.							
19 user SQL statements in trace file.							
69 internal SQL statements in trace file.							
88 SQL statements in trace file.							
41 unique SQL statements in trace file.							
17 SQL statements EXPLAINed using schema:							
SCOTT.prof\$plan_table							
Default table was used.							
Table was created.							
Table was dropped.							
1017 lines in trace file.							

---

## Oracle Trace の使用方法

この章では、Oracle Server のイベント・データを収集するために Oracle Trace を使用方法を説明します。内容は次のとおりです。


- [Oracle Trace について](#)
- [Oracle Trace Manager の使用方法](#)
- [Oracle Trace Data Viewer の使用方法](#)
- [Oracle Trace データの手動収集](#)

### Oracle Trace について

Oracle Trace は汎用のデータ収集製品であり、Oracle Enterprise Manager のシステム管理製品ファミリに含まれます。Oracle Server では、SQL の Parse、Execute、Fetch の統計や Wait 統計などのパフォーマンスやリソース使用状況のデータを収集するために Oracle Trace が使用されます。

**関連項目：** Oracle Diagnostics Pack マニュアル・セットの『Oracle Trace User's Guide』および『Oracle Trace Developer's Guide』を参照してください。これらのマニュアルには、Oracle Server について収集できるイベントとデータの詳しいリストと、独自の製品やアプリケーションでのトレース機能のインプリメント方法が説明されています。

### Oracle Trace データの使用方法

Oracle Trace を使用することの多くの利点の 1 つに、Oracle Trace と他の多数のアプリケーションとの統合があります。 15-1 に示すように、次のアプリケーションで、Oracle サーバーについて収集された Oracle Trace データを使用できます。

- Oracle Expert

Oracle Expert で SQL 作業負荷データのオプション・ソースとして Oracle Trace で収集した情報を使用できます。この SQL データは、索引の追加や削除を薦める場合に使用されます。詳細は、『Oracle Tuning Pack』を参照してください。

- Oracle Trace Data Viewer

Oracle Trace Data Viewer は、SQL および Wait の統計を含む Oracle Trace の収集情報を調べるための簡単なビューアです。Oracle Trace データを次の製品にエクスポートするとさらに詳しく分析できます。

- SQL Analyze

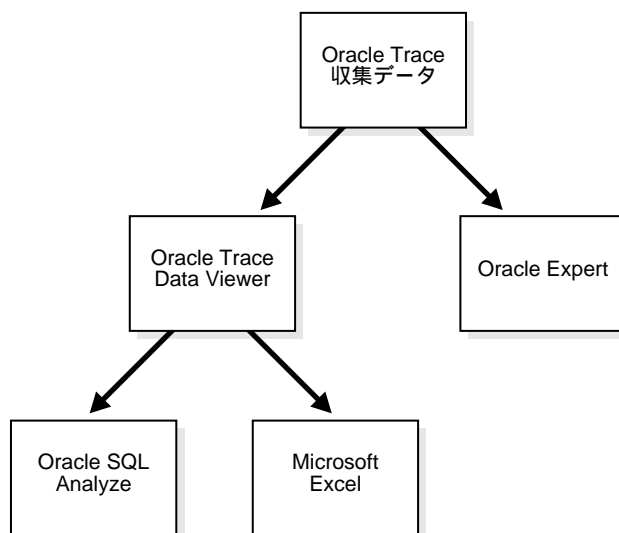
Data Viewer で 1 つ以上の行を選択し、SQL 文テキストをファイルに保存して SQL Analyze にインポートできます。その後、SQL Analyze を使用してそれぞれの文をチューニングできます。

- Microsoft Excel

Data Viewer の SQL は CSV (カンマ区切り値) ファイルに保存して、Microsoft Excel で表示できます。



図 15-1 Oracle Trace と他のアプリケーションの統合



### Oracle Expert への Oracle Trace データのインポート

Oracle Trace を使用して、Oracle Expert アプリケーションで使用する作業負荷データを収集できます。Oracle Trace は、データベースに対して実行する SQL 文のリソース使用状況の統計をリアルタイムで収集します。Oracle Trace を使用すると、パフォーマンスが低下したときに、データベースに対して実行しているすべての SQL 文についてのデータを収集できます。

Oracle Trace のデータ収集期間を制御できます。15 分間低パフォーマンス時の SQL 作業負荷データを収集すると設定した場合でも、低パフォーマンスの期間が終了したらすぐに収集を停止します。

### Data Viewer の SQL の Oracle SQL Analyze へのインポート

Data Viewer の使用中に、「Data View」ウィンドウの上部で 1 つ以上の行を選択してファイルに保存できます。「File/Save」から「SQL (SQL Analyze Format)」を選択すると、問合せテキストを含むファイルが保存されます。その後、この \*.sql ファイルを Oracle SQL Analyze にインポートして、選択した文をチューニングできます。

Oracle SQL Analyze では個々の問合せの実行計画が表示され、さまざまなオプティマイザ・モードやヒントを試すことができます。

## データ・ビューア情報の Microsoft Excel へのインポート

データ・ビューアの使用中に、「Data View」ウィンドウの上部で 1 つ以上の行を選択してファイルに保存できます。CSV ファイル形式を選択すると、Microsoft Excel のスプレッドシートにロードできる \*.csv ファイルが Oracle Trace で作成されます。

## Oracle Trace Manager の使用方法

Oracle Trace には、Oracle Trace Manager というグラフィカルなアプリケーションが提供されていて、Oracle Trace コールを含む製品のために Oracle Trace の収集の作成、スケジューリングおよび管理を行います。

Oracle Server は、SQL と Wait の統計を最小限のオーバーヘッドで収集するために、Oracle Trace API コールを使用してコーディングされています。Oracle Trace Manager グラフィカル・ユーザー・インタフェースを使用すると、次のことができます。

- 収集のスケジューリング
- ユーザーによる収集のフィルタ処理
- Wait イベントの種類による収集のフィルタ処理
- 履歴データを保存するため、収集データをデータベース表用に形式設定
- Oracle Trace Data Viewer による SQL および Wait 統計の表示

## 収集の管理

Oracle Trace の使用と制御には " 収集 " という概念が関係しています。収集とは、対象製品の実行中に発生したイベントについて収集されたデータです。

Oracle Trace Manager を使用すると、収集のスケジューリングと管理を行うことができます。収集を作成するときは、収集名、収集に含める製品やイベント・セット、開始時刻と終了時刻など収集の属性を定義します。Oracle Trace Manager には、収集の作成と実行を促進する収集ウィザードが含まれます。

いったん収集を作成すると、その収集をすぐに実行することも、特定の時刻または指定した間隔でその収集を実行するようにスケジューリングすることもできます。収集が実行されると、その収集に含まれる製品のイベント・データを含むファイルが生成されます。また、類似した他の収集を作成するためのテンプレートとして収集を使用することもできます。

## イベント・データの収集

イベントとは、製品内部でなんらかのアクティビティが発生することです。Oracle Trace は、Oracle Trace API を使用して作成されたソフトウェア製品で発生する事前定義済みのイベントについてのデータを収集します。つまり、製品には Oracle Trace API コールが埋め込まれています。イベントの例としては、解析またはフェッチがあります。

イベントには次の 2 種類があります。

- ポイント・イベント

ポイント・イベントは、対象の製品で瞬間的に何かが発生することを表します。ポイント・イベントの例としては、エラーの発生があります。

- 期間イベント

期間イベントには開始と終了があります。期間イベントの例としては、トランザクションがあります。期間イベントの間に別のイベントが発生する可能性があります。たとえば、エラーがトランザクション中に発生します。

Oracle Server には 13 のイベントがあります。そのうちの 3 つは次のとおりです。

- データベース接続：サーバーのログイン・ユーザー名などのデータを記録するポイント・イベント。
- SQL 解析：一連の SQL 処理期間イベントの 1 つ。このイベントは、ソート、リソース使用率、カーソル番号などの大容量のデータを記録します。
- RowSource: 実行計画の 1 つの行ソースで処理される、SQL 操作、位置、オブジェクト ID、行数など、実行計画に関するデータ。

## 収集したデータへのアクセス

収集の間には、Oracle Trace はイベント・データをメモリーにバッファし、そのデータを定期的に収集バイナリ・ファイルに書き込みます。この方法では、収集プロセスに関連するオーバーヘッドが小さくなります。バイナリ・ファイルに収集されたイベント・データにアクセスするには、データを事前定義済みの表にフォーマットします。これによって、高速かつ柔軟性の高いアクセスが可能になります。このような事前定義済みの表は、"Oracle Trace フォーマット設定表" と呼ばれます。

Oracle Trace Manager は、収集の直後またはしばらくしてから収集データをフォーマットするメカニズムを提供します。

収集をフォーマットする場合、Oracle Trace Manager によってフォーマット済みの収集が作成されるデータベースを次のように指定します。

1. Oracle Trace Manager を使用して、フォーマットする収集を選択します。
2. 「Format」コマンドを選択します。
3. データを入れるターゲット・データベースを指定します。

選択した収集によって、使用される収集定義ファイルとデータ収集ファイルが決まります。フォーマット済みのターゲット・データベースによって、フォーマット済みの収集データを格納する場所が決まります。

データがフォーマットされると、データ・ビューアまたは SQL のレポート作成ツールやスク립トを使用してデータにアクセスできます。

また、Oracle Trace レポート作成ユーティリティから Detail レポートを実行することによってイベント・データにアクセスすることもできます。このレポートでは、収集の結果を表示するための基本的なメカニズムが提供されます。レポート作成するデータとその表示方法の制御は限られています。

**関連項目：** 事前定義済み SQL スクリプトおよび Detail レポートの詳細は、Oracle Diagnostics Pack マニュアル・セットの『Oracle Trace Developer's Guide』を参照してください。

## Oracle Trace Data Viewer の使用方法

Oracle Trace を使用してデータを収集した後で、Oracle Trace 「Collection」メニューから「View Formatted Data...」を選択して Oracle Trace Data Viewer を実行します。または、Oracle Diagnostics Pack ツールバーから直接選択することもできます。データ・ビューアは、収集されたロー・データから SQL と Wait の統計およびリソース使用状況メトリックを計算できます。データ・ビューアで統計を計算すると、リソースを消費する SQL の特定作業が非常に容易になります。

データ・ビューアは、収集期間におけるすべての問合せ実行について Oracle Trace Manager によって収集されたデータから SQL 統計を計算します。SQL 文を 1 回実行する間のリソース使用状況は、データベースまたはノード上の他の同時アクティビティの影響を受けるため、判断を誤らせることがあります。すべての実行についての統計を組み合わせることによって、特定の問合せを実行するときに発生する典型的なリソース使用状況についてより確かな情報が得られます。

---

**注意：** すべてのデータ・ビューで再帰的 SQL を省略できます。

---

## Oracle Trace 事前定義済みデータ・ビュー

SQL および Wait 統計は、Oracle Trace の包括的な一連の事前定義済みデータ・ビューで表されます。Wait 統計では、データ・ビューは Oracle Trace によって収集されるデータに対する問合せの定義です。データ・ビューは、戻される項目すなわち統計で構成されます。オプションでは、戻されるソート順序や行数制限も含まれます。

データ・ビューアで提供されるデータ・ビューを使用して、次のことができます。

- 経過時間やディスク対論理読込みのヒット率など、重要な統計データの調査。
- 文の実行に関する詳細を得るために必要なドリル・ダウン。

事前定義済みデータ・ビューの他に、Oracle Trace Data View Wizard を使用して独自のデータ・ビューを定義できます。

データ・ビューアで SQL および Wait 統計の計算が出ると、使用可能なデータ・ビューを示すダイアログ・ボックスが表示されます。SQL 統計データ・ビューは、[図 15-2](#) に示すように、I/O、Parse（解析）、Elapsed Time（経過時間）、CPU、Row（行）、Sort（ソート）お

よび Wait 統計に分かれます。データ・ビューを選択すると、その説明が画面の右側に表示されます。

図 15-2 Oracle Trace Data Viewer-「Collection」画面



表 15-1 は、Oracle Trace によって提供された上図に示されている事前定義済みデータ・ビューの説明を含みます。

表 15-1 Oracle Trace で提供される事前定義済みデータ・ビュー

ビュー名	ソート基準	表示されるデータ	説明
<b>Logical Reads (論理読み込み)</b>	個別の問合せごとに実行される論理読み込みの合計数。	解析、実行、またはフェッチで読み込まれるブロックの合計数。 問合せの解析、実行およびフェッチでの論理読み込み。	メモリーとディスク両方からのデータ・ブロック読み込みを含む論理データ・ブロック読み込み。  入出力は、データベース・システムでの最もコストがかかる操作の 1 つです。I/O 集中型の文は、メモリーおよびディスクの使用を独占し、他のデータベース・アプリケーションとリソースの競合が発生する可能性があります。
<b>Disk Reads (ディスク読み込み)</b>	ディスク読み込みが最も多く行われる問合せ。	解析、実行、およびフェッチのディスク読み込み。	物理 I/O ととも呼ばれるディスク読み込みは、ディスクから読み込まれるデータベース・ブロックです。ディスク読み込み統計は、読み込み要求がマルチブロック読み込みかシングル・ブロック読み込みのどちらであるかには関係なく、ブロック読み込みのたびに増分されます。ほとんどの物理読み込みでは、ロード・データ、索引、およびロールバック・ブロックがディスクからバッファ・キャッシュに読み込まれます。  物理読み込み件数によって、データ・バッファ・キャッシュでミス・レートが高いことを検出できます。
<b>Logical Reads/Rows Fetched Ratio (論理読み込み / 取出し行の割合)</b>	現在の問合せのすべての実行について論理読み込み回数をフェッチされた行数で割ったもの。	論理 I/O の合計。 フェッチされた行数の合計。	実際に戻される行数に比べてアクセスされるブロック数が多いと、戻される行ごとのコストは高くなります。  問合せの相対的なコストをおおまかに示すことができます。
<b>Disk Reads/Rows Fetched Ratio (ディスク読み込み / 取出し行の割合)</b>	現在の問合せのすべての実行についてディスク読み込み回数をフェッチされた行数で割ったもの。	ディスク I/O の合計。 フェッチされた行数の合計。	戻される各行についてディスクから読み込むブロック数が増えると、戻される行ごとのコストが高くなります。  問合せの相対的なコストをおおまかに示すことができます。
<b>Disk Reads/Execution Ratio (ディスク読み込み / 実行の割合)</b>	個別問合せごとのディスク読み込みの合計数をその問合せの実行回数で割ったもの。	ディスク I/O の合計。 問合せの実行回数とその問合せの論理 I/O。	1 回の実行につき最も多く読み込みを発生させる文を示します。
<b>Disk Reads/Logical Reads Ratio (ディスク読み込み / 論理読み込みの割合)</b>	論理読み込みに対するディスク読み込みの最大ミス・レートの割合。	個々の論理読み込み。 問合せのミス・レートとディスク読み込み。	ミス・レートにより、Oracle Server がメモリー内のデータ・バッファでデータベース・ブロックを見つけた回数に対して、ディスク上のデータベース・ブロックを検索する必要があった回数の割合が示されます。  データ・ブロック・バッファ・キャッシュのミス・レートは、データ検索のために一貫したモードでブロック・バッファにアクセスした回数と 1 回のブロック取得によってアクセスされるブロック数を加えた値で、物理読み込みを割ることによって求められます。  メモリーへのアクセスは、ディスクへのアクセスよりも非常に高速であり、ヒット率も高く、パフォーマンスも上回ります。

表 15-1 Oracle Trace で提供される事前定義済みデータ・ビュー

ビュー名	ソート基準	表示されるデータ	説明
<b>Re-Parse Frequency (再解析頻度)</b>	再解析頻度が最も大きい問合せ。	キャッシュ・ミス数。 解析の合計回数。 解析の合計経過時間。 解析の合計 CPU クロック・カウント。	<p>Oracle Server は、ライブラリ・キャッシュに解析済みの文表現を含む既存の共有 SQL 領域があるかどうかを判別します。ある場合は、ユーザー・プロセスがこの解析済みの表現を使用して、すぐに文を実行します。</p> <p>ライブラリ・キャッシュになかった場合は、文の構文、有効オブジェクトおよびセキュリティについて再チェックしてください。また、オプティマイザが新しい実行計画を判別する必要があります。</p> <p>解析件数統計は、SQL 文がすでに共有 SQL 領域にあるかどうかに関係なく、解析要求ごとに増分されます。</p>
<b>Parse/Execution Ratio (解析 / 実行の割合)</b>	解析回数を 1 文あたりの実行回数で割ったもの。	個々の解析回数。 実行回数。	<p>実行に対する解析の回数はできるだけ 1 に近づく必要があります。1 回の実行あたりの解析回数が多い場合、その文は不必要に再解析されています。これは、SQL 文でのバインド変数が足りないか、カーソルの再使用が有効に行われていないことを示す場合があります。</p> <p>問合せの再解析は、SQL 文の構文、有効オブジェクト、セキュリティを再チェックする必要があることを意味します。また、新しい実行計画が、オプティマイザによって判別される必要があります。</p>
<b>Average Elapsed Time (平均経過時間)</b>	問合せのための解析、実行、フェッチにかかった最大の平均時間。	解析、実行およびフェッチそれぞれの平均。	解析、実行、実行 1 回当たりのフェッチすべての平均経過時間が計算されて、収集内の個別の SQL 文それぞれに対して合計が求められます。
<b>Total Elapsed Time (合計経過時間)</b>	問合せのための解析、実行、フェッチにかかった最大の合計経過時間。	解析、実行およびフェッチそれぞれの経過時間。	すべての解析、実行、フェッチの合計経過時間が計算されて、収集内の個別の SQL 文それぞれに対して合計が求められます。
<b>Parse Elapsed Time (解析経過時間)</b>	個別の SQL 文に関連するすべての解析の合計経過時間。	SQL キャッシュ・ミス。 実行とフェッチの経過時間。 合計経過時間。	<p>Oracle Server は、解析時に、ライブラリ・キャッシュに解析済みの文表現を含む既存の共有 SQL 領域があるかどうかを判別します。ある場合は、ユーザー・プロセスがこの解析済みの表現を使って、すぐに文を実行します。</p> <p>ライブラリ・キャッシュになかった場合は、文の構文、有効オブジェクトおよびセキュリティについてその文を再チェックする必要があります。また、新しい実行計画が、オプティマイザによって判別される必要があります。</p>
<b>Execute Elapsed Time (実行経過時間)</b>	個別の SQL 文に関連するすべての実行の最大の合計経過時間。	合計経過時間。 解析とフェッチそれぞれの経過時間。	Oracle Trace 収集内の問合せのすべてのオカレンスの実行イベントすべての合計経過時間。

表 15-1 Oracle Trace で提供される事前定義済みデータ・ビュー

ビュー名	ソート基準	表示されるデータ	説明
<b>Fetch Elapsed Time</b> (フェッチ経過時間)	個別の SQL 文に関連するすべてのフェッチの最大の合計経過時間。	フェッチされた行数。 フェッチ回数。 実行回数。 合計経過時間。 解析と実行それぞれの経過時間。	Oracle Trace 収集内の現在の問合せのすべてのオカレンスについてデータをフェッチするために費やされた合計経過時間。
<b>CPU Statistics</b> (CPU 統計)	SQL 文のために解析、実行およびフェッチに費やされた合計 CPU クロック・カウント。	解析、実行およびフェッチの CPU クロック・カウント。 SQL キャッシュ・ミスとメモリー内ソートの数。	SQL 文および他の種類のコールが Oracle Server に対して行われるとき、そのコールを処理するために一定量の CPU 時間が必要です。平均的なコールが必要とする CPU 時間は少量です。ただし、大容量のデータ、リソース集中型の問合せ、メモリー内ソートまたは過剰な再解析を含む SQL 文は、大容量の CPU 時間を消費する可能性があります。  表示される CPU 時間は、データベースが存在するオペレーティング・システムの CPU クロック・カウント数です。
<b>Number of Rows Returned</b> (戻される行数)	SQL 文の実行およびフェッチ時に戻される最大の合計行数。	実行行数とフェッチ操作時に戻される行数。	フェッチおよび実行時に最大の行数を操作する問合せを対象にします。行集中型の問合せをチューニングすることによって高い効果が得られることを意味します。
<b>Rows Fetched/Fetch Count Ratio</b> (取出し行 / フェッチ回数の割合)	フェッチされる行数をフェッチ回数で割ったもの。	フェッチされた個別の行数。 フェッチ回数。	この値は一度にフェッチされる行数を示します。配列フェッチ機能が利用されたレベルを示すこともあります。値が 1 に近い場合は、配列フェッチを使用することによってコードを最適化する機会があることを意味します。
<b>Sorts on Disk</b> (ディスク上ソート)	ディスク上ソートを最も多く行った問合せ。	SQL 文のソート統計。 メモリー内ソート回数。 ソートされた行数の合計。	ディスク上ソートは、メモリー内で実行できなかったソートです。メモリーへのアクセスはディスクへのアクセスよりも高速なため、ディスク上ソートはコストが高くなります。
<b>Sorts in Memory</b> (メモリー内ソート)	メモリー内ソートを最も多く行った問合せ。	SQL 文のソート統計。 ディスク・ソート回数。 ソートされた行数の合計。	メモリー内ソートは、一時表領域セグメントを使用することなく、完全にメモリーのソート・バッファ内で実行できるソートです。
<b>Rows Sorted</b> (ソートされる行)	ソートした行数が最も多い問合せ。	メモリー内ソート回数。 ディスク上ソート回数。	ソートした行数が最も多い問合せから順に並べた SQL 文のソート統計を戻します。



表 15-1 Oracle Trace で提供される事前定義済みデータ・ビュー

ビュー名	ソート基準	表示されるデータ	説明
Waits by Total Wait Time (待機 - 合計待機時間)	待機の個別タイプごとの最大の合計待機時間。	平均待機時間、合計待機時間、および待機タイプごとの待機数。	待機は、待機の説明、または収集内ですべてのオカレンスの累積待機時間が最も長い待機タイプの順にソートされます。
Waits by Average Wait Time (待機 - 平均待機時間)	待機タイプごとの最大の平均待機時間。	平均待機時間、合計待機時間、および待機タイプごとの待機数。	待機は、待機の説明、または収集内ですべてのオカレンスの平均待機時間が最も長い待機タイプの順にソートされます。
Waits by Event Frequency (待機 - イベント頻度)	待機タイプごとの待機の頻度。	待機タイプごとの待機数、平均待機時間、および合計待機時間。	待機は、収集内に最も頻繁に出現する待機イベントまたは待機説明によってソートされます。

## Oracle Trace データの表示

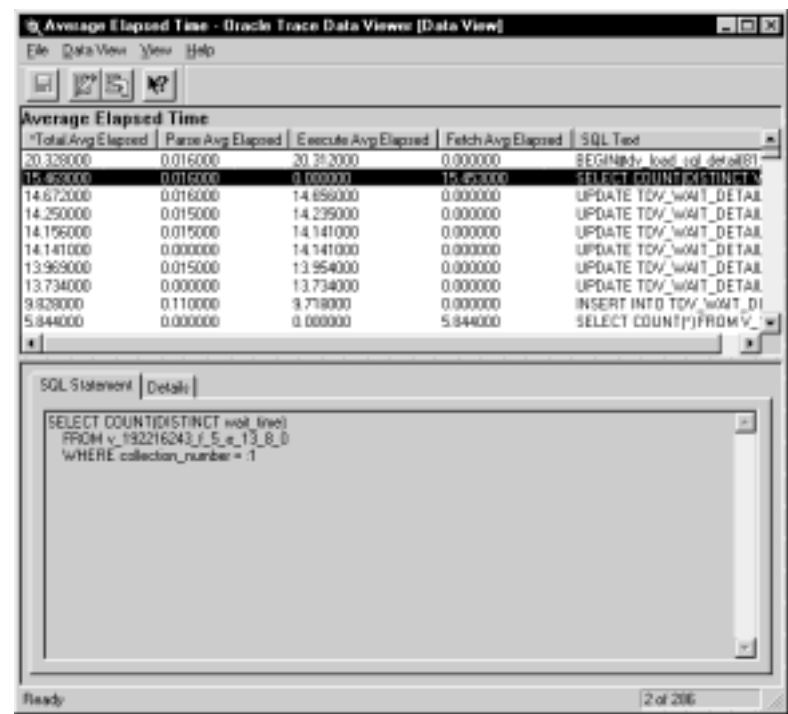
データ・ビューによって提供される SQL または Wait イベントのデータ・ビューをダブル・クリックすると、Oracle Trace が収集データを問い合わせ、データ・ビューの説明に示されている基準でソートされたデータを表示します。

たとえば、「Disk Reads/Log Reads Ratio ( ディスク読み込み / ログ読み込みの割合 )」ビューをダブル・クリックすると、データ・バッファ・キャッシュのミス・レートが最も高い問合せから順にソートされたデータが戻されます。また、ディスク読み込みと論理読み込みそれぞれの値も表示されます。

「Average Elapsed Time ( 平均経過時間 )」データ・ビューをダブル・クリックすると、解析、実行、フェッチにかかった平均経過時間が最も長い問合せから順にソートされたデータが戻されます。また、解析、実行、フェッチの平均経過時間も表示されます。

図 15-3 に、「Average Elapsed Time ( 平均経過時間 )」データ・ビューのデータを示します。問合せテキストと統計はウィンドウの上部に表示されます。列ヘッダーをクリックすると、データ・ビューはその列の統計によって行をソートします。

図 15-3 Oracle Trace Data Viewer-「Data View」画面



現在選択されているデータ・ビューのSQLテキストは、ウィンドウ下部の「SQL Statement」プロパティ・シートに表示されています。現在選択されているデータ・ビューについての全項目に関する統計の詳細も、「Details」プロパティ・シートに表示されます。

図 15-3 に示されるようなデータ・ビューを調べる場合は、次のものを印刷できます。

- 画面の上部にあるデータ・ビューの統計。
- 現在のSQL文（フォーマットされた出力形式）および「Details」プロパティ・ページの現在選択されている問合せについて収集されたすべての統計データの詳細。

印刷時にウィンドウのどの部分がアクティブになっているかによって、印刷される画面部分が決まります。たとえば、画面の上部がアクティブになっている場合は、このデータ・ビューのすべての統計とSQLが表形式で印刷されます。

## 「SQL Statement」プロパティ・ページ

「SQL Statement」プロパティ・ページには、現在選択されている問合せがフォーマットされた出力形式で表示されます。

## 「Details」プロパティ・ページ

「Details」プロパティ・ページには、Oracle Trace 収集内の指定された問合せのすべての実行の統計についての詳細なレポートが表示されます。現在選択されている SQL 文のテキストは、プロパティ・ページの下に示されます。

## 「Details」プロパティ・ページの例

SQL 文のすべての解析、実行およびフェッチについての統計。

解析時のライブラリ・キャッシュでのミス数: 1.000000

SQL 文の経過時間統計:

Average Elapsed Time:	0.843000
Total Elapsed Time:	0.843000
Total Elapsed Parse:	0.000000
Total Elapsed Execute:	0.843000
Total Elapsed Fetch:	0.000000
Average Elapsed Parse:	0.000000
Average Elapsed Execute:	0.843000
Average Elapsed Fetch:	0.000000

解析、実行およびフェッチのコール回数:

Number of Parses:	1
Number of Executions:	1
Number of Fetches:	0

解析、実行およびフェッチ・コールの論理 I/O 統計:

Logical I/O for Parses:	1
Logical I/O for Executions:	247
Logical I/O for Fetches:	0
Logical I/O Total:	0

解析、実行およびフェッチ・コールのディスク I/O 統計:

Disk I/O for Parses:	0
Disk I/O for Executions:	28
Disk I/O for Fetches:	0

Disk I/O Total: 0

解析、実行およびフェッチ・コールの CPU 統計:

CPU for Parses:	0
CPU for Executions:	62500
CPU for Fetches:	0
CPU Total:	62500

実行およびフェッチ・コールの行統計:

Rows processed during Executions:	104
Rows processed during Fetches:	0
Rows Total:	104

実行およびフェッチ・コールのソート統計:

Sorts on disk:	0
Sorts in memory:	2
Sort rows:	667

ヒット率 - 1 から、ディスク I/O を論理 I/O で割った値を引いた値: 0.112903

実行した論理 I/O を実際に処理された行数で割った値: 2.384615

実行したディスク I/O を実行数で割った値: 28.000000

解析回数を実行数で割った値: 1.000000

フェッチされた行数をフェッチ回数で割った値: 0.000000

```
INSERT INTO tdv_sql_detail
(collection_number, sql_text_hash,
"LIB_CACHE_ADDR")
SELECT DISTINCT collection_number,
                 sql_text_hash,
                 "LIB_CACHE_ADDR"
FROM v_192216243_f_5_e_7_8_0
WHERE collection_number = :b1;
```

## 選択された問合せの追加情報の取得

現在選択されている SQL 文の追加情報を容易に取得できる 2 つの方法があります。

- データ・ビューを修正して統計や項目の追加または削除を行うには、「Data View」メニューから「Modify」を選択します。「Items」プロパティ・シートで統計の追加や削除を行うことができます。これらの統計は、データ・ビューで新規列として表示されます。[図 15-3](#) の選択された問合せは次のとおりです。

```
SELECT COUNT(DISTINCT WAIT_TIME)
FROM v_192216243_f_5_e_13_8_0
WHERE collection_number = :1;
```

この問合せは、v\_192216243\_f\_5\_e\_13\_8\_0 表の WAIT\_TIME 列の個別の値の数を求めます。既存のデータ・ビューを修正することによって、ソートされた行数である「Sort Rows」や実行時の CPU クロック・カウント数である「Execute CPU」など、他の必要な統計を追加できます。

この問合せは、WAITS 表の WAIT\_TIME 列の個別の値の数を求めます。既存のデータ・ビューを修正することによって、実行時に処理された行数である「Execute Rows」や実行時の CPU クロック・カウント数である「Execute CPU」など、他の必要な統計を追加できます。

既存の列の削除、ソート順序の変更、または表示するデフォルトの行数の変更もできます。修正したビューを新規のユーザー定義データ・ビューとして保存できます。Oracle は、ユーザー定義データ・ビューを SQL および Wait データ・ビューのデータ・ビュー提供リストの後にある「Custom」データ・ビュー・コンテナに格納します。

- ツールバーの「Drill」アイコンをクリックすることによって、選択されている問合せのすべての解析、実行およびフェッチの統計をドリル・ダウンします。「Drill down Data View」ダイアログを[図 15-4](#) に示します。

図 15-4 Oracle Trace Data Viewer- 「Drill Down Data View」 画面



ドリルダウン・データ・ビューには、すべての解析、実行およびフェッチの個々の統計が表示されます。

図 15-4 では、「Basic Statistics for Parse/Execute/Fetch ( 解析 / 実行 / フェッチの基本統計 )」ドリルダウン・データ・ビューが選択されています。TKPROF のものに類似した統計が表示されます。

---

**注意：** TKPROF の詳細は、[第 12 章「診断ツールの概要」](#)を参照してください。

---

表 15-2 ドリルダウン・データ・ビュー

ドリルダウン名	ソート基準	表示されるデータ	説明
Basic Statistics for Parse/Execute/Fetch ( 解析 / 実行 / フェッチの基本統計 )	最大経過時間。	各個別コールについて : CPU。 経過時間。 ディスク I/O。 論理 I/O。 処理された行数。	TKPROF の統計に類似した解析、実行およびフェッチの統計。
CPU Statistics for Parse/Execute/Fetch ( 解析 / 実行 / フェッチの CPU 統計 )	CPU の最大数。	CPU 合計。 ページフォルト。	現在の問合せの解析、実行およびフェッチの CPU およびページフォルト統計。  CPU 合計は、ユーザー・モードとシステム・モード両方でのクロック・カウント数です。クロック・カウントの細かさは、データベースが存在するオペレーティング・システムに固有です。
I/O Statistics for Parse/Execute/Fetch ( 解析 / 実行 / フェッチの I/O 統計 )	ディスク I/O の最大数。	論理 I/O とディスク I/O の統計。 ページフォルト I/O ( ハード・ページフォルト数 ) 入力 I/O ( ファイル・システムが入力を実行した回数 ) 出力 I/O ( ファイル・システムが出力を実行した回数 )	解析、実行、およびフェッチの I/O 統計。
Parse Statistics ( 解析統計 )	最大経過時間。	現在のユーザー ID。 スキーマ ID。	解析統計、たとえば、現在の文がライブラリ・キャッシュで見つからなかったかどうか、Oracle オプティマイザ・モード、現在のユーザー ID、およびスキーマ ID。

表 15-2 ドリルダウン・データ・ビュー

ドリルダウン名	ソート基準	表示されるデータ	説明
Row Statistics for Execute/Fetch ( 実行 / フェッチの行統計 )	戻された最大の行数。	戻された行数。 ソートされた行数。 全表走査時に戻された行数。	実行とフェッチの行統計。
Sort Statistics for Parse/Execute/Fetch ( 解析 / 実行 / フェッチのソート統計 )	最大経過時間。	ディスク上ソート。 メモリー内ソート。 ソートされた行数。 全表走査で戻された行数。	解析、実行およびフェッチのソート統計。
Wait Parameters ( 待機パラメータ )	Wait_time。	説明。 Wait_time。 P1。 P2。 P3。	待機について調査すると、競合の原因を検出するのに役立つことがあります。  P1、P2、および P3 というパラメータは、特定の待機イベントについて詳細を提供する値です。パラメータは、待機イベントに依存するビューの外部キーです。たとえば、ラッチ待機については、P2 は VSLATCH に対する外部キーであるラッチ番号です。  各パラメータの意味は、各待機タイプに固有です。

## Oracle Trace データの手動収集

Oracle Trace Manager は Oracle Trace に対する主なインタフェースですが、オプションで Oracle Trace データの手動収集を強制的に実行できます。これを行うには、コマンド行インタフェースを使って初期化パラメータを編集するか、ストアド・プロシージャを実行します。

### Oracle Trace コマンド行インタフェースの使用

Oracle Trace のサーバー収集を制御するもう 1 つのオプションは、Oracle Trace CLI ( コマンド行インタフェース ) です。CLI によって、収集の開始時にデータベースにアタッチされているすべてのサーバー・セッションのイベント・データが収集されます。収集が開始した後でアタッチしたセッションは収集から除外されます。CLI は次の機能に対する OTRCCOL コマンドによって起動されます。

- OTRCCOL START job\_id input\_parameter\_file
- OTRCCOL STOP job\_id input\_parameter\_file
- OTRCCOL FORMAT input\_parameter\_file
- OTRCCOL DCF col\_name cdf\_file
- OTRCCOL DFD col\_name username password service



パラメータ JOB\_ID は任意の数値を指定できますが、一意でなければなりません。また、収集を停止するためにはこの値を覚えておく必要があります。入力パラメータ・ファイルには、次の例に示す各機能に必要な特定のパラメータ値が含まれます。COL\_NAME (収集の名前) および CDF\_FILE (収集定義ファイル) は、START 機能の入力パラメータ・ファイルで初期値が定義されています。

OTRCCOL START コマンドによって、入力パラメータ・ファイルに含まれるパラメータ値に基づく収集が起動されます。次に例を示します。

```
OTRCCOL START 1234 my_start_input_file
```

ここで、ファイル MY\_START\_INPUT\_FILE には次の入力パラメータが含まれます。

col_name	my_collection
dat_file	< 通常、収集の名前と同じ >.dat
cdf_file	< 通常、収集の名前と同じ >.cdf
fdf_file	< サーバー・イベント・セット >.fdf
regid	1 192216243 0 0 5 < データベース SID>

fdf\_file の値として使用できるサーバー・イベント・セットは、ORACLE、ORACLEC、ORACLEDE、ORACLEE、および ORACLESM です。

**関連項目：** サーバー・イベント・セットの詳細は、15-20 ページの「[Oracle Trace の制御のための初期化パラメータの使用](#)」を参照してください。

次のように、OTRCCOL STOP コマンドを使用すると収集の実行を一時停止します。

```
OTRCCOL STOP 1234 my_stop_input_file
```

このとき、my\_stop\_input\_file には収集の名前と cdf\_file の名前が含まれます。

OTRCCOL FORMAT コマンドを使うと、バイナリの収集ファイルを Oracle 表にフォーマットできます。FORMAT コマンドの例は次のとおりです。

```
otrccol format my_format_input_file
```

ここで、my\_format\_input\_file には次の入力パラメータが含まれます。

username	< データベース・ユーザー名 >
password	< データベース・パスワード >
service	< データベース・サービス名 >
cdf_file	< 通常、収集の名前と同じ >.cdf
full_format	<0/1>

full\_format 値に 1 を指定すると完全フォーマットが行われます。値 0 を指定すると部分フォーマットが行われます。

**関連項目：** Oracle Trace 収集の部分的または全体的なフォーマットの詳細、および format コマンドを実行するに先立って Oracle Trace フォーマット表を作成するためのその他の重要な情報は、15-24 ページの「[Oracle Trace データの Oracle 表へのフォーマット設定](#)」を参照してください。

OTRCCOL DCF コマンドを使うと、特定の収集の収集ファイルを削除できます。OTRCCOL DFD コマンドを使うと、特定の収集について Oracle Trace フォーマット設定表からフォーマット済みデータを削除できます。

## Oracle Trace の制御のための初期化パラメータの使用

Oracle Trace を制御するためにデフォルトで 6 つのパラメータが設定されています。データベースの管理者アカウントにログインし、SHOW PARAMETERS TRACE コマンドを実行すると、[表 15-3](#) に示されている次のパラメータが表示されます。

表 15-3 Oracle Trace 初期化パラメータ

名前	型	値
ORACLE_TRACE_COLLECTION_NAME	string	[NULL]
ORACLE_TRACE_COLLECTION_PATH	string	\$ORACLE_HOME/otrace/admin/cdf
ORACLE_TRACE_COLLECTION_SIZE	integer	5242880
ORACLE_TRACE_ENABLE	boolean	FALSE
ORACLE_TRACE_FACILITY_NAME	string	oracled
ORACLE_TRACE_FACILITY_PATH	string	\$ORACLE_HOME/otrace/admin/cdf

Oracle Trace 初期化パラメータは変更できます。また、初期化ファイルに追加して使用することもできます。

**注意：** この章では、UNIX ベースのシステムにおけるファイルのパス名を参照しています。その他のオペレーティング・システムでの正確なパスは、プラットフォーム固有の Oracle マニュアルを参照してください。

**関連項目：** これらのパラメータの詳細な説明は、『Oracle8i リファレンス・マニュアル』に記載されています。

## Oracle Trace の収集を使用可能にする

ORACLE\_TRACE\_ENABLE パラメータは、デフォルトでは FALSE に設定されています。FALSE 値に設定されていると、その Oracle Server 用の Oracle Trace を使うことはできません。

サーバーの Oracle Trace 収集を使用可能にするには、パラメータを TRUE に設定してください。パラメータを TRUE に設定しても Oracle Trace 収集は開始されませんが、そのかわりに Oracle Trace がそのサーバーに対して使用可能になります。さらに、次の方法のいずれかを行うと、Oracle Trace を起動できます。

- Oracle Diagnostics Pack とともに提供される Oracle Trace Manager アプリケーションの使用。
- ORACLE\_TRACE\_COLLECTION\_NAME パラメータの設定。

ORACLE\_TRACE\_ENABLE を TRUE に設定したら、Oracle Diagnostics Pack とともに提供される Oracle Trace Manager を使用するか、ORACLE\_TRACE\_COLLECTION\_NAME パラメータに収集の名前を入力することによって、Oracle Trace サーバー収集を開始および終了できます。このパラメータのデフォルト値は NULL です。収集の名前は、長さ 16 文字以内で指定できます。パラメータを有効にするには、データベースをシャットダウンしてから、再び開始する必要があります。収集の名前を指定し、サーバーを開始すると、すべてのデータベース・セッションについての Oracle Trace 収集が自動的に開始されます。この機能は SQL Trace のものと似ています。

ORACLE\_TRACE\_COLLECTION\_NAME パラメータを使用して開始した収集を停止するには、サーバー・インスタンスをシャットダウンして、ORACLE\_TRACE\_COLLECTION\_NAME を NULL にリセットします。このパラメータ値の収集の名前は、収集定義ファイル ( *collection\_name.cdf* ) およびバイナリ・データ・ファイル ( *collection\_name.dat* ) の、2 つの収集出力ファイルの名前としても使用されます。

## Oracle Trace で収集するイベント・セットを決定する

ORACLE\_TRACE\_FACILITY\_NAME 初期化パラメータでは、Oracle Trace が収集するイベント・セットを指定します。DEFAULT イベント・セットの名前は ORACLED です。ALL イベント・セットは ORACLE、EXPERT イベント・セットは ORACLEE、SUMMARY イベント・セットは ORACLESUM、CACHEIO イベント・セットは ORACLEC です。

データベースが再起動したときにデータ収集が開始されない場合は、以下の項目を調べてください。

- ORACLE\_TRACE\_FACILITY\_NAME で識別されるイベント・セット・ファイル ( 拡張子は .fdf ) が、ORACLE\_TRACE\_FACILITY\_PATH 初期化パラメータで識別されるディレクトリにあるかどうか。このパラメータで指定する正確なディレクトリはプラットフォーム固有です。
- ファイル REGID.DAT および PROCESS.DAT、COLLECT.DAT が Oracle Trace の管理ディレクトリにあるかどうか。それらのファイルがない場合は、OTRCCREF 実行可能ファイルを実行して作成します。

- 初期化ファイルで変更した値に Oracle Trace パラメータが設定されているかどうか。Instance Manager を使用して、Oracle Trace パラメータの設定を識別します。
- EPC\_ERROR.LOG ファイルを探して、収集が失敗した原因について詳しい情報を調べます。Oracle Trace は、Oracle Trace Collection Services OTRCCOL イメージを実行するときに、Oracle Intelligent Agent の現在のデフォルト・ディレクトリに EPC\_ERROR.LOG ファイルを作成します。Oracle Trace を Oracle Trace Manager から実行しているか、コマンド行インタフェースから実行しているかによって異なりますが、次のいずれかの位置に EPC\_ERROR.LOG ファイルがあるはずです。
  - \$ORACLE\_HOME または \$ORACLE\_HOME/network/agent ( UNIX )
  - %ORACLE\_HOME%\network\agent または %ORACLE\_HOME%\net80\agent ( NT )
  - NT の %ORACLE\_HOME%\rdbms ( UNIX の \$ORACLE\_HOME/rdbms )
  - 現在の作業ディレクトリ ( コマンド行インタフェースを使用している場合 )
  - UNIX で EPC\_ERROR.LOG ファイルを探すには、\$ORACLE\_HOME ディレクトリに移動して、次のコマンドを実行してください。

```
find . -name EPC_ERROR.LOG -print .
```

---

**注意：** UNIX では、EPC\_ERROR.LOG ファイル名は大文字小文字の区別があり、大文字になっています。

---

- USER\_DUMP\_DEST 初期化パラメータで指定されたディレクトリで \*.trc ファイルを探します。\*.trc ファイルについて「epc」を検索すると、エラーが発生することがあります。これらのエラーとその説明は、\$ORACLE\_HOME/otrace/include/epc.h ファイルに含まれます。

## Oracle Trace の制御のためのストアド・プロシージャの使用

Oracle Trace ストアド・プロシージャを使用すると、自分のセッションまたは他のセッションについての Oracle Trace の収集を起動できます。自分のデータベース・セッションについて Oracle Trace データを収集するには、次のストアド・プロシージャ・パッケージ構文を実行します。

```
DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE(TRUE/FALSE,  
collection_name, serverevent_set)
```

このとき、次のようになります。

True/false	Boolean: オンの場合は TRUE、オフの場合は FALSE。
Collection_name	VARCHAR2: 収集の名前 ( ファイル拡張子はなし。最大 8 文字 )

Server\_event\_set      VARCHAR2: サーバー・イベント・セット (oracled、oraclesm、  
oralec、oracle または oraclee)。

例：

```
EXECUTE DBMS_ORACLE_TRACE_USER.SET_ORACLE_TRACE (TRUE, 'MYCOLL', 'oracle');
```

自分以外のデータベース・セッションについて Oracle Trace データを収集するには、次のストアド・プロシージャ・パッケージ構文を実行します。

```
DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION  
(sid, serial#, true/false, collection_name, server_event_set)
```

このとき、次のようになります。

sid                      Number: V\$SESSION.SID のセッション・インスタンス。

serial#                 Number: V\$SESSION.SERIAL# のセッション・シリアル番号。

例：

```
EXECUTE DBMS_ORACLE_TRACE_AGENT.SET_ORACLE_TRACE_IN_SESSION  
(8,12,TRUE, 'NEWCOLL', 'oracled');
```

収集が発生しない場合は、次の項目を調べてください。

- SERVER\_EVENT\_SET で識別されるサーバー・イベント・セット・ファイルが存在するかどうか。このフィールドに完全なファイル指定がない場合は、そのファイルは初期化ファイルの ORACLE\_TRACE\_FACILITY\_PATH で識別されるディレクトリにあります。
- ファイル REGID.DAT、PROCESS.DAT および COLLECT.DAT が Oracle Trace admin ディレクトリにあるかどうか。それらのファイルがない場合は、OTRCCREF 実行可能ファイルを実行して作成します。
- ストアド・プロシージャ・パッケージがデータベースにあるかどうか。パッケージがない場合は、OTRCSVR.SQL ファイル (Oracle Trace admin ディレクトリの) を実行してパッケージを作成してください。
- ユーザーがストアド・プロシージャの EXECUTE 権限を持っているかどうか。

## Oracle Trace の収集結果

Oracle Trace 収集によって、次の収集ファイルが作成されます。

- collection\_name.CDF は、収集のための Oracle Trace 収集定義ファイルです。
- collection\_name.DAT ファイルは、バイナリ・フォーマットのトレース・データを含む Oracle Trace 出力ファイルです。

収集ファイル内の Oracle Trace データは次の方法でアクセスできます。

- バイナリ・ファイルから Oracle Trace レポートを作成できます。
- データ・ビューア、SQL アクセスおよびレポート作成のために、データを Oracle 表にフォーマットできます。

## Oracle Trace データの Oracle 表へのフォーマット設定

Oracle Trace のサーバー収集を Oracle 表にフォーマットして、SQL レポート作成ツールによってさらに柔軟にアクセスできるようにします。Oracle Trace によって収集対象のイベントごとに別の表が作成されます。たとえば、解析イベントの表は、サーバー収集中に発生したすべての解析イベントについてのデータを格納するために作成されます。データをフォーマットする前に、サーバー・ホスト・マシンで OTRCFMTC.SQL スクリプトを実行することによって、Oracle Trace フォーマット設定表を設定する必要があります。

---

---

**注意：** Oracle Server リリース 7.3.4 および 8.0.4 以降では、フォーマット設定表は自動的に作成されます。

---

---

次の構文を使用して Oracle Trace 収集をフォーマットします。

```
OTRCFMT [optional parameters] collection_name.cdf [user/password@database]
```

user/password@database を省略すると、この情報を求めるプロンプトが表示されます。

Oracle Trace を使うと、収集を行っている間にもデータをフォーマットできます。デフォルトでは、以前にフォーマットされたことがない収集の一部だけが Oracle Trace によってフォーマットされます。収集ファイル全体を再フォーマットする場合は、オプション・パラメータ `-f` を使ってください。

Oracle Trace では、いくつかの SQL スクリプトが提供されており、それらを使ってサーバー・イベント表にアクセスできます。サーバー・イベント表の詳細、およびイベント・データへアクセスしたりイベント表のパフォーマンスを改善したりするためのスクリプトの詳細は、『Oracle Trace User's Guide』を参照してください。

## Oracle Trace 統計レポート作成ユーティリティ

Oracle Trace 統計レポート作成ユーティリティを使用すると、サーバー・イベントの各オカレンスに関連するすべての項目の統計が表示されます。これらのレポートは非常に大きくなる可能性があります。コマンド・パラメータを使うことによって、レポート出力を制御できます。次のコマンドとオプションのパラメータを使用してレポートを作成します。

```
OTRCREP [optional parameters] collection_name.CDF
```

最初は、PROCESS.txt というレポートを実行するとよいでしょう。このレポートを作成すると、別のレポートを実行する対象となる特定のプロセス識別子の一覧が生成されます。

Oracle Trace レポート作成ユーティリティの出力を操作するには、次のようなオプションのレポート・パラメータを使用します。

output_path	レポート・ファイルの出力のフル・パスを指定します。指定しない場合、ファイルは現行ディレクトリに作成されます。
-p	イベント・データをプロセスごとに編成します。プロセス ID (pid) を指定すると、そのプロセスによって生成されたすべてのイベントを時系列順に含む 1 つのファイルが作成されます。プロセス ID を省略すると、収集に含まれる各プロセスごとに 1 つのファイルが作成されます。出力ファイルの名前は、 <i>collection_Ppid.txt</i> になります。
-P	収集に含まれるすべてのプロセスをリストする <i>collection_PROCESS.txt</i> というレポートが作成されます。これには、イベント・データは含まれません。最初にこのレポートを作成して、さらに詳しいレポートを作成する特定のプロセスを判別してもかまいません。
-w#	レポート幅を設定します。たとえば、-w132 など。デフォルトは 80 文字です。
-l#	レポートの 1 ページの行数を設定します。デフォルトは 1 ページあたり 63 行です。
-h	すべてのイベントと項目のレポート・ヘッダーを抑制し、簡潔なレポートを作成します。
-s	Net8 データにのみ使用します。このオプションでは、SQLNet Tracing ファイルに類似したファイルが作成されます。
-a	すべての製品についてのすべてのイベントを含むレポートを、データ収集 (.dat) ファイルで発生した順序に従って作成します。





# 第 IV 部

---

## インスタンスのパフォーマンスの最適化

第 IV 部では、Oracle インスタンスのパフォーマンスを最適化するために、データベース・システムのさまざまな要素をチューニングする方法について説明します。次の章が含まれます。

- [第 16 章「動的パフォーマンス・ビュー」](#)
- [第 17 章「システムのパフォーマンス問題の診断」](#)
- [第 18 章「CPU リソースのチューニング」](#)
- [第 19 章「メモリー割当てのチューニング」](#)
- [第 20 章「I/O のチューニング」](#)
- [第 21 章「リソースの競合のチューニング」](#)
- [第 22 章「ネットワークのチューニング」](#)
- [第 23 章「マルチスレッド・サーバー・アーキテクチャのチューニング」](#)
- [第 24 章「オペレーティング・システムのチューニング」](#)
- [第 25 章「インスタンス回復パフォーマンスのチューニング」](#)



---

## 動的パフォーマンス・ビュー

この章では、次の目的での V\$ ビューの使用方法について説明します。

- チューニングのためのインスタンスレベルのビュー
- チューニングのためのセッションレベルのビューまたは一時的なビュー
- 現行の統計値および変更率

**関連項目：** すべての動的パフォーマンス表の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

動的パフォーマンス・ビューすなわち "V\$" ビューは、インスタンスのレベルのパフォーマンス問題を識別するときに役立ちます。すべての V\$ ビューは V\$FIXED\_TABLE ビューにリストされています。

V\$ ビューの内容は基礎となる X\$ 表によって提供されます。X\$ 表は、SQL 文で修正できる内部データ構造です。このため、これらの表が使用可能になるのは、インスタンスが NOMOUNT または MOUNT 状態にある場合のみです。

この章では、パフォーマンスのチューニングに最も役立つ V\$ ビューについて説明します。また、V\$ ビューは、応答時間の急激な低下がユーザーから報告された場合などの非定型的調査にも役立ちます。

V\$ ビューはユーザー SYS に属しますが、SYS 以外のユーザーは V\$ ビューに対して読み込み専用アクセスを行えます。V\$ ビューおよび X\$ 表は、インスタンス起動時に Oracle によってデータが移入されます。インスタンスをシャットダウンするとその内容はフラッシュされます。

X\$ 表とそれに関連する V\$ ビューは動的であるため、その内容も常に変化しています。init.ora のパラメータ TIMED\_STATISTICS を TRUE に設定した場合、または次の SQL コマンドを実行した場合は、X\$ 表に時間の情報が含まれます。

```
ALTER SYSTEM SET TIMED_STATISTICS=true;
```

# チューニングのためのインスタンスレベルのビュー

これらのビューはインスタンス全体に関係し、そのインスタンスの開始時からの統計を記録するか、（SGA 統計の場合は）現行の値を記録します。後者は、SGA 領域を再割当てする必要に応じて変更されるまでは一定です。累積の統計は開始時からのものです。

表 16-1 チューニングで重要なインスタンス・レベルのビュー

ビュー	説明
V\$FIXED_TABLE	リリースに存在する固定オブジェクトをリストします。
V\$INSTANCE	現行のインスタンスの状態を表示します。
V\$LATCH	親以外のラッチの統計および親ラッチの要約統計をリストします。
V\$LIBRARYCACHE	ライブラリ・キャッシュのパフォーマンスおよびアクティビティについての統計を含みます。
V\$ROLLSTAT	すべてのオンライン・ロールバック・セグメントの名前をリストします。
V\$ROWCACHE	データ・ディクショナリのアクティビティの統計を表示します。
V\$SGA	システム・グローバル領域についての要約情報を含みます。
V\$SGASTAT	システム・グローバル領域についての詳細情報を含みます。
V\$SORT_USAGE	一時セグメントのサイズと一時セグメントを作成するセッションが表示されます。この情報は、ディスク・ソートを行っているプロセスを識別するときに役立ちます。
V\$SQLAREA	共有 SQL 領域についての統計をリストします。SQL 文字列ごとに 1 行を含みます。メモリー内にある SQL 文、解析済みの SQL 文、実行準備ができた SQL 文についての統計を示します。テキストは 1000 文字までに制限されています。完全なテキストは 64 バイトのチャンクで V\$SQLTEXT から利用できます。
V\$SQLTEXT	SGA 内の共有 SQL カーソルに属する SQL 文のテキストを含みます。
V\$SYSSTAT	基本のインスタンス統計を含みます。
V\$SYSTEM_EVENT	あるイベントの合計待機時間についての情報を含みます。
V\$WAITSTAT	ブロック競合の統計をリストします。時間の統計（timed statistics）が使用可能な場合にのみ更新されます。

最も重要な固定ビューは V\$SYSSTAT です。これには、値に加えて統計名も含まれます。この表の値は、インスタンス・チューニング・プロセスの基本入力を形成します。

## チューニングのためのセッションレベルのビューまたは一時的なビュー

これらのビューは、セッション・レベルで操作することも、主に一時的な値を取り扱うこともできます。セッション・データは接続時から累計されます。

表 16-2 チューニングで重要なセッション・レベルのビュー

ビュー	説明
V\$LOCK	Oracle8 Server が現在保持しているロック、およびロックまたはラッチへの保留中の要求をリストします。
V\$MYSTAT	現行のセッションの統計を表示します。
V\$PROCESS	現在アクティブなプロセスについての情報を含みます。
V\$SESSION	各現行セッションのセッション情報をリストします。SID を他のセッション属性にリンクします。行ロック情報を含みます。
V\$SESSION_EVENT	セッションごとのイベントの待機時間についての情報をリストします。
V\$SESSION_WAIT	現行のイベントに対して WAIT_TIME = 0 の場合に、アクティブ・セッションが待機しているリソースまたはイベントをリストします。
V\$SESSTAT	ユーザーのセッションの統計をリストします。 V\$STATNAME および V\$SESSION との結合が必要です。

V\$SESSION\_WAIT の構造によって、任意のセッションが待機しているかどうか、および待機している理由をリアル・タイムでチェックすることが容易です。次に例を示します。

```
SELECT sid,  
       EVENT  
FROM V$SESSION_EVENT  
WHERE WAIT_TIME = 0;
```

また、そのような待機が頻繁に発生するかどうか、その他のイベント（特定モジュールの使用など）と関連付けられるかどうかを調査できます。

## 現行の統計値および変更率

この項では、以下の手順について説明します。

- [統計の現在の設定値の検索](#)
- [統計の変更率の検索](#)

### 統計の現在の設定値の検索

キーとなる率は、インスタンス統計に基づいて表されます。たとえば、consistent changes 率は、consistent changes を consistent gets で割ったものです。統計の現在の設定値を検索するための非常に単純で効果的な SQL\*Plus スクリプトは次の形式です。

```
COL NAME format a35
COL VALUE format 999,999,990
SELECT NAME, VALUE from V$SYSSTAT S
WHERE lower(NAME) LIKE lower('%&stat_name%')
/
```

---

---

**注意：** 上記の問合せの 2 つの LOWER ファンクションによって、大文字と小文字を区別しないようにし、名前が "CPU" または "DBWR" で始まる 11 の統計のデータをレポート作成するようにします。他の大文字は統計名にはありません。

---

---

たとえば、次の問合せを使って、名前に "get" という語を含むすべての統計のレポート作成ができます。

```
@STAT GET
```

ただし、この章の次の項で説明するように、一定の期間の統計の変化を記録するための方法を使用の方がよいでしょう。

## 統計の変更率の検索

以下のスクリプトを追加すると、すべての統計、ラッチまたはイベントの変更率を表示できます。指定した統計について、このスクリプトによってその値の2回のチェック間の秒数と変更率がわかります。

```

set veri off
define secs=0
define value=0
col value format 99,999,999,990 new_value value
col secs format a10 new_value secs noprint
col delta format 9,999,990
col delta_time format 9,990
col rate format 999,990.0
col name format a30
select name,value, to_char(sysdate,'sssss') secs,
       (value - &value) delta,
       (to_char(sysdate,'sssss') - &secs) delta_time,
       (value - &value)/ (to_char(sysdate,'sssss') - &secs) rate
from v$sysstat
where name = '&&stat_name'

/

```

---

---

**注意：** 最初の実行時には SQL\*Plus 変数が初期化されるため、このスクリプトは少なくとも2回実行してください。

---

---





---

## システムのパフォーマンス問題の診断

---

この章では、適切に設計されたシステムでパフォーマンスに影響する要因について説明します。設計に問題がある場合は、この章のガイドラインに従ってもシステムは改善されません。

- 適切に設計された既存のシステムの要因のチューニング
- 不十分な CPU
- 不十分なメモリー
- 不十分な I/O
- ネットワークの制約
- ソフトウェアの制約

後の章では、これらの各要因を詳細に説明します。

### 適切に設計された既存のシステムの要因のチューニング

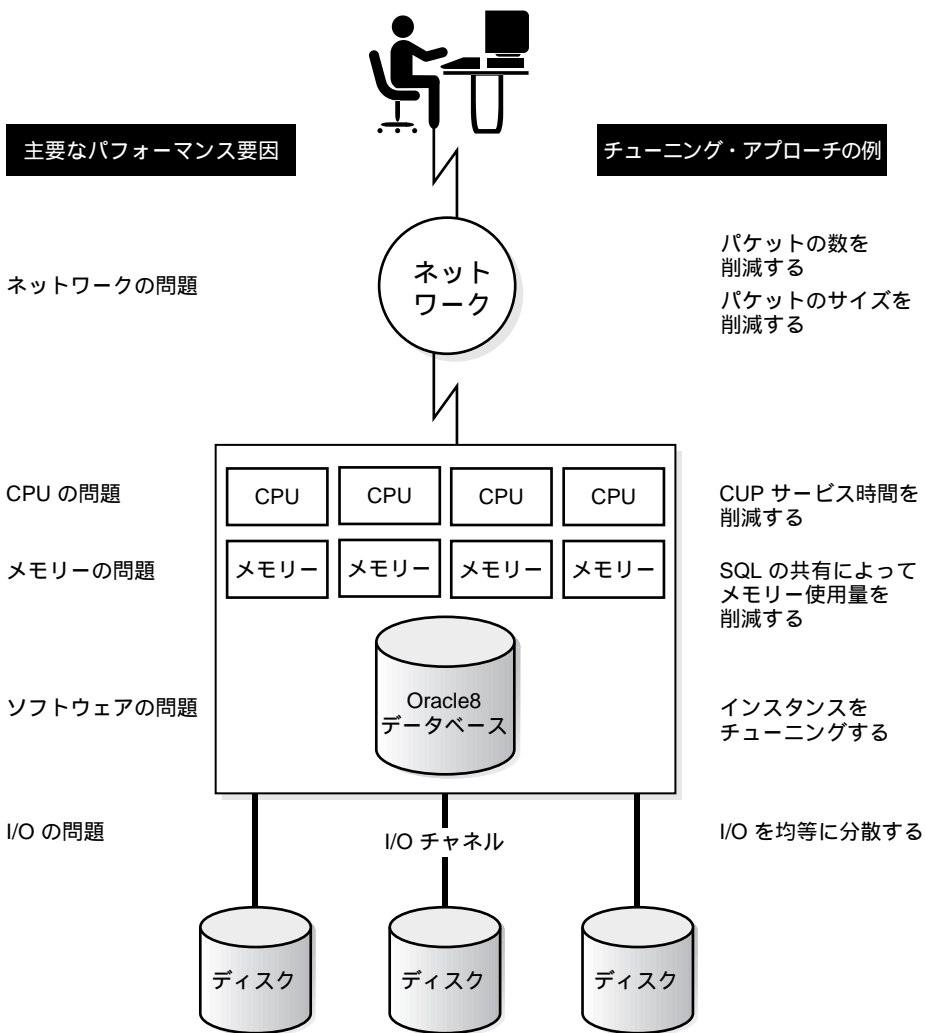
図 17-1 は、適切に設計されたアプリケーションについて、Oracle システムのパフォーマンスに関連する要因を示しています。

---

**注意：** これらの要因のチューニングは、第 2 章「パフォーマンス・チューニングの方法」で説明した業務処理とアプリケーションのチューニングの後でなければ効果がありません。

---

図 17-1 適切に設計されたシステムでの主要なパフォーマンス要因



パフォーマンス問題は、個々に独立していて無関係なのではなく相互に関連する傾向があります。表 17-1 は、既存のシステムのキーとなるパフォーマンス要因と、症状が現れる可能性のある領域を示しています。たとえば、バッファ・キャッシュの問題は、CPU、メモリーまたは I/O の問題として現れることがあります。このため、CPU のためにバッファ・キャッシュをチューニングすると、I/O が改善されることがあります。

表 17-1 既存システムでのチューニング領域のキー

ORACLE チューニング領域	制限リソース				
	CPU	メモリー	I/O	ネット ワーク	ソフト ウェア
<b>アプリケーション</b>					
設計 / アーキテクチャ	X	X	X	X	X
DML SQL	X	X	X	X	X
問合せ SQL	X	X	X	X	X
クライアント / サーバーの往復	X			X	
<b>インスタンス</b>					
バッファ・キャッシュ	X	X	X		
共有プール	X	X			
ソート領域	X	X	X		
データ / DB ファイルの物理構造 の I/O	X		X		
ログ・ファイルの I/O		X	X		
アーカイバの I/O	X		X		
ロールバック・セグメント			X		X
ロック	X	X	X		X
バックアップ	X		X	X	X
<b>オペレーティング・システム</b>					
メモリー管理	X	X	X		
I/O 管理	X	X	X		
プロセス管理	X	X			
ネットワーク管理		X		X	

## 不十分な CPU

CPU バウンドのシステムでは、CPU リソースがすべて割り当てられ、サービス時間が非常に長くなることがあります。このような状況では、システムの処理能力を改善する必要があります。または、アイドル時間が非常に長い、CPU が十分に使用されているわけではないことがあります。どちらの場合も、待機時間が非常に長い原因を調べる必要があります。

CPU が不十分となった原因を調べるには、システム全体での CPU の使用状況を識別してください。Oracle Server プロセスでの CPU 使用状況のみで判断しないでください。たとえば、就業日の開始時には、従業員がメッセージをチェックするので、使用可能な CPU をメール・システムが大量に消費する可能性があります。それ以降は、メール・システムの使用率はずっと低くなり、それに応じてメール・システムの CPU 使用率も低下します。

システムの CPU 使用レベルを評価するときには、作業負荷が非常に重要な要因となります。作業負荷がピークの時間帯に、CPU 使用率が 90%、アイドル時間および待機時間が 10% となるのは理解および許容できます。作業負荷が低い時間帯で使用率が 30% となるのも理解できます。しかし、システムが標準の作業負荷で高い使用率を示している場合は、ピーク時の作業負荷に耐える余裕がありません。標準または低い作業負荷で、アイドル時間と I/O の待機時間が両方とも 0 に近い（または 5% 未満）場合は、CPU に問題があります。

**関連項目：** [第 18 章「CPU リソースのチューニング」](#)

## 不十分なメモリー

メモリーの問題は、I/O の問題として検出されることがあります。メモリー所要量には、Oracle とシステムの 2 種類があります。Oracle のメモリー所要量は、システムの所要量に影響します。メモリーの問題は、マシンで発生するページングとスワッピングの原因になることがあります。そのため、システムがページングとスワッピングを開始しないことを確認してください。内部メモリーによって設定された制限内でシステムを実行できなければなりません。

Oracle 以外のプロセスのためのシステム・メモリー所要量と Oracle のメモリー所要量を加算した容量は、使用可能な内部メモリー量の合計以下でなければなりません。これを達成するためには、Oracle のいくつかのメモリー構造（バッファ・キャッシュ、共有プールまたは REDO ログ・バッファなど）のサイズを縮小します。システム・レベルでは、プロセスの数または各プロセスが使用するメモリー量、あるいはその両方を削減できます。どのプロセスが最も多くメモリーを使っているかを識別することもできます。メモリー使用量を削減する方法の 1 つは、SQL の共有です。

**関連項目：** [第 19 章「メモリー割当てのチューニング」](#)

## 不十分な I/O

I/O は、各ディスクとチャンネル上に均等に分散するようにしてください。I/O リソースには次のものが含まれます。

- チャンネル帯域幅 : I/O チャンネルの数
- デバイス帯域幅 : ディスクの数
- デバイス待ち時間 : 要求の開始から要求の受取りまでの経過時間。待ち時間 (latency) は待機時間 (wait time) に含まれます。

I/O の問題は、ハードウェアの制約から生じることがあります。システムには、必要なトランザクション・スループットをサポートするのに十分なディスクと SCSI バスが必要です。すべてのディスクとバスが潜在的にサポートできるメッセージ数を計算し、それをピークの作業負荷で必要とされるメッセージ数と比較することによって、構成を評価できます。

I/O の応答時間が長すぎる場合、最も多い問題は待機時間の増加です (応答時間 = サービス時間 + 待機時間)。待機時間が増加している場合は、デバイスに対する I/O 要求が多すぎることを意味します。サービス時間が増加すると、通常は大規模な I/O 要求を伴い、より多くのバイト数がディスクに書き込まれます。

さまざまなバックグラウンド・プロセス (DBWR、ARCH など) が実行する I/O の種類はさまざまであり、各プロセスは異なる I/O 特性を持ちます。データベースのブロック・サイズで読み込みと書き込みを行うプロセスもあれば、より大きな単位で読み込みと書き込みを行うプロセスもあります。サービス時間が長すぎる場合は、さまざまなデバイスにファイルをストライプ化してください。

元のデータベースと同数のディスクを持つデータベースにデータがミラー化されていないかぎり、ミラー化も I/O ボトルネックの原因となります。

**関連項目：** [第 20 章「I/O のチューニング」](#)

## ネットワークの制約

ネットワークの制約は、I/O の制約と似ています。次のことを考慮する必要があります。

- ネットワーク帯域幅 : 各トランザクションは、特定の個数のパケットがネットワーク上で送信されることを要求します。あるトランザクションで要求されるパケットの個数がわかっている場合は、それを帯域幅と比較し、希望する作業負荷をサポートする能力がシステムにあるかどうかを判断できます。
- メッセージ割合 : 小さなパケットを多数送信するかわりに、それらをバッチ化することによって、ネットワーク上のパケットの個数を削減できます。
- 送信時間

ユーザー数と需要が増えるにつれて、ネットワークがアプリケーションのボトルネックとなることがあります。ネットワーク使用の待機に長時間を費すことがあります。使用可能なオ

オペレーティング・システム・ツールを使って、ネットワークがどの程度ビジーかを確認してください。

**関連項目：** [第 22 章「ネットワークのチューニング」](#)

## ソフトウェアの制約

オペレーティング・システム・ソフトウェアは、次のことを決定します。

- サポートできるプロセスの最大数
- 接続できるプロセスの最大数

Oracle を効率的にチューニングするには、その前にオペレーティング・システムのパフォーマンスが最適であることを確認してください。ハードウェア / ソフトウェア・システムの管理者と緊密に作業を進め、Oracle が適切なオペレーティング・システム・リソースに割り当てられるようにしてください。

---

**注意：** NT システムでは、サポートまたは接続できるプロセスの最大数を事前設定または設定変更できません。

---

**関連項目：** オペレーティング・システムのチューニングはプラットフォームによって異なります。詳細については、オペレーティング・システムのハードウェアとソフトウェアのマニュアル、およびオペレーティング・システム固有の Oracle マニュアルを参照してください。さらに、[第 24 章「オペレーティング・システムのチューニング」](#)を参照してください。

---

## CPU リソースのチューニング

この章では、CPU のリソースの問題を解決する方法を説明します。トピックは次のとおりです。

- [CPU の問題について](#)
- [CPU 問題の検出と解決](#)
- [システム・アーキテクチャの変更による CPU の問題の解決](#)

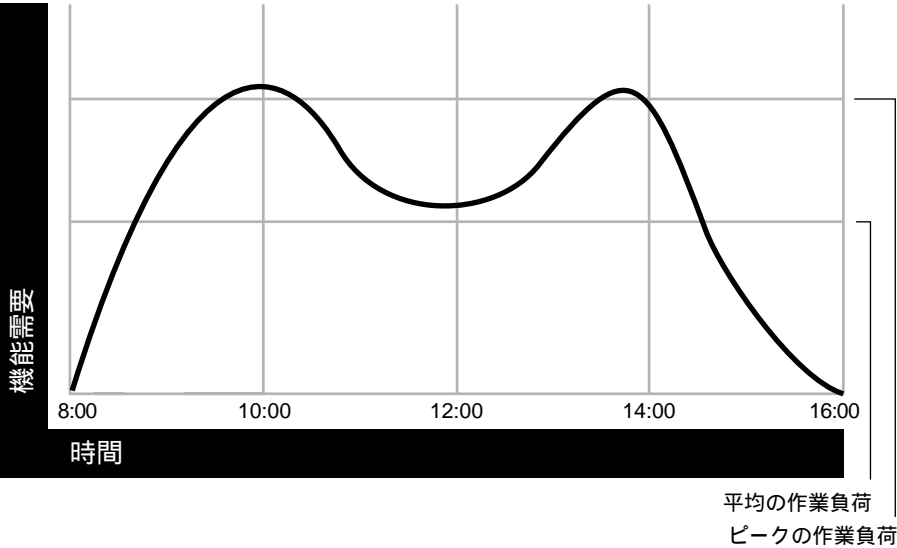
### CPU の問題について

CPU の問題を扱うには、まずシステムが使用する CPU リソースの量について、適切な見積りを設定します。次に、十分な CPU リソースが使用可能かどうかを判別し、システムがいつリソースを使用しすぎているかを認識します。最初に、次の 3 つのシステムの状況について Oracle インスタンスが利用する CPU リソースの量を判別します。

- アイドルのとき
- 作業負荷が平均のとき
- 作業負荷がピークのとき

システムの CPU 使用率のレベルを評価するときには、作業負荷が重要な要因となります。作業負荷がピークの時間帯に、CPU 使用率が 90% でアイドル状態が 10% となり待機時間が発生するのは受容できます。作業負荷が低い時間帯で使用率が 30% となるのも理解できます。しかし、システムが標準の作業負荷で高い使用率を示している場合は、ピークの作業負荷に耐える余裕がありません。たとえば、[図 18-1](#) に、午前 10:00 と午後 2:00 にピークとなるアプリケーションの作業負荷の時間に伴う変化を示します。

図 18-1 平均の作業負荷とピークの作業負荷



この例のアプリケーションでは、1 日に 8 時間作業する 100 人のユーザーがいるので、合計作業時間は 1 日あたり 800 時間となります。各ユーザーが 5 分ごとに 1 つのトランザクションを入力すると、1 日のトランザクションは 9,600 になります。8 時間にわたってシステムは 1 時間あたり 1,200 のトランザクションをサポートする必要がある、1 分あたり平均 20 のトランザクションをサポートしなければならないことになります。需要率が一定の場合は、この平均作業負荷を満たすようにシステムを構築します。

ただし、使用率のパターンは一定ではないので、この場合、1 分あたり 20 のトランザクションというのは単なる最低条件と考えられます。達成しなければならないピークの割合が 1 分あたり 120 トランザクションの場合は、このピークの作業負荷をサポートできるシステムを構成する必要があります。

この例で、作業負荷がピークのときに Oracle が CPU リソースの 90% を使えるとします。その場合、作業負荷が平均の期間では、次の等式が示すように、Oracle は使用可能な CPU リソースの 15% を使用するにすぎません。

$$20 \text{ tpm} / 120 \text{ tpm} * 90\% = 15\%$$

ここで、tpm は 1 分あたりのトランザクション数を表します。



20tpm を達成するためにシステムが CPU リソースの 50% を必要とする場合は、問題があることは明かです。つまり、このシステムは、CPU の 90% を使用しても 1 分あたり 120 のトランザクションを処理できません。ただし、CPU の 15% のみを使用して 20tpm を達成するようにこのシステムをチューニングした場合は、（線形の拡張を前提とすると）CPU リソースの 90% を使用して 1 分あたり 120 のトランザクションを処理できるでしょう。

アプリケーションを使用するユーザーが増加するにつれて、作業負荷がかつてのピーク・レベルにまで上昇する可能性があります。そのときには、以前よりも実際に高くなった新しいピーク割合に対応できる CPU 能力はありません。

## CPU 問題の検出と解決

CPU の使用率に問題があると思われる場合は、次の 2 つの項目をチェックしてください。

- システムの CPU 使用率
- Oracle の CPU 使用率

### システムの CPU 使用率

Oracle の統計は、Oracle セッションの CPU 使用率のみをレポートしますが、使用可能な CPU リソースには、システム上で実行しているすべてのプロセスが影響します。したがって、Oracle 以外の要因をチューニングする作業によって、Oracle のパフォーマンスも向上することがあります。

オペレーティング・システムの監視ツールを使用して、システム全体で実行されているプロセスを確認してください。システムの負荷が非常に高い場合は、この項で後述するメモリ、I/O およびプロセス管理の各項目をチェックしてください。

多くの UNIX ベースのシステムにある `sar -u` などのツールを使用すると、システム全体での CPU 使用率のレベルを調べることができます。UNIX での CPU 使用率は、ユーザー時間、システム時間、アイドル時間および I/O の待機時間を示す統計で説明されます。作業負荷が標準または低いときに、アイドル時間と I/O の待機時間が両方とも 0 に近い（5% 未満）場合は、CPU の問題が存在します。

NT では、パフォーマンス モニタを使用して CPU 使用率を調べることができます。Performance Manager は、プロセッサ時間およびユーザー時間、特権時間 (privileged time)、割込み時間、DPC 時間の統計を提供します（NT のパフォーマンス モニタは Performance Manager とは違います。Performance Manager は Oracle Enterprise Manager ツールです）。

---

**注意：** この項では、ほとんどの UNIX ベース・システムおよび NT システムにおける CPU 使用率をチェックする方法を説明します。その他のプラットフォームについては、使用しているオペレーティング・システムのマニュアルを参照してください。

---

## メモリー管理

次のメモリー管理項目をチェックします。

**ページングとスワッピング** 適切なツール（UNIX の **sar**、**vmstat** または NT のパフォーマンス モニタなど）を使用して、ページングとスワッピングの原因を調査します。

**大きくなりすぎるページ・テーブル** UNIX では、処理領域が大きくなりすぎると、ページ・テーブルが大きくなりすぎることがあります。これは NT では問題になりません。

## I/O 管理

次の I/O 管理の項目をチェックします。

**スラッシング** マシンのスラッシング（メモリー内外のスワッピングおよびページング処理）が発生しないように、作業負荷がメモリーに適合するようにしてください。オペレーティング・システムは固定サイズのタイム・スライスユーザーのプロセスに割り当て、プロセスはそのタイム・スライス中に CPU リソースを使用できます。プロセスが実行可能かどうか、および必要な構成要素がすべてマシンにあるかどうかを確認するときに、プロセスが大半の時間を浪費すると、実際の作業の実行に割り当てられる時間はわずか 50% となることがあります。

**クライアント / サーバーの往復メッセージ送信の待ち時間**は、CPU の過負荷の原因となることがあります。アプリケーションは、ネットワークを介して何度も送信する必要のあるメッセージを生成することがよくあります。このために、メッセージが実際に送信される前に大きなオーバーヘッドが発生します。この問題を軽減するためには、これらのメッセージをまとめてオーバーヘッドを一度だけ実行するか、作業の量を削減してください。たとえば、配列挿入、配列フェッチなどを使用できます。

---

**関連項目：** I/O のチューニングの詳細は、[第 20 章「I/O のチューニング」](#)を参照してください。

---

## プロセス管理

次のプロセスの管理の項目をチェックします。

**スケジューリングとスイッチング** オペレーティング・システムはスケジューリングおよびスイッチング処理に大量の時間を消費することがあります。使用しているプロセスが多すぎる場合があるので、オペレーティング・システムの使用方法を検証してください。NT システムでは、Oracle 以外の大量の処理によってサーバーが過負荷にならないようにしてください。

**コンテキストのスイッチング** オペレーティング・システム固有の特性によって、システムがコンテキストのスイッチングに大量の時間を消費することがあります。コンテキストのスイッチングは、特に超大規模 SGA の場合に不経済です。インスタンスあたりのプロセスが 1 つしかない NT では、コンテキストのスイッチングは問題になりません。すべてのスレッドが同じページ・テーブルを共有します。

プログラマは、単一目的のプロセスを作成し、プロセスを終了し、また新規のプロセスを作成することがよくあります。この場合、そのつどプロセスの再作成と破壊が行われます。この方法では、特に大規模な SGA を持つアプリケーションで CPU が集中して使用されます。これは、そのたびにページ・テーブルを構築する必要があるからです。共有メモリを確保またはロックするときは、すべてのページにアクセスしなければならないので、問題が増大します。

たとえば、1GB の SGA がある場合は、4KB ごとにページ・テーブル・エントリがあり、1 つのページ・テーブル・エントリは 8 バイトになります。エントリのサイズの合計は  $(1\text{G} / 4\text{K}) * 8\text{B}$  となります。ページ・テーブルがロードされていることを絶えず確認する必要があるので、これは不経済になります。

パラレル実行とマルチスレッド・サーバーについては、MINSERVICE の設定が低すぎるとき（たとえば、20 が必要な場合に 10 に設定されているとき）に検討する必要があります。小規模な検索を実行するアプリケーションについては、これは賢明とはいえません。このような状況は、アプリケーションとシステムの両方にとって非効率的です。

## Oracle の CPU 使用率

この項では、Oracle で実行されているプロセスの検証方法を説明します。次の 2 つの動的パフォーマンス・ビューによって、Oracle プロセスについての情報が提供されます。

- V\$SYSSTAT は、すべてのセッションにおける Oracle の CPU 使用率を示します。統計「CPU Used」は、すべてのセッションで使用されている CPU の合計を示します。
- V\$SESSTAT は、セッションごとの Oracle の CPU 使用率を示します。このビューを使用して、特にどのセッションが CPU の大部分を使用しているかを判断できます。

たとえば、8 個の CPU がある場合は、実時間のどの 1 分間をとっても CPU 時間上の 8 分間が使用可能です。NT および UNIX では、これはユーザー時間またはシステム・モードの時間（NT では「特権モード」）となります。ユーザーのプロセスは実行されていない場合は、待機しています。このように、すべてのシステムが利用する CPU 時間は、1 間隔あたり 1 分よりも長くなります。

ある時点で Oracle が使用したシステムの時間の長さはわかります。したがって、8 分が使用可能で Oracle がそのうちの 4 分を使っている場合は、総 CPU 時間の 50% が Oracle によって使用されていることがわかります。ユーザーのプロセスがその時間を消費していない場合は、他のプロセスが消費しています。その場合は、CPU 時間を使用しているプロセスを識別する必要があります。識別できたら、そのプロセスが大量の CPU 時間を消費している理由を判断し、チューニングを試みてください。

Oracle の CPU 使用率についてチェックする主な項目を次に示します。

- SQL 文の再解析
- 効率の悪い SQL 文
- 読み込みの一貫性
- アプリケーションの拡張性の制約

## ■ ラッチの競合

この項では、それぞれの項目を説明し、それに対応してチェックする必要のある Oracle 統計を示します。

## SQL 文の再解析

Oracle が SQL 文を実行する場合、SQL 文を解析して構文とその内容が正しいかどうかを判別します。このプロセスが大きなオーバーヘッドとなることがあります。Oracle では、いったん解析が行われると、メモリー・キャッシュ内の解析情報が古くなって使用不能にならないかぎり、文の再解析は行われません。

SQL 文における効率の悪いメモリー共有は再解析の原因になります。次の手順に従って、再解析が発生するかどうかを判別します。

1. 最初に、V\$SYSSTAT をチェックして、解析が一般に問題となっているかどうかを確認します。

```
SELECT * FROM V$SYSSTAT
WHERE NAME IN
('parse time cpu', 'parse time elapsed', 'parse count (hard)');
```

このとき、次のようになります。

応答時間 = サービス時間 + 待機時間。したがって、応答時間 = 経過時間となります。

サービス時間 = CPU 時間。したがって、経過時間 - CPU 時間 = 待機時間となります。

このようにして、解析のおおよその応答時間を検出できます。アプリケーションの解析が長くなるほど、競合が多くなり、システムの待機時間が長くなります。次のことに注意してください。

待機時間 / 解析件数 = 解析 1 件あたりの平均待機時間

---

---

**注意：** 平均待機時間は、非常に低くし、ゼロに近づける必要があります (V\$SYSSTAT も、解析 1 回あたりの平均待機時間を示します)。

---

---

2. V\$SQLAREA に問い合せて、頻繁に再解析される文を検出します。

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS FROM V$SQLAREA
ORDER BY PARSE_CALLS;
```

解析コールが多い文をチューニングします。

3. 文のチューニングには次の 3 つのオプションがあります。

- 文が繰り返し再解析されないようにアプリケーションを書き直します。
- 初期化パラメータ `SESSION_CACHED_CURSORS` を使用して解析を削減します。
- 解析件数と実行件数が少なく、WHERE 句を除く SQL 文が非常に似ている場合は、バインド変数のかわりにハード・コーディングされた値が使用されていることがあります。バインド変数を使用して解析を削減します。

## 効率の悪い SQL 文

効率の悪い SQL 文は、大量の CPU リソースを消費する可能性があります。そのような文を検出するには、次の問合せを入力します。多くのバッファを取得する SQL 文をチューニングすることで、CPU 使用率を削減できることがあります。

```
SELECT BUFFER_GETS, EXECUTIONS, SQL_TEXT FROM V$SQLAREA;
```

**関連項目：** 4-6 ページの「[SQL 文のチューニングのアプローチ](#)」

## 読み込みの一貫性

システムは、一貫したビューを維持するために、ブロックの変更内容のロールバックに長い時間を費やすことがあります。次の状況を考慮してください。

- 小さなトランザクションが大量にあり、挿入を実行中の表に対して、長時間を要するアクティブな問合せがバックグラウンドで実行されている場合に、問合せが多くの変更内容をロールバックしなければならないことがあります。
- ロールバック・セグメントの数が少なすぎる場合は、トランザクション表のロールバックにも長い時間を費やすことがあります。その問合せがかなり以前に開始している場合があります。つまり、ロールバック・セグメントとトランザクション表の数が非常に少ないので、トランザクション・スロットを終始再使用する必要があるからです。

さらにロールバック・セグメントを作成するか、コミットの割合を増やすとよいでしょう。たとえば、10 個のトランザクションを 1 つのバッチにまとめてコミットを 1 回で済ませると、トランザクション数が 10 分の 1 に削減されます。

- システムが使用可能バッファを見つけるために、あまりにも多くのバッファをフォアグラウンドで走査する必要がある場合は、CPU リソースが浪費されています。この問題を軽減するには、`DBWn` プロセスをチューニングして、より頻繁に書き込みを行うようにします。

また、データベース・ライター・プロセスが継続できるようにバッファ・キャッシュのサイズを増やすこともできます。使用可能バッファを見つけるために、最低使用頻度リスト (LRU) の末尾でシステムが走査するバッファ数の平均を算出するには、次の計算式を使います。

$$\frac{1 + \text{"free buffers inspected"}}{\text{"free buffers inspected"}} = \text{走査された平均バッファ数}$$

平均して 1 個または 2 個のバッファが走査されると予想されます。走査される数がこれよりも多い場合は、バッファ・キャッシュのサイズを増やすか、DBWn プロセスをチューニングします。

次の計算式を使用して、使用済バッファ数を LRU の最後で検索します。

$$\frac{\text{"dirty buffers inspected"}}{\text{"free buffers inspected"}} = \text{使用済みバッファ数}$$

使用済バッファが多い場合は、DBWn プロセスが継続できないことを意味します。この場合も、バッファ・キャッシュ・サイズを増やすか、DBWn プロセスをチューニングします。

## アプリケーションの拡張性の制約

CPU のチューニングに関する説明のほとんどは、線形の拡張性が達成できることを想定していますが、実際は線形にはなりません。拡張性がどの程度平坦または非線形であるかによって、システムが最適なパフォーマンスからどれだけかけ離れているかが示されます。アプリケーションの問題が、拡張性に悪影響を与えることがあります。例として、索引が多すぎる、重要な索引の問題、ブロックのデータが多すぎる、データが適切にパーティション化されていないことなどがあります。この種の競合の問題は、CPU サイクルを浪費し、アプリケーションの拡張性が線形となることを阻害します。

## ラッチの競合

ラッチの競合は CPU の問題の症状であり、通常は原因ではありません。これを解決するには、ラッチの競合をアプリケーション内で検索し、原因を識別し、アプリケーションのどの部分に問題があるかを判断してください。

場合によっては、スピン件数 (spin count) が高く設定されすぎていることがあります。また、あるプロセスが確保しようとするラッチを別のプロセスが保持していることもあります。ラッチを確保しようとするプロセスは、無限にスピンし続けることになります。しばらくすると、このプロセスはスリープしますが、その後、処理を再開し無駄なスピンを繰り返します。これを解決するには、次のようにします。

- Oracle ラッチ統計をチェックします。V\$SYSTEM\_EVENT の "latch free" イベントは、プロセスがラッチを待機していた時間の長さを示します。ラッチの競合が発生していない場合は、この統計は表示されません。

競合が大量に発生した場合に、プロセスがラッチを取得できないときは、スピンを行って CPU を使用するよりも、プロセスをすぐにスリープさせる方がよいでしょう。

- プロセスに対する CPU の比率を調べます。どちらも非常に多い場合は、多くのプロセスを実行できます。ただし、10 個の CPU があるシステムで 1 つのプロセスがラッチを保持している場合は、そのプロセスをリスケジュールして、実行を停止してください。それでも、他の 10 個のプロセスが同じラッチを確保しようとして無駄に実行する場合があります。このような状態は並行して CPU リソースを浪費します。
- ほとんどの競合が Oracle のコードのどの部分で発生しているかを示す `VSLATCH_MISSES` をチェックします。

## システム・アーキテクチャの変更による CPU の問題の解決

システムの CPU 能力を最大限にし、システムの CPU 使用をチューニングするあらゆる手段を実行し尽くした場合は、別のアーキテクチャでシステムを再設計することを考慮してください。別のアーキテクチャに移動すると、CPU の使用状況が改善されることがあります。この項では、次のような考慮すべきアーキテクチャについて説明します。

- [単層から 2 層へ](#)
- [複数層：より小さなクライアント・マシンの使用](#)
- [2 層から 3 層へ：トランザクション処理モニターの使用](#)
- [3 層：複数の TP モニターの使用](#)
- [Oracle Parallel Server](#)

---

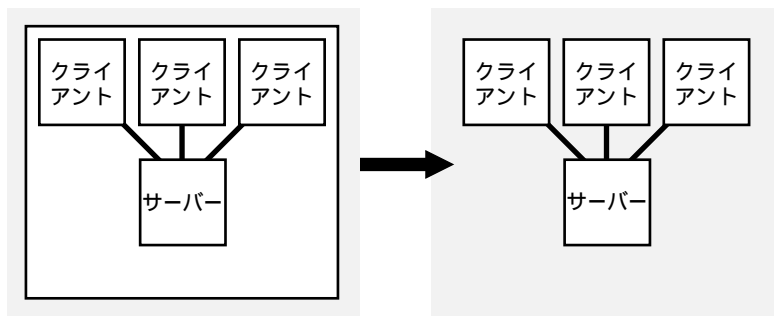
**注意：** 複数層システムを実行している場合は、CPU 使用率のあらゆるレベルを完全にチェックしてください。たとえば、3 層システムで、サーバーがほとんどアイドルであるのに対して 2 番目の層は非常にビジーということがあります。これを解決するには、サーバーや 3 番目の層ではなく 2 番目の層をチューニングします。複数層のシステムでは、パフォーマンスに問題があるのは通常はサーバーではありません。通常は、クライアントおよび中間の層に問題があります。

---

## 単層から 2 層へ

1 台のマシンで 1 つのサーバーといくつかのクライアントをすべて実行している状態（単層）から、2 層のクライアント / サーバー構成への変更が、CPU 問題を軽減するかどうかを検討します。

図 18-2 単層から 2 層へ

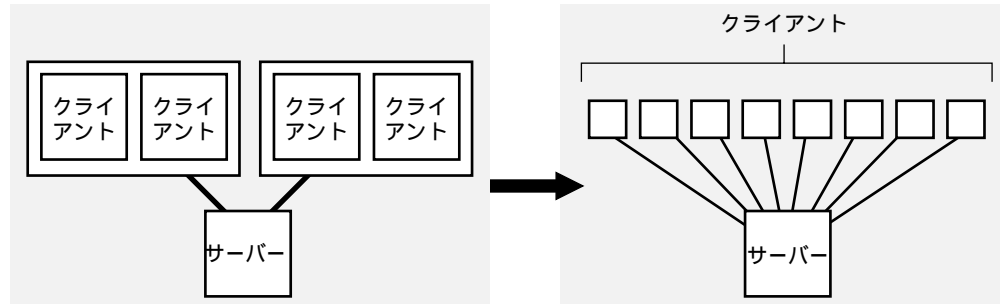




## 複数層：より小さなクライアント・マシンの使用

大きなマシン上で複数のクライアントを使用するかわりに小さなクライアントを使用すると、CPU 使用率が改善されるかどうかを検討します。この方針は、2 層または 3 層の構成に役立つことがあります。

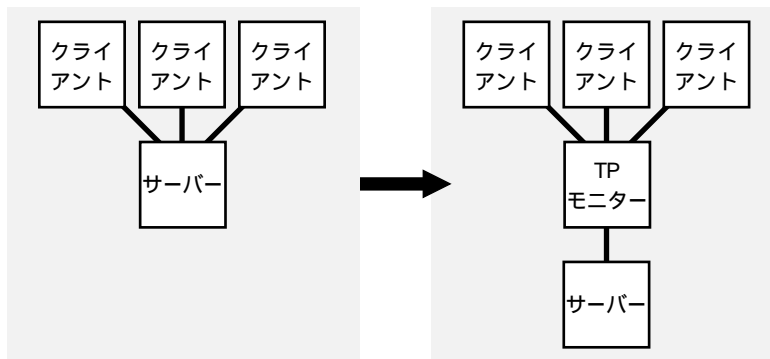
図 18-3 より小さなクライアント・マシンを使用する複数層



## 2 層から 3 層へ：トランザクション処理モニターの使用

システムが複数の層で稼働している場合は、2 層構成から 3 層構成への移行とトランザクション処理モニターの導入が優れた解決策となるかどうかを検討します。

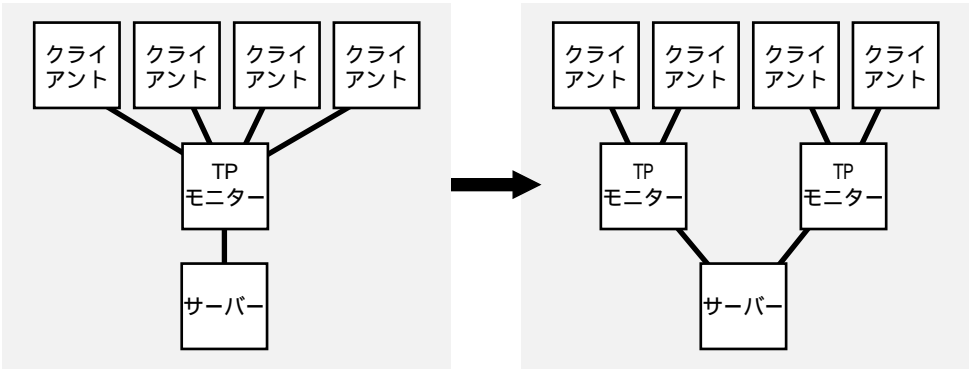
図 18-4 2 層から 3 層へ



### 3 層 : 複数の TP モニターの使用

複数のトランザクション処理モニターの使用を検討します。

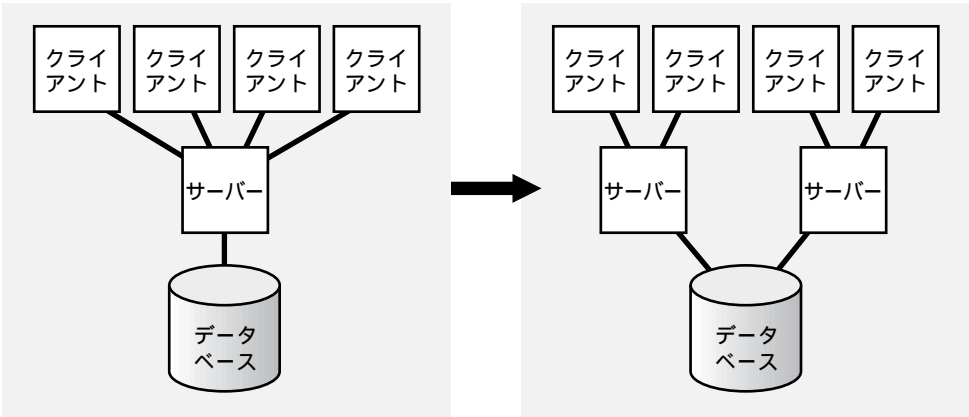
図 18-5 複数の TP モニターを持つ 3 層



### Oracle Parallel Server

Oracle Parallel Server を組み込むことによって、CPU の問題が解決するかどうかを検討します。

図 18-6 Oracle Parallel Server



---

## メモリー割当てのチューニング

この章では、データベース構造体にメモリーを割り当てる方法について説明します。これらの構造体のサイズを適切に設定すると、データベースのパフォーマンスが大幅に向上します。トピックは次のとおりです。

- [メモリー割当ての問題について](#)
- [メモリー割当ての問題の検出方法](#)
- [メモリー割当ての問題の解決方法](#)
  - [オペレーティング・システムのメモリー所要量のチューニング](#)
  - [REDO ログ・バッファのチューニング](#)
  - [プライベート SQL と PL/SQL 領域のチューニング](#)
  - [共有プールのチューニング](#)
  - [バッファ・キャッシュのチューニング](#)
  - [複数バッファ・プールのチューニング](#)
  - [ソート領域のチューニング](#)
  - [メモリーの再割当て](#)
  - [合計メモリー使用量の低減](#)

### メモリー割当ての問題について

Oracle はメモリー内およびディスク上に情報を格納します。メモリー・アクセスはディスク・アクセスよりもかなり高速なため、ディスクよりもメモリーをアクセスすることによってデータ要求に応えることが望まれます。パフォーマンスを改善するには、ディスク上ではなくメモリー内にできる限り多くのデータを格納してください。しかし、オペレーティング・システム上のメモリー・リソースは限られています。メモリー割当てのチューニングには、Oracle メモリー構造に利用可能なメモリーを配分することが含まれます。

Oracle のメモリー所要量はアプリケーションによって異なります。このため、アプリケーションおよび SQL 文をチューニングした後でメモリー割当てをチューニングしてください。アプリケーションと SQL 文をチューニングする前にメモリーを割り当てると、修正した文とアプリケーションのニーズに合わせるために、Oracle メモリー構造のサイズを再設定することが必要となることもあるでしょう。

また、I/O をチューニングする前にメモリー割当てをチューニングします。メモリーを割り当てることによって、Oracle を作動させるために必要な I/O の総量が決まります。この章では、最小限の I/O を実行するようにメモリーを割り当てる方法を示します。

この説明では、次の用語を使用します。

ブロック	主メモリーとディスク間でのデータ転送の単位。メモリー・アドレス領域の 1 セクションにある多数のブロックが 1 つのセグメントを形成します。
バッファ	ディスクから読み取られて、現在使用されているデータまたは最近使用されたデータを、バッファ・マネージャがキャッシュする主メモリーのアドレス。時間の経過につれて、バッファが保持するブロックは変わる場合があります。新しいブロックが必要なときは、バッファ・マネージャは古いブロックを破棄して、新しいブロックで置き換えることができます。
バッファ・プール	バッファの集まり。
キャッシュまたはバッファ・キャッシュ	すべてのバッファおよびバッファ・プール。
セグメント	セグメントは、表、索引、クラスタなど、特定の種類のデータベース・オブジェクトのために割り当てられている一連のエクステンツです。

**関連項目：** [第 20 章「I/O のチューニング」](#) では、可能な限り効率的に I/O を実行する方法について説明します。

## メモリ割当ての問題の検出方法

UNIX で `ps -efl` または `ps -aux` などのオペレーティング・システムのツールを使用して Oracle プロセスのサイズを調べると、プロセスが大きいことがわかるでしょう。表示されている統計を解釈するには、共有メモリ、ヒープおよび実行可能スタックに起因するプロセス・サイズの割合と、指定されたプロセスが消費する実際のメモリ容量を判別してください。

SZ 統計はページ・サイズ単位（通常 4KB）で提供され、通常、共用オーバーヘッドを含みます。プライベートまたはプロセスごとのメモリ使用量を計算するには、SZ 値から共有メモリと実行可能スタックの数値を引きます。次に例を示します。

SZ	+20,000
SHM を引く	- 15,000
EXECUTABLE を引く	- <u>1,000</u>
実際のプロセスごとのメモリ	4,000

この例では、個々のプロセスは 4,000 ページしか消費していません。残りの 16,000 ページはすべてのプロセスが共有しています。

**関連項目：**『Oracle for UNIX Performance Tuning Tips』または使用しているオペレーティング・システムのマニュアル

## メモリ割当ての問題の解決方法

この章では、この後、メモリ割当てのチューニング方法を説明します。最もよい結果を得るには、次に示す順序でメモリの問題を解決してください。

1. [オペレーティング・システムのメモリ所要量のチューニング](#)
2. [REDO ログ・バッファのチューニング](#)
3. [プライベート SQL と PL/SQL 領域のチューニング](#)
4. [共有プールのチューニング](#)
5. [バッファ・キャッシュのチューニング](#)
6. [複数バッファ・プールのチューニング](#)
7. [ソート領域のチューニング](#)
8. [メモリの再割当て](#)
9. [合計メモリ使用量の低減](#)

## オペレーティング・システムのメモリー所要量のチューニング

以下を目標に、オペレーティング・システムをチューニングすることによって、メモリー割当てのチューニングを開始します。

- [ページングとスワッピングの低減](#)
- [主メモリーへのシステム・グローバル領域 \(SGA\) の格納](#)
- [個々のユーザーへの十分なメモリーの割当て](#)

一般にこれらの目標はほとんどのオペレーティング・システムに適用されますが、チューニングの詳細はオペレーティング・システムによって異なります。

**関連項目：** オペレーティング・システムのメモリー使用方法のチューニングの詳細は、オペレーティング・システムのハードウェアとソフトウェアのマニュアル、およびオペレーティング・システム固有の Oracle マニュアルを参照してください。

### ページングとスワッピングの低減

オペレーティング・システムは、次の場所に情報を格納します。

- 実メモリー
- 仮想メモリー
- 拡張記憶
- ディスク

オペレーティング・システムは、ある記憶位置から別の記憶位置へ情報を移動することもあります。このプロセスは、" ページング " または " スワッピング " と呼ばれます。多くのオペレーティング・システムは、実メモリーに入りきれない大量の情報を収容するために、ページングとスワッピングを行います。ただし、過度のページングやスワッピングは、多くのオペレーティング・システムのパフォーマンスを低下させます。

オペレーティング・システム・ユーティリティによって、オペレーティング・システムの動作を監視してください。過度のページングとスワッピングは、頻繁に新しい情報がメモリーに移動されていることを示します。この場合、システムの全体のメモリーが、メモリーを割り当てたすべての情報を保持するには十分でない可能性があります。システム上の全体のメモリーを増やすか、割り当てたメモリー量を減らします。

**関連項目：** 27-5 ページの「[ページングを意識したオーバーサブスクリプ](#)」

## 主メモリーへのシステム・グローバル領域 (SGA) の格納

システム・グローバル領域 (SGA) の目的は、迅速なアクセスのためにメモリー内にデータを格納することなので、SGA は常に主メモリー内に存在すべきです。SGA のページがディスクにスワップされると、データを迅速にアクセスできなくなります。多くのオペレーティング・システムでは、過度のページングによる損失は、大きな SGA がもたらす利益をかなり上回ります。

SGA 全体をメモリーに保持しておくのが最も適切ですが、SGA の内容はホット・パートとコールド・パートに論理的に分割されます。ホット・パートは常に参照されるため、常にメモリー内に存在します。コールド・パートの一部はページ・アウトされる場合があります、それを戻そうとするとパフォーマンスが低下することがあります。ただし、SGA のホット・パートがメモリー内に存在できない場合も、パフォーマンスの問題が発生しやすいといえます。

データがディスクにスワップされるのは参照されていないためです。初期化パラメータ `PRE_PAGE_SGA` の値を YES に設定してインスタンスを開始すると、Oracle は SGA 全体を読み込み、メモリーに記憶することができます。そこで、オペレーティング・システムのページ・テーブル・エントリが SGA の各ページのために事前に構築されます。この設定では、インスタンスの起動に必要な時間が長くなることがありますが、起動後に Oracle のパフォーマンスが最高になるまでに必要な時間が短くなる傾向があります。

---

**注意：** この設定では、SGA が最初にメモリーに読み込まれた後では、オペレーティング・システムによる SGA のページングやスワッピングを防ぎません。

---

`PRE_PAGE_SGA` では、開始するすべてのプロセスが SGA に連結されていなければならないため、プロセスが開始するための時間が長くなることがあります。ただし、この方法のコストは固定しています。つまり、プロセスが開始するたびに 20,000 ページにアクセスしなければならないとユーザーが決定するだけです。アプリケーションによってはこのアプローチが有効でも、すべてのアプリケーションで有効なわけではありません。システムが頻繁にプロセスの作成と破壊を行う（たとえばログオンとログオフを続ける）場合、オーバーヘッドが大きくなることがあります。

`PRE_PAGE_SGA` による利点はページ・サイズによって異なります。たとえば、SGA のサイズが 80MB で、ページ・サイズが 4KB の場合、SGA をリフレッシュするためには 20,000 ページにアクセスしなければなりません ( $80,000/4 = 20,000$ )。

4MB のページ・サイズを設定できる場合は、SGA をリフレッシュするためには 20 ページしかアクセスする必要はありません ( $80,000/4,000 = 20$ )。ページ・サイズはオペレーティング・システム固有であり、通常は変更できません。ただし、オペレーティング・システムによっては、共有メモリーのための特殊な実装があり、それによりページ・サイズを変更できる場合があります。

次の SQL 文を発行することによって、SGA とその各内部構造に割り当てられるメモリー量を確認できます。

```
SHOW SGA
```

この文の出力例を以下に示します。

Total System Global Area	18847360 bytes
Fixed Size	63104 bytes
Variable Size	14155776 bytes
Database Buffers	4096000 bytes
Redo Buffers	532480 bytes

IBM メインフレーム・コンピュータ用のオペレーティング・システムの中には、主メモリーに加えて、非常に高速にページングを実行できる拡張記憶（特別なメモリー）を持つものがあります。これらのオペレーティング・システムは、Oracle が SGA とディスク間でデータの読み書きを行うよりも高速に、主メモリーと拡張記憶の間でデータのページングを行う可能性があります。このため、スワップされる SGA を大きくすると、主メモリーに常駐させる SGA を小さくするよりも、よいパフォーマンスにつながることがあります。オペレーティング・システムが拡張記憶を持っている場合、ページングは発生しますが、より大きな SGA を割り当ててすることで、その拡張記憶を有効に利用してください。

## 個々のユーザーへの十分なメモリーの割当て

いくつかのオペレーティング・システムでは、各ユーザーに割り当てられる物理的なメモリー量を制御できます。必ずすべてのユーザーに十分なメモリーを割り当て、Oracle でアプリケーションを使用するために必要となるリソースを収容できるようにしてください。

オペレーティング・システムに依存しますが、これらのリソースには以下が含まれます。

- Oracle 実行可能イメージ
- SGA
- Oracle アプリケーション・ツール
- アプリケーション固有のデータ

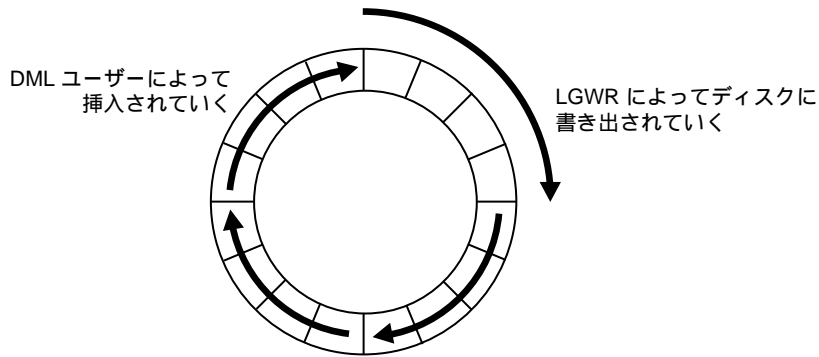
いくつかのオペレーティング・システムでは、多くのユーザーが単一の実行可能イメージを共有できるように、Oracle ソフトウェアをインストールすることができます。複数のユーザー間で実行可能イメージを共有することによって、各ユーザーに必要なメモリー量を節減することができます。

## REDO ログ・バッファのチューニング

LOG\_BUFFER パラメータによって REDO ログ・バッファの領域が予約されます。このサイズは固定です。高速のプロセッサと比較的低速のディスクを持つマシンでは、REDO ログ書き込み機能によってバッファの一部がディスクに移動される時間に、プロセッサがバッファの残りにデータを挿入していることがあります。バッファへのデータの挿入が始まると、常にログ・ライター・プロセス (LGWR) が開始します。このため、使用するバッファが大きいほど、新規入力と書き込み中のバッファの一部が衝突することは少なくなります。



図 19-1 REDO ログ・バッファ



通常、ログ・バッファは SGA の合計サイズに比べて小さいので、少し増やすだけで大幅にスループットを向上できます。重要なのは領域要求率 (REDO ログ領域要求 (redo log space requests) を REDO エントリ (redo entries) で割った値) です。この率が 1:5000 よりも大きいときは、領域要求率が下降を停止するまで REDO ログ・バッファのサイズを増やします。

## プライベート SQL と PL/SQL 領域のチューニング

この項では、プライベート SQL と PL/SQL 領域のチューニング方法について説明します。

- 不要な解析コールの識別
- 不要な解析コールの低減

メモリーと再解析の間にはトレードオフがあります。再解析を頻繁に行うと、必要なメモリーは少なくなります。SQL 文を増やすことによって再解析を低減すると、クライアント側のメモリー所要量は増加します。これは、オープン・カーソルの数が増加するためです。

プライベート SQL 領域のチューニングには、アプリケーションによって作成される不要な解析コールの識別とそれらの低減が含まれます。解析コールを低減するには、アプリケーションがただちに割り当てることのできるプライベート SQL 領域の数を増やす必要があるかもしれません。この項を通して、プライベート SQL 領域と SQL 文に関する説明は、プライベート PL/SQL 領域と PL/SQL ブロックにも適用されます。

## 不要な解析コールの識別

この項では、不要な解析コールを識別するための 3 つの手法を説明します。

### 手法 1

不要な解析コールを識別する 1 つの方法は、SQL トレース機能を使用可能にしてアプリケーションを実行することです。トレース出力の SQL 文ごとに、解析 (Parse) ステップの "count" 統計によって、その文に対してアプリケーションが行った解析コールの回数がわかります。この統計には、実際に文を解析した解析コールだけでなく、ライブラリ・キャッシュへのアクセスによって完了する解析コールも含まれます。

---

---

**注意：** この統計には暗黙の解析は含まれていません。暗黙の解析は、共有 SQL 領域がライブラリ・キャッシュ内に存在しなくなった文をアプリケーションが実行するときに発生します。暗黙の解析の検出に関する詳細は、19-13 ページの「[ライブラリ・キャッシュのアクティビティの検査](#)」を参照してください。

---

---

解析ステップの "count" 値が、文の実行ステップの "count" 値に近い場合、アプリケーションはその文を実行するたびに解析コールを行っている可能性があります。アプリケーション・ツールによって、これらの解析コールを低減するように試みてください。

### 手法 2

不要な解析コールを識別するもう 1 つの方法は、V\$SQLAREA ビューをチェックすることです。以下の問合せを入力してください。

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS
FROM V$SQLAREA
```

PARSE\_CALLS の値が指定の文の EXECUTION 値に近い場合は、その文の再解析を続けます。

### 手法 3

また、不要な解析コールが発生するセッションを識別することで、それらのコールを識別することもできます。特定のバッチ・プログラムやある種のアプリケーションがほとんどの再解析を行っている場合があります。これを行うには、次の問合せを実行します。

```
SELECT * FROM V$STATNAME
WHERE NAME IN ('PARSE_COUNT (HARD)', 'EXECUTE_COUNT')
```

Oracle によって次のような結果が返されます。

```

STATISTIC#,  NAME
-----
100          PARSE_COUNT
90           EXECUTE_COUNT
    
```

次に、以下の問合せを実行します。

```

SELECT * FROM V$SESSTAT
WHERE STATISTICS# IN (90,100)
ORDER BY VALUE, SID;
    
```

結果として、すべてのセッションとそこで行われる再解析の回数がリスト表示されます。各システム識別子 (sid) について、V\$SESSION で再解析の原因となっているプログラムの名前を検索します。

## 不要な解析コールの低減

使用している Oracle アプリケーション・ツールに応じて、アプリケーションによる解析コールの実行頻度、およびプライベート SQL 領域の割当てと割当て解除の頻度を制御することがあります。アプリケーションが複数の SQL 文のプライベート SQL 領域を再利用するかどうか、アプリケーションが実行する解析コールの数とそのアプリケーションが必要とするプライベート SQL 領域の数を決定します。

通常、複数の SQL 文のプライベート SQL 領域を再利用するアプリケーションでは、プライベート SQL 領域を再利用しないアプリケーションほど多くの SQL 領域を必要としません。ただし、プライベート SQL 領域を再利用するアプリケーションは、さらに多くの解析コールを実行する必要があります。これは、既存のプライベート SQL 領域が新しい SQL 文のために再利用されると、アプリケーションが必ず新しい解析コールを実行しなければならないためです。

アプリケーションがすべての SQL 文を収容するために十分なプライベート SQL 領域をオープンできることを確認してください。割り当てるプライベート SQL 領域を増やす場合、セッションに対して許可されるカーソル数の制限を増やす必要があるかもしれません。初期化パラメータ OPEN\_CURSORS の値を増やすことによって、この制限の値を増やすことができます。このパラメータの最大値はオペレーティング・システムによって異なります。最小値は 5 です。

解析コールを制御し、プライベート SQL 領域を割り当て、解除する方法は、Oracle アプリケーション・ツールによって異なります。以降の項では、いくつかのツールに対して使用される方法を紹介します。なお、これらの方法はプライベート SQL 領域にのみ適用され、共有 SQL 領域には適用されません。

## Oracle プリコンパイラによる解析コールの低減

Oracle プリコンパイラを使って、3 つのオプションを設定することでプライベート SQL 領域と解析コールを制御できます。Oracle モードでは、オプションとそのデフォルトは以下のとおりです。

- HOLD\_CURSOR = yes
- RELEASE\_CURSOR = no
- MAXOPENCURSORS = 希望の値

ANSI モードでは、HOLD\_CURSOR と RELEASE\_CURSOR の値が切り替えられますが、これはお薦めしません。

プリコンパイラのオプションは以下の 2 つの方法で指定できます。

- プリコンパイラ・コマンド行
- プリコンパイラ・プログラム

これらのオプションによって、プログラムの進行中にプライベート SQL 領域を管理するための異なる方法を採用できます。

**関連項目：** これらのコールの詳細は、『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。

### Oracle Forms による解析コールの低減

Oracle Forms を使用すると、アプリケーションがプライベート SQL 領域を再利用するかどうかを多少制御できます。この制御は次の 3 つのレベルで実行することができます。

- トリガー・レベル
- フォーム・レベル
- 実行時

**関連項目：** Oracle Forms によるプライベート SQL 領域の再利用の詳細は、『Oracle Forms Reference』マニュアルを参照してください。

## 共有プールのチューニング

この項は、重要なメモリ構造体である共有プールへのメモリの割当て方法について説明します。構造体はチューニングの重要性の順に示しています。

- [ライブラリ・キャッシュのチューニング](#)
- [データ・ディクショナリ・キャッシュのチューニング](#)
- [マルチスレッド・サーバー・アーキテクチャでの大規模プールと共有プールのチューニング](#)
- [共有プールの予約領域のチューニング](#)

---

**注意：** 予約済みサイズの共有プールを使用している場合は、19-24 ページの「**SHARED\_POOL\_SIZE が小さすぎる場合**」を参照してください。

---

Oracle が共有プール内のデータの管理に使用するアルゴリズムでは、ライブラリ・キャッシュ・データよりも長時間、ディクショナリ・キャッシュ・データをメモリーに保持する傾向があります。このため、ライブラリ・キャッシュを許容できるキャッシュ・ヒット率にチューニングすると、多くの場合、データ・ディクショナリ・キャッシュ・ヒット率も許容可能になります。セッション情報に対する共有プール内の領域の割当ては、MTS (マルチスレッド・サーバー) アーキテクチャを使用している場合にのみ必要です。

共有プールではキャッシュの一部が動的です。つまり、必要に応じてサイズが増大または縮小します。これらの動的キャッシュには、ライブラリ・キャッシュおよびデータ・ディクショナリ・キャッシュが含まれます。共有プールの空きがなくなると、これらのキャッシュでは古いオブジェクトから順になります。このために、頻繁に使用されるデータのセットが共有プールに入りきらない場合は、サイズを増やす必要があるでしょう。データ・ディクショナリ・キャッシュまたはライブラリ・キャッシュでのキャッシュ・ミスは、バッファ・キャッシュでのミスよりも影響が大きくなります。このため、共有プールに十分なメモリーを割り当ててから、バッファ・キャッシュへの割当てを行ってください。

ほとんどのアプリケーションでは、共有プールのサイズが Oracle のパフォーマンスにとって重要です (ごく少数の不連続な SQL 文しか発行しないアプリケーションの場合に限り、共有プールのサイズはそれほど重要ではありません)。共有プールには、データ・ディクショナリ・キャッシュと、PL/SQL ブロックおよび SQL 文の完全に解析済みまたはコンパイル済みの表現が、両方とも保持されます。PL/SQL ブロックには、プロシージャ、ファンクション、パッケージ、トリガーおよびクライアントのプログラムから送られた任意の無名 PL/SQL ブロックが含まれます。

共有プールが小さすぎる場合は、限られた容量の使用可能領域の管理のためだけにサーバーがリソースを使用する必要があります。このため、Oracle によってさまざまなキャッシュの並列管理において制限が課されるので、CPU リソースが消費され競合が発生します。トリガーやストアド・プロシージャを多く使用するほど、共有プールを大きくしなければなりません。数百 MB というサイズになることさえあります。

開始からの統計ではなく限定期間の統計を測定する方が望ましいため、次の問合せを使用してライブラリ・キャッシュおよび行キャッシュ (データ・ディクショナリ・キャッシュ) のヒット率を判別できます。結果として、ライブラリ・キャッシュおよび行キャッシュのミス率が表示されます。一般に、再解析の回数はライブラリ・キャッシュを反映します。率が 1 に近ければ、プールのサイズを増やす必要はありません。

```
SELECT (SUM(PINS - RELOADS)) / SUM(PINS) "LIB CACHE"
FROM V$LIBRARYCACHE;
```

```
SELECT (SUM(GETS - GETMISSES - USAGE - FIXED)) / SUM(GETS) "ROW CACHE"
FROM V$ROWCACHE;
```

共有プールの空きメモリーの容量は、V\$SGASTAT でレポートされます。次の問合せを使用してこのビューの現在の値についてレポートを作成します。

```
SELECT * FROM V$SGASTAT WHERE NAME = 'FREE MEMORY';
```

