Oracle9*i*

データベース・パフォーマンス・チューニング・ガイドおよびリファレンス

リリース 2（9.2）

2003 年 2 月

部品番号：J06248-02

ORACLE®

Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス , リリース 2（9.2）

部品番号：J06248-02

原本名：Oracle9*i* Database Performance Tuning Guide and Reference, Release 2 (9.2)

原本部品番号：A96534-02（Vol.1）、A96535-02（Vol.2）

原本著者：Connie Dialeris Green

原本協力者：Valarie Moore, James Barlow, Eric Belden, Qiang Cao, Sumanta Chatterjee, Benoit Dageville, Vinayagam Djegaradjane, Harvey Eneman, Bjorn Engsig, Cecilia Gervasio, Ray Glasstone, Leslie Gloyd, Lester Gutierrez, Karl Haas, Brian Hirano, Andrew Holdsworth, Mamdouh Ibrahim, Christopher Jones, Srinivas Kareenhalli, Stella Kister, Herve Lejeune, Yunrui Li, Juan Loaiza, George Lumpkin, Joe McDonald, Bill McKenna, Sujatha Muthulingam, Gary Ngai, Michael Orlowski, Kant C. Patel, Richard Powell, Shankar Raman, Vinay Srihari, Sankar Subramanian, Margaret Susairaj, Hal Takahara, Nitin Vengurlekar, Stephen Vivian, Simon Watt, Andrew Witkowski, Graham Wood and Mohamed Zait.

目次

2 オプティマイザ操作

3 オプティマイザ統計の収集

4 索引およびクラスタ

5 オプティマイザ・ヒント

6 SQL 文の最適化

7 プラン・スタビリティの使用方法

8 ルールベース・オプティマイザの使用

第 II 部 SQL 関連のパフォーマンス・ツール

9 EXPLAIN PLAN の使用方法

10 SQL トレースおよび TKPROF の使用

11 SQL*Plus での自動トレースの使用

12 Oracle Trace の使用

第 III 部　優れたパフォーマンスを得るためのデータベースの作成

13　パフォーマンスを考慮したデータベースの作成

14　メモリーの構成と使用方法

15 I/O 構成および設計

16 オペレーティング・システム・リソース

17 インスタンス・リカバリ・パフォーマンスの構成

第 IV 部　システム関連のパフォーマンス・ツール

20　データベース統計収集用の Oracle のツール製品

21　Statspack の使用方法

第 V 部　インスタンスのパフォーマンスの最適化

22　インスタンスのチューニング

23 ネットワークのチューニング

第 VI 部 パフォーマンス関連の参照情報

24 チューニングに使用する動的パフォーマンス・ビュー

はじめに

この章には、次の項目が含まれます。

対象読者

『Oracle9*i* データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』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。このマニュアルでは、パフォーマンス・ツールを使用して、SQL を適切に作成およびチューニングし、インスタンスのパフォーマンスを最適化して、Oracle パフォーマンスを向上させる方法を詳しく説明します。また、優れたパフォーマンスを実現するための初期データベースの作成方法を説明するとともに、パフォーマンス関連の参照情報を示します。

データベース管理者、アプリケーション設計者およびプログラマに役立つガイドです。読者が、Oracle9*i*、『Oracle9*i* データベース・パフォーマンス・プランニング』、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いることを前提としています。

多くのクライアント / サーバー・アプリケーションのプログラマが SQL をメッセージ言語とみなすのは、問合せが発行され、データが戻されるためです。しかし、クライアント・ツールでは非効率的な SQL 文が生成される場合がよくあります。したがって、データベース SQL 処理エンジンについて理解することは、最適な SQL を作成するために必要です。このことは特に、大量トランザクション処理システムについて言えます。

一般に、OLTP アプリケーションから発行される SQL 文では、一度にわずかな行しか実行されません。要求する行を索引で正確に指示できれば、最短のパスからこれらの行に効率よくアクセスする計画を作成できます。DSS 環境では、表の行のほとんどにアクセスする場合が多いため、選択性は重要視されません。そのような状況では、全表スキャンが一般的であり、索引は使用しません。このマニュアルは、主として、OLTP タイプのアプリケーションを中心に説明しています。DSS 環境、および DSS と OLTP の混合環境の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

このマニュアルを読む前に、『Oracle9*i* データベース・パフォーマンス・プランニング』を読んでおくことをお薦めします。オラクル社では、長年にわたる設計およびパフォーマンス経験に基づき、新しいパフォーマンス手法を設計しました。このマニュアルでは、システム・パフォーマンスを大幅に向上させる明瞭で簡単なアクティビティについて説明します。次の項目について説明しています。

- 投資対効果
- 拡張性
- システム・アーキテクチャ
- アプリケーション設計の原理
- ワークロードのテスト、モデル化および実装
- 新規アプリケーションの配布

このマニュアルの構成

このマニュアルには、次の章が含まれます。

第 I 部「SQL の作成とチューニング」

第 I 部では、SQL 文の理解と管理に役立つ情報を示します。

第 1 章「オプティマイザの概要」

SQL 処理、Oracle による最適化および SQL 文の実行を Oracle オプティマイザが選択する方法について説明します。

第 2 章「オプティマイザ操作」

CBO により提供される特定の操作について詳細に説明します。

第 3 章「オプティマイザ統計の収集」

コストベース・オプティマイザにとって統計が重要である理由、および統計の収集方法と使用方法を説明します。

第 4 章「索引およびクラスタ」

索引とクラスタの作成方法と、どのような場合にそれらを使用するかを説明します。

第 5 章「オプティマイザ・ヒント」

コストベース・オプティマイザのヒントを使用して Oracle パフォーマンスを拡張する方法に関する推奨事項を説明します。

第 6 章「SQL 文の最適化」

コストベース・オプティマイザ（CBO）を使用して、Oracle が SQL を最適化する方法を説明します。

第 7 章「プラン・スタビリティの使用方法」

パフォーマンス特性を維持するためにプラン・スタビリティ（ストアド・アウトライン）を使用する方法を説明します。

第 8 章「ルールベース・オプティマイザの使用」

この章では、Oracle のルールベース・オプティマイザ（RBO）について説明します。

第 II 部「SQL 関連のパフォーマンス・ツール」

第 II 部では、Oracle SQL 関連のパフォーマンス・ツールに関する情報を示します。

第 9 章「EXPLAIN PLAN の使用方法」

SQL 文 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 10 章「SQL トレースおよび TKPROF の使用」

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 11 章「SQL*Plus での自動トレースの使用」

SQL オプティマイザで使用される実行パスに関するレポートを、自動的に取得できる自動トレースの使用方法と、文のパフォーマンスの監視とチューニングに役立つ、文の実行統計について説明します。

第 12 章「Oracle Trace の使用」

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。

注意： Oracle Trace は、将来のリリースでは使用できなくなる可能性があります。かわりに、SQL トレースおよび TKPROF を使用することをお薦めします。

第 III 部「優れたパフォーマンスを得るためのデータベースの作成」

第 III 部では、優れたパフォーマンス実現のためのデータベースの作成および構成方法について説明します。

第 13 章「パフォーマンスを考慮したデータベースの作成」

意図する要求に合うデータベースの設計および作成方法について説明します。

第 14 章「メモリーの構成と使用方法」

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。

第 15 章「I/O 構成および設計」

基本的な I/O 概念を紹介し、データベースの様々な部分の I/O 要件について説明し、I/O サブシステムの設計のための構成例を示します。

第 16 章「オペレーティング・システム・リソース」

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 17 章「インスタンス・リカバリ・パフォーマンスの構成」
リカバリ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 18 章「UNDO セグメントと一時セグメントの構成」
UNDO セグメントを構成する方法（自動 UNDO 管理またはロールバック・セグメントを使用）と、一時表領域を構成する方法について説明します。

第 19 章「共有サーバーの構成」
ディスパッチャ・プロセスと共有サーバーのための、競合の識別および軽減の方法について説明します。

第 IV 部「システム関連のパフォーマンス・ツール」
第 IV 部では、Oracle のシステム関連のパフォーマンス・ツールに関する情報を示します。

第 20 章「データベース統計収集用の Oracle のツール製品」
Oracle では、パフォーマンス・エンジニアがインスタンスとデータベースのパフォーマンスに関する情報を収集できるようにする多数のツールを提供しています。この章では、パフォーマンス・データの収集が重要な理由を説明するとともに、提供しているツールの使用方法について説明します。

第 21 章「Statspack の使用方法」
Statspack を使用してシステム・データの収集、格納および解析を行う方法について説明します。

第 V 部「インスタンスのパフォーマンスの最適化」
第 V 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・システムの様々な要素をチューニングする方法について説明します。

第 22 章「インスタンスのチューニング」
パフォーマンス・チューニングに使用される手法について説明します。また、Oracle 統計と待機イベントについても説明します。

第 23 章「ネットワークのチューニング」
チューニングに影響を与える様々な接続モデルとネットワーク上の問題点について説明します。

第 VI 部「パフォーマンス関連の参照情報」

第 VI 部では、動的パフォーマンス・ビューおよび待機イベントに関する参照情報を示します。

第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」

システムのチューニングとパフォーマンスの問題の調査を容易にする動的パフォーマンス・ビューに関する詳細情報を示します。

付録

付録 A「パフォーマンスの例で使用されているスキーマ」

第 9 章「EXPLAIN PLAN の使用方法」に示す例で使用されている表について説明します。

関連文書

このマニュアルを読む前に、『Oracle9*i* データベース・パフォーマンス・プランニング』、『Oracle9*i* データベース概要』、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』および『Oracle9*i* データベース管理者ガイド』をお読みください。

Oracle Enterprise Manager とそのオプションのアプリケーションの詳細は、『Oracle Enterprise Manager 概要』、『Oracle Enterprise Manager 管理者ガイド』および『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

データ・ウェアハウス環境をチューニングする方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

このマニュアルに記載されている例の多くは、Oracle のインストール時にデフォルトでインストールされるシード・データベースのサンプル・スキーマを使用しています。これらのスキーマがどのように作成されているか、およびその使用方法については、『Oracle9*i* サンプル・スキーマ』を参照してください。

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

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

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

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

表記規則

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

本文の表記規則

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

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

コード例の表記規則

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

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

次の表は、コード例の記載上の表記規則と使用例を示しています。

表記規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	`DECIMAL (digits [ , precision ])`
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	`{ENABLE | DISABLE}`
|	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	`{ENABLE | DISABLE}` `[COMPRESS | NOCOMPRESS]`
...	水平の省略記号は、次のいずれかを示します。	
	■ 例に直接関係のないコード部分が省略されていること。	`CREATE TABLE ...AS subquery;`
	■ コードの一部が繰返し可能なこと。	`SELECT col1, col2, ..., col`*n*` FROM employees;`

. . .	垂直の省略記号は、例に直接関係のない数行のコードが省略されていることを示します。	`SQL> SELECT NAME FROM V$DATAFILE;`
		`NAME`
		`----------`
		`/fs1/dbs/tbs_01.dbf`
		`/fs1/dbs/tbs_02.dbf`
		`.`
		`.`
		`.`
		`/fs1/dbs/tbs_09.dbf`
		`9 rows selected.`

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

表記規則	意味	例

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

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

Oracle のパフォーマンスの新機能

この項では、Oracle9*i* リリース 2（9.2）の新規パフォーマンス機能について説明すると同時に、追加情報の掲載箇所も記載しています。この項で説明する機能と拡張機能は、データベースのパフォーマンスを最適化することを目標としています。

注意： このマニュアルを読む前に、『Oracle9*i* データベース・パフォーマンス・プランニング』を読んでおくことをお薦めします。オラクル社では、長年にわたる設計およびパフォーマンス経験に基づき、新しいパフォーマンス手法を設計しました。このマニュアルでは、システム・パフォーマンスを大幅に向上させる明瞭で簡単なアクティビティについて説明します。次の項目について説明しています。

- 投資対効果
- 拡張性
- システム・アーキテクチャ
- アプリケーション設計の原理
- ワークロードのテスト、モデル化および実装
- 新規アプリケーションの配布

新規および更新されたパフォーマンス機能は次のとおりです。

- 強制リライト

  新しい設定の FORCE は、QUERY_REWRITE_ENABLED セッション・パラメータで使用可能です。

  QUERY_REWRITE_ENABLED を FORCE に設定すると、Oracle では事前にコストの評価を行わず、常にリライトが使用されます。FORCE は、問合せで常にリライトによる利点がある場合、コンパイル時の減少が重要な場合、およびオプティマイザによりマテリアライズド・ビューの利点が低く評価されている場合に便利です。

  関連項目： 4-9 ページ「QUERY_REWRITE_ENABLED」

- グルーピング・セットによる問合せの UNION-All リライト

  新しいヒント EXPAND_GSET_TO_UNION では、コンパイル時間が重視され問合せのリライトが常に有利に働く（OLAP）場合の問合せにファンクション・ベース索引が使用されていると、強制リライトが使用可能です。

  EXPAND_GSET_TO_UNION ヒントは、グルーピング・セットが含まれる問合せに使用されます（GROUP BY GROUPING SET または GROUP BY ROLLUP などによる問合せ）。ヒントは、問合せを強制的に個々のグルーピングの UNION ALL を持つ対応する問合せに変換します。

  関連項目： このヒントの使用方法の詳細は、5-19 ページの「EXPAND_GSET_TO_UNION」を参照してください。

- オプティマイザの動的サンプリング

  動的サンプリングの目的は、選択性およびカーディナリティの見積りをより正確に判別することにより、サーバー・パフォーマンスを改善することです。選択性およびカーディナリティをより正確に見積ると、オプティマイザでより適切な実行計画を作成できます。

  動的サンプリングを使用すると、次の作業が可能になります。

  - 収集された統計が使用できない、あるいは見積りで重大なエラーを引き起こす可能性がある場合に、単一表の述語の選択性を見積ります。
  - 統計のない表、あるいは統計が古すぎて信頼できない表に対する表カーディナリティを見積ります。

  DYNAMIC_SAMPLING ヒントを使用して、より正確な選択性およびカーディナリティの見積りを判別することにより、動的サンプリングを制御してサーバーのパフォーマンスを改善できます。DYNAMIC_SAMPLING の値は 0 から 10 までの値に設定できます。レベルが高くなるにつれ、コンパイラで動的サンプリングに対してさらに多くの作業が行われ、より広く適用されます。サンプリングでは、表を指定しない場合、カーソル・レベルがデフォルトとなります。

関連項目：

- 動的サンプリングの使用時機および使用方法の詳細は、6-7 ページの「動的サンプリング」を参照してください。
- このヒントの使用方法の詳細は、5-39 ページの「DYNAMIC_SAMPLING」を参照してください。

- ローカル管理 SYSTEM 表領域

  Oracle9*i* リリース 2（9.2）では、ローカル管理 SYSTEM 表領域を持つデータベースが作成できます。CREATE DATABASE 文の EXTENT MANAGEMENT LOCAL 句を使用して、ローカル管理 SYSTEM 表領域を作成します。

  EXTENT MANAGEMENT LOCAL を指定する場合、デフォルトの一時表領域も指定する必要があります。

  関連項目：

  - 13-7 ページ「一時表領域の作成」
  - 13-3 ページ表 13-2「初期作成用データベース・オプション」
  - CREATE DATABASE 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

- データ・セグメント圧縮

  データ・セグメントの圧縮により、ディスク使用およびメモリー使用（具体的には、バッファ・キャッシュ）を削減します。多くの場合、これにより、読取り専用操作のスケールアップがうまく機能します。また、データ・セグメントの圧縮では、問合せの実行が高速化されます。

  Oracle9*i* リリース 2（9.2）では、多くの場合、特別なチューニングなしで、優れた圧縮率を達成しています。ただし、さらに圧縮率を上げる必要がある場合にチューニングを行うと、圧縮率が若干改善される場合もあれば大きく改善される場合もあります。

  関連項目： 13-9 ページ「データ・セグメント圧縮」

- 共有プールのアドバイザ統計

  ライブラリ・キャッシュに使用可能なメモリー量は、Oracle インスタンスの解析率に大きな影響を与えます。Oracle9*i* のリリース 2（9.2）以降では、共有プールのアドバイザ統計により、データベース管理者に、ライブラリ・キャッシュ・メモリーについての情報と、共有プールのサイズ変更が解析率に及ぼす影響の予測が提供されます。

  共有プールのアドバイザ統計では、共有プール・メモリーにおけるライブラリ・キャッシュの使用率が追跡され、異なるサイズの共有プールでライブラリ・キャッシュがどのように動作するかが予測されます。2 つの固定ビューにより、ライブラリ・キャッシュ

のメモリー使用量、現在の確保量、共有プールの LRU リスト上にある量、さらに共有プールのサイズ変更により損失または獲得できる時間を判別する情報が提供されます。

関連項目： 14-29 ページ「共有プールのアドバイザ統計」

- PGA 集計ターゲット・アドバイザ

  自動 PGA メモリー管理モードにおいて Oracle が主な目標としたものは、SQL 作業領域に割り当てられる PGA メモリーの量を動的に制御することで、DBA が設定した PGA_AGGREGATE_TARGET の制限を尊重することです。これと同時に、使用する PGA メモリー（キャッシュ・メモリー）の量が最適である作業領域の数を最大にして、すべてのメモリー集中型 SQL 演算子のパフォーマンスを最大限に引き出す試みも行っています。パラメータ PGA_AGGREGATE_TARGET で DBA により設定された PGA メモリーの制限が低すぎて、マルチ・パスを実行して PGA メモリーの消費をさらに削減して、PGA のターゲット制限を尊重する必要がある場合を除き、残りの作業領域はワン・パス・モードで実行されます。

  新規インスタンスを構成する場合には、PGA_AGGREGATE_TARGET の適切な設定を正確に知ることは困難です。この設定は、次の 3 つの段階を実行して判別します。

  1. PGA_AGGREGATE_TARGET の最初の見積りは経験に基づいて行う。
  2. インスタンスで代理のワークロードを実行し、Oracle により収集された PGA 統計を使用してパフォーマンスを監視して、最大 PGA サイズが高く構成されたか低く構成されたかを確認する。
  3. Oracle PGA のアドバイスの統計を使用して、PGA_AGGREGATE_TARGET をチューニングする。

  関連項目：

  - 14-49 ページ「PGA_AGGREGATE_TARGET の初期設定」
  - 14-49 ページ「自動 PGA メモリー管理のパフォーマンスの監視」
  - 14-57 ページ「PGA_AGGREGATE_TARGET のチューニング」

- FILESYSTEMIO_OPTIONS

  Oracle9*i* リリース 2（9.2）では、FILESYSTEMIO_OPTIONS 初期化パラメータを使用して、ファイル・システムのファイルについて非同期 I/O あるいはダイレクト I/O を有効化することも無効化することもできます。このパラメータは、プラットフォーム固有であり、それぞれのプラットフォームに最適なデフォルト値が設定されています。このパラメータは、動的に変更して、デフォルト設定を更新できます。

  関連項目： 16-4 ページ「FILESYSTEMIO_OPTIONS 初期化パラメータ」

- 平均リカバリ時間（MTTR）アドバイザ

  Oracle9*i* リリース 2（9.2）から、MTTR アドバイザが使用できます。追加の物理書込みに関して、異なる MTTR の設定がシステム・パフォーマンスに与える影響を評価する際に役立ちます。

  MTTR アドバイザが使用可能な場合に、しばらくの間通常のワークロードでシステムを実行して V$MTTR_TARGET_ADVICE を問い合せると、他の MTTR 設定におけるキャッシュ書込み回数の見積りに対する、現在の MTTR 設定におけるキャッシュ書込み回数の比率がわかります。たとえば、比率が 1.2 の場合、キャッシュ書込み回数が 20%多いことを示します。

  異なる MTTR の設定と、それに対応するキャッシュ書込み比率を確認することで、リカバリおよびパフォーマンス要件に適合する MTTR の値を判別できます。V$MTTR_TARGET_ADVICE では、総物理書込み（ダイレクト書込みを含む）の比率と、総 I/O（読込みを含む）の比率が提供されます。

  関連項目：

  - 17-17 ページ「MTTR アドバイザ」
  - 24-22 ページ「V$MTTR_TARGET_ADVICE」

- 統計収集のレベル

  Oracle9*i* リリース 2（9.2）では、初期化パラメータ STATISTICS_LEVEL が提供され、すべての主要統計収集またはアドバイザをデータベースで制御します。このパラメータでは、データベースに統計収集レベルを設定します。

  BASIC、TYPICAL または ALL のレベルがあります。デフォルトは TYPICAL です。

  関連項目： 各レベルで収集される統計の種類の詳細は、22-10 ページの「統計収集のレベルの設定」を参照してください。

- セグメント・レベルの統計

  Oracle9*i* リリース 2（9.2）以上では、個別のセグメントに関連するパフォーマンス問題を確認するために役立つ、セグメント・レベルの統計を収集できます。セグメント・レベルの統計を収集して表示することは、インスタンスで競合度の高い表あるいは索引を効果的に識別するための優れた方法です。

  パフォーマンスの問題を識別するために待機イベントまたはシステム統計を表示した後で、セグメント・レベルの統計を使用して問題の原因となっている特定の表または索引を検索できます。

  セグメント・レベルの統計は、新規の動的ビュー V$SEGMENT_STATISTICS、V$SEGSTAT および V$SEGSTAT_NAME で問い合せます。

関連項目：

- 22-13 ページ「セグメント・レベルの統計」
- 第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」

- 実行時行ソースの統計

  Oracle9*i* リリース 1（9.0.1）で提供されている、新規の動的パフォーマンス表 V$SQL_PLAN では、キャッシュされたカーソルの実行計画が表示できます。Oracle9*i* リリース 2（9.2）では、これ以外の動的パフォーマンス表 V$SQL_PLAN_STATISTICS が提供されています。このビューでは、キャッシュされたカーソルそれぞれについて、実行計画内の各操作に対する実行統計が提供されます。

  このビューで行ソースの統計を表示するには、DBA がパラメータ STATISTICS_LEVEL を ALL に設定する必要があります。

  追加のビュー V$SQL_PLAN_STATISTICS では、V$SQL_PLAN からの情報と、V$SQL_PLAN_STATISTICS および V$SQL_WORKAREA からの実行統計が連結されます。V$SQL_WORKAREA には、SQL メモリー（たとえば、ハッシュ結合やソート）を使用する行ソースのメモリー使用量の統計が含まれています。

  関連項目：

  - 24-51 ページ「V$SQL_PLAN_STATISTICS」
  - 24-53 ページ「V$SQL_PLAN_STATISTICS_ALL」

- 将来のリリースで削除される Oracle Trace

  Oracle Trace は、将来のリリースでは使用できなくなる可能性があります。かわりに、SQL トレースおよび TKPROF を使用することをお薦めします。

  関連項目： 第 10 章「SQL トレースおよび TKPROF の使用」

第 I 部

SQL の作成とチューニング

第 I 部では、最適なパフォーマンスを得るための SQL 文の理解と管理に関する情報を示します。各章を順を追って読むことをお薦めします。

第 I 部には次の章が含まれます。

1

オプティマイザの概要

この章では、SQL 処理、最適化方式および SQL 文を実行する特定の計画をオプティマイザが選択する方法を説明します。

この章には次の項があります。

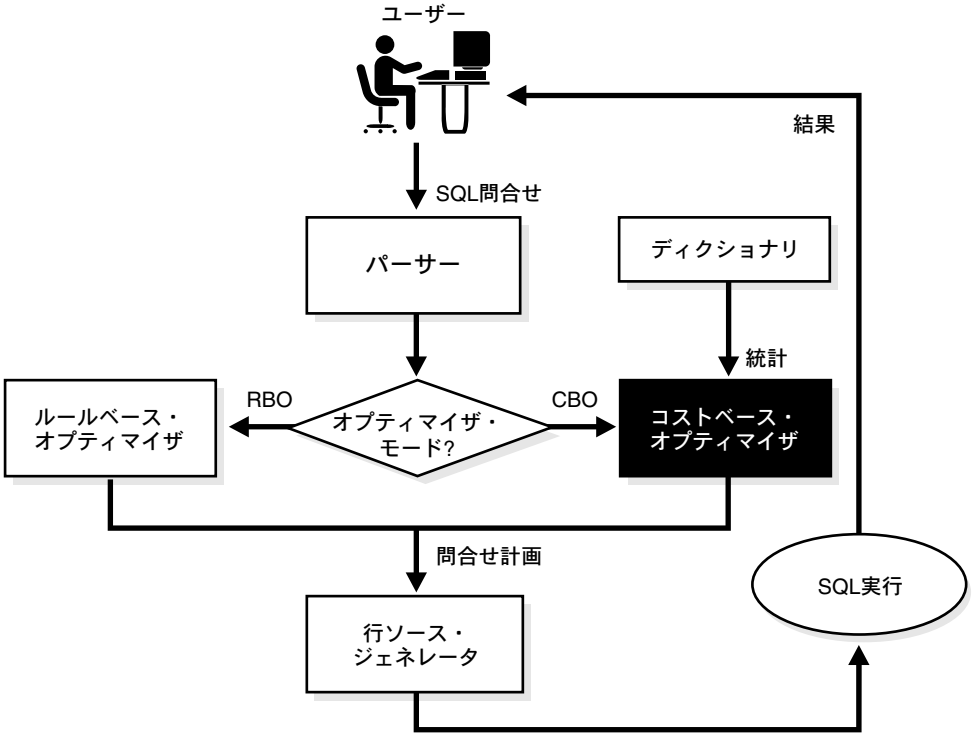
SQL 処理の概要

SQL 処理では、次の主なコンポーネントを使用して SQL 問合せを実行します。

- パーサーは構文および意味の両方の分析をチェックします。
- オプティマイザは、コスト計算方法、コストベース・オプティマイザ（CBO）、内部規則、またはルールベース・オプティマイザ（RBO）を使用して、問合せの結果を生成する最も効率的な方法を決定します。
- オプティマイザから最適な計画を受け取り、SQL 文の実行計画を出力します。
- SQL 実行エンジンは SQL 文に関連付けられた実行計画を処理し、問合せ結果を生成します。

図 1-1 に SQL 処理を示します。

図 1-1 SQL 処理の概要



オプティマイザの概要

オプティマイザは、問合せで指定した参照オブジェクトおよび条件に関連した多くの要素を考慮して SQL 文を最も効率よく実行する方法を判断します。この判断は、SQL 文の処理で重要なステップであり、実行時間が大きく変化します。

SQL 文は様々な方法で実行できます。次に例をあげます。

- 全表スキャン
- 索引スキャン
- ネステッド・ループ
- ハッシュ結合

> **注意：** オプティマイザは、Oracle のあるバージョンとその次のバージョンで同じ決定を行うとは限りません。最新バージョンのオプティマイザは、より高度な情報を使用できるので、異なる決定を行います。

オプティマイザからは、最適な実行方法を説明する計画が出力されます。Oracle データベースには、コストベース・オプティマイザ（CBO）とルールベース・オプティマイザ（RBO）という 2 つの最適化の方法があります。一般には、コストベースのアプローチを使用してください。オラクル社では、継続的に CBO を改善しています。新機能には CBO が必要です。

> **注意：** コストベースの最適化を使用することを強くお薦めします。ルールベースの最適化は、レガシー・アプリケーションとの下位互換性のために使用可能ですが、将来のリリースでは使用されなくなります。

オプティマイザのアプローチと目標を設定して CBO の代表的な統計を収集することによって、オプティマイザの選択を変えることができます。オプティマイザの目標はスループットまたは応答時間です。1-6 ページの「オプティマイザのアプローチと目標の選択」を参照してください。

特定のアプリケーション・データに関して、オプティマイザよりも多くの情報を所有するアプリケーション・デザイナが、より効率良く SQL 文を実行する方法を選択できる場合があります。アプリケーション・デザイナは、SQL 文のヒントを使用して文を実行する方法を指定できます。

関連項目：

- 最適化の目標の詳細は、1-6 ページの「オプティマイザのアプローチと目標の選択」を参照してください。
- 統計を収集および使用する方法の詳細は、第 3 章「オプティマイザ統計の収集」を参照してください。
- SQL 文のヒントを使用する方法の詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

CBO を必要とする機能

次の機能には、CBO を使用する必要があります。

- パーティション表とパーティション索引
- 索引構成表
- 逆キー索引
- ファンクション・ベース索引
- SELECT 文の SAMPLE 句
- パラレル問合せとパラレル DML
- スター型変換とスター型結合
- 拡張可能オプティマイザ
- マテリアライズド・ビューを使用したクエリー・リライト
- Enterprise Manager プログレス・バー
- ハッシュ結合
- ビットマップ索引とビットマップ・ジョイン・インデックス
- 索引スキップ・スキャン

注意： これらの機能を使用するときには、パラメータ OPTIMIZER_MODE が RULE に設定されている場合でも、CBO が使用可能になります。

オプティマイザ操作

Oracle によって処理される SQL 文について、オプティマイザは表 1-1 にリストした操作を実行します。

表 1-1 オプティマイザ操作

操作	説明
式と条件の評価	オプティマイザは、まず、定数が含まれている式と条件を可能なかぎり完全に評価します。2-2 ページの「オプティマイザによる操作の実行方法」を参照してください。
文の変換	たとえば、相関副問合せやビューなどに関連する複合文について、オプティマイザは元の文を同等の結合文に変換する場合があります。2-29 ページの「オプティマイザによる SQL 文の変換方法」を参照してください。
オプティマイザ・アプローチの選択	オプティマイザは、コストベースまたはルールベースどちらかのアプローチを選択して、最適化の目標を判断します。1-6 ページの「オプティマイザのアプローチと目標の選択」を参照してください。
アクセス・パスの選択	文がアクセスするそれぞれの表について、オプティマイザは、表データを取得するために 1 つ以上の使用可能なアクセス・パスを選択します。1-23 ページの「CBO のアクセス・パスについて」を参照してください。
結合順序の選択	2 つ以上の表を結合する結合文について、オプティマイザは、最初に結合する表のペアを選択し、その後、その結果に結合する表を順次選択していきます。1-40 ページの「CBO による結合タイプの実行計画の選択方法」を参照してください。
結合方法の選択	結合文について、オプティマイザは、結合の実行に使用する操作を選択します。1-40 ページの「CBO による結合方法の選択方法」を参照してください。

オプティマイザのアプローチと目標の選択

デフォルトでは、CBO の目標は最高の**スループット**です。つまり、CBO は文からアクセスされたすべての行を処理するのに必要な最小リソースを選択します。また最短**応答時間**を目標とした文の最適化も可能です。つまり、CBO は SQL 文からアクセスされた最初の行を処理するのに必要な最小リソースを使用します。

> **関連項目：** 1-9 ページ「CBO を使用した SQL 文の最適化による応答の高速化」

オプティマイザが生成する実行計画は、オプティマイザの目標によって様々です。スループットを最高にするための最適化であれば、索引スキャンよりも全表スキャン、ネステッド・ループ結合よりもソート / マージ結合が選択されます。応答時間を最短にするための最適化であれば、索引スキャンまたはネステッド・ループ結合が選択されます。

たとえば、ネステッド・ループ操作またはソート / マージ操作で実行可能な結合文が存在する場合について考えます。ネステッド・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ます。目標がスループットの改善である場合、オプティマイザは通常、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、オプティマイザは通常、ネステッド・ループ結合を選択します。

アプリケーションのニーズに基づいて、オプ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は、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は、最短の応答時間を目標に最適化してください。対話形式では、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。

SQL 文に対する最適化のアプローチと目標を選択する場合のオプティマイザの動作は、次の要因の影響を受けます。

- OPTIMIZER_MODE 初期化パラメータ
- データ・ディクショナリ内の CBO 統計
- CBO の目標変更に対するオプティマイザ SQL のヒント

OPTIMIZER_MODE 初期化パラメータ

OPTIMIZER_MODE 初期化パラメータで、インスタンスに最適化アプローチを選択するためのデフォルト動作を設定します。指定可能な値およびその説明を表 1-2 に示します。

表 1-2 OPTIMIZER_MODE パラメータ値

値	説明
CHOOSE	オプティマイザは、統計が使用可能かどうかによって、コストベースまたはルールベースのどちらかのアプローチを選択します。これはデフォルト値です。 ■ データ・ディクショナリに、アクセスされる 1 つ以上の表の統計が含まれている場合、オプティマイザは、コストベースのアプローチを使用し、最高のスループットを目標にして最適化します。 ■ データ・ディクショナリに含まれている統計がわずかな場合も、コストベース・アプローチが使用されますが、オプティマイザは統計のない対象について、統計を推測します。このため、作成される実行計画は最適にはならない場合があります。 ■ データ・ディクショナリに文がアクセスするどの表の統計も含まれていない場合、オプティマイザはルールベースのアプローチを使用します。
ALL_ROWS	オプティマイザは、セッション内の SQL 文に対して、統計の存在の有無にかかわりなく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（最小のリソースを使用して文全体を完成させること）を目標として最適化します。
FIRST_ROWS_*n*	オプティマイザは統計の有無とは関係なく、コストベースのアプローチを使用し、最短応答時間で最初の *n* 行を戻すように最適化します。*n* は 1、10、100 または 1000 です。
FIRST_ROWS	オプティマイザは、コストと経験則を組み合せて、最初の数行の高速配信のための最適な計画を見つけます。 **注意 :** CBO は、経験則を使用すると、経験則を適用しない場合に比べ、コストがはるかに大きい計画を生成する場合があります。FIRST_ROWS は、下位互換性とプラン・スタビリティのためのものです。
RULE	オプティマイザは、SQL 文に対して、統計の存在の有無にかかわりなくルールベースのアプローチを選択します。

初期化ファイル内のパラメータ値の変更、または ALTER SESSION SET OPTIMIZER_MODE 文によって、セッション内のすべての SQL 文の CBO の目標を変更できます。次に例を示します。

- 初期化パラメータ・ファイル内の次の文は、インスタンスのすべてのセッションの CBO の目標を最短の応答時間に変更します。

```
OPTIMIZER_MODE = FIRST_ROWS_1
```

- 次の SQL 文は、現行セッションの CBO の目標を最短の応答時間に変更します。

```
ALTER SESSION SET OPTIMIZER_MODE = FIRST_ROWS_1;
```

オプティマイザがコストベースのアプローチを SQL 文に使用するときに、文がアクセスする一部の表に統計が存在しない場合、オプティマイザはそれらの表に割り当てられているデータ・ブロック数などの内部情報を使用して表に対する別の統計を見積ります。

CBO の目標変更に対するオプティマイザ SQL のヒント

各 SQL 文に対する CBO の目標を指定する場合は、次のリストのヒントのいずれかを使用します。各 SQL 文にあるこれらのヒントは、いずれも、その SQL 文の OPTIMIZER_MODE 初期化パラメータを上書きできます。

- FIRST_ROWS(*n*)、*n* が任意の正の整数
- FIRST_ROWS
- ALL_ROWS
- CHOOSE
- RULE

> **関連項目：** ヒントを使用する方法の詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

データ・ディクショナリ内の CBO 統計

CBO で使用される統計は、データ・ディクショナリに格納されます。DBMS_STATS パッケージまたは ANALYZE 文を使用して、これらのスキーマ・オブジェクト内の物理的な記憶域の特性とデータ配分に関する正確な統計または見積り統計を収集できます。

注意： オプティマイザ統計の収集に、ANALYZE よりも DBMS_STATS パッケージの使用を強くお薦めします。このパッケージでは、パラレル統計の収集、パーティション・オブジェクトについてのグローバル統計の収集、および統計収集の微調整を他の方法で行うことができます。さらに、コストベース・オプティマイザでは、DBMS_STATS により収集された統計のみが最終的に使用されます。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、コストベース・オプティマイザに関連しない次のような統計収集の場合には、DBMS_STATS よりも ANALYZE 文を使用する必要があります。

- VALIDATE または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックの情報を収集する場合

CBO の有効性を維持するには、データを代表する統計が必要です。偏ったデータという、値の重複数のバリエーションが多いデータが存在する表列については、ヒストグラムを収集する必要があります。

その結果の統計によって、データの一意性と配分についての情報が CBO に提供されます。この情報を使用することにより、CBO は計画コストを精密に計算します。その結果 CBO は、最小のコストを基に最良の実行計画を選択できるようになります。

関連項目： 第 3 章「オプティマイザ統計の収集」

CBO を使用した SQL 文の最適化による応答の高速化

パラメータ OPTIMIZER_MODE を FIRST_ROWS_*n* に設定すると、*n* が、1、10、100、1000 のいずれか、または FIRST_ROWS の場合に、CBO は高速応答用に SQL 文を最適化できます。ヒント FIRST_ROWS(*n*)（*n* は正の整数）または FIRST_ROWS は、応答が高速になるように個々の SQL 文を最適化する際に使用できます。

高速応答の最適化は、Oracle Forms または Web アクセスを利用するユーザーなどのオンライン・ユーザーに適しています。一般に、オンライン・ユーザーは最初の数行を見るので、結果のサイズが大きいと、問合せ結果全体を見ようとしません。オンライン・ユーザーの場合、問合せ結果全体を作成する時間を最小にしなくても、できるだけ早く最初の数行を作成するように問合せを最適化することは有効です。

高速応答の最適化を行うと、CBO は最小コストで最初の行または最初の数行を生成する計画を作成します。CBO では、新旧の方法としてここで参照される、2 通りの高速応答の最適化を採用しています。旧来の方法は、FIRST_ROWS ヒントまたはパラメータ値を使用する方法です。この方法では、CBO はコストとルールの組合せを使用して計画を作成します。この計画は、下位互換性のために保持されています。

この新しい方法はすべてコストに基づいており、*n* の値が重要です。*n* の値を小さくすると、CBO は、索引参照を伴うネステッド・ループ結合で構成された計画を生成する傾向があります。*n* の値を大きくすると、CBO はハッシュ結合と全表スキャンで構成された計画を生成する傾向があります。

n の値は、オンライン・ユーザーの要求に基づいて選択する必要があり、この値は、結果がユーザーに表示される方法により明確に異なります。一般に、Oracle Forms ユーザーが一度に見るのは結果の 1 行のみで、最初のいくつかの画面を見る傾向があります。他のオンライン・ユーザーは、一度に行のグループからなる結果を見ます。

高速応答にする方法を使用すると、CBO は様々な計画を調べ、最初の *n* 行を生成するためのコストを行ごとに計算します。CBO は、最小コストで最初の *n* 行を生成する計画を採用します。高速応答の最適化に基づく、最小コストで最初の *n* 行を生成する計画が、全体的な結果を生成する計画として最適であるとは限りません。問合せの結果全体を取得することを条件としている場合は、高速応答の最適化を使用しないでください。かわりに、ALL_ROWS パラメータの値またはヒントを使用してください。

コストベース・オプティマイザについて

CBO は、SQL 文がアクセスするスキーマ・オブジェクト（表または索引）について、使用可能なアクセス・パスを検討し、統計に基づいた情報要素を考慮することによって最も効果的な実行計画を判断します。また、CBO は文のコメントに配置された、最適化の提案であるヒントも考慮します。

> **関連項目：** ヒントの詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

CBO は次のステップを実行します。

1. 使用可能なアクセス・パスおよびヒントに基づき、SQL 文の可能な計画のセットを生成します。
2. 文がアクセスする表、索引およびパーティションのデータ配分および記憶特性に関するデータ・ディクショナリ内の統計に基づき、計画のそれぞれのコストを見積ります。

   コストとして見積られる値は、特定の計画での文の実行に必要と予測されるリソース使用量に比例しています。オプティマイザは、I/O、CPU、メモリーなどのコンピュータ・リソースの見積りに基づいて、アクセス・パスや結合順序のコストを計算します。

   コストの大きいシリアル計画の実行には、コストの小さい計画の実行よりも多くの時間が必要です。ただし、パラレル計画を使用する場合は、リソース使用量は経過時間に直接関係しません。
3. 計画のコストを比較して、コストが最も小さいものを選択します。

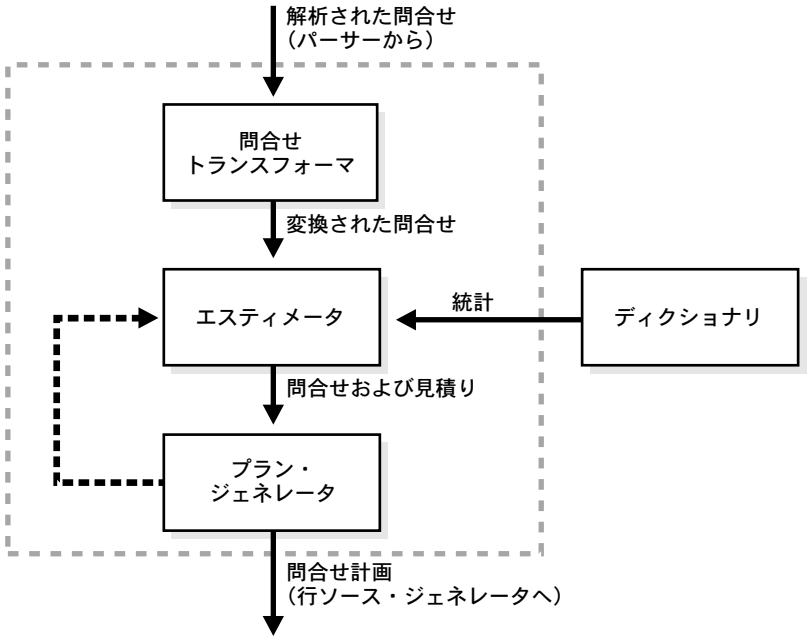
CBO の構成要素

CBO は、次の 3 つの主な構成要素で構成されています。

- 問合せトランスフォーマ
- エスティメータ
- プラン・ジェネレータ

CBO の構成要素を図 1-2 に示します。

図 1-2 コストベース・オプティマイザの構成要素



問合せトランスフォーマ

問合せトランスフォーマへの入力は、一連の問合せブロックによって表される解析済み問合せです。問合せブロックは、互いにネストされているかまたは相互関係を持っています。問合せの形式によって、問合せブロックの相互関係が判断されます。問合せトランスフォーマの主な目的は、問合せの形式を変える必要があるかどうかを判断して、より適切な問合せ計画を生成できるようにすることです。問合せトランスフォーマでは、4 通りの問合せ変換手法を採用しています。

- ビューのマージ
- 述語のプッシュ
- 副問合せのネスト解除
- マテリアライズド・ビューを使用したクエリー・リライト

これらの変換は任意に組み合せて指定の問合せに適用できます。

ビューのマージ 問合せで参照されるビューは、パーサーによって個別の問合せブロックに拡張されます。問合せブロックは、基本的にはビュー定義を表しますが、このため必然的にビューの結果も表すことになります。オプティマイザには、ビューの問合せブロックの１つを分離して分析し、ビュー・サブプランを生成するというオプションがあります。オプティマイザは、問合せ計画全体の生成にビュー・サブプランを使用することで、残りの問合せを処理します。この手法は、そのビューが問合せの残りの部分とは別に最適化されるため、通常は最適な問合せ計画とはなりません。

問合せトランスフォーマは、ビューが含まれている問合せブロックにビューの問合せブロックをマージして、潜在的に最適とは言えない計画を除去します。ほとんどのタイプのビューがマージされます。ビューをマージするときには、ビューを表す問合せブロックが包含的な問合せブロックにマージされます。ビューの問合せブロックは除去されているので、サブプランを生成する必要はありません。

述語のプッシュ マージされていないビューについては、問合せトランスフォーマにより、関連する述語を、ビューを包含する問合せブロックからビューの問合せブロックに組み込むことができます。この手法は、プッシュされた述語を索引へのアクセスやフィルタに使用できるので、マージされていないビューのサブプランが改善されます。

副問合せのネスト解除 ビューの場合と同様に、副問合せが個別の問合せブロックによって表されます。副問合せは、主問合せまたは別の副問合せにネストされているため、副問合せのコストが最も低い計画を検索する前には、使用可能な別の計画について判断できないという制約がプラン・ジェネレータに課されています。そのため、生成された問合せ計画が最適化されないことがあります。副問合せがネストされていることによる制約は、副問合せのネストを解除して結合に変換することによって取り除けます。大半の副問合せは、問合せトランスフォーマでネスト解除されます。ネスト解除されない副問合せの場合は、個別のサブプランが生成されます。問合せ計画全体の実行速度を上げるため、サブプランは効果的な順序で並べられます。

マテリアライズド・ビューを使用したクエリー・リライト マテリアライズド・ビューは、結果がマテリアライズされて表に格納された問合せと類似しています。マテリアライズド・ビューに関連付けられた問合せと互換性のあるユーザー問合せが検出されると、ユーザー問合せはマテリアライズド・ビューにリライトできます。この手法は、ほとんどの問合せ結果があらかじめ計算されているため、ユーザー問合せの実行状態を改善します。問合せトランスフォーマは、ユーザー問合せと互換性のあるマテリアライズド・ビューを検索し、１つ以上のマテリアライズド・ビューを選択してユーザー問合せをリライトします。問合せをリライトするためのマテリアライズド・ビューの使用はコストベースです。したがって、マテリアライズド・ビューを使用せずに生成された計画のコストが、マテリアライズド・ビューを使用して生成された計画のコストより低い場合、問合せはリライトされません。

関連項目：

- 2-29 ページ「オプティマイザによる SQL 文の変換方法」
- クエリー・リライトの詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

エスティメータ

エスティメータは、異なるタイプのメジャーを 3 通り生成します。

- 選択性
- カーディナリティ
- コスト

これらのメジャーは相互に関連性があり、あるメジャーは別のメジャーから派生します。エスティメータの最終目標は、指定された計画のコスト全体を算出することです。統計が使用可能な場合、エスティメータは統計を使用してメジャーを計算します。統計によって、メジャーの正確さの度合いは改良されます。

選択性 最初のメジャーは行セットの一部の行を表す選択性です。行セットとは、実表、ビュー、結合結果または GROUP BY 演算子の結果です。選択性は、`last_name = 'Smith'` などの問合せ述語、または `last_name = 'Smith' AND job_type = 'Clerk'` などの述語の組合せに拘束されています。述語はフィルタとして機能します。このフィルタは、行セットから特定数の行を選別します。したがって、述語の選択性が述語テストに合格する行セットの行数を示しています。選択性の値範囲は、0.0 から 1.0 です。選択性が 0.0 の場合、行セットから行は選択されず、選択性が 1.0 の場合はすべての行が選択されます。

エスティメータは、使用可能な統計が存在しない場合、選択性に内部デフォルト値を使用します。使用される内部デフォルトは、述語のタイプによって異なります。たとえば、等価述語（`last_name = 'Smith'`）の内部デフォルトは範囲述語（`last_name > 'Smith'`）の内部デフォルトより低くなっています。エスティメータがこの仮定を行うのは、等価述語が範囲述語より少ない行数の行を戻すことが予測されるためです。

統計が使用可能な場合、エスティメータは統計を使用して選択性を推測します。たとえば、等価述語（`last_name = 'Smith'`）の選択性は、`last_name` の個別値である数値 *n* の逆数に設定されます。これは、問合せが *n* の内から 1 つの個別値をすべて含む行を選択するためです。`last_name` 列でヒストグラムが使用可能な場合、エスティメータは個別値のかわりにヒストグラムを使用します。ヒストグラムには列内の異なる値の分散が示されているので、より適切な選択性の見積りが行われます。偏ったデータ（値の重複数のバリエーションが多いデータ）が含まれている列のヒストグラムが存在すると、CBO が適切な選択性の見積りを生成するのに役立ちます。

カーディナリティ カーディナリティは、行セット内の行数を表します。ここでの行セットとは、実表、ビュー、結合結果または GROUP BY 演算子の結果です。

- **ベース・カーディナリティ**は、実表の行数です。ベース・カーディナリティは、表を分析することによって得られます。表統計が使用できない場合は、表に使用されているエクステント数を使用してエスティメータがベース・カーディナリティを推測します。
- **有効なカーディナリティ**は、実表から選択される行数です。有効なカーディナリティは、実表の別の列に指定されている述語により異なりますが、各述語は実表の行に対する連続フィルタとして機能します。有効なカーディナリティは、ベース・カーディナリティと表に示されている全ての述語の選択性の積として計算されます。表に述語が存在しない場合は、有効なカーディナリティはベース・カーディナリティと等価です。
- **結合カーディナリティ**は、2 つの行セットが結合されるときに生成される行数です。**結合**は、結果へのフィルタとして適用される述語結合による 2 つの行セットのデカルト積です。したがって、結合カーディナリティは、述語結合の選択性によって乗算される、2 つの行セットのカーディナリティの積です。
- **個別カーディナリティ**は、行セットの列内の個別値の数です。行セットの個別カーディナリティは、列のデータに基づいています。たとえば、100 行の行セットにおいてある列の個別値が 20 行に存在する場合、個別カーディナリティは 20 です。
- **グループ・カーディナリティ**は、GROUP BY 演算子が適用された後で行セットから生成される行数です。GROUP BY 演算子の役割は、行セット内の行数を減らすことです。グループ・カーディナリティは、各グループ列の個別カーディナリティと行セットの行数により異なります。グループ・カーディナリティの説明は、例 1-1 を参照してください。

例 1-1 グループ・カーディナリティ

個別カーディナリティが 30 の colx で 100 行の行セットをグループ化する場合、そのグループ・カーディナリティは 30 です。

ただし、100 行の行セットが colx と coly でグループ化されているとすると、これらのグループが持つ個別カーディナリティはそれぞれ 30 と 60 です。このケースでは、グループ・カーディナリティは colx と coly の個別カーディナリティでの最大値と、colx と coly の個別カーディナリティの積と行セットの行数の値で少ない値の間に存在します。

この例のグループ・カーディナリティは、次の方程式で表すことができます。

```
group cardinality lies between max ( dist. card. colx , dist. card. coly )
                           and min ( (dist. card. colx * dist. card. coly) ,
                           num rows in row set )
```

この例からの数字を代入すると、グループ・カーディナリティは、最大値（30 および 60）と、最小値（30*60 および 100）の間になります。つまり、グループ・カーディナリティは 60 と 100 の間になります。

コスト コストは、作業単位または使用されるリソースを表します。CBO はディスク I/O、CPU 使用量、メモリー使用量を作業単位として使用します。しかし、CBO によって使用されるコストは、操作の実行で使用した CPU とメモリーの量の見積り数になります。すなわち、操作には表をスキャンすること、索引を使用して表から行にアクセスすること、2 つの表を結合すること、または行セットをソートすることなどがあります。問合せ計画のコストは、問合せが実行されてその結果が生成されるときに発生すると予測される作業単位の数です。

アクセス・パスは、実表からデータを得るときに実行される作業単位数です。アクセス・パスには、表スキャン、高速全索引スキャンまたは索引スキャンがなどがあります。表スキャンまたは高速全索引スキャンの実行中には、単独の I/O 操作で複数のブロックがディスクから読み込まれます。したがって、表スキャンまたは高速全索引スキャンのコストは、スキャンされるブロック数およびマルチブロック READ カウントに左右されます。索引スキャンのコストは、B ツリーにおけるレベル、つまりスキャンされる索引リーフ・ブロック数、索引キーの ROWID を使用してフェッチされる行数に左右されます。ROWID を使用して行をフェッチするコストは、索引クラスタ化係数に依存します。

クラスタ化係数は索引のプロパティですが、実際には、表のデータ・ブロック内の類似した索引付きの列の値の拡散の度合いに関連します。低いクラスタ化係数は、個々の行が表の少数のブロック内に集中されることを示します。逆に、高いクラスタ化係数は、個々の行が表の複数のブロックによりランダムに分散されることを示します。したがって、高いクラスタ化係数ではレンジ・スキャンを使用して ROWID で行をフェッチすると、よりコストがかかります。データを戻すために表の中のさらに多くのブロックにアクセスする必要があるためです。例 1-2 では、クラスタ化係数がコストにどのような影響を与えるかが示されています。

例 1-2 クラスタ化係数がコストに与える影響

次の状況を想定します。

- 9 つの行を持つ表があります。
- `col1`（`tab1` 内）に一意でない索引があります。
- `c1` 列は現在、値 `A`、`B` および `C` を格納しています。
- 表には、Oracle ブロックが 3 つしかありません。

ケース 1: 次の図のように配置されている場合、行に対して索引クラスタ化係数は低くなります。

```
Block 1       Block 2       Block 3
-------       -------       --------
A  A  A       B  B  B       C  C  C
```

これは、`c1` に対して同じ索引付きの列の値を持つ行が、表の中の同じ物理ブロック内にあるためです。レンジ・スキャンを使用して、値 `A` を持つすべての行を戻す際のコストが低いのは、表の中のブロックを 1 つのみ読み取れば済むためです。

ケース 2: 同じ行が、索引値が表ブロック間に分散する（並べるのではなく）ように再配置されると、索引クラスタ化係数が高くなります。

```
Block 1       Block 2       Block 3
-------       -------       --------
A  B  C       A  B  C       A  B  C
```

これは、表の中の 3 つのブロックを、col1 内に値 A を持つすべての行を取得するために、すべて読み取る必要があるためです。

結合コストは、結合されている 2 つの行セットの個別のアクセス・コストの組合せを表します。結合では、片方の行セットが内部と呼ばれ、もう一方が外部と呼ばれます。

- **ネステッド・ループ結合**では、外部行セットの各行に対して内部行セットがアクセスされ、結合するすべての一致行が検索されます。このため、ネステッド・ループ結合では、外部行セット内の行数と同じ回数、内部行セットがアクセスされます。

  ```
  cost  = outer access cost + (inner access cost * outer cardinality)
  ```

- **ソート / マージ結合**では、結合されている 2 つの行セットがまだキー順序になっていない場合、2 つの行セットは結合キーによってソートされます。

  ```
  cost = outer access cost + inner access cost + sort costs (if sort is used)
  ```

- **ハッシュ結合**では、内部行セットがメモリーにハッシュされ、結合キーを使用してハッシュ表が作成されます。外部行セットの各行が後にハッシュされ、ハッシュ表がプローブされてすべての一致行を結合します。内部行セットが非常に大きい場合は、その一部のみがメモリーにハッシュされます。この部分をハッシュ・パーティションといいます。

  外部行セットの各行がハッシュされ、ハッシュ・パーティション内の一致行がプローブされます。内部行セットの次の部分が、この後メモリーにハッシュされ、外部行セットのプローブが実行されます。このプロセスは、内部行セットのすべての部分が実行し終わるまで繰り返されます。

  ```
  cost = (outer access cost * # of hash partitions) + inner access cost
  ```

  関連項目： 結合に関する詳細は、1-39 ページの「結合について」を参照してください。

プラン・ジェネレータ

プラン・ジェネレータの主な機能は、指定の問合せに対して使用できる可能性のある別の計画を割り出し、コストの最も低いものを取り出すことです。異なる方法でデータをアクセスおよび処理し、かつ結果が同じになる、様々なアクセス・パス、結合方法および結合順序の組合せが存在するので、多数の異なる計画が使用できます。

結合順序は、異なる結合項目（表など）がアクセスされて結合される順序です。たとえば、t1、t2 および t3 の結合順序では、表 t1 が最初にアクセスされます。次に、t2 がアクセスされ、そのデータが t1 のデータに結合されて t1 と t2 の結合が生成されます。最後に t3 がアクセスされ、そのデータが t1 と t2 の結合結果に結合されます。

問合せのための計画は、まず最初にネストされた副問合せおよびマージされていないビューのそれぞれにサブプランを生成することによって構築されます。ネストされた副問合せ、またはマージされていないビューは、それぞれ、個別の問合せブロックによって表されます。問合せブロックは、それぞれ、下位から上位へ順番に最適化されます。つまり、最も内側の問合せブロックが最初に最適化され、サブプランが生成されます。そして、問合せ全体を表す一番外側の問合せブロックは、最後に最適化されます。

プラン・ジェネレータは、別のアクセス・パス、結合方法および結合順序を試行することによって、問合せブロックに対する様々な計画を探索します。問合せブロックに使用できる可能性のある計画の数は、FROM 句にある結合項目の数に比例します。この数は、結合項目の数によって指数関数的に上昇します。

プラン・ジェネレータは内部カットオフを使用して計画数を削減し、最もコストの低い計画を検索しようとします。カットオフの基準は、現行の最適な計画のコストです。現行の最適コストが大きい場合、プラン・ジェネレータはコストがより低く、より適切な計画（つまり、さらに別の計画）を探索します。現行の最適コストが低い場合は、これ以上のコストの改善を追及しても大きな効果は得られないため、プラン・ジェネレータは検索を速やかに終了します。

最適な状態に最も近いコストで計画を生成する初期結合順序で、プラン・ジェネレータが始動する場合は、カットオフが有効に機能します。適切な初期結合順序の検索は困難です。プラン・ジェネレータは、初期結合順序に対する単純な経験則を使用します。これは、結合項目をその有効なカーディナリティ別に並べます。有効なカーディナリティの最も小さい結合項目が最初で、有効なカーディナリティの最も大きい結合項目が最後になります。

実行計画について

SQL 文を実行するために、Oracle は、多数のステップを実行する必要があります。実行される各ステップでは、データベースからのデータ行の物理的な取出し、またはユーザーが発行する文に返すデータ行の準備のどちらかが実行されます。文を実行するために Oracle が使用するステップの組合せのことを**実行計画**と呼びます。実行計画には、文がアクセスする各表への**アクセス・パス**と、適切な**結合方法**に基づく表の順序（**結合順序**）が含まれています。

関連項目：

- 1-23 ページ「CBO のアクセス・パスについて」
- 8-3 ページ「RBO のアクセス・パス」
- 第 9 章「EXPLAIN PLAN の使用方法」

EXPLAIN PLAN の概要

EXPLAIN PLAN 文を使用することにより、オプティマイザが SQL 文に対して選択した実行計画を確認できます。文が発行されると、オプティマイザが実行計画を選択した後で、計画を説明するデータがデータベース表に挿入されます。単純に、EXPLAIN PLAN 文を発行し、出力表を問い合せます。

EXPLAIN PLAN 文の使用方法の基本は次のとおりです。

- SQL スクリプト `UTLXPLAN.SQL` を使用し、使用しているスキーマ内に `PLAN_TABLE` というサンプル出力表を作成してください。9-5 ページの「PLAN_TABLE 出力表の作成」を参照してください。
- SQL 文の前に EXPLAIN PLAN FOR 句を挿入します。9-5 ページの「EXPLAIN PLAN の実行」を参照してください。
- EXPLAIN PLAN 文を発行した後、Oracle から提供されるスクリプトのいずれかを使用して最新の計画表出力を表示します。9-7 ページの「PLAN_TABLE 出力の表示」を参照してください。
- EXPLAIN PLAN の出力実行順序は、最も右端にインデントされている行から始まります。次のステップは、その行の親です。2 つの行が等しくインデントされている場合、最上位の行が最初に実行されます。

注意：

- この章では、EXPLAIN PLAN 出力表は `utlxpls.sql` スクリプトで表示されました。
- この章の EXPLAIN PLAN 出力表のステップは、システムによって異なる場合があります。データベース構成によって、オプティマイザは異なる実行計画を選択する場合があります。

例 1-3 では EXPLAIN PLAN を使用して、ID が 103 より小さい従業員の、employee_id、job_title、salary および department_name を選択する SQL 文について説明しています。

例 1-3 EXPLAIN PLAN の使用方法

```
EXPLAIN PLAN FOR
SELECT e.employee_id, j.job_title, e.salary, d.department_name
    FROM employees e, jobs j, departments d
    WHERE  e.employee_id < 103
       AND e.job_id = j.job_id
       AND e.department_id = d.department_id;
```

例 1-4 の結果の出力表では、例にある SQL 文を実行するためにオプティマイザで選択された実行計画が示されています。

例 1-4 EXPLAIN PLAN 出力

```
-------------------------------------------------------------------------------------
| Id  | Operation                     |  Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |              |     3 |   189 |    10  (10)|
|   1 |  NESTED LOOPS                 |              |     3 |   189 |    10  (10)|
|   2 |   NESTED LOOPS                |              |     3 |   141 |     7  (15)|
|*  3 |    TABLE ACCESS FULL          | EMPLOYEES    |     3 |    60 |     4  (25)|
|   4 |    TABLE ACCESS BY INDEX ROWID| JOBS         |    19 |   513 |     2  (50)|
|*  5 |     INDEX UNIQUE SCAN         | JOB_ID_PK    |     1 |       |            |
|   6 |   TABLE ACCESS BY INDEX ROWID | DEPARTMENTS  |    27 |   432 |     2  (50)|
|*  7 |    INDEX UNIQUE SCAN          | DEPT_ID_PK   |     1 |       |            |
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   3 - filter("E"."EMPLOYEE_ID"<103)
   5 - access("E"."JOB_ID"="J"."JOB_ID")
   7 - access("E"."DEPARTMENT_ID"="D"."DEPARTMENT_ID")
```

Oracle では、EXPLAIN PLAN 出力を表示するためのもう 1 つのグラフィカル・ツールを提供しています。6-4 ページの図 6-1「Oracle SQL Analyze」は、Oracle SQL アナライザで表示される SQL 文の例を示したものです。図 1-3 は、Oracle Enterprise Manager SQL スクラッチパッド・ウィンドウで生成された例 1-3 の SQL 文の EXPLAIN PLAN をグラフィカル表示したものです。

図 1-3 SQL スクラッチパッド内の SQL Explain Plan のグラフィカル表示

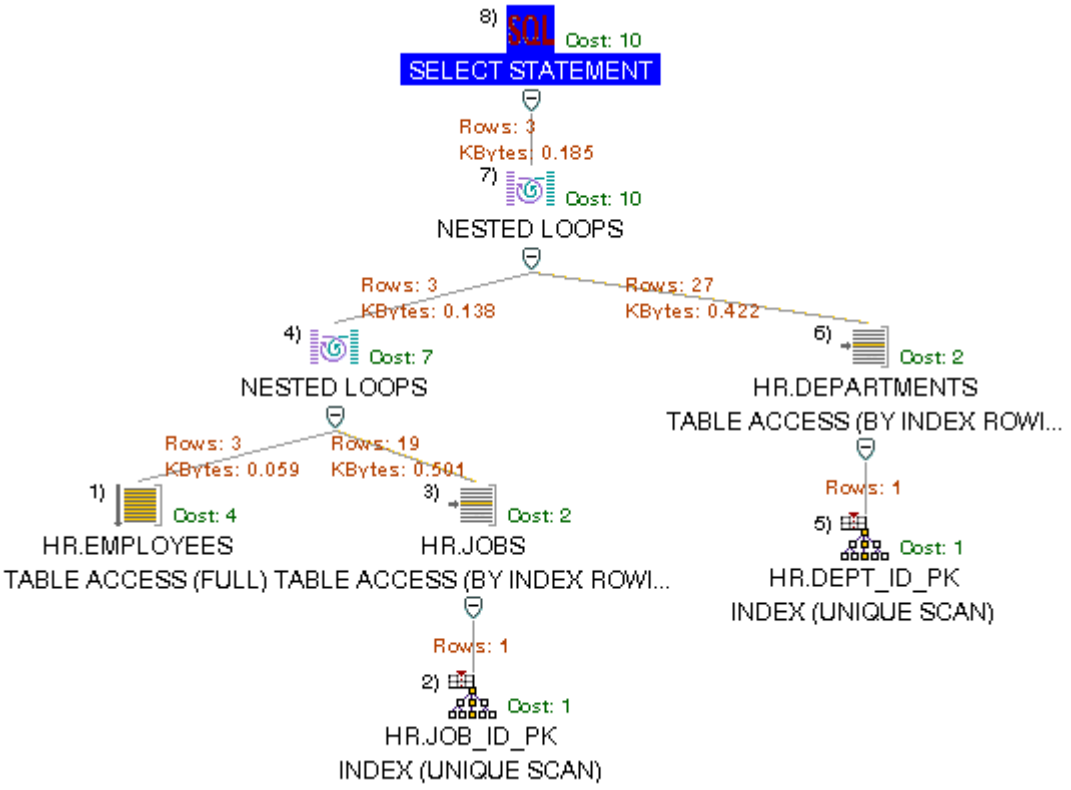


図 1-3 の実行ステップは、例 1-4 の ID ではなく、実行順序によって識別されます。

関連項目： Oracle Enterprise Manager とそのオプションのアプリケーションの詳細は、『Oracle Enterprise Manager 概要』、『Oracle Enterprise Manager 管理者ガイド』および『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

実行計画のステップ

出力表内の各行は、実行計画内の 1 つのステップに対応しています。アスタリスクの付いたステップ ID は、Predicate Information セクションにリストされています。

関連項目：

- 1-21 ページ「実行計画のステップ」
- 第 9 章「EXPLAIN PLAN の使用方法」

実行計画の各ステップで、行のセットが戻されます。この行のセットは、次のステップで使用されます。最後のステップでは、SQL 文を発行しているユーザーまたはアプリケーションに対し、行のセットが戻されます。各ステップで戻される行のセットを、**行セット**と呼びます。

ステップ ID の番号は、EXPLAIN PLAN 文で返された実行計画の順序 ID に対応しています。実行計画の各ステップでは、データベースから行を取り出すか、1 つ以上の行ソースから行を入力として受け入れます。

- 例 1-4 の次のステップでは、データベース内のオブジェクトからデータが物理的に取り出されます。
  - ステップ 3 では `employees` 表にあるすべての行を読み込みます。
  - ステップ 5 では、`JOB_ID_PK` 索引の各 `job_id` を参照して、`jobs` 表内の対応する行の ROWID を検索します。
  - ステップ 4 では、`jobs` 表からステップ 5 で戻された ROWID を持つ行を取得します。
  - ステップ 7 では、`DEPT_ID_PK` 索引の各 `department_id` を参照して、`departments` 表内の対応する行の ROWID を検索します。
  - ステップ 6 では、`departments` 表からステップ 7 で戻された ROWID を持つ行を取得します。
- 例 1-4 の次のステップでは、直前の行ソースから戻された行を処理します。
  - ステップ 2 では、ステップ 3 およびステップ 4 から戻される行ソースを受け入れ、ステップ 3 からの各行をステップ 4 の対応する行に結合し、その結果の行をステップ 2 に戻す、ネステッド・ループ操作を、`jobs` 表および `employees` 表の `job_id` で実行します。
  - ステップ 1 では、ステップ 2 およびステップ 6 から戻される行ソースを受け入れ、ステップ 2 からの各行をステップ 6 の対応する行に結合し、その結果の行をステップ 1 に戻すネステッド・ループ操作を実行します。

関連項目：

- アクセス・パスに関する詳細は、1-23 ページの「CBO のアクセス・パスについて」と 8-3 ページの「RBO のアクセス・パス」を参照してください。
- Oracle が行ソースを結合する方法の詳細は、1-39 ページの「結合について」を参照してください。

実行順序について 実行計画のステップは、1-19 ページの例 1-3 の番号順に実行されるわけではありません。Oracle は、EXPLAIN PLAN 出力で最も右にインデント表示されるステップを最初に実行します。1-20 ページの図 1-3 では、ステップは実行順に番号が付けられています。各ステップで戻された行が、その親ステップの行ソースになります。そして、Oracle がその親ステップを実行します。

1-19 ページの例 1-4 の次のステップを実行して、1-19 ページの例 1-3 の文を実行します。

- ステップ 3 およびステップ 4 を実行し、その結果得られた行をステップ 2 に戻します。ステップ 2 に戻される各行に対してはステップ 5 が実行され、その結果の ROWID をステップ 4 に戻します。
- ステップ 1 を実行して、ステップ 2 から戻された単一行をステップ 6 から戻された単一行に結合し、結果行がある場合は SQL 文を発行したユーザーに戻します。ステップ 6 に戻される各行に対してはステップ 7 が実行され、その結果の ROWID をステップ 4 に戻します。

親ステップの実行に必要な子ステップからの戻り行が 1 行のみの場合は、子ステップから 1 行が戻されるとすぐに親ステップが実行されます。実行された親ステップの親も単一行の戻りによってアクティブになる場合は、同様に実行されます。

ツリーの上位へと連鎖的に文を実行し、実行計画をできるかぎり完了させます。Oracle は、子ステップによって行が戻されると、各行に対して親ステップとそれに連鎖したステップを 1 度ずつ実行します。子ステップによって戻された各行についてトリガーされる親ステップには、表アクセス、索引アクセス、ネステッド・ループ結合およびフィルタがあります。

親ステップの実行に、子ステップから戻される行すべてが必要な場合は、子ステップからすべての行が戻されるまで Oracle は親ステップを実行できません。このような親ステップには、ソート、ソート / マージ結合および集計関数があります。

CBO のアクセス・パスについて

アクセス・パスは、データベースからデータを取り出す経路です。一般に、表の行の小さいサブセットを取得する文には索引アクセス・パスを指定する必要がありますが、表の大きい部分にアクセスするときは全体スキャンのほうが効率が良くなります。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、選択性が高く実行の短い SQL 文から構成されており、多くの場合は索引アクセス・パスを使用するという特徴があります。それに対して、意思決定支援システムはパーティション表を使用し、関連するパーティションの全体スキャンを実行する傾向があります。

この項では、表内の任意の行の検索および取出しに使用できるデータ・アクセス・パスについて説明します。

- 全表スキャン
- ROWID スキャン
- 索引スキャン
- クラスタ・スキャン
- ハッシュ・スキャン
- サンプル表スキャン
- CBO によるアクセス・パスの選択方法

全表スキャン

このタイプのスキャンでは、表にあるすべての行の読取り、選択基準を満たしていない行のフィルタが実行されます。全表スキャンで、最高水位標以下の、表中のすべてのブロックがスキャンされます。各行が文の `WHERE` 句を満たすかどうかを判断するために、各行が検査されます。

全表スキャンを行うと、ブロックが順に読み取られます。ブロックは隣接しているため、単一ブロックより大きい I/O コールを使用してプロセスを高速化することができます。リード・コールのサイズの範囲は、1 ブロックから初期化パラメータ `DB_FILE_MULTIBLOCK_READ_COUNT` で示されるブロック数までです。マルチブロック READ を使用すると、全表スキャンを効率よく実行できます。各ブロックは 1 回のみ読み取られます。

1-19 ページの例 1-4「EXPLAIN PLAN 出力」には、`employees` 表の全表スキャン例が含まれています。

大量データにアクセスする場合に全表スキャンのほうが高速になる理由

表の中の大部分のブロックにアクセスする場合は、索引レンジ・スキャンより全表スキャンのほうがコストが低くなります。これは、全表スキャンのほうが大きい I/O コールを使用しており、少数の大きい I/O コールのほうが、多数の小さい I/O コールよりもコストが低いためです。

オプティマイザが全表スキャンを使用する場合

オプティマイザは、次のいずれかの場合に全表スキャンを使用します。

索引の欠落 問合せで既存の索引を使用できない場合、全表スキャンを使用します。たとえば、問合せに索引付けされた列で使用する関数がある場合は、オプティマイザは索引を使用できず、かわりに例 1-5 のような全表スキャンを使用します。

例 1-5 全表スキャン

```
SELECT last_name, first_name
  FROM employees
 WHERE UPPER(last_name) LIKE :b1
```

大小文字を区別しない検索に索引を使用する必要がある場合は、大小文字混合データを検索列に許可しないか、検索列に UPPER(last_name) のような関数ベースの索引を作成します。4-9 ページの「ファンクション・ベース索引の使用方法」を参照してください。

大量のデータ 問合せが表のブロックの大部分にアクセスするとオプティマイザが判断した場合、索引が使用できる場合でも全表スキャンが使用される可能性があります。

小さい表 1 回の I/O コールで読み取れる DB_FILE_MULTIBLOCK_READ_COUNT より少ないブロックが表の最高水位標以内に格納されている場合には、表のどの部分にアクセスされるかや、索引があるかどうかに関係なく、全表スキャンを行うほうが索引レンジ・スキャンよりもコストが低くなる可能性があります。

高い並列度 表の並列度が高いと、レンジ・スキャンよりも全表スキャンの方向にオプティマイザを偏らせます。表の並列度を判断するには、ALL_TABLES 内の DEGREE 列を調べます。

全表スキャンのヒント

全表スキャンを強制実行する場合は、ヒント FULL(*table alias*) を使用します。FULL ヒントの詳細は、5-11 ページの「FULL」を参照してください。

例 1-6 は、索引レンジ・スキャンを使用する問合せを示したものです。例 1-7 は、FULL ヒントを使用して全表スキャンを強制実行する同じ問合せを示したものです。

例 1-6 FULL ヒントを使用する前

```
SELECT employee_id, last_name
  FROM employees
 WHERE last_name LIKE :b1;

-------------------------------------------------------------------------------
| Id  | Operation                    |  Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |              |     5 |    95 |     3  (34)|
|   1 |  TABLE ACCESS BY INDEX ROWID| EMPLOYEES    |     5 |    95 |     3  (34)|
|*  2 |   INDEX RANGE SCAN           | EMP_NAME_IX  |     2 |       |     3  (34)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("EMPLOYEES"."LAST_NAME" LIKE :Z)
       filter("EMPLOYEES"."LAST_NAME" LIKE :Z)
```

例 1-7 は、FULL ヒントを使用して全表スキャンを強制実行する例 1-6 の問合せを示したものです。

例 1-7 FULL ヒントを使用した後

```
SELECT /*+ FULL(e) */ employee_id, last_name
  FROM employees e
  WHERE last_name LIKE :b1;

---------------------------------------------------------------------------
| Id  | Operation            |  Name       | Rows  | Bytes | Cost (%CPU)|
---------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |             |     5 |    95 |     4  (25)|
|*  1 |  TABLE ACCESS FULL   | EMPLOYEES   |     5 |    95 |     4  (25)|
---------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("E"."LAST_NAME" LIKE :Z)
```

ブロックの I/O（行ではなく）の想定

Oracle は、ブロック単位で I/O を実行します。したがって、全表スキャンを使用するかどうかのオプティマイザの決定は、行でなくアクセスされるブロックのパーセンテージに影響されます。これを索引クラスタ化係数といいます。ブロックに単一行が含まれている場合、アクセスされる行とアクセスされるブロックは同じです。

ただし、大半の表には各ブロック内に複数の行があります。したがって、目的の行が、少ないブロック内にまとまってクラスタ化されていたり、大量のブロックにわたって拡散されていることがあります。

> **関連項目：** 索引クラスタ化係数の詳細は、1-13 ページの「エスティメータ」を参照してください。

DBA_TABLES 内の最高水位標

データ・ディクショナリは、行が占めているブロックの情報を保持します。最高水位標は、全表スキャンの終了マーカーとして使用されます。最高水位標は、DBA_TABLES.BLOCKS に保存されます。最高水位標は、表を削除または切り捨てる場合にリセットされます。

たとえば、過去に大量の行があった表を考えてみます。大半の行は削除されており、現在、最高水位標以下のブロックの大半は空白です。この表の全表スキャンはパフォーマンスが低下します。これは、最高水位標以下のすべてのブロックがスキャンされるためです。

パラレル問合せの実行

全表スキャンが必要である場合は、表のスキャンに複数のパラレル実行サーバーを使用して、応答時間を向上できます。パラレル問合せは、リソース使用の可能性があるために、一般的には並行性の低いデータ・ウェアハウス環境で使用されます。

> **関連項目：** 『Oracle9*i* データ・ウェアハウス・ガイド』

ROWID スキャン

行の ROWID には、行が含まれているデータ・ファイルおよびデータ・ブロックと該当するブロック内の位置を指定します。行の ROWID の特定による、行の位置特定は、単一行を取得する最も高速な方法です。これは、取得する行のデータベース内での正確な位置が指定されるためです。

ROWID を使用して表にアクセスする場合、まず、Oracle は選択された行の ROWID を、文の WHERE 句、または 1 つ以上の表の索引の索引スキャンを使用して取得します。次に、Oracle は ROWID に従って、それぞれの選択された行を表から探します。

1-19 ページの例 1-4「EXPLAIN PLAN 出力」では、索引スキャンが jobs 表および departments 表で実行されます。取り出された ROWID は、行データを戻す場合に使用します。

オプティマイザが ROWID を使用する場合

通常、これは索引から ROWID を取得した後の第 2 のステップです。索引内に存在しない文の中の列には、表アクセスが必要になる場合があります。

ROWID によるアクセスでは、すべての索引スキャンに従う必要はありません。文に必要な列がすべて索引に含まれていると、ROWID による表アクセスは行われない場合があります。

注意： ROWID は、データが格納されている場所を表す Oracle の内部表現です。ROWID はバージョン間で変更される場合があります。位置に基づいたデータのアクセスは、お薦めしません。行の移行や連鎖によって、行が移動するためです。また、エクスポートやインポートの後も同様です。外部キーは主キーに基づいている必要があります。詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

索引スキャン

この方法では、文で指定された索引付きの列の値を使用して索引が検索され、行が取得されます。索引スキャンでは、索引の 1 つ以上の列値に基づいて索引からデータが取得されます。索引スキャンを実行するために、Oracle は文によってアクセスされた列に対応する索引を検索します。文が索引付けされた列にしかアクセスしない場合、Oracle は索引付けされた列値を表からではなく、索引から直接読み込みます。

索引には、索引の値の他に、その値を持っている行の ROWID も含まれています。したがって、索引付けされた列の他に別の列にも文がアクセスする場合、Oracle は、ROWID またはクラスタ・スキャンによる表アクセスのどちらかを使用して、表内の行を検索できます。

索引スキャンには次のタイプがあります。

- 索引一意スキャン
- 索引レンジ・スキャン
- 索引レンジ・スキャン降順
- 索引スキップ・スキャン
- 全体スキャン
- 高速全索引スキャン
- 索引結合
- ビットマップ結合

索引一意スキャン

このスキャンは 1 つの ROWID しか戻しません。文が単一行にしかアクセスしないことが保証されている一意制約または主キー制約が存在する場合、Oracle は一意スキャンを実行します。

1-19 ページの例 1-4「EXPLAIN PLAN 出力」では、それぞれ job_id_pk 索引と dept_id_pk 索引を使用して、jobs 表および departments 表で索引スキャンが実行されます。

オプティマイザが索引一意スキャンを使用する場合 このアクセス・パスは、一意（B ツリー）索引のすべての列が等価条件で指定される場合に使用します。

> **関連項目：** 索引構造の詳細と B ツリーの検索方法の詳細は、『Oracle9*i* データベース概要』を参照してください。

索引一意スキャンのヒント 一般に、一意スキャンを行うためのヒントを使用する必要はありません。ただし、表がデータベース・リンクにまたがっていて、ローカル表からアクセスされる場合や、表が小さく、オプティマイザが全表スキャンを選択する場合があります。

ヒント INDEX(*alias index_name*) は使用する索引を指定しますが、アクセス・パス（レンジ・スキャンや一意スキャン）は指定しません。INDEX ヒントの詳細は、5-12 ページの「INDEX」を参照してください。

索引レンジ・スキャン

索引レンジ・スキャンは、選択性の高いデータにアクセスする共通の操作です。このスキャンは、境界（両側で境界付き）スキャンまたは非有界（片側または両側で）スキャンとすることができます。データは、索引列の昇順に戻されます。同じ値を持つ複数の行は、ROWID で昇順にソートされます。

データを一定順序でソートする必要がある場合は、ORDER BY 句を使用し、索引には依存しません。索引を使用して ORDER BY 句を満たせる場合、オプティマイザはこのオプションを使用し、ソートを回避します。

例 1-8 では順序がレガシー・システムからインポートされており、レガシー・システム内での参照で順序の問合せを行います。この参照が order_date であると仮定します。

例 1-8 索引レンジ・スキャン

```
SELECT order_status, order_id
  FROM orders
 WHERE order_date = :b1;

-----------------------------------------------------------------------------------------
| Id  | Operation                    |  Name              | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                    |     1 |    20 |     3  (34)|
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDERS             |     1 |    20 |     3  (34)|
|*  2 |   INDEX RANGE SCAN           | ORD_ORDER_DATE_IX  |     1 |       |     2  (50)|
-----------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ORDERS"."ORDER_DATE"=:Z)
```

これは選択性の高い問合せである必要があり、問合せでは列の索引を使用して目的の行を取得します。戻されたデータは、order_date の ROWID により昇順でソートされます。索引列 order_date は、ここで選択された行と同じなので、データは ROWID でソートされます。

オプティマイザが索引レンジ・スキャンを使用する場合 オプティマイザは、次のように条件で指定された索引の 1 つ以上の先頭列を検出したとき、レンジ・スキャンを使用します。

- col1 = :b1
- col1 < :b1
- col1 > :b1
- 索引内の先頭列に対する前述の条件の AND 組合せ
- col1 like '%ASD' ワイルド・カードの検索は先頭に置かないでください。条件 col1 like '%ASD' は、レンジ・スキャンでは結果が出ません。

レンジ・スキャンでは、一意索引または非一意索引を使用できます。レンジ・スキャンでは、索引列が ORDER BY/GROUP BY 句を構成しているときにソートを回避します。

索引レンジ・スキャンのヒント オプティマイザが他の索引を選択したり、全表スキャンを使用する場合は、ヒントが必要になる場合があります。ヒント INDEX(*table_alias index_name*) では、使用する索引を指定します。INDEX ヒントの詳細は、5-12 ページの「INDEX」を参照してください。

order_id に偏ったデータ分布があると仮定します。列にはヒストグラムがあるため、オプティマイザがその分散について理解します。ただし、オプティマイザはバインド変数では値がわからず、全表スキャンを選択する可能性があります。次の 2 つのオプションがあります。

- バインド変数でなくリテラルを使用します（この場合は、SQL 文が共有されていないため、問題が生じるおそれがあります）。
- 文を共有するためにヒントを使用します。

例 1-9 は、INDEX ヒント使用前の問合せを示したものです。

例 1-9 INDEX ヒントを使用する前

```
SELECT l.line_item_id, order_id, l.unit_price * l.quantity
  FROM order_items l
 WHERE l.order_id = :b1;

-------------------------------------------------------------------------
| Id  | Operation            |  Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |     6 |    90 |     4  (25)|
|*  1 |  TABLE ACCESS FULL   | ORDER_ITEMS  |     6 |    90 |     4  (25)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - filter("L"."ORDER_ID"=TO_NUMBER(:Z))
```

例 1-10 は、INDEX ヒントを使用した例 1-9 の問合せを示したものです。

例 1-10 INDEX ヒントとバインド変数を使用

```
SELECT /*+ INDEX(l item_order_ix) */ l.line_item_id, order_id,
                                     l.unit_price * l.quantity
  FROM order_items l
 WHERE l.order_id = :b1;

-----------------------------------------------------------------------------------
| Id  | Operation                    |  Name          | Rows  | Bytes | Cost (%CPU)|
-----------------------------------------------------------------------------------
|   0 | SELECT STATEMENT             |                |     6 |    90 |    10  (10)|
|   1 |  TABLE ACCESS BY INDEX ROWID| ORDER_ITEMS    |     6 |    90 |    10  (10)|
|*  2 |   INDEX RANGE SCAN           | ITEM_ORDER_IX  |     6 |       |     2  (50)|
-----------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("L"."ORDER_ID"=TO_NUMBER(:Z))
```

索引レンジ・スキャン降順

索引レンジ・スキャン降順は、データが降順で戻されること以外、索引レンジ・スキャンと同じです。デフォルトでは、索引は昇順に格納されます。通常、このスキャンが使用されるのは、最初に最新のデータを戻すためにデータを降順に並べる場合や、指定された値より小さい値を探す場合です。

例 1-11 では、order_id と line_item_id の 2 列一意索引が使用されています。

例 1-11 2 列一意索引を使用した索引レンジ・スキャン降順

```
SELECT line_item_id, order_id
  FROM order_items
 WHERE order_id < :b1
 ORDER BY order_id DESC;

-------------------------------------------------------------------------------------
| Id  | Operation                     |  Name          | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------------
|   0 | SELECT STATEMENT              |                |    33 |   231 |     3  (34)|
|   1 |  TABLE ACCESS BY INDEX ROWID  | ORDER_ITEMS    |    33 |   231 |     3  (34)|
|*  2 |   INDEX RANGE SCAN DESCENDING| ITEM_ORDER_IX  |     6 |       |     3  (34)|
-------------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("ORDER_ITEMS"."ORDER_ID"<TO_NUMBER(:Z))
       filter("ORDER_ITEMS"."ORDER_ID"<TO_NUMBER(:Z))
```

データは、選択された行の order_id、line_item_id および ROWID により、降順でソートされます。ただし、order_id と line_item_id（item_order_ix が 2 つの列の一意索引）には 1 行しかないため、行は order_id および line_item_id でソートされます。

オプティマイザが索引レンジ・スキャン降順を使用する場合 オプティマイザは、索引で降順句による順序付けを満たせる場合に、索引レンジ・スキャン降順を使用します。

索引レンジ・スキャン降順のヒント ヒント INDEX_DESC(*table_alias index_name*) は、このアクセス・パスに使用します。INDEX_DESC ヒントの詳細は、5-15 ページの「INDEX_DESC」を参照してください。

索引スキップ・スキャン

索引スキップ・スキャンにより、接頭辞の付いていない列による索引スキャンが改善されます。多くの場合、索引ブロックをスキャンするほうが、表データ・ブロックをスキャンするより高速です。

スキップ・スキャンにより、コンポジット索引をさらに小さい副索引に論理的に分割できます。スキップ・スキャンでは、コンポジット索引の初期列が問合せで指定されていません。つまり、その列がスキップされます。

論理副索引の個数は、初期列内の個別値の数で決まります。スキップ・スキャンは、コンポジット索引の先頭列に個別値がほとんどなく、索引の非先頭キーに値が多数ある場合に便利です。

例 1-12 索引スキップ・スキャン

たとえば、コンポジット索引（sex、employee_id）を持つ表 employees（sex、employee_id、address）を想定します。このコンポジット索引を分割すると、結果は 2 つの論理副索引 M および F になります。

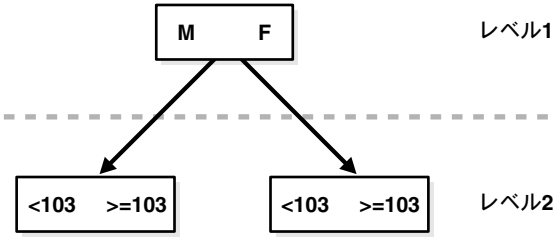
この例で、次の索引データがあるとします。

```
('F',98)
('F',100)
('F',102)
('F',104)
('M',101)
('M',103)
('M',105)
```

索引は、論理的に次の 2 つの副索引に分割されます。

- 値 F を持つキーがある第 1 の副索引。
- 値 M を持つキーがある第 2 の副索引。

図 1-4 索引スキップ・スキャン



列 sex が、次の問合せでスキップされます。

```
SELECT *
   FROM employees
WHERE employee_id = 101;
```

索引の完全なスキャンは行われませんが、まず値 F を持つ副索引が検索され、次に値 M を持つ副索引が検索されます。

全体スキャン

全体スキャンを使用できるのは、述語が索引内の列の 1 つを参照している場合です。述語が索引のドライバである必要はありません。述語がなく、次の条件が 2 つとも満たされる場合も、全体スキャンを使用できます。

- 問合せで参照される表の列すべてが索引に含まれている。
- 少なくとも索引列の 1 つが NOT NULL。

全体スキャンを使用してソート操作を解消できるのは、データが索引キーで並べられるためです。全体スキャンではブロックが単独で読み込まれます。

高速全索引スキャン

高速全索引スキャンは、問合せに必要なすべての列が索引に含まれ、索引キー内の 1 つ以上の列に NOT NULL 制約が存在する場合に、全表スキャンの代用として使用されます。高速全スキャンは、表にアクセスすることなく索引そのものに存在するデータにアクセスします。このスキャンでソート操作を解消できないのは、データが索引キーで並べられないためです。高速全スキャンでは、全索引スキャンとは異なりマルチブロック READ を使用して索引全体が読み込まれ、パラレル実行も可能です。

高速全スキャンは、CBO でのみ使用できます。高速全スキャンは、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または INDEX_FFS ヒントを使用して指定できます。高速全索引スキャンはビットマップ索引に対しては実行できません。

高速全スキャンは、表スキャンと同様にマルチブロック I/O を使用してパラレル化できるため、通常の全索引スキャンより高速です。

高速全索引スキャンのヒント 高速全索引スキャンには、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。INDEX_FFS ヒントの詳細は、5-16 ページの「INDEX_FFS」を参照してください。

高速全索引スキャンの制限 高速全索引スキャン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。
- 高速全索引スキャンをパラレルで実行する場合は、索引に対する parallel 句が必要です。索引のパラレル化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。
- 索引は分析済である必要があります。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。

索引結合

索引結合は、問合せで参照される表の列すべてが含まれている複数の索引のハッシュ結合です。索引結合が使用された場合は、関連するすべての列値が索引から取り出されるので、表アクセスは必要ありません。索引結合は、ソート操作の絞込みには使用できません。索引結合は、CBO でのみ使用できます。

索引結合のヒント 索引結合は、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または INDEX_JOIN ヒントを使用して指定できます。INDEX_JOIN ヒントの詳細は、5-15 ページの「INDEX_JOIN」を参照してください。

ビットマップ結合

ビットマップ結合は、キー値用のビットマップおよび各ビット位置を ROWID に変換するマッピング機能を使用します。ビットマップは、AND 条件と OR 条件の変換にブール演算を使用して、WHERE 句内の複数の条件に対応する索引を効果的にマージすることができます。

ビットマップ・アクセスは、CBO でのみ使用できます。

注意： ビットマップ索引とビットマップ・ジョイン・インデックスが使用可能なのは、Oracle9*i* Enterprise Edition をご購入されている場合のみです。

関連項目： ビットマップ索引の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

クラスタ・スキャン

クラスタ・スキャンは、索引クラスタに格納された表から、クラスタ・キー値の等しい行すべてを取得するときに使用されます。索引クラスタ内においては、同一のクラスタ・キー値を持つすべての行が同じデータ・ブロックに格納されています。クラスタ・スキャンを実行するために、Oracle は、クラスタ索引をスキャンすることによって、選択されている行の ROWID を最初に取得します。次に、Oracle はその行を ROWID に従って探します。

ハッシュ・スキャン

ハッシュ・スキャンは、ハッシュ値に基づいて行をハッシュ・クラスタに配置するために使用します。ハッシュ・クラスタ内においては、同一のハッシュ値を持つすべての行が同じデータ・ブロックに格納されています。ハッシュ・スキャンを実行するために、Oracle は、文によって指定されたクラスタ・キー値にハッシュ関数を適用することによって、最初にハッシュ値を取得します。次に、Oracle はそのハッシュ値を持つ行が含まれているデータ・ブロックを操作します。

サンプル表スキャン

サンプル表スキャンでは、表からデータのランダムなサンプルが取り出されます。このアクセス・パスは、文の FROM 句に SAMPLE 句または SAMPLE BLOCK 句が含まれているときに使用されます。行単位でサンプリングする（SAMPLE 句）ときにサンプル表スキャンを実行するには、表の中の指定されたパーセントの行を読み取ります。ブロック単位（SAMPLE BLOCK 句）でサンプリングするときにサンプル表スキャンを実行するには、表のブロックの中の指定されたパーセントのブロックを読み取ります。

問合せが結合またはリモート表に関連する場合のサンプル表スキャンはサポートされていません。ただし、CREATE TABLE AS SELECT 問合せを使用して同等のスキャンを実行し、基礎となる表のサンプルをマテリアライズできます。また、新たに作成された表サンプルを参照するために元の問合せをリライトできます。追加の問合せを書き込んで、その他の表のサンプルをマテリアライズすることができます。サンプル表スキャンには CBO が必要です。

例 1-13 は、サンプル表スキャンを使用して、ブロックによるサンプリングを行って employees 表の 1% にアクセスします。

例 1-13 サンプル表スキャン

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

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
-------------------------------------------------------------------------
| Id  | Operation            |  Name       | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |             |     1 |    68 |     3  (34)|
|   1 |  TABLE ACCESS SAMPLE | EMPLOYEES   |     1 |    68 |     3  (34)|
-------------------------------------------------------------------------
```

CBO によるアクセス・パスの選択方法

CBO は、次の要因に従ってアクセス・パスを選択します。

- 文で使用可能なアクセス・パス
- 各アクセス・パスまたはパスの組合せを使用して文を実行するための見積りコスト

アクセス・パスを選択する場合、オプティマイザは、文の WHERE 句、および SAMPLE または SAMPLE BLOCK 句の FROM 句の条件を検査して、使用可能なアクセス・パスを最初に判断します。次に、オプティマイザは、使用可能なアクセス・パスを使用して可能な実行計画のセットを生成し、文にアクセス可能な索引、列および表の統計を使用して各見積りコストを生成します。そして最後に、オプティマイザは見積りコストが最も少ない実行計画を選択します。

アクセス・パスを選択する場合、CBO は次の影響を受けます。

- オプティマイザ・ヒント

  オプティマイザが選択した使用可能なアクセス・パスは、ヒントを使用して上書きできますが、文の FROM 句に SAMPLE または SAMPLE BLOCK が含まれているときには上書きできません。

  > **関連項目：** SQL 文のヒントの詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

- 古い統計

  たとえば、表が作成された後に解析されなかった場合およびブロックが DB_FILE_MULTIBLOCK_READ_COUNT 最高水位標より少ない場合は、オプティマイザはその表は小さいと判断し、全表スキャンが使用されます。ALL_TABLES 表内の LAST_ANALYZED 列および BLOCKS 列を見て、統計が何を反映しているかを確認してください。

アクセス・パスの選択の例

この項では、オプティマイザによるアクセス・パスの選択方法を説明します。

ケース 1 例 1-14 では、問合せの WHERE 句に等価条件が使用されて Jackson という名前のすべての従業員が選択されます。

例 1-14 アクセス・パスの選択

```
SELECT *
  FROM employees
  WHERE last_name = 'JACKSON';
```

last_name 列が、一意キーまたは主キーである場合は、Jackson という名前の従業員は 1 人のみであることがオプティマイザによって判断され、問合せから 1 つの行のみが戻されます。この場合、問合せの選択性は非常に高く、オプティマイザは、一意キーまたは主キーを規定する索引に対して一意スキャンを使用して表にアクセスする傾向があります。

ケース 2 例 1-14 の問合せについてもう一度検討します。last_name 列が一意キーまたは主キーでない場合、オプティマイザは問合せの選択性の見積りに次の統計を使用する可能性があります。

- USER_TAB_COLUMNS.NUM_DISTINCT（表内の各列の値の数）
- USER_TABLES.NUM_ROWS（各表内の行数）

employees 表内の行数を、last_name 列内の個別値の数で除算することによって、オプティマイザは、同一の名前を持つ従業員の割合を見積ります。last_name 値が均一に分散していると想定し、オプティマイザは、問合せの選択性の見積り値としてこの割合を使用します。

ケース 3 次の問合せでは、ID 番号が 7500 未満のすべての従業員が選択されます。

```
SELECT *
  FROM employees
  WHERE employee_id < 7500;
```

問合せの選択性を見積もるため、オプティマイザは、WHERE 句条件に 7500 の境界値を使用します。また、使用可能な場合は、employee_id 列に対する HIGH_VALUE 統計と LOW_VALUE 統計も使用します。これらの統計は、USER_TAB_COL_STATISTICS ビュー（または USER_TAB_COLUMNS ビュー）で検索できます。オプティマイザは、employee_id 値が最低値と最高値の範囲内で均一に分布しているものと想定します。そして、この範囲のどの割合が値 7500 より小さいかを判断し、判断した値を、問合せの選択性の見積り値として使用します。

ケース 4 次の問合せでは、WHERE 句条件の境界値にリテラル値ではなくバインド変数が使用されます。

```
SELECT *
  FROM employees
  WHERE employee_id < :e1;
```

オプティマイザは、バインド変数 e1 の値を認識していません。e1 の値は問合せの実行ごとに異なっている可能性があります。そのため、オプティマイザはこの問合せの選択性を判断するのに、前述の例で説明した手段を使用できません。この場合、オプティマイザは内部デフォルト値を使用して、経験則から小さい値を選択性に推測します。オプティマイザは、演算子 <、>、<= または >= のいずれかを使用する条件の境界値としてバインド変数が使用されているかぎり、この仮説を作成します。

バインド変数に対するオプティマイザの処理によって、定数ではなくバインド変数の使用のみが異なる SQL 文に別の実行計画が選択されます。たとえば、オプティマイザは、Oracle プリコンパイラ・プログラム内のバインド変数を使用する埋込み SQL 文および SQL*Plus 内の定数を使用する同一の SQL 文に、異なる実行計画を選択する可能性があります。

ケース 5 次の問合せでは、BETWEEN 演算子を持っている条件に境界値として 2 つのバインド変数が使用されます。

```
SELECT *
  FROM employees
  WHERE employee_id BETWEEN :low_e AND :high_e;
```

オプティマイザは、BETWEEN 条件を、次の 2 つの条件にリライトします。

```
employee_id >= :low_e
employee_id <= :high_e
```

オプティマイザは、索引の使用を優先するために、索引が作成された列の選択性を低く（内部デフォルト値）見積ります。

ケース 6 次の問合せでは、BETWEEN 演算子を使用して 7500 から 7800 までの従業員 ID を持つすべての従業員が選択されます。

```
SELECT *
  FROM employees
  WHERE employee_id BETWEEN 7500 AND 7800;
```

この問合せの選択性を判断するため、オプティマイザは、WHERE 句条件を次の 2 つの条件にリライトします。

```
employee_id >= 7500
employee_id <= 7800
```

オプティマイザは、1-37 ページのケース 4 で説明した手段を使用して、条件それぞれの選択性を個別に見積ります。そして、オプティマイザは、これらの選択性（S1 と S2）および絶対値関数（ABS）を使用して、BETWEEN 条件の選択性（S）を見積ります。

```
S = ABS(S1 + S2 - 1)
```

結合について

結合は、複数の表からデータを取り出す文です。結合は FROM 句の中の複数の表で特性化され、各表の関係は WHERE 句の中に結合条件を設定することで定義されます。

この項では、次の内容を説明します。

- CBO による結合文の実行方法
- CBO による結合方法の選択方法
- CBO による結合タイプの実行計画の選択方法
- 結合方法
  - ネステッド・ループ結合
  - ハッシュ結合
  - ソート / マージ結合
  - デカルト結合
  - 外部結合

関連項目： 『Oracle9*i* SQL リファレンス』における結合に関する説明

CBO による結合文の実行方法

結合文に実行計画を選択するために、オプティマイザは、相互に関連する次の決定を行う必要があります。

アクセス・パス 単純な文では、オプティマイザは、結合文の各表からデータを取り出すアクセス・パスを選択する必要があります。

結合方法 行ソースの各ペアを結合するには、結合操作を Oracle が実行する必要があります。結合方法には、ネステッド・ループ結合、ソート / マージ結合、デカルト結合およびハッシュ結合があります。

結合順序 3 つ以上の表を結合する文を実行する場合、Oracle は 2 つの表を結合し、その結果作成された行ソースを次の表に結合します。このプロセスは、すべての表がその結果に結合されるまで続行されます。

関連項目：

- 8-3 ページ「RBO のアクセス・パス」
- 1-23 ページ「CBO のアクセス・パスについて」

CBO による結合方法の選択方法

オプティマイザは各結合方法のコストを見積り、コストが最も小さい方法を選択します。結合によって多数の行が戻される場合は、次の 3 つの要因が検討されます。

- 結合によって多数の行（通常、10,000 より多い行の場合に多数とみなされます）が戻されるときには、ネステッド・ループ結合は有効ではなく、オプティマイザはこの方法の使用を選択しない可能性があります。ネステッド・ループ結合のコストは、次の式で計算されます。

  ```
  cost= access cost of A + (access cost of B * number of rows from A)
  ```

- CBO を使用している場合は、結合によって多くの行が戻されるときにハッシュ結合が最も効果的です。ハッシュ結合のコストは、次の式で計算されます。

  ```
  cost= (access cost of A * number of hash partitions of B) + access cost of B
  ```

- RBO を使用している場合は、結合によって多くの行が戻されるときにマージ結合が最も効果的です。マージ結合のコストは、次の式で計算されます。

  ```
  cost= access cost of A + access cost of B +(sort cost of A + sort cost of B)
  ```

  データが事前ソートされている場合、ソート・コストは両方とも 0（ゼロ）です。

注意： コストベースの最適化を使用することを強くお薦めします。ルールベースの最適化は、将来のリリースでは使用されなくなります。

CBO による結合タイプの実行計画の選択方法

次の考慮事項は、コストベースとルールベース両方のアプローチに適用されます。

- オプティマイザは、2 つ以上の表を結合した結果を、最大 1 つの行を含む行ソースに限定するかどうかを最初に判断します。オプティマイザは、このような状況を表の UNIQUE 制約および PRIMARY KEY 制約に基づいて認識します。このような状況が存在する場合、オプティマイザはまずこれらの表を結合順序に並べます。その後で、残りの表の結合を最適化します。
- 外部結合条件を持つ結合文では、外部結合演算子のある表の結合順序は、条件内のその他の表の後にしてください。オプティマイザは、この規則に違反する結合順序を考慮しません。

CBO では、適用可能な結合順序、結合方法および使用可能なアクセス・パスに従ってオプティマイザが実行計画のセットを生成します。次に、オプティマイザは各計画のコストを見積り、コストが最も小さいものを選択します。オプティマイザは、次の方法でコストを見積ります。

- ネステッド・ループ操作のコストは、外部表で選択されている各行およびその行に対する内部表の一致行をメモリーに読み込むコストが基準になっています。オプティマイザは、データ・ディクショナリ内の統計を使用してこれらのコストを見積ります。
- ソート / マージ結合のコストは、主に、すべてのソースをメモリーに読み込んでソートするコストを基準にしています。

オプティマイザは、各操作のコストを判断するときにはその他の要因についても考慮します。次に例を示します。

- ソート領域のサイズが小さいと、小さいソート領域内でのソートに CPU 時間と I/O がより多く消費されるため、ソート / マージ結合のコストが大きくなる傾向があります。ソート領域のサイズは、初期化パラメータ SORT_AREA_SIZE によって指定されます。

  注意： インスタンスが共有サーバー・オプションで設定されない限り SORT_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定が使用できるようにすることをお薦めします。SORT_AREA_SIZE は、下位互換性のためにのみ残されています。

- マルチブロック READ カウントが大きいと、ネステッド・ループ結合に関してソート / マージ結合のコストが少なくなる傾向があります。多数の連続したブロックが単独の I/O でディスクから読み込まれる場合は、全表スキャンよりもパフォーマンスを改善するために、ネステッド・ループ結合の内部表についての索引が少なくなる傾向があります。マルチブロック READ カウントは、初期化パラメータ DB_FILE_MULTIBLOCK_READ_COUNT によって指定されます。

CBO では、ORDERED ヒントを使用して結合順序に関するオプティマイザの選択を上書きできます。ORDERED ヒントによって、外部結合に関するこの規則に違反する結合順序が指定された場合、オプティマイザはこのヒントを無視して順序を選択します。結合方法に関するオプティマイザの選択も、ヒントを使用して上書きできます。

関連項目： オプティマイザ・ヒントの詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

CBO によるアンチ結合の実行方法

アンチ結合では、述語の右側に対応する行が存在しない、左側の行が戻されます。つまり、右側における（NOT IN）副問合せとの適合に失敗した、左側の行が戻されます。たとえば、アンチ結合では、特定の部門に属していない従業員のリストなどを選択できます。

例 1-15 従業員および部門のアンチ結合

```
SELECT * FROM employees
  WHERE department_id NOT IN
    (SELECT department_id FROM departments
      WHERE location_id = 1700);
```

オプティマイザは、NOT IN 副問合せに対してネステッド・ループ結合をデフォルトで使用します。ただし、MERGE_AJ ヒント、HASH_AJ ヒント、あるいは NL_AJ ヒントが使用されている場合で、様々な必須条件が一致するときは、NOT IN 非相関副問合せがソート / マージ結合またはハッシュ・アンチ結合に変換されます。

CBO によるセミ結合の実行方法

セミ結合では、右側の複数の行が副問合せの基準を満たしているときに、述語の左側の行から、EXISTS 副問合せに重複することなく適合した行が戻されます。次に例を示します。

例 1-16 従業員および部門のセミ結合

```
SELECT * FROM departments
  WHERE EXISTS
    (SELECT * FROM employees
      WHERE departments.department_id = employees.department_id
        AND employees.salary > 2500);
```

この問合せでは、employees 内の多数の行が副問合せに適合している場合にも、departments から戻される必要があるのは 1 つの行のみです。employees に salary 列の索引が存在しない場合は、セミ結合を使用して問合せのパフォーマンスを改善できます。

オプティマイザは、デフォルトで、内部の問合せとマージできない IN または EXISTS 副問合せに、ネステッド・ループ・アルゴリズムを使用します。ただし、MERGE_SJ ヒント、HASH_SJ ヒント、あるいは NL_SJ ヒントが使用されていて、様々な必須条件が一致する場合は、副問合せがソート / マージ結合またはハッシュ・セミ結合に変換されます。

> **関連項目：** オプティマイザ・ヒントの詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

注意： 副問合せが OR ブランチにある場合は、セミ結合は変換できません。これはアンチ結合にも適用されます。

CBO によるスター・クエリーの実行方法

データ・ウェアハウスには、大きいファクト表といくつかの小さいディメンション（参照）表を含むスター・スキーマを中心に設計されているものがあります。ファクト表は主要な情報を格納します。各ディメンション表は、ファクト表の属性に関する情報を格納します。

スター・クエリーは、ファクト表と複数の参照表との間の結合です。各参照表は主キーを使用して、対応するファクト表の外部キーに結合されますが、参照表どうしは結合されません。

CBO は、スター・クエリーを認識すると、そのスター・クエリーにとって効果的な実行計画を生成します。スター・クエリーは、RBO では認識されません。

通常、ファクト表にはキーおよびメジャーが含まれています。たとえば、単純なファクト表には、メジャー `Sales` とキー `Time`、`Product`、`Market` などが含まれています。この場合、`Time`、`Product` および `Market` に対応するディメンション表が存在することが考えられます。たとえば、`Product` ディメンション表には、通常、ファクト表に表示されている各製品番号についての情報などが登録されています。

スター型結合はファクト表の外部キーの結合を、対応するディメンション表の主キーに使用します。ファクト表には、通常、このタイプの結合を容易にするために外部キー列のコンポジット索引または各外部キー列の個別のビットマップ索引が存在します。

> **関連項目：** スター・クエリーをチューニングする方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

ネステッド・ループ結合

ネステッド・ループ結合は、データの小さいサブセットを結合する場合や、結合条件が第 2 の表にアクセスする効率的な方法である場合に有効です。

内部表が外部表から（外部表によって）駆動されることを確認することが重要です。内部表のアクセス・パスが外部表とは独立している場合は、外部ループを繰り返すたびに同じ行が取得されます。これは、パフォーマンスをかなり低下させてしまいます。そのような場合は、2 つの独立した行ソースを結合するハッシュ結合のほうがパフォーマンスが優れています。

関連項目： 1-50 ページ「デカルト結合」

ネステッド・ループ結合には、次のステップがあります。

1. オプティマイザで駆動表が決定され、これが外部表に指定されます。
2. その他の表は、内部表に指定します。
3. 外部表にあるすべての行について、内部表にあるすべての行がアクセスされます。外部ループは外部表にあるすべての行に対するものであり、内部ループは内部表の中にあるすべての行に対するものです。次のように、外部ループは実行計画の内部ループの前に表示されます。

```
NESTED LOOPS
  outer_loop
  inner_loop
```

ネステッド・ループ結合

この項では、例 1-3 の問合せにおける次のネステッド・ループの外部ループおよび内部ループを説明します。

```
...
|   2 |   NESTED LOOPS                |             |     3 |   141 |     7  (15)|
|*  3 |    TABLE ACCESS FULL          | EMPLOYEES   |     3 |    60 |     4  (25)|
|   4 |    TABLE ACCESS BY INDEX ROWID| JOBS        |    19 |   513 |     2  (50)|
|*  5 |     INDEX UNIQUE SCAN         | JOB_ID_PK   |     1 |       |            |
...
```

この例では、外部ループが employees 表のすべての行を取得します。外部ループによって取得された各従業員に対して、内部ループが jobs 表内の関連行を取得します。

外部ループ 例 1-4 の実行計画では、外部ループおよびそれと等価の文は次のとおりです。

```
3 |    TABLE ACCESS FULL         | EMPLOYEES

SELECT e.employee_id, e.salary
  FROM employees e
 WHERE e.employee_id < 103
```

内部ループ 次のように、例 1-4 の実行計画は、外部ループから取得されたすべての行について繰り返される内部ループであることを示します。

```
4 |    TABLE ACCESS BY INDEX ROWID| JOBS
5 |     INDEX UNIQUE SCAN         | JOB_ID_PK

SELECT j.job_title
 FROM jobs j
    WHERE e.job_id = j.job_id
```

オプティマイザがネステッド・ループ結合を使用する場合

オプティマイザは、2 つの表の間で適切な駆動条件で少数の行を結合する場合に、ネステッド・ループ結合を使用します。外部ループから内部ループに起動するので、実行計画の中の表の順序が重要になります。

外部ループは駆動行ソースです。このループは、結合条件を駆動するための一連の行を生成します。行ソースは、索引スキャンまたは全表スキャンでアクセスされる表とすることができます。また、他の操作からでも行を生成できます。たとえば、ネステッド・ループ結合からの出力は別のネステッド・ループ結合の行ソースとして使用できます。

内部ループは外部ループから戻された行ごとに、索引スキャンによって反復されます。内部ループのアクセス・パスが外部ループに依存していない場合は、デカルト積で終了することが可能で、外部ループの反復ごとに、内部ループは同じ行セットを生成します。したがって、2 つの独立した行ソースをまとめて結合する場合は、他の結合方法を使用することをお薦めします。

ネステッド・ループ結合のヒント

オプティマイザが他の結合方法を選択する場合は、`USE_NL(table1 table2)` ヒントを使用します。`table1` と `table2` は、結合される表の別名です。

データが十分に小さい SQL 例の場合は、オプティマイザは全表スキャンを優先してハッシュ結合を使用します。1-47 ページの例 1-17「ハッシュ結合」に示したのが、その SQL の例です。ただし、`USE_NL` ヒントを追加して結合方法をネステッド・ループに変更できます。`USE_NL` ヒントの詳細は、5-24 ページの「USE_NL」を参照してください。

ネステッド・ループのネスト

ネステッド・ループの外部ループ自体もネステッド・ループにできます。2 つ以上の外部ループをまとめてネストし、必要な数の表に結合できます。次に示すように、各ループはデータ・アクセス方法です。

```
SELECT STATEMENT
 NESTED LOOP 3
  NESTED LOOP 2           (OUTER LOOP 3.1)
   NESTED LOOP 1          (OUTER LOOP 2.1)
    OUTER LOOP 1.1      - #1
    INNER LOOP 1.2      - #2
   INNER LOOP 2.2       - #3
  INNER LOOP 3.2        - #4
```

ハッシュ結合

ハッシュ結合は、大きなデータ・セットを結合する場合に使用します。オプティマイザは、2 つの表またはデータ・ソースの小さいほうを使用して、メモリー内の結合キーにハッシュ表を作成します。次に、大きいほうの表をスキャンし、ハッシュ表を調べて結合された行を見つけます。

この方法は、小さいほうの表が使用可能なメモリー内に収まる場合に最適です。これにより、コストが 2 つの表のデータに対する 1 回のリード・パスに制限されます。

ただし、ハッシュ表が大きすぎてメモリー容量を超えると、オプティマイザはハッシュ表を異なるパーティションに分割します。パーティションが割り当てられたメモリー容量を超えると、各部分がディスク上の一時セグメントに書き込まれます。一時エクステント・サイズが大きくなるほど、ディスクにパーティションを書き込む際の I/O が改善されます。一時エクステントのサイズは約 1 MB にすることをお薦めします。一時エクステント・サイズは、永続的な表領域が INITIAL および NEXT で指定され、一時表領域が UNIFORM SIZE で指定されています。

ハッシュ表が完成した後、次の処理が行われます。

1. 2 番目の、大きい方の表がスキャンされます。
2. この表は、小さい表のようなパーティションに分割されます。
3. パーティションがディスクに書き込まれます。

ハッシュ表の作成が終了すると、ハッシュ表パーティション全体をメモリーに常駐させることができます。その場合、第 2 の（より大きな）表のための対応するパーティションを作成する必要はありません。この表がスキャンされると、常駐ハッシュ表パーティションにハッシュされる行を結合し、ただちに戻すことができます。

各ハッシュ表パーティションはメモリーに読み込まれ、次の処理が発生します。

1. 第 2 の表の対応するパーティションがスキャンされます。
2. 結合された行を戻すために、ハッシュ表が調べられます。

この処理は、パーティションの残りの部分について繰り返されます。コストは、データに対し、読込み 2 回と書込み 1 回まで増加する可能性があります。

ハッシュ表がメモリーに収まらない場合は、第 2 の表から取得された行によって、その一部をスワップ・インおよびスワップ・アウトする必要がある場合もあります。このシナリオのパフォーマンスは極端に低くなる可能性があります。

オプティマイザがハッシュ結合を使用する場合

オプティマイザは、2 つの表が等価結合で結合され、次の条件のいずれかが真である場合に、ハッシュ結合で 2 つの表を結合します。

- 大量のデータを結合する必要がある。
- 表の大きな部分を結合する必要がある。

例 1-17 では、表 `orders` がハッシュ表の作成に使用されます。また、後でスキャンされる `order_items` は、これより大きな表です。

例 1-17 ハッシュ結合

```
SELECT o.customer_id, l.unit_price * l.quantity
  FROM orders o ,order_items l
 WHERE l.order_id = o.order_id;

-------------------------------------------------------------------------------
| Id  | Operation            |  Name        | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |              |   665 | 13300 |     8  (25)|
|*  1 |  HASH JOIN           |              |   665 | 13300 |     8  (25)|
|   2 |   TABLE ACCESS FULL  | ORDERS       |   105 |   840 |     4  (25)|
|   3 |   TABLE ACCESS FULL  | ORDER_ITEMS  |   665 |  7980 |     4  (25)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   1 - access("L"."ORDER_ID"="O"."ORDER_ID")
```

ハッシュ結合のヒント

2 つの表をまとめて結合するときに、ハッシュ結合を適用するようにオプティマイザにアドバイスするには、USE_HASH ヒントを使用します。ハッシュ結合の使用に問題がないか、HASH_AREA_SIZE と HASH_JOIN_ENABLED の各パラメータの値を調べてください。

注意： インスタンスが共有サーバー・オプションで設定されない限り HASH_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。HASH_AREA_SIZE は、下位互換性のためにのみ残されています。

USE_HASH ヒントの詳細は、5-26 ページの「USE_HASH」を参照してください。

ソート / マージ結合

ソート / マージ結合を使用して、2 つの独立したソースからの行を結合できます。ハッシュ結合は、一般に、ソート / マージ結合よりパフォーマンスが優れています。次の条件が 2 つとも存在する場合は、ハッシュ結合よりソート / マージ結合のほうがパフォーマンスの点で優れています。

- 行ソースはソート済み。
- ソート操作を終了する必要なし。

ソート / マージ結合に、より低速のアクセス方法（全表スキャンとは対照的な索引スキャン）の選択が含まれている場合、ソート / マージを使用する利点が失われる可能性があります。

ソート / マージ結合は、2 つの表の間の結合条件が <、<=、> または >= などの等価条件ではない（ただし、非等価ではない）場合に有効です。ソート / マージ結合は、大きいデータ・セットの場合にネステッド・ループ結合よりパフォーマンスが優れています。等価条件がないかぎり、ハッシュ結合を使用できません。

マージ結合には、駆動表の概念はありません。結合には、2 つのステップが含まれています。

1. ソート結合操作 : 両方の入力が、結合キーでソートされる。
2. マージ結合処理 : ソートされたリストがマージされる。

入力がすでに結合列でソートされている場合、その行ソースに対してソート結合操作は行われません。

オプティマイザがソート / マージ結合を使用する場合

オプティマイザは、次の条件が真である場合に、ハッシュ結合よりソート / マージ結合を選択して大量のデータを結合します。

- 2 つの表の間の結合条件が、等価結合ではない。
- OPTIMIZER_MODE が RULE に設定されている。
- HASH_JOIN_ENABLED が false である。
- ソートが他の操作ですでに要求されているため、オプティマイザは、ハッシュ結合よりソート / マージを使用するほうがコストが低いと判断した。
- オプティマイザが、HASH_AREA_SIZE と SORT_AREA_SIZE の設定に基づいて、ハッシュ結合のコストのほうが高いと判断した。

注意： インスタンスが共有サーバー・オプションで設定されない限り HASH_AREA_SIZE および SORT_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定が使用できるようにすることをお薦めします。HASH_AREA_SIZE および SORT_AREA_SIZE は、下位互換性のためにのみ残されています。

ソート / マージ結合のヒント

ソート / マージ結合を使用するようオプティマイザに指示するには、USE_MERGE ヒントを適用します。また、アクセス・パスを設定するためのヒントを与える必要がある場合があります。

USE_MERGE ヒントでオプティマイザを上書きした方がよい場合があります。たとえば、オプティマイザは表に対する全体スキャンを選択して問合せ内でのソート操作を回避できます。ただし、大きい表は全表スキャンによる高速アクセスの場合とは異なり、索引および単一ブロック読込みでアクセスされるため、コストがかかります。

USE_MERGE ヒントの詳細は、5-25 ページの「USE_MERGE」を参照してください。

デカルト結合

他の表への結合条件を文中に持たない表が 1 つ以上ある場合に、デカルト結合が使用されます。オプティマイザは、データ・ソースにあるすべての行を、2 つのセットのデカルト積を作成しながら、別のデータ・ソースにあるすべての行と結合します。

オプティマイザがデカルト結合を使用する場合

オプティマイザは、結合条件のない 2 つの表を結合する指示を受けると、デカルト結合を実行します。場合によっては、2 つの表の間に共通のフィルタ条件が存在して、オプティマイザが可能な結合条件として採用する可能性があります。これは、実行計画で、この結合にはデカルト積とするフラグが付けられていないため、より危険性が高くなります。

デカルト結合は一般に、SQL が適切に作成されていないことから生じます。結合基準を指定して、デカルト積を回避する必要があります。問合せが大量の表を参照することができ、FROM 句に表が追加される場合がありますが、WHERE 句に追加されることはありません。このような問合せには、DISTINCT 句を使用して複数行を取り除くことができます。ただし、DISTINCT 句がデカルト積で生成された余計なタプルを削除することは可能ですが、パフォーマンスが大幅に低下することがあります。

ネステッド・ループ操作の内部表が、外部表からではなく独立した行ソースから駆動される SQL 問合せの場合、アクセスされた行はデカルト積の場合と同じである可能性があります。結合条件は存在しますが、表にアクセスした後に適用されるため、結果はデカルト積ではありません。ただし、表にアクセスするコスト（アクセスされる行）はほぼ同じです。

デカルト結合のヒント

ORDERED ヒントを適用すると、オプティマイザはデカルト結合を使用します。結合表を指定する前に表を指定すると、オプティマイザはデカルト結合を行います。ORDERED ヒントの詳細は、5-23 ページの「ORDERED」を参照してください。

外部結合

外部結合は単純結合の結果を拡張したものです。外部結合は、結合条件に一致するすべての行および結合条件が他の表のどの行とも一致しない、表の一部またはすべての行を戻します。

ネステッド・ループ外部結合

この操作は、外部結合が 2 つの表の間で使用されるときに使用します。外部結合は、内部（オプション）表に対応する行がない場合でも外部（保たれている）表の行を戻します。

正規の外部結合で、オプティマイザはコストに基づいて表（駆動する側と駆動される側）の順序を選択します。ただし、ネステッド・ループ外部結合では、表の順序は結合条件で決定されます。行が保たれる外部表は、内部表に駆動する場合に使用します。

オプティマイザは、ネステッド・ループ結合を使用し、次の状況で外部結合を処理します。

- 外部表から内部表まで起動できる。
- データ量が少なく、ネステッド・ループ方法が効果的と判断できる。

ネステッド・ループの外部結合例の場合は、USE_NL ヒントを例 1-18 に追加してネステッド・ループを使用するようにできます。その例を次に示します。

```
SELECT /*+ USE_NL(c o) */ cust_last_name, sum(nvl2(o.customer_id,0,1)) "Count"
```

ハッシュ結合外部結合

データ量が十分大きく、ハッシュ結合方法が有効と判断される場合や、外部表から内部表を駆動できない場合、オプティマイザは外部結合の処理にハッシュ結合を使用します。

外部結合と同様に、表の順序はコストではなく結合条件で判断されます。行が保たれている外部表はハッシュ表を作成する場合に使用され、内部表はハッシュ表を調べる場合に使用されます。

例 1-18 では、一般的なハッシュ結合外部結合の問合せが示されています。この例では、与信限度が 1000 を超えるすべての顧客が問い合されます。外部結合は、オーダーを持たない顧客を見逃さないようにするために必要です。

例 1-18 ハッシュ結合外部結合

```
SELECT cust_last_name, sum(nvl2(o.customer_id,0,1)) "Count"
  FROM customers c, orders o
 WHERE c.credit_limit > 1000
   AND c.customer_id = o.customer_id(+)
 GROUP BY cust_last_name;

-------------------------------------------------------------------------
| Id  | Operation            |  Name       | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------
|   0 | SELECT STATEMENT     |             |   168 |  3192 |    11  (28)|
|   1 |  SORT GROUP BY       |             |   168 |  3192 |    11  (28)|
|*  2 |   HASH JOIN OUTER    |             |   260 |  4940 |    10  (20)|
|*  3 |    TABLE ACCESS FULL | CUSTOMERS   |   260 |  3900 |     6  (17)|
|*  4 |    TABLE ACCESS FULL | ORDERS      |   105 |   420 |     4  (25)|
-------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------

   2 - access("C"."CUSTOMER_ID"="O"."CUSTOMER_ID"(+))
   3 - filter("C"."CREDIT_LIMIT">1000)
   4 - filter("O"."CUSTOMER_ID"(+)>0)
```

この問合せは、様々な条件に一致する顧客を検索します。内部表に対応する行が見つからないと、外部結合は外部（保たれている）表の行とともに内部表の列に対して NULL を戻します。この操作で、orders 行も持たない customers 行がすべて検索されます。

この場合、外部結合条件は次のとおりです。

```
customers.customer_id = orders.customer_id(+)
```

この条件の構成要素を次に示します。

- 外部表は customers です。
- 内部表は orders です。
- この結合は、orders 内に対応する行を持たない行を含む customers 行を保存します。
- ハッシュ表は、customers で作成されます。
- ハッシュ表は、orders で調べられます。

NOT EXISTS 副問合せを使用して行を戻す可能性がありますが、これは、表にあるすべての行の問合せを行うためであり、NOT EXISTS 副問合せがネストされていない場合は、ハッシュ結合のパフォーマンスは良くなります。

例 1-19 では、外部結合はマルチ表ビューに対して行われます。オプティマイザは通常の結合のようにビューを操作したり、述語をプッシュできないので、ビューの行セット全体を作成します。

例 1-19 マルチ表ビューへの外部結合

```
SELECT c.cust_last_name, sum(revenue)
  FROM customers c, v_orders o
 WHERE c.credit_limit > 2000
   AND o.customer_id(+) = c.customer_id
 GROUP BY c.cust_last_name;

-------------------------------------------------------------------------------
| Id  | Operation              |  Name         | Rows  | Bytes | Cost (%CPU)|
-------------------------------------------------------------------------------
|   0 | SELECT STATEMENT       |               |   144 |  4608 |    16  (32)|
|   1 |  SORT GROUP BY         |               |   144 |  4608 |    16  (32)|
|*  2 |   HASH JOIN OUTER      |               |   663 | 21216 |    15  (27)|
|*  3 |    TABLE ACCESS FULL   | CUSTOMERS     |   195 |  2925 |     6  (17)|
|   4 |    VIEW                | V_ORDERS      |   665 | 11305 |            |
|   5 |     SORT GROUP BY      |               |   665 | 15960 |     9  (34)|
|*  6 |      HASH JOIN         |               |   665 | 15960 |     8  (25)|
|*  7 |       TABLE ACCESS FULL| ORDERS        |   105 |   840 |     4  (25)|
|   8 |       TABLE ACCESS FULL| ORDER_ITEMS   |   665 | 10640 |     4  (25)|
-------------------------------------------------------------------------------

Predicate Information (identified by operation id):
---------------------------------------------------
   2 - access("O"."CUSTOMER_ID"(+)="C"."CUSTOMER_ID")
   3 - filter("C"."CREDIT_LIMIT">2000)
   6 - access("O"."ORDER_ID"="L"."ORDER_ID")
   7 - filter("O"."CUSTOMER_ID">0)
```

ビュー定義は、次のようになります。

```
CREATE OR REPLACE view v_orders AS
SELECT l.product_id, SUM(l.quantity*unit_price) revenue,
       o.order_id, o.customer_id
  FROM orders o, order_items l
 WHERE o.order_id = l.order_id
 GROUP BY l.product_id, o.order_id, o.customer_id;
```

ソート / マージ外部結合

外部結合で、外部（保たれている）表から内部（オプション）表への駆動ができない場合、外部結合ではハッシュ結合またはネステッド・ループ結合が使用できません。その場合、外部結合では結合操作を実行するためにソート / マージ外部結合を使用します。

オプティマイザは、ネステッド・ループ結合の効率が悪い場合に外部結合にソート / マージを使用します。データ量が大きいためにネステッド・ループ結合の効率が悪いか、他の操作ですでにソートが要求されているために、ハッシュ結合よりソート / マージを使用するほうがコストが低いと判断された場合です。

完全外部結合

完全外部結合は、左と右の外部結合の組合せのように動作します。内部結合では、内部結合の結果で戻されなかった両方の表からの行は保たれており、NULL で拡張されます。つまり、完全外部結合を使用すると、表をまとめて結合できますが、結合される表内に対応する行を持たない行も示すことができます。

例 1-20 の問合せでは、全部門と、その各部門に属する全社員を取得しますが、これには次の内容も含まれます。

- 部門に属さない全社員
- 社員のいない全部門

例 1-20 完全外部結合

```
SELECT d.department_id, e.employee_id
  FROM employees e
  FULL OUTER JOIN departments d
    ON e.department_id = d.department_id
 ORDER BY d.department_id;
```

文からは、次の出力が作成されます。

```
DEPARTMENT_ID EMPLOYEE_ID
------------- -----------
           10         200
           20         201
           20         202
           30         114
           30         115
           30         116
...
          270
          280
                     178
                     207

125 rows selected.
```

コストベース・オプティマイザのパラメータの設定

この項には、オプティマイザに固有の、一部のパラメータが含まれています。次の項は、Oracle のアプリケーションをチューニングするときに特に役に立ちます。

CBO 機能の有効化

OPTIMIZER_FEATURES_ENABLE 初期化パラメータを設定して、オプティマイザの機能を有効化できます。

OPTIMIZER_FEATURES_ENABLE パラメータ

OPTIMIZER_FEATURES_ENABLE パラメータは、CBO のアンブレラ・パラメータの役割を果たします。リリースにより異なりますが、このパラメータを使用して一連の CBO 関連機能を有効にできます。このパラメータは、8.0.4、8.1.5 などのリリース番号に対応する有効な文字列値のリストのうちの 1 つを受け入れます。たとえば、次の文は、Oracle9*i* リリース 2（9.2）のオプティマイザ機能を使用可能にします。

```
OPTIMIZER_FEATURES_ENABLE=9.2.0;
```

この文では、問合せ計画の生成時に、リリース 2（9.2）オプティマイザ機能が使用可能にされています。たとえば、PL/SQL プロシージャで生成された再帰的ユーザー SQL に ALL_ROWS あるいは FIRST_ROWS オプティマイザ・モードを使用できます。Oracle8*i* リリース 8.1.6 以前のリリースでは、そのような再帰的 SQL には RULE または CHOOSE オプティマイザ・モードのみが使用されており、ユーザーが明示的に OPTIMIZER_MODE パラメータを FIRST_ROWS または ALL_ROWS に設定した場合は、かわりに CHOOSE モードが使用されていました。

OPTIMIZER_FEATURES_ENABLE パラメータは、Oracle8 リリース 8.0.4 で導入されました。この主な目的は、Oracle Server をアップグレード可能にすること、さらにアップグレード後も CBO の以前の動作を保持できることです。たとえば、リリース 8.1.5 からリリース 8.1.6 に Oracle Server をアップグレードすると、OPTIMIZER_FEATURES_ENABLE パラメータのデフォルトの値は 8.1.5 から 8.1.6 に変わります。このアップグレードのために、CBO は、8.1.5 ではなく 8.1.6 に基づいた最適化機能が有効にされています。

プラン・スタビリティまたは下位互換性の理由から、問合せ計画が、リリース 8.1.6 の新規オプティマイザ機能によって変更されることを望まない場合があります。そのような場合は、OPTIMIZER_FEATURES_ENABLE パラメータを以前のバージョンに設定します。たとえば、CBO の動作をリリース 8.1.5 に保持するには、次のようにパラメータを設定します。

```
OPTIMIZER_FEATURES_ENABLE=8.1.5;
```

この文により、8.1.5 より後のリリースで追加されたすべての新規オプティマイザ機能が無効になります。

注意： リリースをアップグレードし、使用可能な新規機能を有効にする場合は、OPTIMIZER_FEATURES_ENABLE パラメータを明示的に設定する必要はありません。

以前のリリースに OPTIMIZER_FEATURES_ENABLE パラメータを明示的に設定することはお薦めしません。実行計画または問合せパフォーマンスの問題点は、必要に応じて解決することをお薦めします。

表 1-3 では、OPTIMIZER_FEATURES_ENABLE パラメータに設定する各リリースの値と、それにより有効になるオプティマイザ機能について説明しています。

表 1-3 OPTIMIZER_FEATURES_ENABLE パラメータに含まれる機能

設定値	含まれる新機能
8.0.4	高速全索引スキャン
	順序付けられたネステッド・ループ結合方法
8.0.5	なし
8.0.6	改善された外部結合カーディナリティの見積り
8.1.4	なし
8.1.5	改善された B ツリー索引内の NULL 包含の検証
8.1.6	ユーザーの再帰的 SQL に対する FIRST_ROWS または ALL_ROWS モードの使用
	ネステッド・ループ結合で残された入力のランダム配分
	改善された行の長さの計算
	改善された、ヒストグラムに基づく選択性の計算方法
	副問合せの述語に基づくパーティション・プルーニング
8.1.7	共通副次式の最適化
	選択性の計算のために、TO_CHAR などの選択されたファンクションに埋め込まれた列の統計
	改善されたパーティション統計の集計（操作）

表 1-3 OPTIMIZER_FEATURES_ENABLE パラメータに含まれる機能（続き）

設定値	含まれる新機能
9.0.1	ユーザー定義のバインド変数の確認
	複合ビューのマージ
	述語結合のプッシュ
	B ツリー索引のみを持つ表へのビットマップ・アクセス・パスの考慮
	副問合せのネスト解除
	索引結合
9.2.0	小さい参照または参照表への結合に使用するパラレル問合せ内のパラレル・ブロードキャスト。

ユーザー定義のバインド変数の照合

CBO は、カーソルの最初の起動時にユーザー定義バインド変数の値を照合します。この機能により、WHERE 句の条件の選択性と、バインド変数のかわりにリテラルが使用されたかどうかが判断されます。カーソルのその後の起動時は照合が行われず、また、その後の起動で異なるバインド値を使用しても、カーソルは標準カーソル共有基準に基づいて共有されます。

バインド変数が文に使用されている場合、カーソルを共有するために異なる起動が同じ実行計画を使用するものとみなします。カーソルの異なる起動で様々な実行計画が使用される場合、バインド変数が SQL 文で適切に使用されていない可能性があります。

CBO の動作の制御

この項には、コストベース・オプティマイザの動作の管理に使用できる初期化パラメータの一部がリストされています。SQL 実行のパフォーマンスを向上するために、これらのパラメータを使用して様々なオプティマイザ機能を有効にすることができます。

CURSOR_SHARING

このパラメータは、SQL 文のリテラル値をバインド変数に変換します。値を変換するとカーソル共有が改善され、SQL 文の実行計画は影響を受けます。オプティマイザは、実際のリテラル値でなくバインド変数の有無に基づいて実行計画を生成します。

DB_FILE_MULTIBLOCK_READ_COUNT

このパラメータは、全表スキャンまたは高速全索引スキャン時に単一 I/O で読み取られるブロックの個数を指定します。オプティマイザは、DB_FILE_MULTIBLOCK_READ_COUNT の値を使用して、全表スキャンと高速全索引スキャンのコストを計算します。値が大きいほど全表スキャンのコストは低くなり、オプティマイザは索引スキャンより全表スキャンを選択します。

HASH_AREA_SIZE

このパラメータは、ハッシュ結合に使用するメモリー量をバイト単位で指定します。CBO は、このパラメータを使用してハッシュ結合操作のコストを計算します。HASH_AREA_SIZE の値が大きいほど、ハッシュ結合のコストが少なくなります。

注意： インスタンスが共有サーバー・オプションで設定されない限り HASH_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定が使用できるようにすることをお薦めします。HASH_AREA_SIZE は、下位互換性のためにのみ残されています。

HASH_JOIN_ENABLED

このパラメータを使用して、オプティマイザで結合方法として選択されたハッシュ結合の使用を有効または無効にすることができます。true に設定すると、オプティマイザはハッシュ結合を可能な結合方法とみなします。コストがネステッド・ループやマージ結合などの他の結合方法より低い場合、CBO はハッシュ結合を選択します。

OPTIMIZER_INDEX_CACHING

このパラメータは、ネステッド・ループとともに索引プローブのコスト計算の管理に使用します。OPTIMIZER_INDEX_CACHING の 0 から 100 の範囲は、ネステッド・ループおよび IN リスト・イテレータの索引キャッシュに関するオプティマイザの仮定を変更するバッファ・キャッシュ内の索引ブロックのキャッシュ率を管理します。この値が 100 となっている場合、索引ブロックの 100% がバッファ・キャッシュに見つかる可能性が推測されます。オプティマイザはそれに応じて索引プローブあるいはネステッド・ループのコストを調整します。実行計画は索引のキャッシュに応じて変更される可能性があります。このパラメータを使用するときは注意してください。

OPTIMIZER_INDEX_COST_ADJ

このパラメータを使用して、索引プローブのコストを調整できます。値の範囲は 1 から 10000 です。デフォルト値は 100 ですが、これは索引が標準のコスト計算モデルに基づいてアクセス・パスとして評価されることを意味します。値 10 は、索引アクセス・パスのコストが標準コストの 1/10 であることを意味します。

OPTIMIZER_MAX_PERMUTATIONS

このパラメータは、結合を含む SQL 文の実行計画を作成するときに CBO が考慮する順列の最大数を制御します。値の範囲は 4 から 80000 です。値 80000 は無制限に対応しています。通常、このパラメータを 1000 より小さい値に設定すると、解析時間は数秒またはそれ以下となります。

OPTIMIZER_MAX_PERMUTATIONS パラメータを使用すると、多数の表を結合する複合 SQL 文の解析時間を減らすことができます。ただし、その値を低くすると、オプティマイザは最適な結合順列を見逃すおそれがあります。

OPTIMIZER_MODE

この初期化パラメータは、インスタンスの起動時のオプティマイザのモードを設定します。可能値は、RULE、CHOOSE、ALL_ROWS、FIRST_ROWS_*n* および FIRST_ROWS です。これらのパラメータ値の詳細は、1-7 ページの「OPTIMIZER_MODE 初期化パラメータ」を参照してください。

注意： コストベースの最適化を使用することをお薦めします。ルールベースの最適化は、将来のリリースでは使用されなくなります。

PARTITION_VIEW_ENABLED

このパラメータは、パーティション・ビュー・プルーニング機能を有効にします。true に設定すると、CBO は、ビューの述語またはフィルタに基づいて、必要なパーティションのみをスキャンします。

QUERY_REWRITE_ENABLED

このパラメータは、マテリアライズド・ビューに連動して動作するクエリー・リライト機能を有効にします。true に設定すると、CBO はマテリアライズド・ビューを使用して、元の問合せを満たすクエリー・リライトを考慮します。このパラメータは、ファンクション・ベース索引の使用も制御します。

SORT_AREA_SIZE

このパラメータは、ソートを実行するためのメモリーの量をバイト単位で指定します。ソート操作が実行され、ソートされるデータ量が SORT_AREA_SIZE の値を超えると、SORT_AREA_SIZE を超過したデータが一時表領域に書き込まれます。CBO は、SORT_AREA_SIZE の値を使用して、ソート / マージ結合などのソート操作のコストを計算します。SORT_AREA_SIZE の値が大きくなるほど、ソート操作のための CBO のコストが低くなります。

注意： インスタンスが共有サーバー・オプションで設定されない限り SORT_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定が使用できるようにすることをお薦めします。SORT_AREA_SIZE は、下位互換性のためにのみ残されています。

STAR_TRANSFORMATION_ENABLED

このパラメータを true に設定すると、CBO はスター・クエリーのためのスター型変換のコストを計算できます。スター型変換では、デカルトのアプローチを使用するのではなく、様々なファクト表の列でビットマップ索引を結合します。

関連項目： 各パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

拡張可能オプティマイザの概要

拡張可能オプティマイザは CBO の一部です。これにより、実行計画を選択するために CBO が使用する、統計、選択性およびコスト評価の 3 つの主要構成要素を、ユーザー定義ファンクションおよびドメイン索引の作成者が制御できるようになります。

拡張可能オプティマイザでは次のことを実行できます。

- コスト関数とデフォルト索引を、ドメイン索引、索引タイプ、パッケージおよびスタンドアロン関数に関連付けること。
- 選択関数とデフォルト選択性を、オブジェクト・タイプ、パッケージ関数およびスタンドアロン関数に関連付けること。
- 統計収集関数をドメイン索引および表の列に関連付けること。
- コスト基準を使用して関数を伴う述語を順序付けすること。
- アクセス・コストに基づいて、表へのユーザー定義のアクセス・パス（ドメイン索引）を選択すること。
- DBMS_STATS パッケージまたは ANALYZE 文を使用して、ユーザー定義統計の収集と削除関数を起動すること。

注意： オプティマイザ統計の収集に、ANALYZE よりも DBMS_STATS パッケージの使用を強くお薦めします。このパッケージでは、パラレル統計の収集、パーティション・オブジェクトについてのグローバル統計の収集、および統計収集の微調整を他の方法で行うことができます。さらに、コストベース・オプティマイザでは、DBMS_STATS により収集された統計のみが最終的に使用されます。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、コストベース・オプティマイザに関連しない次のような統計収集の場合には、DBMS_STATS よりも ANALYZE 文を使用する必要があります。

- VALIDATE 句または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックの情報を収集する場合

- 列、ドメイン索引、索引タイプまたは関数に関連付けられている選択関数、コストおよび選択性に関する情報を含めるために新しいデータ・ディクショナリ・ビューを使用すること。
- 関数述語の評価順序を保持するためにヒントを追加すること。

関連項目： 拡張可能オプティマイザの詳細は、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

ユーザー定義統計

ドメイン索引、表の個別の列およびユーザー定義のデータ型に、統計収集関数を定義できます。

統計を収集するためにドメイン索引が分析されるときには、必ず、関連付けられた統計収集関数が Oracle によって呼び出されます。表の列が分析されるときには、必ず、Oracle によって該当の列に標準統計が収集され、関連付けられた統計収集関数が呼び出されます。データ型に統計収集関数が存在する場合は、分析する表に該当のデータ型を持っている列それぞれに対して統計収集関数が呼び出されます。

ユーザー定義の選択性

SQL 文内の述語の選択性は、特定のアクセス・パスのコストを見積もるために使用されます。また、最適な結合順序の判断にも使用されます。オプティマイザにはユーザー定義の演算子に関する情報が存在しないため、オプティマイザは、ユーザー定義の演算子が含まれている述語についての正確な選択性を計算できません。

ユーザー定義演算子、スタンドアロン関数、パッケージ関数またはタイプ・メソッドが含まれている述語に選択関数を定義できます。オプティマイザは、リレーション <、<=、=、>=、> または LIKE のいずれかに、定数とともに演算子、関数またはメソッドを含む述語が発見されたときには、常に、ユーザー定義の選択関数をコールします。

ユーザー定義コスト

オプティマイザには索引の内部格納構造は不明であるため、ドメイン索引のコストの正確な見積りを計算できません。また、PL/SQL を起動するユーザー定義関数、再帰的 SQL を使用するユーザー定義関数、BFILE をアクセスするユーザー定義関数または CPU を消費するユーザー定義関数のコストを低く見積もる可能性があります。

ドメイン索引、ユーザー定義スタンドアロン関数、パッケージ関数およびタイプ・メソッドのコストを定義できます。これらのユーザー定義コストは、オプティマイザが単に参照するのみのデフォルト・コストの形式にもできます。または、実際にコストを計算するコスト関数として定義することもできます。この場合、オプティマイザはこれらのユーザー定義コスト関数をコールして、計算を実行します。

2

オプティマイザ操作

この章では、第 1 章「オプティマイザの概要」で紹介した概念を展開し、特定のケースについてオプティマイザのアクションをさらに詳しく説明します。コストベース・オプティマイザが式を評価し、特定の操作を実行する方法について説明します。また、CBO が SQL 文を他の文に変換して、同じ目的をさらに効率良く実現する方法についても説明します。

この章には次の項があります。

- オプティマイザによる操作の実行方法
- オプティマイザによる SQL 文の変換方法

オプティマイザによる操作の実行方法

オプティマイザは、可能な場合は必ず式を完全に評価して、ある構文の構造を同等の構造に変換します。構造は次の場合に変換されます。

- Oracle では、結果の式を元の式よりも迅速に評価できます。
- 元の式は、結果の式と構文上同等です。異なる SQL 構造がまったく同じように機能する場合があります。たとえば、Oracle は = ANY（副問合せ）と IN（副問合せ）を単一の構造にマップします。

この項では、次を説明しています。

CBO による IN リスト・イテレータの評価方法

IN リスト・イテレータは、問合せに値とともに IN 句が含まれている場合に使用します。実行計画は、1 つの追加ステップを除き IN のかわりに等価句を持つ文に対する結果と同じです。もう 1 つのステップとして、IN リスト・イテレータでは、IN リストから一意の値を持つ等価句が供給されます。

例 2-1 と例 2-2 の両方の文は等価であり、同じ計画を作成します。

例 2-1 IN リスト・イテレータの最初の文

```
SELECT header_id, line_id, revenue_amount
  FROM so_lines_all
 WHERE header_id IN (1011,1012,1013);

SELECT header_id, line_id, revenue_amount
  FROM so_lines_all
 WHERE header_id = 1011
    OR header_id = 1012
    OR header_id = 1013;

Plan
-------------------------------------------------
SELECT STATEMENT
 INLIST ITERATOR
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
```

例 2-1 の同等の文は IN リスト・イテレータにより異なる一意の値へ :b1 をバインドします。

例 2-2 同等の代替文

```
SELECT header_id, line_id, revenue_amount
  FROM so_lines_all l
 WHERE header_id = :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
  INDEX RANGE SCAN SO_LINES_N1
```

例 2-3 では一意索引を使用します。IN リストには関連するソートがあるので、一意索引の完全なキーを使用しても、依然としてレンジ・スキャンは存在します。

例 2-3 一意索引による IN リスト・イテレータ

```
SELECT header_id, line_id, revenue_amount
  FROM so_lines_all
 WHERE line_id IN (1011,1012,1013);

Plan
-------------------------------------------------
SELECT STATEMENT
 INLIST ITERATOR
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_U1
```

例 2-4 では、IN リスト演算子はネステッド・ループ操作で表を駆動するときに使用できます。

例 2-4 ネステッド・ループによる IN リスト・イテレータ

```
SELECT h.header_id, l.line_id, l.revenue_amount
  FROM so_headers_all h, so_lines_all l
 WHERE l.inventory_item_id = :b1
   AND h.order_number = l.header_id
   AND h.order_type_id IN (1,2,3);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N5
  INLIST ITERATOR
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_U2
```

IN リスト演算子は、最初のステップのコストが大きいために、すべての IN リスト要素で同じステップの繰返しを避ける場合に特に便利です。例 2-5 では、3 つの IN リスト要素がある場合でも、so_lines_all の全体スキャンは 1 回のみ行われます。

例 2-5 コストの高い最初のステップを回避するための IN リスト・イテレータの使用

```
SELECT h.header_id, l.line_id, l.revenue_amount
  FROM so_headers_all h, so_lines_all l
 WHERE l.s7 = :b1
   AND h.order_number = l.header_id
   AND h.order_type_id IN (1,2,3);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS FULL SO_LINES_ALL
  INLIST ITERATOR
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_U2
```

オプティマイザが IN リスト・イテレータを使用する場合

オプティマイザは IN 句に値が指定されると IN リスト・イテレータを使用し、その列に対する選択性の高い索引を検出します。同じ索引を使用する OR 句が複数ある場合は、この方法がより効率的であるため、オプティマイザは CONCATENATION または UNION ALL よりもこの操作を選択します。

IN リスト・イテレータのヒント

この操作のヒントはありません。関連した索引を使用するためにヒントを提供できますが、この操作が必要になる場合があります。例 2-6 では、INDEX ヒントがない場合の問合せ、および実行計画の結果が示されています。

例 2-6 INDEX ヒントなしの IN リスト・イテレータのコール

```
SELECT h.customer_id, l.line_id, l.revenue_amount
  FROM so_lines_all l, so_headers_all h
 WHERE l.s7 = 20
   AND h.original_system_reference = l.attribute5
   AND h.original_system_source_code IN (1013,1014);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS FULL SO_LINES_ALL
```

```
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N5
```

例 2-7 では、INDEX ヒントがある場合の問合せ、および実行計画の結果が示されています。

例 2-7 INDEX ヒントによる IN リスト・イテレータのコール

```
SELECT /*+INDEX(h so_headers_n9 */ h.customer_id, l.line_id, l.revenue_amount
  FROM so_lines_all l, so_headers_all h
 WHERE l.s7 = 20
   AND h.original_system_reference = l.attribute5
   AND h.original_system_source_code IN (1013,1014);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS FULL SO_LINES_ALL
  INLIST ITERATOR
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N9
```

CBO による連結の評価方法

連結は、様々な条件が OR 句で結合されている文に便利です。連結を使用すると、適切な索引を持つ優れた実行計画が作成できます。2-8 と 2-9 の例では、2 つの計画が示されています。それぞれ、適切な索引から表にアクセスが行われ、連結を使用して結合されます。

例 2-8 の計画は重複行を戻さないので、各コンポーネントに旧コンポーネントの否定が追加されます。

例 2-8 CBO による連結の評価方法

```
SELECT l.header_id, l.line_id, l.revenue_amount
  FROM so_lines_all l
 WHERE l.parent_line_id = :b1
    OR l.service_parent_line_id = :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 CONCATENATION
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N20
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N17
```

例 2-8 では、コンポーネントは次のとおりです。

- `l.parent_line_id = :b1`
- `l.service_parent_line_id = :b1` および `l.parent_line_id != :b1`

例 2-9 では、オプティマイザにより、ネストされた OR 文を最適化するために使用される結合方法が示されています。

例 2-9 ネストされた OR 文による問合せの結合

```
SELECT p.header_id, l.line_id, l.revenue_amount
  FROM so_lines_all p , so_lines_all l
 WHERE p.header_id = :b1
   AND (l.parent_line_id = p.line_id
       OR l.service_parent_line_id = p.line_id);

Plan
-------------------------------------------------
SELECT STATEMENT
 CONCATENATION
  NESTED LOOPS
   TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
    INDEX RANGE SCAN SO_LINES_N1
   TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
    INDEX RANGE SCAN SO_LINES_N20
  NESTED LOOPS
   TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
    INDEX RANGE SCAN SO_LINES_N1
   TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
    INDEX RANGE SCAN SO_LINES_N17
```

例 2-10 では、例 2-9 で示されている問合せと同じ実行計画の結果が示されていますが、連結では NO_EXPAND ヒントにより明確に無効とされます。

例 2-10 連結が無効なネストされた OR による問合せ

```
SELECT /*+NO_EXPAND */ p.header_id, l.line_id, l.revenue_amount
  FROM so_lines_all p, so_lines_all l
 WHERE p.header_id = :b1
   AND (l.parent_line_id = p.line_id
       OR l.service_parent_line_id = p.line_id);
```

```
Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
  TABLE ACCESS FULL SO_LINES_ALL
```

1 回の問合せで文を実行しようとすると、不十分な実行計画が作成されます。オプティマイザには追跡する 2 つのパスがあり、問合せを分解しないように指示されているので、いずれかの条件に一致する行があるかどうかを調べるには、第 2 の表のすべての行にアクセスする必要があります。

連結のヒント

この操作にはヒント USE_CONCAT を使用します。

連結を使用しない場合

次の場合に、連結のコストが高くなるので使用しないでください。

- OR 条件が同じ列にあり、IN リスト演算子を使用できる場合。このほうが連結よりも効率的です。
- すべての連結にコストの高いステップが繰り返される場合。

例 2-11 では、この点が示されています。

例 2-11 連結を使用しない場合

次の文を検討してください。

```
SELECT h.customer_id, l.line_id, l.revenue_amount
  FROM so_lines_all l, so_headers_all h
 WHERE l.s7 = 20
   AND h.original_system_reference = l.attribute5
   AND h.original_system_source_code IN (1013,1014);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS FULL SO_LINES_ALL
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N5
```

最初の文は、第 1 段階として so_lines_all の全体スキャンをコールします。オプティマイザは第 2 の表に 1 列索引の使用を選択しますが、ここではオプティマイザに 2 列索引を使用する必要があります。

例 2-12 で示したように、ヒントを使用して連結を強制できますが、最初の全体スキャンは引き続き繰り返されます（望ましくありません）。

例 2-12 連結のヒントを使用する場合

```
SELECT /*+USE_CONCAT*/ h.customer_id, l.line_id, l.revenue_amount
  FROM so_lines_all l, so_headers_all h
 WHERE l.s7 = 20
   AND h.original_system_reference = l.attribute5
   AND h.original_system_source_code IN (1013,1014);

Plan
-------------------------------------------------
SELECT STATEMENT
 CONCATENATION
  NESTED LOOPS
   TABLE ACCESS FULL SO_LINES_ALL
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N9
  NESTED LOOPS
   TABLE ACCESS FULL SO_LINES_ALL
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N9
```

そのかわりに、2 列索引を使用するヒントを与えると、オプティマイザは IN リスト・イテレータで 2 列索引に切り替えます。次のように、最初のスキャンは繰り返されず、より優れた実行計画が作成されます。

例 2-13 索引のヒントを使用する場合

```
SELECT /*+INDEX(h so_headers_n9 */ h.customer_id, l.line_id, l.revenue_amount
  FROM so_lines_all l, so_headers_all h
 WHERE l.s7 = 20
   AND h.original_system_reference = l.attribute5
   AND h.original_system_source_code IN (1013,1014);

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS FULL SO_LINES_ALL
  INLIST ITERATOR
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N9
```

CBO によるリモート操作の評価方法

リモート操作は、データベース・リンクを通して別のデータベースからアクセスされる表があることを示します。例 2-14 では、リモート駆動表が示されています。

例 2-14 CBO によるリモート駆動表での問合せの評価方法

```
SELECT c.customer_name, count(*)
  FROM ra_customers c, so_headers_all@oe h
 WHERE c.customer_id = h.customer_id
   AND h.order_number = :b1
GROUP BY c.customer_name;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT GROUP BY
  NESTED LOOPS
   REMOTE
   TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
    INDEX UNIQUE SCAN RA_CUSTOMERS_U1
```

ライブラリ・キャッシュから取得されたリモート・データベースの問合せ

```
SELECT "ORDER_NUMBER","CUSTOMER_ID"
  FROM "SO_HEADERS_ALL" "H"
 WHERE "ORDER_NUMBER"=:"SYS_B_0";
```

例 2-15 では、ローカル駆動表が示されています。

例 2-15 CBO によるローカル駆動表での問合せの評価方法

```
SELECT c.customer_name, h.order_number
  FROM ra_customers c, so_headers_all@oe h
 WHERE c.customer_id = h.customer_id
   AND c.customer_name LIKE :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 MERGE JOIN
  REMOTE
  SORT JOIN
   TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
    INDEX RANGE SCAN RA_CUSTOMERS_N1
```

ライブラリ・キャッシュから取得されたリモート・データベースの問合せ

```
SELECT "ORDER_NUMBER","CUSTOMER_ID"
  FROM "SO_HEADERS_ALL" "H"
 WHERE "CUSTOMER_ID" IS NOT NULL
ORDER BY "CUSTOMER_ID";
```

いくつかの要因が実行計画に影響します。

- ネットワーク・ラウンドトリップは、物理および論理 I/O に比べ、はるかにコストがかかります。
- オプティマイザは、Oracle データベース以外のリモート・データベース上では統計をとりません。

通常、オプティマイザはローカル表にアクセスする前に、まずリモート表にアクセスすることを選択します。これは、例 2-14 のように、駆動表がリモート表の場合に効率的です。ただし、駆動表がローカル表である場合、最初にローカル表にアクセスせずに、リモート表にアクセスする方法を選択できない場合があります。そのような場合、パフォーマンスの問題を回避するために、適切なヒントを与える必要がある場合があります。

例 2-16 CBO がローカル駆動表およびネステッド・ループのヒントを使用して問合せを評価する方法

```
SELECT /*+USE_NL(c h) */ c.customer_name, h.order_number
  FROM ra_customers c, so_headers_all@oe h
 WHERE c.customer_id = h.customer_id
   AND c.customer_name LIKE :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
   INDEX RANGE SCAN RA_CUSTOMERS_N1
  REMOTE
```

ライブラリ・キャッシュから取得されたリモート・データベースの問合せ

```
SELECT /*+ USE_NL("H") */ "ORDER_NUMBER","CUSTOMER_ID"
  FROM "SO_HEADERS_ALL" "H" WHERE :1="CUSTOMER_ID"
FILTER;
```

オプティマイザで使用された構造では、行を除外するためにフィルタ条件が適用されています。このフィルタは表のアクセス時には適用できませんでした。例 2-17 では、フィルタは使用されません。

例 2-17 フィルタを使用しないリモート表の問合せ

```
SELECT h.order_number
  FROM so_headers_all h
 WHERE h.open_flag = 'Y'
   AND attribute1 IS NOT NULL;

Plan
-------------------------------------------------
SELECT STATEMENT
 TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
  INDEX RANGE SCAN SO_HEADERS_N2
```

表には、アクセス・パスで使用した条件の他に、表を閲覧するときに行をフィルタにかけるための追加条件がある場合があります。attribute1 IS NOT NULL のように、表にアクセスするときに適用される条件は、FILTER として表示されません。例 2-18 および例 2-19 では、このような条件が示されています。

例 2-18 では、フィルタを作成する GROUP BY 条件による問合せが示されています。

例 2-18 GROUP BY 条件により作成されたフィルタでの問合せ

```
SELECT h.order_number, count(*)
  FROM so_headers_all h
 WHERE h.open_flag = 'Y'
   AND attribute1 IS NOT NULL
GROUP BY h.order_number
HAVING COUNT(*) = 1 ß Filter condition;

Plan
-------------------------------------------------
SELECT STATEMENT
 FILTER
  SORT GROUP BY
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N2
```

例 2-19 では、フィルタを作成する副問合せを持つ問合せが示されています。

例 2-19 副問合せにより作成されたフィルタでの問合せ

```
SELECT h.order_number
  FROM so_headers_all h
 WHERE h.open_flag = 'Y'
   AND EXISTS (SELECT null FROM so_lines_all l
               WHERE l.header_id = h.header_id
               AND l.revenue_amount > 10000);
```

```
Plan
-------------------------------------------------
SELECT STATEMENT
 FILTER
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N2
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
```

この例では、外部問合せの条件に合うすべての行について、相関 EXISTS 副問合せが実行されます。条件に合う行が so_lines_all 表に見つかると、so_headers_all からの行が戻されます。

CBO による分散型の文の評価方法

オプティマイザは、リモート・データベースにあるデータにアクセスする SQL 文の実行計画を選択します。その方法は、ローカル・データのみにアクセスする文の実行計画を選択する方法とほとんど同じです。

- SQL 文によりアクセスされるすべての表が同じリモート・データベースに同時に置かれている場合は、SQL 文がそのリモート・データベースに送信されます。送信された文は、リモート Oracle インスタンスによって実行され、ローカル・データベースにはその結果のみが戻されます。
- SQL 文によってアクセスされる表が、異なるデータベースに置かれている場合、その SQL 文は、それぞれが単一のデータベース上の表にアクセスする断片に分解されます。断片はそれぞれ、Oracle によってアクセス先のデータベースに送信されます。送信された断片は、各データベースのリモート Oracle インスタンスによって実行され、その結果がローカル・データベースに戻されます。ローカル・データベースでは、ローカル Oracle インスタンスにより文に必要な追加の処理が実行されます。

分散型の文にコストベースの実行計画を選択する場合、ローカル・データベースの索引の場合と同様に、オプティマイザはリモート・データベースで使用可能な索引を考慮します。オプティマイザは、また、CBO のためのリモート・データベース上の統計も考慮します。さらに、アクセス・コストを見積もるときには、オプティマイザはデータの位置についても考慮します。たとえば、リモート表の全体スキャンの見積りコストが、個別のローカル表の全体スキャンの見積りコストより大きい、というように検討します。

ルールベースの実行計画については、リモート表の索引についてオプティマイザは考慮しません。

> **関連項目：** 分散問合せをチューニングする方法の詳細は、第 6 章「SQL 文の最適化」を参照してください。

CBO によるソート操作の評価方法

ソートを必要とする操作がユーザーにより指定されると、ソート操作が行われます。一般に発生する操作には次のものがあります。

- SORT UNIQUE
- SORT AGGREGATE
- SORT GROUP BY
- SORT JOIN
- SORT ORDER BY

SORT UNIQUE

SORT UNIQUE は、ユーザーにより DISTINCT 句が指定される場合（例 2-20）、または次のステップに一意の値を必要とする操作がある場合（例 2-21）に行われます。

例 2-20 SORT UNIQUE を発生させる DISTINCT 句

```
SELECT DISTINCT last_name, first_name
 FROM per_all_people_f
 WHERE full_name LIKE :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT UNIQUE
  TABLE ACCESS BY INDEX ROWID PER_ALL_PEOPLE_F
   INDEX RANGE SCAN PER_PEOPLE_F_N54
```

例 2-21 SORT UNIQUE を発生させる IN 副問合せ

SORT UNIQUE では、header_id の一意のリストによる外部問合せが行われます。計画は IN 副問合せがネスト解除され、VW_NSO_1 に変換されたことが示されています。

```
SELECT c.customer_name, h.order_number
  FROM ra_customers c, so_headers_all h
 WHERE c.customer_id = h.customer_id
   AND h.header_id in
      (SELECT l.header_id FROM so_lines_all l
       WHERE l.inventory_item_id = :b1
       AND ordered_quantity > 10);
```

```
Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  NESTED LOOPS
   VIEW  VW_NSO_1
    SORT UNIQUE
     TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
      INDEX RANGE SCAN SO_LINES_N5
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX UNIQUE SCAN SO_HEADERS_U1
  TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
   INDEX UNIQUE SCAN RA_CUSTOMERS_U1
```

例 2-22 のように、オプティマイザにより（一意キーの使用により）重複値が渡されないことが保証される場合は、ソートを回避できます。

例 2-22 SORT UNIQUE を発生させない IN 副問合せ

```
UPDATE so_lines_all l
   SET line_status = 'HOLD'
 WHERE l.header_id IN
     ( SELECT h.header_id FROM so_headers_all h
        WHERE h.customer_id = :b1);

Plan
-------------------------------------------------
UPDATE STATEMENT
 UPDATE  SO_LINES_ALL
  NESTED LOOPS
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX RANGE SCAN SO_HEADERS_N1
   INDEX RANGE SCAN SO_LINES_N1
```

SORT AGGREGATE

SORT AGGREGATE は、実際にはソートを行いません。例 2-23 で示すように、この操作は行セット全体にわたって集計を計算する場合に使用されます。

例 2-23 SORT AGGREGATE を発生させる問合せ

```
SELECT SUM(l.revenue_amount)
  FROM so_lines_all l
 WHERE l.header_id = :b1;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT AGGREGATE
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
```

SORT GROUP BY

SORT GROUP BY は、データ内の異なるグループについての集計を計算する場合に使用します。例 2-24 で示すように、このソートは行を異なるグループに分けるために必要です。

例 2-24 SORT GROUP BY を発生させる問合せ

```
SELECT created_by, SUM(l.revenue_amount)
  FROM so_lines_all l
 WHERE header_id > :b1
GROUP BY created_by;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT GROUP BY
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
```

SORT JOIN

例 2-25 で示すように、結合キーで行をソートする必要がある場合、ソート / マージ結合の間に SORT JOIN は行われます。

例 2-25 SORT JOIN を発生させる問合せ

```
SELECT SUM(l.revenue_amount), l2.creation_date
  FROM so_lines_all l, so_lines_All l2
 WHERE l.creation_date < l2.creation_date
   AND l.header_id <> l2.header_id
GROUP BY l2.creation_date, l2.line_id;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT GROUP BY
  MERGE JOIN
   SORT JOIN
    TABLE ACCESS FULL SO_LINES_ALL
   FILTER
    SORT JOIN
     TABLE ACCESS FULL SO_LINES_ALL
```

SORT ORDER BY

例 2-26 で示すように、索引で満たすことのできない ORDER BY が文で指定されている場合に、SORT ORDER BY の操作が必要となります。

例 2-26 SORT ORDER BY を発生させる問合せ

```
SELECT h.order_number
  FROM so_headers_all h
 WHERE h.customer_id = :b1
 ORDER BY h.creation_date DESC;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT ORDER BY
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N1
```

CBO によるビューの評価方法

次のいずれかの場合に、CBO によりビューが作成できます。

- 複雑なビューが分解されていない場合（例 2-27）。
- 一時 / インライン・ビューが使用されている場合（例 2-28 および 2-29）。

例 2-27 ビューを発生させる問合せ

```
SELECT order_id
FROM orders
 WHERE customer_id = :b1
   AND revenue > :b2;

Plan
-------------------------------------------------
SELECT STATEMENT
 VIEW  ORDERS
  FILTER
   SORT GROUP BY
    NESTED LOOPS
     TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
      INDEX RANGE SCAN SO_HEADERS_N1
     TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
      INDEX RANGE SCAN SO_LINES_N1
```

例 2-28 では、IN 副問合せにより選択される値として SORT UNIQUE が必要とされるため、ビューが作成されます。選択される列が一意であり、ソートを要求しない場合、このビューは不要です。

例 2-28 ビューを発生させる IN 副問合せ

```
SELECT c.customer_name, h.order_number
  FROM ra_customers c, so_headers_all h
 WHERE c.customer_id = h.customer_id
   AND h.header_id IN
     (SELECT l.header_id FROM so_lines_all l
      WHERE l.inventory_item_id = :b1
      AND ordered_quantity > 10);
Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  NESTED LOOPS
   VIEW  VW_NSO_1
    SORT UNIQUE
     TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
```

```
      INDEX RANGE SCAN SO_LINES_N5
   TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
    INDEX UNIQUE SCAN SO_HEADERS_U1
  TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
   INDEX UNIQUE SCAN RA_CUSTOMERS_U1
```

例 2-29 の問合せでは、オーダーの分布および収益が各オーダーごとに明細数により検証されます。CBO では、二重のグループ化を行うために一時インライン・ビューが使用されます。

例 2-29 ビューを発生させるインライン問合せ

```
SELECT COUNT(*) "Orders", cnt "Lines", sum(rev) "Revenue"
FROM (SELECT header_id, COUNT(*) cnt, SUM(revenue_amount) rev
        FROM so_lines_all
       GROUP BY header_id)
GROUP BY cnt;

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT GROUP BY
  VIEW
   SORT GROUP BY
    TABLE ACCESS FULL SO_LINES_ALL
```

CBO による定数の評価方法

定数の計算は、文の実行時に毎回行われるのではなく、文を最適化するときに 1 回のみ実行されます。

たとえば、次の条件では、2000 より多い月給についてすべてテストされます。

```
salary > 24000/12
salary > 2000
salary*12 > 24000
```

SQL 文に 1 番目の条件が含まれている場合、オプティマイザはその条件を 2 番目の条件に単純化します。

注意： オプティマイザは、比較演算子を超えて式を単純化することはありません。前述の例において、オプティマイザは 3 番目の式を 2 番目の式に単純化しません。したがって、アプリケーションの開発者は、可能なかぎり、列に関する式を使用する条件ではなく、定数を使用して列を比較する条件を書きます。

CBO による UNION および UNION ALL 演算子の評価方法

この演算子は、OR 句を 1 つの複合文に結合する場合や、最適化と理解を容易にする目的で、複雑な文を単純な SELECT 文を含む複合文にブレークする場合に便利です。

連結と同様に、コストの高い操作の重複を避けるには、UNION ALL を使用しないようにします。

オプティマイザが UNION または UNION ALL を使用する場合

SQL 文に UNION または UNION ALL 句が含まれている場合、オプティマイザは UNION または UNION ALL を使用します。

例 2-30 では、UNION 句のない問合せが示されています。次の問合せでは、新規の顧客またはオープン・オーダーを持つ顧客が検索されます。

例 2-30 UNION 句のない問合せ

```
SELECT c.customer_name, c.creation_date
  FROM ra_customers c
 WHERE c.creation_date > SYSDATE - 30
    OR customer_id IN
     (SELECT customer_id FROM so_headers_all h
        WHERE h.open_flag = 'Y');

Plan
-------------------------------------------------
SELECT STATEMENT
 FILTER
  TABLE ACCESS FULL RA_CUSTOMERS
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N1
```

駆動条件は異なる表から供給されるので、単一文では問合せを効果的に実行できません。

UNION 句で、問合せを 2 つの文にブレークできます。

- 新規顧客
- オープン・オーダーを持つ顧客

これらの 2 つの文は容易に最適化できます。重複を避けるため（両方の基準を満たす顧客もいます）、ソートを使用して重複を排除する UNION を使用します。個々の 2 つのセットを使用する場合は、UNION ALL を使用して、ソートを除外できます。例 2-31 では、UNION を使用して例 2-30 の問合せをしています。

例 2-31 UNION 句を使用した問合せ

```
SELECT c.customer_name, c.creation_date
  FROM ra_customers c
 WHERE c.creation_date > SYSDATE - 30
UNION ALL
SELECT c.customer_name, c.creation_date
  FROM ra_customers c
 WHERE customer_id IN
     (SELECT customer_id FROM so_headers_all h
        WHERE h.open_flag = 'Y');

Plan
-------------------------------------------------
SELECT STATEMENT
 SORT UNIQUE
  UNION-ALL
   TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
    INDEX RANGE SCAN RA_CUSTOMERS_N2
   NESTED LOOPS
    TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
     INDEX RANGE SCAN SO_HEADERS_N2
    TABLE ACCESS BY INDEX ROWID RA_CUSTOMERS
     INDEX UNIQUE SCAN RA_CUSTOMERS_U1
```

UNION および UNION ALL のヒント

この操作のヒントはありません。

CBO による LIKE 演算子の評価方法

オプティマイザは、ワイルド・カード文字のない式を比較する LIKE 比較演算子を使用する条件を、等価演算子を使用する同等の条件に単純化します。

次の例では、オプティマイザは最初の条件を 2 番目の条件に単純化します。

```
last_name LIKE 'SMITH'
```

は次のように変換されます。

```
last_name = 'SMITH'
```

オプティマイザがこれらの式を単純化できるのは、比較が可変長データ型に関連している場合のみです。たとえば、last_name の型が CHAR(10) の場合、オプティマイザは LIKE 操作を同等の操作に変換できません。これは、等価演算子の後には空白埋めの意味が存在するのに対し、LIKE の後には空白埋めの意味が存在しないためです。

CBO による IN 演算子の評価方法

オプティマイザは、IN 比較演算子を使用する条件を、等価比較演算子および OR 論理演算子を使用する同等の条件に拡張します。

次の例では、オプティマイザは最初の条件を 2 番目の条件に拡張します。

```
last_name IN ('SMITH', 'KING', 'JONES')
```

は次のように変換されます。

```
last_name = 'SMITH' OR last_name = 'KING' OR last_name = 'JONES'
```

関連項目： 2-37 ページ「CBO による IN 副問合せのマージ方法」

CBO による ANY または SOME 演算子の評価方法

オプティマイザは、カッコで括られた値のリストが後に続く ANY 比較演算子または SOME 比較演算子を使用する条件を、等価比較演算子および OR 論理演算子を使用する同等の条件に拡張します。

次の例では、オプティマイザは最初の条件を 2 番目の条件に拡張します。

```
salary > ANY (:first_sal, :second_sal)
```

は次のように変換されます。

```
salary > :first_sal OR salary > :second_sal
```

オプティマイザは、副問合せが後に続く ANY 演算子または SOME 演算子を、EXISTS 演算子および相関副問合せを含む条件に変換します。

次の例では、オプティマイザは最初の条件を 2 番目の条件に変換します。

```
x > ANY (SELECT salary
    FROM employees
    WHERE job_id = 'IT_PROG')
```

は次のように変換されます。

```
EXISTS (SELECT salary
    FROM employees
    WHERE job_id = 'IT_PROG'
      AND x > salary)
```

CBO による ALL 演算子の評価方法

オプティマイザは、カッコで括られた値のリストが後に続く ALL 比較演算子を使用する条件を、等価比較演算子および AND 論理演算子を使用する同等の条件に拡張します。

次の例では、オプティマイザは最初の条件を 2 番目の条件に拡張します。

```
salary > ALL (:first_sal, :second_sal)
```

は次のように変換されます。

```
salary > :first_sal AND salary > :second_sal
```

オプティマイザは、副問合せが後に続く ALL 比較演算子を使用する条件を、ANY 比較演算子および NOT 論理演算子を使用する同等の条件に変換します。次の例では、オプティマイザは最初の条件を 2 番目の条件に変換します。

```
x > ALL (SELECT salary
    FROM employees
    WHERE department_id = 50)
```

は次のように変換されます。

```
NOT (x <= ANY (SELECT salary
      FROM employees
      WHERE department_id = 50) )
```

そして、オプティマイザは、さらに 2 番目の問合せを相関副問合せの後に ANY 比較演算子を持つ条件の変換規則を使用して、次に示す問合せに変換します。

```
NOT EXISTS (SELECT salary
      FROM employees
      WHERE department_id = 50
        AND x <= salary)
```

CBO による BETWEEN 演算子の評価方法

オプティマイザは、常に、BETWEEN 比較演算子を使用する条件を、>= および <= 比較演算子を使用する同等の条件に置換します。次の例では、オプティマイザは最初の条件を 2 番目の条件に置換します。

```
salary BETWEEN 2000 AND 3000
```

は次のように変換されます。

```
salary >= 2000 AND salary <= 3000
```

CBO による NOT 演算子の評価方法

オプティマイザは、条件を単純化して NOT 論理演算子を除去します。この単純化では、NOT 論理演算子が除去され、比較演算子がその逆の比較演算子に置換されます。例 2-32 では、オプティマイザは最初の条件を 2 番目の条件に単純化します。

例 2-32 CBO による NOT 演算子の評価方法

```
NOT department_id = (SELECT department_id FROM employees WHERE last_name = 'Taylor')
```

は次のように変換されます。

```
department_id <> (SELECT department_id FROM employees WHERE last_name = 'Taylor')
```

NOT 論理演算子を含む条件は、様々な記述方法で指定できる場合があります。オプティマイザは条件を変換して、結果の条件にさらに NOT が含まれることになっても、NOT で否定された副条件ができるだけ単純になるようにします。例 2-33 では、オプティマイザは第 1 の条件を第 2 の条件に単純化し、さらに第 3 の条件に単純化します。

例 2-33 CBO による NOT 文の単純化方法

```
NOT (salary < 1000 OR commission_pct IS NULL)
```

は次のように変換されます。

```
NOT salary < 1000 AND commission_pct IS NOT NULL
```

そしてさらに次のように変換されます。

```
salary >= 1000 AND commission_pct IS NOT NULL
```

CBO による推移性の評価方法

WHERE 句の 2 つの条件が共通の列に関連するものであるとき、オプティマイザは推移性の原理を使用して第 3 の条件を推論できる場合があります。その場合、オプティマイザは文を最適化するためにその推論条件を使用できます。推論条件によって、元の条件では使用できなかった索引アクセス・パスを使用可能にできます。

> **注意：** 推移性は CBO によってのみ使用されます。

これらの形式の 2 つの条件を含む WHERE 句を検討します。

```
WHERE column1 comp_oper constant
  AND column1 = column2
```

この場合、オプティマイザは次の条件を推論します。

```
column2 comp_oper constant
```

各項目は次のとおりです。

- comp_oper は、=、!=、^=、<、<>、>、<=、または >= の任意の比較演算子です。
- constant は、演算子、SQL ファンクション、リテラル、バインド変数および相関変数が含まれた任意の定数の式です。

例 2-34 では、WHERE 句に 2 つの条件が含まれており、その 2 つの条件はそれぞれ employees.department_id 列を使用しています。

例 2-34 推移性評価を発生させる問合せ

```
SELECT *
  FROM employees, departments
  WHERE employees.department_id = 20
    AND employees.department_id = departments.department_id;
```

推移性を使用して、オプティマイザは次の条件を推論します。

```
departments.department_id = 20
```

索引が departments.department_id 列に存在する場合は、その索引を使用するアクセス・パスがこの条件によって使用可能になります。

オプティマイザは、列とその他の列に関する条件ではなく、列と定数式に関する条件のみを推論します。これらの形式の 2 つの条件を含む WHERE 句を検討します。

```
WHERE column1 comp_oper column3
  AND column1 = column2
```

この場合、column2 comp_oper column3 の条件はオプティマイザでは推論されません。

CBO による共通の副次式の最適化方法

共通の副次式の最適化は、問合せにおける分離性（OR）のブランチから共通の副次式を識別、除去および収集する経験則的最適化です。多くの場合、実行される結合の数が削減されます。この最適化は、初期化パラメータ OPTIMIZER_FEATURES_ENABLE で有効にされます。

問合せが共通の副次式の最適化に対して妥当であるとみなされるのは、WHERE 句の形式が次のような場合です。

- トップレベルが分離している場合（OR 済みのログのリスト）。
- 各分離部分が、単純な述語または連結の場合（AND 済みのログのリスト）。
- 各連結部分が、単純な述語または単純な述語の分離になっている場合。（述語は、AND または OR が含まれていない場合に単純とみなされます。）
- 式が、問合せのすべての分離ブランチに表示される場合。

共通の副次式の最適化例

例 2-35 の問合せは、ロンドンにある部門で働いており、かつ給与が 60000 より多いか、または経理士である従業員の名前を検索します。問合せには、その 2 つの分離性のブランチに共通の副次式が含まれています。

例 2-35 2 つの OR ブランチにおける共通の副次式の最適化

最初の問合せ