共有プール内に使用可能な空きメモリーが常にある場合、プールのサイズを増やしても、効果はほとんど（またはまったく）ありません。ただし、共有プールが満杯というだけでは、問題があるとはいえません。

エントリがいったん共有プールにロードされると、それを移動することはできません。エントリが多数ロードされるにつれて、空きメモリーが分断されて共有プールが断片化する場合があります。

**dbmspool.sql** にある PL/SQL パッケージ **DBMS\_SHARED\_POOL** を使用して、共有プールを管理できます。コード内のコメントによって、パッケージ内のプロシージャの使用方法が説明されています。

**関連項目：** **DBMS\_SHARED\_POOL** の詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。

### 共有プールへの PL/SQL オブジェクトのロード

Oracle では、サイズ 4KB の " ページ " を使用して共有プールにオブジェクトをロードします。このようなページによって、セグメント化された PL/SQL コードのまとまりがロードされます。ページは連続している必要はありません。このため、Oracle では、オブジェクトの共有プールへのロードに対して、連続したメモリーからなる大きなセクションを割り当てる必要はありません。これによって、連続したメモリーへの必要性が低減し、パフォーマンスが向上します。ただし、Oracle では、パッケージの一部がコールされた場合でもそのパッケージ全体をロードします。

ユーザーのニーズによって異なりますが、共有プールにパッケージを確保しておくのが賢明な場合とそうでない場合があります。それでもなお、Oracle では、頻繁に使用されるアプリケーション・オブジェクトの場合はとりわけ確保することをお勧めします。

### ライブラリ・キャッシュと行キャッシュのヒット率

ライブラリ・キャッシュと行キャッシュのヒット率は重要です。空きメモリーがゼロに近い場合、ライブラリ・キャッシュ・ヒット率または行キャッシュ・ヒット率のいずれかが 0.95 未満の場合は、率が向上しなくなるまで共有プールを増やします。

## ライブラリ・キャッシュのチューニング

この項では、ライブラリ・キャッシュのヒット率のチューニング方法を説明します。

ライブラリ・キャッシュは、SQL カーソル、PL/SQL プログラムおよび JAVA クラスの実行可能な形式を保持します。また、スキーマ・オブジェクトを説明する情報すなわちメタデータもキャッシュされます。メタデータが使用されるのは、SQL カーソルの解析時か PL/SQL プログラムのコンパイル時です。後者のメモリーはパフォーマンスにほとんど関連しないの

で、この項では、カーソル、PL/SQL プログラムおよび JAVA クラスに関連するチューニングについて説明します。これらをまとめて "アプリケーション・ロジック" と呼びます。

## ライブラリ・キャッシュのアクティビティの検査

ライブラリ・キャッシュ・ミスは、SQL 文の処理の解析ステップまたは実行ステップのいずれかで発生します。

**解析** アプリケーションが SQL 文に対して解析コールを作成し、その文の解析された表現がライブラリ・キャッシュ内の共有 SQL 領域に存在していない場合、Oracle はその文を解析し、共有 SQL 領域を割り当てます。解析コールのライブラリ・キャッシュ・ミスは、可能なときはいつでも SQL 文が共有 SQL 領域を共有できるようにすることで低減できる場合があります。

**実行** アプリケーションが SQL 文に対して実行コールを作成し、その文の解析された表現を含んでいる共有 SQL 領域が、別の SQL 文のための場所を作るためにライブラリ・キャッシュから割当て解除された場合、Oracle はその文を暗黙に再解析し、新しい共有 SQL 領域を割り当て、実行します。実行コールのライブラリ・キャッシュ・ミスは、ライブラリ・キャッシュに割り当てるメモリーを増やすことによって低減できることがあります。

ライブラリ・キャッシュ・ミスが Oracle のパフォーマンスに影響を及ぼしているかどうかを判断するには、動的パフォーマンス表 V\$LIBRARYCACHE を問い合わせます。

**V\$LIBRARYCACHE ビュー** 動的パフォーマンス・ビュー V\$LIBRARYCACHE を調べることで、ライブラリ・キャッシュのアクティビティを反映する統計を監視できます。これらの統計は、最も最近のインスタンス起動以降のライブラリ・キャッシュのアクティビティを反映しています。デフォルトでは、ユーザー SYS、および SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。

このビューの各行には、ライブラリ・キャッシュ内に保持される項目の 1 つに対応する統計が収録されます。各行ごとに記述される項目は、NAMESPACE 列の値によって識別されます。次の NAMESPACE 値を持つ表の行は、SQL 文と PL/SQL ブロックのライブラリ・キャッシュのアクティビティを反映します。

- SQL AREA
- TABLE/PROCEDURE
- BODY
- TRIGGER

他の NAMESPACE 値を持つ行は、Oracle が依存関係のメンテナンスのために使用するオブジェクト定義に対するライブラリ・キャッシュのアクティビティを反映します。

以下の V\$LIBRARYCACHE 表の列は、実行コールのライブラリ・キャッシュ・ミスを反映します。

PINS	この列はライブラリ・キャッシュ内の項目が実行された回数を示します。
RELOADS	この列は実行ステップのライブラリ・キャッシュ・ミスの数を示します。

**V\$LIBRARYCACHE 表の問合せ** 次の問合せによって、V\$LIBRARYCACHE 表の統計をある期間にわたって監視してください。

```
SELECT SUM(PINS) "EXECUTIONS",
       SUM(RELOADS) "CACHE MISSES WHILE EXECUTING"
FROM V$LIBRARYCACHE;
```

次に、この問合せの出力例を示します。

```
EXECUTIONS CACHE MISSES WHILE EXECUTING
-----
320871                                549
```

**V\$LIBRARYCACHE 表の解釈** サンプル問合せが戻したデータを調べると、以下のようなことがわかります。

- EXECUTIONS 列の合計は、SQL 文、PL/SQL ブロックおよびオブジェクト定義が、実行のために合計 320,871 回アクセスされたことを示しています。
- CACHE MISSES WHILE EXECUTING 列の合計は、それらの実行のうち 549 回がライブラリ・キャッシュ・ミスに終わり、それが原因で、Oracle が文またはブロックを暗黙に再解析した、またはオブジェクト定義がライブラリ・キャッシュから除去されたためにそれを再ロードしたことを示しています。
- 実行の合計に対するミスの合計の比率は、およそ 0.17% である。この値は実行の 0.17% だけが再解析されたことを意味しています。

ミスの合計が 0 に近くなるようにしてください。実行に対するミスの比率が 1% を超える場合、次の項で説明する方法で、これらのライブラリ・キャッシュ・ミスを低減してください。

**ライブラリ・キャッシュ・ミスの低減**

次のようにして、ライブラリ・キャッシュ・ミスを低減できます。

- ライブラリ・キャッシュに追加のメモリーを割り当てる。
- 可能なときはいつでも同一の SQL 文を書く。

**ライブラリ・キャッシュへの追加のメモリー割当て** ライブラリ・キャッシュに追加のメモリーを割り当てることによって、実行コールのライブラリ・キャッシュ・ミスを低減できる可能性があります。共有 SQL 領域が SQL 文を一度解析したキャッシュ内に確実に残るようにするには、V\$LIBRARYCACHE.RELOADS の値が 0 の近傍になるまで、ライブラリ・



キャッシュに利用できるメモリーの容量を増やしてください。ライブラリ・キャッシュに利用できるメモリーの容量を増やすには、初期化パラメータ `SHARED_POOL_SIZE` の値を増やしてください。このパラメータの最大値はオペレーティング・システムによって異なります。この処置によって、実行のための SQL 文と PL/SQL ブロックの暗黙の再解析が減少します。

共有 SQL 領域に利用可能な追加のメモリーを利用するために、セッションに対して許可されるカーソル数を増やす必要があるかもしれません。初期化パラメータ `OPEN_CURSORS` の値を増やすことによってこれを行います。

ライブラリ・キャッシュに過剰なメモリーを割り当てることによって、ページングやスワッピングが発生しないように注意してください。ライブラリ・キャッシュ・ミス避けるためにライブラリ・キャッシュを十分大きくしたことで得られる利益が、共有 SQL 領域をアクセスするたびにディスクからメモリーに読み込むことで相殺される可能性があります。

#### 関連項目： 19-24 ページの「[SHARED\\_POOL\\_SIZE が小さすぎる場合](#)」

**同一の SQL 文の記述: 基準** SQL 文と PL/SQL ブロックが可能なのは常に共有 SQL 領域を使用することによって、解析コールのライブラリ・キャッシュ・ミスを低減できる可能性があります。別々に発行された 2 つの SQL 文もしくは PL/SQL ブロックが次の基準に従って同一である場合、それらは共有 SQL 領域を使用できます。

- SQL 文や PL/SQL ブロックのテキストは、空白や大文字小文字の区別も含め、一字一句、同一でなければなりません。たとえば、次の 2 つの文は同じ共有 SQL 領域を使用できません。

```
SELECT * FROM emp;
SELECT *   FROM emp;
```

次の 2 つの文も同じ共有 SQL 領域を使用できません。

```
SELECT * FROM emp;
SELECT * FROM EMP;
```

- SQL 文や PL/SQL ブロック内でスキーマ・オブジェクトを参照するさいには、同じスキーマ内の同じオブジェクトでなければなりません。

たとえば、ユーザー BOB と ED のスキーマの両方に EMP 表が存在し、両方のユーザーが次の文を発行する場合、それらの文は同じ共有 SQL 領域を使用できません。

```
SELECT * FROM EMP;
SELECT * FROM EMP;
```

2 つの文が同じ表を問い合わせ、次の文のようにスキーマで表を修飾する場合、それらの文は同じ共有 SQL 領域を使用できます。

```
SELECT * FROM BOB.EMP;
```

- SQL 文の中のバインド変数は、名前とデータ型で一致していなければなりません。たとえば、次の 2 つの文は同じ共有 SQL 領域を使用できません。

```
SELECT * FROM EMP WHERE DEPTNO = :DEPARTMENT_NO;  
SELECT * FROM EMP WHERE DEPTNO = :D_NO;
```

- 同じ最適化アプローチを使用して、SQL 文を最適化しなければなりません。コストベースのアプローチの場合は同じ最適化の目標を使用しなければなりません。最適化のアプローチと目標の詳細は、7-3 ページの「[コストベース・アプローチの目標の選択](#)」を参照してください。

**同一の SQL 文の記述: 計画** 共有 SQL 領域は、同じアプリケーションを実行する複数のユーザーのライブラリ・キャッシュ・ミスを低減するために最も有効です。そのようなアプリケーションの開発者と以下の基準について検討し、アプリケーションの SQL 文と PL/SQL ブロックが同じ共有 SQL 領域を使用できることを保証するための計画を話し合いの上で決定してください。

- 文の中では、可能なときはいつでもバインド変数を使用し、明示的に指定した定数は使用しないようにします。

たとえば、次の 2 つの文は字面が完全には一致しないので、同じ共有領域は使用できません。

```
SELECT ENAME, EMPNO FROM EMP WHERE DEPTNO = 10;  
SELECT ENAME, EMPNO FROM EMP WHERE DEPTNO = 20;
```

バインド変数を含む次の文を使用し、文の一方には 10、他方に 20 をバインドすることによって、上の 2 つの文の目標を達成できます。

```
SELECT ENAME, EMPNO FROM EMP WHERE DEPTNO = :DEPARTMENT_NO;
```

この文を 2 つ発行した場合には、同じ共有 SQL 領域を使用できます。

- アプリケーションのユーザーが最適化アプローチと目標を各セッションに対して変更しないことを確認してください。
- また、アプリケーション開発者の間で以下に示す方針を設定することにより、異なるアプリケーションによって発行される SQL 文が SQL 領域を共有できる可能性を高めることもできます。
  - SQL 文と PL/SQL ブロックに対して、バインド変数の命名規定とスペーシング規定を標準化します。
  - 可能なときはいつでもストアド・プロシージャを使用します。そうすれば、同じストアド・プロシージャを発行している複数のユーザーが、同じ共有 PL/SQL 領域を自動的に使用します。ストアド・プロシージャは解析済みの形式で格納されるため、実行時の解析はまったくなくなります。

## 共有 SQL 領域への高速アクセスのための CURSOR\_SPACE\_FOR\_TIME の使用

ライブラリ・キャッシュ・ミスがない場合も、初期化パラメータ

CURSOR\_SPACE\_FOR\_TIME の値を設定することによって実行コールを高速化できる可能性があります。このパラメータは、新しい SQL 文の領域を作成するために、ライブラリ・キャッシュから共有 SQL 領域の割当てを解除するかどうかを指定します。

CURSOR\_SPACE\_FOR\_TIME の値には次の意味があります。

- このパラメータが FALSE に設定されていると（デフォルト）、SQL 文に対応付けられているアプリケーション・カーソルがオープンされているかどうかにかかわらず、ライブラリ・キャッシュから共有 SQL 領域の割当てを解除できます。この場合、Oracle では、SQL 文を含む共有 SQL 領域がライブラリ・キャッシュ内にあることを検証する必要があります。
- このパラメータが TRUE に設定されていると、共有 SQL 領域を割当て解除できるのは、その文に対応付けられているすべてのアプリケーション・カーソルがクローズされているときだけです。この場合、共有 SQL 領域がキャッシュ内にあることを Oracle で検証する必要はありません。共有 SQL 領域に関連するアプリケーション・カーソルがオープンしている間は、その共有 SQL 領域を割当て解除できないからです。

パラメータの値を TRUE に設定することで、Oracle 側の時間が少し短縮されるので、わずかながら実行コールのパフォーマンスが改善する可能性があります。また、対応付けられたアプリケーション・カーソルがクローズされるまで、プライベート SQL 領域の割当て解除も防止されます。

実行コールでライブラリ・キャッシュ・ミスがあった場合は、CURSOR\_SPACE\_FOR\_TIME の値を TRUE に設定しないでください。そのようなライブラリ・キャッシュ・ミスは、共有プールが十分大きくないので同時にオープンしている全カーソルの共有 SQL 領域を保持できないことを示しています。値が TRUE であり、共有プール内に新しい SQL 文のための領域がない場合、文は解析されず、共有メモリーがなくなったことを示すエラーが Oracle によって戻されます。値が FALSE であり、そして新しい文のための領域がない場合には、Oracle が既存の共有 SQL 領域の割当てを解除します。共有 SQL 領域の割当てを解除するとライブラリ・キャッシュ・ミスが後で発生しますが、SQL 文が解析できるため、アプリケーションを停止させるエラーよりも望ましい対処と言えます。

各ユーザーに利用できるプライベート SQL 領域のメモリー量が不十分な場合、CURSOR\_SPACE\_FOR\_TIME の値を TRUE に設定しないでください。また、この値は、オープンしているカーソルに対応付けられているプライベート SQL 領域の割当て解除も防ぎます。ユーザーの利用可能なメモリーが同時にオープンしている全カーソルのプライベート SQL 領域で満杯になっていて、新しい SQL 文のプライベート SQL 領域を割り当てる領域がない場合、その文は解析できません。そして Oracle は十分なメモリーがないことを伝えるエラーを戻します。

## セッション・カーソルのキャッシュ

アプリケーションから何度も同じ SQL 文に解析コールが発行されると、セッション・カーソルの再オープンにより、システム・パフォーマンスに影響が出ることがあります。セッ

セッション・カーソルは、セッション・カーソル・キャッシュに保存できます。フォーム間で切替えを行うと、最初のフォームに関連するすべてのセッション・カーソルがクローズされるため、この機能は、Oracle Forms を使用して設計されたアプリケーションで特に有用です。

Oracle では、共有 SQL 領域を使って、指定の文で 3 つ以上の解析要求が発行されたかどうかを識別します。発行された場合、Oracle では、文に関連するセッション・カーソルをキャッシュしなければならないことを想定し、カーソルをセッション・カーソル・キャッシュに移動します。同じセッションでその SQL 文の解析要求が続けて出されると、セッション・カーソル・キャッシュ内のカーソルが検索されます。

セッション・カーソルのキャッシュを使用可能にするには、初期化パラメータ `SESSION_CACHED_CURSORS` を設定する必要があります。このパラメータの値は、キャッシュに保持されるセッション・カーソルの最大数を指定する正の整数です。LRU（最低使用頻度）のアルゴリズムでは、セッション・カーソル・キャッシュ内の項目を除去し、必要に応じて新しい項目のための空間を作ります。

また、`ALTER SESSION SET SESSION_CACHED_CURSORS` 文を使うと、セッション・カーソル・キャッシュを動的に使用可能にすることもできます。

セッション・カーソル・キャッシュがインスタンスに対して十分な大きさであるかどうかを判断するには、`V$SESSTAT` ビュー内のセッション統計 "session cursor cache hits" を調べます。この統計では、解析コールによってセッション・カーソル・キャッシュ内でカーソルが検出された回数を数えます。この統計で、セッションの合計解析コール数が相対的に低い割合である場合には、`SESSION_CACHED_CURSORS` を大きい値に設定してください。

## データ・ディクショナリ・キャッシュのチューニング

この項では、データ・ディクショナリ・キャッシュのアクティビティを監視する方法と、ミスを低減する方法について説明します。

### データ・ディクショナリ・キャッシュのアクティビティの監視

データ・ディクショナリ・キャッシュのミスが、Oracle のパフォーマンスに影響を及ぼしているかどうか判断してください。次の項で説明されるように、`V$ROWCACHE` 表を問い合わせることによってキャッシュ・アクティビティを調べることができます。

データ・ディクショナリ・キャッシュ・ミスは、いくつかの場合に予想されます。インスタンスの起動時に、データ・ディクショナリ・キャッシュはデータを含んでいないため、発行されるどの SQL 文もキャッシュ・ミスになるでしょう。キャッシュに読み込まれるデータが増え、キャッシュ・ミスの可能性は減少します。最終的に、データベースは、最も頻繁に使用されるディクショナリ・データがキャッシュ内に存在する "安定状態" に到達するでしょう。この時点で、キャッシュ・ミスはほとんど発生しないはずで、キャッシュをチューニングするには、必ずアプリケーションの実行後にキャッシュのアクティビティを調べてください。

**V\$ROWCACHE ビュー** データ・ディクショナリのアクティビティを反映する統計は、動的パフォーマンス表 `V$ROWCACHE` に保持されます。デフォルトでは、ユーザー `SYS`、および

SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこの表を利用できます。

この表の各行は、データ・ディクショナリ項目について単一のタイプの統計を収録します。これらの統計は、直前のインスタンス起動以降のデータ・ディクショナリ・アクティビティを反映しています。VSROWCACHE 表の中の以下に示す列は、データ・ディクショナリ・キャッシュの使用と有効性を反映します。

PARAMETER	特定のデータ・ディクショナリ項目を識別します。各行で、この列の値は接頭辞 dc_ が付いた項目です。たとえば、ファイル記述の統計を含む行では、この列の値は dc_files です。
GETS	対応する項目に関する情報への要求の総数を示します。たとえば、ファイル記述の統計を含む行では、この列はファイル記述データへの要求の総数を持ちます。
GETMISSES	キャッシュ・ミスとなったデータ要求の数を示します。

**VSROWCACHE 表の問合せ** 次の問合せによって、アプリケーションの実行中、ある期間にわたって VSROWCACHE 表の統計を監視してください。

```
SELECT SUM(GETS)   "DATA DICTIONARY GETS",
       SUM(GETMISSES) "DATA DICTIONARY CACHE GET MISSES"
FROM VSROWCACHE;
```

次に、この問合せの出力例を示します。

```
DATA DICTIONARY GETS   DATA DICTIONARY CACHE GET MISSES
-----
1439044                3120
```

**VSROWCACHE 表の解釈** サンプル問合せが戻したデータを調べると、以下のようなことがわかります。

- GETS 列の合計は、ディクショナリ・データへの要求が全体で 1,439,044 あったことを示しています。
- GETMISSES 列の合計は、ディクショナリ・データへの要求の 3,120 がキャッシュ・ミスに終わったことを示しています。
- GETS と GETMISSES の合計の比率はおよそ 0.2% となります。

## データ・ディクショナリ・キャッシュ・ミスの低減

GETS と GETMISSES 列の合計を監視することによって、キャッシュ・アクティビティを調べてください。ディクショナリ・キャッシュが頻繁にアクセスされる場合、GETS の合計に対する GETMISSES の合計の割合は、10% あるいは 15% より低くしてください。アプリケーションの実行中に割合がこのしきい値を超えて増え続ける場合、データ・ディクショナリ・キャッシュに利用できるメモリーの容量を増やすことを検討すべきです。キャッシュに

利用できるメモリーを増やすには、初期化パラメータ `SHARED_POOL_SIZE` の値を増やしてください。このパラメータの最大値はオペレーティング・システムによって異なります。

## マルチスレッド・サーバー・アーキテクチャでの大規模プールと共有プールのチューニング

MTS セッションのメモリーは大規模プールから割り当てることをお勧めします。このためには、パラメータ `LARGE_POOL_SIZE` に値を指定します。指定しないと、MTS のメモリーは共有プールから割り当てられます。どちらの場合も、MTS を使用するときは、大規模プールまたは共有プールのサイズを増やして、セッション情報を収容できるようにする必要がありますでしょう。

---

**注意：** Oracle のデフォルトは専用サーバー処理です。

---

ユーザーが MTS を介して接続するとき、ユーザー・プロセス、ディスパッチャおよびサーバーの間の接続に関する情報を格納するために SGA 内の領域が割り当てられます。各ユーザーのプライベート SQL 領域に関するセッション情報も格納されます。

大規模プールまたは共有プールで各 MTS ユーザー接続のために必要となる領域の容量は、アプリケーションによって異なります。

**関連項目：** 第 23 章「マルチスレッド・サーバー・アーキテクチャのチューニング」、『Oracle8i 概要』、『Oracle8i 管理者ガイド』、『Oracle8i SQL リファレンス』、『Oracle8i Net8 管理者ガイド』を参照してください。

### 3 層の接続でのメモリー使用の低減

接続ユーザーが非常に多数の場合は、"3 層の接続" をインプリメントすることでメモリー使用を受容可能なレベルに低減できます。これは TP モニター使用の副産物であり、ロックやコミットされていない DML を複数のコールにわたって保持できないため、純粋なトランザクション・モデルでしか実現できません。MTS は、TP モニターに比べてアプリケーション設計の制限が少なくなります。ユーザーがサーバーのプールを共有できるので、オペレーティング・システム・プロセスのカウンとコンテキストの切替えが大幅に減ります。また、MTS では、MTS モードでより大きな SGA が使用されるとしても、全体のメモリー使用量はかなり低減されます。

---

**注意：** NT では、共有サーバーはプロセスではなく "スレッド" としてインプリメントされます。

---

## V\$SESSTAT ビュー

Oracle はセッションによって使用された全体のメモリーの統計を収集し、動的パフォーマンス・ビュー V\$SESSTAT に格納します。デフォルトでは、ユーザー SYS、および SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。以下の統計は、セッション・メモリーの使用を測定する上で役に立ちます。

session UGA memory	この統計の値は、セッションに割り当てられたメモリー容量です（単位はバイト）。
session UGA memory max	この統計の値は、セッションに割り当てたメモリー容量の最大値です（単位はバイト）。

値を検索するには、19-8 ページの「[手法 3](#)」で説明している V\$STATNAME を問い合せてください。

## V\$SESSTAT ビューの問合せ

マルチスレッド・サーバーを使用している場合、この問合せを使用して、どの程度共有プールを大きくするか判断できます。アプリケーションの実行中に、次の問合せを発行してください。

```
SELECT SUM(VALUE) || ' BYTES' "TOTAL MEMORY FOR ALL SESSIONS"
FROM V$SESSTAT, V$STATNAME
WHERE NAME = 'SESSION UGA MEMORY'
      AND V$SESSTAT.STATISTIC# = V$STATNAME.STATISTIC#;
SELECT SUM(VALUE) || ' BYTES' "TOTAL MAX MEM FOR ALL SESSIONS"
FROM V$SESSTAT, V$STATNAME
WHERE NAME = 'SESSION UGA MEMORY MAX'
      AND V$SESSTAT.STATISTIC# = V$STATNAME.STATISTIC#;
```

また、これらの問合せでは、動的パフォーマンス表 V\$STATNAME から選択して、セッション・メモリーと最大セッション・メモリーの内部識別子を取得します。以下にこれらの問合せの結果例を示します。

```
TOTAL MEMORY FOR ALL SESSIONS
-----
157125 BYTES

TOTAL MAX MEM FOR ALL SESSIONS
-----
417381 BYTES
```

## V\$SESSTAT ビューの解釈

最初の問合せの結果は、現在、全セッションに割り当てられているメモリーは 157,125 バイトであることを示しています。この値は、セッションが Oracle に接続されている方法にその位置が依存するメモリーの全体の容量です。セッションが専用サーバー・プロセスで接続されている場合、このメモリーはユーザー・プロセスのメモリーの一部です。セッションが共有サーバー・プロセスで接続されている場合、このメモリーは共有プールの一部です。

2 番目の問合せの結果は、全セッションのメモリーの最大サイズの合計が 417,381 バイトであることを示しています。2 番目の結果は、いくつかのセッションが最大の容量を割り当てた後でメモリーを割当て解除したため、最初の結果よりも大きくなっています。

マルチスレッド・サーバーを使用している場合、これらの問合せの結果を使用してどの程度共有プールを大きくするか判断できます。全セッションがほとんど同時にそれらの最大割当てに到達しそうでない限り、2 番目の値よりも最初の値の方がよい見積りになります。

## 共有プールの予約領域のチューニング

ビジー・システムでは、大容量のメモリー要求を満たすために、データベースが連続したメモリーを探しだすのは困難です。この検索のために共有プールの動作が混乱するので、断片化を引き起こし、パフォーマンスに影響を与えることになるかもしれません。

DBA は共有プール内のメモリーを予約しておき、PL/SQL のコンパイルやトリガーのコンパイルなどの操作中に大容量の割当てを満たすことができます。より小さいオブジェクトによって予約リストの断片化が生じることがないため、予約リストが大きな連続メモリーを保持するのに役立ちます。予約リストから割り当てられたメモリーは、いったん解放されると、予約リストに戻ります。

## 予約リストのチューニング・パラメータ

予約リストのサイズは、予約リストから割り当てられるオブジェクトの最小サイズと同様に、次の初期化パラメータによって制御されます。

SHARED\_POOL\_RESERVED\_  
SIZE

大容量の割当てのために確保されている SHARED\_POOL\_SIZE の容量を制御します。固定ビュー V\$SHARED\_POOL\_RESERVED を使うと、これらのパラメータをチューニングするのに役立ちます。共有プールの他のすべてのチューニングを実行した後で、このチューニングを開始してください。

## 共有プールのスペース再利用の制御

パッケージ DBMS\_SHARED\_POOL の ABORTED\_REQUEST\_THRESHOLD プロシージャを使用すると、空きリストが要求サイズを満たさない場合に共有プールのフラッシュに使用できる割当てのサイズをユーザーが制限できます。データベースは、割当て要求を満たすの



に十分なメモリーができるまで、未使用のオブジェクトを増分的に共有プールからフラッシュします。ほとんどの場合、これによって、割当てが正常に終了するための十分なメモリーが解放されます。データベースが現在使用されていないすべてのオブジェクトをフラッシュしても、十分な大きさのある連続したメモリーが検出されない場合は、エラーが発生します。ただし、すべてのオブジェクトをフラッシュすると、システム・パフォーマンスと同様にシステムの他のユーザーにも影響します。ABORTED\_REQUEST\_THRESHOLD プロシージャを使用すると、メモリーを割り当てられなかったプロセスにエラーをとどめることができます。

## 初期化パラメータの値

SHARED\_POOL\_RESERVED\_SIZE の初期値は SHARED\_POOL\_SIZE の 10% に設定します。すでに共有プールのチューニングを行っている場合、ほとんどのシステムではこの値で十分です。この値を増やすと、データベースで予約リストからの割当てが少なくなり、共有プール・リストのメモリーの要求が増えます。

理想的には、共有プールからオブジェクトをフラッシュせずに、予約リストのメモリーの要求をすべて満たすのに十分な大きさに SHARED\_POOL\_RESERVED\_SIZE を設定します。ただし、オペレーティング・システムのメモリー容量が SGA のサイズを制約する場合があります。そのため、すべてのメモリー要求を満たすほど SHARED\_POOL\_RESERVED\_SIZE を大きくするというのは、実現可能な目標ではありません。

V\$SHARED\_POOL\_RESERVED ビューの統計を使うと、これらのパラメータをチューニングするのに役立ちます。豊富な空きメモリーがあるシステムで SGA のサイズを増やすには、目標は REQUEST\_MISSES = 0 です。システムがオペレーティング・システム・メモリーの制約を受ける場合は、目標は REQUEST\_FAILURES をなくすこと、または少なくともこの値が増えないようにすることです。

これが達成できない場合は、SHARED\_POOL\_RESERVED\_SIZE の値を増やしてください。また、予約リストは共有プールからとられるため、SHARED\_POOL\_SIZE の値も同じだけ増やします。

**関連項目：** LARGE\_POOL\_SIZE パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## SHARED\_POOL\_RESERVED\_SIZE が小さすぎる場合

REQUEST\_FAILURES の値がゼロよりも大きく、増加している場合は、予約プールが小さすぎます。SHARED\_POOL\_RESERVED\_SIZE と SHARED\_POOL\_SIZE の値をそれぞれ増やすと、これを解決できます。ここで選択する設定は、システムの SGA サイズの制約によって異なります。

このオプションでは、予約リストで利用可能なメモリーの容量が増えます。予約リストからメモリーを割り当てないユーザーには影響がありません。2 番目のオプションとしては、予約リストからメモリーを割り当てするのに利用できる割当て数を減らしてください。ただし、

この場合、標準の共有プールのサイズが増加し、システムの他のユーザーが影響を受けることがあります。

### SHARED\_POOL\_RESERVED\_SIZE が大きすぎる場合

予約リストに割り当てられているメモリーが多すぎる場合があります。次の場合です。

- REQUEST\_MISS = 0 (または増加しない)
- FREE\_MEMORY => SHARED\_POOL\_RESERVED\_SIZE の最小値の 50%

これらのどちらかが真の場合、SHARED\_POOL\_RESERVED\_SIZE の値を減らします。

### SHARED\_POOL\_SIZE が小さすぎる場合

V\$SHARED\_POOL\_RESERVED 固定表を使用すると、SHARED\_POOL\_SIZE の値が小さすぎる場合を示すこともできます。以下のような場合です。

- REQUEST\_FAILURES > 0 (および増加する)

予約リストを使用可能にしている場合は、SHARED\_POOL\_RESERVED\_SIZE の値を減らします。予約リストを使用可能にしていない場合は、SHARED\_POOL\_SIZE を増やします。

## バッファ・キャッシュのチューニング

Oracle バッファ・キャッシュは、特定の操作に対して使用またはバイパスできます。ソートやパラレル読み込みの場合には、Oracle ではバッファ・キャッシュはバイパスされます。バッファ・キャッシュを使用する操作について、この項では以下の項目を説明します。

- [キャッシュ・ヒット率によるバッファ・キャッシュのアクティビティの評価](#)
- [バッファ・キャッシュ・ミスの低減によるキャッシュ・ヒット率の増加](#)
- [キャッシュ・ヒット率が高い場合の不必要なバッファの削除](#)

プライベート SQL と PL/SQL 領域、および共有プールをチューニングした後、残りの利用可能なメモリーをバッファ・キャッシュに当てることができます。プロセスを一とおり実行した後で、メモリー割当てのステップを繰り返すことが必要となる可能性もあります。実行を繰り返すことによって、後のステップの変更に基づいて前のステップの調整が可能となります。たとえば、バッファ・キャッシュのサイズを増やすと、ページングやスワッピングを回避するために、より多くのメモリーを Oracle に割り当てる必要があるかもしれません。

### キャッシュ・ヒット率によるバッファ・キャッシュのアクティビティの評価

物理 I/O にはかなり時間がかかります (通常は 15 ミリ秒を上回ります)。また、物理 I/O では、デバイス・ドライバやオペレーティング・システムのイベント・スケジューラのパス

長のために必要な CPU リソースも増加します。必要なブロックがメモリー内に常駐するようにして、このオーバーヘッドをできる限り低減することが目標になります。目標達成の度合いは、キャッシュ・ヒット率を使って測定されます。Oracle では、この用語は特にデータベース・バッファ・キャッシュについて使用します。

### キャッシュ・ヒット率の計算

Oracle はデータ・アクセスを反映する統計を収集し、動的パフォーマンス・ビュー V\$SYSSTAT に格納します。特に何も指定しない場合、ユーザー SYS、および SELECT ANY TABLE システム権限を持つユーザー（SYSTEM など）だけがこの表を使用できます。また、V\$SYSSTAT ビューの情報はシンプル・ネットワーク管理プロトコル（SNMP）を使用しても取得できます。

以下の統計は、バッファ・キャッシュをチューニングする上で役に立ちます。

DB block gets、consistent gets	これらの統計の値の合計はデータ要求の総数です。この値は、メモリー内のバッファへのアクセスによって満たされる要求を含んでいます。
physical reads	この統計は、ディスク上のデータ・ファイルにアクセスしたデータ要求の総数。

アプリケーションの実行中、ある期間にわたってこれらの統計を以下のように監視してください。

```
SELECT NAME, VALUE
FROM V$SYSSTAT
WHERE NAME IN ('DB BLOCK GETS', 'CONSISTENT GETS',
               'PHYSICAL READS');
```

次に、この問合せの出力例を示します。

NAME	VALUE
-----	-----
DB BLOCK GETS	85792
CONSISTENT GETS	278888
PHYSICAL READS	23182

次の公式でバッファ・キャッシュのヒット率を計算してください。

$$\text{ヒット率} = 1 - (\text{physical reads} / (\text{db block gets} + \text{consistent gets}))$$

例題の問合せによって取得された統計に基づくと、バッファ・キャッシュ・ヒット率は 94% です。

## バッファ確保の統計

次の統計はバッファ確保を評価する場合に役立ちます。

Buffer pinned	この統計は、必要とするバッファが確保されているかどうかを判断するためにクライアントが調べたときに、そのバッファがクライアントによってすでに確保されていた回数を測定します。
Buffer not pinned	この統計は、必要とするバッファが確保されているかどうかを判断するためにクライアントが調べたときに、そのバッファがクライアントによって確保されていなかった回数を測定します。

クライアントはバッファを解放する前にこのチェックを行います。この場合はクライアントがバッファを使用する意図がないので、これらの統計は増分されません。

これらの統計は、バッファ上の長期にわたる読み一貫性の確保が利益をもたらす頻度についての規準を提供します。確保されたバッファをクライアントが何回も再使用できる場合は、バッファの確保が有益であることを示します。

## キャッシュ・ヒット率の評価

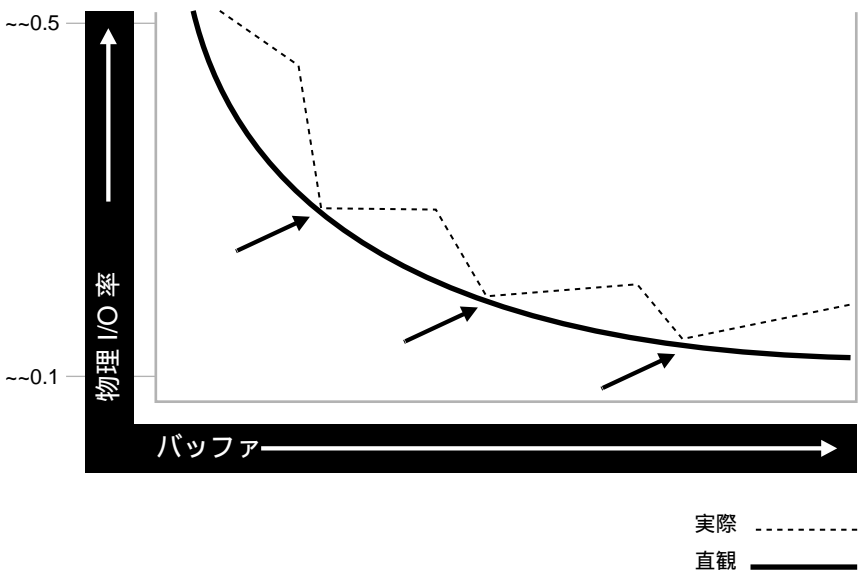
キャッシュ・ヒット率を調べるとき、" 長時間の " 全表走査の間に検出されるブロックは LRU リストの先頭に追加されないことを覚えておいてください。このため、走査を繰り返してもブロックはキャッシュされません。

同一の大容量の表を繰り返し走査するのが、問題の最も効果的なアプローチということはまれです。たとえ、一晩中かかるパッチの組合せを PL/SQL を含まない SQL\*Plus スクリプトとして実装できないとしても、すべての処理を単一パスで実行する方がよいでしょう。そのため、解決策は設計や実装のレベルにあります。

数千または数万のバッファを使用して実行している実稼働サイトが、メモリーを効果的に使用していることはほとんどありません。OLTP アプリケーションを実行するどの大容量データベースでも、常にほとんどの行は 1 回ないし 0 回しかアクセスされません。これに基づけば、行、または行を含むブロックを使用後に長期間メモリーに保存しておいても、ほとんど意味がありません。

最後に、キャッシュ・ヒット率とバッファ数の関係はなめらかな分布ではありません。バッファ・プールをチューニングするときは、キャッシュ・ヒット率の向上にまったく貢献しない（または、ほとんど貢献しない）追加バッファは使用しないでください。図 19-2 で示すように、考慮する価値があるのはごく狭い範囲の DB\_BLOCK\_BUFFERS の値だけです。

図 19-2 バッファ・プールのキャッシュ・ヒット率



**注意：** DB\_BLOCK\_BUFFERS の値を継続して増やすのはよくある間違いです。全表走査や、バッファ・プールを使用しない他の操作を実行している場合は、このように値を増やしても何の効果もありません。

大まかな指針として、以下の場合に DB\_BLOCK\_BUFFERS を増やします。

- キャッシュ・ヒット率が 0.9 未満である場合。
- 不適切なページ・フォルトの形跡がない場合。
- 直前の DB\_BLOCK\_BUFFERS の増加で効果があった場合。

### プール内のバッファの判別

CATPARR.SQL スクリプトによってビュー V\$BH が作成されます。これには、現在 SGA に存在しているブロックのブロック番号とファイル番号が表示されます。CATPARR.SQL は本来はパラレル・サーバー環境で使用されることを目的としていますが、シングル・インスタンス環境を操作している場合でも、SYS として実行できます。

以下の問合せを実行します。

```
SELECT file#, COUNT(block#), COUNT (DISTINCT file# || block#)
FROM V$BH
```

GROUP BY file#;

### バッファ・キャッシュ・ミスの低減によるキャッシュ・ヒット率の増加

60% あるいは 70% を下回るほどヒット率が低い場合、パフォーマンスを改善するために、キャッシュ内のバッファ数を増やしてください。バッファ・キャッシュを大きくするには、初期化パラメータ `DB_BLOCK_BUFFERS` の値を増やしてください。

Oracle では、バッファ・キャッシュのサイズを増やした結果生じるパフォーマンスの利得を見積もる統計を収集できます。これらの統計によって、キャッシュをいくつ追加すべきか見積もることができます。

### キャッシュ・ヒット率が高い場合の不必要なバッファの削除

キャッシュ・ヒット率が高い場合、キャッシュが十分大きく、最も頻繁にアクセスされるデータも保持できる状態になっています。この場合、キャッシュ・サイズを小さくしても、なお優れたパフォーマンスを維持できる可能性があります。バッファ・キャッシュを小さくするには、初期化パラメータ `DB_BLOCK_BUFFERS` の値を小さくします。このパラメータの最小値は 4 です。残りのメモリーは他の Oracle メモリー構造のために使用できます。

Oracle では、小さくしたキャッシュ・サイズに基づくバッファ・キャッシュのパフォーマンスを予測する統計を収集できます。これらの統計を調べることは、パフォーマンスに悪影響を及ぼすことなくバッファ・キャッシュを小さくできる余裕の程度を判断する上で有効です。

## バッファ・キャッシュでの LOB の調整

一時 LOB と永続 LOB の両方ともバッファ・キャッシュを使用できます。

### 一時 LOB

CACHE パラメータを TRUE に設定したときに作成された一時 LOB は、バッファ・キャッシュを介して移動します。CACHE パラメータが FALSE に設定されたときに作成された一時 LOB は、ディスクとの間で直接読み込みと書き込みが行われます。

自動クリーン・アップの期間を使用して時間と手間を省くことができます。また、データベースにとっては、各一時 LOB を明示的に解放するよりも、期間を終了して期間に関連するすべての一時 LOB を解放する方が効果的です。

一時 LOB は、割当て時に自らのディープ・コピーを作成します。次に例を示します。

```
LOCATOR1 BLOB;  
LOCATOR2 BLOB;  
DBMS_LOB.CREATETEMPORARY (LOCATOR1,TRUE,DBMS_LOB.SESSION);  
LOCATOR2 := LOCATOR1;
```

上記のコードでは、LOCATOR1 が指す一時 LOB のコピーが作成されます。PL/SQL の参照方法によるパスの使用を考慮してもよいでしょう。

または、OCI では、次の例のようにロケータに対するポインタを宣言できます。

```
OCILOBDESCRIPTOR *LOC1;  
OCILOBDESCRIPTOR *LOC2;  
OCILOBCREATETEMPORARY (LOC1,TRUE,OCIDURATIONSESSION);  
LOC2 = LOC1;
```

OCILobAssign() コマンドでも一時 LOB のディープ・コピーが行われるので、このコマンドは使用しないでください。つまり、一時 LOB の新規コピーが作成されます。

ポインタの割当てではディープ・コピーは作成されません。ポインタが同じ対象を指すだけです。

## 複数バッファ・プールのチューニング

この項では次の内容を取り上げます。

- [複数バッファ・プールの機能の概要](#)
- [複数バッファ・プールの用途](#)
- [複数バッファ・プールを使ったバッファ・キャッシュのチューニング](#)
- [複数バッファ・プールの使用可能化](#)
- [複数バッファ・プールの使用方法](#)
- [デフォルトのバッファ・プールを表示するディクショナリ・ビュー](#)
- [各バッファ・プールのサイズ設定](#)
- [LRU ラッチ競合の識別と排除](#)

## 複数バッファ・プールの機能の概要

スキーマ・オブジェクトはさまざまな使用パターンで参照されるので、そのキャッシュの動作はかなり異なる場合があります。複数バッファ・プールによって、これらの違いに対処することができます。KEEP バッファ・プールを使用してバッファ・キャッシュ内のオブジェクトをメンテナンスし、RECYCLE バッファ・プールを使用してオブジェクトがキャッシュ内のスペースを不必要に占めるのを防ぐことができます。オブジェクトがキャッシュに割り当てられると、そのオブジェクトのすべてのブロックがそのキャッシュに置かれます。どのバッファ・プールにも割り当てられていないオブジェクトのために、DEFAULT バッファ・プールがメンテナンスされています。

Oracle の各バッファ・プールは多数の作業セットで構成されます。各バッファ・プールにさまざまな数のセットを割り当てられます。すべてのセットが同じ LRU（最低使用頻度）置換

方針を使用します。厳密な LRU 除去方針は、たいいていの場合に優れたヒット率を提供しますが、いくつかのヒントを与えることでさらに改善できることもあります。

LRU リストの主な問題は、大規模セグメントがランダム方式で頻繁にアクセスされるときに発生します。ここで、"大規模"とは、キャッシュのサイズと比較して大きいという意味です。非逐次物理読み込みのかなりの割合（10% を超える）を 1 つのセグメントが占める場合、そのセグメントは、おそらく大規模セグメントといえます。そのような大規模セグメントに対するランダム読み込みは、他のセグメントのデータを含むバッファがキャッシュから除去される原因となります。大規模セグメントは、キャッシュの大きな割合を消費しますが、キャッシングによる利益はありません。

非常に頻繁にアクセスされるセグメントは、バッファが頻繁にウォームされるのでキャッシュから除去されないため、大規模セグメントの読み込みの影響を受けません。主な問題は、大規模セグメントの読み込みによるバッファのフラッシュを免れるほど頻繁にはアクセスされない"ウォーム"セグメントで発生します。

この問題を解決するためのオプションは 2 つあります。1 つは、大規模セグメントを別の RECYCLE キャッシュに移動し、他のセグメントを妨害しないようにすることです。RECYCLE キャッシュは DEFAULT バッファ・プールよりも小さくし、DEFAULT バッファ・プールよりも迅速にバッファを再使用する必要があります。

もう 1 つのアプローチは、大規模セグメントが使わない別の KEEP キャッシュに小さなウォーム・セグメントを移動することです。KEEP キャッシュをサイズ設定して、キャッシュでのミスを最小におさえられます。特定の問合せによってアクセスされるセグメントを KEEP キャッシュに置き、決して除去されないようにすることで、その問合せの応答時間をより予測可能にすることができます。

## 複数バッファ・プールの用途

システム I/O のパフォーマンスを調べるときに、スキーマを分析し、複数バッファ・プールが有利かどうかを判断してください。頻繁にアクセスされる小さな表があり、高速の応答時間を必要とする場合には、KEEP キャッシュを検討してください。ランダム I/O が行われる大規模表は、RECYCLE キャッシュに適した候補です。

次のステップに従って、ある時点で個々のオブジェクトによって使用されるキャッシュの割合を判断してください。

1. 次のように入力して、セグメントの Oracle 内部オブジェクトの数を検索します。

```
SELECT DATA_OBJECT_ID, OBJECT_TYPE FROM USER_OBJECTS
WHERE OBJECT_NAME = '<SEGMENT_NAME>';
```

2 つのオブジェクトが同じ名前を持つことがあるので（異なる型のオブジェクトの場合）、OBJECT\_TYPE 列を使用して目的のオブジェクトを識別できます。オブジェクトが他のユーザーによって所有されている場合は、ビュー USER\_OBJECTS のかわりに DBA\_OBJECTS または ALL\_OBJECTS を使います。



2. SEGMENT\_NAME に対するバッファ・キャッシュ内のバッファ数を検索します。

```
SELECT COUNT(*) BUFFERS FROM V$BH WHERE OBJD = <DATA_OBJECT_ID>;
```

ここで、DATA\_OBJECT\_ID はステップ 1 のものです。

3. インスタンス内にあるバッファの総数を検索します。

```
SELECT VALUE "TOTAL BUFFERS" FROM V$PARAMETER  
WHERE NAME = 'DB_BLOCK_BUFFERS';
```

4. バッファの総数に対するバッファの比率を計算し、現在 SEGMENT\_NAME で使用されているキャッシュの割合を取得します。

$$\text{segment\_nameで使われているキャッシュの割合 (\%)} = \frac{\text{バッファ数 (ステップ 2)}}{\text{バッファ総数 (ステップ 3)}}$$

---

**注意：** この手法は、1 つのセグメントに対してのみ有効です。パーティション・オブジェクトについては、パーティションごとに問合せを実行する必要があります。

---

ローカル・ブロック取得の数が、オブジェクトに関連する文の物理読み数と等しい場合は、そのオブジェクトのバッファ・キャッシュの有用性が限定されるので、RECYCLE キャッシュの使用を検討してください。

## 複数バッファ・プールを使ったバッファ・キャッシュのチューニング

バッファ・キャッシュを複数バッファ・プールにパーティション化するとき、さまざまな方法でアクセスされるオブジェクトのブロックに各バッファ・プールを使用できます。特定のオブジェクトのブロックが再使用されそうな場合は、そのブロックが次に使用されるときにディスク I/O を必要としないように、そのオブジェクトをバッファ・キャッシュ内に確保してください。逆に、ブロックが一定期間再使用されそうにない場合は、そのブロックを破棄して、もっと頻繁に使用されるブロックに入れられるようにします。

適切なバッファ・プールにオブジェクトを適切に割り当てることによって、次のことが可能になります。

- I/O を低減または排除できます。
- キャッシュ内にオブジェクトを分離できます。
- オブジェクトをキャッシュの一部に制限または限定できます。

## 複数バッファ・プールの使用可能化

データベース・インスタンスごとに複数バッファ・プールを作成できます。データベースの各インスタンスについて、必ずしも同じバッファ・プール・セットを定義する必要はありません。インスタンスごとにバッファ・プールのサイズを変えることも、バッファを定義しないこともできます。各インスタンスを個別にチューニングしてください。

### 新しいバッファ・プールの定義

各バッファ・プールは、`BUFFER_POOL_name` 初期化パラメータを使用して定義できます。各バッファ・プールには、2つの属性を指定できます。1つはバッファ・プール内のバッファ数であり、もう1つはバッファ・プールに割り当てられる LRU ラッチの数です。

バッファ・プールの定義に使用される初期化パラメータは次のとおりです。

<code>BUFFER_POOL_KEEP</code>	KEEP バッファ・プールを定義します。
<code>BUFFER_POOL_RECYCLE</code>	RECYCLE バッファ・プールを定義します。
<code>DB_BLOCK_BUFFERS</code>	データベース・インスタンスのためのバッファ数を定義します。この数を総数として個々のバッファ・プールがこの中から作成され、残りは DEFAULT バッファ・プールに割り当てられます。
<code>DB_BLOCK_LRU_LATCHES</code>	データベース・インスタンス全体のための LRU ラッチ数を定義します。定義される各バッファ・プールは、 <code>DB_BLOCK_BUFFERS</code> の場合と同じ方法でこの総数からとられます。

次に例を示します。

```
BUFFER_POOL_KEEP=(buffers:400, lru_latches:3")
BUFFER_POOL_RECYCLE=(buffers:50, lru_latches:1")
```

各バッファ・プールのサイズは、バッファ・キャッシュ全体に関して定義されたバッファの総数（つまり、`DB_BLOCK_BUFFERS` パラメータの値）に含まれます。したがって、すべてのバッファ・プール内のバッファ数の合計は、この値を超えることはできません。同様に、各バッファ・プールに割り当てられた LRU ラッチの数は、`DB_BLOCK_LRU_LATCHES` パラメータを介してインスタンスに割り当てられた総数からとられます。いずれかの制約に違反すると、エラーが表示され、データベースはマウントされません。

各バッファ・プールに割り当てる必要のあるバッファ数の最小値は、LRU ラッチ数の 50 倍です。たとえば、3つの LRU ラッチがあるバッファ・プールは、少なくとも 150 個のバッファを持つ必要があります。

Oracle では、KEEP（保持）、RECYCLE（リサイクル）、DEFAULT（デフォルト）という3つのバッファ・プールが自動的に定義されます。DEFAULT バッファ・プールは常に存在します。DEFAULT バッファ・プールのサイズまたは DEFAULT バッファ・プールに割り当てられる作業セット数は、明示的には定義しません。これらの値は、割り当てられた総数から

他のすべてのバッファ・プールに割り当てられる数を引くことによって推測されます。使用される別のバッファ・プールに対してバッファ・プールの定義する必要性はありません。

## 複数バッファ・プールの使用方法

この項では、オブジェクトに対してデフォルトのバッファ・プールを設定する方法を説明します。オブジェクトのすべてのブロックは、指定されたバッファ・プールに入ります。

BUFFER\_POOL 句は、オブジェクトのデフォルトのバッファ・プールを定義するために使用します。この句は、CREATE 表と ALTER 表、クラスタおよび索引 DDL 文に対して有効です。バッファ・プール名の大文字と小文字は区別されません。バッファ・プールが明示的に設定されていないオブジェクトのブロックは、DEFAULT バッファ・プールに入ります。

バッファ・プールがパーティション表または索引に対して定義されている場合、オブジェクトの各パーティションは、特定のバッファ・プールで上書きされない限り、表または索引定義からバッファ・プールを継承します。

オブジェクトのデフォルトのバッファ・プールが ALTER 文を使用して変更された場合、変更されたセグメントのブロックを現在格納しているすべてのバッファは、ALTER 文を発行する前にあったバッファ・プールに残ります。新たにロードされたブロック、および除去されて再ロードされたブロックは、新しいバッファ・プールに入ります。

BUFFER\_POOL 句の構文は次のとおりです。BUFFER\_POOL {KEEP | RECYCLE | DEFAULT}  
次に例を示します。

```
BUFFER_POOL KEEP
```

または

```
BUFFER_POOL RECYCLE
```

次の DDL 文は、バッファ・プール句を受け付けます。

- CREATE TABLE *table name*... STORAGE (*buffer\_pool\_clause*)

バッファ・プールは、クラスタ化表では許可されていません。クラスタ化表のバッファ・プールは、クラスタ・レベルで指定されます。

索引構成表では、索引とオーバーフロー・セグメントの両方に対してバッファ・プールを定義できます。

パーティション表では、パーティションごとにバッファ・プールを定義できます。バッファ・プールは、各パーティションに対する記憶域句の一部として指定されます。

次に例を示します。

```
CREATE TABLE TABLE_NAME (COL_1 NUMBER, COL_2 NUMBER)
PARTITION BY RANGE (COL_1)
(PARTITION ONE VALUES LESS THAN (10)
STORAGE (INITIAL 10K BUFFER_POOL RECYCLE),
PARTITION TWO VALUES LESS THAN (20) STORAGE (BUFFER_POOL KEEP));
```

- CREATE INDEX *index name*... STORAGE (*buffer\_pool\_clause*)  
グローバル、またはローカルのパーティション索引では、パーティションごとにバッファ・プールを定義できます。
- CREATE CLUSTER *cluster\_name*...STORAGE (*buffer\_pool\_clause*)
- ALTER TABLE *table\_name*...STORAGE (*buffer\_pool\_clause*)  
バッファ・プールは、単純な表の変更 (ALTER TABLE)、パーティションの修正 (MODIFY PARTITION)、パーティションの移動 (MOVE PARTITION)、パーティションの追加 (ADD PARTITION) および両新規パーティションのためのパーティションの分割 (SPLIT PARTITION) のコマンドで定義できます。
- ALTER INDEX *index\_name*...STORAGE (*buffer\_pool\_clause*)  
バッファ・プールは、単純な索引の変更 (ALTER INDEX)、再作成 (REBUILD)、パーティションの修正 (MODIFY PARTITION)、新規パーティションと再作成パーティション両方のためのパーティションの分割 (SPLIT PARTITION) のコマンドで定義できます。
- ALTER CLUSTER *cluster\_name*...STORAGE (*buffer\_pool\_clause*)

## デフォルトのバッファ・プールを表示するディクショナリ・ビュー

次のディクショナリ・ビューには、指定のオブジェクトのデフォルトのバッファ・プールを示す BUFFER POOL 列があります。

USER_CLUSTERS	ALL_CLUSTERS	DBA_CLUSTERS
USER_INDEXES	ALL_INDEXES	DBA_INDEXES
USER_SEGMENTS	DBA_SEGMENTS	
USER_TABLES	USER_OBJECT_TABLES	USER_ALL_TABLES
ALL_TABLES	ALL_OBJECT_TABLES	ALL_ALL_TABLES
DBA_TABLES	DBA_OBJECT_TABLES	DBA_ALL_TABLES
USER_PART_TABLES	ALL_PART_TABLES	DBA_PART_TABLES
USER_PART_INDEXES	ALL_PART_INDEXES	DBA_PART_INDEXES

USER_TAB_PARTITIONS	ALL_TAB_PARTITIONS	DBA_TAB_PARTITIONS
USER_IND_PARTITIONS	ALL_IND_PARTITIONS	DBA_IND_PARTITIONS

ビュー V\$BUFFER\_POOL\_STATISTICS はローカル・インスタンスに対して割り当てられたバッファ・プールを示し、GV\$BUFFER\_POOL\_STATISTICS はデータベース全体に対して割り当てられたバッファ・プールを示します。これらのビューを作成するには、CATPERF.SQL ファイルを実行する必要があります。

## 各バッファ・プールのサイズ設定

この項では、KEEP バッファ・プールと RECYCLE バッファ・プールのサイズを設定する方法を説明します。

### KEEP バッファ・プール

KEEP バッファ・プールの目的は、メモリー内にオブジェクトを保存して、I/O 操作を避けることにあります。したがって、KEEP バッファ・プールのサイズは、バッファ・キャッシュに保持するオブジェクトによって異なります。KEEP バッファ・プールのおおよそのサイズは、このプールに割り当てられるすべてのオブジェクトのサイズを加算することで計算できます。各オブジェクトのサイズを取得するには、ANALYZE コマンドを使用します。ESTIMATE オプションを使用してサイズを大まかに計算することもできますが、COMPUTE STATISTICS オプションの方ができる限り正確な値が提供されるので適しています。

バッファ・プール・ヒット率は、次の計算式を使用して判断できます。

$$\text{ヒット率} = 1 - \frac{\text{physical reads}}{(\text{block gets} + \text{consistent gets})}$$

KEEP バッファ・プールに関する物理読み込み (physical reads) およびブロック取得 (block gets) および一貫取得 (consistent gets) の値は、次の問合せによって取得できます。

```
SELECT PHYSICAL_READS, BLOCK_GETS, CONSISTENT_GETS
FROM V$BUFFER_POOL_STATISTICS WHERE NAME = 'KEEP';
```

KEEP バッファ・プールのヒット率が 100% になるのは、バッファがバッファ・プールにロードされた後だけです。したがって、ヒット率の計算は、システムが実行されてしばらくたち、定常状態のパフォーマンスを達成するまでは行わないでください。ヒット率を計算するには、上記の問合せを使用してシステム・パフォーマンスの 2 つのスナップショットを時間をおいて取ります。物理読み込み (physical reads)、ブロック取得 (block gets) および一貫取得 (consistent gets) について、古い値から新しい値を引いて、これらの値を使用してヒット率を計算します。

100% のバッファ・プール・ヒット率が最適とはかぎりません。KEEP バッファ・プールのサイズを減らしても、十分に高いヒット率が維持されることがよくあります。KEEP バッファ・プールから除去したブロックは、別のバッファ・プールに割り当ててください。

---

**注意：** オブジェクトのサイズが大きくなった場合に、KEEP バッファ・プールに入りきらなくなることがあります。この場合、キャッシュからブロックが失われ始めます。

---

各オブジェクトをメモリー内に保持するとトレードオフが発生します。頻繁にアクセスされるブロックをキャッシュに保持することは有効ですが、頻繁に使用しないブロックを保持すると、よりアクティブな他のブロックのためのスペースが減ることになります。

## RECYCLE バッファ・プール

RECYCLE バッファ・プールの目的は、不要になったブロックをすぐにメモリーから排除することにあります。アプリケーションがラージ・オブジェクトのブロックをランダム方式でアクセスする場合は、バッファ・プールに格納されているブロックが除去される前に、再使用できる可能性はほとんどありません。（使用可能物理メモリーの量の制約により）これはバッファ・プールのサイズに関係なくあてはまります。そのため、そのオブジェクトのブロックはキャッシュしてはなりません。これらのキャッシュ・バッファは、他のオブジェクトに割り当てることができます。

ただし、メモリーからあまりに迅速にブロックを廃棄しないように注意してください。バッファ・プールが小さすぎると、トランザクションまたは SQL 文が実行を完了する前に、ブロックがキャッシュから除外されてしまう可能性があります。たとえば、アプリケーションが表から値を選択し、その値を使用してデータを処理し、レコードを更新する場合があります。文を選択した後でブロックがキャッシュから削除された場合は、更新を実行するために再度ディスクから読み込まなければなりません。ブロックは、ユーザー・トランザクションの所要時間中は保存される必要があります。

Oracle Trace などの SQL 文のチューニング・ツールを使用して、または SQL トレース機能を使用可能にして文を実行し、トレース・ファイル上で TKPROF を実行すると、ディスクから物理的に読み込まれるデータ・ブロックの総数のリストを取得できます（この数値は、TKPROF 出力の "disk" 列に表示されます）。特定の SQL 文に関するディスク読み回数は、DEFAULT バッファ・プールから割り当てられたすべてのオブジェクトに関する同じ SQL 文のディスク読み回数を超えてはなりません。

RECYCLE バッファ・プールが小さすぎるとかを判断するために使用できる統計は、他に 2 つあります。"free buffer waits" 統計値が非常に高い場合は、おそらくプールが小さすぎます。"log file sync" 待機イベントが増加している場合も同様です。RECYCLE バッファ・プールのサイズを設定する 1 つの方法は、RECYCLE バッファ・プールを使用禁止にしてシステムを実行することです。定常状態で、通常は RECYCLE バッファ・プールに入るセグメントが消費する DEFAULT バッファ・プール内のバッファ数を定数 4 で除算します。この結果を使用して RECYCLE キャッシュのサイズを設定します。

## KEEP バッファ・プールと RECYCLE バッファ・プールに入れるセグメントの識別

RECYCLE バッファ・プールに入れるのに適したセグメントとは、DEFAULT バッファ・プールの少なくとも 2 倍のサイズで、システム内の I/O 総数の少なくとも数 % を実行しているセグメントです。

KEEP バッファ・プールに入れるのに適したセグメントとは、DEFAULT バッファ・プールの 10% 未満のサイズで、システム内の I/O 総数の少なくとも 1% を実行しているセグメントです。

これらのルールにおける問題は、表領域に複数のセグメントがある場合に、セグメントあたりの I/O 回数を判断するのが難しいことです。この問題を解決する 1 つの方法は、V\$SESSION\_WAIT から選択することで、ある期間中に発生する I/O をサンプリングし、セグメントあたりの I/O の統計分布を判断することです。

もう 1 つのオプションは、バッファ・キャッシュ内のセグメントのブロックの位置を調べることです。特に、キャッシュのホット・ハーフにあるセグメントについてのブロックのカウントが、同一のセグメントについてのコールド・ハーフのカウントに対する比率を見ると、どれがホット・セグメントでどれがコールド・セグメントかがかなりよくわかります。セグメントで、この比率が 1 に近くなるほど、そのセグメントのバッファは頻繁に使用されず、セグメントは RECYCLE キャッシュに入れるのに適した候補となります。比率が高い場合（おそらく 3.0）は、バッファが頻繁に使用されているので、そのセグメントは KEEP キャッシュの候補として適切です。

## LRU ラッチ競合の識別と排除

LRU ラッチは、バッファ・キャッシュが使う最低使用頻度 (LRU) バッファ・リストを管理するために使います。ラッチ競合が発生している場合、プロセスはラッチを取得する前に待機とスピンを行います。

データベース・インスタンスのラッチの総数は、DB\_BLOCK\_LRU\_LATCHES パラメータを使って設定できます。各バッファ・プールが定義されるときに、これらの LRU ラッチ数をバッファ・プール用に確保できます。バッファ・プールのバッファは、バッファ・プールの LRU ラッチ間で均等に分割されます。

システムでラッチ競合が発生しているかどうかを判断するには、いずれかのラッチで LRU ラッチ競合が発生しているかどうかを判断することから始めます。

```
SELECT child#, sleeps / gets ratio
FROM V$LATCH_CHILDREN
WHERE name = 'cache buffers lru chain';
```

各 LRU ラッチのミス率は、1% 未満でなければなりません。特定のラッチで 1% を超える率になった場合は、LRU ラッチ競合が発生していることを示しているので、対処する必要があります。次の問合せを使って、ラッチが対応付けられているバッファ・プールを判断できます。

```
SELECT name FROM V$BUFFER_POOL_STATISTICS
WHERE lo_setid <= child_latch_number
AND hi_setid >= child_latch_number;
```

ここで、*child\_latch\_number* は、前の問合せの *child#* です。

LRU ラッチの競合は、システム内のラッチの総数と、2 番目の問合せで示されたバッファ・プールに割り当てられているラッチ数を増やすことで軽減できます。

許可されているラッチの最大数は、次の小さい方の値です。

$$\text{number\_of\_cpus} * 2 * 3 \quad \text{または} \quad \text{number\_of\_buffers} / 50$$

この制限が存在するのは、50 未満のバッファを持つセットはないためです。最大を超える値を指定すると、ラッチ数はこの公式による最大値に自動的にリセットされます。

たとえば、CPU 数が 4 でバッファ数が 200 の場合、ラッチの最大数として 4 が許可されます (  $4 * 2 * 3$  と  $200 / 50$  の小さい方 )。CPU 数が 4 でバッファ数が 10000 の場合、ラッチの最大数として 24 が許可されます (  $4 * 2 * 3$  と  $10000 / 50$  の小さい方 )。

## ソート領域のチューニング

大容量のソートが頻繁に発生する場合は、以下の 2 つの目標のどちらかまたは両方を目指して、パラメータ *SORT\_AREA\_SIZE* の値を増やすことを考慮してください。

- メモリー内で完全に実施できるソートの数を増やす。
- メモリー内では完全に実施できないソートの速度を上げる。

大きい *SORT\_AREA\_SIZE* と小さい *SORT\_AREA\_RETAINED\_SIZE* を組み合わせると、大きなソート領域を効果的に使用できます。ユーザーがデータベースから切断するまでメモリーが解放されない場合、ソートの作業領域が大きいと問題が発生する可能性があります。パラメータ *SORT\_AREA\_RETAINED\_SIZE* を使って、ソートの後にすぐ解放できるメモリーのレベルを指定します。多数の同時ユーザーがいるシステムで大きなソート領域が使われている場合は、このパラメータをゼロに設定します。

*SORT\_AREA\_RETAINED\_SIZE* は問合せのソート操作ごとに保持されます。このため、4 つの表がソート・マージでソートされている場合、Oracle は 4 つの領域の *SORT\_AREA\_RETAINED\_SIZE* を保持しています。

**関連項目：** [第 26 章「パラレル実行のチューニング」](#)



## メモリーの再割当て

Oracle のメモリー構造のサイズを変更したら、ライブラリ・キャッシュ、データ・ディクショナリ・キャッシュおよびバッファ・キャッシュのパフォーマンスを再評価してください。これらの構造のいずれかのメモリー消費を節減した場合、別の構造に割り当てるメモリーを増やすとよいでしょう。たとえば、バッファ・キャッシュのサイズを減らした場合は、その分のメモリーをライブラリ・キャッシュに使用するとよいでしょう。

もう一度オペレーティング・システムをチューニングしてください。Oracle のメモリー構造体のサイズを変更したので、Oracle のメモリー要件が変更されているかもしれません。特に、ページングとスワッピングが過度に発生しないことを確認してください。たとえば、データ・ディクショナリ・キャッシュまたはバッファ・キャッシュのサイズを大きくすると、SGA が大きくなって主メモリーに収まらないこともあります。この場合、SGA のページングやスワッピングが発生します。

メモリーの再割当て時に、Oracle のメモリー構造体のサイズを最適化するにはオペレーティング・システムで提供できるメモリーよりも大きなメモリーが必要な場合があります。この場合、コンピュータにメモリーを追加することによって、さらにパフォーマンスを改善できます。

## 合計メモリー使用量の低減

サーバーに十分なメモリーがないので、現在構成されているアプリケーションが実行できないこと、さらにそのアプリケーションが論理的に単一のアプリケーションである（つまり複数のサーバー間でセグメント化または分散できない）ことがパフォーマンス上の最優先の問題である場合、可能性のある解決策は次の 2 つです。

- 使用可能なメモリー容量を増やす。
- 使用されているメモリー容量を減らす。

サーバーのメモリー使用量の大幅な減少は、常にデータベースの接続数の減少に起因します。また、これによって、オープンしているネットワーク・ソケット数やオペレーティング・システムのプロセス数に関する問題も解決されます。ただし、ユーザー数を減らさずに接続数を減らすためには、残っている接続を共有する必要があります。これによって、ユーザー・プロセスは強制的に枠組みに組み込まれます。そこでは、データベースに送られるすべてのメッセージ要求は、完全またはアトミックなトランザクションを説明します。

このモデルに準拠するアプリケーションの作成は、必ずしも制約が多いまたは難しいということはありませんが、異なっていることは確かです。Oracle Forms などの既存アプリケーションを変換して準拠させるのは、完全に再作成しないかぎり通常は不可能です。

Oracle マルチスレッド・サーバー・アーキテクチャは、サーバーのオペレーティング・システム・プロセス数の低減には非常に効果的なソリューションです。また、MTS は、全体のメモリー所要量の低減にも高い効果があります。また、MTS を接続プーリングや接続多重化といっしょに使用すると、ネットワーク接続数を低減することもできます。

クライアントでもある中間サーバーを使用すると、Oracle Forms 環境では共有接続が可能です。この構成では、DBMS\_PIPE 機構を使用して、中間サーバー上のユーザー個々の接続から中間サーバーの共有デーモンにアトミック要求を転送することができます。かわりにデーモンが中央サーバーに対する接続を所有します。

---

## I/O のチューニング

この章では、Oracle の機能が最大限に発揮されない原因となる I/O のボトルネックを回避する方法を説明します。この章のトピックは次のとおりです。

- I/O の問題について
- I/O の問題の検出
- I/O 問題の解決
  - I/O の分散によるディスク競合の低減
  - ディスクのストライプ化
  - 動的な領域管理の回避
  - ソートのチューニング
  - チェックポイントのチューニング
  - LGWR および DBWn の I/O のチューニング
  - 大規模ブールの構成

### I/O の問題について

この項では、I/O のパフォーマンス問題について説明します。内容は次のとおりです。

- I/O のチューニング: トップ・ダウンとボトム・アップ
- I/O 要件の分析
- ファイル格納の計画
- データ・ブロック・サイズの選択
- デバイス帯域幅の評価

多くのソフトウェア・アプリケーションのパフォーマンスは、本質的にディスク入出力 (I/O) によって制限されます。CPU アクティビティは、I/O アクティビティが完了するまで中断しなければなりません。このようなアプリケーションは "I/O バウンド" と呼ばれます。Oracle は、I/O によってパフォーマンスが制限されることのないように設計されています。

データベース・ファイルを含んでいるディスクが最大限の性能で作動している場合、I/O のチューニングはパフォーマンスの向上に役立ちます。しかし、"CPU バウンド" の場合、すなわちコンピュータの CPU がその最大限の性能で作動している場合には、I/O のチューニングをパフォーマンスに役立てることができません。

**第 19 章「メモリー割当てのチューニング」**で提示された推奨事項に従った後で、I/O をチューニングすることが重要です。その章では、I/O を最小限まで低減するためのメモリー割当て方法を説明しています。最小限の I/O を実現したら、この章の指示に従ってさらに効果的な I/O パフォーマンスを達成してください。

## I/O のチューニング: トップ・ダウンとボトム・アップ

新しい設計をするときに、I/O のニーズをトップ・ダウンで分析し、目的のパフォーマンスを達成するために必要なリソースは何か判断してください。

既存のシステムでは、I/O のチューニングをボトム・アップで行うアプローチをとる必要があります。

1. システム上のディスクの数を判断する。
2. Oracle が使っているディスクの数を判断する。
3. システムが実行する I/O のタイプを判断する。
4. I/O がファイル・システムとロー・デバイスのどちらに対して行われるかを確かめる。
5. マニュアル・ストライプ化またはストライプ化ソフトウェアを使ってオブジェクトが複数のディスクにどのように分散されているかを判断する。
6. 予想されるパフォーマンスのレベルを計算する。

## I/O 要件の分析

この項では、システムの I/O 要件を判断する方法を説明します。

1. アプリケーションが必要とするスループットの合計を計算します。

まず、各トランザクションに関連する読み込みと書き込みの回数を計算し、各操作の実行対象となるオブジェクトを判別します。

たとえば、ある OLTP アプリケーションでは、各トランザクションが次の操作を実行します。

- オブジェクト A からの 1 回の読み
- オブジェクト B からの 1 回の読み
- オブジェクト C への 1 回の書き

この例では、1 つのトランザクションがすべての異なるオブジェクトに対して読みを 2 回と書きを 1 回行う必要があります。

2. システムがサポートする必要のある tps (1 秒あたりのトランザクション数) を指定することによって、このアプリケーションの I/O パフォーマンス目標を定義します。

この例では、設計者は、許容できるパフォーマンス・レベルを 100 tps で実現することを指定できます。これを達成するには、システムは 1 秒間に 300 回の I/O を実行できなければなりません。

- オブジェクト A からの 100 回の読み
- オブジェクト B からの 100 回の読み
- オブジェクト C への 100 回の書き

3. このレベルのパフォーマンスを達成するために必要なディスクの数を判別します。

これを計算するには、各ディスクが 1 秒間に実行できる I/O の回数を確認めます。この数値は次の 3 つの要因によって異なります。

- 使用している特定ディスク・ハードウェアのスピード
- 必要な I/O が読みと書きのどちらであるか
- ファイル・システムとロー・デバイスのどちらを使用しているか

一般に、ディスク・スピードには次の特性があります。

表 20-1 相対ディスク・スピード

ディスク・スピード:	ファイル・システム	ロー・デバイス
1 秒あたりの読み	速い	遅い
1 秒あたりの書き	遅い	速い

4. 表 20-2 のような表に、使用しているディスクの 1 操作あたりの相対スピードをまとめてください。

表 20-2 ディスク I/O 分析のワークシート

ディスク・スピード:	ファイル・システム	ロー・デバイス
1 秒あたりの読み		
1 秒あたりの書き		

5. この例のディスクには、表 20-3 に示す特性があります。

表 20-3 サンプル・ディスク I/O 分析

ディスク・スピード:	ファイル・システム	ロー・デバイス
1 秒あたりの読み	50	45
1 秒あたりの書き	20	50

6. I/O パフォーマンス目標を達成するために必要なディスク数を計算します。表 20-4 のような表を使用します。

表 20-4 ディスク I/O 要件のワークシート

オブジェクト	ファイル・システムに格納されている場合			ロー・デバイスに格納されている場合		
	1 秒間に必要とされる読み書き	1 秒あたりのディスク読み書き能力	必要なディスク	1 秒間に必要とされる読み書き	1 秒あたりのディスク読み書き能力	必要なディスク
A						
B						
C						
必要なディスク						

表 20-5 に、この例の値を示します。

表 20-5 サンプル・ディスクの I/O 要件

オブジェクト	ファイル・システムに格納されている場合			ロー・デバイスに格納されている場合		
	1 秒間に必要とされる読み書き	1 秒あたりのディスク読み書き能力	必要なディスク	1 秒間に必要とされる読み書き	1 秒あたりのディスク読み書き能力	必要なディスク
A	100 回の読み込み	50 回の読み込み	2 個のディスク	100 回の読み込み	45 回の読み込み	2 個のディスク
B	100 回の読み込み	50 回の読み込み	2 個のディスク	100 回の読み込み	45 回の読み込み	2 個のディスク
C	100 回の書き込み	20 回の書き込み	5 個のディスク	100 回の書き込み	50 回の書き込み	2 個のディスク
必要なディスク			9 個のディスク			6 個のディスク

## ファイル格納の計画

この項では、次の方法を使用したときにアプリケーションが最も効率よく実行されるかどうかを判別する方法を説明します。

- 使用可能なディスク上でのアプリケーションの実行
- ロー・デバイスへのデータの格納
- ブロック・デバイスへのデータの格納
- ファイル・システムへのデータの直接格納

### 設計アプローチ

次のアプローチに従ってファイル格納を設計します。

1. アプリケーションが必要とする操作を識別します。
2. アプリケーションが必要とするさまざまな操作についてシステムのディスクのパフォーマンスをテストします。
3. 最後に、どの種類のディスク・レイアウトが、アプリケーションで多く行われる操作について最高のパフォーマンスを達成できるかを評価します。

この手順は、以降の項目で詳しく説明します。

必要な読み書き操作の識別

アプリケーションを評価し、それが各タイプの I/O 操作を要求する頻度を判断します。表 20-6 に、各バックグラウンド・プロセス、フォアグラウンド・プロセスおよびパラレル実行サーバーによって実行される読みおよび書き操作のタイプを示します。

表 20-6 Oracle プロセスが実行する読み書き操作

操作	プロセス							
	LGWR	DBWn	ARCH	SMON	PMON	CKPT	フォアグラウンド	PQ プロセス
順次読み			X	X		X	X	X
順次書き	X		X			X		
ランダム読み				X			X	
ランダム書き		X						

この説明では、サンプル・アプリケーションは 50% のランダム読み、25% の順次読みおよび 25% のランダム書きを含みます。

ディスクのパフォーマンスのテスト

この項では、特定のテスト・システムによる読み書き操作の相対パフォーマンスを示します。ロー・デバイスでは、読みと書きはキャラクタ・レベルで行われます。ブロック・デバイスでは、これらの操作はブロック・レベルで行われます（多くの同時プロセスは、ディスク・ドライブのヘッドとアームの移動によるオーバーヘッドを生成する可能性があります）。

**注意：** この例で提供される数値は、経験則を構成するものではありません。これらは、特定のディスクを使用する実際の UNIX のテスト・システムによって生成されました。プラットフォームとディスクが異なると、これらの数値も大幅に異なります。正確な判断を行うには、この項で示すようなアプローチに従って、使用しているシステムをテストしてください。あるいは、さまざまな操作のディスク・パフォーマンスについてはシステム・ベンダーに問い合せてください。



表 20-7 および図 20-1 に、テスト・システム上の 3 つのディスク・レイアウト・オプションそれぞれについて、1 回の I/O あたりの順次読み込みのスピードをミリ秒で示します。

表 20-7 順次読み込みのブロック・サイズとスピード (サンプル・データ)

順次読み込みのスピード:			
ブロック・サイズ	ロー・デバイス	ブロック・デバイス	UNIX ファイル・システム
512 バイト	1.4	0.6	0.4
1KB	1.4	0.6	0.3
2KB	1.5	1.1	0.6
4KB	1.6	1.8	1.0
8KB	2.7	3.0	1.5
16KB	5.1	5.3	3.7
32KB	10.1	10.3	8.1
64KB	20.0	20.3	18.0
128KB	40.4	41.3	36.1
256KB	80.7	80.3	61.3

このような調査を行うことは、正しいストライプ・サイズの判断に役立ちます。この例では、16KB を読み込むのに最大でも 5.3 ミリ秒しかかかりません。データが 256KB ずつまとまりになっている場合、データを 16 のディスクにストライプ化でき (20-22 ページで説明) このように短い読み込み時間を維持できます。

これに対して、すべてのデータが 1 つのディスク上にある場合は、読み込み時間は 80 ミリ秒となります。このように、テスト結果は、この特定のディスク・セット上では結果が予想とかなり異なることを示しています。I/O のサイズによっては、ストライプ・サイズを小さくすることが有益な場合があります。

図 20-1 順次読み込みのブロック・サイズとスピード (サンプル・データ)

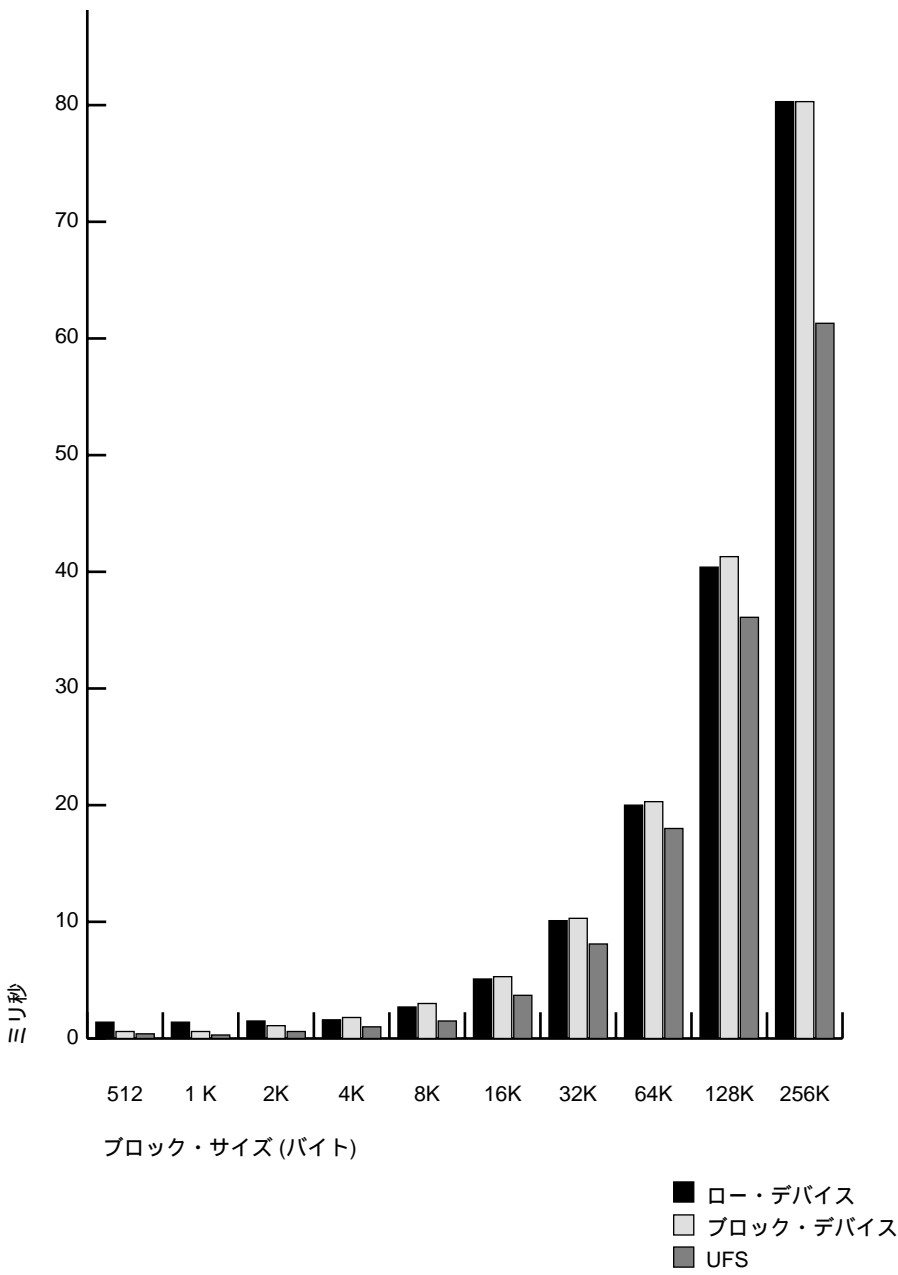


表 20-8 および図 20-2 に、テスト・システム上の 3 つのディスク・レイアウト・オプションそれぞれについて、1 回の I/O あたりの順次書込みのスピードをミリ秒で示します。

表 20-8 順次書込みのブロック・サイズとスピード (サンプル・データ)

順次書込みのスピード			
ブロック・サイズ	ロー・デバイス	ブロック・デバイス	UNIX ファイル・システム
512 バイト	11.2	11.8	17.9
1KB	11.7	11.9	18.3
2KB	11.6	13.0	19.0
4KB	12.3	13.8	19.8
8KB	13.5	13.8	21.8
16KB	16.0	27.8	35.3
32KB	19.3	55.6	62.2
64KB	31.5	111.1	115.1
128KB	62.5	224.5	221.8
256KB	115.6	446.1	429.0

図 20-2 順次書込みのブロック・サイズとスピード (サンプル・データ)

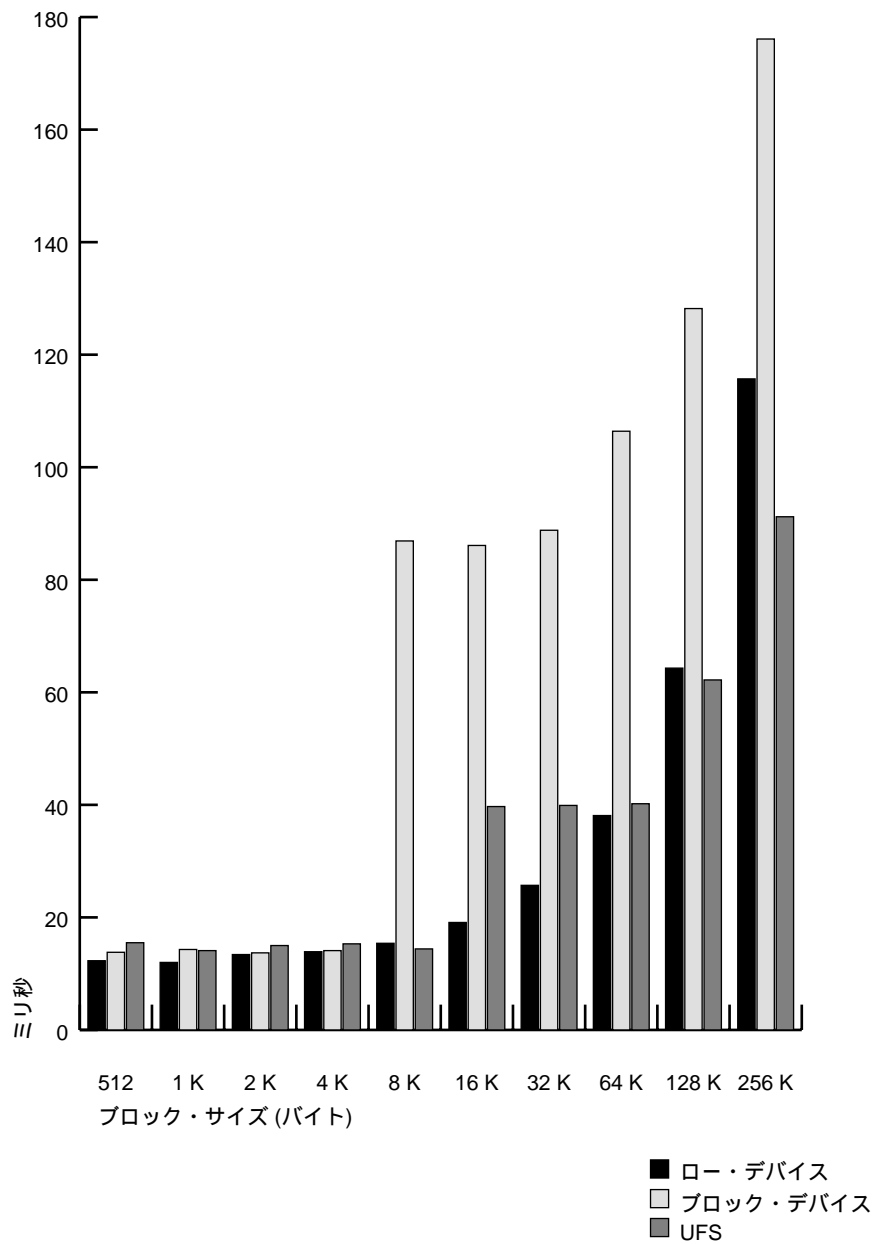


表 20-9 および図 20-3 に、テスト・システム上の 3 つのディスク・レイアウト・オプションそれぞれについて、1 回の I/O あたりのランダム読み込みのスピードをミリ秒で示します。

表 20-9 ランダム読み込みのブロック・サイズとスピード (サンプル・データ)

ランダム読み込みのスピード			
ブロック・サイズ	ロー・デバイス	ブロック・デバイス	UNIX ファイル・システム
512 バイト	12.3	13.8	15.5
1KB	12.0	14.3	14.1
2KB	13.4	13.7	15.0
4KB	13.9	14.1	15.3
8KB	15.4	86.9	14.4
16KB	19.1	86.1	39.7
32KB	25.7	88.8	39.9
64KB	38.1	106.4	40.2
128KB	64.3	128.2	62.2
256KB	115.7	176.1	91.2

図 20-3 ランダム読み込みのブロック・サイズとスピード (サンプル・データ)

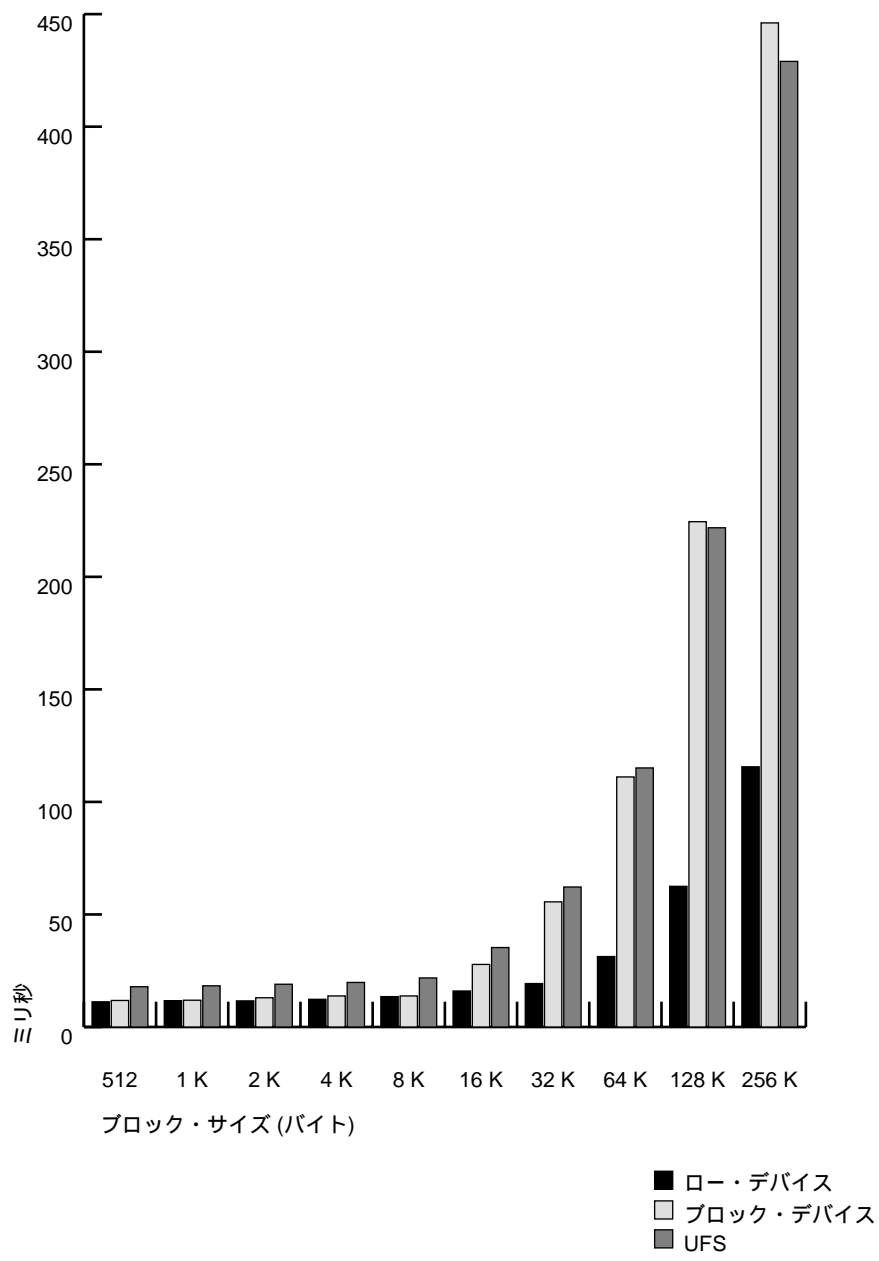
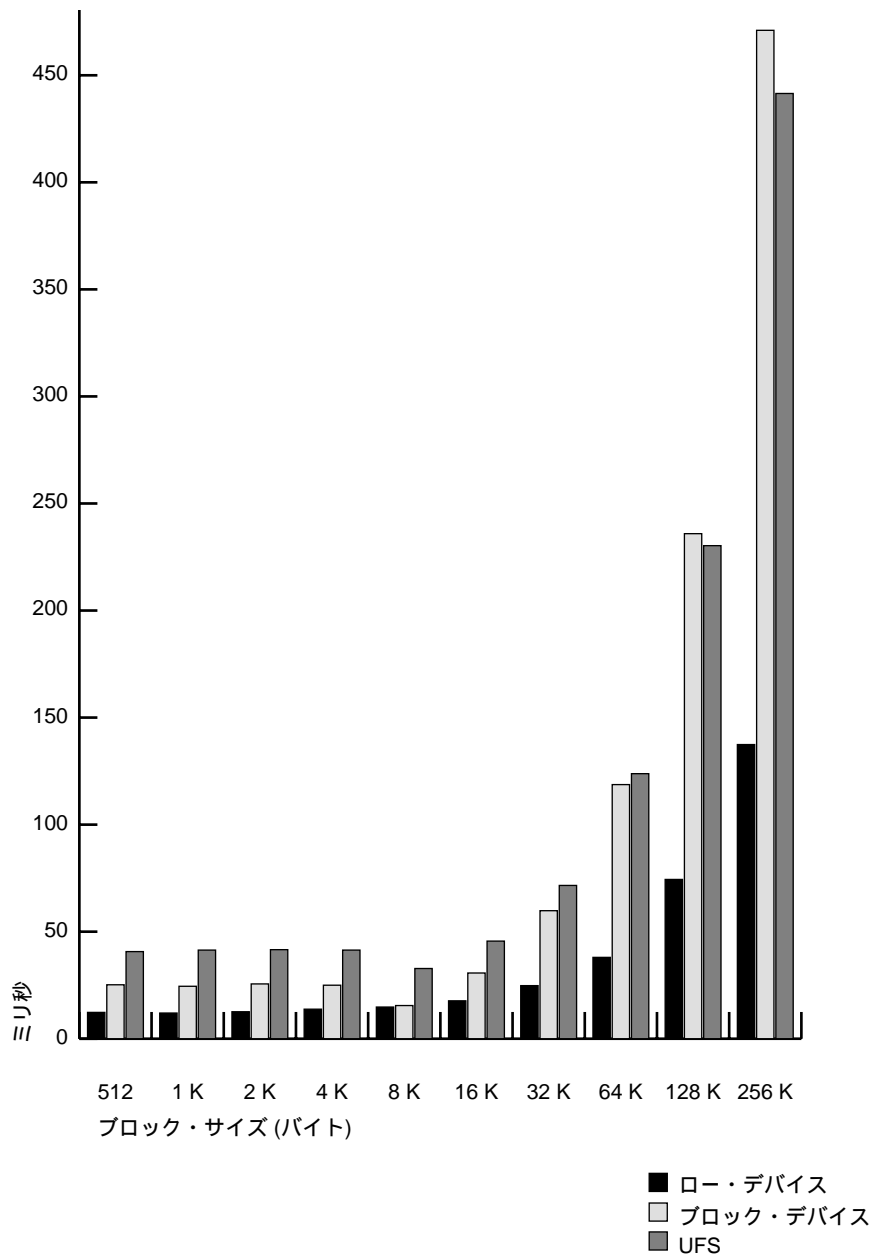


表 20-10 および図 20-4 に、テスト・システム上の 3 つのディスク・レイアウト・オプションそれぞれについて、1 回の I/O あたりのランダム書込みのスピードをミリ秒で示します。

表 20-10 ランダム書込みのブロック・サイズとスピード (サンプル・データ)

ランダム書込みのスピード			
ブロック・サイズ	ロー・デバイス	ブロック・デバイス	UNIX ファイル・システム
512 バイト	12.3	25.2	40.7
1KB	12.0	24.5	41.4
2KB	12.6	25.6	41.6
4KB	13.8	25.0	41.4
8KB	14.8	15.5	32.8
16KB	17.7	30.7	45.6
32KB	24.8	59.8	71.6
64KB	38.0	118.7	123.8
128KB	74.4	235.9	230.3
256KB	137.4	471.0	441.5

図 20-4 ランダム書込みのブロック・サイズとスピード (サンプル・データ)





## ディスク・レイアウト・オプションの評価

アプリケーション内で多く行われる操作のタイプと、その操作に対応する I/O をシステムが処理できるスピードがわかると、パフォーマンスを最大化するディスク・レイアウトを選択できます。

たとえば、前述のサンプル・アプリケーションとテスト・システムでは、UNIX ファイル・システムが適切な選択肢です。ランダム読み込みが多く行われる（すべての I/O 操作の 50%）場合は、8KB が適切なブロック・サイズです。UNIX ファイル・システムでのロー・デバイスは、このブロック・サイズでランダム読み込みの遜色のないパフォーマンスを達成できます。さらに、この例の UNIX ファイル・システムでは、ブロック・サイズが 8KB の場合に、ロー・デバイスのほぼ 2 倍の速さで順次読み込み（すべての I/O 操作の 25%）を処理します。

---

**注意：** プラットフォームおよびディスクが異なると、前の例で示した数字は大幅に異なります。効果的な計画を行うには、自分のシステムで I/O のパフォーマンスをテストしてください。

---

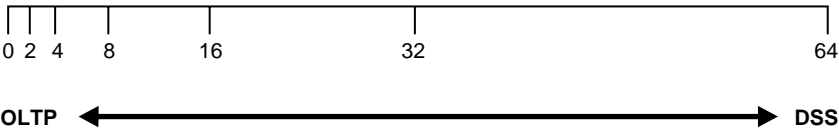
## データ・ブロック・サイズの選択

データベースの表データはデータ・ブロックに格納されます。この項では、最高のパフォーマンスを達成するためのデータ・ブロック内の領域の割当て方法を説明します。単一ブロック I/O（ランダム読み込み）では、最高のパフォーマンスを得るために 1 回の読み込みで 1 つのブロックから目的のデータすべてを取り出す必要があります。データの格納方法によって、このパフォーマンス目標が達成されるかどうかが決まります。これは 2 つの要因に依存します。すなわち、行の格納方法とブロック・サイズです。

オペレーティング・システムの I/O サイズは、データベース・ブロック・サイズと同等以上であることが必要です。（「[ディスクのパフォーマンスのテスト](#)」で説明したように）オペレーティング・システムの I/O サイズがデータベース・ブロック・サイズの 2 倍または 3 倍であると、順次読み込みのパフォーマンスが向上します。これは、次のブロックが特定のバッファから読み込まれるように、オペレーティング・システムが I/O をバッファリングできることを前提としています。

図 20-5 に、オンライン・トランザクション処理（OLTP）アプリケーションまたは意志決定支援（DSS）アプリケーションに適したさまざまなブロック・サイズを示します。

図 20-5 ブロック・サイズとアプリケーション・タイプ



**関連項目：** 使用しているプラットフォームの最小および最大のブロック・サイズの詳細は、プラットフォーム固有の Oracle マニュアルを参照してください。

ブロック・サイズの利点と欠点

この項では、さまざまなブロック・サイズの利点と欠点を説明します。

表 20-11 ブロック・サイズの利点と欠点

ブロック・サイズ	利点	欠点
小 (2KB ~ 4KB)	ブロック競合が低減されます。 行数が少ない場合または大量のランダム・アクセスに適しています。	オーバーヘッドが比較的大きいこと。 行サイズによってはわずかな行数を保存するだけで使い果たされます。
標準 (8KB)	行が標準サイズの場合は、1 回の I/O でバッファ・キャッシュに複数行を入れることができます。2KB または 4KB のブロック・サイズでは、1 行しか入れられないことがあります。	ブロック・サイズが大きい場合に少数の行にランダム・アクセスすると、バッファ・キャッシュ内の領域が浪費されます。たとえば、8KB のブロック・サイズと 50 バイトの行サイズでは、ランダム・アクセスを行うときにバッファ・キャッシュ内の 7,950 バイトが浪費されます。
大 (16KB ~ 32KB)	オーバーヘッドが比較的小さいので、有用なデータを格納する空間が多くなります。 順次アクセスまたは非常に大きな行に適しています。	大きなブロック・サイズは、OLTP 型の環境で使われる索引ブロックには適していません。これは、索引リーフ・ブロック上のブロック競合が増えるためです。

デバイス帯域幅の評価

ディスクが実行できる I/O の回数は、オブジェクトに対する操作が読み込みと書き込みのどちらであるか、およびそのオブジェクトがロー・デバイスとファイル・システムのどちらに格納されているかによって異なります。これは、目的のパフォーマンス・レベルを達成するために使う必要のあるディスク数に影響します。

## I/O の問題の検出

この項では、I/O の使用率に問題があると考える場合に実行する 2 つの作業について説明します。

- システムの I/O 使用率の検査
- Oracle の I/O 使用率の検査

Oracle はデータベース・ファイルへのディスク・アクセスを反映するファイル I/O 統計を収集します。これらの統計は、Oracle セッションの I/O 使用率だけをレポートしますが、使用可能な I/O リソースにはすべてのプロセスが影響します。このため、Oracle 以外の要因をチューニングするとパフォーマンスが向上することがあります。

### システムの I/O 使用率の検査

オペレーティング・システムの監視ツールを使用して、システム全体で実行されているプロセスを判別し、すべてのファイルに対するディスク・アクセスを監視してください。データ・ファイルと REDO ログ・ファイルを保持しているディスクは、Oracle に関連しないファイルも保持している可能性があることを忘れないでください。データベース・ファイルを含むディスクに対する過度のアクセスを低減するようにしてください。Oracle 以外のファイルへのアクセスは、VSFILESTAT ビューを介してではなく、オペレーティング・システムの機能を介してのみ監視できます。

多くの UNIX システムにある `sar -d` などのツールを使用すると、システム全体の `iostat`、つまり I/O 統計を調べることができます（一部の UNIX ベース・プラットフォームには `iostat` コマンドがあります）。NT システムでは、パフォーマンス モニタを使います。

---

**注意：** その他のプラットフォームの情報については、使用しているオペレーティング・システムのマニュアルで確認してください。

---

### Oracle の I/O 使用率の検査

この項では、Oracle の I/O 統計を提供するビューとそのプロセスを説明し、VSFILESTAT を使用して統計をチェックする方法を示します。

#### I/O 統計を含む動的パフォーマンス・ビュー

表 20-12 に、Oracle データベース・ファイル、ログ・ファイル、アーカイブ・ファイルおよび制御ファイルに関連する I/O 統計をチェックするための動的パフォーマンス・ビューを示します。

表 20-12 Oracle ファイルについての統計を含むビュー

ファイル・タイプ	統計を含むビュー
データベース・ファイル	V\$FILESTAT
ログ・ファイル	V\$SYSSTAT、V\$SYSTEM_EVENT、V\$SESSION_EVENT
アーカイブ・ファイル	V\$SYSTEM_EVENT、V\$SESSION_EVENT
制御ファイル	V\$SYSTEM_EVENT、V\$SESSION_EVENT

Oracle のファイル I/O を反映するプロセス

表 20-13 に、プロセスの統計がさまざまな Oracle ファイル・タイプに対する I/O スループットを反映するプロセスのリストを示します。

表 20-13 Oracle プロセスのファイル・スループット統計

プロセス								
ファイル	LGWR	DBWn	ARCH	SMON	PMON	CKPT	フォアグラウンド	PQ プロセス
データベース・ファイル		X		X	X	X	X	X
ログ・ファイル	X							
アーカイブ・ファイル			X					
制御ファイル	X	X	X	X	X	X	X	X

たとえば V\$SYSTEM\_EVENT は、I/O のタイプ別に I/O 数の合計と平均所要時間を示しています。これによって、どの I/O タイプが遅すぎるかを判断します。Oracle に関連する I/O の問題がある場合は、それをチューニングします。ユーザーのプロセスが使用可能な I/O リソースを消費していない場合は、他のプロセスが消費しています。システムに戻って I/O が多すぎるプロセスを識別し、その理由を判断します。その後、そのプロセスをチューニングします。

**注意：** Oracle の I/O タイプが異なると、必要なチューニング・アプローチも異なります。データ・ウェアハウス・アプリケーションの I/O のチューニングは、OLTP アプリケーションの I/O のチューニングとは異なります。データ・ウェアハウス・アプリケーションは大規模な順次読みを行うのが特徴ですが、OLTP はランダム読みと書き込みを行うのが特徴です。

関連項目： 20-5 ページの「[ファイル格納の計画](#)」

## V\$FILESTAT を使用して Oracle データ・ファイル I/O をチェックする方法

動的パフォーマンス・ビュー V\$FILESTAT によって、データベース・ファイルへのディスク・アクセスを調べてください。このビューは、データベース I/O ( ログ・ファイル I/O でなく ) に関する次の情報を示します。

- 物理的な読み込みと書き込みの回数
- 読み込みおよび書き込みを行ったブロック数
- 読み込みと書き込みの I/O 時間の合計

デフォルトでは、ユーザー SYS、および SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。次の列値は、各データ・ファイルのディスク・アクセス回数を反映します。

PHYRDS            各データベース・ファイルからの読み込み回数。

PHYWRTS          各データベース・ファイルへの書き込み回数。

次の問合せによって、アプリケーションの実行中に、ある程度の期間にわたってこれらの値を監視します。

```
SELECT name, phylds, phywrts
FROM v$datafile df, v$filestat fs
WHERE df.file# = fs.file#;
```

また、この問合せは動的パフォーマンス・ビュー V\$DATAFILE から各データ・ファイルの名前も検索します。出力例は次のようになります。

NAME	PHYRDS	PHYWRTS
/oracle/ora70/dbs/ora_system.dbf	7679	2735
/oracle/ora70/dbs/ora_temp.dbf	32	546

V\$FILESTAT の PHYRDS 列と PHYWRTS 列は、SNMP を使用して取得できます。

単一ディスクに対する全体の I/O は、そのディスク上の Oracle インスタンスによって管理される、全データベース・ファイルの PHYRDS と PHYWRTS の合計になります。ディスクごとにこの値を決定してください。また、統計が収集された時間間隔で全体の I/O を割って、ディスクごとの I/O 発生率も求めてください。

## I/O 問題の解決

この章の残りの部分では、I/O の問題を解決するためのさまざまな手法を説明します。

- [I/O の分散によるディスク競合の低減](#)
- [ディスクのストライプ化](#)
- [動的な領域管理の回避](#)
- [ソートのチューニング](#)
- [チェックポイントのチューニング](#)
- [LGWR および DBWn の I/O のチューニング](#)
- [大規模プールの構成](#)

## I/O の分散によるディスク競合の低減

この項では、ディスク競合を低減する方法を説明します。

- [ディスク競合とは](#)
- [データ・ファイルと REDO ログ・ファイルの分離](#)
- [表データのストライプ化](#)
- [表と索引の分離](#)
- [Oracle と関係のないディスク I/O の低減](#)

## ディスク競合とは

複数のプロセスが同時に同じディスクにアクセスしようとするとき、ディスク競合が発生します。多くのディスクには、アクセス数と 1 秒あたりに転送できるデータ量の両方について制限があります。これらの制限に達すると、ディスクにアクセスするためにプロセスを待機させることが必要になります。

通常は、V\$FILESTAT ビューの統計およびオペレーティング・システムの機能を検討してください。ディスク性能の限界を判断するために、ハードウェアのマニュアルを調べてください。最大性能やそれに近い性能で作動しているディスクはディスク競合の候補です。たとえば、VMS や UNIX オペレーティング・システム上の多くのディスクでは、1 秒あたり 40 以上の I/O は過剰です。

負荷が過剰なディスクに対するアクティビティを削減するには、そのディスク上にあるアクセス頻度の激しい 1 つ以上のファイルを、それほど負荷のないディスクに移動します。すべてのディスクの I/O 量がだいたい同じになるまで、各ディスクにこの原則を適用してください。これを I/O の分散といいます。

## データ・ファイルと REDO ログ・ファイルの分離

Oracle プロセスは、絶えずデータ・ファイルと REDO ログ・ファイルにアクセスします。これらのファイルが同じディスク上に存在している場合、ディスク競合が発生する可能性があります。各データ・ファイルを別々のディスク上に配置してください。そうすると、複数のプロセスがディスク競合せずに同時に異なるファイルにアクセスできます。

REDO ログ・ファイルの各セットは、他のアクティビティがない別々のディスクに配置してください。REDO ログ・ファイルは、トランザクションがコミットされるとき、ログ・ライター・プロセス (LGWR) によって書き込まれます。REDO ログ・ファイル内の情報は順次書き込まれます。同じディスクに対する同時実行のアクティビティが存在しない場合、この順次書込みはさらに高速で行われる可能性があります。REDO ログ・ファイルに別々の専用ディスクを割り当てると、さらにチューニングしなくても通常は LGWR が円滑に実行されます。LGWR に関連するパフォーマンス上のボトルネックはめったにありません。LGWR のチューニングの詳細は、21-15 ページの「[REDO ログ・バッファ・ラッチの競合の検出](#)」を参照してください。

---

**注意：** REDO ログ・ファイルをミラー化する、すなわち各 REDO ログ・ファイルの複数のコピーを別々のディスク上に保持することで、LGWR が大幅に遅くはなりません。LGWR は、各ディスクに対して並列して書込みを行い、並列書込みの各部が完了するまで待機します。オペレーティング・システムが単一のディスク書込みを実行するために必要な時間は、変動することがあるため、コピーの数が増えると、並列書込みでの単一のディスク書込みにかかる時間が平均よりも長くなる傾向が増します。ただし、並列書込みが、最も長い単一のディスク書込みよりも長くなることはありません。また、並列した書込みに関連するオーバーヘッドがオペレーティング・システムで多少発生する場合があります。

---

専用ディスクを用意することと REDO ログ・ファイルをミラー化することは、重要な安全対策です。データファイルと REDO ログ・ファイルに専用のディスクを用意することによって、データファイルと REDO ログ・ファイルの両方を単一のディスク障害で失う可能性がないことが保証されます。REDO ログ・ファイルのミラー化によって、REDO ログファイルを単一のディスク障害で失う可能性はないことが保証されます。

## 表データのストライプ化

ストライプ化、すなわち大きな表のデータを別々のディスク上の別々のデータ・ファイルに分散させることも、競合の低減に役立ちます。この方針については、20-22 ページの「[ディスクのストライプ化](#)」で詳しく説明します。

## 表と索引の分離

頻繁に使用される表は、索引と分離する必要があります。一連のトランザクション中は、索引が最初に読み込まれてから表が読み込まれます。これらの I/O は順次に発生するので、表と索引を同じディスク上に格納しても競合は発生しません。

## Oracle と関係のないディスク I/O の低減

可能であれば、データベース・ファイルを含むディスクについて、Oracle と関連のない I/O を取り除いてください。この処置は、REDO ログ・ファイルへのアクセスを最適化する上で特に有効です。これによってディスク競合が減少するだけでなく、動的パフォーマンス表 V\$FILESTAT を通じて、そのようなディスク上のアクティビティをすべて監視することもできます。

## ディスクのストライプ化

この項では、次のトピックについて説明します。

- [ストライプ化の目的](#)
- [I/O のバランス化とストライプ化](#)
- [ディスクを手動でストライプ化する方法](#)
- [オペレーティング・システム・ソフトウェアでディスクをストライプ化する方法](#)
- [RAID でハードウェア・ストライプ化を行う方法](#)

## ストライプ化の目的

"ストライプ化" によって、大きな表のデータが小さな部分に分割され、これらの部分が別々のディスク上の別々のデータ・ファイルに格納されます。これによって、複数のプロセスがディスク競合なしで表の異なる部分に同時にアクセスできます。ストライプ化は、数多くの行を持つ表へのランダム・アクセスを最適化する上で特に有効です。ストライプ化は、手動で実行することも（下記参照）オペレーティング・システムのストライプ化ユーティリティを使って実行することもできます。

## I/O のバランス化とストライプ化

以前、ベンチマークのチューニング担当者は、使用可能な各デバイス上で I/O の負荷のバランスを均一にすることを懸命に試行していました。現在は、オペレーティング・システムによって、頻繁に使用されるコンテナ・ファイルを多数の物理デバイスにストライプ化する機能が提供されています。ただし、このような手法は、負荷の再分散によってなんらかの形態のキューが排除または削減される場合にのみ有効です。

I/O キューが存在する場合、または存在すると思われる場合は、使用可能なデバイス上での負荷分散は当然のチューニング・ステップです。多数の物理ドライブを使用可能な場合は、



2つの専用ドライブで REDO ログをとることを検討してください（2つである理由は、REDO ログはオペレーティング・システムによって、または Oracle REDO ログ・グループ機能を使って常にミラー化する必要があるからです）。REDO ログはシリアルに書き込まれるので、REDO ログ・アクティビティ専用のドライブでは通常はヘッドの移動はわずかで、す。このため、ログ書き込みのスピードが大幅に向上します。

アーカイブする場合は、LGWR および ARCH が同じ読み込み / 書き込みヘッドを競合しないように、別のディスクを使用することが効果的です。これは、ログを代替ドライブに配置することで行います。

ミラー化は、I/O ボトルネックの原因となる可能性もあります。各ミラーへの書き込みプロセスは、通常は並列して行われるので、ボトルネックの原因にはなりません。ただし、各ミラーが別々にストライプ化されている場合は、低速のミラー・メンバーが終了するまで I/O は完了しません。I/O の問題を回避するためには、接続先データベース（つまりコピー）で元データベースと同数のディスクを使用してストライプ化を行ってください。

たとえば、8 個のディスクに 160KB のデータをストライプ化し、データが 1 つのディスクにしかミラー化されていない場合は、データが 8 個のディスク上でどれだけ高速に処理されるかにかかわらず、160KB がミラー・ディスクに書き込まれるまで I/O は完了しません。したがって、データベースへの書き込みには 20.48 ミリ秒しかかかりませんが、ミラーへの書き込みには 137 ミリ秒を要します。

## ディスクを手動でストライプ化する方法

ディスクを手動でストライプ化するには、オブジェクトの記憶要件を I/O 要件と関連付ける必要があります。

1. 最初に、次の項目を調べて、オブジェクトのディスク記憶要件を評価します。

- オブジェクトのサイズ
- ディスクのサイズ

たとえば、オブジェクトが 5GB の Oracle 記憶領域を必要とする場合は、それを収容する 1 つの 5GB のディスクまたは 2 つの 4GB のディスクが必要です。一方、システムが 1GB または 2GB のディスクで構成されている場合は、オブジェクトはそれぞれ 5 個または 3 個のディスクを必要とすることがあります。

2. 20-2 ページの「[I/O 要件の分析](#)」で説明したアプリケーションの I/O 要件とこれを比較してください。記憶要件と I/O 要件の大きい方をとる必要があります。

たとえば、記憶要件が 5 つのディスク（それぞれ 1GB）であり、I/O 要件が 2 つのディスクである場合は、アプリケーションは大きい方の値である 5 ディスクを必要とします。

3. CREATE TABLESPACE 文で表領域を作成します。DATAFILE 句にデータ・ファイルを指定します。各ファイルは異なるディスク上に作成してください。

```
CREATE TABLESPACE stripedtabspace
  DATAFILE 'file_on_disk_1' SIZE 1GB,
```

```
'file_on_disk_2' SIZE 1GB,  
'file_on_disk_3' SIZE 1GB,  
'file_on_disk_4' SIZE 1GB,  
'file_on_disk_5' SIZE 1GB;
```

4. CREATE TABLE 文で表を作成します。TABLESPACE 句に新たに作成した表領域を指定します。

また、STORAGE 句に表のエクステントのサイズを指定します。別々のデータ・ファイルに各エクステントを格納します。表のエクステントは、オーバーヘッドを考慮して表領域内のデータ・ファイルより少し小さくしてください。たとえば、1GB (1024MB) のデータ・ファイルを準備するときは、表エクステントを 1023MB に設定できます。

```
CREATE TABLE stripedtab (  
  col_1  NUMBER(2),  
  col_2  VARCHAR2(10) )  
TABLESPACE stripedtabspace  
STORAGE ( INITIAL 1023MB NEXT 1023MB  
  MINEXTENTS 5 PCTINCREASE 0 );
```

(あるいは、DATAFILE 'size' SIZE 句を指定した ALTER TABLE ALLOCATE EXTENT 文を入力することで表をストライプ化することもできます。)

これらのステップによって、表 STRIPEDTAB が作成されます。STRIPEDTAB には、それぞれサイズが 1023MB の初期エクステントが 5 つあります。各エクステントは、CREATE TABLESPACE 文の DATAFILE 句に指定されたデータ・ファイルの 1 つを取り上げます。これらのファイルはすべて別々のディスク上に存在します。MINEXTENTS が 5 なので、これら 5 つのエクステントはすべて即時に割り当てられます。

**関連項目：** MINEXTENTS および他の記憶領域パラメータの詳細は、『Oracle8i SQL リファレンス』を参照してください。

# オペレーティング・システム・ソフトウェアでディスクをストライプ化する方法

手動でディスクをストライプ化する方法のかわりとして、LVM（論理ボリューム・マネージャ）などのオペレーティング・システムのストライプ化ソフトウェアを使用して、ディスクをストライプ化します。ストライプ化ソフトウェアでは、正しいストライプ・サイズを選択が考慮事項になります。これは、Oracle のブロック・サイズとディスク・アクセス方法によって異なります。

表 20-14 最小ストライプ・サイズ

ディスク・アクセス	最小ストライプ・サイズ
ランダム読み込みおよび書き込み	最小ストライプ・サイズは、Oracle ブロック・サイズの 2 倍です。
順次読み込み	最小ストライプ・サイズは、DB_FILE_MULTIBLOCK_READ_COUNT の値の 2 倍です。

ストライプ化では、データへの統一的なアクセスが前提とされています。ストライプ・サイズが大きすぎる場合は、1 つまたは少数のディスクでホット・スポットが発生する可能性があります。これは、ストライプ・サイズを小さくし、データをより多くのディスクに分散することで回避できます。

固定サイズの 100 行が 5 つのディスクに均一に分散され、各ディスクが 20 の順次行を含んでいる例を考えます。アプリケーションが行 35 ~ 55 へのアクセスだけを必要とする場合は、2 つのディスクだけですべての I/O を実行しなければなりません。この割合では、システムは目的のパフォーマンス・レベルを達成できません。