```
SELECT employees.last_name
 FROM employees E, departments D
 WHERE (D.department_id = E.department_id AND E.job_id = 'AC_ACCOUNT' AND D.location
= 2400)
  OR
  E.department_id = D.department_id AND E.salary > 60000 AND D.location = 2400);
```

共通の副次式の絞込みによって、この問合せは次に示す問合せに変換され、結合の数が 2 から 1 に減少します。

最適化された問合せ

```
SELECT employees.last_name
FROM employees E, departments D
WHERE (D.department_id = E.department_id AND D.location = 2400)
   AND (E.job_id = 'AC_ACCOUNT' OR E.salary > 60000);
```

例 2-36 の問合せには、3 つの分離性のブランチに共通の副次式が含まれています。

例 2-36 3 つの OR ブランチにおける共通の副次式の最適化

最初の問合せ

```
SELECT SUM (l_extendedprice* (1 - l_discount))
FROM PARTS, LINEITEM
WHERE (p_partkey = l_partkey
        AND p_brand = 'Brand#12'
        AND p_container IN ('SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND l_quantity >= 1 AND l_quantity <= 1 + 10
        AND p_size >= 1 AND p_size <= 5
        AND l_shipmode IN ('AIR', 'REG AIR')
        AND l_shipinstruct = 'DELIVER IN PERSON')
    OR (l_partkey = p_partkey)
        AND p_brand = 'Brand#23'
        AND p_container IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
        AND l_quantity >= 10 AND l_quantity <= 10 + 10
        AND p_size >= 1 AND p_size <= 10 AND p_size BETWEEN 1 AND 10
        AND l_shipmode IN ('AIR', 'REG AIR')
        AND l_shipinstruct = 'DELIVER IN PERSON')
    OR (p_partkey = l_partkey
        AND p_brand = 'Brand#34'
        AND p_container IN ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND l_quantity >= 20 AND l_quantity <= 20 + 10
        AND p_size >= 1 AND p_size <= 15
        AND l_shipmode IN ('AIR', 'REG AIR')
        AND l_shipinstruct = 'DELIVER IN PERSON');
```

この問合せは、共通の副次式の最適化によって次の問合せに変換され、結合の数が 3 から 1 に減少します。

最適化された問合せ

```
SELECT SUM (l_extendedprice* (1 - l_discount))
FROM PARTS, LINEITEM
WHERE (p_partkey = l_partkey  /* these are the four common subexpressions */
    AND p_size >= 1
    AND l_shipmode IN ('AIR', 'REG AIR')
    AND l_shipinstruct = 'DELIVER IN PERSON')
    AND
        ((p_brand = 'Brand#12'
        AND p_container IN  ( 'SM CASE', 'SM BOX', 'SM PACK', 'SM PKG')
        AND l_quantity >= 1 AND l_quantity <= 1 + 10
        AND p_size <= 5)
    OR (p_brand = 'Brand#23'
        AND p_container IN ('MED BAG', 'MED BOX', 'MED PKG', 'MED PACK')
```

```
        AND l_quantity >= 10 AND l_quantity <= 10 + 10
        AND p_size <= 10)
    OR (p_brand = 'Brand#34'
        AND p_container IN ('LG CASE', 'LG BOX', 'LG PACK', 'LG PKG')
        AND l_quantity >= 20 AND l_quantity <= 20 + 10
AND p_size <= 15));
```

CBO による DETERMINISTIC 関数の評価方法

ユーザーが書いたファンクションを実行するかわりに、以前計算した値をオプティマイザが使用できる場合があります。これが安全なのは、限定的な方法で機能するファンクションについてのみです。このファンクションでは、入力引数値の指定の集合に対して、同一の出力戻り値が戻される必要があります。

パッケージ変数またはデータベースの内容、グローバリゼーション・サポート・パラメータなどのセッション・パラメータの内容が異なることによって、このファンクションの結果が変化してはいけません。また、将来このファンクションを再定義する場合にも、その出力戻り値は、入力引数値の指定の集合に対して以前の定義で計算されたものと同じである必要があります。さらに、ファンクションを実行するかわりに事前計算済みの値を再度使用しても、意味のある副作用がもたらされないことが確かである必要があります。

このファンクションの作成者は、CREATE FUNCTION 文を使用してこのファンクションを宣言するとき、または CREATE PACKAGE、CREATE TYPE 文にこのファンクションを宣言するときに、キーワード DETERMINISTIC を使用することによって、このファンクションが前述の制限事項に従って機能するように Oracle Server を規定できます。実際にデータベースまたはパッケージ変数を扱うファンクションに DETERMINISTIC が宣言されていたとしても、Oracle Server はこの宣言の検証を行いません。このキーワードが適切な場合にのみ使用されているかどうかの検証はプログラマの責任で行ってください。

DETERMINISTIC 関数に対するコールが、すでに計算されている値の使用に置き換えられる場合がありますが、これは、同一の問合せ内でこのファンクションが複数回コールされるとき、またはこのファンクションへの関連コールの包含が定義されているファンクション・ベース索引あるいはマテリアライズド・ビューが存在するときに行われます。

関連項目：

- DETERMINISTIC 関数の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。
- CREATE FUNCTION、CREATE INDEX、および CREATE MATERIALIZED VIEW の表記は、『Oracle9*i* SQL リファレンス』を参照してください。
- ファンクション・ベース索引の表記は、『Oracle9*i* データベース概要』を参照してください。
- マテリアライズド・ビューの詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

オプティマイザによる SQL 文の変換方法

SQL は、非常に柔軟性に富んだ問合せ言語であり、多くの場合、同一の目標に到達するために使用できる文が多数存在します。同じ目的を達成する別の文のほうが効率よく実行できる場合、オプティマイザ（問合せトランスフォーマ）は文を別の文に変換する場合があります。

この項では、次の項目について説明します。

- CBO による OR の複合問合せへの変換方法
- CBO による副問合せのネスト解除方法
- CBO によるビューのマージ方法
- CBO による述語のプッシュ方法
- CBO による複合問合せの実行方法

> **関連項目：** 結合、セミ結合またはアンチ結合が含まれている文の最適化の詳細は、1-39 ページの「結合について」を参照してください。

CBO による OR の複合問合せへの変換方法

問合せに、OR 演算子によって組み合された複数の条件を持つ WHERE 句が含まれているときに、その問合せを UNION ALL 集合演算子を使用する同等の複合問合せに変換すると、問合せをより効率よく実行できる場合は、オプティマイザがその問合せを複合問合せに変換します。

- それぞれの条件によって索引アクセス・パスが個別に使用可能になる場合は、オプティマイザは変換を実行します。オプティマイザは、異なる索引を使用して複数回にわたって表にアクセスし、その結果を 1 つにまとめる実行計画を、結果の文に選択します。
- 索引を使用可能にできない条件であるためにその条件に全表スキャンが必要な場合、オプティマイザはその文を変換しません。オプティマイザは文を実行するために全表スキャンを選択します。そして、表内の各行が Oracle によって検査され、条件が満たされているかどうかが判断されます。
- CBO を使用する文では、元の文とその変換結果の文とのコストを見積って比較することにより、変換を実行するかどうかを判別するため、オプティマイザは統計を使用する場合があります。
- CBO は、IN リスト、または同一列上の複数の OR に対して OR 変換は使用しません。かわりに、IN リスト・イテレータ（反復）演算子を使用します。

> **関連項目：** アクセス・パスについてと、索引でアクセス・パスを使用可能にする方法については、8-3 ページの「RBO のアクセス・パス」および 1-36 ページの「CBO によるアクセス・パスの選択方法」を参照してください。

例 2-37 では、OR 問合せの複合問合せへの変換方法が示されています。この例では、WHERE 句に、OR 演算子によって組み合された 2 つの条件が含まれています。

例 2-37 OR 問合せの複合問合せへの変換

最初の問合せ

```
SELECT *
  FROM employees
  WHERE job_id = 'ST_CLERK'
    OR department_id = 50;
```

job_id 列と department_id 列の両方に索引がある場合、オプティマイザはこの問合せを次の同等の問合せに変換することがあります。

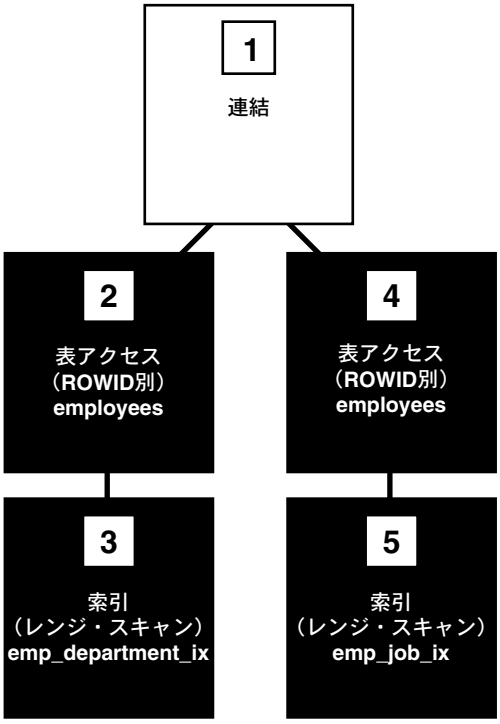
最適化された問合せ

```
SELECT *
  FROM employees
  WHERE job_id = 'ST_CLERK'
UNION ALL
SELECT *
  FROM employees
  WHERE department_id = 50
    AND job_id <> 'ST_CLERK';
```

変換を実行するかどうかを CBO が決定するときに、オプティマイザは、全表スキャンを使用する元の問合せの実行コストを変換問合せの実行コストと比較します。

変換された文の実行計画は、図 2-1 の図のようになります。影付きボックスは物理的にデータを取得するステップを表し、影なしボックスは直前のステップから戻されたデータを処理するステップを表します。

図 2-1 OR が含まれている変換済問合せの実行計画



変換された問合せを実行するには、Oracle では、図 2-1 のステップを次の順序で実行します。

1. ステップ 3 とステップ 5 で、各問合せ要素の条件を使用して、job_id 列と department_id 列の索引をスキャンします。この 2 つのステップによって各問合せ要素を満足させる行の ROWID が取得されます。
2. ステップ 2 とステップ 4 で、ステップ 3 とステップ 5 によって取得された ROWID を使用して、各問合せ要素を満足させる行を探します。
3. ステップ 1 で、ステップ 2 とステップ 4 によって戻された行ソースを 1 つにまとめます。

job_id または department_id のどちらかの列に索引が作成されていない場合は、変換した複合問合せの 1 つの問合せ要素を実行するために複合問合せに全表スキャンが必要になるので、オプティマイザはこの変換を検討しません。索引スキャンの他に全表スキャンも使用する複合問合せが、全表スキャンを使用する元の問合せより高速で実行される可能性はほとんどありません。

例 2-38 では、変換されていない、このような問合せが示されています。この例での問合せでは、last_name 列のみに索引が存在すると仮定します。

例 2-38 変換されていない OR が含まれている問合せ

```
SELECT *
  FROM employees
  WHERE last_name = 'Smith'
    OR salary > commission_pct*100000;
```

この問合せを変換すると、例 2-39 の複合問合せが発生します。

例 2-39 変換対象の OR を含む複合問合せ

```
SELECT *
  FROM employees
  WHERE last_name = 'Smith'
UNION ALL
SELECT *
  FROM employees
  WHERE salary > commission_pct*100000;
```

2 番目の構成要素の問合せ（salary > commission_pct）の WHERE 句の条件では索引は使用可能にはならないので、この複合問合せには全表スキャンが必要です。このため、オプティマイザは変換を実行しませんが、全表スキャンを選択して元の文を実行します。

CBO による副問合せのネスト解除方法

複合文を最適化するには、オプティマイザでは次のいずれかの実行が選択されます。

- 複合文を同等の結合文に変換し、結合文を最適化します。
- 複合文をそのまま最適化します。

オプティマイザは、変換結果の結合文が、複合文とまったく同じ行を戻すことが確実な場合に、複合文を結合文に変換します。この変換によって、Oracle は、オプティマイザの結合の手法の長所を利用して文を実行できるようになります。

例 2-40 では、副問合せをネスト解除するためにオプティマイザの結合を使用する方法が示されています。この例の複合文では、customers 表に所有者が表示されている orders 表からすべての行が選択されます。

例 2-40 CBO による副問合せのネスト解除方法

最初の問合せ

```
SELECT *
  FROM orders
  WHERE customer_id IN
    (SELECT customer_id FROM customers);
```

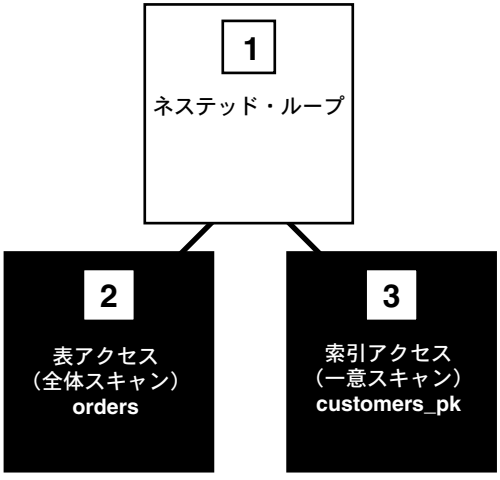
customers 表の customer_id が主キーである場合、またはその列に一意制約が含まれている場合、オプティマイザは、同じデータを戻すことが保証されている次の結合文に複合文を変換できます。

最適化された問合せ

```
SELECT orders.*
  FROM orders, customers
  WHERE orders.customer_id = customers.customer_id;
```

この文を実行するために Oracle は、ネステッド・ループ結合操作を実行します。この文の実行計画は、図 2-2 のようになります。

図 2-2 ネステッド・ループ結合の実行計画



複合文を結合文に変換できない場合、オプティマイザは、親文および副問合せを個別の文とみなしてそれぞれに実行計画を選択します。このとき Oracle は、その副問合せを実行し、親の問合せを実行するために戻された行を使用します。

例 2-41 では、変換されていない、文の一種が示されています。この例における複合文は、customers 表から、平均与信限度より大きい与信限度を持つすべての行を戻します。

例 2-41 変換されていない複合文

```
SELECT *
  FROM customers
  WHERE credit_limit >
    (SELECT AVG(credit_limit) FROM customers);
```

この文の関数を実行できる結合文は存在しないため、オプティマイザはこの文を変換しません。

注意： 副問合せに AVG などの集計関数が存在する複合問合せは、結合文に変換できません。

関連項目： ネステッド・ループ結合の詳細は、1-39 ページの「結合について」を参照してください。

CBO によるビューのマージ方法

ビューにアクセスする文の参照問合せブロックにビューの問合せをマージする場合、オプティマイザは、問合せブロック内のビューの名前をビューの実表の名前に置換し、アクセスする問合せブロックの WHERE にビューの問合せの WHERE 句の条件を追加します。

この最適化は、選択、投影および結合のみが含まれている**選択表示結合**ビューに適用されます。つまり、このビューには集合演算子、集計関数、DISTINCT、GROUP BY、CONNECT BY などの項目は含まれていません。2-35 ページの「マージ可能およびマージ不可能ビュー」を参照してください。

例 2-42 では、CBO によるビューのマージ方法が示されています。この例でのビューは、部門 10 で働くすべての従業員のビューです。

例 2-42 CBO によるビューのマージ方法

```
CREATE VIEW emp_10
  AS SELECT employee_id, last_name, job_id, manager_id, hire_date, salary,
commission_pct, department_id
    FROM employees
    WHERE department_id = 10;
```

このビューは、次の問合せによってアクセスされます。この問合せは、170 より大きい ID を持つ部門 10 で働く従業員を選択します。

```
SELECT employee_id
  FROM emp_10
  WHERE employee_id > 170;
```

オプティマイザはこの問合せを、ビューの実表にアクセスする次の問合せに変換します。

```
SELECT employee_id
  FROM employees
  WHERE department_id = 10
    AND employee_id > 170;
```

department_id 列または employee_id 列に索引が存在する場合は、変換結果の WHERE 句によってその問合せが使用可能になります。

マージ可能およびマージ不可能ビュー オプティマイザがビューを参照問合せブロックにマージできるのは、ビューに 1 つ以上の実表が存在し、提供されたビューに次のいずれも含まれていない場合です。

- 集合演算子（UNION、UNION ALL、INTERSECT、MINUS）
- CONNECT BY 句
- ROWNUM 疑似列
- 選択リスト内の集計関数（AVG、COUNT、MAX、MIN、SUM）

次に示す構成のいずれかがビューに存在する場合、複合ビューのマージが使用可能である場合にのみ、そのビューを参照問合せブロックにマージできます。

- GROUP BY 句
- 選択リスト内の DISTINCT 演算子

複数の実表を持つビューに対するビューのマージは、そのビューが外部結合の右側を構成するビューである場合には実行できません。ただし、外部結合の右側のビューの実表が 1 つのみの場合は、ビュー内の式が NULL に非 NULL 値を戻すことがあっても、オプティマイザは複合ビューのマージを使用できます。

問合せに CURSOR 式が存在する場合、通常ではマージ可能なビューでも、ビューのマージは行われません。次の例のとおりです。

```
CREATE VIEW emp_v AS
       SELECT last_name,employee_id FROM employees;
SELECT CURSOR(select * from sys.dual), last_name, employee_id from emp_v;
```

この問合せでは、emp_v ビューはマージされません。

関連項目： 1-39 ページ「結合について」

複合ビューのマージ ビューの問合せが GROUP BY 句または DISTINCT 演算子を選択リストに持っている場合は、複合ビューのマージが使用可能になっているときのみ、オプティマイザはビューの問合せをアクセス文にマージできます。GROUP BY 句を持つビューの複合マージは、例 2-43 で示しています。

例 2-44 で示すように、複合マージは、副問合せに相互関係がない場合に、IN 副問合せをアクセス文にマージするためにも使用できます。複合マージはコストベースではありませんので、初期化パラメータ OPTIMIZER_FEATURES_ENABLE または MERGE ヒントによって使用可能にする必要があります。このヒントまたはパラメータが設定されていないと、述語をプッシュして、オプティマイザは別のアプローチを使用します。

関連項目：

- 別の方法は、2-37 ページの「CBO による述語のプッシュ方法」を参照してください。
- MERGE および NO_MERGE ヒントの詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

例 2-43 CBO による GROUP BY 句を持つビューのマージ方法

ビュー avg_salary_view には、各部門の給与の平均が表示されます。

```
CREATE VIEW avg_salary_view AS
  SELECT department_id, AVG(salary) AS avg_sal_dept,
    FROM employees
    GROUP BY department_id;
```

複合ビューのマージを使用できる場合は、London に存在する部門の平均給与を検索する次の問合せをオプティマイザが変換します。

```
SELECT departments.location_id, avg_sal_dept
  FROM departments, avg_salary_view
  WHERE departments.department_id = avg_salary_view.department_id
    AND departments.location_id = 2400;
```

その結果は次の問合せになります。

```
SELECT departments.loc, AVG(salary)
  FROM departments, employees
  WHERE departments.department_id = employees.department_id
    AND departments.location_id = 2400
  GROUP BY departments.rowid, departments.location_id;
```

変換された問合せは、ビューの実表にアクセスして、London で働いている従業員の行のみを選択し、選択した行を部門別にグループ化します。

例 2-44 CBO による IN 副問合せのマージ方法

複合マージは、非相関副問合せを使用する IN 句についてもビューの場合と同様に使用することができます。ビュー min_salary_view には、各部門の最低給与が表示されます。

```
CREATE VIEW min_salary_view AS
SELECT department_id, MIN(salary) min_sal
  FROM employees
 GROUP BY department_id;
```

複合マージを使用できる場合は、London に存在する部門の最低給与の従業員を検索する次の問合せをオプティマイザが変換します。

```
SELECT employees.last_name, employees.salary
  FROM employees, departments
 WHERE (employees.department_id, employees.salary) IN
         (select department_id, min_sal from min_salary_view)
   AND employees.department_id = departments.department_id
   AND departments.location_id = 2400;
```

結果は次の問合せになります（アクセスする問合せブロックおよびビューの問合せブロックでそれぞれ employees 表が参照されているため、e1 および e2 は、employees 表を表します）。

```
SELECT e1.last_name, e1.salary
  FROM employees e1, departments, employees e2
 WHERE e1.department_id = departments.department_id
   AND departments.location_id = 2400
   AND e1.department_id = e2.department_id
 GROUP BY e1.rowid, departments.rowid, e1.last_name, e1.salary
HAVING e1.salary = MIN(e2.salary);
```

CBO による述語のプッシュ方法

オプティマイザは、問合せブロックの述語をビューの問合せ内にプッシュすることによって、マージ不可能ビューをアクセスする問合せブロックを変換します。例 2-45 および 2-46 では、このプロセスを説明しています。

例 2-45 では、two_emp_tables ビューは 2 つの従業員表の連結結果です。このビューは、UNION 集合演算子を使用する複合問合せによって定義されます。

例 2-45 CBO による述語のプッシュ方法 - UNION 集合演算子

```
CREATE VIEW two_emp_tables
  (employee_id, last_name, job_id, manager_id, hire_date, salary,
                                  commission_pct, department_id) AS
  SELECT employee_id, last_name, job_id, manager_id, hire_date, salary, commission_
pct, department_id
    FROM emp1
   UNION
  SELECT employee_id, last_name, job_id, manager_id, hire_date,
                         salary, commission_pct, department_id
    FROM emp2;
```

このビューは、次の問合せによってアクセスされます。この問合せは、部門 50 で働くすべての従業員の ID と名前を両方の表から選択します。

```
SELECT employee_id, last_name
  FROM two_emp_tables
 WHERE department_id = 50;
```

このビューは複合問合せとして定義されているため、オプティマイザは、アクセスする問合せブロックにこのビューの問合せをマージできません。問合せをマージするかわりに、オプティマイザは、アクセス文の述語である WHERE 句条件（department_id = 50）をプッシュして、アクセス文をビューの複合問合せに変換します。

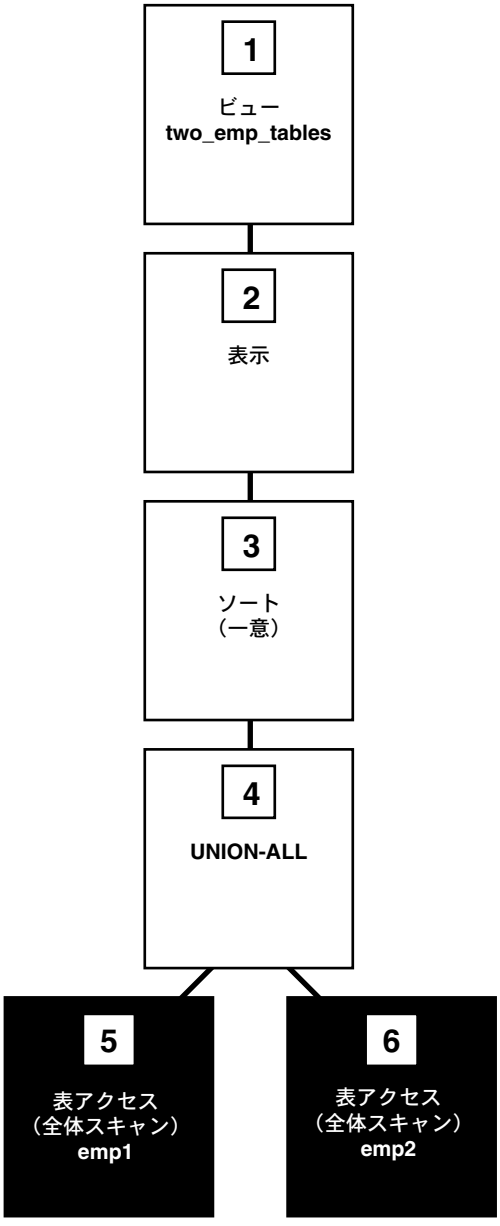
その結果の文は次のようになります。

```
SELECT employee_id, last_name
  FROM ( SELECT employee_id, last_name, job_id, manager_id, hire_date,
                               salary, commission_pct, department_id
           FROM emp1
          WHERE department_id = 50
          UNION
         SELECT employee_id, last_name, job_id, manager_id, hire_date,
                                salary, commission_pct, department_id
           FROM emp2
          WHERE department_id = 50 );
```

department_id 列に索引が存在する場合は、変換結果の WHERE 句によって索引が使用可能になります。

図 2-3 に、変換結果の文の実行計画を示します。

図 2-3 UNION 集合演算子によって定義されたビューのアクセス



この文を実行するには、Oracle では、図 2-3 のステップを次の順序で実行します。

1. ステップ 5 およびステップ 6 で、emp1 表と emp2 表の全表スキャンを実行します。
2. ステップ 4 で、UNION-ALL 操作を実行し、ステップ 5 またはステップ 6 で戻されたすべての行を重複のコピーといっしょに戻します。
3. ステップ 3 で、ステップ 4 の結果をソートして重複した行を除去します。
4. ステップ 2 で、ステップ 3 の結果から必要な行を抽出します。
5. ステップ 1 で、アクセスする問合せにビューの問合せがマージされなかったことを示します。

例 2-46 では、ビュー emp_group_by_deptno には、従業員が存在するすべての部門の部門番号、平均給与、最低給与および最高給与が表示されます。

例 2-46 CBO による述語のプッシュ方法 - GROUP BY 句

```
CREATE VIEW emp_group_by_deptno
  AS SELECT department_id,
        AVG(salary) avg_sal,
        MIN(salary) min_sal,
        MAX(salary) max_sal
       FROM employees
      GROUP BY department_id;
```

次の問合せは、emp_group_by_deptno ビューから部門 50 の平均給与、最低給与および最高給与を選択します。

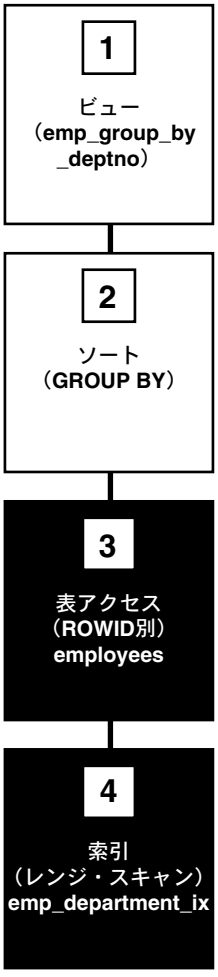
```
SELECT *
  FROM emp_group_by_deptno
 WHERE department_id = 50;
```

オプティマイザは、文の述語（WHERE 句条件）をビューの問合せにプッシュして文を変換します。その結果の文は次のようになります。

```
SELECT department_id,
   AVG(salary) avg_sal,
   MIN(salary) min_sal,
   MAX(salary) max_sal,
  FROM employees
 WHERE department_id = 50
 GROUP BY department_id;
```

department_id 列に索引が存在する場合は、WHERE 句によって索引が使用可能になります。図 2-4 に、変換結果の文の実行計画を示します。この実行計画は、department_id 列の索引を使用します。

図 2-4 GROUP BY 句によって定義されたビューのアクセス



この文を実行するには、Oracle では、図 2-4 のステップを次の順序で実行します。

1. ステップ 4 で、索引 emp_department_ix（employees 表の department_id 列の索引）でレンジ・スキャンを実行し、50 の department_id 値を持っている employees 表のすべての ROWID を取り出します。
2. ステップ 3 で、ステップ 4 によって取り出された ROWID を使用して employees 表をアクセスします。
3. ステップ 2 で、ステップ 3 によって戻された行をソートして平均、最低、および最高の salary 値を計算します。
4. ステップ 1 で、アクセスする問合せにビューの問合せがマージされなかったことを示します。

CBO が集計関数をビューに適用する方法 オプティマイザは、ビューの問合せに関数を適用することによって、集計関数（AVG、COUNT、MAX、MIN、SUM）を含む問合せを変換します。

図 2-47 の問合せは、図 2-46 で定義された emp_group_by_deptno ビューにアクセスします。問合せは、部門の平均給与、最低給与および最高給与の平均を従業員表から導出します。

例 2-47 CBO が集計関数をビューに適用する方法

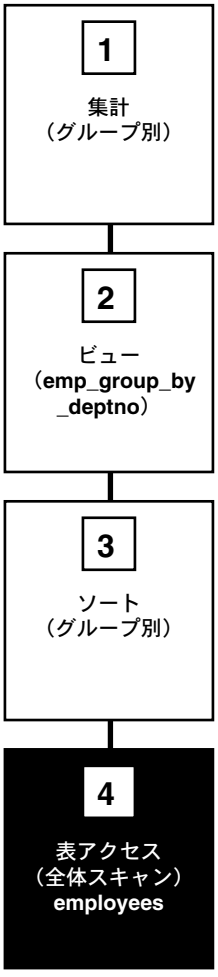
```
SELECT AVG(avg_sal), AVG(min_sal), AVG(max_sal)
  FROM emp_group_by_deptno;
```

オプティマイザは、ビューの問合せの選択リストに AVG 集計関数を適用して、この文を変換します。

```
SELECT AVG(AVG(salary)), AVG(MIN(salary)), AVG(MAX(salary))
  FROM employees
 GROUP BY department_id;
```

図 2-5 に、変換結果の文の実行計画を示します。

図 2-5 GROUP BY 句によって定義されたビューに対する集計関数の適用



この文を実行するには、Oracle では、図 2-5 のステップを次の順序で実行します。

1. ステップ 4 で、employees 表の全体スキャンを実行します。
2. ステップ 3 で、ステップ 4 によって戻された行をその department_id 値を基準とするグループにソートし、各グループの平均、最低、最高の salary 値を計算します。
3. ステップ 2 で、アクセスする問合せにビューの問合せがマージされなかったことを示します。
4. ステップ 1 で、ステップ 2 によって戻された値の平均を計算します。

CBO が外部結合でビューを実行する方法

外部結合の右側に置かれているビューについては、そのビューがアクセスする実表の数に従って次の 2 つの方法のいずれかがオプティマイザによって使用されます。

- ビューの実表が 1 つのみである場合、オプティマイザはビューのマージを使用できます。
- ビューに複数の実表が存在する場合、オプティマイザは、述語結合を ビューにプッシュできます。

CBO が元の文でビューの行にアクセスする方法

オプティマイザは、ビューにアクセスするすべての文を、実表にアクセスする同等の文に変換するわけではありません。たとえば、問合せがビューの ROWNUM 疑似列にアクセスする場合、ビューは問合せにマージされず、問合せの述語はビューにプッシュされません。

実表にアクセスする形式には変換されない文を実行するには、Oracle は、ビューの問合せを発行してその結果の行セットを収集します。そして、表を使用するのと同じように元の文を使用して、収集された行セットにアクセスします。例 2-48 では、このプロセスを説明しています。

例 2-48 CBO がビューの行にアクセスする方法

図 2-46 で定義された emp_group_by_deptno ビューについて検討します。

```
CREATE VIEW emp_group_by_deptno
  AS SELECT department_id,
        AVG(salary) avg_sal,
        MIN(salary) min_sal,
        MAX(salary) max_sal
       FROM employees
      GROUP BY department_id;
```

このビューは、次の問合せによってアクセスされます。次の問合せは、ビュー内に表示されている各部門の平均給与、最低給与、最高給与を、departments 表にある部門の名前と位置に結合します。

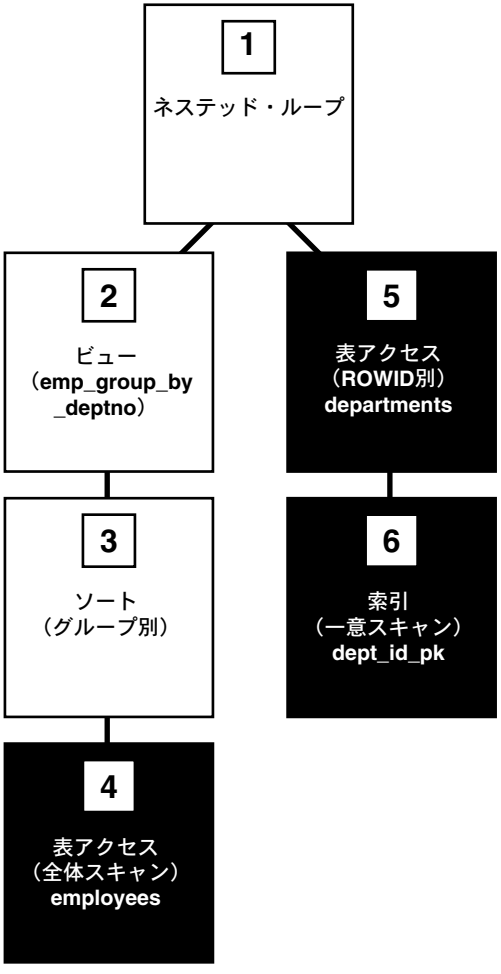
```
SELECT emp_group_by_deptno.department_id, avg_sal, min_sal,
       max_sal, department_name, location_id
  FROM emp_group_by_deptno, departments
 WHERE emp_group_by_deptno.department_id = departments.department_id;
```

実表のみにアクセスする同等の文は存在しないので、オプティマイザはこの文を変換できません。したがって、オプティマイザはビューの問合せを発行する実行計画を選択し、その結果の行セットを、実表から取得した結果の行であるものとみなして使用します。

> **関連項目：** Oracle がネステッド・ループ結合の操作を実行する方法の詳細は、1-39 ページの「結合について」を参照してください。

図 2-6 はこの文の実行計画です。

図 2-6 GROUP BY 句によって定義されたビューと表の結合



この文を実行するには、Oracle では、図 2-6 のステップを次の順序で実行します。

1. ステップ 4 で、employees 表の全体スキャンを実行します。
2. ステップ 3 で、ステップ 4 の結果のソートを実行し、emp_group_by_deptno ビューに対する問合せによって選択された、平均、最低、および最高 salary 値を計算します。
3. ステップ 2 で、ビューに対する前述の 2 つのステップから取得されたデータを使用します。
4. ステップ 2 で戻された各行に対して、ステップ 6 で department_id 値を使用して、dept_id_pk 索引の一意スキャンを実行します。
5. ステップ 5 で、ステップ 6 によって戻された各 ROWID を使用して、department_id 値が一致する departments 表に行を配置します。
6. Oracle は、ステップ 2 で戻された各行をステップ 5 で戻された一致行と組み合せて、その結果を戻します。

CBO による複合問合せの実行方法

複合問合せに実行計画を選択するため、オプティマイザは、複合問合せの構成要素の問合せ、それぞれに実行計画を選択し、その複合問合せに使用されている集合演算子に従って、結果の行ソースを、UNION、INTERSECTION または MINUS 操作と組み合せます。例 2-49、2-50 および 2-51 では、このプロセスを説明しています。

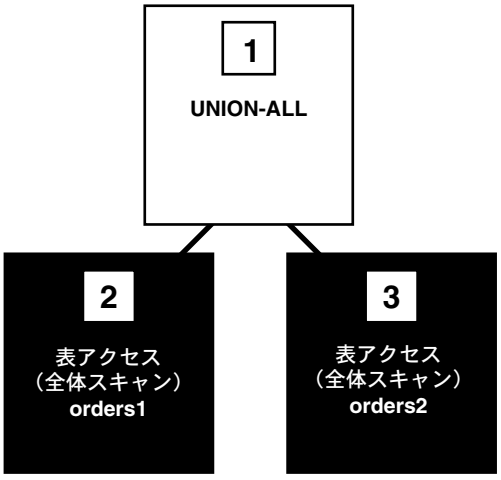
例 2-49 の問合せは、UNION ALL 演算子を使用して orders1 表または orders2 表のいずれかのすべての part 列の出現をすべて選択します。

例 2-49 CBO による UNION ALL での複合問合せの実行方法

```
SELECT part FROM orders1
 UNION ALL
SELECT part FROM orders2;
```

図 2-7 では、この文の実行計画が示されています。

図 2-7 UNION ALL 集合演算子を使用する複合問合せ



この文を実行するには、Oracle では、図 2-7 のステップを次の順序で実行します。

1. ステップ 2 およびステップ 3 で、orders1 表と orders2 表の全表スキャンを実行します。
2. ステップ 1 で、UNION-ALL 操作を実行し、ステップ 2 またはステップ 3 によって戻されたすべての行を重複のコピーといっしょに戻します。

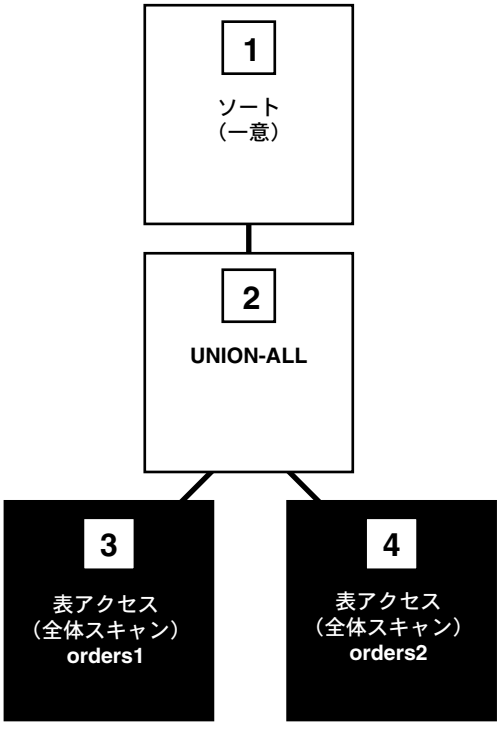
例 2-50 の問合せは、UNION 演算子を使用して orders1 表または orders2 表のいずれかに表示されているすべての part 列を選択します。

例 2-50 CBO による UNION での複合問合せの実行方法

```
SELECT part FROM orders1
 UNION
SELECT part FROM orders2;
```

図 2-8 では、この文の実行計画が示されています。

図 2-8 UNION 集合演算子を使用する複合問合せ



SORT 操作を使用して UNION ALL 操作で戻された重複を取り除いている部分以外は、この実行計画は、図 2-7 に記載されている UNION ALL 操作に対する実行計画と同じものです。

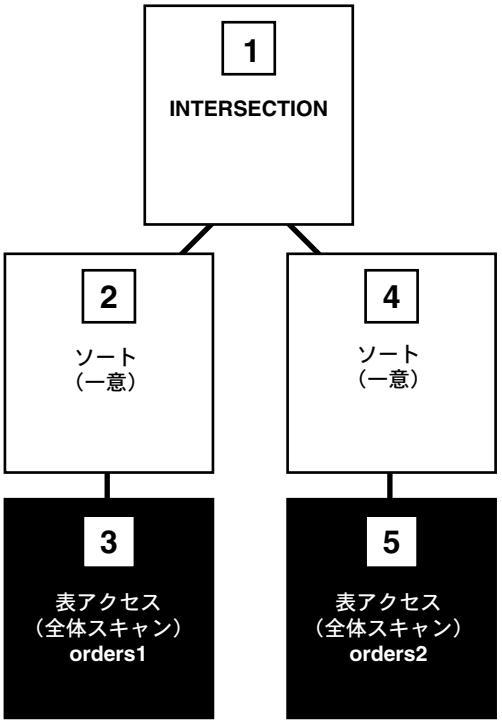
例 2-51 の問合せは、INTERSECT 演算子を使用して orders1 表および orders2 表の両方に表示されている part 列のみを選択します。

例 2-51 CBO による INTERSECT での複合問合せの実行方法

```
SELECT part FROM orders1
INTERSECT
SELECT part FROM orders2;
```

図 2-9 では、この文の実行計画が示されています。

図 2-9 INTERSECT 集合演算子を使用する複合問合せ



この文を実行するには、Oracle では、図 2-9 のステップを次の順序で実行します。

1. ステップ 3 とステップ 5 で、orders1 表と orders2 表の全表スキャンを実行します。
2. ステップ 2 とステップ 4 で、ステップ 3 とステップ 5 の結果をソートして各行ソース内の重複を除去します。
3. ステップ 1 で、ステップ 2 とステップ 4 の両方によって戻された行のみを戻す INTERSECTION 操作を実行します。

3

オプティマイザ統計の収集

この章では、コストベース・オプティマイザにとって統計が重要である理由、および統計の収集方法と使用方法を説明します。

この章には次の項があります。

統計について

データベース管理者は、表、列、索引およびパーティションのデータ配分および記憶特性を検証する統計を生成できます。コストベースの最適化アプローチでは、述語の選択性の計算および各実行計画のコストの見積りにこれらの統計が使用されます。**選択性**は、SQL 文の述語により選択される表に存在する行の割合を表します。オプティマイザでは述語の選択性を使用して、特定のアクセス方法のコストを見積り、最適な結合順序および結合方法が判別されます。

統計はデータ・ディクショナリに格納され、あるデータベースからエクスポートして別のデータベースにインポートできます。たとえば、本番環境をシミュレートするために統計をテスト・システムに転送する場合があります。

注意： この項で説明する統計は CBO 統計であり、V$ ビューから参照できるインスタンス・パフォーマンス統計ではありません。

データ量の変化や列の値の変化により、統計が時間の経過とともに失効するので、定期的にオブジェクトの統計を収集する必要があります。スキーマ・オブジェクトのデータまたは構成を変更するとそれまでの統計が不正確になるようなときには、新しい統計を収集する必要があります。たとえば、大量の行を表にロードした後には、行数についての統計を新たに収集します。表内のデータを更新した後に、行数の統計を新たに収集する必要はありませんが、行の平均長さについては新規の統計が必要になることがあります。

統計の生成には、DBMS_STATS パッケージを使用します。

> **関連項目：** 『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』

生成される統計には次のものがあります。

- 表統計
  - 行数
  - ブロック数
  - 行の平均長さ
- 列統計情報
  - 列内の個別値（NDV）数
  - 列内の NULL 数
  - データ配分（ヒストグラム）

- 索引統計
  - リーフ・ブロック数
  - レベル
  - クラスタ化係数
- システム統計
  - I/O パフォーマンスと使用率
  - CPU パフォーマンスと使用率

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配分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、次の手法を使用して統計を生成します。

- 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

正確な計算を実行する場合、Oracle では、表のスキャン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のみをスキャンおよびソートするのに十分な領域が必要です。索引の場合、計算には時間や領域はそれほど必要ありません。

表中のデータが存在しているデータ・ブロックの数や、索引のルート・ブロックからリーフ・ブロックまでの深さなどの一部の統計は、正確に計算されます。

DBMS_STATS 収集プロシージャの ESTIMATE_PERCENT パラメータを DBMS_STATS.AUTO_SAMPLE_SIZE に設定して、必要な統計精度を実現しながらパフォーマンスを最大にすることをお薦めします。AUTO_SAMPLE_SIZE を使用して、Oracle では、良い統計を作成するために最適なサンプル・サイズが決定されます。たとえば、自動サンプリングで OE スキーマ内のすべての表に関する表統計および列統計情報を収集するには、次のように行います。

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS('OE',DBMS_STATS.AUTO_SAMPLE_SIZE);
```

統計を見積もるために Oracle は、データのサンプルをランダムに選択します。サンプリング・パーセントの指定や、サンプリングが行またはブロックのどちらに基づくかの指定ができます。オラクル社では、DBMS_STATS.AUTO_SAMPLE_SIZE を使用することをお薦めします。どちらに基づくか不明な場合は、行サンプリングを使用してください。

- **行サンプリング**では、ディスク上の行の物理的な位置にかかわらず行が読み込まれます。その結果、ランダムなデータが見積りのために提供されますが、必要以上に多数のデータが読み込まれることにもなります。たとえば、最も不適切な事例は、行サンプリングで各ブロックから 1 行ずつが選択される場合で、これを行うために、表または索引が全体スキャンされる例です。
- **ブロック・サンプリング**では、ブロックのサンプルがランダムに読み込まれ、読み込まれたブロック内のすべての行が見積りに使用されます。これにより、指定されたサンプル・サイズに対する I/O アクティビティの量は削減されますが、行がディスク上に散らばって分布していない場合には、サンプルのランダム性が減少することになります。ブロック・サンプリングは、索引統計には使用できません。

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現行の解析済みの 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・データベース上に発行される分散型の文が、分析したオブジェクトにアクセスする文である場合、その文が次に解析されるときには新しい統計が使用されます。

統計タイプを列またはドメイン索引に対応付ける場合、列またはドメイン索引を分析すると、Oracle では統計タイプの統計コレクション・メソッドがコールされます。

パーティション・スキーマ・オブジェクトの統計の取得

パーティション・スキーマ・オブジェクトには、統計の複数のセットが含まれています。パーティション・スキーマ・オブジェクトでは、次のいずれかを参照する統計を使用できます。

- 総括としての全スキーマ・オブジェクト（グローバル統計）
- 個別のパーティション
- コンポジット・パーティション・オブジェクトの個別のサブパーティション

問合せ述語が問合せ範囲を狭めて単一パーティションにならないかぎり、オプティマイザはグローバル統計を使用します。ほとんどの問合せにはこのような制限は適用されないので、正確なグローバル統計の存在は非常に重要です。感覚的には、パーティション・レベル統計からのグローバル統計の生成は単純に行われているように見えますが、このことが当てはまるのは一部の統計についてのみです。たとえば、各パーティションで検出された個別値数から列の個別値数を算出することは、値が重複している可能性があるために非常に困難です。したがって、`ANALYZE` 文を使用して計算するのではなく、`DBMS_STATS` パッケージを使用して実際にグローバル統計を収集する方法をお薦めします。

注意： オプティマイザ統計の収集に、ANALYZE よりも DBMS_STATS パッケージの使用を強くお薦めします。このパッケージでは、パラレル統計の収集、パーティション・オブジェクトについてのグローバル統計の収集、および統計収集の微調整を他の方法で行うことができます。さらに、コストベース・オプティマイザでは、DBMS_STATS により収集された統計のみが最終的に使用されます。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、コストベース・オプティマイザに関連しない次のような統計収集の場合には、DBMS_STATS よりも ANALYZE 文を使用する必要があります。

- VALIDATE または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックの情報を収集する場合

DBMS_STATS パッケージの使用

PL/SQL パッケージ DBMS_STATS は、コストベースの最適化のための統計を生成および管理できます。このパッケージを使用して、統計を収集、変更、表示、エクスポート、インポートおよび削除できます。このパッケージを使用して、収集された統計を識別または命名することもできます。

DBMS_STATS パッケージで収集できるのは、索引、表およびパーティションの統計、スキーマまたはデータベース内のすべてのスキーマ・オブジェクトの統計です。ただし、クラスタ統計は収集できません。DBMS_STATS を使用して、全クラスタのかわりに個別の表の統計を収集できます。

統計の収集操作は、シリアルまたはパラレルのどちらでも実行できます。索引統計は、パラレルでは収集されません。

パーティション表および索引に対して、DBMS_STATS は、各パーティションの個別の統計を収集できます。また、全表または全索引のグローバル統計も収集できます。コンポジット・パーティションについても同様に、DBMS_STATS はサブパーティション、パーティション、全表および全索引の個別の統計を収集できます。最適化された SQL 文によっては、オプティマイザがパーティション（サブパーティション）統計またはグローバル統計の使用を選択する場合があります。

DBMS_STATS により収集されるのは、コストベースの最適化に必要な統計のみで、それ以外の統計は収集されません。たとえば、DBMS_STATS によって収集された表統計には、行数、データが存在しているブロック数、および行の平均の長さは含まれていますが、連鎖行の数、平均空き領域または未使用のデータ・ブロック数は含まれていません。

関連項目：

- DBMS_STATS パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ユーザー定義統計の詳細は、『Oracle9*i* Data Cartridge Developer's Guide』を参照してください。

DBMS_STATS パッケージでの統計収集

表 3-1 は、DBMS_STATS パッケージにおける統計収集のためのプロシージャです。

表 3-1 DBMS_STATS パッケージの統計収集プロシージャ

プロシージャ	収集対象
GATHER_INDEX_STATS	索引統計
GATHER_TABLE_STATS	表、列および索引の統計
GATHER_SCHEMA_STATS	スキーマ内のすべてのオブジェクトの統計
GATHER_DATABASE_STATS	データベース内のすべてのオブジェクトの統計
GATHER_SYSTEM_STATS	システムの CPU と I/O 統計

関連項目： すべての DBMS_STATS プロシージャの構文と例については、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

システム統計の収集

システム統計を使用すると、オプティマイザでシステムの I/O と CPU のパフォーマンスおよび使用率が考慮できます。各計画候補について、オプティマイザは I/O と CPU のコストの見積りを計算します。I/O と CPU のコスト間の最適な比率を持つ最も効率的な計画を採用するために、システムの特性について把握しておくことが重要です。

システム I/O の特性は多くの要因で決まり、常に一定ではありません。データベース管理者はシステム統計管理ルーチンを使用して、システムのワークロードが最も一般的である場合の統計を一定期間収集できます。たとえば、データベース・アプリケーションは昼間に OLTP トランザクションを処理し、夜間に OLAP レポートを実行できます。管理者は、両方の状態に関する統計を収集し、必要に応じて適切な OLTP または OLAP 統計をアクティブにできます。このため、オプティマイザでは使用可能なシステム・リソース計画に関連するコストが生成できます。

システム統計を生成する場合は、指定期間内のシステム・アクティビティが分析されます。表、索引、列の統計とは異なり、システム統計の更新時には、すでに解析されている SQL

文は無効にされません。新しい SQL 文はすべて、新しい統計を使用して解析されます。システム統計を収集することを強くお薦めします。

DBMS_STATS.GATHER_SYSTEM_STATS ルーチンは、ユーザーが定義した期間内のシステム統計を収集します。DBMS_STATS.SET_SYSTEM_STATS を使用してシステム統計値を明示的に設定することもできます。システム統計を検証するには、DBMS_STATS.GET_SYSTEM_STATS を使用します。

注意： ディクショナリ・システム統計を更新するには、DBA 権限が必要です。

例 3-1 では、昼間に OLTP トランザクションを処理し、夜間にレポートを実行するデータベース・アプリケーションが示されています。まず、システム統計を収集する必要があります。この例の値は、ユーザー定義です。つまり、環境に適した期間および名前を決定する必要があります。

例 3-1 システム統計の生成

昼間に統計を収集します。収集は 720 分後に終了し、統計は mystats 表に格納されます。

```
BEGIN
DBMS_STATS.GATHER_SYSTEM_STATS(
             gathering_mode => 'interval',
             interval => 720,
             stattab => 'mystats',
             statid => 'OLTP');
END;
/
```

夜間に統計を収集します。収集は 720 分後に終了し、統計は mystats 表に格納されます。

```
BEGIN
DBMS_STATS.GATHER_SYSTEM_STATS(
             gathering_mode => 'interval',
             interval => 720,
             stattab => 'mystats',
             statid => 'OLAP');
END;
/
```

必要であれば、収集した統計を切り替えることができます。この処理は、ディクショナリを適切な統計で更新するためのジョブを発行して自動化できます。

次のジョブは、昼間の実行に関する OLTP 統計を、昼間にインポートします。

```
VARIABLE jobno number;
BEGIN
   DBMS_JOB.SUBMIT(:jobno,
      'DBMS_STATS.IMPORT_SYSTEM_STATS(''mystats'',''OLTP'');'
      SYSDATE, 'SYSDATE + 1');
   COMMIT;
END;
/
```

次のジョブは、夜間の実行に関する OLAP 統計を、夜間にインポートします。

```
BEGIN
   DBMS_JOB.SUBMIT(:jobno,
      'DBMS_STATS.IMPORT_SYSTEM_STATS(''mystats'',''OLAP'');'
      SYSDATE + 0.5, 'SYSDATE + 1');
   COMMIT;
END;
/
```

> **関連項目：** システム統計を実装するための DBMS_STATS パッケージのプロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

索引統計の収集

Oracle では、B ツリー索引またはビットマップ索引の作成または再作成中に、一部の統計を自動的に収集できます。この統計収集は、CREATE INDEX または ALTER INDEX ... REBUILD の COMPUTE STATISTICS オプションで実行できます。

COMPUTE STATISTICS オプションで収集される統計は、索引がパーティションの場合と非パーティションの場合とで異なります。

- 非パーティション索引については、索引の作成または再作成中に Oracle が索引統計、表統計および列統計情報を収集します。複合キー索引において列統計情報が参照するのは、キーの先頭列のみです。
- パーティション索引については、索引の作成中または索引のパーティションの再作成中に、Oracle が表統計または列統計情報を収集することはありません。
  - パーティション索引の作成中に、Oracle は各パーティションおよび索引全体についての索引統計を収集します。索引にコンポジット・パーティションが使用されている場合、Oracle はサブパーティションそれぞれについての統計も収集します。
  - 索引のパーティションまたはサブパーティションの再作成中に、Oracle は作成中のパーティションまたはサブパーティションについてのみ索引統計を収集します。

統計の正確さを保つために、Oracle では、COMPUTE STATISTICS オプションによる索引の作成には、常に実表を使用します。これは、索引の作成に使用可能な別の索引が存在している場合でも同様です。

COMPUTE STATISTICS 句を使用しない場合や、データに対する大幅な変更をした場合は、DBMS_STATS.GATHER_INDEX_STATS プロシージャを使用して索引統計を収集します。

> **関連項目：** COMPUTE STATISTICS 句の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ファンクション・ベース索引のための統計の収集 Oracle により式が表わすのと同等の列統計情報を収集できるように、ファンクション・ベース索引の作成後に表を分析する必要があります。オプションとして、METHOD_OPT 引数内の `for all hidden columns size` *`number_of_buckets`* を DBMS_STATS プロシージャに指定してファンクション・ベース索引の式のヒストグラムを収集できます。

> **関連項目：**
>
> - 3-23 ページ「ヒストグラムの使用方法」
> - METHOD_OPT 引数の構文については、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

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統計を収集する前に、DBMS_STATS.EXPORT_SCHEMA_STATS プロシージャを使用して既存の統計を抽出し、保存します。その後で DBMS_STATS.GATHER_SCHEMA_STATS を使用して新しい統計を収集します。GATHER_SCHEMA_STATS プロシージャへの 1 回のコールで、追加パラメータを指定して、この両方を実装することもできます。

> **関連項目：** 3-12 ページ「複数バージョンの統計の保存方法」

重要な 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・サイズを使用する、または次の手順を実行してもう一度統計を収集します。

1. DBMS_STATS.EXPORT_SCHEMA_STATS を使用して、異なる統計表または異なる統計識別子を持つ統計表に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リストアします。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。

新しい統計を使用すると大半の SQL 文でパフォーマンスの向上が見込め、問題のある SQL 文が少ない場合は、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。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。

   関連項目： 第 7 章「プラン・スタビリティの使用方法」

2. DBMS_STATS.IMPORT_SCHEMA_STATS を使用して新しい統計をリストアします。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。

自動統計収集

統計を自動的に収集したり、統計が失効しているか存在していない表のリストを自動的に作成することができます。統計を自動的に収集するには、OPTIONS パラメータと objlist パラメータを使用して DBMS_STATS.GATHER_SCHEMA_STATS プロシージャおよび DBMS_STATS.GATHER_DATABASE_STATS プロシージャを実行します。options パラメータには、表 3-2 にリストした値を使用します。

表 3-2 統計収集用オプション・パラメータ

値	意味
GATHER STALE	失効している統計が存在する表の統計を収集します。
GATHER	すべての表の統計を収集します。（デフォルト）
GATHER EMPTY	統計が存在しない表の統計のみを収集します。
LIST STALE	失効している統計が存在する表のリストを作成します。
LIST EMPTY	統計が存在しない表のリストを作成します。
GATHER AUTO	最新でない特定のスキーマ（または、データベース。DBMS_STATS.GATHER_DATABASE_STATS() を使用。）のオブジェクトのすべての統計を収集します。

objlist パラメータは、LIST STALE オプションと LIST EMPTY オプションの出力パラメータを識別します。objlist パラメータは、型 DBMS_STATS.OBJECTTAB に属します。

監視と自動統計収集のために表を指定する方法 特定の表に自動統計収集を使用する前に、特定のスキーマの表または完全なデータベースの表を監視モードにする必要があります。これは、DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING プロシージャまたは DBMS_STATS.ALTER_DATABASE_TAB_MONITORING プロシージャで行います。別の方法として、MONITORING キーワードを使用して監視属性を有効にできます。このキーワードは、CREATE TABLE および ALTER TABLE 文構文に属します。監視では、最新の統計収集以降の、

表に対する INSERT、UPDATE および DELETE の概数を追跡します。Oracle はこのデータを使用して、統計が失効している表を識別します。次に、GATHER STALE オプションで DBMS_STATS.GATHER_TABLE_STATS を起動する再帰ジョブを設定する（おそらくジョブ・キューを使用して行う）ことで、自動化された統計収集をアプリケーションにとって適切な時間帯に有効にすることができます。

オブジェクトは、総行数の 10% が変更されたときに失効状態とみなされます。GATHER STALE とともに GATHER_TABLE_STATS を発行すると、プロシージャで USER_TAB_MODIFICATIONS ビューがチェックされます。監視される表の変更が 10% を超えた場合、統計が再度収集されます。表の変更に関する情報は、USER_TAB_MODIFICATIONS ビューに示すように、DBMS_STATS.FLUSH_DATABASE_MONITORING_INFO プロシージャで SGA からデータ・ディクショナリへフラッシュできます。

注意： Oracle がこのビューに情報を伝える間、数分間の遅延が出る場合もあります。

監視を無効にするには、DBMS_STATS.ALTER_SCHEMA_TAB_MONITORING または DBMS_STATS.ALTER_DATABASE_TAB_MONITORING プロシージャを使用するか、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 構文と MONITORING キーワードおよび NOMONITORING キーワードの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

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、統計が失効しており、また監視が使用可能になっている表の統計のみが収集されます。

GATHER STALE オプションを使用すると、コストベース・オプ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の統計収集に関連するオーバーヘッドを避けることもできます。GATHER オプションによってオーバーヘッドが多大に発生する場合があります。これは、このオプションによって、GATHER STALE の場合よりも非常に多数の表の統計が収集されるためです。

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、オプ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。

統計が失効している表または統計のない表のリストの作成方法 統計が失効している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、手動で統計を収集する表を特定します。

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 DBMS_STATS パッケージに指定すると、表の複数バージョンの統計を保持できます。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、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。このような表は、DBMS_STATS パッケージの CREATE_STAT_TABLE プロシージャを使用して、最初に作成しておく必要があります。

> **関連項目：** DBMS_STATS プロシージャとパラメータの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ANALYZE 文の使用

ANALYZE 文を使用して、コストベースの最適化のための統計を生成できます。

注意： オプティマイザ統計の収集に、ANALYZE よりも DBMS_STATS パッケージの使用を強くお薦めします。このパッケージでは、パラレル統計の収集、パーティション・オブジェクトのためのグローバル統計の収集、および統計収集の微調整を他の方法で行うことができます。さらに、コストベース・オプティマイザでは、DBMS_STATS により収集された統計のみが最終的に使用されます。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、次のようなコストベース・オプティマイザに関連しない統計収集の場合には、DBMS_STATS よりも ANALYZE 文を使用する必要があります。

- VALIDATE または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックの情報を収集する場合

データ分散の検索

収集された統計は、データが表の中でどのように分散しているかを判断するのに役立ちます。オプティマイザでは、データは均一に分布しているものとみなされます。表における実際のデータ分散は、該当するディクショナリ表を参照することにより分析できます。つまり、表については DBA_TABLES、そして列統計情報は DBA_TAB_COL_STATISTICS です。

属性の偏りを判断するには、ヒストグラムを使用できます。

関連項目：

- 3-2 ページ「統計について」
- 3-20 ページ「ヒストグラムの使用方法」

統計の欠落

統計が存在しない場合、オプティマイザでは表 3-4 および表 3-4 で示されているデフォルト値が使用されます。

表 3-3 統計が欠落しているときの表のデフォルト値

表統計	オプティマイザによって使用されるデフォルト値
カーディナリティ	ブロック数 *（ブロック・サイズ − キャッシュ層）/ 行の平均の長さ
行の平均長さ	100 バイト
ブロック数	1
リモート・カーディナリティ	2000 行
リモートの行の平均長さ	100 バイト

表 3-4 統計が欠落しているときの索引のデフォルト値

索引統計	オプティマイザによって使用されるデフォルト値
レベル	1
リーフ・ブロック	25
リーフ・ブロック / キー	1
データ・ブロック / キー	1
個別キー	100
クラスタ化係数	800（8* ブロック数）

統計の使用方法

この項では、統計の使用方法と参照方法の概要を示します。内容は次のとおりです。

統計の管理

この項では、統計表を説明し、データ・ディクショナリに格納されている統計の情報を表示するビューのリストを示します。

統計表の使用

DBMS_STATS パッケージでは、統計を統計表に格納できます。列、表、索引またはスキーマの統計を統計表に転送したり、後の操作でこれらの統計をデータ・ディクショナリにリストアしたりできます。オプティマイザは、統計表に格納されている統計は使用しません。

統計表によって別の統計のセットを試せます。たとえば、統計の削除、統計の変更、または新しい統計の生成を行う前に、一連の統計のバックアップを作成できます。そして、別の統計を使用して最適化した SQL 文のパフォーマンスと比較して、表に格納されている統計のパフォーマンスが最適であることが判明した場合には、その統計をデータ・ディクショナリにリストアできます。

統計表には複数の個別統計セットを登録できますが、個別統計セットを別々に格納するために、複数の統計表を作成することもできます。

統計の参照

データ・ディクショナリまたは統計表に格納されている統計を表示するには、DBMS_STATS パッケージを使用します。例 3-2 では、統計表を問い合せています。

例 3-2 統計表における統計の表示

```
DECLARE
  num_rows NUMBER;
  num_blocks NUMBER;
  avg_row_len NUMBER;

BEGIN
  -- retrieve the values of table statistics on OE.ORDERS
  -- statistics table name: OE.SAVESTATS    statistics ID: TEST1

  DBMS_STATS.GET_TABLE_STATS('OE','ORDERS',null,
    'SAVESTATS','TEST1',
    num_rows,num_blocks,avg_row_len);

  -- print the values
DBMS_OUTPUT.PUT_LINE('num_rows='||num_rows||',num_blocks='||num_blocks||
                     ',avg_row_len='||avg_row_len);
END;
```

> **注意：** 統計は、DBMS_STATS パッケージを使用することによってのみ参照可能な形式で、統計表に保持されます。

データ・ディクショナリ・ビュー内の統計を表示するには、適切なデータ・ディクショナリ・ビューを問い合せます（USER_、ALL_ または DBA_）。次の表は、DBA_ ビューを示します。

- DBA_TABLES
- DBA_TAB_COL_STATISTICS
- DBA_INDEXES
- DBA_CLUSTERS
- DBA_TAB_PARTITIONS
- DBA_TAB_SUBPARTITIONS
- DBA_IND_PARTITIONS
- DBA_IND_SUBPARTITIONS

- DBA_PART_COL_STATISTICS
- DBA_SUBPART_COL_STATISTICS

関連項目： これらのビューの統計の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

表統計の検証

表統計が使用可能かどうかを検証するには、例 3-3 で示されているような文を使用して、データ・ディクショナリ・ビュー DBA_TABLES を問い合せます。

例 3-3 表統計の検証

```
SQL> SELECT TABLE_NAME, NUM_ROWS, BLOCKS, AVG_ROW_LEN,
       TO_CHAR(LAST_ANALYZED, 'MM/DD/YYYY HH24:MI:SS')
     FROM DBA_TABLES
     WHERE TABLE_NAME IN ('SO_LINES_ALL','SO_HEADERS_ALL','SO_LAST_ALL');
```

これにより、次に示す特有のデータが戻されます。

```
TABLE_NAME                 NUM_ROWS    BLOCKS  AVG_ROW_LEN        LAST_ANALYZED
-------------------------  --------   -------  -----------  -------------------
SO_HEADERS_ALL              1632264    207014          449  07/29/1999 00:59:51
SO_LINES_ALL               10493845   1922196          663  07/29/1999 01:16:09
SO_LAST_ALL ...
```

注意： SO_LAST_ALL 表には統計がないため、すべての列に空白を戻します。

索引統計の検証

索引統計が使用可能かどうかを検証し、アプリケーションで使用する最適な索引を決定するには、例 3-4 で示されているような文を使用して、データ・ディクショナリ・ビュー DBA_INDEXES を問い合せます。

例 3-4 索引統計の検証

```
SQL> SELECT INDEX_NAME "NAME", NUM_ROWS, DISTINCT_KEYS "DISTINCT",
  1  LEAF_BLOCKS, CLUSTERING_FACTOR "CF", BLEVEL "LEVEL",
  2  AVG_LEAF_BLOCKS_PER_KEY "ALFBPKEY"
  3  FROM DBA_INDEXES
  4  WHERE OWNER = 'SH'
  5* ORDER BY INDEX_NAME;
```

出力は、通常次のようになります。

```
NAME                         NUM_ROWS DISTINCT LEAF_BLOCKS      CF   LEVEL   ALFBPKEY
-------------------------- -------- -------- ----------- ------- ------- ----------
CUSTOMERS_PK                   50000    50000         454    4405       2          1
PRODUCTS_PK                    10000    10000          90    1552       1          1
PRODUCTS_PROD_CAT_IX           10000        4          99    4422       1         24
PRODUCTS_PROD_SUBCAT_IX        10000       37         170    6148       2          4
SALES_PROD_BIX                  6287      909        1480    6287       1          1
SALES_PROMO_BIX                 4727      459         570    4727       1          1

6 rows selected.
```

オプティマイザによる索引判断の基準

オプティマイザは、次の基準に従って使用する索引を判断します。

- 索引内の行数（カーディナリティ）。
- 個別キーの数。個別キーによって索引の選択性が定義されます。
- 索引のレベルまたは高さ。これによって、データを検出するためにデータのプローブが検索する深さが指定されます。
- 索引内のリーフ・ブロック数。これは、指定のデータ行を検索するのに必要な I/O 数です。
- クラスタ化係数（CF）。これは、配置された索引ブロックの量をデータ・ブロックとの比較によって表したものです。CF が大きくなるに従って、その索引をオプティマイザが選択する確率が低くなります。
- 各キーの平均リーフ・ブロック（ALFBKEY）。索引内の個別値が表示されるリーフ・ブロックの平均数で、一番近い整数に丸められます。UNIQUE および PRIMARY KEY の各制約を規定する索引の場合、この値は常に 1 です。

正しい索引を選択したかどうかの判断

次の注記は、表、データおよび問合せに適した索引を選択しているかどうかを判断する目安として使用してください。

DISTINCT 2 つの個別キーを持つ索引 ap_invoices_n3 について考えます。索引 ap_invoices_n3 に基づいた場合の選択性は低く、オプティマイザはこの索引を使用しない可能性があります。この索引を使用すると、表のデータの 50% がフェッチされます。この場合、索引 ap_invoices_n3 を使用するよりも全表スキャンの方がコストが低くなります。

索引コストの結合 オプティマイザでは、判断方式がアルファベット順に使用されます。2 つの最終的な索引の選択性、コストおよびカーディナリティが同一であるとオプティマイザが判断した場合、索引の名前を参照し、アルファベットの順位の低いものまたは数字の小さいもので始まる名前を選択します。

列統計情報の検証

列統計情報が使用可能かどうかを検証するには、例 3-5 で示されているような文を使用して、データ・ディクショナリ・ビュー DBA_TAB_COL_STATISTICS を問い合せます。

例 3-5 列統計情報の検証

```
SQL> SELECT COLUMN_NAME, NUM_DISTINCT, NUM_NULLS, NUM_BUCKETS, DENSITY
    FROM DBA_TAB_COL_STATISTICS
    WHERE TABLE_NAME ="PA_EXPENDITURE_ITEMS_ALL"
    ORDER BY COLUMN_NAME;
```

これにより、次のデータが戻されます。

```
COLUMN_NAME                    NUM_DISTINCT  NUM_NULLS NUM_BUCKETS    DENSITY
------------------------------ ------------ ---------- ----------- ----------
BURDEN_COST                            4300      71957           1 .000232558
BURDEN_COST_RATE                        675    7376401           1 .001481481
CONVERTED_FLAG                            1   16793903           1          1
COST_BURDEN_DISTRIBUTED_FLAG              2      15796           1         .5
COST_DISTRIBUTED_FLAG                     2          0           1         .5
COST_IND_COMPILED_SET_ID                 87    6153143           1 .011494253
EXPENDITURE_ID                      1171831          0           1 8.5337E-07
TASK_ID                                8648          0           1 .000115634
TRANSFERRED_FROM_EXP_ITEM_ID        1233787   15568891           1 8.1051E-07
```

列統計情報を検証することは、次の条件にとって重要です。

- 結合条件。
- バインド変数のある列が WHERE 句に含まれている場合。次に例を示します。`column x = :variable_y`

このような場合は、与えられた式についての典型的なカーディナリティの見積りを取得して、格納されている列統計情報を使用できます。

次の項では、例 3-5 での問合せにより戻されたデータが検証されます。

NUM_DISTINCT 列統計情報

NUM_DISTINCT では、列の個別値の数が示されています。

低 例 3-5 では、列 CONVERTED_FLAG の個別値の数は 1 です。この場合、この列の値は 1 つのみです。バインド変数が WHERE 句内の列 CONVERTED_FLAG に存在する場合（たとえば、CONVERTED_FLAG =:variable_y）、カーディナリが小さくなり CONVERTED_FLAG が索引として使用される可能性は低くなります。

列 COST_BURDEN_DISTRIBUTED_FLAG:NUM_DISTINCT = 2。前の例と同様に、この値も低くなります。偏りが大きい場合や多数の NULL がある場合以外、COST_BURDEN_DISTRIBUTED_FLAG は索引に適した候補ではありません。データの偏りが 90% であるとすると、90% のデータに特定の同じ値が存在し、残りの 10% のデータの値が異なるということになります。その 10% にのみ問合せによるアクセスが必要な場合は、オプティマイザがその偏りを認識してその列の索引を利用するために、その列のヒストグラムが必要になります。

高 例 3-5 において、NUM_DISTINCT は、EXPEDITURE_ID 列では 100 万を超えています。列 EXPENDITURE_ID にバインド変数が存在すると、選択性が高くなります（この列の密度が高いことが暗示されます）。言いかえると、EXPENDITURE_ID は索引として使用する適切な候補ということになります。

NUM_NULL 列統計情報

NUM_NULLS は、その列の NULL 値の数を示します。

低 例 3-5 の COST_DISTRIBUTED_FLAG 列のように、単一の列索引にわずかしか NULL が存在しない場合、およびその列が索引として使用された場合、その結果のデータ・セットは大きくなります。

高 たとえば、例 3-5 の CONVERTED_FLAG 列などのように、特定の列に NULL が多く存在する場合、その列を索引として使用すると、その結果のデータ・セットは小さくなります。つまり、COST_DISTRIBUTED_FLAG は、索引により適した列ということになります。

DENSITY 列統計情報

この情報は、その列の値の密度を示します。この密度は、1 を NUM_DISTINCT で除算して算出します。

列統計情報と結合方法

列統計情報は、戻された行数を基準とするものですが、最も効果的な結合方法を判断するのに役立つ有用な情報です。

ヒストグラムの使用方法

コストベース・オプティマイザは、データ値ヒストグラムを使用して、列データの分布の正確な見積りを取得できます。**ヒストグラム**は、列の値を帯域にパーティション化して、1 つの帯域内の値の範囲が等しくなるようにします。ヒストグラムによって、データが偏っている場合の選択性の見積りの精度が改善され、均一でないデータ配分が存在する最適な実行計画が得られます。

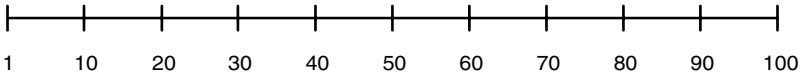
コストベース・オプティマイザの基本的なタスクの 1 つは、問合せに使用される述語の選択性を判断することです。選択性の見積りは、索引を使用するときおよび表を結合する順序を判断するために使用されます。属性ドメイン（表の列）には、均一に分散されていないものもあります。

コストベース・オプティマイザは、指定の属性について高さベースのヒストグラムを使用して、不均等なドメインの分布を説明します。高さベースのヒストグラムでは、列値が帯域に分割され、各帯域にほぼ同数の値が存在するようになっています。したがって、ヒストグラムによって提示される有用な情報が存在するのは、値範囲の終点が位置するところです。

関連項目： 3-22 ページ「ヒストグラムのタイプ」

値が 1 ～ 100 の間に存在し、ヒストグラムが 10 バケットである列 C について検討します。C のデータ配分が均一な場合のヒストグラムは、図 3-1 のようになります。数字は終点の値です。

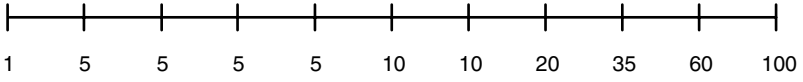
図 3-1 データ配分が均一のヒストグラム



各バケット内の行数は、表内の全行数の 10 分の 1 です。均一に分布しているこの例では、4/10 の行の値が、60 ～ 100 の間にあります。

データ配分が均一でない場合のヒストグラムの例を図 3-2 に示します。

図 3-2 データ配分が非均一のヒストグラム



この場合、ほとんどの行で、この列の値が 5 になっています。60 ～ 100 の間の値を持っている行は、行全体の 1/10 のみです。

ヒストグラムの用途

ヒストグラムの使用はパフォーマンスに影響を与えるため、問合せ計画がヒストグラムによって大幅に改善される場合にのみ使用します。ヒストグラム統計データは永続的なものであるため、データの保存に必要な領域はサンプル・サイズにより異なります。通常は、問合せの WHERE 句で頻繁に使用されている列で、データ配分が非常に偏っている列に対してヒストグラムを作成します。均一に分布しているデータについては、ヒストグラムを使用しなくても、コストベース・オプティマイザが特定の文の実行コストをかなり正確に予測できます。

ヒストグラムは、その他すべてのオプティマイザ統計と同様に静的です。ヒストグラムが有効なのは、指定の列の現行のデータ配分が反映されているときのみです。（分散が一定に保たれているかぎりは、列のデータが変化してもかまいません。）列のデータ配分が頻繁に変化する場合は、その列のヒストグラムを頻繁に再計算する必要があります。

ヒストグラムは、次の特性を持つ列において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用されています。
- 列データが均一に分散しています。
- 列は一意であり、等価述語でのみ使用されます。

ヒストグラムの作成

ヒストグラムは、DBMS_STATS パッケージを使用して生成します。ヒストグラムを生成できるのは、表またはパーティションの列についてです。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、給与額が同じ従業員数が極端に多く、それ以外の給与額の従業員がほとんど存在し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。

どの列にヒストグラムが必要かをデータベースに自動的に決定させるために、DBMS_STATS パッケージを使用することをお薦めします。この決定は、SIZE AUTO を指定して行います。

> **関連項目：** DBMS_STATS パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ヒストグラムのバケット数の選択

列内に発生頻度の高い個別値が比較的少ない場合は、バケット数の値をその個別値数より大きい値に設定してください。ヒストグラムに初期設定されているデフォルトのバケット数は 75 です。この値は、ほとんどのデータ配分に適した詳細レベルです。ただし、バケット数およびデータ分散は、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様々なバケット数で試してみる必要があります。

ヒストグラムのタイプ

ヒストグラムには次の 2 つのタイプがあります。

- 高さベースのヒストグラムについて
- 値ベースのヒストグラムについて

高さベースのヒストグラムについて

高さベースのヒストグラムでは、それぞれの範囲内にほぼ同数の値が配置されますので、範囲の終点がどこになるかはその範囲内に存在する値の数によって異なります。各バケット内の最後（最大）の値のみがバケット（終了点）値として表示されます。

表の問合せ結果によって、次の 4、18、30 および 35 のサンプル値が発生するとします。

高さベースのヒストグラムでは、この 4 つの値はそれぞれ、1 つのバケットの一部をそのサイズに比例した大きさで占有します。その結果の選択性は、次の計算式によって計算されます。

```
S = Height(35) / Height(4 + 18 + 30 + 35)
```

値ベースのヒストグラムについて

個別値の個数が、指定されたヒストグラム・バケットの個数以下であれば、値ベースのヒストグラムが作成されます。値ベースのヒストグラムでは、列内の値はすべて対応するバケットを持ち、バケット数は各値の繰返しの回数を反映します。値ベースのヒストグラムは頻度ヒストグラムとも呼ばれています。

前述の例と同じ 4 つのサンプル値を使用して説明します。値ベースのヒストグラムでは、4 つの個別値それぞれを表すためにバケットが 1 つずつ使用されます。つまり、あるバケットは 4 を表し、もう 1 つのバケットは 18、別のバケットが 30、そしてまた別のバケットが 35 を表します。その結果の選択性は、次の計算式によって計算されます。

```
S = [#rows(35)/(#rows(4) + #rows(18) + #rows(30) + #rows(35))] / #buckets
```

使用する表の特定の列に多数の異なる値が存在することが予測される場合には、高さベースのヒストグラムより値ベースのヒストグラムの方が適しています。これは、高さにおけるデータの偏りが非常に大きいと、その偏りによって選択性の計算がオフセットされ、意味のない選択性の値が指定されるためです。

ヒストグラムの使用方法

例 3-6 では、実行計画を改善するためのヒストグラムの使用および索引のある s6 列の偏りの影響を示します。

例 3-6 実行計画を改善するためのヒストグラムの使用

```
UPDATE so_lines l
SET open_flag=null,
     s6=10,
     s6_date=sysdate,
WHERE l.line_type_code in ('REGULAR','DETAIL','RETURN') AND
      l.open_flag = 'Y' AND NVL(l.shipped_quantity, 0)=0 OR
      NVL(l.shipped_quantity, 0) != 0 AND
      l.shipped_quantity +NVL(l.cancelled_quantity, 0)= l.ordered_quantity)) AND
      l.s6=18
```

この問合せによって、s6 のデータ値の偏った配分が示されます。この場合、データ値は個別の非 NULL 値の 10 と 18 です。多数の行は、s6 = 10（1,589,464）で構成され、少数の行は s6 = 18（13,091）で構成されています。

```
S6:      COUNT(*)
======================
10        1,589,464
18           13,091
NULL         21,889
```

列 s6 の選択性は、s6 = 18 のときに次のようになります。

```
S = 13,091 / (13,091 + 1,589,464) = 0.008
```

ヒストグラムを使用しない場合 : 列 s6 の選択性は、50% で 10 と 18 に均一に分散しているものと想定されます。これは選択的ではないので、s6 を索引に使用するのは理想的な選択とはいえません。

ヒストグラムを使用する場合 : データ配分情報はディクショナリに格納されます。その結果、この情報がオプティマイザによって使用され、データ配分に基づく適正な選択性が計算されます。例 3-6 では、ヒストグラム・データを基準とした選択性は 0.008 です。この比較的高い適切な選択性によって、列 s6 の索引の実行計画での使用がオプティマイザに指示されるようになります。

ヒストグラムの表示

ヒストグラム情報を表示するには、適切なデータ・ディクショナリ・ビューを問い合せます（USER_、ALL_ または DBA_）。次の表は、DBA_ ビューを示します。

- DBA_HISTOGRAMS
- DBA_PART_HISTOGRAMS
- DBA_SUBPART_HISTOGRAMS
- DBA_TAB_COL_STATISTICS

行数 各列のバケット数、つまり行数についての次の DBA_HISTOGRAMS ディクショナリ表を表示します。

- ENDPOINT_NUMBER
- ENDPOINT_VALUE

> **関連項目：** データ・ディクショナリ・ビューの列表記、ヒストグラムの用途およびヒストグラムの制限事項は、『Oracle9*i* データベース・リファレンス』を参照してください。

ヒストグラム統計の検証

ヒストグラム統計が使用可能かどうかを検証するには、例 3-7 で示されているような文を使用して、データ・ディクショナリの DBA_HISTOGRAMS 表を問い合せます。

例 3-7 ヒストグラム統計の検証

```
SQL> SELECT ENDPOINT_NUMBER, ENDPOINT_VALUE
     FROM DBA_HISTOGRAMS
     WHERE TABLE_NAME ="SO_LINES_ALL" AND COLUMN_NAME="S2"
     ORDER BY ENDPOINT_NUMBER;
```

この問合せにより、次に示す特有のデータが戻されます。

```
ENDPOINT_NUMBER   ENDPOINT_VALUE
---------------  ---------------
          1365                4
          1370                5
          2124                8
          2228               18
```

ヒストグラム内で 1 行が 1 つのバケットに対応します。表 3-5 にリストした例 3-7 における、ENDPOINT_NUMBER の値間での差異を考慮します。

表 3-5 ENDPOINT_NUMBER 差異

バケット（値）	ENDPOINT_NUMBER 差異	バケットでの値の数
1（0 から 4）	N/A	N/A
2（4 から 5）	1370 - 1365	5
3（5 から 8）	2124 - 1370	754
4（8 から 18）	2228 - 2124	104

表 3-5 では、バケットに非常に様々な値の数が保持されていることが示されています。データは、偏っており、754 の値は 5 ～ 8 の間ですが、104 のみが 8 ～ 18 の間です。さらに多くのバケットを使用する必要があります。

4

索引およびクラスタ

この章では、索引およびクラスタを使用してパフォーマンスを強化できる、または低下させるデータ・アクセス方法の概要を説明します。

この章には次の項があります。

索引について

この項では、次の項目について説明します。

論理構造のチューニング

コストベースの最適化は、問合せ実行におけるあまり有効ではない索引の使用を避けるために役立ちますが、SQL エンジンは、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します。書込み集中型アプリケーションでは、索引のメンテナンスに CPU と I/O リソースが大量に必要となる場合があります。したがって、必要がなければ索引を作成しないでください。

最適なパフォーマンスを保つために、アプリケーションで使用していない索引を削除してください。使用されていない索引は、`ALTER INDEX MONITORING USAGE` 機能を典型的な負荷を一定期間かけたあとに使用することで検出できます。この監視機能は、索引が使用されたかどうかを記録します。使用されていない索引が検出された場合は、削除してください。代表的なワークロードを選択して監視するよう注意してください。

> **関連項目：**『Oracle9*i* SQL リファレンス』

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。その例が、共有ロックが子表上に取り出されないようにする親表上の外部キー索引です。

また、新しい索引を作成して SQL 文をチューニングするかどうかを決める場合、オプ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を判断するために、`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は、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、オプ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型の文をチューニングする場合は、その文が次に解析されるとき、オプティマイザがこれらの索引について検討します。

また、ある文のチューニング方法が、他の文の実行計画に対するオプ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、ある文によって使用される索

引を作成した場合、オプティマイザは、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、最初にチューニングすることに決めた文をチューニングした後に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します。

注意： Oracle Index Tuning Wizard を使用して、効率の悪い索引を持つ表を検出できます。Oracle Index Tuning Wizard は、Oracle Tuning Pack に用意されている Oracle Enterprise Manager Integrated Applications です。Virtual Index Advisor（SQL Analyze の機能）と Oracle Expert にも、同様の機能があります。

関連項目：『Oracle Tuning Pack によるデータベース・チューニング』

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。次のガイドラインに従って、索引を付けるキーを選択します。

- `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。結合の最適化に関する詳細は、4-21 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。
- 高度な選択性のキーのみに索引を付けます。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、暗黙に索引を作成するか、既存の索引を使用します。

  選択性の低い列への索引付けは、データ配分が偏っているために、1 つまたは 2 つの値がその他の値よりはるかに使用頻度が低い場合に便利です。
- 個別値をほとんど持たないキーまたは式には標準の B ツリー索引は使用しません。通常そのようなキーや式は選択性が劣っているので、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、ビットマップ索引を使用すると効果的です。ただし、並行性の高い OLTP アプリケーションが関与していて、索引が頻繁に変更される場合には向きません。

- 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、表のデータのみでなく、索引のデータも修正する必要があります。また、取消しと再実行も追加的に生成されます。
- 関数や演算子を含む WHERE 句のみに指定されるキーには索引を付けません。索引付きのキーに MIN や MAX 以外の関数や演算子を使用する WHERE 句は、ファンクション・ベース索引を除く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向上が、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薦めします。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： ロックへの外部キーの影響の詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

コンポジット索引の選択

コンポジット索引には複数のキー列が含まれています。コンポジット索引には、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せてコンポジット索引にすると、より高い選択性を得ることができます。

- I/O の削減

  問合せによって選択される列がすべてコンポジット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 文にコンポジット索引の先頭部分を利用する条件が含まれていれば、その文はコンポジット索引に関係するアクセス・パスを使用します。

注意： このことは、索引スキップ・スキャンには現在、該当しなくなっています。1-32 ページ「索引スキップ・スキャン」を参照してください。

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 CREATE INDEX 文を例とします。

```
CREATE INDEX comp_ind
ON table1(x, y, z);
```

- x、xy、xyz の各列の組合せは、索引の先頭部分です。
- yz、y、z の各列の組合せは索引の先頭部分ではありません。

コンポジット索引のキーの選択

コンポジット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、コンポジット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、コンポジット索引を作成してください。
- 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コンポジット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。

コンポジット索引のキーの順序付け

コンポジット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 WHERE 句で使用される頻度が高い場合には、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、これらのキーのみ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 WHERE 句で使用される頻度が同程度で、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、そのキーをコンポジット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のみでは、オプティマイザはその索引を使用するアクセス・パスを選択できません。オプティマイザは、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かぎり、そのようなアクセス・パスを選択できます。索引アクセス・パスを使用するというオプションを CBO で可能にするには、文が索引アクセス・パスを使用可能にする構文になっていることを確認してください。

索引を使用しない文の記述

場合によっては、既存の索引を使用するアクセス・パスを SQL 文で使用しないことも可能です。索引の選択性があまり優れておらず、全表スキャンの方が効率的であることがわかっている場合がそうです。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は、次に示す方法の 1 つを使用して、オプティマイザに全表スキャンを使用するように強制できます。

- NO_INDEX ヒントを使用して特定索引の使用を禁止にし、CBO に最大限の柔軟性を持たせることができます。
- FULL ヒントを使用して、オプティマイザに索引スキャンのかわりに全表スキャンを選択するように強制します。
- INDEX ヒント、INDEX_COMBINE ヒント、または AND_EQUAL ヒントを使用して、オプティマイザに、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します。

> **関連項目：** NO_INDEX、FULL、INDEX、INDEX_COMBINE および AND_EQUAL の各ヒントに関する詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

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レンジ・スキャンは実行しませんが、ネステッド・ループ結合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（各索引エントリの行数が少ない）場合は、パラレル表スキャンよりも順次索引参照を使用することをお薦めします。

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索引作成のパフォーマンスを向上させる場合があります。

注意： 索引の作成または再作成の後で DBMS_STATS をコールしないように、CREATE または REBUILD に、COMPUTE STATISTICS 文を含めてください。Oracle Enterprise Manager の Reorg Wizard を使用して、再作成を必要とする索引を識別できます。Reorg Wizard は索引の再作成にも使用できます。

ただし、既存の索引よりも実表を使用し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用した方が、索引の再作成が速くできます。

ALTER INDEX ... REBUILD 文は、既存の索引を再編成または縮小するため、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（エクステントの割当て用）、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、高速全体スキャン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、再作成の実行中の問合せには、旧索引を利用できるという利点があります。

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の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

索引の縮小

COALESCE オプションを持つ ALTER INDEX 文を使用して索引のリーフ・ブロックを結合できます。このオプションでは、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。

関連項目： この文の構文に関する詳細は、『Oracle9*i* SQL リファレンス』および『Oracle9*i* データベース管理者ガイド』を参照してください。

一意でない索引による一意性の規程

UNIQUE 制約、または PRIMARY KEY 制約の一意性の側面に関して、一意性を規程するために、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、妥当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規程すると、索引の重複を排除できます。主キー列がすでにコンポジット索引の同一キーとして組み込まれている場合、その列に対する一意索引は不要です。Oracle では、制約を使用可能または規程するときに、既存の索引を使用できます。索引を複製しないので、領域を大幅に節約できます。ただし、既存の索引がパーティション化されている場合は、索引のパーティション・キーも UNIQUE キーのサブセットである必要があります。サブセットでなければ、Oracle はさらに一意索引を追加作成し、制約を規程します。

ENABLE NOVALIDATE 制約の使用方法

ENABLE NOVALIDATE 制約は、新しいデータの ENABLE VALIDATE 制約と同じように動作します。制約を ENABLE NOVALIDATE 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する必要があることを意味します。既存のデータはチェックされません。制約を ENABLE NOVALIDATE 状態にすることにより、表をロックしないで制約を使用可能にできます。

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機能がないため、新たに DML、問合せまたは DDL を実行できません。ENABLE NOVALIDATE 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。

表のパラレル一貫読込み問合せによって ENABLE NOVALIDATE 制約を検査済にすることで、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込み機能または書込み機能をブロックしません。さらに、ENABLE NOVALIDATE 制約はパラレルで検査済にすることができます。複数の制約を同時に検査済にし、パラレル問合せを使用して各制約の妥当性チェックを実行できます。

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含め、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、インポートまたは高速ローダーを使用してデータを表 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;
```

この時点で、ユーザーは、表 t で INSERT、UPDATE、DELETE および SELECT を実行できます。

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

これで、制約は ENABLE VALIDATE になりました。

関連項目： 整合性制約の詳細は、『Oracle9*i* データベース概要』を参照してください。

ファンクション・ベース索引の使用方法

ファンクション・ベース索引には、関数（たとえば、UPPER 関数）で変換される列、または式（たとえば、col1 + col2）に含まれている列が含まれています。

変換された列または式でファンクション・ベース索引を定義すると、その関数または式を WHERE 句または ORDER BY 句で使用するとき、その索引を使用してデータを戻すことができます。したがって、ファンクション・ベース索引は、頻繁に実行される SQL 文の WHERE 句または ORDER BY 句の中に変換列（または式の中の列）が含まれているときに有益です。

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ファンクション・ベース索引を使用すると、大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 employees (UPPER(last_name));
```

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 employees
    WHERE UPPER(last_name) = 'MARKSON';
```

問合せでファンクション・ベース索引を使用するためのパラメータの設定

問合せでファンクション・ベース索引を使用する場合は、QUERY_REWRITE_ENABLED パラメータおよび QUERY_REWRITE_INTEGRITY パラメータを設定する必要があります。

QUERY_REWRITE_ENABLED

QUERY_REWRITE_ENABLED セッション・パラメータを TRUE に設定して、ファンクション・ベース索引を問合せで使用できるようにします。QUERY_REWRITE_ENABLED は、次の値に設定できます。

- TRUE: コストベース・リライト
- FALSE: リライトなし
- FORCE: 強制リライト

QUERY_REWRITE_ENABLED が FALSE に設定されている場合、ファンクション・ベース索引内における式の値の取得にファンクション・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ファンクション・ベース索引を使用することはできます。

QUERY_REWRITE_ENABLED が FORCE に設定されている場合、Oracle では常にリライトが使用され、それ以前にコストは評価されません。FORCE は、問合せで常にリライトによる利点がある場合、コンパイル時の減少が重要な場合、およびオプティマイザによりマテリアライズド・ビューの利点が低く評価されている場合に便利です。

QUERY_REWRITE_ENABLED はセッション・レベルのパラメータですが、インスタンス・レベルのパラメータでもあります。

QUERY_REWRITE_INTEGRITY

QUERY_REWRITE_INTEGRITY パラメータの値を設定すると、ファンクション・ベース索引の使用方法が決まります。

- QUERY_REWRITE_INTEGRITY パラメータが ENFORCED（デフォルト）に設定されていると、ファンクション・ベース索引を使用して SQL 式の値のみが導出されます。このパラメータには SQL ファンクションも含まれています。
- QUERY_REWRITE_INTEGRITY が ENFORCED 以外の値に設定されていると、ユーザー定義（SQL でない）関数に基づく場合でも、ファンクション・ベース索引が使用されます。

ファンクション・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。ファンクション・ベース索引を作成して、索引で計算集中型の式を保存できます。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);
```

すると、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;
```

また、ファンクション・ベース索引は、SQL 文で効率的な言語照合を実現する言語ソート索引にも使用できます。

Oracle では、降順の索引は、ファンクション・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。

関連項目：

- ファンクション・ベース索引の使用に関する詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』および『Oracle9*i* データベース管理者ガイド』を参照してください。
- CREATE INDEX 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

索引構成表の使用

索引構成表は、表のデータが、対応付けられた索引に保持されるという点で普通の表とは異なります。新しい行の追加、行の更新、行の削除など、表データを変更すると、索引のみが更新されます。データ行は索引に格納されるため、索引構成表では完全一致、範囲検索またはその両方を含む問合せの表データに対するさらに高速なキー・ベースのアクセスが可能になります。

関連項目：『Oracle9*i* データベース概要』

ビットマップ索引の使用方法

この項では、次の項目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
- 良好なパフォーマンスのビットマップ索引の使用方法
- 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制限事項

関連項目： ビットマップ索引の詳細は、『Oracle9*i* データベース概要』および『Oracle9*i* データ・ウェアハウス・ガイド』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の使用を決定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、記憶域およびメンテナンス）について説明します。

- ビットマップ索引のパフォーマンスの考慮事項
- B ツリー索引とビットマップ索引の比較
- ビットマップ索引のメンテナンスの考慮事項

ビットマップ索引のパフォーマンスの考慮事項

ビットマップ索引は、次のすべての特性を持つ問合せのパフォーマンスを大幅に向上できます。

- カーディナリティが低いか中位である列に関する述語が WHERE 句に複数含まれています。
- カーディナリティが低いか中位である列に関する個々の述語が大量の行を選択しています。
- カーディナリティが低いか中位である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ます。
- 問合せの対象となる表に多数の行が含まれています。

複数のビットマップ索引を使用して、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：** アンチ結合とセミ結合の最適化の詳細は、『Oracle9*i* データベース概要』を参照してください。

B ツリー索引とビットマップ索引の比較

ビットマップ索引は、B ツリー索引が使用する記憶域に比べると、はるかに記憶域を節約できます。B ツリー索引のみ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、大量の領域が必要になるのみでなく、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 のみにアクセスする問合せでは無効です。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判断するときは、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用すると、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・ウェアハウス環境では特に、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。

一意のキー列にはビットマップ索引を作成しないでください。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、圧縮形式で格納されます。

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列では B ツリー索引を維持し、カーディナリティが低い列ではビットマップ索引を作成してください。

ビットマップ索引のメンテナンスの考慮事項

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、INSERT、UPDATE および DELETE 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、挿入された行はソート・バッファに置かれ、その後 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 回のみ更新されます。

注意： 前述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。4-17 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 初期化パラメータおよび CREATE_BITMAP_AREA_SIZE 初期化パラメータは、パラメータ名が示す特定の操作のみに影響します。

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（ただし小さくなる可能性もあります）。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。多数の DML 操作によって索引のサイズが大きくなったり問合せのパフォーマンスが低下した場合、ALTER INDEX ... REBUILD 文を使用して索引を縮小し、効率よいパフォーマンスをリストアできます。

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、単独の一行のみがロックされます。ビットマップ索引では、1 つのエントリにある範囲の ROWID が含まれています。したがって、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、並行性に影響を与えます。並行性は、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。他の型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。それでも、単独の文で多数の行が挿入されたり、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用し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。

良好なパフォーマンスのビットマップ索引の使用方法

この項では、ビットマップ索引のパフォーマンスに関して説明します。

ビットマップ索引でのヒントの使用方法

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用できます。

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、これらすべてを使用する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薦めします。

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せて使おうとします。

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ます。

> **関連項目：** INDEX_COMBINE ヒントに関する詳細は、第 5 章「オプティマイザ・ヒント」を参照してください。

ビットマップ索引のパフォーマンスのヒント

ビットマップ索引を作成する場合に、Oracle ではデータ・ブロックに適合する論理的な最大行数を考慮する必要があります。このため、ビットマップ索引を使用して最適なパフォーマンスとディスク領域の使用状況を達成するには、次のヒントを考慮してください。

- ビットマップ索引をできるかぎり縮小するためには、NULL 値を持てないすべての列に対して NOT NULL 制約を宣言する必要があります。
- 縮小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ます。

> **関連項目：** ビットマップ EXPLAIN PLAN 出力に関する詳細は、第 9 章「EXPLAIN PLAN の使用方法」を参照してください。

ROWID のビットマップへの効率よいマッピング

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ピングを最適化します。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。

MINIMIZE RECORDS_PER_BLOCK を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明確になり、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各ブロック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ピングを使用することはお薦めしません。

> **関連項目：**
>
> - 詳細は、4-11 ページの「ビットマップ索引の使用方法」を参照してください。
> - MINIMIZE および NOMINIMIZE の構文については、『Oracle9*i* SQL リファレンス』を参照してください。

索引構成表でのビットマップ索引の使用方法

索引構成表のビットマップ索引で使用される ROWID は、実表でなくマッピング表にあります。マッピング表は、物理 ROWID（ビットマップ索引コードで必要）に対する論理 ROWID（索引構成表へのアクセスに必要）のマッピングを保守します。各索引構成表には 1 つのマッピング表があり、このマッピング表は索引構成表で作成されたすべてのビットマップ索引により使用されます。

注意： 索引構成表の行を移動しても、索引構成表で作成されたビットマップ索引が使用できなくなることはありません。

関連項目： ビットマップ索引と索引構成表の詳細は、『Oracle9*i* データベース概要』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 に対しては作成できますが、他の型の索引には作成できません。たとえば、次の問合せを実行する、STATE 列と PARTY 列がある表を例とします。

```
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 AGGREGATE
    BITMAP CONVERSION COUNT
      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

次の初期化パラメータは、ビットマップ索引の使用に影響を与えます。

- CREATE_BITMAP_AREA_SIZE は、ビットマップの作成に割り当てられたメモリーに影響を与えます。
- BITMAP_MERGE_AREA_SIZE は、索引レンジ・スキャンからビットマップをマージするために使用するメモリーに影響を与えます。
- SORT_AREA_SIZE は、ビットマップ索引の挿入または更新のときに使用するメモリーに影響を与えます。

> **関連項目：** これらのパラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

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 1 つ以上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が、ビットマップ索引はまったく関係しないこともあります。ただし、ヒントの中で特に指示されないかぎり、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・パスを生成しません。

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、索引内に格納されている 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様々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
```

注意： この文は索引へのアクセスでのみ実行されるので、表へのアクセスは必要ありません。

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、B ツリー索引から取得された ROWID からビットマップを生成するための計画の中には、ROWID からの変換が含まれています。計画内に ORDER BY ソートが表れるのは、列 c3 の条件により B ツリー索引から 2 つ以上の ROWID リストが戻されるためです。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制限事項

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、参照整合性をチェックするためには使用できません。

ビットマップ・ジョイン・インデックスの使用

単一表のビットマップ索引に加えて、ビットマップ・ジョイン・インデックスを作成できます。このビットマップ・ジョイン・インデックスは、複数の表を結合するためのビットマップ索引です。ビットマップ・ジョイン・インデックスは、事前の制限事項の実行により結合される必要のあるデータ量を削減するためにスペースを節約するよい方法です。ビットマップ・ジョイン・インデックスは、表の列の値ごとに、別の表の対応する行の ROWID を格納します。データ・ウェアハウス環境では、結合条件は、ディメンション表の主キー列、およびファクト表の外部キー列との間の等価内部結合です。

ビットマップ・ジョイン・インデックスは、マテリアライズド結合ビューより格納の効率がはるかに良く、事前に結合をマテリアライズする方法の代替手段です。これは、マテリアライズド結合ビューがファクト表の ROWID を圧縮しないためです。

関連項目： ビットマップ・ジョイン・インデックスの例と制限については、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

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。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。通常、ユーザー定義の索引タイプは、Spatial オプションと同様 Oracle のオプションの一部です。たとえば、SpatialIndextype を使用すると、ある条件ボックスにオーバーラップする Spatial データを効率よく取り出せます。

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。

次の情報は、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： 索引の型は CREATE INDEXTYPE 文でも作成できます。

関連項目： SpatialIndextype の詳細は、『Oracle Spatial User's Guide and Reference』を参照してください。

クラスタの使用方法

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

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： クラスタの詳細は、『Oracle9*i* データベース概要』を参照してください。

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する必要もあります。このため、非クラスタ化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 つについてのみ頻繁に全表スキャンを行う場合、それらの表はクラスタ化しません。クラスタ化表の全表スキャンに要する処理時間は、非クラスタ化表の全表スキャンよりも長くなる可能性があります。複数の表が合せて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、ディテール表のみをクラスタに格納します。このようにすれば、同じマスターについて多くの詳細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スキャンのパフォーマンスも低下させずに済みます。別の方法として、索引構成表を使用する方法があります。
- 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用している場合、単一行をアクセスすることによって、非クラスタ化表で同じ行をアクセスする場合よりも多くの読込み処理が必要になる場合があります。
- 各クラスタ・キー値の行数が大きく異なっている場合は表をクラスタ化しないてください。クラスタ化した場合、カーディナリティの低いキー値の場合には領域が無駄になり、カーディナリティの高いキー値の場合には衝突が発生します。衝突が発生するとパフォーマンスが下がります。

アプリケーションのニーズに応じて、クラスタの長所と短所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を上回る場合もあります。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、処理時間を比較してください。

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

ハッシュ・クラスタの使用方法

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、ディスク上にまとめて格納されます。アプリケーションのニーズに応じて、ハッシュ・クラスタの長所と短所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、処理時間を比較してください。

ハッシュ・クラスタを使用する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が WHERE 句に含まれる場合、WHERE 句を持つ SQL 文により頻繁にアクセスされる表を格納するために、ハッシュ・クラスタが使用されます。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判断できる場合、ハッシュ・クラスタに表を格納します。
- アプリケーションが全表スキャンを実行する場合が多く、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。そのような全表スキャンでは、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む必要があります。表を単独で格納することによって、全表スキャンにより読み込まれるブロック数が減少します。
- 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する必要もあります。このため、非クラスタ化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。

前述の 4 項目に基づいてハッシングが適切である限り、他の表と頻繁に結合されるかどうかにかかわらず、単一の表をハッシュ・クラスタに格納することは有益です。

5

オプティマイザ・ヒント

オプティマイザ・ヒントを SQL 文で使用して実行計画を変更できます。ヒントを使用して様々なアプローチを強制する方法を説明します。

この章には次の項があります。

オプティマイザ・ヒントの理解

ヒントを使用することにより、通常オプ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。アプリケーション設計者が、オプティマイザの認知しない、データに関する情報を把握している場合があります。

たとえば、ある問合せに対しては、特定の索引を選択する方がよい場合もあります。この情報に基づいて、アプリケーション設計者がオプティマイザよりも効率的に実行計画を選択することができます。その場合は、ヒントを使用して、オプ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・オプティマイザの目標
- 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。

ヒントは、次の基準に基づいて特定の問合せ実行計画を選択するオプティマイザを割り当てる機構を提供します。

- 結合順序
- 結合方法
- アクセス・パス
- パラレル化

ヒント（RULE ヒント以外）は、コストベース・オプティマイザ（CBO）を起動します。収集した統計が存在しない場合は、デフォルトが使用されます。

関連項目： デフォルト値の詳細は、第 3 章「オプティマイザ統計の収集」を参照してください。

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のみに適用されます。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のみに適用され、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。

オプティマイザへヒントを送るには、SQL 文内でそのヒントをコメントとして囲みます。

> **関連項目：** コメントの詳細は、『Oracle9*i* 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 つのみ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。

> **例外：** APPEND ヒントは常に INSERT キーワードの後に付けられ、PARALLEL ヒントは INSERT キーワードの後に付けられます。

次の構文では、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が示されています。

```
{DELETE|INSERT|SELECT|UPDATE} /*+ hint [text] [hint[text]]... */
```

または

```
{DELETE|INSERT|SELECT|UPDATE} --+ hint [text] [hint[text]]...
```

各項目は次のとおりです。

- DELETE、INSERT、SELECT および UPDATE は、文ブロックを開始するキーワードです。ヒントを含むコメントは、これらのキーワードの直後にかぎり指定できます。
- + により、Oracle はコメントをヒントのリストとして解釈します。プラス記号は、コメント区切り記号の直後に続ける必要があります（空白は許されません）。
- hint は、この項で説明するヒントの 1 つです。コメントに複数のヒントが含まれる場合、各ヒントはその他のヒントと最低 1 つの空白で区切られる必要があります。
- text は、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 はそれらを無視し、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 はそのヒントを無視します。
- Oracle は構文エラーを含むヒントを無視しますが、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します。
- Oracle は矛盾するヒントの組合せを無視しますが、同じコメント内の他のヒントは考慮します。
- 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 を使用する環境では、Oracle はすべての SQL 文でヒントを無視します。これらのヒントをサーバーに渡すことはできますが、サーバーは渡されたヒントを無視します。

オプティマイザは、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。文ブロックにヒント（RULE ヒント以外）が含まれている場合、オプティマイザはコストベースのアプローチを自動的に使用します。

関連項目：

- 各ヒントの構文については、5-6 ページの「オプティマイザ・ヒントの使用方法」を参照してください。
- 索引の型に固有の条件については、5-12 ページの「INDEX」および以降の項を参照してください。

ヒント全セットの指定方法

ヒントを使用するとき、ある状況では、最適な実行計画を確認するためにヒントの全セットの指定が必要な場合があります。たとえば、多数の表が結合された非常に複雑な問合せが存在するときに、指定の表に対して INDEX ヒントのみを指定すると、オプティマイザは、使用される残りのアクセス・パスとともに、対応する結合方法も判断する必要があります。したがって、INDEX ヒントを指定しても、オプティマイザによってそのヒントが使用されるとはかぎりません。これは、オプティマイザの選択した結合方法およびアクセス・パスによっては、要求された索引を使用できないとオプティマイザが判断するためです。

例 5-1 では、ORDERED ヒントにより、使用する正確な結合順序が、別の表で使用される結合方法とともに指定されています。

例 5-1 ヒント全セットの指定方法

```
SELECT /*+ ORDERED INDEX (b, jl_br_balances_n1) USE_NL (j b)
           USE_NL (glcc glf) USE_MERGE (gp gsb) */
 b.application_id ,
 b.set_of_books_id ,
 b.personnel_id,
 p.vendor_id Personnel,
 p.segment1 PersonnelNumber,
 p.vendor_name Name
FROM  jl_br_journals j,
      jl_br_balances b,
      gl_code_combinations glcc,
      fnd_flex_values_vl glf,
      gl_periods gp,
      gl_sets_of_books gsb,
      po_vendors p
WHERE  ...
```

ヒントは次のフォーマットでも指定できます。

```
SELECT --+ ORDERED INDEX (b, jl_br_balances_n1) USE_NL (j b)
           USE_NL (glcc glf) USE_MERGE (gp gsb)
```

ビューに対するヒントの使用方法

デフォルトでは、複合ビューでヒントは使用できません。たとえば、複合ビューに対して選択する問合せでヒントを指定しても、そのヒントは機能しません。これは、ビュー内にヒントがプッシュされないためです。

> **注意：** ビューが単一表の場合、ヒントは伝播されません。

ヒントがベース・ビュー内に存在しないかぎり、ビューに対する問合せからヒントが機能することはありません。

グローバル・ヒントと比較したローカル・ヒント

表ヒント（表を指定するヒント）は、通常、ヒントが呼び出される場所である DELETE 文、SELECT 文または UPDATE 文内の表を参照します。ビューまたは文に参照される副問合せ内の表は参照されません。ビュー内に表示される表のヒントを指定する場合は、ビューに埋め込まれているヒントではなくグローバル・ヒントを使用することをお薦めします。この章で説明するヒントは、表の名前に対する拡張構文を使用してグローバル・ヒントに変換できます。

注意： Oracle Tuning Pack で使用可能な SQL Analyze ツールには、オプティマイザ・ヒントを処理するためのグラフィカル・ユーザー・インタフェースがあります。Hint Wizard（SQL Analyze の機能）では、SQL 文でヒントを容易に追加または変更できます。

関連項目：

- グローバル・ヒントの作成方法の詳細は、5-44 ページの「グローバル・ヒント」を参照してください。
- Oracle SQL Analyze の詳細は、『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

オプティマイザ・ヒントの使用方法

オプティマイザ・ヒントは、次のように分類できます。

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によって、コストベースの最適化アプローチ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。コストベースのアプローチには、最高のスループットまたは最短応答時間という目標も含まれます。

- ALL_ROWS
- FIRST_ROWS(n)
- CHOOSE
- RULE

最適化アプローチと目標を指定するヒントが SQL 文に含まれている場合、オプティマイザは、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用します。

> **注意：** オプ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）文ブロックを最適化するために、コストベースのアプローチを選択します。

all_rows_hint::=

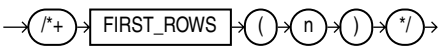
たとえば、オプティマイザは、次の文を最適化するために、最高のスループットを目標としたコストベースのアプローチを使用します。

```
SELECT /*+ ALL_ROWS */ employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;
```

FIRST_ROWS(*n*)

ヒント FIRST_ROWS(*n*)（*n* は正の整数）または FIRST_ROWS は、応答が高速になるように個々の SQL 文を最適化することを指示します。FIRST_ROWS(n) を使用すると、Oracle に対して、最初の *n* 行を最も効率よく戻す計画を選択する指示が出されるため、より高い精度が得られます。FIRST_ROWS ヒントは、最初の 1 行を戻す計画を最適化するためのものであり、下位互換性とプラン・スタビリティのために保持されています。

first_rows_hint::=



たとえば、オプティマイザは、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用します。

```
SELECT /*+ FIRST_ROWS(10) */ employee_id, last_name, salary, job_id
FROM employees
WHERE department_id = 20;
```

この例では、各部門に多数の社員がいます。ユーザーは、部門 20 の最初の 10 人の社員を可能なかぎり素早く表示するとします。

オプティマイザは、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が含まれている 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
- 集計関数
- DISTINCT 演算子
- 順序付けする列に索引がない場合の ORDER BY 句

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取り出す必要があるため、これらの文は最短の応答時間を目標に最適化することができません。このような文にこのヒントを指定しても、オプ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、オプティマイザは、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域など）を使用して、欠けている統計を見積ってから、実行計画を選択します。

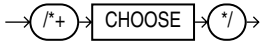
これらの見積りは DBMS_STATS パッケージによって生成された見積りほど正確ではありません。したがって、統計の収集には DBMS_STATS パッケージを使用します。ALL_ROWS ヒントまたは FIRST_ROWS ヒントとともに、アクセス・パスまたは結合操作を指定した場合、オプティマイザはヒントによって指定されたアクセス・パスと結合操作を優先します。

関連項目： FIRST_ROWS(*n*) および FIRST_ROWS の違いの説明は、1-9 ページの「CBO を使用した SQL 文の最適化による応答の高速化」を参照してください。

CHOOSE

CHOOSE ヒントによって、オプティマイザは SQL 文に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どちらかを選択します。オプティマイザによる選択の基準は、文がアクセスする表に対する統計の存在です。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、オプティマイザは、コストベースのアプローチを使用し、最高のスループットを目標にして最適化します。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、オプティマイザは、ルールベースのアプローチを使用します。

choose_hint::=



次に例を示します。

```
SELECT /*+ CHOOSE */ employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;
```

RULE

rule_hint::=

次に例を示します。

```
SELECT /*+ RULE */
employee_id, last_name, salary, job_id
FROM employees
WHERE employee_id = 7566;
```

注意： コストベースの最適化の使用を強くお薦めします。ルールベースの最適化は、将来のリリースでは、使用されなくなります。

アクセス・パスに関するヒント

この項では、表のアクセス・パス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
- INDEX
- INDEX_ASC
- INDEX_COMBINE
- INDEX_JOIN
- INDEX_DESC
- INDEX_FFS
- NO_INDEX
- AND_EQUAL

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 文の構文構造体に基づいて使用できる場合のみ、オプティマイザはそのアクセス・パスを選択します。ヒントを使用できないアクセス・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てください。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。

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。

関連項目： SAMPLE オプションの詳細は、『Oracle9*i* SQL リファレンス』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スキャンを選択します。

full_hint::=

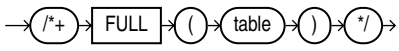


table は、全表スキャンが行われる表の名前または別名です。文に別名を使用していない場合は、表の名前がデフォルトの別名となります。

次に例を示します。

```
SELECT /*+ FULL(e) */ employee_id, last_name
  FROM employees e
  WHERE last_name LIKE :b1;
```

Oracle は、WHERE 句の条件によって使用可能になる last_name 列に索引が付けられている場合でも、employees 表で全表スキャンを実行して、この文を実行します。

> **注意：** employees 表には別名 e が存在するため、ヒントでは、名前ではなく別名で表を参照する必要があります。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 による表スキャンを選択します。

ROWID_hint::=

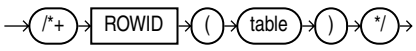


table は、ROWID によってアクセスされる表の名前または別名です。

次に例を示します。

```
SELECT /*+ROWID(employees)*/ *
FROM employees
WHERE rowid > 'AAAAtkAABAAAFNTAAA' AND employee_id = 155;
```

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・スキャンを選択します。これはクラスタ化オブジェクトにのみ適用されます。

cluster_hint::=

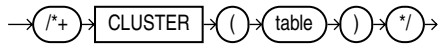


table は、クラスタ・スキャンによってアクセスされる表の名前または別名です。

次に例を示します。

```
SELECT /*+ CLUSTER */
employees.last_name, department_id
FROM employees, departments
WHERE department_id = 10
AND employees.department_id = departments.department_id;
```

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・スキャンを選択します。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_hint::=

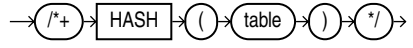
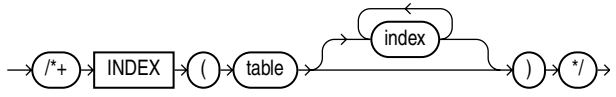


table は、ハッシュ・スキャンによってアクセスされる表の名前または別名です。

INDEX

INDEX ヒントは、指定された表に対して索引スキャンを選択します。INDEX ヒントはドメイン索引、B ツリー索引およびビットマップ・ジョイン・インデックスに使用できます。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 の使用をお薦めします。

index_hint::=



各項目は次のとおりです。

- table は、スキャンされる索引に対応付けられている表の名前または別名を指定します。
- index は、索引スキャンが行われる索引を指定します。

このヒントでは、次のように 1 つ以上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スキャンを行います。オプティマイザは、全表スキャンや、表の別の索引に対するスキャンを検討しません。
- 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スキャン・コストを検討し、コストの最も低い索引に対してスキャンを行います。また、このリストから複数の索引をスキャンし、その結果をマージするアクセス・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、全表スキャンや、ヒントでリストされていない索引に対するスキャン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スキャンのコストを検討し、コストの最も低い索引に対してスキャンを行います。また、複数の索引をスキャンし、その結果をマージするアクセス・パスのコストが最も低い場合、オプティマイザはそのアクセス・パスを選択します。オプティマイザは、全表スキャン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 列に索引があり、この列に値 m と f が含まれているとします。入院中の男性と女性の患者が同数いて、問合せで戻される行の割合が比較的大きい場合、全表スキャンの方が索引スキャンより高速にな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、比較的小さな割合の行を戻します。この場合は、索引スキャンの方が全表スキャンよりも高速で処理される可能性があります。

頻度ヒストグラムを使用する場合以外は、個別の列値の発生数はオプティマイザには使用不可能です。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されます。異なる値を 2 つのみ持つ列の場合、オプ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、索引スキャンではなく、全表スキャンのほうが選択されます。

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 ヒントを使用することによって、オプティマイザが索引スキャンを選択するように強制できます。次の文で、INDEX ヒントは、sex 列の索引 sex_index に対して索引スキャンを選択します。

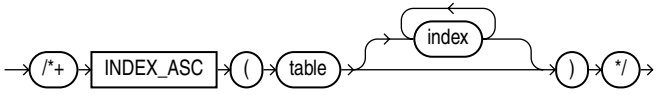
```
SELECT /*+ INDEX(patients sex_index) use sex_index because there are few
   male patients  */ name, height, weight
FROM patients
WHERE sex = 'm';
```

INDEX ヒントは、IN リスト述語に適用されます。このヒントは、可能な場合には、IN リスト述語に関して、ヒントで指定した索引をオプティマイザに強制的に使用させます。複数列の IN リストは索引を使用しません。

INDEX_ASC

INDEX_ASC ヒントは、指定された表に対して索引スキャンを選択します。文が索引レンジ・スキャンを使用する場合、Oracle は索引付きの値について昇順に索引エントリをスキャンします。

index_asc_hint::=



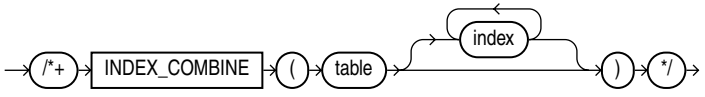
各パラメータは、INDEX ヒントと同じ目的で機能します。

Oracle のレンジ・スキャンのデフォルト動作は、索引付けされた値の昇順に索引エントリをスキャンする作業なので、このヒントでは、INDEX ヒント以外に何も指定されません。INDEX_ASC ヒントを使用して昇順のレンジ・スキャンを明示的に指定する場合は、デフォルト動作を変更してください。

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ていない場合、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_hint::=



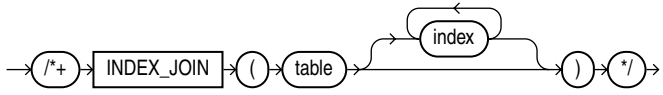
次に例を示します。

```
SELECT /*+INDEX_COMBINE(employees salary_bmi hire_date_bmi)*/ *
FROM employees
WHERE salary < 50000 AND hire_date < '01-JAN-1990';
```

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。このヒントを効果的にするには、問合せの解決に必要な列をすべて含んだ、少数の索引が必要です。

index_join_hint::=



各項目は次のとおりです。

- table は、スキャンされる索引に対応付けられている表の名前または別名を指定します。
- index は、索引スキャンが行われる索引を指定します。

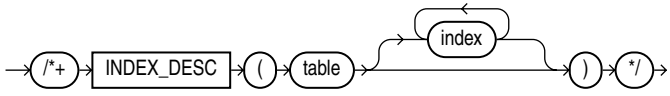
たとえば、次の問合せでは索引結合を使用して、employees 表に索引が作成されている employee_id および department_id 列にアクセスします。

```
SELECT /*+index_join(employees emp_emp_id_pk emp_department_ix)*/
    employee_id, department_id
    FROM employees
    WHERE department_id > 50;
```

INDEX_DESC

INDEX_DESC ヒントは、指定された表に対して索引スキャンを選択します。文に索引レンジ・スキャンが使用される場合、Oracle は索引エントリを索引付きの値について降順にスキャンします。パーティション索引では、各パーティション内で結果が降順に表示されます。

index_desc_hint::=



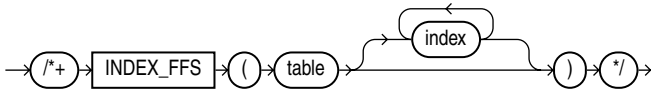
各パラメータは、INDEX ヒントと同じ目的で機能します。次に例を示します。

```
SELECT /*+ INDEX_DESC(a ord_order_date_ix) */
       a.order_date, a.promotion_id, a.order_id
  FROM orders a
 WHERE a.order_date = :b1;
```

INDEX_FFS

INDEX_FFS ヒントによって、全表スキャンではなく高速全索引スキャンが実行されます。

index_ffs_hint::=



次に例を示します。

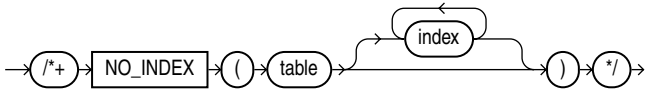
```
SELECT /*+ INDEX_FFS ( o order_pk ) */ COUNT(*)
FROM order_items l, orders o
WHERE l.order_id > 50
     AND l.order_id = o.order_id;
```

関連項目： 1-33 ページ「全体スキャン」

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_hint::=



- 使用可能な索引がヒントによって 1 つ指定されている場合、オプティマイザはその索引に対してスキャンを検討しません。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。
- 使用可能な索引のリストがヒントによって指定されている場合、オプティマイザはその索引のどれに対してもスキャンを検討しません。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。
- 索引がヒントによって指定されていない場合、オプティマイザは表のどの索引に対してもスキャンを検討しません。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、ファンクション・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索引に適用されます。NO_INDEX ヒントと索引ヒント（INDEX、INDEX_ASC、INDEX_DESC、INDEX_COMBINE または INDEX_FFS）の両方で同じ索引が指定されると、NO_INDEX ヒントおよび索引ヒントは両方とも指定された索引で無視され、オプティマイザがその索引を検討します。

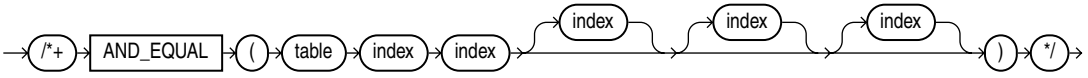
次に例を示します。

```
SELECT /*+NO_INDEX(employees emp_empid)*/ employee_id
FROM employees
WHERE employee_id > 200;
```

AND_EQUAL

AND_EQUAL ヒントは、複数の単一列索引のスキャンをマージするアクセス・パスが使用される実行計画を、明示的に選択します。

and_equal_hint::=



各項目は次のとおりです。

- table は、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を指定します。
- index は、索引スキャンが行われる索引を指定します。索引は 2 つ以上指定してください。6 つ以上の索引を指定することはできません。

問合せの変換に関するヒント

この項で説明する各ヒントは、SQL 問合せ変換を示唆するものです。

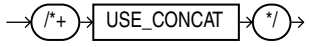
- USE_CONCAT
- NO_EXPAND
- REWRITE
- EXPAND_GSET_TO_UNION
- NOREWRITE
- MERGE
- NO_MERGE
- STAR_TRANSFORMATION
- FACT
- NO_FACT

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、使用しない場合のコストよりも低い場合にかぎり実行します。

USE_CONCAT ヒントは、IN リスト処理および OR 拡張のすべての分離処理（IN リスト自体の分離を含む）をオフにします。

use_concat_hint::=



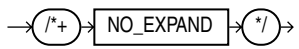
次に例を示します。

```
SELECT /*+USE_CONCAT*/ *
FROM employees
WHERE employee_id > 50 OR salary < 50000;
```

NO_EXPAND

NO_EXPAND ヒントは、コストベース・オプティマイザが OR 条件または WHERE 句の IN リストを持っている問合せの OR 拡張を検討するのを防ぎます。通常、オプティマイザは 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した場合にこのメソッドを使用します。

no_expand_hint::=



次に例を示します。

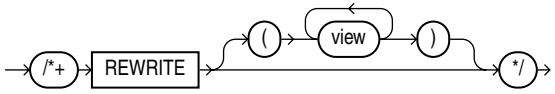
```
SELECT /*+NO_EXPAND*/ *
FROM employees
WHERE employee_id = 50 OR employee_id = 100;
```

REWRITE

REWRITE ヒントでは、可能な場合には、コストベース・オプティマイザがコストを考慮せずに、問合せを強制的にマテリアライズド・ビューに書き換えます。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、リストに適切なマテリアライズド・ビューが存在すると、Oracle はコストとは関係なく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 によって適切なマテリアライズド・ビューが検索され、コストとは関係なくそのビューが使用されます。

rewrite_hint::=



関連項目：

- マテリアライズド・ビューの詳細は、『Oracle9*i* データベース概要』および『Oracle9*i* アドバンスト・レプリケーション』を参照してください。
- マテリアライズド・ビューでの REWRITE の使用方法の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

EXPAND_GSET_TO_UNION

EXPAND_GSET_TO_UNION ヒントは、グルーピング・セットを含む問合せに使用されます（GROUP BY GROUPING SET または GROUP BY ROLLUP での問合せなど）。ヒントは、問合せを強制的に個々のグルーピングの UNION ALL を持つ対応する問合せに変換します。

expand_gset_to_union_hint::=

次に例を示します。

```
SELECT year, quarter, month, sum(sales)
FROM T
GROUP BY year, rollup(quarter, month)
```

まず、次のように変換されます。

```
SELECT year, quarter, month, sum(sales)
FROM T
GROUP BY year, quarter, month UNION ALL
SELECT year, quarter, null, sum(sales)
FROM T
GROUP BY year, quarter UNION ALL
SELECT year, null, null, sum(sales)
FROM T
GROUP BY year
```

次に、Oracle では、UNION ALL の各ブランチに、マテリアライズド・ビューでリライトが試行されます。リライトでは、マテリアライズド・ビューの再結合およびロールアップが行われる場合があります。最終的に、Oracle ではリライトされていないブランチが確認され、グルーピング・セットを持つ 1 つの問合せブロックとして示そうと試みられます。たとえば、UNION ALL の最後のブランチのみがマテリアライズド・ビュー MV でリライトされた場合、Oracle では、次のように、最初の 2 つのブランチが同等の GROUPING SET 問合せで置換されます。

```
SELECT year, quarter, month, sum(sales)
FROM T
GROUP BY grouping set ( (year, quarter, month), (year, quarter) ) UNION ALL
SELECT year, null, null, sum_sales
FROM MV
```

NOREWRITE

NOREWRITE ヒントは、パラメータ QUERY_REWRITE_ENABLED の設定を上書きして、問合せブロックに対する問合せのリライトを無効にします。NOREWRITE ヒントは、要求の問合せブロックで使用します。

> **注意：** NOREWRITE ヒントで、ファンクション・ベース索引の使用を使用禁止にします。

norewrite_hint::=

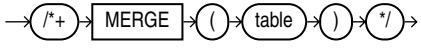
MERGE

MERGE ヒントは、各問合せにビューをマージできます。

ビューの問合せが GROUP BY 句または DISTINCT 演算子を SELECT リストに持っている場合は、複合ビューのマージが使用可能になっているときのみ、オプティマイザがビューの問合せをアクセス文にマージできます。また、複合マージは、副問合せに相互関係がない場合に、アクセス文に IN 副問合せをマージするのにも使用できます。

複合マージはコストベースではありません。つまり、アクセスする問合せブロックには、MERGE ヒントが含まれている必要があります。このヒントが存在しないと、オプティマイザは別のアプローチを使用します。

merge_hint::=



次に例を示します。

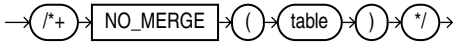
```
SELECT /*+MERGE(v)*/ e1.last_name, e1.salary, v.avg_salary
FROM employees e1,
  (SELECT department_id, avg(salary) avg_salary
   FROM employees e2
   GROUP BY department_id) v
WHERE e1.department_id = v.department_id AND e1.salary > v.avg_salary;
```

> **注意：** この例では、複合ビューのマージが使用可能になっている必要があります。

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_hint::=



このヒントを使用すると、ビューのアクセス方法に、ユーザーがより多くの影響を与えることができます。

次に例を示します。

```
SELECT /*+NO_MERGE(dallas_dept)*/ e1.last_name, dallas_dept.dname
FROM employees e1,
  (SELECT department_id, dname
   FROM departments
   WHERE loc = 'DALLAS') dallas_dept
WHERE e1.department_id = dallas_dept.department_id;
```

これにより、ビュー dallas_dept はマージされなくなります。

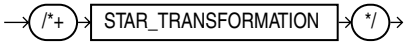
引数を指定せずに NO_MERGE ヒントを使用するときは、ビュー問合せブロック内に挿入してください。ビュー名を引数として指定して NO_MERGE を使用するときは、周囲の問合せに挿入してください。

STAR_TRANSFORMATION

STAR_TRANSFORMATION ヒントにより、オプティマイザは、変換（TRANSFORMATION）が使用されている最適な計画を使用します。このヒントを使用しない場合、オプティマイザは、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適な計画を使用する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。オプティマイザは、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存在しないので、変換されていない問合せの中で最適な計画がヒントとは無関係に使用されます。

star_transformation_hint::=



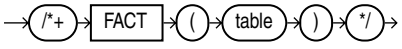
関連項目：

- スター型変換の詳細は、『Oracle9*i* データベース概要』を参照してください。
- STAR_TRANSFORMATION_ENABLED 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

FACT

FACT ヒントをスター型変換のコンテキストで使用すると、変換で、ヒントで指定された表がファクト表とみなされます。

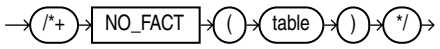
fact_hint::=



NO_FACT

NO_FACT ヒントをスター型変換のコンテキストで使用すると、変換で、ヒントで指定された表はファクト表とみなされません。

no_fact_hint::=



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。

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 ヒントを使用して結合順序を指定できます。そのような情報を把握している場合は、オプティマイザよりも適切に内部表と外部表を選択できます。

ordered_hint::=

次の問合せは ORDERED ヒントの使用例です。

```
SELECT  /*+ORDERED */ o.order_id, c.customer_id, l.unit_price * l.quantity
  FROM customers c, order_items l, orders o
 WHERE c.cust_last_name = :b1
   AND o.customer_id = c.customer_id
   AND o.order_id = l.order_id;
```

STAR

STAR ヒントは、可能な場合にはスター・クエリー・プランの使用を強制します。スター・プランは、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テッド・ループ結合に結合します。STAR ヒントが適用されるのは、3 つ以上の表があり、最大の表の連結索引が 3 つ以上の列を持ち、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、小規模表の別の並替えも考慮します。

star_hint::=

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・プランを選択します。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) */
```

facts は表、fact_concat は索引です。より一般的な方法は、STAR ヒントを使用する方法です。

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
- LEADING
- HASH_AJ、MERGE_AJ および NL_AJ
- HASH_SJ、MERGE_SJ および NL_SJ

ヒントでは、文に指定する場合と同じように正確に表を指定する必要があります。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。しかし、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薦めします。Oracle では、参照表が結合の内部表になった場合にこれらのヒントを使用し、参照表が外部表の場合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、Oracle は、別の行ソースに指定された各表を、指定の表が内部表として使用されているネステッド・ループ結合に結合します。

use_nl_hint::=

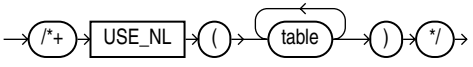


table は、ネステッド・ループ結合の内部表として使用される表の名前または別名です。

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.customer_id = customers.customer_id;
```

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、ネステッド・ループ操作、ソート / マージ操作またはハッシュ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、応答時間を最短にする、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 ヒントを使用することによって、オプティマイザにネステッド・ループ結合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テッド・ループ結合を選択します。

```
SELECT /*+ ORDERED USE_NL(customers) to get first row faster */
    accounts.balance, customers.last_name, customers.first_name
FROM accounts, customers
WHERE accounts.customer_id = customers.customer_id;
```

多くの場合、ネステッド・ループ結合は、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、ネステッド・ループ結合は、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。

ネステッド・ループがヒントによって強制的に使用される次の文の場合は、orders に対しては全表スキャンによってアクセスされ、フィルタ条件 l.order_id = h.order_id がすべての行に適用されます。フィルタ条件を満たすどの行についても、order_items は索引 order_id からアクセスされます。

```
SELECT /*+ USE_NL(l h) */ h.customer_id, l.unit_price * l.quantity
  FROM orders h ,order_items l
 WHERE l.order_id = h.order_id;
```

問合せに INDEX ヒントを追加すると、さらに大きいシステムで使用されるものに類似した実行計画になり、orders での全表スキャンが実行されるのを回避できます（今回は特に有効ではない場合も）。

USE_MERGE

USE_MERGE ヒントにより、Oracle はソート / マージ結合を使用して、それぞれの指定された表を別の行ソースと結合します。

use_merge_hint::=

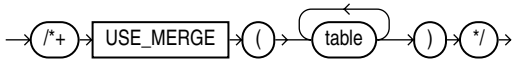


table は、結合順序の前の表を結合した結果の行ソースに、ソート / マージ結合を使用して結合される表です。

次に例を示します。

```
SELECT /*+USE_MERGE(employees departments)*/ *
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

次の問合せは、USE_MERGE ヒントが指定するソート / マージ操作を使用して、オプティマイザが GROUP BY 操作のソートを回避している在庫使用量レポートを示しています。

```
SELECT /*+ USE_MERGE(inv l) */inv.product_id, SUM(l.quantity)
  FROM inventories inv, order_items l
 WHERE inv.product_id = l.product_id(+)
   GROUP BY inv.product_id;
```

次の問合せでは、USE_MERGE ヒントと FULL ヒントが適用されています。

```
SELECT /*+USE_MERGE(h l) FULL(h l) */ h.customer_id, l.unit_price * l.quantity
  FROM orders h ,order_items l
 WHERE l.order_id = h.order_id;
```

USE_HASH

USE_HASH ヒントにより、Oracle はハッシュ結合を使用して、それぞれの指定された表と別の行ソースを結合します。

use_hash_hint::=

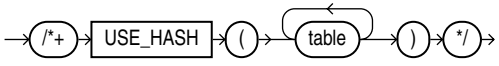


table は、結合順序の前の表を結合した結果の行ソースに、ハッシュ結合を使用して結合される表です。

次に例を示します。

```
SELECT /*+USE_HASH(l l2) */ l.order_date, l.order_id, l2.product_id,
                            SUM(l2.unit_price*quantity)
  FROM orders l, order_items l2
 WHERE l.order_id = l2.order_id
GROUP BY l2.product_id, l.order_date, l.order_id;
```

他の例は次のとおりです。

```
SELECT /*+use_hash(employees departments)*/ *
FROM employees, departments
WHERE employees.department_id = departments.department_id;
```

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を実行できます。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。

driving_site_hint::=

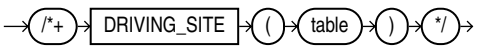


table は、実行が行われるサイトにある表の名前または別名です。

次に例を示します。

```
SELECT /*+DRIVING_SITE(departments)*/ *
FROM employees, departments@rsite
WHERE employees.department_id = departments.department_id;
```

この問合せがヒントなしで実行されると、departments 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。ヒントを使用して実行されると、employees の行はリモート・サイトに送られます。そこで、問合せが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。

LEADING

LEADING ヒントによって、Oracle は指定された表を結合順序の最初の表として使用します。

別の表にある 2 つ以上の LEADING ヒントを指定した場合は、指定したヒントはすべて無視されます。ORDERED ヒントを指定した場合は、LEADING ヒントがすべて上書きされます。

leading_hint::=

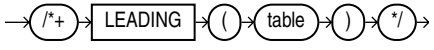


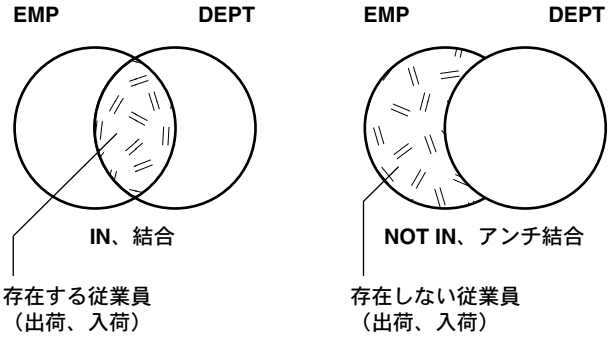
table は、結合順序の最初の表として使用される表の名前または別名です。

HASH_AJ、MERGE_AJ および NL_AJ

特定の問合せについては、MERGE_AJ ヒント、HASH_AJ または NL_AJ ヒントを NOT IN 副問合せに挿入します。MERGE_AJ はソート / マージ・アンチ結合を使用し、HASH_AJ はハッシュ・アンチ結合を使用し、NL_AJ はネステッド・ループ・アンチ結合を使用します。

図 5-1 に示すように、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できます。したがって、employees.department_id を departments.department_id に結合して、部門集合内の従業員のリストを作成できます。

図 5-1 パラレル・ハッシュ・アンチ結合



または、SQL の述語 NOT IN は、2 つの集合を減算するアンチ結合を使用して評価されます。したがって、employees.department_id を departments.department_id にアンチ結合して、部門集合内に存在しないすべての従業員を選択できます。また、出荷部門または入荷部門に所属していない従業員すべてのリストを取得できます。

HASH_SJ、MERGE_SJ および NL_SJ

特定の問合せについては、HASH_SJ ヒント、MERGE_SJ ヒントまたは NL_SJ ヒントを EXISTS 副問合せに挿入します。HASH_SJ はハッシュ・セミ結合を使用し、MERGE_SJ はソート / マージ・セミ結合を使用し、NL_SJ はネステッド・ループ・セミ結合を使用します。

たとえば、次のようにします。

```
SELECT * FROM departments
WHERE exists (SELECT /*+HASH_SJ*/ *
  FROM employees
  WHERE employees.department_id = departments.department_id
        AND salary > 200000);
```

これは、副問合せを、emp と dept の間で特定の型の結合に変換しますが、そのとき副問合せの意味は保持されます。つまり、emp の行に対する複数の一致行が dept 内に存在していても、emp が戻されるのは 1 度のみということになります。

副問合せがセミ結合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
- 副問合せに GROUP BY、CONNECT BY または ROWNUM の参照が含まれていない場合

> **関連項目：** 結合に関する詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

パラレル実行のヒント

この項では、パラレル実行を行ったときに、文がどのようにパラレル化されるか、またはパラレル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
- PARALLEL_INDEX
- NOPARALLEL_INDEX

> **関連項目：** パラレル実行の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

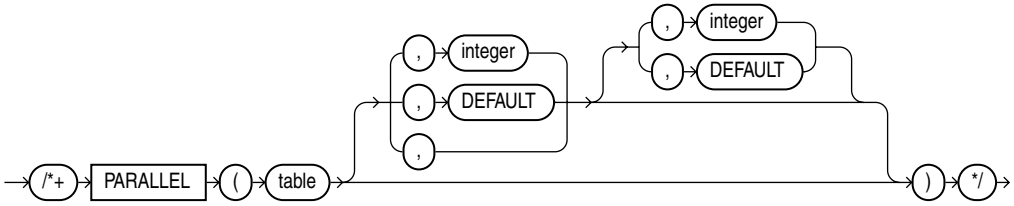
PARALLEL

PARALLEL ヒントを使用して、パラレル操作に使用できる同時サーバーの数を自由に指定します。このヒントは、文の SELECT 部分、INSERT 部分、UPDATE 部分および DELETE 部分と、表スキャン部分に適用されます。

注意： 使用できるサーバーの数は、同時にソート操作またはグルーピング操作が実行されると、PARALLEL ヒント内の値の 2 倍になります。

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。

parallel_hint::=



PARALLEL には、別名が問合せに指定されている場合、表の別名を使用する必要があります。このヒントでは 2 つの値を取得でき、表名の後に、これらをカンマで区切ります。最初の値は、指定の表の並列度を指定し、2 番目の値は、Oracle Real Application Clusters インスタンスにおける表の分割方法を指定します。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の設定を調べ、デフォルトの並列度を判別します。次の例では、PARALLEL ヒントによって、employees 表定義に指定されている並列度が上書きされます。

```
SELECT /*+ FULL(hr_emp) PARALLEL(hr_emp, 5) */ last_name
FROM hr.employees hr_emp;
```

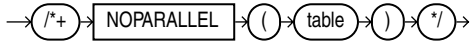
その次の例では、PARALLEL ヒントによって、employees 表定義に指定されている並列度が上書きされ、オプティマイザは、初期化パラメータに指定されているデフォルトの並列度を使用するように命令されます。また、このヒントでは、表をすべての使用可能なインスタンスに分割し、各インスタンスでデフォルトの並列度を使用するように指定します。

```
SELECT /*+ FULL(hr_emp) PARALLEL(hr_emp, DEFAULT,DEFAULT) */ last_name
FROM hr.employees hr_emp;
```

NOPARALLEL

NOPARALLEL ヒントは表句の PARALLEL 指定を上書きします。一般に、ヒントは表句よりも優先されます。

noparallel_hint::=



次の例に、NOPARALLEL ヒントを示します。

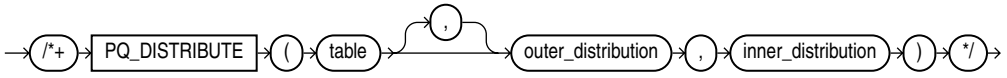
```
SELECT /*+ NOPARALLEL(hr_emp) */ last_name
FROM hr.employees hr_emp;
```

PQ_DISTRIBUTE

PQ_DISTRIBUTE ヒントで、パラレル結合操作のパフォーマンスが向上します。これを行うには、結合した表の列を生成側とコンシューマ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、オプティマイザが通常行う決定は上書きされます。

オプ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。両方の表がシリアルの場合、オプティマイザは配布ヒントを無視します。

pq_distribute_hint::=



各項目は次のとおりです。

- table は、結合の内部表として使用された表の名前または別名です。
- outer_distribution は、外部表の配布です。
- inner_distribution は、内部表の配布です。

> **関連項目：** Oracle が結合操作をパラレル化する方法の詳細は、『Oracle9*i* データベース概要』を参照してください。

表の配布には次の 6 つの組合せがあります。表 5-1 で説明するように、結合表の配布方式の組合せはサブセットのみが妥当です。

表 5-1 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、各表の行をコンシューマ問合せサーバーにマップします。マッピングが完了すると、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 / マージ結合で実装されている場合にお薦めします。
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薦めします。一般規則として、ブロードキャスト対なしのヒントは、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コンシューマ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薦めします。一般的に、なし対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。

表 5-1 配布ヒントの組合せ（続き）

配布	説明
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薦めします。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。 **注意 :** 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薦めします。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。 **注意 :** 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、各表から 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、結合キーで同一レベル・パーティション化する必要があります。

次に例を示します。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;
```

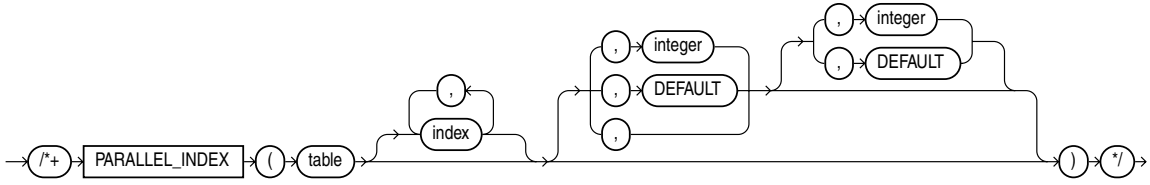
外部表 r をブロードキャストするには、問合せを次のようにします。

```
SELECT column list /*+ORDERED PQ_DISTRIBUTE(s BROADCAST, NONE) USE_HASH (s) */
FROM r,s
WHERE r.c=s.c;
```

PARALLEL_INDEX

PARALLEL_INDEX ヒントは、パーティション索引の索引レンジ・スキャンをパラレル化するために使用できる同時サーバーの数を指定します。

parallel_index_hint::=



各項目は次のとおりです。

- table は、スキャン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 は、索引スキャンが行われる索引です（オプション）。

このヒントでは 2 つの値を取ることができ、これらを表名の後に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。第 2 の値は、Oracle Real Application Clusters のインスタンス間における表の分割方法を指定します。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の設定を調べ、デフォルトの並列度を判別します。

次に例を示します。

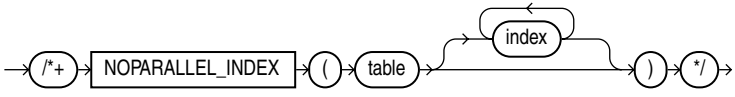
```
SELECT /*+ PARALLEL_INDEX(table1, index1, 3, 2) +/
```

この例では、2 つのインスタンスそれぞれに対して 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 ヒントは、索引の PARALLEL 属性の設定を上書きしてパラレル索引スキャン操作を回避します。

noparallel_index_hint::=



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。

APPEND

APPEND ヒントを使用すると、データベースがシリアル・モードで動作している場合にダイレクト・パス INSERT が使用可能になります。Enterprise Edition を使用していない場合、データベースはシリアル・モードです。シリアル・モードでは従来の INSERT がデフォルトであり、パラレル・モードではダイレクト・パス INSERT がデフォルトです。

ダイレクト・パス INSERT では、表に現在割り当てられている既存の領域を使用せず、表の終わりにデータが付加されます。その結果、ダイレクト・パス INSERT は従来の INSERT よりかなり高速になる可能性があります。

append_hint::=

NOAPPEND

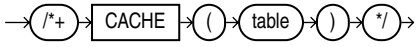
NOAPPEND ヒントを使用すると、INSERT 文の実行中にパラレル・モードを無効にして、従来の INSERT を使用可能にできます。（従来の INSERT はシリアル・モードがデフォルトであり、ダイレクト・パス INSERT はパラレル・モードがデフォルトです。）

noappend_hint::=

CACHE

CACHE ヒントでは、全表スキャンが実行され、取得されたブロックが、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。

cache_hint::=



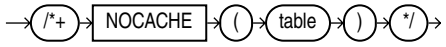
次の例では、CACHE ヒントによって、表のデフォルトのキャッシュ仕様が上書きされます。

```
SELECT /*+ FULL (hr_emp) CACHE(hr_emp) */ last_name
FROM hr.employees hr_emp;
```

NOCACHE

NOCACHE ヒントでは、全表スキャンが実行され、取得されたブロックが、バッファ・キャッシュ内で LRU リストの最初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。

nocache_hint::=



次に例を示します。

```
SELECT /*+ FULL(hr_emp) NOCACHE(hr_emp) */ last_name
FROM hr.employees hr_emp;
```

注意： CACHE ヒントおよび NOCACHE ヒントは、V$SYSSTAT ビューに示されているように、システム統計 "table scans(long tables)" および "table scans(short tables)" に影響を与えます。

小規模表の自動キャッシュ Oracle9*i* リリース 2（9.2）以降では、小規模表は、表 5-2 の基準に従って自動的にキャッシュされるようになりました。

表 5-2 表のキャッシュ基準

表サイズ	サイズ基準	キャッシュ
小規模	ブロックの数 < 20、またはキャッシュされているブロックの合計の 2% のうち、大きいもの	常にキャッシュされます
標準	小規模表よりも大きいが、キャッシュされているブロックの合計が < 10%	Oracle では、表スキャンの基礎およびワークロード履歴に表をキャッシュするかどうかが判別されます。今後の表スキャンでキャッシュされるブロックが見つかる可能性がある場合にのみ、表がキャッシュされます。
大規模	キャッシュされているブロックの合計が > 10%	キャッシュされません

小規模表の自動キャッシュは、CACHE 属性で作成または変更された表に対しては使用禁止です。

UNNEST

UNNEST ヒントは、副問合せのネスト解除を指定します。副問合せのネストの解除によって、副問合せ本体のネストが解除され、その副問合せが含まれている文の本体にマージされます。これにより、オプティマイザは、アクセス・パスと結合を評価するときに、副問合せと本文の両方をいっしょに考慮できるようになります。

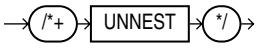
UNNEST ヒントが使用された場合、Oracle では、まず最初に文が妥当かどうかが検証されます。文が妥当でないと、副問合せのネストの解除は処理できません。次に、文は、副問合せのネストを解除することが最も効率がよいかどうかをチェックされます。

UNNEST ヒントは、副問合せブロックの妥当性のみを調査するよう Oracle に通知します。副問合せブロックが妥当な場合は、Oracle が経験則をチェックしなくても、副問合せのネスト解除が使用可能になります。

関連項目：

- ネストされた副問合せのネスト解除および副問合せブロックを妥当にする条件の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- ネスト解除の使用方法の詳細は、1-12 ページの「副問合せのネスト解除」および 2-32 ページの「CBO による副問合せのネスト解除方法」を参照してください。

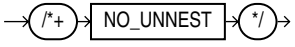
unnest_hint::=



NO_UNNEST

NO_UNNEST ヒントは、特定の副問合せブロックのネスト解除をオフに切り替えます。

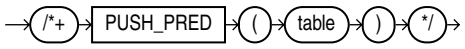
no_unnest_hint::=



PUSH_PRED

PUSH_PRED ヒントを使用すると、述語結合が強制的にビューへプッシュされます。

push_pred_hint::=



次に例を示します。

```
SELECT /*+ PUSH_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_PRED

NO_PUSH_PRED ヒントを使用すると、述語結合がビューへプッシュされません。

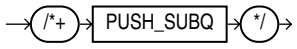
no_push_pred_hint::=

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、実行計画の最初に使用可能なステップで評価されます。一般に、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用し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。

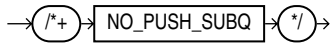
push_subq_hint::=



NO_PUSH_SUBQ

NO_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、実行計画の最後のステップとして評価されます。副問合せのコストが相対的に高く、副問合せにより行数が大幅に減少されない場合には、パフォーマンスが向上し、副問合せを最後に評価します。

no_push_subq_hint::=



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述語を除く述語評価の順序をオプティマイザに保持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、Oracle は次の順序ですべての述語を評価します。

1. ユーザー定義ファンクション、タイプ・メソッドまたは副問合せを持たない述語は、WHERE 句で指定された順序で最初に評価されます。
2. ユーザーが計算したコストのあるユーザー定義ファンクションおよびタイプ・メソッドを持つ述語が、コストの昇順で次に評価されます。
3. ユーザーが計算したコストのないユーザー定義ファンクションおよびタイプ・メソッドを持つ述語が、WHERE 句で指定された順序で次に評価されます。
4. WHERE 句で指定されていない述語（たとえば、オプティマイザにより過渡的に生成された述語）が次に評価されます。
5. 副問合せを持つ述語が、WHERE 句で指定された順序で最後に評価されます。

注意： ORDERED_PREDICATES ヒントを使用して索引キーの述語評価の順序を保持できないことを覚えておいてください。

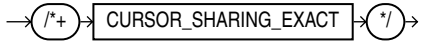
ordered_predicates_hint::=

関連項目：『Oracle9*i* データベース概要』

CURSOR_SHARING_EXACT

安全であれば、Oracle で SQL 文のリテラルをバインド変数に置き換えられます。この処理は、CURSOR_SHARING 起動パラメータで制御されます。CURSOR_SHARING_EXACT ヒントを使用すると、この処理がオフに切り替えられます。つまり、リテラルをバインド変数に置き換える処理をしなくても SQL 文が実行されます。

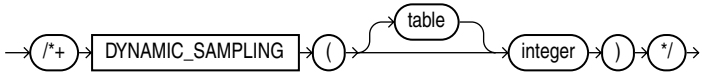
cursor_sharing_exact_hint::=



DYNAMIC_SAMPLING

DYNAMIC_SAMPLING ヒントを使用して、より正確な選択性およびカーディナリティの見積りを判別することにより、動的サンプリングを制御してサーバーのパフォーマンスを改善できます。DYNAMIC_SAMPLING の値は 0 から 10 までの値に設定できます。レベルが高くなるにつれ、コンパイラで動的サンプリングに対してさらに多くの作業が行われ、より広く適用されます。サンプリングでは、表を指定しない場合、カーソル・レベルがデフォルトとなります。

dynamic_sampling_hint::=



各項目は次のとおりです。

- table は、全表スキャンが行われる表の名前または別名です。
- integer は、0 から 10 の値で、サンプリングの度合いを示しています。文に別名を使用していない場合は、表の名前がデフォルトの別名となります。

次に例を示します。

```
SELECT /*+ dynamic_sampling(1) */ *
FROM ...
```

動的サンプリングは、次のすべての条件にあてはまる場合に使用可能になります。

- 2 つ以上の表が問合せにある場合。
- 分析されていない表があり、索引がない場合。
- オプティマイザにより、分析されていない表に相対的に高コストの表スキャンが実行されるかどうかを判別する場合。

サンプリング・レベルは、使用された動的サンプリング・レベルがカーソル・ヒントまたは `optimizer_dynamic_sampling` パラメータからの場合、次のようになります。

- レベル 0: 動的サンプリングは使用しないでください。
- レベル 1: 次の条件を満たす場合、すべての分析されていない表をサンプリングします。(1) 分析されていない表が問合せに少なくとも 1 つある場合。(2) この分析されていない表が、別の表と結合、または副問合せかマージ不可能ビューにある場合。(3) この分析されていない表に索引がない場合。(4) この分析されていない表に、この表の動的サンプリングに使用されるブロックの数よりも多いブロックがある場合。サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数です (32)。
- レベル 2: 動的サンプリングをすべての分析されていない表に適用します。サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数です。
- レベル 3: レベル 2 の基準を満たすすべての表と、標準の選択性の見積りで動的サンプリングの可能性がある述語の推論が使用されるすべての表に、動的サンプリングを適用します。サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数です。
- レベル 4: 動的サンプリングをレベル 3 の基準を満たすすべての表、および 2 つ以上の列を参照する単一表の述語を持つすべての表に適用します。サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数です。
- レベル 5: 動的サンプリング・ブロックのデフォルトの数の 2 倍を使用して、動的サンプリングをレベル 4 の基準を満たすすべての表に適用します。
- レベル 6: 動的サンプリング・ブロックのデフォルトの数の 4 倍を使用して、動的サンプリングをレベル 5 の基準を満たすすべての表に適用します。
- レベル 7: 動的サンプリング・ブロックのデフォルトの数の 8 倍を使用して、動的サンプリングをレベル 6 の基準を満たすすべての表に適用します。
- レベル 8: 動的サンプリング・ブロックのデフォルトの数の 32 倍を使用して、動的サンプリングをレベル 7 の基準を満たすすべての表に適用します。

- レベル 9: 動的サンプリング・ブロックのデフォルトの数の 128 倍を使用して、動的サンプリングをレベル 8 の基準を満たすすべての表に適用します。
- レベル 10: 表内のすべてのブロックを使用して、動的サンプリングをレベル 9 の基準を満たすすべての表に適用します。

使用された動的サンプリング・レベルが表ヒントからである場合、サンプリング・レベルは次のようになります。

- レベル 0: 動的サンプリングは使用しないでください。
- レベル 1: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数です (32)。
- レベル 2: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 2 倍です。
- レベル 3: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 4 倍です。
- レベル 4: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 8 倍です。
- レベル 5: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 16 倍です。
- レベル 6: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 32 倍です。
- レベル 7: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 64 倍です。
- レベル 8: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 128 倍です。
- レベル 9: サンプリングされたブロック数は、動的サンプリングのブロックのデフォルト数の 256 倍です。
- レベル 10: 表内のすべてのブロックを読み込みます。

特定の表に動的サンプリングを適用するには、次のヒントのフォームを使用します。

```
SELECT /*+ dynamic_sampling(employees 1) */ *
FROM employees
WHERE ..,
```

表ヒントがある場合、表が分析済で表に述語がない場合を除いて、動的サンプリングが使用されます。たとえば、`employees` が分析された場合、次の問合せでは、動的サンプリングが発生しません。

```
SELECT /*+ dynamic_sampling(e 1) */ count(*)
FROM employees e;
```

カーディナリティ統計が存在する場合、それが使用されます。述語がある場合、動的サンプリングが表ヒントを使用して行われ、カーディナリティは見積もられません。

分析された表にもカーディナリティの見積りを強制するには、次の例のように、dynamic_sampling_est_cdn というヒントをさらに使用できます。

```
SELECT /*+ dynamic_sampling(e 1) dynamic_sampling_est_cdn(t) */ count(*)
FROM employees e;
```

表が分析された場合でも、employees にカーディナリティの見積りを強制します。次の問合せは、選択性およびカーディナリティの見積りを employees に行います。

```
SELECT /*+ dynamic_sampling(e 1) dynamic_sampling_est_cdn(e) */ count(*)
FROM employees e
WHERE cols > 3;
```

ビューでのヒントの使用方法

ビュー（または副問合せ）内、またはビューに対してのヒントの使用は、お薦めしません。これは、1 つのコンテキストに定義したビューを他のコンテキストでも使用できるためです。このようなヒントによって予想外の実行計画が発生する可能性があります。特に、ビュー内のヒントまたはビューに対するヒントは、そのビューがトップレベル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場合は、この後の項で状況ごとの動作についての説明を参照してください。

- 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

表のヒントをビューまたは副問合せ内で指定する場合は、グローバル・ヒント構文の使用をお薦めします。これについては、次の項で詳しく説明します。

- グローバル・ヒント

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、トップレベルの問合せ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トップレベルの問合せにある場合は、ビューの内側にあるヒントとは関係なくそのヒントが使用されます。
- トップレベルのオプティマイザ・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かぎり、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 つ以上のモード・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（デフォルトか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・パスとヒント結合 参照されるビューに対するアクセス・パスとヒント結合は、そのビューが単一の表を含んでいないかぎり（または単一の表を持つその他のヒント・ビューを参照していないかぎり）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・パスやヒント結合は、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・パスとヒント結合 アクセス・パスとヒント結合は、ビュー定義に含めることができます。

- ビューが副問合せである場合（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・パスとヒント結合は、そのビューがトップレベル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・パスとヒント結合が保存されるのは、トップレベル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）のみです。

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繰り返し適用されます。トップレベル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。

ビューの内側のパラレル実行ヒント ビューの内側の PARALLEL、NOPARALLEL、PARALLEL_INDEX および NOPARALLEL_INDEX ヒントは、ビューがトップレベルの問合せにマージされるときに保存されます。トップレベルの問合せにあるビュー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す。つまり、トップレベルの問合せにより最適化モードが決定されます。

マージ不可能なビューはトップレベル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・パスとヒント結合は保存されます。同じ理由から、トップレベルの問合せ内のビューに対するアクセス・パスも無視されます。

ただし、トップレベルの問合せ内のビューに対するヒント結合は保存されます。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。

グローバル・ヒント

表ヒント（表を指定するヒント）は、一般に、ヒントが呼び出される場所である DELETE 文、SELECT 文または UPDATE 文内の表を参照します。文によって参照されるビュー内の表は参照しません。ビュー内に表示される表のヒントを指定する場合は、ビューに埋め込まれているヒントではなくグローバル・ヒントを使用してください。この章で説明するヒントは、表の名前に対する拡張構文を使用してグローバル・ヒントに変換できます。

次のビュー定義および SELECT 文について検討します。

```
CREATE OR REPLACE VIEW v1 AS
   SELECT *
   FROM employees
   WHERE employee_id < 150;

CREATE OR REPLACE VIEW v2 AS
   SELECT v1.employee_id employee_id, departments.department_id department_id
   FROM v1, departments
   WHERE v1.department_id = departments.department_id;

SELECT /*+ INDEX( v2.v1.employees emp_emp_id_pk ) FULL(v2.departments) */ *
   FROM v2
   WHERE department_id = 30;
```

ビュー V1 は、従業員番号が 150 未満の従業員をすべて取り出します。ビュー V2 は、ビュー V1 と部門表との結合を実行します。SELECT 文は、番号が 30 の部門にビューを制限することによって、ビュー V2 から行を取り出します。

SELECT 文には 2 つのグローバル・ヒントが存在します。最初のヒントは、ビュー V1 で参照された従業員表に対して索引スキャンを指定します。これは、ビュー V2 で参照されます。次のヒントは、ビュー V2 で参照された部門表に対して全表スキャンを指定します。これらのビュー表の点線で表された構文に注意してください。

ヒント：

```
INDEX(employees emp_emp_id_pk)
```

SELECT 文のこのヒントは、SELECT 文の FROM 句に従業員表が表示されないため、無視されます。

グローバル・ヒント構文は、マージ不可能なビューにも適用されます。次の SELECT 文について検討します。

```
SELECT /*+ NO_MERGE(v2) INDEX(v2.v1.employees emp_emp_id_pk)
                        FULL(v2.departments) */ *
FROM v2
WHERE department_id = 30;
```

この文は、V2 をマージ不可能にし、従業員および部門表のアクセス・パス・ヒントを指定します。これらのヒントは、マージされていないビュー V2 にプッシュされます。

グローバル・ヒントが UNION または UNION ALL ビューを参照している場合は、ヒント付きの表が含まれている最初のブランチにヒントが適用されます。次の SELECT 文の INDEX ヒントを参照してください。

```
SELECT /*+ INDEX(v.employees emp_emp_id_pk) */ *
FROM (SELECT *
   FROM employees
   WHERE employee_id < 150
   UNION ALL
   SELECT *
   FROM employees
   WHERE employee_id > 175) v
WHERE department_id = 30;
```

INDEX ヒントは、UNION ALL ビュー v の最初のブランチにある従業員表に適用されます。2 番目のブランチにある従業員の表ではありません。

6

SQL 文の最適化

この章では、多くのリソースを消費する SQL 文の識別、収集する内容の説明、およびチューニングの提案を示します。

この章には次の項があります。

チューニングの目的

システムをチューニングする目的は、システムのエンド・ユーザーへの応答時間を短縮したり、同じ作業の処理に使用されるリソースを削減することです。これには、次の方法があります。

- ワークロードの削減
- ワークロードの均衡化
- ワークロードのパラレル化

ワークロードの削減

一般に SQL のチューニングを行う目的は、同じワークロードをより効率的に処理する方法を見つけ出すことです。機能性を変えることなく文の実行計画を変更し、リソース使用量を削減することは可能です。

リソース使用量を削減する方法の 2 つの例を、次に示します。

1. 一般に実行される問合せで、アクセスするデータの表中での割合が少ない場合、効率的な問合せの実行方法として、索引の使用があります。索引を作成すれば、使用するリソースの量を削減できます。
2. ユーザーが、特定のソート順序で戻される 10,000 行の最初の 20 行を見る場合でかつ、索引で問合せ（およびソート順序）を満たすことができる場合、最初の 20 行を見るために、10,000 行にアクセスしてソートする必要はありません。

ワークロードの均衡化

システムは、実ユーザーがシステムに接続している昼間に使用量が最大に達し、夜間には低下する傾向があります。重要でないレポートやバッチ・ジョブを夜間に実行するようにスケジューリングし、昼間の並行性が削減されれば、昼間の、より重要なプログラムのためにリソースが解放されます。

ワークロードのパラレル化

大量のデータにアクセスする問合せ（代表的なものは、データ・ウェアハウス問合せ）は、多くの場合、パラレル化できます。これは特に、並行性が低いデータ・ウェアハウスで応答時間を短縮する場合に有効です。ただし、並行性が高い傾向のある OLTP 環境の場合は、プログラム全体のリソース使用量を増加させることになり、他のユーザーに影響を与える可能性があります。

多くのリソースを消費する SQL の識別およびデータ収集

この項では、パフォーマンスの低い SQL 文について、識別およびデータ収集を行う手順を説明します。

多くのリソースを消費する SQL の識別

多くのリソースを消費する SQL を識別する最初の手順は、検討する問題を分類することです。単一（または少数）のプログラムに固有な問題であるか、アプリケーション全般にわたる一般的な問題であるかに分類します。

特定のプログラムのチューニング

特定のプログラム（GUI または 3GL）をチューニングする場合、検討する SQL の識別は、プログラム内で実行された SQL を見るだけの簡単な作業です。

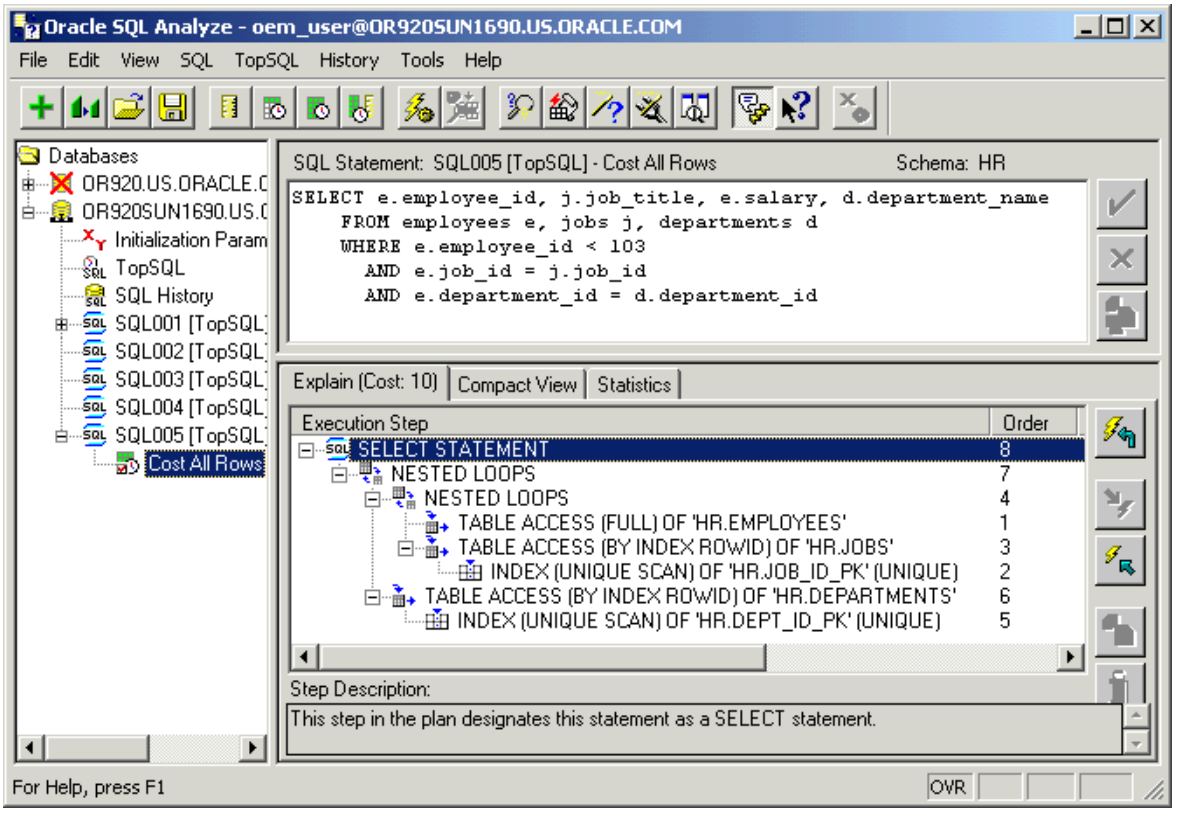
SQL を識別できない（たとえば、SQL が動的に生成される）場合は、`SQL_TRACE` を使用して、実行された SQL を含むトレース・ファイルを生成し、次に `TKPROF` を使用して出力ファイルを生成します。

`TKPROF` 出力ファイル内の SQL 文は、実行経過時間（`exeela`）などの各種パラメータで順序付けできるため、通常は、SQL 文を経過時間で順序付けする（経過時間が最も長い SQL 文をファイルの一番上に置く）ことで識別に利用されます。これにより、ファイル内に SQL 文が多数ある場合に、パフォーマンスの低い SQL を識別するジョブの実行が容易になります。

関連項目： 第 10 章「SQL トレースおよび TKPROF の使用」

Oracle SQL Analyze を使用して、リソースを多く使用する SQL 文の識別、EXPLAIN PLAN の生成および SQL パフォーマンスの評価ができます。図 6-1 は、1-19 ページの例 1-3「EXPLAIN PLAN の使用方法」で使用する SQL 文を表示した SQL Analyze を図解したものです。

図 6-1 Oracle SQL Analyze



関連項目： Oracle SQL Analyze の詳細は、『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

アプリケーションのチューニング / 負荷の軽減

アプリケーション全体のパフォーマンスが十分に最適化されていない場合や、データベース・サーバーの CPU または I/O 全体の負荷を減らそうとする場合は、次の手順で、多くのリソースを消費する SQL を識別します。

1. 検討する時間帯を判別します。通常は、アプリケーションの処理のピーク時間です。
2. その期間のオペレーティング・システムおよび Oracle 統計を収集します。収集する最小限の Oracle 統計は、ファイル I/O（V$FILESTAT）、システム統計（V$SYSSTAT）および SQL 統計（V$SQLAREA または V$SQL、V$SQLTEXT、および V$SQL_PLAN）です。

   関連項目： Statspack を使用して Oracle インスタンス・パフォーマンス・データを収集する方法については、第 21 章「Statspack の使用方法」を参照してください。

3. ステップ 2 で収集したデータを使用して、多くのリソースを使用する SQL 文を識別します。候補の SQL 文を識別するには、V$SQLAREA を問い合せる方法が適しています。V$SQLAREA には、共有プール内のすべての SQL 文に関するリソース使用率の情報が含まれています。V$SQLAREA 内のデータを、リソース使用率で順序付けしてください。共通リソースの主なものは、次のとおりです。
   - バッファ取得（V$SQLAREA.BUFFER_GETS。CPU 使用率の高い文の問合せ。）
   - ディスク読取り（V$SQLAREA.DISK_READS。I/O 使用率の高い文の問合せ。）
   - ソート（V$SQLAREA.SORTS。ソートの多い文の問合せ。）

負荷の最も大きい SQL 文を識別する方法の 1 つは、その期間内に SQL 文で使用されたリソースを、同じ期間内に使用されたそのリソースの総量と比較することです。BUFFER_GETS の場合、各 SQL 文の BUFFER_GETS の回数を、期間中のバッファ取得の総数で除算します。システム内のバッファ取得の総数は V$SYSSTAT 表の session logical reads の統計情報からわかります。

同様に、V$SQL_AREA.DISK_READS を V$SYSSTAT 統計の physical reads の値で除算すると、システムによって実行されるディスク読取りの総数のうち、文によって実行されるディスク読取りの割合を計算できます。Statspack レポートの SQL セクションにはこのデータが含まれているため、割合を手動で計算する必要はありません。

関連項目： V$SQLAREA および V$SQL の詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

候補の SQL 文を識別した後、文を調べるために必要な情報を収集し、チューニングします。

識別した SQL に関するデータの収集

CPU が問題となっている場合は、一定期間内に最も多くの BUFFER_GETS を実行した上位の SQL 文を調べます。その他の場合は、最も多くの DISK_READS を実行した SQL 文から始めます。

チューニング中に収集する情報

チューニング・プロセスではまず、基礎となる表と索引の構造を判別します。

収集する情報は、次のとおりです。

1. V$SQLTEXT からの完全な SQL テキスト
2. SQL 文で参照される表の構造（通常は SQL*Plus の describe コマンドを実行）
3. すべての索引（列、列の順序付け）の定義と、各索引が一意かどうか
4. セグメントの CBO 統計（表ごとの行数、索引列の選択性など）、およびセグメントが最後に分析された日付
5. SQL 文で参照されるビューの定義
6. ステップ 4 で検出された、ビュー定義で参照されている表について、ステップ 2 と 3 を繰り返します。
7. SQL 文（EXPLAIN PLAN、V$SQL_PLAN または TKPROF 出力のいずれかからのもの）のオプティマイザ計画の安定性
8. その SQL 文の以前のオプティマイザ計画の安定性

注意： アプリケーション内の主要な SQL 文すべてについて、実行計画を生成し、見直すことが重要です。これにより、SQL 文が効率良く実行されたときのオプティマイザの実行計画と、そうでないときの計画とを比較できます。データ量の変化などの情報とあわせて比較を行うと、パフォーマンスの低下の原因を正確に識別できます。

動的サンプリング

動的サンプリングの目的は、選択性およびカーディナリティをより正確に見積ることにより、サーバーのパフォーマンスを改善することです。選択性およびカーディナリティをより正確に見積ると、オプティマイザでより適切な実行計画を作成できます。

動的サンプリングを使用すると、次の作業が可能になります。

- 収集された統計が使用できない、あるいは見積りで重大なエラーを引き起こす可能性がある場合に、単一表の述語の選択性を見積ります。
- 統計のない表、あるいは統計が古すぎて信頼できない表に対する表のカーディナリティを見積ります。

動的サンプリングの動作

重要なパフォーマンス属性は、コンパイル時に決定されます。Oracle では、問合せで動的サンプリングを使用する利点があるかどうか、コンパイル時に判別されます。利点がある場合、再帰的 SQL 文が発行されて表のブロックの小さなランダム・サンプルがスキャンされ、関連する単一表の述語を適用することで述語の選択性の見積りが行われます。サンプルしたカーディナリティが、表のカーディナリティの見積りに使用される場合もあります。

`OPTIMIZER_DYNAMIC_SAMPLING` 初期化パラメータの値によって、動的サンプリングの問合せによって読み込まれるブロック数が決まります。

動的サンプリング使用のタイミング

通常、迅速に（数秒以内で）完了する問合せに対しては、動的サンプリングのコストが発生するのは望ましくありません。しかし、次のいずれかの条件が当てはまる場合は、動的サンプリングが有効です。

- 動的サンプリングを使用すると、より優れた計画になる場合。
- サンプリングにかかる時間が、問合せの実行時間全体のごく一部である場合。
- 問合せが何度も実行される場合。

動的サンプリングは、単一表の述語によるサブセットに適用したり、動的サンプリングが行われていない述語の通常の選択性の見積りと組み合せることができます。

動的サンプリングを使用したパフォーマンスの改善方法

動的サンプリングは、OPTIMIZER_DYNAMIC_SAMPLING パラメータを使用して制御します。このパラメータの値は、0 ～ 10 に設定できます。

- 値が 0 の場合、動的サンプリングが行われません。
- 値が 1（デフォルト）の場合は、次のすべての条件が満たされたときに動的サンプリングが実行されます。
  - 問合せに複数の表が含まれる場合。
  - 一部の表が分析されておらず、索引がない場合。
  - 分析されていない表に対し、比較的コストの高い表スキャンが必要であるとオプティマイザが判別した場合。
- パラメータの値が大きくなるにつれて、サンプリングされる表（分析された表、あるいは分析されていない表）のタイプについても、サンプリングで使用される I/O の量に関しても、動的サンプリングがより積極的に適用されるようになります。

動的サンプリングは、サンプリング対象の表内で行が挿入、削除または更新されていない場合、同じものが繰り返し使われます。

パラメータ OPTIMIZER_FEATURES_ENABLE が 9.2.0 より前のリリースに設定されている場合、動的サンプリングはオフになります。

> **関連項目：** このヒントを使用する場合の詳細は、5-39 ページの「DYNAMIC_SAMPLING」を参照してください。

SQL 文のチューニングの概要

この項では、SQL 文の効率を高める方法を説明します。

- オプティマイザ統計の確認
- 実行計画の検討
- SQL 文の再構成
- 索引の再構成
- トリガーおよび制約の変更または無効化
- データの再構成
- 実行計画の長期的な保持
- データへのアクセスを最小限に削減

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。この項でお薦めし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。

オプティマイザ統計の確認

CBO では、最適な実行計画を判別するときに、表と索引について収集された統計を使用します。これらの統計が収集されなかった場合、または、統計がデータベース内に格納されているデータをすでに反映しなくなっている場合、オプティマイザには最適な計画を生成するための十分な情報がありません。