この問題は、行 35 ~ 55 をより多くのディスクに分散することで解決できます。現行の例では、1 ブロックあたりに 2 行とすれば、行 35 と 36 が同じディスク上に存在し、行 37 と行 38 は別のディスクに存在することになります。このアプローチをとると、データはすべてのディスクに分散され、I/O スループットが改善されます。

## RAID でハードウェア・ストライプ化を行う方法

RAID（Redundant arrays of inexpensive disk）は、障害回復機能に大きな利点をもたらします。また、RAID を使用すると、ストライプ化を非常に簡単にこなしますが、パフォーマンスにはあまり役立ちません。実際に、RAID は I/O オーバーヘッドに高いコストを課すことがあります。

RAID テクノロジーの機能を一部しか使用しないことによってパフォーマンスを改善できることもあります。または、単一コンポーネントの障害に対する RAID テクノロジーの回復力によって、パフォーマンスに関するコストを正当化できることもあります。

## 動的な領域管理の回避

表やロールバック・セグメントのようなオブジェクトを作成すると、データのために領域がデータベース内に割り当てられます。この領域をセグメントと呼びます。後続のデータベース操作によってデータ容量が増大し、割り当てられた領域を上回るようになると、Oracle はそのセグメントを拡張します。この場合、動的拡張によってパフォーマンスが低下します。

この項では、次のことについて説明します。

- [動的拡張の検出](#)
- [エクステントの割当て](#)
- [無制限のエクステントの評価](#)
- [複数のエクステントの評価](#)
- [ロールバック・セグメント内の動的領域管理の回避](#)
- [移行行と連鎖行の削減](#)
- [SQL.BSQ ファイルの修正](#)

## 動的拡張の検出

動的拡張によって、Oracle はユーザー・プロセスが発行した SQL 文に加えて、それらの SQL 文を実行するようになります。Oracle 自体がこれらの文を発行するため、この SQL 文を再帰的コールと呼びます。また、再帰的コールは以下のアクティビティによっても生成されます。

- データ・ディクショナリ・キャッシュ・ミス
- データベース・トリガーの起動
- データ定義言語の実行
- ストアド・プロシージャ、ファンクション、パッケージおよび無名 PL/SQL ブロック内の SQL 文の実行
- 参照整合性制約の施行

RECURSIVE CALLS 統計を、動的パフォーマンス・ビュー V\$SYSSTAT を介して調べます。デフォルトでは、ユーザー SYS、および SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。次の問合せを使用して、一定期間にわたってこの統計を監視してください。

```
SELECT name, value
       FROM v$sysstat
      WHERE NAME = 'recursive calls';
```

Oracle によって次のような結果が返されます。

NAME	VALUE
recursive calls	626681

アプリケーションの実行時に Oracle で過度の再帰的コールが発生する場合には、これらのコールが再帰的コールを生成する 1 つのアクティビティによるものであり、動的拡張によるものではないことを確認してください。これらの再帰的コールが動的拡張によるものと判断される場合、より大きなエクステントを割り当てることによって、この拡張を低減してください。

## エクステントの割当て

動的拡張を避けるには、以下の手順に従ってください。

1. オブジェクトの最大サイズを決定します。オブジェクトのスペース要件を見積もる式の詳細は、『Oracle8i 管理者ガイド』を参照してください。
2. オブジェクトを作成するときに、Oracle が全データを収容するのに十分大きなエクステントを割り当てるように、記憶領域パラメータ値を選びます。

エクステントを大きくすると、以下の理由でパフォーマンス上有利になる傾向があります。

- 単一エクステント内のブロックは連続しているため、1 つの大きなエクステントは、複数の小さなエクステントよりも連続性に優れています。Oracle は、多くの小さなエクステントを読み込むために必要となるより少ないマルチブロック読み込みで、ディスクから 1 つの大きなエクステントを読み込むことができます。
- エクステントが大きいほど、そのセグメントが拡張される可能性は低くなります。

ただし、大きなエクステントにはより多くの連続ブロックが必要となるため、Oracle がそれらを格納できる十分な連続領域を見つけることは難しいかもしれません。割り当てるエクステントを大きくしてその数を減らすか、または小さくしてその数を増やすかを判断するには、オブジェクトの成長と使用の計画を考慮に入れ、それぞれの利点と欠点を評価してください。

また、自動的にサイズ変更できるデータファイルも、動的拡張によって問題が発生する可能性があります。自動拡張の使用は避けてください。かわりに、システムが相対的に非アクティブであるときに、より多くの領域をデータ・ファイルに手動で割り当ててください。

## 無制限のエクステントの評価

オブジェクトはエクステントを無制限に持つことができますが、これは大量の小さなエクステントを許容できるという意味ではありません。パフォーマンスを最適化するためには、エクステントの数を削減します。

エクステント・マップは、特定のセグメントのすべてのエクステントをリストします。Oracle のブロックあたりのエクステントの数は、オペレーティング・システムのブロック・サイズとプラットフォームによって決まります。エクステントは Oracle 内部のデータ構造ですが、このデータ構造のサイズはプラットフォームに依存します。それに応じて、データ構造のサイズは 1 つのオペレーティング・システム・ブロックに Oracle が格納できるエクステントの数に影響します。一般に、この値は次のようになります。

表 20-15 ブロック・サイズとエクステントの最大数（標準値）

ブロック・サイズ (KB)	エクステントの最大数
2	121
4	255
8	504
16	1032
32	2070

最適なパフォーマンスを得るには、エクステント・マップを 1 回の I/O で読み込めなければなりません。エクステント・マップを入手するための全表走査に複数の I/O が必要な場合は、パフォーマンスが低下します。

ディクショナリ対応の表領域では動的拡張を行わないでください。ディクショナリにマップされた表領域では、エクステント数が 1,000 を超えないようにします。エクステント割当てがローカルの場合は、2,000 を超えるエクステントを使用しないでください。エクステントが多すぎると、表の削除や切捨て時にパフォーマンスが低下します。

ほとんどの状況で最適な選択は、AUTOEXTEND を使用可能にすることです。また、証明済みの値によって最適なパフォーマンスが得られることが確実な場合は、エクステントを割り当てるときにその値を使用することもできます。

## 複数のエクステントの評価

この項では、複数のエクステント使用のさまざまな影響を説明します。

- 大規模なセグメントは、ファイル・サイズやファイル・システムのサイズ制限により、1つのエクステントに入れることができません。セグメントが新しいエクステントを割り当てられるようになると、より高速で安価なディスクを利用できます。
- 全表走査が行われない表では、表のエクステントが1つか複数にかかわらず、問合せのパフォーマンスに違いはありません。
- 索引を使った検索のパフォーマンスは、索引のエクステントが1つであるか複数であるかの影響を受けません。
- 表、クラスタまたは一時セグメントで複数のエクステントを使用しても、マルチユーザー・システム上での全走査のパフォーマンスに影響はありません。
- 表、クラスタまたは一時セグメントで複数のエクステントを使用しても、エクステントが正しくサイズ設定されている場合、およびアプリケーションが過度の DDL 操作を避けるように設計されている場合は、単一ユーザー専用のバッチ処理システム上の全走査のパフォーマンスに実質的な影響はありません。
- エクステント・サイズが I/O サイズに適したものであれば、セグメント内に多数のエクステントを持つ場合のパフォーマンス・コストは最小化されます。
- ロールバック・セグメントでは、多数のエクステントの方が少数のエクステントよりも望ましいとされます。多数のエクステントを持つと、セグメント上で動的エクステント割当てを実行する再帰的 SQL コールの回数が減ります。

## ロールバック・セグメント内の動的領域管理の回避

ロールバック・セグメントのサイズが、パフォーマンスに影響を及ぼす可能性もあります。ロールバック・セグメントのサイズは、ロールバック・セグメントの記憶領域パラメータの値によって決まります。ロールバック・セグメントには、トランザクションのロールバック・エントリを保持するのに十分な大きさが必要です。他のオブジェクトと同様に、ロールバック・セグメントでの動的領域管理を避けるべきです。

SET TRANSACTION 文を使用して、次の項で提示される推奨事項に基づく適切なサイズのロールバック・セグメントをトランザクションに割り当ててください。トランザクションをロールバック・セグメントに明示的に割り当てないと、Oracle によってトランザクションが自動的にロールバック・セグメントに割り当てられます。

たとえば、次の文は、現行トランザクションをロールバック・セグメント OLTP\_13 に割り当てます。

```
SET TRANSACTION USE ROLLBACK SEGMENT oltp_13
```

---

**注意：** 同じアプリケーションの複数のコピーを同時に実行する場合は、すべてのコピーのトランザクションを同じロールバック・セグメントに割り当てないように注意してください。同じロールバック・セグメントに割り当てると、ロールバック・セグメントの競合につながります。

---

OPTIMAL 記憶領域パラメータに基づく、ロールバック・セグメントの縮小または動的割当て解除を監視してください。このパラメータの値の選択、ロールバック・セグメントの縮小の監視、OPTIMAL パラメータの調整の詳細は、『Oracle8i 管理者ガイド』を参照してください。

### 長い問合せの場合

長い問合せが同時実行で選択しているデータを修正するトランザクションには、大きなロールバック・セグメントを割り当ててください。そのような問合せは、修正されたデータの読み込み一貫バージョンを再構築するために、ロールバック・セグメントへのアクセスを必要とする可能性があります。ロールバック・セグメントには、問合せの実行中に、そのデータに対するすべてのロールバック・エントリを保持するのに十分な大きさが必要です。

### 長いトランザクションの場合

大量のデータを修正するトランザクションには、大きなロールバック・セグメントを割り当ててください。そのようなトランザクションでは大きなロールバック・エントリが生成されるため、大きなロールバック・セグメントによってパフォーマンスが改善されます。ロールバック・エントリがロールバック・セグメントに収まらない場合は、Oracle がセグメントを拡張します。動的拡張はパフォーマンスを低下させるため、可能なかぎり避けてください。

### OLTP トランザクションの場合

OLTP アプリケーションの特徴は、頻繁な同時実行のトランザクションです。これらのトランザクションはそれぞれが少量のデータを修正します。データに対して同時実行の問合せが行われない場合、OLTP トランザクションを小さなロールバック・セグメントに割り当ててください。ロールバック・セグメントを小さくすると、バッファ・キャッシュ内に格納されたままになることが多く、高速でアクセスできます。典型的な OLTP ロールバック・セグメントでは、エクステンツが 2 つあり、サイズはおよそ 10K です。最も完全に競合を避けるには、多くのロールバック・セグメントを作成し、各トランザクションをそれぞれ専用のロールバック・セグメントに割り当ててください。

## 移行行と連鎖行の削減

UPDATE 文が行のデータ量を増やし、行がそのデータ・ブロックに収まらなくなった場合、Oracle は行全体を保持するのに十分な空き領域を持つ別のブロックを見つけようとします。そのようなブロックが利用可能であれば、Oracle は新しいブロックへ行全体を移動します。これを行の移行といいます。行が大きすぎて利用可能なブロックに収まらない場合、Oracle



はその行を複数の断片に分割し、各断片を別々のブロックに格納します。これを行の連鎖といいいます。行は挿入時にも連鎖される可能性があります。

動的な領域管理、特に移行と連鎖はパフォーマンスに悪影響を及ぼします。

- 移行と連鎖の原因となる UPDATE 文のパフォーマンスが悪くなります。
- 移行行や連鎖行を選択する問合せは、より多くの I/O を実行する必要があります。

LIST CHAINED ROWS オプションを指定した ANALYZE 文を使用して、表やクラスタ内の移行行と連鎖行を識別します。この文は、それぞれの移行行と連鎖行に関する情報を収集し、この情報を指定された表に出力します。

サンプルの出力表 CHAINED\_ROWS の定義が、配布媒体上の使用可能な SQL スクリプトに収録されています。このスクリプトの一般的な名前は UTLCHAIN.SQL ですが、正確な名前と位置は使用しているプラットフォームに依存して異なる可能性があります。出力表は、CHAINED\_ROWS 表と同じ列名、データ型およびサイズでなければなりません。

移行行または連鎖行は、V\$SYSSTAT の TABLE FETCH CONTINUED ROW 列を調べることによって検出できます。PCTFREE を増やして移行行を回避してください。ブロック内に使用可能な空き領域を多く残しておく、行の拡張に対処できます。削除割合が高い表と索引を再編成すなわち再作成することもできます。

---

**注意：** PCTUSED の内容は PCTFREE の逆ということではありません。  
PCTUSED は領域管理を制御します。

---

**関連項目：** 詳細は、『Oracle8i 概要』を参照してください。

既存の表での移行行と連鎖行を低減するには、以下の手順に従ってください。

1. ANALYZE 文を使用して、移行行と連鎖行に関する情報を収集します。次に例を示します。

```
ANALYZE TABLE order_hist LIST CHAINED ROWS;
```

2. 出力表を問い合わせます。

```
SELECT *
FROM chained_rows
WHERE table_name = 'ORDER_HIST';
```

OWNER_NAME	TABLE_NAME	CLUST...	HEAD_ROWID	TIMESTAMP
-----	-----	-----	-----	-----
SCOTT	ORDER_HIST	...	AAAAluAAHAAAAAIAAA	04-MAR-96
SCOTT	ORDER_HIST	...	AAAAluAAHAAAAAIAAB	04-MAR-96
SCOTT	ORDER_HIST	...	AAAAluAAHAAAAAIAAC	04-MAR-96

出力では、移行または連鎖されたすべての行がリストされます。

3. 移行行や連鎖行が数多くあることが出力表に示された場合、次の手順に従って移行行を取り除くことができます。

- a. 移行行と連鎖行を保持するために既存の表と同じ列を持つ中間表を作成します。

```
CREATE TABLE int_order_hist
AS SELECT *
FROM order_hist
WHERE ROWID IN
(SELECT head_rowid
FROM chained_rows
WHERE table_name = 'ORDER_HIST');
```

- b. 既存の表から移行行と連鎖行を削除します。

```
DELETE FROM order_hist
WHERE ROWID IN
(SELECT head_rowid
FROM chained_rows
WHERE table_name = 'ORDER_HIST');
```

- c. 中間表の行を既存の表に挿入します。

```
INSERT INTO order_hist
SELECT *
FROM int_order_hist;
```

- d. 中間表を削除します。

```
DROP TABLE int_order_history;
```

4. 出力表からステップ 1 で収集した情報を削除します。

```
DELETE FROM chained_rows
WHERE table_name = 'ORDER_HIST';
```

5. 再び ANALYZE 文を使用して、出力表を問い合わせます。

6. 出力表にあるすべての行は連鎖しています。データ・ブロック・サイズを増やすことによってのみ、連鎖行を取り除くことができます。すべての状況で連鎖を避けることは不可能です。表に LONG 列または長い CHAR 列や VARCHAR2 列がある場合には、連鎖を避けられません。

移行行の検索はリソースを集中的に使用するので、UPDATE の対象となるすべての表は、更新されそうなブロック内で十分な領域を使用できるようにするために、分散空き領域セットを持つ必要があります。

## SQL.BSQ ファイルの修正

SQL.BSQ ファイルは、CREATE DATABASE 文が発行されたときに実行されます。このファイルには、Oracle Server を構成する表定義が含まれています。DBA が使用するビューは、これらの表を基にしています。ユーザーによる SQL.BSQ への修正を厳しく制限することをお勧めします。

- 必要であれば、次の記憶パラメータの値を増やせます。INITIAL、NEXT、MINEXTENTS、MAXEXTENTS、PCTINCREASE、FREELISTS、FREELIST GROUPS および OPTIMAL。
- PCTINCREASE を除き、記憶領域パラメータにデフォルトを下回る値を設定しないでください（MAXEXTENTS の値が大きい場合は、PCTINCREASE を小さな値または 0 にすることができます）。
- SQL.BSQ に対するその他の変更はサポートされません。特に、列の追加、削除または改名は行わないでください。

---

**注意：** 内部データ・ディクショナリ表は、リリースごとに追加、削除または変更される可能性があります。このため、ディクショナリが新しいリリースに移行されるときにユーザーによる修正が引き継がれることはありません。

---

**関連項目：** これらのパラメータの詳細は、『Oracle8i SQL リファレンス』を参照してください。

## ソートのチューニング

パフォーマンスとメモリー使用率の間にはトレード・オフがあります。最高のパフォーマンスを得るには、ほとんどのソートをメモリー内で行う必要があります。ディスクに書き込むソートはパフォーマンスに悪影響を与えます。ソート領域のサイズが大きすぎると、使用されるメモリーが多くなりすぎることがあります。ソート領域のサイズが小さすぎると、ソートをディスクに書き込まなければならず、上述したようにパフォーマンスが大幅に低下する可能性があります。

この項では、次のトピックについて説明します。

- [メモリーでのソート](#)
- [ディスクでのソート](#)
- [一時表領域の使用によるソート・パフォーマンスの最適化](#)
- [NOSORT の使用によるソートを行わない索引作成](#)
- [GROUP BY NOSORT](#)

## メモリーでのソート

デフォルトのソート領域のサイズは、多くのソートですべてのデータを保持するためには十分です。ただし、アプリケーションがソート領域に収まらないデータに対して大規模なソートを頻繁に実行する場合、そのソート領域のサイズを大きくしてください。数多くの行を操作するソートを実行する SQL 文が原因で、大規模なソートが発生する可能性があります。

**関連項目：** ソートを実行する SQL 文の詳細は、『Oracle8i 概要』を参照してください。

### 大規模ソートの認識

Oracle は、ソート・アクティビティを反映する統計を収集し、それらを動的パフォーマンス・ビュー V\$SYSSTAT に格納します。デフォルトでは、ユーザー SYS、および SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。以下の統計がソートの動作を反映します。

SORTS(MEMORY)	十分に小さいため、ディスク上の一時セグメントへの I/O を行わずにソート領域で完全に実行できるソートの数。
SORTS(DISK)	大きすぎるためにソート領域で完全に実行できずに、ディスク上の一時セグメントへの I/O を必要とするソートの数。

次の問合せを使用し、一定期間にわたってこれらの統計を監視します。

```
SELECT name, value
FROM v$sysstat
WHERE name IN ('sorts (memory)', 'sorts (disk)');
```

次に、この問合せの出力例を示します。

NAME	VALUE
-----	-----
sorts(memory)	965
sorts(disk)	8

V\$SYSSTAT 内の情報は、SNMP (シンプル・ネットワーク管理プロトコル) を介して取得することもできます。

### ディスクでのソートを回避するための SORT\_AREA\_SIZE の拡大

SORT\_AREA\_SIZE は、各ソートに使用するメモリーの最大容量を指定する、動的に変更可能な初期化パラメータです。膨大な数のソートが一時セグメントのディスク I/O を必要とする場合、アプリケーションのパフォーマンスは、ソート領域のサイズを大きくすることで向上することがあります。この場合には、SORT\_AREA\_SIZE の値を増やしてください。

このパラメータの最大値はオペレーティング・システムによって異なります。SORT\_AREA\_SIZE をどの程度大きくすることが有効かを判断する必要があります。

`SORT_AREA_SIZE` を十分に大きな値に設定していれば、（たとえば 10 ギガバイトの表をソートしているのではない限り）ほとんどのソートはディスクでは行われません。

**関連項目：**「[チェックポイントのチューニング](#)」の「[チェックポイントのチューニング](#)」および 26-20 ページの「[SORT\\_AREA\\_SIZE](#)」

## 大規模ソート領域のパフォーマンスの利点

上述したように、ソート領域のサイズを増やすとソートがディスクで行われる機会が減少します。このため、大きなソート領域がある場合は、ほとんどのソートは I/O なしで短時間に処理されます。

ソート操作がディスクに書き込まれる場合は、ソート・ランにおいて部分的にソート済みのデータが書き込まれます。ソート処理からすべてのデータを受け取ると、Oracle はランをマージして最終的なソート済み出力を生成します。すべてのランを一度にマージできるほどソート領域が大きい場合は、いくつかのマージ・パスにランが分かれてマージされます。ソート領域が大きい場合は、生成されるラン数は少なく、時間は長くなります。また、ソート領域が大きな場合は、1 つのマージ・パスでより多くのランをマージできます。

## 大規模ソート領域のパフォーマンスのトレードオフ

ソート領域のサイズを大きくすると、Oracle の各ソート・プロセスがより多くのメモリーを割り当ててようになります。この増大によって、プライベート SQL と PL/SQL 領域のためのメモリー量が減少します。さらに、オペレーティング・システムのメモリー割当てに影響が及び、ページングとスワッピングが発生する可能性もあります。ソート領域のサイズを大きくする前に、オペレーティング・システム上で大規模なソート領域を収容するのに十分な空きメモリーが利用できることを確認してください。

ソート領域のサイズを増やす場合は、`SORT_AREA_RETAINED_SIZE` パラメータの値を減らすことを考慮してください。このパラメータは、ソート・プロセスの一部またはすべてが完了したときに、ソート領域のサイズを削減できる下限を制御します。つまり、ソート処理がソート済みのデータをユーザーまたは問合せの次の部分へ送信し始めると、Oracle によってソート領域が削減されます。ソート領域の保持サイズが小さくなると、メモリーの使用は節減されますが、ディスク上の一時セグメントに対して読み書きを行うための I/O が増加します。

## ディスクでのソート

ディスク上でソートする場合は、ソートに使用する表領域の `PCTINCREASE` を 0 に設定してください。また、`INITIAL` と `NEXT` は同じサイズにする必要があります。これによって、ソートに使用する表領域の断片化を削減できます。これらのパラメータは `ALTER TABLE` の `STORAGE` オプションを使用して設定します

**関連項目：** `PCTINCREASE` の詳細は、『Oracle8i 概要』を参照してください。

## 一時表領域の使用によるソート・パフォーマンスの最適化

ソートのパフォーマンスを最適化するために、一時表領域でソートを実行します。一時表領域を作成するには、TEMPORARY キーワードを CREATE TABLESPACE 文または ALTER TABLESPACE 文に含めて使用します。

通常、ソートは、一時セグメントの割当てと割当て解除を行うために多数の領域割当て呼出しを必要とします。表領域を TEMPORARY と指定する場合、Oracle はソート操作を要求するインスタンスごとに 1 つのソート・セグメントをその表領域にキャッシュします。このスキームは、通常の領域割当てを回避するため、すべてをメモリー内で実行できない中規模のソートのパフォーマンスを大幅に向上させます。

表やロールバック・セグメントなどの永久オブジェクトを含む表領域では、TEMPORARY キーワードは使用できません。

**関連項目：** CREATE TABLESPACE 文と ALTER TABLESPACE 文の構文  
の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## 一時表領域のストライプ化によるソート・パフォーマンスの改善

できればオペレーティング・システムのストライプ化ツールを使用して、一時表領域を多数のディスクにストライプ化します。たとえば、2 つのディスクだけに一時表領域をストライプ化し、それぞれが 1 秒あたり最大 50 回の I/O が可能な場合は、1 秒あたり 100 回の I/O しか実行できません。この制限によって、ソート操作の時間が長くなることがあります。

この例では、一時表領域を 10 個のディスクにストライプ化すると、ソート処理を 5 倍の速にすることができます。これによって、1 秒あたり 500 回の I/O が可能になります。

## SORT\_MULTIBLOCK\_READ\_COUNT を使用するソート・パフォーマンスの改善

一時表領域を使用するもう 1 つのソート・パフォーマンスの改善方法は、パラメータ SORT\_MULTIBLOCK\_READ\_COUNT をチューニングすることです。一時セグメントに対しては、SORT\_MULTIBLOCK\_READ\_COUNT はパラメータ DB\_FILE\_MULTIBLOCK\_READ\_COUNT とほぼ同じ効果があります。

SORT\_MULTIBLOCK\_READ\_COUNT の値を増やすことで、各マージ・パスでディスクからメモリーによりソート・プロセスが大きなソート・ランの部分を読み込むようになります。また、これによって、ソート・プロセスが 1 つのマージ・パスでマージできるマージ幅（つまり実行数）を減らすことになります。この場合、マージ・パスの数が増えることがあります。

各マージ・パスではディスクでのソート・ランが新たに行われるため、マージ・パス数が増加するとソート時に実行される I/O の総数も増加します。

SORT\_MULTIBLOCK\_READ\_COUNT パラメータの増加によって得られる I/O スループットの向上と、実行される I/O 総数の増加のバランスがとれるようにしてください。

## NOSORT の使用によるソートを行わない索引作成

ソートが行われる原因の 1 つは索引の作成です。表に対して索引を作成すると、索引が作成される列の値に基づいて、その表のすべての行のソートが行われます。Oracle で、ソートが発生しないように索引を作成することもできます。表の中の行が昇順でロードされていると、ソートを実行せずに、索引をより高速に作成できます。

### NOSORT オプション

ソートが発生しないように索引を作成するには、索引付きの列値について昇順で表に行をロードします。ロードする前に行をソートするためのソート・ユーティリティが、オペレーティング・システムで用意されていることもあります。索引を作成するときには、CREATE INDEX 文の NOSORT オプションを使用してください。たとえば、次の CREATE INDEX 文は、EMP 表で行のソートが発生しないように、EMP 表の ENAME 列に対して索引 EMP\_INDEX を作成します。

```
CREATE INDEX emp_index  
ON emp(ename)  
NOSORT;
```

---

**注意：** CREATE INDEX 文に NOSORT を指定すると、PARALLEL (DEGREE *n*) が指定されている場合でも PARALLEL INDEX CREATE の使用が否定されます。

---

### NOSORT オプションの用途

データを事前にソートし、その順序でロードすることが、必ずしも表をロードする最も高速な方法ではありません。

- 複数 CPU のコンピュータを使用している場合、各プロセッサがデータの異なる部分をロードするように、複数のプロセッサをパラレルで使用してより高速にデータをロードできる可能性があります。並列処理を利用するには、まずソートを発生させずにデータをロードします。その後、NOSORT オプションなしで索引を作成してください。
- CPU が 1 つのコンピュータの場合、可能であればロード前にデータをソートします。その後、NOSORT オプションを使用して索引を作成してください。

## GROUP BY NOSORT

GROUP BY 操作を実行するとき、入力データがすでに整列して、各グループに属するすべての行がひとかたまりになっている場合、ソート操作を回避できます。この事例としては、グループ列と一致する行を索引から検索した場合や、ソート / マージ結合によって行が正しい順序で作成された場合が考えられます。同じ状況で ORDER BY ソートも回避できます。ソートが実行されない場合は、EXPLAIN PLAN 出力の中に GROUP BY NOSORT として示されます。

## チェックポイントのチューニング

チェックポイントは、Oracle が自動的に実行する操作です。この項では、次のトピックを説明します。

- [チェックポイントがパフォーマンスに与える影響](#)
- [チェックポイントの頻度の選択](#)
- [ファースト・スタート・チェックポイント](#)

## チェックポイントがパフォーマンスに与える影響

チェックポイントは次のパフォーマンスに影響を及ぼします。

- インスタンス回復時のパフォーマンス
- 実行時のパフォーマンス

頻繁なチェックポイントによって、インスタンス障害時のインスタンス回復時間が減少します。チェックポイントが比較的頻繁であれば、チェックポイント間にデータベースに対して行われる変更は比較的わずかなものです。この場合、インスタンス回復のためにロールフォワードされる変更も比較的わずかなものになります。

チェックポイントでは DBWn プロセスが I/O を実行するため、一時的に実行時のパフォーマンスが低下することがあります。ただし、チェックポイントに関連するオーバーヘッドは通常小さなものであり、Oracle がチェックポイント取得を実行している間のみパフォーマンスに影響を与えます。

## チェックポイントの頻度の選択

パフォーマンス上の関心事に基づいてチェックポイントの頻度を選択します。回復時よりも実行時のパフォーマンスの効率を重要視する場合、選択するチェックポイントの頻度は低くします。最適な実行時パフォーマンスの達成よりも、迅速なインスタンス回復を重要視する場合は、選択するチェックポイントの頻度は高くします。

チェックポイントは REDO ログのメンテナンス上必須であるため、完全にチェックポイントをなくすことはできません。ただし、以下のパラメータを設定することによって、チェックポイントの頻度を最低限まで下げることができます。

- 最大の REDO ログ・ファイルのサイズよりも大きな値（物理ブロック・サイズの倍数）を LOG\_CHECKPOINT\_INTERVAL 初期化パラメータに設定します。
- LOG\_CHECKPOINT\_TIMEOUT 初期化パラメータの値をゼロに設定します。この値によって、時間ベースのチェックポイントがなくなります。

また、「[ファースト・スタート・チェックポイント](#)」で説明するように I/O 操作の回数に制限を設けることでパフォーマンスを制御できます。



さらに、これらのパラメータの設定の他に、ログ・ファイルのサイズについても考慮してください。小さなログ・ファイルのメンテナンスによって、チェックポイント・アクティビティが増加し、パフォーマンスが低下することがあります。

---

**注意：** Oracle は現在の読み込みブロックでチェックポイント取得を実行します。反対に、ソート・ブロックや一貫読み込みブロックではチェックポイント取得が実行されません。

---

**関連項目：** チェックポイントの詳細は、『Oracle8i 概要』を参照してください。

## ファースト・スタート・チェックポイント

ファースト・スタート・チェックポイントという機能によって、使用済みバッファ数が制限されるので、インスタンスの回復に必要な時間も制限されます。インスタンス回復を実行するのに処理しなければならない I/O 操作の回数が非常に多い場合は、パフォーマンスが悪影響を受けます。パラメータ FAST\_START\_IO\_TARGET に適切な値を設定することによってこのオーバーヘッドを制御できます。

FAST\_START\_IO\_TARGET によって、インスタンス回復のために Oracle が許可すべき I/O 操作の回数が制限されます。回復に必要な操作の回数がこの制限を超えると、Oracle によって使用済みバッファがディスクに書き込まれます。これは、インスタンス回復に必要な I/O 操作の回数が FAST\_START\_IO\_TARGET に設定された制限に減少するまで行われます。

回復に必要な操作回数によってそのプロセスにかかる時間がわかるので、インスタンス回復の時間を制御できます。チェックポイントのこの機能を使用禁止にするには、FAST\_START\_IO\_TARGET をゼロ (0) に設定します。

---

**注意：** ファースト・スタート・チェックポイントが使用可能なのは、Oracle8i Enterprise Edition のみです。

---

## LGWR および DBWn の I/O のチューニング

この項では、ログ・ライターおよびデータベース・ライターのバックグラウンド・プロセスの I/O のチューニング方法について説明します。

- [LGWR の I/O のチューニング](#)
- [DBWn の I/O のチューニング](#)

### LGWR の I/O のチューニング

多くの INSERT または LONG/RAW アクティビティを実行するアプリケーションは、LGWR の I/O をチューニングすることで利益を受けることがあります。各 I/O の書き込みサイズは、初期化パラメータ LOG\_BUFFER によって設定されるログ・バッファのサイズに依存します。そのため、正しいログ・バッファ・サイズを選択することが重要です。LGWR は、バッファの 3 分の 1 が満たされた場合、または COMMIT などのフォアグラウンド・プロセスによって通知されたときに書き込みを開始します。大きすぎるログ・バッファ・サイズは、書き込みの遅延を発生させることがあります。小さすぎるログ・バッファも効率が悪く、小規模な I/O を頻繁に行う結果となります。

I/O の平均サイズがかなり大きい場合は、ログ・ファイルがボトルネックとなることがあります。REDO ログ・ファイルをストライプ化し、複数のディスク上で並列に使用すると、この問題が避けられます。この状況では手動のストライプ化が不可能なので、オペレーティング・システムのストライプ化ツールを使う必要があります。

ストライプ・サイズも重要です。適切な値は、バッファをストライプ化するディスクの数で平均再実行 I/O サイズを除算することによって計算できます。

大量のデータ・ファイルがある場合、または大容量の OLTP 環境にある場合は、CHECKPOINT\_PROCESS 初期化パラメータを常に TRUE に設定する必要があります。この設定によって CKPT プロセスが使用可能になり、チェックポイント中に LGWR が REDO 情報の書き込みを続け、一方で CKPT プロセスがデータファイル・ヘッダーを更新することが保証されます。

### 増分チェックポイント

増分チェックポイントを使用すると、クラッシュ時およびインスタンス回復時のパフォーマンスが向上しますが、メディア回復のパフォーマンスは向上しません。増分チェックポイントでは、クラッシュ / インスタンス回復が開始する必要がある REDO スレッド（ログ）内の位置が記録されます。このログ位置は、バッファ・キャッシュ内の最も古い使用済みバッファによって判断されます。増分チェックポイント情報は、通常処理の際に定期的にメンテナンスされ、オーバーヘッドはごく小さいかまったく発生しません。

インスタンス回復の時間は、回復プロセスがディスクから読み込む必要があるデータ・ブロック数によって最も大きな影響を受けます。パラメータ DB\_BLOCK\_MAX\_DIRTY\_TARGET を使用して回復処理の時間を制御できます。このパラメータを使用すると、インスタンス回復プロセスが回復時にディスクから読み込む必要があるブロック数を制限できます。

DB\_BLOCK\_MAX\_DIRTY\_TARGET に適切な値を設定するには、まず、ディスクから 1 ブロックを読み込むのにかかる時間を判断してください。次に、希望するインスタンス回復時間をこの値で割ります。たとえば、1 ブロックを読み込むのに 10 ミリ秒かかり、回復プロセスを 30 秒以内に納めたい場合は、DB\_BLOCK\_MAX\_DIRTY\_TARGET の値を 3000 に設定します。DB\_BLOCK\_MAX\_DIRTY\_TARGET を低い値に設定することによってインスタンス回復時間は短縮されますが、その代償として通常処理時の書き込み数が増加します。

このパラメータを小さな値に設定すると、Oracle はより多くのバッファをディスクに書き込む必要があるため、通常処理時のオーバーヘッドが高くなります。また、この値を小さくすれば、回復対象のブロック数が少なくなるため回復のパフォーマンスは向上します。また、DB\_BLOCK\_MAX\_DIRTY\_TARGET を使用すると、インスタンス回復時に読み込まれるブロック数を制限でき、回復処理時間に影響を与えることもできます。

増分チェックポイント情報は、Oracle によって自動的にメンテナンスされます。ユーザー指定チェックポイントなど他のチェックポイントには影響を与えません。つまり、増分チェックポイントは、インスタンスで発生する他のチェックポイントとは関係なく発生します。

増分チェックポイントは、単一インスタンスでも複数インスタンス環境でも回復に好影響をもたらします。

**関連項目：**『Oracle8i 概要』、『Oracle8i リファレンス・マニュアル』および『Oracle8i バックアップおよびリカバリ・ガイド』

## DBWn の I/O のチューニング

この項では、DBW の I/O をチューニングする場合の次の問題について説明します。

- [複数のデータベース・ライター \(DBWn\) プロセス](#)
- [内部書き込みバッチ・サイズ](#)
- [単一バッファ・プールを持つ LRU ラッチ](#)
- [複数のバッファ・プールを持つ LRU ラッチ](#)

### 複数のデータベース・ライター (DBWn) プロセス

DB\_WRITER\_PROCESSES 初期化パラメータを使用すると、複数のデータベース・ライター・プロセス (DBW0 から DBW9 まで) を作成できます。これらは、NUMA マシンや、多数の CPU を備えた SMP システムのようなハイエンド・システムでは有効な場合があります。これらのバックグラウンド・プロセスは I/O サーバー・プロセス (DBWR\_IO\_SLAVES によって設定される) と同じではありません。I/O サーバー・プロセスはインスタンス障害がなくても停止する可能性があります。同じシステムで I/O サーバー・プロセスと複数の DBWn プロセスを同時に実行することはできません。

## 内部書込みバッチ・サイズ

データベース・ライター（DBW<sub>n</sub>）プロセスは、3つの値（A、BまたはC）のうち最も小さいものに設定される内部書込みバッチ・サイズを使用します。

- 値 A は次のように計算します。

$$\frac{DB\_FILES * DB\_FILE\_SIMULTANEOUS\_WRITES}{2} = \text{値 A}$$

- 値 B はポート固有の制限です（プラットフォーム固有の Oracle マニュアルを参照してください）。
- 値 C は、DB\_BLOCK\_BUFFERS の 4 分の 1 の値です。

内部書込みバッチ・サイズの設定が大きすぎると、応答時間が不均一になります。

最善の結果を得るために、上記の計算式の値 A を計算する元になるパラメータ値を変更することで、内部書込みバッチ・サイズに影響を及ぼすことができます。次のアプローチに従ってください。

1. 書き込む必要のあるファイルと、これらのファイルが存在するディスクの数を判断します。
2. これらのディスクに対して実行できる I/O の回数を判断します。
3. トランザクションで必要とされる書込み回数を判断します。
4. この割合に十分に対応できるディスクがあることを確認します。

## 単一バッファ・プールを持つ LRU ラッチ

複数のデータベース・ライター（DBW<sub>n</sub>）プロセスとただ1つのバッファ・プールがある場合、バッファ・キャッシュは LRU（最低使用頻度）ラッチごとにプロセス間で分割されます。各 LRU ラッチは1つの LRU リストに対応します。

DB\_BLOCK\_LRU\_LATCHES パラメータのデフォルト値はシステムの CPU 数です。この値を調整するときは、CPU 数と等しくするか、その倍数にします。各 DBW<sub>n</sub> プロセスが同一数の LRU リストを持ち、それらのプロセスの負荷を等しくするのが目的です。

たとえば、2つのデータベース・ライター・プロセスがあり、4つの LRU リスト（つまり4つのラッチ）がある場合、DBW<sub>n</sub> プロセスはラウンドロビン法でラッチを取得します。DBW0 がラッチ 1 を獲得し、DBW1 がラッチ 2 を獲得し、次に DBW2 がラッチ 3 を獲得し、DBW3 がラッチ 4 を獲得します。同様に、使用しているシステムに8つの CPU と3つの DBW<sub>n</sub> プロセスがある場合は9つのラッチを持つ必要があります。

## 複数のバッファ・プールを持つ LRU ラッチ

ただし、複数のバッファ・プールと複数のデータベース・ライター（DBWn）プロセスを使用している場合、各プール（DEFAULT、KEEP および RECYCLE）のラッチ数はプロセス数と同じかその倍数にしてください。これをお薦めするのは、各 DBWn プロセスの負荷を等しくするためです

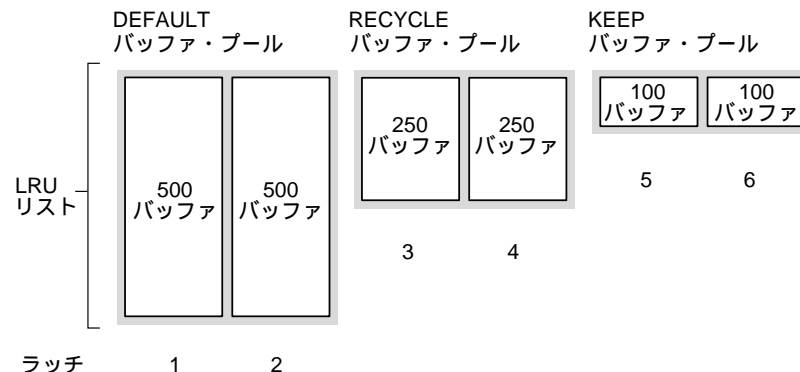
---

**注意：** 複数のバッファ・プールがあるときは、各バッファ・プールが連続範囲の LRU ラッチを持ちます。

---

3 つの DBWn プロセスと、3 つのバッファ・プールそれぞれに対して 2 つのラッチ（合計 6 つのラッチ）がある図 20-6 の例について考えます。各バッファ・プールは、ラウンドロビン法でラッチを取得します。

図 20-6 複数のバッファ・プールを持つ LRU ラッチ：例 1



DEFAULT バッファ・プールには、各 LRU リストに対して 500 のバッファがあります。  
 RECYCLE バッファ・プールには、各 LRU リストに対して 250 のバッファがあります。  
 KEEP バッファ・プールには、各 LRU に対して 100 のバッファがあります。

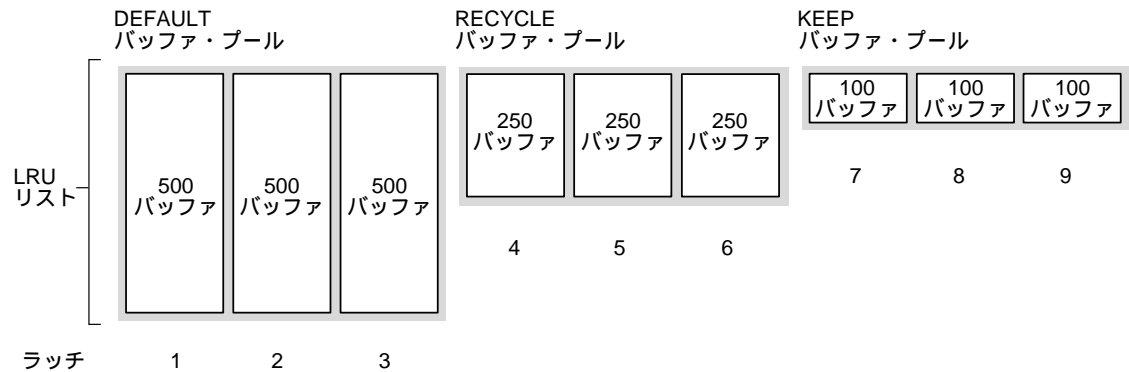
DBW0 はラッチ 1（500）とラッチ 4（250）を獲得し、合計 750 になります。  
 DBW1 はラッチ 2（500）とラッチ 6（100）を獲得し、合計 600 になります。  
 DBW2 はラッチ 3（250）とラッチ 5（100）を獲得し、合計 350 になります。

このように、各 DBWn プロセスにかかる負荷は異なり、パフォーマンスも影響を受けます。ただし、各プールに 3 つのラッチがある場合は、DBWn プロセスの負荷は等しくなり、パフォーマンスは最適化されます。

バッファ・プールが異なると、ブロック置換率も異なります。通常、ブロックは KEEP プール内で修正されることはほとんどなく、RECYCLE プール内では頻繁に修正されます。つまり、RECYCLE プールからは、KEEP プールからよりも頻繁にブロックを書き出す必要があります。結果として、あるプールの 100 のバッファを所有するのは、別のプールの 100 のバッファを所有するのとは違います。完全に負荷を平衡させるには、各 DBWn プロセスが、各種類のバッファ・プールから同一数の LRU リストを持つ必要があります。

システムが適切に構成されていれば、3 つの DBWn プロセスと 9 つのラッチを持ち、各バッファ・プールに 3 つのラッチがあるでしょう。

図 20-7 複数のバッファ・プールを持つ LRU ラッチ：例 2



DEFAULT バッファ・プールには、各 LRU リストに対して 500 のバッファがあります。  
RECYCLE バッファ・プールには、各 LRU リストに対して 250 のバッファがあります。  
KEEP バッファ・プールには、各 LRU リストに対して 100 のバッファがあります。

DBW0 はラッチ 1 ( 500 ) とラッチ 4 ( 250 ) とラッチ 7 ( 100 ) を獲得し、合計 850 になります。  
DBW1 はラッチ 2 ( 500 ) とラッチ 5 ( 250 ) とラッチ 8 ( 100 ) を獲得し、合計 850 になります。  
DBW2 はラッチ 3 ( 500 ) とラッチ 6 ( 250 ) とラッチ 9 ( 100 ) を獲得し、合計 850 になります。

## バックアップおよび復元操作のチューニング

バックアップと復元のチューニングの主要な目的は、ディスクと記憶デバイス間に適切なデータのフローを作成することです。バックアップと復元操作のチューニングでは、次の作業を完了することが必要です。

- ボトルネックの原因の発見
- バックアップ操作を監視するための固定ビューの使用
- バックアップのスループットの改善

### ボトルネックの原因の発見

通常は、バックアップと復元操作は次の 3 つの段階で実行します。

- 入力（ディスクまたはテープ）の読み込み
- ブロックの評価、および入力から出力バッファへのコピーによるデータ処理
- テープまたはディスクへの出力の書き込み

すべての段階で時間の長さが同じということはありません。このため、3 つのうち最も時間がかかる段階がボトルネックであるといえます。

### I/O のタイプについて

Oracle のバックアップと復元では、ディスクとテープという 2 つのタイプの I/O が使用されます。バックアップを実行するとき、入力ファイルはディスク I/O を使用して読み込まれ、出力バックアップ・ファイルはディスクまたはテープ I/O を使用して書き込まれます。復元を実行するときは、これらの役割が逆になります。ディスク I/O とテープ I/O はどちらも同期でも非同期でも使用できます。つまり、それぞれは互いに独立しています。

### 同期 I/O 率と非同期 I/O 率の測定

同期 I/O を使用すると、デバイスは一度に 1 回の I/O しか実行しないので、バックアップ・ジョブが必要とする時間を容易に判断できます。非同期 I/O を使用すると、次の理由で 1 秒あたりのバイト数を測定することが同期 I/O よりも難しくなります。

- 非同期処理は一度に複数のタスクが発生することを意味します。
- Oracle の I/O は、各 I/O 要求がいつ完了するかを判断するために割込みメカニズムではなくポーリングを使用します。オペレーティング・システムによって I/O の完了がバックアップまたは復元プロセスにすぐに通知されないため、各 I/O の時間を判断できません。

次の項では、バックアップのボトルネックを判別するために V\$BACKUP\_SYNC\_IO ビューおよび V\$BACKUP\_ASYNC\_IO ビューを使用する方法を説明します。

**関連項目：** これらのビューの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## 同期 I/O でのボトルネックの判別

同期 I/O では、すべての同期 I/O がプロセスにとってのボトルネックであるので、特定のボトルネックを識別するのは困難です。同期 I/O をチューニングする唯一の方法は、デバイスの最大スループット率と 1 秒あたりのバイト数を比較することです。1 秒あたりのバイト数がデバイス指定の値よりも小さい場合は、バックアップおよび復元のその部分のチューニングを検討してください。I/O 率を調べるには、

V\$BACKUP\_SYNC\_IO.DISCRETE\_BYTES\_PER\_SECOND 列を使用します。

## 非同期 I/O でのボトルネックの判別

LONG\_WAITS と SHORT\_WAITS を合せたものが IO\_COUNT の大半を占める場合は、V\$BACKUP\_SYNC\_IO および V\$BACKUP\_ASYNC\_IO で示されるファイルがおそらくボトルネックになっています。一部のプラットフォームの非同期 I/O の実現方法では、I/O 完了の非ブロック化ポーリングを実行するときコール元に I/O の完了を待機させる場合があります。この動作はプラットフォームによって異なりますが、V\$BACKUP\_ASYNC\_IO ビューに " 短期 " と " 長期 " 両方の合計待機時間が示されます。

" 長期 " 待機は、バックアップ / 復元プロセスが、オペレーティング・システムに I/O が完了するまで待機するように指示した回数です。" 短期 " 待機は、バックアップ / 復元プロセスがオペレーティング・システムのコールに I/O 完了を非ブロック化モードでポーリングさせた回数です。どちらの待機の場合も、オペレーティング・システムは迅速に応答する必要があります。

SHORT\_WAIT\_TIME\_TOTAL 列が LONG\_WAIT\_TIME\_TOTAL 列と同じかそれより大きい場合は、" 非ブロック化 " I/O ポーリングを実行するときに、使用しているプラットフォームは I/O 完了に対しておそらくブロックしています。この場合、SHORT\_WAIT\_TIME\_TOTAL はこのファイルの本当の I/O 時間を表します。SHORT\_WAIT\_TIME\_TOTAL がこのファイルの合計時間に比べて低い場合は、遅延の原因はプロセス・スワッピングなどの他の要因だと考えられます。可能な場合は、オペレーティング・システムをチューニングして、I/O 待機時間が LONG\_WAIT\_TIME\_TOTAL 列に現れるようにしてください。

## バックアップ操作を監視するための固定ビューの使用

V\$BACKUP\_SYNC\_IO および V\$BACKUP\_ASYNC\_IO を使用して、バックアップまたは復元でのボトルネックの原因を判別し、バックアップ・ジョブの進捗を判別します。

バックアップを実行するプロセス（または一部プラットフォームでの " スレッド "）に対して I/O が同期であるとき、V\$BACKUP\_SYNC\_IO に行が含まれます。I/O が非同期の場合は、V\$BACKUP\_ASYNC\_IO に行が含まれます。非同期 I/O は、I/O プロセスによって行なわれるか、基礎となるオペレーティング・システムでサポートされているために実行されます。



**V\$BACKUP\_SYNC\_IO および V\$BACKUP\_ASYNC\_IO で共通の列**

表 20-16 に、V\$BACKUP\_SYNC\_IO ビューと V\$BACKUP\_ASYNC\_IO ビューで共通な列とその説明を示します。

**表 20-16 V\$BACKUP\_SYNC\_IO および V\$BACKUP\_ASYNC\_IO の共通の列**

列	説明
SID	バックアップまたは復元を行うセッションの Oracle SID。
SERIAL	バックアップまたは復元を行う SID の使用状況カウンタ。
USE_COUNT	さまざまなバックアップ・セットの行を識別するために使用できるカウンタ。V\$BACKUP_SYNC_IO または V\$BACKUP_ASYNC_IO で一連の行が新たに作成されるたびに、それらの行は直前の行を上回る USE_COUNT を持ちます。USE_COUNT は、各バックアップまたは復元操作によって使用されるすべての行について同一です。
DEVICE_TYPE	ファイルが存在するデバイスの種類（通常は DISK または SBT_TAPE）。
TYPE	INPUT: 読み込まれるファイル。 OUTPUT: 書き込まれるファイル。 AGGREGATE: この行は、バックアップまたは復元に関係するすべての DISK ファイルの合計 I/O 回数を表します。
STATUS	NOT STARTED: このファイルはまだオープンされていません。 IN PROGRESS: このファイルは現在読み込みまたは書き込み中です。 FINISHED: このファイルの処理は完了しました。
FILENAME	読み込みまたは書き込み対象のバックアップ・ファイルの名前。
SET_COUNT	読み込みまたは書き込み対象のバックアップ・セットの SET_COUNT。
SET_STAMP	読み込みまたは書き込み対象のバックアップ・セットの SET_STAMP。
BUFFER_SIZE	このファイルの読み込み / 書き込みに使用されるバッファのサイズ（バイト単位）。
BUFFER_COUNT	このファイルの読み込み / 書き込みに使用されるバッファ数。
TOTAL_BYTES	明らかな場合は、このファイルについて読み込みまたは書き込みが行われた合計バイト数。不明の場合、この列は NULL になります。
OPEN_TIME	このファイルがオープンされた時刻。TYPE = 'AGGREGATE' の場合、集合内の最初のファイルがオープンされた時刻になります。

表 20-16 V\$BACKUP\_SYNC\_IO および V\$BACKUP\_ASYNC\_IO の共通の列

列	説明
CLOSE_TIME	このファイルがクローズされた時刻。TYPE = 'AGGREGATE' の場合、集合内の最後のファイルがクローズされた時刻になります。
ELAPSED_TIME	ファイルがオープンしていた時間（100 分の 1 秒単位）。
MAXOPENFILES	同時にオープンしている DISK ファイル数。この値は、TYPE = 'AGGREGATE' の行にのみ表示されます。
BYTES	これまでに読み込みまたは書き込みが行われたバイト数。
EFFECTIVE_BYTES_PER_SEC	バックアップ時にこのデバイスで達成した I/O 率。読み込みまたは書き込みのバイト数を経過時間で割った値です。この値は、ボトルネックを引き起こしているバックアップ・システムのコンポーネントについてのみ意味があります。このコンポーネントがボトルネックの原因ではない場合は、実際は、この列で測定された値はシステム内の他のより遅いコンポーネントの速度を反映しています。
IO_COUNT	このファイルに対して実行された I/O 回数。各要求は 1 つのバッファ（BUFFER_SIZE のサイズ）の読み込みまたは書き込みです。

V\$BACKUP\_SYNC\_IO 固有の列

表 20-17 に、V\$BACKUP\_SYNC\_IO ビュー固有の列を示します。

表 20-17 V\$BACKUP\_SYNC\_IO 固有の列

列	説明
IO_TIME_TOTAL	このファイルの I/O を実行するのに要した合計時間（100 分の 1 秒単位）。
IO_TIME_MAX	1 つの I/O 要求にかかった最大時間。
DISCRETE_BYTES_PER_SEC	このファイルの平均転送率。これは、個々の I/O 要求の開始時と終了時に行われた測定に基づきます。この値はこのデバイスの実際の速度を反映しているはずですが。

## V\$BACKUP\_ASYNC\_IO 固有の列

表 20-18 に、V\$BACKUP\_ASYNC\_IO ビュー固有の列を示します。

表 20-18 V\$BACKUP\_ASYNC\_IO 固有の列

列	説明
READY	バッファの使用準備が即座に完了した非同期要求の数。
SHORT_WAITS	バッファがただちに使用可能にはならなかったが、I/O 完了の非ブロック化ポーリングを行った後でバッファが使用可能になった回数。非ブロック化待機の時間が計測される理由は、一部の "非同期 I/O" の実現方法では、要求が非ブロック化要求であると推測されるときでも I/O の完了を待機することがあるためです。
SHORT_WAIT_TIME_TOTAL	I/O 完了の非ブロック化ポーリングにかかった合計時間（100 分の 1 秒単位）。
SHORT_WAIT_TIME_MAX	I/O 完了の非ブロック化ポーリングにかかった最長時間（100 分の 1 秒単位）。
LONG_WAITS	バッファがただちに使用可能にならずに、I/O の完了に対するブロック待機を発行した場合のみ使用可能になった回数。
LONG_WAIT_TIME_TOTAL	I/O 完了のブロック待機にかかった合計時間（100 分の 1 秒単位）。
LONG_WAIT_TIME_MAX	I/O 完了のブロック待機にかかった最長時間（100 分の 1 秒単位）。

## バックアップのスループットの改善

最適にチューニングされたバックアップでは、テープ・コンポーネントだけがボトルネックの原因になるはずですが、テープとそのデバイスを "ストリーミング" の状態にする、つまり常に回転させておく必要があります。テープがストリーミング状態でない場合は、テープへのデータ・フローが適切とはいえません。

この項では、バックアップのスループットを改善することによってストリーミングを保つための次の項目を説明します。

- データ転送率に影響する要因について
- 同期 I/O の場合のテープのストリーミングについての判別
- 非同期 I/O の場合のテープのストリーミングについての判別
- テープのストリーミングを可能にするスループットの増大

データ転送率に影響する要因について

ホストのデータ送信によってテープがストリーミング状態に保たれる速度は、次の要因によって異なります。

- テープ・デバイスのロー・キャパシティ
- 圧縮

テープ・デバイスのロー・キャパシティは、テープをストリーミング状態に保つために必要な最小のデータ容量です。

圧縮は、テープ・ハードウェアまたはメディア管理ソフトウェアのどちらかによってインプリメントされます。圧縮を使用しない場合は、テープ・デバイスのロー・キャパシティによってテープがストリーミング状態に保たれます。圧縮を使用する場合、テープをストリーミング状態にするために送る必要があるデータ容量は、圧縮係数をデバイスのロー・キャパシティに掛けたものです。圧縮係数は、データの型によって異なります。

同期 I/O の場合のテープのストリーミングについての判別

I/O が同期の場合にテープがストリーミング状態かどうかを判別するには、V\$BACKUP\_SYNC\_IO ビューの EFFECTIVE\_BYTES\_PER\_SECOND 列を問い合わせます。

EFFECTIVE_BYTES_PER_SECOND の値	状態
ハードウェアのロー・キャパシティより少ない値	テープはストリーミング状態ではありません。
ハードウェアのロー・キャパシティより多い値	データの圧縮率によってテープがストリーミング状態になっている可能性があります。

非同期 I/O の場合のテープのストリーミングについての判別

I/O が非同期の場合、LONG\_WAITS と SHORT\_WAITS を合せたものが I/O 回数の大半を占める場合はテープはストリーミング状態です。SHORT\_WAIT\_TIME\_TOTAL 列で示される時間が LONG\_WAIT\_TIME\_TOTAL 列と同じか上回る場合は、SHORT\_WAITS の方に重点を置いてください。

テープのストリーミングを可能にするスループットの増大

テープがストリーミング状態でない場合、それに対応する基本方針はテープの 1 秒あたりのバイト数を上げることです。ディスクから読み込む必要があるブロック数と、アクセスする必要があるディスク数に応じて、この方針を修正してください。

**複数ディスクへの I/O の分散** BACKUP 文の DISKRATIO パラメータを使用してバックアップの I/O を複数のボリュームに分散し、複数の同時データファイルをバックアップするとき RMAN がファイル読み込みを分散するために使用するディスク・ドライブ数を指定します。たとえば、システムに 10 個のディスクがあり、そのディスクでは 10 バイト / 秒でデータが供給され、テープ・ドライブをストリーミング状態に保つには 50 バイト / 秒が必要であるとして、この場合、DISKRATIO を 5 に設定して、バックアップの負荷を 5 個のディスクに分散します。

DISKRATIO を設定するときは、テープの状態をストリーミングに保つのに必要最低限のディスクに I/O を分散します。それ以上に設定すると、1 つのファイルを復元するためにかかる時間が長くなり、パフォーマンスの利益が得られません。DISKRATIO を指定しないが FILESPERSET を指定する場合、DISKRATIO はデフォルトで FILESPERSET になることに注意してください。どちらも指定しない場合、DISKRATIO のデフォルトは 4 です。

**空ファイルまたはほとんど変更されていないファイルのバックアップ** 大半が空のファイルの全体バックアップを実行するとき、またはごく少数のブロックだけが変更された場合に増分バックアップを実行するときは、テープがストリーミング状態を保つのに十分な速度でテープにデータを供給できないことがあります。

この場合、次のようにして最適なパフォーマンスを達成してください。

- Recovery Manager の SET LIMIT CHANNEL 文の MAXOPENFILES パラメータへの最大限の値の設定
- 非同期ディスク I/O

後者は、1 つのファイルのデータを入力バッファに入れる一方で他のファイルのデータを処理する非同期先読みを利用します。

**関連項目：** RMAN SET 文の詳細は、『Oracle8i バックアップおよびリカバリ・ガイド』を参照してください。

**満杯のファイルのバックアップ** ほぼ満杯のファイルの全体バックアップを実行する場合に、テープの状態がストリーミングでないときは、[表 20-19](#) に示すいくつかの方法でパフォーマンスを改善できます。

表 20-19 スループット・パフォーマンスの改善方法

方法	結果
BACKUP_TAPE_IO_SLAVES を設定	複数のプロセスを発生させて非同期テープ I/O をシミュレートし、バックアップまたは復元操作の負荷を分割します。このパラメータを設定しないと、テープ・レイヤーへのすべての I/O は同期になります。このパラメータを設定すると LARGE_POOL_SIZE も設定されます。
LARGE_POOL_SIZE を設定	<p>BACKUP_TAPE_IO_SLAVES を設定すると、テープ I/O のバッファを共有メモリから割り当てて、バッファを 2 つのプロセスで共有できるようにする必要があります。I/O スレーブのための共有バッファを取得しようとすると、Oracle では次のことが行われます。</p> <ul style="list-style-type: none"> <li>■ LARGE_POOL_SIZE が設定されると、Oracle は大規模プールからメモリを取得しようとします。この値の大きさが十分でない場合は、Oracle はバッファを共有プールから取得しようとはしません。</li> <li>■ LARGE_POOL_SIZE が設定されないと、Oracle は共有プールからメモリを取得しようとします。</li> <li>■ Oracle が十分なメモリを取得できない場合は、ローカル・プロセス・メモリから I/O バッファ・メモリを取得し、メッセージを alert.log ファイルに書き込んで、このバックアップでは同期 I/O が使用されることを通知します。</li> </ul>
DB_FILE_DIRECT_IO_COUNT を増加	<p>RMAN がディスク I/O でより大きなバッファを使用するようにします。バックアップや復元のディスク I/O で使用されるデフォルトのバッファ・サイズは、<math>DB\_FILE\_DIRECT\_IO\_COUNT * DB\_BLOCK\_SIZE</math> です。</p> <p>DB_FILE_DIRECT_IO_COUNT のデフォルト値は 64 なので、DB_BLOCK_SIZE が 2048 の場合、バッファ・サイズは 128KB になります。一部のプラットフォームでは、最も効果的な I/O バッファ・サイズは 128KB を超える場合があります。</p>
RMAN のパラメータ MAXOPENFILES または FILESPERSET が小さすぎないことを確認	<p>RMAN が一度に処理できるファイル数を増やします。デフォルトのバッファ・サイズを使用すると、同時にオープンされるファイルは、それぞれ 512KB のプロセス・メモリ（I/O プロセスが使用されている場合は SGA 大規模プール・メモリ）をバッファのために使用します。同時ファイルの数は、テープをストリーミング状態に保つのに十分な数だけにしてください。</p> <p>未使用ブロックの圧縮がテープ・ドライブに送られるディスク・データの容量に大きな影響を与えるため、何度も試して適切な数を求める必要があります。テープ・ドライブがディスク・ドライブよりも遅い場合は、MAXOPENFILES の値として 1 で十分なはずですが。</p>

## 大規模プールの構成

大規模プールを任意に構成して、Oracle が別々のプールから大規模メモリー割当てを要求できるようにすることができます。これによって、同じメモリーに対する他のサブシステムとの競合を避けることができます。

Oracle がマルチスレッド・サーバーのセッション・メモリーに割り当てる共有プール・メモリーが多いほど、共有 SQL キャッシュが利用できる共有プール・メモリーが減少します。共有メモリーの別の領域からセッション・メモリーを割り当てると、Oracle は、主に共有 SQL のキャッシングのために共有プールを使えるので、共有 SQL キャッシュの減少によるパフォーマンス・オーバーヘッドは発生しません。

I/O サーバー・プロセスと、バックアップおよびリストア操作では、Oracle は数百 K バイトの単位でバッファを割り当てます。共有プールでこの要求を満たせなくても、大規模プールでは満たすことが可能です。大規模プールには LRU リストがありません。Oracle は大規模プールからメモリーを除去しません。

LARGE\_POOL\_SIZE パラメータを使用して、大規模プールを構成します。どのプール（共有プールまたは大規模プール）にオブジェクト用のメモリーが存在するかを確認するには、V\$SGASTAT で列 POOL を参照します。

**関連項目：** 大規模プールの詳細は、『Oracle8i 概要』を参照してください。初期化パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。





---

## リソースの競合のチューニング

複数のプロセスが同時に同じリソースにアクセスしようとする、競合が発生します。競合が発生すると、いくつかのプロセスは各種データベース構造体へのアクセスを待機しなければなりません。この章で説明するトピックは、次のとおりです。

- 競合の問題の理解
- 競合の問題の検出
- 競合の問題の解決
- ロールバック・セグメントの競合の低減
- マルチスレッド・サーバー・プロセスの競合の低減
- パラレル実行サーバーの競合の低減
- REDO ログ・バッファ・ラッチの競合の低減
- LRU ラッチの競合の低減
- 空きリスト競合の低減

### 競合の問題の理解

リソースの競合の問題の症状は、VSSYSTEM\_EVENT で見ることができます。このビューは、ラッチの競合、バッファの競合および I/O の競合など、パフォーマンスに悪影響を与える可能性のあるシステムのさまざまな問題を明らかにします。このビューに示されるのは問題の症状でしかなく、実際の原因ではないことに注意してください。

たとえば、VSSYSTEM\_EVENT を参照することで、多くのバッファ・ビジー待機が発生していると気づくことがあります。おそらく、多数のプロセスを同じブロックに挿入しようとするときに、各プロセスが他のプロセスの挿入を待機してからでないと挿入できないことが原因です。問題となっているオブジェクトに空きリストを導入することで解決できる可能性があります。

バッファ・ピジー待機は、ラッチ空き待機の原因となることもあります。ラッチ空き待機のほとんどはキャッシュ・バッファのハッシュ連鎖ラッチのミスによって発生したものであり、やはり同じブロックへ挿入しようとしたときの副作用です。SPIN\_COUNTを増やしてラッチ空き待機（症状）を低減するのではなく、オブジェクトを変更して複数のプロセスが空きブロックに挿入できるようにする必要があります。このアプローチによって、競合が効果的に低減されます。

**関連項目：** さまざまな Oracle の機能で使用するリソースの詳細は、『Oracle8i 管理者ガイド』を参照してください。

## 競合の問題の検出

VSRESOURCE\_LIMIT ビューでは、いくつかのシステム・リソースについて、現行のグローバル・リソース使用率および最大のグローバル・リソース使用率の情報を提供します。この情報は、リソース制限を制御するパラメータに指定する値を決定する際の参考になります。

システムにアイドル時間がある場合は、V\$SYSTEM\_EVENT を検査することから調査を開始してください。平均待機時間が最も高いイベントを調べ、それぞれに適切なアクションを実行します。たとえば、空きラッチ待機の数が多い場合は、V\$LATCH を見てどのラッチに問題があるかを確認します。

バッファ・ピジー待機が過度の場合は、V\$WAITSTAT を見て、待機件数が最も多く待機時間が最も長いブロック・タイプを確認します。V\$SESSION\_WAIT でキャッシュ・バッファ待機を見ると、オブジェクトのファイルおよびブロック数をデコードできます。

この章の残りの部分では、一般的な競合の問題を説明します。競合のさまざまな形態は症状であり、次の 2 つのいずれかを変更することで修正できることに注意してください。

- アプリケーションの変更
- Oracle の変更

パフォーマンスの制約を克服する方法は、アプリケーションを変更する以外にないことがあります。

## 競合の問題の解決

この章の残りの部分では、さまざまな種類の競合について検討し、問題の解決方法を説明します。競合は、ロールバック・セグメント、マルチスレッド・サーバー・プロセス、パラレル実行サーバー、REDO ログ・バッファ・ラッチ、LRU ラッチまたは空きリストに対するものがあります。

## ロールバック・セグメントの競合の低減

この項では、ロールバック・セグメントの競合を低減する方法について説明します。次の問題を取り上げます。

- [ロールバック・セグメントの競合の識別](#)
- [ロールバック・セグメントの作成](#)

## ロールバック・セグメントの競合の識別

ロールバック・セグメントの競合は、ロールバック・セグメント・ブロックを含むバッファの競合によって反映されます。動的パフォーマンス表 V\$WAITSTAT を検査することによって、ロールバック・セグメントの競合がパフォーマンスに悪影響を与えているかどうかを判断できます。

V\$WAITSTAT は、ブロックの競合を反映する統計を収録しています。デフォルトでは、ユーザー SYS、および SYSTEM のような、SELECT ANY TABLE システム権限を持っているユーザーだけがこの表を利用できます。これらの統計は、次のようなブロックの異なるクラスの競合を反映します。

SYSTEM UNDO HEADER	SYSTEM ロールバック・セグメントのヘッダー・ブロックを含んでいるバッファの待機回数。
SYSTEM UNDO BLOCK	ヘッダー・ブロック以外の SYSTEM ロールバック・セグメントのブロックを含んでいるバッファの待機回数。
UNDO HEADER	SYSTEM ロールバック・セグメント以外のロールバック・セグメントのヘッダー・ブロックを含んでいるバッファの待機回数。
UNDO BLOCK	SYSTEM ロールバック・セグメント以外のロールバック・セグメントのヘッダー・ブロック以外のブロックを含んでいるバッファの待機回数。

次の問合せによって、アプリケーションの実行中に、ある程度の期間にわたってこれらの統計を監視します。

```
SELECT CLASS, COUNT
FROM V$WAITSTAT
WHERE CLASS IN ('SYSTEM UNDO HEADER', 'SYSTEM UNDO BLOCK',
                'UNDO HEADER', 'UNDO BLOCK');
```

次にこの問合せの結果例を示します。

CLASS	COUNT
-----	-----
SYSTEM UNDO HEADER	2089
SYSTEM UNDO BLOCK	633
UNDO HEADER	1235
UNDO BLOCK	942

全体のデータ要求回数とブロックの各クラスの待機回数を、同じ期間にわたって比較します。次の問合せによって、ある期間にわたるデータ要求の総数を監視することができます。

```
SELECT SUM(VALUE)
FROM V$SYSSTAT
WHERE NAME IN ('DB BLOCK GETS', 'CONSISTENT GETS');
```

次に、この問合せの出力例を示します。

```
SUM(VALUE)
-----
929530
```

V\$SYSSTAT 内の情報は、SNMP を介して取得することもできます。

ブロックの待機回数が全体の要求回数の 1% を上回る場合は、さらにロールバック・セグメントを作成して競合を低減することを検討してください。

## ロールバック・セグメントの作成

ロールバック・セグメント・ブロックが含まれているバッファの競合を低減するには、さらにロールバック・セグメントを作成します。表 21-1 に、データベース上の同時トランザクションの数に基づいてロールバック・セグメントの割当て数を選択する際の一般的なガイドラインを示します。これらのガイドラインは、ほとんどのアプリケーションの組み合わせに対して適切なものです。

表 21-1 ロールバック・セグメントの数の選択

現行トランザクション数 ( <i>n</i> )	ロールバック・セグメントの推奨数
$n < 16$	4
$16 \leq n < 32$	8
$32 \leq n$	$n/4$

## マルチスレッド・サーバー・プロセスの競合の低減

この項では、Oracle マルチスレッド・サーバー・アーキテクチャで使用されるプロセスの競合を低減する方法を説明します。

- [ディスパッチャ・プロセスの競合の低減](#)
- [共有サーバー・プロセスの競合の低減](#)

## ディスパッチャ固有のビューを使用する競合の識別

次のビューによってディスパッチャのパフォーマンス統計が提供されます。

- V\$DISPATCHER
- V\$DISPATCHER\_RATE

V\$DISPATCHER は、ディスパッチャ・プロセスの一般的な情報を提供します。  
V\$DISPATCHER\_RATE ビューはディスパッチャ処理の統計を提供します。

**関連項目：** これらのビューの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

### V\$DISPATCHER\_RATE 統計の分析

V\$DISPATCHER\_RATE ビューは、いくつかのカテゴリについてディスパッチャ統計の現在、平均および最大の値を含みます。接頭辞 "CUR\_" が付いている統計は、現在のセッションについての統計です。接頭辞 "AVG\_" が付いている統計は、収集期間が開始してからの統計の平均値です。接頭辞 "MAX\_" が付いている統計は、統計の収集が開始してからのこれらのカテゴリでの最大値です。

ディスパッチャのパフォーマンスを調べるには、V\$DISPATCHER\_RATE ビューを問い合せて、最大値と現在の値を比較します。現在のシステム・スループットが適切な応答時間を提供し、このビューの現在の値が平均値に近く最大値を下回る場合、MTS 環境は最適にチューニングされていると考えられます。

現在の値と平均値が最大値を大きく下回る場合は、ディスパッチャ数の削減を検討してください。反対に、現在の値と平均値が最大値に近い場合は、ディスパッチャを追加する必要が

あるかもしれません。システム使用率が低い期間と高い期間の両方で V\$DISPATCHER\_RATE 統計を調べてみることを経験則としてお勧めします。MTS の負荷パターンを識別した後で、それに従ってパラメータを調整します。

必要であれば、システムのストレス・テストを実行し、定期的に V\$DISPATCHER\_RATE 統計をポーリングすることによって処理負荷をシミュレートすることもできます。これらの統計の正しい解釈方法は、プラットフォームごとに異なります。アプリケーションの種類が変わると、V\$DISPATCHER\_RATE に記録される統計値も大幅に異なる可能性があります。

## ディスパッチャ・プロセスの競合の低減

この項では、ディスパッチャ・プロセスの競合を識別する方法、ディスパッチャ・プロセスを追加する方法および接続プーリングを使用可能にする方法を説明します。

### ディスパッチャ・プロセスの競合の識別

ディスパッチャ・プロセスの競合は、次のいずれかの徴候によって示されます。

- 既存のディスパッチャ・プロセスの使用率が高いこと。
- ディスパッチャ・プロセスの応答キューにおける応答の待機時間が確実に増加していること。

**ディスパッチャ・プロセスの使用率の検査** V\$DISPATCHER には、ディスパッチャ・プロセスのアクティビティを反映した統計が含まれます。デフォルトでは、ユーザー SYS、および SYSTEM のような、SELECT ANY TABLE システム権限を持っているユーザーのみがこのビューを利用できます。次の列がディスパッチャ・プロセスの使用率を反映します。

IDLE	ディスパッチャ・プロセスのアイドル時間（100 分の 1 秒単位）を示します。
BUSY	ディスパッチャ・プロセスの使用（ビジー）時間（100 分の 1 秒単位）を示します。

次の問合せによって、アプリケーションの実行中に、ある程度の期間にわたってこれらの統計を監視します。

```
SELECT NETWORK "PROTOCOL",  
       SUM(BUSY) / ( SUM(BUSY) + SUM(IDLE) ) "TOTAL BUSY RATE"  
FROM V$DISPATCHER  
GROUP BY NETWORK;
```

この問合せは、各プロトコルのディスパッチャ・プロセスに対する全体の使用率、すなわち各プロトコルのディスパッチャ・プロセスが使用中になっている時間の割合を戻します。次にこの問合せの結果例を示します。

PROTOCOL	TOTAL BUSY RATE
-----	-----
DECNET	.004589828
TCP	.029111042

この結果から、以下のようなことがわかります。

- DECnet ディスパッチャ・プロセスは時間の 0.5% 近くが使用中です。
- TCP ディスパッチャ・プロセスは時間の 3% 近くが使用中です。

データベースが 1 日に 8 時間しか使われない場合は、有効作業時間によって統計を正規化する必要があります。単純にインスタンス開始時からの統計を参照することはできません。かわりに、ピーク・ワークロード時の統計を記録してください。特定のプロトコルのディスパッチャ・プロセスが、ピーク・ワークロード時の 50% を上回る割合で使用中心となる場合は、ディスパッチャ・プロセスを追加することによって、そのプロトコルを使用して Oracle に接続するユーザーのためにパフォーマンスを改善できます。

**ディスパッチャ・プロセス応答キューの待機時間の検査** VSQUEUE には、ディスパッチャ・プロセスの応答キュー・アクティビティを反映した統計が含まれます。デフォルトでは、ユーザー SYS、および SYSTEM のような、SELECT ANY TABLE システム権限を持っているユーザーのみがこの表を利用できます。これらの列には、キューにおける応答の待機時間が示されます。

WAIT                    キューに存在していた全応答についての待機時間の合計（100 分の 1 秒単位）

TOTALQ                キューに存在していた応答の総数。

次の問合せを使用して、アプリケーションの実行中にこれらの統計をときどき監視してください。

```
SELECT NETWORK          "PROTOCOL",
       DECODE( SUM(TOTALQ), 0, 'NO RESPONSES',
              SUM(WAIT)/SUM(TOTALQ) || ' HUNDREDTHS OF SECONDS')
       "AVERAGE WAIT TIME PER RESPONSE"
FROM VSQUEUE Q, V$DISPATCHER D
WHERE Q.TYPE = 'DISPATCHER'
      AND Q.PADDR = D.PADDR
GROUP BY NETWORK;
```

この問合せは、ユーザー・プロセスへの経路を定めるために、応答がそれぞれの応答キューでディスパッチャ・プロセスを待機した平均時間を 100 分の 1 秒単位で戻します。この問合せでは、ネットワーク・プロトコルによって VSQUEUE 表の行をグループ化するために、V\$DISPATCHER 表を使用します。また、この問合せは、キューに応答が存在しなかったプロトコルを識別するために、DECODE 構文を使用しています。以下にこの問合せの結果例を示します。

```

PROTOCOL  AVERAGE WAIT TIME PER RESPONSE
-----  -----
DECNET     .1739130 HUNDREDTHS OF SECONDS
TCP        NO RESPONSES

```

この結果から、DECNET ディスパッチャ・プロセスのキューにおいて応答は平均 0.17 待機し (単位は 100 分の 1 秒)、TCP ディスパッチャ・プロセスのキューには応答が存在しなかったことがわかります。

アプリケーションの実行時に特定のネットワーク・プロトコルの平均待機時間が確実に増加し続ける場合は、ディスパッチャ・プロセスを追加することによって、そのプロトコルを使用して Oracle に接続するユーザー・プロセスのパフォーマンスを改善できる可能性があります。

## ディスパッチャ・プロセスの追加

Oracle の実行時にディスパッチャ・プロセスを追加するには、ALTER SYSTEM コマンドの SET オプションを使用して MTS\_DISPATCHERS パラメータの値を増やします。

全プロトコルにわたるディスパッチャ・プロセスの総数は、初期化パラメータ MTS\_MAX\_DISPATCHERS の値によって制限されます。ディスパッチャ・プロセスを追加する前に、この値を増やす必要があるかもしれません。このパラメータのデフォルト値は 5 です。最大値は使用しているオペレーティング・システムに応じて異なります。

**関連項目：** ディスパッチャ・プロセスの詳細は、『Oracle8i 管理者ガイド』を参照してください。

## 接続プーリングを使用可能にする

システム・ロードが増加し、ディスパッチャ・スループットが最大限になった場合は、すぐにディスパッチャを追加することが必ずしも適切とはかぎりません。そのかわりに、多重化によってより多くのユーザーをサポートできるようにディスパッチャを構成することを検討してください。このためには、Connection Manager ソフトウェアをインストールする必要があります。

MTS\_DISPATCHERS を使用すると、それぞれのディスパッチャにさまざまな属性を設定できます。Oracle は、位置に依存せず大文字と小文字を区別しない方式で属性を指定できる、名前 = 値構文をサポートしています。次に例を示します。

```
MTS_DISPATCHERS = "(PROTOCOL=TCP)(DISPATCHERS=3)(POOL=ON)(TICK=1)"
```

オプション属性 POOL (または POO) は、Net8 接続プーリング機能を使用可能にするために使用します。TICK は、ネットワーク・ティックのサイズ (秒数) です。TICK のデフォルトは 15 秒です。

**関連項目：** MTS\_DISPATCHER パラメータとそのオプションの詳細は、『Oracle8i SQL リファレンス』および『Oracle8i Net8 管理者ガイド』を参照してください。



## 接続多重化を使用可能にする

多重化では、Connection Manager (CM) プロセスを使用して、複数ユーザーの個別ディスパッチャ・プロセスに対する接続の確立とメンテナンスを行います。たとえば、いくつかのユーザー・プロセスが 1 つの CM プロセスを経由して 1 つのディスパッチャ・プロセスに接続できます。

CM は、ユーザーとディスパッチャの通信を単一接続経路で管理します。CM プロセス経由でディスパッチャにリンクしているユーザー・プロセスがアイドル状態のとき、接続を必要とするユーザーが 0 のこともあれば、1 つまたは少数の場合もあります。このように、ユーザー対ディスパッチャ・プロセスの接続を最大限に利用できるため、多重化は有効な方法です。

CM プロセスは、ユーザーおよびディスパッチャ・プロセスと同一のマシンに存在しても、他のマシンに存在してもかまいません。いずれの場合も、使用しているプラットフォームのネットワーク・プロトコルが、ユーザー・プロセス、CM およびディスパッチャ・プロセスの通信リンクとして使用されます。

各ディスパッチャの接続数の制限はプラットフォームによって異なります。1 ディスパッチャあたりの接続の割当ては、250 以下にすることをオススメします。ほとんどの 32 ビット・マシンでは、接続数が 250 を上回るとパフォーマンスが低下する傾向があります。

## 共有サーバー・プロセスの競合の低減

この項では、共有サーバー・プロセスの競合を識別する方法と共有サーバー・プロセスの最大数を増やす方法を説明します。

### 共有サーバー・プロセスの競合の識別

要求キューにおいて待機時間が一貫して増加する場合、それは共有サーバー・プロセスの競合を示します。待機時間のデータを調べるには、動的パフォーマンス・ビュー VSQUEUE を使用します。このビューには、共有サーバー・プロセスの要求キューのアクティビティを示す統計が含まれます。デフォルトでは、ユーザー SYS、および SYSTEM のような、SELECT ANY TABLE システム権限を持っているユーザーだけがこのビューを利用できます。次の列がキューにおける要求の待機時間を示します。

WAIT	キューに存在していた全要求についての待機時間の合計 (100 分の 1 秒単位)
TOTALQ	キューに存在していた要求の総数

アプリケーションの実行時に次の SQL 文を発行して、この統計を数回監視してください。

```
SELECT DECODE( totalq, 0, 'No Requests',
              WAIT/TOTALQ || ' HUNDREDTHS OF SECONDS')
      "AVERAGE WAIT TIME PER REQUESTS"
FROM VSQUEUE
WHERE TYPE = 'COMMON';
```

この問合せは、次のような計算結果を戻します。

```
AVERAGE WAIT TIME PER REQUEST
-----
.090909 HUNDREDTHS OF SECONDS
```

この結果から、要求は処理される前に要求キューにおいて平均 0.09 待機したことがわかります (単位は 100 分の 1 秒)。

また、次の問合せを発行することによって、現在実行中の共有サーバー・プロセスの数が判断できます。

```
SELECT COUNT(*) "Shared Server Processes"
FROM v$shared_servers
WHERE status != 'QUIT';
```

以下にこの問合せの結果例を示します。

```
SHARED SERVER PROCESSES
-----
10
```

MTS でリソース競合を検出した場合は、まず、MTS を介して接続している各ユーザーに対して、LARGE\_POOL\_SIZE パラメータによって 5KB が割り当てられていることを確認してください。パフォーマンスが改善されない場合は、共有サーバー・プロセス競合を低減するためにさらにリソースを作成するとよいでしょう。このためには、次に説明するサーバー・プロセスのオプション・パラメータを変更します。

**関連項目：** LARGE\_POOL\_SIZE の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## MTS プロセスの設定と変更

この項では、マルチスレッド・サーバー・アーキテクチャのプロセスに影響するオプション・パラメータの設定方法を説明します。また、この項では、パフォーマンスをチューニングするためにこれらのパラメータを変更する方法とタイミングについても説明します。

この項で説明する静的初期化パラメータは次のとおりです。

- MTS\_MAX\_DISPATCHERS
- MTS\_MAX\_SERVERS

この項では、次の初期化 / セッション・パラメータについても説明します。

- MTS\_DISPATCHERS
- MTS\_SERVERS

**ディスパッチャおよびサーバーの静的パラメータ** init.ora パラメータである MTS\_MAX\_DISPATCHERS および MTS\_MAX\_SERVERS は、1 インスタンスで実行するディスパッチャ数およびサーバー数の上限を定義します。これらのパラメータは静的であり、データベースの実行中には変更できません。ディスパッチャ・プロセスとサーバー・プロセスは必要に応じていくつでも作成できますが、プロセスの合計数はホスト・オペレーティング・システムの実行プロセス数の制限を超えることはできません。

---

**注意：** MTS\_MAX\_DISPATCHERS を設定すると、組み合わされたすべてのネットワーク・プロトコルでのディスパッチャ数が制限されます。

---

**ディスパッチャおよびサーバーの動的パラメータ** また、MTS\_DISPATCHERS パラメータおよび MTS\_SERVERS パラメータを設定することで、ディスパッチャおよびサーバー数の開始値を定義することもできます。システムが起動した後で、ALTER SYSTEM コマンドの SET オプションを使用して、これらのパラメータの値を動的にリセットしディスパッチャおよびサーバーの数を変更できます。静的パラメータによって設定された制限を超えてこれらのパラメータに値を入力すると、静的パラメータの値が使用されます。

**静的および動的な MTS パラメータの依存性** MTS\_MAX\_SERVERS のデフォルト値は、MTS\_SERVERS の値に依存します。MTS\_SERVERS が 10 以下の場合、MTS\_MAX\_SERVERS のデフォルトは 20 になります。MTS\_SERVERS が 10 よりも大きい場合、MTS\_MAX\_SERVERS のデフォルトは MTS\_SERVERS の値の 2 倍になります。

**MTS アーキテクチャの自己調整機能** データベースが開始すると、MTS\_SERVERS の数は共有サーバー数と等しくなります。この数は下限としても使用されます。そのため、共有サーバー数がこの最小値を下回ることはありません。処理中に、既存プロセスの負荷が急激に増加した場合は、MTS\_MAX\_SERVERS によって定義された制限内で共有サーバー・プロセスが自動的に追加されます。そのため、明示的に共有サーバー・プロセスを追加することによって、パフォーマンスが向上することはありません。ただし、リソースの特定の問題を解決するために、システムを調整する必要があるかもしれません。

共有サーバー・プロセスの数が初期化パラメータ MTS\_MAX\_SERVERS によって設定されている制限に到達し、要求キューにおける平均待機時間が依然として好ましくない場合、MTS\_MAX\_SERVERS の値を増やすことによってパフォーマンスを改善できることもあります。

リソースの需要が予想を上回る場合、Oracle によって自動的に共有サーバー・プロセスを追加するか、MTS\_SERVERS の値を変更して共有プロセスを追加することができます。INIT.ORA ファイルでこのパラメータの値を変更するか、ALTER SYSTEM コマンドの MTS\_SERVERS パラメータを使用して値を変更します。この制限を使用して試行し、共有サーバーを監視して、このパラメータの理想的な設定を判別してください。

## 最適なディスパッチャ数および共有サーバー・プロセス数の判別

前述したように、MTS\_SERVERS は、インスタンス起動時にアクティブ化する共有サーバー・プロセス数を決定します。MTS\_SERVERS のデフォルト設定は 1 であり、これは MTS\_DISPATCHERS がアクティブ化されときのデフォルト設定でもあります。

ディスパッチャおよび共有サーバーの最適な数を判別するには、通常データベースにアクセスするユーザー数と各ユーザーが要求する処理量を考慮します。ユーザーおよび処理の負荷が時間とともに変化することも考慮に入れます。たとえば、顧客サービス・システムの負荷は、日中のピークの OLTP 指向の使用状況と夜間の DSS 指向の使用状況では大きく変化します。会計システムにかかる負荷が月の中旬と月末では大きく異なるように、より長期的な時間経過にともなうシステム使用状況の変化も予測できます。

一定の期間に各ユーザーが比較的小数の要求しか行わない場合、関連する各ユーザー・プロセスはほとんどの時間アイドルになります。この場合、10 ~ 20 のユーザーが 1 つの共有サーバー・プロセスを使用できます。各ユーザーが大量の処理を要求する場合は、ユーザー・プロセスに対するサーバー・プロセスの割合を高く設定してください。

最初は、共有サーバー・プロセスを少なめに割り当てるのが最適です。必要であれば追加の共有サーバーが自動的に開始し、アイドル時間が長すぎる場合は自動的に割当てが解除されます。ただし、初期サーバーはアイドルであっても常に割り当てられたままになります。

サーバーの初期数の設定が大きすぎると、システムで不要なオーバーヘッドが発生する場合があります。共有サーバー・プロセスの初期数について試行し、共有サーバーを監視して、通常のデータベース・アクティビティにとって理想的なシステム・パフォーマンスを達成してください。

**ディスパッチャ・プロセスの最大数の見積り** MTS\_MAX\_DISPATCHERS および MTS\_DISPATCHERS に対して、最大同時セッション数を 1 ディスパッチャあたりの接続数で割った値と等しい値、またはそれよりも大きい値を使用します。ほとんどのシステムでは、値が 1 ディスパッチャあたり接続数 250 であればパフォーマンスは良好です。

**同時 MTS 使用による MTS 追加使用の禁止** 前述したように、ALTER SYSTEM コマンドの SET オプションを使用して、アクティブな共有サーバー・プロセス数を変更できます。追加のユーザーが共有サーバー・プロセスにアクセスするのを防ぐには、MTS\_SERVERS を 0 に設定します。これは、MTS の追加使用を一時的に禁止します。MTS\_SERVERS を正の値にリセットすると、現在のすべてのユーザーにとって MTS が使用可能になります。

**関連項目：** ディスパッチャの詳細は、『Oracle8i リファレンス・マニュアル』の V\$DISPATCHER ビューおよび V\$DISPATCHER\_RATE ビューの説明を参照してください。ALTER SYSTEM コマンドの詳細は、『Oracle8i SQL リファレンス』を参照してください。共有サーバー数の変更の詳細は、『Oracle8i 管理者ガイド』を参照してください。

## パラレル実行サーバーの競合の低減

この項では、パラレル実行の使用時に、パラレル実行サーバーの競合を検出し、軽減する方法について説明します。

- [パラレル実行サーバーの競合の識別](#)
- [パラレル実行サーバーの競合の低減](#)

### パラレル実行サーバーの競合の識別

VSPQ\_SYSSTAT ビューに含まれている統計は、1 インスタンスについて適切なパラレル実行サーバーの数を決定するときに役立ちます。特に役に立つ統計は、SERVERS BUSY、SERVERS IDLE、SERVERS STARTED および SERVERS SHUTDOWN です。

1 インスタンスあたりのパラレル実行サーバーの最大数は、CPU の容量および I/O 帯域幅に大きく依存しているため、増やせない場合がほとんどです。ただし、サーバーが継続的に起動とシャットダウンを繰り返す場合には、PARALLEL\_MIN\_SERVERS 初期化パラメータの値を増やすことを検討してください。

たとえば、使用しているマシンで管理できる同時パラレル実行サーバーの最大数を 100 と判断した場合には、PARALLEL\_MAX\_SERVERS を 100 に設定します。次に、平均的なパラレル操作に必要なパラレル実行サーバーの数を判断し、さらに、同時に実行させるパラレル操作の数を判断します。この例では、同時操作が 2 つあり、その平均並列度を 20 と想定します。その場合、1 インスタンスで 80 のパラレル実行サーバーがビジー状態になる可能性が常にあります。このため、PARALLEL\_MIN\_SERVERS パラメータを 80 に設定する必要があります。

VSPQ\_SYSSTAT を定期的に調べて、インスタンスに対して 80 のパラレル実行サーバーが実際にビジー状態になっているかどうかを判別してください。これを行うには、次の問合せを発行します。

```
SELECT * FROM VSPQ_SYSSTAT
WHERE STATISTIC = "SERVERS BUSY";
```

以下にこの問合せの結果例を示します。

STATISTIC	VALUE
SERVERS BUSY	70

### パラレル実行サーバーの競合の低減

ビジー状態のプロセスが常に PARALLEL\_MIN\_SERVERS よりも少ない場合、アイドル状態のパラレル実行サーバーが使用されていないシステム・オーバーヘッドになります。この場合は、パラメータ PARALLEL\_MIN\_SERVERS の値を小さくすることを検討してください。アクティブなパラレル実行サーバー数が PARALLEL\_MIN\_SERVERS の値よりも平均して多

く、SERVERS STARTED 統計が継続的に増加している場合には、パラメータ PARALLEL\_MIN\_SERVERS の値を増やすことを検討してください。

## REDO ログ・バッファ・ラッチの競合の低減

REDO ログ・バッファのアクセスの競合がデータベース・パフォーマンスを抑制することはめったにありませんが、Oracle では、発生するラッチ競合を監視し、低減する方法を提供しています。この項では次の内容を取り上げます。

- [REDO ログ・バッファ内の領域の競合の検出](#)
- [REDO ログ・バッファ・ラッチの競合の検出](#)
- [REDO ログのアクティビティの検査](#)

### REDO ログ・バッファ内の領域の競合の検出

LGWR が REDO ログ・ファイルに REDO ログ・バッファから REDO エントリを書き込むとき、ユーザー・プロセスはディスクに書き込まれたメモリー内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激しいときであっても、通常 LGWR は高速に書き込みを行い、新しいエントリのバッファ内の領域が常に利用できることを保証します。

値 REDO BUFFER ALLOCATION RETRIES は、ユーザー・プロセスが REDO ログ・バッファ内の領域の使用を待機した回数を反映します。この統計は、動的パフォーマンス・ビュー V\$SYSSTAT を通じて利用できます。デフォルトでは、ユーザー SYS、および SYSTEM のような SELECT ANY TABLE システム権限を付与されているユーザーだけがこのビューを利用できます。

次の問合せによって、アプリケーションの実行中に、ある程度の期間にわたってこれらの統計を監視します。

```
SELECT NAME, VALUE
FROM V$SYSSTAT
WHERE NAME = 'REDO BUFFER ALLOCATION RETRIES';
```

V\$SYSSTAT 内の情報は、シンプル・ネットワーク管理プロトコル (SNMP) を介して取得することもできます。

REDO BUFFER ALLOCATION RETRIES は、ゼロに近い値でなければなりません。この値が一貫して増分する場合は、プロセスがバッファ内の領域を待機する必要があったということです。この待機は、ログ・バッファが小さすぎる、あるいはチェックポイント機能が原因となっていることがあります。必要であれば、初期化パラメータの LOG\_BUFFER の値を変更することによって、REDO ログ・バッファのサイズを大きくできます。このパラメータの値は、バイト単位で表現され、DB\_BLOCK\_SIZE の倍数でなければなりません。あるいは、チェックポイント機能またはアーカイブ・プロセス改善してください。

---

**注意：** 複数のアーカイバ・プロセスの使用はお薦めしません。単一の自動 ARCH プロセスは、LGWR プロセスと速度を合わせて REDO ログをアーカイブします。

---

## REDO ログ・バッファ・ラッチの競合の検出

REDO ログ・バッファへのアクセスは、REDO 割当てラッチおよび REDO コピー・ラッチという 2 つのタイプのラッチによって調節されます。

### REDO 割当てラッチ

REDO 割当てラッチは、REDO ログ・バッファ内の REDO エントリに対する領域の割当てを制御します。Oracle ユーザー・プロセスは、バッファ内の領域を割り当てるために、REDO 割当てラッチを獲得しなければなりません。REDO 割当てラッチは 1 つしかないので、一度にただ 1 つのユーザー・プロセスがバッファ内の領域を割り当てることができます。単一の REDO 割当てラッチは、バッファ内のエントリに対して順次処理の性質を施行します。

REDO エントリに対して領域を割り当てた後で、ユーザー・プロセスは、そのエントリをバッファにコピーすることができます。このようなコピーを "REDO 割当てラッチのコピー" といいます。REDO エントリがしきい値サイズよりも小さい場合のみ、プロセスは REDO 割当てラッチでコピーすることができます。

### REDO コピー・ラッチ

ユーザー・プロセスは最初にコピー・ラッチを取得します。これによってプロセスがコピーできます。次に、割当てラッチを取得し、割当てを実行して、割当てラッチを解放します。次に、プロセスはコピー・ラッチのもとでコピーを実行し、コピー・ラッチを解放します。したがって、割当てラッチは非常に短い期間のみ保持されます。ユーザー・プロセスは、割当てラッチを保持している間はコピー・ラッチを取得しようとしません。

REDO エントリが大きすぎて REDO 割当てラッチでコピーできない場合、ユーザー・プロセスは、そのエントリをバッファにコピーする前に、REDO コピー・ラッチを獲得する必要があります。ユーザー・プロセスは、REDO コピー・ラッチを保持している間、バッファ内の割り当てられた領域に REDO エントリをコピーし、それから REDO コピー・ラッチを解放します。

コンピュータが複数の CPU を持っている場合、REDO ログ・バッファは複数の REDO コピー・ラッチを持つことができます。複数の REDO コピー・ラッチによって、複数のプロセスがエントリを同時に REDO ログ・バッファにコピーできます。

シングル CPU のコンピュータでは、一度に 1 つのプロセスしかアクティブにできないため、REDO コピー・ラッチは使用できません。この場合、サイズにかかわらず、すべての REDO エントリが REDO 割当てラッチでコピーされます。

## REDO ログのアクティビティの検査

REDO ログ・バッファへのアクセスが激しいと、REDO ログ・バッファ・ラッチの競合が起こる可能性があります。ラッチ競合はパフォーマンスを低下させる可能性があります。Oracle はすべてのラッチのアクティビティに対する統計を収集し、それらを動的パフォーマンス・ビュー V\$LATCH に格納します。デフォルトでは、ユーザー SYS、および SYSTEM のような、SELECT ANY TABLE システム権限を持っているユーザーだけがこの表を利用できます。

V\$LATCH 表の各行には、ラッチの異なるタイプに対する統計が収録されます。表の列は、ラッチ要求の異なるタイプのアクティビティを反映します。要求のこれらのタイプの違いは、ラッチが使用できない場合に、要求プロセスがラッチを要求し続けるかどうかということです。

WILLING-TO-WAIT	willing-to-wait 要求で要求したラッチが利用できない場合、要求プロセスは少し時間をおいてから再びラッチを要求します。プロセスは、ラッチが利用できるまで待機と要求を繰り返し続けます。
-----------------	---

IMMEDIATE	immediate 要求で要求したラッチが利用できない場合、要求プロセスは待機しないで処理を継続します。
-----------	--

V\$LATCH ビューの次の列は willing-to-wait 要求を反映します。

GETS	ラッチに対して成功した willing-to-wait 要求の数を示します。
MISSES	最初の willing-to-wait 要求が成功しなかった回数を示します。
SLEEPS	最初の willing-to-wait 要求後にプロセスがラッチを待機し、要求した回数を示します。

たとえば、プロセスが利用できないラッチに対して willing-to-wait 要求を作成する場合について検討します。プロセスは待機してから、そのラッチを再要求しますが、ラッチはまだ使用できません。さらに、プロセスは待機し、3 度目のラッチ要求を行い、ラッチを獲得します。このアクティビティは、次のように統計を増やします。

- 1 つのラッチ要求（3 度目の要求）が成功したので、GETS の値は 1 だけ増えます。
- 最初のラッチ要求は待機したので、MISSES 値は毎回 1 だけ増えます。
- プロセスは最初の要求と 2 度目の要求の後に一度ずつラッチを待機したので、SLEEPS 値は 2 増えます。



V\$LATCH 表の以下の列は immediate 要求を反映します。

IMMEDIATE GETS	各ラッチに対して成功した immediate 要求の数を示します。
IMMEDIATE MISSES	各ラッチに対して成功しなかった immediate 要求の数を示します。

次の問合せによって、REDO 割当てラッチの統計と REDO コピー・ラッチの統計をある期間にわたって監視してください。

```
SELECT ln.name, gets, misses, immediate_gets, immediate_misses
FROM v$latch l, v$latchname ln
WHERE ln.name IN ('redo allocation', 'redo copy')
AND ln.latch# = l.latch#;
```

次に、この問合せの出力例を示します。

NAME	GETS	MISSES	IMMEDIATE_GETS	IMMEDIATE_MISSES
redo allocation	252867	83	0	0
redo copy	0	0	22830	0

この問合せの出力から、要求の各タイプについて待機率を計算してください。

以下の条件のどちらかが真であれば、ラッチの競合がパフォーマンスに影響を及ぼしている可能性があります。

- GETS に対する MISSES の比率が 1% を超える。
- IMMEDIATE\_GETS と IMMEDIATE\_MISSES の合計に対する IMMEDIATE\_MISSES の比率が 1% を超える。

ラッチに対するこれらの条件のどちらかが真であれば、そのラッチの競合を低減するように試みてください。これらの競合のしきい値はほとんどのオペレーティング・システムに適したものです。多くの CPU を持ついくつかのコンピュータでは、パフォーマンスを低下させずに、より多くの競合を許容できる可能性があります。

## LRU ラッチの競合の低減

LRU (least recently used) ラッチは、バッファ・キャッシュ内のバッファの置換を制御します。対称型マルチプロセッサ (SMP) システムでは、Oracle によって、LRU ラッチの数がシステムの CPU 数の 2 分の 1 に等しい値になるように自動的に設定されます。非 SMP システムでは、LRU ラッチは 1 つあれば十分です。

LRU ラッチの競合は、多数の CPU を備えた SMP マシンでのパフォーマンスを低下させることがあります。LRU ラッチの競合は、V\$LATCH、V\$SESSION\_EVENT および V\$SYSTEM\_EVENT に問い合わせることによって検出できます。競合を回避するには、バッ

ファ・キャッシュをバイパスすること、またはアプリケーションを再設計することを検討してください。

システム上の LRU ラッチの数は、DB\_BLOCK\_LRU\_LATCHES 初期化パラメータを用いて指定できます。このパラメータは、希望の LRU ラッチ数のための最大値を設定します。各 LRU ラッチがバッファ・セットを制御し、Oracle がセット間の置換バッファの割当てのバランスを取ります。

DB\_BLOCK\_LRU\_LATCHES の最適値を選択するためには、次の事項を考慮します。

- ラッチの最大数は、システム内の CPU 数の 2 倍です。つまり、DB\_BLOCK\_LRU\_LATCHES の値は 1 ~ CPU 数の 2 倍です。
- 各ラッチのセット内のバッファは、50 を下回ってはなりません。小さなバッファ・キャッシュでは、より大きいセット数を選択した場合の付加価値はありません。バッファ・キャッシュのサイズによって、セット数に関する最大境界条件が決まります。
- Oracle がシングル・プロセス・モードで実行中のときは、複数のラッチを作成しないでください。シングル・プロセス・モード中は LRU ラッチは自動的に 1 つしか使われません。
- インスタンスにかかる負荷が大きい場合は、ラッチ数を増やしてください。たとえば、システム内に 32 個の CPU がある場合は、システム内の CPU の半数（16）と実際の数（32）との間の数を選択してください。

---

---

**注意：** インスタンスの稼働期間中は、セット数を動的に変更できません。

---

---

## 空きリスト競合の低減

空きリスト競合によって、いくつかのアプリケーションのパフォーマンスが低下する可能性があります。この項では次の内容を取り上げます。

- [空きリストの競合の識別](#)
- [空きリストの追加](#)

### 空きリストの競合の識別

空きリストの競合は、バッファ・キャッシュ内の空きデータ・ブロックの競合によって反映されます。動的パフォーマンス・ビュー `V$WAITSTAT` に問合せを実行することによって、空きリストの競合がパフォーマンスを低下させているかどうかを判断できます。

**関連項目：** 空きリストの詳細は、『Oracle8i 概要』を参照してください。

`V$WAITSTAT` 表は、ブロック競合統計を収録します。デフォルトでは、ユーザー `SYS`、および `SYSTEM` のような、`SELECT ANY TABLE` システム権限を持っているユーザーだけがこのビューを利用できます。

次の手順に従って、競合しているセグメント名と空きリストを検出してください。

1. `V$WAITSTAT` で `DATA BLOCKS` の競合をチェックします。
2. `V$SYSTEM_EVENT` で `BUFFER BUSY WAITS` をチェックします。  
数値が高い場合は、なんらかの競合が発生していることを示します。
3. この場合は、`V$SESSION_WAIT` をチェックして、各バッファ・ピジー待機について `FILE`、`BLOCK` および `ID` の値を確認してください。
4. 次のような問合せを組み立てて、バッファ・ピジー待機をしているオブジェクトおよび空きリストの名前を獲得してください。

```
SELECT SEGMENT_NAME, SEGMENT_TYPE
FROM DBA_EXTENTS
WHERE FILE_ID = file
AND BLOCK BETWEEN block_id AND block_id + blocks;
```

この問合せは、セグメント名 (*segment*) とタイプ (*type*) を返します。

5. 空きリストを見つけるには、次の問合せを実行してください。

```
SELECT SEGMENT_NAME, FREELISTS
FROM DBA_SEGMENTS
WHERE SEGMENT_NAME = SEGMENT
AND SEGMENT_TYPE = TYPE;
```

## 空きリストの追加

表の空きリストの競合を低減するには、FREELISTS 記憶領域パラメータの値を大きくしてその表を再作成します。このパラメータの値を、表にデータを同時に挿入する Oracle プロセスの数まで増やすと、INSERT 文のパフォーマンスが向上する可能性があります。

表の再作成では、単にその表を削除してから、もう一度作成することが必要となります。ただし、かわりに次のいずれかの方法を使用できます。

- 古い表から新しい表にデータを選択し、古い表を削除し、新しい表を改名します。
- Export と Import を使って、表をエクスポートし、表を削除し、表をインポートします。この方法では、一時表の作成による領域の消費を回避します。

---

## ネットワークのチューニング

この章では、チューニングに影響するネットワークの問題を説明します。トピックは次のとおりです。

- ネットワークの問題の検出
- ネットワークの問題の解決

### ネットワークの問題の検出

ネットワークには、処理をある程度遅延させるオーバーヘッドが伴います。パフォーマンスを最適化するには、ネットワーク・スループットを高速にし、ネットワーク上に送信する必要のあるメッセージ数を削減する必要があります。

ネットワークによって加えられる遅延を測定するのは困難な場合があります。この作業に役立つ、V\$SESSION\_EVENT、V\$SESSION\_WAIT および V\$SESSTAT という 3 つの動的パフォーマンス・ビューがあります。

V\$SESSION\_EVENT では、Oracle がメッセージを待機する時間間隔を AVERAGE\_WAIT 列が示します。この統計を判断基準として使用して、ネットワークの効率を評価できます。

V\$SESSION\_WAIT には、アクティブなセッションが待機しているイベントをリストする EVENT 列が含まれます。"sqlnet message from client" 待機イベントは、共有プロセスまたはフォアグラウンド・プロセスがクライアントからのメッセージを待機していることを示します。この待機イベントが発生した場合、メッセージがユーザーによって送信されたかどうか、または Oracle によって受信されたかどうかをチェックできます。

V\$SESSION\_WAIT を参照することによってセッションが待機している対象を確認し、ハングアップが調査できます。クライアントがメッセージを送信済みの場合は、Oracle がそれに応答中か、あるいはまだ待機中であるかを判断できます。

V\$SESSTAT では、クライアントから受信したバイト数、クライアントに送信されたバイト数およびクライアントが実行したコール数を確認できます。

## ネットワークの問題の解決

この項では、パフォーマンスを向上させる手法およびネットワークの問題を解決する手法をいくつか説明します。

- [配列インタフェースの使用方法](#)
- [事前開始プロセスの使用方法](#)
- [セッション・データ単位バッファ・サイズの調整](#)
- [リスナー・キューのサイズの増大](#)
- [TCP.NODELAY の使用方法](#)
- [専用サーバー・プロセスではなく共有サーバー・プロセスを使用する方法](#)
- [Connection Manager の使用](#)

**関連項目：**『Oracle8i Net8 管理者ガイド』

### 配列インタフェースの使用方法

配列インタフェースを使ってネットワーク・コールを削減します。一度に 1 行をフェッチするよりも、ネットワークの 1 往復で 10 行をフェッチする方が効率的です。

**関連項目：** 配列インタフェースの詳細は、『Pro\*C/C++ プリコンパイラ・プログラマーズ・ガイド』および『Pro\*COBOL プリコンパイラ・プログラマーズ・ガイド』を参照してください。

### 事前開始プロセスの使用方法

事前開始プロセスは、専用サーバーとの接続時間を改善できます。これは、マルチスレッド・サーバーを使用していないために負荷が重く、接続時間が低速なシステムに特に効果があります。事前開始プロセスが使用可能にされている場合、リスナーは接続要求を受信するたびに、待機しないで既存のプロセスに接続を渡せます。接続要求は、新しいプロセスが開始されるのを待機する必要がありません。

### セッション・データ単位バッファ・サイズの調整

ネットワーク上にデータを送信する前に、Net8 はデータをセッション・データ単位 (SDU) でバッファリングします。SQL Net は、バッファが満杯になったとき、またはアプリケーションがデータを読み込もうとしたときに、バッファに格納されているデータを送信します。大量のデータが検索されるとき、およびパケット・サイズが一貫して同じときは、デフォルトの SDU サイズを調整すると検索が高速になる可能性があります。

最適な SDU サイズは、標準パケット・サイズによって決まります。検出ツールを使用してフレーム・サイズを調べるか、トレースを最大レベルに設定して送受信されたパケット数を

チェックし、それらが断片化されているかどうかを確認します。システムをチューニングして、断片化の量を制限してください。

Net8 Assistant を使用して、クライアントとサーバーの両方でデフォルト SDU サイズを変更します。SDU サイズは、一般にクライアントとサーバーの両方で同じにする必要があります。

**関連項目：**『Oracle8i Net8 管理者ガイド』

## リスナー・キューのサイズの増大

データベース・サーバー上でアクティブなネットワーク・リスナーは、接続要求をモニターし、接続要求に応答します。より多数の同時要求を動的に処理するために、リスニング・プロセスのリスニング・キューを増やすことができます。

## TCP.NODELAY の使用方法

セッションが確立すると、Net8 はデータをパッケージ化し、パケットを使ってサーバーとクライアント間でデータを送信します。TCP.NODELAY オプションを使うと、パケットがネットワーク上にフラッシュされる間隔を短くすることができます。大量のデータをストリーミングしても、バッファリングが発生せず遅延もありません。

Net8 は多くのネットワーク・プロトコルをサポートしていますが、最も拡張性が高いのは TCP です。

**関連項目：** プラットフォーム固有の Oracle マニュアル

## 専用サーバー・プロセスではなく共有サーバー・プロセスを使用する方法

マルチスレッド・サーバー・ディスパッチャのような共有サーバー・プロセスは、専用サーバー・プロセスよりもパフォーマンスが高い傾向にあります。専用サーバー・プロセスは 1 つのセッションだけを処理し、そのセッションが持続している間存在します。それに対して、共有サーバー・プロセスでは、多数のクライアントが同じサーバーに接続することができます。クライアントごとの専用サーバー・プロセスは必要ありません。共有サーバーに対する複数のセッション要求が入ってくると、ディスパッチャによって処理されます。

---

**注意：** Oracle のデフォルトは専用サーバー処理です。マルチスレッド・サーバーの詳細は、『Oracle8i 概要』および『Oracle8i 管理者ガイド』を参照してください。

---

## Connection Manager の使用

Net8 では、多重化（多数のクライアント・セッションを 1 つのトランスポート接続を通じてサーバーへ送り込む）によってシステム・リソースを節約するために、Connection Manager を使用することができます。この方法を使用すると、1 つのプロセスが処理できるセッションの数を増やすことができます。

Connection Manager を使用すると、専用サーバーへのクライアントのアクセスを制御できます。また、Connection Manager では、異なるネットワーク・プロトコルを持つサーバーとクライアントが通信できる、複数プロトコルのサポートが提供されます。

**関連項目：**『Oracle8i Net8 管理者ガイド』



---

## マルチスレッド・サーバー・アーキテクチャ のチューニング

マルチスレッド・サーバー（MTS）は、Oracle Server テクノロジーの戦略的コンポーネントであり、同時にデータベースに接続する大多数のクライアントをサポートするアプリケーションに対して、優れたユーザー拡張性を提供します。接続プーリングや多重化などの MTS の機能は、アプリケーションにとって有益です。

MTS のアーキテクチャによってユーザー拡張性の増大、スループットの向上および応答時間の短縮が可能になるため、MTS は Oracle に戦略的な機能性をもたらします。ユーザー数が増加しているときでも、MTS はより少ないリソースを使用してこれを行います。MTS は、ほとんどのアプリケーションを拡張して、大多数のユーザーを処理できるようにします。アプリケーションを変更する必要はありません。

---

**注意：** また、Oracle を構成して、MTS と専用サーバー構成の両方を使用することもできます。この詳細は、『Oracle8i Net8 管理者ガイド』を参照してください。

---

### MTS の設定

MTS を設定するには、使用するアプリケーションに合わせて MTS 関連のパラメータを設定します。

**関連項目：** 設定方法の詳細は、『Oracle8i 管理者ガイド』を参照してください。

### MTS の恩恵を受けるアプリケーションの種類

データベース・サーバーのメモリー使用量を最小化する一方、多数のユーザーが同時にデータベースに接続する必要のあるアプリケーションが、MTS の恩恵を最も強く受けるアプリケーションの種類です。例としては、トランザクション量が多い OLTP アプリケーション

や、Java ベース・アーキテクチャの IIOP を使用する Web ベースのシン・クライアント・アプリケーションです。

MTS はユーザー拡張性とパフォーマンス拡張をもたらし、Oracle Server での 3 層アプリケーションの使用を可能にします。多くの場合、MTS はトランザクション処理 (TP) モニターのかわりとして最適です。これは、MTS には TP モニターに関連するパフォーマンス・オーバーヘッドがないためです。

また、MTS を OPS (Oracle Parallel Server) とともに使用すると、優れた拡張性と高い可用性を備えたシステムが得られます。MTS を OPS とともに使用すると、年中無休のアップタイムを達成することが可能です。

次に、Oracle8i の重要なパフォーマンス拡張と MTS 関連の新機能を示します。

- 非同期ネットワーク I/O
- 主要なメモリー管理の拡張
- 動的な複数のプレゼンテーションのサポート
- 改善された管理性と非常に容易になった構成
- 接続プーリングと接続多重化
- 改善されたロード・バランシング・アルゴリズム (この機能は "コネクション・ロード・バランシング" と呼ばれます)

**関連項目：** OPS の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

## MTS によるユーザー・スケーラビリティの改善

MTS の使用によって、システムをチューニングし、リソース使用量を最小化できます。一般的なアプリケーションでの MTS 関連パラメータの値は、次のいくつかの要因によって決定します。

- ディスパッチャ数に対する接続数の割合
- 共有サーバーに対する接続
- アプリケーションそのもの

次の項では、ディスパッチャのチューニング方法と、MTS の接続プーリングおよび接続多重化機能の使用方法について説明します。

## ディスパッチャの構成

実行中のシステムのアクティブなディスパッチャの数は動的に増加しません。システムが起動した後でディスパッチャ数を増やして、より多くのユーザーを扱えるようにするには、MTS\_DISPATCHERS パラメータの DISPATCHERS 属性の値を変更します。このパラメータの値を変更するときは、要求されるディスパッチャの合計数がパラメータ MTS\_MAX\_DISPATCHERS に設定された値を超えないようにします。MTS\_MAX\_DISPATCHERS のデフォルト値は 5 です。

構成するディスパッチャ数が MTS\_MAX\_DISPATCHERS の値よりも大きい場合は、Oracle によって MTS\_MAX\_DISPATCHERS の値がその値にまで増やされます。同時接続の数が時間とともに増加することが予測されないかぎり、このパラメータを設定する必要はありません。

一般的なシステムでは、接続 250 に対してディスパッチャ 1 という割合が適切に作動します。たとえば、ピーク時に 1000 の接続を予測する場合は、ディスパッチャを 4 つ構成するとよいでしょう。見積りが多すぎても利点はありません。構成するディスパッチャが多すぎるとパフォーマンスが低下する場合があります。

---

**注意：** NT では、ディスパッチャはスレッドであり、個別のプロセスではありません。

---

## 接続プーリングと接続多重化

多数のディスパッチャを構成するためのリソースがないが、システムでは同時接続をさらに必要とする場合は、MTS とともに接続プーリングと接続多重化を使用します。

MTS は接続プーリングを使用可能にします。接続プーリングでは、接続がアイドルになったときに一時的にクライアント接続をリリースすることによって、クライアントがサーバー側のプールの接続スロットを共有します。MTS は、Connection Manager を使用して接続多重化を使用可能にします。これによって、複数のクライアント接続が、Connection Manager からデータベースへの 1 つの接続を共有できます。

---

**注意：** 接続のプーリングと多重化の機能は、MTS でのみ作動します。

---

**関連項目：** 多重化のインプリメントの例は、『Oracle8i Net8 管理者ガイド』を参照してください。

## MTS によるスループットの最大化と応答時間の最短化

この項では、MTS を使用してスループットを最大化し応答時間を最短化する方法を説明します。次のトピックを含みます。

- 共有サーバー数の構成と管理
- SDU サイズのチューニング

### 共有サーバー数の構成と管理

生成される共有サーバー・プロセス数は、必要に応じて動的に変化します。システムの開始時には、MTS\_SERVERS の設定値で初期値として指定された共有サーバー数が生成されます。必要であれば、MTS\_MAX\_SERVERS パラメータに設定された値内でシステムがさらに共有サーバーを生成します。

システムの負荷が下がると、MTS\_SERVERS で指定された最小限のサーバー数が保持されます。このため、システム起動時の MTS\_SERVERS の値の設定は高くしすぎないでください。一般的なシステムでは、10 の接続に対して共有サーバー 1 つという割合で安定するようです。

OLTP アプリケーションでは、要求数が少なく、要求に対するサーバー使用率が低いため、サーバーに対する接続の割合は大きくなるでしょう。要求の割合が高いか、要求に対するサーバー使用率が高いアプリケーションでは、サーバーに対する接続の割合は小さくなります。

この場合、使用するアプリケーションに基づいた妥当な値に MTS\_MAX\_SERVERS を設定します。MTS\_MAX\_SERVERS のデフォルト値は 20、MTS\_SERVERS のデフォルトは 1 です。

NT では、前述したように各サーバーは共通プロセス内のスレッドなので、MTS\_MAX\_SERVERS の値を高く設定する場合には注意を払ってください。これらの設定の最適な値は、構成によって変化します。つまり、これらは、一般的な構成での動作を予測したものにはすぎません。

MTS\_MAX\_SERVERS は静的な INIT.ORA パラメータなので、データベースをシャットダウンしないで変更することはできません。ただし、MTS\_SERVERS は動的パラメータなので、ALTER SYSTEM コマンドを使用してアクティブなセッション中に変更することができます。

**OLTP アプリケーションの例** :2000 という同時接続が必要になると予測される場合は、200 の共有サーバー、すなわち接続 10 に対して共有サーバー 1 で開始します。

MTS\_MAX\_SERVERS を 400 に設定します。これは OLTP アプリケーションなので、各接続による共有サーバーに対する負荷は通常のアプリケーションよりも低いと考えられます。かわりに、200 ではなく 100 の共有サーバーで開始してもよいでしょう。共有サーバーの追加が必要になると、MTS\_MAX\_SERVERS の値内でシステムが数を調整します。