チェックする内容

- データベース内のいくつかの表に関する統計が必要な場合は、すべての表に関する統計を収集するのが望ましい方法です。このことは、結合を実行する SQL 文がアプリケーションに含まれている場合に特に言えます。
- データ・ディクショナリ内のオプティマイザ統計が表と索引内のデータを反映しなくなっている場合は、新しい統計を収集します。ディクショナリ統計が失効しているかどうかをチェックする方法の 1 つは、表の実際のカーディナリティ（行数）と、DBA_TABLES.NUM_ROWS の値を比較することです。述語列に大きなデータの偏りがある場合は、ヒストグラムを使用することを検討してください。

実行計画の検討

OLTP 環境で SQL 文をチューニング（または作成）する場合、最も選択性の高いフィルタを持つ表から駆動することを目標とします。つまり、次のステップに渡される行を少なくするということです。次のステップが結合である場合は、少数の行しか結合されないということになります。アクセス・パスが最適かどうかを確認してください。

オプティマイザの実行計画を調べる場合は、次の内容を確認します。

- 駆動表が最適なフィルタを持つ計画であること。
- 各ステップの結合順序は、最少数の行が次のステップに戻されるようにします（つまり、可能であれば、結合の順序は最適な未使用フィルタを選んで進むようにすべきです）。
- 結合方法が、それによって戻される行数からみた場合に、適切なものであること。たとえば、索引でのネステッド・ループ結合は、多数の行が戻される場合には最適ではありません。
- ビューが効果的に使用されていること。SELECT 構文のリストを見て、ビューへのアクセスが必要であるかどうかを確認してください。
- 意図しないデカルト積（小さい表の場合も含む）があるかどうか。

- 各表が効率的にアクセスされていること。

  SQL 文の述語と、表の行数を評価します。大量の行を持つ表の全表スキャンなど、WHERE 句に述語を持つ疑わしいアクティビティを探します。そのような選択的な述語に索引が使用されない理由を判別します。

  全表スキャンが非効率的というわけではありません。小さい表で全表スキャンを行う場合や、戻される行数に対してよりよい結合方法（たとえば、hash_join）を活用するために、全表スキャンを行うほうが効率がよい場合があります。

これらの条件のうち最適でないものがある場合は、SQL 文の再構成や、表で使用できる索引について考慮します。

SQL 文の再構成

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。元の文の意図を理解していれば、要件を満たす新しい文を迅速かつ容易に作成できます。

AND と = を使用した条件の組立て

SQL の効率性を高めるには、可能なかぎり等価結合を使用します。変換されない列値に対して等価結合を実行する文は、最も容易にチューニングできます。

WHERE 句での変換列の回避

変換されない列値を使用します。たとえば、次の例のように使用します。

```
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 ファンクションを使用しないでください。列を引数として持つ関数など、列を使用する式は、使用できる定義済みファンクション・ベース索引がないかぎり、その列の索引を使用できる可能性をオプティマイザが無視する原因となります（一意索引も例外ではありません）。

型が混在する式は避け、暗黙的な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
```

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
```

次に示すタイプの複合式は使用しないようにしてください。

- col1 = NVL (:b1,col1)
- NVL(col1,-999) = ...
- TO_DATE(), TO_NUMBER() など

ここに示した式によって、オプティマイザは有効なカーディナリティまたは選択性の見積りを割り当てることができなくなります。この結果全体の計画および結合方法に悪い影響を与えます。

NVL() のかわりに述語を追加する

次に例を示します。

```
SELECT employee_num, full_name Name, employee_id
FROM mtl_employees_current_view
WHERE (employee_num = NVL (:b1,employee_num)) AND (organization_id=:1)
ORDER BY employee_num;
```

これは次のようにします。

```
SELECT employee_num, full_name Name, employee_id
FROM mtl_employees_current_view
WHERE (employee_num = :b1) AND (organization_id=:1)
ORDER BY employee_num;
```

フィルタまたは述語結合で SQL ファンクションを使用する必要がある場合、索引を持つ列に対しては使用しないでください。その代わり、次の文に示すように、述語の反対側に SQL ファンクションを使用してください。

```
TO_CHAR(numcol) = varcol
```

次の例は使用しません。

```
varcol = TO_CHAR(numcol)
```

> **関連項目：** ファンクション・ベース索引の詳細は、第 4 章「索引およびクラスタ」を参照してください。

特定のタスクに対する専用の SQL 文の作成

SQL は、手続き型言語ではありません。1 つの SQL を使用して様々なことを実行すると、通常は各タスクに最適でない結果が生じます。SQL を使用して様々なタスクを実行する場合は、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のではなく、様々な文を作成してください。

注意： Oracle Forms と Oracle Reports は、PL/SQL（トリガーまたはプログラム・ユニット）を使用してアプリケーション・ロジックをコード化するための強力な開発ツールです。Forms または Reports で複雑なロジックを処理することによって、SQL 文の複雑さを減らすことができます。また、規模の大きな単一の複雑な SQL 文のかわりに、少数の SQL 文を実行するサーバー側の PL/SQL パッケージを起動することもできます。このパッケージはサーバー側のユニットであるため、クライアントとデータベースの間のラウンドトリップやネットワークの通信量の問題は発生しません。

通常、異なるタスクには個別の SQL 文を作成することが適していますが、使用する SQL 文を 1 つにする必要がある場合は、UNION ALL 演算子を使用することによって、非常に複雑な文を多少簡略化することができます。

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。この列を含む式が、BETWEEN の両辺で同じ列を使用するためです。

このことは、選択性の高い、索引作成可能な他の条件が別にあって、それを使用し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用することは多くありますが、:loval などの値を、あらかじめ知っておく必要があります。この情報があれば、索引を使用できない ALL のケースを除外できます。

:loval と :hival に実際の値が指定されている場合には必ず索引を使用する場合（:loval と :hival の間が狭く、多くの場合等しいことが期待できる場合）は、この例を論理的に等しい、次の形式にリライトすることができます。

```
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');
```

この新しい問合せで EXPLAIN PLAN を実行した場合、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るように思われます。しかし、データベースが UNION ALL の前半と後半のどちらを実行するかを決めるために最初に評価する条件は、:hival と :loval が ALL であるかどうかの複合条件です。データベースは、問合せの前半と後半のどちらかの実行計画から実際に行を取得する前に、この条件を評価します。

UNION ALL 問合せの一方に関して条件が false であれば、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のみが実際に実行されます。:hival と :loval に関する最終条件はどちらか一方のみが true であることが保証されているので、実際に行を戻すのは UNION ALL の半分のみです。（UNION ALL 内の ALL は、この排他性により論理的に有効です。これにより、問合せの両半分の結果から重複行を除外するためにコストの高いソートを実行することなく、計画を実行できます。）

副問合せに対する EXISTS と IN の使用

ある環境では、EXISTS より IN を使用したほうが適していることがあります。一般に、選択的述語が副問合せにある場合は、IN を使用します。選択的述語が親問合せの中にある場合は、EXISTS を使用します。

注意： この説明は、親 SQL または副問合せへのアクセス・パスが選択性の高い索引付きの列を経由するような OLTP 環境で最も当てはまります。DSS 環境では、親 SQL または副問合せの選択性が低い場合があり、結合列には索引がない可能性もあります。DSS 環境では、EXISTS の場合にセミ結合を使用することを考慮してください。

関連項目：

- 1-42 ページ「CBO によるアンチ結合の実行方法」
- 5-28 ページ「HASH_AJ、MERGE_AJ および NL_AJ」および 5-28 ページ「HASH_SJ、MERGE_SJ および NL_SJ」
- 『Oracle9*i* データ・ウェアハウス・ガイド』

副問合せに IN 句を使用した場合に、副問合せで指定される選択性の利点を活用するために、Oracle が副問合せをリライトすることがあります。これは、最も選択性の高いフィルタが副問合せにある場合、結合列に索引がある場合に、最も有効です。これに対し、EXISTS は、最も選択性の高いフィルタが親問合せにある場合に有効です。その場合、EXISTS 基準に照らして行をフィルタにかける前に、親問合せの選択的述語を適用できるからです。

注意： 使用したリソース（V$SQL または V$SQLAREA からの BUFFER_GETS、DISK_READS または CPU_TIME）の実際の数で、文の CBO コストを検証する必要があります。データの偏り（ヒストグラムを使用しない）などの状況は、オプティマイザの操作コストの見積りにマイナスの影響を与える可能性があります。

次に、IN および EXISTS の利点を実証する 2 つの例を示します。いずれの例でも、次の特性を持つ同じスキーマを使用します。

- employees.employee_id フィールドに一意索引があります。
- orders.customer_id フィールドに索引があります。
- employees.department_id フィールドに索引があります。
- employees 表には 27,000 行あります。
- orders 表には 10,000 行あります。
- OE および HR スキーマに、これらのセグメントがあり、ともに COMPUTE で分析されています。

例 1: IN の使用 ー 副問合せ内の選択的フィルタ この例は、IN を使用するように問合せをリライトすると、パフォーマンスがどのように向上するかを示しています。この問合せでは、顧客 144 のオーダーを発注したすべての社員を識別します。

EXISTS を使用する SQL 文

```
SELECT /* EXISTS example */
       e.employee_id
     , e.first_name
     , e.last_name
     , e.salary
  FROM employees e
 WHERE EXISTS (SELECT 1 FROM orders o                  /* Note 1 */
                WHERE e.employee_id = o.sales_rep_id   /* Note 2 */
                  AND o.customer_id = 144);            /* Note 3 */
```

注意 :

- Note 1: EXISTS を含む行です。
- Note 2: 副問合せを相関副問合せにする行です。
- Note 3: 相関副問合せに選択性の高い述語 customer_id = *number* が含まれている行です。

次に、前述の文の実行計画（V$SQL_PLAN からのもの）を示します。この計画では、employees 表の全表スキャンを必要とし、そのため、多数の行が戻されます。次に、戻された各行が orders 表で（索引を経由して）フィルタにかけられます。

```
  ID OPERATION            OPTIONS         OBJECT_NAME            OPT       COST
---- -------------------- --------------- ---------------------- --- ----------
   0 SELECT STATEMENT                                            CHO
   1  FILTER
   2   TABLE ACCESS       FULL            EMPLOYEES              ANA        155
   3   TABLE ACCESS       BY INDEX ROWID  ORDERS                 ANA          3
   4    INDEX             RANGE SCAN      ORD_CUSTOMER_IX        ANA          1
```

IN を使用して文をリライトすると、使用されるリソースが大幅に減少します。

IN を使用する SQL 文

```
SELECT /* IN example */
       e.employee_id
     , e.first_name
     , e.last_name
     , e.salary
  FROM employees e
 WHERE e.employee_id IN (SELECT o.sales_rep_id         /* Note 4 */
                           FROM orders o
                          WHERE o.customer_id = 144);  /* Note 3 */
```

注意 :

- Note 3: 相関副問合せに選択性の高い述語 customer_id = *number* が含まれている行です。
- Note 4:IN が使用されている行です。副問合せは、相関ではなくなっています。これは、IN 句が副問合せ内の結合を置換するためです。

次に、前述の文の実行計画（V$SQL_PLAN からのもの）を示します。オプティマイザは副問合せをビューにリライトし、次にそれが一意索引で employees 表に結合されます。この結果、計画は大幅に改善されます。ビュー（すなわち、副問合せ）に選択的述語があるため、わずかな employee_id のみが戻されるためです。次に、このわずかな employee_id で、一意索引から employees 表にアクセスします。

```
  ID OPERATION            OPTIONS         OBJECT_NAME            OPT       COST
---- -------------------- --------------- ---------------------- --- ----------
   0 SELECT STATEMENT                                            CHO
   1  NESTED LOOPS                                                            5
   2   VIEW                                                                   3
   3    SORT               UNIQUE                                             3
   4     TABLE ACCESS      FULL            ORDERS                 ANA         1
   5   TABLE ACCESS        BY INDEX ROWID  EMPLOYEES              ANA         1
   6    INDEX              UNIQUE SCAN     EMP_EMP_ID_PK          ANA
```

例 2: EXISTS の使用 ー 親の中の選択的述語 この例は、EXISTS を使用するように問合せをリライトすると、パフォーマンスがどのように向上するかを示しています。この問合せでは、オーダーを発注した、部門 80 の全営業社員を識別します。

IN を使用する SQL 文

```
  SELECT /* IN example */
         e.employee_id
       , e.first_name
       , e.last_name
       , e.department_id
       , e.salary
    FROM employees   e
   WHERE e.department_id = 80                                    /* Note 5 */
     AND e.job_id        = 'SA_REP'                              /* Note 6 */
     AND e.employee_id IN (SELECT o.sales_rep_id FROM orders o); /* Note 4 */
```

注意 :

- Note 4: IN が使用されている行です。副問合せは、相関ではなくなっています。これは、IN 句が副問合せ内の結合を置換するためです。
- Note 5 および 6: 親 SQL 内の選択的述語です。

次に、前述の文の実行計画（V$SQL_PLAN からのもの）を示します。SQL 文は、orders 表でビューを使用するようにオプティマイザでリライトされています。この文では、orders 表に存在するすべての一意の employee_id を戻すためにデータのソートが必要です。述語がないため、多数の employee_ids が戻されます。この多数の employee_id からなる大きなリストが、一意索引を用いて employees 表へのアクセスに使用されることになります。

```
  ID OPERATION            OPTIONS         OBJECT_NAME            OPT       COST
---- -------------------- --------------- ---------------------- --- ----------
   0 SELECT STATEMENT                                            CHO
   1  NESTED LOOPS                                                          125
   2   VIEW                                                                 116
   3    SORT               UNIQUE                                           116
   4     TABLE ACCESS      FULL            ORDERS                 ANA        40
   5   TABLE ACCESS        BY INDEX ROWID  EMPLOYEES              ANA         1
   6    INDEX              UNIQUE SCAN     EMP_EMP_ID_PK          ANA
```

EXISTS を使用する SQL 文

```
SELECT /* EXISTS example */
       e.employee_id
     , e.first_name
     , e.last_name
     , e.salary
  FROM employees   e
 WHERE e.department_id = 80                          /* Note 5 */
   AND e.job_id        = 'SA_REP'                    /* Note 6 */
   AND EXISTS (SELECT 1                              /* Note 1 */
                 FROM orders o
                WHERE e.employee_id = o.sales_rep_id);  /* Note 2 */
```

注意：

- Note 1: EXISTS を含む行です。
- Note 2: 副問合せを相関副問合せにする行です。
- Note 5 および 6: 親 SQL 内に選択的述語があります。

次に、前述の文の実行計画（V$SQL_PLAN からのもの）を示します。計画のコストは、EXISTS を使用するように SQL 文をリライトすることで削減されます。この計画は、より効率的です。これは、2 つの索引を使用して親問合せ内の述語を満たすため、戻される employee_id がきわめて少ないためです。次に、この employee_id を使用して、索引から orders 表にアクセスします。

```
  ID OPERATION            OPTIONS         OBJECT_NAME            OPT       COST
---- -------------------- --------------- ---------------------- --- ----------
   0 SELECT STATEMENT                                            CHO
   1  FILTER
   2   TABLE ACCESS       BY INDEX ROWID  EMPLOYEES              ANA         98
   3    AND-EQUAL
   4     INDEX            RANGE SCAN      EMP_JOB_IX             ANA
   5     INDEX            RANGE SCAN      EMP_DEPARTMENT_IX      ANA
   6   INDEX              RANGE SCAN      ORD_SALES_REP_IX       ANA          8
```

注意： より効果的なアプローチは、department_id および job_id の連結索引を作成することです。これによって 2 つの索引にアクセスする必要がなくなり、使用されるリソースが削減されます。

ヒントによるアクセス・パスおよび結合順序の制御

オプティマイザのアプローチと目標を設定して CBO の代表的な統計を収集することによって、オプティマイザの選択を変えることができます。特定のアプリケーション・データに関して、オプティマイザよりも多くの情報を持つアプリケーション・デザイナであれば、より効率良く SQL 文を実行する方法を選択できる場合があります。SQL 文のヒントを使用すれば、文を実行する方法を指定できます。

/*+FULL */ などのヒントは、アクセス・パスを制御します。その例を次に示します。

```
SELECT /*+ FULL(e) */ e.ename
FROM emp e
WHERE e.job = 'CLERK';
```

> **関連項目：** 第 1 章「オプティマイザの概要」および 第 5 章「オプティマイザ・ヒント」

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、結果に影響しない不要な行にアクセスする作業を回避することです。このことから次の 3 つの一般ルールが導かれます。

- 索引を介して必要な行を取得するほうが効率的な場合には、全表スキャンを避けてください。
- 100 行をフェッチする別の索引を使用できる場合には、駆動表から 10,000 行をフェッチする索引を使用するようなことは避けてください。
- 結合順序の後ろへ行くほど結合する行が少なくなるように結合順序を選択してください。

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 100 AND 200
    AND b.bcol BETWEEN 10000 AND 20000
    AND c.ccol BETWEEN 10000 AND 20000
    AND a.key1 = b.key1
    AND a.key2 = c.key2;
```

1. 駆動表と駆動索引（存在する場合）を選択します。

   前述の例における最初の 3 つの条件は、それぞれ 1 つの表にのみ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、駆動表と駆動索引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は駆動表です。この例では、100 ～ 200 の範囲は `acol` の範囲に比べて狭く、10000 ～ 20000 の範囲は相対的に大きいため、`taba` は駆動表であり、他はすべて同じです。

ネステッド・ループ結合の場合、結合はすべて結合索引を介して行う必要があります。この結合索引は、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結びつけるために使用します。駆動表を除いて、非結合条件にこの結合索引を使用することはほとんどありません。そのため、taba を駆動表として選択した後は、b.key1 と c.key2 の索引を使用して tabb と tabc を駆動します。

2. 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します。

   最適な未使用フィルタを持つ表に先に結合することでその後の結合の作業は、削減できます。したがって、「bcol BETWEEN ...」が「ccol between ...」よりも限定的な（行をより高い比率で排除する）場合は、tabb が tabc よりも前に結合されると、最後の結合はより簡単に（より少ない行で）実行できます。

3. ORDERED または STAR ヒントを使用して、結合順序を強制的に設定できます。

関連項目： 5-23 ページ「結合順序のヒント」

ビューを管理するときの注意

ビューの結合、ビューへの外部結合、および既存ビューの新規目的への再利用に対しては、注意が必要です。

複合ビューを結合するときの注意 複合ビューへの結合、特に、ある複合ビューから別の複合ビューへの結合はお薦めできません。そのような結合を行うと、多くの場合、ビュー全体がインスタンス化され、ビュー・データに対して問合せが行われる結果になります。

たとえば、次の文は従業員および部門をリストするビューを作成します。

```
CREATE OR REPLACE VIEW emp_dept
AS
SELECT d.department_id
     , d.department_name
     , d.location_id
     , e.employee_id
     , e.last_name
     , e.first_name
     , e.salary
     , e.job_id
FROM  departments d
     ,employees e
WHERE e.department_id (+) = d.department_id
/
```

次の問合せは指定した状態の従業員を検索します。

```
SELECT v.last_name, v.first_name, l.state_province
FROM locations l, emp_dept v
WHERE l.state_province = 'California'
AND   v.location_id = l.location_id (+)
/
```

次の計画では、emp_dept ビューがインスタンス化されます。

```
Plan Table
-------------------------------------------------------------------------------
| Operation                 |  Name    |  Rows | Bytes|  Cost  | Pstart| Pstop |
-------------------------------------------------------------------------------
| SELECT STATEMENT          |          |       |      |        |       |       |
|  FILTER                   |          |       |      |        |       |       |
|   NESTED LOOPS OUTER      |          |       |      |        |       |       |
|    VIEW                   |EMP_DEPT  |       |      |        |       |       |
|     NESTED LOOPS OUTER    |          |       |      |        |       |       |
|      TABLE ACCESS FULL    |DEPARTMEN |       |      |        |       |       |
|      TABLE ACCESS BY INDEX|EMPLOYEES |       |      |        |       |       |
|       INDEX RANGE SCAN    |EMP_DEPAR |       |      |        |       |       |
|    TABLE ACCESS BY INDEX R|LOCATIONS |       |      |        |       |       |
|     INDEX UNIQUE SCAN     |LOC_ID_PK |       |      |        |       |       |
-------------------------------------------------------------------------------
```

ビューの再利用禁止 ある用途のために作成したビューを他の用途に使用することは、不適切な場合があるため注意してください。ビューから問合せを行うと、データを戻すために、そのビューに関連するすべての表がアクセスされます。ビューを再利用する前に、ビュー内のすべての表にアクセスしてデータを戻す必要があるかどうかを判別してください。その必要がない場合は、ビューを使用しないでください。かわりに、実表を使用するか、必要に応じて新しいビューを定義してください。その目的は、必要なデータを戻すために参照する表およびビューの数を最小限にすることにあります。

次の例を見てください。

```
SELECT dname
FROM emp_dept
WHERE deptno=10;
```

ビュー全体はまず、emp および dept 表の結合を行ってインスタンス化され、次に、データを集めます。ただし、dname と deptno は、dept 表から直接取得できます。（前の例で宣言されている）emp_dept 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。

副問合せのネストを解除するときの注意 副問合せのネストの解除によって、副問合せ本体が解除され、その副問合せが含まれている文の本体にマージされます。これにより、オプティマイザは、アクセス・パスと結合を評価するときに、副問合せと本体の両方をいっしょに考慮させてしまいます。

> **関連項目：** 副問合せのネスト解除における危険性については、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

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の場合、等価述語が定義されていれば、CBO（リリース 8.1.6 以上）は外部結合列から駆動できます。

ビュー内での外部結合は、外部結合のパフォーマンスに与える影響が読めないため、問題が発生しやすくなります。

中間結果の格納

中間の表、すなわちステージング表がリレーショナル・データベース・システムで比較的よく利用されるのは、それらの表に中間結果を一時的に格納するためです。これは、多くのアプリケーションで役に立つものですが、作成するにはさらにリソースが必要になります。したがって、これらの表による利益が、作成のコストに見合うものかどうかを常に考慮してください。ステージング表は、その情報が何回も再利用されない場合は、作成しないようにしてください。

他の考慮事項

- ステージング表に中間結果を格納することで、アプリケーション・パフォーマンスを向上できる場合があります。一般に、中間結果が、引き続きその後の複数の問合せで使用可能であれば、その結果をステージング表に格納する価値があります。中間結果の再利用により、複雑な文を用いて何度もデータを取り出すことをしないで済む利益は、中間結果をマテリアライズするコストを上回ります。
- 長く複雑な問合せは、理解し、最適化することが困難です。ステージング表を用いることで、複雑な SQL 文をいくつかの小さい文に分解することができ、このとき、各ステップの結果を格納します。
- マテリアライズド・ビューを使用することを検討してください。マテリアライズド・ビューは、ファクト表やディメンション表からの集計データや結合データを格納する、あらかじめ計算済の表です。

> **関連項目：** マテリアライズド・ビューの使用方法に関する詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

索引の再構成

多くの場合は、索引を再構成すると、パフォーマンスが向上します。これには、次の内容が含まれます。

- 非選択的な索引を削除して、DML の速度を上げます。
- パフォーマンス重視のアクセス・パスの索引を作成します。
- 既存の連結索引で列を並び換えることを考慮します。
- 索引に列を追加して、選択性を向上します。

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、索引を多く作成すれば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 名のプログラマが別々に索引を作成すると、アプリケーションのパフォーマンス改善は望めません。

トリガーおよび制約の変更または無効化

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ぶので、トリガーを変更または使用禁止にする必要がある場合があります。

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。

- 導出された値を用意しておきます。応答時間重視のコードでは、GROUP BY の使用を回避します。
- データ設計を検討します。変更によりパフォーマンスの向上が見込める場合は、システムの設計を変更します。
- 必要に応じて、パーティション化を考慮します。

実行計画の長期的な保持

格納されている統計または SQL 実行計画を使用すると、SQL 文の既存の実行計画を長期的に保持できます。表のオプティマイザの統計を格納すると、その表を参照するすべての SQL 文にその統計が適用されます。実行計画を格納すると（すなわち、プラン・スタビリティ）、単一の SQL 文の計画が保持されます。統計およびストアド・プランの両方が SQL 文に対して使用可能な場合は、オプティマイザはストアド・プランのほうを使用します。

関連項目：

- 第 3 章「オプティマイザ統計の収集」
- 第 7 章「プラン・スタビリティの使用方法」

データへのアクセスを最小限に削減

アプリケーションから各行へのアクセスを、可能な限り 1 回のみにします。そうすることで、ネットワークの通信量が削減され、データベースの負荷が軽減されます。次を実行することを考慮してください。

- CASE 文による複数のスキャンの結合
- RETURNING 句を持つ DML の使用
- 1 つの文での必要なすべてのデータの変更

CASE 文による複数のスキャンの結合

多くの場合、様々な表セットで異なった集計を行う必要があります。通常、この計算は、表に複数のスキャンを行って処理されますが、1 回の単一のスキャンですべての集計を計算すると簡単です。n-1 回のスキャンを排除することで、パフォーマンスを大幅に向上できます。

複数のスキャンを 1 つのスキャンに結合するには、各スキャンの WHERE 条件の内容を CASE 文の中に移動します。CASE 文は、集計対象のデータをフィルタにかけます。各集計では、データを取り出す別の列があっても構いません。

次の例では、収入が毎月 2000 より少ない社員、2000 ～ 4000 の社員、4000 を超える社員の数を問い合せています。これは、次の 3 つの問合せで行うことができます。

```
SELECT COUNT (*)
FROM employees
WHERE salary < 2000;

SELECT COUNT (*)
FROM employees
WHERE salary BETWEEN 2000 AND 4000;

SELECT COUNT (*)
FROM employees
WHERE salary>4000;
```

しかし、1 つの文で問合せ全体を実行するほうが効率的です。各数値は 1 つの列として計算されます。count ファンクションは、CASE 文によるフィルタを使用して、条件が一致する行のみを数えます。次に例を示します。

```
SELECT COUNT (CASE WHEN salary < 2000
                   THEN 1 ELSE null END) count1,
       COUNT (CASE WHEN salary BETWEEN 2001 AND 4000
                   THEN 1 ELSE null END) count2,
       COUNT (CASE WHEN salary > 4000
                   THEN 1 ELSE null END) count3
  FROM employees;
```

これは、きわめて単純な例です。範囲が重なっていたり、集計の関数が異なっていることもあります。

RETURNING 句を持つ DML の使用

適時、INSERT、UPDATE または DELETE... RETURNING を使用して、1 回のコールでデータを選択および変更します。この技法は、データベースのコール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 の各文の構文については、『Oracle9*i* SQL リファレンス』を参照してください。

1 つの文での必要なすべてのデータの変更

可能であれば、配列処理を使用します。つまり、バインド変数の値の配列が繰返し実行のために Oracle に渡されます。これは、あるセットの複数行が同じ操作の対象である場合のくり返し処理に適しています。

次に例を示します。

```
BEGIN
 FOR pos_rec IN (SELECT *
   FROM order_positions
   WHERE order_id = :id) LOOP
      DELETE FROM order_positions
      WHERE order_id = pos_rec.order_id AND
        order_position = pos_rec.order_position;
 END LOOP;
 DELETE FROM orders
 WHERE order_id = :id;
END;
```

別の方法として、orders に対するカスケード制約を定義できます。前述の例では、1 つの SELECT に対して *n* 個の DELETE が実行されます。orders 表に対する DELETE 要求として、DELETE FROM orders WHERE order_id = :id に対して DELETE を発行すると、データベースは、1 回の DELETE 文で複数の行を自動的に削除します。

> **関連項目：** 分散問合せをチューニングする方法の詳細は、『Oracle9*i* データベース管理者ガイド』または『Oracle9*i* Heterogeneous Connectivity Administrator's Guide』を参照してください。

7

プラン・スタビリティの使用方法

この章では、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持する方法を説明します。

この章には次の項があります。

実行計画を保持するためのプラン・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ても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。このような変更には、オプティマイザ統計の変更、オプティマイザ・モード設定の変更および SORT_AREA_SIZE や BITMAP_MERGE_AREA_SIZE などのメモリー構造のサイズに影響するパラメータの変更があります。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が変化してしまう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・アウトラインに保持します。Oracle では、1 つまたはすべての SQL 文についてパブリックまたはプライベート・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、オプティマイザはアウトラインから同じ実行計画を生成します。アウトラインをグループ化してカテゴリに分け、Oracle が使用するカテゴリを制御することによって、アウトラインの管理と配置を単純化できます。

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かかわらず一貫しています。また、ストアド・アウトラインを使用すると、以降の Oracle リリースでオプティマイザが変更されても、生成した実行計画の安定性は保たれます。プラン・スタビリティは、新規の Oracle リリースへアップグレードする際に、ルールベース・オプティマイザからコストベース・オプティマイザへ移行するときにも役立ちます。

注意： 市場を通じて多量に販売する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。

プラン・スタビリティでのヒントの使用

Oracle ではヒントを使用してストアド・プランを記録するので、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・メカニズムが実行計画を制御する程度によって決まります。

SQL テキストは、そのストアド・アウトラインと 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。

関連項目： リテラルをシステム生成のバインド変数に置き換えて、SQL を共有するように類似文を設定できます。これは、CREATE OUTLINE 文でなく CREATE_STORED_OUTLINES パラメータを使用してアウトラインが生成されている場合のプラン・スタビリティに限り有効です。さらに、CURSOR_SHARING パラメータを SIMILAR または FORCE に設定してアウトラインを作成してあり、アウトラインを使用するときにも、そのパラメータを SIMILAR または FORCE に設定する必要があります。詳細は、第 14 章「メモリーの構成と使用方法」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、実行計画の保持に依存します。しかし、多くの環境では、日付やオーダー番号などのデータ・タイプの属性はすぐに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、データ特性の変更につれて、パフォーマンスが低下していく結果となります。

つまり、動的な環境では、計画の保持に依存するという技法は、コストベースの最適化の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、データ特性の変更への適合性を持つオプティマイザの利点とのバランスを考慮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 は実行計画そのものではなく実行計画への入力を使用して、アウトラインを生成します。

注意： Oracle は、SYS 表領域に USER_OUTLINES ビューと USER_OUTLINE_HINTS ビューを、それぞれ OL$ 表と OL$HINTS 表のデータに基づいて作成します。OL$、OL$HINTS および OL$NODES 表の直接操作は禁止されています。

SQL 文にヒントを組み込むことはできますが、その結果が Oracle によるアウトラインの使用方法に影響することはありません。Oracle は、ヒントを使用して修正された SQL 文を、アウトラインに格納されている元の SQL 文とは異なるものとして認識します。

アウトラインの格納

アウトライン・データは、OL$、OL$HINTS および OL$NODES の各表に格納します。アウトラインは、削除しなければ無期限に保持されます。

実行計画がキャッシュ内に存在しているかどうかの識別には、SQL テキストのみでなくアウトラインのカテゴリ名が使用されます。アウトラインが実行計画のキャッシュに及ぼす影響はこの点に限定されています。これにより、別のカテゴリの下でコンパイルした SQL 文を実行するときに、Oracle が、それとは別のあるカテゴリの下でコンパイルした実行計画を使用することはありません。

プラン・スタビリティを使用可能にする方法

アウトラインを適切に機能させるためには、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設定が、実行環境全体で一貫したものになっている必要があります。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

提供されるパッケージを使用したストアド・アウトラインの管理

DBMS_OUTLN および DBMS_OUTLN_EDIT パッケージによって、ストアド・アウトラインとそのアウトライン・カテゴリの管理に使用するプロシージャが提供されます。

ユーザーは DBMS_OUTLN を実行するために EXECUTE_CATALOG_ROLE ロールを必要としますが、パブリックには DBMS_OUTLN_EDIT に対する実行権限があります。DBMS_OUTLN_EDIT パッケージは、実行者権限のパッケージです。

> **関連項目：** DBMS_OUTLN および DBMS_OUTLN_EDIT の各プロシージャの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

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、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。そのどちらの場合においても、アウトラインの入力はオプティマイザから導出されます。

パラメータ CREATE_STORED_OUTLINES を TRUE に設定すると、ストアド・アウトラインは Oracle によって自動的に作成されます。パラメータを有効にすると、Oracle はコンパイルされた SQL 文すべてに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。

注意： アウトラインを作成するスキーマに CREATE ANY OUTLINE 権限があることを必ず確認してください。この権限が存在しない場合は、CREATE_STORED_OUTLINE パラメータをオンにしてもアプリケーションの実行後にデータベース内でアウトラインを見つけることはできません。

また、CREATE_STORED_OUTLINES パラメータが有効で、実行中のアプリケーションに多数のリテラル SQL 文がある場合、デフォルトのシステム表領域がすべて使用される可能性があります。その場合は、DBMS_OUTLN.DROP_UNUSED プロシージャを使用して、これらのリテラル SQL のアウトラインを削除します。

DBMS_OUTLN_EDIT パッケージ内の CREATE_EDIT_TABLES プロシージャは、実行者のスキーマ内に表を作成します。これは、プライベート・アウトラインを編集するために必要です。このプロシージャは、DBMS_OUTLN_EDIT に関する EXECUTE 権限があれば呼び出すことができます。

関連項目：

- CREATE OUTLINE 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- DBMS_OUTLN および DBMS_OUTLN_EDIT の各パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。
- ルールベース・オプティマイザからコストベース・オプティマイザに移行する方法の詳細は、7-12 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。
- 使用しやすいグラフィカル・インタフェースでストアド・アウトラインの作成、編集、削除および管理を行う Outline Management および Outline Editor の各ツールの使用方法については、『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

ストアド・アウトラインにカテゴリ名を使用する方法

管理タスクを単純にするために、アウトラインをカテゴリに分類できます。CREATE OUTLINE 文を使用すると、カテゴリを指定できます。指定されていなければ、DEFAULT カテゴリが選択されます。同様に、CREATE_STORED_OUTLINES パラメータでカテゴリの名前を指定できますが、このパラメータに true を指定すると DEFAULT カテゴリ内にアウトラインを作成できます。

CREATE_STORED_OUTLINES パラメータを使用してカテゴリ名を指定すると、その後で作成されるアウトラインはすべて Oracle によってそのカテゴリに割り当てられます。この割当ては、そのカテゴリ名がリセットされるまで変更されません。アウトラインの生成を中断するには、このパラメータを false に設定します。

CREATE_STORED_OUTLINES を true に設定するか、またはカテゴリ名を使用しない CREATE OUTLINE 文を使用した場合は、Oracle はアウトラインを DEFAULT のカテゴリ名に割り当てます。

注意： CREATE_STORED_OUTLINES、USE_STORED_OUTLINES および USE_PRIVATE_OUTLINES は、システムまたはセッション固有のパラメータです。これらは初期化パラメータではありません。これらのパラメータの詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

ストアド・アウトラインの使用および編集

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・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。

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 はアウトラインを default カテゴリで使用します。USE_STORED_OUTLINES パラメータを使用して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名に再設定するか、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ているときにそのカテゴリ内に 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 カテゴリ内の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 文のコンパイルのみを制御します。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。

USE_PRIVATE_OUTLINES パラメータを使用すると、プライベート・アウトラインの使用を制御できます。プライベート・アウトラインは、現行のセッション内のみで見られるアウトラインで、そのデータは現行の解析スキーマ内に常駐します。このアウトラインに対して行った変更はシステム上の他のセッションからは見られず、文のコンパイルへの適用は、現行セッションで USE_PRIVATE_OUTLINES パラメータを指定することによってのみ行えます。編集内容をパブリック領域に保存するように明示的に選択した場合のみ、他のユーザーにも編集内容が表示されます。

オプティマイザは通常、問合せに最適な計画を選択しますが、ユーザーが実行環境に関して理解している事柄と、オプティマイザが従う経験則的方法とが整合しない場合があります。アウトラインを直接編集することで、アプリケーションを変更しなくても SQL 問合せをチューニングできます。

プライベート・アウトラインを作成する場合に、アウトライン・データを保持するためのアウトライン表があらかじめローカル・スキーマに存在しないと、エラーが戻されます。アウトライン表は、DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES プロシージャを使用して作成できます。UTLEDITOL.SQL スクリプトを使用することもできます。

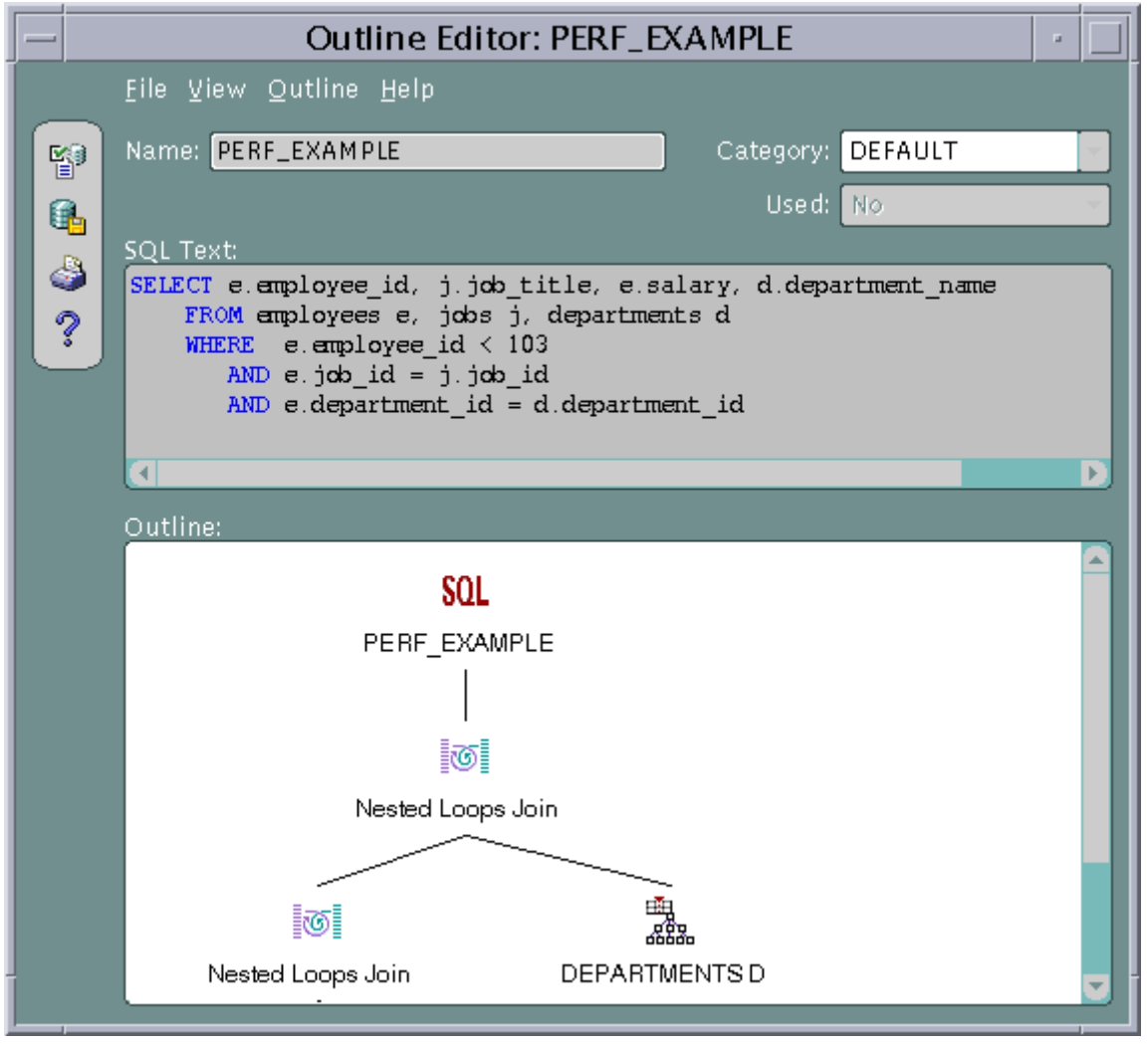
USE_PRIVATE_OUTLINES パラメータを有効にし、アウトラインを使用する SQL 文を発行すると、オプティマイザは、USE_STORED_OUTLINES を有効にした場合に使用されるパブリック領域ではなく、セッションのプライベート領域からアウトラインを取り出します。セッションのプライベート領域にアウトラインが存在しない場合、オプティマイザは、文のコンパイルにアウトラインを使用しません。

CREATE OUTLINE 文はすべて、CREATE ANY OUTLINE 権限が必要です。FROM 句を指定する場合は、SELECT 権限も必要です。この権限は、アウトラインを使用する文に関連する SQL テキストとヒント・テキストを表示する権限を持つユーザーのみに与える必要があります。このロールは、CREATE OUTLINE FROM コマンドに必要です。コマンドの発行者がアウトラインの所有者でもある場合は、このロールは不要です。

編集セッションを開始するときには、USE_PRIVATE_OUTLINES を、編集するアウトラインが属するカテゴリに設定する必要があります。編集を終了するときには、このパラメータを false に設定して、USE_STORED_OUTLINES パラメータに従って、アウトラインを参照するように元に戻しておく必要があります。

Oracle Tuning Pack の GUI Outline Editor を使用してアウトラインを更新できます。図 7-1 は Outline Editor を図解したものです。

図 7-1 Outline Editor



関連項目： アウトライン編集用の GUI ツールの詳細は、『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。

アウトラインの編集例

アウトライン ol1 を編集すると仮定します。手順は次のとおりです。

1. アウトラインを使用する文を実行できるスキーマに接続し、CREATE ANY OUTLINE および SELECT 権限が与えられているかどうかを確認します。
2. DBMS_OUTLN_EDIT.CREATE_EDIT_TABLES プロシージャでアウトライン編集表をローカルに作成します。
3. 次のコードを使用して、編集するアウトラインのクローンをプライベート領域に作成します。

   ```
   CREATE PRIVATE OUTLINE p_ol1 FROM ol1;
   ```

4. Enterprise Manager の Outline Editor でアウトラインを編集するか、ローカルの OL$HINTS 表に問い合せて、適切なヒント・タプルに対して DML を実行する方法で手動でアウトラインを編集します。DBMS_OUTLN_EDIT.CHANGE_JOIN_POS を使用して、結合順序を変更できます。
5. アウトラインを手動で編集する場合は、次のいわゆる認証文でストアド・アウトライン定義を再同期化します。

   ```
   CREATE PRIVATE OUTLINE p_ol1 FROM PRIVATE p_ol1;
   ```

   これは、DBMS_OUTLN_EDIT.REFRESH_PRIVATE_OUTLINE または ALTER SYSTEM FLUSH SHARED_POOL を使用して行うこともできます。
6. 編集内容をテストします。USE_PRIVATE_OUTLINES=TRUE と設定し、アウトライン文を発行するか、文で EXPLAIN PLAN を実行します。
7. パブリックで使用するためにこれらの編集内容を保持する場合は、次の文で編集内容を公開します。

   ```
   CREATE OR REPLACE OUTLINE ol1 FROM PRIVATE p_ol1;
   ```

8. 次のように設定してプライベート・アウトラインとしての使用を無効にします。

   ```
   USE_PRIVATE_OUTLINES=FALSE
   ```

関連項目：

- SQL 構文については、『Oracle9*i* SQL リファレンス』を参照してください。
- アウトラインを編集するための GUI ツールの詳細は、『Oracle Tuning Pack によるデータベース・チューニング』を参照してください。
- DBMS_OUTLN および DBMS_OUTLN_EDIT の各パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

アウトラインが使用されているかどうかを知る方法

アウトラインが V$SQL で使用されているかどうかをテストできます。SQL 文で OUTLINE_CATEGORY 列の問合せを行います。アウトラインが適用されている場合は、そのアウトラインが属しているカテゴリが列に挿入されます。適用されていない場合は、NULL になります。OUTLINE_SID 列は、この特定のカーソルがパブリック・アウトラインを使用しているか（値は 0）、プライベート・アウトラインを使用しているか（そのアウトラインを使用しているセッションの SID）を知らせます。

次に例を示します。

```
SELECT OUTLINE_CATEGORY, OUTLINE_SID
FROM V$SQL
WHERE SQL_TEXT LIKE 'SELECT COUNT(*) FROM emp%';
```

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
- ALL_OUTLINES
- ALL_OUTLINE_HINTS
- DBA_OUTLINES
- DBA_OUTLINE_HINTS

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取得するには、次の構文を使用します。

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

その結果、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 に対して生成されたすべてのヒントを表示するには、次の構文を使用します。

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

注意： 必要な場合は、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を使用できます。詳細は、7-11 ページの「アウトライン表の移動」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 OL$ 表と OL$HINTS 表のデータに基づいて作成します。OUTLN と呼ばれるスキーマを使用して、SYS 表領域にこれらの表と OL$NODES 表も作成します。アウトラインが SYS 表領域の領域を過剰に使用している場合は、アウトラインを移動できます。そのためには、次の手順を使用して別の表領域を作成し、そこにアウトライン表を移動します。

1. CREATE_STORED_OUTLINES パラメータがオンであり、かつ、実行中のアプリケーションに多数のリテラル SQL 文がある場合、デフォルトのシステム表領域すべてが使用される可能性があります。その場合は、DBMS_OUTLN.DROP_UNUSED プロシージャを使用して、これらのリテラル SQL アウトラインを削除します。

   関連項目： DBMS_OUTLN パッケージの使用方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

2. OL$ 表、OL$HINTS 表および OL$NODES 表をエクスポートします。

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

3. 以前の OL$ 表、OL$HINTS 表および OL$NODES 表を削除します。

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

4. 表に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;
   ```

5. 次の文を入力してください。

   ```
   ALTER USER OUTLN DEFALUT TABLESPACE outln_ts;
   ```

6. OL$ 表、OL$HINTS 表および OL$NODES 表をインポートします。

```
IMP OUTLN/outln_password
FILE=exp_file TABLES = 'OL$' 'OL$HINTS' 'OL$NODES'
```

IMPORT 文によって、OUTLN という名前のスキーマに OL$ 表、OL$HINTS 表および OL$NODES 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配置されます。

注意： Oracle8*i* アウトラインを Oracle9*i* データベースにインポートする場合、DBMS_OUTLN.UPDATE_SIGNATURES プロシージャを実行してください。そうすると、システム上のすべてのアウトラインの署名が Oracle9*i* セマンティクスとの互換性を持つように更新されます。この手順を実行しないと、Oracle8*i* データベースからのアウトラインは使用されません。

コストベース・オプティマイザでのプラン・スタビリティの使用

この項では、コストベース・オプ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手順を説明します。プラン・スタビリティは、システムが目標としている、パフォーマンスの良い実行計画を保ちながら、一方で、残りの SQL 文に対するコストベース・オプティマイザの新機能の利点も利用する手段を提供します。

この項で説明する項目は次のとおりです。

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

ルールベース・オプ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持することによって、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・オプ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コストベースの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、次のプロセスを実行します。

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. アウトラインを作成するスキーマに CREATE ANY OUTLINE 権限があることを確認します。たとえば、SYS からは次のようになります。

   ```
   GRANT CREATE ANY OUTLINE TO user-name
   ```

2. 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;
   ```

3. 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。

4. アウトライン生成を中断します。

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

5. DBMS_STATS パッケージを使用して統計を収集します。

6. パラメータ OPTIMIZER_MODE を CHOOSE に変更します。

7. 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;
   ```

8. 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： 手順 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、以前の問合せの動作が獲得される可能性があります。その場合は、オプティマイザ・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、オプティマイザ・モードを再び CHOOSE に戻します。

アップグレードとコストベース・オプティマイザ

コストベースの最適化を使用していて、新しい Oracle リリースにアップグレードする場合は、オプティマイザの変更に伴って変更された実行計画がいくつかの SQL 文に存在する可能性が常にあります。このような変更によりパフォーマンスが向上しますが、アプリケーションによっては、パフォーマンスがすでに十分であるため、動作の変更は不要なリスクと考える場合もあります。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こともできますし、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格納されていたアウトラインをバックアップとして使用することもできます。

バックアップとして使用する場合は、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ができます。

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・システムを用意すれば、アップグレードと合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 カテゴリにアウトラインを作成した後、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なく、選択したアウトラインのみを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' 'OL$NODES'
QUERY='WHERE CATEGORY="problemcat"'
```

8

ルールベース・オプティマイザの使用

この章では、Oracle のルールベース・オプティマイザ（RBO）について説明します。通常は、コストベースのアプローチが使用されます。ルールベースのアプローチは、下位互換性のために使用します。

注意： コストベースの最適化を使用することを強くお薦めします。ルールベースの最適化は、将来のリリースでは使用できなくなる可能性があります。

関連項目： 第 1 章「オプティマイザの概要」

この章には次の項があります。

ルールベース・オプティマイザ（RBO）の概要

Oracle では、ルールベース・オプティマイザ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・オプティマイザ（CBO）を使用して作成することをお薦めします。また、CBO では DSS のために拡張された機能がサポートされているため、データ・ウェアハウスのアプリケーションにも CBO を使用することをお薦めします。パーティション表、改良されたスター・クエリー処理およびマテリアライズド・ビューなど多数の新しいパフォーマンス機能は、CBO でのみ使用可能です。

注意： 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、オプティマイザのルールに基づいて SQL 文を慎重にチューニングしている場合は、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、RBO を引き続き使用することもできます。

サード・パーティ・ベンダーが提供するアプリケーションを使用している場合は、ベンダーに確認して、そのアプリケーションに最も適している最適化のタイプを判断してください。

`OPTIMIZER_MODE=CHOOSE` の場合に統計が存在せず、かつ使用している SQL 文にヒントを追加していないと、RBO が使用されます。リレーショナル・データとオブジェクト・タイプの両方にアクセスするのに RBO が使用できます。`OPTIMIZER_MODE=FIRST_ROWS`、`FIRST_ROWS_`*n* または `ALL_ROWS` に設定されていて、統計が存在しない場合は、CBO はデフォルトの統計を使用します。コストベースのアプローチを使用するために既存のアプリケーションを移行します。

単純に統計を収集することによって、試験的に CBO 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` 句の値を `RULE` に設定することで、RBO 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、この値を使用することもできます。

関連項目： 統計を収集する方法の説明は、第 3 章「オプティマイザ統計の収集」を参照してください。

RBO のアクセス・パス

RBO を使用する場合、オプティマイザは、使用可能なアクセス・パスおよび次に示すアクセス・パスの順位に基づいて実行計画を選択します。Oracle によるアクセス・パスの順位付けには経験則が使用されています。SQL 文を実行する方法が複数存在する場合、RBO は常に順位の低い操作を使用します。通常、順位の低い操作は、順位の高い構成に関連付けられた操作よりも先に実行されます。

次のリストで、アクセス・パスとその順位を示します。

RBO パス 1: ROWID による単一行

RBO パス 2: クラスタ結合による単一行

RBO パス 3: 一意キーまたは主キーを持つハッシュ・クラスタ・キーによる単一行

RBO パス 4: 一意キーまたは主キーによる単一行

RBO パス 5: クラスタ結合

RBO パス 6: ハッシュ・クラスタ・キー

RBO パス 7: 索引クラスタ・キー

RBO パス 8: コンポジット索引

RBO パス 9: 単一列索引

RBO パス 10: 索引付けされた列の境界レンジ・スキャン

RBO パス 11: 索引付けされた列の非有界レンジ・スキャン

RBO パス 12: ソート / マージ結合

RBO パス 13: 索引付けされた列の MAX または MIN

RBO パス 14: 索引付けされた列における ORDER BY

RBO パス 15: 全表スキャン

RBO アクセス・パスの詳細

次の各項で、アクセス・パスおよびアクセス・パスを使用できる場合を説明し、アクセス・パスに対して EXPLAIN PLAN 文が生成する出力を示します。

RBO パス 1: ROWID による単一行

このアクセス・パスは、文の WHERE 句が、ROWID によって選択された行または Oracle プリコンパイラによってサポートされた CURRENT OF CURSOR の埋込み SQL 構文を使用する行を識別する場合にのみ使用可能です。この文を実行するために Oracle は、ROWID を用いて表にアクセスします。

次に例を示します。

```
SELECT * FROM emp WHERE ROWID = 'AAAA7bAA5AAAA1UAAA';
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
    TABLE ACCESS          BY ROWID       EMP
```

RBO パス 2: クラスタ結合による単一行

このアクセス・パスは、次の両方の条件が真の場合に同一のクラスタに格納されている表を結合する文でのみ使用可能です。

- 文の WHERE 句に、1 つの表にあるクラスタ・キーの各列と別の表の対応する列との等価条件が存在します。
- 文の WHERE 句に、結合が単独の行のみを戻すことを保証する条件が存在します。このような条件は、一意キーまたは主キーの列の等価条件である場合がほとんどです。

これらの条件は、AND 演算子と組み合されている必要があります。この文を実行するために Oracle は、ネステッド・ループ操作を実行します。

関連項目： 1-51 ページ「ネステッド・ループ外部結合」

たとえば、次の文では、emp 表と dept 表は deptno 列にクラスタ化されており、empno 列は emp 表の主キーです。

```
SELECT *
  FROM emp, dept
  WHERE emp.deptno = dept.deptno
    AND emp.empno = 7900;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
    NESTED LOOPS
       TABLE ACCESS       BY ROWID       EMP
       INDEX              UNIQUE SCAN    PK_EMP
       TABLE ACCESS       CLUSTER        DEPT
```

pk_emp は、主キーを規定する索引の名前です。

RBO パス 3: 一意キーまたは主キーを持つハッシュ・クラスタ・キーによる単一行

このアクセス・パスは、次の両方の条件が真である場合に使用可能です。

- 文の WHERE 句によって、ハッシュ・クラスタ・キーのすべての列が等価条件で使用されます。コンポジット・クラスタ・キーでは、等価条件が AND 演算子と組み合されている必要があります。
- ハッシュ・クラスタ・キーを構成する列が一意キーまたは主キーも構成しているため、この文が単独の行のみを戻すことが保証されています。

この文を実行するために、Oracle は、文に指定されているハッシュ・クラスタ・キー値にクラスタのハッシュ関数を適用してハッシュ値を取得します。次に、Oracle はハッシュ値を使用して表のハッシュ・スキャンを実行します。

次に例を示します。

次の文では、orders 表と line_items 表はハッシュ・クラスタに格納されており、orderno 列は orders 表のクラスタ・キーであると同時に主キーです。

```
SELECT *
    FROM orders
    WHERE orderno = 65118968;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
    TABLE ACCESS          HASH           ORDERS
```

RBO パス 4: 一意キーまたは主キーによる単一行

このアクセス・パスは、文の WHERE 句によって一意キーまたは主キーのすべての列が等価条件で使用される場合に使用可能です。コンポジット・キーでは、等価条件が AND 演算子と組み合されている必要があります。この文を実行するために Oracle は、一意キーまたは主キーの索引において一意スキャンを実行して単独の ROWID を取り出し、次に該当の ROWID を使用して表にアクセスします。

次に例を示します。

次の文では、empno 列は emp 表の主キーです。

```
SELECT *
    FROM emp
    WHERE empno = 7900;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 UNIQUE SCAN    PK_EMP
```

pk_emp は、主キーを規定する索引の名前です。

RBO パス 5: クラスタ結合

このアクセス・パスは、1 つの表のクラスタ・キーの各列と別の表の対応する列との等価条件が文の WHERE 句に含まれている場合に、同一のクラスタに格納された表を結合する文で使用可能です。コンポジット・クラスタ・キーでは、等価条件が AND 演算子と組み合されている必要があります。この文を実行するために Oracle は、ネステッド・ループ操作を実行します。

関連項目： 1-51 ページ「ネステッド・ループ外部結合」

次に例を示します。

次の文では、emp 表と dept 表は deptno 列にクラスタ化されます。

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

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  NESTED LOOPS
    TABLE ACCESS          FULL           DEPT
    TABLE ACCESS          CLUSTER        EMP
```

RBO パス 6: ハッシュ・クラスタ・キー

このアクセス・パスは、文の WHERE 句によってハッシュ・クラスタ・キーのすべての列が等価条件で使用される場合に使用可能です。コンポジット・クラスタ・キーでは、等価条件が AND 演算子と組み合されている必要があります。この文を実行するために、Oracle は、文に指定されているハッシュ・クラスタ・キー値にクラスタのハッシュ関数を適用してハッシュ値を取得します。次に、ハッシュ値を使用して表のハッシュ・スキャンを実行します。

次に例を示します。次の文では、orders 表と line_items 表はハッシュ・クラスタに格納されており、orderno 列は orders 表のクラスタ・キーです。

```
SELECT *
    FROM line_items
    WHERE orderno = 65118968;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
    TABLE ACCESS          HASH           LINE_ITEMS
```

RBO パス 7: 索引クラスタ・キー

このアクセス・パスは、文の WHERE 句によって索引クラスタ・キーのすべての列が等価条件で使用される場合に使用可能です。コンポジット・クラスタ・キーでは、等価条件が AND 演算子と組み合されている必要があります。

この文を実行するために Oracle は、クラスタ索引に一意スキャンを実行して、指定のクラスタ・キー値を持つ 1 つの行の ROWID を取り出します。そして、Oracle は、取り出された ROWID を使用してクラスタ・スキャンを行い表にアクセスします。同一のクラスタ・キー値を持つ行はすべて同じ場所にまとめて格納されるので、同一のクラスタ・キー値を持つ行すべてを検索するためにクラスタ・スキャンが必要とする ROWID は 1 つのみです。

次に例を示します。

次の文では、emp 表は索引クラスタに格納され、deptno 列はクラスタ・キーです。

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

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            CLUSTER        EMP
    INDEX                 UNIQUE SCAN    PERS_INDEX
```

pers_index は、クラスタ索引の名前です。

RBO パス 8: コンポジット索引

このアクセス・パスは、文の WHERE 句によって AND 演算子と組み合された等価条件でコンポジット索引のすべての列が使用される場合に使用可能です。この文を実行するために Oracle は、索引にレンジ・スキャンを実行して選択された行の ROWID を取り出し、その ROWID を使用して表にアクセスします。

次に例を示します。

次の文では、job 列と deptno 列にコンポジット索引があります。

```
SELECT *
    FROM emp
    WHERE job = 'CLERK'
      AND deptno = 30;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 RANGE SCAN     JOB_DEPTNO_INDEX
```

job_deptno_index は、job 列および deptno 列に設けたコンポジット索引の名前です。

RBO パス 9: 単一列索引

このアクセス・パスは、文の WHERE 句によって 1 つ以上の単一列索引が等価条件で使用される場合に使用可能です。複数の単一列索引では、条件が AND 演算子と組み合されている必要があります。

WHERE 句によって 1 つの索引のみの列が使用される場合、この文を実行するために Oracle は、索引にレンジ・スキャンを実行して選択された行の ROWID を取り出し、検出された ROWID を使用して表にアクセスします。

次に例を示します。

次の文では、emp 表の job 列に索引があります。

```
SELECT *
    FROM emp
    WHERE job = 'ANALYST';
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 RANGE SCAN     JOB_INDEX
```

job_index は、emp.job の索引です。

WHERE によって多数の単一列索引の列が使用される場合、この文を実行するために Oracle は、各索引でレンジ・スキャンを実行し、それぞれの条件を満たす行の ROWID を取り出します。次に、Oracle は取り出した ROWID のセットをマージして、すべての条件を満たす行の ROWID のセットを取得します。そして、取得した ROWID を使用して表にアクセスします。

Oracle は、最大 5 つの索引をマージできます。WHERE 句によって 5 つより多くの単一列索引の列が使用される場合、Oracle はそのうちの 5 つをマージして ROWID 別に表をアクセスします。そして、その結果得られた行を検査して残りの条件を満たしているかどうかを判断した後、結果を戻します。

次の文では、emp 表の job と deptno の両方の列に索引があります。

```
SELECT *
    FROM emp
    WHERE job = 'ANALYST'
      AND deptno = 20;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    AND-EQUAL
      INDEX               RANGE SCAN     JOB_INDEX
      INDEX               RANGE SCAN     DEPTNO_INDEX
```

job_index および deptno_index のスキャンによって取得された ROWID が AND-EQUAL 操作によってマージされた結果が、問合せを満足させる ROWID のセットです。

RBO パス 10: 索引付けされた列の境界レンジ・スキャン

このアクセス・パスは、単一列索引の列またはコンポジット索引の先頭部分を構成する 1 つ以上の列のいずれかを使用する条件が文の WHERE 句に含まれている場合に使用可能です。

```
column = expr

column >[=] expr AND column <[=] expr

column BETWEEN expr AND expr

column LIKE 'c%'
```

これらの各条件によって、文がアクセスする索引付けされた値の境界範囲が指定されます。この範囲を境界と呼ぶのは、これらの条件によって最小値と最大値の両方が指定されるためです。このような文を実行する場合、Oracle は索引にレンジ・スキャンを実行し、ROWID を用いて表をアクセスします。

このアクセス・パスは、式 *expr* が索引付けされた列を参照しているときには使用できません。

次に例を示します。

次の文では、emp 表の sal 列に索引があります。

```
SELECT *
    FROM emp
    WHERE sal BETWEEN 2000 AND 3000;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 RANGE SCAN     SAL_INDEX
```

sal_index は、emp.sal の索引の名前です。

次の文では、emp 表の ename 列に索引があります。

```
SELECT *
    FROM emp
    WHERE ename LIKE 'S%';
```

RBO パス 11: 索引付けされた列の非有界レンジ・スキャン

このアクセス・パスは、単一列索引の列またはコンポジット索引の先頭部分の 1 つ以上の列を使用する次のいずれかの条件が文の WHERE 句に含まれている場合に使用可能です。

```
WHERE column >[=] expr

WHERE column <[=] expr
```

これらの各条件によって、文がアクセスする索引値の非有界範囲が指定されます。この範囲を非有界と呼ぶのは、これらの条件によって指定されるのが最小値と最大値の両方ではなくどちらか一方であるためです。このような文を実行する場合、Oracle は索引にレンジ・スキャンを実行し、ROWID 別に表をアクセスします。

次に例を示します。

次の文では、emp 表の sal 列に索引があります。

```
SELECT *
    FROM emp
    WHERE sal > 2000;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 RANGE SCAN     SAL_INDEX
```

次の文では、line_items 表の order 列と line 列にコンポジット索引があります。

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

このアクセス・パスを使用できるのは、索引の先頭部分である order 列が WHERE 句によって使用されているためです。

このアクセス・パスは、order 列と line 列にコンポジット索引が存在する次の文では使用できません。

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

このアクセス・パスを使用できないのは、索引の先頭部分ではない line 列のみが WHERE 句によって使用されているためです。

RBO パス 12: ソート / マージ結合

このアクセス・パスは、文の WHERE 句によって各表からの列が等価条件で使用されている場合に、同じクラスタ内には格納されていない表を結合する文に対して使用可能です。このような文を実行するために Oracle は、ソート / マージ操作を使用します。また、結合文を実行するために Oracle は、ネステッド・ループ操作も使用します。

> **関連項目：** これらの操作については、1-39 ページの「結合について」を参照してください。

次に例を示します。

次の文では、emp 表と dept 表は同一のクラスタに格納されていません。

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

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  MERGE JOIN
    SORT                  JOIN
      TABLE ACCESS        FULL           EMP
    SORT                  JOIN
  TABLE ACCESS            FULL           DEPT
```

RBO パス 13: 索引付けされた列の MAX または MIN

このアクセス・パスは、次のすべての条件が真であるときに SELECT 文で使用可能です。

- 問合せは、単一列索引の列またはコンポジット索引の先頭列のいずれかで列の最大値または最小値を選択するのに、MAX 関数または MIN 関数を使用します。索引は、クラスタ索引ではありません。MAX 関数または MIN 関数の引数には、列、定数、加算演算子（+）、連結演算子（||）または CONCAT 関数が使用できます。
- 選択リストにその他の式は存在していません。
- 文には、WHERE 句または GROUP BY 句はありません。

この問合せを実行するために、Oracle は、索引の全体スキャンを実行して最大または最小の索引付けされた値を検索します。この値のみが選択されるので、索引のスキャンの後に Oracle が表をアクセスする必要はありません。

たとえば、次の文では、emp 表の sal 列に索引があります。

```
SELECT MAX(sal) FROM emp;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
  0      SELECT STATEMENT Optimizer=CHOOSE
  1    0  SORT (AGGREGATE)
  2    1    INDEX (FULL SCAN (MIN/MAX)) OF 'SAL_INDEX' (NON-UNIQUE)
```

RBO パス 14: 索引付けされた列における ORDER BY

このアクセス・パスは、次のすべての条件が真であるときに SELECT 文で使用可能です。

- 問合せに、単一列索引の列またはコンポジット索引の先頭部分のいずれかを使用する ORDER BY 句が含まれています。索引はクラスタ索引ではありません。
- ORDER BY 句の索引付けされた列リストの中に NULL が含まれていない列が 1 つ以上存在することを保証する PRIMARY KEY または NOT NULL 整合性制約があります。
- NLS_SORT 初期化パラメータは BINARY に設定されています。

この問合せを実行するために、Oracle は索引のレンジ・スキャンを実行して、選択された行の ROWID をソート順に並べかえたものを取り出します。そして、取得した ROWID を使用して表にアクセスします。

次に例を示します。

次の文では、emp 表の empno 列に主キーがあります。

```
SELECT *
    FROM emp
    ORDER BY empno;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            BY ROWID       EMP
    INDEX                 RANGE SCAN     PK_EMP
```

pk_emp は、主キーを規定する索引の名前です。この主キーによってこの列に NULL が含まれていることが保証されます。

RBO パス 15: 全表スキャン

このアクセス・パスは、FROM 句に SAMPLE または SAMPLE BLOCK が含まれている場合を除いて、WHERE 句条件にかかわりなく、すべての SQL 文で使用可能です。

全表スキャンは、このリストの中で最も順位の低いアクセス・パスです。つまり、全表スキャンの方が高速で実行できるとしても、索引を使用するアクセス・パスが使用可能であれば、RBO は常に索引を使用するアクセス・パスを選択します。

次の条件は、索引アクセス・パスを使用不可にします。

- column1 > column2
- column1 < column2
- column1 >= column2
- column1 <= column2

column1 と *column2* は同じ表内にあります。

- 列 IS NULL
- 列 IS NOT NULL
- 列 NOT IN
- 列 != expr
- 列 LIKE '%pattern'

列に索引が作成されているかどうかは関係ありません。

- expr = expr2

expr は、列に索引が作成されているかどうかにかかわらず、演算子または関数を用いて列を操作する式です。

- NOT EXISTS 副問合せ
- ビュー内の ROWNUM 擬似列
- 索引付けされていない列に関連する条件

これらの制約事項が含まれているだけで、索引アクセス・パスを使用可能にする、その他の事項が含まれていない SQL 文では、全表スキャンを使用する必要があります。

次に例を示します。次の文は、全表スキャンを使用して emp 表にアクセスします。

```
SELECT *
    FROM emp;
```

この文の EXPLAIN PLAN 出力は、次のような形式になります。

```
OPERATION                 OPTIONS        OBJECT_NAME
-------------------------------------------------------
SELECT STATEMENT
  TABLE ACCESS            FULL           EMP
```

RBO を使用した結合の実行計画の選択

注意： 次の考慮事項は、コストベースとルールベース両方のアプローチに適用されます。

- オプティマイザは、2 つ以上の表を結合した結果が最大 1 行の行ソースに限定するかどうかを最初に判断します。オプティマイザは、このような状況を表の UNIQUE 制約および PRIMARY KEY 制約に基づいて認識します。このような状況が存在する場合、オプティマイザはこれらの表を結合順序の最初に並べます。その後で、残りの表の結合を最適化します。
- 外部結合条件を持つ結合文では、外部結合演算子のある表の結合順序は、条件内のその他の表の後にしてください。オプティマイザは、この規則に違反する結合順序を考慮しません。

ルールベースのアプローチでは、オプティマイザが次のステップを実行して、R 個の表を結合する文の実行計画を選択します。

1. オプティマイザは、最初の表がそれぞれ異なる一連の R 個の結合順序を生成します。オプティマイザは、次のアルゴリズムを使用して適用可能な結合順序を生成します。
   - 結合順序の各順位の決定において、オプティマイザは、8-3 ページの「RBO のアクセス・パス」で説明したアクセス・パスの順位に従って、使用可能な最も順位の高いアクセス・パスを使用している表を選択します。オプティマイザは、このステップを繰り返して結合順序の次の順位を決定していきます。
   - 結合順序の決定した各表に対して、オプティマイザは、その表を結合順序が前の表、または行ソースに結合するのに使用する操作も選択します。このことをオプティマイザは、ソート / マージ操作をアクセス・パス 12 と順位を比較しながら、次の規則を適用することによって決定します。
     * 選択された表のアクセス・パスの順位が 11 位またはそれより高い場合、オプティマイザは、結合順序が前の表または行ソースを外部表として使用して、ネステッド・ループ操作を選択します。

- 選択された表のアクセス・パスの順位が 12 位より低く、選択した表と結合順序がその前の表または行ソースとの間に等価結合条件が存在する場合、オプティマイザはソート / マージ結合を選択します。
- 選択された表のアクセス・パスの順位が 12 位より低く、等価結合条件が存在しない場合、オプティマイザは、結合順序が前の表または行ソースを外部表として使用して、ネステッド・ループ操作を選択します。

2. この後、オプティマイザは得られた実行計画の結果セットについて選択を行います。オプティマイザによる選択の目標は、索引スキャンを使用して内部表がアクセスされるネステッド・ループ結合操作の数を最大限にすることです。ネステッド・ループ結合は内部表に何度もアクセスするため、内部表に索引があるとネステッド・ループ結合のパフォーマンスは大きく改善されます。

通常、実行計画を選択するときに、オプティマイザは FROM 句に表が指定される順序については考慮しません。実行計画を選択するのに、オプティマイザは次の規則を適用します。

- オプティマイザは、全表スキャンを使用して内部表をアクセスするネステッド・ループ操作の回数が最も少ない実行計画を選択します。
- 同順位のものが存在する場合は、ソート / マージ操作の回数が最も少ない実行計画を選択します。
- それでもまだ同順位のものが存在する場合は、結合順序が最初の表のアクセス・パスの順位が最も高い実行計画を選択します。
  - 最初の表が単一列索引のアクセス・パスによってアクセスされる計画の中に同順位のものがある場合、オプティマイザは、最もマージされた索引を使用して最初の表がアクセスされる計画を選択します。
  - 最初の表が境界レンジ・スキャンによってアクセスされる計画の中に同順位のものがある場合、オプティマイザは、コンポジット索引の先頭から最も多くの列を使用して最初の表がアクセスされる計画を選択します。
- それでも同順位のものが存在する場合、オプティマイザは、最初の表が問合せの FROM 句の後ろに現れる実行計画を選択します。

RBO を使用するときの文の変換および最適化

SQL は、非常に柔軟性に富んだ問合せ言語であり、多くの場合、同一の目標に到達するために使用できる文が多数存在します。オプティマイザは、ある文と同一の目標を達成する別の文があり、それがより効率的な場合、文を変換します。

この項では、次の項目について説明します。

- RBO を使用するときの OR から複合問合せへの変換
- 代替 SQL 構文の使用

RBO を使用するときの OR から複合問合せへの変換

問合せに、OR 演算子によって組み合された複数の条件を持つ WHERE 句が含まれているときに、その問合せを UNION ALL 集合演算子を使用する同等の複合問合せに変換するとより効率よく実行できる場合は、オプティマイザはその問合せを複合問合せに変換します。

- それぞれの条件において索引アクセス・パスがそれぞれ使用できる場合は、オプティマイザは変換を実行します。オプティマイザは、それぞれ異なる索引を使用して複数回にわたって表にアクセスし、その結果を 1 つにまとめる実行計画を、結果の文に選択します。
- 索引を使用できないために全表スキャンが必要な条件がある場合、オプティマイザはその文を変換しません。オプティマイザは文を実行するために全表スキャンを選択します。Oracle は表内の各行を検査し、それぞれの条件が満たされているかどうかを判断します。

> **関連項目：** アクセス・パスの詳細と、索引でそのアクセス・パスを使用できるようにする方法については、8-3 ページの「RBO のアクセス・パス」および 2-29 ページの「CBO による OR の複合問合せへの変換方法」を参照してください。

RBO では、変換した複合問合せのそれぞれの問合せは、索引を使用して実行できるので、オプティマイザはこれを UNION ALL 変換します。RBO は、2 つの索引スキャンを使用する複合問合せの方が、全表スキャンを使用する元の問合せより高速に実行できると想定します。

代替 SQL 構文の使用

SQL は柔軟性がある言語なので、アプリケーションの要求に応える SQL 文が複数ある可能性があります。2 つの SQL 文が同じ結果を生成するとしても、Oracle では一方が他方よりも処理が速い場合があります。EXPLAIN PLAN 文の結果を使用して、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。

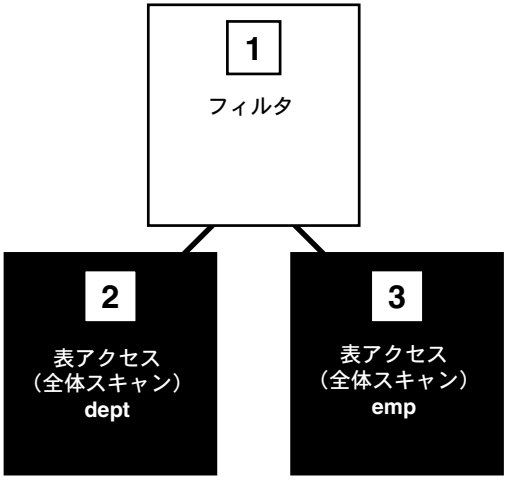
次の例は、同じ機能を実現する 2 つの SQL 文の実行計画を示します。どちらの文も、emp 表に従業員が存在しない dept 表のすべての部門を戻します。それぞれの文は、副問合せを使用して emp 表を検索します。emp 表の deptno 列には索引 deptno_index が存在するものとします。

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);
```

変換された文の実行計画は、図 8-1 の図のようになります。影付きボックスは物理的にデータを取り出すステップを表し、影なしボックスは直前のステップから戻されたデータを処理するステップを表します。

図 8-1 2 つの全表スキャンを行う実行計画



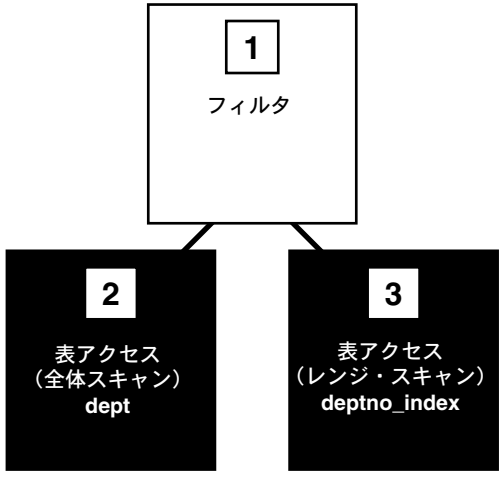
図のステップ 3 の出力は、deptno 列に索引が存在しても、Oracle が emp 表の全表スキャンを行うことによってこの文が実行されることを示します。この全表スキャンは、時間のかかる処理となる可能性があります。emp 表を検索する副問合せには索引を使用するための WHERE 句がないため、Oracle は索引を使用しません。

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);
```

変換された文の実行計画は、図 8-2 の図のようになります。影付きボックスは物理的にデータを取り出すステップを表し、影なしボックスは直前のステップから戻されたデータを処理するステップを表します。

図 8-2 全表スキャンと索引スキャンを行う実行計画



副問合せの WHERE 句で emp 表の deptno 列が参照されているため、索引 deptno_index が使用されます。この索引は、実行計画のステップ 3 で使用されます。deptno_index の索引レンジ・スキャンでは、最初の SQL 文の emp 表の全表スキャンよりも処理時間が短くなります。さらに、最初の SQL 文では、deptno 表のあらゆる deptno について emp 表の全表スキャンが 1 回ずつ実行されます。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 演算子を使用している文が存在する場合は、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してください。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。

注意： 代替 SQL 構文は、ルールベース・オプティマイザでのみ有効です。

第 II 部

SQL 関連のパフォーマンス・ツール

第 II 部では Oracle の SQL 関連パフォーマンス・ツールについて説明します。これらのツールは SQL 文の実行計画を調べ、文をより最適化できるかどうかを判断します。EXPLAIN PLAN の SQL 文、V$SQL_PLAN の問合せ、SQL トレースのいずれかから、実行計画を取得できます。

開発段階では、EXPLAIN PLAN を使用して適切なアクセス計画を判断し、次に大量データ・テストによりそのアクセス計画が最適な計画であるかどうかを検証します。計画を評価する場合、V$SQLAREA と V$SQL_PLAN、Oracle Trace、または SQL トレース機能と TKPROF を使用して、文の実際のリソース使用量を調べます。V$SQL_PLAN ビューの情報は、EXPLAIN PLAN 文の出力に非常に似ています。ただし、EXPLAIN PLAN は文を実行する場合の理論上の計画を示すのに対して、V$SQL_PLAN には実際に使用された計画が含まれています。したがって、V$SQL_PLAN とともに V$SQLAREA の問合せを行うと、その結果は SQL トレースと TKPROF を併用した結果と類似したものになります。

> **関連項目：** 第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」

自動トレース・ツールを使用すると、SQL オプティマイザが使用する実行パスと文の実行統計に関するレポートを自動的に取得できます。このレポートは、これらの文のパフォーマンスを監視したりチューニングする場合に役立ちます。Oracle Trace は、Oracle Server がパフォーマンスとリソースの使用状況データを収集する際に使用する GUI のイベント・ドリブンのデータ収集製品です。

第 II 部には次の章が含まれます。

- 第 9 章「EXPLAIN PLAN の使用方法」
- 第 10 章「SQL トレースおよび TKPROF の使用」
- 第 11 章「SQL*Plus での自動トレースの使用」
- 第 12 章「Oracle Trace の使用」

9

EXPLAIN PLAN の使用方法

この章では、実行計画について紹介し、SQL コマンドの EXPLAIN PLAN を解説し、その出力の解釈方法を説明します。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。

この章には次の項があります。

> **関連項目：** EXPLAIN PLAN の構文については、『Oracle9*i* SQL リファレンス』を参照してください。

EXPLAIN PLAN について

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。

行ソース・ツリーは、実行計画の中核です。行ソース・ツリーは次の情報を示します。

- 文によって参照される表の順序
- 文で言及される各表へのアクセス方法
- 文の結合操作の影響を受ける表の結合方法
- フィルタ、ソート、集計などのデータ操作

PLAN TABLE には、行ソース・ツリーの他、次の情報が含まれています。

- 最適化。各操作のコストとカーディナリティについて。
- パーティション化。アクセスされたパーティションのセットなど。
- パラレル実行。結合入力の配分方法など。

EXPLAIN PLAN の結果により、オプティマイザが特定の実行計画（たとえば、ネステッド・ループ結合）を選択するかどうかを判断できます。また、オプティマイザの決定（たとえば、オプティマイザがハッシュ結合でなくネステッド・ループ結合を選択した理由）について理解し、問合せのパフォーマンスを知るために役立ちます。

注意： Oracle Performance Manager チャートと Oracle SQL Analyze は、EXPLAIN PLAN を自動的に作成して表示できます。EXPLAIN PLAN の使用方法の詳細は、『Oracle Tuning Pack によるデータベース・チューニング』マニュアルを参照してください。

実行計画の変化理由

コストベース・オプティマイザを使用すると、実行計画は基礎となるコストが変化するたびに変化します。EXPLAIN PLAN の出力は、SQL 文の説明段階での実行方法を示します。この方法は、実行環境と EXPLAIN PLAN 環境との違いにより、SQL 文の実際の実行計画とは異なる場合があります。

実行計画は、次の理由により異なります。

- スキーマの相違
- コストの相違

スキーマの相違

- 実行と EXPLAIN PLAN が、異なるデータベース上で起こる場合。
- 文を EXPLAIN するユーザーが、文を実行するユーザーとは異なる場合。2 人のユーザーが同じデータベース内の異なるオブジェクトを指していれば、異なる実行計画が発生します。
- 2 つの操作間でスキーマが変更された場合（多くは索引の変更）。

コストの相違

スキーマが同じであっても、コストが異なる場合にオプティマイザは異なる実行計画を選択する可能性があります。コストに影響を与えるいくつかの要因には次のものがあります。

- データ量と統計
- バインド変数の型
- 初期化パラメータ（グローバル設定またはセッション・レベルでの設定）

排除行数の最少化

EXPLAIN PLAN を調べることにより、次の場合の排除行数を確認できます。

- 全表スキャン
- 選択性のないレンジ・スキャン
- 遅延した述語フィルタ
- 誤った結合順序
- 遅延したフィルタ処理

たとえば、次の EXPLAIN PLAN では、最後のステップは非常に選択性のないレンジ・スキャンです。このレンジ・スキャンは 76563 回実行され、11432983 行にアクセスし、アクセスした行の 99 パーセントを排除して 76563 行を保持します。11432983 行にアクセスした結果、必要な行が 76563 行のみであると判別された理由について考えます。

例 9-1 EXPLAIN PLAN 内の排除行数の確認

```
Rows      Execution Plan
--------  -------------------------------------------------
      12  SORT AGGREGATE
       2   SORT GROUP BY
   76563    NESTED LOOPS
   76575     NESTED LOOPS
      19      TABLE ACCESS FULL CN_PAYRUNS_ALL
   76570      TABLE ACCESS BY INDEX ROWID CN_POSTING_DETAILS_ALL
   76570       INDEX RANGE SCAN (object id 178321)
   76563     TABLE ACCESS BY INDEX ROWID CN_PAYMENT_WORKSHEETS_ALL
11432983      INDEX RANGE SCAN (object id 186024)
```

実行計画以外の考慮事項

実行計画の操作だけでは、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、文による索引の使用が EXPLAIN PLAN 出力で示されたとしても、その文が効率的に機能するとはかぎりません。索引は、非常に非効率的である場合もあります。この場合、次のことを調べる必要があります。

- 使用される索引の列
- その索引の選択性（アクセスされる表の一部）

したがって、EXPLAIN PLAN でアクセス計画を判断し、後からテストによってそれが最適な計画であることを確認するのが最もよい方法です。計画を評価する際は、文の正確なリソース使用量を調べてください。Oracle Trace、または SQL トレース機能と 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：** TKPROF の解釈については、第 10 章「SQL トレースおよび TKPROF の使用」を参照してください。

PLAN_TABLE 出力表の作成

EXPLAIN PLAN 文を発行する前に、出力結果を入れる表が必要です。PLAN_TABLE は、EXPLAIN PLAN 文が実行計画について記述している行を挿入するデフォルトのサンプル出力表です。SQL スクリプト UTLXPLAN.SQL を使用し、使用しているスキーマ内に PLAN_TABLE 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、使用するオペレーティング・システムによって異なります。このスクリプトは UNIX 上では $ORACLE_HOME/rdbms/admin ディレクトリにあります。

たとえば、SQL*Plus セッションで例 9-2 のコマンドを実行し、HR スキーマに PLAN_TABLE を作成します。

例 9-2 PLAN_TABLE の作成

```
CONNECT HR/your_password
@$ORACLE_HOME/RDBMS/ADMIN/UTLXPLAN.SQL

Table created.
```

データベースのバージョンを更新した場合は、列が変更される可能性があるため、PLAN_TABLE 表を削除して再作成することをお薦めします。表を指定する場合は、スクリプトの実行が失敗したり、TKPROF が失敗する場合があります。

別の名前の出力表が必要な場合は、PLAN_TABLE を作成し、RENAME SQL 文でその名前を変更します。

EXPLAIN PLAN の実行

SQL 文を EXPLAIN する場合は、次を使用します。

```
EXPLAIN PLAN FOR
  SQL_Statement
```

次に例を示します。

```
EXPLAIN PLAN FOR
SELECT last_name FROM employees;
```

計画を EXPLAIN したものが PLAN_TABLE 表に挿入されます。PLAN_TABLE から実行計画を選択できるようになります。これは、PLAN_TABLE 内に他の計画がない場合や、最後の文のみを見る場合に便利です。

EXPLAIN PLAN での文の指定

複数の文があるときは、文の識別子を指定し、その識別子で特定の実行計画を識別できます。SET STATEMENT ID を使用する前に、その文と同じ識別子を持つ既存の行を削除してください。

例 9-3 の場合は、bad1 が文の識別子として指定されています。

例 9-3 STATEMENT ID 句での EXPLAIN PLAN の使用方法

```
EXPLAIN PLAN
    SET STATEMENT_ID = 'bad1' FOR
SELECT last_name FROM employees;
```

EXPLAIN PLAN での別の表の指定

INTO 句を指定して、別の表を指定できます。

例 9-4 INTO 句での EXPLAIN PLAN の使用方法

```
EXPLAIN PLAN
   INTO my_plan_table
    FOR
SELECT last_name FROM employees;
```

INTO 句を使用する場合は、文の識別子を指定できます。

```
EXPLAIN PLAN
   INTO my_plan_table
    SET STATEMENT_ID = 'bad1' FOR
SELECT last_name FROM employees;
```

関連項目： EXPLAIN PLAN 構文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

PLAN_TABLE 出力の表示

計画を EXPLAIN した後、Oracle から提供される 2 つのスクリプトを使用して最新の PLAN TABLE 出力を表示します。

- UTLXPLS.SQL - シリアル処理のための PLAN TABLE 出力を表示します。
- UTLXPLP.SQL - パラレル実行列をもつ PLAN TABLE 出力を表示します。

1-19 ページの例 1-4「EXPLAIN PLAN 出力」は、UTLXPLS.SQL スクリプト使用時の計画表出力例です。

文の識別子を指定した場合は、PLAN_TABLE を問い合せるための独自のスクリプトを書くことができます。次に例を示します。

- START WITH ID = 0 および STATEMENT_ID を指定します。
- CONNECT BY 句を使用して親から子へツリーを移動します。結合キーは、STATEMENT_ID = PRIOR STATEMENT_ID と PARENT_ID = PRIOR ID です。
- 疑似列 LEVEL（CONNECT BY に関連付けられている）を使用して子をインデントします。

```
SELECT cardinality "Rows",
       lpad(' ',level-1)||operation||' '||
       options||' '||object_name "Plan"
  FROM PLAN_TABLE

CONNECT BY prior id = parent_id
        AND prior statement_id = statement_id
  START WITH id = 0
        AND statement_id = 'bad1'
  ORDER BY id;

   Rows Plan
------- ------------------------------------------
        SELECT STATEMENT
         TABLE ACCESS FULL EMPLOYEES
```

Rows 列の NULL は、オプティマイザが表に統計を持っていないことを示します。表を ANALYZE すると、次の内容が表示されます。

```
   Rows Plan
------- ------------------------------------------
  16957 SELECT STATEMENT
  16957  TABLE ACCESS FULL EMPLOYEES
```

COST も選択できます。これは、実行計画を比較する場合や、オプティマイザが複数の中からある実行計画を選択した理由を理解する場合に便利です。

注意： これらの単純な例は、再帰的 SQL の場合には有効ではありません。

EXPLAIN PLAN 出力の読み方

この項では、さらに複雑な例を使用して実行計画を説明します。

関連項目： 付録 A「パフォーマンスの例で使用されているスキーマ」

例 9-5 の文は、実行計画の表示に使用されます。

例 9-5 EXPLAIN PLAN を表示する文

```
SELECT lpad(' ',level-1)||operation||' '||options||' '||
        object_name "Plan"
   FROM plan_table
CONNECT BY prior id = parent_id
        AND prior statement_id = statement_id
  START WITH id = 0 AND statement_id = '&1'
  ORDER BY id;
```

EXPLAIN PLAN の例

次は、EXPLAIN PLAN の例を示したものです。

例 9-6 EXPLAIN PLAN の例 1

```
EXPLAIN PLAN SET statement_id = 'example_plan1' FOR
SELECT full_name FROM per_all_people_f
 WHERE UPPER(full_name) LIKE 'Pe%' ;

Plan
----------------------------------------------
SELECT STATEMENT
 TABLE ACCESS FULL PER_ALL_PEOPLE_F
```

この計画は、SELECT 文の実行を示します。表 per_all_people_f は、全表スキャンでアクセスされます。

- 表 per_all_people_f のすべての行がアクセスされ、各行が WHERE 句の条件に基づいて評価されます。
- SELECT 文により、WHERE 句の条件に一致する行が戻されます。

例 9-7 EXPLAIN PLAN の例 2

```
EXPLAIN PLAN SET statement_id = 'example_plan2' FOR
SELECT full_name FROM per_all_people_f
 WHERE full_name LIKE 'Pe%' ;

Plan
---------------------------------------------
SELECT STATEMENT
 TABLE ACCESS BY INDEX ROWID PER_ALL_PEOPLE_F
  INDEX RANGE SCAN PER_PEOPLE_F_N54
```

この計画は、SELECT 文の実行を示します。

- 索引 per_people_f_n54 を使用して、レンジ・スキャン操作が実行されます。
- 表 per_all_people_f が、ROWID によってアクセスされます。ROWID は、WHERE 句の条件に一致するキーに対して、前のステップの索引から取得されます。表にアクセスするときに、レンジ・スキャンで評価できなかった（列が索引の中でなく表の中にあるため）、追加の WHERE 句の条件も評価されます。
- SELECT 文により、WHERE 句の条件を満たす（前のステップで評価された）行が戻されます。

例 9-8 EXPLAIN PLAN の例 3

```
EXPLAIN PLAN SET statement_id = 'example_plan3' FOR
SELECT segment1, segment2, description, inventory_item_id
  FROM mtl_system_items msi
 WHERE segment1 = :b1
   AND segment2 LIKE '%-BOM'
AND NVL(end_date_active,sysdate+1) > SYSDATE ;

Plan
-------------------------------------------------
SELECT STATEMENT
 TABLE ACCESS BY INDEX ROWID MTL_SYSTEM_ITEMS
  INDEX RANGE SCAN MTL_SYSTEM_ITEMS_N8
```

この計画は、SELECT 文の実行を示します。

- 索引 mtl_system_items_n8 を使用して、レンジ・スキャン操作が実行されます。この索引は、（segment1、segment2 および segment3）への索引です。レンジ・スキャンは、次の条件で発生します。

  ```
  segment1 = :b1
  ```

このステップで戻される行は、索引列で評価できるすべての WHERE 句の条件を満たします。したがって、次の条件もこの段階で評価されます。

```
segment2 LIKE '%-BOM'
```

- 表 per_all_people_f は、前のステップの索引から取得された ROWID によってアクセスされます。表にアクセスするとき、レンジ・スキャンで評価できなかった（列が索引の中でなく表の中にあるため）、追加の WHERE 句の条件も評価されます。したがって、次の条件はこの段階で評価されます。

```
NVL(end_date_active,sysdate+1) > SYSDATE
```

- SELECT 文により、WHERE 句の条件を満たす（前のステップで評価された）行が戻されます。

例 9-9 EXPLAIN PLAN の例 4

```
EXPLAIN PLAN SET statement_id = 'example_plan4' FOR
SELECT h.order_number, l.revenue_amount, l.ordered_quantity
  FROM so_headers_all h, so_lines_all l
 WHERE h.customer_id = :b1
   AND h.date_ordered > SYSDATE-30
   AND l.header_id = h.header_id ;

Plan
-------------------------------------------------
SELECT STATEMENT
 NESTED LOOPS
  TABLE ACCESS BY INDEX ROWID SO_HEADERS_ALL
   INDEX RANGE SCAN SO_HEADERS_N1
  TABLE ACCESS BY INDEX ROWID SO_LINES_ALL
   INDEX RANGE SCAN SO_LINES_N1
```

この計画は、SELECT 文の実行を示します。

- 索引 so_headers_n1 を使用して、レンジ・スキャン操作が実行されます。これは、customer_id 上の索引です。レンジ・スキャンは、次の条件で発生します。

```
customer_id = :b1
```

- 表 so_headers_all は、前のステップの索引から取得された ROWID によってアクセスされます。表にアクセスするとき、レンジ・スキャンで評価できなかった（列が索引の中でなく表の中にあるため）、追加の WHERE 句の条件も評価されます。したがって、次の条件はこの段階で評価されます。

```
h.date_ordered > sysdate-30
```

- WHERE 句の条件を満たす so_headers_all のあらゆる行について、レンジ・スキャンが次の条件を使用して so_lines_n1 で実行されます。

  ```
  l.header_id = h.header_id
  ```

- 表 so_lines_all は、前のステップの索引から取得された ROWID からアクセスされます。表にアクセスするとき、レンジ・スキャンで評価できなかった（列が索引の中でなく表の中にあるため）、追加の WHERE 句の条件も評価されます。ここで評価される追加条件はありません。
- SELECT 文により、WHERE 句の条件を満たす（前のステップで評価された）行が戻されます。

EXPLAIN PLAN によるビットマップ索引の表示

ビットマップ索引を使用する索引行ソースは、索引のタイプを示すワード BITMAP とともに EXPLAIN PLAN 出力に表示されます。例 9-10 の問合せおよび計画の次の例を検討します。

例 9-10 ビットマップ検索による EXPLAIN PLAN

```
EXPLAIN PLAN FOR
    SELECT * FROM t
    WHERE c1 = 2
    AND c2 <> 6
    OR c3 BETWEEN 10 AND 20;

SELECT STATEMENT
   TABLE ACCESS T BY INDEX ROWID
      BITMAP CONVERSION TO ROWID
         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 に対応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 に対応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わかります。NULL 減算は、NOT NULL 制約が列に付いていないかぎり、意味上の正確さを保つために必要です。TO ROWIDS オプションは、表アクセスに必要な ROWID を生成するのに使用されます。

注意： ビットマップ・ジョイン・インデックスを使用した問合せは、ビットマップ・ジョイン・インデックスのアクセス・パスを指示します。ビットマップ・ジョイン・インデックスの操作は、ビットマップ索引と同じです。

EXPLAIN PLAN によるパーティション・オブジェクトの表示

EXPLAIN PLAN を使用すると、特定の問合せでのパーティション・オブジェクトに Oracle がアクセスする方法を参照できます。

プルー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。

結合されるいずれかの表の PLAN TABLE の DISTRIBUTION 列に PARTITION（KEY）が存在する場合、結合はパーシャル・パーティション・ワイズ結合を使用して実装されます。パーシャル・パーティション・ワイズ結合が可能なのは、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、表がパラレル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・ワイズ結合を使用して実装されます。フル・パーティション・ワイズ結合が可能なのは、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のみです。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・パーティション化およびハッシュ・パーティション化の表示の例

hire_date で範囲ごとにパーティション化されている次の emp_range 表を参考に、プルーニングの表示方法を例示します。標準 Oracle スキーマの表 emp および dept が存在することを想定しています。

```
CREATE TABLE emp_range
PARTITION BY RANGE(hire_date)
(
    PARTITION emp_p1 VALUES LESS THAN (TO_DATE('1-JAN-1991','DD-MON-YYYY')),
    PARTITION emp_p2 VALUES LESS THAN (TO_DATE('1-JAN-1993','DD-MON-YYYY')),
    PARTITION emp_p3 VALUES LESS THAN (TO_DATE('1-JAN-1995','DD-MON-YYYY')),
    PARTITION emp_p4 VALUES LESS THAN (TO_DATE('1-JAN-1997','DD-MON-YYYY')),
    PARTITION emp_p5 VALUES LESS THAN (TO_DATE('1-JAN-1999','DD-MON-YYYY'))
)
AS SELECT * FROM employees;
```

最初の例では、次の文を検討します。

```
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について繰り返されます。この例では、述語がプルー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（ALL）を対象とします。PLAN_TABLE の PARTITION_START 列と PARTITION_STOP 列は、1 ～ 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。

次の例では、次の文を検討します。

```
EXPLAIN PLAN FOR SELECT * FROM emp_range
WHERE hire_date >= TO_DATE('1-JAN-1995','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.
```

前の例では、パーティション行ソースがパーティション 4 ～ 5 までを反復します。これは、hire_date についての述語を使用してその他のパーティションをプルーニングするためです。

最後に、次の文を検討します。

```
EXPLAIN PLAN FOR SELECT * FROM emp_range
WHERE hire_date < TO_DATE('1-JAN-1991','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.
```

この例では、パーティション 1 のみがアクセスされ、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、プルーニングが可能なのは等価述語か IN リスト述語を使用している場合のみです。

コンポジット・パーティション・オブジェクトでのプルーニング情報の例

Oracle がコンポジット・パーティション・オブジェクトのプルーニング情報を表示する方法を例示するために、hire_date でレンジ・パーティション化され、department_id でハッシュ・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(hire_date) SUBPARTITION BY HASH(department_
id)
SUBPARTITIONS 3
(
    PARTITION emp_p1 VALUES LESS THAN (TO_DATE('1-JAN-1991','DD-MON-YYYY')),
    PARTITION emp_p2 VALUES LESS THAN (TO_DATE('1-JAN-1993','DD-MON-YYYY')),
    PARTITION emp_p3 VALUES LESS THAN (TO_DATE('1-JAN-1995','DD-MON-YYYY')),
    PARTITION emp_p4 VALUES LESS THAN (TO_DATE('1-JAN-1997','DD-MON-YYYY')),
    PARTITION emp_p5 VALUES LESS THAN (TO_DATE('1-JAN-1999','DD-MON-YYYY'))
)
AS SELECT * FROM employees;
```

最初の例では、次の文を検討します。

```
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.
```

この例では、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計画を示します。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。

次の例では、プルー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 までを反復します。各パーティション内では、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 までを反復します。その結果、表アクセス行ソースがサブパーティション 1 ～ 15 にアクセスします。つまり、コンポジット・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することになります。

```
EXPLAIN PLAN FOR SELECT * FROM emp_comp
WHERE hire_date = TO_DATE('15-FEB-1997', '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.
```

この例では、最後のパーティション 5 のみ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 ～ 3 が表示されることになりますが、これは emp_comp 表のサブパーティション 13 ～ 15 までと変換されます。

次に、次の文を検討します。

```
EXPLAIN PLAN FOR SELECT * FROM emp_comp WHERE department_id = 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・ディメンションでプルー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。

最後に、次の文を検討します。

```
VARIABLE dno NUMBER;
EXPLAIN PLAN FOR SELECT * FROM emp_comp WHERE department_id = :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.
```

最後の 2 つの例は、deptno = 20 が department_id = :dno に置き換えられたこと以外は同じです。この最後の場合、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 つのみなので、その行ソースのオプションは SINGLE です。PARTITION_START および PARTITION_STOP は KEY に設定されます。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・ワイズ結合の例

次の例では、emp_range がパーティション化列で結合され、パラレル化されます。dept 表がパーティション化されていないことにより、パーシャル・パーティション・ワイズ結合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;
    Table altered.
ALTER TABLE dept PARALLEL 2;
    Table altered.
```

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.hire_date > TO_DATE('29-JUN-1996','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、PQ Distrib 列にパーティション・キーを示すテキスト PART（KEY）が存在するので、オプティマイザがパーティション・ワイズ結合を選択したことを示しています。

次の例では、emp_comp がハッシュ・パーティション化列の deptno で結合され、パラレル化されています。dept 表がパーティション化されていないことにより、パーシャル・パーティション・ワイズ結合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;
    Table altered.
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-1995','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・ワイズ結合の例

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・ワイズ結合が使用可能になります。PARTITION HASH 行ソースが、PLAN TABLE 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-1996','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 および EXPLAIN PLAN の例

INLIST ITERATOR 操作は、索引が IN リスト述語を実装する場合に、EXPLAIN PLAN 出力に表示されます。次に例を示します。

```
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、計画内の次の操作を反復します。パーティション表およびパーティション索引には 3 種類の IN リスト列が使用可能ですが、これについては次の項で説明します。

IN リスト列が索引列である場合

IN リスト列 empno が索引列で、パーティション列ではない場合、計画は次のようになります（IN リスト演算子は表操作の前に表示されますが、パーティションの操作よりは後に表示されます）。

```
OPERATION        OPTIONS              OBJECT_NAME PARTITION_START PARTITION_STOP
---------------- ------------         ----------- --------------- --------------
SELECT STATEMENT
PARTITION RANGE  ALL                                KEY(INLIST)     KEY(INLIST)
INLIST ITERATOR
TABLE ACCESS     BY LOCAL INDEX ROWID EMP           KEY(INLIST)     KEY(INLIST)
INDEX            RANGE SCAN           EMP_EMPNO     KEY(INLIST)     KEY(INLIST)
```

PARTITION_START 列と PARTITION_STOP 列に 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。

IN リスト列が索引でありパーティション列である場合

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、計画にはパーティション操作の前に INLIST ITERATOR 操作が含まれています。

```
OPERATION        OPTIONS              OBJECT_NAME PARTITION_START PARTITION_STOP
---------------- ------------         ----------- --------------- --------------
SELECT STATEMENT
INLIST ITERATOR
PARTITION RANGE  ITERATOR                           KEY(INLIST)     KEY(INLIST)
TABLE ACCESS     BY LOCAL INDEX ROWID EMP           KEY(INLIST)     KEY(INLIST)
INDEX            RANGE SCAN           EMP_EMPNO     KEY(INLIST)     KEY(INLIST)
```

IN リスト列がパーティション列である場合

empno がパーティション列で、索引が存在しない場合は、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 RANGE   INLIST                       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
```

ドメイン索引および EXPLAIN PLAN の例

また、EXPLAIN PLAN を使用して、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、これらの統計を PLAN_TABLE の OTHER 列に表示します。

たとえば、resume 列にドメイン索引 emp_resume を持つユーザー定義演算子 CONTAINS が表 emp に存在し、emp_resume の索引タイプが演算子 CONTAINS をサポートしている場合に、次の問合せをします。

```
SELECT * FROM emp WHERE CONTAINS(resume, 'Oracle') = 1
```

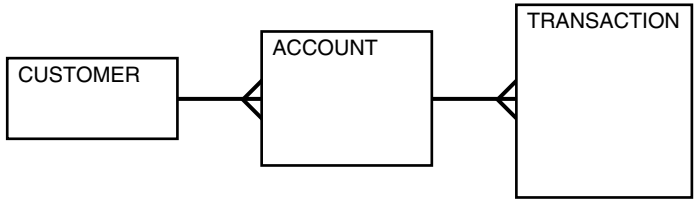
その結果、次のような計画が表示されます。

```
OPERATION            OPTIONS      OBJECT_NAME     OTHER
-----------------    -----------  ------------    ----------------
SELECT STATEMENT
TABLE ACCESS         BY ROWID     EMP
DOMAIN INDEX                      EMP_RESUME      CPU: 300, I/O: 4
```

EXPLAIN PLAN によるパラレル実行の表示

パラレル問合せのチューニングは、パラレルでない問合せのチューニングの場合と同様に駆動表を選択することにより、開始されます。ただし、選択を管理するルールは異なります。パラレルでない問合せの場合は、通常、制限条件が適用された後に最も少ない行が生成される駆動表が最適です。少数の行は、一意でない索引を使用して大きな表に結合されます。たとえば、CUSTOMER、ACCOUNT および TRANSACTION で構成された表階層の場合について考えます。

図 9-1 表階層



ここでは CUSTOMER が最も小さな表、TRANSACTION が最も大きな表です。通常の OLTP 問合せでは、特定の顧客のアカウントに関する取引情報が取得できます。問合せは CUSTOMER 表から駆動されます。この場合の目標は、論理 I/O を最少化することです。それにより、通常、物理 I/O や CPU タイムを含むその他の重要なリソースも最少化されます。

パラレル問合せの場合は、通常、最も大きな表が駆動表として選択されます。これは、最も大きな表でパラレル問合せが有効に利用できるためです。ここでの問合せにパラレル問合せを使用するのは効率的ではありません。各表から最終的にアクセスされる行がごくわずかであるためです。ここで、たとえば前月に特定のタイプの取引を持つすべての顧客を識別する必要が生じた場合を考えます。顧客表には制限条件がないため、問合せは TRANSACTION 表から行ったほうが効率的です。TRANSACTION 表から取り出した行は ACCOUNT 表に結合され、最終的には CUSTOMER 表に結合されます。この場合、ACCOUNT および CUSTOMER 表で使用される索引は、通常、最初の問合せで使用される一意でない索引ではなく、選択性の高い主キーまたは一意の索引になります。TRANSACTION 表は大きく、列に選択性がないため、TRANSACTION 表から行われるパラレル問合せを使用したほうが有効です。

```
Parallel operation
------------------------------
PARALLEL_TO_SERIAL
PARALLEL_TO_PARALLEL
PARALLEL_COMBINED_WITH_PARENT
PARALLEL_FROM_SERIAL
PARALLEL_COMBINED_WITH_PARENT
PARALLEL_TO_PARALLEL
```

通常は、各操作のワークロードがほぼ同じであるかぎり、PARALLEL_TO_PARALLEL 操作によって最適なパフォーマンスが得られます。

ステップが親ステップと同時に実行される場合、PARALLEL_COMBINED_WITH_PARENT 操作が発生します。

PARALLEL_TO_SERIAL 操作は、パラレル操作からの行が問合せコーディネータによって使用される場合、常に発生するステップです。この問合せでは発生しない他の種類の操作には、SERIAL 操作があります。これらの操作が発生するとボトルネックになる可能性があります。パフォーマンスを向上させるために、これらの操作をパラレル操作にすることを考慮します。

パラレル・ステップによって多数の行を生成する場合、それらの行を問合せコーディネータがすみやかに受けつけて処理することができないことがあります。このような状況を改善する方法はありません。

CPU のコスト計算モデル

CPU は、すべてのデータベース操作で使用されます。ほとんどの場合、CPU 使用率は I/O 使用率と同様に重要です。多くの場合、CPU 使用率のみがコストに影響します（メモリー内ソート、ハッシュ、述語評価およびキャッシュされた I/O の場合）。Oracle9*i* ではオプティマイザに新規モデルが提供されますが、このモデルには CPU 使用率のコストが含まれています。コスト計算モデルに CPU 使用率を含めたことにより、より最適な計画が生成できます。

CPU コスト計算モデルに基づく例を次に示します。

```
Cost =  (#SRds * sreadtim +
         #MRds * mreadtim +
         #CPUCycles / cpuspeed ) / sreadtim
```

各項目は次のとおりです。

- #SRDs は、単一ブロック読込みの数です。
- #MRDs は、複数ブロック読込みの数です。
- #CPUCycles は、CPU サイクルの数です*。
- sreadtim は、単一ブロック読込み時間です。
- mreadtim は、複数ブロック読込み時間です。
- cpuspeed は、1 秒あたりの CPU サイクルです。

CPUCycles には、問合せ処理にかかる CPU コスト（純粋な CPU コスト）およびデータの取得にかかる CPU コスト（バッファ・キャッシュ取得の CPU コスト）が含まれます。

このモデルは、シリアル実行の場合は簡単です。パラレル実行の場合は、#SRD、#MRD および #CPUCycles の見積り計算中に、必要な調整が行われます。

EXPLAIN PLAN の制限事項

EXPLAIN PLAN は、日付バインド変数の暗黙的な型変換を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 が実際の実行計画を表していない場合があります。

TKPROF は、SQL 文のテキストからバインド変数の型を判断することはできません。型は 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 10 章「SQL トレースおよび TKPROF の使用」

PLAN_TABLE 列

EXPLAIN PLAN 文に使用される PLAN_TABLE には、表 9-1 にリストした列があります。

表 9-1 PLAN_TABLE 列

列	型	説明
STATEMENT_ID	VARCHAR2(30)	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	DATE	EXPLAIN PLAN 文が発行された日時です。
REMARKS	VARCHAR2(80)	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 文を使用して PLAN_TABLE の行を修正してください。
OPERATION	VARCHAR2(30)	このステップで実行された内部操作の名前です。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、表 9-4 を参照してください。
OPTIONS	VARCHAR2(225)	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、表 9-4 を参照してください。
OBJECT_NODE	VARCHAR2(128)	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	VARCHAR2(30)	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。

表 9-1 PLAN_TABLE 列（続き）

列	型	説明
OBJECT_NAME	VARCHAR2(30)	表または索引の名前です。
OBJECT_INSTANCE	NUMERIC	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。ビューを展開した場合、この数値は予測できません。
OBJECT_TYPE	VARCHAR2(30)	たとえば、索引に対する NON-UNIQUE のような、オブジェクトに関して説明を与える修飾子です。
OPTIMIZER	VARCHAR2(255)	オプティマイザの現行モードです。
SEARCH_COLUMNS	NUMBERIC	現在は使用されていません。
ID	NUMERIC	実行計画の各ステップに割り当てられた番号です。
PARENT_ID	NUMERIC	ID のステップの出力について処理を行う次の実行ステップの ID です。
POSITION	NUMERIC	最初の出力行の場合、この列はオプティマイザが見積った、文を実行するためのコストを示します。その他の行の場合は、同じ親の他の子に対応する相対位置を示します。
COST	NUMERIC	オプ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です。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。この列の値は、CPU_COST 列と IO_COST 列の関数です。
CARDINALITY	NUMERIC	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	NUMERIC	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。
OTHER_TAG	VARCHAR2(255)	OTHER 列の内容です。この列に使用可能な値の詳細は、表 9-2 を参照してください。

表 9-1 PLAN_TABLE 列（続き）

列	型	説明
PARTITION_START	VARCHAR2(255)	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。 *n* は、開始パーティションが SQL コンパイラで識別され、そのパーティション番号が *n* で示されることを意味します。 KEY は、開始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。 ROW REMOVE_LOCATION は、開始パーティション（終了パーティションと同じになります）が実行時に、取得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。 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。
PARTITION_STOP	VARCHAR2(255)	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。 *n* は、終了パーティションが SQL コンパイラで識別され、そのパーティション番号が *n* で示されることを意味します。 KEY は、終了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。 ROW REMOVE_LOCATION は、終了パーティション（開始パーティションと同じになります）が実行時に、取得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。 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。
PARTITION_ID	NUMERIC	PARTITION_START と PARTITION_STOP 列の値の対を計算したステップです。
OTHER	LONG	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。
DISTRIBUTION	VARCHAR2(30)	プロデューサ問合せサーバーからコンシューマ問合せサーバーへ行を分配する方法です。 この列に使用可能な値の詳細は、表 9-3 を参照してください。コンシューマ問合せサーバーおよびプロデューサ問合せサーバーの詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

表 9-1 PLAN_TABLE 列（続き）

列	型	説明
CPU_COST	NUMERIC	オプ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 CPU の操作コスト。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。この列の値は、操作に必要なマシン・サイクル数に比例します。
IO_COST	NUMERIC	オプ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 I/O 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。この列の値は、操作で読み込まれるデータ・ブロックの数に比例します。
TEMP_SPACE	NUMERIC	オプティマイザのコストベース・アプローチで見積もられた、操作で使用される一時領域をバイト単位で表したもの。ルールベース・アプローチを使用する文の場合、または一時領域を使用しない操作の場合、この列は NULL です。

表 9-2 で、OTHER_TAG 列に使用される値を説明します。

表 9-2 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。
PARALLEL_TO_PARALLEL (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、同じパラレル処理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 9-3 で、DISTRIBUTION 列に使用される値を説明します。

表 9-3 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・ワイズ結合、PARALLEL INSERT、パーティション表の CREATE TABLE AS SELECT および CREATE PARTITIONED GLOBAL INDEX に使用します。
HASH	結合キーについて、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。PARALLEL JOIN または PARALLEL 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 9-4 に、EXPLAIN PLAN 文によって生成される OPERATION と OPTION の各組合せおよびその実行計画におけるそれぞれの意味を示します。

表 9-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値

操作	オプション	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。この処理は単一列索引のアクセス・パスに対して使用されます。
BITMAP	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、実際の値を必要としない場合に ROWID の数を戻します。
BITMAP	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取り出します。 FULL SCAN は、開始キーまたは終了キーがない場合にビットマップ索引の全体スキャンを実行します。
BITMAP	MERGE	レンジ・スキャン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します。
BITMAP	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用されます。減算が発生する可能性があるビットマップを作成する非否定述語がある場合にのみ使用できます。9-11 ページの「EXPLAIN PLAN によるビットマップ索引の表示」で例を示します。
BITMAP	OR	2 つのビットマップのビット単位の OR を計算します。
BITMAP	AND	2 つのビットマップのビット単位の AND を計算します。
BITMAP	KEY ITERATION	表の行ソースから各行を取り出し、ビットマップ索引から対応するビットマップを検索します。その後、このビットマップのセットは、次の BITMAP MERGE 操作で 1 つのビットマップにマージされます。
CONNECT BY	.	CONNECT BY 句を含んでいる問合せについて階層順に行を取り出します。
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。

表 9-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値（続き）

操作	オプション	説明
DOMAIN INDEX	.	ドメイン索引からの 1 つ以上の ROWID の取出し。オプション列には、ユーザー定義ドメイン・インデックス・コスト関数から与えられた情報が含まれています。
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のみの取出し。
FOR UPDATE	.	FOR UPDATE 句が含まれている問合せによって選択される行を取り出し、ロックする処理。
HASH JOIN （これらは結合操作です。）	.	2 つのセットの行を結合し、結果を戻す操作。この結合方法は、データのラージ・データ・セット（DSS やバッチなど）の結合に役立ちます。この結合条件は、第 2 の表にアクセスする場合に有効です。 CBO は、2 つの表またはデータ・ソースの小さいほうを使用して、メモリー内に結合キーについてのハッシュ表を作成します。次に、大きいほうの表をスキャンし、ハッシュ表を調べて結合された行を見つけます。
HASH JOIN	ANTI	ハッシュ・アンチ結合。
HASH JOIN	SEMI	ハッシュ・セミ結合。
INDEX （これらはアクセス方法です。）	UNIQUE SCAN	索引からの単一の ROWID の取出し。
INDEX	RANGE SCAN	索引からの 1 つ以上の ROWID の取出し。索引値は昇順でスキャンされます。
INDEX	RANGE SCAN DESCENDING	索引からの 1 つ以上の ROWID の取出し。索引値は降順でスキャンされます。
INDEX	FULL SCAN	スタート・キーおよびストップ・キーがない場合の、索引からのすべての ROWID の取得。索引値は昇順でスキャンされます。
INDEX	FULL SCAN DESCENDING	スタート・キーおよびストップ・キーがない場合の、索引からのすべての ROWID の取得。索引値は降順でスキャンされます。
INDEX	FAST FULL SCAN	マルチブロック READ を使用した全 ROWID（および列の値）の取得。ソート順は定義できません。索引付けされた列に対してのみ、全表スキャンと比較されます。コストベース・オプティマイザでのみ使用可能です。
INDEX	SKIP SCAN	索引内の先頭列を使用しない、連結索引からの ROWID の取得。Oracle9*i* で導入されています。コストベース・オプティマイザでのみ使用可能です。
INLIST ITERATOR	.	IN リスト述語内の各値に対して、計画内の次の操作を反復します。

表 9-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値（続き）

操作	オプション	説明
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 （これらは結合操作です。）	.	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。
MERGE JOIN	OUTER	外部結合文を実行するマージ結合処理。
MERGE JOIN	ANTI	マージ・アンチ結合。
MERGE JOIN	SEMI	マージ・セミ結合。
MERGE JOIN	CARTESIAN	文中に他の表への結合条件を持たない 1 つ以上の表について発生する操作です。結合とともに発生する可能性がありますが、計画内では CARTESIAN とフラグが付かないことがあります。
CONNECT BY	.	CONNECT BY 句を含んでいる問合せに対する、階層順での行の取出し。
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 （これらは結合操作です。）	.	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、条件を満たす行を戻します。この結合方法は、小さいサブセットのデータを結合する場合（OLTP）に役立ちます。この結合条件は、第 2 の表にアクセスする場合に有効です。
NESTED LOOPS	OUTER	外部結合文を実行するネステッド・ループ操作。
PARTITION	SINGLE	1 つのパーティションへのアクセス。
PARTITION	ITERATOR	多数のパーティション（サブセット）へのアクセス。
PARTITION	ALL	すべてのパーティションへのアクセス。
PARTITION	INLIST	IN リスト述語を基準にしたイテレータ。
PARTITION	INVALID	アクセスするよう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。 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、計画内の次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）や同じ数でパーティション化された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区間を示します。パーティションの区間は、PARTITION の PARTITION_START および PARTITION_STOP の値で指定されます。PARTITION_START および PARTITION_STOP の有効な値は、表 9-1 を参照してください。

表 9-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値（続き）

操作	オプション	説明
REMOTE	.	リモート・データベースからのデータの取出し。
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取得される単一行の取出し。
SORT	UNIQUE	行のセットをソートし、重複をなくす処理。
SORT	GROUP BY	GROUP BY 句を持つ問合せで、行のセットを複数のグループにソートする処理。
SORT	JOIN	マージ結合の前に、一連の行をソートする操作。
SORT	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 （これらはアクセス方法です。）	FULL	表のすべての行の取出し。
TABLE ACCESS	SAMPLE	表のサンプル採取された行の取出し。
TABLE ACCESS	CLUSTER	索引クラスタのキーの値に基づいた、表からの行の取出し。
TABLE ACCESS	HASH	ハッシュ・クラスタのキーの値に基づいた、表からの行の取出し。
TABLE ACCESS	BY ROWID RANGE	ROWID 範囲に基づいた表からの行の取出し。
TABLE ACCESS	SAMPLE BY ROWID RANGE	ROWID 範囲に基づいた表からのサンプル行の取出し。
TABLE ACCESS	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。
TABLE ACCESS	BY INDEX ROWID	表がパーティション化されておらず、索引を使用して行が指定される場合。
TABLE ACCESS	BY GLOBAL INDEX ROWID	表がパーティション化されており、グローバル索引のみを使用して行が指定される場合。

表 9-4 EXPLAIN PLAN によって生成される OPERATION 値と OPTION 値（続き）

操作	オプション	説明
TABLE ACCESS	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、行が指定される場合。 パーティション区間 : パーティション区間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 ステップ内の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 および PARTITION_STOP に使用できる値は、NUMBER(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 および PARTITION_STOP に使用できる値は、NUMBER(n)、KEY、ROW REMOVE_LOCATION（TABLE ACCESS のみ）および 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。

関連項目： 第 1 章「オプティマイザの概要」

10

SQL トレースおよび TKPROF の使用

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チューニングに役立つ 2 つの基本的なパフォーマンス診断ツールです。

この章には次の項があります。

SQL トレースと TKPROF について

SQL トレース機能および TKPROF を使用すると、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 を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。

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
- 物理読込みと論理読込み
- 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。

注意： SQL トレースは、特定のセッションにおける統計収集に対してのみ使用可能にするようにしてください。この機能を本番環境全体で使用可能にする必要がある場合は、次を行うことによりパフォーマンスの影響を最小限にとどめます。

- 最低 25% の CPU のアイドル状態を維持すること
- USER_DUMP_DEST 位置に対する適切なディスク領域の維持
- 十分なディスクでのディスク領域のストライプ化

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できます。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。この情報を使用すると、リソースを最も多く使用し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、使用さ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 の使用方法

SQL トレース機能および TKPROF を使用するには、次の手順に従います。

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。

   10-4 ページの「手順 1: トレース・ファイル管理用の初期化パラメータの設定」を参照してください。

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。手順 2 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。

   10-6 ページの「手順 2: SQL トレース機能を使用可能にする方法」を参照してください。

3. 手順 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。手順 3 ではオプションとして、データベースに統計を格納するのに使用できる SQL スクリプトを作成できます。

   10-7 ページの「手順 3: TKPROF によるトレース・ファイルのフォーマット」を参照してください。

4. 手順 3 で作成した出力ファイルを解釈します。

   10-13 ページの「手順 4: TKPROF 出力の解釈」を参照してください。

5. 任意で、手順 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。

   10-18 ページの「手順 5: SQL トレース機能統計の格納」を参照してください。

次の項では、これらの各手順について詳しく説明します。

手順 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 トレース機能を有効にする前に、次のことを行ってください。

1. TIMED_STATISTICS、MAX_DUMP_FILE_SIZE および USER_DUMP_DEST の初期化パラメータ設定をチェックします。表 10-1 を参照してください。

表 10-1 SQL トレース機能を有効にする前にチェックする初期化パラメータ

パラメータ	説明
TIMED_STATISTICS	これにより、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様々な統計の収集が使用可能または使用禁止にできます。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。true にすること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。これは動的パラメータです。これはセッション・パラメータでもあります。
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。これは動的パラメータです。これはセッション・パラメータでもあります。
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。デフォルト値は、使用するオペレーティング・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。これは動的パラメータです。これはセッション・パラメータでもあります。

注意： 初期化パラメータ STATISTICS_LEVEL が TYPICAL または ALL に設定されている場合、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL が BASIC に設定されている場合、TIMED_STATISTICS を TRUE に設定して時間統計の収集を使用可能にする必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を初期化パラメータ・ファイル内で、または ALTER_SYSTEM や ALTER SESSION を使用して明示的に設定した場合、STATISTICS_LEVEL から導出された値は明示的に設定された値によって上書きされます。

---

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、このディレクトリは通常、生成された名前を持つ何百ものファイルですぐに一杯になります。このため、トレース・ファイルを生成元のセッションやプロセスに対応づけることは困難な場合があります。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・ユーザーの場合があります。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。

   関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

手順 2: SQL トレース機能を使用可能にする方法

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、次のいずれかを使用します。

- DBMS_SESSION.SET_SQL_TRACE プロシージャ
- ALTER SESSION SET SQL_TRACE = TRUE;

DBMS_SYSTEM.SET_SQL_TRACE_IN_SESSION プロシージャを使用すると、別のセッションに対して SQL トレースを使用可能にできます。

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 文をチューニングするときにのみ使用可能にし、チューニングが終了してから使用禁止にしてください。

ALTER SESSION 文を挿入するには、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 statistics 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 リファレンス』マニュアルを参照してください。

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。

初期化ファイルの SQL_TRACE 初期化パラメータの値を TRUE に設定すると、インスタンスに対して SQL トレース機能が使用可能になります。

```
SQL_TRACE = TRUE
```

更新済み初期化パラメータ・ファイルを使用してインスタンスを再起動すると、インスタンスに対して SQL トレースが使用可能になり、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場合は、SQL_TRACE パラメータの値を FALSE に設定すると使用禁止にできます。

注意： SQL_TRACE を TRUE に設定すると、サーバーのパフォーマンスに重大な影響を与えることがあります。詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

手順 3: TKPROF によるトレース・ファイルのフォーマット

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受け入れ、フォーマットされた出力ファイルを生成します。TKPROF は、実行計画の生成にも使用できます。

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、次を実行できるようになります。

- 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 つずつ作成できます。
- 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できます。

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         0
Execute      1      0.00       0.00          0          0           0         0
Fetch        1      0.03       0.26          2          2           4        14

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'
```

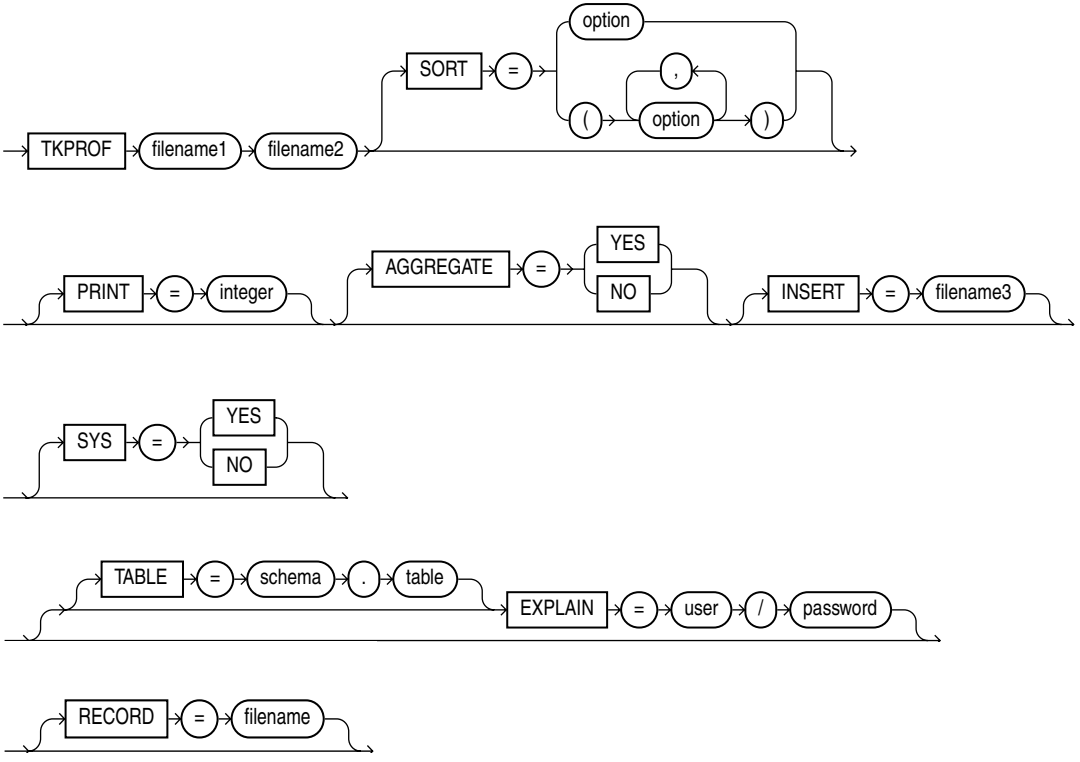
この文では、TKPROF 出力に次の情報が含まれています。

- 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::=



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。TKPROF を実行するときには表 10-2 の引数を使用します。

表 10-2 TKPROF 引数

引数	説明
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	TKPROF が、フォーマット済みの出力を書き出すファイルを指定します。
WAITS	トレース・ファイル内の待機イベントのサマリーを記録するかどうかを指定します。値は YES または NO のいずれかです。
SORTS	トレースした SQL 文のリストを出力ファイルに作成する前に、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 はそれぞれの文のリストを使用順に出力ファイルに作成します。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込みの回数
PRSQRY	解析中の一貫モード・ブロック読込みの回数
PRSCU	解析中の現行モード・ブロック読込みの回数
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込みの回数
EXEQRY	実行中の一貫モード・ブロック読込みの回数
EXECU	実行中の現行モード・ブロック読込みの回数
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

表 10-2 TKPROF 引数（続き）

引数	説明
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込みの回数
FCHQRY	フェッチ中の一貫モード・ブロック読込みの回数
FCHCU	フェッチ中の現行モード・ブロック読込みの回数
FCHROW	フェッチされた行数
PRINT	ソートされた SQL 文の先頭の integer 個のみのリストを作成します。このパラメータを指定しないと、TKPROF はトレースした SQL 文すべてのリストを作成します。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 文に対する挿入データを生成します。
AGGREGATE	AGGREGATE = NO を指定すると、TKPROF は同じ SQL テキストに対する複数のユーザーのデータを集計しません。
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、名前 filename3 を使用して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、つまり再帰的 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 文のリストを作成します。NO の値が指定されると、TKPROF はこれらの SQL 文のリストを作成しません。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。

表 10-2 TKPROF 引数（続き）

引数	説明
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、TKPROF はその表の行をすべて削除し、EXPLAIN PLAN 文（より多くの行を表に書き込む）でその表を使用してから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、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。 指定されたユーザーは、表に対して INSERT、SELECT および DELETE 文を発行できる必要があります。表がまだ存在しない場合は、この指定のユーザーが前述の文に加えて CREATE TABLE 文と DROP TABLE 文も発行できる必要があります。これらの文を発行するための権限については、『Oracle9*i* 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、EXPLAIN の値に指定されている同一のユーザーについて複数のユーザーが同時に TKPROF を実行できます。これらの複数のユーザーが個別に、TABLE に異なる値を指定しておくことで、一時的な PLAN TABLE の処理時に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。 TABLE パラメータを指定せずに EXPLAIN パラメータを使用すると、TKPROF は EXPLAIN パラメータで指定されたユーザーのスキーマにある表 PROF$PLAN_TABLE を使用します。EXPLAIN パラメータを指定せずに TABLE パラメータを使用した場合は、TKPROF が TABLE パラメータを無視します。 PLAN TABLE が存在しない場合、TKPROF では表 PROF$PLAN_TABLE が作成され、最後に削除されます。
EXPLAIN	トレース・ファイルの各 SQL 文の実行計画を判断して、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たユーザーとパスワードを使用して Oracle に接続した後、EXPLAIN PLAN 文を発行して実行計画を判断します。指定されたユーザーは、CREATE SESSION システム権限を持っている必要があります。EXPLAIN オプションが使用されている場合は、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。
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。
WIDTH	EXPLAIN PLAN など、一部の TKPROF 出力の出力行幅を制御する整数です。このパラメータは、TKPROF 出力の後処理に役立ちます。

TKPROF 文の例

この項では、TKPROF の 2 つの使用例を簡単に説明します。TKPROF 出力例の詳細は、10-24 ページの「TKPROF の出力例」を参照してください。

TKPROF の例 1 SORT パラメータと PRINT パラメータの組合せを使用して大規模なトレース・ファイルを処理する場合は、リソースを最も多く使用する文のみを含む 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 ora53269.prf SORT = (PRSDSK, EXEDSK, FCHDSK) PRINT = 10
```

TKPROF の例 2 この例では、TKPROF を実行して、dlsun12_jane_fg_sqlplus_007.trc というトレース・ファイルを取り込み、outputa.prf という名前のフォーマット済みの出力ファイルに書き込みます。

```
TKPROF dlsun12_jane_fg_sqlplus_007.trc OUTPUTA.PRF
EXPLAIN=scott/tiger TABLE=scott.temp_plan_table_a INSERT=STOREA.SQL SYS=NO
SORT=(EXECPU,FCHCPU)
```

この例は、スクリーン上で複数行にわたって表示される可能性があるので、使用し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する必要があります。

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 がユーザー scott として接続され、トレースされた各 SQL 文の実行計画を生成するために EXPLAIN PLAN 文が使用されます。これを使用してアクセス・パスおよび行ソース行数を取得できます。
- TABLE の値によって、TKPROF がスキーマ scott の表 temp_plan_table_a を一時的な PLAN TABLE として使用します。
- INSERT の値によって、TKPROF が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 が指定されていることにより、TKPROF は出力ファイルに再帰的 SQL 文のリストを作成しません。そのため、一時表操作などの内部 Oracle 文は無視されます。
- 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。効率を最大にするためには、SORT パラメータを常に使用してください。

手順 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。

- TKPROF の表形式の統計
- TKPROF のライブラリ・キャッシュ・ミス
- SQL トレースでの文の切捨て
- TKPROF での SQL 文を発行するユーザーの識別
- TKPROF の実行計画
- チューニングする文の決定

TKPROF は非常に有用な分析を提供しますが、効率の最も正確なメジャーは、対象アプリケーションの実際のパフォーマンスです。TKPROF 出力の最後の部分は、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です。

TKPROF の表形式の統計

TKPROF は、SQL トレース機能によって戻される SQL 文の統計のリストを行と列に作成します。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。統計は、次に示す CALL 列の値によって識別されます。表 10-3 を参照してください。

表 10-3 CALL 列の値

CALL の値	意味
PARSE	適切なセキュリティ認可のチェック、および表、列、その他の参照オブジェクトの存在のチェックを行って SQL 文を実行計画に変換します。
EXECUTE	Oracle によって行われる実際の文の実行です。INSERT 文、UPDATE 文および DELETE 文では、データの変更が行われます。SELECT 文では、選択された行が識別されます。
FETCH	問合せを満たす行を取得します。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、全ての文の解析、実行、フェッチについての統計です。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。これは論理 I/O（LIO）とも呼ばれます。表 10-4 を参照してください。

表 10-4 解析、実行およびフェッチの SQL トレース統計

SQL トレース統計	意味
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。TIMED_STATISTICS がオンになっていない場合、この値は 0（ゼロ）になります。
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。TIMED_STATISTICS がオンになっていない場合、この値は 0（ゼロ）になります。
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、一貫モードで取り出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取り出されます。
CURRENT	現行モードで取り出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取り出されます。

処理された行に関する統計は、ROWS 列に表示されます。表 10-5 を参照してください。

表 10-5 ROWS 列の SQL トレース統計

SQL トレース統計	意味
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 つしかないため、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。

統計の精度の解釈

タイミング統計の分解能は 100 分の 1 秒なので、100 分の 1 秒以下のカーソル操作は 正確に計測できません。統計を解読するときには、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読するときには特に注意してください。

再帰的コールについて

ユーザーが発行した SQL 文を実行するために、Oracle は内部的に追加の文を実行する必要があります。このような文を再帰的コールまたは再帰的 SQL 文と呼びます。たとえば、十分な領域のない表に行を挿入しようとすると、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・キャッシュにないため、ディスクから取り出す必要がある場合にも、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。SYS コマンドライン・パラメータを NO に設定して、出力ファイルへの Oracle 内部再帰的コール（たとえば、領域管理）のリスト表示を抑止できます。再帰的 SQL 文の統計は、再帰的コールを発生させた SQL 文のリストでなく、再帰的 SQL 文のリストに表示されます。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、その文自体の統計に加えて、その文を原因とする再帰的コールの統計も合わせて考慮する必要があります。

注意： 再帰的 SQL 統計は、SQL レベルの操作には組み込まれません。ただし、再帰的 SQL 統計は、トリガーなど SQL レベルより下の操作には組み込まれます。詳細は、10-24 ページの「トリガー・トラップの回避」を参照してください。

TKPROF のライブラリ・キャッシュ・ミス

TKPROF 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・ミス回数のリストも作成します。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 はその統計のリストを作成しません。10-7 ページの「TKPROF の出力例」における解析ステップでは、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・ミスは発生しませんでした。

SQL トレースでの文の切捨て

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
```

TKPROF での SQL 文を発行するユーザーの識別

TKPROF は、各 SQL 文を発行したユーザーのユーザー ID を出力します。SQL トレース入力ファイルが複数の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、TKPROF は文を解析した最後のユーザーの ID を出力します。すべてのデータベース・ユーザーのユーザー ID は、列 ALL_USERS.USER_ID のデータ・ディクショナリに表示されます。

TKPROF の実行計画

TKPROF のコマンドラインに 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）のキャッシュがいっぱいになったときの不均衡な量の I/O アクティビティを反映します。チューニングを行うときには、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： 実行計画の解釈に関する詳細は、第 9 章「EXPLAIN PLAN の使用方法」を参照してください。

チューニングする文の決定

CPU リソースまたはディスク・リソースを最も消費する SQL 文を検出する必要があります。TIMED_STATISTICS パラメータがオンになっている場合は、CPU の高いアクティビティを CPU 列で見つけられます。TIMED_STATISTICS がオンになっていない場合は、QUERY 列と CURRENT 列をチェックします。

関連項目： リソースを消費する文の検索例は、10-12 ページの「TKPROF 文の例」を参照してください。

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、読込み一貫性の対象）と現行モード（読込み一貫性の非対象）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。これらのメジャーは、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。高ディスク・アクティビティはディスク列で見つけられます。

次のリストには、ある SQL 文の TKPROF 出力が出力ファイルに表示されるときと同じ状態で示されています。

```
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           0
Execute     11      0.23       0.66          0          3          6           0
Fetch       35      6.70       6.83        100      12326          2         824
------------------------------------------------------------------------------
total       57      7.01       7.67        100      12329          8         826

Misses in library cache during parse: 0
```

7.01 CPU 秒で、824 行を取り出せる場合は、これ以上このトレース出力を検索する必要はありません。事実上、チューニング作業での TKPROF レポートの主な用途は、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。

1 つの文に対して 11 の解析コールが存在していたため、10 の不要な解析コールが作成され、その配列フェッチ操作が実行されたことも確認できます。これは、フェッチで取り出された行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。CPU 時間と経過時間の大きなギャップは、物理 I/O（PIO）の存在を示します。

手順 5: SQL トレース機能統計の格納

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保持し、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、次の内容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、SQL スクリプトを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。これにより、スクリプトが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);
```

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。

表 10-6 の列は、統計が入っている行を識別する際に役立ちます。

表 10-6 統計の行を識別する TKPROF_TABLE 列

列	説明
SQL_STATEMENT	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。この列のデータ型は LONG なので式または WHERE 句条件ではこの列を使用できません。
DATE_OF_INSERT	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	SQL 文が発行された再帰レベルを示します。たとえば、値 0 はユーザーがその文を発行したことを示します。値 1 は、Oracle が値 0 の文（ユーザー発行の文）を処理するための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 *n* は、Oracle がその文を値 *n*-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 が使用します。

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、10-7 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 によって次のような結果が返されます。

```
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 の解釈における誤りの回避

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。

引数トラップの回避

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。EXPLAIN PLAN は、SQL 文からバインド変数の型を判別できないので、型は常に varchar であると想定されます。バインド変数が実際には番号または日付である場合、TKPROF が暗黙的データ変換を行い、その結果、効率の悪い計画が実行される可能性があります。これを回避するには、異なるデータ型を使用して問合せを試みます。

この問題を回避するには、各自で変換を行ってください。

> **関連項目：** TKPROF およびバインド変数の詳細は、9-23 ページの「EXPLAIN PLAN の制限事項」を参照してください。

読込み一貫性トラップの回避

次の例は、読込み一貫性トラップを示しています。コミットされていない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困難です。

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           0
Execute      1      0.00       0.00          0          0          0           0
Fetch        1      0.11       0.21          2        101          0           1
```

```
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・トラップの回避

この例は極端な場合を示しているので、スキーマ・トラップは容易に検出できます。最初は、明らかに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    0
Execute         1       0.02         0.02        0      0       0    0
Fetch           1       0.23         0.30       31     31       3    1

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、問合せが全表スキャンを使用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。

新規トレース・ファイルを生成すると次のデータが与えられます。

```
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で注目する機能の 1 つは、解析コールには 10 ミリ秒の CPU 時間と 20 ミリ秒の経過時間を要する一方で、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。これらの例外的事態が発生するのは、10 ミリ秒というクロック刻みがデータの実行およびフェッチに要する時間と比べて非常に長いためです。このような場合、文を多く実行して統計的に有効な数値を得ることが重要になります。

タイム・トラップの回避

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           0
Execute        1      0.62      19.62       22      526      12           7
Fetch          0      0.00       0.00        0        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 で共有ロックが保持されています。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・トラップの回避

ある文についてレポートされたリソースには、文が処理されていた間に発行されたすべての SQL 用のリソースが含まれます。したがって、これらには、トリガーで使用されるリソースと、他の再帰的 SQL で使用されるリソース（領域割当てで使用されるリソースなど）が含まれます。SQL トレース機能が使用可能になっている場合は、TKPROF が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使用されている場合は、DML 文をチューニングすることは避けてください。

DML 文が予想よりはるかに多くのリソースを消費していると思われる場合は、トリガーと制約についての SQL 文に関連する表をチェックして、トリガーと制約がリソースの使用量を大幅に増やしていないか、調べてください。

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。

TKPROF ヘッダーのサンプル

```
Copyright (c) Oracle Corporation 1979, 1999. 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
********************************************************************************
```

TKPROF 本体のサンプル

```
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
------- ------  -------- ---------- ---------- ---------- ----------  ----------
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
------- ------  -------- ---------- ---------- ---------- ----------  ----------
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'
********************************************************************************
```

TKPROF サマリーのサンプル

```
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 cache during parse: 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.
```

11

SQL*Plus での自動トレースの使用

この章では、SQL*Plus および *i*SQL*Plus 統計レポートにおける自動トレース機能の使用方法を説明します。

この章には次の項があります。

自動トレース・レポートの概要

SQL*Plus では、SQL オプティマイザが使用する実行パス、および文の実行統計に関するレポートを自動的に取得できます。このレポートは、SELECT、DELETE、UPDATE または INSERT などの SQL DML 文が正常に実行された後に生成されます。これらの DML 文のパフォーマンスを監視したりチューニングする場合には、このレポートが役立ちます。

自動トレース・レポートの構成

AUTOTRACE システム変数を設定してレポートを管理できます。表 11-1 を参照してください。

表 11-1 自動トレースの設定

自動トレースの設定	結果
SET AUTOTRACE OFF	AUTOTRACE レポートは生成されません。これはデフォルトです。
SET AUTOTRACE ON EXPLAIN	AUTOTRACE レポートには、オプティマイザ実行パスのみ表示されます。
SET AUTOTRACE ON STATISTICS	AUTOTRACE レポートには、SQL 文実行統計のみ表示されます。
SET AUTOTRACE ON	AUTOTRACE レポートには、オプティマイザ実行パスと SQL 文実行統計の両方が含まれます。
SET AUTOTRACE TRACEONLY	SET AUTOTRACE ON と似ていますが、ユーザーの問合せ出力（ある場合）の印刷を抑止します。STATISTICS が有効の場合は、問合せデータはフェッチされますが印刷はされません。

自動トレース・レポートの必須設定

この機能を使用するには、HR などの PLUSTRACE ロールがユーザーに付与されている必要があります。PLUSTRACE ロールを付与するには、DBA 権限が必要です。

さらに、HR スキーマなどのユーザーのスキーマに PLAN_TABLE 表を作成する必要があります。PLAN_TABLE の作成方法については、9-5 ページの「PLAN_TABLE 出力表の作成」を参照してください。

PLUSTRACE ロールを作成して DBA に付与するには、SQL*Plus セッションで例 11-1 のコマンドを実行します。

例 11-1 PLUSTRACE ロールの作成

```
CONNECT / AS SYSDBA
@$ORACLE_HOME/SQLPLUS/ADMIN/PLUSTRCE.SQL

drop role plustrace;
Role dropped.
create role plustrace;
Role created.
.
grant plustrace to dba with admin option;
Grant succeeded.
```

PLUSTRACE ロールを HR ユーザーに付与するには、SQL*Plus セッションから例 11-2 のコマンドを実行します。

例 11-2 PLUSTRACE ロールの付与

```
CONNECT / AS SYSDBA
GRANT PLUSTRACE TO HR;
Grant succeeded.
```

関連項目：

- ロールの付与および PLAN_TABLE 表の作成の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。
- PLUSTRACE ロール、および SET コマンドの AUTOTRACE 変数の詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。
- ロールの付与については、『Oracle9*i* データベース管理者ガイド』を参照してください。

SQL 文の実行計画

実行計画には、SQL オプティマイザの問合せ実行パスが示されます。実行計画の各行には、シーケンシャル番号が付いています。SQL*Plus は、親操作の行番号も表示します。実行計画の詳細および例は、1-18 ページの「実行計画について」を参照してください。

実行計画出力は、EXPLAIN PLAN コマンドで生成されます。出力フォーマットは設定によって異なります。PLAN_TABLE の列のフォーマットは、COLUMN コマンドで変更できます。たとえば、PARENT_ID 列の表示を停止するには、次のように入力します。

```
COLUMN PARENT_ID NOPRINT
```

デフォルト・フォーマットは、このサイトのプロファイル（たとえば、GLOGIN.SQL）にあります。

関連項目： EXPLAIN PLAN の出力の生成と解釈の詳細は、第 9 章「EXPLAIN PLAN の使用方法」を参照してください。

SQL 文のデータベース統計

統計は、文が実行されるとサーバーにより記録され、文の実行に必要なシステム・リソースを示します。結果には表 11-2 に示した統計が含まれます。

表 11-2 データベース統計

データベース統計名	説明
recursive calls	ユーザー・レベルおよびシステム・レベルの両方で生成された再帰的コールの数。Oracle では、内部処理に使用される表が管理されます。これらの表を変更する必要がある場合、Oracle で内部 SQL 文が内部的に生成されます。この SQL 文により、再帰的コールが生成されます。
db block gets	CURRENT ブロックが要求された回数。
consistent gets	ブロックに対して読込み一貫性が要求された回数。
physical reads	ディスクから読み取られたデータ・ブロックの合計数。この数は、physical reads direct とバッファ・キャッシュへのすべての読込みを合計した値と等しくなります。
redo size	生成された REDO の合計バイト数。
bytes sent via SQL*Net to client	フォアグラウンド・プロセスからクライアントへ送られた合計バイト数。
bytes received via SQL*Net from client	Oracle Net でクライアントから受信した合計バイト数。
SQL*Net roundtrips to/from client	クライアントに送られた Oracle Net メッセージとクライアントから受信した Oracle Net メッセージの合計数。
sorts (memory)	メモリー内で完全に実行され、ディスク書込みを必要としなかったソート操作の数。
sorts (disk)	少なくとも 1 回のディスク書込みを必要としたソート操作の数。
rows processed	操作中に処理された行数

統計でのクライアントとは SQL*Plus です。Oracle Net とは、それがインストールされているかどうかにかかわらず、SQL*Plus とサーバー間の汎用プロセス間通信のことを指します。統計レポートのデフォルト・フォーマットは変更できません。

関連項目：

- データベース統計の完全なリストは、『Oracle9*i* データベース・リファレンス』を参照してください。
- 統計とその解釈方法に関する詳細は、第 3 章「オプティマイザ統計の収集」を参照してください。

文のトレース例

この項では、AUTOTRACE 機能の使用例を示します。

パフォーマンス統計文と問合せ実行パス文のトレース

SQL バッファに次の文が含まれている場合、

```
SELECT E.LAST_NAME, E.SALARY, J.JOB_TITLE
  FROM EMPLOYEES E, JOBS J
    WHERE E.JOB_ID=J.JOB_ID AND E.SALARY>12000;
```

文は、次のように指定して実行すると自動的にトレースできます。

```
SET AUTOTRACE ON
/
```

出力は、次のようなものです。

```
LAST_NAME                 SALARY JOB_TITLE
------------------------- ---------- ----------------------------------
King                       24000 President
Kochhar                    17000 Administration Vice President
De Haan                    17000 Administration Vice President
Russell                    14000 Sales Manager
Partners                   13500 Sales Manager
Hartstein                  13000 Marketing Manager

6 rows selected.

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE
   1    0   TABLE ACCESS (BY INDEX ROWID) OF 'EMPLOYEES'
   2    1     NESTED LOOPS
   3    2       TABLE ACCESS (FULL) OF 'JOBS'
   4    2       INDEX (RANGE SCAN) OF 'EMP_JOB_IX' (NON-UNIQUE)
```

```
Statistics
----------------------------------------------------------
          0  recursive calls
          2  db block gets
         34  consistent gets
          0  physical reads
          0  redo size
        848  bytes sent via SQL*Net to client
        503  bytes received via SQL*Net from client
          4  SQL*Net roundtrips to/from client
          0  sorts (memory)
          0  sorts (disk)
          6  rows processed
```

注意： 接続先のサーバーのバージョンとサーバーの構成により、出力が異なる場合があります。

問合せデータを表示しない文のトレース

問合せデータを表示せずに同じ文をトレースするには、次のように入力します。

```
SET AUTOTRACE TRACEONLY
```

このオプションは、大きな問合せをチューニングしても問合せレポートを見る必要のない場合に便利です。

データベース・リンクによる文のトレース

データベース・リンクで文をトレースするには、次のように入力します。

```
SET AUTOTRACE TRACEONLY EXPLAIN
SELECT * FROM EMPLOYEES@MY_LINK;

Execution Plan
----------------------------------------------------------
0      SELECT STATEMENT (REMOTE) Optimizer=CHOOSE
1    0   TABLE ACCESS (FULL) OF 'EMPLOYEES'  MY_LINK.DB_DOMAIN
```

実行計画は、行 1 でアクセスされる表が、データベース・リンク MY_LINK.DB_DOMAIN を経由することを示します。

タイミング統計の収集

SQL*Plus TIMING コマンドは、1 つ以上のコマンドまたはブロックを実行する際に使用するコンピュータ・リソース量に関するデータを収集して表示する場合に使用します。TIMING は、経過時間データを収集し、タイマー期間中に実行されたコマンドに関するデータを保存します。

すべてのタイマーを削除するには、コマンド・プロンプトに CLEAR TIMING と入力します。

> **関連項目：** TIMING コマンドの詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

パラレル問合せおよび分散問合せのトレース

パラレル問合せまたは分散問合せの文をトレースすると、実行計画はカーディナリティというコストベース・オプティマイザによる行数の見積りを示します。一般に、各ノードのコスト、カーディナリティおよびバイトは、累積結果を表します。たとえば、結合ノードのコストには、結合操作を行うコストのみでなく、その結合でリレーションにアクセスする総コストも計算に入れます。

例 11-3 は、パラレル問合せオプションを使用した文のトレース例です。出力はシステム構成によって異なります。アスタリスク（*）の付いた行は、パラレル操作またはリモート操作を示します。各操作については、レポートの第 2 の部分で説明しています。例 11-3 の実行計画は、次の情報を含む 3 つの列からなります。

- 各実行ステップの行番号
- SQL 文の関数
- Oracle Real Application Clusters データベースまたはリモート・データベースの問合せのテキスト

PLAN_TABLE の列の詳細は、9-23 ページの「PLAN_TABLE 列」を参照してください。

例 11-3 パラレル問合せオプションによる文のトレース

パラレル問合せオプションを実行するパラレル問合せをトレースするには、次のようにします。

```
SQL> CREATE TABLE D2_t1 (unique1 NUMBER) PARALLEL -(DEGREE 6);
Table created.

SQL> CREATE TABLE D2_t2 (unique1 NUMBER) PARALLEL -(DEGREE 6);
Table created.

SQL> CREATE UNIQUE INDEX d2_i_unique1 ON d2_t1(unique1);
Index created.
```

```
SQL> SET LONG 500 LONGCHUNKSIZE 500
SQL> SET AUTOTRACE ON EXPLAIN
SQL> SELECT /*+ INDEX(B,D2_I_UNIQUE1) USE_NL(B) ORDERED -*/ COUNT (A.UNIQUE1)
     FROM D2_T2 A, D2_T1 B
     WHERE A.UNIQUE1 = B.UNIQUE1;

Execution Plan
----------------------------------------------------------
   0      SELECT STATEMENT Optimizer=CHOOSE (Cost=1 Card=1 Bytes=26)
   1    0  SORT (AGGREGATE)
   2    1   SORT* (AGGREGATE)                                      :Q2000
   3    2    NESTED LOOPS* (Cost=1 Card=41 Bytes=1066)             :Q2000
   4    3     TABLE ACCESS* (FULL) OF 'D2_T2' (Cost=1 Card=41 Byte :Q2000 s=533)
   5    3      INDEX* (UNIQUE SCAN) OF 'D2_I_UNIQUE1' (UNIQUE)     :Q2000

   2 PARALLEL_TO_SERIAL            SELECT /*+ PIV_SSF */ SYS_OP_MSR(COUNT(A1.C0
                                   )) FROM (SELECT /*+ ORDERED NO_EXPAND USE_NL
                                   (A3) INDEX(A3 "D2_I_UNIQUE1") */ A2.C0 C0,A3
                                   .ROWID C1,A3."UNIQUE1" C2 FROM (SELECT /*+ N
                                   O_EXPAND ROWID(A4) */ A4."UNIQUE1" C0 FROM "
                                   D2_T2" PX_GRANULE(0, BLOCK_RANGE, DYNAMIC)
                                   A4) A2,"D2_T1" A3 WHERE A2.C0=A3."UNIQUE1")
                                   A1

   3 PARALLEL_COMBINED_WITH_PARENT
   4 PARALLEL_COMBINED_WITH_PARENT
   5 PARALLEL_COMBINED_WITH_PARENT
```

実行計画の行 0 では、コストベース・オプティマイザが、行数が 1 で、26 バイトの使用を見積っていることが示されています。文の総コストは 1 です。行 2、3、4 および 5 にはアスタリスクが付いており、パラレル操作が示されています。たとえば、行 3 の NESTED LOOPS ステップは、PARALLEL_TO_SERIAL 操作です。PARALLEL_TO_SERIAL 操作では、SQL 文が実行されて逐次出力が生成されます。行 2 は、パラレル問合せサーバーの識別子が *Q2000* であることも示しています。

パラレル・レポート行を識別する番号は、親レポートの行を相互参照しています。たとえば、例の最後の行では、次のようになります。

```
4 PARALLEL_COMBINED_WITH_PARENT
```

4 は、実行計画内の `4 3 TABLE ACCESS*....` 行を参照しています。

ディスク読取りおよびバッファ取得の監視

ディスク読取りとバッファ取得の監視を行うには、次の文を実行します。

```
SET AUTOTRACE ON TRACEONLY STATISTICS
```

戻される結果は、通常、例 11-4 のようになります。

例 11-4 ディスク読取りおよびバッファ取得の監視

```
Statistics
----------------------------------------------------------
        70 recursive calls
         0 db block gets
       591 consistent gets
       404 physical reads
         0 redo size
       315 bytes sent via SQL*Net to client
       850 bytes received via SQL*Net from client
         3 SQL*Net roundtrips to/from client
         3 sorts (memory)
         0 sorts (disk)
         0 rows processed
```

consistent gets または physical reads が戻されたデータの量に対して多すぎる場合、その問合せのコストは高く、見直して最適化する必要があることを示しています。たとえば、1,000 未満の戻り行を予測している場合に consistent gets が 1,000,000 で、physical reads が 10,000 であれば、その問合せをさらに最適化する必要があります。

注意： V$SQL または TKPROF を使用して、ディスク読取りとバッファ取得を監視することもできます。

SQL*Plus のパフォーマンスに影響するシステム変数

次の変数は、SQL*Plus のパフォーマンスに影響します。

SET APPINFO OFF

DBMS_APPLICATION_INFO パッケージを介したスクリプトの自動登録を設定します。APPINFO を OFF に設定すると登録が無効化され、パフォーマンスおよびスクリプトのリソース使用率の監視が無効化されます。このようにしてオーバーヘッドを削減すると、パフォーマンスが改善される場合があります。

SET ARRAYSIZE

SQL*Plus によってデータベースから一度にフェッチされるバッチと呼ばれる行数を設定します。有効な値は 1 ～ 5000 です。値を大きくすると、多数の行をフェッチする問合せと副問合せの効率が上がりますが、より多くのメモリーが必要になります。値が 100 あたりを超えると、パフォーマンスはあまり改善されません。ARRAYSIZE は、効率を上げるという点以外、SQL*Plus 操作の結果に影響を与えません。

SET DEFINE OFF

置換変数に対するスクリプトを SQL*Plus で解析するかどうかを制御します。DEFINE が OFF の場合、SQL*Plus では置換変数に対するスクリプトが解析されません。スクリプトで置換変数が使用されていない場合に DEFINE を OFF に設定すると、パフォーマンスがある程度向上される場合があります。

SET FLUSH OFF

出力が、ユーザーの表示デバイスにいつ送られるかを制御します。OFF に設定すると、ホスト・オペレーティング・システムが出力をバッファに入れます。これによってプログラム I/O の量が削減され、パフォーマンスが改善されます。

ユーザーの介入が不要で、スクリプト実行が終了するまでその出力を参照する必要がないスクリプトを実行する場合にのみ OFF を使用します。

SET FLUSH は、*i*SQL*Plus ではサポートされていません。

SET SERVEROUTPUT

SQL*Plus が DBMS 出力をチェックして表示するかどうかを制御します。SERVEROUTPUT が OFF の場合、SQL*Plus では DBMS 出力がチェックされず、適用可能な SQL または PL/SQL 文の後に出力が表示されません。この出力をチェックおよび表示しないようにすると、パフォーマンスが向上する場合があります。

SET TRIMOUT ON

SQL*Plus で、表示された各行の終わりに後続の空白を許可するかどうかを決定します。ON に設定すると、各行の終わりにある空白が削除されます。これにより、特に低速の通信デバイスから SQL*Plus にアクセスしている場合、パフォーマンスが改善されます。TRIMOUT を ON に設定しても、スプーリングされた出力には影響しません。

SET TRIMOUT は、*i*SQL*Plus ではサポートされていません。

SET TRIMSPOOL ON

SQL*Plus で、スプーリングされた各行の終わりに後続の空白を許可するかどうかを決定します。ON に設定すると、各行の終わりにある空白が削除されます。これにより、特に低速の通信デバイスから SQL*Plus にアクセスしている場合、パフォーマンスが改善されます。TRIMSPOOL を ON に設定しても、端末への出力には影響しません。

SET TRIMSPOOL は、*i*SQL*Plus ではサポートされていません。

*i*SQL*Plus サーバー統計レポート

*i*SQL*Plus サーバー統計レポートにより、*i*SQL*Plus セッションに関する静的および動的な環境情報が提供されます。レポート内のアクティブ統計は、*i*SQL*Plus の監視およびチューニングをする場合に便利です。*i*SQL*Plus サーバー統計レポートを表示するには、次のようにします。

```
http://machine_name.domain:port/isqlplusdba?statistics={active|full}
[&refresh=number]
```

各項目の意味は次のとおりです。

- *machine_name.domain* は、*i*SQL*Plus サーバー統計の生成対象である Oracle HTTP Server の URL です。
- *port* は、*i*SQL*Plus サーバーで使用されるポート番号です。デフォルトは 7777 です。
- ?statistics={active|full} は、レポートの詳細レベルを指定します。
  - full に指定すると、生成可能なすべての統計が生成されます。この値がデフォルトです。
  - active に指定すると、*i*SQL*Plus サーバーに対して動的に変更されるセッション統計が提供されます。これらの統計は、全レポートの終わりにも含まれます。
- [&refresh=*number*] はオプションで、統計レポートが自動的にリフレッシュされるまでの時間を秒単位で指定します。最小値は 10 秒です。

レポートを実行するには、Oracle HTTP Server 認証を取得して *i*SQL*Plus DBA URL にアクセスする必要があります。ただし、データベースには接続しないため、Oracle9*i* へのログインは必要ありません。リソースの可用性を最大にするには、*i*SQL*Plus の各ユーザーに、適切な制限が定義されたデータベース・スキーマのプロファイルを設定する必要があります。

> **関連項目：** 統計のフル・セットを含む *i*SQL*Plus サーバー統計レポートの詳細は、『SQL*Plus ユーザーズ・ガイドおよびリファレンス』を参照してください。

アクティブ統計

アクティブ統計レポートには、表 11-3 にリストされた統計に対する現在の値が表示されます。これらの統計では、*i*SQL*Plus サーバーのチューニングに役立つフィードバックが提供されています。

表 11-3 アクティブ統計

統計	説明
アクティブ・セッション	アクティブな同時セッションの数、あるいは現在 *i*SQL*Plus にログインしているユーザーの数。
起動後のセッション	*i*SQL*Plus サーバーの起動後に確立されたセッションの累積件数。
最大同時セッション	*i*SQL*Plus サーバーの起動後の、同時セッションの最大数またはピークの件数。
起動後の時間切れセッション	*i*SQL*Plus サーバーの起動後、セッションが非アクティブになったためにタイムアウトになった累積件数。
アクティブ要求	アクティブな同時 HTTP リクエストの数。各要求は、ユーザーのアクション（ボタンのクリックなど）や、*i*SQL*Plus による要求の処理に対応します。Requests active の最大値は、`iSQLPlusNumberOfThreads` によって設定されます。値がこの制限に達し、ユーザーの応答時間が長くなった場合、`iSQLPlusNumberOfThreads` の値を大きくすると応答時間が改善されます。
起動後の要求	*i*SQL*Plus サーバー起動後の、アクティブな HTTP リクエストの累積件数。
次の期限切れ操作	次に処理が時間切れになるまでの、分単位に丸められた時間。
起動後の期限切れ操作	*i*SQL*Plus サーバーの起動後に、時間切れとなった処理が実行された回数。
ハッシュ表のコリジョン	現在ハッシュテーブルの衝突が発生しているアクティブなセッションの数。この統計と Sessions active を比較し、衝突によって現在問題が発生しているかどうかを調べます。

表 11-3 アクティブ統計（続き）

統計	説明
起動後のハッシュ表のコリジョン	*i*SQL*Plus サーバーの起動後に、ハッシュテーブルの衝突が発生したセッションの累積件数。この統計と Sessions since startup を比較し、衝突によって現在進行中の問題があるかどうかを調べます。

アクティブ統計の解析

次にアクティブ統計の解釈方法を説明します。

スレッド数の増加

アクティブ時間よりアイドル時間が長い場合は、`iSQLPlusNumberOfThreads` の値は大きくする必要があります。

- アクティブ時間とは、ユーザー要求が進行中でそれを処理するためにスレッドが消費される時間を指します。
- アイドル時間とは、ユーザー要求が処理済みで、関連処理スレッドが他の *i*SQL*Plus セッションに使用可能な時間を指します。

各スレッドが処理できるのは 1 つのユーザー要求です。ボタンをクリックするか、または *i*SQL*Plus のコマンドとリンクに従うと要求が開始され、すべての結果がユーザーに戻されると終了します。

`iSQLPlusNumberOfThreads` の値を設定する場合は、次の点に注意してください。

- `isqlplus.conf` ファイルに `iSQLPlusHashTableSize` が指定されていない場合は、`iSQLPlusNumberOfThreads` が増加するとその値は増加します。
- `iSQLPlusNumberOfThreads` の設定値が小さすぎる場合は、*i*SQL*Plus は要求負荷を処理できません。この問題の例は次のような場合です。
  - `ORACLE_HOME/Apache/Apache/logs/error_log` で頻繁に接続拒否エラーが発生する場合
  - 次の Oracle HTTP サーバーの内部エラーが発生する場合

    ```
    The server encountered an internal error or misconfiguration and was unable
    to complete your request.
    ```

ハッシュテーブルのサイズの拡大

通常、アクティブ時間よりアイドル時間が長い場合は、`iSQLPlusNumberOfThreads` の指定値に対して高めの `iSQLPlusHashTableSize` の値が必要になります。セッションがアイドルの場合も、各ユーザー・セッションでハッシュ表内のエントリを 1 つ消費します。

タイムアウト時間の削減

多数のセッションがタイムアウトになっている場合、ユーザーが正しくログアウトせず、セッションがアイドル状態のままである可能性があります。このような状況で、*i*SQL*Plus サーバーの負荷が高い場合、`iSQLPlusTimeOutInterval` の値を小さくしてセッションをより積極的にタイムアウトさせることを考慮します。

アイドル・タイムアウト

アイドル・タイムアウトとは、Oracle HTTP Server で *i*SQL*Plus からの結果を待機する時間です。FastCGI タイムアウト・パラメータのパラメータ値、`-idle-timeout` は 3600 秒に設定されています。この値は、Web ブラウザへアクセスする前に *i*SQL*Plus がタイムアウトになる可能性を低くし、長い問合せが多数ある場合でも、*i*SQL*Plus がタイムアウトになる前に結果を戻すには十分な値です。

ユーザー・セッションの存続期間を管理する `iSQLPlusTimeOutInterval` と、アイドル・タイムアウトとを混同しないようにします。

12

Oracle Trace の使用

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 を使用する方法を説明します。

この章には次の項があります。

注意： Oracle Trace は、将来のリリースでは使用できなくなる可能性があります。かわりに、SQL トレースおよび TKPROF を使用することを強くお薦めします。

関連項目： 第 10 章「SQL トレースおよび TKPROF の使用」

Oracle Trace の概要

Oracle Trace は汎用のイベント・ドリブンのデータ収集製品で、Oracle Server はこの製品を使用して、SQL の解析、実行、フェッチの統計や、待ち統計などの、パフォーマンスおよびリソース使用率のデータを収集します。

イベント・データ

イベントとは、製品内部でのアクティビティによる状態変化のことです。Oracle Trace は、Oracle Trace API が組み込まれたソフトウェア製品で発生する、事前定義済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、製品内での瞬間的な状態変化を表します。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することなどです。

Oracle Server には、10 種類を超えるイベントがあります。これらのうちの 3 つのイベントを次に示します。

- 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 処理の期間イベントの 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多数のデータを記録します。
- 行ソース : SQL 操作、位置、オブジェクト ID、実行計画の 1 つの行ソースで処理される、行数など、実行計画に関するデータ。

イベント・セット

Oracle Trace イベントは、データ収集を特定のイベントに制限する**イベント・セット**に編成できます。パフォーマンスの監視、監査、診断、その他の論理イベント・グループ化のためのイベント・セットを設定できます。

各イベント・セットは、それ自体の**製品定義ファイル**（fdf）で記述されます。製品定義ファイルは、イベントとそれらに関連するデータ項目を集めたものです。組み込まれた製品用に定義されたイベントの完全セットを、ALL イベント・セットと呼びます。他のイベントは、この ALL セットから導出されます。たとえば、Oracle Server には EXPERT セットと呼ばれるイベント・セットがあります。このセットには Oracle Expert チューニング・アプリケーションで使用される SQL イベント・データが含まれていますが、待機イベントなどの他のイベントは含まれていません。

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格納し、そのデータを定期的にコレクション・バイナリ・ファイルに書き込みます。この方法により、収集プロセスのリソース・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、データをフォーマットしてデータベースの表にします。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。このようなデータベース・テーブルは、「Oracle Trace フォーマッタ表」と呼ばれます。

Oracle Trace データの収集

次のメカニズムのいずれかを使用して Oracle Trace データを収集できます。

- Oracle Trace コマンドライン・インタフェース（CLI）
- データベース ORACLE_TRACE_* 初期化パラメータ
- PL/SQL から実行された Oracle Trace ストアド・プロシージャ

Oracle Trace コマンドライン・インタフェースの使用

Oracle Trace CLI（コマンドライン・インタフェース）で Oracle Trace のサーバー収集を制御できます。CLI は次の機能に対する OTRCCOL 実行可能ファイル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 CHECK *collection_name*
- OTRCCOL FORMAT *input_parameter_file*
- OTRCCOL DCF *col_name* cdf_file
- OTRCCOL DFD *col_name* username password service [*col_id*]

job_id パラメータは、値 1 に設定する必要があります。

入力パラメータ・ファイルには、次の例に示すように各関数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 関数の入力パラメータ・ファイルで初期値が定義されています。

注意： サーバー・パラメータ ORACLE_TRACE_ENABLE を true に設定して、Oracle Trace がサーバー・イベント・データを収集できるようにします。この設定は、Oracle8 以降のサーバーに必要です。

収集の開始

OTRCCOL START 関数によって、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 1 my_start_input_file
```

ここで、ファイル my_start_input_file には少なくとも次の入力パラメータが含まれます。

```
col_name= collection name
cdf_file= collection name.cdf
dat_file= collection name.dat
fdf_file= facility definition file.fdf
```

fdf_file パラメータの値として使用できるサーバー・イベント・セットは、ORACLE、ORACLEC、ORACLED、ORACLEE および ORACLESM の他に、CONNECT、SQL_ONLY、SQL_PLAN、SQL_TXN、SQLSTATS、SQLWAITS および WAITS があります。

> **関連項目：** サーバー・イベント・セットの説明は、表 12-2 を参照してください。

コレクション .cdf およびコレクション .dat ファイルは、EPC_COLLECTION_PATH 環境変数で上書きされないかぎり、CLI（または PL/SQL プロシージャ）で開始される収集のために、デフォルトで、$ORACLE_HOME/otrace/admin/cdf ディレクトリに作成されます。

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。これらのパラメータの詳細な説明は、『Oracle9*i* データベース・リファレンス』に記載されています。

---

collections name パラメータは、任意の有効な一意のファイル名にすることができます。

Oracle データベース収集には、1 つの追加パラメータが必要です。

```
regid= 1 192216243 0 0 5 database_SID
```

> **注意：** CLI の以前のバージョンでは、この構文どおりであることが必要で、'=' の前は空白なし、'=' の後には、1 つ以上の空白が必要でした。現在は、CLI は空白の有無を区別しません。

regid パラメータ・レコードは、Oracle Trace の収集が行われるデータベースを SID で識別します。regid パラメータ・レコードを構成する 6 つの要素を、表される順に次に示します。

- フラグ（この用途では、1 に設定）
- ベンダー番号（Oracle のベンダー番号は 192216243）
- cf_num
- cf_val
- 機能番号（Oracle Server は機能番号 5）
- データベース SID

cf_num と cf_val の各要素は、Oracle データベース収集の基本の regid では、0（ゼロ）に設定する必要があります。

収集されるデータの制限

収集するデータ量を制限する方法はいくつかあります。たとえば、収集するデータ量を減らすように追加の regid レコードを指定できます。この場合は、cf_num および cf_val に 0（ゼロ）以外の値を指定します。Oracle Server では、データベースのユーザー ID の値を記録するために、Oracle Trace 機能間項目 6（cf_num = 6）が予約されています。

ユーザーによる制限

Oracle Server では、データベースのユーザー ID の値を記録するために、Oracle Trace 機能間項目 6（cf_num = 6）が予約されています。したがって、たとえば追加の regid レコードに cf_num = 6 および cf_val = *some_DB_userID* を指定すると、データベース・イベント・データの収集はそのデータベース・ユーザーが実行したイベントに制限されます。

ユーザー 23 および 45 のデータベース・アクティビティについてのみを収集する場合は、次の 3 つの regid レコードを指定します。

```
regid= 1 192216243 0  0 5 ORCL
regid= 1 192216243 6 23 5 ORCL
regid= 1 192216243 6 45 5 ORCL
```

プロセスによる制限

収集を開始するときに CLI で使用される入力パラメータ・ファイルには、データベースおよびデータベース以外の Oracle Trace 収集を行うための次のオプション・パラメータを含めることもできます。

```
prores= process restriction
max_cdf= maximum collection file size
```

プロセス制限レコードを指定しない場合、収集の対象となるプロセスに制限はありません。プロセス制限を使用する場合は、1 つ以上のプロセス ID（PID）値を指定できる他、対象の各プロセスの所有者のオペレーティング・システム・ユーザー名も指定できます。

ファイル・サイズでの制限の設定

複数の異なるモードにおいて max_cdf パラメータは便利です。このパラメータは、収集する Oracle Trace データの最大量を、バイト単位（つまり、*collection*.dat ファイルのサイズ）で指定します。

値 0（ゼロ）は、制限を課してはならないことを示します。最大 2 GB までの正の値を指定すると、その制限サイズに達したときにデータ収集が停止します。負の値（ただし、-2 GB 以上）を指定すると、Oracle Trace は、「循環データ・ファイル」モードでデータを収集します。*collection*.dat が最大（max_cdf）に達した場合、そのデータを保存し（保存されていた dat ファイルはすべて削除）、新しい *collection*.dat ファイルへの収集を開始します。この方法により、使用するディスクの総容量は制限されますが、Oracle Trace のデータ収集は手動で収集を停止するまで続けることが可能になります。

収集のステータスのチェック

収集が開始されていることを確認します。

```
otrccol check collection_name
```

収集は、アクティブ、アクティブでない、見つからないのいずれかで表示されます。

収集の停止

次のように、OTRCCOL STOP 関数を使用すると収集の実行を一時停止します。

```
OTRCCOL STOP 1 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。

収集のフォーマット

OTRCCOL FORMAT 関数を使用すると、バイナリのコレクション・ファイルを Oracle 表にフォーマットできます。FORMAT 文の例は次のとおりです。

```
OTRCCOL FORMAT my_format_input_file
```

my_format_input_file には次の入力パラメータが含まれます。

```
username= database username
password= database password
service= database service name
cdf_file= usually same as collection_name.cdf
full_format= 0/1
```

full_format 値 1 は、完全なフォーマットを生成します。full_format 値 0 は部分的なフォーマットを生成します。部分的なフォーマットでは新しいデータ、つまり以前の任意のフォーマット以降に収集されたデータのみがフォーマットされます。

> **関連項目：** otrcfmt ユーティリティ・プログラムを使用して Oracle Trace の収集をフォーマットする操作の詳細は、12-12 ページの「Oracle Trace データの Oracle 表へのフォーマット」を参照してください。

収集の削除

OTRCCOL DCF（コレクション・ファイルの削除）関数は、特定の収集のためのコレクション .cdf ファイルとコレクション .dat ファイルを削除します。OTRCCOL DFD（フォーマット済みデータの削除）関数を使用すると、特定の収集について Oracle Trace フォーマッタ表からフォーマット済みデータを削除できます。選択 DFD に col_id オプション・パラメータを指定できます。この DFD には、1 つの収集に対して複数の col_id が複数の（完全な）フォーマットで作成されています。

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 個の Oracle データベース初期化パラメータが設定されています。データベースの特権アカウントにログインし、SHOW PARAMETER ORACLE_TRACE 文を実行すると、次のパラメータが表示されます。

表 12-1 Oracle Trace 初期化パラメータ

名前	型	デフォルト値
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

これらの Oracle Trace サーバー・パラメータを変更して、サーバー・イベント・データの Oracle Trace 収集を可能にし、初期化ファイルに追加してサーバー・イベント・データを使用できます。

ただし、この Oracle Trace 収集の制御方法はやや柔軟性に欠けます。データベースをシャットダウンしないと、収集の名前を変更できません （8.1.6 以下の Oracle リリースの場合は、収集はシャットダウンによってのみ停止されます。シャットダウン後、ORACLE_TRACE_ENABLE = FALSE を設定してから再起動します)。ただし、ORACLE_TRACE_ENABLE = TRUE で、ORACLE_TRACE_COLLECTION_NAME = "" [すなわち、空の名前文字列] の場合、データベース・イベント・データの Oracle Trace 収集は、他の収集制御メカニズムのうちの 1 つ、たとえば、Oracle Trace CLI などを使用して行えます。他のメカニズムには、データベース初期化パラメータより高い柔軟性があります。一般に収集制御には、パラメータよりもその他のメカニズムが優先して使用されます。

Oracle Trace の収集の使用可能化

ORACLE_TRACE_ENABLE データベース初期化パラメータは、デフォルトでは false です。このため、収集制御に使用するメカニズムとは無関係に、そのサーバーの Oracle Trace データの収集は使用禁止になっています。

DBinit.ora の ORACLE_TRACE_ENABLE を true に設定すると、そのサーバーの Oracle Trace 収集は使用可能になります。ただしデータベース・インスタンスが起動されるときに、収集を起動する必要はありません。データベース・パラメータのみを使用してデータベース・イベント・データの Oracle Trace 収集を起動する場合、6 つの ORACLE_TRACE_* パラメータをすべて指定するか、またはデフォルトで NULL でない値を持っている必要があります。つまり、一般には、ORACLE_TRACE_ENABLE を true に設定するとともに、NULL で

ない ORACLE_TRACE_COLLECTION_NAME（最大文字長は 16 文字）の値を指定する必要があります。

> **注意：** 収集の名前は *collection name*.cdf および .dat バイナリ・ファイル名を作成する場合にも使用されるため、プラットフォームによっては 8.3 ファイル・ネーミング規則が適用される場合があります。8.3 ファイルのネーミングとは、ファイル名が 8 文字以下、ファイル拡張子が 3 文字以下に制限されているシステムを意味します。

ORACLE_TRACE_ENABLE は現在（Oracle8*i* リリース 8.1.7 では）、動的パラメータであるため、データベースの実行中に true または false に設定できます。この設定は、ALTER SESSION 文または ALTER SYSTEM 文を使用して、カレント・データベース・セッションまたはすべてのセッション（将来のセッションも含む）に対して行えます。データベースがシャットダウンされた後に再起動されると、ORACLE_TRACE_ENABLE の *DBinit*.ora 設定が再度使用されて、データベース・イベント・データの Oracle Trace 収集が使用可能または使用禁止になります。