## SDU サイズのチューニング

Net8 は、バッファ・データをセッション・データ単位 (SDU) で格納し、バッファが満杯になるか、アプリケーションがデータを読み取ろうとすると、このバッファに格納されたデータを送信します。大容量のデータが取り出されるとき、およびパケット・サイズが一貫して同じ場合に、デフォルトの SDU サイズをチューニングします。これによって取出しの速度が向上し、断片化も減少します。

**関連項目：** 詳細は、22-2 ページの「[セッション・データ単位バッファ・サイズの調整](#)」を参照してください。

## 接続負荷の調整

接続負荷の調整では、次の要因に基づいて負荷を分散します。

- 1 ディスパッチャあたりの接続数。" ディスパッチャ・レベルのロード・バランシング" とも呼ばれます。
- ノードでの負荷。" ノード・レベルのロード・バランシング" とも呼ばれます。

接続負荷の調整は、MTS が使用可能な場合にのみ動作します。

OPS またはレプリケーション環境のデータベースを使用する場合、MTS の接続負荷の調整機能によって、DESCRIPTION\_LISTS を使用する場合よりもすぐれたロード・バランシングが提供されます。これは、この機能ではクライアント接続を実際の CPU 負荷に基づいて分散させるためです。また、MTS によって、アプリケーション依存の単純化したルーティング構成が可能になります。この構成では、同一アプリケーションからの要求が毎回同じノードにルーティングされます。これによって、データを転送するアプリケーションの効率が良くなります。

---

---

**注意：** 接続負荷の調整は、MTS が使用可能な場合にのみ動作します。

---

---

**関連項目：** 詳細は、『Oracle8i Parallel Server 概要および管理』の「複数インスタンスの管理」の章を参照してください。

## MTS によるメモリー使用状況のチューニング

この項では、MTS によってメモリー使用状況をチューニングする方法を説明します。

### MTS での大規模プールと共有プールの構成

MTS 関連の UGA (ユーザー・グローバル領域) の割当てには、共有プールではなく大規模プールの使用をお薦めします。共有メモリーは、Oracle によって、共有 SQL や PL/SQL プ

ロシージャなどの他の目的で SGA メモリーを割り当てるためにも使用されるためです。共有プールのかわりに大規模プールを使用すると、SGA の断片化も減少します。

大規模プールに MTS 関連の UGA を格納するには、パラメータ `LARGE_POOL_SIZE` に値を指定します。`LARGE_POOL_SIZE` に値を指定しないと、MTS ユーザー・セッション・メモリーを共有プール内に確保します。

MTS を使用するときは、大規模プールまたは共有プールは通常よりも大きく構成してください。これは、UGA のすべてのユーザー状態情報が MTS の SGA（共有グローバル領域）に格納されるため、必要な処置です。

共有プールを使用する場合、`SHARED_POOL_SIZE` のデフォルト値は 32 ビット・システムで 8MB、64 ビット・システムでは 64MB になります。`LARGE_POOL_SIZE` にはデフォルト値はありませんが、最小値は 300K です。

---

---

**注意：** `LARGE_POOL_SIZE` に値を設定した場合でも、Oracle によってセッションごとに一定量のメモリーが共有プールから割り当てられます。

---

---

### MTS の UGA 記憶領域のための効果的な設定の判別

Oracle が使用する UGA の厳密な容量は、各アプリケーションによって異なります。大規模プールまたは共有プールの効果的な設定を判別するには、一般的なユーザーでの UGA の使用状況を観察して、その容量をユーザー・セッションの見積り数に掛けます。

MTS の使用により共有メモリーの使用が増加するとしても、合計のメモリー使用量は減少します。これは、プロセス数が減少するので、MTS では PGA メモリーの使用量が減るためです。専用サーバー環境での動作とは逆になります。

## PRIVATE\_SGA の設定によるユーザー・セッションごとのメモリー使用量の制限

MTS を使用すると、`PRIVATE_SGA` パラメータを設定して、各クライアント・セッションによる SGA のメモリー使用量を制限できます。`PRIVATE_SGA` によって、1 セッションで SGA から使用されるメモリーのバイト数が定義されます。ただし、ほとんどの DBA はユーザー単位での SGA 消費量の制限は行わないため、このパラメータを使用することはほとんどありません。

**関連項目：** 詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## 接続、負荷および統計データを含む MTS 関連ビュー

Oracle では、ディスパッチャ、共有サーバー、接続が確立される速さ、キュー内のメッセージ、使用される共有メモリーなどに関する情報を含むビューが提供されています。

V\$DISPATCHER_RATE	各ディスパッチャがメッセージやイベントなどを受信および処理する速度に関する情報と統計を提供するビュー。
V\$MTS	インスタンスが開始されてからの、システムがこれまでに起動した共有サーバーの最大数、ディスパッチャによって処理された接続の最大数などについての統計を提供するビュー。
V\$DISPATCHER	ディスパッチャ・プロセスについての情報を提供するビュー。
V\$SHARED_SERVERS	共有サーバー・プロセスについての情報を提供するビュー。
V\$CIRCUITS	仮想サーキットについての情報を提供するビュー。仮想サーキットは、データベース・セッション中にアクセスされる必要があるユーザー関連の状態情報すべてのリポジトリとして作動する状態オブジェクトです。1 つのクライアント接続ごとに 1 つの仮想サーキットがあります。
V\$QUEUE	共通メッセージ・キューおよびディスパッチャ・メッセージ・キュー内のメッセージについての情報を提供するビュー。
V\$SGA	システム・グローバル領域についての情報を提供するビュー。
V\$SGASTAT	システム・グローバル領域についての詳細な統計情報を提供するビュー。
V\$SHARED_POOL_RESERVE	統計を提供するビュー。

## MTS 機能のパフォーマンス問題

MTS を使用すると、特定のデータベース機能のパフォーマンスがわずかに低下することがあります。このような機能には、BFILE、パラレル実行、ノード間パラレル実行およびハッシュ結合が含まれます。これは、これらの機能がアクティブなときには、セッションが別の共有サーバーへ移行することを妨げる場合があるためです。

クライアントの要求が処理された後で、セッションが移行不可能なままになることがあります。これらの機能はすべてのユーザー状態情報を UGA に格納せずに、状態の一部を PGA に残すため、これらの機能を使用するとセッションが移行不可能になることがあります。結果として、さまざまな共有サーバーがクライアントの要求を処理する場合、PGA に格納されているユーザー状態の一部はアクセスできません。これを回避するために、個々の共有サーバーは、しばしばユーザー・セッションにバインドされた状態を続ける必要があります。これによって、共有サーバー間でセッションが移行不可能になります。

これらの機能を使用するときは、より多くの共有サーバーを構成することが必要になる場合があります。長期にわたって一部のサーバーがセッションにバインドすることがあるためです。



---

## オペレーティング・システムのチューニング

---

この章では、Oracle Server のパフォーマンスを最適化するためにオペレーティング・システムをチューニングする方法を説明します。トピックは次のとおりです。

- [オペレーティング・システムのパフォーマンスの問題の理解](#)
- [オペレーティング・システムの問題の検出](#)
- [オペレーティング・システムの問題の解決](#)

**関連項目：** この章の情報の他に、使用しているオペレーティング・システム対応のマニュアルも参照してください。

### オペレーティング・システムのパフォーマンスの問題の理解

オペレーティング・システムのパフォーマンスの問題は、一般にプロセス管理、メモリー管理およびスケジューリングに関係します。Oracle インスタンスをチューニングした後で、さらにパフォーマンスを改善する必要がある場合は、作業方法を検証するか、システム時間の短縮を試行してください。I/O 帯域幅、CPU 能力およびスワップ・スペースが十分にあることを確認してください。ただし、オペレーティング・システムをさらにチューニングしても、それがアプリケーションのパフォーマンスに大きな効果を与えることは期待できません。単にオペレーティング・システムをチューニングするよりも、Oracle の構成またはアプリケーションを変更する方が、オペレーティング・システムの効率を大きく変えます。

たとえば、アプリケーションでバッファ・ビジー待機が非常に頻繁に発生する場合は、システム・コールの回数が増加します。アプリケーションをチューニングすることでバッファ・ビジー待機を削減すると、システム・コールの数が減少します。同様に、Oracle 初期化パラメータ `TIMED_STATISTICS` をオンにすると、システム・コールの数は増加します。このパラメータをオフにすると、システム・コールは減少します。

**関連項目：** 詳細は、使用しているプラットフォーム固有の Oracle マニュアルおよび使用しているオペレーティング・システム・ベンダーのマニュアルを参照してください。

## オペレーティング・システムおよびハードウェア・キャッシュ

オペレーティング・システムとデバイス・コントローラが提供するデータ・キャッシュは、Oracle 独自のキャッシュ管理と直接は衝突しません。ただし、パフォーマンスにほとんど利益がもたらされないにもかかわらず、これらの構造ではリソースが消費されます。これは、データベース・ファイルが UNIX のファイル・システム上にある UNIX システムにおいて最も顕著です。デフォルトでは、すべてのデータベース I/O がファイル・システムのキャッシュを介して行われます。一部の UNIX システムでは、ファイル・システムへの直接 I/O が使用可能です。これによって、データベースを含んでいるファイルは、ファイル・システム・キャッシュをバイパスして UNIX ファイル・システム内でアクセスできます。これによって CPU リソースを節約でき、ファイル・システム・キャッシュを、プログラム・テキストやスプール・ファイルなどのデータベース以外のアクティビティ専用とすることができ

ます。

NT では、この問題は発生しません。データベースから要求されたファイルは、すべてファイル・システム内のキャッシュをバイパスします。

## ロー・デバイス

システム上のロー・デバイスの使用方法を評価してください。ロー・デバイスを使用すると作業量は増加しますが、パフォーマンスも大きく改善されることがあります。

ロー・デバイスは、構成によっては、全表走査に悪影響を与えますが、実装が "ライト・スルー" キャッシュをサポートしていない UNIX システムでは必須です。UNIX ファイル・システムは、連続するデータ・ブロックの要求をサーバーが開始したときに、先読みすることで全表走査のスピードを上げます。全表走査のキャッシュも行います。UNIX のシステムが、ファイル・システムへの書込みでライト・スルー・オプションをサポートしていない場合は、コミットとチェックポイントの実行時に、ディスク上に支障なく設定されていることをサーバーが前提とするデータが、実際にその場所に存在することを保証するために、ロー・デバイスを使用する必要があります。そうしないと、UNIX オペレーティング・システムのクラッシュからの回復が不可能になることがあります。

NT 上のロー・デバイスは UNIX のロー・デバイスと似ていますが、すべての NT デバイスはライト・スルー・キャッシュをサポートしています。

## プロセス・スケジューラ

多数のプロセス、または NT システムでは "スレッド" が、Oracle の操作に関与しています。これらはすべて、SGA の共有メモリー・リソースにアクセスします。

すべての Oracle プロセス、つまりバックグラウンド・プロセスとユーザー・プロセスの両方に、同じプロセス優先順位が設定されていることを確認してください。Oracle のインストール時に、すべてのバックグラウンド・プロセスに、オペレーティング・システムのデフォルトの優先順位が指定されます。バックグラウンド・プロセスの優先順位は変更しないでください。すべてのユーザー・プロセスにデフォルトのオペレーティング・システム優先順位が限定されるようにしてください。

Oracle プロセスに異なる優先順位を割り当てると、競合の影響が悪化するかもしれません。オペレーティング・システムは、優先順位の高いプロセスが処理時間を要求している場合、優先順位の低いプロセスに処理時間を与えない場合があります。優先順位の高いプロセスが、優先順位の低いプロセスによって保持されているメモリー・リソースにアクセスする必要があると、優先順位の高いプロセスは、優先順位の低いプロセスが CPU を獲得し、要求を処理し、そしてリソースを解放するのを無期限に待機する可能性があります。

## オペレーティング・システムの問題の検出

オペレーティング・システムのツールから抽出することが重要な統計は次のとおりです。

- CPU の負荷
- デバイス・キュー
- ネットワーク・アクティビティ（キュー）
- メモリーの管理（ページング / スワッピング）

CPU 使用率を調べて、アプリケーション・モードでの実行に消費している時間とオペレーティング・システム・モードでの実行に消費している時間の比率を確認します。実行キューを調べて、実行可能なプロセスの数と実行中のシステム・コールの数を確認します。ページングまたはスワッピングが発生しているかどうかを確認し、実行されている I/O の数をチェックします。

**関連項目：** 使用しているプラットフォーム固有の Oracle マニュアルおよび使用しているオペレーティング・システム・ベンダーのマニュアル

## オペレーティング・システムの問題の解決

この項では、さまざまなシステムでのチューニングのヒントを示します。次のトピックについて説明します。

- [UNIX ベース・システムでのパフォーマンス](#)
- [NT システムでのパフォーマンス](#)
- [メインフレーム・コンピュータでのパフォーマンス](#)

プラットフォーム固有の問題をよく理解し、使用しているオペレーティング・システムが提供するパフォーマンス・オプションを認識してください。たとえば、いくつかのプラットフォームには、システム時間をマップしてシステム・コールを削減し、I/O を高速にすることができるポストウェイト・ドライバがあります。

**関連項目：** 使用しているプラットフォーム固有の Oracle マニュアルおよび使用しているオペレーティング・システム・ベンダーのマニュアル

## UNIX ベース・システムでのパフォーマンス

UNIX システムでは、オペレーティング・システムが費やす時間量（システム・コールの処理およびプロセス・スケジューリングの実行）とアプリケーションが実行する時間量の妥当な比率を確立するようにしてください。アプリケーション・モードの時間を 60 ~ 75% とし、オペレーティング・システム・モードの時間を 25 ~ 40% とすることを目標としてください。各モードの時間の 50% をシステムが消費している場合は、問題点を判断してください。

各モードで消費される時間の比率は、本来の問題の単なる症状であり、次の項目に関係している可能性があります。

- スワッピング
- 実行している O/S システム・コールが多すぎる状態
- 実行しているプロセスが多すぎる状態

このような条件が存在する場合は、アプリケーションの実行に使用できる時間は少なくなります。オペレーティング・システム側の時間を多く解放するほど、アプリケーションが実行できるトランザクションの量が増えます。

## NT システムでのパフォーマンス

UNIX ベースのシステムと同様に、NT システムでは、アプリケーション・モードの時間とシステム・モードの時間の比率を適切に設定する必要があります。NT システムでは、パフォーマンス モニタを使用して多くの要因を容易に監視できます。CPU、ネットワーク、I/O およびメモリーは、すべて同じグラフに表示され、これらの分野のいずれかでボトルネックが発生しないようにすることができます。

## メインフレーム・コンピュータでのパフォーマンス

メインフレームのページング・パラメータを検討し、Oracle がパラメータの非常に大きな作業セットを利用できることを覚えておいてください。

VAX/VMS 環境での空きメモリーは、どのオペレーティング・システム・プロセスにも実際にマップされていないメモリーです。ビジーなシステムでは、1 つ以上の現行のアクティブ・プロセスに属するページを空きメモリーが含むことがよくあります。これがアクセスされると "ソフト・ページ・フォルト" が発生し、ページにはそのプロセスの作業セットが組み込まれます。プロセスが作業セットを拡張できない場合は、プロセスによって現在マップされているページの 1 つを空きセットに移動する必要があります。

任意の数のプロセスが、作業セット内に共有メモリーのページを持つことができます。したがって、作業セットのサイズの合計が使用可能メモリーを著しく超過することがあります。Oracle Server の実行中は、SGA、Oracle カーネル・コードおよび Oracle Forms ランタイム実行可能ファイルは一般にすべて共有可能であり、アクセスされるページのおよそ 80% または 90% に相当します。

バッファを追加することがよいとはかぎりません。各アプリケーションには、キャッシュ・ヒット率の上昇が止まるバッファ数のしきい値があります。これは、一般にかなり低い数字です（およそ 1500 バッファ）。より高い値を設定しても、Oracle とオペレーティング・システムの両方で管理負荷が増えるだけです。



---

## インスタンス回復パフォーマンスのチューニング

この章では、インスタンス回復をチューニングする場合のガイドラインを説明します。トピックは次のとおりです。

- [インスタンス回復について](#)
- [インスタンスおよびクラッシュの回復時間のチューニング](#)
- [インスタンス回復の監視](#)
- [インスタンス回復のフェーズのチューニング](#)
- [透過的アプリケーション・フェイルオーバー](#)

### インスタンス回復について

インスタンス回復およびクラッシュ回復とは、クラッシュまたはシステム障害の後で REDO ログ・レコードが Oracle データ・ブロックに自動的に適用されることです。単一インスタンスのデータベースがクラッシュした場合や、OPS（Oracle Parallel Server）構成の全インスタンスがクラッシュした場合には、Oracle は次の起動時にインスタンス回復を実行します。OPS 構成の 1 つ以上のインスタンスがクラッシュした場合は、残りのインスタンスが回復を実行します。

インスタンスおよびクラッシュの回復は 2 つのフェーズで行われます。フェーズ 1 では、REDO ログ・ファイル中のコミット済みおよび未コミットの全変更内容が、影響を受けたデータブロックに適用されます。フェーズ 2 では、ロールバック・セグメントの情報を適用して、未コミット・トランザクションによってデータ・ブロックに対して行われた変更が取り消されます。

## REDO ログ情報の適用方法

通常の運用時は、Oracle の DBW $n$  プロセスが使用済みバッファ（メモリー内で変更されたバッファ）をディスクに定期的書き込みます。Oracle は定期的に全変更内容のうち最も番号の大きい SCN をブロックに記録して、その SCN よりも番号が小さい変更内容を含むすべてのデータ・ブロックが DBW $n$  によってディスクに書き込まれるようにします。この SCN が "チェックポイント" です。

チェックポイントが示す変更レコードの後に REDO ログ・ファイルに追加されたレコードは、Oracle がまだディスクに書き込んでいない変更内容です。障害が発生した場合は、チェックポイントよりも番号の大きい SCN を変更内容を含む REDO ログ・レコードだけが回復時に再生される必要があります。

回復処理の長さは、チェックポイントの SCN よりも番号の大きい SCN を変更内容を含むデータ・ブロックの数によって直接影響を受けます。たとえば、1 つのデータ・ブロックに影響する 100 のエントリを含む REDO ログの方が、10 の異なるデータ・ブロックに対して 10 のエントリを含む REDO ログよりも短時間で回復できます。これは、回復時に処理される各ログ・レコードについて、REDO ログ・エントリが表す変更内容に対応するデータ・ブロックに適用できるように、そのブロックをディスクから読み取る必要があるためです。

## 回復時間最短化のトレードオフ

インスタンス回復時間と日常業務のパフォーマンスのバランスをとるための主要な手段は、Oracle がチェックポイントを動かす頻度です。最新 REDO ログ・レコードよりも数ブロックだけ遅れてチェックポイントをとるようにすると、回復時に処理するブロック数が最小化されます。

ただし、回復時間を最短にすると、トレードオフとして、通常のデータベース操作でのパフォーマンスのオーバーヘッドが増加します。日常的な操作効率が回復時間の最短化よりも重要な場合は、データファイルへの書き込み頻度を減らすとインスタンス回復時間は長くなります。

## インスタンスおよびクラッシュの回復時間のチューニング

インスタンスやクラッシュの回復をチューニングして、回復時間をユーザー指定範囲に収める方法がいくつかあります。その方法は次のとおりです。

- 初期化パラメータを使用して、回復の際に処理される REDO ログ・レコードとデータ・ブロックの数に影響を与える
- REDO ログ・ファイルのサイズを変更して、チェックポイントの頻度に影響を与える
- SQL 文を使用してチェックポイントを開始する
- インスタンス回復操作をパラレル化して、回復時間をさらに短縮する

Oracle8i Enterprise Edition でも、インスタンス回復を制御するファースト・スタート・リカバリ機能が提供されます。



## インスタンスとクラッシュの回復時間に影響を与える初期化パラメータの使用

回復時には、Oracle は主に次の 2 つの作業を実行します。

- 変更された部分を判別するための REDO ログの読み込み
- 変更を適用するかどうかを判別するためのデータ・ブロックの読み込み

Oracle がチェックポイントを実行する頻度に影響を与えるために、表 25-1 に示す次の 3 つの初期化パラメータを使用できます。

**表 25-1 チェックポイントに影響する初期化パラメータ**

パラメータ	目的
LOG_CHECKPOINT_TIMEOUT	最新の REDO レコードとチェックポイントの間の秒数を制限します。
LOG_CHECKPOINT_INTERVAL	最新の REDO レコードとチェックポイントの間の REDO レコード数を制限します。
FAST_START_IO_TARGET	インスタンス回復時に Oracle が処理するデータ・ブロック数を制御することによってインスタンス回復時間を制限します。

---

**注意：** FAST\_START\_IO\_TARGET パラメータを使用できるのは、Oracle8i Enterprise Edition だけです。

---

### 回復に影響を与える LOG\_CHECKPOINT\_TIMEOUT の使用

初期化パラメータ LOG\_CHECKPOINT\_TIMEOUT を値  $n$  (ここで  $n$  は整数) に設定すると、最新のチェックポイントの位置は最新の REDO ブロックから  $n$  秒以内にあることが必要になります。つまり、最新のチェックポイント位置と最後の REDO ログの間で、最大で  $n$  秒のログ・アクティビティが発生する可能性があるということです。これによって、チェックポイント位置が最新 REDO ブロックと一定の間隔を保つようになります。

また、LOG\_CHECKPOINT\_TIMEOUT は、バッファがキャッシュ内で使用済みの状態になってから、DBW $n$  がバッファをディスクに書き出すまでの時間の上限を指定すると解釈することもできます。たとえば、LOG\_CHECKPOINT\_TIMEOUT を 60 に設定すると、60 秒を超えて使用済みのままキャッシュに残るバッファはなくなります。LOG\_CHECKPOINT\_TIMEOUT のデフォルト値は 1800 です。

---

**注意：** Standard Edition での LOG\_CHECKPOINT\_TIMEOUT の最小値は 900 です。Standard Edition で 900 未満の値を設定すると 900 に実行されます。

---

## 回復に影響を与える LOG\_CHECKPOINT\_INTERVAL の使用

初期化パラメータ LOG\_CHECKPOINT\_INTERVAL を値  $n$  (ここで  $n$  は整数) に設定すると、チェックポイントの位置は最新の REDO ブロックから  $n$  ブロック以内にあることが必要になります。つまり、チェックポイント位置と REDO ログに書き込まれた最後のブロックの間には、最大でも  $n$  個の REDO ブロックしか存在しないということです。結果として、チェックポイントと最後のログの間に存在できる REDO ブロック数を制限することになります。

Oracle では LOG\_CHECKPOINT\_INTERVAL の最大値は最小ログの 90% までに制限して、" ログ・ラップ " を解消するのに十分なほどチェックポイントが実行されるようにしています。ログ・ラップが発生するのは、使用可能な最後の REDO ログ・ファイルにデータを移しているときに、チェックポイントが十分に実行されていなかったために別のログ・ファイルに書き込めない場合です。チェックポイントがログの最後から離れすぎないようにすることで、Oracle はログを切り替えるのに、チェックポイントの実行を待機する必要がなくなります。

LOG\_CHECKPOINT\_INTERVAL は REDO ブロックで指定されます。REDO ブロックのサイズは、オペレーティング・システムのブロックと同じです。VSINSTANCE\_RECOVERY の LOG\_FILE\_SIZE\_REDO\_BLKs 列を使用して、最小のログ・ファイル・サイズの 90% に相当する REDO ブロック数を確認してください。

## インスタンスの回復時間に影響を与える FAST\_START\_IO\_TARGET の使用

初期化パラメータ FAST\_START\_IO\_TARGET を使用できるのは、Oracle8i Enterprise Edition を使用する場合だけです。このパラメータを  $n$  (ここで  $n$  は整数) に設定すると、クラッシュまたはインスタンスの回復時に Oracle が処理するバッファ数を  $n$  に制限できます。回復時に処理される I/O 数は回復時間と密接な相関関係があるので、FAST\_START\_IO\_TARGET パラメータを使用すると回復時間の制御が最も厳密に行えます。

DBW $n$  は FAST\_START\_IO\_TARGET の値を使用して書き込み量を決めるため、FAST\_START\_IO\_TARGET を使用するとチェックポイントが実行されます。ユーザーがデータベースに対して更新を多く行うと仮定する場合は、このパラメータに小さな値を設定すると DBW $n$  は変更済みのバッファをディスクに書き込みます。チェックポイントが実行するにつれて、CKPT プロセスはこの進捗を反映します。もちろん、ユーザー・アクティビティが低いか存在しない場合は、DBW $n$  には書き込むバッファがないので、チェックポイントは実行されません。

FAST\_START\_IO\_TARGET の値を小さくすると、回復を必要とするブロックが減少するため回復のパフォーマンスは良くなります。ただし、このパラメータの値を小さくすると、DBW $n$  がディスクに書き込むバッファの量と回数が増えるので、通常処理時のオーバーヘッドは大きくなります。

**関連項目：** 詳細は、「[回復時間の見積り](#)」25-3 ページおよび「[パフォーマンスのオーバーヘッドの計算](#)」25-10 ページを参照してください。初期化パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## チェックポイントの頻度に影響を与える REDO ログ・サイズの使用

REDO ログ・ファイルのサイズは、チェックポイントのパフォーマンスに直接影響を与えます。最小のログのサイズが小さくなると、使用済みバッファをディスクに書き込む回数が多くなり、現在のログが満杯になる前にチェックポイントの位置が現在のログにまで実行されるようになります。Oracle はこの動作を実現するために、チェックポイントと最新 REDO レコードの間の REDO ブロック数が、最小のログ・サイズの 90% 未満になるようにします。

データベースに対して行われる変更数に比べて REDO ログが小さい場合は、ログを頻繁に切り替える必要があります。LOG\_CHECKPOINT\_INTERVAL の値が最小のログ・サイズの 90% 未満であれば、このパラメータはチェックポイントの動作に最も大きな影響を与えます。

データベース作成時にオンライン REDO ログ・ファイルの数とサイズを指定しますが、起動後にも REDO ログ・ファイルの特徴を変更できます。ALTER DATABASE コマンドの ADD LOGFILE 句を使用して、REDO ログ・ファイルの追加やサイズの指定を行います。または、DROP LOGFILE 句を使用して REDO ログを削除します。

VSINSTANCE\_RECOVERY 動的パフォーマンスの LOG\_FILE\_SIZE\_REDO\_BLKs 列に REDO ログのサイズが表示されます。この値は、最小のオンライン REDO ログのサイズがチェックポイントに影響を与える方法を示します。オンライン REDO ログのサイズを変更すると、チェックポイントの書き込み頻度が間接的に影響されます。

**関連項目：** インスタンス回復をチューニングするための VSINSTANCE\_RECOVERY ビューの使用の詳細は、「[回復時間の見積り](#)」25-8 ページを参照してください。

## チェックポイントを開始するための SQL 文の使用

初期化パラメータの設定や REDO ログ・ファイルのサイズ変更の他に、SQL 文を使用してチェックポイントに影響を与えることもできます。Oracle は、ALTER SYSTEM CHECKPOINT ではノードに対するチェックポイントを記録し、ALTER SYSTEM CHECKPOINT GLOBAL ではクラスタ内のすべてのノードに対するチェックポイントを記録します。

SQL で指定されたチェックポイントはシステム負荷が大です。つまり、チェックポイントはすべての REDO スレッドが共有する制御ファイルに記録されます。また、データファイル・ヘッダーも更新されます。SQL で指定されたチェックポイントによって、コマンドが起動されたときのログの最後に相当するポイントにチェックポイントの位置が移されます。データファイルへの書き込みの追加によってシステム・オーバーヘッドが増加するので、このようなチェックポイントはパフォーマンスに悪影響を与えることがあります。

**関連項目：** これらの文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## インスタンス回復の監視

V\$INSTANCE\_RECOVERY ビューを使用して回復パラメータの現在の設定を確認します。このビューの統計を使用して、チェックポイントに最も大きな影響を持つパラメータを計算して求めることもできます。V\$INSTANCE\_RECOVERY には表 25-2 に示す列が含まれます。

表 25-2 V\$INSTANCE\_RECOVERY

列	説明
RECOVERY_ESTIMATED_IOS	ファースト・スタート・チェックポイントのパラメータのメモリー内の値に基づいた、回復時に処理されるデータ・ブロック数の見積り。
ACTUAL_REDO_BLOCKS	回復に必要な REDO ブロックの現在の数。
TARGET_REDO_BLOCKS	回復時に処理される REDO ブロックの最大数の目標。この値はこの後の 4 つの列の最大値です。
LOG_FILE_SIZE_REDO_BLOCKS	ログ切替えがチェックポイントのために待機する必要があることを保証する、回復時に処理される REDO ブロック数。
LOG_CHKPT_TIMEOUT_REDO_BLOCKS	LOG_CHECKPOINT_TIMEOUT を満たす、回復時に処理される必要がある REDO ブロック数。
LOG_CHKPT_INTERVAL_REDO_BLOCKS	LOG_CHECKPOINT_INTERVAL を満たす、回復時に処理される必要がある REDO ブロック数。
FAST_START_IO_TARGET_REDO_BLOCKS	FAST_START_IO_TARGET を満たす、回復時に処理される必要がある REDO ブロック数。

TARGET\_REDO\_BLOCKS 列に現れる値と同じ値が、このビューのもう 1 つの列に現れます。その別の列は、回復時に処理される REDO ブロックの最大数を決定するパラメータまたはログ・ファイルに対応しています。この列のパラメータの設定によって、REDO ブロックに対する最も強い要件が課せられます。

### チェックポイントに対する最大の影響を判別する：例

例として、次のような初期化パラメータの設定を仮定します。

```
FAST_START_IO_TARGET = 1000
LOG_CHECKPOINT_TIMEOUT = 1800 # default
LOG_CHECKPOINT_INTERVAL = 0 # default: disabled interval checkpointing
```

次の問合せを実行します。

```
SELECT * FROM V$INSTANCE_RECOVERY;
```

Oracle から次の結果が戻されます。

RECOVERY_ESTIMATED_IOS	ACTUAL_REDO_BLK	TARGET_REDO_BLK	LOG_FILE_SIZE_REDO_BLK	LOG_CHKPT_TIMEOUT_REDO_BLK	LOG_CHKPT_INTERVAL_REDO_BLK	FAST_START_IO_TARGET_BLK
1025	6169	4215	55296	35485	4294967295	4215

1 row selected.

最後の 3 つの列の値からわかるように、FAST\_START\_IO\_TARGET パラメータは他の 2 つのパラメータよりも Oracle に対して強く回復を要求します。つまり、このパラメータは、回復時に Oracle が処理する REDO ブロックは 4,215 未満であることを要求します。LOG\_FILE\_SIZE\_REDO\_BLK 列は、Oracle が回復時に最大 55,296 ブロックを処理できることを示します。このため、ログ・ファイルのサイズはチェックポイントに対する最大の影響にはなりません。

**注意：** LOG\_CHKPT\_INTERVAL\_REDO\_BLK の値 4294967295 は、可能な最大値に相当し、この列がチェックポイントに対して最大の影響を持たないことを示します。

TARGET\_REDO\_BLK 列は、最後の 5 つの列の最小値を表示します。これは、Oracle のチェックポイント処理に対して最大の要件を課すパラメータすなわち条件を示します。この例では、FAST\_START\_IO\_TARGET パラメータが 4,215 という値によって最大の影響を持ちます。

データベースでいくつかの更新を行ったと仮定し、3 時間後に V\$INSTANCE\_RECOVERY を問い合わせます。Oracle から次の結果が戻されます。

RECOVERY_ESTIMATED_IOS	ACTUAL_REDO_BLK	TARGET_REDO_BLK	LOG_FILE_SIZE_REDO_BLK	LOG_CHKPT_TIMEOUT_REDO_BLK	LOG_CHKPT_INTERVAL_REDO_BLK	FAST_START_IO_TARGET_BLK
1022	916	742	55296	44845	4294967295	742

1 row selected.

FAST\_START\_IO\_TARGET が、チェックポイント動作に対して依然として最も強い影響を及ぼしますが、このターゲットに対応する REDO ブロック数も大きく変わりました。この変化は、FAST\_START\_IO\_TARGET の変化やそれに対応する RECOVERY\_ESTIMATED\_IOS の変化のせいではありません。これは、回復時に I/O を必要とする操作が REDO ログで増えたため、現在同じ FAST\_START\_IO\_TARGET に対応する REDO ブロックが少なくなったことを意味します。

Oracle が回復時に処理する REDO ブロックの最大数に対して FAST\_START\_IO\_TARGET によって過度の制限を設けることにしたと仮定します。FAST\_START\_IO\_TARGET を 8000 に調整し、LOG\_CHECKPOINT\_TIMEOUT を 60 に設定し、いくつかの更新を実行します。VSINSTANCE\_RECOVERY に問合せを再発行すると、Oracle が次の結果を戻します。

RECOVERY_ ESTIMATED_ IOS	ACTUAL_ REDO_BKLS	TARGET_ REDO_BKLS	LOG_FILE_ SIZE_REDO_ BKLS	LOG_CHKPT_ TIMEOUT_ REDO_BKLS	LOG_CHKPT_ INTERVAL_ REDO_BKLS	FAST_START_ IO_TARGET_ BKLS
1640	6972	6707	55296	6707	4294967295	10338

1 row selected.

TARGET\_REDO\_BKLS 列の値 6707 は LOG\_CHKPT\_TIMEOUT\_REDO\_BKLS 列の値に対応するため、現在は LOG\_CHECKPOINT\_TIMEOUT がチェックポイントの動作に対して最も強い影響を及ぼします。

回復時間の見積り

VSINSTANCE\_RECOVERY ビューの統計を使用して、次の計算式を使用して回復時間を見積ります。

RECOVERY\_ESTIMATED\_JOBS

システムで実行できる 1 秒あたりの最大 I/O 数

たとえば、RECOVERY\_ESTIMATED\_IOS が 2,500 で、システムが実行する書込みの最大数が 1 秒あたり 500 回の場合、回復時間は 5 秒になります。次の制限事項に注意してください。

- システムが実行できる 1 秒あたりの最大 I/O 回数を正確に測定するのは困難である
- 回復時にシステムが最大の I/O 速度を保つ保証はない
- 回復時間の見積りが有効となるのは、FAST\_START\_IO\_TARGET が使用可能で、かつこのパラメータがチェックポイントの動作に決定的な影響を及ぼす場合に限られる

回復時間を調整するには、チェックポイントに最も大きな影響を持つ初期化パラメータを変更します。「[インスタンス回復の監視](#)」25-6 ページで説明したように VSINSTANCE\_RECOVERY ビューを使用して、調整するパラメータを判別します。その後、パラメータを調整して、必要に応じて回復時間を短縮または延長します。

回復時間の調整 : 例

例として、「[チェックポイントに対する最大の影響を判別する : 例](#)」25-6 ページのように、次のような初期化パラメータの設定を仮定します。

```
FAST_START_IO_TARGET = 1000
LOG_CHECKPOINT_TIMEOUT = 1800 # default
LOG_CHECKPOINT_INTERVAL = 0 # default: disabled interval checkpointing
```

次の問合せを実行します。

```
SELECT * FROM V$INSTANCE_RECOVERY;
```

Oracle から次の結果が戻されます。

RECOVERY_ ESTIMATED_ IOS	ACTUAL_ REDO_BKLS	TARGET_ REDO_BKLS	LOG_FILE_ SIZE_REDO_ BKLS	LOG_CHKPT_ TIMEOUT_ REDO_BKLS	LOG_CHKPT_ INTERVAL_ REDO_BKLS	FAST_START_ IO_TARGET_ BKLS
1025	6169	4215	55296	35485	4294967295	4215

1 row selected.

25-8 ページの式を使用して回復時間を計算します。ここで、RECOVERY\_ESTIMATED\_JOBS は 1025、システムが実行できる 1 秒あたりの最大 I/O 回数は 500 です。

$$\frac{1025}{500} = 2.05$$

2.05 をわずかに上回る回復時間が可能であると決定します。データに常にアクセスすることはクリティカルではありません。パラメータ FAST\_START\_IO\_TARGET の値を 2000 に増やして、更新をいくつか実行します。問合せを再発行すると、表示は次のようになります。

RECOVERY_ ESTIMATED_ IOS	ACTUAL_ REDO_BKLS	TARGET_ REDO_BKLS	LOG_FILE_ SIZE_REDO_ BKLS	LOG_CHKPT_ TIMEOUT_ REDO_BKLS	LOG_CHKPT_ INTERVAL_ REDO_BKLS	FAST_START_ IO_TARGET_ BKLS
2007	8301	8012	55296	40117	4294967295	8012

1 row selected.

同じ計算式を使用して再び回復時間を計算します。

$$\frac{2007}{500} = 4.01$$

回復時間は 1.96 秒増加しました。もっと時間がかかってもよい場合は、受け入れられる回復時間になるまでこの手順を繰り返します。

パフォーマンスのオーバーヘッドの計算

パフォーマンスのオーバーヘッドを計算するには、V\$SYSSTAT ビューを使用します。たとえば、次の問合せを実行するとします。

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ( 'PHYSICAL READS', 'PHYSICAL WRITES', );
```

Oracle から次の結果が戻されます。

NAME	VALUE
physical reads	2376
physical writes	14932
physical writes non checkpoint	11165
3 rows selected.	

最初の行には、ディスクから取り出されるデータ・ブロック数が表示されます。2 番目の行には、ディスクに書き込まれるデータ・ブロック数が表示されます。最後の行には、チェックポイントをオフにした場合に発生するディスクへの書き込み回数の値が表示されます。

このデータを使用して、FAST\_START\_IO\_TARGET 初期化パラメータを設定することでかかるオーバーヘッドを計算します。余分の書き込みの割合を効果的に測定するために、異なる時点における統計の値を T\_1 と T\_2 とマークします。次の計算式を使用します。変数は次の内容を表します。

変数	定義
*_1	時刻 T_1 における接頭辞付き変数の値。T_1 はデータベースが実行してしばらく経過したとき
*_2	時刻 T_2 における接頭辞付き変数の値。T_2 は T_1 よりも後で、チェックポイント・パラメータのいずれかを変更してから少し経過したとき
PWNC	チェックポイントではない物理書き込み
PW	物理書き込み
PR	物理読み込み



変数	定義
EIO	チェックポイントを使用可能にすることで生成される余分な I/O の割合の見積り

次の計算式を使用して、ファースト・スタート・チェックポイントによって生成される余分な I/O の割合を計算します。

$$(((PW\_2 - PW\_1) - (PWNC\_2 - PWNC\_1)) / ((PR\_2 - PR\_1) + (PW\_2 - PW\_1))) \times 100\% = EIO$$

インスタンスの起動または初期化パラメータの動的な変更後は、データベース統計が安定するまで少し時間がかかることがあります。そのような操作の後では、すべてのブロックがバッファ・キャッシュから少なくとも 1 回は除去されるまで待ってから測定を行ってください。

余分な I/O の割合が高い場合は、FAST\_START\_IO\_TARGET の値を増やします。「[チェックポイントに対する最大の影響を判別する : 例](#)」25-6 ページで説明したように、V\$INSTANCE\_RECOVERY の RECOVERY\_ESTIMATED\_IOS の値が受入れ可能になるまでこのパラメータを調整します。

FAST\_START\_IO\_TARGET をゼロ以外の値に設定することで発生する余分な書込みの数は、アプリケーションによって異なります。同じバッファを繰り返して修正するアプリケーションでは、そうでないアプリケーションよりもファースト・スタート・チェックポイントによる書込み回数が多くなります。余分な書込みの問題はキャッシュ・サイズには関係ありません。

### パフォーマンスのオーバーヘッドの計算 : 例

例として、次のような初期化パラメータの設定を仮定します。

```
FAST_START_IO_TARGET = 2000
LOG_CHECKPOINT_TIMEOUT = 1800 # default
LOG_CHECKPOINT_INTERVAL = 0 # default: disabled interval checkpointing
```

統計が安定したら、V\$SYSSTAT に次の問合せを発行します。

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('PHYSICAL READS', 'PHYSICAL WRITES',
'PHYSICAL WRITES NON CHECKPOINT');
```

Oracle から次の結果が戻されます。

Name	Value
physical reads	2376
physical writes	14932
physical writes non checkpoint	11165

3 rows selected.

数時間、更新を行った後で、問合せを再発行すると、Oracle から次の結果が戻されます。

Name	Value
physical reads	3011
physical writes	17467
physical writes non checkpoint	13231

3 rows selected.

25-10 ページの説明のように、SELECT 文から戻された値を計算式にあてはめて、パフォーマンスのオーバーヘッドがどれくらい発生しているかを判別します。

$$(((17467 - 14932) - (13231 - 11165)) / ((3011 - 2376) + (17467 - 14932))) \times 100\% = 14.8\%$$

この結果が示すように、ファースト・スタート・チェックポイントを使用可能にすることで、ファースト・スタート・チェックポイントを使用可能にできなかった場合に比べて I/O がおよそ 15% 増加しました。余分な I/O を計算した後で、回復時間を短縮した場合に、さらに大きなシステム・オーバーヘッドを負担できるかどうかを判断します。

回復時間を短縮するには、パラメータ FAST\_START\_IO\_TARGET の値を 1000 に減らします。バッファ・キャッシュ内の項目が除去された後で、1 秒間隔で V\$SYSSTAT 統計を計算し、新たなパフォーマンス・オーバーヘッドを判別します。V\$SYSSTAT を問い合わせます。

```
SELECT NAME, VALUE FROM V$SYSSTAT
WHERE NAME IN ('PHYSICAL READS', 'PHYSICAL WRITES',
'PHYSICAL WRITES NON CHECKPOINT');
```

Oracle から次の結果が戻されます。

Name	Value
physical reads	4652
physical writes	28864
physical writes non checkpoint	21784

3 rows selected.

更新を行った後で、問合せを再発行すると、Oracle から次の結果が戻されます。

Name	Value
physical reads	6000
physical writes	35394
physical writes non checkpoint	26438

3 rows selected.

この 2 つの SELECT 文から戻される値を使用して、パフォーマンスのオーバーヘッドがどれくらい発生しているかを計算します。

$$[(35394 - 28864) - (26438 - 21784)] / ((6000 - 4652) + (35394 - 28864)) \times 100\% = 23.8\%$$

パラメータを変更した後では、Oracle が実行する I/O の割合は、ファースト・スタート・チェックポイントを使用禁止にした場合よりもおよそ 24% 増加しました。

## インスタンス回復のフェーズのチューニング

チェックポイントを使用してインスタンス回復をチューニングする他に、インスタンス回復のロール・フォワードおよびロールバックのフェーズで、さまざまなパラメータを使用して Oracle の動作を制御することができます。場合によっては、操作をパラレル化して回復の効率を向上できます。

この項のトピックは次のとおりです。

- [ロール・フォワード・フェーズのチューニング](#)
- [ロールバック・フェーズのチューニング](#)

## ロール・フォワード・フェーズのチューニング

パラレル・ブロック回復を使用して、回復のロール・フォワード・フェーズをチューニングします。パラレル・ブロック回復は " 作業分割 " アプローチを使用して、回復のロール・フォワード・フェーズでさまざまなプロセスをさまざまなデータ・ブロックに割り当てます。たとえば、REDO ログが多数のエントリを含む場合、プロセス 1 はログ・ファイルの一部を処理し、プロセス 2 が別の部分を処理し、プロセス 3 がまた別の部分を処理します。多数のデータファイルがさまざまなディスク・ドライブに存在する、クラッシュ、インスタンスおよびメディアの回復は、パラレル・ブロック回復に適しています。

RECOVERY\_PARALLELISM 初期化パラメータを使用して、インスタンスまたはメディア回復操作での同時回復プロセス数を指定します。クラッシュ回復はインスタンス起動時に行われるため、このパラメータはクラッシュ回復で使用するプロセス数を指定するのにも有効です。RECOVER コマンドの PARALLEL 句を指定していない場合は、このパラメータの値は、メディア回復で使用されるプロセス数のデフォルトになります。このパラメータの値は 1 より大きくなければなりませんが、PARALLEL\_MAX\_SERVERS パラメータの値を超えることはできません。パラレル・ブロック回復がシリアル回復の効率を上回るためには、少なくとも 8 個の回復プロセスが必要です。

回復は、通常はデータ・ブロックの読み込みで I/O バウンドになります。その結果、ブロック・レベルのパラレル化によって合計 I/O が増加するとしても、回復のパフォーマンスに役立つだけです。つまり、ブロック・レベルのパラレル化によって、非同期 I/O に関するオペレーティング・システムの制約が回避されます。通常は、効果的な非同期 I/O を備えたシステムのパフォーマンスは、パラレル・ブロック回復を使用しても大幅には改善されません。

## ロールバック・フェーズのチューニング

インスタンス回復の 2 番目のフェーズでは、未コミットのトランザクションがロールバックされます。Oracle では、ファースト・スタート・オンデマンド・ロールバックとファースト・スタート・パラレル・ロールバックという 2 つの機能を使用して、この回復フェーズの効率を上げます。

---

**注意：** これらの機能はファースト・スタート・リカバリに含まれ、Oracle8i Enterprise Edition でのみ使用可能です。

---

この項のトピックは次のとおりです。

- [ファースト・スタート・オンデマンド・ロールバックの使用](#)
- [ファースト・スタート・パラレル・ロールバックの使用](#)

### ファースト・スタート・オンデマンド・ロールバックの使用

ファースト・スタート・オンデマンド・ロールバック機能を使用すると、回復のロール・フォワード・フェーズが終了するとすぐに新しいトランザクションを自動的に開始できます。障害が発生したトランザクションによってロックされている行をユーザーがアクセスしようとする、Oracle はそのトランザクションを完了するために必要な変更だけをロールバックします。つまり、要求時 (" オンデマンド ") にロールバックを行います。その結果、新しいトランザクションは、長いトランザクションの全体がロールバックされるまで待機する必要がありません。

---

**注意：** Oracle ではこれは自動的に行われます。この機能を使用するため、パラメータの設定やコマンドの発行を行う必要はありません。

---

### ファースト・スタート・パラレル・ロールバックの使用

ファースト・スタート・パラレル・ロールバックでは、バックグラウンド・プロセス SMON はコーディネータとして作動し、複数のサーバー・プロセスを使用してパラレルで一連のトランザクションをロールバックします。基本的に、ファースト・スタート・パラレル・ロールバックは、ロールバックを対象とし、パラレル・ブロック回復はロール・フォワードを対象とします。

ファースト・スタート・パラレル・ロールバックが主に役立つのは、特にパラレルの INSERT、UPDATE および DELETE 操作をコミットするまでに長時間実行するトランザクションがシステムにある場合です。SMON は、回復作業の量が一定のしきい値を超えていると判断した場合、作業をいくつかのパラレル・プロセスに分散することによって、パラレル・ロールバックを自動的に開始します。つまり、プロセス 1 が 1 つのトランザクションをロールバックし、プロセス 2 が 2 番目のトランザクションをロールバックするというよう

に、順にロールバックします。しきい値は、パラレル回復のコストが有効になる分岐点です。つまり、パラレル回復がシリアル回復よりも時間がかからなくなるところです。

ファースト・スタート・パラレル・ロールバックの特殊な形式の1つは、トランザクション内回復です。トランザクション内回復では、1つのトランザクションがいくつかのプロセスに分割されます。たとえば、8つのトランザクションが、各トランザクションに1つのパラレル・プロセスを割り当てて回復を行う必要があるとします。これらのトランザクションのサイズはすべて同じくらいですが、トランザクション5は非常にサイズが大きくなっています。このトランザクションをロールバックするプロセスは、他のトランザクションをロールバックするプロセスよりも時間がかかることになります。

このような状況では、トランザクション5をいくつかのプロセスに分散することによって、トランザクション内回復が自動的に開始されます。プロセス1が1部分を処理し、プロセス2が別の部分を処理します。プロセス3以降も同様です。

パラメータ `FAST_START_PARALLEL_ROLLBACK` を次の3つの値のいずれかに設定することによって、トランザクション回復に関係するプロセス数を制御します。

FALSE	ファースト・スタート・パラレル・ロールバックをオフにします。
LOW	回復サーバー数が <code>CPU_COUNT</code> パラメータの値の2倍を超えないことを指定します。
HIGH	回復サーバー数が <code>CPU_COUNT</code> パラメータの値の4倍を超えないことを指定します。

**OPS 構成でのパラレル・ロールバック** OPS では、ファースト・スタート・パラレル・ロールバックを各インスタンスで実行できます。各インスタンス内では、次のようなトランザクションでパラレル・ロールバックを実行できます。

- 指定のインスタンスでオンラインになるトランザクション
- オフラインであり、指定のインスタンス以外のインスタンスでは回復されないトランザクション

指定のインスタンスについてロールバック・セグメントがいったんオンラインになると、そのインスタンスだけがそのセグメントのトランザクションについてパラレル・ロールバックを実行できます。

**ファースト・スタート・パラレル・ロールバックの進捗の監視** ファースト・スタート・パラレル・ロールバックの進捗を監視するには、`V$FAST_START_SERVERS` 表および `V$FAST_START_TRANSACTIONS` 表を調べます。`V$FAST_START_SERVERS` は、ファースト・スタート・パラレル・ロールバックを処理するすべての回復についての情報を提供します。`V$FAST_START_TRANSACTIONS` には、トランザクションの進捗についての情報が含まれます。

**関連項目：** OPS 環境でのファースト・スタート・パラレル・ロールバックの詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。初期化パラメータの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## 透過的アプリケーション・フェイルオーバー

この項のトピックは次のとおりです。

- [透過的アプリケーション・フェイルオーバーとは](#)
- [透過的アプリケーション・フェイルオーバーの仕組み](#)
- [透過的アプリケーション・フェイルオーバーについて（DBA 向け）](#)
- [透過的アプリケーション・フェイルオーバーについて（アプリケーション開発者向け）](#)
- [透過的アプリケーション・フェイルオーバーの制限事項](#)

---

**注意：** 透過的アプリケーション・フェイルオーバーを使用するには、Oracle8i Enterprise Edition が必要です。

---

## 透過的アプリケーション・フェイルオーバーとは

透過的アプリケーション・フェイルオーバー（TAF）は、接続に障害が発生したときに、アプリケーションが自動的にデータベースに再接続する機能です。クライアントがデータベース・トランザクションに関係していない場合は、ユーザーはサーバーの障害に気付かないことがあります。この再接続は OCI ライブラリから自動的に行われるので、TAF を使用するためにクライアント・アプリケーション・コードを変更する必要はないでしょう。

## 透過的アプリケーション・フェイルオーバーの仕組み

通常のクライアント / サーバー・データベース操作では、クライアントがデータベースへの接続をメンテナンスし、クライアントとサーバーが通信を行えます。サーバーに障害が発生すると、通信にも障害が発生します。たとえば、その後、クライアントが接続を使用して新たに SQL 文を実行しようとする、オペレーティング・システムのエラーがクライアントに表示されます。Oracle では、通常このような場合にはエラー "ORA-3113: end-of-file on communication channel" を発行します。この時点で、ユーザーはデータベースに再びログインする必要があります。

ただし、TAF を使用すると、Oracle が自動的にデータベースへの新たな接続を取得します。これによって、ユーザーは、最初の接続に障害が発生しなかったかのように、新たな接続を使用して作業を継続することができます。

アクティブなデータベース接続に関連するいくつかの要素があります。これには次のものがあります。

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

TAF は自動的にこれらの要素の一部を復元します。ただし、その他の要素はアプリケーション・コードに埋め込んで、TAF によって接続を回復できるようにする必要があります。

## クライアント / サーバー・データベース接続

TAF は自動的にデータベース接続を再確立します。デフォルトでは、TAF は、同じ接続文字列を使用して新しいデータベース接続を獲得しようとします。または、フェイルオーバーでは別の接続文字列を使用するように構成することもできます。つまり、代用フェイルオーバー接続を事前に確立しておくこともできます。この構成の詳細は、「[アプリケーション・フェイルオーバーの構成](#)」25-20 ページを参照してください。

## ユーザーのデータベース・セッション

TAF は、自動的に、障害前に使用していたのと同じユーザー ID を使用してユーザーをログインさせます。複数のユーザーが接続を使用していた場合は、TAF は、それらのユーザーがデータベース・コマンドの処理を試行するときに自動的にログインさせます。あいにく、TAF は他のセッション・プロパティを自動的に復元することはできません。アプリケーションが ALTER SESSION コマンドを発行していた場合は、TAF の処理が完了した後にはアプリケーションでそれらのコマンドを再発行する必要があります。これは、フェイルオーバー・コールバック処理で行われます。これは『Oracle8i コール・インタフェース・プログラマーズ・ガイド』で詳しく説明します。

## コマンドの実行

通常、クライアントが接続の障害に気付くのは、コマンドをサーバーに発行して結果がエラーになったときです。クライアントは、サーバー障害の前にコマンドが完全に実行されたかどうか判別できません。コマンドが完全に実行され、データベースの状態が変更された場合は、コマンドは再送信されません。データベースを変更したコマンドに応答して TAF が再接続した場合は、TAF がアプリケーションに "ORA-25408: can not safely replay call" というメッセージを発行します。

TAF は、フェイルオーバーの後で自動的に SELECT コマンドおよび FETCH コマンドをデータベースに再送信します。この種のコマンドはデータベースの状態を変更しないためです。

## フェッチに使用されるオープン・カーソル

TAF は、フェイルオーバーの前にカーソルによる行のフェッチを開始したアプリケーションが、フェイルオーバーの後に行のフェッチを続けられるようにします。これは、"SELECT" フェイルオーバーと呼ばれます。同じスナップショットを使用して SELECT 文を再実行し、同数の行を取り出すことによってこれを達成します。

また、TAF は、選択結果が一貫性を持つことを保証する保護対策を提供します。この保護対策が失敗すると、アプリケーションはエラー・メッセージ "ORA-25401 can not continue fetches" を受け取ります。

## アクティブなトランザクション

TAF はフェイルオーバー後にアクティブなトランザクションを保存できないので、障害時にはアクティブなトランザクションはロールバックされます。そのかわり、ROLLBACK が送られるまでアプリケーションはエラー・メッセージ "ORA-25402 transaction must roll back" を受け取ります。

## サーバー側のプログラム変数

PL/SQL パッケージの状態などのサーバー側のプログラム変数は障害時に失われ、TAF によって回復することはできません。これらの変数は、フェイルオーバー・コールバックからコールすることで初期化できます。これは、『Oracle8i コール・インタフェース・プログラマーズ・ガイド』で詳しく説明します。

## 透過的アプリケーション・フェイルオーバーの実現例

TAF がデータベース障害を効果的に隠すためには、クライアントが再接続できる場所が必要になります。この項では、次のデータベース構成と TAF との関係について説明します。

- OPS
- フェイル・セーフ・システム
- レプリケート・システム
- スタンバイ・データベース
- 単一インスタンス Oracle データベース

### OPS

TAF は、最初は Oracle Parallel Server 環境のために開発されました。TAF のすべての機能は OPS で作動し、特別な設定は必要ありません。OPS の詳細は、『Oracle8i Parallel Server セットアップおよび構成ガイド』を参照してください。



## フェイル・セーフ・システム

TAF は Oracle フェイル・セーフで使用できます。ただし、プライマリ・データベースに障害が発生したときに、バックアップ・インスタンスは接続を受けることができないので、一部のクライアントは、データベース・サーバーが使用不可能なときに再接続を試みる場合があります。フェイルオーバー・コールバックを使用してこれを処理できます。フェイルオーバー・コールバックの詳細は、『Oracle8i 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。

## レプリケート・システム

TAF は、すべてのデータベース・オブジェクトが両方のレプリケーションで同じであれば、レプリケート・システムといっしょに作動します。これは、パスワードが同じということも含みます。表のデータが互いにわずかも同期が取れていないと、ほとんどの場合に "ORA-25401: can not continue fetches" が表示されます。レプリケーションの詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

## スタンバイ・データベース

TAF は、フェイル・セーフの場合と同じようにスタンバイ・データベースで作動します。データベースにクライアントがログインできない時間が発生することがあるため、フェイルオーバー・コールバックを用意する必要があります。また、最新のアーカイブ・ログよりも後で行われた変更がなくなる場合は、データ・スキューが発生し、"ORA-25401: can not continue fetches" が表示される可能性が高くなります。

## 単一インスタンス Oracle データベース

単一インスタンス Oracle データベース環境でも TAF を使用できます。単一インスタンス環境では障害の後に、データベースが使用できない時間が発生する可能性があり、TAF は接続を再確立できません。このため、フェイルオーバー・コールバックを使用して、定期的にフェイルオーバーを再試行できます。データベースが再び使用可能になったら、TAF は正常に接続を再確立します。

# 透過的アプリケーション・フェイルオーバーについて (DBA 向け)

この項のトピックは次のとおりです。

- TAF の構成
- クライアントによるアプリケーション・フェイルオーバー使用の判別方法
- ロード・バランシング
- 計画的シャットダウン
- TAF のチューニング

## アプリケーション・フェイルオーバーの構成

アプリケーションの接続文字列をネーム・サーバーで構成するか、TNSNAMES.ORA ファイルに入力することができます。または、接続文字列をアプリケーション内にハードコーディングすることもできます。

各アプリケーションについて、ネーム・サーバーがリスナー、インスタンス・グループおよびフェイルオーバー・モードについての情報を提供します。接続文字列 *failover\_mode* のフィールドによって、フェイルオーバーの種類と方法が指定されます。構文の詳細は、『Oracle8i Net8 管理者ガイド』を参照してください。

**TYPE: フェイルオーバー・モードの機能オプション** クライアントのフェイルオーバー機能は、接続文字列の TYPE キーワードで決まります。TYPE キーワードの選択肢は次のとおりです。

SELECT	これによって、オープン・カーソルを使用していたユーザーは、障害の後もそのカーソルでフェッチを継続できます。ただし、このモードでは通常の SELECT 操作でクライアント側にオーバーヘッドが生じます。このため、ユーザーは SELECT フェイルオーバーを使用禁止にすることができます。
SESSION	これはセッションのフェイルオーバーです。つまり、ユーザーの接続が失われると、ユーザーのためにバックアップに 2 番目のセッションが自動的に作成されます。この種のフェイルオーバーでは SELECT の回復は行われません。
NONE	これはデフォルトです。フェイルオーバー機能を使用しません。フェイルオーバーを行わないようにするために明示的に指定することもできます。

**METHOD: フェイルオーバー・モードのパフォーマンス・オプション** アプリケーション・フェイルオーバーの速度を高めると、バックアップ・インスタンスの負担が大きくなる可能性があります。DBA は、接続文字列の METHOD キーワードを使用して、BASIC または PRECONNECT のパフォーマンス・オプションを構成することができます。

BASIC	フェイルオーバー時に接続を確立する。このオプションは、フェイルオーバー時までバックアップ・サーバーでの作業はほとんど不要です。
PRECONNECT	接続を事前定義する。これはフェイルオーバーがさらに速く行われますが、バックアップ・インスタンスが、サポートするすべてのインスタンスのすべての接続をサポートできなければなりません。

**BACKUP: 代替バックアップ接続文字列** 多くの場合、最初の接続とバックアップの接続の両方で同じ接続文字列を使用することは不便です。このようなインスタンスでは、さまざまな TNS 別名を指定する BACKUP キーワードを接続文字列で使用するか、バックアップ接続のための接続文字列を明示できます。

## V\$SESSION のフェイルオーバー・フィールド

ビュー V\$SESSION には、フェイルオーバーに関連する次のフィールドがあります。

FAILED_OVER	バックアップを使用する場合は TRUE、使用しない場合は FALSE。
TYPE	SELECT、SESSION または NONE のいずれか。
METHOD	BASIC または PRECONNECT。

## 現在のトランザクションの後でのインスタンスのシャットダウン

SHUTDOWN コマンドに TRANSACTIONAL オプションを指定すると、1 インスタンスの計画的シャットダウンを実行することができ、クライアントの中断を最小限にできます。このオプションは、処理中のトランザクションが完了するのを待機します。

TRANSACTIONAL オプションは、パッチ・リリースをインストールするときに役立ちます。また、サービスを中断しないでインスタンスを停止する必要があるときにこのオプションを使用します。

待機中には、クライアントはそのインスタンスで新しいトランザクションを開始できません。トランザクションを開始しようとするクライアントは切断され、フェイルオーバーが使用可能になっている場合は、これによってフェイルオーバーが起動されます。最後のトランザクションが完了すると、プライマリ・インスタンスが SHUTDOWN IMMEDIATE を実行します。

## 現在のトランザクションの後でのセッションの切断

ALTER SYSTEM DISCONNECT SESSION POST\_TRANSACTION 文によって、現在のトランザクションが完了した後の最初のコールでセッションが切断されます。アプリケーションのフェイルオーバーは自動的に行われます。

この文は、負荷を制御する方法として TAF で作動します。1 つのインスタンスの負荷が多い場合は、このオプションを使用して一連のセッションを手動で切断できます。このオプションは、セッションが切断されるときにトランザクションが存在しないことを保証するため、ユーザーは変化に気付くことはありませんが、切断後のコマンドの実行がわずかに遅れます。この構文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## フェイルオーバーのパフォーマンスのチューニング

フェイルオーバーの経過時間には、インスタンス回復とデータベースへの再接続に必要な時間が含まれます。フェイルオーバーの最適なパフォーマンスのためには、チェックポイントを頻繁に設定することでインスタンス回復をチューニングしてください。

複数のリスナーの使用やマルチスレッド・サーバー (MTS) の使用でも、パフォーマンスが改善されます。MTS 接続は、専用サーバー経由の接続よりもかなり高速になる傾向があります。

## 透過的アプリケーション・フェイルオーバーについて (アプリケーション開発者向け)

この項では、複数ユーザー・ハンドルとフェイルオーバー・コールバックについて説明します。

### 複数ユーザー・ハンドル

フェイルオーバーは複数ユーザー・ハンドルでサポートされます。OCI では、サーバー・コンテキスト・ハンドルとユーザー・ハンドルは分離されています。サーバー・コンテキスト・ハンドルに関連する複数ユーザー・ハンドルを使用することができるので、複数ユーザーはデータベースに対する同じ接続を共有できます。

接続が破壊されると、その接続に関連するすべてのユーザーにフェイルオーバーが行われます。ただし、1 つのユーザー・セッションが破壊されても、接続は存在するのでフェイルオーバーは行われません。フェイルオーバーは、移行可能なユーザー・ハンドルを再認証しません。

**関連項目：**『Oracle8i コール・インタフェース・プログラマーズ・ガイド』

### フェイルオーバー・コールバック

1 インスタンスの障害と別のインスタンスへのフェイルオーバーには時間がかかる場合がよくあります。このような遅延が発生するため、ユーザーに対してフェイルオーバーが進行中であることを通知したい場合があります。さらに、最初のインスタンスにおけるセッションが ALTER SESSION コマンドを受け取った場合もあります。これらは、2 番目のインスタンスでは自動的に再生されません。これらのコマンドが 2 番目のインスタンスで確実に再生されるようにすることが必要な場合もあります。このような問題に対処するために、コールバック関数を登録できます。

フェイルオーバーは、ユーザー・セッションを再確立するときにコールバック関数を数回コールします。最初のコールは、インスタンスの接続障害が最初に検出されたときに、アプリケーションがユーザーに遅延を予告するために発生します。フェイルオーバーが正常に終了すると、接続が再確立され使用可能になったときに、2 番目のコールが発生します。

このときは、クライアントは ALTER SESSION 文を再生し、フェイルオーバーが発生したことをユーザーに通知するとよいでしょう。フェイルオーバーが正常に終了しなかった場合は、フェイルオーバーが発生しなかったことをアプリケーションに通知するためにコールバックがコールされます。この場合は、フェイルオーバーが再試行されるように指定できます。さらに、ユーザー・ハンドルがフェイルオーバー後に接続を使用しようとしたときに、各ユーザー・ハンドルについてコールバックがコールされます。

**関連項目：**『Oracle8i コール・インタフェース・プログラマーズ・ガイド』

## 透過的アプリケーション・フェイルオーバーの制限事項

接続が失われると、次の影響が現れます。

- サーバーのすべての PL/SQL パッケージの状態がフェイルオーバー時に失われます。
- ALTER SESSION 文が失われます。
- トランザクションの処理中にフェイルオーバーが発生すると、ユーザーが OCITransRollback コールを発行するまで、後続の各コールによってエラー・メッセージが出されます。その後、OCI 成功メッセージが発行されます。必ずこの情報メッセージを調べて、追加操作を実行する必要があるかどうかを確認してください。
- フェイルオーバーが行われたカーソルで作業を続けるとエラー・メッセージが出される場合があります。
- フェイルオーバー後の最初のコマンドが SQL SELECT 文または OCIStmtFetch 文でない場合は、エラー・メッセージが出されます。
- フェイルオーバーの効果があるのは、アプリケーションが OCI リリース 8.0 以上を使用してプログラミングされている場合のみです。
- 進行中の問合せがあると、フェイルオーバー時に再発行されもう一度最初から処理されます。元の実行に時間がかかった場合は、次の問合せにも時間がかかることがあります。



# 第 V 部

---

## パラレル実行

第 V 部では、パラレル実行の最適化について説明します。第 V 部には次の章が含まれます。

- [第 26 章「パラレル実行のチューニング」](#)
- [第 27 章「パラレル実行のパフォーマンス上の問題について」](#)





---

## パラレル実行のチューニング

パラレル実行を使用すると、通常は意思決定支援システム（DSS）に関連する大規模なデータベースでのデータ処理集中型の操作で、応答時間が大幅に削減されます。パラレル実行は、一定のタイプの OLTP（オンライン・トランザクション処理）とハイブリッド・システム上でもインプリメントできます。

この章では、パラレル実行をインプリメントする方法と、パラレル実行のパフォーマンス最適化のために、どのようにシステムをチューニングするかについて説明します。

**関連項目：** パラレル実行の基本概念は、『Oracle8i 概要』を参照してください。パラレル実行のチューニングの詳細は、プラットフォーム固有の Oracle マニュアルを参照してください。

---

**注意：** パラレル実行は、Oracle8i Enterprise Edition でのみ利用できます。

---

この章では、パラレル実行を 4 つのフェーズに分けて説明します。フェーズ 1 では、自動チューニングと手動チューニングのどちらを行うかを決定します。多くのアプリケーションでは、自動チューニングによってシステム構成に基づくデフォルトのパラメータ値を自動的に設定することで、許容できるパフォーマンスを達成できます。

フェーズ 2 では、並列性とパーティション化を利用する方法を説明します。フェーズ 2 では、自システムのニーズに基づいて、使用するのに最適なタイプの並列性を選択します。このフェーズでは、Oracle のパーティション化機能を最大限に利用する方法も説明します。

フェーズ 3 では、データベースの作成、移入およびリフレッシュの方法を説明します。フェーズ 4 では、パフォーマンスを最適化するために、パラレル実行を監視して微調整する方法を説明します。

フェーズとその内容は次のとおりです。

### フェーズ 1: パラレル実行のためのパラメータの初期化とチューニング

- ステップ 1: パラレル実行の自動または手動チューニングの選択

- ステップ 2: 並列度の設定とマルチユーザー問合せ調整の使用可能化
- ステップ 3: 一般パラメータのチューニング

フェーズ 2: パラレル実行の物理データベース・レイアウトのチューニング

- 並列性のタイプ

フェーズ 3: データベースの作成、移入およびリフレッシュ

- パラレル・ロードによるデータベースへの移入
- パラレル・ソートおよびハッシュ結合のための一時表領域の作成
- パラレルでの索引の作成
- パラレル DML のその他の考慮事項

フェーズ 4: パラレル実行のパフォーマンスの監視

- 動的パフォーマンス・ビューを使ったパラレル実行のパフォーマンスの監視
- セッション統計の監視

## パラレル実行のチューニングの概要

パラレル実行は、大量のデータにアクセスする多くのタイプの操作に役立ちます。パラレル実行によって、次のような処理が改善されます。

- 大規模な表走査および結合
- 大規模な索引の作成
- パーティション索引の走査
- 大量の挿入、更新および削除
- 集計とコピー

パラレル実行を行って、Oracle データベース内のオブジェクト・タイプにもアクセスできます。たとえば、パラレル実行を行って、LOB ( Large Object ) にアクセスできます。

次の特性をすべて備えたシステムでは、パラレル実行の効果が大きいです。

- 対称型マルチプロセッサ ( SMP )、クラスタまたは超並列システム
- 十分な I/O 帯域幅
- 十分に利用されない CPU や断続的に使用される CPU ( たとえば、CPU の標準使用率が 30% 未満のシステム )
- ソート、ハッシング、I/O バッファなどのメモリーを大量に必要とする処理の追加に応じられる十分なメモリー

システムにこれらの特性のいずれかが欠けている場合は、パラレル実行を行ってもパフォーマンスに大きな改善が見られない場合があります。実際、使用率超過のシステムまたは I/O 帯域幅の小さいシステムでは、パラレル実行を行うと、システムのパフォーマンスが低下することがあります。

---

---

**注意：** " パラレル実行サーバー " という用語は、パラレル操作を実行するサーバー・プロセス（または、Windows NT システムでは " スレッド "）を指します。これは、同じデータベースにアクセスする複数の Oracle インスタンスを表す Oracle Parallel Server とは異なります。

---

---

**関連項目：** Oracle Parallel Server の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

## パラレル実行をインプリメントするタイミング

パラレル実行は、意思決定支援システム（DSS）のパフォーマンスを大幅に改善します。オンライン・トランザクション処理（OLTP）システムもパラレル実行の効果がありますが、通常はバッチ処理の間だけです。

業務時間帯では、ほとんどの OLTP システムではおそらくパラレル実行を行うべきではありません。しかし、業務時間外は、パラレル実行によって大容量のバッチ操作を効果的に実行できます。たとえば銀行は、パラレル・バッチ・プログラムを使用して、口座に利子を加算する何百万もの更新を実行する場合があります。

パラレル実行を行う例としては、DSS の方が一般的です。複数の表の結合や非常に大規模な表の検索を必要とするような複雑な問合せは、多くの場合パラレルで実行するのが最適です。そのため、この章の残りの部分では、DSS 環境でのパラレル実行について特に詳しく説明します。

## フェーズ 1: パラレル実行のためのパラメータの初期化とチューニング

パラレル実行は、パラメータ `PARALLEL_AUTOMATIC_TUNING` を `TRUE` に設定することによって初期化し、自動的にチューニングできます。自動パラレル実行は、いったん使用可能にすると、パラレル実行に関連するすべてのパラメータの値を制御します。これらのパラメータは、サーバー処理のいくつかの側面、つまり DOP（並列度）、マルチユーザー問合せ調整機能およびメモリー・サイズ設定に影響します。

パラレル実行自動チューニング機能が使用可能になっている場合、Oracle は、システム上の CPU の数と、`PARALLEL_THREADS_PER_CPU` に設定した値に基づいて、各環境のパラメータ設定を決定します。`PARALLEL_AUTOMATIC_TUNING` が `TRUE` のときに Oracle がパラレル実行処理に対して設定するデフォルト値は、通常はほとんどの環境で最適な値です。たいていの場合、Oracle で自動的に導出された設定は、少なくとも手動で導出された設定と同程度効果的です。

パラレル実行パラメータを手動でチューニングすることもできますが、自動パラレル実行を行うことをお勧めします。パラレル実行の手動チューニングは、自動チューニングを行うよりも 2 つの理由で複雑です。つまり、パラレル実行の手動チューニングには自動チューニングよりも慎重な管理が必要であることと、手動チューニングではユーザーの負荷とシステム・リソースの計算を間違いやすいことです。

パラレル実行の初期化とチューニングには、次の見出し以降で説明する 3 つのステップが関係します。これらのステップは次のとおりです。

- [ステップ 1: パラレル実行の自動または手動チューニングの選択](#)
- [ステップ 2: 並列度の設定とマルチユーザー問合せ調整の使用可能化](#)
- [ステップ 3: 一般パラメータのチューニング](#)

ステップ 3 では、一般パラメータのチューニングについて説明します。最初の 2 つのステップを完了した後で、パラレル実行のパフォーマンスをさらにチューニングする必要がある場合は、一般パラメータ情報が役立ちます。

パラレル実行チューニングを説明するいくつかの例を、この項の最後に示します。例としてあげたシナリオでは、完全な自動システムから完全な手動システムに及ぶ広範囲の構成について説明します。

# ステップ 1: パラレル実行の自動または手動チューニングの選択

パラレル実行の初期化とチューニングを行う方法はいくつかあります。前述したように、PARALLEL\_AUTOMATIC\_TUNING を TRUE に設定することで、パラレル実行を完全に自動化した環境にすることができます。このタイプの環境は、自動的に導出された値のいくつかを上書きすることで、さらにカスタマイズできます。

PARALLEL\_AUTOMATIC\_TUNING をデフォルト値の FALSE のままにして、パラレル実行に影響するパラメータを手動で設定することもできます。通常の OLTP 環境と、パラレル実行の利益を受けないその他のタイプのシステムでは、パラレル実行を使用可能にしないでください。

**注意：** 正しく設定され、目標のリソース使用パターンを達成している手動でチューニングされたシステムは、自動パラレル実行の効果が表れないことがあります。

## 完全に自動化されたパラレル実行環境で自動的に導出されるパラメータ設定

PARALLEL\_AUTOMATIC\_TUNING が TRUE のときは、表 26-1 で示すように、Oracle が自動的に他のパラメータを設定します。ほとんどのシステムでは、さらに調整を行わなくても、十分にチューニングされた完全自動パラレル実行環境になります。

表 26-1 PARALLEL\_AUTOMATIC\_TUNING の影響を受けるパラメータ

パラメータ	デフォルト	PARALLEL_AUTOMATIC_TUNING=TRUE の場合 のデフォルト	コメント
PARALLEL_ADAPTIVE_MULTI_USER	FALSE	TRUE	-
PROCESSES	6	次の式より大きい値 : 1.2 x PARALLEL_MAX_SERVERS または PARALLEL_MAX_SERVERS + 6 + 5 + (CPU 数 x 4)	PARALLEL_AUTOMATIC_TUNING が TRUE の場合は、値が強制的に最小値にされます。
SESSIONS	(PROCESSES x 1.1) + 5	(PROCESSES x 1.1) + 5	自動パラレル・チューニングは、SESSIONS に間接的に影響します。SESSIONS を設定しない場合は、Oracle が PROCESSES の値に基づいて設定します。

表 26-1 PARALLEL\_AUTOMATIC\_TUNING の影響を受けるパラメータ

パラメータ	デフォルト	PARALLEL_AUTOMATIC_TUNING=TRUE の場合 のデフォルト	コメント
PARALLEL_MAX_SERVERS	5	CPU x 10	この制限は、パラレル実行が使用するプロセス数を最大化するために使用します。このパラメータの値はポートに固有なので、システムによって処理が異なる場合があります。
LARGE_POOL_SIZE	なし	PARALLEL_EXECUTION_POOL+ MTS ヒープ要件 + バックアップ・バッファ要求 + 600KB	Oracle は、パラレル実行バッファを SHARED_POOL から割り当てません。
PARALLEL_EXECUTION_MESSAGE_SIZE	2KB (ポート固有)	4KB (ポート固有)	Oracle はメモリーを LARGE_POOL から割り当てるので、デフォルトが増やされません。

前述したように、PARALLEL\_AUTOMATIC\_TUNING を TRUE に設定した場合でも、[表 26-1](#) で示されたパラメータを手動で調整できます。環境を高度にカスタマイズしている場合や、完全に自動化された設定を使用するとシステムが最適に実行されない場合は、手動調整が必要な場合があります。

パラレル実行は、さまざまなタイプのシステム・パフォーマンスを改善するので、この項の最後にある例を手始めとして活用ください。これらの初期設定を使用したシステムのパフォーマンスを見きわめた後で、パラレル実行のためにシステムをさらにカスタマイズできます。

## ステップ 2: 並列度の設定とマルチユーザー問合せ調整の使用可能化

このステップでは、システムの並列度（DOP）を設定し、マルチユーザー問合せ調整を使用可能にするかどうかを検討します。

### 並列度とマルチユーザー問合せ調整、およびその相互作用

DOP は、パラレル操作で使用可能なプロセス（スレッド）数を指定します。各パラレル・スレッドは、問合せの複雑さに応じて 1 セットか 2 セットの問合せプロセスを使用します。

マルチユーザー問合せ調整機能は、ユーザーの負荷に基づいて DOP を調整します。たとえば、DOP が 5 の表があるとします。この DOP は、10 人のユーザーを受け付けることができます。しかし、さらに 10 人のユーザーがシステムに入った場合に、PARALLEL\_ADAPTIVE\_MULTI\_USER 機能を使用可能にすると、Oracle は DOP を減らし、認識されているシステム負荷に従ってリソースをより均等に分散します。

---

---

**注意：** Oracle が問合せの DOP をいったん決定すると、その問合せが終わるまで DOP は変更されません。

---

---

複数のユーザーがパラレル実行操作を同時に処理するときは、パラレル・マルチユーザー問合せ調整機能を使用するのが最適です。PARALLEL\_AUTOMATIC\_TUNING を使用可能にすると、Oracle は PARALLEL\_ADAPTIVE\_MULTI\_USER を自動的に TRUE に設定します。

---

---

**注意：** 単一ユーザーやバッチ処理システムの場合、またはシステムのパフォーマンスがすでに最適化されている場合は、マルチユーザー問合せ調整を使用禁止にしてください。

---

---

### マルチユーザー問合せ調整アルゴリズムを機能させる方法

マルチユーザー問合せ調整アルゴリズムには、いくつかの入力があります。アルゴリズムは、データベース・リソース・マネージャによって計算された割当て済のスレッド数を最初に検討します。次にアルゴリズムは、INIT.ORA で設定された並列性のデフォルト設定と、CREATE TABLE コマンド、ALTER TABLE コマンドおよび SQL ヒントで使用されているパラレル・オプションを検討します。

システムが過負荷で、入力 DOP がデフォルト DOP よりも大きい場合、アルゴリズムはデフォルトの並列度を入力として使用します。それから、システムは入力 DOP に適用する縮小係数を計算します。

## 表と問合せでの並列性の使用可能化

パラレル操作に関連する表の DOP は、それらの表に対する操作の DOP に影響します。したがって、パラレル・チューニング関連のパラメータを設定した後で、CREATE TABLE または ALTER TABLE コマンドの PARALLEL オプションを使用して、パラレル化する各表についてパラレル実行を使用可能にします。SQL 文の PARALLEL ヒントを使用して、その操作に対してのみ並列性を使用可能にすることもできます。

表を並列化するとき、DOP を自分で指定することも、PARALLEL\_THREADS\_PER\_CPU の値に基づいて Oracle に自動的に設定させることもできます。

## PARALLEL\_THREADS\_PER\_CPU を使ったパフォーマンスの制御

初期化パラメータ PARALLEL\_THREADS\_PER\_CPU は、DOP とマルチユーザー問合せ調整機能の両方を制御するアルゴリズムに影響します。Oracle は、PARALLEL\_THREADS\_PER\_CPU の値にインスタンスあたりの CPU 数を乗算して、パラレル操作で使用するスレッド数を導出します。

マルチユーザー問合せ調整機能は、デフォルト DOP を使用して、システムに存在する問合せサーバー・プロセスの目標数も計算します。システムが目標数より多くのプロセスを実行していると、適応アルゴリズムは新しい問合せの DOP を必要に応じて削減します。したがって、PARALLEL\_THREADS\_PER\_CPU を使用して適応アルゴリズムを制御することもできます。

PARALLEL\_THREADS\_PER\_CPU のデフォルトは、ほとんどのシステムで適切です。ただし、I/O サブシステムがプロセッサの速度に合せられない場合は、PARALLEL\_THREADS\_PER\_CPU の値を増やすことが必要になる場合があります。この場合、システムの拡張性を高めるためには、より多くのプロセスが必要です。実行されているプロセスが多すぎる場合は、数を減らしてください。

ほとんどのプラットフォームでは、PARALLEL\_THREADS\_PER\_CPU のデフォルトは 2 です。ただし、比較的低速の I/O サブシステムを持つマシンでは、デフォルトが 8 まで上げられることがあります。



## ステップ 3: 一般パラメータのチューニング

この項では、次のタイプのパラメータについて説明します。

- パラレル操作のリソース制限を設定するパラメータ
- リソース消費に影響するパラメータ
- I/O に関連したパラメータ

### パラレル操作のリソース制限を設定するパラメータ

リソース制限を設定するパラメータは、次のとおりです。

- `PARALLEL_MAX_SERVERS`
- `PARALLEL_MIN_SERVERS`
- `LARGE_POOL_SIZE/SHARED_POOL_SIZE`
- `SHARED_POOL_SIZE`
- `PARALLEL_MIN_PERCENT`
- `PARALLEL_SERVER_INSTANCES`

#### `PARALLEL_MAX_SERVERS`

推奨値は、 $2 \times DOP \times \text{number\_of\_concurrent\_users}$  (同時ユーザー数) です。

`PARALLEL_MAX_SERVERS` パラメータは、パラレル実行で利用できるプロセスの最大数に対するリソース制限を設定します。`PARALLEL_AUTOMATIC_TUNING` を `FALSE` に設定する場合は、`PARALLEL_MAX_SERVERS` の値を手動で指定する必要があります。

ほとんどのパラレル操作に必要な問合せサーバー・プロセス数は、最大でも、操作で任意の表に使用される最大 DOP の 2 倍です。

`PARALLEL_AUTOMATIC_TUNING` が `FALSE` に設定されている場合、`PARALLEL_MAX_SERVERS` デフォルト値は 5 です。この値は、いくつかの最小限の操作には十分ですが、パラレル実行には不十分です。パラメータ `PARALLEL_MAX_SERVERS` を手動で設定する場合は、CPU 数の 10 倍に設定してください。この値は最初に設定するのに適切な値です。

同時ユーザーをサポートするには、問合せサーバー・プロセスをさらに追加します。CPU バウンズのプロセスの数は、CPU 数の倍数になりますが、少なめに抑えたいものです (CPU 数の 4 ~ 16 倍くらい)。すると、同時パラレル実行文の数は、2 ~ 8 の範囲に抑えられます。

データベースのユーザーが非常に多くの同時操作を開始した場合、Oracle の問合せサーバー・プロセス数が十分でないことがあります。この場合に、`PARALLEL_MIN_PERCENT`

がデフォルト値の 0 (ゼロ) 以外の値に設定されていると、Oracle は操作の逐次処理をするか、エラーの表示を行います。

**ユーザー・プロセスの数が多すぎるとき** 同時ユーザーが使用する問合せサーバー・プロセスが多すぎると、メモリー競合 (ページング)、I/O 競合または過度のコンテキスト・スイッチが起こるおそれがあります。競合が起こると、システムのスループットは、パラレル実行を使わない場合よりも低いレベルまで下がることがあります。PARALLEL\_MAX\_SERVERS 値を増やすのは、その結果で発生する負荷に十分なメモリーおよび I/O 帯域幅がシステムに備わっている場合のみにしてください。問合せサーバー・プロセスの合計数を抑えると、パラレル操作を実行できる同時ユーザーの数は限られますが、システムのスループットは安定しやすくなります。

### 同時ユーザー数の増加

同時ユーザーの数を増やすために、リソース・コンシューマ・グループの同時セッション数を制限することもできます。次に例を示します。

- PARALLEL\_ADAPTIVE\_MULTI\_USER を使用可能にすることができます。
- バッチ・ジョブを実行するユーザーに対し、大きめの制限値を設定できます。
- 分析を実行するユーザーに対し、中程度の制限値を設定できます。
- 特定のクラスのユーザーに対し、並列性の使用を完全に禁止できます。

**関連項目：** リソース・コンシューマ・グループの詳細は、『Oracle8i 管理者ガイド』と『Oracle8i 概要』の「データベース・リソース・マネージャ」の説明を参照してください。

### ユーザーのリソース数の制限

ユーザーのリソース・コンシューマ・グループを設定することで、特定のユーザーが使用可能な並列性の量を制限できます。これによって、1 人のユーザーまたはユーザー・グループのセッション数、同時ログオンおよびパラレル・プロセス数を制限します。

パラレル実行文を処理する各問合せサーバー・プロセスに、セッション ID を使用してログオンします。各プロセスは、現行セッションのユーザーの制限に対してカウントされます。たとえば、ユーザーを 10 プロセスに制限するには、ユーザーの制限を 11 に設定します。1 つのプロセスはパラレル・コーディネータ用で、残りの 10 のパラレル・プロセスは 2 セットの問合せサーバーから構成されます。したがって、ユーザーの最大 DOP は 5 になります。

**関連項目：** 同時ユーザー、DOP および消費されるリソースのバランス化の詳細は、27-2 ページの「[メモリー、ユーザーおよびパラレル実行サーバー・プロセスの計算式](#)」を参照してください。また、ユーザー・プロファイルを使ったリソース管理の詳細は『Oracle8i 管理者ガイド』を参照し、GV\$ ビューの問合せの詳細は『Oracle8i Parallel Server 概要および管理』を参照してください。

## PARALLEL\_MIN\_SERVERS

推奨値は 0 (ゼロ) です。

システム・パラメータ PARALLEL\_MIN\_SERVERS を使うと、単一インスタンスの起動時に起動してパラレル操作用に確保しておくプロセス数を指定できます。構文は次のとおりです。

```
PARALLEL_MIN_SERVERS=n
```

ここで、*n* は、パラレル操作のために起動して確保しておくプロセス数です。

PARALLEL\_MIN\_SERVERS を設定すると、メモリー使用量と起動コストのバランスがとられます。PARALLEL\_MIN\_SERVERS を使用して開始されたプロセスは、データベースがシャットダウンされるまで終了しません。このため、問合せが発行されたときにプロセスが使用可能な可能性が高くなります。ただし、これらのプロセスが使用するメモリーは断片化されることがあり、高水位標が少しずつ増加する原因となるので、問合せサーバー・プロセスは定期的によりサイクルするのが望ましい処理です。PARALLEL\_MIN\_SERVERS を設定しないと、プロセスは 5 分間アイドル状態が続いた後に終了します。

## LARGE\_POOL\_SIZE/SHARED\_POOL\_SIZE

大規模プールのチューニング方法に関する次の説明は、26-16 ページの「[SHARED\\_POOL\\_SIZE](#)」という見出しの下に記載されている注記を除いて、共有プールの調整にも適用されます。決定した量によって、このメモリー設定の値も増やす必要があります。

LARGE\_POOL\_SIZE に対する推奨値はありません。オラクル社では、このパラメータを設定しないで PARALLEL\_AUTOMATIC\_TUNING パラメータを TRUE に設定し、Oracle によって設定される値を使用することをお勧めします。ただし、システムで割り当てられた値が処理要件に対して不十分な場合は例外です。

---

---

**注意:** PARALLEL\_AUTOMATIC\_TUNING が TRUE に設定されていると、Oracle は、パラレル実行バッファを大規模プールから割り当てます。このパラメータが FALSE の場合、Oracle はパラレル実行バッファを共有プールから割り当てます。

---

---

Oracle は、PARALLEL\_AUTOMATIC\_TUNING が TRUE に設定されている場合は LARGE\_POOL\_SIZE を自動的に計算します。LARGE\_POOL\_SIZE の値を手動で設定するには、V\$SGASTAT ビューを問い合せて、ニーズに応じて LARGE\_POOL\_SIZE の値を増減します。

たとえば、Oracle は起動時に次のようなエラーを表示することがあります。

```
ORA-27102: out of memory
SVR4 Error: 12: Not enough space
```

この場合は、データベースが起動できるように、LARGE\_POOL\_SIZE の値を十分に低くすることを検討してください。LARGE\_POOL\_SIZE の値を低くした後で、次のようなエラーが表示される場合があります。

```
ORA-04031: unable to allocate 16084 bytes of shared memory ("large
pool","unknown object","large pool hea","PX msg pool")
```

この場合は、次の問合せを実行して、Oracle が 16,084 バイトを割り当てられない理由を判断します。

```
SELECT NAME, SUM(BYTES) FROM V$SGASTAT WHERE POOL='LARGE POOL' GROUP BY
ROLLUP (NAME) ;
```

Oracle は、次のような出力で応答します。

NAME	SUM(BYTES)
-----	-----
PX msg pool	1474572
free memory	562132
	2036704

3 rows selected.

これを解決するには、LARGE\_POOL\_SIZE の値を増やします。この例では、LARGE\_POOL\_SIZE を約 2MB にしています。使用可能なメモリー量に応じて、LARGE\_POOL\_SIZE の値を 4MB に増やし、データベースの起動を試行します。Oracle が ORA-4031 メッセージを表示し続ける場合は、起動が成功するまで LARGE\_POOL\_SIZE の値を徐々に増やしてください。

## メッセージ・バッファの追加メモリー要件の計算

大規模プールまたは共有プールの初期設定を決定した後で、メッセージ・バッファの追加メモリー必要量を計算し、カーソルに必要な追加スペース量を判断する必要があります。

**メッセージ・バッファの追加メモリー** LARGE\_POOL\_SIZE または SHARED\_POOL\_SIZE パラメータの値を増やして、メッセージ・バッファに充当する必要があります。メッセージ・バッファにより、問合せサーバー・プロセスが相互に通信できます。自動パラレル・チューニングを使用可能にすると、Oracle はメッセージ・バッファの領域を大規模プールから割り当てます。自動パラレル・チューニングを使用可能にしない場合、Oracle は領域を共有プールから割り当てます。

Oracle は、生成側の問合せサーバーと受取側の問合せサーバー間の仮想接続あたり、固定のバッファ数を使用します。接続は、DOP の増加の 2 乗で増加します。そのため、パラレル実行で使用される最大メモリー量は、システムで許可されている最大 DOP によって制限されます。この値は、PARALLEL\_MAX\_SERVERS パラメータ、またはポリシーとプロファイルを使用して制御できます。

**関連項目：** Oracle がサーバー間の接続を行う方法は、『Oracle8i 概要』を参照してください。

メッセージ・バッファに必要な追加メモリー量は、次の 5 つのステップに従って計算します。これら 5 つのステップは、PARALLEL\_AUTOMATIC\_TUNING パラメータを TRUE に設定したときに Oracle が実行するステップとほぼ同じです。自動チューニングを使用可能にし、計算された値をチェックすると、同じ結果が得られます。

1. システムで使用可能な最大 DOP を判断します。この値を判断するときは、バッチ・ジョブをどのようにパラレル化するかを検討してください。大きな DOP を使用する単一ジョブでは、小さな DOP で複数のジョブを実行する場合よりも多くのメモリーを使用します。したがって、メッセージ・バッファに十分なメモリーがあることを確認するには、" 上限 "DOP を計算します。この DOP では、複数インスタンスも考慮する必要があります。つまり、2 つのインスタンスで並列度 4 を使用するには、計算する数値は 4 ではなく 8 にする必要があります。最大値を控えめに計算する方法として、PARALLEL\_MAX\_SERVERS の値をインスタンス数で乗算し、4 で割ります。この数値は、ステップ 5 の後に記載されている計算式の DOP です。
2. SQL 文に関するインスタンス数を判断します。ほとんどのインストレーションでは、この数は 1 になります。この値は、計算式の INSTANCES です。
3. この DOP で実行する同時問合せの最大数を推定します。PARALLEL\_ADAPTIVE\_MULTI\_USER が TRUE に設定されている場合、または PARALLEL\_MAX\_SERVERS を 4 で割った値以上の値を DOP に設定している場合は、1 という数字が妥当です。DOP はサーバー数によって制限されるからです。この数値は、後述の計算式の USERS です。
4. 問合せあたりの問合せサーバー・プロセス・グループの最大数を計算します。通常、Oracle は、問合せあたり 1 グループの問合せサーバー・プロセスのみを使用します。ただし、副問合せでは、副問合せごとに 1 グループの問合せサーバー・プロセスを使用することもあります。この数値の控えめな開始値は 2 です。この数値は、ステップ 5 の後に記載されている計算式の GROUPS です。
5. パラメータ PARALLEL\_MESSAGE\_SIZE の値を使用して、パラレル・メッセージ・サイズを判断します。通常、この値は 2KB か 4KB です。PARALLEL\_MESSAGE\_SIZE の現在値を参照するには、SQL\*Plus SHOW PARAMETERS コマンドを使用します。

**SMP システムのメモリー計算式** ほとんどの SMP システムでは、次の計算式を使用します。

$$\text{メモリーのバイト数} = (3 \times \text{SETS} \times \text{USERS} \times \text{SIZE} \times \text{CONNECTIONS})$$

ここで、CONNECTIONS = (DOP<sup>2</sup> + 2 x DOP) です。

**MPP システムのメモリー計算式** OPS を使用していて、INSTANCES の値が 1 より大きい場合は、次の計算式を使用します。この計算式は、ローカル仮想接続とリモート物理接続に必要なバッファ数を計算します。REMOTE の値をノード間のリモート接続数として使用し、オペレーティング・システムをチューニングできます。計算式は次のとおりです。

$$\text{メモリーのバイト数} = (\text{GROUPS} \times \text{USERS} \times \text{SIZE}) \times ((\text{LOCAL} \times 3) + (\text{REMOTE} \times 2))$$

各項目は次のとおりです。

- $\text{CONNECTIONS} = (\text{DOP}^2 + 2 \times \text{DOP})$
- $\text{LOCAL} = \text{CONNECTIONS} / \text{INSTANCES}$
- $\text{REMOTE} = \text{CONNECTIONS} - \text{LOCAL}$

各インスタンスは、計算式によって計算されたメモリーを使用します。

この量を、大規模プールまたは共有プールのオリジナルの設定に加算します。ただし、これらのメモリー構造体のいずれかの値を設定する前に、次の見出しの下で説明するカーソル用の追加メモリーも考慮する必要があります。

**カーソル用の追加メモリーの計算** パラレル実行計画は、シリアル実行計画よりも多くの SQL 領域を使用します。共有プール・リソースの使用率を定期的に監視し、システムの処理要件に十分に適合するメモリーが両方の構造体にあることを確認する必要があります。

**関連項目：** 実行計画の詳細は、[第 13 章「EXPLAIN PLAN の使用方法」](#)を参照してください。問合せサーバーの通信方法の詳細は、『Oracle8i 概要』を参照してください。

---

---

**注意：** 前のリリースでパラレル実行を使用し、今回これを手動でチューニングしようとしている場合は、このプールでの需要の減少を考慮して、LARGE\_POOL\_SIZE に割り当てられるメモリー量を減らしてください。

---

---

## 処理開始後のメモリーの調整

この項の計算式は、これ以降の検討の起点となるものです。自動チューニングと手動チューニングのどちらを行うかにかかわらず、システムの実行中に使用率を監視して、メモリーのサイズが大きすぎたり小さすぎたりしないかを確認します。これを確認するには、次の問合せ構文を使用して大規模プールの構造体のサイズを調べた後で、大規模プールと共有プールをチューニングします。

```
SELECT POOL, NAME, SUM(BYTES) FROM V$SGASTAT WHERE POOL LIKE '%pool%'
GROUP BY ROLLUP (POOL, NAME);
```

Oracle は、次のような出力で応答します。

POOL	NAME	SUM(BYTES)
-----	-----	-----
large pool	PX msg pool	38092812
large pool	free memory	299988
large pool		38392800
shared pool	Checkpoint queue	38496
shared pool	KGFF heap	1964
shared pool	KGK heap	4372
shared pool	KQLS heap	1134432
shared pool	LRMPD SGA Table	23856
shared pool	PLS non-lib hp	2096
shared pool	PX subheap	186828
shared pool	SYSTEM PARAMETERS	55756
shared pool	State objects	3907808
shared pool	character set memory	30260
shared pool	db_block_buffers	200000
shared pool	db_block_hash_buckets	33132
shared pool	db_files	122984
shared pool	db_handles	52416
shared pool	dictionary cache	198216
shared pool	dln shared memory	5387924
shared pool	enqueue_resources	29016
shared pool	event statistics per sess	264768
shared pool	fixed allocation callback	1376
shared pool	free memory	26329104
shared pool	gc_*	64000
shared pool	latch nowait fails or sle	34944
shared pool	library cache	2176808
shared pool	log_buffer	24576
shared pool	log_checkpoint_timeout	24700
shared pool	long op statistics array	30240
shared pool	message pool freequeue	116232
shared pool	miscellaneous	267624
shared pool	processes	76896
shared pool	session param values	41424
shared pool	sessions	170016
shared pool	sql area	9549116
shared pool	table columns	148104
shared pool	trace_buffers_per_process	1476320
shared pool	transactions	18480
shared pool	trigger inform	24684
shared pool		52248968
		90641768

41 rows selected.

出力で示されている使用メモリーを評価し、処理ニーズに基づいて LARGE\_POOL\_SIZE の設定を変更します。

より詳細なメモリー使用率統計を取得するには、次の問合せを実行します。

```
SELECT * FROM V$PX_PROCESS_SYSSTAT WHERE STATISTIC LIKE 'Buffers%';
```

Oracle は、次のような出力で応答します。

STATISTIC	VALUE
-----	-----
Buffers Allocated	23225
Buffers Freed	23225
Buffers Current	0
Buffers HWM	3620
4 Rows selected.	

使用中のメモリー量は、統計 "Buffers Current" と "Buffers HWM" に表示されます。バイト値は、バッファ数に PARALLEL\_EXECUTION\_MESSAGE\_SIZE の値を乗算することで計算します。高水位標をパラレル実行のメッセージ・プール・サイズと比較して、メモリーが割り当てられすぎているかどうかを判断します。たとえば、最初の出力では、大規模プールの値は 'px msg pool' で示されているように 38092812、つまり 38MB です。2 番目の出力の 'Buffers HWM' は 3,620 です。これは、パラレル実行のメッセージ・サイズである 4,096 で乗算すると、14,827,520、つまり約 15MB になります。この場合、高水位標は容量の約 40% に達します。

### SHARED\_POOL\_SIZE

前述のように、PARALLEL\_AUTOMATIC\_TUNING が FALSE の場合、Oracle は、問合せ サーバープロセスを共有プールから割り当てます。この場合は、次の例外を除き、前の見出しの大規模プールの説明に従って共有プールをチューニングします。

- 共有カーソルやストアド・プロシージャなどの共有プールの他の使用用途を考慮する。
- 大きな値を設定するとマルチユーザー・システムでのパフォーマンスが改善されるが、小さな値を設定するとメモリー使用量が少なくなる。

パラレル実行を使用すると、生成されるカーソル数が増えることも考慮する必要があります。V\$SQLAREA ビューの統計を参照して、Oracle がカーソルを再コンパイルする頻度を判断してください。カーソルのヒット率が悪い場合は、プールのサイズを増やします。

次の問合せを使用して、各コンポーネントが使用するメモリー量を判断し、SHARED\_POOL\_SIZE の値をチューニングします。

```
SELECT POOL, NAME, SUM(BYTES) FROM V$SGASTAT WHERE POOL LIKE '%pool%'
GROUP BY ROLLUP (POOL, NAME);
```



Oracle は、次のような出力で応答します。

POOL	NAME	SUM(BYTES)
shared pool	Checkpoint queue	38496
shared pool	KGFF heap	1320
shared pool	KGK heap	4372
shared pool	KQLS heap	904528
shared pool	LRMPD SGA Table	23856
shared pool	PLS non-lib hp	2096
shared pool	PX msg pool	373864
shared pool	PX subheap	65188
shared pool	SYSTEM PARAMETERS	55828
shared pool	State objects	3877520
shared pool	character set memory	30260
shared pool	db_block_buffers	200000
shared pool	db_block_hash_buckets	33132
shared pool	db_files	122984
shared pool	db_handles	36400
shared pool	dictionary cache	181792
shared pool	dln shared memory	5387924
shared pool	enqueue_resources	25560
shared pool	event statistics per sess	189120
shared pool	fixed allocation callback	1376
shared pool	free memory	36255072
shared pool	gc_*	64000
shared pool	latch nowait fails or sle	34944
shared pool	library cache	559676
shared pool	log_buffer	24576
shared pool	log_checkpoint_timeout	24700
shared pool	long op statistics array	21600
shared pool	message pool freequeue	116232
shared pool	miscellaneous	230016
shared pool	network connections	17280
shared pool	processes	53736
shared pool	session param values	58684
shared pool	sessions	121440
shared pool	sql area	1232748
shared pool	table columns	148104
shared pool	trace_buffers_per_process	1025232
shared pool	transactions	18480
shared pool	trigger inform	16456
shared pool		51578592
		51578592

40 rows selected.

前述したのと同じ方法で、パラレル実行によって使用されるバッファ数を監視し、'shared pool PX msg pool' を、ビュー VSPX\_PROCESS\_SYSSTAT からの出力で報告された現行の高水位標と比較します。

### PARALLEL\_MIN\_PERCENT

このパラメータの推奨値は 0 (ゼロ) です。

このパラメータを使用すると、ユーザーは、使用中のアプリケーションに応じて、許容される DOP を待機できます。このパラメータに 0 (ゼロ) 以外の値を設定すると、Oracle は、必要な最小 DOP がシステムで満たされないときにエラーを戻します。

たとえば、PARALLEL\_MIN\_PERCENT を 50 に設定すると、これは "50%" と解釈され、適応アルゴリズムまたはリソース制限により DOP が削減され、Oracle は ORA-12827 を返します。次に例を示します。

```
SELECT /*+ PARALLEL(e, 4, 1) */ d.deptno, SUM(SAL)
FROM emp e, dept d WHERE e.deptno = d.deptno
GROUP BY d.deptno ORDER BY d.deptno;
```

Oracle は、次のメッセージで応答します。

```
ORA-12827: INSUFFICIENT PARALLEL QUERY SLAVES AVAILABLE
```

### PARALLEL\_SERVER\_INSTANCES

このパラメータに設定する推奨値は、パラレル・サーバー環境のインスタンス数です。

PARALLEL\_SERVER\_INSTANCES パラメータは、パラレル・サーバー環境で構成されるインスタンス数を指定します。Oracle は、PARALLEL\_AUTOMATIC\_TUNING が TRUE に設定されているときに、このパラメータの値を使用して LARGE\_POOL\_SIZE の値を計算します。

## リソース消費に影響するパラメータ

リソース消費に影響するパラメータは、次のとおりです。

- [HASH\\_AREA\\_SIZE](#)
- [SORT\\_AREA\\_SIZE](#)
- [PARALLEL\\_EXECUTION\\_MESSAGE\\_SIZE](#)
- [OPTIMIZER\\_PERCENT\\_PARALLEL](#)
- [PARALLEL\\_BROADCAST\\_ENABLE](#)

この項で説明するパラメータの最初のグループは、すべてのパラレル操作、特にパラレル実行でのメモリーとリソースの消費に影響します。この項で説明するパラメータの 2 番目のサ

ブセットは、パラレル DML と DDL に影響します。第 27 章「パラレル実行のパフォーマンス上の問題について」では、これらのパラメータの相互関係について詳しく説明します。

リソース消費を制御するには、次の 2 つのレベルでメモリーを構成します。

- Oracle のレベル。システムが適切な容量のオペレーティング・システム・メモリーを使用するようにします。
- オペレーティング・システムのレベル（一貫性を保つため）。プラットフォームによっては、すべてのプロセスについて合計された、使用可能な仮想メモリーの総容量を制御するオペレーティング・システムのパラメータを設定しなければならないことがあります。

通常、SGA は実際の物理メモリーに含まれます。SGA は静的で、サイズも固定です。サイズを変更したい場合は、データベースをシャットダウンし、変更を行ってからデータベースを再起動します。Oracle は、SGA の内部に大規模ブールと共有ブールを割り当てます。

データ・ウェアハウス操作で使用するメモリーの大部分は、より動的で、このメモリーはプロセス・メモリーに含まれますが、プロセス・メモリーのサイズとプロセス数は、どちらも非常に多様に設定できます。このメモリーは、HASH\_AREA\_SIZE および SORT\_AREA\_SIZE パラメータによって制御されます。これらのパラメータが連携して、Oracle で使用される仮想メモリー量を制御します。

また、プロセス・メモリーは仮想メモリーに含まれます。仮想メモリーの合計は使用可能な実メモリーよりもある程度は多くなければなりません。実メモリーとは、物理メモリーから SGA のサイズを除いたものです。一般的には、仮想メモリーは物理メモリーから SGA を除いたサイズの 2 倍を超えないようにしてください。仮想メモリーを実メモリーの複数倍のサイズに設定すると、マシンが過負荷になったときにページング率が増加することがあります。

メモリー・サイズ設定の一般的なルールとして、各プロセスにはハッシュ結合を行うのに十分なアドレス空間が必要です。大容量のデータ・ウェアハウス操作では、メモリー、プロセス数およびハッシュ結合操作の回数の関係が重要な要因になります。ハッシュ結合や大規模なソートは多くのメモリーが必要な操作なので、少数のプロセスを構成し、それぞれの使用可能なメモリー容量の限界を多めにしてください。ただし、ソートのパフォーマンスはメモリー使用率が増えるにつれて低下します。

## HASH\_AREA\_SIZE

HASH\_AREA\_SIZE は、2 つのアプローチのいずれかを使用して設定します。最初のアプローチでは、SGA を構成し、システムが通常の負荷で使用するメモリー・プロセス量を計算した後で、使用可能なメモリー量を調べます。

Oracle プロセスが使用を許可される合計メモリー量は、通常の負荷でのプロセス数で割る必要があります。これらのプロセスには、パラレル実行サーバーが含まれます。この数値は、プロセスあたりの作業メモリーの合計量を決定します。この量を、特定の問合せの異なる操作間で共有する必要があります。たとえば、HASH\_AREA\_SIZE または SORT\_AREA\_SIZE はこの数値の半分または 3 分の 1 に設定するのが妥当です。

これらのパラメータは、スワッピングが発生しない最大の数値に設定します。説明に従ってこれらのパラメータを設定した後で、スワッピングと空きメモリを監視する必要があります。スワッピングが発生している場合は、これらのパラメータの値を小さくします。空きメモリが大量に残っている場合は、これらのパラメータの値を増やすことができます。

HASH\_AREA\_SIZE を設定する 2 番目のアプローチでは、実行するハッシュ結合のタイプを十分に理解し、問い合わせるデータ量も認識しておく必要があります。実行する問合せと問合せ計画をよく理解している場合は、このアプローチが妥当です。

HASH\_AREA\_SIZE は  $S$  の平方根のおよそ半分にします。ここで、 $S$  は結合操作に対する入力の小さい方のサイズ（メガバイト）です。いずれの場合も、HASH\_AREA\_SIZE の値は 1MB 以上にする必要があります。

この関係は次のように表すことができます。

$$\text{HASH\_AREA\_SIZE} \geq \frac{\sqrt{S}}{2}$$

たとえば、 $S$  が 16MB の場合、HASH\_AREA\_SIZE の適切な最小値は、すべてのパラレル・プロセスで合計して 2MB となります。したがって、2 つのパラレル・プロセスがある場合は、HASH\_AREA\_SIZE の最小値は 1MB になります。これよりも小さいハッシュ領域はお薦めしません。

大規模データ・ウェアハウスの場合は、HASH\_AREA\_SIZE の範囲は 8MB ~ 32MB、あるいはさらに大きくなる場合があります。このパラメータによって、ハッシュ結合に適したメモリが提供されます。パラレル・ハッシュ結合を実行する各プロセスが、HASH\_AREA\_SIZE と等しい容量のメモリを使用します。

ソートのパフォーマンスが SORT\_AREA\_SIZE に影響されるのにくらべて、ハッシュ結合のパフォーマンスは HASH\_AREA\_SIZE によってよりいっそう影響されます。SORT\_AREA\_SIZE の場合と同様に、ハッシュ領域サイズを大きくしすぎると、システムのメモリ不足を引き起こす場合があります。

ハッシュ領域は、ブロックをバッファ・キャッシュにキャッシュしません。小さい値を HASH\_AREA\_SIZE に設定してもキャッシュは発生しません。ただし、設定値を小さくしすぎるとパフォーマンスが悪影響を受けます。

HASH\_AREA\_SIZE は、パラレル実行操作、および DML 文や DDL 文の問合せ部分に関係します。

### **SORT\_AREA\_SIZE**

このパラメータの推奨値は、256KB ~ 4MB の範囲です。

このパラメータによって、ソート操作のために問合せサーバー・プロセスごとに割り当てるメモリ容量が指定されます。システムのメモリに余裕がある場合は、SORT\_AREA\_SIZE を大きな値に設定することをお薦めします。これによって、ソート・プ

プロセス全体がメモリー内で実行される可能性が高くなるので、操作のパフォーマンスが大幅に改善されます。ただし、メモリーがシステムの動作に大きく影響する場合には、ソートおよびハッシュ操作に割り当てるメモリー容量を制限してください。

ソート領域が小さすぎる場合は、多数のソートの実行をマージするために過度の量の I/O が必要になります。ソート領域のサイズがソートするデータの容量よりも小さい場合は、ソートがディスクに移動されて、ソートの実行が作成されます。これらは後でソート領域を使用して再びマージされる必要があります。ソート領域のサイズが非常に小さい場合は、マージする必要のあるソート実行が多数生成され、複数のパスが必要になることもあります。SORT\_AREA\_SIZE が小さくなるにつれて、I/O の量が増えます。

ソート領域が大きすぎる場合は、オペレーティング・システムのページング率が過剰になります。各問合せサーバー・プロセスがこのメモリー容量を各ソートに割り当てるので、ソート領域は累積されすぐに増えていきます。このような状況では、オペレーティング・システムのページング率を監視して、要求されているメモリーが多すぎないかどうかを調べてください。

SORT\_AREA\_SIZE は、パラレル実行操作、および DML 文や DDL 文の問合せ部分に関係します。すべての CREATE INDEX 文では、索引を生成するためにソートを実行する必要があります。ソートを必要とするコマンドには、次のものが含まれます。

- CREATE INDEX (索引の作成)
- ダイレクト・ロード挿入 (INSERT) (索引が関係している場合)
- ALTER INDEX ... REBUILD

**関連項目：** 26-19 ページの「[HASH\\_AREA\\_SIZE](#)」

## PARALLEL\_EXECUTION\_MESSAGE\_SIZE

PARALLEL\_EXECUTION\_MESSAGE\_SIZE の推奨値は 4KB です。

PARALLEL\_AUTOMATIC\_TUNING が TRUE の場合、デフォルトは 4KB です。

PARALLEL\_AUTOMATIC\_TUNING が FALSE の場合、デフォルトは 2KB より少し多くなります。

PARALLEL\_EXECUTION\_MESSAGE\_SIZE パラメータは、パラレル実行メッセージのサイズの上限を指定します。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が、この値はほとんどのアプリケーションに対して十分です。

PARALLEL\_EXECUTION\_MESSAGE\_SIZE により大きな値を設定する場合は、パラレル自動チューニングを使用可能にしているかどうかに応じて、LARGE\_POOL\_SIZE または SHARED\_POOL\_SIZE にも大きな値を設定する必要があります。

PARALLEL\_EXECUTION\_MESSAGE\_SIZE の値を増やすと応答時間が大幅に改善することがありますが、メモリー使用量も大幅に増えます。たとえば、PARALLEL\_EXECUTION\_MESSAGE\_SIZE の値を 2 倍にすると、パラレル実行に必要なメッセージ・ソース・プールも 2 倍になります。

したがって、`PARALLEL_AUTOMATIC_TUNING` に `FALSE` を設定した場合は、`SHARED_POOL_SIZE` を調整して、パラレル実行メッセージに適合させる必要があります。`PARALLEL_AUTOMATIC_TUNING` に `TRUE` を設定し、`LARGE_POOL_SIZE` を手動で設定した場合は、`LARGE_POOL_SIZE` を調整してパラレル実行メッセージに適合させる必要があります。

### OPTIMIZER\_PERCENT\_PARALLEL

推奨値は、`100/number_of_concurrent_users` です。

このパラメータによって、オブティマイザが与えられた実行計画に対してパラレル化を試みるしきい値が決まります。`OPTIMIZER_PERCENT_PARALLEL` は、使用されているリソースの合計が最小化されていない場合でも、パラレル実行が原因で応答時間が遅くなっている計画をオブティマイザが使用することを促進します。

`OPTIMIZER_PERCENT_PARALLEL` のデフォルト値は 0 (ゼロ) です。デフォルト値が設定されている場合は、できるだけ少ないリソースを使用して計画がパラレル化されます。このとき、ごく少量のリソースしか使われないために、操作の実行時間が長くなる場合があります。

---

**注意：** 適切な索引があれば、Oracle は 1 レコードを表から高速に選択できます。これを行うために並列性は必要ありません。その 1 行を検索するための全走査をパラレルで実行できます。ただし、通常は、各パラレル・プロセスでは多数の行が調べられます。この場合、索引を使用するシリアル計画で行った場合に比べて、パラレル計画の応答時間は長くなり、システム・リソース使用量の合計は非常に大きくなります。パラレル計画を使用すると、より多くのリソースが使用されるので遅延は短縮されます。パラレル計画では、最大  $n$  倍のリソースが使用されます。ここで  $n$  は、並列度に設定された値と同じです。0 ~ 100 の値によって、スループットと応答時間のバランスをとるトレードオフが設定されます。索引がある場合は小さい値を設定し、表走査を行う場合は大きい値を設定します。

---

`OPTIMIZER_PERCENT_PARALLEL` の設定がゼロ以外の場合は、`FIRST_ROWS` ヒントを使用するか `OPTIMIZER_MODE` を `FIRST_ROWS` に設定するとこの値は上書きされます。

### PARALLEL\_BROADCAST\_ENABLE

デフォルト値は `FALSE` です。

非常に大規模な結合結果セットとごく小さな結合結果セット（サイズは行数ではなくバイト数で測定する）を結合する場合は、このパラメータを `TRUE` に設定します。この場合、オブティマイザには、大きいセットの行を処理する各問合せサーバー・プロセスに小さいセットの行をブロードキャストするオプションがあります。以上のことにより、パフォーマンスが改善されます。

PARALLEL\_BROADCAST\_ENABLE パラメータは、動的に設定できず、ハッシュ結合とマージ結合にのみ影響します。

## パラレル DML およびパラレル DDL のリソース消費に影響するパラメータ

パラレル DML とパラレル DDL のリソース消費に影響するパラメータは、次のとおりです。

- [TRANSACTIONS](#)
- [ROLLBACK\\_SEGMENTS](#)
- [FAST\\_START\\_PARALLEL\\_ROLLBACK](#)
- [LOG\\_BUFFER](#)
- [DML\\_LOCKS](#)
- [ENQUEUE\\_RESOURCES](#)

パラレル挿入、更新および削除には、シリアル DML 操作の場合よりも多くのリソースが必要です。同様に、PARALLEL CREATE TABLE ... AS SELECT および PARALLEL CREATE INDEX にも、より多くのリソースが必要となる場合があります。このため、さらにいくつかの初期化パラメータの値を増やす必要が生じることもあります。これらのパラメータは、問合せのためのリソースには影響しません。

**関連項目：** パラメータの詳細は、『Oracle8i SQL リファレンス』を参照してください。

## TRANSACTIONS

パラレル DML および DDL の場合は、各問合せサーバー・プロセスがトランザクションを開始します。パラレル・コーディネータは、2 フェーズ・コミット・プロトコルを使用してトランザクションをコミットするので、処理されるトランザクションの数は DOP に応じて増加します。したがって、TRANSACTIONS 初期化パラメータの値を増やすことが必要な場合があります。

TRANSACTIONS パラメータは、同時トランザクションの最大数を指定します。デフォルトは並列性なしとみなされます。たとえば、DOP が 20 の場合は、20 の新しいサーバー・トランザクション（2 つのサーバー・セットがある場合は 40）と 1 つのコーディネータ・トランザクションがあることになるので、それらを同じインスタンスで実行するなら、TRANSACTIONS に 21（または 41）を加える必要があります。このパラメータを設定しない場合は、Oracle が 1.1 x SESSIONS に設定します。

## ROLLBACK\_SEGMENTS

パラレル DML および DDL のトランザクション数が増えると、より多くのロールバック・セグメントが必要になります。たとえば、DOP が 5 のコマンドでは 5 つのサーバー・トランザクションが使用されますが、それらのトランザクションは別々のロールバック・セグメントに分散されます。ロールバック・セグメントは空き領域のある表領域に属する必要がある

ります。ロールバック・セグメントを無制限にするか、STORAGE 句の MAXEXTENTS パラメータに高い値を設定する必要があります。それによって拡張が可能になり、領域不足を防ぐことができます。

### FAST\_START\_PARALLEL\_ROLLBACK

コミットされていないパラレル DML または DDL トランザクションがあるときにシステムがクラッシュした場合は、FAST\_START\_PARALLEL\_ROLLBACK パラメータを使用することで起動時のトランザクション回復を高速化できます。

このパラメータは、"デッド・トランザクション"を回復するときに使用される DOP を制御します。デッド・トランザクションとは、システムがクラッシュする前にアクティブだったトランザクションです。デフォルトでは、DOP には CPU\_COUNT パラメータの値の 2 倍以内で選択されます。

デフォルトの DOP で不十分な場合は、パラメータを HIGH に設定します。これによって、最大の DOP が CPU\_COUNT パラメータの値の 4 倍以内に設定されます。この機能は、デフォルトで使用可能になっています。

### LOG\_BUFFER

VSSYSSTAT ビューの統計 "redo buffer allocation retries" を確認します。値が高い場合は、LOG\_BUFFER サイズを増やしてみてください。多数のログを生成するシステムの場合、LOG\_BUFFER サイズは通常 3 ~ 5MB です。LOG\_BUFFER サイズを増やしてもまだ再試行回数が多いようなら、ログ・ファイルが存在するディスクに問題のある可能性があります。その場合は、I/O サブシステムをチューニングして、REDO の I/O 率を増やしてください。これを行う方法の 1 つとして、複数のディスク間でファイン・グレイン・ストライプ化を行う方法があります。たとえば、16KB のストライプ・サイズを使用します。より単純なアプローチは、REDO ログを独自のディスクに分離することです。

### DML\_LOCKS

このパラメータは DML ロックの最大数を指定します。この値は、すべてのユーザーによって参照されるすべての表のロック総数と等しい必要があります。パラレル DML 操作のロックおよびエンキューのリソース要件は、シリアル DML とはかなり異なります。パラレル DML は、より多くのロックを保持するので、ENQUEUE\_RESOURCES および DML\_LOCKS パラメータの値を同じ量だけ増やす必要があります。

表 26-2 は、さまざまなタイプのパラレル DML 文について、コーディネータおよび問合せサーバー・プロセスによって取得されるロックのタイプを示しています。この情報を使用すると、これらのパラメータで必要となる値を判断できます。1 つの問合せサーバー・プロセスは 1 つ以上のパーティションまたはサブパーティション上で動作できますが、1 つのパーティションまたはサブパーティションでは 1 つのサーバー・プロセスのみが動作できます（これはパラレル実行と異なります）。



表 26-2 パラレル DML 文によって取得されるロック

文のタイプ	コーディネータ・プロセスが取得するもの	各パラレル実行サーバーが取得するもの
パーティション表に対するパラレル UPDATE または DELETE。パーティション / サブパーティションのサブセットにブルニングされた WHERE 句	1 つの表ロック SX  ブルニングされた (サブ) パーティションあたり 1 つのパーティション・ロック X	1 つの表ロック SX  問合せサーバー・プロセスによって所有される、ブルニングされた (サブ) パーティションあたり 1 つのパーティション・ロック NULL  問合せサーバー・プロセスによって所有される、ブルニングされた (サブ) パーティションあたり 1 つのパーティション待機ロック S
パーティション表に対するパラレル行移行 UPDATE。(サブ)パーティションのサブセットにブルニングされた WHERE 句	1 つの表ロック SX  ブルニングされた (サブ) パーティションあたり 1 つのパーティション・ロック X  その他すべての (サブ) パーティションに対する 1 つのパーティション・ロック SX	1 つの表ロック SX  問合せサーバー・プロセスによって所有される、ブルニングされた (サブ) パーティションあたり 1 つのパーティション・ロック NULL  問合せサーバー・プロセスによって所有される、ブルニングされたパーティションあたり 1 つのパーティション待機ロック S  その他すべての (サブ) パーティションに対する 1 つのパーティション・ロック SX
パーティション表に対するパラレル UPDATE、DELETE または INSERT	1 つの表ロック SX  すべての (サブ) パーティションに対するパーティション・ロック X	1 つの表ロック SX  問合せサーバー・プロセスによって所有される (サブ) パーティションあたり 1 つのパーティション・ロック NULL  問合せサーバー・プロセスによって所有される (サブ) パーティションあたり 1 つのパーティション待機ロック S
非パーティション表に対するパラレル INSERT	1 つの表ロック X	なし

---

---

**注意：** 表ロック、パーティション・ロックおよびパーティション待機 DML ロックのすべてが、V\$LOCK ビューでは TM ロックとして表されます。

---

---

100 の DOP で実行されている 600 のパーティションがある表について考えます。すべてのパーティションが、行移行のないパラレル UPDATE/DELETE 文に関与しているとします。

コーディネータが獲得するもの：	1 つの表ロック SX
	600 のパーティション・ロック X
全サーバー・プロセスが獲得するもの：	100 の表ロック SX
	600 のパーティション・ロック NULL
	600 のパーティション待機ロック S

### ENQUEUE\_RESOURCES

このパラメータは、ロック・マネージャによってロックできるリソース数を設定します。パラレル DML 操作には、シリアル DML の場合よりも多くのリソースが必要です。したがって、ENQUEUE\_RESOURCES および DML\_LOCKS パラメータの値は、同じ量だけ増やしてください。

**関連項目：** 26-24 ページの「[DML\\_LOCKS](#)」

## I/O に関連したパラメータ

I/O に影響するパラメータは、次のとおりです。

- [DB\\_BLOCK\\_BUFFERS](#)
- [DB\\_BLOCK\\_SIZE](#)
- [DB\\_FILE\\_MULTIBLOCK\\_READ\\_COUNT](#)
- [HASH\\_MULTIBLOCK\\_IO\\_COUNT](#)
- [SORT\\_MULTIBLOCK\\_READ\\_COUNT](#)
- [DISK\\_ASYNC\\_IO](#) および [TAPE\\_ASYNC\\_IO](#)

これらのパラメータは、パラレル実行 I/O 操作のパフォーマンスを最適化するオプティマイザにも影響します。

## DB\_BLOCK\_BUFFERS

パラレル更新やパラレル削除を実行したときのバッファ・キャッシュの動作は、大容量の更新を実行するときのシステムの動作とよく似ています。詳細は、19-24 ページの「[バッファ・キャッシュのチューニング](#)」を参照してください。

## DB\_BLOCK\_SIZE

推奨値は 8KB または 16KB です。

データベースを作成するときは、データベース・ブロック・サイズを設定します。新しいデータベースを作成する場合は、大きなブロック・サイズを使ってください。

## DB\_FILE\_MULTIBLOCK\_READ\_COUNT

推奨値は、8KB ブロック・サイズでは 8、16KB ブロック・サイズでは 4 です。

このパラメータによって、オペレーティング・システムの単一の READ コールで読み込まれるデータベース・ブロック数が決まります。このパラメータの上限は、プラットフォームに依存します。DB\_FILE\_MULTIBLOCK\_READ\_COUNT に非常に高い値を設定した場合、オペレーティング・システムは、データベースを起動するときに許容される最高のレベルに値を下げます。この場合は、各プラットフォームが可能な最大の値を使用します。最大値は、一般に 64KB ~ 1MB の範囲です。

## HASH\_MULTIBLOCK\_IO\_COUNT

推奨値は 4 です。

このパラメータは、ハッシュ結合が一度に読み込みおよび書き込みを行うブロック数を指定します。HASH\_MULTIBLOCK\_IO\_COUNT の値を増やすと、ハッシュ・バケット数が減少します。システムが I/O バウンドの場合、1 回の I/O の転送量を多くすることによって I/O の効率を向上できます。

I/O バッファのメモリーは HASH\_AREA\_SIZE で指定されるので、I/O バッファが大きいということはハッシュ・バケットが少ないということを意味します。ただし、トレードオフがあります。大きな表（サイズが数百ギガバイト）では、ハッシュ・バケット数は多くして、I/O 効率は少し低くするのが適切です。ハッシュ結合の間に一時領域で I/O バウンドの条件が見つかったら、HASH\_MULTIBLOCK\_IO\_COUNT の値を増やすことを検討してください。

## SORT\_MULTIBLOCK\_READ\_COUNT

推奨値は、デフォルト値の使用です。

SORT\_MULTIBLOCK\_READ\_COUNT パラメータは、ソートが一時セグメントの読み込みを実行するたびに読み取られるデータベース・ブロック数を指定します。一時セグメントは、データがメモリーの SORT\_AREA\_SIZE に収まらないときにソートによって使用されます。

ソート操作中にシステムが実行する秒あたりの I/O 数が多すぎ、そのときに CPU が比較的アイドルになっている場合は、SORT\_MUTLIBLOCK\_READ\_COUNT パラメータの値を増やして、1 回の I/O で処理するデータ量を増やし I/O 数を減らすようにソート操作を変更してください。

関連項目： 詳細は、20-33 ページの「ソートのチューニング」を参照してください。

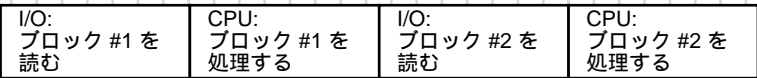
DISK\_ASYNC\_IO および TAPE\_ASYNC\_IO

推奨値は TRUE です。

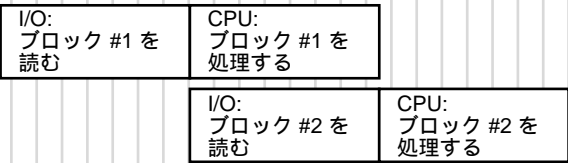
これらのパラメータは、オペレーティング・システムの非同期 I/O 機能を使用可能にしたり使用禁止にしたりします。これにより、表走査を実行するときに問合せサーバー・プロセスが I/O 要求を処理とオーバーラップできます。オペレーティング・システムが非同期 I/O をサポートしている場合、これらのパラメータはデフォルト値 TRUE のままにしておいてください。

図 26-1 非同期読み込み

同期読み込み



非同期読み込み



非同期操作は、現在はパラレル表走査、ハッシュ結合、ソートおよびシリアル表走査でサポートされています。ただし、この機能はオペレーティング・システム固有の構成を必要とする場合があります、すべてのプラットフォームでサポートされるとは限りません。オペレーティング・システム固有の Oracle マニュアルで確認してください。

## パラレル実行でのパラメータ設定シナリオの例

これから述べる例では、可能なパラレル実行インプリメンテーションのいくつかについて説明します。各例は、自動または手動パラレル実行チューニングの説明ではじまります。Oracle は、各サンプル・システムの特性と、パラレル実行チューニングがどのように初期化されたかに基づいて、その他のパラメータを自動的に設定します。さらに、並列度の設定とマルチユーザー問合せ調整機能の使用可能化について説明します。

これらの例のパラメータ設定が、内部的に導出された設定と全体的なパフォーマンスに与える効果は概算です。システムのパフォーマンス特性は、オペレーティング・システムの依存性とユーザーの作業負荷によって異なります。

追加的な調整によって、これらの例を微調整し、自分の環境により近いものにすることができま。PARALLEL\_AUTOMATIC\_TUNING を TRUE に設定した結果をさらに分析するには、26-5 ページの表 26-1 を参照してください。

本番環境では、表に対して DOP を設定し、マルチユーザー問合せ調整機能を使用可能にした後で、26-73 ページの「フェーズ 4: パラレル実行のパフォーマンスの監視」の説明に従ってシステムのパフォーマンスを分析します。システムのパフォーマンスが改善されない場合は、26-9 ページの「ステップ 3: 一般パラメータのチューニング」のフェーズ 1 の説明を参照してください。

次の 4 つの例では、異なるシステム・タイプをサイズと複雑度の低いものから順に説明します。

### 例 1: 小規模なデータ・マート

この例では、DBA はパラレル実行について限られた経験しかなく、システムを綿密に監視する時間ありません。

データベースは主としてスター型スキーマで、第 3 正規形にいくつかのサマリー表と少数の表があります。作業負荷は、主として " 非定型 " な性質があります。ユーザーは、パラレル実行によって大容量問合せのパフォーマンスが改善されることを期待しています。

システムに関するその他の実状は次のとおりです。

- CPU 数 = 4
- メイン・メモリー = 750MB
- ディスク = 40GB
- ユーザー数 = 16

DBA は次の設定を行います。

- PARALLEL\_AUTOMATIC\_TUNING = TRUE
- SHARED\_POOL\_SIZE = 12MB
- TRANSACTIONS = 設定しないでシステム・デフォルトを使用する

Oracle は、次のデフォルトを自動的に設定します。

- `PARALLEL_MAX_SERVERS = 64`
- `PARALLEL_ADAPTIVE_MULTI_USER = TRUE`
- `PARALLEL_THREADS_PER_CPU = 2`
- `PROCESSES = 76`
- `SESSIONS = 88`
- `TRANSACTIONS = 96`
- `LARGE_POOL_SIZE = 29MB`

## DOP とマルチユーザー問合せ調整機能のパラメータ設定

DBA は、次のようなコマンドを使用して、10,000 を超える行を持つすべての表をパラレル化します。

```
ALTER TABLE employee PARALLEL;
```

この例では、`PARALLEL_THREADS_PER_CPU` が 2 で、CPU 数が 4 なので、DOP は 8 です。`PARALLEL_ADAPTIVE_MULTI_USER` は `TRUE` に設定されているので、Oracle は、問合せの開始時に存在するシステム負荷に応じて、この DOP を減らすことがあります。

## 例 2: 中規模サイズのデータ・ウェアハウス

この例では、DBA は経験者ですが、やはり他の職務で多忙です。DBA は、ユーザーをリソース・コンシューマ・グループに編成し、ビューやその他のロールを使用して並列性へのアクセスを制御する方法を理解しています。DBA は、自動パラレル・チューニングで生成された設定を手動で調整して実験し、DBA が `FALSE` に設定した `PARALLEL_ADAPTIVE_MULTI_USER` パラメータを除くすべてのパラメータについて、生成された設定を使用することを選択しました。

システムの作業負荷には、いくつかの非定型問合せと、中央リポジトリをサマリー表とスター・スキーマに変換する大容量バッチ操作が含まれます。このシステム上のほとんどの問合せは、Oracle Express とその他のツールによって生成されています。

データベースには、第 3 正規形のソース表と、スター・スキーマおよびサマリー・フォーム内のエンドユーザー表しかありません。

システムに関するその他の実状は次のとおりです。

- CPU 数 = 8
- メイン・メモリー = 2GB
- ディスク = 80GB
- ユーザー数 = 40

DBA は次の設定を行います。

- PARALLEL\_AUTOMATIC\_TUNING = TRUE
- PARALLEL\_ADAPTIVE\_MULTI\_USER = FALSE
- PARALLEL\_THREADS\_PER\_CPU = 4
- SHARED\_POOL\_SIZE = 20MB

DBA は、並列性に関係しないその他のパラメータも設定します。結果として、Oracle は次のパラメータ設定を自動的に調整します。

- PROCESSES = 307
- SESSIONS = 342
- TRANSACTIONS = 376
- PARALLEL\_MAX\_SERVERS = 256
- LARGE\_POOL\_SIZE = 78MB

### DOP とマルチユーザー問合せ調整機能のパラメータ設定

DBA は、データ・ウェアハウス内のいくつかの表をパラレル化する一方で、特別なユーザーに対して他のビューを作成します。

```
ALTER TABLE sales PARALLEL;
CREATE VIEW invoice_parallel AS SELECT /*+ PARALLEL(P) */ * FROM invoices P;
```

DBA は、8 つの CPU で PARALLEL\_THREADS\_PER\_CPU 設定を 4 にすることをシステムに対して許可します。表の DOP は 32 です。したがって、単純な問合せは 32 のプロセスを使用しますが、より複雑な問合せは 64 のプロセスを使用します。

## 例 3: 大規模なデータ・ウェアハウス

この例では、DBA は経験者で、主にこのシステムの管理に従事しています。DBA は、リソースをよく制御し、システムのチューニング方法を理解しています。DBA は、大規模な問合せをバッチ・モードでスケジュールします。

作業負荷には、いくつかの非定型パラレル問合せが含まれます。また、大量のシリアル問合せがスター・スキーマに対して処理されます。サマリー表と索引を生成するバッチ処理もいくつかあります。データベースは完全に非正規化され、Oracle Parallel Server が使用されています。

システムに関するその他の実状は次のとおりです。

- 24 ノード、ノードあたり 1 つの CPU
- MPP アーキテクチャ (Massively Parallel Processing) を使用

- メイン・メモリー = ノードあたり 750MB
- ディスク = 200GB
- ユーザー数 = 256

DBA は、次のパラメータを設定することによって、手動パラレル・チューニングを行います。

- PARALLEL\_AUTOMATIC\_TUNING = FALSE
- PARALLEL\_THREADS\_PER\_CPU = 1
- PARALLEL\_MAX\_SERVERS = 10
- SHARED\_POOL\_SIZE = 75MB
- PARALLEL\_SERVER\_INSTANCES = 24
- PARALLEL\_SERVER = TRUE
- PROCESSES = 40
- SESSIONS = 50
- TRANSACTIONS = 60

DBA は、パラレル実行に関係しないその他のパラメータも設定します。PARALLEL\_AUTOMATIC\_TUNING が FALSE に設定されているので、Oracle は、パラレル実行バッファを SHARED\_POOL から割り当てます。

### DOP とマルチユーザー問合せ調整機能のパラメータ設定

DBA は、次のような構文を使用して DOP を明示的に設定することで、データ・ウェアハウスの表をパラレル化します。

```
ALTER TABLE department1 PARALLEL 10;  
ALTER TABLE department2 PARALLEL 5;  
CREATE VIEW current_sales AS SELECT /*+ PARALLEL(P, 20) */ * FROM sales P;
```

この例では、DBA がすべてのパラレル実行パラメータを手動で設定するので、Oracle はパラレル実行の計算を行いません。

## 例 4: 非常に大規模なデータ・ウェアハウス

この例では、DBA は非常に熟練しており、このシステムの管理に専念しています。DBA は環境をよく制御していますが、さまざまなユーザーは DBA がシステムに絶えず注意を払うことを要求しています。

DBA は PARALLEL\_AUTOMATIC\_TUNING を TRUE に設定しているので、Oracle はパラレル実行バッファを大規模プールから割り当てます。PARALLEL\_ADAPTIVE\_MULTI\_USER は自動的に使用可能にされます。システムに熟練し



た後、DBA はシステム提供のデフォルトを微調整して、パフォーマンスをさらに改善します。

データベースは非常に大規模なデータ・ウェアハウスで、データ・マートは同じマシン上にあります。データ・マートはウェアハウス内のデータから生成され、リフレッシュされます。データ・ウェアハウスは主として正規化されていますが、データ・マートは主としてスター・スキーマとサマリー表です。DBA は、実験によってシステム・パラメータを慎重にカスタマイズしました。

システムに関するその他の実状は次のとおりです。

- CPU 数 = 64
- メイン・メモリー = 32GB
- ディスク = 3TB
- ユーザー数 = 1,000

DBA は次の設定を行います。

- PARALLEL\_AUTOMATIC\_TUNING = TRUE
- PARALLEL\_MAX\_SERVERS = 600
- PARALLEL\_MIN\_SERVER = 600
- LARGE\_POOL\_SIZE = 1,300MB
- SHARED\_POOL\_SIZE = 500MB
- PROCESSES = 800
- SESSIONS = 900
- TRANSACTIONS = 1,024

## DOP とマルチユーザー問合せ調整機能のパラメータ設定

DBA は、どのユーザーと表が並列性を必要としているかを慎重に評価し、それらの要件に従って値を設定しました。DBA は、これまでの例で言及したすべてのステップを実行しましたが、さらに、ユーザーのピーク時間帯に次のコマンドを使用して DOP 適応アルゴリズムも使用可能にします。

```
ALTER SYSTEM SET PARALLEL_ADAPTIVE_MULTI_USER = TRUE;
```

バッチ処理が開始する業務時間外には、DBA はコマンドを発行することで適応処理を使用禁止にします。

```
ALTER SYSTEM SET PARALLEL_ADAPTIVE_MULTI_USER = FALSE;
```

## フェーズ 2: パラレル実行の物理データベース・レイアウトのチューニング

この項では、パラレル実行の最適なパフォーマンスのために、物理データベース・レイアウトを調整する方法について説明します。

- 並列性のタイプ
- データのストライプ化
- データのパーティション化
- 並列度の決定
- パラレル・ロードによるデータベースへの移入
- パラレル・ソートおよびハッシュ結合のための一時表領域の設定
- パラレルでの索引の作成
- パラレル DML のその他の考慮事項

### 並列性のタイプ

使われる並列性のタイプは、パラレル操作によって異なります。最適な物理データベース・レイアウトは、アプリケーションの主要なパラレル操作に依存します。

並列性の基本単位は、グラニュルと呼ばれます。並列化される操作（表走査または表更新、索引作成など）は、Oracle によってグラニュルに分けられます。問合せサーバー・プロセスでは、操作が一度に 1 グラニュルずつ実行されます。グラニュルの数とそのサイズは、使用できる DOP（並列度）に影響します。また、問合せサーバー・プロセス間で作業のバランスがどの程度うまくとられるかにも影響します。

### ブロック範囲グラニュル

ブロック範囲グラニュルは、ほとんどのパラレル操作の基本単位です。これは、パーティション表にもあてはまります。このため、Oracle では、並列度はパーティション数に関連しません。

ブロック範囲グラニュルとは、表の物理ブロックの範囲のことです。これらは物理データ・アドレスに基づいているので、Oracle はブロック範囲グラニュルのサイズを変更して、より優れたロード・バランシングを実現します。ブロック範囲グラニュルによって、表または索引の静的な事前割当てに依存しない動的な並列性が実現されます。SMP（対称型マルチプロセッサ）システムでは、できる限り多数のディスクを駆動できるように、グラニュルは異なるデバイスに配置されます。多くの MPP（超並列）システムでは、ブロック範囲グラニュルは、グラニュルを保管するディスクに物理的に近い問合せサーバー・プロセスに優先的に割り当てられています。ブロック範囲グラニュルは、グローバルなストライプ化でも使用されます。

ブロック範囲グラニュルが、表または索引へのパラレル・アクセスで優先的に使用されている場合、管理上の考慮事項（回復や、パーティションを使ったデータの部分削除など）はパフォーマンス上の考慮事項よりもパーティション・レイアウトに影響を与えることがあります。パーティションをストライプ化するディスク数は、パラレル実行操作がパーティションのブルニングを利用できるように、少なくとも DOP の値と同じにする必要があります。

**関連項目：** MPP システムについては、プラットフォーム固有のマニュアルを参照してください。

## パーティション・グラニュル

パーティション・グラニュルを使用しているとき、問合せサーバー・プロセスは、表や索引のパーティション全体またはサブパーティションに対して作業をします。パーティション・グラニュルは、表または索引の作成時に静的に決定されるので、操作をそれほど柔軟に並列化できるわけではありません。したがって、許容される DOP が制限されたり、問合せサーバー・プロセス間で負荷のバランスをうまくとれなかったりする可能性があります。

パーティション・グラニュルは、パラレル索引範囲走査、およびパーティション表やパーティション索引の複数パーティションを修正するパラレル操作の基本単位です。これらの操作には、パラレル更新およびパラレル削除、パーティション表へのパラレル・ダイレクト・ロード・インサート、パーティション索引のパラレル作成、パーティション表のパラレル作成が含まれます。パラレル DML や CREATE LOCAL INDEX などの操作は、ブロック範囲グラニュルを認識しません。

パーティション・グラニュルを表または索引へのパラレル・アクセスに使用する際に大切なのは、Oracle が問合せサーバー・プロセス間で作業のバランスを効果的にとれるように、パーティションの数が多い（少なくとも DOP の 3 倍）であることです。

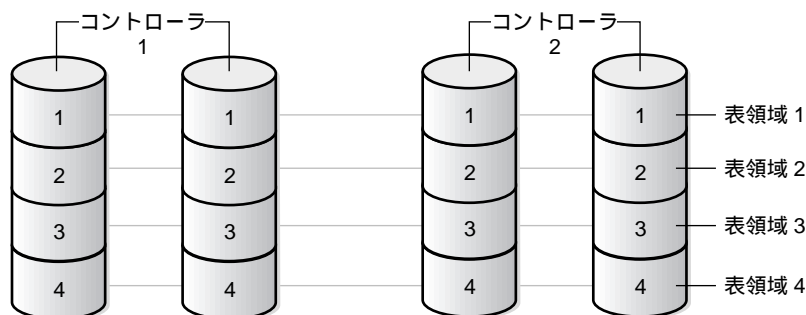
**関連項目：** ディスクのストライプ化とパーティション化の詳細は、『Oracle8i 概要』を参照してください。

## データのストライプ化

パラレル処理中の I/O のボトルネックを回避するには、パラレル操作によってアクセスされる表領域をストライプ化する必要があります。図 26-2 で示すように、表領域は、常に少なくとも CPU と同じ数のデバイスにストライプ化する必要があります。この例では 4 つの CPU があります。パーティション・グラニュルで説明したように、これらの表領域をストライプ化するディスク数は、少なくとも DOP に設定された値と等しくする必要があります。

表、索引の表領域、ロールバック・セグメントおよび一時表領域をストライプ化します。また、コントローラまたは I/O チャンネル、内部バス（あるいはすべて）にデバイスを振り分けることも必要です。

図 26-2 最少でも CPU 数と同数のデバイスへのオブジェクトのストライプ化



ロード中にデータをストライプ化するには、パラレル・ローダーの「FILE=」句を使用して、複数のロード・セッションから表領域内の異なるファイルにデータをロードします。ストライプ化を効果的にするには、ストライプ化した表領域にデータを並列で I/O する帯域幅をサポートするために、十分なコントローラおよびその他の I/O コンポーネントを確保する必要があります。

オペレーティング・システムのボリューム・マネージャでもストライプ化を実行できます (OS ストライプ化)。または、パラレル操作のためのストライプ化を手動で行うこともできます。

可能な限り、OS ストライプ化を使用し少なくとも 64KB の大きなストライプ・サイズを指定することをお勧めします。この方法は、特にマルチユーザー環境では、手動ストライプ化よりも常に効果的に実行されます。

**オペレーティング・システムのストライプ化** オペレーティング・システムのストライプ化は、一般に柔軟性が高く、管理が容易です。並列に実行するシングル・ユーザーだけでなく、順次実行する複数ユーザーもサポートします。システムが非常に小さい場合、または可用性が主な目標である場合以外は、次の 2 つの利点により OS ストライプ化の方が手動ストライプ化よりも有効です。

- パラレル走査操作では (全表走査または高速全走査) オペレーティング・システムのストライプ化によってディスク・シーク回数が増加します。それにもかかわらず、これは I/O サイズ ( $DB\_BLOCK\_SIZE * MULTIBLOCK\_READ\_COUNT$ ) を大きくすることによってかなり補正されます。このようにすると、この操作が、使用しているプラットフォームで最大の I/O スループットを達成できます。一般に、この最大値はプラットフォームのコントローラまたは I/O バスの数によって制限されるもので、(ディスク数の構成が特に少なくない限り) ディスク数によっては制限されません。
- 索引プロープ (たとえば、ネスト・ループ・ジョインまたはパラレル索引範囲走査に含まれる) では、オペレーティング・システムのストライプ化を行うとホット・スポットを回避できます。つまり、I/O がディスク間でさらに均等に分散されます。

ストライプ・サイズは少なくとも I/O サイズと同じ大きさをなければなりません。ストライプ・サイズが I/O サイズの 2 倍または 4 倍の大きさである場合は、特定のトレードオフが発生する場合があります。ストライプ・サイズが大きいと、システムが各ディスクに対する順次操作をさらに多く実行でき、ディスクのシーク回数が減少するので効果があります。不利な点は、I/O の並列性が低下するので、同時にアクティブになるディスク数が減少することです。これによって問題が発生した場合は、ストライプ・サイズを変更するのではなく、走査操作の I/O サイズを増やしてください（たとえば、64KB を 128KB にします）。最大 I/O サイズはプラットフォームに固有です（たとえば、64KB ~ 1MB の範囲）。

OS ストライプ化では、パフォーマンスの面から見た、最適なレイアウトは、プラットフォームのすべてのディスク上にデータおよび索引、一時表領域をストライプ化することです。この方法では、1 つのオブジェクトがパラレル操作でアクセスされるときに、（スループットと 1 秒あたりの I/O 数の両方の面で）最大の I/O パフォーマンスに到達できます。同時に複数のオブジェクトがアクセスされると（マルチユーザー構成の場合など）、ストライプ化によって自動的に競合が制限されます。可用性が主要な考慮事項である場合は、この方式とハードウェアの冗長性（たとえば RAID5）を組み合わせてください。これによって、パフォーマンスと可用性の両方が実現されます。

**手動ストライプ化** 手動ストライプ化は、すべてのプラットフォームで行えます。手動ストライプ化を行うには、それぞれ別のディスク上にある複数のファイルを各表領域に追加します。手動ストライプ化を適正に行うと、システムのパフォーマンスが大幅に改善されます。ただし、適正にストライプ化しないとパフォーマンスに悪影響を与えるおそれのある欠点がいくつかあるので、これらを認識しておく必要があります。

第 1 に、手動ストライプ化を行うと、DOP は CPU 数よりもディスク数に依存します。その理由は、すべてのディスクを駆動し、I/O ボトルネックが発生する危険性を減らすために、データファイルあたり 1 つのサーバー・プロセスを持つ必要があるからです。また、手動ストライプ化は、データファイル・サイズの偏りの影響を大きく受けます。データファイル・サイズの偏りはパラレル走査操作の拡張性に影響することがあります。第 2 に、手動ストライプ化には、オペレーティング・システムのストライプ化よりも多くの計画と設定作業が必要です。

**関連項目：** ディスクのストライプ化とパーティション化の詳細は、『Oracle8i 概要』を参照してください。MPP システムでオペレーティング・システムのストライプ化を行うときのディスク親和性の使用禁止の適否は、プラットフォーム固有の Oracle マニュアルを参照してください。

## ローカルおよびグローバルのストライプ化

ローカルのストライプ化はパーティション表またはパーティション索引にのみ適用され、非オーバーラップのディスク対パーティションというストライプ化の形式をとります。各パーティションには、表 26-3 で示されているように、独自のディスクおよびファイルの集合があります。オーバーラップ・ディスク・アクセスやファイルのオーバーラップはありません。

ローカルのストライプ化の利点は、あるディスクで障害が発生しても他のパーティションに影響しないことです。また、データが1つのパーティションにしかない場合でもストライプ化されていることです。

ローカルのストライプ化の欠点は、各パーティションにそれぞれ数台のディスクが必要なので、インプリメントするにはかなり余分にディスクが必要となることです。もう1つの欠点は、1つまたは少数のパーティションのみに対するパーティションのプルニングの後で、システムの I/O 帯域幅が制限されることです。結果として、ローカルのストライプ化は、パラレル操作ではそれほど実際的ではないといえます。この理由で、ローカルのストライプ化を考慮するのは、主要な目標が可用性であり、パラレル実行ではない場合だけにしてください。妥協案としては、グローバルなストライプ化を RAID5 と組み合わせて使うのがよいでしょう。これによって、パフォーマンスと可用性の両方が認められます。

図 26-3 ローカルのストライプ化

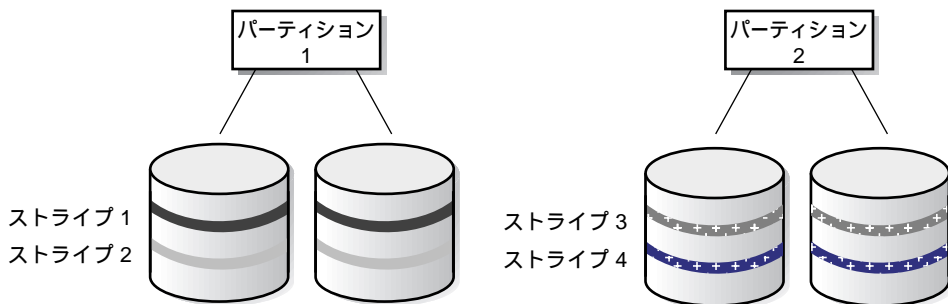
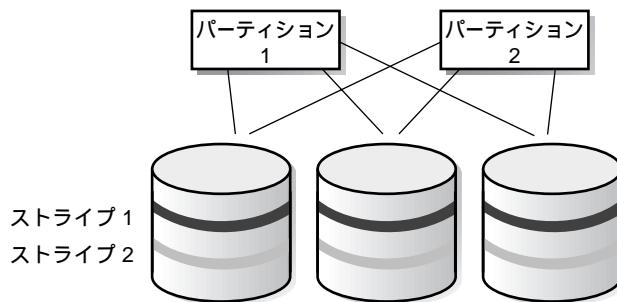


図 26-4 に示されているグローバルのストライプ化には、オーバーラップ・ディスクおよびオーバーラップ・パーティションが含まれます。

図 26-4 グローバルなストライプ化



グローバルなストライプ化は、パーティション・プルニングがあり、1つのパーティションのデータにしかアクセスする必要のない場合に便利です。そのパーティションのデータを多数のディスクに分散することで、パラレル実行操作のパフォーマンスが改善されます。グローバルなストライプ化の欠点は、1つのディスクに障害が起こると、すべてのパーティションに影響が及ぶことです。

**関連項目：** 26-42 ページの「[ストライプ化およびメディア回復](#)」

## ストライプ化の分析

アプリケーションのストライプ化の問題を分析するときには、2つの考慮事項があります。第1に、記憶システム内のオブジェクト間の関連のカーディナリティを考慮します。第2に、全表走査、一般の表領域の可用性、パーティション走査、これらの目標のいくつかの組合せなど、ストライプ化作業で何を最適化できるかを検討します。これら2つのトピックについては、次の見出しで説明します。

**記憶オブジェクトの関連のカーディナリティ** ストライプ化を分析するには、次の関連を考慮する必要があります。

図 26-5 関連のカーディナリティ

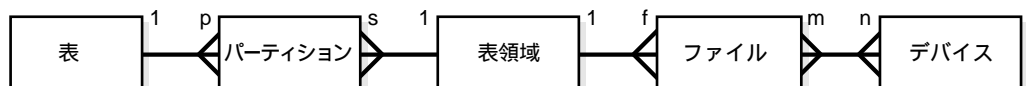


図 26-5 は、典型的な Oracle 記憶システム内のオブジェクト間の関連のカーディナリティを示しています。それぞれの表に対して、次のものがある可能性があります。

- 図 26-5 で示す、1 対多関連としての  $p$  個のパーティション。
- 図 26-5 で示す、それぞれの表領域に対する多対 1 関連としての  $s$  個のパーティション。
- 図 26-5 で示す、それぞれの表領域に対する 1 対多関連としての  $f$  個のファイル。
- 図 26-5 で示す、 $n$  個のデバイスに対する多対多関連としての  $m$  個のファイル。

**目標。** デバイス間にオブジェクトをストライプ化して、次の 3 つの目標を達成します。

- 目標 1: 全表走査の最適化。これは、1 つの表を多数のデバイスへ配置することを意味します。
- 目標 2: 可用性の最適化。これは、表領域を少数のデバイスに制限することを意味します。
- 目標 3: パーティション走査の最適化。これは、各パーティションを多数のデバイスに配置することで、パーティション内並列性を達成することを意味します。

目標 1 と目標 2 の両方を達成する（表を多数のデバイスに常駐させ、かつできるだけ高い可用性を実現する）には、パーティションの数（ $p$ ）を最大にし、表領域あたりのパーティションの数（ $s$ ）を最小にします。

最大の可用性と最小のパーティション内並列性を実現するには、各パーティションを個々の表領域に置きます。ストライプ化されたファイルは使わず、表領域ごとにファイルを 1 つ使用します。目標 2 を最小化して可用性を最小化するには、 $f$  と  $n$  が 1 と等しくなるように設定します。

可用性を最小化するとパーティション内並列性は最大になります。目標 3 の計算式の最大化と目標 2 の計算式の最小化を同時に行うことはできないので、目標 3 と目標 2 は競合します。両方の目標をある程度達成するには、妥協が必要です。

**目標 1: 全表走査の最適化。** 1 つの表を多数のデバイスに配置すると、全表走査がスケーラブルになるので有効です。

パーティション数および表領域内のファイル数、ファイルあたりのデバイス数の 3 つを掛け合わせます。その値を、同じ表領域を共有するパーティション数と、同じデバイスを共有するファイル数との積で割ります。計算式は次のようになります。

$$\text{Number of devices per table} = \frac{p \times f \times n}{s \times m}$$



それぞれの表領域にファイルが 1 つずつあり、そのファイルがストライプ化されていない場合は、各パーティションが独自の表領域にある  $t$  個のパーティションでこれを達成できます。

$$t \times 1 / p \times 1 \times 1, \text{ up to } t \text{ devices}$$

表がパーティション化されていないで、1 つのファイル内の 1 つの表領域にある場合は、 $n$  個のデバイスにストライプ化します。

$$1 \times 1 \times n \text{ devices}$$

それぞれ独自の表領域にあるパーティションが最大  $t$  個あり、各表領域に  $f$  個のファイルがあり、各表領域がストライプ化されたデバイス上にある場合。

$$t \times f \times n \text{ devices}$$

**目標 2: 可用性の最適化。** 各表領域を少数のデバイスに制限し、できるだけ多数のパーティションを持つと、高い可用性を達成するのに役立ちます。

$$\text{Number of devices per tablespace} = \frac{f \times n}{m}$$

可用性が最大になるのは、 $f = n = m = 1$  であり、 $p$  が 1 を大きく上回るときです。

**目標 3: パーティション走査の最適化。** パーティション内並列性を達成すると、パーティションの走査がスケーラブルになるので有効です。これを行うには、各パーティションを多数のデバイスに配置します。

$$\text{Number of devices per partition} = \frac{f \times n}{s \times m}$$

パーティションは、多数のファイルを持つ表領域に常駐させることができます。次の 2 通りの可能性があります。

- 表領域ごとに多数のファイルがある場合。
- ファイルがストライプ化されている場合。

## ストライプ化およびメディア回復

ストライプ化はメディア回復に影響を与えます。通常、ディスクの損害は、そのディスクに格納されていた全オブジェクトにアクセスできなくなることを意味します。すべてのオブジェクトがすべてのディスク上にストライプ化されている場合、どのディスクに損害が発生しても全データベースが停止します。さらに、障害が発生したディスクに格納されているデータが全体のごく一部であったとしても、すべてのデータベース・ファイルをバックアップから復元しなければならない可能性があります。

ストライプ化を提供しているのと同じ OS のサブシステムがミラー化も提供していることがよくあります。ディスク価格の下落にともない、ミラー化でバックアップおよびログ・アーカイブを効果的に補足できますが、代替にはなりません。ミラー化は、バックアップを使用するよりも短時間でシステムがデバイス障害から回復するのに役立ちますが、バックアップほど確実ではありません。独立したバックアップはソフトウェア障害やその他の問題からシステムを保護しますが、ミラー化はこれらに対して保護は行いません。

元のソース・テープから読み専用データを再ロードできる場合は、ミラー化を効果的に行えます。ディスク障害がある場合に、バックアップからデータを復元するとダウン時間が長くなることがありますが、ミラー化ディスクからデータを復元すればシステムを短時間でオンライン状態に戻すことができます。

ミラー化よりもさらに費用がかからないのが RAID テクノロジーです。書き込み操作にはコストがかかるので、RAID は完全な複写を回避します。"ほとんど読みみしか行わない" アプリケーションでは、これで十分な場合があります。

---

**注意：** RAID テクノロジーは書き込み操作では特に遅くなります。このような低速性がデータベースの復元時間に影響し、RAID のパフォーマンスが許容できなくなる場合があります。

---

**関連項目：** データファイル間での表の手動ストライプ化の詳細は、20-22 ページ「[ディスクのストライプ化](#)」を参照してください。メディア回復の問題の説明は、11-8 ページの「[データ・ウェアハウスのバックアップとリカバリ](#)」を参照してください。

自動ファイル・ストライプ化、およびデバイス間の I/O 分散を判別するために使うツールの詳細は、オペレーティング・システムのマニュアルを参照してください。

## データのパーティション化

この項では、データ・アクセスとアプリケーション全体のパフォーマンスを大幅に改善するパーティション化機能について説明します。これは、数 100 万の行や何ギガバイトものデータが収められている表や索引にアクセスするアプリケーションに特にあてはまります。

パーティション表およびパーティション索引を使用すると、管理上の操作をデータのサブセットに対して実行できるので、操作が容易になります。たとえば、新しいパーティションの追加、既存のパーティションの編成またはパーティションの削除は、読み込み専用アプリケーションへの 1 秒未満の割込みで行えます。

この項で説明するパーティション化方式を使用して、SQL 文をチューニングし、(パーティションのプルニングを使用して) 索引や表の不要な走査を回避できます。大量のデータ(たとえば数 100 万行)を結合するときに、大規模な結合走査のパフォーマンスも改善できます。これは、パーティション・ワイズ・ジョインを使用して行います。最後に、データのパーティション化によって、非常に大規模なデータベースの管理容易性が非常に増し、バックアップや復元などの管理作業に必要な時間が大幅に削減されます。

### パーティション化のタイプ

Oracle には、3 つのパーティション化方式が用意されています。

- レンジ
- ハッシュ
- コンポジット

各パーティション化方式には、それぞれ異なる長所と短所があります。したがって、各方式は、他の方式が適さない特定の状況に適しています。

**関連項目：** パーティション化の詳細は、『Oracle8i 概要』を参照してください。

**レンジ・パーティション化** レンジ・パーティション化は、各パーティションに対して設定した列値の範囲によって判別される境界に基づいて、データをパーティションにマップします。この方式は、主に、履歴データを管理する DSS アプリケーションに役立ちます。

**ハッシュ・パーティション化** ハッシュ・パーティション化は、ユーザーがパーティション化キーを指定し、Oracle がそのキーに適用するハッシング・アルゴリズムに基づいて、データをパーティションにマップします。ハッシング・アルゴリズムは、行をパーティション間に均等に分散します。したがって、結果として生成されるパーティションのセットは、ほぼ同じサイズになります。このため、ハッシュ・パーティション化は、データをデバイス間に均等に分散するのに理想的です。またハッシュ・パーティション化は、データの内容が履歴データでないときに、レンジ・パーティション化のかわりに使用するのに適した使いやすい方法です。

---

**注意：** ユーザー指定のハッシング・アルゴリズムは作成できません。

---

**コンポジット・パーティション化** コンポジット・パーティション化は、レンジ・パーティション化とハッシュ・パーティション化の機能を組み合わせたものです。コンポジット・パーティション化では、Oracle は最初にパーティションのレンジによって設定された境界に従って、データをパーティションに分散します。さらに Oracle は、各レンジ・パーティション内でデータをサブパーティションに分割します。Oracle は、ハッシング・アルゴリズムを使用してデータをサブパーティションに分散します。

**関連項目：** 詳細は、『Oracle8i 概要』と『Oracle8i 管理者ガイド』を参照してください。

## 索引パーティション化

レンジ、ハッシュまたはコンポジットによってパーティション化された表には、ローカルとグローバルの両方の索引を作成できます。ローカル索引は、関連する表のパーティション化属性を継承します。たとえば、コンポジット・パーティション表にローカル索引を作成すると、Oracle は、コンポジット方式を使用してローカル索引を自動的にパーティション化します。

Oracle は、グローバル索引に対してはレンジ・パーティション化のみをサポートしています。したがって、ハッシュ・パーティション化方式やコンポジット・パーティション化方式を使用してグローバル索引をパーティション化することはできません。

## レンジ、ハッシュおよびコンポジット・パーティション化のパフォーマンス上の考慮点

次の項では、レンジ、ハッシュおよびコンポジット・パーティション化のパフォーマンス上の考慮点について説明します。

**レンジ・パーティション化のパフォーマンスに関する考慮事項** 前述したように、レンジ・パーティション化は、履歴データをパーティション化するのに適した方式です。レンジ・パーティションの境界は、表または索引のパーティションの順序を定義します。

レンジ・パーティション化の最も一般的な使用方法では、「日付」型の列上の期間でデータをパーティション化します。このため、レンジ・パーティションにアクセスする SQL 文は、時間枠に焦点を合わせる傾向があります。この例として、「特定期間のデータを検索する」のような SQL 文があります。このようなシナリオでは、各パーティションが 1 か月のデータを表す場合に、「98 年 12 月のデータを検索する」問合せがアクセスする必要があるのは、98 年の December パーティションだけです。これによって、走査されるデータ量は、使用可能なデータ全体の数分の 1 に削減されます。この最適化方式を、「パーティションのブルニング」といいます。

レンジ・パーティションは、新規データを定期的にロードして古いデータを削除する場合にも理想的です。このようなパーティションの '追加' と '削除' は、管理容易性の主要な改善点です。

データの 'ローリング・ウィンドウ' を保持することはよく行われます。たとえば、最近 36 か月のデータをオンラインに保持する場合などです。レンジ・パーティション化によって、このプロセスが単純化されます。新しい月のデータを追加するには、そのデータを別の表にロードして、データを整形します。さらに、索引を作成し、EXCHANGE PARTITION コマンドを使用してそのデータをレンジ・パーティション表に追加します。これらのすべての処理は、表をオンラインにしたまま行うことができます。新しいパーティションを追加したら、DROP PARTITION コマンドを使用して対象から外れた月を削除できます。

結論として、次のような場合にはレンジ・パーティション化を検討してください。

- 非常に大規模な表が、ある列に対する範囲述語によって頻繁に走査される場合。その列は適切なパーティション化列になります (ORDER\_DATE や PURCHASE\_DATE など)。その列で表をパーティション化すると、パーティションのプルニングが使用可能になります。
- データの 'ローリング・ウィンドウ' をメンテナンスする場合。
- 大規模な表に対するバックアップや復元などの管理操作が、割り当てられた時間枠内に完了しない場合。
- パラレル DML (PDML) 操作をインプリメントする必要がある場合。

次の SQL 例は、1994 年と 1995 年の 2 年間について表 "Sales" を作成し、その表を列 s\_saledate の範囲によってパーティション化することで、それぞれ 1 つのパーティションに対応する 8 つの四半期にデータを分割します。

```
CREATE TABLE sales
(s_productid NUMBER,
s_saledate DATE,
s_custid NUMBER,
s_totalprice NUMBER)
PARTITION BY RANGE(s_saledate)
(PARTITION sal94q1 VALUES LESS THAN TO_DATE ('01-APR-1994', 'DD-MON-YYYY'),
PARTITION sal94q2 VALUES LESS THAN TO_DATE ('01-JUL-1994', 'DD-MON-YYYY'),
PARTITION sal94q3 VALUES LESS THAN TO_DATE ('01-OCT-1994', 'DD-MON-YYYY'),
PARTITION sal94q4 VALUES LESS THAN TO_DATE ('01-JAN-1995', 'DD-MON-YYYY'),
PARTITION sal95q1 VALUES LESS THAN TO_DATE ('01-APR-1995', 'DD-MON-YYYY'),
PARTITION sal95q2 VALUES LESS THAN TO_DATE ('01-JUL-1995', 'DD-MON-YYYY'),
PARTITION sal95q3 VALUES LESS THAN TO_DATE ('01-OCT-1995', 'DD-MON-YYYY'),
PARTITION sal95q4 VALUES LESS THAN TO_DATE ('01-JAN-1996', 'DD-MON-YYYY'));
```

**ハッシュ・パーティション化のパフォーマンスに関する考慮点** Oracle がデータをハッシュ・パーティションに分散する方法には、データの内容によるものではありません。したがって、ハッシュ・パーティション化は、履歴データを管理するのに効果的な方法ではありません。

ません。ただし、ハッシュ・パーティションは、それ以外のレンジ・パーティションのパフォーマンス特性はすべて備えています。これは、パーティションのプルニングの使用が等価述語に制限されることを意味します。パーティション・ワイズ・ジョイン、パラレル索引アクセスおよび PDML を使用することもできます。

**関連項目：** パーティション・ワイズ・ジョインについては、この章の 26-50 ページの「[パーティション・ワイズ・ジョイン](#)」で後述します。

一般的なルールとして、ハッシュ・パーティション化は次のような目標に使います。

- データをパーティション化するため。たとえば、大規模な表の可用性や管理容易性を改善したり、PDML を使用可能にしたりする必要があるときに、表に履歴データが格納されていないためレンジ・パーティション化が適さない場合。
- パーティション間でのデータの偏りを回避するため。Oracle はいくつかのパーティションにデータをハッシュします。各パーティションはそれぞれ別のデバイス上にあってもかまわないので、ハッシュ・パーティション化はデータの分散手段として効果的です。このため、I/O スループットを最大化するのに必要な数のデバイスに、データを均等に分散できます。同様に、ハッシュ・パーティション化を使って、Oracle Parallel Server を使用する MPP プラットフォームのノード間にデータを均等に分散できます。
- パーティション化キーに従ってパーティションのプルニングとパーティション・ワイズ・ジョインを使用することが重要な場合。

---

**注意：** ハッシュ・パーティション化では、パーティションのプルニングは等価述語またはインリスト述語を使用する場合に制限されます。

---

ハッシュ・パーティションの追加や連結を行うと、Oracle は行を自動的に並べ替えて、パーティションやサブパーティションの数の変更を反映します。Oracle が使用するハッシュ機能は、特にこの再編成のコストを軽減するように設計されています。表内のすべての行を入れ替えるかわりに、Oracle は、既存のハッシュ・パーティションの 1 つだけを分割するパーティション追加論理を使用します。反対に、Oracle は、既存の 2 つのハッシュ・パーティションをマージすることでパーティションを連結します。

これによってハッシュ・パーティション表の管理容易性は大幅に改善されますが、ハッシュ・パーティション表のパーティション数、またはコンポジット・パーティション表の各パーティションのサブパーティション数が 2 の累乗でない場合は、ハッシュ機能によって偏りが引き起こされることがあります。パーティション数を 2 の累乗にしないと、最悪の場合には最大パーティションが最小パーティションの 2 倍のサイズになることがあります。したがって、パフォーマンスを最適化するために、パーティション数またはパーティションあたりのサブパーティション数は、2 の累乗で作成してください。たとえば、2、4、8、16、32、64、128、256、512、1024 などにします。

次の例は、列 `s_productid` をパーティション・キーとして使用し、表 "Sales" に対して 4 つのハッシュ・パーティションを作成します。

```
CREATE TABLE sales
(s_productid NUMBER,
 s_saledate DATE,
 s_custid NUMBER,
 s_totalprice NUMBER)
PARTITION BY HASH(s_productid)
PARTITIONS 4;
```

パーティション名は、一部のパーティションのプロパティを表とは異なるプロパティにしたい場合にのみ指定します。それ以外の場合は、Oracle がパーティションの内部名を自動的に生成します。また、STORE IN 句を使用して、ラウンドロビン方式で表領域にパーティションを割り当てることができます。

**コンポジット・パーティション化のパフォーマンスに関する考慮点** コンポジット・パーティション化は、レンジ・パーティション化とハッシュ・パーティション化の両方の利点を備えています。コンポジット・パーティション化では、Oracle は最初に範囲によってパーティション化します。さらに、各範囲内にサブパーティションを作成し、ハッシュ・アルゴリズムを使用してデータをサブパーティション内で分散します。Oracle は、ハッシュ・パーティション表の場合と同じハッシュ・アルゴリズムを使用して、コンポジット・パーティション表のハッシュ・サブパーティション間にデータを分散します。

コンポジット・パーティションに置かれるデータは、レンジ・レベル・パーティションを定義するために使用するパーティション境界についてだけ論理的に並べられます。各パーティション内のデータのパーティション化では、サブパーティションが属するパーティションを識別する以外に論理的な編成は行われません。

したがって、コンポジット方式を使用してパーティション化された表とローカル索引には、次のことがあてはまります。

- パーティション・レベルで履歴データをサポートします。
- PDML などのパラレル操作（たとえば、領域管理やバックアップとリカバリ）の並列性の単位として、サブパーティションの使用をサポートします。
- レンジ・ディメンションとハッシュ・ディメンションでのパーティションのプルニングとパーティション・ワイズ・ジョインの対象となります。

**コンポジット・パーティション化の使用法** 表とローカル索引に対するコンポジット・パーティション化方式は、次のような場合に使用します。

- 履歴データを効率よくサポートするために、パーティションに論理的な意味が必要な場合。
- パーティションの内容が、複数の表領域、デバイスまたは MPP システムのノードに分散することがある場合。

- ブルニングと結合の述語がパーティション表の異なる列を使用する場合にも、パーティションのブルニングとパーティション・ワイズ・ジョインの両方を使用する必要がある場合。
- バックアップ、回復およびパラレル操作で、パーティション数より多い並列度を使用したい場合。

コンポジット方式を使用すると、Oracle は各サブパーティションを異なるセグメントに格納します。したがって、サブパーティションのプロパティは、そのサブパーティションが属する表やパーティションのプロパティとは異なる場合があります。

次の SQL 例は、表 "Sales" を列 `s_saledate` の範囲によってパーティション化し、4 つのパーティションを作成します。この SQL は、時間枠によるデータの並べ替えの利点があります。また、各レンジ・パーティション内では、列 `s_productid` のハッシュによって、データがさらに 4 つのサブパーティションに分割されます。

```
CREATE TABLE sales(
  s_productid NUMBER,
  s_saledate DATE,
  s_custid NUMBER,
  s_totalprice)
PARTITION BY RANGE (s_saledate)
SUBPARTITION BY HASH (s_productid) SUBPARTITIONS 4
(PARTITION sal94q1 VALUES LESS THAN TO_DATE (01-APR-1994, DD-MON-YYYY),
PARTITION sal94q2 VALUES LESS THAN TO_DATE (01-JUL-1994, DD-MON-YYYY),
PARTITION sal94q3 VALUES LESS THAN TO_DATE (01-OCT-1994, DD-MON-YYYY),
PARTITION sal94q4 VALUES LESS THAN TO_DATE (01-JAN-1995, DD-MON-YYYY));
```

各ハッシュ・サブパーティションには、製品コード別に並べられた 1 つの四半期の売上が含まれます。サブパーティションの合計数は 16 です。

## パーティション・ブルニング

パーティション・ブルニングは、コストベースのオブティマイザを使用して SQL 文の FROM 句と WHERE 句を分析することによって問合せ実行を改善し、パーティション・アクセス・リストを作成するときに不要なパーティションを排除します。これによって、Oracle は、SQL 文に関連するパーティションに対してのみ操作を実行できます。Oracle は、レンジ・パーティション化列に対して範囲、等価およびインリスト述語を使用するとき、およびハッシュ・パーティション化列に対して等価およびインリスト述語を使用するときのみ、これを行うことができます。

また、パーティション・ブルニングによって、ディスクから検索するデータ量が大幅に削減され、処理時間が短縮されます。その結果、問合せのパフォーマンスとリソース使用がかなり改善されます。索引と表を異なる列でパーティション化する場合は、基礎となる表のパーティションを排除できないときでも、パーティション・ブルニングによって索引パーティションも排除されます。これは、グローバル・パーティション索引を作成することで行います。



コンジョイント・パーティション・オブジェクトでは、Oracle は関連する述語を使用して、レンジ・パーティション・レベルとハッシュ・サブパーティション・レベルの両方でプルニングできます。たとえば、列 `s_saledate` の範囲によってパーティション化され、列 `s_productid` のハッシュによってサブパーティション化されている前の例の表 `Sales` を参照し、次の SQL 文を検討します。

```
SELECT * FROM sales
WHERE s_saledate BETWEEN TO_DATE(01-JUL-1994, DD-MON-YYYY) AND
TO_DATE(01-OCT-1994, DD-MON-YYYY) AND s_productid = 1200;
```

Oracle は、パーティション列に対する述語を使用してパーティションのプルニングを次のように実行します。

- レンジ・ディメンションでは、Oracle は `sal94q2` と `sal94q3` にのみアクセスします。
- ハッシュ・ディメンションでは、Oracle は、`s_productid` が 1200 と等しい行がマップされる 3 番目のパーティション `h3` にのみアクセスします。

## DATE 列を使ったプルニング

前の例では、`TO_DATE` ファンクションを使用して日付値が 4 桁の年で完全に指定されています。これは日付値の指定の推奨形式ですが、次の例のように他の形式を使用するときでも、オブティマイザは述語 `s_saledate` を使用してパーティションをプルニングできます。

```
SELECT * FROM sales
WHERE s_saledate BETWEEN TO_DATE(01-JUL-94, DD-MON-YY) AND
TO_DATE(01-OCT-94, DD-MON-YY) AND s_productid = 1200;

SELECT * FROM sales
WHERE s_saledate BETWEEN '01-JUL-1994' AND
'01-OCT-1994' AND s_productid = 1200;
```

ただし、SQL 文の `EXPLAIN PLAN` コマンド出力の `partition_start` および `partition_stop` 列で通常示されるような、Oracle がアクセスするパーティションを参照することはできません。かわりに、両方の列のキーワード `'KEY'` が表示されます。

## I/O ボトルネックの回避

前述したように、一部のパーティションがプルニングによって排除されたため Oracle がすべてのパーティションを走査しない場合に I/O のボトルネックを回避するには、複数のデバイスに各パーティションを分散します。MPP システムでは、それらのデバイスを複数のノードに分散します。

## パーティション・ワイズ・ジョイン

パーティション・ワイズ・ジョインは、結合がパラレルで実行されるときに問合せサーバー間で交換されるデータ量を最小化することで、問合せの応答時間を短縮します。これによって、CPU とメモリーの両方について、応答時間とリソース使用が大幅に削減されます。OPS (Oracle Parallel Server) 環境では、パーティション・ワイズ・ジョインによって、インターコネクト上のデータ通信量が回避されるか、少なくとも制限されます。これは、大規模な結合走査の拡張性を高めるキーとなります。

パーティション・ワイズ・ジョインには、次の見出しの下で説明するように、フルおよびパーシャルという 2 つのバリエーションがあります。

### フル・パーティション・ワイズ・ジョイン

フル・パーティション・ワイズ・ジョインは、2 つの結合表の 1 対のパーティション間で、大きな結合を小さな結合に分割します。この機能を使うには、両方の表を結合キーで同一レベル・パーティション化する必要があります。たとえば、sales 表と customer 表の間での列 'customerid' に対する大規模な結合について考えます。問合せ "1994 年第 3 四半期に 100 個より多い数の商品を購入したすべての顧客のレコードを検索する" は、大規模な結合を実行する SQL 文の典型的な例です。次に、この SQL 文の例を示します。

```
SELECT c_customer_name, COUNT(*)
FROM sales, customer
WHERE s_customerid = c_customerid
AND s_saledate BETWEEN TO_DATE('01-jul-1994', 'DD-MON-YYYY') AND
TO_DATE('01-oct-1994', 'DD-MON-YYYY')
GROUP BY c_customer_name HAVING
COUNT(*) > 100;
```

この結合は、データ・ウェアハウス環境で典型的な大規模結合です。customer 表全体が、販売データの 1 つの四半期に結合されます。大規模なデータ・ウェアハウス・アプリケーションでは、数 100 万行が結合される場合があります。このような場合に使用する結合方式は、明らかにハッシュ結合です。しかし、両方の表が customerid 列で同一レベル・パーティション化されている場合は、このハッシュ結合の処理時間をさらに短縮できます。これによって、フル・パーティション・ワイズ・ジョインが使用可能になります。

26-34 ページの「[並列性のタイプ](#)」で説明したように、フル・パーティション・ワイズ・ジョインをパラレルで実行するときは、並列性のグラニクルはパーティションです。その結果、並列度はパーティション数に制限されます。たとえば、問合せの並列度を 16 に設定するには、少なくとも 16 のパーティションが必要です。

両方の表を列 customerid で 16 のパーティションに同一レベル・パーティション化するには、さまざまなパーティション化方式を使用することができます。これらの方式については、次のサブセクション以降で説明します。

**ハッシュ/ハッシュ** これは、最も単純な方式です。Customer 表と Sales 表は、それぞれ s\_customerid と c\_customerid に対するハッシュによって、どちらも 16 のパーティションにパーティション化されています。表が customerid 列で結合されているときは、このパー

パーティション化方式によってフル・パーティション・ワイズ・ジョインが使用可能になります。

シリアルでは、この結合は一度に一組の対応するハッシュ・パーティションのペアの間で実行されます。1つのパーティション・ペアが結合されると、別のパーティション・ペアの結合が始まります。結合は、16のパーティション・ペアが処理されると完了します。

---

**注意：** 一致するハッシュ・パーティションのペアは、各表で同じパーティション番号を持つパーティションと定義されています。たとえば、フル・パーティション・ワイズ・ジョインでは、sales のパーティション 0 を customer のパーティション 0 に結合し、sales のパーティション 1 を customer のパーティション 1 に結合します。

---

フル・パーティション・ワイズ・ジョインのパラレル実行は、シリアル実行の単純なパラレル化です。1つのパーティション・ペアを一度に結合するかわりに、16のパーティション・ペアを16の問合せサーバーによってパラレルで結合します。図 26-6 に、フル・パーティション・ワイズ・ジョインのパラレル実行を示します。

図 26-6 フル・パーティション・ワイズ・ジョインのパラレル実行

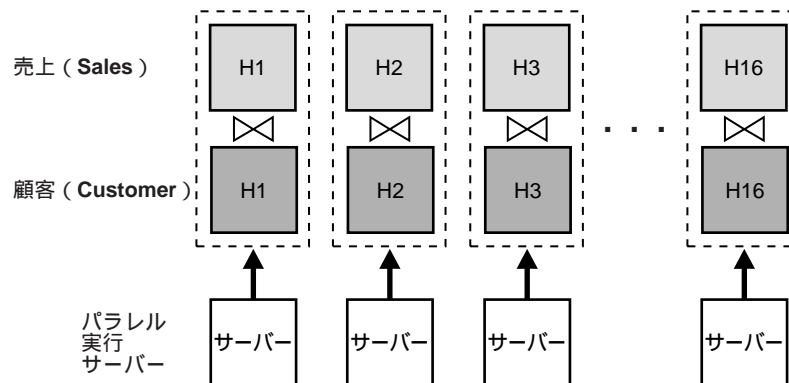


図 26-6 では、並列度とパーティション数が同じであること、つまりどちらも 16 であることを想定しています。パーティション数を並列度よりも多くすると、ロード・バランシングを改善し、実行での偏りの可能性を制限できます。問合せサーバー数よりも多くのパーティションがある場合、問合せサーバーは、1 ペアのパーティションの結合を完了すると、結合する別のペアを問合せコーディネータに要求します。このプロセスは、すべてのペアが処理されるまで繰り返されます。この方式では、パーティション・ペアの数が並列度よりも多い

とき、たとえばパーティション数が 64 で並列度が 16 のときに、動的なロード・バランシングが可能になります。

---

**注意：** パーティション数は、必ず並列度の倍数にしてください。

---

シェアード・ナッシング・プラットフォームまたは MPP で稼働している Oracle Parallel Server 環境で、優れた拡張性を達成するためには、ノードへのパーティションの配置が重要です。リモート I/O を回避するために、対応するパーティションの両方が同じノードに対して親和性を持っている必要があります。ボトルネックを回避し、システムで使用可能なすべての CPU リソースを使用するために、パーティション・ペアをすべてのノードに分散する必要があります。

ただし、ノード数よりもペアの数が多いときは、1 つのノードに複数のペアを置くことができます。たとえば、8 ノードのシステムに 16 のパーティション・ペアがある場合は、各ノードに 2 つのペアを置く必要があります。

**関連項目：** データの親和性の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

**コンボジット/ハッシュ** この方式は、ハッシュ/ハッシュ方式のバリエーションです。sales 表は、履歴データを格納する表の典型的な例です。26-44 ページの「[レンジ・パーティション化のパフォーマンスに関する考慮事項](#)」で説明したすべての理由により、sales に対するより論理的なパーティション化方式は、おそらくハッシュ方式ではなくレンジ方式です。

たとえば、Sales 表を s\_saledate の範囲によって 8 パーティションにパーティション化するとします。また、2 年分のデータがあり、各パーティションが四半期を表すとします。レンジ・パーティション化のかわりにコンボジット・パーティション化を行うと、フル・パーティション・ワイズ・ジョインを使用可能にする一方で、s\_saledate でのパーティション化も保持できます。これを行うには、s\_saledate の範囲によって Sales 表をパーティションしてから、パーティションあたり 16 のサブパーティション、合計で 128 のサブパーティションを使用して、各パーティションを s\_customerid に対するハッシュでサブパーティション化します。customer 表では、依然として 16 パーティションのハッシュ・パーティション化を行います。

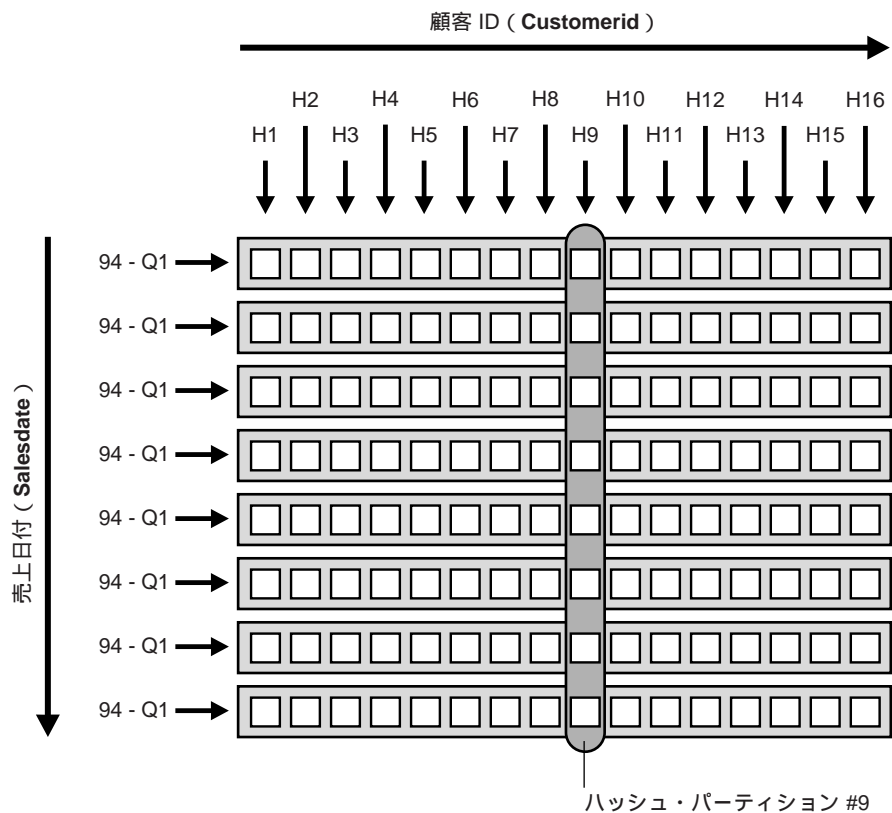
この新しいパーティション化方式では、フル・パーティション・ワイズ・ジョインはハッシュ/ハッシュ方式と同じように機能します。結合は、依然として両方の表のハッシュ・パーティション・ペアの間で 16 の小さい結合に分割されます。違いは、Sales 表の各ハッシュ・パーティションが、8 つのサブパーティション（各レンジ・パーティションから 1 つずつ）からなる 1 つのセットで構成されていることです。

[図 26-7](#) に、Sales 表でハッシュ・パーティションがどのように形成されるかを示します。[図 26-7](#) では、各セルがサブパーティションを表しています。各行が 1 つのレンジ・パーティションに対応し、合計で 8 つのレンジ・パーティションがあります。それぞれのレンジ・パーティションには、16 のサブ・パーティションがあります。対称的に、図の各列は 1 つの

ハッシュ・パーティションに対応し、合計で 16 のハッシュ・パーティションがあります。各ハッシュ・パーティションには 8 つのサブパーティションがあります。ハッシュ・パーティションを定義できるのは、すべてのパーティションに同じ数（この例では 16）のサブパーティションがある場合のみであることに注意してください。

コンボジット表のハッシュ・パーティションは暗黙的ですが、ただし、Oracle は、ハッシュ・パーティションをデータ・ディクショナリには記録しないので、レンジ・パーティションの場合のように DDL コマンドを使用してこれら进行操作することはできません。

図 26-7 コンボジット表のレンジ・パーティションとハッシュ・パーティション



このパーティション化方式は、(s\_salesdate での) プルニングを (customerid での) フル・パーティション・ワイズ・ジョインと組み合わせることができるので効果的です。前の例の問合せでは、プルニングは 1994 年の Q3 に対応するサブパーティション（図 26-7 の行番号 3）

を走査することでのみ達成できます。Oracle は、フル・パーティション・ワイズ・ジョインを使用して、これらのサブパーティションを customer 表と結合します。

ハッシュ / ハッシュ方式のすべての特性は、コンボジット / ハッシュ方式にも適用されます。この例では特に、次の 2 つのポイントが両方の方式に共通しています。

- Sales 表に 128 のサブパーティションがある場合でも、ハッシュ・パーティションは 16 しかないので、フル・パーティション・ワイズ・ジョインの並列度が 16 を超えてはありません。
- MPP システムのデータ配置のルールがここにも適用されます。違いは、ハッシュ・パーティションがサブパーティションの集合であることだけです。これらすべてのサブパーティションは、他の表の一致するハッシュ・パーティションと同じノードに置く必要があります。たとえば、図 26-7 では、丸で囲まれた 8 つのサブパーティションで示されている Sales 表のハッシュ・パーティション 9 を、Customer 表のハッシュ・パーティション 9 と同じノードに格納する必要があります。

**コンボジット / コンボジット (ハッシュ・ディメンション)** 必要に応じて、Customer 表をコンボジットでパーティション化することもできます。たとえば、この表を郵便番号列の範囲によってパーティション化し、郵便番号に基づくプルニングを使用可能にできます。さらに、customerid のハッシュによってサブパーティション化し、ハッシュ・ディメンションでのパーティション・ワイズ・ジョインを使用可能にします。

**レンジ / レンジ** パーティション・ワイズ・ジョインは、レンジ・パーティション化でも使用することができます。ただし、結合を実行する前にデータの分布を調査しておく必要がありますので、インプリメントはより複雑です。さらに、パーティション・バウンドを正しく指定していないため、実行中にデータの偏りを引き起こすことがあるので、パーティションのサイズを均一にする必要があります。

レンジ / レンジを使用する場合の基本的な原則は、ハッシュ / ハッシュの場合と同じです。つまり、両方の表を同一レベル・パーティション化する必要があります。したがって、パーティションの数が同じで、パーティション・バウンドが同一であることが必要です。たとえば、1000 万人の顧客がいることがあらかじめわかっている、customerid の値が 1 ~ 10000000 の範囲であるとしてします。つまり、1000 万の異なる値のある可能性があるとします。16 のパーティションを作成するには、両方の表をレンジ・パーティション化します。Sales は s\_customerid で、Customer は c\_customerid でパーティション化します。両方の表のパーティション・バウンドを定義して、同じサイズのパーティションを生成する必要があります。この例では、パーティション・バウンドは 625001、1250001、1875001、...、10000001 として定義されているので、パーティションには 625000 行が含まれます。

**レンジ / コンボジット、コンボジット / コンボジット (レンジ・ディメンション)** 最後に、一方または両方の表を別の列でサブパーティション化することもできます。したがって、レンジ・ディメンションでのフル・パーティション・ワイズ・ジョインを使用可能にするために、レンジ・ディメンションでのレンジ / コンボジット方式とコンボジット / コンボジット方式も有効です。

## パーシャル・パーティション・ワイズ・ジョイン

Oracle は、パーシャル・パーティション・ワイズ・ジョインのみをパラレルで実行できます。フル・パーティション・ワイズ・ジョインとは異なり、パーシャル・パーティション・ワイズ・ジョインでは、両方ではなく一方の表のみを結合キーでパーティション化する必要があります。パーティション表は、参照表と呼ばれます。他の表は、パーティション化されている場合とされていない場合があります。パーシャル・パーティション・ワイズ・ジョインは、結合された表の 1 つだけを結合キーでパーティション化する必要があるため、フル・パーティション・ワイズ・ジョインよりも一般的です。

パーシャル・パーティション・ワイズ・ジョインを実行するために、Oracle は、参照表のパーティション化に基づいて他の表を動的に再パーティション化します。他の表が再パーティション化された後の実行は、フル・パーティション・ワイズ・ジョインの場合と同様です。

パーシャル・パーティション・ワイズ・ジョインのパフォーマンスが従来のパラレル結合よりも優れている点は、参照表が結合操作中に移動されないことです。従来のパラレル結合では、両方の入力表を結合キーで再分散する必要があります。この再分散操作には、問合せサーバー間での行の交換が必要です。この交換は非常に CPU を使用する操作であり、OPS 環境で過度のインターコネクト通信量が発生する原因になります。外部キーまたは主キーのいずれかの結合キーで大規模表をパーティション化することによって、表がそのキーで結合されるたびにこの再分散が行われることを回避できます。もちろん、外部キーを選択して表をパーティション化する場合（最も一般的なシナリオ）は、多くの問合せに関係する外部キーを選択します。

パーシャル・パーティション・ワイズ・ジョインを図示するために、前の Sales/Customer の例を考えます。customer は、パーティション化されていないか、c\_customerid 以外の列でパーティション化されているとします。Sales は customerid で Customer に結合されることが多く、この結合はアプリケーションの作業負荷の中心となるので、s\_customerid で Sales をパーティション化すると、Customer と Sales が結合されるたびにパーシャル・パーティション・ワイズ・ジョインが使用可能になります。フル・パーティション・ワイズ・ジョインの場合と同様に、いくつかの代替方法があります。

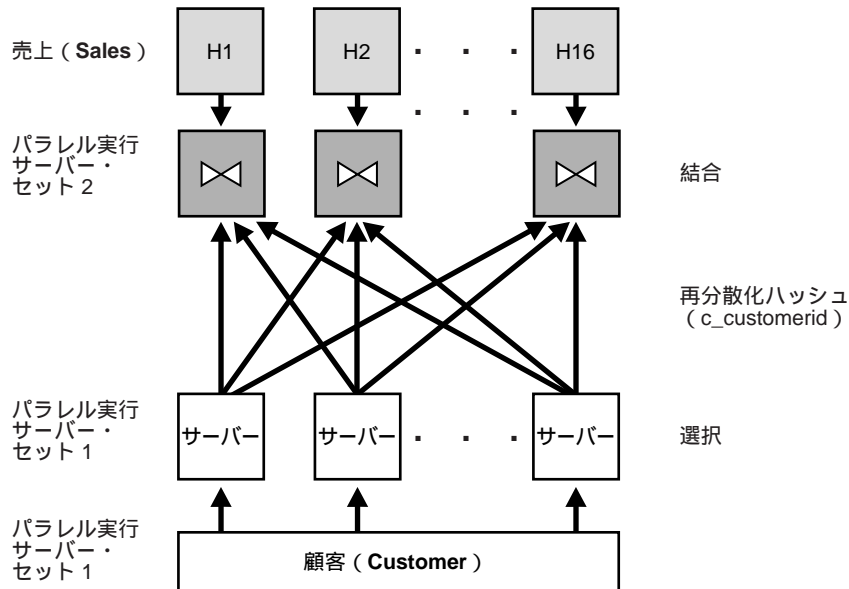
**ハッシュ** パーシャル・パーティション・ワイズ・ジョインを使用可能にする最も単純な方法は、c\_customerid のハッシュによって Sales をパーティション化することです。パーティションは、パラレル・パーティション・ワイズ・ジョイン操作の並列性の最小グラニクルなので、パーティションの数によって最大並列度が決まります。

並列度と Sales のパーティション数が両方とも 16 だとすると、パーシャル・パーティション・ワイズ・ジョインのパラレル実行は図 26-8 で示されるようになります。実行には、2 セットの問合せサーバーが関係します。図で 'set 1' のラベルが付いている一方のセットは、customer 表をパラレルで走査します。走査操作の並列性のグラニクルは、ブロック範囲です。

最初のセットで選択された Customer の行（この場合はすべての行）は、customerid をハッシュすることで問合せサーバーの 2 番目のセットに再分散されます。たとえば、Sales のパーティション H1 に一致する行を持つ Customer のすべての行は、2 番目のセットの問合せサーバー 1 に送信されます。問合せサーバーの 2 番目のセットによって受信された行は、

Sales 内の対応するパーティションの行と結合されます。たとえば、2 番目のセットの間合せサーバー番号 1 は、受信したすべての Customer 行を Sales のパーティション H1 に結合します。

図 26-8 パーシャル・パーティション・ワイズ・ジョイン



フル・パーティション・ワイズ・ジョインに関する考慮事項は、パーシャル・パーティション・ワイズ・ジョインにも適用されます。

- 並列度は、パーティション数と同じである必要はありません。図 26-8 では、間合せは 8 セットの間合せサーバーで実行されています。この場合、Oracle は、2 つのパーティションを 2 番目のセットの各間合せサーバーに割り当てます。この場合も、パーティション数は必ず並列度の倍数とする必要があります。
- シェアード・ナッシング・プラットフォーム (MPP) で稼動している Oracle Parallel Server 環境では、リモート I/O を回避するために、Sales の各ハッシュ・パーティションはできるだけ 1 つのノードに対してのみ親和性を持つ必要があります。また、ボトルネックを回避し、システムで使用可能なすべての CPU リソースを使用するために、パーティションをすべてのノードに分散する必要があります。ノード数よりもパーティション数の方が多い場合は、複数のパーティションを収容するノードが 1 つあれば十分です。



**関連項目：** データの親和性の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

**コンポジット フル・パーティション・ワイズ・ジョイン**の場合と同様に、Sales 表に最適なパーティション化方式は、列 `s_salesdate` に対してレンジ方式を使用することです。これは、Sales 表が履歴データを格納する表の典型的な例であることが理由です。このレンジ・パーティション化を保持したままパーシャル・パーティション・ワイズ・ジョインを使用可能にするには、パーティションあたり 16 のサブパーティションを使用して、列 `s_customerid` に対するハッシュによって Sales をサブパーティション化します。問合せが Customer と Sales を結合する場合、また問合せが `s_salesdate` に対して選択述語を持つ場合は、ブルニングとパーシャル・パーティション・ワイズ・ジョインを一緒に使用することができます。

Sales がコンポジットである場合は、パーシャル・パーティション・ワイズ・ジョインの並列性のグラニクルは、サブパーティションではなくハッシュ・パーティションです。コンポジット表のハッシュ・パーティションの定義については、[図 26-7](#) を参照してください。ここでも、ハッシュ・パーティション数は、並列度の倍数とする必要があります。また、MPP システムでは、各ハッシュ・パーティションが単一のノードに対して親和性を持つようにしてください。前の例では、ハッシュ・パーティションを構成する 8 つのサブパーティションが、同じノードに対して親和性を持っている必要があります。

**レンジ** 最後に、`s_customerid` でのレンジ・パーティション化を使用して、パーシャル・パーティション・ワイズ・ジョインを使用可能にできます。これは、ハッシュ方式と同じような働きをしますが、お薦めはしません。パーティションのサイズが異なる場合は、結果のデータ分布が偏る可能性があります。さらにこの方式は、結合キーでもあるパーティション列の値をあらかじめ知っておく必要があるので、インプリメントがより複雑です。

## パーティション・ワイズ・ジョインの利点

パーティション・ワイズ・ジョインには、この項で説明するような利点があります。

- **通信オーバーヘッドの削減**
- **メモリ要件の削減**

**通信オーバーヘッドの削減** パーティション・ワイズ・ジョインは、パラレルで実行されるときに、通信オーバーヘッドを削減します。その理由は、デフォルトのケースでは、1 セットのパラレル実行サーバーで結合操作をパラレル実行するためには、各表を結合列で行の重複のないサブセットに再分散する必要があるからです。これらの行の重複のないサブセットは、単一のパラレル実行サーバーによってペア単位で結合されます。

2 つの表は結合列ですでにパーティション化されているので、Oracle はパーティションの再分散を回避できます。これによって、各パラレル実行サーバーは、一致するパーティションのペアを結合できます。

このパフォーマンス強化は、ノード間パラレル実行を行う OPS 構成でより顕著です。パーティション・ワイズ・ジョインにより、インターコネクト通信量が大幅に削減されるからです。この機能の使用は、OPS を使用する大規模な DSS 構成では必須の最適化手法です。

現在、MPP や SMP クラスタなどのほとんどの OPS プラットフォームでは、処理能力に比べてインターコネクト帯域幅が非常に制限されています。インターコネクト帯域幅は、ディスク帯域幅と同等であるのが理想ですが、このようなケースはめったにありません。その結果、OPS でのほとんどの結合操作では、この最適化を行わないとインターコネクトの待ち時間が非常に長くなります。

**メモリー要件の削減** パーティション・ワイズ・ジョインは、より少ないメモリー量しか必要としません。シリアル結合の場合は、一度に一組の対応するパーティションのペアに対して結合が実行されます。したがって、データがパーティションに均等に分散されている場合は、メモリー要件がパーティション数で割られます。この場合は偏りが発生しません。

パラレルの場合は、パラレルで結合されるパーティション・ペアの数に依存します。たとえば、並列度が 20 でパーティション数が 100 の場合は、2 つのパーティションの 20 の結合のみが同時に実行されるので、必要なメモリー量は 5 分の 1 になります。パーティション・ワイズ・ジョインでは少ないメモリー量しか必要でないという事実は、パフォーマンスに直接影響します。たとえば、結合は、ハッシュ結合の構築フェーズ中にブロックをディスクへ書き込む必要がありません。

### パラレル・パーティション・ワイズ・ジョインのパフォーマンスに関する考慮点

パラレル・パーティション・ワイズ・ジョインによるパフォーマンスの改善は、欠点も伴います。コストベースのオプティマイザは、パーティション・ワイズ・ジョインを使用するかどうかを決定するときに、利点と欠点をはかりにかけます。

- レンジ・パーティション化の場合は、パーティションのサイズが異なると、データの偏りによって応答時間が長くなります。一部のパラレル実行サーバーで、結合を終了するのに他のサーバーよりも長い時間がかかるからです。パーティション数が 2 の累乗だとすると、ハッシュ・パーティション化によって偏りの危険性が制限されるので、ハッシュ（サブ）パーティション化を使用してパーティション・ワイズ・ジョインを使用可能にすることをお勧めします。
- パーティション・ワイズ・ジョインに使用されるパーティション数は、できるだけ問合せサーバー数の倍数にしてください。たとえば、並列度が 16 の場合は、パーティション（またはサブパーティション）数を 16、32、または 64 とすることができます。ただし、並列度がたとえば 17 の場合、Oracle は結合の最後のフェーズをシリアルで実行します。その理由は、実行の開始時に、各パラレル問合せサーバーが異なるパーティション・ペアを処理するからです。この最初のフェーズの終わりに、1 つのペアのみが残されています。したがって、1 つの問合せサーバーがこの残りのペアを結合している間、他の問合せサーバーはアイドル状態になります。
- パラレル結合は、リモート I/O の原因となることがあります。たとえば、MPP 構成で稼働している Oracle Parallel Server 環境では、一致するパーティションのペアが同じノードに置かれていないと、リモート I/O により、パーティション・ワイズ・ジョインで余分なノード間通信が必要になります。これは、結合が実行されるノードに Oracle が少なくとも 1 つのパーティションを転送する必要があることが原因です。この場合は、

パーティション・ワイズ・ジョインを使用するよりも、データを明示的に再分散する方が賢明です。

## フェーズ 3: データベースの作成、移入およびリフレッシュ

この項では、次のトピックについて説明します。

- [パラレル・ロードによるデータベースへの移入](#)
- [パラレル・ソートおよびハッシュ結合のための一時表領域の作成](#)
- [パラレルでの索引の作成](#)
- [パラレル SQL 文の実行](#)
- [EXPLAIN PLAN を使ったパラレル操作計画の表示](#)
- [パラレル DML のその他の考慮事項](#)

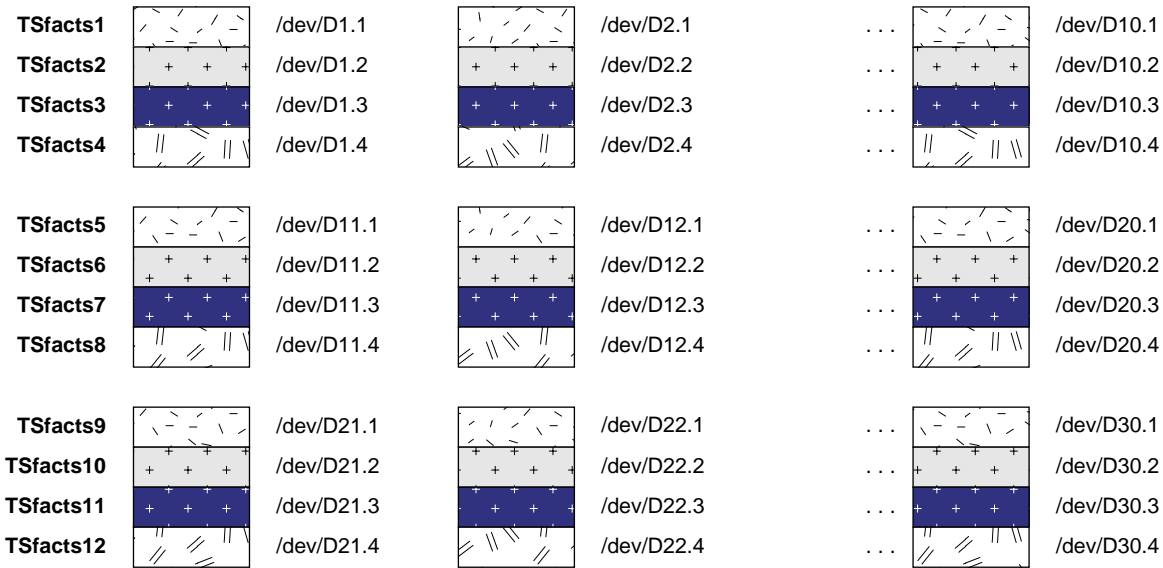
### パラレル・ロードによるデータベースへの移入

この項では、パーティションのある大規模なデータ・ウェアハウスにおける、一般的なスター・スキーマのファクト表の作成、ロード、索引付けおよび分析方法を説明するケース・スタディを示します。この例では、SQL Loader を使用して 30 個のディスクに明示的にデータをストライプ化します。

- 例の 120GB の表に、FACTS という名前を付けます。
- システムは、100 台を超えるディスク・ドライブを備え、10 個の CPU を持つ共有メモリー・コンピュータです。
- 30 台のディスク（各 4GB）が実表データのために使用され、10 台のディスクが索引、30 台のディスクが一時領域のために使用されます。その他のディスクは、ロールバック・セグメント、制御ファイル、ログ・ファイルおよびローダー・フラット・ファイル用に作成される可能性のあるステージング領域などのために必要になります。
- FACTS 表は、月ごとに 12 の論理パーティションにパーティション化されます。バックアップおよび回復を行いやすくするために、各パーティションは個々の表領域に保管されます。
- 各パーティションは 10 台のディスクに均等に分散されるので、一部のパーティションまたは 1 つのパーティションにアクセスする走査は、完全に並列に実行できます。このため、問合せでパーティションのプルニングによってデータ・アクセスが制限されているときに、パーティション内並列性が実現できます。
- 各ディスクはさらに、OS ユーティリティを使用して、/dev/D1.1、/dev/D1.2、...、/dev/D30.4 のような名前を持つ 4 つの OS ファイルに分けられます。
- 4 つの表領域が、10 ディスクずつの各グループに割り当てられます。I/O のバランスをとり表領域作成を並列化する（Oracle では、表領域にデータファイルが追加されると、

ブロックの1つ1つがデータ・ファイルに書き込まれる) ために最も望ましいのは、10 ディスクの各グループの4つの表領域がそれぞれ、異なるディスク上に最初のデータ・ファイルを持つことです。したがって、最初の表領域には最初のデータファイルとして /dev/D1.1 があり、2 番目の表領域には最初のデータファイルとして /dev/D4.2 があります。図 26-9 に示すように、その他も同様になります。

図 26-9 パラレル・ロード例のデータファイル・レイアウト



**ステップ 1: 表領域を作成し、データ・ファイルをパラレルで追加する**  
次に、"Tsfacts1" という表領域を作成するコマンドを示します。その他の表領域も、類似のコマンドで作成されます。CPU が 10 個のマシンでは、12 の CREATE TABLESPACE コマンドをすべて一緒に実行できます。あるいは、2 つのバッチに分けて 6 つずつ実行することもできます ( ディスクの 3 グループのそれぞれから 2 つ )

```
CREATE TABLESPACE Tsfacts1
DATAFILE /dev/D1.1'  SIZE 1024MB REUSE
DATAFILE /dev/D2.1'  SIZE 1024MB REUSE
DATAFILE /dev/D3.1'  SIZE 1024MB REUSE
DATAFILE /dev/D4.1'  SIZE 1024MB REUSE
DATAFILE /dev/D5.1'  SIZE 1024MB REUSE
DATAFILE /dev/D6.1'  SIZE 1024MB REUSE
```

```
DATAFILE /dev/D7.1' SIZE 1024MB REUSE
DATAFILE /dev/D8.1' SIZE 1024MB REUSE
DATAFILE /dev/D9.1' SIZE 1024MB REUSE
DATAFILE /dev/D10.1' SIZE 1024MB REUSE
DEFAULT STORAGE (INITIAL 100MB NEXT 100MB PCTINCREASE 0)
CREATE TABLESPACE Tsfacts2
DATAFILE /dev/D4.2' SIZE 1024MB REUSE
DATAFILE /dev/D5.2' SIZE 1024MB REUSE
DATAFILE /dev/D6.2' SIZE 1024MB REUSE
DATAFILE /dev/D7.2' SIZE 1024MB REUSE
DATAFILE /dev/D8.2' SIZE 1024MB REUSE
DATAFILE /dev/D9.2' SIZE 1024MB REUSE
DATAFILE /dev/D10.2' SIZE 1024MB REUSE
DATAFILE /dev/D1.2' SIZE 1024MB REUSE
DATAFILE /dev/D2.2' SIZE 1024MB REUSE
DATAFILE /dev/D3.2' SIZE 1024MB REUSE
DEFAULT STORAGE (INITIAL 100MB NEXT 100MB PCTINCREASE 0)
...
CREATE TABLESPACE Tsfacts4
DATAFILE /dev/D10.4' SIZE 1024MB REUSE
DATAFILE /dev/D1.4' SIZE 1024MB REUSE
DATAFILE /dev/D2.4' SIZE 1024MB REUSE
DATAFILE /dev/D3.4' SIZE 1024MB REUSE
DATAFILE /dev/D4.4' SIZE 1024MB REUSE
DATAFILE /dev/D5.4' SIZE 1024MB REUSE
DATAFILE /dev/D6.4' SIZE 1024MB REUSE
DATAFILE /dev/D7.4' SIZE 1024MB REUSE
DATAFILE /dev/D8.4' SIZE 1024MB REUSE
DATAFILE /dev/D9.4' SIZE 1024MB REUSE
DEFAULT STORAGE (INITIAL 100MB NEXT 100MB PCTINCREASE 0)
...
CREATE TABLESPACE Tsfacts12
DATAFILE /dev/D30.4' SIZE 1024MB REUSE
DATAFILE /dev/D21.4' SIZE 1024MB REUSE
DATAFILE /dev/D22.4' SIZE 1024MB REUSE
DATAFILE /dev/D23.4' SIZE 1024MB REUSE
DATAFILE /dev/D24.4' SIZE 1024MB REUSE
DATAFILE /dev/D25.4' SIZE 1024MB REUSE
DATAFILE /dev/D26.4' SIZE 1024MB REUSE
DATAFILE /dev/D27.4' SIZE 1024MB REUSE
DATAFILE /dev/D28.4' SIZE 1024MB REUSE
DATAFILE /dev/D29.4' SIZE 1024MB REUSE
DEFAULT STORAGE (INITIAL 100MB NEXT 100MB PCTINCREASE 0)
```

STORAGE 句のエクステント・サイズは、次のようにマルチブロック読み込みサイズの倍数にしてください。

$blocksize \text{ (ブロック・サイズ)} * MULTIBLOCK\_READ\_COUNT = multiblock \text{ read size (マルチブロック読み込みサイズ)}$

INITIAL と NEXT は、通常は同じ値に設定する必要があります。パラレル・ロードの場合は、適切なエクステント数を保持でき、データ・ディクショナリのボトルネックによるシリアル化とオーバーヘッド過多を回避できるように、エクステント・サイズを十分に大きくしてください。PARALLEL=TRUE がパラレル・ローダーに使われているとき、INITIAL エクステントは使われません。この場合、表領域のデフォルト記憶域句で指定されている INITIAL エクステント・サイズを、ローダー制御ファイルで指定した値（たとえば 64KB など）で上書きできます。

COMPATIBLE システム・パラメータを設定して現在のリリース番号と一致させ、表領域またはオブジェクトに対する CREATE または ALTER コマンドで、MAXEXTENTS キーワードを使用すると、表や索引は無限の数のエクステントを持つことができます。ただし、実際は、オブジェクトごとに 10,000 エクステントの制限が適切です。表または索引のエクステント数は無限なので、エクステント・サイズを等しくするために、PERCENT\_INCREASE パラメータをゼロに設定してください。

---

**注意：** エクステントの割当てが 1 分あたり 2 つまたは 3 つよりも速いペースで行われるのは、望ましくありません。詳細は、27-11 ページの「[ソートと一時データのための ST \(領域トランザクション\) エンキュー](#)」を参照してください。このため、各プロセスは 3 ~ 5 分もつエクステントを取得する必要があります。通常、ラージ・オブジェクトに対するこのようなエクステントは少なくとも 50MB になります。エクステント・サイズが小さすぎると、オーバーヘッドが大量に発生し、パラレル操作のパフォーマンスと拡張性に影響します。4 つのパーティションに均等に分けられた 4GB ディスクで可能な最大エクステント・サイズは 1GB です。100MB のエクステントで十分に実行できます。各パーティションは 100 のエクステントを持ちます。必要に応じて、表領域で作成されたオブジェクトごとに、デフォルトの記憶領域パラメータをカスタマイズできます。

---

## ステップ 2: パーティション表を作成する

12 個のパーティションがそれぞれ個々の表領域にあるパーティション表を作成します。表は複数ディメンションと複数メジャーを含みます。パーティション列は "dim\_2" という名前、日付を含みます。また、その他の列もあります。

```
CREATE TABLE fact (dim_1 NUMBER, dim_2 DATE, ...
meas_1 NUMBER, meas_2 NUMBER, ... )
PARALLEL
PARTITION BY RANGE (dim_2)(
PARTITION jan95 VALUES LESS THAN ('02-01-1995') TABLESPACE
TSfacts1,
```

```

PARTITION feb95 VALUES LESS THAN ('03-01-1995') TABLESPACE
TSfacts2,
...
PARTITION dec95 VALUES LESS THAN ('01-01-1996') TABLESPACE
TSfacts12);

```

### ステップ 3: パラレルでパーティションをロードする

この項では、パラレルでパーティションをロードする、4 つの方法を説明します。

ロードの方法が複数あることで、個々のパーティションをパラレルでロードするかどうかを制御する、SQL\*Loader の PARALLEL=TRUE キーワードの影響を管理しやすくなります。PARALLEL キーワードには、次のような制限が付随します。

- インデックスを定義できません。
- 各ローダー・セッションは、開始時に新たなエクステントを取得し、オブジェクトと対応付けられている既存の領域を使わないので、初期エクステントの設定は小さくする必要があります。
- 領域断片化問題が起こります。

ただし、パーティションごとにローダー・プロセスが 1 つある場合は、このキーワードの設定に関係なく、効率よく表をパラレルでロードできます。

#### ケース 1

この方法では、表と同じやり方でパーティション化されている 12 個の入力ファイルを想定しています。DBA は、ロードされる表のパーティションごとに入力ファイルを 1 つ持っています。DBA は、次のような文を入力し、12 個の SQL\*Loader セッションをパラレルで同時に開始します。

```

SQL*Loader DATA=jan95.dat DIRECT=TRUE CONTROL=jan95ctl
SQL*Loader DATA=feb95.dat DIRECT=TRUE CONTROL=feb95ctl
...
SQL*Loader DATA=dec95.dat DIRECT=TRUE CONTROL=dec95ctl

```

この例では、キーワード PARALLEL=TRUE は設定されません。制御ファイルでロード先のパーティションを指定しなければならないので、パーティションごとに別々の制御ファイルが必要です。これには、次のような文が含まれます。

```

LOAD INTO fact partition(jan95)

```

この方法の利点は、ローカルの索引が SQL\*Loader でメンテナンスされることです。パラレル・ロードはそれでも行われますが、パーティション・レベルなので、PARALLEL キーワードの制限は受けません。

この方法の欠点は、手動でロードする前に入力データをパーティション化しなければならないことです。

## ケース 2

これも一般的な方法ですが、ここでは、表と同じやり方でパーティション化されていない、任意の数の入力ファイルがあると想定しています。DBA は、各入力ファイルに対し個別にパラレル・ロードを実行する方法をとることができます。したがって、7つの入力ファイルがある場合、DBA は次のような文を使って7つの SQL\*Loader セッションを開始できます。

```
SQLLDR DATA=file1.dat DIRECT=TRUE PARALLEL=TRUE
```

Oracle では、入力データが正しいパーティションに入るようにパーティション化されます。この場合は、すべてのローダー・セッションが同じ制御ファイルを共有できるので、文にそれを書き込む必要はありません。

7つのローダー・セッションそれぞれがすべてのパーティションに書き込むことができるので、キーワード PARALLEL=TRUE を使う必要があります。ケース 1 では、データがロードの前にパーティション化されているので、それぞれのローダー・セッションが1つのパーティションにしか書き込みません。そのため、PARALLEL キーワードによるすべての制限が有効です。

このケースでは、Oracle は、12個の各表領域のすべてのファイルヘデータを均一に分散しようとしします。ただし、データの均一な分散が保証されているわけではありません。また、複数のローダー・プロセスが同時に同じデバイスに書き込もうとすると、ロード中に I/O 競合が起こる可能性があります。

## ケース 3

ケース 3（例で示されています）では、DBA がロードを厳密に制御する必要があります。そのため、DBA は、データ・ファイルが Oracle でパーティション化されるのと同じやり方で、入力データをパーティション化する必要があります。

この例では、30個のディスクに10プロセスを使ってロードします。これを行うには、DBA が入力を120個のファイルに前もって分割する必要があります。10プロセスが、最初のパーティションを最初の10個のディスクに並列でロードし、次に2番目のパーティションを2番目の10個のディスクに並列でロードし、12番目のパーティションまで同様にロードします。DBA は、次のコマンドをバックグラウンド・プロセスとして同時に実行します。

```
SQLLDR DATA=jan95.file1.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D1.1
...
SQLLDR DATA=jan95.file10.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D10.1
WAIT;
...
SQLLDR DATA=dec95.file1.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D30.4
...
SQLLDR DATA=dec95.file10.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D29.4
```

Oracle Parallel Server では、ローダー・セッションをノード間で均等に分割してください。読み込まれるデータ・ファイルは、常にローダー・セッションと同一のファイルに存在する必要があります。



複数のローダー・セッションが同一のパーティションに書き込むことができるので、キーワード `PARALLEL=TRUE` を使う必要があります。そのため、`PARALLEL` キーワードによって起こる制限がすべて有効になります。それでもこの方法は、すべてのデータがパーティション化を正確に反映し、精密にバランスをとることが保証されるという利点があります。

---

**注意：** この例では、パラレル・ロードがパーティション表とともに使用されていますが、この 2 つの機能はそれぞれ独立して使用できます。

---

#### ケース 4

この方法では、すべてのパーティションが同じ表領域に存在する必要があります。表領域内のデータファイルと同数の入力ファイルが必要ですが、表がパーティション化されているのと同じやり方で入力データをパーティション化する必要はありません。

たとえば、30 個のデバイスのすべてが同じ表領域内にある場合は、入力データを 30 個のファイルにパーティション化し、その後 30 個の `SQL*Loader` セッションをパラレルで開始します。最初のセッションを開始する文は、次のようになります。

```
SQLLDR DATA=file1.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D1
. . .
SQLLDR DATA=file30.dat DIRECT=TRUE PARALLEL=TRUE FILE=/dev/D30
```

この方法の利点は、ケース 3 の場合と同様に、`FILE` キーワードを使用するのでデータファイルの正確な配置を制御できることです。ただし、入力データのパーティション化は Oracle によって自動的に行われるので、値によってパーティション化する必要はありません。

この方法の欠点は、すべてのパーティションが同じ表領域に存在しなければならないことです。このため、可用性が最小化されます。

## パラレル・ソートおよびハッシュ結合のための一時表領域の作成

領域管理のパフォーマンスを最適化するために、専用の一時表領域を使用します。TSfacts 表領域では、次の例で示すように、最初に単一のデータファイルを追加し、後から残りのデータファイルをパラレルで追加します。

```
CREATE TABLESPACE TStemp TEMPORARY DATAFILE '/dev/D31'  
SIZE 4096MB REUSE  
DEFAULT STORAGE (INITIAL 10MB NEXT 10MB PCTINCREASE 0);
```

### 一時エクステントのサイズ

サーバーでは、PCTINCREASE 設定や INITIAL 設定を無視し、一時エクステントに対して NEXT 設定だけしか使われないため、一時エクステントのサイズはすべて同じになります。これは断片化の防止に役立ちます。

一般的な規則として、一時領域の需要は多く、また、パラレル・プロセスや、同時実行するその他の操作は一時表領域を共有しなければならないため、一時エクステントは永続エクステントよりも小さくする必要があります。通常、一時エクステントは 1MB ~ 10MB の範囲にしてください。ユーザーがいったんエクステントを割り当てると、そのユーザーの操作の間そのエクステントはユーザーのものになります。大規模なエクステントを割り当てたが、領域のごく一部しか使う必要がない場合は、エクステント内の未使用の領域は拘束されません。

同時に、一時エクステントは、プロセスが領域待ちに時間を費やす必要がないように、十分な大きさでなければなりません。新規エクステントの割当てと解放のときには、一時表領域が使うオーバーヘッドは永続表領域よりも少なくなります。ただし、新規一時エクステントを獲得しても、ラッチの獲得や SGA 構造体の検索には、ソートのエクステント・プールとして、SGA 領域が消費されるのと同様に、オーバーヘッドがかかります。また、エクステントが小さすぎる場合には、新しいインスタンスの起動時に、SMON が古いソート・セグメントを削除するのに時間がかかります。

### 一時表領域のオペレーティング・システムによるストライブ化

オペレーティング・システムのストライブ化は、一時表領域とともに使える代替技法です。ただし、大規模な一時表領域の場合は、メディア回復は少々困難です。ミラー化の意味がないので、RAID を使うか、一時表領域のバックアップをとります。OS でストライブ化した一時領域内のディスクが失われた場合、おそらく表領域を削除して再作成しなければなりません。120GB の例では数時間かかるでしょう。Oracle のストライブ化では、表領域から不良ディスクが削除されるだけです。たとえば、/dev/D50 に障害が起きた場合は、以下のように入力します。

```
ALTER DATABASE DATAFILE '/dev/D50' RESIZE 1K;  
ALTER DATABASE DATAFILE '/dev/D50' OFFLINE;
```

ディクショナリに対してサイズが 1KB になり、これはエクステント・サイズよりも小さいので、この破損ファイルはアクセスされなくなります。その結果、いずれ表領域の再作成が必要になります。

以下のようにして必ず一時表領域を使用可能にしてください。

```
ALTER USER scott TEMPORARY TABLESPACE TStemp;
```

**関連項目：** MPP システムでオペレーティング・システムのストライブ化を行うときのディスク親和性の使用禁止の適否は、プラットフォーム固有のマニュアルを参照してください。

## パラレルでの索引の作成

facts 表の索引は、パーティション化してもしなくてもかまいません。パーティション化されたローカル索引は、管理の点では最も簡単です。唯一不都合な点は、ローカルな非同一次元索引を検索するときに、すべての索引パーティションを検索しなければならないことです。

索引の表領域を作成する場合の考慮事項は、他の表領域を作成する場合と同様です。多くの場合は、小さいストライブ幅でのオペレーティング・システムのストライブ化が適していますが、管理を単純化するために最もよい方法は、個々の索引ごと別々の表領域を使うことです。ローカル索引の場合、対応するパーティションと同じ表領域に索引を作成したい場合があります。それぞれのパーティションがいくつかのディスクにストライブ化されている場合、回復のために個々の索引パーティションをパラレルで再作成することができます。また、オペレーティング・システムによるミラー化を行うこともできます。このような理由から、データ・ウェアハウスのために、索引を作成する文に NOLOGGING オプションを指定すると効果的です。

パーティション索引の表領域は、パーティション表の表領域と同じ方法でパラレルで作成する必要があります。

パーティション索引は、パーティション・グラニクルを使用してパラレルで作成されるので、可能な DOP の最大値はグラニクルの数と同じです。ローカル索引を作成する場合、もともとの並列度がグローバル索引作成の場合より少ないので、DOP を上げると実行が速くなる可能性があります。fact 表にローカル索引を作成する場合、次の文を使用できます。

```
CREATE INDEX I on fact(dim_1,dim_2,dim_3) LOCAL
PARTITION jan95 TABLESPACE Tsidx1,
PARTITION feb95 TABLESPACE Tsidx2,
...
PARALLEL(DEGREE 12) NOLOGGING;
```

1 月のデータのバックアップや復元をするには、表領域 Tsidx1 を管理するだけで十分です。

---

**注意：** 最適なパフォーマンスを維持するために、表および索引を頻繁に分析して、統計を最新にしてください。

---

**関連項目：** パーティション索引の説明は、『Oracle8i 概要』を参照してください。統計の分析の詳細は、第 7 章「オブティマイザ・モード、プラン・スタビリティおよびヒント」を参照してください。

## パラレル SQL 文の実行

表および索引の分析が終わると、使用する並列度に比例してパフォーマンスが向上することがわかります。以下の操作で改善されます。

- 表の走査
- NESTED LOOP JOIN ( ネスト・ループ・ジョイン )
- SORT MERGE JOIN ( ソート・マージ結合 )
- HASH JOIN
- "NOT IN"
- GROUP BY
- SELECT DISTINCT
- UNION および UNION ALL
- AGGREGATION ( 集計 )
- SQL からコールされた PL/SQL ファンクション
- ORDER BY
- CREATE TABLE AS SELECT
- CREATE INDEX ( 索引の作成 )
- REBUILD INDEX ( 索引の再作成 )
- REBUILD INDEX PARTITION ( 索引パーティションの再作成 )
- MOVE PARTITION ( パーティションの移動 )
- SPLIT PARTITION ( パーティションの分割 )
- UPDATE ( 更新 )
- DELETE ( 削除 )
- INSERT ...SELECT
- ENABLE CONSTRAINT
- STAR TRANSFORMATION ( スター型変換 )

単純なパラレル操作から始めます。SELECT COUNT(\*) FROM facts を使って、I/O スループットの合計を評価します。複雑な WHERE 句を追加することによって、CPU 性能の合計を評価します。I/O の不均衡によって、適切な物理データベース・レイアウトが提案される場合があります。単純な走査の動作を理解したら、作業負荷全体のそれぞれの局面に影響を与える、集計や結合、およびその他の操作を追加します。ボトルネックを調べてください。

問合せのパフォーマンスの他にも、パラレル・ロードおよびパラレル索引作成、パラレル DML を監視し、I/O および CPU リソースの使用状況が適切になるようにしてください。

## EXPLAIN PLAN を使ったパラレル操作計画の表示

EXPLAIN PLAN コマンドを使用して、パラレル問合せの実行計画を表示します。EXPLAIN PLAN 出力は、COST、BYTES および CARDINALITY 列にオプティマイザ情報を示します。EXPLAIN PLAN の使用方法の詳細は、[第 13 章「EXPLAIN PLAN の使用方法」](#)を参照してください。

結合文のパラレル実行を最適化する方法はいくつかあります。システムの構成の変更、この章で前述したパラメータの調整または DISTRIBUTION ヒントなどのヒントの使用が可能です。パラレル操作のヒントの詳細は、7-51 ページの「[パラレル実行のヒント](#)」を参照してください。

## パラレル DML のその他の考慮事項

データ・ウェアハウス上でパラレル挿入、更新または削除を実行してデータ・ウェアハウス・データベースをリフレッシュしようとする場合、物理データベースを設計する時点で考慮する必要がある追加的な問題があります。これらの考慮事項はパラレル実行操作には影響しません。これらの問題は次のとおりです。

- 並列度の制限
- ローカルおよびグローバルなストライプ化の使用
- INITTRANS および MAXTRANS の増加
- 使用可能なトランザクション空きリスト数の制限
- 複数のアーカイバの使用
- [NO]LOGGING オプション

### PDML とダイレクト・ロードの制限事項

PDML およびダイレクト・ロード挿入の制限事項の一覧は、『Oracle8i 概要』を参照してください。パラレルの制限に違反すると、操作はシリアルに実行されます。ダイレクト・ロード挿入の制限に違反すると、APPEND ヒントが無視され、標準の挿入が実行されます。エラー・メッセージは戻されません。

### 並列度の制限

パラレル挿入、更新または削除操作を実行している場合、DOP は表のパーティション数以下になります。

**関連項目：** 26-59 ページの「[フェーズ 3: データベースの作成、移入およびリフレッシュ](#)」

## ローカルおよびグローバルなストライプ化の使用

パラレル DML は主としてパーティション表で動作します。パラレル DML では、パラレル UPDATE や DELETE 操作による索引メンテナンスを行う間、非同期 I/O を使わずに、膨大な回数のランダム I/O 要求を生成することがあります。ローカル索引のメンテナンスでは、ローカルのストライプ化による最大の効果は I/O 競合が削減されることです。これは、1 つのサーバー・プロセスが、固有のディスク・セットとディスク・コントローラに限られるためです。また、ローカルのストライプ化により、1 つのディスクに障害が起こった場合の可用性が高まります。

グローバルな索引のメンテナンスでは（パーティション化されているか否かにかかわらず）、多数のディスクおよびディスク・コントローラ間で索引をグローバルにストライプ化することが、I/O 数を分散させる最もよい方法です。

## INITRANS および MAXTRANS の増加

グローバル索引がある場合は、グローバル索引セグメントとグローバル索引ブロックが同じパラレル DML 文のサーバー・プロセスによって共有されます。操作が同じ行に対して実行されていない場合でも、サーバー・プロセスは同じ索引ブロックを共有する可能性があります。各サーバー・トランザクションは、ブロックを変更する前に、索引ブロック・ヘッダーにトランザクション・エントリを 1 つ持つ必要があります。したがって、CREATE INDEX 文または ALTER INDEX 文では、INITRANS（各データ・ブロックに割り当てられるトランザクションの初期数）には、その索引の最大 DOP などの大きな値を設定する必要があります。MAXTRANS（データ・ブロックを更新できる同時トランザクションの最大数）は、システムがサポートできる最大数であるデフォルト値のままとしてください。この値は 255 を超えないようにする必要があります。

グローバル索引を持つ表を 10 の DOP で実行すると、10 個のサーバー・プロセスすべてが同じグローバル索引ブロックを変更しようとしています。そのため、MAXTRANS には少なくとも 10 を設定し、すべてのサーバー・プロセスが同時に変更を行えるようにする必要があります。MAXTRANS が十分に大きくないと、パラレル DML 操作が失敗します。

## 使用可能なトランザクション空きリスト数の制限

セグメントが作成されると、プロセス数とトランザクション空きリスト数は固定され、変更できなくなります。セグメント・ヘッダー内でプロセスの空きリストに大きな数を指定すると、使用可能なトランザクション空きリストの数が制限されます。次にセグメント・ヘッダーを再作成するときに、プロセスの空きリストの数を減らすことによってこの制限を軽減できます。これによって、セグメント・ヘッダー内のトランザクション空きリスト用の領域が大きくなります。

UPDATE および DELETE 操作では、各サーバー・プロセスが固有のトランザクション空きリストを要求します。したがって、パラレル DML 文がメンテナンスする必要のある任意のグローバル索引で使用可能なトランザクション空きリスト数の最小値によって、パラレル DML の DOP が効果的に制限されます。たとえば、2 つのグローバル索引があり、1 つが 50 のトランザクション空きリストを持ち、もう 1 つが 30 のトランザクション空きリストを持つ場合は、DOP が 30 に制限されます。

STORAGE 句の FREELISTS パラメータを使用して、プロセス空きリストの数を設定できます。デフォルトでは、プロセス空きリストは作成されません。

デフォルトのトランザクション空きリスト数は、ブロック・サイズによって決まります。たとえば、プロセス空きリストの数が明示的に設定されていない場合は、4KB のブロックはデフォルトで 80 のトランザクション空きリストを持ちます。トランザクション空きリストの最小値は 25 です。

**関連項目：** トランザクション空きリストの詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

## 複数のアーカイバの使用

パラレル DDL およびパラレル DML 操作で大量の REDO ログを生成することがあります。ただし、単一の ARCH プロセスで大量の REDO ログをアーカイブし続けることはできません。この問題を避けるために、複数のアーカイバ・プロセスを作成することができます。作成は、手動で行うかジョブ・キューを使用します。

## データベース・ライター・プロセス (DBWn) ワークロード

パラレル DML 操作により、短時間にバッファ・キャッシュにある大量のデータおよび索引、UNDO ブロックがダーティになります。次の構文を使用して V\$SYSTEM\_EVENT ビューを問い合わせた後で、"FREE\_BUFFER\_WAITS" の値が大き場合があります。

```
SELECT TOTAL_WAITS FROM V$SYSTEM_EVENT WHERE EVENT = 'FREE BUFFER WAITS';
```

この場合は、DBWn プロセスをチューニングしてください。空きバッファの待機がない場合は、上記の問合せで行は戻されません。

**関連項目：** 19-6 ページの「[REDO ログ・バッファのチューニング](#)」

## [NO]LOGGING オプション

[NO]LOGGING オプションは、表、パーティション、表領域および索引に適用されます。実際、NOLOGGING オプションを使うと、特定の操作（ダイレクト・ロード挿入など）のログが生成されません。NOLOGGING 属性は、INSERT 文レベルでは指定されず、表、パーティション、索引または表領域に対して、ALTER または CREATE コマンドを使用して指定されます。

表または索引に NOLOGGING が設定されている場合、パラレル・ダイレクト・ロード INSERT でも、シリアル・ダイレクト・ロード INSERT でも、UNDO および REDO ログを生成しません。NOLOGGING オプションを設定して実行しているプロセスは、REDO が生成されないので高速に実行されます。ただし、表、パーティションまたは索引に対して NOLOGGING 操作を行った後で、バックアップをとる前にメディアが故障した場合は、変更した表およびパーティション、索引のすべてに障害が発生する可能性があります。

---

---

**注意：** ダイレクト・ロード INSERT 操作（ディクショナリ更新は除く）では、UNDO ログが生成されることはありません。NOLOGGING 属性は UNDO には作用せず REDO にのみ影響します。厳密には、NOLOGGING を設定しても、ダイレクト・ロード INSERT 操作では、ほとんど取るに足りない程度の REDO が生成されます（フル・イメージ REDO ではなく無効範囲 REDO）。

---

---

下位互換性を保つために、CREATE TABLE コマンドでは依然として [UN]RECOVERABLE が代替キーワードとしてサポートされています。ただし、この機能は、将来のリリースではサポートされない可能性があります。

表領域レベルの logging 句は、表領域に作成されたすべての表、索引およびパーティションに、デフォルトのロギング属性を指定します。ALTER TABLESPACE 文を使用して既存の表領域ロギング属性を変更すると、ALTER 文の後に作成されたすべての表、索引およびパーティションは新しいロギング属性を持ちます。既存の表、索引およびパーティションのロギング属性は変わりません。表領域レベルのロギング属性は、表、索引またはパーティション・レベルでの指定によって上書きできます。

デフォルトのロギング属性は LOGGING です。ただし、ALTER DATABASE NOARCHIVELOG の発行によってデータベースを NOARCHIVELOG モードにした場合は、ロギングなしで実行できるすべての操作は、指定されたロギング属性にかかわらずログを生成しません。

**関連項目：** 『Oracle8i SQL リファレンス』



## フェーズ 4: パラレル実行のパフォーマンスの監視

フェーズ 4 では、パラレル実行のパフォーマンスの監視に関する次のトピックについて説明します。

- 動的パフォーマンス・ビューを使ったパラレル実行のパフォーマンスの監視
- セッション統計の監視
- オペレーティング・システム統計の監視

### 動的パフォーマンス・ビューを使ったパラレル実行のパフォーマンスの監視

システムを数日間にわたって実行した後で、パラレル実行のパフォーマンス統計を監視して、パラレル処理が最適かどうかを判断します。これは、このフェーズで説明するビューのいずれかを使用して行います。

#### Oracle Parallel Server でのビュー名

Oracle Parallel Server では、このフェーズで説明するビューのグローバルなバージョンによって、複数のインスタンスの統計が集計されます。このグローバルなビューには、たとえばビュー `V$FILESTAT` に対して `GV$FILESTAT` というように、"G" で始まる名前が付けられています。

#### V\$PX\_SESSION

`V$PX_SESSION` ビューでは、問合せサーバーのセッション、グループ、セットおよびサーバー数に関するデータが示されます。パラレル実行のために作業しているプロセスについてのリアルタイム・データが表示されます。この表には、要求された DOP と、操作に対して実際に付与された DOP に関する情報が含まれます。

#### V\$PX\_SESSTAT

`V$PX_SESSTAT` ビューには、`V$PX_SESSION` のセッション情報と `V$SESSTAT` 表の結合が示されます。したがって、通常のセッションで利用できるすべてのセッション統計は、パラレル実行を使用して実行されるすべてのセッションで利用できます。

#### V\$PX\_PROCESS

`V$PX_PROCESS` ビューには、パラレル処理に関する情報が含まれます。状態、セッション ID、プロセス ID およびその他の情報が含まれます。

#### V\$PX\_PROCESS\_SYSSTAT

`V$PX_PROCESS_SYSSTAT` ビューでは、問合せサーバーの状態と、バッファ割当て統計が示されます。

## V\$PQ\_SESSTAT

V\$PQ\_SESSTAT ビューでは、問合せがプロセスをどのように割り当てるか、また複数ユーザーとロード・バランシングのアルゴリズムがデフォルト値とヒント値にどのように影響するかなどの、システムの現行のすべてのサーバー・グループに関する状態が示されます。

V\$PQ\_SESSTAT は、将来のリリースでは廃止されます。

場合によっては、これらのビューからデータを検索した後に、いくつかのパラメータ設定を調整してパフォーマンスを改善する必要があります。この場合は、26-9 ページの「[ステップ 3: 一般パラメータのチューニング](#)」の説明を参照してください。これらのビューを定期的に問い合わせ、長時間実行のパラレル操作の進行状況を監視してください。

---

**注意：** 多くの動的パフォーマンス・ビューでは、Oracle がそれぞれのビューの統計を収集できるようにするために、パラメータ TIMED\_STATISTICS を TRUE に設定する必要があります。ALTER SYSTEM または ALTER SESSION コマンドを使用して、TIMED\_STATISTICS のオンとオフを切り替えることができます。

---

**関連項目：** DBMS\_STATS パッケージを使った統計の収集の詳細は、[第 7 章「オプティマイザ・モード、プラン・スタビリティおよびヒント」](#)を参照してください。

## V\$FILESTAT

V\$FILESTAT ビューでは、すべての表領域のすべてのデータファイルに対する、読み込みおよび書き込み要求、ブロック数、サービス時間が集計されます。V\$FILESTAT は、I/O や作業負荷の分散についての問題を診断するときに使用します。

V\$FILESTAT の統計を DBA\_DATA\_FILES ビューの統計と結合して、表領域ごとに I/O をグループ化することや、特定のファイル番号のファイル名を検索することができます。比率の分析を使用すると、表領域のすべてのアクティビティに対して表領域の各ファイルが使用している割合がわかります。頻繁にアクセスされる大規模なオブジェクトを 1 つだけ表領域に入れる場合、このテクニックを使うと、物理レイアウトに問題があるオブジェクトを識別できます。

DBA\_EXTENTS ビューを使うと、さらにディスク領域割当ての問題も診断できます。表領域のすべてのファイルから均等に領域が割り当てられていることを確認してください。長時間実行の操作中に V\$FILESTAT を監視して、EXPLAIN PLAN の出力に I/O アクティビティを結び付けるのは、進捗を追跡するのに適した方法です。

## V\$PARAMETER

V\$PARAMETER ビューでは、すべてのシステム・パラメータの名前、現在の設定値およびデフォルト値がリストされます。さらにこのビューは、パラメータが、ALTER SYSTEM または ALTER SESSION コマンドを使用してオンラインで変更できるセッション・パラメータかどうかを示します。

## V\$PQ\_TQSTAT

V\$PQ\_TQSTAT ビューでは、テーブル・キューのレベルでのメッセージ通信量の詳細なレポートが示されます。V\$PQ\_TQSTAT データは、パラレル SQL 文を実行しているセッションから問い合わせた場合のみ有効です。テーブル・キューは、問合せサーバー・グループ間、パラレル・コーディネータと問合せサーバー・グループの間、または問合せサーバー・グループとコーディネータの間のパイプラインです。これらのテーブル・キューは、EXPLAIN PLAN 出力内でそれぞれ行ラベル PARALLEL\_TO\_PARALLEL、SERIAL\_TO\_PARALLEL または PARALLEL\_TO\_SERIAL によって表されます。

V\$PQ\_TQSTAT には、各テーブル・キューの読み込みまたは書き込みを行う問合せサーバー・プロセスごとに 1 行があります。10 の受取側プロセスを 10 の生成側プロセスに接続しているテーブル・キューの場合、このビューには 20 行が含まれます。バイト列を合計し、TQ\_ID (テーブル・キュー識別子) ごとにグループ分けして、各テーブル・キューを介して送信されるバイト数の合計を取得します。これをオプティマイザの見積りと比較します。大きな相違があると、より大きなサンプルを使ってデータを分析する必要があることを示す場合があります。

TQ\_ID ごとにグループ分けしたバイト数の違いを計算します。大きな相違があれば、作業負荷の不均衡があることを示しています。大きな相違を調べて、生成側が不均等なデータ配分から開始しているかどうか、または配分そのものが偏っていたかどうかを判別する必要があります。データ自体が偏っている場合は、カーディナリティが低いこと、つまり固有値の数が少ないことを示している場合があります。

---

**注意：** V\$PQ\_TQSTAT ビューは、将来のリリースで V\$PX\_TQSTAT に改名されます。

---

## V\$SESSTAT および V\$SYSSTAT

V\$SESSTAT ビューでは、各セッションのパラレル実行に関連する統計が提供されます。この統計には、セッションで実行された問合せ、DML 文および DDL 文の合計数と、セッションでパラレル実行の間に交換されたインスタンス内部メッセージおよびインスタンス間メッセージの合計数が含まれます。

V\$SYSSTAT は、システム全体に対して V\$SESSTAT と同様の情報を提供します。

**関連項目：** [第 16 章「動的パフォーマンス・ビュー」](#)

## セッション統計の監視

これらの例では、これまでに説明した動的パフォーマンス・ビューを使用しています。

V\$PX\_SESSION を使用して、パラレルで実行するサーバー・グループの構成を判断します。この例では、Session ID 9 が問合せコーディネータで、セッション 7 と 21 は最初のグループの最初のセットにあります。セッション 18 と 20 は、最初のグループの 2 番目のセットにあ

ります。次の問合せに対する Oracle の応答で示されるように、この問合せについて要求され、付与された DOP は 2 です。

```
SELECT QCSID, SID, INST_ID "Inst",
SERVER_GROUP "Group", SERVER_SET "Set",
DEGREE "Degree", REQ_DEGREE "Req Degree"
FROM GV$PX_SESSION
ORDER BY QCSID, QCINST_ID, SERVER_GROUP, SERVER_SET;
```

Oracle は、次のように応答します。

QCSID	SID	Inst	Group	Set	Degree	Req Degree
	9	9	1			
	9	7	1	1	1	2
	9	21	1	1	1	2
	9	18	1	1	2	2
	9	20	1	1	2	2

5 rows selected.