Oracle Trace で収集するイベント・セットの判断

ORACLE_TRACE_FACILITY_NAME データベース初期化パラメータは、データベース・パラメータでデータ収集を制御する場合に、Oracle Trace が収集するイベント・セットを指定します。このパラメータのデフォルトは ORACLED（すなわち、Oracle デフォルト・イベント・セット）です。

> **注意：** ORACLE_TRACE_FACILITY_NAME パラメータはファイル拡張子を使用しません。したがって、.fdf 拡張子をこのパラメータの一部として指定しないでください。

Oracle Trace 収集を開始するようにデータベース・パラメータを設定した状態で、データベースがデータ収集を開始しない場合は、次の項目をチェックしてください。

- 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。
- Oracle Trace admin ディレクトリに、COLLECT.DAT、FACILITY.DAT（Oracle 7 リリース 7.3 の場合は PROCESS.DAT）および REGID.DAT が必要です。こ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または再作成します。
- 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 Collection Service の OTRCCOL イメージがエラーを記録するようになっている場合、Oracle Trace は現行のデフォルト・ディレクトリに EPC_ERROR.LOG ファイルを作成します。
- サーバー 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 ファイルを検索します。*.trc ファイルについて「epc」を検索すると、エラーを見つけることができます。所定のプラットフォームで使用可能な場合、これらのエラーとその説明は、$ORACLE_HOME/otrace/mesg/epcus.msg（英語）で見ることができます。

PL/SQL による Oracle Trace の収集の制御

Oracle では、データベースと非データベースの両方の Oracle Trace 収集制御を PL/SQL から行えるようにする Oracle Trace ライブラリも提供しています。

注意： リリース 7.3.3 までの Oracle Server の旧バージョンでは、Oracle Trace データベース収集を開始および停止する他のストアド・プロシージャを提供していましたが、大きな制限がありました。たとえば、収集の対象となるのは、収集が開始されたときアクティブだったデータベース・セッションのみでした。Oracle Trace 収集が開始された後で起動されたセッションについては、データベース・イベント・データは収集されませんでした。

Oracle8 では、Oracle Trace CLI から開始されたデータベース収集については、これらの制限がなくなりました。ただし、以前の Oracle7 ストアド・プロシージャから制御されるデータベース収集には、依然として制限がありました。Oracle Trace 8.1.7 で提供された新しいプロシージャでは、データベースと非データベース収集の両方でこれらの制限がなくなり、Oracle Trace CLI と同じレベルの収集制御が可能になりました。

この新しいライブラリの名前と場所はともに、プラットフォームにより異なります。UNIX プラットフォームでは、ライブラリは $ORACLE_HOME/lib/libtracepls9.so です。

Win32 プラットフォーム（たとえば、Windows NT）では、ライブラリは %ORACLE_HOME%¥bin¥oratracepls9.dll です。

otrace/admin ディレクトリには、このライブラリのデータベース LIBRARY オブジェクトを定義したり、PL/SQL からライブラリを呼び出す場合に使用できるプロシージャを定義するための、次の 2 つの新しい SQL スクリプトが含まれています。

- OTRCPLSLIB.SQL
- OTRCPLSCMD.SQL

注意： デフォルトでは、OTRCPLSLIB.SQL はすべてのデータベース・ユーザーに対して LIBRARY へのアクセス権を付与します。この SQL スクリプトは、必要に応じてアクセスを制限するように編集できます。

また、otrace/demo ディレクトリには、Oracle Trace 収集の開始、停止およびフォーマットのそれぞれの PL/SQL の例を示す SQL スクリプトが含まれています。各スクリプトは次のとおりです。

- OTRCPLSSC.SQL
- OTRCPLSCC.SQL
- OTRCPLSFC.SQL

「収集開始」例のスクリプト OTRCPLSSC.SQL では、regid_list に、単一の要素 "1 192216243 0 0 5 ORCL" のみが含まれています。この内部の二重引用符は、6 つの構成要素で形成される単一の regid 文字列を表すために必要です。各構成要素の内容を、表される順に次に示します。

- フラグ（この用途では、1 に設定）
- ベンダー番号（Oracle のベンダー番号は 192216243）
- cf_num
- cf_val
- 機能番号（Oracle Server は機能番号 5）
- データベース SID

Oracle Trace データベース収集の場合、基本的に、データベース SID を識別し、Oracle データベースについての収集を行うことを指定するための、この例のような regid 文字列が必要です。cf_num と cf_val は、この基本の regid レコード内で 0（ゼロ）です。

収集されるデータ量を減らすために、regid レコードを追加して指定できます。cf_num と cf_val 項目は、この指定に使用します。Oracle データベースでは、データベースのユーザー ID の値を記録するために、Oracle Trace 機能間項目 6（cf_num = 6）が予約されています。したがって、追加の regid レコードに cf_num = 6 および cf_val = *some_DB_userID* を指定すると、データベース・イベント・データの収集はそのデータベース・ユーザーが実行したイベントに制限されます。たとえば、ユーザー 23 と 45 のデータベース・アクティビティについてのみ収集を行う場合、regid_list は次の 3 つのレコードから構成されます。

```
regid_list   VARCHAR2(256) := '"1 192216243 0  0 5 ORCL",
                               "1 192216243 6 23 5 ORCL",
                               "1 192216243 6 45 5 ORCL"';
```

注意： 読み取りやすくするために、この regid 文字列のレイアウトは単純化されています。

同様に、fdf_list 引数で 1 つの .fdf ファイル（機能定義ファイル）の名前を指定することもあります。これは、代表的な事例です。実際には、複数の機能が収集に関与している場合は、複数の .fdf が fdf_list に指定されることもあります。ただし、どの機能（たとえば、データベース）に対しても、指定できる .fdf は 1 つのみです。

その一方、プロセス制限リスト prores_list は、空にすることができます。これは、収集に参加できるプロセスに制限を付けないことを示します。プロセス制限を使用する場合は、1 つ以上のプロセス ID（PID）値を指定できる他、各プロセスの所有者のオペレーティング・システム・ユーザー名も指定できます。

OTRCPLSSC.SQL による収集開始の例での他の引数は、ここに示すように単一の数値または文字列値です。たとえば、収集の名前と収集データ・ファイル・サイズの最大値は、それぞれ col_name 変数と maxsize 変数で指定されます。

Oracle Trace の収集結果へのアクセス

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、収集のためのバイナリの Oracle Trace 収集定義ファイルです。このファイルは、収集される対象を記述します。
- *collection name*.dat は、収集された Oracle Trace イベント・データを含むバイナリ形式の出力ファイルです。.dat ファイルを新たに作成した時点では、その空のファイルには、ファイル・ヘッダー情報が 35 バイトのみ含まれています。

収集ファイル内の Oracle Trace データへは、次の方法でアクセスできます。

- 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。
- バイナリ・ファイルから Oracle Trace レポートを作成できます。

Oracle Trace データの Oracle 表へのフォーマット

Oracle データベース表に Oracle Trace バイナリ収集データをフォーマットし、SQL などのツールでこのフォーマットされたデータにアクセスできます。Oracle Trace フォーマットは、収集されたイベント型ごとに個別の表を作成します。たとえば、サーバー収集時に記録されたすべてのデータベース解析イベントのデータを格納する解析イベント表が作成されます。

注意： Oracle Server リリース 7.3.4 以上の場合、Oracle Trace フォーマッタは、必要に応じてフォーマッタ表を自動作成します。

次の構文を使用して、OTRCFMT フォーマッタ・ユーティリティで Oracle Trace 収集をフォーマットします。

```
OTRCFMT [options] collection_name.cdf [user/password@database]
```

コレクション .cdf と .dat の位置が現行のデフォルト・ディレクトリではない場合は、.cdf ファイルのフル・パスを指定します。

user/password@database（またはこの一部、たとえば、password や database など）を省略すると、OTRCFMT はこの情報の入力を要求します。

Oracle Trace を使用すると、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のみが 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フォーマットし直す場合は、オプション・パラメータ -f （フォーマッタ表に新しい収集 ID を生成）を使用します。

Oracle Trace では、いくつかのサンプル SQL スクリプトが提供されており、それらを使用してフォーマット設定サーバー・イベント・データ表にアクセスできます。これらのスクリプトは、otrace ディレクトリ・ツリーの OTRCRPT*.SQL 内にあります。

注意： フォーマッタ表の一部のバージョン間では互換性がないため、Oracle Trace フォーマッタのバージョンごとに個別のデータベース・スキーマを使用してください。

Oracle Trace レポート作成ユーティリティの実行

Oracle Trace レポート作成ユーティリティを使用すると、サーバー・イベントの各状態変化に関連する項目のデータが表示されます。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。オプション文修飾子を使用することによって、レポート出力を制御できます。次のレポート・ユーティリティ文の構文を使用します。

```
OTRCREP [options] collection name.cdf
```

コレクション .cdf と .dat の位置が現行のデフォルト・ディレクトリではない場合は、.cdf ファイルのフル・パスを指定します。

最初は、PROCESS.txt というレポートを実行してください。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プロセスの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修飾子を使用します。

修飾子	説明
output_path	レポート・ファイルの出力のフル・パスを指定します。このパスを指定しない場合、ファイルは現行ディレクトリに作成されます。
-p[*pid*]	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発生順に含む 1 つのファイルが作成されます。PID 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。出力ファイルには、*collection_name*_Ppid.txt という名前が付けられます。
-P	収集に含まれるすべてのプロセスをリストするレポート・ファイル名 PROCESS.txt 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すると、詳細なレポートの作成対象にするプロセスを判断できます。
-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 ページ当たり 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。
-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）ファイル内の発生した順序に従って作成します。

デフォルト OTRCREP レポート出力には、オプションの修飾子の指定はなく、収集されるイベント型当たり 1 つのテキスト・ファイルで構成されます。参加するすべてのプロセスからのデータは、これらのテキスト・ファイルの中で統合されます。

Oracle Server のイベント

この項では、Oracle Server に組み込まれたイベントについて説明します。ほとんどのイベントは、Oracle Expert によるパフォーマンスの解析とチューニング、およびワークロードの解析に利用できます。

イベントには、**ポイント**・イベントと**期間**イベントの 2 種類があります。ポイント・イベントは、対象の製品で発生する瞬間的な状態変化です。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することなどです。

表 12-2 は、Oracle Trace に組み込まれている Oracle Server イベントのリストです。詳細は、各イベントの項を参照してください。

表 12-2 Oracle Server のイベント

イベント	説明	イベントのタイプ
1 Connection	データベースへの接続	ポイント
2 Disconnect	データベースからの切断	ポイント
3 ErrorStack	コア・ダンプのコード・スタック	ポイント
4 Migration	共有サーバー・プロセス間のセッションの移行	ポイント
5 ApplReg	アプリケーション・コンテキスト情報	ポイント
6 RowSource	行情報。Oracle Server リリース 8.0.2 以上の場合は、実行計画に関する情報も含まれます。	ポイント
7 SQLSegment	SQL 文のテキスト	ポイント
8 Parse	SQL 解析情報	期間
9 Execute	SQL の実行情報	期間
10 Fetch	実際にとり出された行の情報	期間
11 LogicalTX	データベースの状態を変更する可能性があるコマンドが最初に発行されたことの記録	期間
12 PhysicalTX	データベースの状態が実際に変更されたイベントの記録	期間
13 Wait	一般待機イベント。コンテキストは、イベント文字列で提供されます。	ポイント

イベントで収集されるデータ項目

項目と呼ばれる情報が、各イベントに対応付けられています。項目には、次の 3 つのタイプがあります。

- リソース使用率項目
- 製品間項目
- Oracle Server イベント固有項目

リソース使用率項目

Oracle Trace には、Oracle Server などのアプリケーションについて収集を行うための、リソース使用率項目と呼ばれる、項目の規格セットがあります。また、Oracle Server 内のすべての期間イベントには、Oracle Server 固有のデータベース統計用の項目が含まれています。

標準のリソース使用率項目については、表 12-3 で説明しています。

Oracle Trace 収集は、アクセス、解析およびレポートのために Oracle 表にフォーマットできます。以降に示す表の右端の列に、Oracle データベースにフォーマットされるときのデータ項目のデータ型を示しています。

表 12-3 標準リソース使用率項目

項目名	説明	項目 ID	フォーマットされたデータのデータ型
UCPU	ユーザー・モードでの CPU 時間	129	NUMBER
SCPU	システム・モードでの CPU 時間	130	NUMBER
INPUT_IO	ファイル・システムが入力を実行した回数	131	NUMBER
OUTPUT_IO	ファイル・システムが出力を実行した回数	132	NUMBER
PAGEFAULTS	ハードとソフト、ページ・フォルト数	133	NUMBER
PAGEFAULT_IO	ハード・ページ・フォルト数	134	NUMBER
MAXRS_SIZE	使用する最大常駐セット・サイズ	135	NUMBER

項目の実装はプラットフォーム固有です。項目が実装されない場合、値は 0 です。たとえば、Windows NT では、現在は CPU 時間のみが記録されます。

製品間項目

Oracle Trace には、製品間項目と呼ばれる（以前は、機能間項目と呼ばれていた）14 項目のセットがあります。プログラマはこれらのデータ項目を使用して、各種製品のイベントを関連付けることができます。たとえば、ビジネス・トランザクションは、アプリケーションとデータベースの 2 つの製品でイベントを生成します。製品間データ項目を使用すると、ビジネス・トランザクション全体の解析のためにこれらの個々のイベントを結びつけることができます。

製品間項目は、表 12-4 に示すような特定の製品または製品タイプ用に予約されています。項目が予約されている製品を使用しない場合、これらの項目を独自の目的に使用できます。

表 12-4 製品間項目

項目名	レイヤー	説明	項目 ID	フォーマットされたデータのデータ型
CROSS_FAC_1	アプリケーション	アプリケーション ID。Oracle Financials、サード・パーティ製または顧客のアプリケーションなどの、ハイレベル・アプリケーションで使用。	136	NUMBER
CROSS_FAC_2	Oracle Forms	Oracle Forms ID	137	NUMBER
CROSS_FAC_3	Oracle Net	リモート・ノード接続 ID	138	NUMBER
CROSS_FAC_4	Oracle Server	トランザクション ID	139	NUMBER
CROSS_FAC_5	Oracle Server	SQL 文のハッシュ ID	140	NUMBER
CROSS_FAC_6	Oracle Server リリース 8.x	ユーザー ID	141	NUMBER
CROSS_FAC_7	Oracle Server リリース 8.x	待機タイプ	142	NUMBER
CROSS_FAC_8	n/a	予約なし	143	NUMBER
CROSS_FAC_9	n/a	予約なし	144	NUMBER
CROSS_FAC_10	n/a	予約なし	145	NUMBER
CROSS_FAC_11	n/a	予約なし	146	NUMBER
CROSS_FAC_12	n/a	予約なし	147	NUMBER
CROSS_FAC_13	n/a	予約なし	148	NUMBER
CROSS_FAC_14	n/a	予約なし	149	NUMBER

注意： Oracle Trace のこのバージョンでは、「機能」という用語が「製品」に変更されました。したがって、CROSS_FAC_*x* という名前の項目は製品間項目です。

製品間項目 1（CROSS_FAC_1 と呼ばれる）にデータが含まれるのは、組み込まれたアプリケーションからデータが提供される場合のみです。

製品間項目 2（CROSS_FAC_2）は、Oracle Forms の将来のリリース用に予約されています。組み込まれたアプリケーションと Oracle Forms は、これらの製品間項目を通して Oracle Server 収集に識別データを渡します。

製品間項目 3（CROSS_FAC_3）は、Oracle Net 用に予約されています。Oracle Net は、CROSS_FAC_3 を通して Oracle Trace に接続 ID を提供します。CROSS_FAC_3 は、クライアント / サーバー間または多層間で Oracle Trace 収集を調整する際の重要な要素です。Oracle Trace は、マージャで一致させる共通要素（クライアントとサーバーの収集ファイルなど）として Oracle Net グローバル接続 ID を使用します。クライアントとサーバーの接続のためのグローバル接続 ID は同じです。

大半の Oracle Server イベントは、製品間項目 1 ～ 6 を記録します（キャッシュ I/O は記録しません）。

Oracle Server イベント固有項目

Oracle Server 製品（機能）の定義ファイル（*.fdf）は、Oracle Server に固有の項目を定義します。リスト内の項目を検索するには、その項目の番号を使用します。フォーマットされたデータ型は、Oracle Trace フォーマッタがデータを Oracle データベースにフォーマットするときにどのように項目を定義するかを表しています。

Oracle Server 項目を、表 12-5 にリストします。

表 12-5 Oracle Server 項目

項目名	説明	項目番号	フォーマットされたデータ型
App_Action	DBMS_APPLICATION_INFO.SET_MODULE プロシージャで設定されたアクション名	23	VARCHAR2 (1020)
App_Module	DBMS_APPLICATION_INFO.SET_MODULE プロシージャで設定されたモジュール名	22	VARCHAR2 (1020)
Commit_Abort	トランザクションがコミットされたか強制終了されたかを示します。	24	NUMBER
Consistent_ Gets	一貫モードで取り出されたブロック数（データを変更せず、ロックや他のユーザーとの競合が発生しません）	104	NUMBER

表 12-5 Oracle Server 項目（続き）

項目名	説明	項目番号	フォーマットされたデータ型
CPU_Session	CPU セッション	112	NUMBER
Current_UID	現行ユーザー ID	36	NUMBER
Cursor_Number	SQL 文に対応付けられたカーソルの番号	25	NUMBER
DB_Block_Change	変更されたブロック数	102	NUMBER
DB_Block_Gets	現行モードで取り出されたブロック数。大規模な問合せの場合、この項目は、必要なすべてのレコードを取り出すために、データベースのいくつのセクション（論理ページ）がフェッチされたかを示します。	103	NUMBER
Deferred_Logging	Oracle Trace が内部で使用した値	14	NUMBER
Depth	SQL 文が処理される再帰的レベル	32	NUMBER
Description	発生するイベントに依存します（たとえば、待機イベントの記述）。	43	VARCHAR2 (1020)
Elapsed_Session	セッションの経過時間	113	NUMBER
End_of_Fetch	取り出されたデータが問合せからの最後のデータである場合に設定されるフラグ	38	NUMBER
Lib_Cache_Addr	ライブラリ・キャッシュ内の SQL 文のアドレス	27	VARCHAR2（64）
Login_UID	セッションのユーザー ID を識別する Oracle データベース内の内部 ID	15	NUMBER
Login_UName	セッションのシステム・アカウント名を識別する Oracle データベース内の内部 ID	16	VARCHAR2 (1020)
Missed	SQL 文がライブラリ・キャッシュ内で欠落していた場合に設定されるフラグ	33	NUMBER
Object_ID[1]	行ソースのオブジェクト ID	46	NUMBER
Operation[1]	操作のテキスト	47	VARCHAR2 (1020)
Operation_ID[1]	文の実行計画内の操作の位置	28	NUMBER
Optimizer_Mode	Oracle オプティマイザ・モード	35	VARCHAR2（128）

表 12-5 Oracle Server 項目（続き）

項目名	説明	項目番号	フォーマットされたデータ型
Oracle_Cmd_Type	Oracle コマンド番号	34	NUMBER
Oracle PID	Oracle プロセス ID	11	NUMBER
OS_Image	オペレーティング・システム・イメージ（プログラム名）	42	LONG
OS_Mach	オペレーティング・システム・ホスト・マシン	20	VARCHAR2 (1020)
OS_Term	オペレーティング・システム端末	19	VARCHAR2 (1020)
OS_UName	オペレーティング・システム・ユーザー名	18	VARCHAR2 (1020)
P1	P1 の定義は、P1 が発生するイベントに依存します。	1	NUMBER
P2	P2 の定義は、P2 が発生するイベントに依存します。	2	NUMBER
P3	P3 の定義は、P3 が発生するイベントに依存します。	3	NUMBER
P4	P4 の定義は、P4 が発生するイベントに依存します。	4	NUMBER
P5	P5 の定義は、P5 が発生するイベントに依存します。	5	NUMBER
P6	P6 の定義は、P6 が発生するイベントに依存します。	6	NUMBER
P7	P7 の定義は、P7 が発生するイベントに依存します。	7	NUMBER
P8	P8 の定義は、P8 が発生するイベントに依存します。	8	NUMBER
P9	P9 の定義は、P9 が発生するイベントに依存します。	9	NUMBER
P10	P10 の定義は、P10 が発生するイベントに依存します。	10	NUMBER
Parent_Op_ID[1]	親操作	44	NUMBER

表 12-5 Oracle Server 項目（続き）

項目名	説明	項目番号	フォーマットされたデータ型
PGA_Memory	プロセス・グローバル領域メモリー	101	NUMBER
Physical Reads	ディスクから読み取られたブロック数	105	NUMBER
Position[1]	同じ親操作を持つイベント内の位置	45	NUMBER
Position_ID[2]	文の実行計画内の操作の位置	28	NUMBER
Redo_Entries	プロセスが生成した REDO エントリ数	106	NUMBER
Redo_Size	REDO エントリのサイズ	107	NUMBER
Row_Count	処理された行数	29	NUMBER
Schema_UID	スキーマ・ユーザー ID	37	NUMBER
Session_Index	Oracle セッション ID	12	NUMBER
Session_Serial	セッション・シリアル番号	13	NUMBER
SID	文字列型のセッション ID	17	VARCHAR2 (1020)
Sort_Disk	実行されたディスク・ソート回数	110	NUMBER
Sort_Memory	実行されたメモリー・ソート回数	109	NUMBER
Sort_Rows	ソートされた行数の合計	111	NUMBER
SQL_Text	SQL 文のテキスト	31	LONG
SQL_Text_Hash	SQL 文へのポインタ	26	NUMBER
SQL_Text_ Segment	SQL テキストのアドレス	30	NUMBER
T_Scan_Rows_ Got	全表スキャンで処理された行	108	NUMBER
TX_ID	ロールバック・セグメント番号、スロット番号およびラップ番号から構成されるトランザクションの一意な ID	41	VARCHAR2（72）
TX_SO_Addr	トランザクション状態オブジェクトのアドレス	40	VARCHAR2（64）

表 12-5 Oracle Server 項目（続き）

項目名	説明	項目番号	フォーマットされたデータ型
TX_Type	トランザクションのタイプ。値はビットマップです（たとえば、2 はアクティブ・トランザクション、0X10 はスペース・トランザクション、0X20 は再帰的トランザクション）。	39	NUMBER
UGA_Memory	ユーザー・グローバル領域セッション・メモリー	100	NUMBER
Wait_Time	待機イベントの 100 分の 1 秒単位の経過時間	21	NUMBER

[1] Oracle Server リリース 8.0.2 以上に固有の項目

[2] Oracle Server リリース 8.0.2 以上では Operation_ID で置き換えられます。

各イベントに対応付けられた項目

この項では、各イベントの詳細を説明するとともに、各イベントに対応付けられた項目のリストを示します。項目の説明は、表 12-5 を参照してください。

> **注意：** Oracle Server リリース 8.0.5 より前のリリースでは、製品間項目 1 ～ 5 がサーバー・コードで設定されていました。Oracle Server リリース 8.0.5 から、製品間項目 6（および待機イベント用の製品間項目 7）が追加されました。

データをフォーマットすると、Oracle Trace は収集された各イベント型に対応する表を作成します。イベント・データ表の名前は **V_vendor#_F_product#_E_event#_version** です。この **version** は Oracle Server リリースの番号です。製品バージョンのピリオドはすべてアンダースコアに置き換えられます。otrcsyn.sql スクリプトを使用して、これらの表のシノニムを作成できます。

> **注意：** 次の表では、例として Oracle7 Server の名前を使用します。

Oracle Trace フォーマッタは、各イベント項目の列を作成します。ポイント・イベントの場合、列名は項目名と同じです。期間イベントの場合、開始イベントの項目は項目名に _START が付き、終了イベントの項目は項目名に _END が付きます。

フォーマッタは、収集番号、プロセス識別子、タイムスタンプ情報の追加列を自動的に付加します。表 12-6 に説明を示します。

表 12-6 Oracle Trace フォーマッタで組み込まれる追加列

列名	説明	データ型
collection_number	フォーマッタで自動的に指定される収集 ID	NUMBER (4)
epid	プロセス ID	NUMBER (8)
timestamp	ポイント・イベントの記録時間	DATE
timestamp_nano	ポイント・イベントの記録時間の小数部	NUMBER
timestamp_start	期間イベントの開始時間	DATE
timestamp_nano_start	期間イベントの開始時間の小数部	NUMBER
timestamp_end	期間イベントの終了時間	DATE
timestamp_nano_end	期間イベントの終了時間の小数部	NUMBER

イベント統計ブロック

いくつかのイベントには、データベース・パフォーマンスに関連する項目が表示されます。便宜的にこれらの項目をイベント統計ブロックと呼びます。イベント統計ブロックの各項目を、次にリストします。

```
UGA_Memory
PGA_Memory
DB_Block_Change
DB_Block_Gets
Consistent_Gets
Physical_Reads
Redo_Entries
Redo_Size
T_Scan_Rows_Got
Sort_Memory
Sort_Disk
Sort_Rows
CPU_Session
Elapsed_Session
```

Connection イベント

Connection イベント（イベント =1）は、データベースへの接続が行われるたびに記録を取ります。Connection イベントに対応付けられた項目を、次にリストします。

```
Session_Index
Session_Serial
Oracle_PID
Login_UID
Login_UName
SID
OS_UName
OS_Term
OS_Mach
OS_Image
製品間項目 1-6
```

Oracle Server は、Session_Index および Session_Serial の組合せを使用して接続を一意に識別します。Oracle Net は、CROSS_FAC_3 に格納されている接続 ID を使用して接続を一意に識別します。

Disconnect イベント

Disconnect イベントは、データベース切断が行われるたびに記録を取ります。Disconnect イベントに対応付けられた項目を、次にリストします。

```
Session_Index
Session_Serial
イベント統計ブロック
Oracle_PID
製品間項目 1-6
```

Disconnect イベントは、1 つの Connection イベントに対応します。したがって、同じフィールド、すなわち Session_Index と Session_Serial の組合せまたは CROSS_FAC_3 のいずれかで、切断を一意に識別します。

ErrorStack イベント

ErrorStack イベントは、エラーのあるプロセスを識別します。ErrorStack イベントに対応付けられた項目を、次にリストします。

```
Session_Index
Session_Serial
Oracle_PID
P1
P2
P3
P4
P5
```

P6
P7
P8
製品間項目 1-6

ErrorStack イベントには明示的な ID がありません。Session_Index、Session_Serial、Timestamp および Timestamp_Nano の組合せで、特定の ErrorStack イベントを一意に識別します。

Migration イベント

Migration イベントは、セッションが共有サーバー・プロセスに移行するたびに記録を取ります。このイベントは現在、Oracle Server コードで無効にされています。

Migration イベントに対応付けられた項目を、次にリストします。

Session_Index
Session_Serial
Oracle_PID
製品間項目 1-6

Migration イベントには明示的な ID がありません。Session_Index、Session_Serial、Timestamp および Timestamp_Nano の組合せで、特定の Migration イベントを一意に識別します。

ApplReg イベント

ApplReg イベント（イベント =5）は、アプリケーションが特定の時点にある場合に、Oracle Trace に登録を行います。ApplReg イベントに対応付けられた項目を、次にリストします。

Session_Index
Session_Serial
App_Module
App_Action
製品間項目 1-6

ApplReg イベントには明示的な ID がありません。Session_Index、Session_Serial、Timestamp および Timestamp_Nano の組合せで、特定の ApplReg イベントを一意に識別します。

RowSource イベント

RowSource イベントは、実行計画内で 1 つの行ソースによって処理された行数を記録します。RowSource イベントに対応付けられた項目を、次にリストします。

```
Session_Index
Session_Serial
Cursor_Number
Position_ID
Row_Count
製品間項目 1-5
```

Session_Index、Session_Serial、Cursor_Number および Position_ID の組合せで、RowSource イベントを一意に識別します。

RowSource イベント

RowSource イベントは、実行計画内の 1 つの行ソースで処理された行数を記録します。Oracle Server リリース 8.0.2 以上の RowSource イベントに対応付けられた項目を、次にリストします。

```
Session_Index
Session_Serial
Cursor_Number
Operation_ID
Row_Count
Parent_Op_ID
Position
Object_ID
Operation
製品間項目 1-6
```

Session_Index、Session_Serial、Cursor_Number および Operation_ID の組合せで、RowSource イベントを一意に識別します。

注意： Operation 項目のテキストは実行計画に関する情報と同等で、この情報は EXPLAIN PLAN を実行して取得されるデータに似ています。

SQLSegment イベント

SQLSegment イベントは SQL 文の説明です。SQLSegment イベントに対応付けられた項目を、次にリストします。

Session_Index
Session_Serial
Cursor_Number
SQL_Text_Hash
Lib_Cache_Addr
SQL_Text_Segment
SQL_Text
製品間項目 1-6

SQL セグメントには明示的な ID がありません。SQL_Text_Hash フィールドは、SQL 文の各状態変化で常に同じですが、複数の文で同じハッシュ値を持つことができます。文がライブラリ・キャッシュから強制的に取り出された後でまたスワップで戻されると、同じ文が Lib_Cache_Addr に複数の値を持つことがあります。Session_Index、Session_Serial、SQL_Text_Hash および Lib_Cache_Addr の組合せは、通常、あるセッションの特定の SQL 文を識別します。Cursor_Number を追加する場合は、セッション内の SQL 文の個々の出現を識別します。

Wait イベント

Wait イベントは、すべての応答に対する待機時間の合計を 100 分の 1 秒単位で示します。Wait イベントに対応付けられた項目を、次にリストします。

Session_Index
Session_Serial
P1
P2
P3
Description
製品間項目 1-7

Wait イベントには明示的な ID がありません。Session_Index、Session_Serial、Description、Timestamp および Timestamp_Nano の組合せで、特定の Wait イベントを一意に識別します。

Parse イベント

Parse イベントは、SQL 文の処理中に解析フェーズの開始と終了を記録します。解析フェーズは、SQL テキストが読み込まれ、各種の構成要素に分解（解析）されるときに開始されます。表とフィールドが識別されるとともに、どのフィールドがソート基準で、どの情報を戻す必要があるかが識別されます。Parse イベントに対応付けられた項目を、次にリストします。

Parse イベントを開始するときの項目	Parse イベントを終了するときの項目
Session_Index	Session_Index
Session_Serial	Session_Serial
イベント統計ブロック	イベント統計ブロック
Cursor_Number	Cursor_Number
リソース項目	Depth
製品間項目 1 ～ 6	Missed
	Oracle_Cmd_Type
	Optimizer_Mode
	Current_UID
	Schema_UID
	SQL_Text_Hash
	Lib_Cache_Addr
	リソース項目

Session_Index、Session_Serial、Cursor_Number および SQL_Text_Hash の組合せで、特定の Parse イベントを一意に識別します。

Execute イベント

Execute イベントは、問合せの計画が実行される場合です。すなわち、解析された入力は、データを取り出すための的確なアクセス方法の判別のために分析され、データは、必要に応じてフェッチできるよう準備されます。Execute イベントに対応付けられた項目を、次にリストします。

Execute イベントを開始するときの項目	Execute イベントを終了するときの項目
Session_Index	Session_Index
Session_Serial	Session_Serial
イベント統計ブロック	イベント統計ブロック
Cursor_Number	Cursor_Number
リソース項目	Depth
	Missed
	Row_Count
	SQL_Text_Hash
	Lib_Cache_Addr
	リソース項目

Session_Index、Session_Serial、Cursor_Number および SQL_Text_Hash の組合せで、特定の Execute イベントを一意に識別します。

Fetch イベント

Fetch イベントは、実際にデータが戻されるイベントです。すべてのデータを取り出すためには同じ文に対して複数のフェッチが実行されます。Fetch イベントに対応付けられた項目を、次にリストします。

Fetch イベントを開始するときの項目	Fetch イベントを終了するときの項目
Session_Index	Session_Index
Session_Serial	Session_Serial
イベント統計ブロック	イベント統計ブロック
Cursor_Number	Cursor_Number
リソース項目	Depth
製品間項目 1 ～ 6	Row_Count

Fetch イベントを開始するときの項目	Fetch イベントを終了するときの項目
	End_of_Fetch
	SQL_Text_Hash
	Lib_Cache_Addr
	リソース項目

Session_Index、Session_Serial、Cursor_Number、SQL_Text_Hash、Timestamp および Timestamp_Nano の組合せで、特定の Fetch イベントを一意に識別します。

LogicalTX イベント

LogicalTX イベントは、論理トランザクションの開始と終了を記録します（発行された文によりデータベース・ステータスが変更される可能性があります）。LogicalTX イベントに対応付けられた項目を、次にリストします。

LogicalTX イベントを開始するときの項目	LogicalTX イベントを終了するときの項目
Session_Index	Session_Index
Session_Serial	Session_Serial
イベント統計ブロック	イベント統計ブロック
TX_Type	TX_Type
TX_SO_Addr	TX_SO_Addr
リソース項目	リソース項目
製品間項目 1 ～ 6	

CROSS_FAC_4 に格納されるトランザクション識別子は、特定のトランザクションを一意に識別するものである必要があります。あるいは、Session_Index、Session_Serial および TX_SO_Addr を使用します。

PhysicalTX イベント

PhysicalTX イベントは、物理トランザクションの開始と終了を記録します（発行された文によりデータベース・ステータスが変更されています）。PhysicalTX イベントに対応付けられた項目を、次にリストします。

PhysicalTX イベントを開始するときの項目	PhysicalTX イベントを終了するときの項目
Session_Index	Session_Index
Session_Serial	Session_Serial
イベント統計ブロック	イベント統計ブロック
TX_Type	TX_Type
TX_ID	TX_ID
リソース項目	Commit_Abort
製品間項目 1 ～ 6	リソース項目

CROSS_FAC_4 に格納されるトランザクション識別子は、特定のトランザクションを一意に識別するものである必要があります。

イベント・セットのファイル名

Oracle Trace イベントは、データ収集を特定のイベントに制限する**イベント・セット**に編成できます。パフォーマンスの監視、監査、診断、または論理イベントのグループ化に対してイベント・セットを設定できます。

表 12-7 サーバー・イベント・セットのファイル名

イベント・セットのファイル名（.fdf）	説明
CONNECT	CONNECT_DISCONNECT イベント・セット データベースとの接続およびデータベースからの切断についての統計を収集します。
ORACLE	ALL イベント・セット 待機イベントを含む Oracle Server 統計を収集します。
ORACLEC	CACHEIO イベント・セット バッファ・キャッシュ I/O のキャッシュ統計を収集します。
ORACLED	Oracle Server DEFAULT イベント・セット Oracle Server 統計を収集します。

表 12-7 サーバー・イベント・セットのファイル名（続き）

イベント・セットのファイル名（.fdf）	説明
ORACLEE	EXPERT イベント・セット Oracle Expert アプリケーションについての統計を収集します。
ORACLESM	SUMMARY イベント・セット Summary Advisor アプリケーションのワークロード統計を収集します。
SQL_ONLY	SQL_TEXT_ONLY イベント・セット データベースとの接続およびデータベースからの切断、および SQL テキストについての統計を収集します。
SQL_PLAN	SQL_STATS_AND_PLAN イベント・セット データベースとの接続およびデータベースからの切断、SQL 統計、SQL テキストおよび行ソース（EXPLAIN PLAN）についての統計を収集します。
SQLSTATS	SQL_AND_STATS イベント・セット SQL テキストと統計のみを収集します。
SQL_TXN	SQL_TXNS_AND_STATS イベント・セット データベースとの接続およびデータベースからの切断、トランザクション、SQL テキストと統計、および行ソース（EXPLAIN PLAN）についての統計を収集します。
SQLWAITS	SQL_AND_WAIT_STATS イベント・セット データベースとの接続、データベースからの切断、行ソース（EXPLAIN PLAN）、SQL テキストと統計、および待機イベントついての統計を収集します。
WAITS	WAIT_EVENTS イベント・セット データベースとの接続、データベースからの切断、および待機イベントについての統計を収集します。

関連項目： 12-3 ページ「イベント・セット」

Oracle Trace のトラブルシューティング

Oracle Trace の使用中に問題をトラブルシューティングするには、次の各項を参考にしてください。

Oracle Trace の構成

Oracle Trace の構成上の問題が考えられる場合は、次のことを行ってください。

- 記録された Oracle Trace エラーの詳細は、EPC_ERROR.LOG ファイルを調べてください。
- $ORACLE_HOME/otrace/admin で管理ファイル（*.dat ファイル）を検索してください。存在しない場合、otrccref を実行して、Oracle Trace *.dat ファイルを再作成してください。
- .fdf ファイルが $ORACLE_HOME/otrace/admin/fdf ディレクトリにあることを確認してください。
- Oracle Trace Collection Services のバージョンが Oracle Server の適切なバージョンと対応しているかどうかを確認してください。

```
% $ORACLE_HOME/bin/otrccol version
```

戻り値	コマンドライン・インタフェース・リリース
1	733
2	803
3	734
4	804
5	805
6	813
7	814
8	815
9	806
10	816
11	817
12	901

- 現在収集を実行中であるかどうかを調べるには、コマンドライン・インタフェースを使用して状態をチェックします。

  ```
  % $ORACLE_HOME/bin/otrccol check collection_name
  ```

CLI をテストするには、次のことを行ってください。

1. CLI は特権付きアカウント、たとえば、Oracle オペレーティング・システムのユーザー・アカウントから実行する必要があります。
2. Oracle ホーム変数と SID 環境変数を正しく設定する必要があります。

   UNIX で設定をチェックするには

   ```
   printenv ORACLE_HOME
   printenv ORACLE_SID
   ```

   UNIX で設定するには

   ```
   setenv ORACLE_HOME path
   setenv ORACLE_SID sid
   ```

   ORACLE_HOME 1 つにつき、CLI は 1 つのみです。たとえば、同じ ORACLE_HOME を Oracle Server リリース 7.3.3 の 2 つのインスタンスで共有する場合でも、CLI は 1 つである必要があります。
3. 収集を開始する前に、収集の名前がまだ使用されていないことを確認してください。

   collection name.cdf および .dat ファイルを検索します。

   - $ORACLE_HOME/otrace/admin/cdf ディレクトリ
   - データベース・パラメータ ORACLE_TRACE_COLLECTION_PATH で指定されたディレクトリ
   - 環境変数 EPC_COLLECTION_PATH で指定されたディレクトリ
4. この収集のためのデータベース・アクティビティを生成する場合は、データベースに接続します。
   - Oracle Server リリース 7.3.x の場合は、収集を作成する前にサービスに接続します。
   - Oracle Server リリース 8.0 の場合は、いつでもデータベースに接続でき、プロセスが登録されます。

サーバー環境

サーバー環境の問題が考えられる場合は、次のことを調べてください。

- サーバー・ノードに収集出力ファイルのための十分なディスク領域があるかどうか。十分なディスク領域がない場合は、収集が停止します。収集のサイズを制限する場合は、Trace のオプションを使用します。収集サイズを制限する方法は、この章の「Oracle Trace CLI の使用」の項を参照してください。

  直接的な問題を解決するには、収集を停止し、Oracle Trace が収集を終了できるように領域を解放します。

  最初に、ユーザーまたは待機イベントを特定して収集を制限することで、収集されるデータ量を容易に制限できます。Oracle Server リリース 8.0.4 以上では、ユーザーと待機イベントの制限が可能です。

- セッションが 6 つ以上の収集に関与していないかどうか。各セッションは、5 つまでの最新の収集データを記録します。したがって、6 つ以上の収集がある場合、最も古い収集に関するデータが失われます。

データの欠落

待機時間が収集されなかった 待機時間は、INIT*sid*.ORA パラメータの TIMED_STATISTICS が true に設定されている場合のみ収集されます。

注意： 初期化パラメータ STATISTICS_LEVEL が TYPICAL または ALL に設定されている場合、時間統計は自動的にデータベースへ収集されます。STATISTICS_LEVEL が BASIC に設定されている場合、TIMED_STATISTICS を TRUE に設定して時間統計の収集を使用可能にする必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を初期化パラメータ・ファイル内で、または ALTER_SYSTEM や ALTER SESSION を使用して明示的に設定した場合、STATISTICS_LEVEL から導出された値は明示的に設定された値によって上書きされます。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

収集での SQL 文の欠落 収集されるはずの SQL 文が表示されない場合、収集が停止されても、Oracle Trace データ収集バッファ内の少量のデータが、収集データ・ファイルにフラッシュされなかった可能性があります。追加のデータベース・アクティビティで、これらのバッファをディスクにフラッシュする必要があります。また、データベースをシャットダウンすることでも、これらのバッファがフラッシュされます。

大きすぎる収集 収集が大きすぎる場合があります。Oracle Server リリース 8.0.4 からは、特定のユーザーおよび特定の待機イベント型のデータのみを収集して、収集のサイズをできるだけ小さくすることができます。ほとんどの場合、サーバーはラッチ、ロック、リソースのいずれかで待機しているので、簡単な収集の待機イベント・データが非常に大きくなる可能性があります。

空の収集 Oracle8 データベース（Oracle 8.1.6 以下）では、サーバー上の `INITsid.ORA` ファイル内の `ORACLE_TRACE_ENABLE` パラメータを、データベースの起動前に `true` に設定する必要があります。Oracle 8.1.7 からは、そのパラメータは動的パラメータであり、`ALTER_SESSION` または `ALTER_SYSTEM` で変更できます。（Oracle7 の場合、init.ora パラメータを使用して収集を開始または停止、`ORACLE_TRACE_ENABLE` パラメータを `false` のままにする必要があります。）同時に実行している収集が多すぎる場合も、この問題が発生することがあります。

Oracle Trace がメモリーにアクセスできなかった

Windows NT システムでは、Oracle Trace 収集を実行して、Oracle Trace がメモリーにアクセスできなかったことを示すエラーが発生した場合は、`collect.dat` ファイルがいっぱいになっています。`$ORACLE_HOME/bin` ディレクトリ内にある `otrccref.exe` イメージを実行して、新しい .dat ファイルを作成する必要があります。これには、データベース・サービスをシャットダウンして、`collect.dat` ファイルを解放し、`otrccref` スクリプトが新しい `collect.dat` ファイルを作成できるようにします。また、`collect.dat` レコード数をデフォルトの 36 より多くすることもできます（たとえば、50 レコードを含む `otrace/admin/collect.dat` ファイルを作成するには `otrccref -c50` を実行します）。

Oracle7 ストアド・プロシージャ

Oracle7 データベースの Oracle Trace データを収集しようとして、メッセージ「error starting or stopping Oracle7 database collection」が発生した場合、Oracle Trace が Oracle7 の収集を開始および停止するためのデータベース・ストアド・プロシージャが欠落している可能性があります。

- ストアド・プロシージャの有無のチェック（Oracle Server リリース 7.3.x の場合）

  Oracle Enterprise Manager のコンソールを使用してストアド・プロシージャの有無をチェックするには、Navigator と次のパスを使用します。

  ```
  Networks > Databases > your_database > Schema Objects > Packages > SYS
  ```

- DBMS_ORACLE_TRACE_xxx から始まるストアド・プロシージャを検索します。

  Oracle Server Manager または Oracle SQL*Plus Worksheet を使用してストアド・プロシージャの有無をチェックするには、次を実行します。

```
select object_name from dba_objects where object_name like '%TRACE%'
and object_type = 'PACKAGE';
OBJECT_NAME
DBMS_ORACLE_TRACE_AGENT
DBMS_ORACLE_TRACE_USER
2 rows selected.
```

Oracle7 の場合、Oracle Trace ではこれらのストアド・プロシージャをデータベースにインストールする必要がありました。これらの SQL スクリプトは、プラットフォーム固有のインストール・プロシージャによりますが、データベースのインストール時に自動的に実行されます。実行されない場合は、これらのスクリプトを手動で実行する必要があります。$ORACLE_HOME/otrace/admin から、特権付きデータベース・アカウント（SYS または INTERNAL）で otrcsvr.sql スクリプトを実行して、これらのストアド・プロシージャをデータベースに追加できます。スクリプトを実行するには、スクリプトがあるパスをデフォルトに設定します。このスクリプトは、パスが指定されていない他のスクリプトを実行します。これらの他のスクリプトは、実行されるディレクトリがカレント・ディレクトリになっていないと、実行できません。

EPC_ERROR.LOG ファイル

EPC_ERROR.LOG ファイルは、収集処理に関する情報の中でも、特に Oracle Trace Collection Services エラーに関する情報を提供します。

EPC_ERROR.LOG ファイルは、現行のデフォルト・ディレクトリに作成されます。

> **関連項目：** 大部分の Oracle Trace メッセージの原因とその対処方法については、『Oracle Enterprise Manager メッセージ・マニュアル』を参照してください。

フォーマッタ表

Oracle Server リリース 7.3.4 および 8.0.4 以上では、フォーマッタ表は自動的に作成されます。Oracle Server リリース 7.3.4 および 8.0.4 より前のリリースでは、データをフォーマットするユーザーとして、Oracle Server Manager または Oracle SQL*Plus Worksheet から otrcfmtc.sql スクリプトを実行する必要があります。otrcfmtc.sql を手動で実行してフォーマッタ表を作成する場合は、収集をフォーマットする Oracle ホームから SQL スクリプトも実行します。

otrcfmtc.sql スクリプトは、$ORACLE_HOME/otrace/admin ディレクトリ内にあります。

フォーマット・エラーは次の原因のいずれかにより発生します。

1. ユーザーがフォーマッタ表を作成するためのスクリプトを実行しませんでした（7.3.4 および 8.0.4 より前の Oracle Server のリリースで有効）。
2. フォーマッタ表が、収集を作成したときのユーザー ID で作成されませんでした（7.3.4 および 8.0.4 より前の Oracle Server のリリースで有効）。

EPC_COLLECTION を検索してください。

SQL Worksheet を使用してフォーマッタ表をチェックするには、次を実行します。

```
CONNECT username/password@service name
DESCRIBE epc_collection
```

第 III 部

優れたパフォーマンスを得るための データベースの作成

第 III 部では、優れたパフォーマンス実現のためのデータベースの構成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。

13

パフォーマンスを考慮したデータベースの作成

データベースおよびOracleインスタンスのどちらも、パフォーマンスの修正は後でも可能ですが、データベース作成時にニーズにあわせてデータベースを慎重に設計することによって、パフォーマンス目標の多くは達成可能です。

この章には次の項があります。

注意： この章では、パフォーマンスを考慮してデータベースを設計するための、オラクル社の新しい方法論の概要を説明しています。この章を読む前に、『Oracle9*i* データベース・パフォーマンス・プランニング』に示されている情報を読んでください。メモリーおよびI/Oの構成の詳細は、第III部のその他の章を参照してください。

初期データベースの作成

データベースの管理における最初の段階の 1 つは、初期データベースの作成です。この項では、Oracle データベースを作成する際の重要なステップを説明します。

インストーラを使用したデータベースの作成

Oracle Installer によって、ソフトウェアのインストール時またはその後に、Database Creation Assistant を使用してデータベースを作成できます。この方法は、中小規模環境のデータベースを作成する効率的な方法であり、単純なグラフィカル・ユーザー・インタフェースを提供します。ただし、このプロシージャは様々なオプションの可能性に対していくつかの制限を課すため、大規模環境におけるデータベースの作成にはお薦めできません。

手動によるデータベース作成

手動アプローチでは、各種の構成オプションを自由に設定できます。このことは特に、大規模環境にとって重要です。一般的に、手動アプローチでは、様々なステージで使用する複数のパラメータ・ファイルの設計、CREATE DATABASE 文および CREATE TABLESPACE 文の SQL スクリプトの実行、必要なデータ・ディクショナリ・スクリプトの実行などの処理を行います。

初期データベースの作成に必要なパラメータ

データベース作成前の初期段階も含め、Oracle インスタンスを起動すると、常に初期化パラメータ・ファイルが読み込まれます。後から変更できないパラメータはごく少数です。データベース作成時に正しく設定する必要がある最も重要なパラメータを、表 13-1 にリストします。

表 13-1 データベース作成時の重要な初期化パラメータ

パラメータ	説明
DB_BLOCK_SIZE	データベース・ファイルに格納され、SGA にキャッシュされる Oracle データベース・ブロックのサイズを設定します。値の範囲はオペレーティング・システムにより異なりますが、一般的に 2 の累乗の 2048 ～ 16384 の範囲です。共通値は、トランザクション処理システムの場合は 4096 または 8192 で、データベース・ウェアハウス・システムの場合はさらに大きい値となります。
DB_NAME および DB_DOMAIN	データベースの名前およびデータベースのドメイン名を、それぞれで設定します。これらのパラメータは後で変更できますが、初期作成前に正しく設定しておくことをお薦めします。選択した名前は、SQL*Net の構成にも反映する必要があります。

表 13-1 データベース作成時の重要な初期化パラメータ（続き）

パラメータ	説明
COMPATIBLE	Oracle サーバーで互換性を保つ必要のあるリリースを指定します。このパラメータを使用すると、新しい機能を自分の環境でテストせずに、ただちに本番システムで新しいリリースで改善された機能を利用できるようになります。アプリケーションは Oracle の特定のリリース用に設計されているが、実際にはそれ以降のリリースをインストールしている場合、このパラメータをアプリケーションが設計された古いリリースのバージョンに設定してください。

注意： ローカル管理 SYSTEM 表領域を確保するには、互換性をリリース 2（9.2）以上に設定する必要があります。CREATE DATABASE 時に EXTENT MANAGEMENT LOCAL を指定すると、データベースを 9.2 以前のリリースに戻すことができなくなり、SYSTEM 表領域をディクショナリ管理の表領域に移行できなくなります。指定後に作成する表領域はすべてローカル管理にする必要があります。

CREATE DATABASE 文

初期インスタンスの起動後に実行される最初の SQL 文は CREATE DATABASE 文です。この文は、初期システム表領域および初期 REDO ログ・ファイルを作成し、一部のデータベース・オプションを設定します。表 13-2 にリストしたオプションは後で変更できないか、変更することが困難なオプションです。

表 13-2 初期作成用データベース・オプション

データベース・オプション	説明
キャラクタ・セット	このオプションで指定されたキャラクタ・セットは、内部データ・ディクショナリの SQL テキストに使用されるキャラクタ・セットを識別しますが、最も重要なのは、データ型 CHAR、VARCHAR または VARCHAR2 として格納されたテキストに使用されるキャラクタ・セットを識別することです。各国語キャラクタを含むデータがロードされた後は、このキャラクタ・セットを変更できません。
各国語キャラクタ・セット	このキャラクタ・セットは、データ型 NCHAR、NVARCHAR および NVARCHAR2 に使用されます。一般的に、通常のキャラクタ・セットの場合と同様、このキャラクタ・セットは変更できません。
SQL.BSQ ファイル	内部データ・ディクショナリを作成します。このファイルの変更の詳細は、第 15 章「I/O 構成および設計」を参照してください。

表 13-2 初期作成用データベース・オプション（続き）

データベース・オプション	説明
初期データ・ファイルの場所	システム表領域を構成する初期データ・ファイルは注意して設定してください。後で設定は変更できますが、その手順にはインスタンスの完全なシャットダウンが含まれています。
EXTENT MANAGEMENT LOCAL	ローカル管理 SYSTEM 表領域を作成するには、EXTENT MANAGEMENT LOCAL 句を使用します。AUTOALLOCATE がデフォルトで、エクステント・サイズはシステムで選択されます。SYSTEM 表領域では EXTENT SIZE UNIFORM は使用できません。
デフォルトの一時表領域	EXTENT MANAGEMENT LOCAL を指定する場合、デフォルトの一時表領域も指定する必要があります。
MAXDATAFILES	データ・ファイルの最大数。
MAXLOGFILES	ログ・ファイルの最大数。

注意： ローカル管理 SYSTEM 表領域を確保するには、互換性をリリース 2（9.2）以上に設定する必要があります。CREATE DATABASE 時に EXTENT MANAGEMENT LOCAL を指定すると、データベースを 9.2 以前のリリースに戻すことができなくなり、SYSTEM 表領域をディクショナリ管理の表領域に移行できなくなります。指定後に作成する表領域はすべてローカル管理にする必要があります。

例 13-1 CREATE DATABASE スクリプトの例

```
CONNECT SYS/ORACLE AS SYSDBA
STARTUP NOMOUNT pfile=/u01/admin/init_create.ora
CREATE DATABASE "dbname"
DATAFILE '/u01/oradata/system01.dbf' size 200M
LOGFILE  '/u02/oradata/redo01.dbf' size 100M,
         '/u02/oradata/redo02.dbf' size 100M
CHARACTER SET "WE8ISO8859P1"
NATIONAL CHARACTER SET "UTF8"
EXTENT MANAGEMENT LOCAL
DEFAULT TEMPORARY TABLESPACE mytemp TEMPFILE 'temp.f' SIZE 1000M
MAXDATAFILES = 50
MAXLOGFILES = 5;
```

関連項目： CREATE DATABASE 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

データ・ディクショナリ・スクリプトの実行

CREATE DATABASE 文を実行した後、特定のカタログ・スクリプトを実行する必要があります。カタログ・スクリプトは、ORACLE_HOME にある UNIX 上の rdbms/admin ディレクトリ内または Windows 上の rdbms¥admin ディレクトリ内にあります。次のスクリプトを実行する必要があります。

- CATALOG.SQL ― すべての正規のデータ・ディクショナリ・ビューに必要です。
- CATPROC.SQL ― 初期の PL/SQL 環境をロードするために必要です。

> **注意：** 特定のオプションまたは機能を使用している場合は（たとえば、Java またはレプリケーション）、さらに多くのスクリプトが必要です。これらのスクリプトは、個々のオプションで記述されます。

例 13-2 必要なデータ・ディクショナリ・スクリプトの実行

```
CONNECT SYS/ORACLE AS SYSDBA
@@CATALOG
@@CATPROC
```

二重のアットマーク（@）を使用すると、適切なディレクトリからこれらのスクリプトが強制的に実行されます。

REDO ログ・ファイルのサイズ指定

REDO ログ・ファイルのサイズは、パフォーマンスに影響を与えます。これは、データベース・ライター・プロセスおよびアーカイバ・プロセスの動作は REDO ログ・サイズに依存するためです。一般的に、大きい REDO ログ・ファイルのほうがパフォーマンスが向上します。小さなログ・ファイルは、チェックポイント・アクティビティを増加し、パフォーマンスを低下させることがあります。高度なパフォーマンスを目的とした I/O 分散では、REDO ログ・ファイル用に個別のディスクを使用することが推奨されるため、REDO ログ・ファイルのサイズを大きくすることが可能です。大きな REDO ログ・ファイルの潜在的な問題点は、REDO ログのミラー化が無効である場合に、これらの大きいファイルが Single Point of Failure（単一障害点）となることです。

REDO ログ・ファイルに特定のサイズを推奨することはできませんが、100MB ～数 GB の範囲内にある REDO ログ・ファイルが妥当であると考えられます。システムが生成する REDO の量に従って、オンライン REDO ログ・ファイルをサイズ設定します。およその目安として、多くても 20 分ごとに 1 回ログを切り替えます。

必要な REDO ログ・ファイルの完全セットは、データベース作成時に作成できます。REDO ログ・ファイルの作成後は、REDO ログ・サイズは変更できません。ただし、後でより大きな新規ファイルを追加でき、この結果元のファイル（小さいファイル）を削除できます。

初期データベースの作成と、CATALOG.SQL ファイルからの必要なデータ・ディクショナリ・ビューのロードを高速化するために、それほど多くのことを行う必要はありません。これらのステップは、相互に順次実行する必要があります。

注意： REDO ログ・ファイルのサイズは LGWR のパフォーマンスに影響を与えませんが、DBWR およびチェックポイントの動作に影響を与える可能性があります。

追加表領域の作成

初期データベースの作成後、いくつかの追加表領域を作成する必要があります。すべてのデータベースには、システム表領域の他に少なくとも 3 つの表領域が必要です。ソートなどに使用される一時表領域、ロールバック・セグメントの格納に使用されたり自動 UNDO 管理セグメントとして指定されるロールバック表領域、および実際にアプリケーションが使用する（少なくとも 1 つの）表領域の 3 つです。ほとんどの場合、アプリケーションにはいくつかの表領域が必要です。多数のデータ・ファイルを持つ非常に大きな表領域の場合は、複数の `ALTER TABLESPACE x ADD DATAFILE Y` 文をパラレルに実行することもできます。

表領域の作成時、表領域を構成するデータ・ファイルは特殊な空のブロック・イメージで初期化されます。`TEMPFILES` は初期化されません。

この初期化は、すべてのデータ・ファイルが、その全体に書込みを行えるようにするためですが、シリアルに行うと、長い処理になる可能性があります。したがって、複数の `CREATE TABLESPACE` 文を同時に実行して、表領域の作成処理を高速化します。13-7 ページの例 13-3 の SQL 文を参照してください。これを行う最も効率的な方法は、使用可能なディスクの各セットに対し 1 つの SQL スクリプトを実行することです。

永続的な表の場合、表領域作成時にローカルまたはグローバルのいずれかのエクステント管理を選択するかで、パフォーマンスに大きな影響を与える可能性があります。読取り操作と比べて、普通または大規模な、挿入、変更または削除操作を行う永続的な表領域の場合は、ローカル・エクステント管理を選択する必要があります。

永続的な表領域の作成ー自動セグメント領域管理

永続的な表領域では、自動セグメント領域管理を使用することをお薦めします。これらの表領域（ビットマップ化表領域と呼ばれる）は、ビットマップ・セグメント領域管理が行われるローカル管理表領域です。Oracle9*i* リリース 1（9.0.1）以上で使用できます。

> **関連項目：** 表領域の自動セグメント領域管理の作成および使用の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

一時表領域の作成

一時表領域はディクショナリ管理を行うこともローカル管理を行うこともできます。Oracle9*i* リリース 1（9.0.1）以上では、ローカル管理の一時表領域を使用することをお薦めします。例 13-3 では、ローカル・エクステント管理の一時表領域の作成方法を示しています。

例 13-3 一時表領域の作成

```
CONNECT SYSTEM/MANAGER
CREATE TABLESPACE appdata DATAFILE
  '/u02/oradata/appdata01.dbf' size 1000M;
CREATE TEMPORARY TABLESPACE mytemp TEMPFILE 'temp.f' SIZE 1000M;
```

別のセッションでは、次のようになります。

```
CONNECT SYSTEM/MANAGER
CREATE TABLESPACE appindex DATAFILE
  '/u03/oradata/appindex01.dbf' size 1000M;
```

適切なパフォーマンスを得る表の作成

アプリケーションをインストールする際、最初のステップで、必要なすべての表および索引を作成します。表のようなセグメントを作成すると、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。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。

表および索引を作成する際には、次の点に注意してください。

- 表領域の自動セグメント領域管理の指定

  最高のパフォーマンスが得られるよう、Oracle によって自動的にセグメント領域が管理されます。

- 記憶領域オプションの注意深い設定

  アプリケーションでは、表または索引の用途によって記憶領域オプションを慎重に設定する必要があります。この設定には、PCTFREE の値の設定が含まれます。自動セグメント領域管理を使用すると、PCTUSED を指定する必要がありません。

注意： 空きリストの使用はお薦めしません。自動セグメント領域管理を使用するには、ローカル管理表領域を作成し、セグメント領域管理の句を AUTO に設定します。

- 必要に応じて、INITRANS 値をさらに高くする設定

  各データブロックには、行のロックを目的として使用される多数のトランザクション・エントリがあります。最初に、この数字は INITRANS パラメータで指定されます。デフォルト値（表の場合は 1、索引の場合は 2）は一般的に十分な値です。ただし、表または索引に 1 ブロック当たり多数の行があり、多数の同時更新の可能性がある場合は、それより高い値を設定することが有効です。この設定は CREATE TABLE/CREATE INDEX の実行時に行い、オブジェクトのすべてのブロックに対し値が設定されていることを確認する必要があります。

  関連項目：

  - Real Application Clusters の記憶域の考慮事項の詳細は、『Oracle9*i* Real Application Clusters 概要』を参照してください。
  - インスタンスおよびユーザーと、空きリストおよび空きリスト・グループとの関連付けの詳細は、『Oracle9*i* Real Application Clusters 管理』を参照してください。
  - マルチブロック・ロック割当ての構成の詳細は、『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』を参照してください。
  - Oracle Real Application Clusters での空きリストおよび空きリスト・グループ使用の詳細は、『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』を参照してください。

注意： 表作成の操作は比較的高速で、パラレルに実行する場合とあまり変わりません。

データ・セグメント圧縮

データ・セグメントの圧縮により、ディスク使用およびメモリー使用（具体的には、バッファ・キャッシュ）を削減します。多くの場合、これにより、読取り専用操作のスケールアップがうまく機能します。また、データ・セグメントの圧縮では、問合せの実行が高速化されます。

チューニングによる圧縮率向上

Oracle9*i* リリース 2（9.2）では、多くのケースで特別なチューニングなしで、優れた圧縮率を達成しています。ただし、さらに圧縮率を上げる必要がある場合にチューニングを行うと、圧縮率が若干改善されるケースもあれば大きく改善されるケースもあります。

ヒープ構成のブロック・レベル圧縮は、個々のブロック内の列値の繰返しを解消することによって機能します。つまり、そのような繰返しの列値を共有ブロック・レベルのシンボル表に移動し、出現個所をシンボル表への参照で置き換えます。したがって、繰返しの値が多いブロックほど圧縮率が高くなります。データベース管理者またはアプリケーション開発者は、繰返しの可能性が高くなるように、圧縮の必要なセグメントの行を再構成することによってその仕組みを利用できます。

単一列の表では、ORDER BY 句を持つ CREATE TABLE AS SELECT を使用して、列値で表の行の順序を指定できます。

この方法は、ある列のカーディナリティが低く、別の列のカーディナリティが高い表にも適用できます。その場合は、カーディナリティの低い列順に表の行の順序を指定します。

次のビューにはセグメント内の列カーディナリティに関する情報が含まれています。

- ALL_TAB_COL_STATISTICS
- ALL_PART_COL_STATISTICS
- ALL_SUBPART_COL_STATISTICS

例 13-4 データ・セグメントの圧縮 / 非圧縮率の見積り

表 table_t の圧縮 / 非圧縮率を見積る手順は次のとおりです。これにより、自動クリーン・アップが可能になります。

1. サンプリングをリピータブルにします。

```
ALTER SESSION SET EVENTS '10193 trace name context forever, level 1';
```

2. 次のボディを使用して、1 分後に DBMS_JOB を起動して実行します（クリーン・アップ手順）。

```
LOCK TABLE table_t$a1 IN EXCLISIVE MODE;
DROP TABLE table_t$a1;
DROP TABLE table_t$a2;
```

3. 圧縮した空の表を作成します。

```
CREATE TABLE table_t$a1 COMPRESS AS SELECT * FROM table_t WHERE ROWNUM < 1;
LOCK TABLE table_t$a1 IN SHARE MODE;
```

4. 圧縮されていない空の表を作成します。

```
CREATE TABLE table_t$a2 NOCOMPRESS AS SELECT * FROM table_t
       WHERE ROWNUM < 1;
INSERT /*+ APPEND */ INTO table_t$a1 SELECT * FROM table_t
       SAMPLE BLOCK(x,y);
INSERT /*+ APPEND */ INTO table_t$a2 SELECT * FROM table_t
       SAMPLE BLOCK(x,y);
```

データ・セグメント圧縮率は、表 table_t$a1 のブロック数を表 table_t$a2 のブロック数で除算します。

注意： 圧縮されていないパーティションのみが含まれているパーティション表に、圧縮されたパーティションを初めて追加する場合は、既存のビットマップ索引およびビットマップ・インデックス・パーティションをすべて削除するか、それらに UNUSABLE のマークを付けた後で圧縮したパーティションを追加する必要があります。これは新しいパーティションにデータが含まれていない場合も同様です。この条件に関連する操作は、パーティションの追加、分割、マージおよび移動です。

関連項目： ブロック・グループ・サンプリング構文 SAMPLE BLOCK(x,y) の例は、『Oracle9*i* SQL リファレンス』を参照してください。

データのロードおよび索引付け

多数のアプリケーションでは、初期のアプリケーション・インストール・プロセスの一部としてデータをロードする必要があります。このデータは、郵便番号やその他の参照データなどの固定データ、または古いシステムで発生する実際のデータです。Oracle の SQL*Loader ツールは、大量のデータをロードする最も効率的な方法です。

適切なパフォーマンスを得る SQL*Loader の使用

SQL*Loader を実行する場合、従来型パスまたはダイレクト・パスを使用するように指定します。従来型パスでは、通常の SQL の INSERT 文を使用してデータを表にロードします。つまり、他のデータベース操作と同時にロードを実行できます。ただし、その場合、ロードは通常の INSERT のパフォーマンスによって制限されます。大量のデータを迅速にロードするには、ダイレクト・パスを選択します。ダイレクト・パスを使用すると、ロード・プロセスは SQL をバイパスし、データベース・ブロックに直接ロードします。このタイプのロード中は、表（またはパーティション表のパーティション）での通常の操作は実行できません。

次のヒントを参考にすると、SQL*Loader を使用してデータ・ロード・プロセスを高速化できます。

- 区切り記号または引用符の付いたフィールドではなく固定長のフィールドを使用します。そうすると、SQL*Loader が入力ファイルの読取りおよび解釈に必要な時間が短縮されます。
- ネットワーク経由ではなくローカルに SQL*Loader を実行します。
- バイナリ・フォーマットではなくテキストとして入力ファイルから読み込まれた数字をロードします。これは特に、計算なしで数字がロードされた場合に高速になります。これは、整数および浮動小数点数値の場合に該当します。
- 非常に大規模なロード操作の場合は、SQL*Loader を使用してパラレルにデータのロードを行います。

> **関連項目：** SQL*Loader の詳細は、『Oracle9*i* データベース・ユーティリティ』を参照してください。

効率的な索引作成

最も効率的な索引の作成方法は、データがロードされた後に索引を作成することです。そうすると、領域管理が大幅に簡単になり、行が挿入されるたびに索引がメンテナンスされることもありません。これは、SQL*Loader により自動的に行われますが、他の方法で初期データをロードする場合は、手動で行う必要があります。また、索引の作成は、CREATE INDEX 文の PARALLEL 句を使用してパラレルで実行します。ただし、SQL*Loader ではこれを実行できないため、データのロード後に索引を手動でパラレルに作成する必要があります。

ソート用データのメモリーの指定

データを含む表での索引の作成中に、データをソートする必要があります。このソートは、すべての使用可能なメモリーをソートに使用する場合に最も高速に行われます。PGA_AGGREGATE_TARGET 初期化パラメータを設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。

関連項目：

- メモリー管理の詳細は、14-46 ページの「PGA 作業メモリーの構成」を参照してください。
- PGA_AGGREGATE_TARGET 初期化パラメータについては、『Oracle9*i* データベース・リファレンス』を参照してください。

SORT_AREA_SIZE による SQL 作業領域のメモリーの指定

SQL 作業領域のメモリーは、SORT_AREA_SIZE 初期化パラメータでも制御できます。

注意： インスタンスが共有サーバー・オプションで設定されない限り SORT_AREA_SIZE パラメータの使用はお薦めしません。そのかわりに、PGA_AGGREGATE_TARGET を設定して、SQL 作業領域の自動サイズ指定を使用可能にすることをお薦めします。SORT_AREA_SIZE は、下位互換性のために残されています。

SORT_AREA_SIZE パラメータの値は、次のルールで設定する必要があります。

1. SGA のサイズおよびオペレーティング・システムのサイズをシステム・メモリーの総量から減算し、使用可能なメモリー量を算出します。
2. このメモリー量を、使用するパラレル・スレーブ数で割ります。通常これは CPU の数と同じです。
3. 通常は 5 ～ 10MB のプロセス・オーバーヘッドを減算し、SORT_AREA_SIZE の値を算出します。

注意： 統計情報を生成しないことによって、索引作成操作または高速な再作成の時間も節約できます。

例 13-5 は、SORT_AREA_SIZE パラメータの設定例です。

例 13-5 効率的な索引の作成例

512 MB のメモリーを持つシステムが 100 MB SGA の Oracle インスタンスを実行し、オペレーティング・システムは 50 MB を使用しているとします。ソートに使用可能なメモリーは 362 MB（512 - 50 - 100）です。システムに 4 つのパラレル・スレーブで実行中の 4 つの CPU がある場合、それぞれ 90 MB のメモリーが使用可能になります。10 MB がプロセス・オーバーヘッド用に確保されているため、SORT_AREA_SIZE を 80 MB に設定する必要があります。この設定は、初期化ファイル内か、または次の文を使用してセッション・ベースで行えます。

```
ALTER SESSION SET SORT_AREA_SIZE = 80000000;
```

初期インスタンス構成

実行中の Oracle インスタンスは、初期化パラメータ・ファイルで設定される起動パラメータを使用して構成されます。これらのパラメータは、パフォーマンスへの影響を含め、実行中のインスタンスの動作に影響を与えます。一般的に、関連する設定をほとんど持たない非常に単純な初期化ファイルでは、ほとんどの状況に対応できます。表 13-4 に示すきわめて少数のパラメータを除き、初期化ファイルはパフォーマンス・チューニングを行う最初の場所ではありません。

関連項目： 第 14 章「メモリーの構成と使用方法」

表 13-3 に、最小初期化ファイルで必要なパラメータを示します。これらのパラメータは必須ですが、パフォーマンスに影響を与えません。

表 13-3 パフォーマンスに影響を与えない必要な初期化パラメータ

パラメータ	説明
DB_NAME	データベースの名前。これは、環境変数 ORACLE_SID に一致する必要があります。
DB_DOMAIN	インターネット・ドット表記法による、データベースの場所。
OPEN_CURSORS	1 セッション当たりのカーソル（アクティブな SQL 文）の最大数に対する制限。設定はアプリケーションにより異なり、多くの場合は、デフォルトで十分です。

表 13-3 パフォーマンスに影響を与えない必要な初期化パラメータ（続き）

パラメータ	説明
CONTROL_FILES	異なるディスク・ドライブ上に少なくとも 2 つのファイルを含めるように設定して、制御ファイルの消失による障害を防ぎます。
DB_FILES	データベースに割り当てることのできるファイルの最大数に設定します。

表 13-4 に、パフォーマンスの考慮事項として設定する、最も重要なパラメータを示します。

表 13-4 パフォーマンスに影響を与える重要な初期化パラメータ

パラメータ	説明
DB_BLOCK_SIZE	データベース・ブロックのサイズを設定します。
DB_CACHE_SIZE	SGA 内のバッファ・キャッシュのサイズ。値を設定するための単純で優れたルールはありません。この値はアプリケーションにより非常に異なりますが、一般的な値はユーザー・セッション当たり 20 ～ 50 の範囲内です。この値は、低めよりも高めに設定するのが一般的です。DB_BLOCK_BUFFERS は使用されなくなりました。
SHARED_POOL_SIZE	SGA 内の共有プールのサイズを設定します。この設定はアプリケーションにより異なりますが、一般的にユーザー・セッション当たり数 MB から数 10MB の範囲内にあります。
PROCESSES	そのインスタンスで起動できるプロセスの最大数を設定します。このパラメータは、他のパラメータ値の多くがこのパラメータから導出されるため、設定する最も重要なプライマリ・パラメータとなります。
SESSIONS	これは、デフォルトで PROCESSES の値から設定されます。ただし、共有サーバーを使用する場合は、導出された値では不十分である可能性があります。
JAVA_POOL_SIZE	Java ストアド・プロシージャを使用する場合は、Java 環境での実際のメモリー要件に基づいてこのパラメータを設定する必要があります。
LOG_ARCHIVE_XXX	REDO ログのアーカイブを使用可能にします。『Oracle9*i* ユーザー管理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。
ROLLBACK_SEGMENTS	このインスタンスに 1 つ以上のロールバック・セグメントを名前で割り当てます。このパラメータを設定すると、ロールバック・セグメント数がインスタンスから要求された最小数（TRANSACTIONS /TRANSACTIONS_PER_ROLLBACK_SEGMENT として計算された数値）を超過する場合でも、インスタンスはこのパラメータで指定されたすべてのロールバック・セグメントを取得します。

最小初期化ファイルの例

多くの場合、Oracle インスタンスを適切にチューニングするためには、次の例のパラメータのみを適切な値に設定する必要があります。ここでは、最小初期化ファイルの例を示します。

```
DB_NAME = finance
DB_DOMAIN = hq.company.com
CONTROL_FILES = ('/u01/database/control1.dbf', '/u02/database/control2.dbf')
DB_BLOCK_SIZE = 8192
DB_BLOCK_BUFFERS = 12000 # this is approximately 100 Mb
DB_FILES = 200 # Maximum 200 files in the database
SHARED_POOL_SIZE = 100000000 # 100 Mb
PROCESSES = 80 # Would be good for approximately 70
# directly connected users
# log_archive_XXX
# Set various archiving parameters
```

UNDO 領域の構成

読込み一貫性、リカバリおよび実際のロールバック文の情報を保持するために、UNDO 領域が必要です。この情報は、ロールバック・セグメント内または 1 つ以上の自動 UNDO 管理表領域内に保持されます。

V$UNDOSTAT ビューには、UNDO 領域を監視およびチューニングするための統計が含まれています。このビューを使用して、現行のワークロードに必要な UNDO 領域の大きさをより的確に見積ることができます。この情報を使用して、システム内の UNDO 使用量を簡単にチューニングすることもできます。V$ROLLSTAT ビューには、UNDO 表領域内の UNDO セグメントの動作に関する情報が含まれています。

関連項目：

- ロールバック・セグメントまたは自動 UNDO 管理を使用した UNDO 領域の管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- V$UNDOSTAT ビューおよび V$ROLLSTAT ビューの詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

オペレーティング・システム、データベースおよびネットワーク監視の設定

パフォーマンスの問題を効果的に診断するには、後でシステムの動作状態が悪い場合に比較するためのパフォーマンス・ベースラインを確立しておくことが不可欠です。ベースライン・データ・ポイントがないと、新しい問題を識別することが非常に困難になる可能性があります。たとえば、システム上のトランザクション量の増加、トランザクション・プロファイルやアプリケーションの変更、またはユーザー数の増加などです。

データベースおよびテーブルが作成され、データがロードおよび索引付けされ、インスタンスが構成された後、監視プロシージャを設定します。

関連項目： 第 20 章「データベース統計収集用の Oracle のツール製品」

14

メモリーの構成と使用方法

Oracle メモリー・キャッシュを適切にサイズ設定して効率的に使用すると、データベースのパフォーマンスが大幅に向上します。この章では、Oracle メモリー・キャッシュにメモリーを割り当てる方法とこれらのキャッシュの使用方法について説明します。

この章には次の項があります。

メモリー割当ての問題について

Oracle はメモリー・キャッシュ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・アクセスよりはるかに高速です。ディスク・スキャン（物理 I/O）は、メモリー・アクセスに比べ、時間がかかります（通常は 10 ミリ秒のオーダー）。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。このため、頻繁にアクセスされるオブジェクトに対するデータ要求は、ディスク・アクセスではなく、メモリー・アクセスで要求するほうが効率的です。

パフォーマンスの目標は、必要なデータがメモリー内にある可能性を高くしたり、必要なデータを取り出すプロセスをさらに効率的にし、できるだけ多くの物理 I/O オーバーヘッドを削減することです。

Oracle メモリー・キャッシュ

パフォーマンスに影響を与える主な Oracle メモリー・キャッシュは次のとおりです。

- 共有プール
- ラージ・プール
- Java プール
- バッファ・キャッシュ
- ログ・バッファ
- プロセス・プライベート・メモリー（たとえば、ソート、ハッシングなどに使用）

これらのメモリー・キャッシュのサイズは、初期化構成パラメータで構成できます。これらのパラメータの値も、ALTER SYSTEM 文（起動後に静的に設定されるログ・バッファおよび Java プールを除く）で動的に構成できます。

キャッシュ・サイズの動的な変更

共有プール、ラージ・プール、バッファ・キャッシュおよびプロセス・プライベート・メモリーのサイズは動的に再構成できます。

共有プール、ラージ・プール、Java プールおよびバッファ・キャッシュのメモリーは、グラニュル単位で割り当てられます。一般的に、ほとんどのプラットフォームで、SGA の総サイズが 128MB 未満の場合、グラニュルのサイズは 4MB で、それより大きい SGA の場合は 16MB となります。プラットフォームへの依存性が多少あります。たとえば、32 ビットの Windows NT では、128MB より大きい SGA の場合、グラニュルのサイズは 8MB になります。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。

SGA で現在使用されているグラニュルのサイズは、ビュー V$SGA_DYNAMIC_COMPONENTS によって表示できます。それと同じグラニュルのサイズが SGA のすべての動的コンポーネントで使用されます。

必要であれば、1 つのキャッシュのサイズを減らし、そのメモリーを別のキャッシュに再度割り当てることが可能です。SGA の総サイズは、SGA_MAX_SIZE パラメータに等しい値に拡張できます。

注意： SGA_MAX_SIZE は、動的にサイズ変更できません。

インスタンスで使用できる最大メモリー量は、インスタンス起動時に SGA_MAX_SIZE 初期化パラメータで決定されます。SGA_MAX_SIZE をすべてのメモリー・コンポーネント（バッファ・キャッシュや共有プールなど）の合計より大きいサイズに指定できます。指定しない場合は、SGA_MAX_SIZE は、これらのコンポーネントで使用されるデフォルトの実際のサイズに設定されます。すべてのコンポーネントで使用される合計メモリーよりも大きい値に SGA_MAX_SIZE を設定すると、別のキャッシュのサイズを小さくせずにキャッシュ・サイズを動的に大きくできます。

動的サイズ変更操作に関する情報の表示

Oracle9*i* リリース 2（9.2）以上では、動的 SGA サイズ変更操作に関する情報を提供するビューが用意されています。

- V$SGA_CURRENT_RESIZE_OPS: 現在進行中の SGA サイズ変更操作に関する情報。動的 SGA コンポーネントの拡張または縮小操作があります。
- V$SGA_RESIZE_OPS: 実行済みの最新の 100 件の SGA サイズ変更操作に関する情報。これには現在進行中の操作は含まれません。
- V$SGA_DYNAMIC_COMPONENTS: SGA の動的コンポーネントに関する情報。このビューでは、起動後のすべての実行済み SGA サイズ変更操作に基づく情報が要約されます。
- V$SGA_DYNAMIC_FREE_MEMORY: 今後の動的 SGA サイズ変更操作で使用可能な SGA メモリーの量に関する情報。

関連項目：

- 動的 SGA の詳細は、『Oracle9*i* データベース概要』を参照してください。
- これらのビューの列の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

アプリケーションの考慮事項

メモリー構成では、アプリケーションの要求に適したキャッシュをサイズ設定することが重要です。逆に、アプリケーションのキャッシュの使用率をチューニングすると、リソース要件を大幅に削減できます。Oracle メモリー・キャッシュを効率的に使用すると、これらのキャッシュを保護するラッチ、CPU、I/O システムなどの関連リソースに対する負荷も削減できます。

最高のパフォーマンスを得るために、次のことを考慮してください。

- アプリケーションを、効率的に Oracle と動作するように設計し、コーディングする必要があります。
- Oracle メモリー構造に対するメモリーの割当ては、アプリケーションの要求を最もよく反映する必要があります。

既存のアプリケーションに対する変更または追加を行う場合は、変更されたアプリケーションの要求を満たすために Oracle メモリー構造のサイズ変更が必要な場合があります。

オペレーティング・システムのメモリー使用量

大半のオペレーティング・システムでは、次のことを考慮することが重要です。

ページングの削減

ページングは、新しいページをメモリーにロードできるようにするため、オペレーティング・システムがメモリー常駐ページをディスクに転送する場合に行われます。多くのオペレーティング・システムは、実メモリーに格納しきれない大量の情報を収容するために、ページングを行います。大半のオペレーティング・システムでは、ページングはパフォーマンスを低下させます。

オペレーティング・システムのユーティリティを使用して、オペレーティング・システムを調べ、システム上に多数のページングがあるかどうかを確認します。ページングが多数ある場合は、システム上の総メモリー量が、メモリーを割り当てたすべてを保持できるほど十分に大きくない場合があります。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。

主メモリーへの SGA の格納

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 は主メモリー内に存在する必要があります。SGA のページがディスクにスワップされると、データに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、ページングによる損失は、大規模な SGA がもたらす利益をかなり上回ります。

注意： LOCK_SGA パラメータを使用すると、SGA が物理メモリーにロックされるため、ページ・アウトを防止できます。

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するには、次の SQL*Plus 文を入力します。

```
SHOW SGA
```

この文の出力例を次に示します。

```
Total System Global Area  840205000 bytes
Fixed Size                   279240 bytes
Variable Size             520093696 bytes
Database Buffers          318767104 bytes
Redo Buffers                1064960 bytes
```

個々のユーザーへの十分なメモリーの割当て

SGA をサイズ設定する場合は、個々のサーバー・プロセスとその他のプログラムがシステム上で作動するように十分なメモリーを使用できるようにします。

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。

構成の繰返し

メモリーの割当てを構成する場合は、アプリケーションの要求により異なりますが、Oracle メモリー構造に使用可能なメモリーを配分します。Oracle の構造にメモリーを配分すると、Oracle が動作するために必要な物理 I/O の量に影響を与える可能性があります。最初にメモリーを適切に構成すると、I/O システムが効果的に構成されているかどうかも表示されます。

プロセスをひと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小さくすると、共有プールなど別のメモリー構造のサイズを大きくすることができます。

バッファ・キャッシュの構成と使用方法

様々なタイプの操作について、Oracle ではバッファ・キャッシュを使用してディスクから読み取られたブロックを格納します。ソートやパラレル読込みなどの特定操作の場合には、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、この項では次の項目を説明します。

- バッファ・キャッシュの効果的な使用
- バッファ・キャッシュのサイズ設定
- バッファ・キャッシュ・アドバイザ統計の解釈および使用方法
- 複数バッファ・プールについて

バッファ・キャッシュの効果的な使用

バッファ・キャッシュを効果的に使用するには、不要なリソース使用を回避するようにアプリケーションの SQL 文をチューニングする必要があります。これを確認するには、頻繁に実行される SQL 文と、多数のバッファ取得を実行する SQL 文がチューニングされたかどうかを検証します。

> **関連項目：** この確認を行う方法の詳細は、第 6 章「SQL 文の最適化」を参照してください。

バッファ・キャッシュのサイズ設定

新規にインスタンスを構成する場合は、バッファ・キャッシュの適切なサイズがわかっていません。通常、データベース管理者はキャッシュ・サイズの最初の見積りを行い、次にインスタンス上で代表的なワークロードを実行し、関連する統計を調べて、キャッシュが構成過小か構成過大かを調べます。

バッファ・キャッシュ・アドバイザの統計

多数の統計が、バッファ・キャッシュ・アクティビティの調査に使用できます。これらの統計には次のものがあります。

- V$DB_CACHE_ADVICE
- バッファ・キャッシュ・ヒット率

V$DB_CACHE_ADVICE の使用 このビューは、DB_CACHE_ADVICE パラメータが ON に設定されるときに移入されます。このビューは、潜在的なバッファ・キャッシュ・サイズ範囲のシミュレーションによるミス率を示します。

このビューには、シミュレートされたキャッシュ・サイズのそれぞれの独自の行と、そのキャッシュ・サイズに対して発生すると予測された物理 I/O アクティビティがあります。

DB_CACHE_ADVICE パラメータは動的であるため、特定のワークロードのアドバイザ・データを収集できるように、アドバイザ機能を動的に有効にしたり、無効にできます。

このアドバイザ機能に関連する小さなオーバーヘッドには、次の 2 つがあります。

- CPU: アドバイザ機能を有効にすると、追加の記録が必要なため、CPU の使用量はわずかに増加します。
- メモリー : アドバイザ機能には、共有プール（バッファ当たり約 100 バイト）からメモリーを割り当てる必要があります。

Oracle9*i* リリース 2（9.2）では、DBA ベースのサンプリングを使用して、キャッシュ・アドバイザ統計を収集します。サンプリングを使用すると、ブックキーピングに関連する CPU およびメモリーのオーバーヘッドが大幅に減少します。サンプリングは、開始時のバッファの数が少ないバッファ・プールでは使用しません。

V$DB_CACHE_ADVICE を使用するには、パラメータ DB_CACHE_ADVICE を ON に設定し、インスタンス上で代表的なワークロードを実行するようにします。V$DB_CACHE_ADVICE ビューを問い合せる前にワークロードを安定化できるようにします。

次の SQL 文は、様々なキャッシュ・サイズについてデフォルト・バッファ・プールに対する I/O 要件の予測を戻します。

```
COLUMN size_for_estimate          FORMAT 999,999,999,999 heading 'Cache Size (MB)'
COLUMN buffers_for_estimate       FORMAT 999,999,999 heading 'Buffers'
COLUMN estd_physical_read_factor  FORMAT 999.90 heading 'Estd Phys|Read Factor'
COLUMN estd_physical_reads        FORMAT 999,999,999 heading 'Estd Phys| Reads'

SELECT size_for_estimate, buffers_for_estimate, estd_physical_read_factor, estd_physical_reads
   FROM V$DB_CACHE_ADVICE
   WHERE name          = 'DEFAULT'
     AND block_size    = (SELECT value FROM V$PARAMETER WHERE name = 'db_block_size')
     AND advice_status = 'ON';
```

次の出力は、キャッシュが現行のサイズ 304MB ではなく 212MB だった場合、物理読込みの予測追加数が 1700 万（17,850,847）を超えることを示しています。現行のサイズよりキャッシュ・サイズを大きくしても、大きな利点にはなりません。

```
                                  Estd Phys    Estd Phys
 Cache Size (MB)      Buffers Read Factor        Reads
---------------- ------------ ----------- ------------
              30        3,802       18.70  192,317,943      現行サイズの 10%
              60        7,604       12.83  131,949,536
              91       11,406        7.38   75,865,861
             121       15,208        4.97   51,111,658
             152       19,010        3.64   37,460,786
             182       22,812        2.50   25,668,196
             212       26,614        1.74   17,850,847
             243       30,416        1.33   13,720,149
             273       34,218        1.13   11,583,180
```

```
304        38,020         1.00    10,282,475      現行のサイズ
334        41,822          .93     9,515,878
364        45,624          .87     8,909,026
395        49,426          .83     8,495,039
424        53,228          .79     8,116,496
456        57,030          .76     7,824,764
486        60,832          .74     7,563,180
517        64,634          .71     7,311,729
547        68,436          .69     7,104,280
577        72,238          .67     6,895,122
608        76,040          .66     6,739,731      現行サイズの 200%
```

このビューは、潜在的な各キャッシュ・サイズの物理読込み数を予測する情報を提供して、キャッシュのサイズ設定を支援します。このデータには物理読込みファクタが含まれています。これは、バッファ・キャッシュが所定の値にサイズ変更された場合、現行の物理読込み回数がその分のみ変化すると予測されるファクタです。

注意： Oracle では、物理読込みは必ずしもディスク読込みを意味しません。物理読込みは、ファイル・システム・キャッシュからで済む場合があります。

キャッシュ内でのブロックの検出成功とキャッシュのサイズ間の関係は、必ずしも滑らかな分布を示しません。バッファ・プールをサイズ設定するときは、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。14-9 ページの図 14-1 の例では、キャッシュ・サイズの増分の狭い帯状部分のみが考慮に値することを示しています。