**注意：** 単一インスタンスの場合は、V\$PX\_SESSION から選択し、列名 "Instance ID" は含めないでください。

GV\$PX\_SESSION を使った前の例の出力で示されているプロセスが協調して、同じタスクを完了します。次の例では、結合問合せを実行して、物理読み込みに関するこれらのプロセスの進行状況を判断します。この問合せを使用して、特定の統計を追跡します。

```
SELECT QCSID, SID, INST_ID "Inst",
SERVER_GROUP "Group", SERVER_SET "Set" ,
NAME "Stat Name", VALUE
FROM GV$PX_SESSTAT A, V$STATNAME B
WHERE A.STATISTIC# = B.STATISTIC#
AND NAME LIKE 'PHYSICAL READS'
AND VALUE > 0
ORDER BY QCSID, QCINST_ID, SERVER_GROUP, SERVER_SET;
```

Oracle は、次のような出力で応答します。

QCSID	SID	Inst	Group	Set	Stat Name	VALUE
	9	9	1		physical reads	3863
	9	7	1	1	1 physical reads	2
	9	21	1	1	1 physical reads	2
	9	18	1	1	2 physical reads	2
	9	20	1	1	2 physical reads	2

5 rows selected.

前のタイプの問合せを使用して、V\$STATNAME の統計を追跡します。この問合せは、問合せサーバー・プロセスの進行状況を観察するのに必要な回数だけ繰り返します。

次の問合せは、V\$PX\_PROCESS を使用して問合せサーバーの状態をチェックします。

```
SELECT * FROM V$PX_PROCESS;
```

出力は次のようになります。

SERV	STATUS	PID	SPID	SID	SERIAL
P002	IN USE	16	16955	21	7729
P003	IN USE	17	16957	20	2921
P004	AVAILABLE	18	16959		
P005	AVAILABLE	19	16962		
P000	IN USE	12	6999	18	4720
P001	IN USE	13	7004	7	234

6 rows selected.

**関連項目：** これらのビューの詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

## システム統計の監視

V\$SYSSTAT ビューと V\$SESSTAT ビューには、パラレル実行を監視するためのいくつかの統計が含まれます。これらの統計を使用して、パラレルの問合せ、DML、DDL、DFO および操作の数を追跡します。各問合せ、DML または DDL には、複数のパラレル操作と複数の DFO がある場合があります。

また、統計は、DOP が削減された問合せ操作数、つまり、マルチユーザー問合せ調整アルゴリズムや使用可能なパラレル実行サーバーの減少によって低下した問合せ操作の数もカウントします。

最後に、これらのビューの統計は、パラレル実行のために送信されたメッセージ数もカウントします。次の構文は、これらの統計を表示する方法の例です。

```
SELECT NAME, VALUE FROM GV$SYSSTAT
WHERE UPPER (NAME) LIKE '%PARALLEL OPERATIONS%'
      OR UPPER (NAME) LIKE '%PARALLELIZED%'
      OR UPPER (NAME) LIKE '%PX%' ;
```

Oracle は、次のような出力で応答します。

NAME	VALUE
-----	-----
queries parallelized	347
DML statements parallelized	0
DDL statements parallelized	0
DFO trees parallelized	463
Parallel operations not downgraded	28
Parallel operations downgraded to serial	31
Parallel operations downgraded 75 to 99 pct	252
Parallel operations downgraded 50 to 75 pct	128
Parallel operations downgraded 25 to 50 pct	43
Parallel operations downgraded 1 to 25 pct	12
PX local messages sent	74548
PX local messages rcv'd	74128
PX remote messages sent	0
PX remote messages rcv'd	0
14 rows selected.	

## オペレーティング・システム統計の監視

Oracle で得られる情報と、オペレーティング・システムのユーティリティ（UNIX ベースのシステムでは `sar` や `vmstat` など）を介して得られる情報はかなり重複しています。オペレーティング・システムでは、I/O、通信、CPU、メモリーとページング、スケジューリング、同期基本命令についてのパフォーマンスの統計が得られます。Oracle の `V$SESSTAT` ビューでも、同様に OS 統計の主なカテゴリが得られます。

通常は、I/O デバイスおよびセマフォ操作についてのオペレーティング・システムの情報を、データベースのオブジェクトおよび操作にマップし直すのは、Oracle の情報の場合よりも困難です。ただし、オペレーティング・システムによっては、高性能の視覚化ツールを持ち、効果的なデータ収集方法を備えていることがあります。

CPU とメモリー使用状況についてのオペレーティング・システムの情報は、パフォーマンスを調べるときに非常に重要です。おそらく、最も重要な統計は CPU 使用状況です。パフォーマンス・チューニングの低レベルの目標は、すべての CPU で CPU バウンドになることです。いったんこれが達成されると、レベルを上げて、より I/O 集中で CPU の使用が少ない代替計画を SQL レベルで探すことができます。

オペレーティング・システムのメモリーとページングの情報は、パラレルの通信、ソートおよびハッシュ結合など多くのメモリーを使うウェアハウス・サブシステム間のメモリー配分方法を制御する、多くのシステム・パラメータの微調整に有効です。

---

## パラレル実行のパフォーマンス上の問題について

この章では、パラレル実行のパフォーマンス上の問題およびその他のパフォーマンス強化の手法に関する概念を説明します。また、パラレル操作のパフォーマンス・フィードバックを得るために使用できるツールと手法について要約し、パフォーマンス上の問題の解決方法も説明します。

- [パラレル実行のパフォーマンス上の問題について](#)
- [パラレル実行のチューニングのヒント](#)
- [問題の診断](#)

**関連項目：** パラレル実行の基本原理については、『Oracle8i 概要』を参照してください。また、パラレル実行を使用するときのチューニングの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。パラレル実行を使用するときのチューニングの詳細は、オペレーティング・システム固有の Oracle マニュアルを参照してください。

### パラレル実行のパフォーマンス上の問題について

- [メモリー、ユーザーおよびパラレル実行サーバー・プロセスの計算式](#)
- [パラレル操作バッファ・プール・サイズの設定](#)
- [計算式のバランス化](#)
- [例：メモリー、ユーザーおよびパラレル実行サーバーのバランス化](#)
- [パラレル実行の領域管理問題](#)
- [Oracle Parallel Server でのパラレル実行のチューニング](#)

## メモリー、ユーザーおよびパラレル実行サーバー・プロセスの計算式

パラレル操作のチューニングのポイントは、メモリー所要量、システムがサポートできるユーザー数（プロセス数）およびパラレル実行サーバーの最大数の関係を理解することです。チューニングの目的は、特定の操作のパラレル化およびソート・マージ結合ではないハッシュ結合の使用によって、パフォーマンスを大幅に向上させることです。このパフォーマンス目標と複数ユーザーのサポートの必要性とのバランスをとる必要があります。

システムがサポートできるプロセスの最大数を考慮する場合、メモリー所要量に基づいてプロセスを3つのクラスに分けると便利です。表 27-1 で、メモリー所要量が大、中および小のプロセスを定義します。

メモリーに収まるプロセスの最大数を次のように分析します。

図 27-1 メモリー / ユーザー / サーバーの関係の計算式

$$\begin{aligned} & sga\_size \\ & + (\# \text{ low\_memory\_processes } * \text{ low\_memory\_required }) \\ & + (\# \text{ medium\_memory\_processes } * \text{ medium\_memory\_required }) \\ & + (\# \text{ high\_memory\_processes } * \text{ high\_memory\_required }) \\ \hline & total \text{ memory required } \end{aligned}$$



表 27-1 3 つのクラスのプロセスのメモリー所要量

クラス	説明
小容量メモリー・プロセス： 100KB ~ 1MB	<p>小容量メモリー・プロセスには、表走査、索引の検索、索引のネスト・ループ・ジョイン、単一行集計（GROUP BY を指定しないか、ごくわずかのグループの合計または平均）、数行しか戻さないソートおよびダイレクト・ロードが含まれます。</p> <p>このクラスのデータ・ウェアハウス・プロセスは、必要なメモリー容量の点では OLTP プロセスに似ています。プロセス・メモリーは、固定オーバーヘッドである数百キロバイトと同じくらい少なくてもかまいません。この種の操作を実行する数千ユーザーを潜在的にサポートできます。マルチスレッド・サーバーを使用すると、この所要量をさらに少なくすることができ、さらに多数のユーザーをサポートできます。</p>
中容量メモリー・プロセス： 1MB ~ 10MB	<p>中容量メモリー・プロセスには、大規模なソート、ソート・マージ結合、非常に多数の行を戻す GROUP BY または ORDER BY 操作、索引メンテナンスを伴うパラレル挿入操作および索引の作成が含まれます。</p> <p>これらのプロセスでは、操作によって異なりますが、小容量メモリー・プロセスによって必要とされる固定オーバーヘッドに加えて 1 つ以上のソート領域が必要です。たとえば、典型的なソート・マージ結合では入力を両方ともソートするので、2 つのソート領域が使用されます。多数のグループや行に関連する GROUP BY または ORDER BY 操作でも、ソート領域が必要になります。</p> <p>結合の数と種類やソートの数と種類を識別するには、その操作の EXPLAIN PLAN 出力を調べます。その計画におけるオプティマイザの統計によって、操作のサイズが表示されます。結合を計画するときは、いくつかの選択肢があることを覚えておいてください。EXPLAIN PLAN 文については、<a href="#">第 13 章「EXPLAIN PLAN の使用方法」</a>で説明します。</p>
大容量メモリー・プロセス： 10MB ~ 100MB	<p>大容量メモリー・プロセスには次の操作が含まれます。1 つ以上のハッシュ結合、または 1 つ以上のハッシュ結合と大規模ソートの組合せです。</p> <p>これらのプロセスでは、小容量メモリー・プロセスによって必要とされる固定オーバーヘッドに加えてハッシュ領域が必要です。必要となるハッシュ領域のサイズは 8MB ~ 32MB であり、2 つの領域が必要になるでしょう。2 つ以上のシリアル・ハッシュ結合を実行している場合、各プロセスが 2 つのハッシュ領域を使います。パラレル操作では、各パラレル実行サーバーは多くても一度に 1 つのハッシュ結合しか行わないので、サーバーごとに 1 つのハッシュ領域サイズが必要です。</p> <p>つまり、1 つの操作のハッシュ結合メモリーの容量は、ハッシュ領域サイズに DOP をかけた値、またはその操作のハッシュ結合数と 2 のうち少ない方に DOP をかけた値に等しくなります。</p>

**注意：** パラレル DML（データ操作言語）およびパラレル DDL（データ定義言語）操作のプロセス・メモリー所要量は、文の間合せ部分にも依存します。

## パラレル操作用バッファ・プール・サイズの設定

ユーザー・システムでサポートできるプロセスの最大数（ここでは *max\_processes* とします）を求める計算式は次のとおりです。

図 27-2 プロセスの最大数を求める計算式

$$\frac{\begin{aligned} &\# \text{ low\_memory\_processes} \\ &+ \# \text{ medium\_memory\_processes} \\ &+ \# \text{ high\_memory\_processes} \end{aligned}}{\text{max\_processes}}$$

一般に、*max\_processes* の値がユーザー数よりも大幅に大きい場合は、パラレル操作の使用を検討してください。*max\_processes* がユーザー数よりもかなり小さい場合は、27-5 ページの「[計算式のバランス化](#)」で説明するようなその他の代替方法を検討します。

パラレル更新および削除以外のパラレル操作では、一般的にバッファ・プール・サイズを大きくしても利点はありません。パラレル更新および削除により索引を更新する場合には、バッファ・プール・サイズが大きい方が効果的です。その理由は、索引の更新はランダム・アクセス・パターンで行われるので、索引全体または索引の内部ノードをバッファ・プールに保持することにより、I/O アクティビティが削減できるからです。他のパラレル操作によって利益が得られるのは、バッファ・プールのサイズを大きくすることができ、それによって、ネスト・ループ・ジョインのための内部表または索引に適応することができた場合だけです。

**関連項目：** バッファ・プール・サイズの設定の詳細は、19-24 ページの「[バッファ・キャッシュのチューニング](#)」を参照してください。

## 計算式のバランス化

図 27-1 で示した、メモリー / ユーザー / サーバーの関係の計算式をバランス化するには、次の手法を使用します。

- ページングを意識したオーバーサブスクライブ
- メモリー負荷集中プロセス数の削減
- プロセスあたりのデータ・ウェアハウス・メモリーの低減
- マルチユーザーのための並列性の低減

### ページングを意識したオーバーサブスクライブ

潜在的な負荷が、計算式で推奨される制限を超過することを許可できます。SGA サイズを除いた必要メモリーの合計に係数 1.2 をかけ、20% のオーバーサブスクライブをもたらすことが可能です。このため、メモリーが 1GB ある場合、1.2GB の要求をサポートできます。超過する 20% はページング・システムによって処理されます。

ただし、特定の度合いのオーバーサブスクライブがシステムで実行可能であることを検証する必要があります。これを検証するには、ページング率を監視して、ページング・サブシステムの待機にごく小さい割合の時間しか費やしていないことを確認します。60% のオーバー・サブ・スクライプだとしても、通常すべてのプロセスが同時にハッシュ結合を一斉に実行することがない場合、システムのパフォーマンスは許容できるでしょう。ユーザーは使用可能なメモリーよりも多くのメモリーを使用しようとすることがあるので、このような場合にはページング・アクティビティを頻繁に監視する必要があります。ページング率が急激に上昇した場合は、別の代替策を検討してください。

通常、ページ・フォルト時にオペレーティング・システムにおいて待機するだけのために 5% を超える時間が費やされるべきではありません。待機時間が 5% を超える場合は、ページング・サブシステムが I/O バウンドであることを示しています。オペレーティング・システムのモニターを使用して待機時間を調べます。待機時間と実行時間の合計は 100% になります。システム負荷の合計が CPU の 100% 近くを占めている場合、システムは待機に多くの時間を費やしていません。待機している場合は、ページングが原因ではありません。

ページング・デバイスへの待機時間が 5% を超える場合は、次のいずれかの方法でメモリー所要量を低減する必要があります。

- プロセスの各クラスで必要なメモリーを低減する。
- メモリー負荷集中クラスでのプロセス数を低減する。
- メモリーを追加する。

待機時間がページング・サブシステムの I/O ボトルネックを示す場合は、ストライプ化によって解決できます。

## メモリー負荷集中プロセス数の削減

この項では、メモリー負荷集中プロセス数を削減するために実行できる 2 つの方法について説明します。

- [並列度の調整](#)
- [パラレル・ジョブのスケジューリング](#)

**並列度の調整** . パラレルで実行する操作の数を調整するだけでなく、操作を実行する DOP (並列度) も調整できます。このためには、PARALLEL 句を指定した ALTER TABLE 文を発行するか、ヒントを使用します。

**関連項目：** パラレル実行の詳細は、[第 26 章「パラレル実行のチューニング」](#)を参照してください。ALTER TABLE 文の詳細は、『Oracle8i SQL リファレンス』を参照してください。

PARALLEL\_MAX\_SERVERS の値を減らすことによって、パラレル・プールを制限できます。これによって、並列度の合計にシステムレベルの制限が設定されます。また、システムの管理も容易になります。さらに多くのプロセスが強制的にシリアル・モードで実行されます。

PARALLEL\_ADAPTIVE\_MULTI\_USER パラメータを TRUE に設定してパラレル・マルチユーザー問合せ調整機能を使用可能にすると、Oracle はユーザー負荷に基づいて DOP の調整を制御します。

**関連項目：** パラレル・マルチユーザー問合せ調整機能の詳細は、26-7 ページの「[並列度とマルチユーザー問合せ調整、およびその相互作用](#)」を参照してください。

**パラレル・ジョブのスケジューリング** ジョブのキューイングは、並列度を減らさずにプロセス数を減らすもう 1 つの方法です。すべての操作の並列度を削減するのではなく、大規模なパラレル・バッチ・ジョブを同時実行しないで、一度に 1 つずつ完全な並列度で実行するようにスケジューリングできます。キューの先頭にある問合せの応答時間は速くなりますが、キューの最後の問合せの応答時間は遅くなります。ただし、この方法では一定の管理オーバーヘッドがかかります。

## プロセスあたりのデータ・ウェアハウス・メモリーの低減

次の説明は、HASH\_AREA\_SIZE とメモリーの関係に関するものですが、SORT\_AREA\_SIZE にもまったく同じ考慮事項が適用されます。ただし、SORT\_AREA\_SIZE の下限は、HASH\_AREA\_SIZE の推奨最小値 8MB ほど重要ではありません。

すべての操作がハッシュ結合とソートを実行する場合、メモリー所要量が大きくなるため、ユーザーが持つことのできるプロセス数が制限されます。さらに多くのユーザーが同時に実行できるようにするには、データ・ウェアハウスのプロセス・メモリーの削減が必要な場合があります。

**大容量メモリーから中容量メモリーへのプロセスの移動** HASH\_AREA\_SIZE の値を低くすることで、プロセスを大容量メモリー・クラスから中容量メモリー・クラスへ移すことができます。メモリー容量が同じ場合、Oracle は常にソート・マージ結合よりも高速にハッシュ結合を処理します。したがって、オラクル社では、ハッシュ領域をソート領域よりも小さくすることはお勧めしません。

**大容量または中容量メモリーから小容量メモリーへのプロセスの移動** 数千人のユーザーをサポートする必要がある場合は、操作によって不要なデータがアクセスされないようなアクセス・パスを作成します。そのためには、次の 1 つ以上を実行します。

- 索引またはサマリー表（またはその両方）を作成することで、索引結合の要求を低減する。
- サマリー表を作成し、ユーザーおよびアプリケーションに詳細データではなくサマリーとマテリアライズド・ビューを参照させることによって、GROUP BY ソートへの要求を低減する。
- 頻繁にソートされる列に索引を作成することによって、ORDER BY ソートへの要求を低減する。

**関連項目：** サマリー表の詳細は、[VI 項の「マテリアライズド・ビュー」](#)を参照してください。

## マルチユーザーのための並列性の低減

マルチユーザーのための並列性を低減する最も簡単な方法は、26-7 ページの「[並列度とマルチユーザー問合せ調整、およびその相互作用](#)」の説明に従って、パラレル・マルチユーザー問合せ調整機能を使用可能にすることです。

ただし、これを手動で制御することにした場合、単一ユーザーでの高速の応答時間とマルチ・ユーザーのための効果的なリソース使用の間にはトレードオフがあります。たとえば、システムのメモリーが 2GB で、HASH\_AREA\_SIZE が 32MB の場合は、およそ 60 のパラレル実行サーバーをサポートできます。10 の CPU を持つマシンは、最大 3 つの同時パラレル操作をサポートできます ( $2 * 10 * 3 = 60$ )。12 の同時パラレル操作をサポートするためには、デフォルトの並列性の上書き（低減）、HASH\_AREA\_SIZE の低減、追加メモリーの購入のいずれかを行います。または、この 3 つの方法を組み合わせ使用します。つまり、すべてのパラレル表 *t* に対して ALTER TABLE *t* PARALLEL (DOP = 5) を使用し、HASH\_AREA\_SIZE を 16MB に設定し、PARALLEL\_MAX\_SERVERS を 120 に増やします。係数 2 を使用して各パラレル・サーバーのメモリーを削減し、係数 2 を使用して単一の操作の並列度を削減することによって、システムはさらに  $2 * 2 = 4$  倍の同時パラレル操作を受け入れることができます。

このようなアプローチの欠点は、単一の操作が実行されると、システムが 10CPU マシンの CPU リソースの半分だけを使用することです。もう半分は別の操作が開始するまでアイドル状態になります。

システムが十分に利用されているかどうかを判断するには、ほとんどのオペレーティング・システムで使用可能なグラフィカル・システム・モニターの 1 つを使用します。これらのモ

ニターを使うと、操作の実行時間を監視するよりも容易に CPU の使用率とシステムのパフォーマンスを調べることができます。オペレーティング・システムのマニュアルを参照して、システムがグラフィカル・システム・モニターをサポートしているかどうかを確認してください。

## 例：メモリー、ユーザーおよびパラレル実行サーバーのバランス化

この項の例では、メモリー、ユーザーおよびパラレル実行サーバー間の関係を評価して、[図 27-1](#) で示す計算式をバランス化する方法を説明します。システムの作業負荷を調整して、必要な数のプロセスおよびユーザーを受け入れる方法を具体的に示します。

### 例 1

システムに 1GB のメモリーがあり、DOP が 10 で、3 つ以上の表のハッシュ結合をユーザーが 2 つ実行すると想定します。SGA として 300MB が必要とすると、処理を行うには 700MB が残ります。32MB などの豊富なハッシュ領域サイズを設定すると、システムは次のものをサポートできます。

図 27-3 メモリー、ユーザーおよびプロセスのバランス化のための計算式

*1 parallel operation* ( $32\text{MB} * 10 * 2 = 640\text{MB}$ )

*1 serial operation* ( $32\text{MB} * 2 = 64\text{MB}$ )

上記の合計は、704MB です。この場合、メモリーは大幅にはオーバーサブスクライブされません。

パラレル、ハッシュ、ソート・マージ結合操作はすべて、DOP の 2 倍の数のパラレル実行サーバーを使用すること（2 つのサーバー・セットを利用）、またパラレル操作の個々のプロセスはそれぞれ大量のメモリーを使用する場合が多いことを覚えておいてください。このため、プロセスをシリアルで実行するか、より低い並列性を使用してプロセスを実行すると、はるかに多くのユーザーをサポートできます。

さらに多くのユーザーにサービスを提供するために、ハッシュ領域サイズは 2MB まで削減できます。この構成によって、17 のパラレル操作または 170 のシリアル操作をサポートできます。ただし、応答時間はハッシュ結合を使う場合よりもかなり長くなることがあります。

この例のトレードオフによって、プロセスあたりのメモリーを係数 16 で削減すると、同時ユーザーの数を係数 16 で増やせることが明らかになります。このため、マシンの物理メモリーの容量によって、ハッシュ結合やソートを行うために実行できるパラレル操作の合計数に別の制限が課されます。

## 例 2

作業負荷が混在している例では、表 27-2 に示すように、さまざまなニーズを持つユーザー集団を考えてください。この状況では、リソースを選択的に割り当てる必要があります。ハッシュ結合がソート・マージ結合よりもパフォーマンスがよいとしても、全ユーザーがハッシュ結合を実行することは許可できません。このレベルの作業負荷をサポートするだけのメモリーがないからです。

日中はバッチ・ジョブが頻繁には実行されないので、50% をオーバーサブスクライブしておけば安全であると考えられます ( $700\text{MB} * 1.5 = 1.05\text{GB}$ )。これによって、作業負荷の合計に対して十分な仮想メモリーが提供されます。

表 27-2 混在する作業負荷への適応方法

ユーザーのニーズ	適応方法
DBA: 夜間にバッチ・ジョブを実行し、日中もバッチ・ジョブを実行することがあります。これらのジョブは、ハッシュ結合を実行し、したがって大容量のメモリーを使用するパラレル操作である可能性があります。	20 のパラレル実行サーバーを使用し、大容量メモリー・クラスの単一の高性能バッチ・ジョブのために、HASH_AREA_SIZE を中位の値、たとえば 20MB に設定します。このバッチ・ジョブは、データのサマリーを生成する結合を伴う大規模な GROUP BY 操作である場合があります。サーバー数 20 に 20MB をかけると、メモリー 400MB に相当します。
分析者: スプレッドシート用のデータを抽出する対話型のユーザー。	大容量データにアクセスする複雑なハッシュ結合を使うシリアル操作を 10 名の分析者が実行するという計画を立てるとします。メモリー所要量により、パラレル操作の実行は許可できません。1 つあたり 40MB のそのようなシリアル・プロセスが 10 あると、メモリー 400MB に相当します。
ユーザー: 個々の顧客アカウントの単純な検索を実行し、すでに結合され、部分的にサマリーされたデータについてレポートを作成する数百名のユーザー。	それぞれ約 0.5MB の小容量メモリー・プロセスを実行する数百のユーザーをサポートするには、200MB を確保しておきます。

## 例 3

使用しているシステムに 2GB のメモリーがあり、200 の問合せサーバー・プロセスと、ハッシュ結合を伴う高負荷のデータ・ウェアハウス操作を実行する 100 名のユーザーがいることを想定します。索引検索や小規模のソートなどのタスクは考慮しないことにします。かわりに、大容量メモリー・プロセスに集中します。300 のプロセスがあり、そのうち 200 はパラレル・プールに由来する必要があり、あとの 100 は単スレッドです。合計 2GB のメモリーの 4 分の 1 は SGA によって使用され、残りの 1.5GB のメモリーですべてのプロセスを処理します。20% のオーバーサブスクライブの係数も含め、大容量のメモリー所要量のみを考慮して計算式を適用します。

図 27-4 メモリー / ユーザー / サーバーの関係の計算式 : 大容量メモリー・プロセス

$$high\_memory\_req'd = \frac{total\_memory}{\#\_high-memory\_processes} * 1.2 = \frac{1.5GB * 1.2}{300} = \frac{1.8GB}{300}$$

このとき、6MB = 1.8GB/300 となります。6MB 未満のハッシュ領域が各プロセスで使用可能になりますが、8MB が推奨最小値です。300 のプロセスが必要な場合は、それらを大容量メモリー・負荷集中クラスから中容量負荷メモリー・集中クラスに変更するためにハッシュ領域サイズを削減することが必要な場合があります。そうすると、それらはシステムの制約に適合します。

#### 例 4

2GB のメモリーを持つシステムで、10 人のユーザーが高負荷のデータ・ウェアハウス・パラレル操作を同時に実行しようとし、なおも良好なパフォーマンスを期待する場合を想定します。10 の DOP を選択すると、10 人のユーザーでは 200 プロセスが必要になります（大規模な結合を実行するプロセスでは、DOP の 2 倍のパラレル実行サーバーが必要になります。そのため PARALLEL\_MAX\_SERVERS は 10 \* 10 \* 2 に設定します）。この例では、各プロセスが 1.8GB/200、すなわち約 9MB のハッシュ領域を獲得します。通常このハッシュ領域の量で十分です。

大規模なハッシュ結合を行うユーザーが 5 名だけの場合は、各プロセスは 16MB を超えるハッシュ領域を獲得します。これは問題ありません。しかし、大容量のハッシュ結合のために 32MB を使用可能にしたい場合、システムは 2 ~ 3 ユーザーしかサポートできません。反対に、ユーザーが集計を計算するだけの場合は、システムで必要なのは十分なサイズのソート領域であり、より多数のユーザーをサポートできます。

#### 例 5

2GB のメモリーを持つシステムで、1000 ユーザーをサポートする必要があり、すべてのユーザーが大容量の問合せを実行しなければならない場合は、状況を注意深く評価する必要があります。このとき、ユーザーあたりの使用予定メモリーはわずか 1.8MB です（つまり、1.8GB を 1,000 で除算）。この数字は、中容量メモリー・クラスの最小値なので、さらに多くのリソースを使うパラレル操作を除外する必要があります。また、大規模なハッシュ結合も除外する必要があります。各シリアル・プロセスには、ソート領域に加えて最大 2 つのハッシュ領域が必要です。そのため、HASH\_AREA\_SIZE には SORT\_AREA\_SIZE と同じ値を設定する必要があります。これは 600KB (1.8MB/3) になります。このように小さなハッシュ領域サイズでは、効果がなくなる可能性が高くなります。

組織のリソースとビジネス・ニーズが与えられる場合、システムのメモリーをアップグレードするのは適切でしょうか？メモリーのアップグレードが選択肢にない場合は、予想を変更しなければなりません。バランスを調整するには、以下のようにします。



- システムが実際にサポートするのは大規模なハッシュ結合を実行する限られた数のユーザーであるという事実を受け入れます。
- データベース全体でなくサマリー表へのアクセスをユーザーに与えます。
- ユーザーをさまざまなグループに分類し、グループごとに与えるメモリの容量を変えます。全ユーザーが小さなソート領域を使ってソートを実行するかわりに、ごく一部のユーザーが大容量メモリのハッシュ結合を行い、大部分のユーザーはサマリー表を使い、小容量メモリの索引結合を実行します（これを達成するには、各グループのユーザーに、ヒントを問合せ内で強制的に使わせて、特定の方法で操作を実行するようにします）。

## パラレル実行の領域管理問題

この項では、パラレル実行を使用するときに発生する領域管理問題について説明します。これらの問題は次のとおりです。

- ソートと一時データのための ST（領域トランザクション）エンキュー
- 外部断片化

これらの問題は、OPS（Oracle Parallel Server）環境のパラレル操作で特に重要になります。ノード数が増加するにつれて、チューニングがますます重要になります。

ローカルに管理される表領域をインプリメントできる場合は、これらの問題をすべて回避できます。

---

**注意：** ローカルに管理される表領域の詳細は、『Oracle8i 管理者ガイド』を参照してください。

---

### ソートと一時データのための ST（領域トランザクション）エンキュー

データベースのすべての領域管理トランザクション（PARALLEL CREATE TABLE での一時セグメントの作成や、非パーティション表のパラレル・ダイレクト・ロード挿入など）は、単一の ST エンキューによって制御されています。ST エンキューでのトランザクション率が高い（たとえば 1 分間に 2 または 3 トランザクション以上）と、多数のノードがある OPS で拡張性が乏しくなったり、領域管理リソースを待機してタイムアウトになったりすることがあります。VSROWCACHE ビューと VSLIBRARYCACHE ビューを使用してこのタイプの競合を突き止めます。

特に以下の領域管理トランザクションの回数を最小化するようにしてください。

- ソート領域管理トランザクションの回数。
- オブジェクトの作成と削除。
- 表領域内の断片化によって発生するトランザクション。

専用の一時表領域を使用して、ソートの領域管理を最適化します。これは、OPS では特に効果があります。V\$SORT\_SEGMENT を使用するとこれを監視できます。

INITIAL および NEXT のエクステント・サイズを 1MB ~ 10MB の値に設定します。プロセスは、1 秒間に最大 1MB の割合で一時領域を使用します。NEXT エクステント・サイズとしてデフォルト値の 40KB を使用すると、1 秒あたりの領域の要求が多くなるので、この値は受け入れないでください。

### 外部断片化

外部断片化は、パラレル・ロードおよびダイレクト・ロード挿入、PARALLEL CREATE TABLE ... AS SELECT の問題です。エクステントが割り当てられ、データの挿入および削除が行われるにつれて、メモリーは断片化する傾向があります。これにより、使用不可能な空き領域が大量に発生します。この領域は、連続していない小さなメモリーの断片で構成されるため使用できません。

パーティション表の外部断片化を削減するために、すべてのエクステントを同じサイズに設定してください。NEXT の値は INITIAL と同じ値に設定し、PCTINCREASE はゼロに設定します。システムが効率的に処理できるのは、オブジェクトあたり数千のエクステントです。したがって、MAXEXTENTS はたとえば 1,000 ~ 3,000 に設定します。MAXEXTENTS には 10,000 を超える値を使用しないでください。パーティション化されていない表に対して、パラレル・ダイレクト・ロード等を行う場合には初期エクステントを小さくする必要があります。

## Oracle Parallel Server でのパラレル実行のチューニング

この項では、OPS でのパラレル実行のいくつかの側面について説明します。

### ロック割当て

この項では、OPS で最適なロック管理を行うためのパラレル実行のチューニングのガイドラインを説明します。

OPS でのパラレル実行を最適化するには、GC\_FILES\_TO\_LOCKS を正しく設定する必要があります。OPS では、特定の数のパラレル・キャッシュ管理 (PCM) ロックが各データ・ファイルに割り当てられています。データ・ブロック・アドレス (DBA) ロッキングのデフォルトの動作では、各ブロックに 1 つのロックが割り当てられます。全表走査の間は、走査で読み込まれる各ブロックに対して 1PCM ロックが獲得されなければなりません。全表走査の速度を上げるためには、以下の 3 つの可能性があります。

- 完全な読み込み専用データを含むデータ・ファイルでは、表領域を読み込み専用を設定します。すると、PCM ロックがかからなくなります。
- または、ほとんど読み込み専用のデータには、ごく少数のハッシュした PCM ロック (たとえば、2 つの共用ロック) を各データ・ファイルに割り当てます。すると、これらは、データを読み込むときに獲得しなければならない唯一のロックになります。

- DBA ロックまたはファイン・グレイン・ロックが必要な場合は、!オプションを使用して、各ロックで制御されるブロックを1つのグループにまとめます。これは、デフォルトの DBA ロックでは有利です。デフォルトでは、100 万ブロックを読み込むためにはロックを 100 万回獲得する必要があるためです。ブロックをグループ分けすると、グループ化係数によって割り当てられるロックの回数が削減されます。このように、!10 によるグループ化は、獲得しなければならない PCM ロック数がデフォルトの 10 分の 1 になることを意味します。ロック割当て数の急激な低下によってパフォーマンスが向上します。大まかな指針としては、!10 でグループ化したときのパフォーマンスは、ハッシュ・ロッキングの速さに匹敵します。

パラレル DML 操作を高速化するには、データベース・アドレス・ロッキングではなくハッシュ・ロッキングまたは高いグループ化係数の使用を検討してください。パラレル実行サーバーは、非オーバーラップ・パーティションで動作します。パーティションがファイルを共有しないようにすることをお勧めします。それによって、1 ファイルにつきハッシュ・ロックはただ1つとすることで、ロック操作の数を削減できます。パラレル実行サーバーが動作するファイルに重複はないので、ロックの ping が発生しません。

次のガイドラインは、メモリーの使用方法に影響を与えます。そのため、間接的にパフォーマンスも影響を受けます。

- 一時表領域のデータ・ファイルには決して PCM ロックを割り当てないでください。
- ロールバック・セグメントしか含まないデータファイルには決して PCM ロックを割り当てないでください。これらは GC\_ROLLBACK\_LOCKS および GC\_ROLLBACK\_SEGMENTS によってプロテクトされています。
- SYSTEM 表領域には特定の PCM ロックを割り当てます。この方法では、領域管理などのデータ・ディクショナリ・アクティビティがキャッシュ管理レベルでデータ表領域に干渉（エラー 1575）しないことが保証されます。

たとえば、読み込み専用データベースで、データ・ウェアハウス・アプリケーションの問合せ専用作業負荷がある場合、ファイル 1 の SYSTEM 表領域に 500PCM ロックを作成し、さらに 50 のロックを作成し、その他のファイルのすべてのデータで共有します。そうすると、領域管理作業は、データベースの他の部分に決して干渉しません。

**関連項目：** PCM ロックとロッキング・パラメータの詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

## 複数の同時パラレル操作のロード・バランシング

ロード・バランシングは、問合せサーバー・プロセスを配分して、ノード間の CPU およびメモリーの使用を均一化します。また、ノード間の通信とリモート I/O を最小化します。Oracle は、実行しているプロセスが最も少ないノードにサーバーを割り当てることによって、ロード・バランシングを行います。

ロード・バランシング・アルゴリズムは、すべてのノードの負荷を均一にしようとします。たとえば、ノードあたり 1 つの CPU を持つ 8 ノードの MPP（超並列処理）システムで 8 の DOP が要求された場合、アルゴリズムは各ノードに 2 つのサーバーを配置します。

問合せサーバー・グループ全体が1つのノードに収まる場合、ロード・バランシング・アルゴリズムは、単一のノード上にすべてのプロセスを配置して、通信オーバーヘッドを回避します。たとえば、ノードあたり16のCPUを持つ2ノードのクラスターで8のDOPが要求された場合、アルゴリズムは1つのノードに16の問合せサーバー・プロセスすべてを配置します。

### パラレル・インスタンス・グループの使用方法

ユーザーまたはDBAは、"Instance Group"機能を使用して、どのインスタンスが問合せサーバー・プロセスを割り当てるかを制御できます。この機能を使用するには、最初に各アクティブ・インスタンスを少なくとも1つ以上のインスタンス・グループに割り当てます。その後で、特定のインスタンス・グループをアクティブ化することによって、どのインスタンスがパラレル・プロセスを生成するかを動的に制御できます。

インスタンス・グループのメンバーシップは、初期化パラメータ `INSTANCE_GROUPS` に1つ以上のインスタンス・グループを表す名前を設定することによって、インスタンスごとに設定します。たとえば、マーケティング組織と販売組織の両方によって所有されている32ノードのMPPシステムでは、インスタンス・グループ名を使用して、ノードの半分を一方の組織に割り当て、もう半分を他方の組織に割り当てます。これを行うには、各 `INIT.ORA` ファイルで次のパラメータ構文を使用して、ノード1～16をマーケティング組織に割り当てます。

```
INSTANCE_GROUPS=marketing
```

次に、残りの `INIT.ORA` ファイルで次の構文を使用して、ノード17～32を販売組織に割り当てます。

```
INSTANCE_GROUPS=sales
```

さらに、ユーザーまたはDBAは、次のように入力することによって、販売組織が所有しているノードをアクティブ化して問合せサーバー・プロセスを生成できます。

```
ALTER SESSION SET PARALLEL_INSTANCE_GROUP = 'sales';
```

Oracleは、応答として問合せサーバー・プロセスをノード17～32に割り当てます。`PARALLEL_INSTANCE_GROUP`のデフォルト値は、すべてのアクティブ・インスタンスです。

---

---

**注意：** 前述したように、インスタンスは1つ以上のグループに属することができます。カンマをセパレーターとして使用して、`INSTANCE_GROUP`パラメータで複数のインスタンス・グループ名を入力できます。

---

---

### ディスク親和性

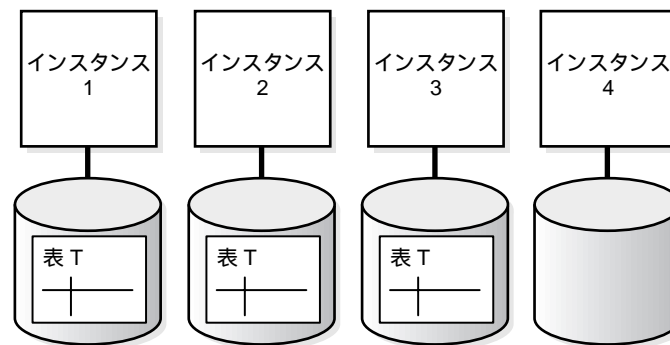
一部のOPSプラットフォームでは、ディスク親和性が使用されます。ディスク親和性を使用しない場合、Oracleは割当てを均等にインスタンスに配置しようとします。ディスク親和

性を使用すると、Oracle は、パラレル表走査のためのパラレル実行サーバーを、要求されたデータに最も近いインスタンスに割り当てようとしています。ディスク親和性は、シェアード・ナッシング・アーキテクチャ上でデータの移動とノード間通信を最小化します。このように、ディスク親和性によって、パラレル操作のスループットおよび応答時間を大幅に改善できます。

ディスク親和性は、パラレル表走査、パラレル一時表領域割当て、パラレル DML およびパラレル索引走査で使用されます。これは、パラレル表作成またはパラレル索引作成では使用されません。一時表領域へのアクセスでは、優先的にローカル・データファイルが使用されます。これによって、最適な領域管理のエクステンツ割当てが保証されます。オペレーティング・システムによってストライプ化されたディスクは、ディスク親和性によって 1 つの単位として扱われます。

以下のディスク親和性の例では、表 T が 3 つのノードに分配されており、表 T の全表走査が実行されています。

図 27-5 ディスク親和性の例



- 1 つの問合せで 2 つのインスタンスが必要な場合、セット 1 およびセット 2、セット 3 から 2 つのインスタンスが使われます。
- 1 つの問合せで 3 つのインスタンスが必要な場合、インスタンス 1 およびインスタンス 2、インスタンス 3 が使われます。
- 1 つの問合せで 4 つのインスタンスが必要な場合、4 つのインスタンスすべてが使われます。
- 表 T に対して 2 つの同時操作がある場合に、それぞれが 3 つのインスタンスを要求すると（両方の操作に対するインスタンスには十分なプロセスがあります）、両方の操作でインスタンス 1、インスタンス 2 およびインスタンス 3 が使用されます。インスタンス 4 は使用されません。反対に、ディスク親和性がない場合はインスタンス 4 が使用されます。

**関連項目：** インスタンスとディスクの親和性の詳細は、『Oracle8i Parallel Server 概要および管理』を参照してください。

## パラレル実行のチューニングのヒント

この項では、パラレル操作のパフォーマンス向上手法について説明します。

- [デフォルトの並列度の上書き](#)
- [SQL 文の書直し](#)
- [パラレルでの表の作成と移入](#)
- [パラレルでの索引の作成](#)
- [問題の診断](#)
- [パラレルでの表のリフレッシュ](#)
- [コストベースの最適化でのヒントの使用](#)

### デフォルトの並列度の上書き

デフォルトの DOP は、任意のパラレル操作に対して CPU および I/O リソースの使用を保証する一方で、応答時間を短縮するのに適しています。操作が I/O バウンドの場合は、デフォルトの DOP を増やすことを検討してください。操作がメモリー・バウンドの場合、または複数の同時パラレル操作が実行中の場合は、デフォルトの DOP を減らすことが必要な場合があります。

デフォルトの DOP は、データ・ディクショナリ内の PARALLEL 属性を持つ表に対して、または PARALLEL ヒントが指定されているときに使用されます。表が PARALLEL 属性を持たない場合、または NOPARALLEL (デフォルト) 属性を持つ場合は、CPU、インスタンスおよびその表を格納するデバイスの数によって示されるデフォルトの DOP に関係なく、その表はパラレルには走査されません。

DOP を調整するときは、次のガイドラインに従います。

- デフォルトの DOP は、PARALLEL\_THREADS\_PER\_CPU パラメータの値を変更することで変更できます。
- DOP は、ALTER TABLE またはヒントを使用して調整できます。
- 同時パラレル操作の数を増やすには、DOP を減らすか、パラメータ PARALLEL\_ADAPTIVE\_MULTI\_USER を TRUE に設定します。
- I/O バウンドのパラレル操作では、最初に、CPU 数よりも多くのディスクにデータを分散させます。次に、並列度を段階的に増やします。問合せが CPU バウンドになったら中止します。

たとえば、CPU 数 =10 かつディスク数 =36 の場合に、パラレルの索引によるネスト・ループ・ジョインが、索引検索を実行するときに I/O バウンドになるとします。デフォルトの DOP は 10 であり、この場合は I/O バウンドになります。最初に、DOP を 12 に増やしてみます。それでもアプリケーションが I/O バウンドになる場合は、DOP を 24 にし、なおかつ I/O バウンドの場合は 36 にしてみます。

## SQL 文の書直し

パラレル実行で最も重要な問題は、相当の量のデータを処理する問合せ計画のすべての部分が並列に実行されるようにすることです。EXPLAIN PLAN を使用して、PARALLEL\_TO\_PARALLEL、PARALLEL\_TO\_SERIAL、PARALLEL\_COMBINED\_WITH\_PARENT または PARALLEL\_COMBINED\_WITH\_CHILD の、OTHER\_TAG がすべての計画ステップにあることを確認してください。他のキーワード（または NULL）は、シリアル実行を指示するため、ボトルネックとなる可能性があります。

次の変更を加えることで、パラレル計画の生成におけるオプティマイザの能力が向上します。

- 副問合せ（特に相関副問合せ）を結合に変換します。Oracle は、副問合せよりも効率よく結合を並列化できます。これは更新にも当てはまります。
- 相関副問合せのかわりに、主問合せの WHERE 句で PL/SQL 関数を使用します。
- 集計操作 distinct をネストした問合せとして書き直します。たとえば、次のようになります。

```
SELECT COUNT(DISTINCT C) FROM T;
```

これは、次のように書き直します。

```
SELECT COUNT(*)FROM (SELECT DISTINCT C FROM T);
```

**関連項目：** 27-24 ページの「[表の更新](#)」

## パラレルでの表の作成と移入

Oracle は、ユーザーおよびプロセスに結果をパラレルで戻せません。問合せから多数の行が戻される場合は、問合せの実行は高速に行われますが、ユーザー・プロセスでは行をシリアルでのみ受け取れます。多数の結果セットを取り出す問合せでパラレル実行のパフォーマンスを最適化するには、PARALLEL CREATE TABLE ... AS SELECT またはダイレクト・ロード挿入を使用して結果セットをデータベースに保存します。後で、ユーザーは結果セットをシリアルに参照できます。

---

**注意：** SELECT の並列性は CREATE 文に影響しません。ただし、CREATE がパラレルの場合は、オブティマイザは SELECT もパラレルで実行しようとします。

---

NOLOGGING オプションと組み合わせると、パラレルの CREATE TABLE ... AS SELECT は、非常に効率のよい中間表機能を提供します。

次に例を示します。

```
CREATE TABLE summary PARALLEL NOLOGGING
AS SELECT dim_1, dim_2 ..., SUM (meas_1) FROM facts
GROUP BY dim_1, dim_2;
```

これらの表は、パラレル挿入で増分的にロードすることもできます。次の手法を使用して、中間表を利用できます。

- 共通の副問合せは、1 回計算しておけば何度でも参照できます。これは、複合ビューを何度も参照するよりもはるかに効率的です。
- アプリケーション・レベルでのチェックポイント / 再始動を提供するために、複雑な問合せをより単純なステップに分解できます。たとえば、1TB のサイズのデータベースで複雑な複数表の結合を実行するには、数十時間を要します。この問合せ中にクラッシュが発生すると、最初からやり直さなければなりません。CREATE TABLE ... AS SELECT または PARALLEL INSERT AS SELECT、あるいはその両方を使用して、それぞれ数時間で実行できるより単純な一連の問合せとして書き直せます。システム障害が発生した場合は、最後に完了したステップから問合せを再起動できます。
- 直積演算を実体化します。これによって、スター・スキーマに対する問合せを並列して実行できます。また、これで結合列内の固有の値の数を増やすことによって、パラレル・ハッシュ結合の拡張性も向上します。

地域と部門の参照表に結合されている大規模な小売りデータ表を考えます。5 つの地域と 25 の部門があります。パラレル・ハッシュ・パーティション化を使用して大規模表が地域に結合されている場合、最大のスピードアップは 5 です。同様に、大規模表が部門に結合されている場合は、最大のスピードアップは 25 です。しかし、地域と部門の直積演算を含む一時表が大規模表に結合されている場合は、最大のスピードアップは 125 となります。

- 元の表から不要な行を除いた新しい表を作成し、元の表を削除することで、手動パラレル削除を効率よくインプリメントできます。あるいは、便利なパラレル削除機能を使うと、元の表から行を直接に削除することもできます。
- 効率のよいマルチディメンション・ドリルダウン分析用のサマリー表を作成します。たとえば、月および銘柄、地域、販売員別にグループ化した収益の合計をサマリー表に保存できます。



- 古い表を新しい表にコピーすることで表を再編成すると、連鎖行が排除され、空き領域が圧縮されます。これは、エクスポート / インポートよりもはるかに高速で、再ロードよりも容易です。

---

**注意：** 新しく作成した表には、必ず ANALYZE 文を使用するようにしてください。索引の作成も検討してください。I/O のボトルネックを避けるには、表領域に少なくとも CPU と同じ数のデバイスを指定します。領域割当てでの断片化を避けるために、表領域内のファイル数を CPU の数の倍数とする必要があります。

---

## パラレルでの索引の作成

索引を作成するときには、複数のプロセスを同時に実行できます。Oracle Server では、索引作成に必要な作業を複数のサーバー・プロセスに分割するので、1 つのサーバー・プロセスで順番に索引を作成する場合よりも速く索引を作成できます。

パラレル索引作成は、ORDER BY 句を使った表走査と非常によく似た働きをします。表はランダムにサンプリングされ、一連の索引キーによって索引が DOP と同じ数に均一に分割されます。問合せプロセスの最初のセットは、表を走査してキーと ROWID の対を抽出し、キーに基づいて、2 番目のセット内のプロセスにそれぞれの対を送ります。2 番目のセット内の各プロセスは、キーをソートして通常的方式で索引を作成します。すべての索引の部分が作成された後で、コーディネータは（順番に並んだ）各部分を単に連結して最終的な索引を形成します。

ローカル索引のパラレル作成には、1 つのサーバー・セットを使います。そのセットの各サーバー・プロセスは割り当てられている表のパーティションを走査して、索引パーティションを作成します。使用されるサーバー・プロセスの数は指定の DOP になるので、ローカル索引のパラレル作成の DOP を高くすることができます。

索引の作成中に REDO および UNDO ロギングが行われないようにすることを任意に指定できます。これによって、パフォーマンスは大幅に改善されますが、索引が一時的に回復不能になります。回復可能性は、新しい索引をバックアップした後で復旧します。索引の回復のために、索引の再作成を必要とするこの期間をアプリケーションが許容できる場合は、NOLOGGING オプションの使用を検討する必要があります。

CREATE INDEX 文で PARALLEL 句を使用するのが、索引の作成で DOP を指定する唯一の方法です。CREATE INDEX のパラレル句に DOP が指定されていない場合は、CPU の数が DOP として使用されます。パラレル句がない場合は、索引作成はシリアルで実行されます。

---

**注意：** 索引をパラレルで作成する場合は、STORAGE 句で、問合せサーバー・プロセスによって作成される各副索引の記憶領域が参照されます。したがって、INITIAL が 5MB、DOP が 12 で作成される索引は、各プロセスが 5MB のエクステントで開始されるので、索引作成時に 60MB 以上の記憶領域を使用します。問合せコーディネータ・プロセスによってソートされた副索引が連結されるときには、エクステントの一部が切り捨てられることがあり、結果として作成される索引が、要求した 60MB より小さくなる場合があります。

---

表に UNIQUE キーまたは PRIMARY KEY 制約を追加したり使用可能にしたりすると、必要な索引を自動的にパラレルで作成できなくなります。そのかわりに、CREATE INDEX 文と適切な PARALLEL 句を使用して、必要な列に索引を手動で作成し、制約を追加または使用可能にしてください。Oracle では、制約を追加または使用可能にするときに、既存の索引を使用します。

同一の表のすべての制約がすでに使用可能未検査状態になっている場合は、それらの複数の制約を同時にパラレルで使用可能にできます。次の例では、ALTER TABLE ... ENABLE CONSTRAINT 文によって、制約をパラレルでチェックする表走査が実行されます。

```
CREATE TABLE a (a1 NUMBER CONSTRAINT ach CHECK (a1 > 0) ENABLE NOVALIDATE)
PARALLEL;
INSERT INTO a values (1);
COMMIT;
ALTER TABLE a ENABLE CONSTRAINT ach;
```

**関連項目：** パラレル実行機能を使用するときのエクステントの割当て方法の詳細は、『Oracle8i 概要』を参照してください。また、CREATE INDEX 文の完全な構文は、『Oracle8i SQL リファレンス』を参照してください。

## パラレル DML のヒント

この項では、パラレル DML 機能の概要を示します。

- [INSERT](#)
- [ダイレクト・ロード INSERT](#)
- [INSERT、UPDATE および DELETE の並列化](#)

**関連項目：** パラレル DML と DOP の詳細は、『Oracle8i 概要』を参照してください。パラレル DML の親和性の説明は、『Oracle8i Parallel Server 概要および管理』を参照してください。

### INSERT

Oracle の INSERT 機能は、次のように要約できます。

表 27-3 INSERT 機能の要約

挿入のタイプ	パラレル	シリアル	NOLOGGING
標準	不可	可	不可
ダイレクト・ロード挿入 (APPEND)	可: 次のものがが必要です。 <ul style="list-style-type: none"><li>ALTER SESSION ENABLE PARALLEL DML</li><li>表の PARALLEL 属性または PARALLEL ヒント</li><li>APPEND ヒント (オプション)</li></ul>	可: 次のものがが必要です。 <ul style="list-style-type: none"><li>APPEND ヒント</li></ul>	可: 次のものがが必要です。 <ul style="list-style-type: none"><li>表またはパーティションの NOLOGGING 属性の設定</li></ul>

パラレル DML が使用可能にされていて、PARALLEL ヒントがあるか、データ・ディクショナリで表に PARALLEL 属性が設定されている場合は、制限が適用されない限り、挿入はパラレルおよび追加となります。PARALLEL ヒントと PARALLEL 属性のうちのいずれかがない場合は、挿入はシリアルで実行されます。

ダイレクト・ロード INSERT

APPEND (追加) モードはパラレル挿入を実行するときのデフォルトです。つまり、データは常に、表に割り当てられた新しいブロックに挿入されます。そのため、APPEND ヒントはオプションです。挿入操作を高速化するには APPEND モードを使用する必要があります。ただし、領域の使用を最適化する必要があるときには使用しないでください。NOAPPEND ヒントを使用することで APPEND モードを上書きできます。

APPEND ヒントはシリアル挿入とパラレル挿入の両方に適用されますが、このヒントを使用する場合はシリアル挿入でも高速になります。ただし、APPEND ではより多くの領域とロックのオーバーヘッドが必要です。

NOLOGGING を APPEND とともに使用すると、処理がさらに高速になります。NOLOGGING では、操作に対する REDO ログが生成されません。NOLOGGING はデフォルトではありません。パフォーマンスを最適化したい場合に使用してください。通常、NOLOGGING は、表またはパーティションを回復する必要がある場合には使用しないでください。回復が必要な場合は、必ず操作の直後にバックアップを取ってください。ALTER TABLE [NO]LOGGING 文を使用して適切な値を設定してください。

関連項目: 『Oracle8i 概要』

INSERT、UPDATE および DELETE の並列化

表またはパーティションがデータ・ディクショナリ内で PARALLEL 属性を指定されている場合、PARALLEL 属性設定は、問合せの並列性だけでなく INSERT、UPDATE および DELETE 文の並列性も決定します。表の PARALLEL ヒントを文で明示的に指定すると、データ・ディクショナリ内の PARALLEL 属性の効果が上書きされます。

NOPARALLEL ヒントを使用すると、データ・ディクショナリ内の表の PARALLEL 属性が上書きされます。一般に、ヒントは属性よりも優先されます。

DML 操作では、セッションが PARALLEL DML を使用可能なモードにある場合にのみ並列が考慮されます（このモードに入るには、ALTER SESSION ENABLE PARALLEL DML を使用します）。モードは、問合せの並列化または DML 文の問合せ部分の並列化には影響しません。

**関連項目：** パラレル INSERT、UPDATE および DELETE の詳細は、『Oracle8i 概要』を参照してください。

**INSERT ...SELECT の並列化** INSERT... SELECT 文では、SELECT キーワードの後のヒントに加えて、INSERT キーワードの後に PARALLEL ヒントを指定できます。INSERT キーワードの後の PARALLEL ヒントは挿入操作だけに、SELECT キーワードの後の PARALLEL ヒントは SELECT 操作だけに影響します。つまり、INSERT 操作と SELECT 操作の並列度は互いに独立しています。1 つの操作をパラレルで実行できない場合でも、他の操作をパラレルで実行できるかどうかには影響しません。

ユーザーがセッションに対してパラレル DML を明示的に使用可能にしている、かつ、対象となる表にデータ・ディクショナリのエントリで PARALLEL 属性が設定されている場合は、INSERT を並列化する機能によって既存の動作が変更されます。この場合に、選択操作が並列化されている既存の INSERT ... SELECT 文では、挿入操作も並列化されます。

複数の表を問合せる場合は、複数の SELECT PARALLEL ヒントと複数の PARALLEL 属性を指定できます。

### 例

ACME 社を吸収した後に雇用された新しい従業員を追加します。

```
INSERT /*+ PARALLEL(EMP) */ INTO EMP  
SELECT /*+ PARALLEL(ACME_EMP) */ *  
FROM ACME_EMP;
```

PARALLEL ヒントで暗黙指定されているので、この例では APPEND キーワードは不要です。

**UPDATE および DELETE の並列化** (UPDATE キーワードまたは DELETE キーワードの直後に指定された) PARALLEL ヒントは、基になる走査操作だけでなく、更新 / 削除操作にも適用されます。修正される表の定義で指定された PARALLEL 句で、更新 / 削除の並列性を指定することもできます。

セッションまたはトランザクションに対して PDML (パラレル・データ操作言語) を明示的に使用可能にしておくと、問合せ操作を並列化する UPDATE/DELETE 文によって、UPDATE/DELETE 操作も並列化されることがあります。文に副問合せまたは更新可能ビューが含まれている場合は、それらは別の PARALLEL ヒントまたは句を独自に持つことができますが、これらのパラレル指示は、更新または削除の並列化の決定には影響しません。

ん。これらの操作をパラレルで実行できない場合でも、UPDATE または DELETE の部分をパラレルで実行できるかどうかには影響しません。

パラレル UPDATE および DELETE は、パーティション表でのみ使用することができます。

### 例 1

Dallas のすべての事務員の給与を 10% 上げます。

```
UPDATE /*+ PARALLEL(EMP) */ EMP
SET SAL=SAL * 1.1
WHERE JOB='CLERK' AND
DEPTNO IN
(SELECT DEPTNO FROM DEPT WHERE LOCATION='DALLAS');
```

PARALLEL ヒントは、走査だけでなく更新操作にも適用されます。

### 例 2

作業がアウトソーシングされる会計部門のすべての従業員を解雇します。

```
DELETE /*+ PARALLEL(EMP) */ FROM EMP
WHERE DEPTNO IN
(SELECT DEPTNO FROM DEPT WHERE DNAME='ACCOUNTING');
```

ここでも、並列性は表 EMP の走査と更新操作に適用されます。

## パラレルでの表のリフレッシュ

更新可能な結合ビュー機能と組み合わせたパラレル DML は、データ・ウェアハウス・システムの表をリフレッシュする効率的なソリューションを提供します。表のリフレッシュとは、OLTP 本番システムから生成された差異データで表を更新することです。

次の例では、CUSTOMER(c\_key, c\_name, c\_addr) という名前の表をリフレッシュすると想定します。差異データには、新しい行またはデータ・ウェアハウスの最後のリフレッシュ以降に更新された行のいずれかが含まれます。この例では、更新されたデータは ASCII ファイルによって本番システムからデータ・ウェアハウスに送られます。これらのファイルは、リフレッシュ処理を開始する前に、DIFF\_CUSTOMER という名前の一時表にロードしなければなりません。このタスクを効率よく実行するために、パラレル・オプションとダイレクト・オプションの両方で SQL\*Loader を使用します。

DIFF\_CUSTOMER のロードが終わると、リフレッシュ処理を開始できます。リフレッシュ処理は次の 2 つのフェーズで実行されます。

- 表の更新。
- 新しい行のパラレルでの挿入。

## 表の更新

更新では、簡単な SQL インプリメンテーションに副問合せが使用されます。

```
UPDATE CUSTOMER
SET (C_NAME, C_ADDR) =
(SELECT C_NAME, C_ADDR
 FROM DIFF_CUSTOMER
 WHERE DIFF_CUSTOMER.C_KEY = CUSTOMER.C_KEY)
WHERE C_KEY IN (SELECT C_KEY FROM DIFF_CUSTOMER);
```

欠点として、上述の文の 2 つの副問合せはパフォーマンスに影響を及ぼす点があります。

または、更新可能な結合ビューを使ってこの問合せを再作成することもできます。そのためには、まず、主キー制約を DIFF\_CUSTOMER 表に追加して、変更した列がキー保存された表にマップされるようにしなければなりません。

```
CREATE UNIQUE INDEX DIFF_PKEY_IND ON DIFF_CUSTOMER(C_KEY)
PARALLEL NOLOGGING;
ALTER TABLE DIFF_CUSTOMER ADD PRIMARY KEY (C_KEY);
```

CUSTOMER 表を次の SQL 文で更新します。

```
UPDATE /*+ PARALLEL(CUST_JOINVIEW) */
(SELECT /*+ PARALLEL(CUSTOMER) PARALLEL(DIFF_CUSTOMER) */
CUSTOMER.C_NAME as C_NAME
CUSTOMER.C_ADDR as C_ADDR,
DIFF_CUSTOMER.C_NAME as C_NEWNAME,
DIFF_CUSTOMER.C_ADDR as C_NEWADDR
 FROM CUSTOMER, DIFF_CUSTOMER
 WHERE CUSTOMER.C_KEY = DIFF_CUSTOMER.C_KEY) CUST_JOINVIEW
SET C_NAME = C_NEWNAME, C_ADDR = C_NEWADDR;
```

結合ビュー CUST\_JOINVIEW を展開する実表走査は、パラレルで実行されます。また、CUSTOMER 表がパーティション化されている場合に限り、更新を並列化することによってパフォーマンスをさらに改善できます。

**関連項目：** 27-17 ページの「[SQL 文の書直し](#)」。また、キー保存された表については、『Oracle8i アプリケーション開発者ガイド 基礎編』を参照してください。

## パラレルでの表への新しい行の挿入

リフレッシュ処理のもう 1 つのフェーズは、DIFF\_CUSTOMER 表から CUSTOMER 表に新しい行を挿入する作業です。この方法では更新の場合と異なり、INSERT 文の副問合せは回避できません。

```
INSERT /*+PARALLEL(CUSTOMER)*/ INTO CUSTOMER
SELECT * FROM DIFF_CUSTOMER
```

---

```
WHERE DIFF_CUSTOMER.C_KEY NOT IN (SELECT /*+ HASH_AJ */ KEY FROM CUSTOMER);
```

ただし、この場合、HASH\_AJ ヒントによって副問合せが逆ハッシュ結合に変換されます。(初期化ファイルでパラメータ ALWAYS\_ANTI\_JOIN がハッシュに設定されている場合には、このヒントは不要です)。このようにすると、パラレル挿入を使用して上の文を非常に効率よく実行できます。表がパーティション化されていない場合でも、パラレル挿入を適用できます。

## コストベースの最適化でのヒントの使用

コストベースの最適化は、SQL 文のための最善の実行計画を検出する非常に洗練されたアプローチです。パラレル実行では、コストベースの最適化が自動的に使われます。

---

---

**注意：** ANALYZE を使用して、コストベースの最適化のための最新の統計を収集する必要があります。特に、並列して使用されている表は、常に分析する必要があります。DDL 操作および DML 操作を行った後は、ANALYZE を実行して統計を最新の状態に保つようにしてください。

---

---

ヒントは慎重に採用してください。ヒントを使用する場合は、それが必要で大幅なパフォーマンスの利点をもたらす場合にのみ、チューニングの最後のステップで採用してください。このような場合には、コストベースの最適化によって推奨される実行計画から開始して、パフォーマンスの見積りを定量化した後でヒントの効果のテストに進みます。ヒントは優先されることを忘れないでください。ヒントを使用しているときに基になっているデータが変更された場合は、ヒントを変更することが必要な場合があります。そうしないと、実行計画の効果が低下する可能性があります。

ルールベース最適化を手動でチューニングした既存のアプリケーションがない限り、常にコストベースの最適化を使用してください。ルールベース最適化を使用する必要がある場合は、SQL 文を書き直してアプリケーション・パフォーマンスを大幅に改善できます。

---

---

**注意：** 問合せの中に DOP が 2 以上 (デフォルトの DOP も含む) の表があると、OPTIMIZER\_MODE = RULE が指定されている場合、またはその問合せ自体に RULE ヒントがある場合でも、Oracle はその問合せにコストベースのオプティマイザを使用します。

---

---

**関連項目：** 26-22 ページの「[OPTIMIZER\\_PERCENT\\_PARALLEL](#)」このパラメータは、パラレル認識を制御します。

## 問題の診断

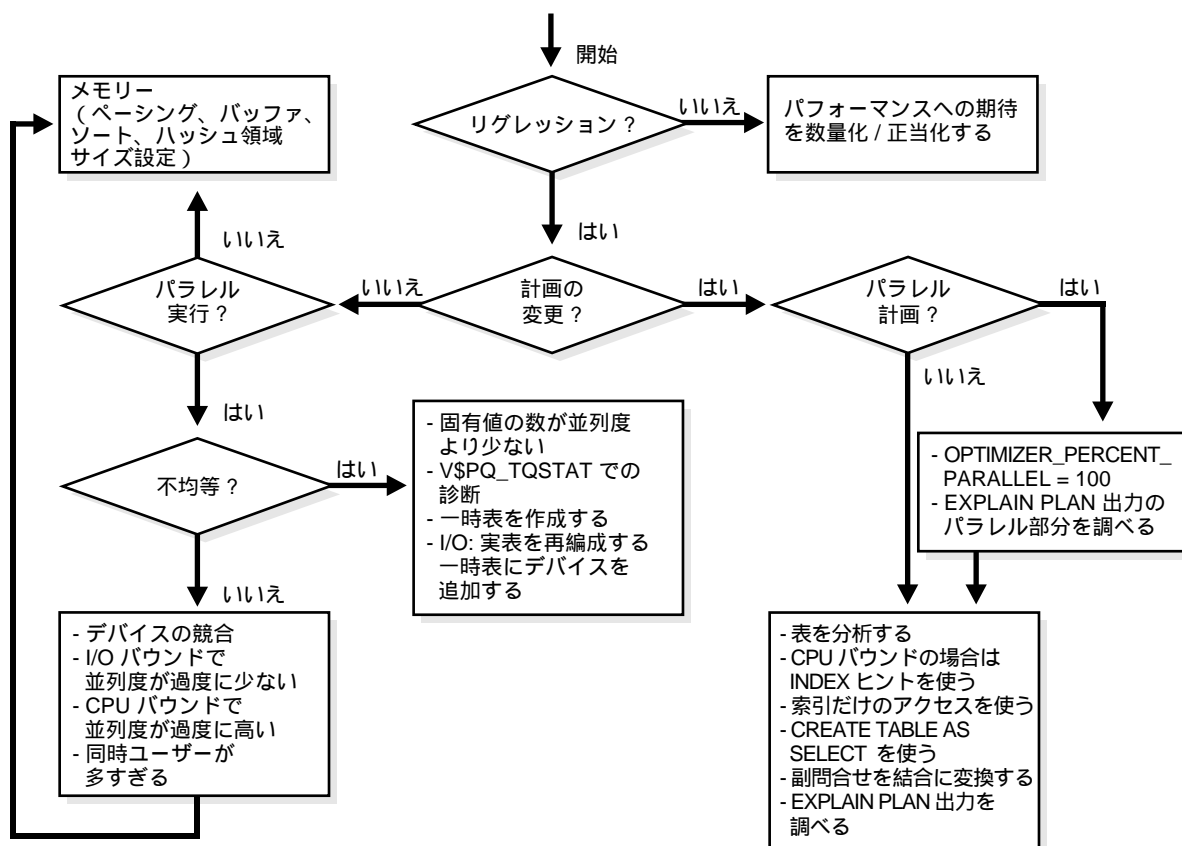
図 27-6 のディシジョン・ツリーを使用して、パラレルのパフォーマンス問題を診断します。  
図 27-6 のディシジョン・ポイントの問題は、図の後で詳細に説明されています。

パラレル実行のパフォーマンス問題の診断における重要事項の一部として、次のものがあります。

- パフォーマンスの期待値を数量化し、問題があるかどうかを判断します。
- 問題が最適化に関係するものか（表の再分析やヒントの追加が必要になる効率の悪い計画など）、実行に関係するものか（実行時間がマニュアルの基準を大幅に超えている走査、ロード、グループ化、索引作成といった単純な操作）を判断します。
- パラレルで実行しているときに問題が発生するのか（不均衡な負荷やリソース・ボトルネックなど）、シリアル操作でも問題が発生するのかを判断します。



図 27-6 パラレル実行のパフォーマンス・チェックリスト



## リグレッションの有無

パラレル実行の実際のパフォーマンスが期待外れではないですか。パフォーマンスが期待どおりの場合は、潜在的なパフォーマンス問題が存在していますか。おそらく、期待される結果があり、それと現在の結果が比較されているでしょう。パフォーマンス期待値は正当であるのに、システムがそれを達成していないのかもしれませんが。過去に、このレベルのパフォーマンスまたは特定の実行計画を達成したのに、現在では同様の環境と操作にもかかわらず、システムでこの目標を達成できない場合もあります。

パフォーマンスが期待外れの場合は、期待との誤差を数量化できますか。データ・ウェアハウス操作では、実行計画が重要です。クリティカルなデータ・ウェアハウス操作では、EXPLAIN PLAN の結果を保存してください。そして、データを分析し、再分析し、Oracle

をアップグレードし、さらに新規データをロードすると、やがて新しい実行計画と元の計画を比較できます。このアプローチは、予防的または反応的に実行します。

または、ヒントを使用すると、計画のパフォーマンスが改善される場合があります。ヒントが必要な理由を理解したり、ヒントを使用せずにオプティマイザで必要な計画を生成する方法を判断したい場合があります。統計サンプルのサイズを増やしてみてください。より詳細な統計をとることによって効果的な計画が作成できる場合があります。PARALLEL ヒントを使用する必要がある場合は、OPTIMIZER\_PERCENT\_PARALLEL を 100% に設定したかどうかを判断してください。

**関連項目：** EXPLAIN PLAN 文の詳細は、[第 13 章「EXPLAIN PLAN の使用方法」](#)を参照してください。プラン・スタビリティとアウトラインを使ったシステムの変更全体での計画の維持の詳細は、[第 7 章「オプティマイザ・モード、プラン・スタビリティおよびヒント」](#)を参照してください。

## 計画の変更

実行計画に変更があった場合は、計画がパラレルかシリアルのどちらか（または、どちらかである必要があるか）を判別します。

## パラレル計画

実行計画がパラレルである場合、またはパラレルでなければならない場合は、次のことを行います。

- パラレル計画が必要であるのに、オプティマイザがパラレル計画を生成しない場合は、OPTIMIZER\_PERCENT\_PARALLEL を 100 に増やしてみます。
- EXPLAIN PLAN の出力を調べます。すべての表を分析しましたか。おそらく、ヒントが必要な場合はまれです。ヒントによってパフォーマンスが改善するかどうかを検証してください。

**関連項目：** パラレル EXPLAIN PLAN のタグは、[表 13-2](#) で定義されています。

## シリアル計画

実行計画がシリアルである場合、またはシリアルでなければならない場合は、次の方法を検討してください。

- 索引を使用します。索引を追加するとパフォーマンスが大幅に向上することがあります。もう 1 つの列を索引に追加することを検討してください。そうすると、おそらく、操作がすべてのデータを索引から取得でき、表走査が不要になるでしょう。おそらく、ヒントが必要な場合はまれです。ヒントによって結果が改善するかどうかを検証してください。

- 分析を頻繁に行っておらず、時間に余裕がある場合は、統計を計算します。これは、多数の結合を実行する場合に特に重要であり、より効果的な計画が生成されます。または、統計を見積もることもできます。

---

**注意：** 異なるサンプル・サイズを使用すると、計画が変化することがあります。一般に、サンプル・サイズが大きいほど、効果的な計画が生成されます。

---

- 分散が一律でないデータの場合はヒストグラムを使用します。
- 初期化パラメータを調べて、値が適当かどうかを確認します。
- バインド変数をリテラルで置き換えます。
- 実行が I/O バウンドと CPU バウンドのどちらであるかを判断します。さらに、オペティマイザのコスト・モデルをチェックします。
- 副問合せを結合に変換します。
- CREATE TABLE ... AS SELECT を使って、複雑な操作を細かく分解します。5 個または 6 個の表を参照する大規模な問合せでは、問合せのどの部分に最も時間がかかっているのか判別するのが困難な場合があります。問合せをいくつかのステップに分けて、各ステップを分析することによって、ボトルネックを分離できます。

**関連項目：** CREATE TABLE ... AS SELECT については、『Oracle8i 概要』を参照してください。

## パラレル実行

リグレッションの原因を調べ、計画の問題ではないことが判明した場合、そのリグレッションは実行での問題ということになります。データ・ウェアハウス操作では、シリアルでもパラレルでも、計画がメモリーをどのように使用しているかを考慮してください。ページング率を調べて、システムができるかぎり効果的にメモリーを使っていることを確認します。バッファ、ソートおよびハッシュの領域のサイズ設定を調べます。問合せまたは DML 操作を実行した後で V\$SESSTAT ビュー、V\$PX\_SESSTAT ビューおよび V\$PQ\_SYSSTAT ビューを参照すると、使用されたサーバー・プロセスの数と、セッションおよびシステムに関するその他の情報を知ることができます。

**関連項目：** 26-73 ページの「[動的パフォーマンス・ビューを使ったパラレル実行のパフォーマンスの監視](#)」

## 作業負荷の均一な分散

パラレル実行を使用している場合に、作業負荷の分配が不均等ですか。たとえば、CPU が 10 あり、ユーザーが 1 の場合、作業負荷が CPU の間で均等に分配されているかどうかわかります。I/O が集中する期間に波があると、これは長期的には変化しますが、一般には CPU はそれぞれおよそ同じ量のアクティビティを持つ必要があります。

VSPQ\_TQSTAT の統計では、パラレル実行サーバーごとに生成および受取される行が表示されます。これは偏りの適切な指標であり、ユーザーの操作はまったく必要ありません。

オペレーティング・システムの統計では、プロセッサごとの CPU 使用率とディスクごとの I/O アクティビティが示されます。ただし、タスクを同時に実行すると、何が行われているかを見るのが難しくなります。シングルユーザー・モードで実行し、システム・レベルの CPU および I/O アクティビティを表示するオペレーティング・システム・モニターをチェックするのが有効です。

作業負荷の分配が不平衡な場合、通常はデータ内の偏りの存在が原因です。ハッシュ結合では、固有値の数が並列度より少ない場合がこれに当たります。4 つの固有値しかない列について 2 つの表を結合する場合、4 を超える拡張性はありません。CPU が 10 ある場合、そのうち 4 つは飽和状態ですが、6 つはアイドル状態になります。この問題を回避するには、問合せを変更します。つまり、一時表を使って結合順序を変更し、すべての操作が結合列に持つ値の数が CPU の数よりも大きくなるようにします。

I/O の問題が発生する場合は、データを再編成して、さらに多数のデバイスにデータを分散させる必要があるかもしれません。パラレル実行の問題が発生する場合は、推奨に従って、少なくとも CPU 数と同数のデバイスにデータが分散していることを確認してください。

作業負荷の分配に偏りがいない場合は、以下の条件を調べてください。

- デバイスの競合はないか。適切な I/O 帯域幅を提供できるだけのディスク制御装置があるか。
- 並列度が過度に少なく、システムが I/O バウンドになっていないか。その場合、並列度をデバイス数まで増やすことを考慮してください。
- 並列度が高すぎて、システムが CPU バウンドになっていないか。オペレーティング・システムの CPU モニターをチェックして、システム・コールに費やされている時間が多いかどうかを調べます。リソースがコミットされ過ぎている可能性があります。また、並列度が高すぎると、プロセスが互いに競合することになります。
- システムがサポートできる数よりも多く同時ユーザーがいらないか。

# 第 VI 部

---

## マテリアライズド・ビュー

第 VI 部では、マテリアライズド・ビューについて説明します。第 VI 部には次の章が含まれます。

- [第 28 章「マテリアライズド・ビューを使用したデータ・ウェアハウス」](#)
- [第 29 章「マテリアライズド・ビュー」](#)
- [第 30 章「ディメンション」](#)
- [第 31 章「クエリー・リライト」](#)
- [第 32 章「マテリアライズド・ビューの管理」](#)



---

## マテリアライズド・ビューを使用したデータ・ウェアハウス

この章のトピックは次のとおりです。

- [マテリアライズド・ビューを使用したデータ・ウェアハウスの概要](#)
- [マテリアライズド・ビュー](#)
- [データ・ウェアハウスのための Oracle Tools](#)
- [スタート・ガイド](#)

### マテリアライズド・ビューを使用したデータ・ウェアハウスの概要

企業データ・ウェアハウスには、組織に関する詳細な履歴データが含まれています。一般に、データは、1 つ以上のオンライン・トランザクション処理 (OLTP) データベースからデータ・ウェアハウスに毎月、毎週または毎日移動します。データは、データ・ウェアハウスに追加される前に、ステージング・ファイルで処理されるのが普通です。データ・ウェアハウスは、一般に数十ギガバイトから数テラバイトの範囲のサイズであり、通常、そのデータの大多数はいくつかの非常に大規模なファクト表に格納されています。

データ・マートには、特定の業務単位、部門またはユーザー・セットにとって価値のある企業データのサブセットが含まれます。一般に、データ・マートは企業データ・ウェアハウスから導出されます。

パフォーマンスを改善するためにデータ・ウェアハウスで採用されている手法の 1 つは、サマリー、すなわち集計の作成です。サマリーは、費用のかかる結合操作や集計操作を実行の前に事前計算し、その結果をデータベース内の表に格納することによって、問合せの実行時間を改善する特別な種類の集計ビューです。たとえば、地域別や製品別の売上合計を含む表を作成できます。

現在、サマリーを使用している組織では、手動でサマリーの作成、作成するサマリー選定、サマリーの索引作成、サマリーの更新およびユーザーへの使用するサマリー通知を行うため

に、大量の時間を費やしています。Oracle サーバーにサマリー管理が導入されたことにより、DBA の作業負荷は大幅に減少し、エンドユーザーはどのサマリーが定義されているかを意識する必要がなくなりました。DBA は、1 つ以上のマテリアライズド・ビューを作成します。マテリアライズド・ビューはサマリーと等価です。エンドユーザーがデータベースの表とビューに問合せを実行すると、Oracle サーバーのクエリー・リライト・メカニズムによって、サマリー表を使用するように SQL 問合せが自動的にリライトされます。この結果、問合せの結果が戻されるときに応答時間が大幅に改善され、エンドユーザーまたはデータベース・アプリケーションは、データ・ウェアハウス内に存在するサマリーを意識する必要がなくなります。

通常はクエリー・リライト・メカニズムを介して間接的にサマリーにアクセスしますが、エンドユーザーまたはデータベース・アプリケーションがサマリーに直接アクセスする問合せも作成できます。ただし、問合せでサマリーを直接参照すると、DBA はアプリケーションに影響を与えずにサマリーの削除や作成を自由に行うことができなくなるので、ユーザーにこの操作を認めるかどうかには慎重な検討が必要です。

このマニュアルおよびデータ・ウェアハウス関連の文書に記述されているサマリーまたは集計は、マテリアライズド・ビューというスキーマ・オブジェクトを使用して Oracle 内に作成されます。次に説明するように、マテリアライズド・ビューを使用して、問合せパフォーマンスの改善やレプリケート・データの提供などのいくつかのロールを実行できます。

## データ・ウェアハウスのマテリアライズド・ビュー

データ・ウェアハウスでは、マテリアライズド・ビューを使用して、売上合計などの集計データの事前計算と格納を行うことができます。このような環境でのマテリアライズド・ビューは、サマリー・データを格納するので、一般にサマリーと呼ばれています。マテリアライズド・ビューを使用して、集計を行う結合、または集計を行わない結合も事前計算できます。したがって、マテリアライズド・ビューを使用すると、大規模または重要な問合せクラスでの費用のかかる結合や集計に関連するオーバーヘッドを排除できます。

## 分散コンピューティングでのマテリアライズド・ビュー

分散環境でのマテリアライズド・ビューは、分散サイトにデータをレプリケートし、競合解決方式を使用して複数のサイトで行われた更新を同期するために使用されます。レプリカとしてのマテリアライズド・ビューによって、レプリカがなければリモート・サイトからアクセスしなければならないデータに、ローカルにアクセスできるようになります。

## モバイル・コンピューティングでのマテリアライズド・ビュー

マテリアライズド・ビューは、中央サーバーからモバイル・クライアントにデータのサブセットをダウンロードするために使用されます。データは中央サーバーから定期的にリフレッシュされ、クライアントによって行われた更新は中央サーバーに伝播されます。

この章では、データ・ウェアハウスでのマテリアライズド・ビューの使用に焦点を合せます。分散コンピューティングとモバイル・コンピューティングの詳細は、『Oracle8i レプリケーション・ガイド』と『Oracle8i 分散システム』を参照してください。



## サマリー管理のコンポーネント

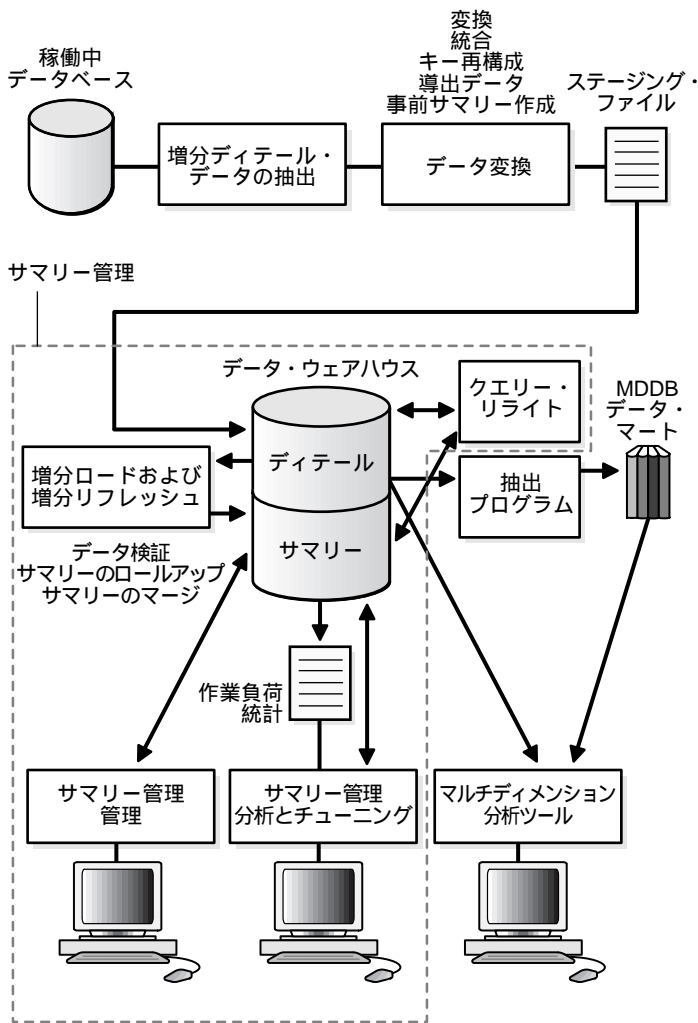
サマリー管理は、次のコンポーネントから構成されています。

- サマリーとディメンションを定義するメカニズム
- すべてのサマリーに最新のデータが含まれることを保証するリフレッシュ・メカニズム
- サマリーを使用するように問合せを透過的にリライトするクエリー・リライト機能
- 作成、保持および削除するサマリーを推奨するアドバイザ・ユーティリティ

多くの大規模な意思決定支援システム（DSS）データベースのスキーマは、従来のデータ・ウェアハウス・スキーマとはあまり似ていませんが、依然として結合と集計を必要とします。サマリー管理機能を使用すると、スキーマ制限は課されず、既存の DSS データベース・アプリケーションのいくつかは、データベースやアプリケーションの再設計をしなくてもパフォーマンスが大幅に改善されます。この機能は、すべてのデータベース・ユーザーが使用できます。

図 28-1 に、ウェアハウス・サイクルのどこでサマリー管理が使用されるかを示します。サマリー管理は、データが変換されてデータ・ウェアハウスにロードされた後で使用可能になります。したがって、図 28-1 で示されているように、データが変換され、ステージングされ、ウェアハウス内のディテール・データにロードされた後で、サマリー管理プロセスを起動できます。つまり、サマリーを作成し、問合せをリライトし、アドバイザを使用してサマリーの使用方法と作成を計画できるようになります。

図 28-1 サマリー管理の概要



データ・ウェアハウス設計の最も早い段階でサマリー管理プロセスを理解しておく、より高いパフォーマンス、より低い管理コストおよび記憶域必要量の削減という形の大きな配当を後で受け取ることができます。

サマリー管理プロセスは、業務の関係とデータベース内の共通アクセス・パターンを表すディメンションと階層の作成から開始します。ディメンションの分析を典型的な作業負荷の理解と組み合わせて、マテリアライズド・ビューの作成に利用することができます。マテリア

ライズド・ビューは、コストのかかる結合操作や集計操作を実行前に事前計算することによって、問合せ実行パフォーマンスを改善します。クエリー・リライトは、既存のマテリアライズド・ビューを使用して要求を満たすことができるとき、および要求を満たす必要のあるときを自動的に認識し、マテリアライズド・ビューを使用するように要求を透過的にリライトして、パフォーマンスを改善します。

ウェアハウスを作成するときのその他の考慮事項は、次のとおりです。

- 時間属性によってファクト表を水平パーティション化します。  
これによって、拡張性が増し、システム管理が単純化され、効率よく再作成できるローカル索引を定義できるようになります。
- SQL\*Loader に対して、表の単一パーティションをロードするように指示できます。  
この場合は、対応するローカル索引パーティションのみが再作成されます。
- グローバル索引は、ダイレクト・ロードの後で完全に再作成する必要があります。したがって、比較的少ない行数を大規模表にロードするときには非常にコストが高くなります。  
このため、ファクト表索引はすべてローカル索引として定義することを強くお勧めします。この定義は、たとえば各キー列に対してビットマップ索引を作成し（ビットマップ索引は常にローカルです）、索引の主要列としてパーティション化属性を持つすべてのキー列を単一の複数キー索引に含めることによって行うことができます。

## 用語

次に、データ・ウェアハウスのいくつかの基本用語を説明します。

- ディメンション表は、企業の業務エンティティを表します。通常、業務エンティティは、階層的に分類された時間、部門、場所、製品などの情報を表します。ディメンション表は、参照表と呼ばれることもあります。  
ディメンション表は、通常は時間の経過につれて徐々に変更されるものであり、定期的なスケジュールでは変更されません。ディメンション表は一般に大きなものではありませんが、ファクト表とディメンション表の結合、およびそれに続くディメンション階層の特定のレベルへの集計から構成される、長時間実行の意思決定支援問合せのパフォーマンスに影響します。
- ファクト表は、企業の業務トランザクションを表します。ファクト表は、ディテール表とも呼ばれます。  
データ・ウェアハウス内のデータの大多数は、いくつかの非常に大規模なファクト表に格納されています。これらは、稼働中の 1 つ以上のオンライン・トランザクション処理 (OLTP) データベースのデータによって定期的に更新されます。  
ファクト表には、売上、単位、在庫などの数量が含まれます。  
- 単純メジャーは、1 つの表の数値列または文字列です (FACT.SALES など)。

- 計算済メジャーは、1 つの表の単純メジャーのみを含む式です。たとえば、FACT.REVENUES - FACT.EXPENSES などです。
- 複数表メジャーは、複数の表で定義された計算済メジャーです。たとえば、FACT\_A.REVENUES - FACT\_B.EXPENSES などです。

ファクト表には、時間、製品、市場などの関連する業務エンティティ別に業務トランザクションを編成する 1 つ以上のキーも含まれます。たいていの場合、ファクト・キーは NULL 以外であり、ファクト表の一意複合キーを形成し、ディメンション表の 1 行のみと結合します。

- マテリアライズド・ビューは、ファクト表（場合によってはディメンション表）の集計データまたは結合データ、あるいはその両方のデータから構成される事前計算表です。データ・ウェアハウスの作成者は、マテリアライズド・ビューをサマリーまたは集計として認識します。

## マテリアライズド・ビュー

マテリアライズド・ビューが役立つ最も一般的なケースは、データ・ウェアハウス・アプリケーションと分散システムです。ウェアハウス・アプリケーションでは、大量のデータが処理され、同じような問合せが頻繁に繰り返されます。これらの問合せが事前計算され、結果がマテリアライズド・ビューとしてデータ・ウェアハウスに格納されている場合は、マテリアライズド・ビューを使用して結果セットを高速に検索することによって、パフォーマンスが大幅に向上します。

マテリアライズド・ビュー定義には、任意の数の集計と任意の数の結合を含めることができます。いくつかの点で、マテリアライズド・ビューは索引と同じように動作します。

- マテリアライズド・ビューの目的は、要求実行のパフォーマンスを向上させることです。
- マテリアライズド・ビューの存在は SQL アプリケーションに対して透過的なので、DBA は、SQL アプリケーションの有効性に影響を与えずに、いつでもマテリアライズド・ビューを作成または削除できます。
- マテリアライズド・ビューは、記憶領域を消費します。
- マテリアライズド・ビューの内容は、基礎となるディテール表が変更されたときにメンテナンスする必要があります。

この章では、マテリアライズド・ビューがデータ・ウェアハウス環境でどのように使用されるかを示します。ただし、サマリー管理のキー・コンポーネントであるマテリアライズド・ビューは、分散環境でレプリケート・データを管理するためにも使用することができます。詳細は、『Oracle8i レプリケーション・ガイド』を参照してください。

## マテリアライズド・ビューのスキーマ・デザイン・ガイド

サマリー管理のさまざまなコンポーネントの定義と使用を開始する前に、スキーマ・デザインを見直して、可能な場合には次のガイドラインに従うことをお勧めします。

**ガイドライン 1:** ディメンションは、非正規化（各ディメンションが 1 つの表に含まれる）する必要があります。または、正規化または部分正規化ディメンションの表間の結合で、子側の各行が親側の 1 行のみと結合されることを保証する必要があります。この条件を維持することの利点は、30-3 ページの「[ディメンションの作成](#)」に記載されています。

必要であれば、この条件は、FOREIGN KEY および NOT NULL 制約を子側の結合キーに追加し、PRIMARY KEY 制約を親側の結合キーに追加することによって施行できます。マテリアライズド・ビューにディテール表が 1 つしか含まれていない場合、またはマテリアライズド・ビューが集計を実行しない場合は、通常の結合のかわりに外部結合を使用するのが望ましい代替策です。この場合は、参照整合性制約を施行しなくても、Oracle オプティマイザが結果の整合性を保証できます。

**ガイドライン 2:** ディメンションが非正規化または部分非正規化の場合は、ディメンション表のキー列間で階層整合性を維持する必要があります。各子キー値は、ディメンション表が非正規化されている場合でも、親キー値を一意に特定する必要があります。非正規化ディメンションの階層整合性は、DBMS\_OLAP パッケージの VALIDATE\_DIMENSION プロシージャをコールすることで検証できます。

**ガイドライン 3:** ファクト表とディメンション表では、同様にファクト表の各行がディメンション表の 1 行のみと結合されることを保証する必要があります。この条件は、FOREIGN KEY 制約と NOT NULL 制約をファクト・キー列に追加し、PRIMARY KEY 制約をディメンション・キー列に追加するか、ガイドライン 1 で説明したように外部結合を使用することによって宣言する必要があります。データ・ウェアハウスでは、制約施行のパフォーマンス・オーバーヘッドを回避するために、制約は一般に NOVALIDATE および RELY オプションを指定して使用可能にします。

**ガイドライン 4:** ディテール・データの増分ロードは、SQL\*Loader のダイレクト・パス・オプションを使用するか、Oracle のダイレクト・パス・インタフェース (APPEND または PARALLEL ヒントを指定した INSERT AS SELECT を含みます) を使用するバルク・ローダーを使用して行う必要があります。マテリアライズド・ビューに複数の表が含まれ、かつ、集計を実行する場合、またはマテリアライズド・ビューのログが定義されていない場合は、データに対して他のタイプの DML を実行すると、完全リフレッシュが必要になります。

- ガイドライン 5:** 可能な場合は、単調に増加する時間列（望ましいタイプは DATE です）によって表を水平パーティション化します。各表について、各キー列のビットマップ索引を作成し、すべてのキー列を含むローカル索引を 1 つ作成します。パフォーマンスを最大化するために、各水平パーティションを複数の記憶デバイスにストライプ化します。
- ガイドライン 6:** 各ロードの後、マテリアライズド・ビューをリフレッシュする前に、DBMS\_OLAP パッケージの `VALIDATE_DIMENSION` プロシーダを使用して、追加したデータに対しディメンションの整合性を検証します。
- ガイドライン 7:** ファクト表と同じように、マテリアライズド・ビューの水平パーティション化と索引作成を行います。マテリアライズド・ビューのすべてのキーを、ローカル複合索引を含めます。

ガイドライン 1、2 および 3 は、それぞれスキーマ・デザインの際に重要ですが、ガイドライン 1 と 2 はガイドライン 3 よりもさらに重要です。スキーマ・デザインでガイドライン 1 と 2 に従わない場合は、ガイドライン 3 に従っても意味がありません。ガイドライン 1、2 および 3 は、クエリー・リライトのパフォーマンスとマテリアライズド・ビューのリフレッシュのパフォーマンスの両方に影響します。ガイドライン 4 は、マテリアライズド・ビューのリフレッシュのパフォーマンスにのみ影響します。スキーマ・デザインがガイドライン 4 に従っていない場合、マテリアライズド・ビューの増分リフレッシュは不可能または非常に効率が悪くなります。

制約を使用可能にするのに必要な時間、および制約違反があるかどうかを考慮する場合は、`ENABLE NOVALIDATE` 句を使用して、既存の制約を検査しない制約チェックをオンにします。このアプローチのリスクは、制約が破られた場合に間違った問合せ結果が生成される可能性があることです。したがって、設計者は、データがどの程度クリーンであるか、および潜在的な不正結果のリスクが非常に重大かどうかを判断します。

マテリアライズド・ビュー管理は、データ・ウェアハウス設計がこれらのガイドラインに従っていない場合でも、クエリー・リライトやマテリアライズド・ビューのリフレッシュなどの多くの有用な機能を実行できます。ただし、スキーマ・デザインでこれらのガイドラインに従うと、問合せ実行のパフォーマンスとマテリアライズド・ビューのリフレッシュのパフォーマンスを大幅に改善でき、必要なマテリアライズド・ビューの数が少なくなることを留意ください。

## データ・ウェアハウスのための Oracle Tools

マテリアライズド・ビューの分析と管理に役立つ強力なツールが提供されているかということは、データ・ウェアハウスのコストの制御における重要な要因です。次の Oracle Tools は、データ・ウェアハウスの作成と管理に役立ちます。

**Data Mart Designer** または **Oracle Designer** を使用して、ウェアハウス・スキーマを設計できます。その後で、稼働中のシステムからデータ・ウェアハウスまたはデータ・マートにデータが抽出、変換および転送（ETT）されます **Data Mart Builder** を使用して、ETT プロセスの指定、ターゲットのデータ・マートへの移入、およびロードと索引再作成の自動スケジュールを行うことができます。

Discoverer を使用すると、データベースを問い合わせることができ、Discoverer を介して実行された問合せは適切なときにリライトできます。Discoverer は問合せ使用に関する独自の作業負荷統計を保持しているため、Discoverer サマリー・ウィザードによって、どのマテリアライズド・ビューを作成すべきかを提示できます。

データ・マートは、Discoverer で直接分析するか、オプションとして Relational Access Manager (RAM) を介して **Express** マルチディメンション・データベース・サーバーにエクスポートできます。Express でのデータの分析は、RAM を介して Oracle8i サーバーに保存されているディテール・データへのアクセスをサポートし、Oracle Sales Analyzer (OSA) や Oracle Express Objects (OEO) などのツールへのリレーショナル・アクセスを可能にします。

## スタート・ガイド

次の章以降では、マテリアライズド・ビューとディメンションの作成方法を説明します。マテリアライズド・ビューは、いつでも作成でき、そのためウェアハウスのリフレッシュやクエリー・リライトなどのサマリー管理の他の機能で使用できますが、いくつかのパラメータを設定する必要があります。これらのパラメータは、初期化パラメータ・ファイルで定義するか、ALTER SYSTEM コマンドまたは ALTER SESSION コマンドを使用して定義できます。必須パラメータは、対象となる領域毎に決められています。

### ■ ウェアハウスのリフレッシュ

`JOB_QUEUE_PROCESSES`

バックグラウンド・プロセスの数。このパラメータは、同時にリフレッシュできるマテリアライズド・ビューの数を決定します。

`JOB_QUEUE_INTERVAL`

新しいジョブがジョブ・キューに送られたかどうかをジョブ・キュー・スケジューラがチェックする秒単位の間隔。

`UTL_FILE_DIR`

リフレッシュ・ログが書き込まれるディレクトリ。このパラメータが指定されていない場合、リフレッシュ・ログは作成されません。

### ■ クエリー・リライト

`OPTIMIZER_MODE="ALL_ROWS"、"FIRST_ROWS" または "CHOOSE"`

分析された表で、クエリー・リライトに必要なコストベースのオプティマイザが使用されることを保証します。

`QUERY_REWRITE_ENABLED = True`

クエリー・リライトをオンにします。

`QUERY_REWRITE_INTEGRITY = enforced、trusted または stale_tolerated`

オプション。クエリー・リライトの対象となるにはマテリアライズド・ビューがどれぐらい新しくなければならぬかをアドバイスします。  
QUERY\_REWRITE\_INTEGRITY の値の詳細は、『Oracle8i リファレンス・マニュアル』を参照してください。

#### COMPATIBLE

8.1 以上であることが必要です。

### ■ アドバイザの作業負荷

推奨パラメータ：

ORACLE\_TRACE\_COLLECTION\_NAME = oraclsm

トレース収集ファイル名。

ORACLE\_TRACE\_COLLECTION\_PATH = ?/otrace/admin/cdf

収集ファイルが格納される場所。

ORACLE\_TRACE\_COLLECTION\_SIZE = 0

収集ファイルの初期サイズ。

必須パラメータとその設定：

ORACLE\_TRACE\_ENABLE=true

Oracle Trace 収集をオンにします。

ORACLE\_TRACE\_FACILITY\_NAME = oraclsm

データを収集するトレース機能。

ORACLE\_TRACE\_FACILITY\_PATH = ?/otrace/admin/cdf

トレース機能定義ファイルの場所。

### ■ 並列性のための推奨パラメータ

PARALLEL\_MAX\_SERVERS

並列性を考慮するには、十分に高い値を設定する必要があります。

SORT\_AREA\_SIZE

HASH\_AREA\_SIZE より小さくする必要があります。

OPTIMIZER\_MODE

CHOOSE (コストベースのオブティマイザ) と等しくする必要があります。

OPTIMIZER\_PERCENT\_PARALLEL

100 と等しくする必要があります。



これらのパラメータに適切な値を設定すると、サマリー管理機能を使用する準備ができます。



---

## マテリアライズド・ビュー

Oracle8i で導入されたマテリアライズド・ビューは、データのサマリー、事前計算、レプリケートおよび分散に使用される汎用オブジェクトです。マテリアライズド・ビューは、データ・ウェアハウス、意思決定支援、分散コンピューティング、モバイル・コンピューティングなどのさまざまなコンピューティング環境に適しています。

さまざまなコンピューティング環境におけるマテリアライズド・ビューの管理と使用に関して包括的で堅牢なサポートを提供するために、いくつかの新しい機能領域が開発されました。新機能には、透過的クエリー・リライト、オブジェクト依存性管理、マテリアライズド・データの古さの追跡、コミット時のトランザクションとして整合性のとれたリフレッシュなどの新しいリフレッシュ方法、およびダイレクト・パスと DML ログを使用し非常に効率のよい増分高速リフレッシュが含まれます。

この章のトピックは次のとおりです。

- マテリアライズド・ビューの必要性
- マテリアライズド・ビューの作成
- 既存のマテリアライズド・ビューの登録
- マテリアライズド・ビューのパーティション化
- マテリアライズド・ビューに対する索引作成の選択
- マテリアライズド・ビューの無効化
- データ・ウェアハウスにおけるマテリアライズド・ビューの使用のガイドライン
- マテリアライズド・ビューの変更
- マテリアライズド・ビューの削除

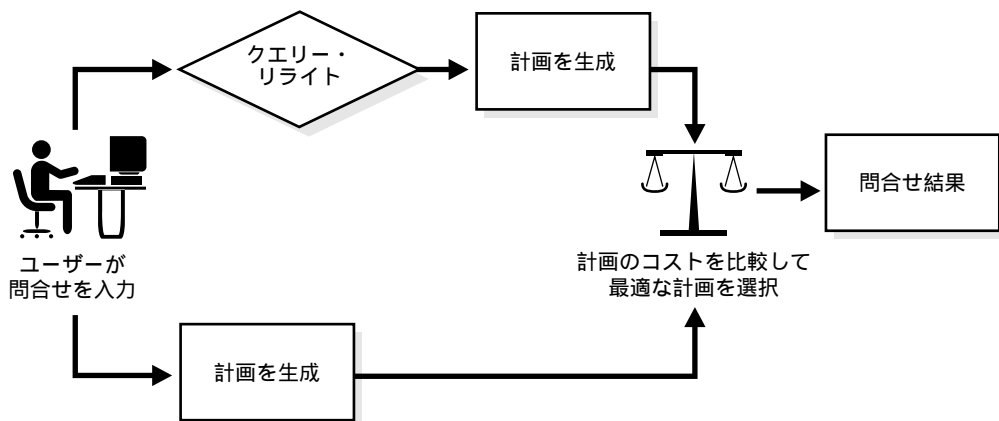
## マテリアライズド・ビューの必要性

マテリアライズド・ビューは、非常に大規模なデータベースの問合せの速度を上げるために、ウェアハウスで使用されます。大規模データベースへの問合せには、表間の結合または SUM などの集計、あるいはその両方が関係することがあります。これらの操作は、時間と処理能力の点で非常にコストがかかります。作成されるマテリアライズド・ビューのタイプによって、どのようにリフレッシュでき、クエリー・リライトでどのように使用されるかが決まります。

マテリアライズド・ビューの使用方法はいくつかあり、いくつかのロールはほぼ同じ構文を使用して実行できます。たとえば、以前は CREATE SNAPSHOT コマンドを使用して行っていたデータのレプリケートを、マテリアライズド・ビューを使用して行うことができます。CREATE MATERIALIZED VIEW は、CREATE SNAPSHOT のシノニムです。

マテリアライズド・ビューは、コストの高いデータベース上の結合操作や集計操作を実行前に事前計算し、その結果をデータベースに格納することによって、問合せパフォーマンスを改善します。問合せ最適化は、既存のマテリアライズド・ビューで要求を満たせるとき、および要求を満たす必要のあるときを自動的に認識することによって、マテリアライズド・ビューを利用します。そして、マテリアライズド・ビューを使用するように要求を透過的にリライトします。問合せは、基礎となるディテール表やビューではなくマテリアライズド・ビューに送られます。詳細なリレーションではなくマテリアライズド・ビューを使用するように問合せをリライトすると、パフォーマンスが大幅に向上します。

図 29-1 透過的なクエリー・リライト



したがって、クエリー・リライトを使用するときは、最大数の問合せを満足させるマテリアライズド・ビューの作成が必要です。たとえば、一般にディテール表やファクト表に適用される 20 の問合せを対象にした場合は、適切に作成された 5 つか 6 つのマテリアライズド・ビューを使用してそれらの問合せを満足させることができるかもしれません。マテリアライ

ズド・ビュー定義には、任意の数の集計 (SUM、COUNT(x)、COUNT(\*)、COUNT(DISTINCT x)、AVG、VARIANCE、STDDEV、MIN および MAX) または任意の数の結合、あるいはその両方を含めることができます。どのマテリアライズド・ビューを作成すればよいかわからない場合は、Oracle が DBMS\_OLAP パッケージに用意している一連のアドバイザ機能が、クエリー・リライトでのマテリアライズド・ビューの設計と評価に役立ちます。

マテリアライズド・ビューをクエリー・リライトで使用する場合は、ファクト表またはディテール表と同じデータベースに格納しておく必要があります。マテリアライズド・ビューはパーティション化できます。また、パーティション表にマテリアライズド・ビューを定義することや、マテリアライズド・ビューに 1 つ以上の索引を定義することもできます。

マテリアライズド・ビューは、いくつかの点で索引と似ています。つまり、記憶領域を消費し、マスター表のデータが変更されたときはリフレッシュする必要があります。また、クエリー・リライトで使用すると SQL 実行のパフォーマンスが改善され、その存在は SQL アプリケーションとユーザーに対して透過的です。索引とは異なり、マテリアライズド・ビューには SELECT 文を使用して直接アクセスできます。また、必要なリフレッシュのタイプに応じて、INSERT、UPDATE または DELETE 文で間接的にアクセスすることもできます。

---

**注意：** マテリアライズド・ビューは、Oracle Replication でも使用できます。この章で説明する手法は、データ・ウェアハウス内のマテリアライズド・ビューの使用法を示しています。

---

## マテリアライズド・ビューの作成

マテリアライズド・ビューを作成するには、CREATE MATERIALIZED VIEW コマンドを使用します。次のコマンドは、マテリアライズド・ビュー *store\_sales\_mv* を作成します。

```
CREATE MATERIALIZED VIEW store_sales_mv
  PCTFREE 0 TABLESPACE mvviews
  STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
  PARALLEL
  BUILD DEFERRED
  REFRESH COMPLETE
  ENABLE QUERY REWRITE
AS
SELECT
  s.store_name,
  SUM(dollar_sales) AS sum_dollar_sales
  FROM store s, fact f
  WHERE f.store_key = s.store_key
  GROUP BY s.store_name;
```

**関連項目：** CREATE MATERIALIZED VIEW の詳細な説明は、『Oracle8i SQL リファレンス』を参照してください。

データ・ウェアハウスには作成済みのサマリーや集計表があることが多いですが、DBA は、新しいマテリアライズド・ビューを作成することでこれらの作成作業を繰返すことは望まないでしょう。この場合は、データベースに存在している表を事前作成マテリアライズド・ビューとして登録できます。この手法は、29-15 ページの「[既存のマテリアライズド・ビューの登録](#)」で説明されています。

作成するマテリアライズド・ビューを選択したら、各マテリアライズド・ビューについて次のステップを実行します。

1. マテリアライズド・ビューの物理設計を行います（既存のユーザー定義マテリアライズド・ビューでは、このステップは不要です）。マテリアライズド・ビューは、時間属性によって水平パーティション化する必要があり（可能な場合）、最大または最も頻繁に更新されるディテール表やファクト表のパーティション化と合致している必要があります（可能な場合）。水平パーティションの数が多いと、Oracle でパラレル機能を利用できるので、一般にリフレッシュのパフォーマンスが向上します。
2. CREATE MATERIALIZED VIEW 文を使用してマテリアライズド・ビューを作成し、オプションで移入を行います。ユーザー定義マテリアライズド・ビューがすでに存在している場合は、CREATE MATERIALIZED VIEW 文で PREBUILT オプションを使用します。それ以外の場合は、BUILD IMMEDIATE オプションを使用してマテリアライズド・ビューに即時に移入を行うか、BUILD DEFERRED オプションを使用して都合のよいときにマテリアライズド・ビューに移入を行います（ENABLE QUERY REWRITE 句が指定されていると、マテリアライズド・ビューは、最初のリフレッシュまでクエリー・リライトで使用不可になり、リフレッシュの後で自動的に使用可能になります）。

**関連項目：** SQL 文 CREATE MATERIALIZED VIEW、ALTER MATERIALIZED VIEW および DROP MATERIALIZED VIEW の説明は、『Oracle8i SQL リファレンス』を参照してください。

## 命名

マテリアライズド・ビューに与えられる名前は、標準の Oracle 命名規則に従っている必要があります。ただし、マテリアライズド・ビューがユーザーの事前作成表をベースとする場合は、マテリアライズド・ビューの名前は表名と正確に一致している必要があります。

表と索引に対する命名規則がすでにある場合は、マテリアライズド・ビューを簡単に識別できるように、この命名スキームをマテリアライズド・ビューに拡張することを検討します。たとえば、マテリアライズド・ビューに *sum\_of\_sales* という名前を付けるかわりに、*sum\_of\_sales\_mv* という名前にして、これが表やビューなどではなくマテリアライズド・ビューであることを示すことができます。

## 記憶領域特性

マテリアライズド・ビューは、ユーザー定義の事前作成表をベースとしていない限り、データベースの内部に記憶領域を必要とします。したがって、マテリアライズド・ビューの記憶領域の必要量は、そのマテリアライズド・ビューが存在する表領域とエクステント・サイズで指定する必要があります。

マテリアライズド・ビューに必要な記憶容量が不明な場合は、32-14 ページの「[サマリー・アドバイザー](#)」で説明する DBMS\_OLAP.ESTIMATE\_SIZE パッケージを使用して、このマテリアライズド・ビューの格納に必要なバイト数を見積ることができます。この情報は、デザイン・チームがマテリアライズド・ビューをどの表領域に入れるかを決定するときにも役立ちます。

**関連項目：** STORAGE 方法の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## 作成方法

次の表で示すように、マテリアライズド・ビューの作成方法は 2 通りあります。BUILD IMMEDIATE を選択した場合は、マテリアライズド・ビュー定義がデータ・ディクショナリスキーマ・オブジェクトに追加された後、ファクト表またはディテール表が SELECT 式によって走査され、その結果がマテリアライズド・ビューに格納されます。走査される表のサイズに応じて、この作成プロセスにはかなり時間がかかることがあります。

BUILD DEFERRED 句を使用する方法もあります。この方法では、データのないマテリアライズド・ビューが作成されるので、32-3 ページの「[ウェアハウス・リフレッシュ](#)」で説明する DBMS\_MVIEW.REFRESH パッケージを使用してデータを後で移入できます。

作成方法	説明
BUILD DEFERRED	マテリアライズド・ビュー定義を作成しますが、データは移入しません。
BUILD IMMEDIATE	マテリアライズド・ビューを作成し、データを移入します。

## クエリー・リライトでの使用

マテリアライズド・ビューが定義されても、クエリー・リライト機能で自動的に使用されるわけではありません。したがって、マテリアライズド・ビューを問合せのリライトに使用できると思われる場合は、ENABLE QUERY REWRITE 句を指定する必要があります。

マテリアライズド・ビューが最初に作成されるときに、この句が省略されるか DISABLE QUERY REWRITE として指定された場合は、ALTER MATERIALIZED VIEW 文を使用して、クエリー・リライトに対してマテリアライズド・ビューを後から使用可能にすることができます。

## クエリー・リライトの制限事項

クエリー・リライトは、すべてのマテリアライズド・ビューで可能なわけではありません。期待したときにクエリー・リライトが行われなかった場合は、マテリアライズド・ビューが次の条件のいずれかを満たしていないかを確認します。

### マテリアライズド・ビューの制限事項

1. 問合せの定義には、結果の再現が不可能な式があってはなりません（ROWNUM、SYSDATE、結果の再現が不可能な PL/SQL 関数など）。
2. RAW または LONG RAW データ型への参照やオブジェクト REF があってはなりません。
3. 問合せは、単一ブロック問合せであることが必要です。つまり、集合関数（UNION、MINUS など）を含むことはできません。ただし、マテリアライズド・ビューは、複数の問合せブロックを持つことができます（たとえば、FROM 句のインライン・ビューや、WHERE 句または HAVING 句のサブセレクト）。
4. マテリアライズド・ビューが PREBUILT として登録された場合、列の精度は、WITH REDUCED PRECISION によって上書きされない限り、対応する SELECT 式の精度と一致している必要があります。

### クエリー・リライトの制限事項

1. ディテール表は、ローカルであることが必要です。問合せでアクセスしたりマテリアライズド・ビューの定義で使用したりできるのは、ローカルのディテール表またはビューだけです。
2. SYS はディテール表を所有できません。また、SYS はマテリアライズド・ビューを所有できません。

### SQL テキスト以外のリライトの制限事項

1. FROM リストには、同じ表またはビューの複数のオカレンスを含めることができません。
2. SELECT リストや GROUP BY リストが存在する場合は、問合せとマテリアライズド・ビューで同じであることが必要です。また、これらのリストが含む列は単純列である必要があります。つまり、列では式が許可されません。
3. 集計演算子は、式の一番外側にのみ許されます。つまり、AVG(AVG(x)) や AVG(x)+AVG(x) などの集計は許可されません。
4. WHERE 句には、AND で連結できる内部結合または外部結合のみが含まれている必要があります。したがって、WHERE 句に OR や単一表の選択があってはなりません。
5. HAVING または CONNECT BY 句は許可されません。



## リフレッシュ・オプション

マテリアライズド・ビューをディテール表やファクト表からリフレッシュする場合は、`CREATE MATERIALIZED VIEW` 文に `REFRESH` 句を付加する必要があります。 `REFRESH` 句を定義するときは、リフレッシュのタイプおよびリフレッシュの実行方法という2つの要素を指定する必要があります。

2つのリフレッシュ実行モードは、`ON COMMIT` と `ON DEMAND` です。どちらの方式を選択するかによって、定義できるマテリアライズド・ビューのタイプが決まります。

リフレッシュ・モード	説明
<code>ON COMMIT</code>	リフレッシュは、マスター表で次に <code>COMMIT</code> が実行されたときに自動的に行われます。単一表集計のマテリアライズド・ビューおよび結合のみを含むマテリアライズド・ビューで使用できます。
<code>ON DEMAND</code>	リフレッシュは、 <code>DBMS_MVIEW</code> パッケージに含まれる使用可能なリフレッシュ・プロシージャ ( <code>REFRESH</code> 、 <code>REFRESH_ALL_MVIEWS</code> 、 <code>REFRESH_DEPENDENT</code> ) の1つをユーザーが手動で実行したときに行われます。

マテリアライズド・ビューが `ON COMMIT` 方式を使用してリフレッシュされる場合は、リフレッシュ操作の後にアラート・ログとトレース・ファイルを調べて、エラーが発生していないことを確認する必要があります。

マテリアライズド・ビューが `COMMIT` 時のリフレッシュでエラーになった場合、ユーザーは、トレース・ファイルで示されたエラーに対処した後で、`DBMS_MVIEW` パッケージを使用してリフレッシュ・プロシージャを明示的に起動する必要があります。この処理を行うまで、ビューはコミット時に自動的にリフレッシュされなくなります。

`ON DEMAND` 実行モードを選択すると、マテリアライズド・ビューのウェアハウス・リフレッシュ機能を利用できます。この機能は、マテリアライズド・ビューの全体リフレッシュまたはディテール・データへの追加のみのリフレッシュで、速く効率のよいリフレッシュ・メカニズムを提供します。

`FORCE`、`COMPLETE`、`FAST` および `NEVER` という4つのオプションの1つを選択することによって、マテリアライズド・ビューをディテール表からリフレッシュする方法を指定できます。

リフレッシュ・オプション	説明
<code>COMPLETE</code>	<code>ATOMIC REFRESH=TRUE</code> の場合は、マテリアライズド・ビューの定義問合せを再計算することによってリフレッシュします。 <code>ATOMIC REFRESH=FALSE</code> の場合、 <code>COMPLETE</code> は <code>FORCE</code> と同じです。

リフレッシュ・オプション	説明
FAST	ダイレクト・パスを使用するかマテリアライズド・ビュー・ログの内容に基づいて、表に挿入された新規データの増分を追加することによってリフレッシュします（高速リフレッシュ、増分リフレッシュ）。
FORCE	最初に、高速リフレッシュが可能かどうかを判断し、可能な場合は適用します。可能でない場合は、完全リフレッシュを適用します。
NEVER	マテリアライズド・ビューのリフレッシュを抑制します。

高速リフレッシュ・オプションが使用可能かどうかは、作成されたマテリアライズド・ビューのタイプに依存します。次の表に、マテリアライズド・ビューのタイプごとに高速リフレッシュが可能になる条件を示します。これらの条件が満たされていないと、マテリアライズド・ビューの作成は失敗し、エラーが発生します。

表 29-1 マテリアライズド・ビューの高速リフレッシュの要件

	マテリアライズド・ビューにあるもの：		
	結合のみ	結合と集計	単一表の集計
ディテール表のみ	X	X	X
単一表のみ	-	-	X
表が FROM リストに 1 回のみ現れる	X	X	X
SYSDATE や ROWNUM などの結果の再現が不可能な式	X	X	X
RAW や LONG RAW への参照がない	X	X	X
GROUP BY がない	X	-	-
すべてのディテール表の ROWID が問合せの SELECT リストに存在する	X	-	-
式は、同一である場合に限り、GROUP BY 句と SELECT 句で許可される	-	X	X
集計は許可されるが、ネストはできない	-	X	X
AVG があるときは COUNT もある	-	X	X
SUM があるときは COUNT もある	-	-	X
VARIANCE があるときは COUNT と SUM がある	-	X	X
STDDEV があるときは COUNT と SUM がある	-	X	X
WHERE 句に含まれる結合述語は、OR ではなく AND でつながれている	X	X	-

表 29-1 マテリアライズド・ビューの高速リフレッシュの要件

WHERE 句はない	-	-	X
HAVING または CONNECT BY はない	X	X	X
副問合せ、インライン・ビュー、または UNION や MINUS などの集合関数はない	X	X	X
COUNT(*) がなければならない	-	-	X
MIN と MAX は許可されない	-	-	X
外部結合の場合は、一意制約が結合の内部表の結合列に存在する必要がある	X	-	-
マテリアライズド・ビューのログが存在する必要がある。このログには、マテリアライズド・ビューで参照されるすべての列を含んでいて、かつ、LOG NEW VALUES 句をつけて作成されている必要がある	-	-	X
すべてのディテール表の ROWID のあるマテリアライズド・ビューのログが存在する必要がある	X	-	-
SELECT および GROUP BY の集計以外の式は、単純列でなければならない	-	-	X
ディテール表に対する DML	X	-	X
ダイレクト・パス・データ・ロード	X	X	X
ON COMMIT	X	-	X
ON DEMAND	X	X	X

## マテリアライズド・ビューのデータの定義

マテリアライズド・ビューの SELECT 句では、ビューに含めるデータを定義します。SELECT 句で指定できるものには、いくつかの制限があります。表はいくつでも結合できますが、クエリー・リライトやウェアハウス・リフレッシュ機能を利用する場合は、リモート表を結合できません。SELECT 句で結合または参照できるのは、表だけではありません。ビュー、インライン・ビュー、副問合せ、マテリアライズド・ビューはすべて許可されません。

### 結合と集計のあるマテリアライズド・ビュー

データ・ウェアハウスでは、通常、次の表に示す集計の 1 つがマテリアライズド・ビューに含まれています。ウェアハウスの増分リフレッシュを行うには、SELECT リストに GROUP BY 列のすべてが含まれている必要があります（存在する場合） また 1 つ以上の集計関数を含めることができます。集計関数は、SUM、ACOUNT(x)、ACOUNT(\*）、COUNT(DISTINCT x)、AVG、VARIANCE、STDDEV、MIN および MAX のいずれかである必要があります、集計される式は任意の SQL 値式です。

次に、作成できるマテリアライズド・ビューのタイプの例をいくつか示します。

#### マテリアライズド・ビューの作成 : 例 1

```
CREATE MATERIALIZED VIEW store_sales_mv
  PCTFREE 0 TABLESPACE mviews
  STORAGE (initial 16k next 16k pctincrease 0)
  BUILD DEFERRED
  REFRESH COMPLETE ON DEMAND
  ENABLE QUERY REWRITE
  AS
  SELECT
    s.store_name,
    SUM(dollar_sales) AS sum_dollar_sales
  FROM store s, fact f
  WHERE f.store_key = s.store_key
  GROUP BY s.store_name;
```

上記の文は、*store* 別の *sales* の合計を計算するマテリアライズド・ビュー *store\_sales\_mv* を作成します。これは、表 *store* と *fact* を列 *store\_key* で結合することによって導出されます。作成方法が DEFERRED なので、最初はマテリアライズド・ビューにデータは含まれていません。マテリアライズド・ビューがリフレッシュされるときは、完全リフレッシュが実行され、このマテリアライズド・ビューは、移入された後でクエリー・リライトで使用できるようになります。

#### マテリアライズド・ビューの作成 : 例 2

```
CREATE MATERIALIZED VIEW store_avgcnt_mv
  PCTFREE 0 TABLESPACE mviews
  STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
  BUILD IMMEDIATE
```

```

REFRESH COMPLETE
ENABLE QUERY REWRITE
AS
SELECT store_name,
       AVG(unit_sales) AS avgcnt_unit_sales,
       COUNT(DISTINCT(f.time_key)) AS count_days
FROM store s, fact f, time t
WHERE s.store_key = f.store_key AND
      f.time_key = t.time_key
GROUP BY store_name, t.time_key;

```

上記の文は、特定の日付に *store* によって販売された平均ユニット数を計算するマテリアライズド・ビュー *store\_avgcnt\_mv* を作成します。これは、表 *store*、*time* および *fact* を列 *store\_key* と *time\_key* で結合することによって導出されます。作成方法が IMMEDIATE なので、マテリアライズド・ビューにはデータが即時に移入され、クエリー・リライトで使用できます。ON DEMAND 句はオプションなので、このマテリアライズド・ビュー定義では省略されていることに注意してください。ON DEMAND 句はデフォルトなので、マテリアライズド・ビューは手動で要求されるまでリフレッシュされません。

### マテリアライズド・ビューの作成 : 例 3

```

CREATE MATERIALIZED VIEW store_stdcnt_mv
PCTFREE 0 TABLESPACE mviews
STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
BUILD IMMEDIATE
REFRESH FAST
ENABLE QUERY REWRITE
AS
SELECT store_name, t.time_key,
       STDDEV(unit_sales) AS stdcnt_unit_sales
       AVG(unit_sales) AS avgcnt_unit_sales
       COUNT(unit_sales) AS count_days
       SUM(unit_sales) AS sum_unit_sales
FROM store s, fact f, time t
WHERE s.store_key = f.store_key AND
      f.time_key = t.time_key
GROUP BY store_name, t.time_key;

```

上記の文は、指定された日付に *store* によって販売されたユニット数の標準偏差を計算するマテリアライズド・ビュー *store\_stdcnt\_mv* を作成します。これは、表 *store*、*time* および *fact* を列 *store\_key* と *time\_key* で結合することによって導出されます。作成方法が IMMEDIATE なので、マテリアライズド・ビューにはデータが即時に移入され、クエリー・リライトで使用できます。この例では、リフレッシュ方法が FAST です。STDDEV 集計の高速リフレッシュをサポートする COUNT および SUM 集計が含まれているので、このリフレッシュ方法が許可されます。

### マテリアライズド・ビューの作成 : 例 4

```
CREATE MATERIALIZED VIEW store_var_mv
PCTFREE 0 TABLESPACE mviews
  STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
PARALLEL
BUILD DEFERRED
REFRESH FORCE
AS
  SELECT s.store_key, store_name,
         VARIANCE(unit_sales) AS var_unit_sales
  FROM fact f, store s, time t
 WHERE s.store_key = f.store_key AND
        f.time_key = t.time_key
 GROUP BY s.store_key, t.time_key, store_name;
```

上記の文は、指定された日付に *store* によって販売されたユニット数の平方偏差を計算するマテリアライズド・ビュー *store\_stdcnt\_mv* を作成します。これは、表 *store*、*time* および *fact* を列 *store\_key* と *time\_key* で結合することによって導出されます。マテリアライズド・ビューに、データは即時に移入されません。また、ENABLE QUERY REWRITE 句が指定されていないので、マテリアライズド・ビューはクエリー・リライトで使用できません。リフレッシュ方法は FORCE であり、最適なりフレッシュ方法が選択されることを意味します。

### 単一表集計マテリアライズド・ビュー

1 つ以上の集計 (SUM、AVG、VARIANCE、STDDEV、COUNT) と GROUP BY を含むマテリアライズド・ビューは、単一表をベースとすることがあります。SUM(a\*b) などのように、集計関数は列の式を伴うことがあります。マテリアライズド・ビューが増分リフレッシュされる場合、マテリアライズド・ビューのログは、INCLUDING NEW VALUES オプションつきでディテール表に作成される必要があり、マテリアライズド・ビュー問合せ定義で参照されるすべての列を含んでいる必要があります。

このリリースでは、マテリアライズド・ビューと、マテリアライズド・ビューが依存するすべての実表が、同じスキーマに属する必要があることを前提としています。

```
CREATE MATERIALIZED VIEW log on fact
  with rowid (store_key, time_key, dollar_sales, unit_sales)
  including new values;

CREATE MATERIALIZED VIEW sum_sales
PARALLEL
BUILD IMMEDIATE
REFRESH FAST ON COMMIT
AS
  SELECT f.store_key, f.time_key,
         COUNT(*) AS count_grp,
         SUM(f.dollar_sales) AS sum_dollar_sales,
         COUNT(f.dollar_sales) AS count_dollar_sales,
```

```
SUM(f.unit_sales) AS sum_unit_sales,
      COUNT(f.unit_sales) AS count_unit_sales
FROM fact f
GROUP BY f.store_key, f.time_key;
```

この例では、単一表の集計を含むマテリアライズド・ビューが作成されています。マテリアライズド・ビューのログが作成されているので、マテリアライズド・ビューは高速リフレッシュが可能です。DML がファクト表に適用される場合は、コミットが発行されると、変更がマテリアライズド・ビューに反映されます。

表 29-2 に、単一表集計マテリアライズド・ビューの集計要件を示します。

**表 29-2 単一表集計の要件**

**集計 X が存在する場合は、集計 Y が必須で、集計 Z はオプションです。**

X	Y	Z
COUNT(expr)	-	-
SUM(expr)	COUNT(expr)	-
AVG(expr)	COUNT(expr)	SUM(expr)
STDDEV(expr)	COUNT(expr)	SUM(expr * expr)
VAR(expr)	COUNT(expr)	SUM(expr * expr)

COUNT(\*) は常に存在することに注意してください。

## 結合のみを含むマテリアライズド・ビュー

*fact* 表を *store* 表に結合するマテリアライズド・ビューが作成されている次の例のように、マテリアライズド・ビューは結合のみを含み、集計を含まないことがあります。このタイプのマテリアライズド・ビューを作成する利点は、コストのかかる結合が事前計算されることです。

REFRESH FAST を指定すると、Oracle は問合せ定義の詳細な検証を実行して、ディテール表のいずれかが変更された場合に必ず高速リフレッシュを実行できることを確認します。これらの追加チェックには、次のものが含まれます。

1. ディテール表ごとにマテリアライズド・ビューのログが存在する必要があります。
2. すべてのディテール表の ROWID が、マテリアライズド・ビュー問合せ定義の SELECT リストに存在する必要があります。
3. 外部結合がある場合は、内部表の結合列に一意制約が存在する必要があります。

たとえば、ファクト表とディメンション表を結合するときに、その結合が外部表であるファクト表との外部結合である場合は、ディメンション表の結合列に一意制約が存在する必要があります。

上記の制限のいずれかが満たされない場合は、マテリアライズド・ビューを REFRESH FORCE として作成する必要があります。表の 1 つがいずれかの基準を満たしていなくても、他の表が基準をすべて満たしていればマテリアライズド・ビューの増分リフレッシュは可能ですが、リフレッシュされるのはすべての基準を満たしている他の表のみです。

このリリースでは、マテリアライズド・ビューと、マテリアライズド・ビューが依存するすべての実表が、同じスキーマに属する必要があることを前提としています。

データ・ウェアハウスのスター・スキーマでは、領域に余裕がない場合には、ファクト表が最も頻繁に更新される表である場合にのみファクト表の ROWID を含めることができ、ユーザーはマテリアライズド・ビューを作成するときに FORCE オプションを指定できます。

マテリアライズド・ビューのログには、マスター表の ROWID が含まれている必要があります。他の列を追加する必要はありません。

結合のみを含むマテリアライズド・ビューの増分リフレッシュは、実表に対する任意のタイプの DML (ダイレクト・ロードまたは標準の INSERT、UPDATE または DELETE) の後で行うことができます。

結合のみを含むマテリアライズド・ビューは、ON COMMIT または ON DEMAND でリフレッシュされるように定義できます。ON COMMIT の場合は、ディテール表に対して DML を行うトランザクションのコミット時にリフレッシュが実行されます。

コミット時のリフレッシュの後に、アラート・ログとトレース・ファイルをチェックして、リフレッシュ時にエラーが発生したかどうかを確認してください。

リフレッシュを高速化するために、ファクト表の ROWID を格納するマテリアライズド・ビューの列に索引を作成することをお勧めします。

```
CREATE MATERIALIZED VIEW LOG ON fact
  WITH ROWID;

CREATE MATERIALIZED VIEW LOG ON time
  WITH ROWID;

CREATE MATERIALIZED VIEW LOG ON store
  WITH ROWID;

CREATE MATERIALIZED VIEW detail_fact_mv
  PARALLEL
  BUILD IMMEDIATE
  REFRESH FAST
  AS
  SELECT
    f.rowid "fact_rid", t.rowid "time_rid", s.rowid "store_rid",
    s.store_key, s.store_name, f.dollar_sales,
    f.unit_sales, f.time_key
  FROM fact f, time t, store s
  WHERE f.store_key = s.store_key(+) AND
    f.time_key = t.time_key(+);
```



上記の例では、REFRESH FAST を実行するために、*s.store\_key* と *t.time\_key* に一意制約が存在している必要があります。また、次に示すように、列 *fact\_rid*、*time\_rid* および *store\_rid* に索引を作成して、リフレッシュのパフォーマンスを改善することをお勧めします。

```
CREATE INDEX mv_ix_factrid ON
  detail_fact_mv(fact_rid);
```

あるいは、上記の例に列 *time\_rid* および *store\_rid* が含まれていない場合、またリフレッシュ方法が REFRESH FORCE の場合は、このマテリアライズド・ビューが高速リフレッシュ可能になるのは、*fact* 表が変更され、表 *time* または *store* が変更されない場合です。

```
CREATE MATERIALIZED VIEW detail_fact_mv
  PARALLEL
  BUILD IMMEDIATE
  REFRESH FORCE
  AS
  SELECT
    f.rowid "fact_rid",
    s.store_key, s.store_name, f.dollar_sales,
    f.unit_sales, f.time_key
  FROM fact f, time t, store s
  WHERE f.store_key = s.store_key(+) AND
    f.time_key = t.time_key(+);
```

## 既存のマテリアライズド・ビューの登録

データ・ウェアハウスのなかには、通常のユーザー表にマテリアライズド・ビューをインプリメントしているものもあります。このソリューションによって、マテリアライズド・ビューのパフォーマンスは向上しますが、すべての SQL アプリケーションでクエリー・リライトを行えるわけではありません。また、あるアプリケーションで定義されているマテリアライズド・ビューを別のアプリケーションで透過的にアクセスすることはできず、マテリアライズド・ビューの高速なパラレル・リフレッシュや高速な増分リフレッシュは一般にサポートされません。

これらの問題があり、既存のマテリアライズド・ビューが非常に大規模だと再作成にコストがかかることがあるので、可能な場合には既存のマテリアライズド・ビュー表を Oracle サーバーに登録してください。ユーザー定義マテリアライズド・ビューは、CREATE MATERIALIZED VIEW ... ON PREBUILT TABLE 文を使用して登録できます。マテリアライズド・ビューは、いったん登録されると、クエリー・リライトで使用したり、いずれかまたは両方のリフレッシュ方法によってメンテナンスしたりできるようになります。

場合によっては、ユーザー定義マテリアライズド・ビューは、更新サイクルよりも長いスケジュールでリフレッシュされます。たとえば、毎月のマテリアライズド・ビューは各月の終わりにのみ更新される場合もあります。その場合、マテリアライズド・ビューの値は、常に完了した期間のデータを参照したものです。これらのマテリアライズド・ビューから直接作成されたレポートは、現在の（未完了の）期間にはないデータのみを暗黙的に選択します。

ユーザー定義マテリアライズド・ビューがすでに時間ディメンションを含んでいる場合は、次のことが当てはまります。

- 登録し、更新サイクルごとに増分リフレッシュする必要があります。
- 検索対象の完了した期間を選択するビューを作成する必要があります。

たとえば、マテリアライズド・ビューがそれまで毎月の終わりにリフレッシュされていた場合、ビューには `WHERE time.month < CURRENT_MONTH()` という選択が含まれています。

- ユーザー定義マテリアライズド・ビューを直接参照するかわりに、ビューを参照するようにレポートを変更する必要があります。

ユーザー定義マテリアライズド・ビューに時間ディメンションが含まれていない場合は、次のことが当てはまります。

- 時間ディメンションを含む新しいマテリアライズド・ビューを作成する必要があります（可能な場合）。
- 新しいマテリアライズド・ビューの時間列で、ビューを集計する必要があります。

表は、マテリアライズド・ビューとして登録するときに、定義問合せのマテリアライゼーションを反映している必要があり、定義問合せの各列は、一致するデータ型を持つ表の列に対応している必要があります。ただし、`WITH REDUCED PRECISION` を指定して、定義問合せの列の精度を表の列の精度と変えることができます。

表とマテリアライズド・ビューは、同じ名前であればなりませんが、表は表としての特性を持ち、マテリアライズド・ビューの定義問合せでは参照されない列（非管理列）を含むことができます。リフレッシュ操作中に行が挿入された場合は、行の各非管理列はデフォルト値に設定されるので、非管理列は、デフォルト値を持たない限り `NOT NULL` 制約を持つことができません。

非管理列は、単一表集計マテリアライズド・ビューまたは結合のみを含むマテリアライズド・ビューではサポートされません。

事前作成表をベースとするマテリアライズド・ビューは、パラメータ `QUERY_REWRITE_INTEGRITY` が少なくとも `TRUSTED` レベルに設定されていれば、クエリー・リライトによる選択の対象になります。整合性レベルの詳細は、[第 31 章「クエリー・リライト」](#)を参照してください。

事前作成表で作成されたマテリアライズド・ビューを削除しても、表は存在したままになります。マテリアライズド・ビューの定義のみ削除されます。

事前作成表がマテリアライズド・ビューとして登録されるときは、作成時にマテリアライズド・ビューは古いものとしてマークされ、古い整合性モードのみ使用できるので、パラメータ `QUERY_REWRITE_INTEGRITY` を少なくとも `STALE_TOLERATED` 設定する必要があります。

```
CREATE TABLE sum_sales_tab
  PCTFREE 0 TABLESPACE mviews
```

```
STORAGE (INITIAL 16k NEXT 16k PCTINCREASE 0)
AS
SELECT f.store_key
       SUM(dollar_sales) AS dollar_sales,
       SUM(unit_sales) AS unit_sales,
       SUM(dollar_cost) AS dollar_cost
FROM fact f GROUP BY f.store_key;

CREATE MATERIALIZED VIEW sum_sales_tab
ON PREBUILT TABLE WITHOUT REDUCED PRECISION
ENABLE QUERY REWRITE
AS
SELECT f.store_key,
       SUM(dollar_sales) AS dollar_sales,
       SUM(unit_sales) AS unit_sales,
       SUM(dollar_cost) AS dollar_cost
FROM fact f GROUP BY f.store_key;
```

この例は、ユーザー定義表を登録するのに必要な 2 つのステップを示しています。最初に表を作成する必要があり、さらに、マテリアライズド・ビューが表とまったく同じ名前を使用して定義されます。このマテリアライズド・ビュー *sum\_sales\_tab* は、クエリー・リライトで使用できます。

## マテリアライズド・ビューのパーティション化

データ・ウェアハウスには大量のデータが保持されているので、パーティション化は、データベース設計者が使用できる非常に有用なオプションです。

時間属性でファクト表を水平パーティション化することによって、拡張性が増し、システム管理が単純化され、効率よく再作成できるローカル索引を定義することが可能になります。SQL\*Loader に対して、表の単一パーティションをロードするように指示できます。この場合は、対応するローカル索引パーティションのみ再作成されます。グローバル索引は、ダイレクト・ロードの後で完全に再作成する必要があります。これは、比較的少ない行数を大規模表にロードするときには非常にコストが高くなります。このため、ファクト表の索引はすべてローカル索引として定義することを強くお勧めします。この定義は、たとえば各キー列に対してビットマップ索引を作成し（ビットマップ索引は常にローカルです）すべてのキー列を単一の複数キー索引に含め、複数キー索引の先頭の列としてパーティション化属性を持たせることによって行うことができます。

リフレッシュ・プロシージャでは、パラレル DML を使用してマテリアライズド・ビューをメンテナンスできるので、マテリアライズド・ビューのパーティション化はリフレッシュに関しても利益があります。これらの利益を実現するには、マテリアライズド・ビューを PARALLEL として定義し、セッションでパラレル DML を使用可能にする必要があります。

データ・ウェアハウスまたはデータ・マートに時間ディメンションが含まれているときは、最も古い情報をアーカイブして、記憶域を新しい情報のために再利用するのが望ましい場合

があります。ファクト表またはマテリアライズド・ビューが時間ディメンションを含み、時間属性によって水平パーティション化されている場合は、ロールアウトされるデータ単位が水平パーティションと等しいか、少なくとも並んでいれば、マテリアライズド・ビューのローリングの管理は、いくつかの高速なパーティション・メンテナンス操作にまで削減できます。

ウェアハウスにローリングマテリアライズド・ビューを持つことを計画している場合は、パーティション・メンテナンス操作の実行を計画する頻度を決定し、ファクト表とマテリアライズド・ビューの水平パーティション化を計画して、古いデータが削除されるときに必要なシステム管理オーバーヘッド量を削減する必要があります。

Oracle8i で導入された新しいパーティション化オプションでは、レンジ・パーティションの使用を制限されません。たとえば、時間値と store\_key 値の両方を使用したコンボジット・パーティションによって、データの理想的なパーティション化が生成可能です。

パーティション化の詳細は、『Oracle8i 概要』を参照してください。

パーティション化を使用するのが理想的なケースは、マテリアライズド・ビューにデータのサブセットが含まれ、そのサブセットが、マテリアライズド・ビューに対する SELECT 式で WHERE time\_key < '1-OCT-1998' 形式の式を定義することによって取得された場合です。このタイプの WHERE 句が含まれている場合、クエリー・リライトは完全一致ケースに制限されるので、マテリアライズド・ビューが使用されるときに厳しい制限が課されます。この問題を克服するには、WHERE 句を使わないでパーティション化したマテリアライズド・ビューを使用します。これによって、クエリー・リライトはマテリアライズド・ビューを使用できるようになり、適切なパーティションのみ検索するようになるので、問合せパフォーマンスが向上します。

マテリアライズド・ビューのパーティション化には、次の 2 つの方法があります。

- マテリアライズド・ビューのパーティション化
- ビルトイン表のパーティション化

## マテリアライズド・ビューのパーティション化

マテリアライズド・ビューのパーティション化には、次の例で示すような標準の Oracle パーティション化句を使用してマテリアライズド・ビューを定義します。この例では、3 つのパーティションを使用し、高速リフレッシュされ、クエリー・リライトの対象となる part\_sales\_mv というマテリアライズド・ビューを作成しています。

```
CREATE MATERIALIZED VIEW part_sales_mv
PARALLEL
PARTITION BY RANGE (time_key)
(
  PARTITION time_key
    VALUES LESS THAN (TO_DATE('31-12-1997', 'DD-MM-YYYY'))
    PCTFREE 0 PCTUSED
    STORAGE (INITIAL 64k NEXT 16k PCTINCREASE 0)
    TABLESPACE sfl,
```

```

PARTITION month2
VALUES LESS THAN (TO_DATE('31-01-1998', 'DD-MM-YYYY'))
PCTFREE 0 PCTUSED
STORAGE INITIAL 64k NEXT 16k PCTINCREASE 0)
TABLESPACE sf2,
PARTITION month3
VALUES LESS THAN (TO_DATE('31-01-1998', 'DD-MM-YYYY'))
PCTFREE 0 PCTUSED
STORAGE (INITIAL 64k NEXT 16k PCTINCREASE 0)
TABLESPACE sf3)
BUILD DEFERRED
REFRESH FAST
ENABLE QUERY REWRITE
AS
SELECT f.store_key, f.time_key,
       SUM(f.dollar_sales) AS sum_dol_sales,
       SUM(f.unit_sales) AS sum_unit_sales
FROM fact f GROUP BY f.time_key, f.store_key;

```

## 事前作成表のパーティション化

また、マテリアライズド・ビューは、次に示すようにパーティション化された事前作成表に対して登録することもできます。

```

CREATE TABLE part_fact_tab(
    time_key, store_key, sum_dollar_sales,
    sum_unit_sale)
PARALLEL
PARTITION BY RANGE (time_key)
(
    PARTITION month1
    VALUES LESS THAN (TO_DATE('31-12-1997', 'DD-MM-YYYY'))
    PCTFREE 0 PCTUSED 99
    STORAGE (INITIAL 64k NEXT 16k PCTINCREASE 0)
    TABLESPACE sf1,
    PARTITION month2
    VALUES LESS THAN (TO_DATE('31-01-1998', 'DD-MM-YYYY'))
    PCTFREE 0 PCTUSED 99
    STORAGE (INITIAL 64k NEXT 16k PCTINCREASE 0)
    TABLESPACE sf2,
    PARTITION month3
    VALUES LESS THAN (TO_DATE('31-01-1998', 'DD-MM-YYYY'))
    PCTFREE 0 PCTUSED 99
    STORAGE (INITIAL 64k NEXT 16k PCTINCREASE 0)
    TABLESPACE sf3)
AS
SELECT f.time_key, f.store_key,

```

```
SUM(f.dollar_sales) AS sum_dollar_sales,
SUM(f.unit_sales)   AS sum_unit_sales
  FROM fact f GROUP BY f.time_key, f.store_key;

CREATE MATERIALIZED VIEW part_fact_tab
ON PREBUILT TABLE
ENABLE QUERY REWRITE
AS
SELECT f.time_key, f.store_key,
       SUM(f.dollar_sales) AS sum_dollar_sales,
       SUM(f.unit_sales)   AS sum_unit_sales
  FROM fact f  GROUP BY f.time_key, f.store_key;
```

この例では、*part\_fact\_tab* 表が 3 か月にパーティション化され、事前作成表を使用するマテリアライズド・ビューが登録されています。ENABLE QUERY REWRITE 句が含まれているので、このマテリアライズド・ビューはクエリー・リライトの対象になります。

## マテリアライズド・ビューに対する索引作成の選択

マテリアライズド・ビューに対する 2 つの主要操作は、問合せ実行と高速増分リフレッシュであり、各操作のパフォーマンス要件は異なります。高速増分リフレッシュは、マテリアライズド・ビューのキーの完全照合を実行する必要がある、マテリアライズド・ビューのすべてのキーを含む複合索引があるときに最もパフォーマンスがよくなります。一方、問合せ実行は、マテリアライズド・ビューのキー列のサブセットにアクセスする必要があり、これらの列のサブセットに対して結合や集計を行うことが必要な場合があります。したがって、問合せ実行は、マテリアライズド・ビューの各キー列に単一列ビットマップ索引が定義されている場合に最もパフォーマンスがよくなるのが一般的です。

マテリアライズド・ビューの索引作成の選択肢の 1 つとして、マテリアライズド・ビューのすべてのキーを含む一意のローカル索引を定義し、記憶領域とリフレッシュ時間が許容される場合にはマテリアライズド・ビューの各キーに対して単一列ビットマップ索引を作成する方法があります。

高速リフレッシュ・オプションを使用する、結合のみを含むマテリアライズド・ビューの場合は、ROWID を含む列に対して索引を作成し、リフレッシュ操作のパフォーマンスを改善することを強くお勧めします。

## マテリアライズド・ビューの無効化

マテリアライズド・ビューに関連する依存関係は、正しい操作を保証するために自動的に維持されます。DDL 時には、マテリアライズド・ビューはその定義で参照されているディテール表に依存します。

共有カーソルは、カーソルで参照されているすべてのオブジェクトに依存します。カーソルが書き直された場合、カーソルはクエリー・リライトによって選択されたマテリアライズド・ビューに依存し、クエリー・リライトでカーソルの表のディメンションが使用されてい

る場合はそれに依存します。これらのディメンションまたはマテリアライズド・ビューを無効にする操作は、カーソルも無効にします。

したがって、マテリアライズド・ビューの依存関係に対する DROP や ALTER などの DDL 操作は、マテリアライズド・ビューが無効になる原因となります。

マテリアライズド・ビューの状態は、表 USER\_MVIEW\_ANALYSIS または ALL\_MVIEW\_ANALYSIS を問い合わせることによってチェックできます。列 UNUSABLE は、Y または N の値を取り、マテリアライズド・ビューを使用できるかどうかを示します。列 KNOWN\_STALE も Y または N の値を取り、マテリアライズド・ビューが古いものかどうかをアドバイスします。また、最終的に、マテリアライズド・ビューが無効な場合は列 INVALID が Y に、無効でない場合は N に設定されます。

マテリアライズド・ビューは、参照されるたびに自動的に再検証されます。ただし、マテリアライズド・ビューで参照されている表で列が削除された場合、またはクエリー・リライト権限の 1 つを持っていなかったマテリアライズド・ビューの所有者がその権限を付与された場合は、次のコマンドを使用してマテリアライズド・ビューを再検証する必要があります。

```
ALTER MATERIALIZED VIEW mview_name ENABLE QUERY REWRITE
```

問題がある場合は、エラーが戻されます。

## セキュリティの問題

マテリアライズド・ビューを作成するには、権限 CREATE MATERIALIZED VIEW が必要です。また、別のスキーマの表を参照するマテリアライズド・ビューを作成するには、権限 CREATE ANY MATERIALIZED VIEW が必要です。

マテリアライズド・ビューをクエリー・リライトで使用する場合は、権限 QUERY REWRITE を付与する必要があります。または、マテリアライズド・ビューがスキーマにない表を参照する場合は、GLOBAL QUERY REWRITE を付与する必要があります。

マテリアライズド・ビューを作成しようとする権限エラーが発生し続け、必要な権限がすべて付与されているように思える場合、その最も多い原因は、権限が明示的に付与されずに、あるロールから継承されたものであることです。マテリアライズド・ビューの所有者は、参照される表に対する SELECT アクセス権を明示的に付与される必要があります。

## データ・ウェアハウスにおけるマテリアライズド・ビューの使用のガイドライン

パフォーマンスの向上に関して、どのマテリアライズド・ビューが最も有効かを判断するには、DBMS\_OLAP パッケージの分析ツールが役立ちます。具体的には、DBMS\_OLAP.RECOMMEND\_MV プロシージャをコールして、Oracle が統計とターゲット・データベースの使用状況に基づいて推奨するマテリアライズド・ビューのリストを参照します。このパッケージでは、現在のところ、複数の表に対する集計を持つマテリアライズド・ビューのみを推奨していることに注意してください。

Oracle 分析ツールに頼らずに独自のマテリアライズド・ビューを作成する場合は、次のガイドラインに従って最大のパフォーマンスを達成してください。

1. 同じ表に対して方法の異なる同じ GROUP BY 列を持つ複数のマテリアライズド・ビューを定義するかわりに、異なる方法をすべて含む単一のマテリアライズド・ビューを定義します。
2. マテリアライズド・ビューに集計方法 AVG(x) が含まれる場合は、COUNT(x) も含めて増分リフレッシュをサポートします。同様に、VARIANCE(x) または STDDEV(x) が存在する場合は、必ず COUNT(x) と SUM(x) も含めて、増分リフレッシュをサポートします。

## マテリアライズド・ビューの変更

マテリアライズド・ビューに対して行うことができる修正は、次の 3 種類のみです。

- リフレッシュ方法の変更
- 再コンパイル
- クエリー・リライトの使用可能化 / 使用不能化

その他すべての変更は、マテリアライズド・ビューを削除してから再作成することによって実行できます。

ALTER MATERIALIZED VIEW 文の COMPILE オプションは、29-20 ページの「[マテリアライズド・ビューの無効化](#)」で説明したように、マテリアライズド・ビューが無効にされたときに使用できます。このコンパイル処理は高速なので、マテリアライズド・ビューをクエリー・リライトで使用できます。

ALTER MATERIALIZED VIEW の詳細は、『Oracle8i SQL リファレンス』を参照してください。

## マテリアライズド・ビューの削除

マテリアライズド・ビューを削除するには、DROP MATERIALIZED VIEW コマンドを使用します。次に例を示します。

```
DROP MATERIALIZED VIEW sales_sum_mv;
```

このコマンドは、マテリアライズド・ビュー sales\_sum\_mv を削除します。事前作成された表を使用したマテリアライズド・ビューである場合、表は削除されませんが、リフレッシュ・メカニズムでメンテナンスできなくなります。



## ディメンション

この章のトピックは次のとおりです。

- データ・ウェアハウスのディメンション
- ディメンションの作成
- ディメンションの検証

### データ・ウェアハウスのディメンション

ディメンションは必ずしも定義する必要はありませんが、ディメンションによってクエリー・リライトがより複雑なタイプのリライトを実行できるようになるので、ディメンションの作成に時間を費やすことには大きな利益があります。アドバイザを使用して、作成、削除または保持するマテリアライズド・ビューを推奨する場合は、ディメンションが必須です。

業務プロセスとは組織内の運用プロセスであり、これに関してデータを収集できます。例として、ビデオ・チェーンの各店舗は、チェックアウト・カウンタでのビデオ・テープの売上とレンタルについて、データを収集および格納できます。ビデオ・チェーン管理では、データ・ウェアハウスを作成して、すべての店舗の製品の長期にわたる売上を分析し、次のような問いに答えることができます。

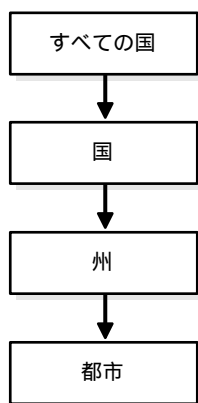
- ある製品の販売促進活動が、販売促進活動を実施していない関連製品の売上に与える影響。
- 販売促進活動の前後の製品の売上。

ビデオ・チェーンのデータ・ウェアハウス・システムのデータには、ディメンションおよびファクトという2つの重要なコンポーネントがあります。ディメンションは、製品、場所（店舗）、販売促進活動および時間です。ディメンションを設定する1つのアプローチは、製品に関するすべての情報を含む product 表や店舗に関するすべての情報を含む store 表などの、参照表を検討することです。ファクトは売上（販売またはレンタル数）と利益です。データ・ウェアハウスには、各店舗における各製品の毎日の売上に関するファクトが含まれます。

ディメンション値は、通常は階層に編成されます。階層レベルを上に移動することをデータのロールアップといい、階層レベルを下に移動することをデータのドリルダウンといいます。ビデオ・チェーン例では、次のことが行われます。

- 時間ディメンションでは、月が四半期に、四半期が年に、年がすべての年にロールアップされます。
- 製品ディメンションでは、製品がカテゴリに、カテゴリが部門に、部門が全部門にロールアップされます。
- 場所ディメンションでは、図 30-1 で示すように、店舗が都市に、都市が州に、州が地域に、地域が国に、国がすべての国にロールアップされます。

図 30-1 地理ディメンション



データ分析は、通常はディメンション階層の上位レベルから開始し、そのような分析が保証される状況であれば徐々にドリルダウンします。

業務プロセスのディメンションは、n ディメンションのデータ・キューブとしてビジュアル化できます。ビデオ・チェーン例では、業務ディメンションである製品、場所および時間をキューブの3つの軸で表現できます。製品軸の各単位はさまざまな製品を表し、場所軸の各単位は店舗を表します。また、時間軸の各単位は月を表します。これらの値の交差部は、販売数量や利益などのファクト情報を含むセルです。上位レベルの分析は、1998年第2四半期中のカリフォルニア店のコメディ・ビデオのレンタル量などの、サブキューブ内のファクト情報の選択および集計から構成されます。

したがって、ディメンションを作成する最初のステップでは、データ・ウェアハウス内のディメンションを明確化し、図 30-1 に示したような階層を描きます。たとえば、(都市レベルのデータは州に集約できるので)都市は州の子であり、州は地域の子です。このアプローチを使用すると、これを実際のディメンションに変換するのが簡単になります。

正規化または部分正規化されたディメンション（複数の表に格納されたディメンション）の場合は、これらの表がどのように結合されているかを明確にしてください。ディメンション表の間の結合によって、各子側の行が親側の 1 行のみと必ず結合されることを保証できるかどうかに注意してください。非正規化ディメンションの場合は、子側の列によって、親側（または属性）の列が一意に決定されるかどうかを判断します。制約によって表される関係を他の手段で保証できる場合は、NOVALIDATE オプションおよび RELY オプションを使用してこれらの制約を使用可能にできます。ファクト表とディメンション表の間の結合がこの関係をサポートしていない場合でも、CREATE DIMENSION 文を使用したディメンションの定義によってパフォーマンスを大幅に改善できることに注意してください。一定の制限の対象となる別の代替策として、マテリアライズド・ビュー定義で（つまり、CREATE MATERIALIZED VIEW 文で）外部結合を使用する方法があります

これらの関係を満たさないスキーマでは、問合せから間違った結果が戻される可能性があるので、ディメンションを作成しないでください。

## ディメンションの作成

ディメンションを作成する前に、このディメンション・データを含む表がデータベースに存在している必要があります。たとえば、「場所」というディメンションを作成する場合は、都市、州および国情報を含む 1 つ以上の表が存在している必要があります。データ・ウェアハウスでは、これらのディメンション表がすでに存在しています。したがって、どれを使用するかを判別するのは単純な作業です。

ディメンションは、CREATE DIMENSION 文を使用して作成します。CREATE DIMENSION 文では、LEVEL...IS 句を使用してディメンション・レベルの名前を指定します。

場所ディメンションには、子レベルから親レベルに矢印が引かれた単一の階層が含まれます。このディメンション・グラフの一番上には、すべての行の集計を表す特殊なレベル ALL があります。グラフの各矢印は、どの子に対しても親が必ず 1 つのみ存在することを示します。たとえば、各都市は 1 つの州にのみ含まれ、各州は 1 つの国にのみ含まれている必要があります。複数の国に属する州や、どの国にも属さない州は、階層整合性に違反します。階層整合性は、集計を含むマテリアライズド・ビューで管理機能を正しく操作するために必要です。

したがって、30-2 ページの図 30-1 で例として示したエンティティを使用して、レベル CITY、STATE および COUNTRY を含むディメンション LOCATION を宣言できます。

```
CREATE DIMENSION location_dim
  LEVEL city      IS location.city
  LEVEL state     IS location.state
  LEVEL country   IS location.country
```

ディメンションの図を使用して、ダイアグラムの各レベルを CREATE DIMENSION 文の LEVEL 句に変換します。したがって、都市、州、国という 3 つのレベルを定義します。さらに、ディメンションの各レベルは、データベース内の表の 1 つ以上の列に対応している必要があります。このため、レベル都市は location という表の列 city によって識別され、レベル国は同じ表の country という列によって識別されます。

この例では、データベース表は非正規化され、すべての列が同じ表に存在します。ただし、これはディメンション作成の前提条件ではありません。30-7 ページの「[正規化ディメンション表の使用方法](#)」に、JOIN KEY 句を使用して正規化スキーマ・デザインを持つディメンションを作成する方法を示します。

次のステップでは、HIERARCHY 文を使用してレベル間の関係を宣言し、階層に名前を割り当てます。階層関係とは、階層内のあるレベルから次のレベルへの機能的な依存関係です。前に定義したレベル名を使用すると、CHILD OF 関係は、各子のレベル値が 1 つの親レベル値のみに関連づけられていることを示します。ここでも 30-2 ページの図 30-1 のエンティティを使用すると、次の文は階層 LOC\_ROLLUP を宣言し、CITY、STATE および COUNTRY 間の関係を定義します。

```
HIERARCHY loc_rollup (
    city      CHILD OF
    state     CHILD OF
    country   )
```

1:n の階層関係に加えて、ディメンションには、階層レベルとその依存するディメンション属性の間の 1:1 の属性関係も含まれます。たとえば、列 *governor* と *mayor* がある場合は、ATTRIBUTE...DETERMINES 文は *governor* に対する *state*、および *mayor* に対する *city* になります。

この例では、*city* ではなく *mayor* によって問い合わせられた問合せが発行されたことを想定します。この 1 対 1 関係が属性とレベルの間に存在するので、*city* を使用してデータを識別できます。

```
ATTRIBUTE    city      DETERMINES    mayor
```

location 表の作成を含む、この完全なディメンション定義を次に示します。

```
CREATE TABLE location (
    city      VARCHAR2(30),
    state     VARCHAR2(30),
    country   VARCHAR2(30),
    mayor     VARCHAR2(30),
    governor  VARCHAR2(30) );

CREATE DIMENSION location_dim
    LEVEL city      IS location.city
    LEVEL state     IS location.state
    LEVEL country   IS location.country
HIERARCHY loc_rollup (
    city      CHILD OF
    state     CHILD OF
    country   )
ATTRIBUTE    city      DETERMINES    location.mayor
ATTRIBUTE    state     DETERMINES    location.governor;
```

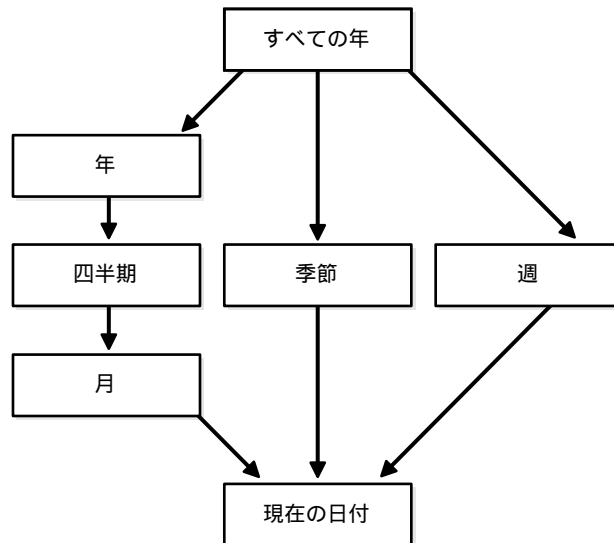
ディメンションの設計、作成およびメンテナンスは、データ・ウェアハウス・スキーマの設計、作成およびメンテナンスの一部です。ディメンションが作成された後で、次の要件を満たしているかどうかを確認します。

- **親と子の間に 1:n 関係が存在する必要があります。**親は複数の子を持つことができますが、子は 1 つの親しか持つことができません。
- **階層レベルとその依存するディメンション属性の間に、1:1 の属性関係が存在する必要があります。**たとえば、列 *corporation* がある場合、可能な属性関係は *corporation* から *president* です。
- **親レベルと子レベルの列が異なるリレーションにある場合、それらの間の接続には 1:n の結合関係も必要であります。**子表の各行は、必ず親表の 1 行のみと結合される必要があります。この関係では、子の結合キーが NULL 以外であること、子の結合キーから親の結合キーへの参照整合性が維持されていること、および親の結合キーが一意であることが必要なので、この関係は参照整合性のみよりも強力です。
- **(必要に応じてデータベース制約を使用して、) 各階層レベルの列が NULL 以外であり、階層整合性が維持されていることを確認します。**
- **ディメンションの階層は、相互にオーバーラップすることも切断することもあります。**ただし、階層レベルの列を複数のディメンションと関連付けることはできません。
- **ディメンション・グラフのサイクルを形成する結合関係はサポートされません。**たとえば、階層レベルは、直接的にも間接的にもそれ自体と結合することはできません。

## 複数の階層

次に示すように、単一のディメンション定義に複数の階層を含めることができます。あるデパートが、特定の品目の売上を長期にわたって追跡するとします。最初のステップでは、売上が追跡される時間ディメンションを定義します。30-6 ページの図 30-2 に、3 つの時間階層を持つ "時間" ディメンションを示します。

図 30-2 3 つの時間階層を持つ Time\_dim ディメンション



この図から、次の非正規化時間ディメンション文を組み立てることができます。関連する CREATE TABLE 文も示されています。

```

CREATE TABLE time (
    curDate    DATE,
    month      INTEGER,
    quarter    INTEGER,
    year       INTEGER,
    season     INTEGER,
    week_num   INTEGER,
    dayofweek  VARCHAR2(30),
    month_name VARCHAR2(30) );

CREATE DIMENSION time_dim
    LEVEL curDate    IS time.curDate
    LEVEL month      IS time.month
    LEVEL quarter    IS time.quarter
    LEVEL year       IS time.year
    LEVEL season     IS time.season
    LEVEL week_num   IS time.week_num

HIERARCHY calendar_rollup (

```

```

        curDate      CHILD OF
        month        CHILD OF
        quarter      CHILD OF
        year         )
HIERARCHY weekly_rollup (
        curDate      CHILD OF
        week_num     )
HIERARCHY seasonal_rollup (
        curDate      CHILD OF
        season       )
ATTRIBUTE curDate    DETERMINES time.dayofweek
ATTRIBUTE month      DETERMINES time.month_name;

```

## 正規化ディメンション表の使用方法

ディメンションの定義に使用される表は、正規化されている場合と非正規化されている場合があります。個々の階層は正規化または非正規化できます。階層のレベルが同じ表にある場合は、この階層を完全非正規化階層と呼びます。たとえば、時間ディメンションの CALENDAR\_ROLLUP は、非正規化階層です。階層のレベルが別の表のものである場合は、この階層を完全または部分正規化階層と呼びます。この項では、正規化階層の定義方法を説明します。

製品、ブランドおよび部門別に、製品の追跡を行うとします。このデータは、表 PRODUCT、BRAND および DEPARTMENT に格納されています。データ・エンティティ ITEM\_NAME、BRAND\_ID および DEPT\_ID は異なる表に属するので、製品ディメンションは非正規化です。ディメンション定義内の句 JOIN KEY 句は、階層内のレベルの結合方法を指定します。PRODUCT、BRAND および DEPARTMENT 表のディメンション文と、関連する CREATE TABLE 文を次に示します。

```

CREATE TABLE product (
    item_name  VARCHAR2(30),
    brand_id   INTEGER );

CREATE TABLE brand (
    brand_id   INTEGER,
    brand_name VARCHAR2(30),
    dept_id    INTEGER);

CREATE TABLE department (
    dept_id    INTEGER,
    dept_name  VARCHAR2(30),
    dept_type  INTEGER);

CREATE DIMENSION product_dim
    LEVEL item      IS product.item_name
    LEVEL brand_id  IS brand.brand_id
    LEVEL dept_id   IS department.dept_id

```

```
HIERARCHY merchandise_rollup
(
    item                CHILD OF
    brand_id            CHILD OF
    dept_id

    JOIN KEY product.brand_id REFERENCES brand_id
    JOIN KEY brand.dept_id    REFERENCES dept_id
)
ATTRIBUTE brand_id DETERMINES product.brand_name
ATTRIBUTE dept_id  DETERMINES (product.dept_name, product.dept_type);
```

## ディメンションの表示

定義されているディメンションを表示するプロシージャは2つあります。最初に、ファイル `smdim.sql` を実行して、次のものを含む `DEMO_DIM` パッケージを提供する必要があります。

- 特定のディメンションを出力する `DEMO_DIM.PRINT_DIM`
- すべてのディメンションを出力する `DEMO_DIM.PRINT_ALLDIMS`

`DEMO_DIM.PRINT_DIM` プロシージャのパラメータは、表示するディメンション名のみです。次の例で、ディメンション `TIME_PD` を表示する方法を示します。

```
DEMO_DIM.PRINT_DIM ('TIME_PD');
```

定義されているすべてのディメンションを表示するには、次に示すように、パラメータを指定しないでプロシージャ `DEMO_DIM.PRINT_ALLDIMS` をコールします。

```
DEMO_DIM.PRINT_ALLDIMS ();
```

どちらのプロシージャをコールするかにかかわらず、出力は同一になります。サンプルの表示を次に示します。

```
DIMENSION GROCERY.TIME_PD
LEVEL FISCAL_QTR IS GROCERY.WEEK.FISCAL_QTR
LEVEL MONTH IS GROCERY.MONTH.MONTH
LEVEL QUARTER IS GROCERY.QUARTER.QUARTER
LEVEL TIME_KEY IS GROCERY.TIME.TIME_KEY
LEVEL WEEK IS GROCERY.WEEK.WEEK
LEVEL YEAR IS GROCERY.YEAR.YEAR
HIERARCHY WEEKLY_ROLLUP (
    TIME_KEY
    CHILD OF WEEK
    JOIN KEY GROCERY.TIME.WEEK REFERENCES WEEK
)
HIERARCHY FISCAL_ROLLUP (
```



```

        TIME_KEY
        CHILD OF WEEK
        CHILD OF FISCAL_QTR
    JOIN KEY GROCERY.TIME.WEEK REFERENCES WEEK
    )
HIERARCHY CALENDAR_ROLLUP (
    TIME_KEY
    CHILD OF MONTH
    CHILD OF QUARTER
    CHILD OF YEAR
    JOIN KEY GROCERY.TIME.MONTH REFERENCES MONTH
    JOIN KEY GROCERY.MONTH.QUARTER REFERENCES QUARTER
    JOIN KEY GROCERY.QUARTER.YEAR REFERENCES YEAR
)

ATTRIBUTE TIME_KEY DETERMINES GROCERY.TIME.DAY_NUMBER_IN_MONTH
ATTRIBUTE TIME_KEY DETERMINES GROCERY.TIME.DAY_NUMBER_IN_YEAR
ATTRIBUTE WEEK DETERMINES GROCERY.WEEK.WEEK_NUMBER_OF_YEAR
ATTRIBUTE MONTH DETERMINES GROCERY.MONTH.FULL_MONTH_NAME

```

## ディメンションと制約

ディメンションにおける重要なロールの制約を計画します。たいていの場合、運用データベースに対して完全な参照整合性が施行され、(データのクリーン化の後に) データ・ウェアハウスへのデータのフローが参照整合性に決して違反しないことは運用手順を使用して保証できますが、実際には、参照整合性制約はデータ・ウェアハウスで使用可能にされる場合と使用可能にされない場合があります。

制約を使用可能にし、検証時間を考慮する場合は、次に示すように NOVALIDATE 句を使用することをお勧めします。前述したように、主キーと外部キーをインプリメントする必要があります。ファクト表に対する参照整合性制約と NOT NULL 制約は、クエリー・リライトがマテリアライズド・ビューの有用性を拡張するために使用できる情報を提供します。

```
ENABLE NOVALIDATE CONSTRAINT pk_time;
```

また、制約に依存するクエリー・リライトを行うため、次に示すように RELY 句を使用する必要があります。

```
ALTER TABLE time MODIFY CONSTRAINT pk_time RELY;
```

## ディメンションの検証

ディメンションによって表される関係が正しくない場合は、間違った結果が発生することがあります。したがって、DBMS\_OLAP.VALIDATE\_DIMENSION プロシージャを定期的に使用して、CREATE DIMENSION によって指定された関係を検証する必要があります。

このプロシージャは、使用方法が簡単で、パラメータは 4 つしかありません。

- ディメンション名
- 所有者名
- このディメンションの表の新規行のみをチェックする場合は TRUE を設定
- すべての列が NULL 以外であることを検証するには TRUE を設定

次に示す例は、Grocery スキーマのディメンション time\_fn を検証します。

```
DBMS_OLAP.VALIDATE_DIMENSION ('TIME_FN', 'GROCERY', FALSE, TRUE);
```

プロシージャ VALIDATE\_DIMENSION によって検出されたすべての例外は、ユーザーのスキーマに作成される表 MVIEW\$\_EXCEPTIONS に入れます。この表に対する問合せで発見された例外を識別します。次に例を示します。

OWNER	TABLE_NAME	DIMENSION_NAME	RELATIONSHIP	BAD_ROWID
GROCERY	MONTH	TIME_FN	FOREIGN KEY	AAAAuWAAJAAAArWAAA

ただし、この表を問い合わせるよりは、次のように、無効な行の ROWID を使用して制約違反のある実際の行を検索する問合せを行う方がよい場合があります。この例では、ディメンション TIME\_FD が month という表をチェックしています。これによって制約に違反している行が見つかり、ROWID を使用して、month 表内のどの行が問題の原因となっているかを正確に調べることができます。

```
SELECT * FROM month
WHERE rowid IN (select bad_rowid from mview$_exceptions);
```

MONTH	QUARTER	FISCAL_QTR	YEAR	FULL_MONTH_NAME	MONTH_NUMB
199903	19981	19981	1998	March	3

## ディメンションの変更

ALTER DIMENSION 文を使用して、ディメンションにいくつかの変更を加えることができます。このコマンドを使用して、ディメンションに対してレベル、階層または属性の追加や削除を行うことができます。

[図 30-2](#) の時間ディメンションを参照すると、属性 month の削除、階層 weekly\_rollup の削除およびレベル week の削除を行うことができます。また、qtr1 という新しいレベルを追加できます。

```
ALTER DIMENSION time_dim DROP attribute month;
ALTER DIMENSION time_dim DROP hierarchy weekly_rollup;
ALTER DIMENSION time_dim DROP LEVEL week;
ALTER DIMENSION time_dim ADD LEVEL qtr1 IS time.fiscal_qtr;
```

ディメンションが参照しているスキーマ・オブジェクトを変更すると、ディメンションは無効になります。たとえば、ディメンションが定義された表が変更された場合などです。

ディメンションのステータスをチェックするには、表 ALL\_DIMENSIONS の列 *invalid* の内容を表示します。

ディメンションを再検証するには、次に示すように COMPILE オプションを使用します。

```
ALTER DIMENSION time_dim COMPILE;
```

## ディメンションの削除

ディメンションは、DROP DIMENSION コマンドを使用して削除します。次に例を示します。

```
DROP DIMENSION time_dim;
```



---

## クエリー・リライト

この章のトピックは次のとおりです。

- [クエリー・リライトの概要](#)
- [コストベースのリライト](#)
- [クエリー・リライトの使用可能化](#)
- [Oracle が問合せをリライトするとき](#)
- [クエリー・リライトの方法](#)
- [制約とディメンションが必要になるとき](#)
- [クエリー・リライトの正確性](#)
- [クエリー・リライトが行われたかどうか](#)
- [クエリー・リライトの使用のガイドライン](#)

### クエリー・リライトの概要

マテリアライズド・ビューを作成およびメンテナンスすることの主要な利点の 1 つは、クエリー・リライトを利用できることです。クエリー・リライトは、表やビューに対する SQL 文を、ディテール表で定義されている 1 つ以上のマテリアライズド・ビューにアクセスする文に変換します。変換はエンド・ユーザーやアプリケーションに対して透過的であり、手動操作や SQL 文でのマテリアライズド・ビューへの参照は必要ありません。クエリー・リライトは透過的なので、マテリアライズド・ビューは、索引と同じように、アプリケーション・コード内の SQL を無効にしないで追加または削除できます。

問合せは、リライトされる前に、クエリー・リライトの候補であるかどうかを判断するためのチェックがいくつか行われます。問合せがいずれかのチェックに適合しなかった場合、その問合せはマテリアライズド・ビューではなくディテール表に適用されます。これには、応答時間と処理能力の点でコストがかかることがあります。

Oracle オプティマイザは、2 つの異なる方法を使用して、1 つ以上のマテリアライズド・ビューについて問合せをいつリライトするかを認識します。第 1 の方法は、問合せの SQL テキストとマテリアライズド・ビュー定義の SQL テキストの照合に基づきます。第 1 の方法が失敗した場合、オプティマイザは、問合せとマテリアライズド・ビュー間で結合条件、データ列、グループ化列および集計関数を比較する、より一般的な方法を使用します。

クエリー・リライトは、次のタイプの SQL 文の問合せと副問合せに対して機能します。

- SELECT
- CREATE TABLE ... AS SELECT
- INSERT INTO ... SELECT

また、集合演算子 UNION、UNION ALL、INTERSECT および MINUS の副問合せに対して機能します。

特定の問合せが 1 つ以上のマテリアライズド・ビューを使用するようにリライトされるかどうかには、いくつかの要因が影響します。

- クエリー・リライトの使用可能化 / 使用禁止
  - 個々のマテリアライズド・ビューに対する CREATE または ALTER 文による
  - 初期化パラメータ QUERY\_REWRITE\_ENABLED による
  - SQL 文の REWRITE および NOREWRITE ヒントによる
- リライトの整合性レベル
- ディメンションと制約

## コストベースのリライト

クエリー・リライトは、コストベースの最適化で使用できます。Oracle は、リライトを使用した場合と使用しない場合の両方について入力問合せの最適化を行い、最もコストの低いものを選択します。オプティマイザは、一度に 1 つ以上の問合せブロックをリライトすることによって、問合せをリライトします。

問合せブロックをリライトするためのマテリアライズド・ビューの選択肢が複数ある場合、リライト・ロジックはその 1 つを選択して、元の問合せブロック内の表の合計カーディナリティに対するリライトされた問合せブロック内の表の合計カーディナリティの比率を最適化します。したがって、選択されるマテリアライズド・ビューは、読み込むデータ量が最も少なくなるマテリアライズド・ビューです。

リライトに使用するマテリアライズド・ビューが選択された後、オプティマイザは、リライトを実行してから、リライトされた問合せを別のマテリアライズド・ビューでさらにリライトできるかどうかをテストします。このプロセスは、それ以上リライトできなくなるまで継続されます。次に、リライトされた問合せが最適化され、元の問合せが最適化されます。オプティマイザは、これらの 2 つの最適化を比較し、コストの少ない方を選択します。

最適化はコストに基づくので、問合せに関係する表とマテリアライズド・ビューを表す表の両方に基づいて統計を収集することが重要です。表の行数などの統計は、(リライトされた) 問合せのコストの計算に使用される基本的な基準です。これらは、ANALYZE 文または DBMS\_STATISTICS パッケージを使用して作成されます。

インライン・ビューや名前付きビューを含む問合せも、クエリー・リライトの候補になります。問合せに名前付きビューが含まれる場合、ビュー名はマテリアライズド・ビューと問合せ間の照合に使用されます。つまり、マテリアライズド・ビュー定義内の一連の名前付きビューは、問合せ内の一連のビューと完全に一致している必要があります。問合せにインライン・ビューが含まれる場合は、マテリアライズド・ビューと問合せ間の照合が行われる前に、インライン・ビューが問合せにマージされることがあります。

## クエリー・リライトの使用可能化

クエリー・リライトを使用可能にするには、いくつかのステップに従う必要があります。

1. 個々のマテリアライズド・ビューには、ENABLE QUERY REWRITE 句が必要です。
2. 初期化パラメータ QUERY\_REWRITE\_ENABLED は、TRUE に設定する必要があります。
3. 初期化パラメータ OPTIMIZER\_MODE を "ALL\_ROWS" または "FIRST\_ROWS" に設定するか、表を分析して OPTIMIZER\_MODE を "CHOOSE" に設定することによって、コストベースの最適化を使用する必要があります。

ステップ 1 が完了していないと、マテリアライズド・ビューはクエリー・リライトの対象になりません。ENABLE QUERY REWRITE は、次に示すようにマテリアライズド・ビューが作成されるときに指定するか、ALTER MATERIALIZED VIEW 文を介して指定できます。

```
CREATE MATERIALIZED VIEW store_sales_mv
  ENABLE QUERY REWRITE
  AS
  SELECT s.store_name,
         SUM(dollar_sales) AS sum_dollar_sales
  FROM store s, fact f
  WHERE f.store_key = s.store_key
  GROUP BY s.store_name;
```

初期化パラメータ QUERY\_REWRITE\_ENABLED を使用して、すべてのマテリアライズド・ビューに対してクエリー・リライトを使用禁止にしたり、個別に使用可能にされているすべてのマテリアライズド・ビューに対してクエリー・リライトを再度使用可能にしたりできます。ただし、QUERY\_REWRITE\_ENABLED パラメータでは、CREATE または ALTER 文で使用禁止にされたマテリアライズド・ビューに対してクエリー・リライトを使用可能にすることはできません。

NOREWRITE ヒントは、QUERY\_REWRITE\_ENABLED パラメータを上書きして SQL 文のクエリー・リライトを使用禁止にします。REWRITE (mview\_name, ...) ヒントは、対象となるマテリアライズド・ビューをヒントで指定されているもののみに制限します。

## クエリー・リライトの初期化パラメータ

クエリー・リライトには、次の初期化パラメータ設定が必要です。

- OPTIMIZER\_MODE = "ALL\_ROWS"、"FIRST\_ROWS" または "CHOOSE"
- QUERY\_REWRITE\_ENABLED = TRUE
- COMPATIBLE = 8.1.0 (またはそれ以上)

QUERY\_REWRITE\_INTEGRITY パラメータはオプションですが、これを指定する場合は、STALE\_TOLERATED、TRUSTED または ENFORCED に設定する必要があります (31-17 ページの「[クエリー・リライトの正確性](#)」を参照してください)。未定義の場合のデフォルトは ENFORCED です。

## クエリー・リライトの使用可能化の権限

マテリアライズド・ビューは、ユーザーがそのマテリアライズド・ビューに対して持っている権限ではなく、ユーザーが問合せのディテール表またはビューに対して持っている権限に基づきます。

システム権限 QUERY REWRITE を使用すると、マテリアライズド・ビューで直接参照されているすべての表がユーザー固有のスキーマにある場合のみ、そのスキーマでマテリアライズド・ビューに対してクエリー・リライトを使用可能にできます。GLOBAL QUERY REWRITE 権限を使用すると、マテリアライズド・ビューが他のスキーマのオブジェクトを参照している場合でも、マテリアライズド・ビューに対してクエリー・リライトを使用可能にできます。

クエリー・リライトでマテリアライズド・ビューを使用するための権限は、定義者権限プロシージャに対する権限と似ています。詳細は、『Oracle8i 概要』を参照してください。

## Oracle が問合せをリライトするとき

問合せは、一定数の条件が満たされたときのみリライトされます。

1. セッションに対してクエリー・リライトが使用可能にされている必要があります。
2. クエリー・リライトに対してマテリアライズド・ビューが使用可能にされている必要があります。
3. リライトの整合性レベルで、マテリアライズド・ビューの使用を許可する必要があります。たとえば、あるマテリアライズド・ビューが古く、クエリー・リライトの整合性が ENFORCED に設定されている場合、そのマテリアライズド・ビューは使用されません。
4. 問合せによって要求される結果のすべてまたは一部が、マテリアライズド・ビューに格納されている事前計算結果から取得可能である必要があります。



これを判断するために、オプティマイザは、ユーザーが制約とディメンションを介して宣言したいくつかのデータ関係に依存します。このようなデータ関係には、階層、参照整合性、キー・データの一意性などが含まれます。

次の項では、スキーマ例といくつかのマテリアライズド・ビューを使用して、オプティマイザが問合せをリライトするためにデータ関係をどのように使用するかを示します。小売りデータベースは、次の表から構成されます。

```
STORE    (store_key, store_name, store_city, store_state, store_country)
PRODUCT  (prod_key, prod_name, prod_brand)
TIME     (time_key, time_day, time_week, time_month)
FACT     (store_key, prod_key, time_key, dollar_sales)
```

これらの表で作成されている 2 つのマテリアライズド・ビューには、結合のみ含まれます。

```
CREATE MATERIALIZED VIEW join_fact_store_time
  ENABLE QUERY REWRITE
  AS
  SELECT s.store_key, s.store_name, f.dollar_sales, t.time_key, t.time_day,
         f.prod_key, f.rowid, t.rowid
  FROM   fact f, store s, time t
  WHERE  f.time_key = t.time_key AND f.store_key = s.store_key;
```

```
CREATE MATERIALIZED VIEW join_fact_store_time_oj
  ENABLE QUERY REWRITE
  AS
  SELECT s.store_key, s.store_name, f.dollar_sales, t.time_key,
         f.rowid, t.rowid
  FROM   fact f, store s, time t
  WHERE  f.time_key = t.time_key(+) AND f.store_key = s.store_key(+);
```

また、2 つのマテリアライズド・ビューには、結合と集計が含まれます。

```
CREATE MATERIALIZED VIEW sum_fact_store_time_prod
  ENABLE QUERY REWRITE
  AS
  SELECT s.store_name, time_week, p.prod_key,
         SUM(f.dollar_sales) AS sum_sales,
         COUNT(f.dollar_sales) AS count_sales
  FROM   fact f, store s, time t, product p
  WHERE  f.time_key = t.time_key AND f.store_key = s.store_key AND
         f.prod_key = p.prod_key
  GROUP BY s.store_name, time_week, p.prod_key;
```

```
CREATE MATERIALIZED VIEW sum_fact_store_prod
  ENABLE QUERY REWRITE
  AS
  SELECT s.store_city, p.prod_name
```

```
SUM(f.dollar_sales) AS sum_sales,  
COUNT(f.dollar_sales) AS count_sales  
FROM fact f, store s, product p  
WHERE f.store_key = s.store_key AND f.prod_key = p.prod_key  
GROUP BY store_city, p.prod_name;
```

マテリアライズド・ビューに対して統計を計算し、オブティマイザがコストに基づいて問合せをリライトするかどうかを判断できるようにする必要があります。

```
ANALYZE TABLE join_fact_store_time COMPUTE STATISTICS;  
ANALYZE TABLE join_fact_store_time_oj COMPUTE STATISTICS;  
ANALYZE TABLE sum_fact_store_time_prod COMPUTE STATISTICS;  
ANALYZE TABLE sum_fact_store_prod COMPUTE STATISTICS;
```

## クエリー・リライトの方法

オブティマイザは、いくつかの異なる方法を使用して問合せをリライトします。最初の最も重要なステップは、問合せによって要求される結果のすべてまたは一部が、マテリアライズド・ビューに格納されている事前計算結果から取得可能かどうかを判断することです。

最も単純なケースは、マテリアライズド・ビューに格納されている結果が、問合せによって要求される結果と完全に一致する場合です。Oracle オブティマイザは、問合せの SQL テキストをマテリアライズド・ビュー定義の SQL テキストと比較することによって、このタイプの判断を行います。この方法は、最も単純ですが、非常に制限があります。

SQL テキストの比較テストに失敗すると、Oracle オブティマイザは、結合、グループ化、集計およびフェッチされた列データに基づいて、一連の汎用チェックを実行します。このチェックは、問合せのさまざまな句 (SELECT、FROM、WHERE、GROUP BY) をマテリアライズド・ビューの句と個別に比較することによって実行されます。

## SQL テキスト照合リライト手法

オブティマイザでは、2 つの方法が使用されます。

1. 完全 SQL テキスト照合
2. 部分 SQL テキスト照合

完全 SQL テキスト照合では、問合せの SQL テキスト全体が、マテリアライズド・ビュー定義の SQL テキスト全体 (つまり、SELECT 式全体) と比較されます。SQL テキストの比較では、空白が無視されます。次の問合せは、

```
SELECT s.store_name, time_week, p.prod_key,  
SUM(f.dollar_sales) AS sum_sales,  
COUNT(f.dollar_sales) AS count_sales  
FROM fact f, store s, time t, product p  
WHERE f.time_key = t.time_key AND  
f.store_key = s.store_key AND
```

```
f.prod_key = p.prod_key
GROUP BY s.store_name, time_week, p.prod_key;
```

`sum_fact_store_time_prod` (空白は除外) と一致し、次のようにリライトされます。

```
SELECT store_name, time_week, product_key, sum_sales, count_sales
FROM   sum_fact_store_time_prod;
```

完全 SQL テキスト照合に失敗すると、オブティマイザは、部分 SQL テキスト照合を試行します。この方法では、問合せの FROM 句から開始する SQL テキストが、マテリアライズド・ビュー定義の FROM 句から開始する SQL テキストと比較されます。したがって、次の問合せは、

```
SELECT s.store_name, time_week, p.prod_key,
       AVG(f.dollar_sales) AS avg_sales
FROM   fact f, store s, time t, product p
WHERE  f.time_key = t.time_key AND
       f.store_key = s.store_key AND
       f.prod_key = p.prod_key
GROUP BY s.store_name, time_week, p.prod_key;
```

次のようにリライトされます。

```
SELECT store_name, time_week, prod_key, sum_sales/count_sales AS avg_sales
FROM   sum_fact_store_time_prod;
```

部分 SQL テキスト照合リライト手法では、問合せによって要求される売上集計の平均が、マテリアライズド・ビューに格納されている売上合計と売上集計件数を使用して計算されることに注意してください。

どちらの SQL テキスト照合も成功しない場合、オブティマイザは一般クエリー・リライト手法を使用します。

## 一般クエリー・リライト手法

一般クエリー・リライト手法は、マテリアライズド・ビューに問合せで要求されるデータの一部しか含まれていない場合、問合せで要求されているよりも多くのデータが含まれている場合、または問合せで要求されている形式に変換できる別の形式でデータが含まれている場合でもマテリアライズド・ビューを使用可能にできるので、SQL テキスト照合方法よりもはるかに強力です。これを実行するために、オブティマイザは、問合せとマテリアライズド・ビューの間で SQL 句 (SELECT、FROM、WHERE、GROUP BY) を個別に比較します。

Oracle オブティマイザは、次の 4 種類のチェックを採用します。

- 結合互換性
- データ充足
- グループ化互換性

■ 集計可能性

次の表に示すように、マテリアライズド・ビューを使用して問合せをリライトできるかどうかを判断するために、マテリアライズド・ビューのタイプに応じて 4 つのチェックの一部またはすべてが実行されます。

表 31-1 マテリアライズド・ビューのタイプと一般クエリー・リライト手法

	結合のみの MV	結合と集計のある MV	単一表に対する集計の ある MV
結合互換性	X	X	-
データ充足	X	X	X
グループ化互換性	-	X	X
集計可能性	-	X	X

これらのチェックを実行するために、オプティマイザは、使用できるデータ関係を使用します。たとえば、主キーと外部キーの関係は、外部キー表の各行が主キー表の 2 行以上とは結合されないことをオプティマイザに示します。さらに、外部キーに NOT NULL 制約がある場合は、外部キー表の各行が、主キー表の 1 行のみと必ず結合されることを示します。

これらのデータ関係は、データの結合、グループ化または集計によって生成される結果のタイプを示すので、クエリー・リライトでは非常に重要です。したがって、このようなデータ関係がデータベースに存在するときに、大部分問合せのリライトの可能性を最大化するには、ユーザーが宣言する必要があります。

結合互換性チェック

このチェックでは、問合せ内の結合がマテリアライズド・ビュー内の結合と比較されます。一般に、この比較の結果として、結合が 3 つのカテゴリに分類されます。

- 1. 問合せとマテリアライズド・ビューの両方に共通する結合
- 2. 問合せにはあり、マテリアライズド・ビューにはないデルタ結合
- 3. マテリアライズド・ビューにはあり、問合せにはないデルタ結合

**共通の結合** 2 つの間で共通する結合の対が同じタイプであるか、問合せ内の結合がマテリアライズド・ビュー内の結合から導出される必要があります。たとえば、マテリアライズド・ビューに表 A と表 B の外部結合が含まれ、問合せに表 A と表 B の結合が含まれる場合、結合の結果は、外部結合の結果から結合されない行をフィルタリングすることによって導出できます。

たとえば、次の問合せについて考えます。

```
SELECT s.store_name, t.time_day, SUM(f.dollar_sales)
FROM   fact f, store s, time t
```

```
WHERE f.time_key = t.time_key AND
      f.store_key = s.store_key AND
      t.time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP BY s.store_name, t.time_day;
```

この問合せとマテリアライズド・ビュー *join\_fact\_store\_time* の間で共通の結合は、次のとおりです。

```
f.time_key = t.time_key AND f.store_key = s.store_key
```

これらは完全に一致し、問合せを次のようにリライトできます。

```
SELECT store_name, time_day, SUM(dollar_sales)
FROM   join_fact_store_time
WHERE  time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP BY store_name, time_day;
```

問合せは、問合せの内部結合をマテリアライズド・ビューの外部結合から導出できるようにマテリアライズド・ビューを *join\_fact\_store\_time\_oj* 使用して回答することもできます。リライトされたバージョンは、(ユーザーに対して透過的に) 結合されない行をフィルタリングします。リライトされた問合せの構造は、次のようになります。

```
SELECT store_name, time_day, SUM(f.dollar_sales)
FROM   join_fact_store_time_oj
WHERE  time_key IS NOT NULL AND store_key IS NOT NULL AND
      time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP BY store_name, time_day;
```

一般に、結合のみを含むマテリアライズド・ビューで外部結合を使用する場合は、マテリアライズド・ビューで外部結合の右側に主キーまたは ROWID を入れる必要があります。たとえば、前述の例の *join\_fact\_store\_time\_oj* では、*store* と *time* の両方に主キーがあります。

結合のみを含むマテリアライズド・ビューが使用されるとき別の例として、セミ・ジョイン・リライトのケースがあります。つまり、問合せには、単一表の EXISTS または IN 副問合せが含まれます。

1997 年のクリスマス・シーズンの売上が \$10,000 を超える店舗をレポートする次の問合せについて考えます。

```
SELECT store_name
FROM   store s
WHERE  EXISTS (SELECT *
               FROM fact f
               WHERE f.store_key = s.store_key
                  AND f.dollar_sales > 10000);
```

この問合せは、次のようにもなります。

```
SELECT store_name
FROM   store s
```

```
WHERE s.store_key in (SELECT f.store_key
                      FROM fact f
                      WHERE f.dollar_sales > 10000
                      and f.time_key between '01-DEC-1997' and '31-DEC-1997');
```

問合せには、store 表と fact 表の間のセミ・ジョイン 'f.store\_key = s.store\_key' が含まれます。この問合せは、外部キー制約が有効な場合は *join\_fact\_store\_time* マテリアライズド・ビューを使用してリライトでき、主キーが有効な場合は *join\_fact\_store\_time\_oj* マテリアライズド・ビューを使用してリライトできます。両方のマテリアライズド・ビューに、問合せのセミ・ジョインを導出するために使用できる 'f.store\_key = s.store\_key' が含まれていることに注目してください。

問合せは、*join\_fact\_store\_time* を使用して次のようにリライトされます。

```
SELECT store_name
FROM (SELECT DISTINCT store_name, store_key
      FROM join_fact_store_time
      WHERE dollar_sales > 10000
      AND f.time_key BETWEEN '01-DEC-1997' and '31-DEC-1997');
```

*store* と *fact* 間の元の結合が回避されているので、マテリアライズド・ビュー *join\_fact\_store\_time* が *time\_key* によってパーティション化されている場合は、この問合せは元の間合せよりも効率がよくなる可能性が高くなります。

問合せは、*join\_fact\_store\_time\_oj* を使用して次のようにリライトできます。

```
SELECT store_name
FROM (SELECT DISTINCT store_name, store_key
      FROM join_fact_store_time_oj
      WHERE dollar_sales > 10000
      AND store_key IS NOT NULL
      AND time_key BETWEEN '01-DEC-1997' and '31-DEC-1997');
```

セミ・ジョインを使用したリライトは、現在は結合のみのマテリアライズド・ビューに制限されており、結合と集計のあるマテリアライズド・ビューでは使用できません。

**問合せデルタ結合** 問合せデルタ結合は、問合せにはあり、マテリアライズド・ビューにはない結合です。問合せでは、任意の数および任意のタイプのデルタ結合が許可され、問合せがマテリアライズド・ビューを使用してリライトされるときは、そのまま単純に保持されます。リライト時に、マテリアライズド・ビューは、デルタ結合の適切な表と結合されます。

たとえば、次の問合せについて考えます。

```
SELECT store_name, prod_name, SUM(f.dollar_sales)
FROM   fact f, store s, time t, product p
WHERE  f.time_key = t.time_key AND
       f.store_key = s.store_key AND
       f.prod_key = p.prod_key AND
       t.time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
```

```
GROUP BY store_name, prod_name;
```

マテリアライズド・ビュー *join\_fact\_store\_time* を使用すると、共通の結合は *f.time\_key = t.time\_key AND f.store\_key = s.store\_key* です。問合せのデルタ結合は、*f.prod\_key = p.prod\_key* です。

リライトされた形式は、*join\_fact\_store\_time* マテリアライズド・ビューと *product* 表を結合します。

```
SELECT store_name, prod_name, SUM(f.dollar_sales)
FROM   join_fact_store_time mv, product p
WHERE  mv.prod_key = p.prod_key AND
       mv.time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP BY store_name, prod_name;
```

**マテリアライズド・ビュー・デルタ結合** マテリアライズド・ビューのすべてのデルタ結合は、共通の結合の結果に関して可逆式であることが必要です。可逆式結合は、共通の結合の結果が制限されないことを保証します。可逆式結合は、A および B という 2 つの表が結合される場合に、表 A の行が常に表 B の行と一致し、データが失われない結合なので、可逆式結合と呼ばれています。たとえば、外部キーで NULL が許可されていない場合は、外部キーを持つすべての行が主キーを持つ行と一致します。したがって、可逆式結合を保証するには、該当する結合キーに対して FOREIGN KEY、PRIMARY KEY および NOT NULL 制約が必要です。また、表 A と B の間の結合が外部結合（A が外部表）の場合は、表 A のすべての行が保持されるので、これは可逆式です。

マテリアライズド・ビューのすべてのデルタ結合は、共通の結合の結果に関して重複していません。非重複結合は、共通の結合の結果が重複しないことを保証します。たとえば、非重複結合は、表 A と表 B が結合された場合に表 A の行が表 B の 2 行以上とは一致せず、重複が発生しない結合です。非重複結合を保証するには、主キーまたは一意制約を使用して、表 B のキーを一意的に値に制約する必要があります。

FACT と TIME を結合する次の問合せについて考えます。

```
SELECT t.time_day, sum(f.dollar_sales)
FROM   fact f, time t
WHERE  f.time_key = t.time_key AND
       t.time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP  t.time_day;
```

マテリアライズド・ビュー *join\_fact\_store\_time* には、fact と store 間の追加結合 '*f.store\_key = s.store\_key*' があります。これは、*join\_fact\_store\_time* のデルタ結合です。

この結合が可逆式および非重複の場合は、問合せをリライトできます。これは、*f.store\_key* が *p.store\_key* に対する外部キーで、NULL 以外の場合に当てはまります。したがって、問合せは次のようにリライトされます。

```
SELECT time_day, SUM(f.dollar_sales)
FROM   join_fact_store_time
```

```
WHERE time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
```

```
GROUP BY time_day;
```

問合せは、マテリアライズド・ビュー *join\_fact\_store\_time\_oj* を使用してリライトすることもでき、この場合は外部キー制約が不要です。このビューには、結合を可逆式にする fact と store 間の外部結合 'f.store\_key = s.store\_key(+)' が含まれます。s.store\_key が主キーの場合は、非重複条件も満たされ、オプティマイザは問合せを次のようにリライトします。

```
SELECT time_day, SUM(f.dollar_sales)
FROM   join_fact_store_time_oj
WHERE  time_key IS NOT NULL AND
       time_day BETWEEN '01-DEC-1997' AND '31-DEC-1997'
GROUP BY time_day;
```

現在の制限事項によって、外部結合を使用したリライトのほとんどは、結合のみのマテリアライズド・ビューに限定されます。外部結合のある集計を含むビューでは、非常に限られたリライトしかサポートされません。これらのビューは、マテリアライズド・ビュー・デルタ結合の可逆性を保証するために、外部キー制約に依存する必要があります。

## データ充足チェック

このチェックでは、オプティマイザは、問合せによって要求される必要な列データをマテリアライズド・ビューから取得できるかどうかを判断します。このためには、ある列と別の列の同等化が使用されます。たとえば、表 A と表 B の間の内部結合が結合述語  $A.X = B.X$  に基づいている場合、結合の結果の列 A.X のデータと列 B.X のデータは同等です。このデータ・プロパティは、問合せの列 A.X をマテリアライズド・ビューの列 B.X と照合するため、またはマテリアライズド・ビューの列 A.X を問合せの列 B.X と照合するために使用されます。

たとえば、次の問合せについて考えます。

```
SELECT s.store_name, f.time_key, SUM(f.dollar_sales)
FROM   fact f, store s, time t
WHERE  f.time_key = t.time_key AND
       f.store_key = s.store_key
GROUP BY s.store_name, f.time_key;
```

この問合せは、マテリアライズド・ビューに f.time\_key がない場合でも、*join\_fact\_store\_time* を使用して回答できます。かわりに、t.time\_key は、結合条件 'f.time\_key = t.time\_key' を介して f.time\_key と同等になります。

したがって、オプティマイザは次のリライトを選択します。

```
SELECT store_name, time_day, SUM(dollar_sales)
FROM   join_fact_store_time
GROUP BY store_name, time_key;
```

問合せによって要求される列データのいくつかがマテリアライズド・ビューから取得できない場合、オプティマイザは、機能依存性と呼ばれるデータ関係に基づいて取得できるかどうかをさらに判断します。列のデータによって別の列のデータを決定できる場合、このような



関係を機能依存性または機能決定性といいます。たとえば、表に *prod\_key* という主キー列および *prod\_name* という別の列が含まれる場合、*prod\_key* 値が与えられると、対応する *prod\_name* を検索できます。逆は真ではなく、*prod\_name* 値は必ずしも一意の *prod\_key* と関連しているとは限りません。

問合せによって要求される列データをマテリアライズド・ビューから取得できない場合でも、機能によって必要な列データを決定するキーがマテリアライズド・ビューに含まれていれば、このような列データは、その列データを含む表にマテリアライズド・ビューを再結合することによって取得できます。

たとえば、次の問合せについて考えます。

```
SELECT  s.store_name, t.time_week, p.prod_name,
        SUM(f.dollar_sales) AS sum_sales,
FROM    fact f, store s, time t, product p
WHERE   f.time_key = t.time_key AND f.store_key = s.store_key AND
        f.prod_key = p.prod_key AND p.prod_brand = 'KELLOGG'
GROUP BY s.store_name, t.time_week, p.prod_name;
```

マテリアライズド・ビュー *sum\_fact\_store\_time\_prod* には、*p.prod\_key* が含まれますが、*p.prod\_brand* は含まれていません。ただし、*prod\_key* の機能によって *prod\_brand* が決定されるので、*sum\_fact\_store\_time\_prod* を PRODUCT に再結合して、*prod\_brand* を検索できます。オプティマイザは、*sum\_fact\_store\_time\_prod* を使用してこの問合せを次のようにリライトします。

```
SELECT  mv.store_name, mv.time_week, p.product_key, mv.sum_sales,
FROM    sum_fact_store_time_prod mv, product p
WHERE   mv.prod_key = p.prod_key AND p.prod_brand = 'KELLOGG'
GROUP BY mv.store_name, mv.time_week, p.prod_key;
```

ここで、PRODUCT 表は、元はマテリアライズド・ビューで結合されていましたが、リライトされた問合せに再度結合されるので、再結合表と呼ばれます。

機能依存性を宣言する方法は 2 通りあります。

1. 主キー制約を使用する
2. ディメンションの DETERMINES 句を使用する

ディメンション定義の DETERMINES 句は、別の列を決定する列を主キー列にできないときに、機能依存性を宣言できる唯一の方法です。たとえば、STORE 表は、列 *store\_key*、*store\_name*、*store\_city*、*city\_name* および *store\_state* を持つ非正規化ディメンション表です。*store\_key* の機能によって *store\_name* が決定され、*store\_city* の機能によって *store\_state* が決定されます。

最初の機能依存性は、*store\_key* を主キーとして宣言することによって確立できますが、*store\_city* 列に重複する値が含まれているので、2 番目の機能依存性はこの方法では確立できません。この状況では、ディメンションの DETERMINES 句を使用して、2 番目の機能依存性を宣言できます。

次のディメンション定義で、機能依存性がどのように宣言されるかを示します。

```
CREATE DIMENSION store_dim
LEVEL store_key      IS store.store_key
LEVEL city           IS store.store_city
LEVEL state          IS store.store_state
LEVEL country        IS store.store_country
  HIERARCHY geographical_rollup      (
    store_key      CHILD OF
    city           CHILD OF
    state          CHILD OF
    country        )
ATTRIBUTE store_key DETERMINES store.store_name;
ATTRIBUTE store_city DETERMINES store.city_name;
```

階層 *geographic\_rollup* は、1:n の機能依存性も持つ階層関係を宣言します。store\_city の機能によって city\_name が決定されるような 1:1 の機能依存性は、DETERMINES 句を使用して宣言されます。

次の問合せは、

```
SELECT  s.store_city, p.prod_name
        SUM(f.dollar_sales) AS sum_sales,
FROM    fact f, store s, product p
WHERE   f.store_key = s.store_key AND f.prod_key = p.prod_key
        AND s.city_name = 'BELMONT'
GROUP BY s.store_city, p.prod_name;
```

*sum\_fact\_store\_prod* を STORE 表と結合することによってリライトできるので、city\_name を使用して述語を評価できます。しかし、結合は STORE 表の主キーではない store\_city 列に基づくので、重複が許可されます。リライトされた次の問合せで示されるように、これは固有値を選択するインライン・ビューを使用して実行され、このビューはマテリアライズド・ビューに結合されます。

```
SELECT  iv.store_city, mv.prod_name, mv.sum_sales
FROM    sum_fact_store_prod mv, (SELECT DISTINCT store_city, city_name
                                FROM store) iv
WHERE   mv.store_city = iv.store_city AND
        iv.store_name = 'BELMONT'
GROUP BY iv.store_city, mv.prod_name;
```

このタイプのリライトが可能なのは、ディメンションで宣言されるときに store\_city の機能によって city\_name が決定されるからです。

## グループ化互換性チェック

このチェックは、マテリアライズド・ビューと問合せの両方に GROUP BY 句が含まれている場合にのみ必要です。オプティマイザは、問合せによって要求されるデータのグループ化

が、マテリアライズド・ビューに格納されているデータのグループ化と完全に同じかどうかを最初に判断します。これは、問合せとマテリアライズド・ビューの両方でグループ化のレベルが同じであることを意味します。たとえば、問合せが *store\_city* によってグループ化されたデータを要求し、マテリアライズド・ビューが *store\_city* と *store\_state* によってグループ化されたデータを格納している場合などです。前述のディメンション例で示した機能依存性のように、*store\_city* の機能によって *store\_state* が決定される場合は、両方のグループ化が同じになります。

問合せによって要求されたデータのグループ化が、マテリアライズド・ビューに格納されているデータのグループに比べて粗いレベルの場合は、オプティマイザはマテリアライズド・ビューを使用して問合せをリライトできます。たとえば、マテリアライズド・ビュー *sum\_fact\_store\_time\_prod* は、*store\_name*、*time\_week* および *prod\_key* によってグループ化します。この問合せは、より粗いグループ化単位である *store\_name* によってグループ化されません。

```
SELECT s.store_name, SUM(f.dollar_sales) AS sum_sales,
FROM   fact f, store s
WHERE  f.store_key = s.store_key
GROUP BY s.store_name;
```

したがって、オプティマイザは、この問合せを次のようにリライトします。

```
SELECT store_name, SUM(sum_dollar_sales) AS sum_sales,
FROM   sum_fact_store_time_prod
GROUP BY s.store_name;
```

別の例では、問合せが *store\_state* によってグループ化されたデータを要求し、マテリアライズド・ビューが *store\_city* によってグループ化されたデータを格納します。*store\_city* が *store\_state* の子である場合（前述のディメンション例を参照）、マテリアライズド・ビューに格納されているグループ化されたデータは、問合せがリライトされるときに *store\_state* によってさらにグループ化されます。つまり、マテリアライズド・ビューに格納されている *store\_city* レベル（より細かい単位）での集計は、*store\_state* レベル（より粗い単位）での集計にロールアップできます。

たとえば、次の問合せについて考えます。

```
SELECT store_state, prod_name, SUM(f.dollar_sales) AS sum_sales
FROM   fact f, store s, product p
WHERE  f.store_key = s.store_key AND f.prod_key = p.prod_key
GROUP BY store_state, prod_name;
```

*store\_city* の機能によって *store\_state* が決定され、*sum\_fact\_store\_prod* を *store* 表への再結合に使用して *store\_state* 列データを検索できるので、集計は次に示すように *store\_state* レベルにロールアップできます。

```
SELECT store_state, prod_name, sum(mv.sum_sales) AS sum_sales
FROM   sum_fact_store_prod mv, (SELECT DISTINCT store_city, store_state
                                FROM store) iv
```

```
WHERE mv.store_city = iv.store_city
GROUP BY store_state, prod_name;
```

このリライトでは、データ充足チェックによって STORE 表への再結合が必要かどうかが判断され、グループ化互換性チェックによって集計のロールアップが必要かどうか判断されることに注意してください。

## 集計可能性チェック

このチェックは、問合せとマテリアライズド・ビューの両方に集計が含まれる場合にのみ必要です。オプティマイザは、問合せによって要求された集計が、マテリアライズド・ビューに格納されている 1 つ以上の集計から導出または計算できるかどうかを判断します。たとえば、問合せが AVG(X) を要求し、マテリアライズド・ビューに SUM(X) と COUNT(X) が含まれている場合、AVG(X) は SUM(X) / COUNT(X) として計算できます。

グループ化互換性チェックによって、マテリアライズド・ビューに格納されている集計のロールアップが必要であると判断された場合、集計可能性チェックは、問合せによって要求された各集計をマテリアライズド・ビューの集計を使用してロールアップできるかどうかを判断します。

たとえば、city レベルの SUM(sales) は、同じ state 値を持つグループ内のすべての SUM(sales) 集計を合計することによって、state レベルの SUM(sales) にロールアップできます。ただし、AVG(sales) は、マテリアライズド・ビューで COUNT(sales) も使用可能でない限り、より粗いレベルにはロールアップできません。同様に、VARIANCE(sales) または STDDEV(sales) は、マテリアライズド・ビューで COUNT(sales) と SUM(sales) も使用可能でない限り、ロールアップできません。たとえば、次のような問合せがあるとします。

```
SELECT p.prod_name, AVG(f.dollar_sales) AS avg_sales
FROM fact f, product p
WHERE f.prod_key = p.prod_key
GROUP BY p.prod_name;
```

FACT と STORE 間の結合が可逆式であり、重複していない場合は、マテリアライズド・ビュー *sum\_fact\_store\_prod* を使用してこれをリライトできます。さらに、問合せは *prod\_name* によってグループ化しますが、マテリアライズド・ビューは *store\_city*、*prod\_name* によってグループ化するので、マテリアライズド・ビューに格納されている集計はロールアップする必要があります。オプティマイザは、問合せを次のようにリライトします。

```
SELECT mv.prod_name, SUM(mv.sum_sales)/SUM(mv.count_sales) AS avg_sales
FROM sum_fact_store_prod mv
GROUP BY mv.prod_name;
```

SUM などの集計の引数は、A+B のような算術式にすることができます。オプティマイザは、問合せの集計 SUM(A+B) を、マテリアライズド・ビューに格納されている集計 SUM(A+B) または SUM(B+A) と照合しようとしています。つまり、問合せ内の集計の引数をマテリアライズド・ビューの同様の集計の引数と照合するときに、式の評価が使用されます。これを実行するために、Oracle は、2 つの異なる等価式を同じ基準形式に変換するような方法で、集計

引数の式を基準形式に変換します。たとえば、 $A*(B-C)$ 、 $A*B-C*A$ 、 $(B-C)*A$  および  $-A*C+A*B$  はすべて同じ基準形式に変換されるので、これらは照合の結果一致します。

## 制約とディメンションが必要になるとき

さまざまなタイプのクエリー・リライトでディメンションと制約が必要になるときを明確にするために、表 31-2 を参照してください。

表 31-2 クエリー・リライトでのディメンションと制約の要件

リライト・チェック	ディメンション		主キー / 外部キー / NOT NULL 制約
SQL テキストの照合	不要		不要
結合互換性	不要		必須
データ充足	必須	または	必須
グループ化互換性	必須		必須
集計可能性	不要		不要

## クエリー・リライトの正確性

クエリー・リライトには、初期化パラメータ `QUERY_REWRITE_INTEGRITY` によって制御できる 3 レベルのリライト整合性があります。この初期化パラメータは、パラメータ・ファイルで設定するか、`ALTER SYSTEM` または `ALTER SESSION` コマンドを使用して制御できます。可能な 3 つの値は次のとおりです。

- ENFORCED

これはデフォルトのモードです。オプティマイザは、新しいデータが含まれていることがわかっているマテリアライズド・ビューのみ使用し、施行された制約に基づく関係のみ使用します。

- TRUSTED

TRUSTED モードでは、オプティマイザは、事前作成表をベースとするマテリアライズド・ビュー内のデータが正しいことと、ディメンションで宣言されている関係と RELY 制約が正しいことを前提にします。このモードでは、オプティマイザは事前作成表によるマテリアライズド・ビューを使用し、施行されていない関係と施行された関係を使用します。このモードでは、オプティマイザは、宣言されているが施行はされていない制約とディメンションを使用して指定されたデータ関係も正しいものとみなします。

- STALE\_TOLERATED

STALE\_TOLERATED モードでは、オプティマイザは、有効ではあるが古いデータを含むマテリアライズド・ビューと、新しいデータを含むマテリアライズド・ビューを使用

します。このモードでは、最大のリライト機能が提供されますが、間違った結果が生成される危険性もあります。

リライトの整合性が最も安全なレベル ENFORCED に設定されている場合、オブティマイザは、施行された主キー制約と参照整合性制約のみ使用して、問合せの結果がディテール表に直接アクセスしたときの結果と同じであることを確認します。

リライトの整合性が ENFORCED 以外のレベルに設定されている場合は、リライトの出力がリライトしない場合の出力と異なる可能性のある状況がいくつかあります。

1. マテリアライズド・ビューは、データのマスター・コピーと同期しなくなることがあります。一般に、この状況は、マテリアライズド・ビューのリフレッシュ・プロシージャが、それに続くマテリアライズド・ビューの1つ以上のディテール表に対する大量ロードや DML 操作を保留していることが原因で発生します。データ・ウェアハウス・サイトによっては、一部のマテリアライズド・ビューが一定の時間間隔でリフレッシュされることがよくあるため、この状況が望ましい場合があります。
2. ディメンション・オブジェクトによって暗示される関係が無効な場合。たとえば、階層内の特定のレベルの値が、1つの親値に正確にロールアップされない場合などです。
3. 事前作成マテリアライズド・ビュー表に格納されている値は、誤っている可能性があります。
4. ディテール表に対する DROP や MOVE PARTITION などのパーティション操作は、マテリアライズド・ビューの結果に影響する可能性があります。

## クエリー・リライトが行われたかどうか

クエリー・リライトは透過的に行われるので、問合せがリライトされたかどうかを確認するには特別なステップが必要です。当然ですが、問合せの実行が速い場合はリライトが行われていることを示していますが、その証拠はありません。したがって、クエリー・リライトが行われたことを確認するには、EXPLAIN PLAN 文を使用します。

### Explain Plan

EXPLAIN PLAN 機能は、『Oracle8i SQL リファレンス』で説明されているように使用されます。クエリー・リライトでは、チェックする必要があるのは、PLAN\_TABLE の *object\_name* 列にマテリアライズド・ビュー名が含まれているかどうかということだけです。この列が含まれている場合は、この問合せが実行されるときにクエリー・リライトが行われます。

次の例では、マテリアライズド・ビュー *store\_mv* が作成されています。

```
CREATE MATERIALIZED VIEW store_mv
  ENABLE QUERY REWRITE
AS
SELECT
  s.region, SUM(grocery_sq_ft) AS sum_floor_plan
```

```
FROM store s
GROUP BY s.region;
```

この SQL 文に対して EXPLAIN PLAN が使用されると、その結果はデフォルトの表 PLAN\_TABLE に入れられます。

```
EXPLAIN PLAN
FOR
SELECT  s.region, SUM(grocery_sq_ft)
FROM store s
GROUP BY s.region;
```

クエリー・リライトを目的とした場合、PLAN\_TABLE 内の重要な情報は、この問合せの実行に使用されるオブジェクトを識別する OBJECT\_NAME のみです。したがって、次に示すように、出力にオブジェクト名 STORE\_MV があることが期待されます。

```
SELECT  object_name FROM plan_table;

OBJECT_NAME
-----

STORE_MV
2 rows selected.
```

**関連項目：** ヒントの詳細は、7-32 ページの「[ヒントの使用](#)」を参照してください。

## クエリー・リライトの制御

マテリアライズド・ビューは、ENABLE QUERY REWRITE 句が指定されている場合にのみクエリー・リライトの対象となります。この句は、マテリアライズド・ビューが最初に作成されたときに指定されるか、後で ALTER MATERIALIZED VIEW コマンドを介して指定されます。

前述の初期化パラメータは、ALTER SYSTEM SET コマンドを使用して設定できます。ユーザーのセッションでは、ALTER SESSION を使用して、そのセッションでのみクエリー・リライトを使用禁止または使用可能にすることができます。次に例を示します。

```
ALTER SESSION SET QUERY_REWRITE_ENABLED = TRUE;
```

クエリー・リライトの正確性はセッションに対して設定できるので、異なるユーザーが異なる整合性レベルで作業できます。

```
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = STALE_TOLERATED;
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = TRUSTED;
ALTER SESSION SET QUERY_REWRITE_INTEGRITY = ENFORCED;
```

### リライト・ヒント

SQL 文には、クエリー・リライトを実行するかどうかを制御するヒントが含まれていることがあります。問合せで NOREWRITE ヒントを引数なしで使用すると、オプティマイザはその問合せをリライトしません。

問合せで REWRITE ヒントを引数なしで指定すると、オプティマイザは、マテリアライズド・ビューを使用して（存在する場合）コストに関係なく強制的に問合せをリライトします。

REWRITE (*mv1*, *mv2*, ...) ヒントを引数とともに指定すると、リライトは指定された名前のリストから最も適切なマテリアライズド・ビューを選択します。

たとえば、リライトを防ぐには、次の文を使用します。

```
SELECT /*+ NOREWRITE */ s.city, SUM(s.grocery_sq_ft)
FROM store s
GROUP BY s.city;
```

また、*mv1* を使用したリライトを強制するには、次の文を使用します。

```
SELECT /*+ REWRITE (mv1) */ s.city, SUM(s.grocery_sq_ft)
FROM store s
GROUP BY s.city;
```

## クエリー・リライトの使用のガイドライン

次のガイドラインは、クエリー・リライトの利点を最大限に引き出すのに役立ちます。これらは、クエリー・リライトの使用で必須ではなく、このガイドラインに従ってもリライトが保証されるわけではありません。これらは、一般的な経験則です。

### 制約

マテリアライズド・ビューで参照されているすべての内部結合に、外部キー列に定義された NOT NULL 制約との参照整合性（外部キー - 主キー制約）があることを確認します。制約は大規模なオーバーヘッドを生成する傾向があるので、これらを NONVALIDATE および RELY にし、パラメータ QUERY\_REWRITE\_INTEGRITY を STALE\_TOLERATED または TRUSTED に設定します。ただし、QUERY\_REWRITE\_INTEGRITY を ENFORCED に設定した場合は、リライト可能性を最大にするためにはすべての制約を施行する必要があります。

### ディメンション

ディメンションの HIERARCHY 句を使用して、正規化または非正規化ディメンション表の階層関係と機能依存性を表現できます。ディメンションは、どの制約でも表現できない内部表関係を表現できます。ディメンションで宣言されている関係をクエリー・リライトで利用



するには、パラメータ `QUERY_REWRITE_INTEGRITY` を `TRUSTED` または `STALE_TOLERATED` に設定します。

## 外部結合

制約を回避する別の方法として、マテリアライズド・ビューで外部結合を使用する方法もあります。B の ROWID または列 B.b がマテリアライズド・ビューで使用可能であれば、クエリー・リライトは、(A.a = B.b) などの問合せの内部結合を、マテリアライズド・ビューの外部結合 (A.a = B.b(+)) から導出できます。外部結合を使用したリライトのサポートの大部分は、結合のみのマテリアライズド・ビューに対して提供されます。これを活用するには、外部結合のあるマテリアライズド・ビューが、外部結合の内部表の ROWID または主キーを格納する必要があります。たとえば、マテリアライズド・ビュー `join_fact_store_time_oj` は、外部結合の内部表の主キー `store_key` と `time_key` を格納します。

## SQL テキスト照合

非常に複雑な長時間実行問合せを高速化する必要がある場合は、問合せの正確なテキストでマテリアライズド・ビューを作成します。

## 集計

クエリー・リライトの利点を最大限に引き出すために、ターゲットの問合せ内の集計の計算に必要なすべての集計がマテリアライズド・ビューに存在することを確認します。集計に対する条件は、増分リフレッシュに対する条件と同様です。たとえば、`AVG(x)` が問合せにある場合は、`COUNT(x)` と `AVG(x)` または `SUM(x)` と `COUNT(x)` をマテリアライズド・ビューに格納する必要があります。29-8 ページの表 29-1 の「[マテリアライズド・ビューの高速リフレッシュの要件](#)」を参照してください。

## グループ化条件

階層の下位レベルを使用するとより多くの問合せをリライトできるので、階層の下位レベルでデータを集計する方が上位レベルで集計するよりも妥当です。ただし、下位レベルでの集計にはより多くの領域が必要なので注意してください。たとえば、州でグループ化するかわりに、都市でグループ化します（領域の制約によって制限されない場合）。

オーバーラップする GROUP BY 列または階層的に関連する GROUP BY 列のある複数のマテリアライズド・ビューを作成するかわりに、それらすべての GROUP BY 列を持つ単一のマテリアライズド・ビューを作成します。たとえば、都市でグループ化するマテリアライズド・ビューと月でグループ化する別のマテリアライズド・ビューを使用するかわりに、都市と月でグループ化するマテリアライズド・ビューを使用します。

ディメンション内のレベルに対応する列には GROUP BY を使用しますが、クエリー・リライトはディメンションの DETERMINES 句に基づいて機能依存性を自動的に使用できるので、機能的に依存する列には GROUP BY を使用しません。たとえば、`city_name` でグループ化するかわりに、`city_id` でグループ化します（属性 `city_id` によって `city_name` が決定される

ことを示すディメンションがある限り、*city\_name* に関連する問合せのリライトを使用可能にできます。

## 統計

マテリアライズド・ビューを使用した最適化はコストをベースとするので、オブティマイザは、コストベースの選択を行うために、マテリアライズド・ビューと問合せ内の表の両方について統計を必要とします。このため、マテリアライズド・ビューは、ANALYZE TABLE 文または DBMS\_STATISTICS パッケージを使用して収集された統計を持つ必要があります。

**関連項目：** 統計の収集の詳細は、7-7 ページの「[統計情報の生成](#)」を参照してください。

---

## マテリアライズド・ビューの管理

この章のトピックは次のとおりです。

- [マテリアライズド・ビュー管理の概要](#)
- [ウェアハウス・リフレッシュ](#)
- [サマリー・アドバイザ](#)
- [マテリアライズド・ビューが使用されているかどうか](#)

### マテリアライズド・ビュー管理の概要

マテリアライズド・ビューを使用する目的は、パフォーマンスを改善することですが、マテリアライズド・ビューの管理に関連するオーバーヘッドはシステム管理上の大きな問題になることがあります。マテリアライズド・ビューの管理アクティビティには、次のものがあります。

- 最初に作成するマテリアライズド・ビューの選択
- マテリアライズド・ビューの索引の作成
- データベースが更新されるたびに、すべてのマテリアライズド・ビューとマテリアライズド・ビューの索引が正しくリフレッシュされることの確認
- どのマテリアライズド・ビューが使用されたかをチェック
- 各マテリアライズド・ビューがパフォーマンス向上に対して与えた効果の判断
- マテリアライズド・ビューによって使用される領域の測定
- 作成する必要がある新しいマテリアライズド・ビューの判別
- 削除する必要がある既存のマテリアライズド・ビューの判別
- 不要になった古いディテール・データとマテリアライズド・ビュー・データのアーカイブ

この章では、これらのタスクをウェアハウス・リフレッシュおよびウェアハウス・アドバイザという2つの領域にグループ化します。ウェアハウス・リフレッシュは、マテリアライズド・ビューに正しい最新のデータが含まれていることの保証に関係し、ウェアハウス・アドバイザは、作成、保持および削除するマテリアライズド・ビューを推奨します。

データ・ウェアハウスまたはデータ・マートの作成および移入の初期作業の後、主要な管理オーバーヘッドとなるのは更新プロセスです。更新プロセスには、運用システムからの増分変更の定期的な抽出、データの変換、増分変更が正しく、一貫性があり、完全であることの検証、ウェアハウスへのデータの大量ロード、およびディテール・データとの一貫性を保つための索引とマテリアライズド・ビューのリフレッシュが含まれます。

更新プロセスは、一般に、更新ウィンドウと呼ばれる制限された期間内に実行する必要があります。更新ウィンドウは、更新頻度（毎日、毎週など）と業務に依存します。更新頻度が毎日の場合は、2～6時間の更新ウィンドウが一般的です。

更新ウィンドウは、通常は次のアクティビティの時間を表示します。

1. ディテール・データのロード
2. ディテール・データの索引の更新または再作成
3. データに対する品質保証テストの実行
4. マテリアライズド・ビューのリフレッシュ
5. マテリアライズド・ビューの索引の更新

ウェアハウスまたはデータ・マートにデータをロードする一般的で効率のよい方法は、`DIRECT` または `PARALLEL` オプションを指定した `SQL*Loader` を使用するか、Oracle ダイレクト・パス API を使用する別のローダー・ツールを使用することです。

**関連項目：** `DIRECT` または `PARALLEL` キーワードを指定した `SQL*Loader` を使用するときの制限事項と考慮事項は、『Oracle8i ユーティリティ・ガイド』を参照してください。

ロード方法は、1 フェーズ方式と2 フェーズ方式に分類できます。1 フェーズ・ロードでは、マテリアライズド・ビューをリフレッシュする前に DML 操作を実行することによって、データがターゲット表に直接ロードされ、品質保証テストが実行され、エラーが解決されます。大量の削除が発生する場合は、記憶域の使用率に悪影響を及ぼすことがありますが、一時的な領域要件とロード時間は最小化されます。1 フェーズ・ロードの後に必要になることのある DML は、複数の表を集計したマテリアライズド・ビューが最も安全なライト整合性レベルで使用できなくなる原因となります。

2 フェーズ・ロード・プロセスでは、次のことが行われます。

- データは最初にウェアハウス内の一時表にロードされます。
- 品質保証プロシージャがデータに適用されます。
- ターゲット表の参照整合性制約が使用禁止にされ、ターゲット・パーティションのローカル索引に使用禁止のマークが付けられます。

- INSERT AS SELECT を PARALLEL または APPEND ヒントとともに使用して、一時領域からターゲット表の適切なパーティションにデータがコピーされます。
- 一時表が削除されます。
- 通常は NOVALIDATE オプション付きで、制約が使用可能にされます。

ディテール・データのロードとディテール・データの索引の更新の直後に、必要に応じて操作するためにデータベースをオープンできます。クエリー・リライトは、すべてのマテリアライズド・ビューがリフレッシュされるまでデフォルトで (ALTER SYSTEM SET QUERY\_REWRITE\_ENABLED = FALSE で) 使用禁止にできますが、最後のロードのデータを反映されたマテリアライズド・ビューを必要としないユーザーに対しては、セッション・レベルで (ALTER SESSION SET QUERY\_REWRITE\_ENABLED = TRUE で) 使用可能にできます。ただし、QUERY\_REWRITE\_INTEGRITY = ENFORCED または TRUSTED である限り、更新されたデータのあるマテリアライズド・ビューだけがクエリー・リライトに関与することをシステムが保証するので、これは必ずしも必要ではありません。これらのパッケージを使用して、結合のみ、結合と集計、または単一表の集計を含むマテリアライズド・ビューなど、任意のタイプのマテリアライズド・ビューをリフレッシュできます。

## ウェアハウス・リフレッシュ

マテリアライズド・ビューを作成するときに、リフレッシュが要求時 (ON DEMAND) とコミット時 (ON COMMIT) のどちらに行われるかを指定するオプションがあります。高速なウェアハウス・リフレッシュ機能を使用するには、ON DEMAND モードを指定する必要があります。この場合、マテリアライズド・ビューは、DBMS\_MVIEW のプロシージャの 1 つをコールすることによってリフレッシュできます。

DBMS\_MVIEW パッケージには、3 種類のリフレッシュ操作が用意されています。

- DBMS\_MVIEW.REFRESH

1 つ以上のマテリアライズド・ビューをリフレッシュします。

- DBMS\_MVIEW.REFRESH\_ALL\_MVIEWS

すべてのマテリアライズド・ビューをリフレッシュします。

- DBMS\_MVIEW.REFRESH\_DEPENDENT

指定されたディテール表またはディテール表のリストに依存する表ベースのマテリアライズド・ビューをすべてリフレッシュします。

このパッケージの詳細は、32-6 ページの「[DBMS\\_MVIEW パッケージを使用した手動リフレッシュ](#)」を参照してください。

リフレッシュ操作を実行するには、索引を再作成するための一時領域が必要であり、リフレッシュ操作自体を実行するために追加領域が必要になることもあります。高速リフレッシュでは、一時表をユーザーの一時表領域に作成することも必要です。

サイトによっては、すべてのマテリアライズド・ビューを同時にリフレッシュしない方が望ましい場合があります。したがって、マテリアライズド・ビューのリフレッシュを遅延させる場合は、ALTER SYSTEM SET QUERY\_REWRITE\_ENABLED = FALSE を指定して、クエリー・リライトを一時的に使用禁止にすることができます。古いマテリアライズド・ビューにもアクセスしたいユーザーは、このデフォルトを ALTER SESSION SET QUERY\_REWRITE\_ENABLED = TRUE で上書きできます。マテリアライズド・ビューをリフレッシュした後で、ALTER SYSTEM SET QUERY\_REWRITE\_ENABLED = TRUE を設定することによって、現行のデータベース・インスタンスのすべてのセッションに対してクエリー・リライトをデフォルトとして再度使用可能にすることができます。

マテリアライズド・ビューのリフレッシュによって、その索引がすべて自動的に更新されます。完全リフレッシュの場合は、この処理のために一時的なソート領域が必要になります。索引の再作成に使用できる一時領域が不十分な場合は、リフレッシュ操作を実行する前に、各索引を明示的に削除するか、使用禁止のマークを付ける必要があります。

マテリアライズド・ビューをリフレッシュするときには、次の表に示す 4 つのリフレッシュ方法の 1 つを指定できます。

リフレッシュ・オプション		説明
COMPLETE	C	atomic refresh=TRUE の場合は、マテリアライズド・ビューの定義問合せを再計算することによってリフレッシュします。atomic refresh=FALSE の場合、COMPLETE は FORCE と同じです。
FAST	F	変更の増分をディテール表に適用することによってリフレッシュします。
FORCE	?	デフォルトのリフレッシュ方法を使用します。デフォルトのリフレッシュ方法が FORCE の場合は、高速リフレッシュを試行します。高速リフレッシュが不可能な場合は、完全リフレッシュを実行します。
ALWAYS	A	無条件に完全リフレッシュを実行します。

完全リフレッシュ

マテリアライズド・ビューが事前作成表を参照していない限り、マテリアライズド・ビューが最初に定義されたときには完全リフレッシュが実行され、完全リフレッシュはマテリアライズド・ビューが存続している間はいつでも要求できます。リフレッシュでは、マテリアライズド・ビューの結果を計算するためにディテール表を読み込む必要があるため、完全リフレッシュは非常に時間のかかるプロセスになることがあります。読み込んで処理する必要のあるデータが大量の場合には、特に時間がかかります。したがって、完全リフレッシュを要求する前に、その処理に必要な時間を常に考慮する必要があります。

ただし、マテリアライズド・ビューが次の項で指定する高速リフレッシュの条件を満たしていないために、使用できるリフレッシュ方法が完全リフレッシュのみの場合があります。

## 高速リフレッシュ

ほとんどのデータ・ウェアハウスでは、ディテール・データに対する定期的な増分更新が必要です。28-7 ページの「[マテリアライズド・ビューのスキーマ・デザイン・ガイド](#)」で説明したように、SQL\*Loader のダイレクト・パス・オプションを使用するか、Oracle のダイレクト・パス・インタフェースを使用する大量ロード・ユーティリティを使用して、ディテール・データの増分ロードを実行できます。Oracle のダイレクト・パス・インタフェースを使用すると、マテリアライズド・ビュー全体を再計算するかわりに変更が既存のデータに追加されるので、マテリアライズド・ビューの高速リフレッシュの効率がよくなります。このため、変更のみの適用によって、リフレッシュ時間が非常に高速になります。

増分リフレッシュを実行するのに必要な時間には、いくつかの要因が影響します。

- マテリアライズド・ビューのコンテナ表のデータが時間属性によってまとめられているかどうか
- マテリアライズド・ビューのキーで複合索引が使用可能かどうか
- マテリアライズド・ビュー内で、参照整合性制約または CREATE/ALTER DIMENSION 文の JOIN KEY 宣言の一部として宣言されていない内部結合の数

最初の 2 つの要因は、ファクト表などのマテリアライズド・ビュー・コンテナを時間別にパーティション化すること、およびマテリアライズド・ビューのキーにローカル複合索引を作成することによって対応できます。3 番目の要因は、スキーマのディメンションと階層を作成すること、および、次に説明するように、可能な場合にはマテリアライズド・ビューのすべての内部結合が厳密に 1:n の関係になるようにすることによって対応できます。

増分ロードが実行された場合、通常は完全リフレッシュよりも高速リフレッシュを実行する方がはるかに高速です。ウェアハウスの高速リフレッシュは、次を除くすべてのケースでサポートされます。

- 集計を含むマテリアライズド・ビューに複数の表があり、完全リフレッシュが最後に実行された後に、ファクト表に対してダイレクト・ロード以外の任意の DML が実行された場合。
- マテリアライズド・ビューに、ビューまたはスナップショットであるディテール・リレーションが含まれている場合。
- マテリアライズド・ビューに、COUNT(x) がなく AVG(x) が含まれている場合。
- マテリアライズド・ビューに、COUNT(x) と SUM(x) がなく、VARIANCE(x) が含まれている場合。
- マテリアライズド・ビューに、COUNT(x) と SUM(x) がなく、STDDEV(x) が含まれている場合。

増分リフレッシュは、マテリアライズド・ビューに対して挿入と更新の両方を実行する場合があるので注意してください。新規行が挿入された場合は、キー列やメジャー列以外のマテリアライズド・ビューの列がデフォルト値に設定されます。

参照される表に UPDATE や DELETE などの DML 操作が適用される場合にも、マテリアライズド・ビューを高速リフレッシュ可能にするには、[第 29 章「マテリアライズド・ビュー」](#)を参照してください。ここには、マテリアライズド・ビューのログが存在する場合に DML 操作が可能なマテリアライズド・ビューのタイプが記載されています。

### DBMS\_MVIEW パッケージを使用した手動リフレッシュ

ON DEMAND リフレッシュを実行するための 3 種類のリフレッシュ・プロシージャが、DBMS\_MVIEW パッケージに用意されています。これらのプロシージャには、それぞれ固有のパラメータ・セットがあります。このパッケージを使用するには、Oracle8i のキューが使用可能になっている必要があるので、初期化パラメータ・ファイルに次のパラメータを設定する必要があります。キューが使用できない場合は、リフレッシュが失敗して該当するメッセージが表示されます。

**関連項目：** DBMS\_MVIEW パッケージの詳細情報は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。レプリケーション環境でこのパッケージを使用する方法は、『Oracle8i レプリケーション・ガイド』に説明されています。

### ウェアハウス・リフレッシュに必須の初期化パラメータ

- JOB\_QUEUE\_PROCESSES

バックグラウンド・プロセスの数。同時にリフレッシュできるマテリアライズド・ビューの数を決定します。

- JOB\_QUEUE\_INTERVAL

新しいジョブがジョブ・キューに発行されたかどうかをジョブ・キュー・スケジューラがチェックする秒単位の間隔。

- UTL\_FILE\_DIR

リフレッシュ・ログが書き込まれるディレクトリを決定します。このパラメータが指定されていない場合、リフレッシュ・ログは作成されません。

これらのパッケージは、リフレッシュ・プロセス中の問題の診断に役立つ refresh.log というログもデフォルトで作成します。ログ・ファイルはプロシージャ DBMS\_OLAP.SET\_LOGFILE\_NAME('log filename') をコールすることによって改名できます。

### 特定のマテリアライズド・ビューのリフレッシュ

DBMS\_MVIEW.REFRESH プロシージャは、FROM リストで明示的に定義されている 1 つ以上のマテリアライズド・ビューをリフレッシュするために使用されます。このリフレッシュ・プロシージャは、レプリケーションで使用するマテリアライズド・ビューのリフレッシュにも使用できるので、すべてのパラメータが必須なわけではありません。このプロシージャを使用するための必須パラメータは、次のとおりです。



- カンマで区切られた、リフレッシュするマテリアライズド・ビューのリスト
- リフレッシュ方法 (A-Always、F-Fast、?-Force、C-Complete)
- 使用するロールバック・セグメント
- エラー後の継続

このパラメータが TRUE に設定されている場合は、複数のマテリアライズド・ビューをリフレッシュするときにその 1 つがリフレッシュでエラーになっても、ジョブ全体は継続されます。

- 続く 4 つのパラメータは、FALSE、0、0、0 に設定する必要があります。

これらのパラメータは、レプリケーション・プロセスによって使用されるので、ウェアハウス・リフレッシュではこれらの値が必須です。

- アトミック・リフレッシュ

このパラメータが TRUE に設定されている場合、ウェアハウス・リフレッシュは使用されません。かわりに、スナップショット / レプリケーション・リフレッシュを使用します。FALSE に設定されている場合は、ウェアハウス・リフレッシュ方法が使用され、各リフレッシュ操作は独自のトランザクション内で実行されます。

したがって、マテリアライズド・ビュー `store_mv` に対して高速なリフレッシュを実行するには、パッケージが次のようにコールされます。

```
DBMS_MVIEW.REFRESH('STORE_MV', 'A', '', TRUE, FALSE, 0,0,0, FALSE);
```

複数のマテリアライズド・ビューは同時にリフレッシュでき、必ずしもすべてが同じリフレッシュ方法を使用する必要はありません。異なるリフレッシュ方法を実行するには、複数の方法コードをマテリアライズド・ビューのリストと同じ順序で指定します (カンマはなし)。たとえば、次のように指定すると、`store_mv` が完全リフレッシュされ、`product_mv` が高速リフレッシュされます。

```
DBMS_MVIEW.REFRESH('STORE_MV,PRODUCT_MV', 'AF', '', TRUE, FALSE, 0,0,0, FALSE);
```

## すべてのマテリアライズド・ビューのリフレッシュ

リフレッシュするマテリアライズド・ビューを指定する別の方法として、プロシージャ `DBMS_MVIEW.REFRESH_ALL_MVIEWS` を使用する方法もあります。これによって、すべてのマテリアライズド・ビューがリフレッシュされます。いずれかのマテリアライズド・ビューのリフレッシュに失敗した場合は、エラーの数がレポートされます。

このプロシージャのパラメータは、次のとおりです。

- エラーの数
- データ型番号
- リフレッシュ方法 (A-Always、F-Fast、?-Force、C-Complete)

- 使用するロールバック・セグメント
- エラー後の継続

すべてのマテリアライズド・ビューのリフレッシュの例を、次に示します。

```
DBMS_MVIEW.REFRESH_ALL_MVIEWS ( failures, 'A', '', FALSE, FALSE );
```

## 依存するマテリアライズド・ビューのリフレッシュ

3 番目のオプションは、プロシージャ DBMS\_MVIEW.REFRESH\_DEPENDENT を使用して、特定の表に依存するマテリアライズド・ビューのみリフレッシュする機能です。たとえば、受注表に対する変更が受信され、顧客の支払はないとします。依存リフレッシュ・プロシージャをコールして、ORDER 表を参照するマテリアライズド・ビューのみリフレッシュできます。

このプロシージャのパラメータは、次のとおりです。

- エラーの数
- 依存表
- リフレッシュ方法 ( A-Always、F-Fast、?-Force、C-Complete )
- 使用するロールバック・セグメント
- エラー後の継続

ブール値パラメータ。これが TRUE に設定されている場合は、エラーの数を表す出力パラメータに失敗したリフレッシュの数が設定され、汎用エラー・メッセージによってエラーの発生が示されます。リフレッシュ・ログには、各エラーの詳細とインスタンスに対するアラート・ログが記録されます。デフォルトの FALSE に設定されている場合は、最初のエラーの発生後にリフレッシュが停止し、リスト内の残りのマテリアライズド・ビューはリフレッシュされません。

- アトミック・リフレッシュ

ブール値パラメータ。

ORDERS 表を参照するすべてのマテリアライズド・ビューに対して完全リフレッシュを実行するには、次のように指定します。

```
DBMS_mview.refresh_dependent (failures, 'ORDERS', 'A', '', FALSE, FALSE );
```

## ウェアハウス・リフレッシュを使用したリフレッシュのヒント

DBMS\_MVIEW.REFRESH を実行しているプロセスが割り込まれた場合、またはインスタンスがシャット・ダウンした場合は、ジョブ・キュー・プロセスで実行されたリフレッシュ・ジョブがもう一度キューに入れられ、実行が継続されます。これらのジョブを取り消すには、DBMS\_JOB.REMOVE プロシージャを使用します。

## 結合と集計のあるマテリアライズド・ビュー

次に、結合と集計のあるマテリアライズド・ビューに対するリフレッシュ・メカニズムの使用のガイドラインをいくつか示します。

1. ウェアハウス・リフレッシュ機能は、集計を含むマテリアライズド・ビューに対してのみ動作します。
2. 可能な場合は、必ずダイレクト・パス・オプションを使用して新規データをロードします。完全リフレッシュが必要になるので、削除と更新は避けます。ただし、マテリアライズド・ビューのパーティションを削除して、高速リフレッシュを実行できます。
3. 固定キー制約をファクト表に置き、ファクト表の主キー制約をディメンション表に置きます。これによってリフレッシュでファクト表を識別できるので、高速リフレッシュに役立ちます。
4. ロード中は、すべての制約を使用禁止にして、ロードが完了したら再度使用可能にします。
5. 複合索引を使用して、外部キー列にマテリアライズド・ビューの索引を作成します。
6. 高速なリフレッシュの速度を上げるために、ジョブ・キュー・プロセス数をプロセス数よりも多くします。
7. リフレッシュするマテリアライズド・ビューの数が多い場合は、個別にコールするよりも、単一のコマンドですべてをリフレッシュする方が高速です。
8. FORCE リフレッシュ方法を使用して、クエリー・リライトに使用できるマテリアライズド・ビューがリフレッシュされるようにします。高速リフレッシュを実行できない場合は、完全リフレッシュが実行されます。一方、高速リフレッシュを要求した場合は、高速リフレッシュできるものがないとマテリアライズド・ビューはまったくリフレッシュされません。
9. 高速リフレッシュの方が速いので、高速リフレッシュ可能なマテリアライズド・ビューを作成するようにします。
10. サマリーに、ファクト表にはなくなったデータに基づくデータが含まれている場合は、高速なリフレッシュを使用してマテリアライズド・ビューをメンテナンスします。ジョブ・キューが開始していない場合は、リフレッシュによって2つのジョブ・キュー・プロセスが開始されます。これは、次の指定によって変更できます。

```
ALTER SYSTEM SET JOB_QUEUE_PROCESSES = value
```

11. 一般に、プロセッサの数が多いほど、より多くのジョブ・キュー・プロセスを作成する必要があります。また、主として完全リフレッシュを実行している場合は、各リフレッシュが高速リフレッシュよりも多くのシステム・リソースを消費するので、ジョブ・キュー・プロセスの数を減らします。ジョブ・キュー・プロセスの数によって、同時にリフレッシュできるマテリアライズド・ビューの数が制限されます。これに対し、主として高速リフレッシュを実行する場合は、ジョブ・キュー・プロセスの数を増やします。

## 集計のある単一表を含むマテリアライズド・ビューのリフレッシュ

集計を含み、単一表をベースとするマテリアライズド・ビューは、データの変更がダイレクト・パスまたは SQL DML 文を使用して行われたときに、29-8 ページの表 29-1 の「マテリアライズド・ビューの高速リフレッシュの要件」のルールに従っていれば、高速リフレッシュが可能な場合があります。リフレッシュ時に、Oracle は、実行された DML のタイプ（ダイレクト・ロードまたは SQL DML）を検出し、マテリアライズド・ビュー・ログまたはダイレクト・パスから入手できる情報を使用して、新規データを判別します。両方の方法を使用してデータが変更された場合は、最後に 1 回リフレッシュを発行するのではなく、各タイプのデータ変更の後にリフレッシュを実行する必要があります。これは、Oracle が 1 つのタイプの DML のみ実行されたことを検出した場合に、顕著な最適化を実行できるためです。したがって、使用例 1 ではなく使用例 2 に従うことをお勧めします。

高速リフレッシュのパフォーマンスを改善するために、ROWID を含む列に索引を作成することを強くお勧めします。

### 使用例 1

- ディテール表へのデータのダイレクト・ロード
- ディテール表への INSERT や DELETE などの SQL DML
- マテリアライズド・ビューのリフレッシュ

### 使用例 2

- ディテール表へのデータのダイレクト・ロード
- マテリアライズド・ビューのリフレッシュ
- ディテール表への INSERT や DELETE などの SQL DML
- マテリアライズド・ビューのリフレッシュ

さらに、ON-COMMIT のリフレッシュの場合、Oracle はコミットされたトランザクションで実行された DML のタイプを追跡します。このため、Oracle がリフレッシュ・フェーズを最適化できない可能性があるので、同じトランザクションで別の表に対してダイレクト・パス・ロードや SQL DML を実行しないことをお勧めします。

表に対する更新が多数ある場合は、それらを 1 つのトランザクションにまとめて、マテリアライズド・ビューのリフレッシュが各更新の後ではなくコミット時に 1 回だけ実行されるようにするのが賢明です。ウェアハウスでは、大量ロードの後に、ユーザーがセッションでパラレル DML を使用可能にし、リフレッシュを実行する必要があります。Oracle は、パラレル DML を使用してリフレッシュを行うので、パフォーマンスが著しく改善されます。マテリアライズド・ビューがパーティション化されている場合は、パフォーマンスがさらに改善されます。

例として、マテリアライズド・ビューがパーティション化され、パラレル句がある場合を想定します。データ・ウェアハウスでは、次のシーケンスが推奨されます。

1. ディテール表への大量ロード

2. ALTER SESSION ENABLE PARALLEL DML;

3. マテリアライズド・ビューのリフレッシュ

## 結合のみを含むマテリアライズド・ビューのリフレッシュ

マテリアライズド・ビューに結合が含まれ、集計は含まれていない場合、このタイプのマテリアライズド・ビューは集計を含むマテリアライズド・ビューよりもはるかに大きくなる傾向があるので、ディテール表の各結合列の ROWID に索引を作成すると、リフレッシュのパフォーマンスが大幅に改善されます。たとえば、次のマテリアライズド・ビューについて考えます。

```
CREATE MATERIALIZED VIEW detail_fact_mv
  BUILD IMMEDIATE
  REFRESH FASY ON COMMIT
  AS
  SELECT
    f.rowid "fact_rid", t.rowid "time_rid", s.rowid "store_rid",
    s.store_key, s.store_name, f.dollar_sales,
    f.unit_sales, f.time_key
  FROM fact f, time t, store s
  WHERE f.store_key = s.store_key(+) and
    f.time_key = t.time_key(+);
```

列 FACT\_RID、TIME\_RID および STORE\_RID に索引を作成する必要があります。パーティション化によってリフレッシュのパフォーマンスが大幅に改善されるので、リフレッシュを実行する前に、セッションでパラレル DML を使用可能にするのと同様にパーティション化を強くお勧めします。

このタイプのマテリアライズド・ビューは、ディテール表に対して DML が実行される場合には高速リフレッシュも行うことができます。したがって、このタイプのマテリアライズド・ビューには、単一表集計の場合と同じプロシージャを適用することをお勧めします。つまり、1 つのタイプの変更（ダイレクト・パス・ロードまたは DML）を実行してから、マテリアライズド・ビューをリフレッシュします。これは、Oracle が 1 つのタイプの変更のみ実行されたことを検出した場合に、顕著な最適化を実行できるためです。

また、すべての表をロードしてからリフレッシュを実行するのではなく、各表のロードの後にリフレッシュを起動することをお勧めします。したがって、リフレッシュ手順では、次の使用例 2 を使用するようにしてください。

### 使用例 1

fact 表に対する変更  
store に対する変更  
detail\_fact\_mv のリフレッシュ

## 使用例 2

```
fact 表に対する変更
detail_fact_mv のリフレッシュ
store 表に対する変更
detail_fact_mv のリフレッシュ
```

ON-COMMIT のリフレッシュの場合、Oracle は、コミットされたトランザクションで実行された DML のタイプを追跡します。したがって、Oracle がリフレッシュ・フェーズを最適化できない可能性があるので、同じトランザクションで別の表に対してダイレクト・パスや標準 DML を実行しないことをお勧めします。たとえば、次の例はお勧めしません。

```
fact 表に対してダイレクト・パスで新しいデータ挿入
store 表に対して標準 DML
commit
```

可能な場合は、異なるタイプの標準 DML 文を混在させないようにしてください。これによって、高速リフレッシュ時にさまざまな最適化が使用されるのを防ぎます。たとえば、次の例は避けてください。

```
insert into fact ..
delete from fact ..
commit
```

多くの更新が必要な場合は、リフレッシュが各更新の後ではなくコミット時に 1 回だけ実行されるように、それらをすべて 1 つのトランザクションにグループ化するようにしてください。

## 使用例 1

```
fact 表の更新
commit
fact 表の更新
commit
fact 表の更新
commit
```

## 使用例 2

```
fact 表の更新
fact 表の更新
fact 表の更新
commit
```

DBMS\_MVIEW パッケージを使用して、"atomic" パラメータを TRUE に設定して結合のみを含む多数のマテリアライズド・ビューをリフレッシュするときに、パラレル DML が使用禁止にされていると、リフレッシュのパフォーマンスが悪くなることがあります。

データ・ウェアハウス環境で、マテリアライズド・ビューにパラレル句がある場合は、次の一連のステップをお勧めします。

1. fact 表への大量ロード
2. ALTER SESSION ENABLE PARALLEL DML;
3. マテリアライズド・ビューのリフレッシュ

## 並列性のための推奨初期化パラメータ

次のパラメータを推奨します。

- 並列性を考慮するには、PARALLEL\_MAX\_SERVERS に十分に高い値を設定する必要があります。
- SORT\_AREA\_SIZE は、HASH\_AREA\_SIZE より小さくする必要があります。
- OPTIMIZER\_MODE は、CHOOSE（コストベース最適化）と等価にする必要があります。
- OPTIMIZER\_PERCENT\_PARALLEL は、100 と等価にする必要があります。

## リフレッシュの監視

ジョブが実行されているときに、SELECT \* FROM V\$SESSION\_LONGOPS 文はリフレッシュされている各マテリアライズド・ビューの進行状況を通知します。

進行しているジョブがどのキューにあるかを参照するには、SELECT \* FROM DBA\_JOBS\_RUNNING 文を使用します。

表 ALL\_MVIEW\_ANALYSIS には、最後にリフレッシュされた時刻と、完全方式と増分方式の両方を使用したリフレッシュの平均時間の値が、移動平均として含まれます。

リフレッシュは、長時間実行ジョブを最初にスケジュールします。リフレッシュ・ログを使用して、各リフレッシュが何を行ったかをチェックします。

## マテリアライズド・ビューをリフレッシュした後のヒント

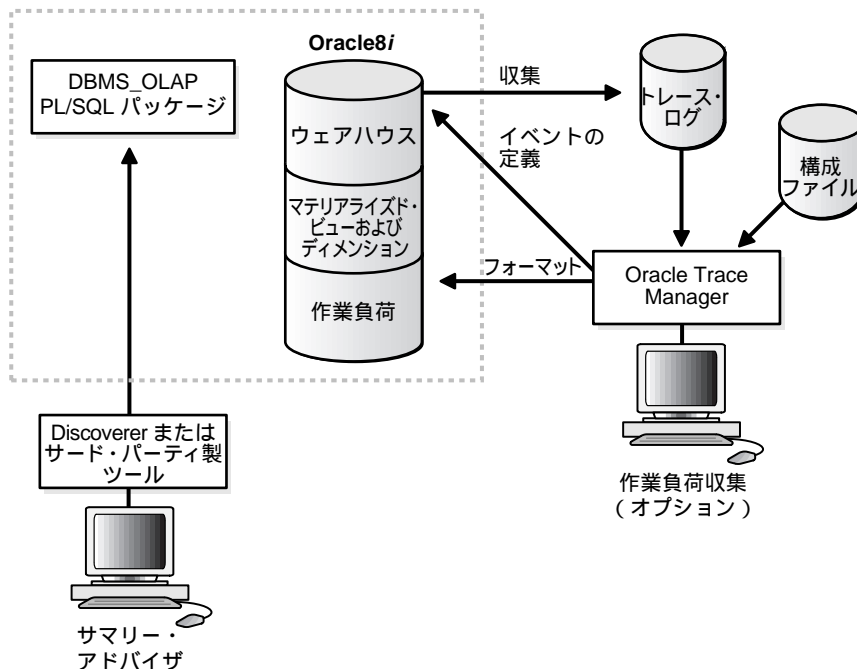
ロードまたは増分ロードを実行し、ディテール表索引を再作成した後で、整合性制約（存在する場合）を再度使用可能にし、ディテール・データから導出されたマテリアライズド・ビューとマテリアライズド・ビュー索引をリフレッシュする必要があります。データ・ウェアハウス環境では、通常は NOVALIDATE または RELY オプションを指定して参照整合性制約が使用可能にされています。リフレッシュ操作を実行する前に行う必要のある重要な決定は、リフレッシュを回復可能にするかどうかということです。マテリアライズド・ビュー・データは重複しており、常にディテール表から再作成できるので、マテリアライズド・ビューについてのロギングは使用禁止にするのが望ましい場合があります。ロギングを使用禁止にして回復不能な増分リフレッシュを実行するには、REFRESH の前に ALTER MATERIALIZED VIEW...NOLOGGING 文を使用します。

マテリアライズド・ビューが ON COMMIT 方式を使用してリフレッシュされる場合は、リフレッシュ操作の後にアラート・ログ (alert\_<SID>.log) とトレース・ファイル (ora\_<SID>\_number.trc) を調べて、エラーが発生していないことを確認する必要があります。

## サマリー・アドバイザー

スキーマで使用可能な多くのマテリアライズド・ビューの中からの選択に役立てるために、Oracle は、一連のマテリアライズド・ビュー分析関数とアドバイザ関数を DBMS\_OLAP パッケージに用意しています。これらの関数は、任意の PL/SQL プログラムからコールできます。

図 32-1 マテリアライズド・ビューとアドバイザ



DBMS\_OLAP パッケージ内のいくつかの機能を使用して、次のことを実行できます。

- マテリアライズド・ビューのサイズを推定します。
- マテリアライズド・ビューを推奨します。



- 収集された作業負荷情報に基づいて、マテリアライズド・ビューを推奨します。
- 収集された作業負荷に基づいて、マテリアライズド・ビューの実際の使用率をレポートします。

マテリアライズド・ビューのサイズのレポートを除き、結果はサマリー・アドバイザが実行されるたびにデータベース内の表に入れられます。したがって、これらは問い合わせることができるので、アドバイザ・プロセスの実行の必要性を低減できます。

## 構造統計の収集

DBMS\_OLAP パッケージのアドバイザ関数では、ファクト表のカーディナリティ、ディメンション表のカーディナリティ、およびすべてのディメンションの LEVEL 列、JOIN KEY 列およびファクト表のキー列の個別のカーディナリティに関する構造統計を収集することを要求されます。これは、データ・ウェアハウスをロードし、DBMS\_STATS パッケージまたは ANALYZE TABLE 文で正確な統計または推定の統計を収集することによって行います。統計収集には時間がかかり、統計は非常に正確である必要はないので、一般には統計を推定する方を選択します。アドバイザは、ディメンションが定義されていない場合には使用できないので、これはディメンションを作成するのに時間を費やすことの正当な理由になります。

## 動的作業負荷統計の収集

オプションとして、Oracle Enterprise Manager Performance Pack を購入した場合は、Oracle Trace を実行して、問合せ作業負荷に関する動的情報も収集し、これをアドバイザ関数で使用できます。Oracle Trace が使用可能な場合は、マテリアライズド・ビューの使用状況の収集に対して重大な考慮事項があります。どのマテリアライズド・ビューが使用されているかを DBA が参照できるようにするだけでなく、アドバイザがユーザーからのまねな問合せ要求を検出することもあるので、いくつかの異なるマテリアライズド・ビューが推奨されます。

Oracle Trace は、マテリアライズド・ビューを分析するために、次の作業負荷統計を収集します。

- クエリー・リライトによって選択された各マテリアライズド・ビューの名前
- マテリアライズド・ビューを使用することによって得られる推定の利益。これは、マテリアライズド・ビューのカーディナリティに対するファクト表のカーディナリティの大きな割合であり、マテリアライズド・ビューをさらに集計したり、他のリレーションに再結合したりする必要性のために調整されています。
- 要求によって使用されたはずの "理想的なマテリアライズド・ビュー"

Oracle Trace には、マテリアライズド・ビューに関するランタイム統計を収集するための 2 つの新しい "ポイント・イベント" が含まれます。その 1 つは、要求実行時に選択されたマテリアライズド・ビュー名を記録するイベントで、もう 1 つは、コンパイル時に推定の利益と理想的なマテリアライズド・ビューを記録するイベントです。必要に応じてマテリアライズド・ビュー分析に対するこれら 2 つのイベントをロギングしたり、他の Trace イベントも収集されている場合には、この情報を Oracle Trace によって収集された他の情報 (SQL テキ

ストや要求の実行時間)と結合したりできます。Oracle Trace Manager GUI の収集オプションは、マテリアライズド・ビュー管理統計を収集する方法を提供します。

サマリー・イベント・セットを収集および分析するには、次のことを行う必要があります。

1. Oracle Trace を介してデータを収集するために、6 つの初期化パラメータを設定します。これらのパラメータを使用可能にすると、データベース接続時に追加オーバーヘッドが発生しますが、それ以外では透過的です。

- ORACLE\_TRACE\_COLLECTION\_NAME = oraclesm
- ORACLE\_TRACE\_COLLECTION\_PATH = 収集ファイルの場所
- ORACLE\_TRACE\_COLLECTION\_SIZE = 0
- ORACLE\_TRACE\_ENABLE = TRUE Trace 収集をオンにします
- ORACLE\_TRACE\_FACILITY\_NAME = oraclesm
- ORACLE\_TRACE\_FACILITY\_PATH = トレース機能ファイルの場所

これらのパラメータの詳細は、『Oracle Trace ユーザーズ・ガイド』を参照してください。

2. Oracle Trace Manager GUI を実行し、収集名を指定して、SUMMARY\_EVENT セットを選択します。Oracle Trace Manager は、関連する構成ファイルから情報を読み込み、ロギングされるイベントを Oracle に登録します。収集が使用可能になっている間は、イベント・セットで定義されている作業負荷情報がフラット・ログ・ファイルに書き込まれます。
3. 収集が完了すると、Oracle Trace は自動的に Oracle Trace ログ・ファイルをリレーションのセットに書式化します。これには、事前定義シノニム V\_192216243\_F\_5\_E\_14\_8\_1 および V\_192216243\_F\_5\_E\_15\_8\_1 があります。作業負荷表は、後続の作業負荷分析が実行されるのと同じスキーマにあることが必要です。または、通常は .CDF 拡張子を持つ収集ファイルは、otrcfmt を使用して手動で書式化できます。手動収集コマンドを次に示します。

```
otrcfmt collection_name.cdf user/password@database
```

4. DBMS\_STATS パッケージの GATHER\_TABLE\_STATS プロシージャまたは ANALYZE...ESTIMATE STATISTICS を実行して、すべてのファクト表、ディメンション表およびキー列 (CREATE DIMENSION 文のディメンション LEVEL 句または JOIN KEY 句に現れる列) に関するカーディナリティ統計を収集します。

これらの 4 つのステップが完了すると、マテリアライズド・ビューに関する推奨を行う準備ができます。

## マテリアライズド・ビューの推奨

マテリアライズド・ビューの分析およびアドバイザ関数は、DBMS\_OLAP パッケージ内の RECOMMEND\_MV および RECOMMEND\_MV\_W です。これらの関数は、作成、保持または削除するマテリアライズド・ビューを自動的に推奨します。

- RECOMMEND\_MV は、構造統計を使用し、作業負荷統計は使用しないで推奨を生成します。
- RECOMMEND\_MV\_W は、作業負荷統計と構造統計の両方を使用します。

これらの関数をコールして、選択、変更または拒否できるマテリアライズド・ビューの推奨リストを取得できます。または、同じ目的のために、PL/SQL プログラムで DBMS\_OLAP パッケージを直接使用することもできます。

サマリー・アドバイザは、次の条件が満たされていないとサマリーを推奨できません。

1. 上記のステップ 4 で説明したように、既存のマテリアライズド・ビューを含むすべての表が分析されている必要があります。
2. ディメンションが存在する必要があります。
3. ファクト表には他の表への外部キー参照が含まれるので、アドバイザはファクト表を識別できる必要があります。

**関連項目：** DBMS\_OLAP パッケージの詳細は、『Oracle8i パッケージ・プロシージャ リファレンス』を参照してください。

これらの関数を使用するには、4 つのパラメータが必要です。

- ファクト表名、またはすべてのファクト表を分析するには NULL
- マテリアライズド・ビューを格納するために使用できる最大記憶領域
- 保持したいマテリアライズド・ビューのリスト
- 保持する必要があるマテリアライズド・ビューのパーセンテージを指定する 0 ~ 100 の数値

メインのファクト表が FACT という名前である場合、パッケージに対する典型的なコールは次のとおりです。

```
DBMS_OLAP.RECOMMEND_MV('fact', 100000, '', 10);
```

この例では、作業負荷統計は使用されていません。

このパッケージのコールの結果は、表 MVIEW\$\$RECOMMENDATIONS に入れられます。この表の内容を問い合わせたり、SQL ファイル sadvdemo.sql を使用して表示したりできます。このプロシージャのコールの出力は、作業負荷統計が使用されている場合も使用されていない場合も同じになります。

推奨は、プロシージャ DEMO\_SUMADV.PRETTYPRINT\_RECOMMENDATIONS をコールすることによって表示できますが、最初に sadvdemo.sql を実行する必要があります。SET SERVEROUTPUT ON SIZE 900000 を使用して、すべての情報が表示できることを確認することをお薦めします。このパッケージをコールした結果の推奨例を次に示します。

### 推奨番号 1

Recommended Action is DROP existing summary GROCERY.QTR\_STORE\_PROMO\_SUM  
Storage in bytes is 196020  
Percent performance gain is null  
Benefit-to-cost ratio is null

### 推奨番号 2

Recommended Action is RETAIN existing summary GROCERY.STORE\_SUM  
Storage in bytes is 21  
Percent performance gain is null  
Benefit-to-cost ratio is null

パッケージをコールして作業負荷統計を使用するには、コールされるプロシージャ名だけが異なります。たとえば、*recommend\_mv* のかわりに *recommend\_mv\_w* を使用します。

```
DBMS_OLAP.RECOMMEND_MV_W('fact', 100000, '', 10);
```

### 推奨番号 3

Recommendation Number = 3  
Recommended Action is CREATE new summary:  
SELECT PROMOTION.PROMOTION\_KEY, STORE.STORE\_KEY, STORE.STORE\_NAME,  
      STORE.DISTRICT, STORE.REGION , COUNT(\*), SUM(FACT.CUSTOMER\_COUNT),  
      COUNT(FACT.CUSTOMER\_COUNT), SUM(FACT.DOLLAR\_COST),  
COUNT(FACT.DOLLAR\_COST),  
      SUM(FACT.DOLLAR\_SALES), COUNT(FACT.DOLLAR\_SALES), MIN(FACT.DOLLAR\_SALES),  
      MAX(FACT.DOLLAR\_SALES), SUM(FACT.RANDOM1), COUNT(FACT.RANDOM1),  
      SUM(FACT.RANDOM2), COUNT(FACT.RANDOM2), SUM(FACT.RANDOM3),  
      COUNT(FACT.RANDOM3), SUM(FACT.UNIT\_SALES), COUNT(FACT.UNIT\_SALES)  
FROM GROCERY.FACT, GROCERY.PROMOTION, GROCERY.STORE  
WHERE FACT.PROMOTION\_KEY = PROMOTION.PROMOTION\_KEY AND FACT.STORE\_KEY =  
      STORE.STORE\_KEY  
GROUP BY PROMOTION.PROMOTION\_KEY, STORE.STORE\_KEY, STORE.STORE\_NAME,  
      STORE.DISTRICT, STORE.REGION  
  
Storage in bytes is 257999.999999976  
Percent performance gain is .533948057298649  
Benefit-to-cost ratio is .00000206956611356085

## 推奨番号 4

Recommended Action is CREATE new summary:

```
SELECT STORE.REGION, TIME.QUARTER, TIME.YEAR , COUNT(*),
      SUM(FACT.CUSTOMER_COUNT), COUNT(FACT.CUSTOMER_COUNT),
SUM(FACT.DOLLAR_COST),
      COUNT(FACT.DOLLAR_COST), SUM(FACT.DOLLAR_SALES),
COUNT(FACT.DOLLAR_SALES),
      MIN(FACT.DOLLAR_SALES), MAX(FACT.DOLLAR_SALES), SUM(FACT.RANDOM1),
COUNT(FACT.RANDOM1), SUM(FACT.RANDOM2), COUNT(FACT.RANDOM2),
SUM(FACT.RANDOM3), COUNT(FACT.RANDOM3), SUM(FACT.UNIT_SALES),
COUNT(FACT.UNIT_SALES)
FROM GROCERY.FACT, GROCERY.STORE, GROCERY.TIME
WHERE FACT.STORE_KEY = STORE.STORE_KEY AND FACT.TIME_KEY = TIME.TIME_KEY
GROUP BY STORE.REGION, TIME.QUARTER, TIME.YEAR
```

Storage in bytes is 86  
Percent performance gain is .523360688578368  
Benefit-to-cost ratio is .00608558940207405

## マテリアライズド・ビューのサイズの見積り

マテリアライズド・ビューはデータベース内の記憶領域を使用するので、マテリアライズド・ビューを作成する前に、必要な記憶容量を知っておくと役に立ちます。表領域内の使用可能な領域の不足を推測したり、作成されるまで待つから不足を検出したりするかわりに、パッケージ DBMS\_ESTIMATE\_SIZE を使用します。このプロシージャをコールすると、マテリアライズド・ビューが占めるサイズの見込みがバイト数で即時に戻されます。

このプロシージャのパラメータは、次のとおりです。

- サイジングの名前
  - SELECT 文
- パッケージは、次のものを戻します。
- マテリアライズド・ビューに予想される行数
  - マテリアライズド・ビューのサイズ (バイト数)

次に示す例では、マテリアライズド・ビューで指定される問合せが ESTIMATE\_SUMMARY\_SIZE パッケージに渡されます。SQL 文は ";" なしで渡されることに注意してください。

```
DBMS_OLAP.estimate_summary_size ('simple_store',
'SELECT
  product_key1, product_key2,
  SUM(dollar_sales) AS sum_dollar_sales,
  SUM(unit_sales) AS sum_unit_sales,
  SUM(dollar_cost) AS sum_dollar_cost,
```

```
SUM(customer_count) AS no_of_customers
FROM fact GROUP BY product_key1, product_key2' ,
no_of_rows, mv_size );
```

プロシージャは、次に示すように、行数の見積りとマテリアライズド・ビューのバイト数の見積りという2つの値を戻します。

```
No of Rows: 17284
Size of Materialized view (bytes): 2281488
```

## マテリアライズド・ビューが使用されているかどうか

マテリアライズド・ビューに関する管理の大きな問題の1つは、マテリアライズド・ビューが使用されているかどうかを知ることです。マテリアライズド・ビューは、定期的に使用されることもあれば、現在は解決されている1回のみの問題のために作成された場合もあります。ただし、このレベルの分析を要求したユーザーグループが、それが不要になったことをDBAに通知することはないので、マテリアライズド・ビューはデータベース内に残り、記憶領域を占め、おそらく定期的にリフレッシュされます。

Oracle Trace オプションが使用可能な場合は、作業負荷統計の収集とまったく同じプロシージャを使用して、どのマテリアライズド・ビューが使用されているかをDBAにアドバイスできます。Trace 収集が使用可能にされ、この場合はTraceが統計の収集中に使用されたマテリアライズド・ビューについてのみレポートするので、問合せ収集の収集周期が長くなる傾向があります。したがって、選択されているウィンドウが小さすぎると、使用されているマテリアライズド・ビューのすべてはレポートされないことがあります。

収集した十分なデータに満足すると、データはOracle Traceによって作業負荷情報と同じように書式化され、パッケージEVALUATE\_UTILIZATION\_Wがコールされます。このパッケージがデータを分析し、その結果を表MVIEWSS\_EVALUATIONSに入れます。

次の例では、マテリアライズド・ビューの使用率が分析され、結果が表示されます。

```
DBMS_OLAP.EVALUATE_UTILIZATION_W();
```

パッケージにはパラメータが渡されていないことに注意してください。

次の情報を提供する表MVIEWSEVALUATIONSを問い合わせることによって取得されたサンプル出力を、その次に示します。

- マテリアライズド・ビューの所有者と名前。
- このマテリアライズド・ビューの利益対コスト率の降順での順位。
- マテリアライズド・ビューのサイズ(バイト数)。
- マテリアライズド・ビューが作業負荷に現れる回数。
- 累積利益は、マテリアライズド・ビューが使用されるたびに計算されます。

- 利益対コスト率は、マテリアライズド・ビューのサイズに対するパフォーマンス改善の増分として計算されます。

MVIEW_OWNER	MVIEW_NAME	RANK	SIZE	FREQ	CUMULATIVE	BENEFIT
GROCERY	STORE_MIN_SUM	1	340	1	9001	26.4735294
GROCERY	STORE_MAX_SUM	2	380	1	9001	23.6868421
GROCERY	STORE_STDCNT_SUM	3	3120	1	3000.38333	.961661325
GROCERY	QTR_STORE_PROMO_SUM	4	196020	2	0	0
GROCERY	STORE_SALES_SUM	5	340	1	0	0
GROCERY	STORE_SUM	6	21	10	0	0





## 数字

---

2 層, 18-10

2 フェーズ・コミット, 26-23

## A

---

ABORTED\_REQUEST\_THRESHOLD プロシージャ, 19-23

ALL\_HISTOGRAMS ビュー, 7-6

ALL\_INDEXES ビュー, 6-15

ALL\_OBJECTS ビュー, 19-30

ALL\_ROWS ヒント, 7-34

ALL\_TAB\_COLUMNS ビュー, 7-5, 7-6

ALTER INDEX REBUILD 文, 6-9

ALTER MATERIALIZED VIEW 文, 29-4  
クエリー・リライトの使用可能化, 31-3

ALTER SESSION 文

SET SESSION\_CACHED\_CURSORS, 19-18  
例, 14-4

ALTER SYSTEM DISCONNECT SESSION, 25-21

ALTER SYSTEM 文

CHECKPOINT オプション, 25-5

MTS\_DISPATCHERS パラメータ, 21-8

ALTER TABLE 文

NOLOGGING オプション, 27-21

ALWAYS\_ANTI\_JOIN パラメータ, 7-19, 7-50, 7-51

ANALYZE 文, 11-4, 20-31, 27-19, 27-25, 32-15

AND\_EQUAL ヒント, 6-7, 7-44

APPEND ヒント, 7-55, 27-21

ARCH プロセス, 21-15

複数の, 26-71

AUTOEXTEND, 20-28

Average Elapsed Time (平均経過時間) データ・  
ビュー, 15-9

## B

---

B\* ツリー索引, 6-15, 6-18

Basic Statistics for Parse/Execute/Fetch (解析 / 実行 /  
フェッチの基本統計) ドリルダウン・データ・  
ビュー, 15-17

BEGIN\_DISCRETE\_TRANSACTION プロシージャ, 9-1, 9-3

BITMAP CONVERSION 行ソース, 6-18

BITMAP\_MERGE\_AREA\_SIZE パラメータ, 6-14, 6-17, 7-20

BITMAP キーワード, 6-15

buffer not pinned 統計, 19-26

buffer pinned 統計, 19-26

BUFFER\_POOL\_name パラメータ, 19-32

BUFFER\_POOL 句, 19-33

## C

---

CACHE ヒント, 7-57

CATPARR.SQL スクリプト, 19-27

CATPERF.SQL ファイル, 19-35

CHECKPOINT オプション

ALTER SYSTEM 文, 25-5

CHOOSE ヒント, 7-36

CKPT プロセス, 20-40

CLUSTER ヒント, 7-39

COMPATIBLE パラメータ, 6-15, 26-62  
パラレル問合せ, 7-51

CONNECT BY, 13-23

Connection Manager, 22-4

マルチスレッド・サーバー, 23-3

consistent gets 統計, 19-25, 21-4, 21-20

count 列、SQL トレース, 14-11

CPU

システム・アーキテクチャ, 18-9  
使用率, 18-1, 26-2  
使用率の検査, 18-3  
チューニング, 18-1  
問題の検出, 18-3  
CPU Statistics (CPU 統計) データ・ビュー, 15-10  
CPU Statistics for Parse/Execute/Fetch (解析 / 実行 /  
フェッチの CPU 統計) ドリルダウン・データ・  
ビュー, 15-17  
CPU\_COUNT 初期化パラメータ, 25-15  
cpu 列、SQL トレース, 14-11  
CREATE CLUSTER 文, 6-25  
CREATE DIMENSION 文, 30-3  
CREATE INDEX 文, 27-19  
NOSORT オプション, 20-37  
例, 20-37  
CREATE MATERIALIZED VIEW 文, 29-4  
クエリー・リライトの使用可能化, 31-3  
CREATE OUTLINE 文, 7-25  
CREATE SNAPSHOT 文, 29-2  
CREATE TABLE AS SELECT, 11-3, 27-17, 27-18, 27-29  
CREATE TABLESPACE 文, 20-23  
CREATE TABLE 文  
STORAGE 句, 20-24  
TABLESPACE 句, 20-24  
CREATE\_BITMAP\_AREA\_SIZE パラメータ, 6-14, 6-17  
CREATE\_STORED\_OUTLINES パラメータ, 7-25  
current 列、SQL トレース, 14-12  
CURSOR\_NUM 列  
TKPROF\_TABLE, 14-17  
CURSOR\_SPACE\_FOR\_TIME パラメータ  
設定, 19-17

## D

---

Data Mart Builder, 28-8  
Data Mart Designer, 28-8  
DATAFILE 句, 20-23  
DATE\_OF\_INSERT 列  
TKPROF\_TABLE, 14-17  
db block gets 統計, 19-25, 21-4, 21-20  
DB\_BLOCK\_BUFFERS パラメータ, 19-28, 19-32, 20-42  
DB\_BLOCK\_LRU\_LATCHES パラメータ, 19-32, 19-37

DB\_BLOCK\_MAX\_DIRTY\_TARGET  
パラメータ, 20-40  
DB\_BLOCK\_SIZE パラメータ  
バックアップのチューニング, 20-52  
パラレル問合せ, 26-27  
DB\_FILE\_DIRECT\_IO\_COUNT  
パラメータ, 20-52  
DB\_FILE\_MULTIBLOCK\_READ\_COUNT パラメータ, 7-19, 20-36, 26-27  
DBA\_DATA\_FILES ビュー, 26-74  
DBA\_EXTENTS ビュー, 26-74  
DBA\_HISTOGRAMS ビュー, 7-6  
DBA\_INDEXES ビュー, 6-15  
DBA\_OBJECTS ビュー, 19-30  
DBA\_TAB\_COLUMNS ビュー, 7-5, 7-6  
DBA ロッキング, 27-13  
DBMS\_APPLICATION\_INFO パッケージ, 5-2, 5-3  
DBMS\_MVIEW.REFRESH\_ALL\_MVIEWS プロシージャ, 32-3  
DBMS\_MVIEW.REFRESH\_DEPENDENT プロシージャ, 32-3  
DBMS\_MVIEW.REFRESH プロシージャ, 32-3  
DBMS\_MVIEW パッケージ, 32-6  
DBMS\_OLAP.RECOMMEND\_MV プロシージャ, 29-21  
DBMS\_OLAP パッケージ, 29-21, 32-15, 32-17  
DBMS\_SHARED\_POOL パッケージ, 10-3, 19-12, 19-23  
DBMS\_STATISTICS パッケージ, 31-3  
DBMS\_STATS パッケージ, 7-8, 32-15  
DBMSPOOL.SQL スクリプト, 10-3, 19-12  
DBMSUTL.SQL, 5-2  
DEPTH 列  
TKPROF\_TABLE, 14-17  
Discoverer, 28-9  
Disk Reads/Execution Ratio (ディスク読み込み / 実行の割合) データ・ビュー, 15-8  
Disk Reads/Logical Reads Ratio (ディスク読み込み / 論理読み込みの割合) データ・ビュー, 15-8  
Disk Reads/Rows Fetched Ratio (ディスク読み込み / 取出し行の割合) データ・ビュー, 15-8  
Disk Reads (ディスク読み込み) データ・ビュー, 15-8  
DISK\_ASYNC\_IO パラメータ, 26-28  
DISKRATIO  
バックアップ I/O の分散, 20-51  
disk 列、SQL トレース, 14-12  
DIUTIL パッケージ, 10-4

DML\_LOCKS パラメータ, 26-24, 26-26  
DOMAIN INDEX  
    EXPLAIN PLAN, 13-21  
DROP MATERIALIZED VIEW 文, 29-4  
    事前作成表, 29-16  
DROP\_BY\_CAT  
    OUTLN\_PKG のプロシージャ, 7-27  
DROP\_UNUSED  
    OUTLN\_PKG のプロシージャ, 7-27  
DSS データベース  
    ディメンション, 30-1  
DSS メモリー, 26-19

## E

---

elapsed 列、SQL トレース, 14-12  
ENFORCED モード, 31-17  
ENQUEUE\_RESOURCES パラメータ, 26-24, 26-26  
Enterprise Manager, 12-7  
EVALUATE\_UTILIZATION\_W パッケージ, 32-20  
Execute Elapsed Time (実行経過時間) データ・  
    ビュー, 15-9  
EXPLAIN PLAN 文, 31-18  
    DOMAIN INDEX, 13-21  
    PLAN\_TABLE, 13-2  
    SQL の分解, 8-5  
    TKPROF プログラムでの起動, 14-9  
    概要, 12-5  
    出力のフォーマット, 13-22  
    出力の例, 4-7, 13-22, 14-14  
    制限事項, 13-25  
    問合せのバラレル化, 26-69  
    パーシャル・パーティション・ワイズ・ジョイン,  
        13-17  
    パーティション・オブジェクト, 13-12  
    バラレル問合せ, 27-28  
    フル・パーティション・ワイズ・ジョイン, 13-19

## F

---

FAST FULL SCAN, 11-3  
FAST\_START\_IO\_TARGET 初期化パラメータ  
    回復時間, 25-4  
    チェックポイントの制御, 20-39  
FAST\_START\_PARALLEL\_ROLLBACK 初期化パラ  
    メータ, 25-15  
FAST\_START\_PARALLEL\_ROLLBACK パラメータ

    回復パラメータ, 26-24  
Fetch Elapsed Time (フェッチ経過時間) データ・  
    ビュー, 15-10  
FILESERSET パラメータ  
    バックアップのチューニング, 20-52  
FIRST\_ROWS ヒント, 7-35, 26-22  
FREELISTS パラメータ, 26-71  
FULL ヒント, 6-7, 7-38

## G

---

GATHER\_INDEX\_STATS  
    DBMS\_STATS のプロシージャ, 7-8  
GATHER\_DATABASE\_STATS  
    DBMS\_STATS のプロシージャ, 7-8  
GATHER\_SCHEMA\_STATS  
    DBMS\_STATS のプロシージャ, 7-8  
GATHER\_TABLE\_STATS  
    DBMS\_STATS のプロシージャ, 7-8  
GC\_FILES\_TO\_LOCKS パラメータ, 27-12  
GC\_ROLLBACK\_LOCKS パラメータ, 27-13  
GC\_ROLLBACK\_SEGMENTS パラメータ, 27-13  
GETMISSES、VSROWCACHE 表, 19-20  
GETS、VSROWCACHE 表, 19-20  
GROUP BY  
    NOSORT, 20-37  
    要求の低減, 27-7  
GV\$FILESTAT ビュー, 26-73

## H

---

HASH\_AJ ヒント, 7-39, 7-50, 7-51  
HASH\_AREA\_SIZE パラメータ, 7-19, 26-19  
    バラレル実行, 26-19, 26-20  
    メモリーとの関係, 27-6  
    例, 27-7  
HASH\_JOIN\_ENABLED パラメータ, 7-19  
HASH\_MULTIBLOCK\_IO\_COUNT パラメータ, 7-20,  
    26-27  
HASH\_SJ ヒント, 7-40, 7-44, 7-51  
HASHKEYS パラメータ  
    CREATE CLUSTER 文, 6-25  
HASH ヒント, 7-39  
HOLD\_CURSOR, 19-10

## I

---

### ID 列

- PLAN\_TABLE 表, 13-4

- INDEX\_ASC ヒント, 7-41

- INDEX\_COMBINE ヒント, 6-7, 6-15

- INDEX\_DESC ヒント, 7-42, 7-43

- INDEX\_FFS ヒント, 6-8, 7-43, 11-3

- INDEX ヒント, 6-7, 6-15, 7-40

- INITIAL エクステント・サイズ, 26-62, 27-12

### INSERT

- 機能, 27-20

- 追加, 7-55

### I/O

- 解析 / 実行 / フェッチの統計, 15-17

- チューニング, 2-10, 20-2

- ディスク・パフォーマンスのテスト, 20-6

- ニーズを分析する, 20-2

- バランス化, 20-22

- パラレル実行, 26-2

- 非同期, 26-28

- 複数バッファ・プール, 19-30

- 不十分な, 17-5

- 分散, 20-20, 20-23

- ボトルネックを回避するためのストライブ化,  
26-35

- I/O の分散, 20-20, 20-23

- ISOLATION LEVEL, 9-5

## K

---

- KEEP プロシージャ, 10-5

## L

---

- LARGE\_POOL\_SIZE, 20-53

- LARGE\_POOL\_SIZE パラメータ, 26-11

### LOB

- 一時, 19-28

- LOG\_BUFFER パラメータ, 19-6, 20-40

- 設定, 21-14

- パラレル実行, 26-24

- LOG\_CHECKPOINT\_INTERVAL 初期化パラメータ,  
20-38

- 回復時間, 25-4

- LOG\_CHECKPOINT\_TIMEOUT 初期化パラメータ,  
20-38

- 回復時間, 25-3

- LOGGING オプション, 26-71

- Logical Reads/Rows Fetched Ratio ( 論理読み込み / 取出  
し行の割合 ) データ・ビュー, 15-8

- Logical Reads ( 論理読み込み ) データ・ビュー, 15-8

### LRU

- 除去方針, 19-30

- ラッチ, 19-32, 19-37

- ラッチの競合, 19-38, 21-18

## M

---

- MAX\_DUMP\_FILE\_SIZE, 14-3

- SQL トレース・パラメータ, 14-3

- MAXEXTENTS キーワード, 26-62, 27-12

- MAXOPENCURSORS, 19-10

- MAXOPENFILES パラメータ

- バックアップのチューニング, 20-52

- MERGE\_AJ ヒント, 7-44, 7-50, 7-51

- MERGE\_SJ ヒント, 7-51

- MERGE ヒント, 7-58

- MIB, 12-5

### MPP

- ディスク親和性, 26-37

### MTS

- 共有プール, 19-20

- 大規模プール, 19-20

- MTS\_DISPATCHERS パラメータ, 21-8

- MTS\_MAX\_DISPATCHERS パラメータ, 21-8, 23-3

- MTS\_MAX\_SERVERS パラメータ, 21-11, 23-4

- MTS\_SERVERS パラメータ, 23-4

- 設定, 21-12

- MULTIBLOCK\_READ\_COUNT パラメータ, 26-62

## N

---

### NAMESPACE 列

- V\$LIBRARYCACHE 表, 19-13

- NEXT エクステント, 27-12

- NO\_EXPAND ヒント, 7-45

- NO\_INDEX ヒント, 7-43

- NO\_MERGE ヒント, 7-58

- NO\_PUSH\_JOIN\_PRED ヒント, 7-60

- NOAPPEND ヒント, 7-55, 27-21

- NOARCHIVELOG モード, 26-72

- NOCACHE ヒント, 7-58

- NOLOGGING オプション, 26-67, 26-71, 27-18,

27-19, 27-21  
NOPARALLEL\_INDEX ヒント, 7-57  
NOPARALLEL 属性, 27-16  
NOPARALLEL ヒント, 7-52  
NOREWRITE ヒント, 7-46, 31-3, 31-20  
NOSORT オプション, 20-37  
NT のパフォーマンス, 24-4  
Number of Rows Processed (処理される行数) データ・ビュー, 15-10

## O

OBJECT\_INSTANCE 列  
PLAN\_TABLE 表, 13-4  
OBJECT\_NAME 列  
PLAN\_TABLE 表, 13-4  
OBJECT\_NODE 列  
PLAN\_TABLE 表, 13-4  
OBJECT\_OWNER 列  
PLAN\_TABLE 表, 13-4  
OBJECT\_TYPE 列  
PLAN\_TABLE 表, 13-4  
OCISmtFetch, 25-23  
OCITransRollback, 25-23  
OPEN\_CURSORS パラメータ  
セッションごとのカーソル数の増加, 19-15  
プライベート SQL 領域の追加割当て, 19-9  
OPERATION 列  
PLAN\_TABLE, 13-3, 13-8  
OPTIMAL 記憶領域パラメータ, 20-30  
OPTIMIZER\_FEATURES\_ENABLED パラメータ, 7-19  
OPTIMIZER\_INDEX\_CACHING  
索引最適化, 7-20  
OPTIMIZER\_INDEX\_COST\_ADJ  
索引最適化, 7-20  
OPTIMIZER\_MODE, 7-3, 7-4, 7-19, 7-22, 7-34, 11-7, 27-25  
OPTIMIZER\_PERCENT\_PARALLEL パラメータ, 7-19, 11-6, 26-22, 27-28  
OPTIMIZER\_SEARCH\_LIMIT パラメータ, 7-20  
OPTIMIZER 列  
PLAN\_TABLE, 13-4  
OPTIONS 列  
PLAN\_TABLE 表, 13-4  
Oracle Designer, 28-8  
Oracle Expert, 2-1, 12-11  
Oracle Express Objects, 28-9  
Oracle Forms, 14-5  
解析とプライベート SQL 領域の制御, 19-10  
Oracle Network Manager, 22-3  
Oracle Parallel Server, 3-8, 11-5  
CPU, 18-12  
ST エンキュー, 27-11  
ディスク親和性, 27-15  
同期点, 2-8  
パラレル実行, 27-12  
パラレル・ロード, 26-64  
Oracle Parallel Server Management (OPSM), 12-11  
Oracle Performance Manager, 12-9  
Oracle Sales Analyzer, 28-9  
Oracle Server  
イベント, 15-5  
クライアント / サーバー構成, 3-9  
構成, 3-6  
Oracle Trace, 15-1, 19-36, 32-15  
「Details」プロパティ・シート, 15-13  
FORMAT 文, 15-18, 15-19  
Oracle Trace Data Viewer, 15-6  
「SQL Statement」プロパティ・シート, 15-13  
START 文, 15-18, 15-19  
STOP 文, 15-18, 15-19  
イベント, 15-4  
期間イベント, 15-5  
コマンド行インタフェース, 15-18  
作業負荷データの収集に使用, 15-3  
収集, 15-4, 15-21  
収集結果, 15-23  
収集したデータへのアクセス, 15-5  
事前定義済みデータ・ビュー, 15-6  
ストアド・プロシージャ, 15-22  
データの表示, 15-11  
データのフォーマット, 15-24  
データ・ビュー, 15-6  
Average Elapsed Time (平均経過時間), 15-9  
CPU Statistics (CPU 統計), 15-10  
ディスク読み込み / 実行の割合, 15-8  
Disk Reads/Logical Reads Ratio (ディスク読み込み / 論理読み込みの割合), 15-8  
Disk Reads/Rows Fetched Ratio (ディスク読み込み / 取出し行の割合), 15-8  
Disk Reads (ディスク読み込み), 15-8  
Execute Elapsed Time (実行経過時間), 15-9  
Fetch Elapsed Time (フェッチ経過時間), 15-10  
Logical Reads/Rows Fetched Ratio (論理読み込み / 取出し行の割合), 15-8

/ 取出し行の割合), 15-8  
Logical Reads (論理読み込み), 15-8  
Number of Rows Processed (処理される行数), 15-10  
Parse Elapsed Time (解析経過時間), 15-9  
Parse/Execution Ratio (解析 / 実行の割合), 15-9  
Re-Parse Frequency (再解析頻度), 15-9  
Rows Fetched/Fetch Count Ratio (取出し行 / フェッチ回数の割合), 15-10  
Rows Sorted (ソートされる行), 15-10  
Sorts in Memory (メモリー内ソート), 15-10  
Sorts on Disk (ディスク上ソート), 15-10  
Total Elapsed Time (合計経過時間), 15-9  
Waits by Average Wait Time (待機 - 平均待機時間), 15-11  
Waits by Event Frequency (待機 - イベント頻度), 15-11  
Waits by Total Wait Time (待機 - 合計待機時間), 15-11  
ドリルダウン・データ・ビュー, 15-15, 15-17  
    Basic Statistics for Parse/Execute/Fetch (解析 / 実行 / フェッチの基本統計), 15-17  
    CPU Statistics for Parse/Execute/Fetch (解析 / 実行 / フェッチの CPU 統計), 15-17  
    Parse Statistics (解析統計), 15-17  
    Row Statistics for Execute/Fetch (実行 / フェッチの行統計), 15-18  
バイナリ・ファイル, 15-5  
パラメータ, 15-20  
    ファイルの削除, 15-20  
    フォーマット設定表, 15-5  
    ポイント・イベント, 15-5  
    レポート作成ユーティリティ, 15-6, 15-24  
Oracle Trace Data Viewer, 15-6  
Oracle Trace Manager, 15-4, 15-21  
    収集のフォーマットに使用, 15-5  
Oracle Trace の「Details」プロパティ・シート, 15-13  
Oracle Trace の FORMAT 文, 15-18, 15-19  
Oracle Trace の「SQL Statement」プロパティ・シート, 15-13  
Oracle Trace の START 文, 15-18, 15-19  
Oracle Trace の STOP 文, 15-18, 15-19  
Oracle Trace のイベント, 15-4  
Oracle Trace の期間イベント, 15-5  
Oracle Trace の収集, 15-4, 15-21  
Oracle Trace の詳細レポート, 15-6

Oracle Trace の初期化パラメータ, 15-20  
Oracle Trace のデータ・ビュー, 15-6  
    Average Elapsed Time (平均経過時間), 15-9  
    CPU Statistics (CPU 統計), 15-10  
    ディスク読み込み / 実行の割合, 15-8  
    Disk Reads/Logical Reads Ratio (ディスク読み込み / 論理読み込みの割合), 15-8  
    Disk Reads/Rows Fetched Ratio (ディスク読み込み / 取出し行の割合), 15-8  
    Disk Reads (ディスク読み込み), 15-8  
    Execute Elapsed Time (実行経過時間), 15-9  
    Fetch Elapsed Time (フェッチ経過時間), 15-10  
    Logical Reads/Rows Fetched Ratio (論理読み込み / 取出し行の割合), 15-8  
    Logical Reads (論理読み込み), 15-8  
    Number of Rows Processed (処理される行数), 15-10  
    Parse Elapsed Time (解析経過時間), 15-9  
    Parse/Execution Ratio (解析 / 実行の割合), 15-9  
    Re-Parse Frequency (再解析頻度), 15-9  
    Rows Fetched/Fetch Count Ratio (取出し行 / フェッチ回数の割合), 15-10  
    Rows Sorted (ソートされる行), 15-10  
    Sorts in Memory (メモリー内ソート), 15-10  
    Sorts on Disk (ディスク上ソート), 15-10  
    Total Elapsed Time (合計経過時間), 15-9  
    Waits by Average Wait Time (待機 - 平均待機時間), 15-11  
    Waits by Event Frequency (待機 - イベント頻度), 15-11  
    Waits by Total Wait Time (待機 - 合計待機時間), 15-11  
Oracle Trace のドリルダウン・データ・ビュー, 15-15  
    Basic Statistics for Parse/Execute/Fetch (解析 / 実行 / フェッチの基本統計), 15-17  
    CPU Statistics for Parse/Execute/Fetch (解析 / 実行 / フェッチの CPU 統計), 15-17  
    Parse Statistics (解析統計), 15-17  
    Row Statistics for Execute/Fetch (実行 / フェッチの行統計), 15-18  
Oracle Trace のフォーマット設定表, 15-5  
Oracle Trace のポイント・イベント, 15-5  
ORACLE\_TRACE\_COLLECTION\_NAME パラメータ, 15-20, 15-21  
ORACLE\_TRACE\_COLLECTION\_PATH パラメータ, 15-20  
ORACLE\_TRACE\_COLLECTION\_SIZE パラメータ,

15-20  
ORACLE\_TRACE\_ENABLE パラメータ, 15-20, 15-21  
ORACLE\_TRACE\_FACILITY\_NAME パラメータ,  
15-20, 15-21  
ORACLE\_TRACE\_FACILITY\_PATH パラメータ,  
15-20  
Oracle ストライプ化, 26-37  
Oracle プリコンパイラ  
解析とプライベート SQL 領域の制御, 19-9  
ORDER BY, 13-23  
要求の低減, 27-7  
ORDERED\_PREDICATES ヒント, 7-61  
ORDERED ヒント, 7-46  
OTHER 列  
PLAN\_TABLE 表, 13-4  
OUTLN\_PKG  
アウトラインを管理するパッケージ, 7-27

## P

---

PARALLEL CREATE INDEX 文, 26-23  
PARALLEL CREATE TABLE AS SELECT, 11-3  
外部断片化, 27-12  
必要なリソース, 26-23  
Parallel Server, 3-8  
ディスク親和性, 27-15  
パラレル実行のチューニング, 27-12  
チューニング, 12-11  
PARALLEL\_ADAPTIVE\_MULTI\_USER パラメータ,  
26-7, 26-30  
PARALLEL\_AUTOMATIC\_TUNING パラメータ,  
26-4  
PARALLEL\_BROADCAST\_ENABLE パラメータ,  
26-22  
PARALLEL\_EXECUTION\_MESSAGE\_SIZE パラメータ,  
26-21  
PARALLEL\_MAX\_SERVERS 初期化パラメータ,  
25-13, 26-10  
PARALLEL\_MAX\_SERVERS パラメータ, 25-13,  
26-9, 26-11, 27-6  
パラレル実行, 26-9  
PARALLEL\_MIN\_PERCENT パラメータ, 26-10,  
26-18  
PARALLEL\_MIN\_SERVERS パラメータ, 26-11  
PARALLEL\_SERVER\_INSTANCES  
パラレル実行, 26-18  
PARALLEL\_THREADS\_PER\_CPU パラメータ, 26-4,

26-8  
PARALLEL 句, 27-21  
RECOVER 文, 25-13  
PARALLEL ヒント, 7-52, 27-16, 27-21, 27-28  
PARENT\_ID 列  
PLAN\_TABLE 表, 13-4  
Parse Elapsed Time (解析経過時間) データ・ビュー,  
15-9  
Parse Statistics (解析統計) ドリルダウン・データ・  
ビュー, 15-17  
Parse/Execution Ratio (解析 / 実行の割合) データ・  
ビュー, 15-9  
PARTITION\_VIEW\_ENABLED パラメータ, 8-6  
PCM ロック, 27-12  
PCTFREE, 2-11, 20-31  
PCTINCREASE パラメータ, 20-35  
および SQL.BSQ ファイル, 20-33  
PCTUSED, 2-11, 20-31  
Performance Manager, 12-9  
PHYRDS 列  
VSFILESTAT 表, 20-19  
physical reads 統計, 19-25  
PHYWRTS 列  
VSFILESTAT 表, 20-19  
ping, 2-11  
ping UNIX コマンド, 12-3  
PINS 列  
VSLIBRARYCACHE 表, 19-14  
PLAN\_TABLE 表  
ID 列, 13-4  
OBJECT\_INSTANCE 列, 13-4  
OBJECT\_NAME 列, 13-4  
OBJECT\_NODE 列, 13-4  
OBJECT\_OWNER 列, 13-4  
OBJECT\_TYPE 列, 13-4  
OPERATION 列, 13-3  
OPTIMIZER 列, 13-4  
OPTIONS 列, 13-4  
OTHER 列, 13-4  
PARENT\_ID 列, 13-4  
POSITION 列, 13-4  
REMARKS 列, 13-3  
SEARCH\_COLUMNS 列, 13-4  
STATEMENT\_ID 列, 13-3  
TIMESTAMP 列, 13-3  
構造体, 13-2  
PL/SQL

PL/SQL 領域のチューニング, 19-7  
パッケージ, 12-6  
POOL 属性, 21-8  
POSITION 列  
    PLAN\_TABLE 表, 13-4  
POST\_TRANSACTION オプション, 25-21  
PQ\_DISTRIBUTE ヒント, 7-53  
PRE\_PAGE\_SGA パラメータ, 19-5  
PRIMARY KEY 制約, 6-10, 27-20  
PRIVATE\_SGA 変数, 23-6  
PRVTPool.PLB, 10-3  
PUSH\_JOIN\_PRED ヒント, 7-59

## Q

---

query 列、SQL トレース, 14-12

## R

---

RAID, 20-25, 26-42, 26-66  
READ\_CLIENT\_INFO  
    DBMS\_APPLICATION\_INFO のプロシージャ, 5-2  
READ\_MODULE  
    DBMS\_APPLICATION\_INFO のプロシージャ, 5-2  
REBUILD, 6-9  
RECOMMEND\_MV\_W 関数, 32-17  
RECOMMEND\_MV 関数, 32-17  
RECOVERY\_PARALLELISM 初期化パラメータ, 25-13  
RECOVER 文  
    PARALLEL 句, 25-13  
RECYCLE キャッシュ, 19-30  
REDO BUFFER ALLOCATION RETRIES, 21-14  
REDO コピー・ラッチ, 21-15  
    数の選択, 21-15  
REDO ログ・バッファのチューニング, 19-6  
REDO ログ・ファイル  
    ディスク上の配置, 20-21  
    ミラー化, 20-21  
REDO 割当てラッチ, 21-15  
Relational Access Manager, 28-9  
RELEASE\_CURSOR, 19-10  
RELOADS 列  
    VSLIBRARYCACHE 表, 19-14  
REMARKS 列  
    PLAN\_TABLE 表, 13-3  
Re-Parse Frequency (再解析頻度) データ・ビュー,  
    15-9

REWRITE ヒント, 7-45, 31-3, 31-20  
RMAN  
    バックアップのチューニング, 20-51  
ROLLBACK\_SEGMENTS パラメータ, 26-24  
Row Statistics for Execute/Fetch (実行 / フェッチの行  
    統計) ドリルダウン・データ・ビュー, 15-18  
ROWID  
    ビットマップへのマップ, 6-16  
ROWID ヒント, 7-39  
Rows Fetched/Fetch Count Ratio ( 取出し行 / フェッ  
    チ回数の割合 ) データ・ビュー, 15-10  
Rows Sorted (ソートされる行) データ・ビュー,  
    15-10  
RowSource イベント, 15-5  
rows 列、SQL トレース, 14-12  
RULE ヒント, 7-36, 27-25

## S

---

sar UNIX コマンド, 18-4, 26-78  
SEARCH\_COLUMN 列  
    PLAN\_TABLE 表, 13-4  
SESSION\_CACHED\_CURSORS パラメータ, 18-7,  
    19-18  
SET TRANSACTION 文, 20-29  
SET\_ACTION  
    DBMS\_APPLICATION\_INFO のプロシージャ, 5-2  
SET\_CLIENT\_INFO  
    DBMS\_APPLICATION\_INFO のプロシージャ, 5-2  
SET\_MODULE  
    DBMS\_APPLICATION\_INFO のプロシージャ, 5-2  
SGA サイズ, 19-7, 26-19  
SGA 統計, 16-2  
SHARED\_POOL\_RESERVED\_SIZE パラメータ, 19-23  
SHARED\_POOL\_SIZE パラメータ, 19-20, 19-24  
    共有プールのチューニング, 19-20  
    ライブラリ・キャッシュの割当て, 19-15  
SHOW SGA 文, 19-5  
SIZES プロシージャ, 10-4  
SMP ( 対称型マルチプロセッサ ), 26-2  
SNMP, 12-5  
SORT\_AREA\_RETAINED\_SIZE パラメータ, 19-38,  
    20-35  
SORT\_AREA\_SIZE パラメータ, 6-14, 7-19, 19-38,  
    26-21  
    ソートのチューニング, 20-34  
    パラレル実行, 26-20



SORT\_MULTIBLOCK\_READ\_COUNT パラメータ ,  
20-36 , 26-27

sorts  
(disk) 統計 , 20-34  
(memory) 統計 , 20-34

Sorts in Memory ( メモリー内ソート ) データ・  
ビュー , 15-10

Sorts on Disk ( ディスク上ソート ) データ・ビュー ,  
15-10

SPIN\_COUNT , 21-2

SQL Loader , 26-59

SQL\*Plus スクリプト , 12-6

SQL\_STATEMENT 列  
TKPROF\_TABLE , 14-16

SQL\_TRACE パラメータ , 14-5

SQL.BSQ ファイル , 20-33

SQLUTLCHAIN.SQL , 12-6

SQL 解析イベント , 15-5

SQL テキスト照合 , 31-6 , 31-21

SQL トレース機能 , 12-6 , 14-2 , 14-5 , 19-8 , 19-36  
解析コール , 19-8  
出力 , 14-11  
出力の例 , 14-14  
使用可能にする , 14-4  
実行するステップ , 14-2  
トレース・ファイル , 12-3 , 14-3 , 14-4  
文の切捨て , 14-13

SQL 文  
アウトラインとマッチングする , 7-24  
効率の悪い , 18-7  
再解析する , 18-6  
再帰的 , 10-1  
索引データの修正 , 6-5  
索引を使う , 6-6  
索引を使わない , 6-7  
チューニング , 2-9  
分解 , 8-3

SQL 文の分解 , 8-3

SQL 領域のチューニング , 19-7

STALE\_TOLERATED モード , 31-17

STANDARD パッケージ , 10-4

STAR\_TRANSFORMATION\_ENABLED パラメータ ,  
7-61 , 11-8

STAR\_TRANSFORMATION ヒント , 7-60 , 11-8

STAR ヒント , 7-47

STATEMENT\_ID 列  
PLAN\_TABLE 表 , 13-3

STORAGE 句  
CREATE TABLE 文 , 20-24  
OPTIMAL , 20-30  
SQL.BSQ の修正 , 20-33  
パラメータの修正 , 20-33  
パラレル問合せ , 27-20  
例 , 20-24

ST エンキュー , 27-11

## T

---

TABLESPACE 句 , 20-24  
CREATE TABLE 文 , 20-24

TAPE\_ASYNC\_IO パラメータ , 26-28

TCP.NODELAY オプション , 22-3

TEMPORARY キーワード , 20-36

TIMED\_STATISTICS パラメータ , 14-3 , 24-1 , 26-74  
SQL トレース , 14-3

TIMESTAMP 列  
PLAN\_TABLE 表 , 13-3

TKPROF\_TABLE , 14-16  
問い合わせる , 14-16

TKPROF プログラム , 14-2 , 14-5 , 19-36  
EXPLAIN PLAN 文の使用 , 14-9  
概要 , 12-6  
構文 , 14-7  
出力 SQL スクリプトの生成 , 14-15  
出力 SQL スクリプトの編集 , 14-15  
出力の例 , 14-14

TNSNAMES.ORA , 25-20

Total Elapsed Time ( 合計経過時間 ) データ・ビュー ,  
15-9

Trace、Oracle , 15-1

TRANSACTIONAL オプション  
SHUTDOWN , 25-21

TRANSACTIONS パラメータ , 26-23

Transparent Gateway , 8-10

TRUSTED モード , 31-17

## U

---

undo block 統計 , 21-3

UNION ALL ビュー , 8-6

UNIQUE 索引 , 6-15

UNIQUE 制約 , 6-10 , 27-20

UNIX システム・パフォーマンス , 24-4

UPDATE\_BY\_CAT

- OUTLN\_PKG のプロシージャ , 7-27 , 7-28
- USE\_CONCAT ヒント , 7-45
- USE\_MERGE ヒント , 7-48
- USE\_NL ヒント , 7-48
- USE\_STORED\_OUTLINES パラメータ , 7-26
- USER\_DUMP\_DEST , 14-3
  - SQL トレース・パラメータ , 14-3
- USER\_HISTOGRAMS ビュー , 7-6
- USER\_ID 列
  - TKPROF\_TABLE , 14-17
- USER\_INDEXES ビュー , 6-15
- USER\_OUTLINE\_HINTS
  - ストアド・アウトライン・ヒントのビュー , 7-26
- USER\_OUTLINES
  - ストアド・アウトラインのビュー , 7-26
- USER\_TAB\_COLUMNS ビュー , 7-5 , 7-6
- UTLBSTAT.SQL , 12-6
- UTLCHAIN.SQL , 20-31
- UTLDTREE.SQL , 12-6
- UTLESTAT.SQ , 12-6
- UTLLOCKT.SQ , 12-6
- UTLXPLAN.SQL , 13-2

## V

---

- V\$BACKUP\_ASYNC\_IO
  - バックアップのチューニング , 20-45
  - ビュー、説明 , 20-46
- V\$BACKUP\_SYNC\_IO
  - バックアップのチューニング , 20-45
  - ビュー、説明 , 20-46
- V\$BH ビュー , 19-27
- V\$BUFFER\_POOL\_STATISTICS ビュー , 19-35 , 19-38
- V\$DATAFILE ビュー , 20-19
- V\$DISPATCHER ビュー , 21-6
- V\$FAST\_START\_SERVERS ビュー , 11-10 , 25-15
- V\$FAST\_START\_TRANSACTIONS ビュー , 11-10 , 25-15
- V\$FILESTAT ビュー
  - PHYRDS 列 , 20-19
  - PHYWRDS 列 , 20-19
  - ディスク I/O , 20-19
  - パラレル問合せ , 26-74
- V\$FIXED\_TABLE ビュー , 16-2
- V\$INSTANCE ビュー , 16-2
- V\$LATCH\_CHILDREN ビュー , 19-37
- V\$LATCH\_MISSES ビュー , 18-9
- V\$LATCH ビュー , 16-2 , 21-2 , 21-16
- V\$LIBRARYCACHE ビュー , 16-2
  - NAMESPACE 列 , 19-13
  - PINS 列 , 19-14
  - RELOADS 列 , 19-14
- V\$LOCK ビュー , 16-3
- V\$MYSTAT ビュー , 16-3
- V\$PARAMETER ビュー , 26-74
- V\$PQ\_SESSTAT ビュー , 26-74 , 27-29
- V\$PQ\_SYSSTAT ビュー , 27-29
- V\$PQ\_TQSTAT ビュー , 26-75 , 27-30
- V\$PROCESS , 16-3
- V\$PX\_PROCESS ビュー , 26-73
- V\$PX\_SESSION ビュー , 26-73
- V\$PX\_SESSTAT ビュー , 26-73
- V\$QUEUE ビュー , 21-7 , 21-9
- V\$RESOURCE\_LIMIT ビュー , 21-2
- V\$ROLLSTAT ビュー , 16-2
- V\$ROWCACHE ビュー , 16-2
  - GETMISSES 列 , 19-20
  - GETS 列 , 19-20
  - 使用 , 19-18
  - パフォーマンス統計 , 19-19
- V\$SESSION\_EVENT ビュー , 16-3
  - ネットワーク情報 , 22-1
- V\$SESSION\_WAIT ビュー , 16-3 , 19-37 , 21-2
  - ネットワーク情報 , 22-1
- V\$SESSION ビュー , 16-3 , 25-21
  - アプリケーションの登録 , 5-1 , 12-7
- V\$SESSTAT ビュー , 16-3 , 18-5 , 26-75 , 26-78
  - 使用 , 19-21
  - ネットワーク情報 , 22-1
- V\$SGASTAT ビュー , 16-2
- V\$SGA ビュー , 16-2
- V\$SHARED\_POOL\_RESERVED ビュー , 19-24
- V\$SORT\_SEGMENT ビュー , 27-12
- V\$SORT\_USAGE ビュー , 4-5 , 16-2
- V\$SQLAREA , 16-2 , 18-6
  - アプリケーションの登録 , 5-1 , 5-5 , 12-7
  - リソース集中型の文 , 4-5
- V\$SQLTEXT ビュー , 16-2
- V\$SYSSTAT ビュー , 16-2 , 18-5 , 18-6 , 26-71 , 26-75
  - REDO バッファ割当て , 21-14
  - 再帰的コールの調査 , 20-27
  - 再実行バッファ割当て再試行 , 26-24
  - 使用 , 19-25
  - ソートのチューニング , 20-34

動的拡張の検出, 20-27  
VSSYSTEM\_EVENT ビュー, 16-2, 18-8, 21-1, 21-2  
V\$WAITSTAT ビュー, 16-2, 21-2  
空きリストの競合の低減, 21-19  
ロールバック・セグメントの競合, 21-3  
VS 動的パフォーマンス・ビュー, 12-5  
vmstat UNIX コマンド, 18-4, 26-78

## W

---

Waits by Average Wait Time (待機 - 平均待機時間)  
データ・ビュー, 15-11  
Waits by Event Frequency (待機 - イベント頻度) データ・ビュー, 15-11  
Waits by Total Wait Time (待機 - 合計待機時間) データ・ビュー, 15-11

## あ

---

アーキテクチャと CPU, 18-9  
アウトライン  
CREATE OUTLINE 文, 7-25  
OUTLN\_PKG で管理する, 7-27  
SQL 文とマッチングする, 7-24  
カテゴリ名を割り当てる, 7-25  
カテゴリを再割当てする, 7-28  
記憶要件, 7-24  
コストベース・オブティマイザへの移行に使用する, 7-29  
削除する, 7-27  
作成する、使用する, 7-25  
作成と使用, 7-25  
使用, 7-26  
使用されていないものを削除する, 7-27  
実行計画とプラン・スタビリティ, 7-23  
データを照会する, 7-26  
表、移動する, 7-28  
ヒント, 7-24  
空きリスト  
競合, 21-19  
競合の低減, 21-20  
追加する, 21-20  
アクセス・パス, 2-9  
アップグレードする  
コストベース・オブティマイザへ, 7-30  
アプリケーション  
OLTP, 3-2

Parallel Server, 3-8  
意思決定支援, 3-3, 26-2  
クライアント / サーバー, 3-9  
データベースへの登録, 5-1, 12-7  
パラレル問合せ, 3-4  
分散データベース, 3-6  
アプリケーション設計, 2-8  
アプリケーションの開発者, 1-8  
アプリケーションの設計者, 1-8  
アプリケーションのフェイルオーバー, 25-16, 25-18  
アラート・ログ, 12-3

## い

---

移行行, 20-31  
意思決定支援, 3-3  
OLTP, 3-5  
システム (DSS), 1-2  
チューニング, 26-2  
問合せの特性, 26-19  
プロセス, 27-3  
意思決定支援システム (DSS)  
ディメンション, 30-1  
依存関係  
ディメンション, 29-20  
マテリアライズド・ビュー, 29-20  
一意性, 6-10  
一時 LOB, 19-28  
一時エクステンツ, 26-66  
一時表領域  
サイズ, 26-66  
ストライプ化, 26-66  
ソートの最適化, 20-36  
一貫性、読み込み, 18-7  
一貫モード、TKPROF, 14-12  
インリスト, 7-41, 7-45

## う

---

ウェアハウス, 28-1  
アドバイザ, 32-2  
リフレッシュ, 29-7, 32-2  
リフレッシュ、ヒント, 32-8

## え

---

エクステンツ

- 一時, 26-66
- サイズ, 26-62
- 無制限の, 20-28

エラー

- 一般的なチューニング, 2-13
- ディスクリット・トランザクション中の, 9-3

## お

---

- 応答時間, 1-2, 1-3
  - 最適化する, 7-3, 7-35
- オーバーヘッド、プロセス, 27-3
- オブティマイザ, 31-2
  - プラン・スタビリティ, 7-23
- オペレーティング・システム
  - 監視ツール, 12-3
  - ストライブ化, 26-36
  - チューニング, 2-11, 17-6, 19-4
  - ディスク I/O の監視, 20-17
  - データ・キャッシュ, 24-2
- オンライン・トランザクション処理 ( OLTP ), 1-2, 3-2
  - 意思決定支援, 3-5
  - プロセス, 27-3

## か

---

- カーディナリティ, 6-19
- 開始列
  - パーティション化と EXPLAIN PLAN, 13-13
- 解析する, 18-6
  - Oracle Forms, 19-10
  - Oracle プリコンパイラ, 19-9
  - 不要なコールの低減, 19-9
- 階層, 30-2
  - 複数の, 30-5
  - ロールアップとドリルダウン, 30-2
- 階層のロールアップ, 30-2
- 回復
  - PARALLEL\_MAX\_SERVERS 初期化パラメータ, 25-13
  - 使用するプロセス数の設定, 25-13
  - チェックポイントの影響, 20-38
  - データ・ウェアハウス, 11-8
  - パラレル
    - トランザクション内回復, 25-15
  - パラレル・プロセス, 25-13

- メディア、ストライブ化, 26-42
- 書込みバッチ・サイズ, 20-42
- 拡張可能な操作, 26-68
- 拡張性, 11-5, 18-8
- 格納
  - ファイル, 20-5
- 仮想メモリー, 26-19
- 偏り
  - 作業負荷, 27-30
- 過負荷のディスク, 20-20
- 監査証跡, 12-4
- 監視
  - パラレル処理, 26-73
- 監視する, 12-5
- 関数ベース索引, 6-11
- 完全リフレッシュ, 32-4
- 管理情報ベース ( MIB ), 12-5
- 外部結合, 31-21

## き

---

- キー, 28-6
- キャッシュ・ヒット率
  - 増加, 19-28
- 競合
  - 空きリスト, 21-19
  - チューニング, 21-1
  - ディスク・アクセス, 20-20
  - メモリー, 19-1
  - メモリー・アクセス, 21-1
  - リソースのチューニング, 2-11
  - ロールバック・セグメント, 21-3
- 共通の結合, 31-8
- 共有 SQL 領域
  - 大きな領域の検出, 10-4
  - 共有プールでの維持, 10-3
  - 考慮される文, 10-1
  - 同一の SQL 文, 10-2
  - メモリー割当て, 19-15
- 共有サーバー
  - 構成, 23-4
  - プロセス, 21-12
- 共有プール, 2-10
  - オブジェクトの確保, 10-3
- 競合, 2-11
- チューニング, 19-10, 19-22
- マルチスレッド・サーバー, 19-20

記録, 2-13  
業務規則, 1-8, 2-3, 2-7

## く

クエリー・リライト  
一致結合グラフ, 29-6  
権限, 31-4  
初期化パラメータ, 31-4  
使用可能にする, 31-3  
正確性, 31-17  
ヒント, 31-3, 31-20  
方法, 31-6  
クライアント / サーバー・アプリケーション, 3-9, 18-4  
クラスタ, 6-23  
グループ化  
互換性チェック, 31-15  
条件, 31-21  
グローバル索引, 26-70

## け

結合, 29-10  
パラレル、PQ\_DISTRIBUTE ヒント, 7-53  
結合互換性, 31-8  
現行モード、TKPROF, 14-12

## こ

更新ウィンドウ, 32-2  
更新頻度, 32-2  
高速全走査, 6-8  
高速リフレッシュ, 32-5  
コールバック  
フェイルオーバー, 25-22  
コストベースの最適化, 7-3, 11-7, 27-25  
アップグレードする, 7-30  
パラレル実行, 27-25  
プラン・スタビリティのプロシージャ, 7-29  
コストベースのリライト, 31-2  
コンテキストのスイッチング, 18-4  
コンテキスト領域, 2-10  
コンボジット・パーティション化, 26-44  
パフォーマンスに関する考慮点, 26-47  
例, 13-15

## さ

サーバー / メモリー / ユーザーの関係, 27-2  
サービス時間, 1-2, 1-3  
再解析する, 18-6  
再帰的 SQL, 10-1  
再帰的コール, 14-12, 20-26  
最大セッション・メモリー統計, 19-21  
最低使用頻度リスト (LRU), 18-7  
最適化  
アプローチと目標の選択, 7-3  
クエリー・リライト  
一致結合グラフ, 29-6  
権限, 31-4  
使用可能にする, 31-3  
ヒント, 31-3, 31-20  
コストベースの, 7-3  
並列を考慮する, 11-6  
ルールベース, 7-22  
作業負荷, 1-6, 18-1  
超過する, 27-5  
調整, 27-8  
不整, 27-30  
索引  
STORAGE 句, 27-20  
値の修正, 6-5  
一意性の施行, 6-10  
一意でない, 6-10  
関数ベース, 6-11  
グローバル, 26-70  
高速全走査, 6-8, 11-3  
再作成, 6-9  
再作成する, 6-8  
作成のタイミング, 6-2  
設計, 2-9  
選択性, 6-4  
使う, 6-6  
使わない, 6-7  
ディスク上の配置, 20-22  
統計、収集, 7-8  
ドメイン、使用, 6-22  
パーティション化, 26-44  
パラレル, 11-3  
パラレル作成, 27-19  
パラレルでの作成, 27-19  
パラレルなローカル索引, 27-19  
ビットマップ, 6-12, 6-13, 6-15, 6-17, 11-7

- 複合, 6-5
- 例, 4-8
- 列の選択, 6-4
- ローカル, 26-70
- 索引の結合, 27-7
- 作成方法, 29-5
- サマリー・アドバイザー, 32-14
- サマリー管理, 28-3
- 参照表, 28-5
  - (ディメンション表), 28-5

## し

---

- 施行された制約, 6-10
- システム・グローバル領域のチューニング, 19-5
- システム固有の Oracle マニュアル
  - USE\_ASYNC\_IO, 26-28
  - ソフトウェアの制約, 17-6
- 集計, 27-17, 28-6, 29-10, 31-21
- 集計可能性チェック, 31-16
- 終了列
  - パーティション化と EXPLAIN PLAN, 13-13
- 手動ストライプ化, 26-37
- 手動リフレッシュ, 32-6
- 初期化パラメータ
  - COMPATIBLE, 28-10, 31-4
  - CPU\_COUNT, 25-15
  - FAST\_START\_PARALLEL\_ROLLBACK, 25-15
  - JOB\_QUEUE\_INTERVAL, 28-9, 32-6
  - JOB\_QUEUE\_PROCESSES, 28-9, 32-6
  - LOG\_CHECKPOINT\_INTERVAL, 25-4
  - LOG\_CHECKPOINT\_TIMEOUT, 25-3
  - MAX\_DUMP\_FILE\_SIZE, 14-3
  - OPTIMIZER\_MODE, 7-22, 7-34, 28-9, 28-10, 31-4, 32-13
  - OPTIMIZER\_PERCENT\_PARALLEL, 28-10, 32-13
  - ORACLE\_TRACE\_COLLECTION\_SIZE, 28-10
  - ORACLE\_TRACE\_ENABLE, 28-10
  - ORACLE\_TRACE\_FACILITY\_NAME, 28-10
  - ORACLE\_TRACE\_FACILITY\_PATH, 28-10
  - PARALLEL\_MAX\_SERVERS, 25-13, 28-10, 32-13
  - PRE\_PAGE\_SGA, 19-5
  - QUERY\_REWRITE\_ENABLED, 28-9, 31-3, 31-4
  - QUERY\_REWRITE\_INTEGRITY, 28-9
  - RECOVERY\_PARALLELISM, 25-13
  - SESSION\_CACHED\_CURSORS, 19-18
  - SORT\_AREA\_SIZE, 28-10, 32-13

- SQL\_TRACE, 14-5
- TIMED\_STATISTICS, 14-3
- USER\_DUMP\_DEST, 14-3
- UTL\_FILE\_DIR, 28-9, 32-6
- 処理、分散, 3-9
- 使用可能にされた制約, 6-10
- 使用禁止, 32-4
- 使用禁止にされた制約, 6-10
- シンプル・ネットワーク管理プロトコル (SNMP), 12-5
- 事後チューニング, 2-3
- 事実, 30-1
- 事前作成マテリアライズド・ビュー, 29-4
- 事前チューニング, 2-1
- 実行計画, 13-1
  - TKPROF, 14-6, 14-9
  - パラレル操作, 27-28
  - プラン・スタビリティ, 7-23
  - プラン・スタビリティで保存する, 7-22
  - 例, 4-7, 14-6
- 需用率, 1-5
- 順次書込み, 20-6
- 順序
  - 保存する, 13-23
- 順序キャッシュ, 2-10
- 順次読込み, 20-6
- ジョイン
  - パーシャル・パーティション・ワイズ, 26-55
  - パーティション式、パーシャル~の例, 13-17
  - パーティション・ワイズ, 26-50
  - パーティション・ワイズ、フル, 13-19
  - パーティション・ワイズ、フル~の例, 13-19
  - フル・パーティション・ワイズ, 26-50

## す

---

- スキーマ・デザイン・ガイド
  - マテリアライズド・ビュー, 28-7
- スキーマ統計
  - 収集, 7-12
- スター・スキーマ, 11-7
- スター問合せ, 11-7
- スター変換, 7-60, 11-8
- ステージング・データベース, 28-1
- ステージング・ファイル, 28-1
- ストアド・アウトライン
  - OUTLN\_PKG で管理する, 7-27

- SQL 文とマッチングする, 7-24
- カテゴリ名を割り当てる, 7-25
- カテゴリを再割り当てする, 7-28
- 記憶要件, 7-24
- 削除する, 7-27
- 作成する、使用する, 7-25
- 使用, 7-26
- 使用されていないものを削除する, 7-27
- 実行計画とプラン・スタビリティ, 7-23
- データを照会する, 7-26
- 表、移動する, 7-28
- ヒント, 7-24
- ストアド・プロシージャ
  - KEEP, 10-5
  - Oracle Trace の, 15-22
  - READ\_MODULE, 5-5
  - SET\_ACTION, 5-3
  - SET\_CLIENT\_INFO, 5-4
  - SET\_MODULE, 5-2
  - SIZES, 10-4
  - データベースへの登録, 5-1, 12-7
- ストライプ化, 20-22, 26-36
  - Oracle, 26-37
  - 一時表領域, 26-66
  - オペレーティング・システム, 26-36
  - オペレーティング・システム・ソフトウェア,  
20-25
  - 手動, 20-23, 26-36, 26-37
  - ディスク親和性, 27-15
  - 分析, 26-39
  - メディア回復, 26-42
  - 例, 20-23, 26-59
  - ローカル, 26-37
- スピン件数, 18-8
- スラッシング, 18-4
- スループット, 1-3
  - 最適化する, 7-3, 7-34
- スレッド, 24-2
- スワッピング, 17-4, 18-4
  - SGA, 19-39
  - 低減, 19-4
  - ライブラリ・キャッシュ, 19-15

## せ

---

- 制約, 6-10, 30-9, 31-20
- セグメント, 20-26

- 設計およびチューニング, 2-9
- 設計ディクショナリ, 12-4
- セッション・データ単位 (SDU), 22-2, 23-5
- セッション・メモリー統計, 19-21
- 接続対ディスパッチャ
  - 推奨の割合, 23-3
- 接続多重化
  - マルチスレッド・サーバー, 23-3
- 接続負荷の調整
  - マルチスレッド・サーバー, 23-5
- 接続プーリング, 21-8
  - マルチスレッド・サーバー, 23-3
- 選択性、索引, 6-4
- 全表走査, 4-8

## そ

---

- 層, 18-10
- 送信時間, 17-5
- ソート
  - 索引作成の回避, 20-37
  - チューニング, 20-33
- ソート領域
  - プロセスのローカル領域, 2-10
  - メモリー割当て, 20-34
- 属性, 30-4

## た

---

- 帯域幅, 26-2
- 待機時間, 1-3, 1-4, 27-5
- 多重化
  - マルチスレッド・サーバー, 23-1
- 多目的アプリケーション, 3-5
- 単一表集計の要件, 29-13
- 短期待機
  - 定義, 20-46
- 単層, 18-10
- 大規模なパラレル・システム, 26-2
- 大規模プール, 20-53
- ダイレクト・ロード挿入, 26-71
  - 外部断片化, 27-12
- 断片化、外部, 27-12

## ち

---

- チェックポイント

- REDO ログのメンテナンス, 20-38
- チェックポイントの頻度の選択, 20-38
- チューニング, 20-38
- パフォーマンス, 20-38
- チャンネル帯域幅, 17-5
- チューニング
  - CPU, 18-1
  - I/O, 2-10, 20-2
  - MTS のための共有プール, 19-20
  - MTS のための大規模プール, 19-20
  - OLTP アプリケーション, 3-2
  - Parallel Server, 3-8
  - SQL, 2-9
  - SQL と PL/SQL 領域, 19-7
  - アクセス・パス, 2-9
  - アプリケーション設計, 2-8
  - 意思決定支援システム, 3-3
  - オペレーティング・システム, 2-11, 17-6, 19-4
  - 期待値, 1-9
  - 競合, 21-1
  - 共有プール, 19-10, 19-20
  - 業務規則, 2-7
  - クライアント / サーバー・アプリケーション, 3-9
  - システム・グローバル領域 (SGA), 19-5
  - 事後, 2-3
  - 事前, 2-1
  - 人員, 1-7
  - 設計, 2-9
  - ソート, 20-33
  - チェックポイント, 20-38
  - データ設計, 2-7
  - データ・ソース, 12-1
  - データベースの論理構造, 2-9
  - 問合せサーバー, 21-13
  - 登録済みアプリケーションの監視, 5-1, 12-7
  - パラレル実行, 3-4, 26-34
  - 分散データベース, 3-6
  - 方法, 2-1
  - 本番システム, 2-4
  - マルチスレッド・サーバー, 21-5
  - メモリー割当て, 2-10, 19-1, 19-39
  - 目標, 1-9, 2-12
  - 問題の診断, 17-1
  - 要因, 17-1
  - ライブラリ・キャッシュ, 19-13
  - 論理構造, 6-3
- チューニングでの役割, 1-7

- チューニングの期待値, 1-9
- チューニングの目標, 1-9, 2-12
- チューニングの問題の診断, 17-1
- チューニングの利益, 2-3
- チューニング用のソース・データ, 12-1
- 長期待機
  - 定義, 20-46
- 直列可能トランザクション, 9-5

## て

- 低減
  - 競合
    - OS プロセス, 24-2
    - REDO ログ・バッファ・ラッチ, 21-14
    - 共有サーバー, 21-9
    - ディスパッチャ, 21-6
    - 問合せサーバー, 21-14
  - データ・ディクショナリ・キャッシュ・ミス, 19-20
  - バッファ・キャッシュ・ミス, 19-28
  - 不要な解析コール, 19-9
  - ページングとスワッピング, 19-4
  - ライブラリ・キャッシュ・ミス, 19-14
  - ロールバック・セグメントの競合, 21-4
- テスト, 2-12
- ディクショナリ対応の表領域, 20-28
- ディスク
  - I/O の分散, 20-20
  - I/O 要件, 20-4
  - OS ファイル・アクティビティの監視, 20-17
  - REDO ログ・ファイルの配置, 20-21
  - 競合, 20-20, 20-21
  - 競合の低減, 20-20
  - スピード特性, 20-3
  - データ・ファイルの配置, 20-21
  - パフォーマンスのテスト, 20-6
  - 必要な数, 20-4
  - レイアウト・オブション, 20-15
- ディスク親和性
  - MPP, 26-67
  - MPP によって使用禁止にする, 26-37
  - パラレル問合せ, 27-15
- ディスクリット・トランザクション
  - 処理, 9-2, 9-3
  - 用途, 9-1
  - 例, 9-3



- ディスパッチャ対接続
  - 推奨の割合, 23-3
- ディスパッチャ・プロセス (Dnnn), 21-8
- ディテール表, 28-5
  - ファクト表と同じ, 28-5
- ディメンション, 30-1, 30-9, 31-21
  - 意思決定支援システムの, 30-1
  - 依存関係, 29-20
  - 階層
    - ロールアップとドリルダウン, 30-2
  - 検証, 30-9
  - 削除, 30-11
  - 作成する, 30-3
  - 使用, 30-1
  - ディメンション表 (参照表), 28-5
  - 変更, 30-10
- ディメンション表, 11-7, 28-6
  - (参照表), 28-5
  - 正規化, 30-7
- データ
  - 設計、チューニング, 2-7
  - チューニング用のソース, 12-1
  - パーティション化, 26-43
  - 比較, 12-4
  - ボリューム, 12-2
- データ・ウェアハウス, 28-1
  - ANALYZE 文, 11-4
  - Oracle Parallel Server, 11-5
  - 階層
    - ロールアップとドリルダウン, 30-2
  - 回復, 11-8
  - 概要, 11-1
  - 機能, 11-1
  - 索引の平行作成, 11-3
  - スター・スキーマ, 11-7
  - ディメンション表
    - (参照表), 28-5
  - データ・マート, 28-1
  - バックアップ, 11-8
  - パーティション, 11-4
  - パーティション表, 26-45
  - 平行・ロード, 11-4
  - ビットマップ索引, 11-7
  - ファクト表 (ディテール表), 28-5
  - 並列を考慮するオプティマイザ, 11-6
- データ・キャッシュ, 24-2
- データ充足チェック, 31-12

- データ・ソースとしての実行可能コード, 12-4
- データ・ディクショナリ, 12-2
- データ・ディクショナリ・キャッシュ, 2-10, 19-20
- データの分析
  - 平行処理, 26-73
- データファイルのディスク上の配置, 20-21
- データ・ブロック・サイズ, 20-15
- データベース
  - ステージング・データベース, 28-1
  - 統計、収集, 7-14
  - パフファ, 19-28
  - レイアウト, 26-34
- データベース管理者 (DBA), 1-8
- データベース接続イベント, 15-5
- データベースの論理構造, 2-9
- データベースへのアプリケーションの登録, 5-1, 12-7
- データベース・ライター・プロセス (DBWn)
  - チェックポイントでの動作, 20-38
  - チューニング, 18-7, 26-71
- データ・マート, 28-1
- デバイス帯域幅, 17-5
  - 評価する, 20-16
- デバイス待ち時間, 17-5
- デフォルト・キャッシュ, 19-30

## と

- 問合せ
  - 索引を使う, 6-6
  - 索引を使わない, 6-7
  - 分散, 8-1, 8-10
  - 並列性の使用可能化, 26-8
- 問合せ計画, 13-1
- 問合せサーバー・プロセス
  - チューニング, 21-13
- 問合せデルタ結合, 31-10
- 等価結合, 4-9
- 透過的アプリケーション・フェイルオーバー, 25-18
- 統計, 16-2, 27-29, 31-22
  - consistent gets, 19-25, 21-4, 21-20
  - db block gets, 19-25, 21-4
  - DBMS\_STATS での収集, 7-8
  - physical reads, 19-25
  - sorts (disk), 20-34
  - sorts (memory), 20-34
  - undo block, 21-3
  - いつ生成するか, 7-21

- オペレーティング・システム, 26-78
- 共有サーバー・プロセス, 21-9, 21-14
- 現在の設定値, 16-4
- コストベース最適化のために生成する, 7-7
- 最大セッション・メモリー, 19-21
- 生成する, 7-7
- セッション・メモリー, 19-21
- ディスパッチャ・プロセス, 21-6
- 問合せサーバー, 21-13
- 変更率, 16-5
- トランザクション
  - 直列可能, 9-5
  - ディスクリット, 9-1
  - 率, 27-11
  - ロールバック・セグメントの割当て, 20-29
- トランザクション処理モニター, 18-11, 18-12
- トランザクション内回復, 25-15
- トリガー
  - OLTP アプリケーションのチューニング, 4-4
- 同時ユーザー
  - 数の増加, 26-10
- 動的拡張, 20-26
  - 回避, 20-28
- 動的パフォーマンス・ビュー
  - チューニング, 16-1
  - 統計を使用可能にする, 14-3
- ドメイン索引
  - 使用, 6-22
- ドリルダウン, 30-2
  - 階層, 30-2

## な

---

- 内部書込みバッチ・サイズ, 20-42

## ね

---

- ネストされた問合せ, 27-17
- ネスト・ループ・ジョイン, 27-3
- ネットワーク
  - 事前開始プロセス, 22-2
  - 制約, 17-5
  - セッション・データ単位, 22-2, 23-5
  - 帯域幅, 17-5
  - チューニング, 22-1
  - 配列インタフェース, 22-2

- パフォーマンス問題の検出, 22-1
- 問題の解決, 22-2

## は

---

- 配布
  - ヒント, 7-53
- 配列インタフェース, 22-2
- ハッシュ
  - 分散値, 13-7
- ハッシュ結合, 26-20, 27-3
- ハッシュ・パーティション
  - パフォーマンスに関する考慮点, 26-46
- ハッシュ・パーティション化, 13-12, 26-43
  - 例, 13-13
- ハッシュ領域, 2-10, 27-3
- ハッシング, 6-24
- 範囲
  - 分散値, 13-7
- ハンドル
  - ユーザー, 25-22
- バイナリ・ファイル
  - Oracle Trace を使用したフォーマット, 15-5
- バインド変数, 19-16
- バックアップ
  - チューニング, 20-51
  - ディスクのミラー化, 26-42
  - データ・ウェアハウス, 11-8
- バッファ・キャッシュ, 2-10
  - キャッシュ・ミスの低減, 19-28
  - チューニング, 19-24
  - バッファ数の低減, 19-28
  - パーティション化, 19-31
  - メモリー割当て, 19-28
- バッファ取得, 4-5
- バッファ・プール
  - 「RECYCLE」キャッシュ, 19-30
  - 構文, 19-33
  - デフォルト・キャッシュ, 19-30
  - 複数の, 19-29, 19-30
  - 保持キャッシュ, 19-30
- パーシャル・パーティション・ワイズ・ジョイン, 26-55
- パーティション
  - プルニング, 26-48
- パーティション・オブジェクト
  - EXPLAIN PLAN, 13-12

- パーティション化
    - 開始列と終了列, 13-13
    - コンボジット, 26-44
    - コンボジットの例, 13-15
    - 索引, 26-44
    - データ, 26-43
    - ハッシュ, 26-43
    - ハッシュごと, 13-12
    - 範囲ごと, 13-12
    - 分散値, 13-7
    - 例, 13-13
    - レンジ, 26-43
  - パーティション絞込み, 8-6
  - パーティション表, 11-4
    - データ・ウェアハウス, 26-45
    - 例, 26-62
  - パーティション・ビュー, 8-6, 11-4
  - パーティション・ワイズ・ジョイン, 26-50
    - パーシャル, 26-55
    - パーシャル、EXPLAIN PLAN 出力, 13-17
    - パラレル、パフォーマンスに関する考慮点, 26-58
    - フル, 13-19
    - フル、EXPLAIN PLAN 出力, 13-19
    - 利点, 26-57
  - パッケージ
    - DBMS\_APPLICATION\_INFO, 5-2, 5-3
    - DBMS\_SHARED\_POOL, 10-3
    - DBMS\_TRANSACTION, 9-3
    - DIUTIL, 10-4
    - OUTLN\_PKG, 7-27
    - STANDARD, 10-4
    - データベースへの登録, 5-1, 12-7
  - パフォーマンス
    - NT, 24-4
    - OLTP アプリケーション, 3-2
    - Parallel Server, 3-8
    - UNIX ベース・システム, 24-4
    - アプリケーションのさまざまなタイプ, 3-1
    - 意思決定支援アプリケーション, 3-3
    - キーとなる要因, 17-2
    - クライアント / サーバー・アプリケーション, 3-9
    - 登録済みアプリケーションの監視, 5-1, 12-7
    - 評価する, 1-10
    - 分散データベース, 3-6
    - メインフレーム, 24-4
  - パフォーマンス モニター、NT, 18-4
  - パラメータ・ファイル, 12-4
  - パラレル回復, 25-13
  - パラレル結合
    - PQ\_DISTRIBUTE ヒント, 7-53
  - パラレル実行, 3-4
    - I/O パラメータ, 26-26
    - Parallel Server, 27-12
    - SQL の書直し, 27-17
    - 概要, 26-2
    - 計画, 27-28
    - コストベースの最適化, 27-25
    - 最大プロセス, 27-2
    - 作業負荷の調整, 27-8
    - 索引作成, 27-19
    - チューニング, 26-1
    - 問合せサーバー, 21-13
    - 問合せサーバーのチューニング, 21-13
    - パフォーマンスの問題, 27-2
    - パフォーマンス問題の検出, 27-1
    - ヒント, 7-52
    - 物理データベース・レイアウトのチューニング, 26-34
    - プロセスの分類, 26-35, 26-37, 26-67, 27-4
    - 問題を解決する, 27-16
    - リソース・パラメータ, 26-19
    - 領域管理, 27-11
  - パラレルでの索引, 27-19
    - 作成, 11-3
  - パラレル・パーティション・ワイズ・ジョイン
    - パフォーマンスに関する考慮点, 26-58
  - パラレル・ロード, 11-4
    - Oracle Parallel Server, 26-64
    - 使用, 26-59
    - 例, 26-64
- ## ひ
- 
- ヒストグラム
    - 作成する, 7-4
    - バケット数, 7-5
    - 表示する, 7-6
  - 非同期 I/O, 26-28
  - 表
    - ストライプ化の例, 20-23
    - ディスク上の配置, 20-22
    - ディテール表, 28-5
    - ディメンション表 (参照表), 28-5
    - ファクト表, 28-5

- フォーマット定義、Oracle Trace , 15-5
- 並列性の使用可能化 , 26-8
- 表統計
  - 収集 , 7-10
- 表のキュー , 26-75
- 表領域
  - 一時 , 20-36
  - 作成、例 , 26-60
  - 専用の一時表領域 , 26-66
  - ディクショナリ対応 , 20-28
- ヒント , 7-32
  - ALL\_ROWS , 7-34
  - AND\_EQUAL , 6-7 , 7-44
  - CACHE , 7-57
  - CLUSTER , 7-39
  - FIRST\_ROWS , 7-35
  - FULL , 6-7 , 7-38
  - HASH , 7-39
  - HASH\_AJ , 7-39 , 7-40 , 7-44 , 7-50
  - HASH\_SJ , 7-51
  - INDEX , 6-7 , 7-40 , 7-47
  - INDEX\_ASC , 7-41
  - INDEX\_DESC , 7-42 , 7-43
  - INDEX\_FFS , 7-43
  - MERGE\_AJ , 7-44 , 7-50
  - MERGE\_SJ , 7-51
  - NO\_MERGE , 7-58
  - NOCACHE , 7-58
  - NOPARALLEL ヒント , 7-52
  - ORDERED , 7-46 , 7-47
  - PARALLEL ヒント , 7-52
  - PQ\_DISTRIBUTE , 7-53
  - PUSH\_SUBQ , 7-60
  - ROWID , 7-39
  - RULE , 7-36
  - STAR , 7-47
  - USE\_CONCAT , 7-45
  - USE\_MERGE , 7-48
  - USE\_NL , 7-48
  - アウトラインで使用する , 7-24
  - アクセス方法 , 7-37
  - クエリー・リライト , 31-3 , 31-20
  - 結合操作 , 7-47
  - 最適化のアプローチと目標 , 7-34
  - 使用方法 , 7-32
  - パラレル問合せオプション , 7-51
  - 並列度 , 7-51

- ビットマップ
  - ROWID へのマップ , 6-16
- ビットマップ索引 , 6-13 , 6-17 , 11-7
  - INLIST ITERATOR , 13-21
  - 記憶領域に関する考慮事項 , 6-13
  - サイズ , 6-19
  - 作成する , 6-15
  - メンテナンス , 6-14
  - 用途 , 6-12
- ビュー
  - USER\_OUTLINE\_HINTS , 7-26
  - USER\_OUTLINES , 7-26
  - V\$FAST\_START\_SERVERS , 25-15
  - V\$FAST\_START\_TRANSACTIONS , 25-15
  - インスタンス・レベル , 16-2
  - チューニング , 16-1
  - マテリアライズド・ビュー
    - 依存関係 , 29-20
  - マルチスレッド・サーバー , 23-7

## ふ

- ファースト・スタート・オンデマンド・ロールバック , 25-14
- ファースト・スタート・チェックポイント
  - FAST\_START\_IO\_TARGET 初期化パラメータ , 25-4
  - LOG\_CHECKPOINT\_INTERVAL 初期化パラメータ , 25-4
  - LOG\_CHECKPOINT\_TIMEOUT 初期化パラメータ , 25-3
  - チェックポイントの制御 , 20-39
- ファースト・スタート・パラレル・ロールバック , 25-14
- ファイル格納、設計 , 20-5
- ファクト表 , 11-7 , 28-5
  - ディテール表に同じ , 28-5
- フェイルオーバー , 11-5 , 25-16
  - BASIC , 25-20
  - METHOD , 25-20
  - PRECONNECT , 25-20
  - TYPE , 25-20
  - アプリケーション , 25-18
  - コールバック , 25-22
  - 制限事項 , 25-23
  - チューニング , 25-21
- フォーマット

- EXPLAIN PLAN 出力, 13-22
- 複合索引, 6-5
- 複数層システム, 18-11
- 複数のアーカイバ・プロセス, 26-71
- 複数の階層, 30-5
- 複数バッファ・プール, 19-29, 19-30, 19-33
- 複数ユーザー・ハンドル, 25-22
- 副問合せ
  - 相関, 27-17
- フル・パーティション・ワイズ・ジョイン, 13-19, 26-50
- 物理データベース・レイアウト, 26-34
- ブロードキャスト
  - 分散値, 13-7
- ブロック・サイズ, 20-15
- ブロックの競合, 2-11
- 分散データベース, 3-6
- 分散問合せ, 8-1, 8-10
- 分析ディクショナリ, 12-4
- プライベート SQL 領域, 19-9
- プラン・スタビリティ, 7-23
  - コストベース・オプティマイザのプロシージャ, 7-29
  - 実行計画を保存する, 7-22
  - 制限事項, 7-23
  - ヒントの使用, 7-23
- ブルニング
  - DATE 列の使用, 26-49
  - パーティション, 26-48
- プロセス
  - DSS, 27-3
  - OLTP, 27-3
  - オーバーヘッド, 27-3
  - 最大数, 17-6, 27-2
  - 事前開始, 22-2
  - スケジューラ, 24-2
  - スケジューリング, 18-4
  - ディスパッチャ・プロセス構成, 21-8
  - パラレル処理でのメモリー競合, 26-10
  - パラレル実行のクラス, 26-35, 26-37, 26-67, 27-4
  - パラレル問合せの最大数, 27-2
  - 優先順位, 24-2
- プロセスのスイッチング, 18-4

## へ

---

並列性

- 程度、上書き, 27-16
- 表と問合せでの使用可能化, 26-8
- 並列度の設定, 26-7
- 並列度
  - 設定, 26-7
  - マルチユーザー問合せ調整, 26-7
- 並列を考慮するオプティマイザ, 11-6
- ページ・テーブル, 18-4
- ページング, 17-4, 18-4, 26-78, 27-5, 27-29
  - SGA, 19-39
  - サブシステム, 27-5
  - 低減, 19-4
  - ライブラリ・キャッシュ, 19-15
  - 率, 26-19

## ほ

---

方法

- チューニング, 2-1
- チューニングのステップ, 2-4
- 適用, 2-12

保持キャッシュ, 19-30

ボトルネック

- ディスク I/O, 20-20
- メモリー, 19-1

## ま

---

マテリアライズド・ビュー, 11-2

- 依存関係, 29-20
- 依存のリフレッシュ, 32-8
- クエリー・リライト
  - 一致結合グラフ, 29-6
  - 権限, 31-4
  - 初期化パラメータ, 31-4
  - ヒント, 31-3, 31-20
- 結合のみを含む, 29-13
- サイズの見積り, 32-19
- 削除, 29-16, 29-22
- 作成する, 29-3
- 作成するビューの選択, 29-21
- 作成方法, 29-5
- 使用, 28-2
- 事前作成, 29-4
- スキーマ・デザイン・ガイド, 28-7
- すべてリフレッシュ, 32-7
- 制限事項, 29-6

- セキュリティ, 29-21
- 単一表集計, 29-12
- デルタ結合, 31-11
- 登録, 29-15
- パーティション化, 29-17
- 命名, 29-4
- リフレッシュ, 29-7
- リフレッシュ・オプション, 29-7
- リライト
  - 使用可能にする, 31-3
- マテリアライズド・ビューのサイズの見積り, 32-19
- マルチスレッド・サーバー, 27-3
  - 競合の低減, 21-5
  - 共有プール, 19-20
  - コンテキスト領域のサイズ, 2-10
  - 設定, 23-1
  - 接続多重化, 23-3
  - 接続プーリング, 23-3
  - 大規模プール, 19-20
  - チューニング, 21-5, 23-1
  - 定義, 23-1
  - ディスパッチャの構成, 23-3
  - 統計を含むビュー, 23-7
  - パフォーマンス問題, 23-8
  - メモリーのチューニング, 23-6
  - 利点, 23-1
- マルチブロック読み込み, 20-27
- マルチユーザー問合せ調整
  - アルゴリズム, 26-7
  - 定義, 26-7

## み

---

- ミラー化
  - REDO ログ・ファイル, 20-21
  - ディスク, 26-42

## む

---

- 無制限のエクステント, 20-28

## め

---

- メッセージ割合, 17-5
- メディア回復, 26-66
- メモリー
  - 2つのレベルでの構成, 26-19

- 仮想, 26-19
- 使用量の低減, 19-39
- チューニング, 2-10
- 不十分な, 17-4
- プロセスの分類, 27-3
- メモリー / ユーザー / サーバーの関係, 27-2
- メモリー割当て
  - MTS, 19-20
  - 共有 SQL 領域, 19-15
  - 重要性, 19-1
  - ソート領域, 20-34
  - チューニング, 19-1, 19-39
  - バッファ・キャッシュ, 19-28
  - ユーザー, 19-6
  - ライブラリ・キャッシュ, 19-15

## ゆ

---

- ユーザー
  - ハンドル, 25-22
  - メモリー割当て, 19-7
- ユーザー / サーバー / メモリーの関係, 27-2
- ユーザー・リソース
  - 制限, 26-10

## よ

---

- 読み込み書き込み操作, 20-6
- 読み込みの一貫性, 18-7

## ら

---

- ライブラリ・キャッシュ, 2-10
  - チューニング, 19-13
  - メモリー割当て, 19-15
- ラウンドロビン
  - 分散値, 13-7
- ラッチ
  - REDO コピー・ラッチ, 21-15
  - REDO 割当てラッチ, 21-15
  - 競合, 2-11, 18-8
- ランダム書き込み, 20-6
- ランダム読み込み, 20-6

## り

---

- リグレッション, 27-27, 27-29

リスニング・キュー, 22-3  
リソース  
    オーバーサブスクライブ, 27-5, 27-9  
    競合のチューニング, 2-11  
    消費、影響するパラメータ, 26-19  
    制限, 26-9  
    追加する, 1-4  
    消費、パラレル DML/DDL に影響するパラメータ, 26-23  
    パラレル問合せの使用, 26-19  
    ユーザーに対する制限, 26-10  
リソースのオーバーサブスクライブ, 27-5, 27-9  
リフレッシュ・オプション, 29-7  
リモート SQL 文, 8-1  
領域管理, 26-66  
    トランザクションの削減, 27-11  
    パラレル実行, 27-11  
リライト  
    権限, 31-4  
    初期化パラメータ, 31-4  
    問合せの最適化  
        一致結合グラフ, 29-6  
        ヒント, 31-3, 31-20  
    ヒント, 31-20

## る

---

ルールベースの最適化, 7-22

## れ

---

例  
    ALTER SESSION 文, 14-4  
    CREATE INDEX 文, 20-37  
    DATAFILE 句, 20-23  
    EXPLAIN PLAN 出力, 4-7, 13-22, 14-14  
    NOSORT オプション, 20-37  
    SET TRANSACTION 文, 20-29  
    SQL トレース機能の出力, 14-14  
    STORAGE 句, 20-24  
    TABLESPACE 句, 20-24  
    索引問合せ, 4-8  
    実行計画, 4-7  
    全表走査, 4-8  
    ディスクリット・トランザクション, 9-3  
    表のストライプ化, 20-23  
列、索引付け, 6-4

連鎖行, 20-31  
レンジ・パーティション化, 13-12, 26-43  
    パフォーマンスに関する考慮点, 26-44  
    例, 13-13

## ろ

---

ローカル索引, 26-70  
ローカルのストライプ化, 26-37  
ロー・デバイス, 24-2  
ロード  
    パラレル, 11-4, 26-64  
ロード・バランシング, 11-5, 20-22  
    マルチスレッド・サーバー, 23-5  
ロールバック  
    ファースト・スタート・オンデマンド, 25-14  
    ファースト・スタート・パラレル, 25-14  
ロールバック・セグメント, 18-7, 26-24  
数の選択, 21-4  
競合, 21-3  
作成する, 21-4  
トランザクションへの割当て, 20-29  
動的拡張, 20-29  
動的拡張の検出, 20-26  
ログ, 21-14  
ログ・バッファのチューニング, 2-10, 19-7  
ログ・ライター・プロセス (LGWR) のチューニング, 20-21, 20-40  
ロックの競合, 2-11

## わ

---

割当て  
    マルチスレッド・サーバー, 19-20  
    メモリーの, 19-1