図 14-1 物理 I/O とバッファ・キャッシュ・サイズ

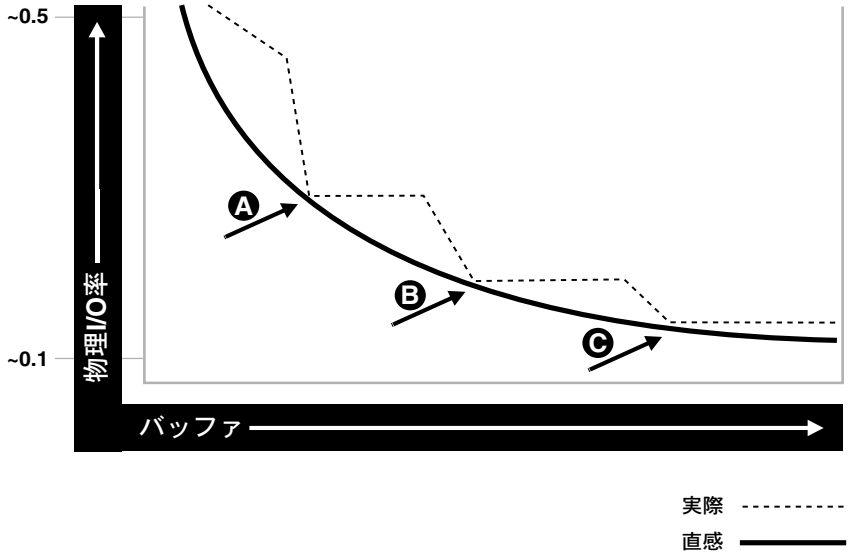


図 14-1 を調べると、次のことがわかります。

- ポイント A からポイント B へバッファを増やす場合の利点は、ポイント B からポイント C へバッファを増やす場合よりかなり大きくなります。
- ポイント A と B およびポイント B と C との間の物理 I/O の減少は、グラフの点線で示されるように滑らかではありません。

バッファ・キャッシュ・ヒット率の計算 バッファ・キャッシュ・ヒット率では、ディスク・アクセスを行わずにバッファ・キャッシュ内で要求されたブロックが検出された頻度を計算します。この率は、動的なパフォーマンス・ビュー V$SYSSTAT から選択したデータを使用して計算されます。バッファ・キャッシュ・ヒット率を使用して、V$DB_CACHE_ADVICE で予測されたように物理 I/O を検証できます。

表 14-1 の統計は、ヒット率の計算に使用されます。

表 14-1 ヒット率を計算するための統計

統計	説明
session logical reads	メモリー内かディスク上にあるブロックにアクセスする要求の総数。
physical reads	ディスク上のデータ・ファイルにアクセスする結果となった、データ・ブロックへのアクセス要求の総数。ブロックは、キャッシュへの読込み、またはローカル・メモリーへのダイレクト読込みで読み込むことが可能。
physical reads direct	バッファ・キャッシュをバイパスし、ラージ・オブジェクト（LOB）のダイレクト読込みを外した、読み込まれたブロック数。
physical reads direct (lob)	バッファ・キャッシュをバイパスして、LOB の読込み中に読み込まれたブロック数。

例 14-1 は V$SYSSTAT 表から直接選択した値を使用して単純化したもので、ある期間の値を選択したものではありません。アプリケーションの実行中のある期間にわたるこれらの統計の差分を計算し、それらの統計を使用してヒット率を判断することが最良の方法です。

> **関連項目：** ある期間内の統計の収集の詳細は、第 20 章「データベース統計収集用の Oracle のツール製品」を参照してください。

例 14-1 バッファ・キャッシュ・ヒット率の計算

```
SELECT NAME, VALUE
  FROM V$SYSSTAT
WHERE NAME IN ('session logical reads','physical reads',
               'physical reads direct','physical reads direct (lob)');
```

この問合せの出力例を次に示します。

```
NAME                                                                  VALUE
---------------------------------------------------------------- ----------
session logical reads                                             464905358
physical reads                                                     10380487
physical reads direct                                                 86850
physical reads direct (lob)                                               0
```

次の公式でバッファ・キャッシュ・ヒット率を計算してください。

```
Hit Ratio = 1 - ((physical reads - physical reads direct - physical reads direct (lob)) /
(db block gets + consistent gets - physical reads direct - physical reads direct (lob))
```

この例のサンプル統計に基づくと、バッファ・キャッシュ・ヒット率は .978 または 97.8% になります。

バッファ・キャッシュ・アドバイザ統計の解釈および使用方法

バッファ・キャッシュ・サイズの増減を考慮する前に、調べるファクタは多数あります。たとえば、V$DB_CACHE_ADVICE データおよびバッファ・キャッシュ・ヒット率を調べる必要があります。

低いキャッシュ・ヒット率は、キャッシュのサイズを大きくすることがパフォーマンスに有益であることを意味しません。キャッシュ・ヒット率がよいことが、ワークロードに対してキャッシュが適切にサイズ設定されていることを示しているとはかぎりません。

バッファ・キャッシュ・ヒット率を解釈する場合は、次の点を考慮する必要があります。

- 大きな表や索引を繰り返しスキャンすると、キャッシュ・ヒット率を低下させる可能性があります。バッファ取得数が多く、頻繁に実行される SQL 文を調べて、実行計画が最適なものであるか確認します。可能であれば、1 つのパスですべての処理を実行するか、SQL 文を最適化して、頻繁にアクセスされるデータを繰り返しスキャンしないようにします。
- 可能であれば、頻繁にアクセスされるデータをクライアント・プログラムまたは中間層にキャッシュして、同じデータを再問合せしないようにします。
- 長い全表スキャンで検出されたブロックは、最低使用頻度（LRU）リストの先頭には配置されません。このようにして、これらのブロックは、索引参照または小規模な表スキャンを実行するときに読み込まれるブロックよりも早く除去されます。このように、妥当な大規模全表スキャンが発生している場合の低いヒット率は、バッファ・キャッシュ・データの解析時にも考慮する必要があります。

  注意： 小規模表のスキャンは、一定のサイズのしきい値を使用して、表に対して実行されるスキャンです。小規模表とは、最大でバッファ・キャッシュの 2% か 20 のいずれかの、大きい方と定義されます。

- 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、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。
- バッファ・キャッシュ・サイズを継続して増やすことはよくある間違いです。全表スキャンや、バッファ・キャッシュを使用しない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。

バッファ・キャッシュに割り当てられたメモリーの増加

一般規則として、キャッシュ・ヒット率が低く、全表スキャンを実行しないようにアプリケーションがチューニングされている場合は、キャッシュのサイズを増やすことを検討してください。

キャッシュ・サイズを増やすには、まず DB_CACHE_ADVICE パラメータを ON に設定し、キャッシュ統計を安定させます。V$DB_CACHE_ADVICE ビュー内のアドバイザ・データを調べて、実行する物理 I/O の量を大幅に減少させるために必要な次の増分を決定します。ホスト・オペレーティング・システムにページングさせずに必要な余分なメモリーをバッファ・キャッシュに割り当てることができる場合は、このメモリーを割り当てます。バッファ・キャッシュに割り当てられたメモリーの量を増やすには、DB_CACHE_SIZE パラメータの値を増やします。

必要であれば、インスタンスをシャットダウンせずに、バッファ・プールを動的にサイズ変更してこの変更を行います。

注意： キャッシュをサイズ変更すると、DB_CACHE_ADVICE が OFF に設定されます。また、V$DB_CACHE_ADVICE はキャッシュの古い値に対する忠告を示します。この値は、DB_CACHE_ADVICE が明示的に READY または ON に設定しなおされるまで変化しません。

DB_CACHE_SIZE パラメータは、データベースの標準ブロック・サイズのデフォルト・キャッシュのサイズを指定します。データベースの標準ブロック・サイズとは異なるブロック・サイズを持つ表領域を作成して使用するには（トランスポータブル表領域をサポートする場合など）、使用するブロック・サイズごとに個別のキャッシュを構成する必要があります。DB_*n*K_CACHE_SIZE パラメータを使用して、必要な標準以外のブロック・サイズを構成できます（*n* は 2、4、8、16 または 32 のいずれかで、*n* は標準ブロック・サイズではありません）。

注意： キャッシュ・サイズを選択するプロセスは、キャッシュがデフォルトの標準ブロック・サイズ・キャッシュ、KEEP または RECYCLE キャッシュ、標準以外のブロック・サイズ・キャッシュのいずれかにかかわらず同様です。

関連項目： DB_*n*K_CACHE_SIZE パラメータの使用方法の詳細は、『Oracle9*i* データベース・リファレンス』および『Oracle9*i* データベース管理者ガイド』を参照してください。

バッファ・キャッシュに割り当てられたメモリーの削減

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。V$DB_CACHE_ADVICE データをチェックして、キャッシュ・サイズを削減すると物理 I/O 数が大幅に増えるかどうかを調べます。物理 I/O への影響がなければ、別のメモリー構造にメモリーを必要とする場合に、キャッシュ・サイズを削減しても、良好なパフォーマンスを維持できます。バッファ・キャッシュを小さくするには、DB_CACHE_SIZE パラメータの値を変更してキャッシュのサイズを削減します。

複数バッファ・プールについて

一般に、ほとんどのシステムでは 1 つのデフォルト・バッファ・プールが適切です。ただし、アプリケーションのバッファ・プールについて詳しい知識を持つユーザーは、複数バッファ・プールを構成すると有益な場合があります。

非定型アクセス・パターンを持つセグメントの場合、それらのセグメントからのブロックを 2 つの異なるバッファ・プールである KEEP プールと RECYCLE プールに格納します。セグメントのアクセス・パターンは、常にアクセスされるか（すなわち、ホット）、またはほとんどアクセスされない（たとえば、1 日に 1 回のみバッチ・ジョブでアクセスされる大きなセグメント）というように、非定型である可能性があります。

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・キャッシュ内の頻繁にアクセスされるセグメン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領域を不必要に占有す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関連付けられると、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。特定のバッファ・プールに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。デフォルト・バッファ・プールのサイズは、DB_CACHE_SIZE です。各バッファ・プールは、同じ LRU 置換方針を使用します（たとえば、KEEP プールがそのプールに割り当てられたすべてのセグメントを格納するほど十分大きくない場合、最も古いブロックがキャッシュから除去されます）。

オブジェクトを適切なバッファ・プールに割り当てると、次の操作を実行できます。

- I/O の低減または排除
- 個別のキャッシュに対するオブジェクトの隔離または制限

大きいセグメントへのランダム・アクセス

非常に大きいセグメントに大きい索引レンジ・スキャンまたは非有界索引レンジ・スキャンでアクセスすると、LRU 除外方法では問題が発生する可能性があります。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。非順次物理読込みのかなりの割合（10% を超える割合）を 1 つのセグメントが占有する場合、そのセグメントは大規模であると考えられます。大規模セグメントに対するランダム読込みは、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、バッファが頻繁にアクセス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込みの影響を受けません。ただし、その問題は、大規模セグメントの読込みによるバッファの除外を免れるほど頻繁にはアクセスされないウォーム・セグメントに影響を与えます。この問題を解決するオプションは、次の 3 つです。

1. アクセスされたオブジェクトが索引である場合は、索引に選択性があるかどうかを調べます。選択性がない場合は、さらに選択性のある索引を使用するように SQL 文をチューニングします。
2. SQL 文をチューニングすると、大きいセグメントを個別の RECYCLE キャッシュに移動できるので、その他のセグメントに影響を与えません。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利用する必要があります。
3. 大規模セグメントではまったく使用されない別の KEEP キャッシュに小さなウォーム・セグメントを移動する方法もあります。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、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。

Oracle Real Application Cluster のインスタンス

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。それぞれのアプリケーション要件に従って、各インスタンスをチューニングします。

複数バッファ・プールの使用方法

オブジェクトのデフォルト・バッファ・プールを定義するには、STORAGE 句の BUFFER_POOL キーワードを使用します。この句は、CREATE TABLE および ALTER TABLE、CLUSTER、および INDEX の各 SQL 文に有効です。バッファ・プールを指定すると、そのオブジェクトに対して読み込まれたブロックは、すべてそのプールに配置されます。

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かぎり、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。

> **関連項目：** STORAGE 句での BUFFER_POOL の指定に関しては、『Oracle9*i* SQL リファレンス』を参照してください。

V$DB_CACHE_ADVICE 内のバッファ・プール・データ

V$DB_CACHE_ADVICE を使用すると、インスタンス上に構成されたすべてのプールをサイズ設定できます。初期キャッシュ・サイズを見積もり、代表的なワークロードを実行し、次に使用する必要のあるプールの V$DB_CACHE_ADVICE ビューを単純に問い合せます。

たとえば、KEEP プールからデータを問い合せるには、次のようにします。

```
SELECT SIZE_FOR_ESTIMATE, BUFFERS_FOR_ESTIMATE, ESTD_PHYSICAL_READ_FACTOR, ESTD_PHYSICAL_READS
  FROM V$DB_CACHE_ADVICE
    WHERE NAME          = 'KEEP'
     AND BLOCK_SIZE    = (SELECT VALUE FROM V$PARAMETER WHERE NAME = 'db_block_size')
     AND ADVICE_STATUS = 'ON';
```

バッファ・プール・ヒット率

V$SYSSTAT のデータは、すべてのバッファ・プールに対する論理読込みと物理読込みを 1 つの統計セットとして表します。バッファ・プールのヒット率を個々に決定するには、V$BUFFER_POOL_STATISTICS ビューを問い合せる必要があります。このビューは、論理読込みと論理書込みの回数に関するプールごとの統計をメンテナンスします。

バッファ・プール・ヒット率は、次の計算式を使用して決定できます。

```
hit ratio = 1 - [physical reads/(block gets + consistent gets)]
```

物理読込み（*physical reads*）、ブロック取得（*block gets*）および一貫取得（*consistent gets*）の値は、次の問合せで取得できます。

```
SELECT NAME, PHYSICAL_READS, DB_BLOCK_GETS, CONSISTENT_GETS,
      1 - (PHYSICAL_READS / (DB_BLOCK_GETS + CONSISTENT_GETS)) "Hit Ratio"
  FROM V$BUFFER_POOL_STATISTICS;
```

プール内に多くのバッファを持つセグメントの判断

V$BH ビューは、SGA 内に現在常駐するすべてのブロックのデータ・オブジェクト ID を示します。プール内にバッファを多く持つセグメントを判断するには、この項で説明する 2 つの方法のいずれかを使用します。すべてのセグメントのバッファ・キャッシュ使用パターンを確認する（方法 1）か、特定のセグメントの使用パターンを調べます（方法 2）。

方法 1 次の問合せでは、ある時点でバッファ・キャッシュ内に常駐するすべてのセグメントのブロック数をカウントします。バッファ・キャッシュ・サイズによっては、このカウントに多数のソート領域を必要とする可能性があります。

```
COLUMN object_name FORMAT a40
COLUMN number_of_blocks FORMAT 999,999,999,999

SELECT o.object_name, COUNT(1) number_of_blocks
  FROM DBA_OBJECTS o, V$BH bh
```

```
 WHERE o.object_id  = bh.objd
   AND o.owner     != 'SYS'
 GROUP BY o.object_name
 ORDER BY count(1);

OBJECT_NAME                            NUMBER_OF_BLOCKS
-------------------------------------- ----------------
OA_PREF_UNIQ_KEY                                      1
SYS_C002651                                           1
..
DS_PERSON                                            78
OM_EXT_HEADER                                       701
OM_SHELL                                          1,765
OM_HEADER                                         5,826
OM_INSTANCE                                      12,644
```

方法 2 次の手順に従って、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決定してください。

1. 次の問合せを入力して、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 DBA_OBJECTS
    WHERE OBJECT_NAME = UPPER('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します。

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_value;
   ```

   data_object_id_value が手順 1 からの場合。

3. インスタンス内にあるバッファ数を検索します。

   ```
   SELECT NAME, BLOCK_SIZE, SUM(BUFFERS)
     FROM V$BUFFER_POOL
    GROUP BY NAME, BLOCK_SIZE
    HAVING SUM(BUFFERS) > 0;
   ```

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。

   ```
   % cache used by segment_name = [buffers(Step2)/total buffers(Step3)]
   ```

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。

KEEP プール

アプリケーションに頻繁に参照されるセグメントがある場合は、KEEP バッファ・プールと呼ばれる個別のキャッシュにそれらのセグメントのブロックをキャッシュします。メモリーは、DB_KEEP_CACHE_SIZE パラメータを必要なサイズに設定することで KEEP バッファ・プールに割り当てられます。KEEP プールのメモリーは、デフォルト・プールのサブセットではありません。保持できる一般的なセグメントは、頻繁に使用される小さい参照表です。アプリケーション開発者と DBA は、どの表が候補かを判断できます。

14-15 ページの「プール内に多くのバッファを持つセグメントの判断」で説明するように、V$BH を問い合せて、候補表からブロック数をチェックできます。

注意： NOCACHE 句は、KEEP キャッシュ内の表に影響を与えません。

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で使用されるブロックを加算することで計算できます。セグメントに関する情報を収集する場合、DBA_TABLES.BLOCKS と DBA_TABLES.EMPTY_BLOCKS を問い合せて使用されるブロック数を判断できます。

ヒット率を計算するには、前述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、これらの結果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限りません。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・プールから除去されたブロックは、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 プール

メモリーに残さないセグメントに属するブロック用の RECYCLE バッファ・プールを構成できます。RECYCLE プールは、ほとんどスキャンされないか頻繁に参照されないセグメントに適しています。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利用できる可能性は（使用可能物理メモリーの量の制約により）ほとんどありません。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。したがって、そのオブジェクトのブロックはキャッシュしないでください。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。

メモリーは、DB_RECYCLE_CACHE_SIZE パラメータを必要なサイズに設定することで RECYCLE バッファ・プールに割り当てられます。この RECYCLE バッファ・プールのメモリーは、デフォルト・プールのサブセットではありません。

あまり早くメモリーからブロックを破棄しないでください。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。SELECT 文の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む必要があります。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。

共有プールとラージ・プールの構成および使用方法

異なるタイプのデータをキャッシュするには、共有プールを使用します。キャッシュされたデータには、PL/SQL ブロックおよび SQL 文のテキストおよび実行可能フォーム、ディクショナリ・キャッシュ・データおよびその他のデータが含まれています。

共有プールを適切な大きさにして使用すると、次の 4 つの方法でリソース使用量を低減できます。

1. SQL 文がすでに共有プールに存在する場合は解析オーバーヘッドをなくせます。このため、ホスト上の CPU リソースとエンド・ユーザーの経過時間が節約されます。
2. リソース使用のラッチングが大幅に減少して、拡張性がさらに増大します。
3. すべてのアプリケーションが SQL 文およびディクショナリ・リソースの同一プールを使用するので、共有プール・メモリーの必要量が低減されます。
4. 共有プールのディクショナリ要素はディスク・アクセスが不要なので、I/O リソースが節約されます。

この項では、次の項目について説明します。

- 共有プールの概念
- 共有プールの効果的な使用方法
- 共有プールのサイズ設定

共有プールの概念

共有プールの主なコンポーネントは、ライブラリ・キャッシュとディクショナリ・キャッシュです。ライブラリ・キャッシュは、最近参照された SQL と PL/SQL コードの実行可能な（解析またはコンパイルされた）形式を格納します。ディクショナリ・キャッシュは、データ・ディクショナリから参照されたデータを格納します。ライブラリ・キャッシュやディクショナリ・キャッシュなどの共有プール内のキャッシュの多くは、必要に応じてサイズを自動的に増減します。共有プールに空き領域がない場合は、新しいエントリを受け入れるために古いエントリがこれらのキャッシュから除去されます。

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、頻繁に使用されるデータがキャッシュされるように共有プールをサイズ設定する必要があります。

共有サーバー、パラレル問合せ、Recovery Manager など、共有プールで大きいメモリーの割当てを行う機能は多数あります。ラージ・プールと呼ばれる個別のメモリー領域を構成して、これらの機能で使用される SGA メモリーを区別することをお薦めします。

> **関連項目：** ラージ・プールの構成の詳細は、14-33 ページの「ラージ・プールの使用」を参照してください。

共有プールからのメモリーの割当ては、チャンクで行われます。このため、1 つの連続領域を必要とせずにラージ・オブジェクト（5k より多い）をキャッシュにロードできるので、フラグメント化のために十分な連続メモリーが不足する可能性が減ります。

Java、PL/SQL または SQL の各カーソルが、まれに、5k より大きい共有プールから割当てを行う場合があります。このような割当てを最も効率よく行うために、Oracle では、少量の共有プールを区別しています。このメモリーは、共有プールに十分な領域がない場合に使用します。共有プールの区別された領域は予約プールと呼ばれます。

> **関連項目：** 共有プールの予約領域の詳細は、14-39 ページの「予約プールの構成」を参照してください。

ディクショナリ・キャッシュの概念

データ・ディクショナリ・キャッシュに格納されている情報には、ユーザー名、セグメント情報、プロファイル・データ、表領域情報および順序番号が含まれています。また、ディクショナリ・キャッシュはスキーマ・オブジェクトを説明する情報すなわちメタデータも格納します。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。

ライブラリ・キャッシュの概念

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。この項では、チューニングを中心に説明します。チューニングはカーソル、PL/SQL プログラムおよび Java クラスに関連するためです。これらをまとめてアプリケーション・コードと呼びます。

アプリケーション・コードを実行するとき、既存のコードが以前に実行されており、共有できる場合は、そのコードを再利用しようとします。解析された文の表現がライブラリ・キャッシュ内に存在し、共有できる場合は、既存のコードを再利用します。これは、ソフト解析またはライブラリ・キャッシュ・ヒットと呼ばれています。

既存のコードを使用できない場合は、アプリケーション・コードの新しい実行可能バージョンを作成する必要があります。これは、ハード解析またはライブラリ・キャッシュ・ミスと呼ばれています。

> **関連項目：** SQL 文と PL/SQL 文を共有できる場合の詳細は、14-21 ページの「SQL 共有基準」を参照してください。

ライブラリ・キャッシュ・ミスは、SQL 文を処理するとき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。

アプリケーションが SQL 文の解析コールを行うとき、解析された文の表現がライブラリ・キャッシュにまだ存在しない場合、Oracle はその文を解析し、共有プールに解析されたフォームを格納します。これはハード解析です。可能な場合は、すべての共有可能な SQL 文が共有プール内にあることを確認して、解析コールのライブラリ・キャッシュ・ミスを低減できます。

アプリケーションが SQL 文に対して実行コールを作成し、すでに作成された SQL 文の実行可能な部分が別の文のための場所を作成するためにライブラリ・キャッシュから除去（すなわち、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 領域を作成し、実行します。この場合も、ハード解析が発生します。通常は、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実行コールのライブラリ・キャッシュ・ミスを低減できます。

ハード解析を実行するには、ソフト解析の実行時より多くのリソースを使用します。ソフト解析に使用するリソースには、CPU およびライブラリ・キャッシュ・ラッチ取得が含まれます。ハード解析に必要なリソースには、追加の CPU、ライブラリ・キャッシュ・ラッチ取得および共有プール・ラッチ取得が含まれます。

SQL 共有基準

Oracle は、発行される SQL 文または PL/SQL ブロックが共有プールに現在存在する別の文と同じかどうかを自動的に判断します。

比較のために、次のステップが実行されます。

1. 発行された文のテキストは、共有プール内の既存の文と比較されます。
2. 文のテキストがハッシュされます。一致するハッシュ値がない場合、SQL 文は共有プール内に現在存在せず、ハード解析が実行されます。
3. 共有プール内の既存の SQL 文に一致するハッシュ値がある場合は、一致した文のテキストが、ハッシュされた文のテキストと比較され、それらが同一であるかどうかが確認されます。SQL 文や PL/SQL ブロックのテキストは、空白、大文字小文字の区別、コメントも含め、完全に同一である必要があります。たとえば、次の文は同じ共有 SQL 領域を使用できません。

   ```
   SELECT * FROM employees;
   SELECT * FROM Employees;
   SELECT * FROM employees;
   ```

   通常は、リテラルのみ異なる SQL 文は同じ共有 SQL 領域を使用できません。たとえば、次の SQL 文は同じ SQL 領域に変換されません。

   ```
   SELECT count(1) FROM employees WHERE manager_id = 121;
   SELECT count(1) FROM employees WHERE manager_id = 247;
   ```

   唯一の例外は、CURSOR_SHARING パラメータが SIMILAR または FORCE に設定されている場合です。CURSOR_SHARING パラメータが、SIMILAR または FORCE に設定されている場合、類似の文は SQL 領域を共有できます。CURSOR_SHARING を使用する場合のコストと効果については、この項の後半で説明します。

   > **関連項目：** CURSOR_SHARING パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

4. 発行された文で参照されたオブジェクトは、共有プール内のすべての既存の文の参照済みオブジェクトと比較され、それらのオブジェクトが同一であるかどうかが確認されます。

   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際には、同じスキーマ内の同じオブジェクトである必要があります。

   たとえば、2 人のユーザーが次の SQL 文を発行するとします。

   ```
   SELECT * FROM employees;
   ```

   各ユーザーに独自の employees 表がある場合、文はユーザーごとに異なる表を参照するので、この文は同一とみなされません。

5. SQL 文の中のバインド変数は、名前、データ型および長さで一致している必要があります。

   たとえば、次の文で同じ共有 SQL 領域を使用できないのは、バインド変数名が異なるためです。

   ```
   SELECT * FROM employees WHERE department_id = :department_id;
   SELECT * FROM employees WHERE department_id = :dept_id;
   ```

   多数の Oracle 製品（Oracle Forms やプリコンパイラなど）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。

6. セッションの環境は同一である必要があります。比較される項目は次のとおりです。

   - 最適化のアプローチと目標。同じ最適化アプローチを使用して、SQL 文を最適化する必要があります。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する必要があります。
   - SORT_AREA_SIZE などのセッション構成可能パラメータ。

共有プールの効果的な使用方法

共有プールの重要な目的は、SQL 文と PL/SQL 文の実行可能バージョンをキャッシュすることです。これにより、ハード解析にリソースを使用することなく、同じ SQL または PL/SQL コードを複数回実行できるので、CPU、メモリーおよびラッチの使用が大幅に減少します。

また、共有プールは、データ・ウェアハウス・アプリケーションで非共有 SQL をサポートできます。これらのアプリケーションでは、並行性が低くリソース使用率の高い SQL 文が実行されます。このような状況では、リテラル値を持つ非共有 SQL を使用することをお薦めします。バインド変数ではなくリテラル値を使用すると、オプティマイザは優れた列選択性の見積りを行えるので、最適なデータ・アクセス・プランを提供します。

関連項目：『Oracle9*i* データ・ウェアハウス・ガイド』

OLTP システムでは、共有プールと関連リソースを効率的に使用できるようにする方法が多数あります。次の項目についてアプリケーション開発者と検討し、共有プールが効果的に使用されるようにする方法を決定します。

- 共有カーソル
- 接続の維持
- 単一のユーザーのログインおよび修飾表の参照
- PL/SQL の使用方法
- DDL の実行の回避

- キャッシュ順序番号
- カーソルのアクセスおよび管理

並行性の高い OLTP システムで共有プールを効率よく使用すると、解析関連アプリケーションの拡張性の問題が発生する確率が大幅に低減します。

共有カーソル

同じアプリケーションを実行する複数のユーザーのために共有 SQL を再利用すると、ハード解析が回避されます。ソフト解析は、共有プール・ラッチやライブラリ・キャッシュ・ラッチなどのリソース使用量を大幅に減少させます。カーソルを共有するには、次のことを行います。

- 可能な場合、SQL 文ではリテラルではなくバインド変数を使用します。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 employee_id FROM employees WHERE department_id = 10;
SELECT employee_id FROM employees WHERE department_id = 20;
```

  リテラルをバインド変数と置換すると、2 回実行可能な SQL 文が 1 つのみ生成されます。

```
SELECT employee_id FROM employees WHERE department_id = :dept_id;
```

  > **注意：** バインド変数を使用するためにコードをリライトすることが実際的ではない既存のアプリケーションについては、CURSOR_SHARING 初期化パラメータを使用してハード解析のオーバーヘッドをある程度回避できます。詳細は、14-42 ページの「既存のアプリケーション用の CURSOR_SHARING」の項を参照してください。

- 多数のユーザーが動的な非共有の SQL 文を発行するようなアプリケーションを設計しないようにしてください。通常、大半のユーザーが必要とするデータの大部分は、事前に設定されている問合せを使用して満たすことができます。そのような機能が必要な場合は、動的 SQL を使用します。
- 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。
- アプリケーション開発者に対し、次のポリシーを設定します。
  - SQL 文と PL/SQL ブロックに対して、バインド変数のネーミング規則と、スペーシング規定を標準化します。
  - 可能な場合、ストアド・プロシージャを使用することを考慮してください。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フォームに格納されているため、ストアド・プロシージャを使用すると実行時間の解析が減少します。

接続の維持

中間層を持つ大きな OLTP アプリケーションでは、データベース要求ごとに接続と切断を行うのではなく、接続を維持します。接続を維持することで、ラッチなどの CPU リソースとデータベース・リソースが節約されます。

単一のユーザーのログインおよび修飾表の参照

ユーザーが独自のユーザー ID でデータベースにログインするような大きい OLTP システムでは、パブリック・シノニムを使用するのではなく、明示的にセグメントの所有者を修飾すると有益です。これにより、ディクショナリ・キャッシュ内のエントリ数が大幅に削減されます。たとえば、次のような利点があります。

```
SELECT employee_id FROM hr.employees WHERE department_id = :dept_id;
```

表名を認定する別の方法として、個々のユーザー ID ではなく単一のユーザー ID でデータベースに接続します。ユーザー・レベルの検証は、中間層でローカルに行われます。個別のユーザー ID の数を削減した場合も、ディクショナリ・キャッシュ上の負荷は低減します。

PL/SQL の使用方法

ストアド PL/SQL パッケージを使用すると、多数のユーザーが個々にユーザー・サインオンとパブリック・シノニムを持つシステムにおける、拡張性の問題を克服できます。これは、パッケージがコール元ではなく所有者として実行されるため、ディクショナリ・キャッシュの負荷がかなり削減されるためです。

注意： 拡張性の問題を克服するには、定義者権限パッケージの使用をお薦めします。ディクショナリ・キャッシュの負荷軽減の利点は、実行者権限パッケージの場合ほど顕著ではありません。

DDL の実行の回避

ピーク時間に使用率の高いセグメントで DDL 操作を実行しないようにします。そのようなセグメントで DDL を実行すると、多くの場合、依存 SQL は無効にされるため、以降の実行で再度解析されることになります。

キャッシュ順序番号

頻繁に更新される順序番号に十分なキャッシュ領域を割り当てると、ディクショナリ・キャッシュ・ロックの回数が大幅に減るため、拡張性が向上します。CREATE SEQUENCE 文または ALTER SEQUENCE 文の CACHE キーワードを使用して、各順序のキャッシュ済みエントリ数を構成できます。

関連項目： CREATE SEQUENCE 文および ALTER SEQUENCE 文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

カーソルのアクセスおよび管理

使用している Oracle アプリケーション・ツールにより異なりますが、アプリケーションが解析コールを実行する頻度を制御できます。

アプリケーションが、カーソルをクローズする、または新しい SQL 文に既存のカーソルを再利用する頻度は、セッションで使用されるメモリー量と、時には、そのセッションで実行される解析の量にも影響を与えます。

（異なる SQL 文の）カーソルをクローズまたは再利用するアプリケーションは、カーソルをオープンした状態を保つアプリケーションほどセッション・メモリーを必要としません。逆に、そのようなアプリケーションでは、解析コールをより多く実行し、そのための追加の CPU および Oracle リソースを使用する可能性があります。

頻繁に実行されない SQL 文に関連するカーソルをクローズしたり、他の文に再利用できるのは、その文を再実行（および再解析）する可能性が低いからです。

再実行される SQL 文を含むカーソルがクローズまたは別の文に再利用される場合は、追加の解析コールが必要になります。カーソルがオープンされた状態であれば、解析コールを発行するためのオーバーヘッドを発生させずに、カーソルを再利用できます。

カーソルの管理を行う方法は、アプリケーション開発ツールにより異なります。次の項では、いくつかのツールで使用される方法を紹介します。

関連項目：

- 各ツールの詳細は、ツール固有のマニュアルを参照してください。
- カーソルの共有と管理の詳細は、『Oracle9*i* データベース概要』を参照してください。

OCI による解析コールの低減 Oracle Call Interface（OCI）を使用する場合、再実行するカーソルはクローズおよび再オープンしないでください。そのかわりに、カーソルをオープンしたままにし、実行前にリテラル値をバインド変数に変更してください。

既存の SQL 文が今後再実行される場合は、新しい SQL 文に文ハンドルを再利用しないようにしてください。

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用する場合、プリコンパイラ句を設定して、いつカーソルをクローズするかを制御できます。Oracle モードでは、プリコンパイラ句は次のとおりです。

- `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。

プリコンパイラ句は、プリコンパイラ・コマンドライン上またはプリコンパイラ・プログラム内で指定できます。これらの句により、様々な方法で、プログラムの実行中にカーソルを管理できます。

> **関連項目：** これらの句の詳細は、使用している言語のプリコンパイラ・マニュアルを参照してください。

SQLJ による解析コールの低減 文を用意し、バインド変数に新しい値を使用して文を再実行します。カーソルは、セッション中はオープンのままです。

JDBC による解析コールの低減 新しいリテラル値は、再実行のためにカーソルにバインドできるので、カーソルを再実行する場合は、カーソルをクローズしないでください。別の方法として、JDBC は `setStmtCacheSize()` メソッドを使用して JDBC クライアント内に SQL 文キャッシュを提供しています。このメソッドを使用して、JDBC は JDBC プログラムに対してローカルな SQL 文キャッシュを作成します。

> **関連項目：** JDBC SQL 文キャッシュの使用方法の詳細は、『Oracle9*i* JDBC 開発者ガイドおよびリファレンス』を参照してください。

Oracle Forms による解析コールの低減 Oracle Forms で、カーソル管理の一部を管理できます。トリガー・レベル、フォーム・レベルまたは実行時のいずれかで、この管理を実施できます。

共有プールのサイズ設定

新規にインスタンスを構成する場合、作成する共有プール・キャッシュの適切なサイズを知ることはできません。通常、DBA はキャッシュ・サイズの最初の見積りを行い、次にインスタンス上で代表的なワークロードを実行し、関連する統計を調べて、キャッシュが過小構成か過大構成かを調べます。

OLTP アプリケーションの多くでは、共有プール・サイズはアプリケーション・パフォーマンスにとって重要な要因です。意思決定支援システム（DSS）などのごく少数の別々な SQL 文のみ発行するアプリケーションの場合は、共有プールのサイズはそれほど重要ではありません。

共有プールが小さすぎる場合、追加リソースを使用して、使用可能領域の限度を補います。このため、CPU とラッチングのリソースが使用され、競合が発生します。

共有プールは、頻繁にアクセスされるオブジェクトをキャッシュするためにちょうど十分な大きさであることが最適です。共有プールに大量の空きメモリーを持つことは、メモリーの無駄になります。

共有プール : ライブラリ・キャッシュの統計

共有プールをサイズ設定するときの目標は、メモリーを割り当てすぎずに、複数回実行される SQL 文がライブラリ・キャッシュにキャッシュされるようにすることです。

以前にキャッシュされ、すでに除去された SQL 文の再ロード（すなわち、再解析）の量の統計は、V$LIBRARYCACHE ビューの RELOADS 列に示されます。効果的に SQL を再利用するアプリケーションでは、システムが最適な共有プール・サイズを持ち、RELOADS 統計が 0（ゼロ）に近い値を示します。

V$LIBRARYCACHE ビューの INVALIDATIONS 列は、ライブラリ・キャッシュのデータが無効にされ、再解析された回数を示しています。INVALIDATIONS は 0（ゼロ）に近い値である必要があります。つまり、共有できた可能性のある SQL 文が、ある操作（たとえば、DDL）により無効にされたことを意味します。この統計は、ピーク・ロード中の OLTP システム上では、0（ゼロ）に近い値となります。

別の重要な統計は、ピーク時の共有プール内の空きメモリー量です。空きメモリー量は、共有プールの空きメモリーを参照する V$SGASTAT から問い合せることができます。空きメモリーは、システム上に再ロードを発生させない程度で、できるだけ小さい値である必要があります。

最終的には、ライブラリ・キャッシュの全般的なインジケータは、ライブラリ・キャッシュ・ヒット率で表されます。この値は、この項で説明されているその他の統計、およびハード解析率や、共有プールまたはライブラリ・キャッシュのラッチ競合があるかどうかなどのその他のデータとともに考慮する必要があります

これらの統計の詳細は、次の項で説明します。

V$LIBRARYCACHE

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、最新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。

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。

関連項目： V$LIBRARYCACHE ビューの列については、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

各 NAMESPACE を個々に調べるには、次の問合せを使用します。

```
SELECT namespace
     , pins
     , pinhits
     , reloads
     , invalidations
  FROM V$LIBRARYCACHE
 ORDER BY namespace;
```

この問合せの出力例を次に示します。

```
NAMESPACE             PINS    PINHITS    RELOADS INVALIDATIONS
--------------- ---------- ---------- ---------- -------------
BODY                  8870       8819          0             0
CLUSTER                393        380          0             0
INDEX                   29          0          0             0
OBJECT                   0          0          0             0
PIPE                 55265      55263          0             0
SQL AREA          21536413   21520516      11204             2
TABLE/PROCEDURE   10775684   10774401          0             0
TRIGGER               1852       1844          0             0
```

ライブラリ・キャッシュ・ヒット率を計算するには、次の計算式を使用します。

```
Library Cache Hit Ratio = sum(pinhits) / sum(pins)
```

ライブラリ・キャッシュ・ヒット率の計算式を使用すると、キャッシュ・ヒット率は次のようになります。

```
SUM(PINHITS)/SUM(PINS)
----------------------
            .999466248
```

注意： これらの問合せでは、ある時間間隔で収集された統計ではなく、インスタンス起動時からのデータを戻します。時間間隔の統計の方が、より的確に問題を特定できます。

関連項目： ある時間間隔の情報を収集する方法の詳細は、第 20 章「データベース統計収集用の Oracle のツール製品」を参照してください。

戻されたデータを調べると、次のことがわかります。

- SQL AREA の NAMESPACE では、21,536,413 回の実行がありました。
- 11,204 回の実行でライブラリ・キャッシュ・ミスが発生したため、文またはブロックを暗黙的に再解析するか、またはライブラリ・キャッシュから除去されている場合は、オブジェクト定義を再ロード（すなわち、RELOAD）する必要があります。
- SQL 文は 2 回無効化されたため、再度ライブラリ・キャッシュ・ミスが発生しました。
- ヒット率は約 99.94% です。これは、実行の 0.06% のみが再解析されたことを意味します。

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'
   AND POOL = 'shared pool';

The output will be similar to the following:

POOL        NAME                            BYTES
----------- -------------------------- ----------
shared pool free memory                   4928280
```

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。また、共有プールがいっぱいというのみでは、問題があるとはいえません。これは、適切に構成されたシステムであることを示している場合があります。

共有プールのアドバイザ統計

ライブラリ・キャッシュに使用可能なメモリー量は、Oracle インスタンスの解析率に大きな影響を与えます。Oracle9*i* のリリース 2（9.2）以降では、共有プールのアドバイザ統計により、データベース管理者に、ライブラリ・キャッシュ・メモリーについての情報と、共有プールのサイズ変更が解析率に及ぼす影響の予測が提供されます。

共有プールのアドバイザ統計では、共有プール・メモリーにおけるライブラリ・キャッシュの使用率が追跡され、異なるサイズの共有プールでライブラリ・キャッシュがどのように動作するかが予測されます。2 つの固定ビューにより、ライブラリ・キャッシュのメモリー使用量、現在の確保量、共有プールの LRU リスト上にある量、さらに共有プールのサイズ変更により損失または獲得できる時間を判別する情報が提供されます。

共有プールのアドバイザ統計では、次のビューが使用できます。共有プール・アドバイザをオンにすると、これらのビューにはあらゆるデータが表示されます。共有プール・アドバイザをオフにすると、それらの統計がリセットされます。

V$SHARED_POOL_ADVICE このビューには、共有プールのサイズを変更した場合の、見積り解析時間短縮率に関する情報が表示されます。サイズの範囲は、同じ時間間隔で現在の共有プール・サイズの 50 ～ 200% です。時間間隔の値は、現在の共有プール・サイズによって異なります。

解析時間短縮率とは、ライブラリ・キャッシュのメモリー・オブジェクトを再ロードするかわりに、共有プールに保持することによって節約できる時間の長さを指します。

V$LIBRARY_CACHE_MEMORY このビューには、別の NAMESPACE のライブラリ・キャッシュのメモリー・オブジェクトに割り当てられるメモリーに関する情報が表示されます。メモリー・オブジェクトとは、効率的な管理を行うためのメモリーの内部グループ化です。ライブラリ・キャッシュ・オブジェクトは 1 つ以上のメモリー・オブジェクトで構成されます。

関連項目：

- 24-45 ページ「V$SHARED_POOL_ADVICE」
- 24-16 ページ「V$LIBRARY_CACHE_MEMORY」

共有プール : ディクショナリ・キャッシュの統計

共有プールがライブラリ・キャッシュに対して適切にサイズ設定されている場合、その設定はディクショナリ・キャッシュ・データに対しても適切であるのが普通です。

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。インスタンス起動時は、データ・ディクショナリ・キャッシュにデータは含まれていません。したがって、発行された SQL 文からキャッシュ・ミスが発生する可能性があります。キャッシュに読み込まれるデータが増えると、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します。この時点で、キャッシュ・ミスはほとんど発生しません。

V$ROWCACHE ビューの各行は、データ・ディクショナリ項目について単一のタイプの統計を収録します。これらの統計は、直前のインスタンス起動以降のデータ・ディクショナリ・アクティビティを反映しています。データ・ディクショナリ・キャッシュの使用と有効性を反映する V$ROWCACHE ビューの中の列を表 14-2 にリストします。

表 14-2 V$ROWCACHE 列

列	説明
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。

表 14-2 V$ROWCACHE 列（続き）

列	説明
GETMISSES	キャッシュで満たされなかったデータ要求で、I/O を必要とするものの個数を示します。
MODIFICATIONS	ディクショナリ・キャッシュ内のデータが更新された回数を示します。

次の問合せによって、アプリケーションの実行中、ある期間にわたって V$ROWCACHE ビューの統計を監視してください。導出された列 PCT_SUCC_GETS は、項目固有のヒット率と考えることができます。

```
column parameter format a21
column pct_succ_gets format 999.9
column updates format 999,999,999

SELECT parameter
     , sum(gets)
     , sum(getmisses)
     , 100*sum(gets - getmisses) / sum(gets)  pct_succ_gets
     , sum(modifications)                     updates
  FROM V$ROWCACHE
 WHERE gets > 0
 GROUP BY parameter;
```

この問合せの出力例を次に示します。

```
PARAMETER              SUM(GETS) SUM(GETMISSES) PCT_SUCC_GETS      UPDATES
---------------------- ---------- -------------- ------------- ------------
dc_database_links              81              1          98.8            0
dc_free_extents             44876          20301          54.8       40,453
dc_global_oids                 42              9          78.6            0
dc_histogram_defs            9419            651          93.1            0
dc_object_ids               29854            239          99.2           52
dc_objects                  33600            590          98.2           53
dc_profiles                 19001              1         100.0            0
dc_rollback_segments        47244             16         100.0           19
dc_segments                100467          19042          81.0       40,272
dc_sequence_grants            119             16          86.6            0
dc_sequences                26973             16          99.9       26,811
dc_synonyms                  6617            168          97.5            0
dc_tablespace_quotas          120              7          94.2           51
dc_tablespaces             581248             10         100.0            0
dc_used_extents             51418          20249          60.6       42,811
```

```
dc_user_grants            76082            18        100.0            0
dc_usernames             216860            12        100.0            0
dc_users                 376895            22        100.0            0
```

サンプル問合せが戻したデータを調べると、次のことがわかります。

- 使用済みエクステント、空きエクステントおよびセグメントには、多数のミスと更新があります。つまり、インスタンスに大量の動的な領域の拡張があったことを示しています。
- 取得成功率およびその統計と、実際の取得数との比較に基づくと、共有プールはディクショナリ・キャッシュ・データを適切に格納できる大きさがあります。

次の計算式を使用して、総合的ディクショナリ・キャッシュ・ヒット率も計算できますが、すべてのキャッシュにわたるデータを合計すると、より細かいデータの粒度は失われます。

```
SELECT (SUM(GETS - GETMISSES - FIXED)) / SUM(GETS) "ROW CACHE" FROM V$ROWCACHE;
```

共有プール統計の解釈

共有プール統計は実行可能な調整方法を示します。この項ではそのうちのいくつかについて説明します。

メモリー割当ての増加

共有プールのメモリー量を増やすと、ライブラリ・キャッシュとディクショナリ・キャッシュの両方で使用できるメモリー量が増えます。

ライブラリ・キャッシュへの追加のメモリー割当て 共有 SQL 領域がそれらの SQL 文を解析した後にキャッシュ内に残るようにするには、V$LIBRARYCACHE.RELOADS 値が 0（ゼロ）に近くなるまでライブラリ・キャッシュに利用できるメモリー量を増やします。ライブラリ・キャッシュに利用できるメモリーを増やすには、SHARED_POOL_SIZE 初期化パラメータ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 ブロックの暗黙的な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こともあります。その場合は、初期化パラメータ OPEN_CURSORS の値を増やします。

データ・ディクショナリ・キャッシュへの追加のメモリーの割当て GETS と GETMISSES 列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 の割合は、アプリケーションによって異なりますが、10% あるいは 15% より低くしてください。

次のすべてに当てはまる場合は、キャッシュに利用できるメモリー量を増やすことを考慮してください。

- アプリケーションは共有プールを効果的に使用している。14-22 ページの「共有プールの効果的な使用方法」を参照してください。
- システムは安定状態に達し、項目固有のヒット率が低く、ヒット率の低いキャッシュへの取得数が多い。

初期化パラメータ SHARED_POOL_SIZE の値を増やして、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を増やします。

メモリー割当ての減少

RELOADS が 0 に近く、共有プール内の空きメモリーが少ない場合、共有プールは、最も頻繁にアクセスされるデータを保持できる十分な大きさがあります。

常に共有プールに多数の空きメモリーがある場合、このメモリーを他の場所に割り当てるために共有プールのサイズを小さくしても、良好なパフォーマンスを維持できます。

共有プールを小さくするには、SHARED_POOL_SIZE パラメータの値を変更してキャッシュのサイズを小さくします。

ラージ・プールの使用

共有プールとは異なり、ラージ・プールには LRU リストがありません。Oracle は、ラージ・プールからオブジェクトを除去しようとしません。

インスタンスが次のいずれかを使用する場合は、ラージ・プールの構成を考慮してください。

- パラレル問合せ

  パラレル問合せでは、共有プール・メモリーを使用してパラレル実行メッセージ・バッファをキャッシュします。

  関連項目： パラレル問合せによるラージ・プールのサイズ設定の詳細は、『Oracle9*i* データ・ウェアハウス・ガイド』を参照してください。

- Recovery Manager

  Recovery Manager は、共有プールを使用してバックアップおよびリストア操作時に I/O バッファをキャッシュします。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 KB 単位でバッファを割り当てます。

  関連項目： Recovery Manager を使用するときのラージ・プールのサイズ設定の詳細は、『Oracle9*i* Recovery Manager ユーザーズ・ガイド』を参照してください。

- 共有サーバー

  共有サーバー・アーキテクチャでは、各クライアント・プロセスのセッション・メモリーが共有プールに含まれています。

共有サーバー・アーキテクチャでのラージ・プールと共有プールのチューニング

Oracle では共有サーバー・セッション・メモリーに共有プールからメモリーを割り当てるため、ライブラリ・キャッシュとディクショナリ・キャッシュに使用可能な共有プール・メモリーの量が減少します。別のプールからこのセッション・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用できるので、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。

共有サーバー関連のユーザー・グローバル領域（UGA）の割当てには、共有プールではなくラージ・プールの使用をお薦めします。共有プールは、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システム・グローバル領域（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ラージ・プールを使用すると、共有プールの断片化も減少します。

ラージ・プールに共有サーバー関連の UGA を格納するには、初期化パラメータ LARGE_POOL_SIZE に値を指定します。どのプール（共有プールまたはラージ・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 をチェックします。ラージ・プールはデフォルトで構成されません。最小値は 300K です。ラージ・プールを構成しないと、共有サーバー・ユーザー・セッション・メモリーに共有プールが使用されます。

ラージ・プールの大きさは、同時にアクティブとなるセッションの数を基準に構成します。各アプリケーションは、必要なセッション情報メモリー量がそれぞれ異なり、ラージ・プールあるいは SGA の構成はメモリー要件を反映する必要があります。たとえば、アクティブな各セッションのセッション情報を格納するために共有サーバーが 200 ～ 300K を必要とすると仮定します。100 個のセッションが同時にアクティブになると予想される場合、30M のラージ・プールを構成するか、ラージ・プールを構成しない場合は、共有プールを増やしてください。

注意： 共有サーバー・アーキテクチャを使用する場合、ラージ・プールを構成した場合でも、Oracle によって各構成セッションに一定量のメモリー（約 10K）が共有プールから割り当てられます。CIRCUITS 初期化パラメータは、データベースで許可される同時共有サーバー接続最大数を指定します。

関連項目：

- ラージ・プールの詳細は、『Oracle9*i* データベース概要』を参照してください。
- 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

共有サーバーの UGA 記憶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。ラージ・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乗算します。

共有サーバー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、専用サーバー環境と比較した場合に共有サーバーでは PGA メモリーの使用量が減るためです。

注意： 共有サーバーを使用したソートのパフォーマンスを最高にするには、SORT_AREA_SIZE と SORT_AREA_RETAINED_SIZE を同じ値に設定します。これによって、ソート結果をディスクに書き込むのではなくラージ・プールに留めておきます。

V$SESSTAT ビューでのシステム統計のチェック Oracle はセッションによって使用された全体のメモリーの統計を収集し、動的パフォーマンス・ビュー V$SESSTAT に格納します。表 14-3 はこれらの統計をリストしたものです。

表 14-3 メモリーを反映した V$SESSTAT 統計

統計	説明
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、V$STATNAME を問い合せます。共有サーバーを使用している場合、次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・ビュー 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。次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
```

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。

共有サーバー・アーキテクチャ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かぎり、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。

PRIVATE_SGA の設定による各ユーザー・セッションのメモリー使用量の制限 PRIVATE_SGA リソース制限を設定して、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：** PRIVATE_SGA リソース・リミットの設定の詳細は、『Oracle9*i* SQL リファレンス』の「ALTER RESOURCE COST 文」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 層の接続を実装することでメモリー使用を低減できます。これはトランザクション処理（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。共有サーバー環境には次の利点があります。

- TP モニターに比べてアプリケーション設計の制限が大幅に少なくなります。
- 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・プロセス数とコンテキストの切替えが大幅に減ります。
- 共有サーバー・モードでさらに多くの SGA が使用される場合でも総メモリー使用量が大幅に減ります。

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 true に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・キャッシュからカーソル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。

- CURSOR_SPACE_FOR_TIME 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・キャッシュからカーソルの割当てを解除できます。この場合、Oracle では、SQL 文を含むカーソルがライブラリ・キャッシュ内にあることを検証する必要があります。
- CURSOR_SPACE_FOR_TIME を true に設定すると、その文に関連するすべてのアプリケーション・カーソルがクローズされる場合のみカーソルの割当てを解除できます。この場合、カーソル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カーソルの割当てを解除できないため、カーソルがキャッシュ内にあるかどうかを確認する必要はありません。

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。この値は、対応付けられているアプリケーション・カーソルがクローズされるまでカーソルの割当て解除も防ぎます。

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 が既存のカーソルの割当てを解除します。カーソル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（カーソルが再度実行される場合のみ）が、SQL 文

が解析できないため、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。同時にオープンしているすべてのカーソルのプライベート SQL 領域が使用可能メモリーを満たしているために、新しい SQL 文の領域がない場合は、文を解析できません。Oracle は、メモリーが十分にないことを示すエラーを戻します。

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 が使用されているアプリケーションで特に有用です。

Oracle では、ライブラリ・キャッシュをチェックして、指定の文で 3 回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することを想定し、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 のアルゴリズムでは、必要に応じてセッション・カーソル・キャッシュ内の項目を除去し、新しい項目のための空間を作成します。

また次の文を使用すると、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。

```
ALTER SESSION SET SESSION_CACHED_CURSORS = value;
```

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$SYSSTAT 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。

予約プールの構成

非常に大きいメモリーの要求は小さいチャンクに分割されますが、システムによっては、メモリーの連続チャンク（たとえば、5KB 以上）を検索する必要性が依然存在する場合があります（デフォルトの最小予約プールの割当ては 4400 バイトです）。

共有プールに十分な空き領域がない場合は、この要求を満たすための十分な空きメモリーを検索する必要があります。この操作では、検出可能期間にラッチ・リソースを保持するため、メモリー割当てで他の同時動作に対して多少の影響が生じる可能性があります。

したがって、共有プールに十分な領域がない場合、Oracle は使用できる共有プールに内部的に小さいメモリー領域を予約します。この予約プールによって、大きいチャンクの割当てがより効率的に行われます。

デフォルトでは、小さな予約プールを構成します。このメモリーは、PL/SQL およびトリガーのコンパイルなどの操作や、Java オブジェクトのロード時の一時領域に使用できます。予約プールから割り当てられたメモリーが解放されると、予約プールに戻ります。

予約されるデフォルトの領域量を変更する必要はほとんどありません。ただし、必要であれば、SHARED_POOL_RESERVED_SIZE 初期化パラメータを設定して予約プール・サイズを変更できます。このパラメータは、極端に大きい割当て用の領域を共有プール内に確保します。

大きい割当ての場合、Oracle は次の順序で共有プールへの領域の割当てを試行します。

1. 共有プールの予約されていない部分。
2. 予約プール。共有プールの予約されていない部分に十分な領域がない場合は、予約プールに十分な領域があるかどうかチェックされます。
3. メモリー。共有プールの予約されていない部分と予約された部分に十分な領域がない場合は、Oracle は割当てのために十分なメモリーの解放を試みます。次に、共有プールの予約されていない部分と予約されている部分が再試行されます。

SHARED_POOL_RESERVED_SIZE の使用 SHARED_POOL_RESERVED_SIZE のデフォルト値は SHARED_POOL_SIZE の 5% です。つまり、デフォルトでは、予約リストは構成されています。

SHARED_POOL_RESERVED_SIZE を SHARED_POOL_SIZE の半分を超える量に設定すると、Oracle はエラー信号を出します。予約プールにメモリーをあまり多くは予約できません。ただし、オペレーティング・システムのメモリー容量が共有プールのサイズを制約する場合があります。一般的には、SHARED_POOL_RESERVED_SIZE 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済ませている場合、ほとんどのシステムではこの値で十分です。この値を大きくすると、データベースは共有プールからメモリーを取り出します。（このため、それより小さい割当てに使用可能な予約されていない共有プールのメモリーの量が減少します。）

V$SHARED_POOL_RESERVED ビューの統計を使用すると、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。SGA のサイズを大きくするための空きメモリーが豊富にあるシステムでは、REQUEST_MISSES の値を 0（ゼロ）にすることが目標です。オペレーティング・システム・メモリーに制約があるシステムの場合は、REQUEST_FAILURES をなくすことが目標で、少なくともこの値が増加しないようにします。

これらの目標値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取られるため、SHARED_POOL_SIZE の値も同じだけ増やします。

> **関連項目：** LARGE_POOL_SIZE パラメータ設定の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

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。これらのパラメータで選択する設定は、システムの SGA サイズの制約によって異なります。

SHARED_POOL_RESERVED_SIZE の値を増やすと、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。

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ことがあります。次の場合です。

- REQUEST_MISSES がゼロの場合（または増加しない場合）
- FREE_MEMORY の最小値が SHARED_POOL_RESERVED_SIZE の 50% 以上になる場合

これらの条件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 固定ビューを使用すると、SHARED_POOL_SIZE の値が小さすぎる場合を示すこともできます。これは REQUEST_FAILURES がゼロより大きいかあるいは増加しているような場合です。

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。

除去防止のためのラージ・オブジェクトの保存

エントリが共有プールにロードされた後は、それを移動することはできません。エントリがロードされ、除去されると、空きメモリーが断片化されることもあります。

共有プールを管理するには、PL/SQL パッケージ DBMS_SHARED_POOL を使用します。共有 SQL 領域と共有 PL/SQL 領域は、古くなると 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ようにすることが可能です。

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。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、オブジェクトはメモリー内に維持されます。こうすることによって、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 パッケージは、次の場合に便利です。

- STANDARD や DIUTIL パッケージなどの大きな PL/SQL オブジェクトをロードする場合。大きな PL/SQL オブジェクトがロードされる場合で、領域を確保するために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場合は、ユーザーの応答時間に影響が現れます。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。
- 頻繁に実行されるトリガー。頻繁に使用する表のコンパイル済みトリガーを共有プール内に維持できます。
- DBMS_SHARED_POOL が順序もサポートする場合。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ぎます。

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、次の手順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 をコールしてオブジェクトを確保します。

   この手順により、保存されている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用し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。

関連項目： DBMS_SHARED_POOL プロシージャの使用方法の詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

既存のアプリケーション用の CURSOR_SHARING

解析の第 1 段階の 1 つは、文のテキストを共有プール内の既存の文と比較して、その文を共有できるかどうかを確認することです。文をテキストとして比較し、なんらかの点で異なる場合は、文は共有されません。

例外は、CURSOR_SHARING パラメータが SIMILAR または FORCE に設定されている場合です。このパラメータを使用する場合、まず共有プールがチェックされ、共有プールに同一の文があるかどうかが確認されます。同一の文が検出されないと、共有プール内で類似する文が検索されます。類似する文があると、解析チェックが引き続き行われ、カーソルの実行可能フォームを使用できるかどうかが検証されます。文がない場合は、文の実行可能フォームを生成するためにハード解析が必要になります。

類似 SQL 文

いくつかのリテラル値以外が同一である文は、類似文と呼ばれます。CURSOR_SHARING パラメータが SIMILAR または FORCE に設定されると、類似文は解析フェーズでテキスト・チェックを省略します。テキストの類似性では、共有は保証されません。SQL 文の新しいフォームでは、解析フェーズの残りのステップを実行して、既存の文の実行計画が新しい文にも同じように適用できるかどうかを確認する必要があります。

関連項目： 実行される各種チェックの詳細は、14-21 ページの「SQL 共有基準」を参照してください。

CURSOR_SHARING

CURSOR_SHARING を EXACT に設定すると、SQL 文はテキストがまったく同一の場合にのみ SQL 領域を共有することができます。これはデフォルトの動作です。この設定では、類似文は共有できません。テキストとしての完全に同一の文のみ共有できます。

CURSOR_SHARING を SIMILAR または FORCE に設定すると、類似文が SQL を共有できます。SIMILAR と FORCE の違いは、実行計画を機能低下させることなく SIMILAR が類似する文に SQL 領域を共有させるという点です。CURSOR_SHARING を FORCE に設定すると、類似文に実行可能な SQL 領域を共有するように強制します。この方法には、潜在的に実行計画の機能を低下させる可能性があります。したがって、計画が最適なものではなくなる可能性があってもカーソル共有率の向上を優先する場合に、FORCE を最後の手段として使用してください。

CURSOR_SHARING を使用する場合

CURSOR_SHARING 初期化パラメータにより一部のパフォーマンス問題を解決できる場合があります。初期化パラメータには次の値があります。FORCE、SIMILAR および EXACT（デフォルト）。このパラメータの使用は、多数の類似 SQL 文を持つ既存のアプリケーションにとっては有益です。

注意： 複雑な問合せを使用している場合は、DSS 環境で CURSOR_SHARING を FORCE に設定することはお薦めしません。また、CURSOR_SHARING が SIMILAR または FORCE に設定されると、スター型変換はサポートされません。詳細は、1-55 ページの「OPTIMIZER_FEATURES_ENABLE パラメータ」を参照してください。

最適な解決方法は、CURSOR_SHARING パラメータに依存するのではなく、共有可能な SQL を書くことです。これは、CURSOR_SHARING によってハード解析がなくなる分、使用されるリソース量は大幅に削減されますが、共有プール内で類似文を検索するために、ソフト解析の一部としてある程度の追加作業が必要になるからです。

注意： CURSOR_SHARING を SIMILAR または FORCE に設定すると、（SELECT 文に指定した）リテラルを含む選択された式の最大長（DESCRIBE からの戻り値）が増加します。ただし、戻されたデータの実際の長さは変わりません。

次の質問の回答がいずれも「はい」の場合は、CURSOR_SHARING を SIMILAR または FORCE に設定することを考慮してください。

1. 共有プール内にリテラルの値のみが異なる文がありますか。
2. 応答時間は、ライブラリ・キャッシュ・ミス数が非常に多いために遅くなっていますか。

注意： CURSOR_SHARING を FORCE または SIMILAR に設定すると、CURSOR_SHARING が EXACT に設定された状態で生成されたリテラルを持つアウトラインは使用されません。

CURSOR_SHARING=FORCE または SIMILAR でストアド・アウトラインを使用するには、FORCE または SIMILAR に設定された CURSOR_SHARING でアウトラインを生成するか、CREATE_STORED_OUTLINES パラメータでアウトラインを生成する必要があります。

CURSOR_SHARING = SIMILAR（または FORCE）を使用すると、多数の類似文を持ついくつかのアプリケーション上でのカーソルの共有を大幅に向上できるため、メモリー使用量が削減され、解析が高速になり、ラッチ競合が減少します。

Java プールの構成と使用

アプリケーションが Java を使用する場合、Java プールのデフォルト構成を変更する必要があるかどうかを調べる必要があります。

関連項目：『Oracle9*i* Java Developer's Guide』

REDO ログ・バッファの構成および使用

バッファ・キャッシュ内のデータ・ブロックに変更を行うサーバー・プロセスでは、REDO データをログ・バッファに生成します。LGWR は、次のいずれかに当てはまる場合に、REDO ログ・バッファからオンライン REDO ログにエントリをコピーする書込みを開始します。

- ログ・バッファの 1/3 が満たされる。
- LGWR が、COMMIT または ROLLBACK を実行するサーバー・プロセスによって通知される。
- DBWR が、書込みをするように LGWR に通知する。

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も、通常 LGWR は高速に書込みを行い、新しいエントリのバッファ内の領域が利用できることを保証します。

大きいバッファにより、新しいエントリのための領域がある可能性が高くなります。また、LGWR に、REDO レコードを効率よく書き出す機会を与えます（大規模な更新を行うシステム上の小さすぎるログ・バッファは、LGWR が REDO を継続的にディスクにフラッシュすることになり、ログ・バッファは 2/3 が空白のままになります）。

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・ライター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。この状況では、大きいログ・バッファは低速のディスクの影響を一時的に隠すことがあります。次のような方法も選択できます。

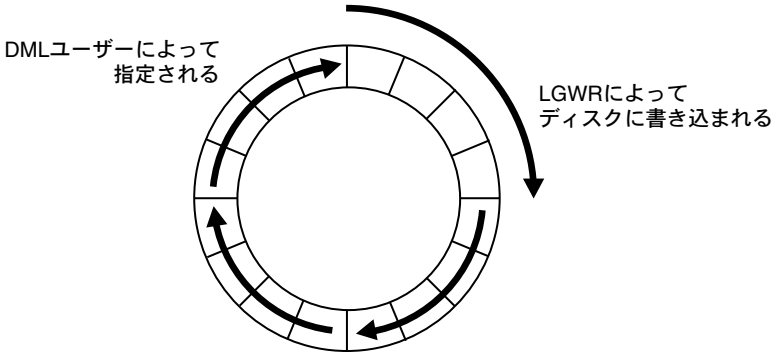
- チェックポイント機能またはアーカイブ・プロセスを改善する。
- すべてのオンライン・ログを高速の RAW デバイスに移動するなどの方法で、ログ・ライターのパフォーマンスを改善する。

REDO ログ・バッファの有効利用の簡単な例を示します。

- ログ・ライターが REDO ログ・エントリを効率的に書き込めるようにする、バッチ・ジョブに対する一括コミット操作
- 大量のデータをロードするときの NOLOGGING 操作の使用

REDO ログ・バッファのサイズは、初期化パラメータ LOG_BUFFER で決定されます。ログ・バッファ・サイズは、インスタンス起動後には変更できません。

図 14-2 REDO ログ・バッファ



ログ・バッファのサイズ設定

大量のデータを挿入、変更または削除するアプリケーションは、通常、デフォルトのログ・バッファ・サイズを変更する必要があります。ログ・バッファは総 SGA サイズと比較すると小さく、中規模サイズのログ・バッファは、多数の更新を実行するシステムでのスループットを大幅に向上させます。

そのようなシステムで合理的な最初の見積りは、ログ・バッファを 1MB にすることです。大半のシステムでは、ログ・バッファを 1MB より大きくサイズ設定しても、パフォーマンスの利点が得られません。バッファ・サイズを増やしても、パフォーマンスまたはリカバリ能力に対してマイナスの影響を及ぼしません。単に追加のメモリーが使用されます。

ログ・バッファの統計

統計 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 で問い合せることができます。

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';
```

redo buffer allocation retries の値は、ある時間間隔に対して 0（ゼロ）に近い値である必要があります。この値が一貫して増分する場合は、プロセスが REDO ログ・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こと、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されます。あるいは、チェックポイント機能またはアーカイブ・プロセスを改善してください。

別のデータ・ソースは、log buffer space 待機イベントがインスタンスの待機時間における重要な要因でないことをチェックするためのものです。重要な要因でなければ、バッファ・サイズはほぼ適切です。

PGA 作業メモリーの構成

プログラム・グローバル領域（PGA）は、サーバー・プロセスのデータおよび制御情報を含むプライベート・メモリー領域です。この領域に対するアクセスはそのサーバー・プロセスに対して排他的であり、そのかわりの役割を果たす Oracle コードでのみ読取りおよび書込みが行われます。そのような情報の例として、カーソルのランタイム領域があります。カーソルを実行するたびに、そのカーソルを実行するサーバー・プロセスの PGA メモリー領域内に、そのカーソルのための新しいランタイム領域が作成されます。

> **注意：** ランタイム領域の一部は、共有サーバーを使用するときに SGA 内に配置できます。

複雑な問合せ（たとえば、意思決定支援の問合せ）の場合、ランタイム領域の大部分が、次のようなメモリー集約型演算子で割り当てられた作業領域に使用されます。

- ソート・ベース演算子（たとえば、ORDER BY、GROUP BY、ROLLUP、ウィンドウ機能）
- ハッシュ結合
- ビットマップ・マージ

- ビットマップ作成
- 一括ロード操作で使用される書込みバッファ

ソート演算子は、作業領域（ソート領域）を使用して一連の行のメモリー内ソートを実行します。同様に、ハッシュ結合演算子は作業領域（ハッシュ領域）を使用して、ハッシュ表を左側から入力して作成します。

作業領域のサイズは、制御およびチューニングできます。一般に、作業領域を大きくすると、メモリー消費量は増えますが特定の演算子のパフォーマンスを大幅に向上できます。作業領域のサイズは、関連する SQL 演算子で割り当てられた入力データや補助メモリー構造を十分収容できるほど大きなサイズが理想的です。これは、作業領域の最適なサイズと呼ばれます。作業領域のサイズが最適なサイズより小さい場合は、入力データの部分に対して追加のパスが実行されるので、応答時間は増えます。これは、作業領域のワン・パス・サイズと呼ばれます。ワン・パスしきい値以下の場合は、作業領域のサイズが入力データ・サイズに比べて小さすぎる場合に入力データに対する複数のパスが必要です。このため、演算子の応答時間が大幅に増加する可能性があります。これは、作業領域のマルチ・パス・サイズと呼ばれます。たとえば、10GB のデータをソートすることが必要なシリアル・ソート操作では、最適なサイズの実行に 10GB を少し超える値を必要とし、ワン・パスを実行するには少なくとも 40MB が必要です。このソートが 40MB より少ない取得を行う場合は、入力データに対して複数のパスを実行する必要があります。

目標は、最適なサイズ（たとえば、90% を超える、または OLTP システム固有の場合は 100%）で大半の作業領域を動作させ、その一部をワン・パス・サイズ（たとえば、10% 未満）で動作させることです。マルチ・パスの実行は避けてください。大きなソートとハッシュ結合を実行する DSS システムの場合であっても、ワン・パスの実行のメモリー要件は相対的に少ない量です。妥当な PGA メモリー量で構成されたシステムは、入力データに対してマルチ・パスを実行する必要がありません。

Oracle9*i* 以前は、これらの作業領域の最大サイズは `SORT_AREA_SIZE`、`HASH_AREA_SIZE`、`BITMAP_MERGE_AREA_SIZE` および `CREATE_BITMAP_AREA_SIZE` の各パラメータで制御されていました。これらのパラメータの設定が難しいのは、最大作業領域のサイズがデータ入力サイズとシステム内でアクティブな作業領域の総数に基づいて理想的に選択されるためです。これらの 2 つの要因は、作業領域ごとに、また時間ごとにかなり異なります。したがって、様々な `*_AREA_SIZE` パラメータを最高の環境でチューニングすることは困難です。

Oracle9*i* では、自動 PGA メモリー管理を有効化することにより、PGA メモリーの割当て方法を単純化および改善できます。このモードでは、DBA が明示的に設定した全般的な PGA メモリー目標に基づいて、作業領域専用の PGA メモリーのサイズが動的に調整されます。自動 PGA メモリー管理を有効化するには、次の項の説明に従って、初期化パラメータ `PGA_AGGREGATE_TARGET` を設定する必要があります。

注意： このメカニズムは、共有サーバー接続には使用できません。

自動 PGA メモリー管理

自動 PGA メモリー管理モードで実行する場合、すべての専用セッションの作業領域のサイズ設定が自動で行われます。したがって、*_AREA_SIZE パラメータは、そのモードで動作するすべてのセッションで無視されます。インスタンスでアクティブな作業領域に使用できる PGA メモリーの総量は、指定時間に、PGA_AGGREGATE_TARGET 初期化パラメータから自動的に導出されます。この量は、システムの他のコンポーネントで割り当てられた PGA メモリー（たとえば、セッションで割り当てられた PGA）の量を PGA_AGGREGATE_TARGET から減算した値に設定されます。次に、その結果の PGA メモリーは、それぞれ特定のメモリー要件に基づいて個々のアクティブな作業領域に割り当てられます。

自動 PGA メモリー管理モードにおいて Oracle が主な目標としたものは、SQL 作業領域に割り当てられる PGA メモリーの量を動的に制御することで、DBA が設定した PGA_AGGREGATE_TARGET の制限を尊重することです。これと同時に、使用する PGA メモリー（キャッシュ・メモリー）の量が最適である作業領域の数を最大にして、すべてのメモリー集中型 SQL 演算子のパフォーマンスを最大限に引き出す試みも行っています。パラメータ PGA_AGGREGATE_TARGET で DBA により設定された PGA メモリーの制限が低すぎて、マルチ・パスを実行して PGA メモリーの消費をさらに削減して、PGA のターゲット制限を尊重する必要がある場合を除き、残りの作業領域は、ワン・パス・モードで実行されます。

新規インスタンスを構成する場合には、PGA_AGGREGATE_TARGET の適切な設定を正確に知ることは困難です。この設定は、次の 3 つの段階を実行して判別します。

1. PGA_AGGREGATE_TARGET の最初の見積りは経験に基づいて行う。
2. インスタンスで代理のワークロードを実行し、Oracle により収集された PGA 統計を使用してパフォーマンスを監視して、最大 PGA サイズが高く構成されたか低く構成されたかを確認する。
3. Oracle PGA のアドバイスの統計を使用して、PGA_AGGREGATE_TARGET をチューニングする。

これについては、次の項で詳しく説明します。

- PGA_AGGREGATE_TARGET の初期設定
- 自動 PGA メモリー管理のパフォーマンスの監視
- PGA_AGGREGATE_TARGET のチューニング

PGA_AGGREGATE_TARGET の初期設定

Oracle インスタンスに使用できる総メモリー量に基づいて、PGA_AGGREGATE_TARGET 初期化パラメータの値（たとえば、100000KB、2500MB、50GB など）を設定する必要があります。この値は後からインスタンス・レベルでチューニングしたり動的に変更できます。例 14-2 に一般的な状況を示します。

例 14-2 PGA_AGGREGATE_TARGET の初期設定

Oracle インスタンスが 4 GB の物理メモリーを持つシステム上で動作するように設定されていると仮定します。そのメモリーの一部は、オペレーティング・システムと同じハードウェア・システムで動作しているその他の Oracle 以外のアプリケーションに残しておく必要があります。たとえば、使用可能なメモリーの 80%（3.2 GB）のみを Oracle インスタンス専用にします。

次に、残りのメモリーを SGA と PGA に分割する必要があります。

- OLTP システムの場合、PGA メモリーの割合は使用可能なメモリーの総量の一部（例：20% が PGA 用、80% が SGA 用）とするのが普通です。
- メモリー集中型の大きな問合せを実行する DSS システムの場合、PGA メモリーには総量の最大 70%（この例では最大 2.2GB）までを使用できます。

PGA_AGGREGATE_TARGET パラメータの適切な初期値の例を次に示します。

- OLTP の場合 : `PGA_AGGREGATE_TARGET = (total_mem * 80%) * 20%`
- DSS の場合 : `PGA_AGGREGATE_TARGET = (total_mem * 80%) * 50%`

  total_mem はシステムで使用可能な物理メモリーの総量です。

この例では、4GB の *total_mem* の値を使用することにより、PGA_AGGREGATE_TARGET を DSS システムの場合は 1600MB に、OLTP システムの場合は 655MB に初期設定できます。

自動 PGA メモリー管理のパフォーマンスの監視

チューニング・プロセスを開始する前に、Oracle で収集される主要統計の監視および解釈方法を理解して、自動 PGA メモリー管理のパフォーマンスを評価する場合の参考にする必要があります。そのための動的パフォーマンス・ビューの例を次に示します。

- V$PGASTAT
- V$PROCESS
- V$SQL_WORKAREA_HISTOGRAM
- V$SQL_WORKAREA_ACTIVE
- V$SQL_WORKAREA

V$PGASTAT このビューは、PGA メモリー使用量および自動 PGA メモリー・マネージャに関するインスタンス・レベルの統計を示します。その例を次に示します。

```
SELECT * FROM V$PGASTAT;
```

この問合せの出力例を次に示します。

```
NAME                                                     VALUE UNIT
---------------------------------------------------------- ------- 
aggregate PGA target parameter                       524288000 bytes
aggregate PGA auto target                            463435776 bytes
global memory bound                                      25600 bytes
total PGA inuse                                        9353216 bytes
total PGA allocated                                   73516032 bytes
maximum PGA allocated                                698371072 bytes
total PGA used for auto workareas                           0 bytes
maximum PGA used for auto workareas                  560744448 bytes
total PGA used for manual workareas                         0 bytes
maximum PGA used for manual workareas                       0 bytes
over allocation count                                       0
total bytes processed                              4.0072E+10 bytes
total extra bytes read/written                     3.1517E+10 bytes
cache hit percentage                                     55.97 percent
```

V$PGASTAT に表示される主な統計は次のとおりです。

- aggregate PGA target parameter: これは初期化パラメータ PGA_AGGREGATE_TARGET の現在の値で、ここでは 500MB に設定されています。このパラメータを設定しない場合、その値は 0 となり、PGA メモリーの自動管理は無効になります。
- aggregate PGA auto target: 自動モードで実行する作業領域に使用できる PGA メモリーの量を示します。この量は、PGA_AGGREGATE_TARGET パラメータの値と現在の作業領域のワークロードから導出されます。したがって、Oracle で継続的に調整されます。この値が PGA_AGGREGATE_TARGET と比べて小さい場合、多くの PGA メモリーがシステムの他のコンポーネント（たとえば、PL/SQL や Java メモリー）で使用され、ソート作業領域にはメモリーがほとんど残されていません。自動モードで実行される作業領域には十分な PGA メモリーが残されている必要があります。
- global memory bound: AUTO モードで実行された作業領域の最大サイズを示します。この値は、作業領域のワークロードの現在の状態を反映するように継続的に調整されます。通常は、システム内のアクティブな作業領域の数が増えると、グローバル・メモリー・バウンドが縮小します。一般的には、グローバル・バウンドの値は 1MB を下回らないようにする必要があります。1MB 未満になった場合は、PGA_AGGREGATE_TARGET の値を増やす必要があります。

- total PGA allocated: インスタンスによって割り当てられた現在の PGA メモリー量を示します。通常この値は、PGA_AGGREGATE_TARGET の値未満に維持されます。ただし、作業領域のワークロードが急速に増えている場合や、初期化パラメータ PGA_AGGREGATE_TARGET の設定値が小さすぎる場合は、少量かつ短時間、その値を超過した PGA が割り当てられることがあります。
- total PGA used for auto workareas: 自動メモリー管理モードで実行する作業領域で現在消費されている PGA メモリーの量を示します。この数値から、PGA メモリーの他のコンシューマ（たとえば、PL/SQL や Java）で消費されるメモリーの量を判断できます。

```
PGA other = total PGA allocated - total PGA used for auto workareas
```

- over allocation count: この統計は、インスタンスの起動時から累積されます。PGA_AGGREGATE_TARGET の値が小さすぎて、前述の等式の PGA other コンポーネントと、作業領域のワークロードを実行するために必要な最小メモリーに対応できない場合は、PGA メモリーの過剰割当てとなる可能性があります。その場合、Oracle は初期化パラメータ PGA_AGGREGATE_TARGET を満たすことができないため、追加の PGA メモリーを割り当てる必要があります。過剰割当てが発生した場合は、アドバイス・ビュー V$PGA_TARGET_ADVICE の情報を使用して、PGA_AGGREGATE_TARGET の値を増やす必要があります。
- total bytes processed: インスタンスの起動後にメモリー集中型 SQL 演算子によって処理されたバイト数を示します。たとえば、ソート操作の入力サイズが、処理されたバイト数によって示されます。この数値は cache hit percentage 測定値を計算する場合に使用します。
- extra bytes read/written: 作業領域が最適に実行できない場合は、1 つ以上の余分なパスが入力データで実行されています。extra bytes read/written は、インスタンス起動後にこれらのパスで処理されたバイト数を示します。この数値は cache hit percentage を計算する場合にも使用します。
- cache hit percentage: この測定値は Oracle によって計算され、PGA メモリー・コンポーネントのパフォーマンスを反映します。この値はインスタンスの起動時から累積されます。値が 100% の場合は、インスタンス起動後にシステムが実行したすべての作業領域で、最適な量の PGA メモリーが使用されたことを意味します。それが理想的ですが、純粋な OLTP システムなどの場合を除き、そのようになることはほとんどありません。実際には、PGA メモリーの総サイズによって、ワン・パスやマルチ・パスを実行する作業領域も発生します。作業領域が最適に実行できない場合は、1 つ以上の余分なパスが入力データで実行されています。その場合、入力データのサイズと実行された余分なパスの数に比例して cache hit percentage が低下します。例 14-3 に、余分なパスによって cache hit percentage が受ける影響を示します。

例 14-3 キャッシュ・ヒット率の計算

4 つのソート操作が実行され、そのうちの 3 つは小さく（1MB の入力データ）、1 つは大きい（100MB の入力データ）という場合の単純事例を示します。4 つの操作で処理されるバイト数（BP）は 103MB です。小さなソートの 1 つがワン・パスを実行すると、1MB の入力データで余分なパスが実行されます。この 1MB という値は、extra bytes read/written（EBP）の数を示します。cache hit percentage は次の計算式で計算されます。

```
BP x 100 / (BP + EBP)
```

この場合の cache hit percentage は 99.03% で、ほぼ 100% です。他のすべてのソートを最適に実行している間、余分なパスを実行する小さなソートが 1 つであったことがこの値に反映されています。したがって、cache hit percentage はほぼ 100% になりますが、それはこの 1MB の余分なパスがわずかなオーバーヘッドであるためです。一方、大きなソートがワン・パスを実行するソートの場合、EBP は 1MB ではなく 100MB となり、cache hit percentage は 50.73% に低下しますが、それはこの余分なパスがもたらす影響が大きくなるためです。

V$PROCESS このビューでは、インスタンスに接続されている Oracle プロセス 1 つにつき行が 1 つあります。PGA_USED_MEM 列、PGA_ALLOC_MEM 列および PGA_MAX_MEM 列を使用して、これらのプロセスの PGA メモリー使用量を監視できます。その例を次に示します。

```
SELECT PROGRAM, PGA_USED_MEM, PGA_ALLOC_MEM, PGA_MAX_MEM
  FROM V$PROCESS;
```

この問合せの出力例を次に示します。

```
PROGRAM                        PGA_USED_MEM PGA_ALLOC_MEM PGA_MAX_MEM
------------------------------ ------------ ------------- -----------
PSEUDO                                    0             0           0
oracle@miflo (PMON)                  120463        234291      234291
oracle@miflo (DBW0)                 1307179       1817295     1817295
oracle@miflo (LGWR)                 4343655       4849203     4849203
oracle@miflo (CKPT)                  194999        332583      332583
oracle@miflo (SMON)                  179923        775311      775323
oracle@miflo (RECO)                  129719        242803      242803
oracle@miflo (TNS V1-V3)            1400543       1540627     1540915
oracle@miflo (P000)                  299599        373791      635959
oracle@miflo (P001)                  299599        373791      636007
oracle@miflo (P002)                  299599        373791      570471
oracle@miflo (P003)                  303899        373791      636007
oracle@miflo (P004)                  299599        373791      635959
```

V$SQL_WORKAREA_HISTOGRAM このビューには、インスタンス起動後に、最適なメモリー・サイズ、ワン・パス・メモリー・サイズおよびマルチ・パス・メモリー・サイズで実行された作業領域の総数を示します。このビューの統計は、作業領域の最適なメモリー要件によって定義されるバケットに副分割されます。各バケットは、列 LOW_OPTIMAL_SIZE および HIGH_OPTIMAL_SIZE の値で指定された最適メモリー要件の範囲によって識別されます。

例 14-4 および 14-5 で、V$SQL_WORKAREA_HISTOGRAM の 2 種類の使用方法を示します。

例 14-4 V$SQL_WORKAREA_HISTOGRAM への問合せ : 空でないバケット

最適に実行する（キャッシュされる）には 3MB のメモリーを必要とするソート操作の事例を示します。このソートで使用される作業領域に関する統計は、LOW_OPTIMAL_SIZE = 2097152（2 MB）および HIGH_OPTIMAL_SIZE = 4194303（4 MB - 1 バイト）で定義されるバケットに配置されます。これは 3MB が最適サイズの範囲内に収まるためです。統計は作業領域のサイズでセグメント化されます。最適、ワン・パスまたはマルチ・パスの各モードで作業領域を実行する場合のパフォーマンスの影響は、その作業領域のサイズに大きく依存するためです。

次の問合せでは、空でないすべてのバケットの統計が示されます。空のバケットは述語 where total_execution != 0 で削除されます。

```
SELECT LOW_OPTIMAL_SIZE/1024 low_kb,
       (HIGH_OPTIMAL_SIZE+1)/1024 high_kb,
       OPTIMAL_EXECUTIONS, ONEPASS_EXECUTIONS, MULTIPASSES_EXECUTIONS
  FROM V$SQL_WORKAREA_HISTOGRAM
 WHERE TOTAL_EXECUTIONS != 0;
```

この問合せの結果を次に示します。

```
LOW_KB HIGH_KB OPTIMAL_EXECUTIONS ONEPASS_EXECUTIONS MULTIPASSES_EXECUTIONS
------ ------- ------------------ ------------------ ----------------------
     8      16             156255                  0                      0
    16      32                150                  0                      0
    32      64                 89                  0                      0
    64     128                 13                  0                      0
   128     256                 60                  0                      0
   256     512                  8                  0                      0
   512    1024                657                  0                      0
  1024    2048                551                 16                      0
  2048    4096                538                 26                      0
  4096    8192                243                 28                      0
  8192   16384                137                 35                      0
 16384   32768                 45                107                      0
 32768   65536                  0                153                      0
 65536  131072                  0                 73                      0
131072  262144                  0                 44                      0
262144  524288                  0                 22                      0
```

1024 ～ 2048KB のバケットでは、551 の作業領域で最適な量のメモリーが使用されたこと、またワン・パス・モードで実行されたものが 16 ある一方で、マルチ・パス・モードで実行されたものはなかったことがこの問合せ結果に示されています。また、1MB 未満のすべての作業領域が最適モードで実行できたことも示されています。

例 14-5 V$SQL_WORKAREA_HISTOGRAM への問合せ : 最適パーセント

V$SQL_WORKAREA_HISTOGRAM を使用すると、起動後に作業領域が最適、ワン・パスまたはマルチ・パスの各モードで実行された回数の割合を調べることもできます。この問合せでは、一定のサイズ（最適メモリー要件が最低 64KB）の作業領域のみ考慮されます。

```
SELECT optimal_count, round(optimal_count*100/total, 2) optimal_perc,
       onepass_count, round(onepass_count*100/total, 2) onepass_perc,
       multipass_count, round(multipass_count*100/total, 2) multipass_perc
FROM
 (SELECT decode(sum(total_executions), 0, 1, sum(total_executions)) total,
         sum(OPTIMAL_EXECUTIONS) optimal_count,
         sum(ONEPASS_EXECUTIONS) onepass_count,
         sum(MULTIPASSES_EXECUTIONS) multipass_count
    FROM v$sql_workarea_histogram
   WHERE low_optimal_size > 64*1024);
```

この問合せの出力例を次に示します。

```
OPTIMAL_COUNT OPTIMAL_PERC ONEPASS_COUNT ONEPASS_PERC MULTIPASS_COUNT MULTIPASS_PERC
------------- ------------ ------------- ------------ --------------- --------------
         2239        81.63           504        18.37               0              0
```

この結果には、最適な量のメモリーを使用して実行できるのは、これらの作業領域の 81.63% であることが示されています。残り（18.37%）はワン・パスで実行されました。マルチ・パスで実行されたものはありませんでした。これは望ましい状態ですが、その理由は次のとおりです。

- マルチ・パス・モードでは、パフォーマンスが大幅に低下する可能性があります。マルチ・パスの作業領域が多いと、関連付けられた SQL 演算子の応答時間に大きな悪影響があります。
- ワン・パスの実行では多量のメモリーを必要としません。ワン・パス・モードで 1GB のデータをソートする場合に必要なメモリーはわずか 22MB です。

V$SQL_WORKAREA_ACTIVE このビューを使用すると、インスタンスでアクティブな（または実行中の）作業領域を表示できます。小さいアクティブなソート（64KB 以下）はビューから除外されます。すべてのアクティブな作業領域のサイズを正確に監視したり、それらの作業領域が一時セグメントに流用されているかどうかを判断するにはこのビューを使用します。例 14-6 に、このビューの代表的な問合せを示します。

例 14-6 V$SQL_WORKAREA_ACTIVE への問合せ

```
SELECT to_number(decode(SID, 65535, NULL, SID)) sid,
       operation_type OPERATION,
       trunc(EXPECTED_SIZE/1024) ESIZE,
       trunc(ACTUAL_MEM_USED/1024) MEM,
       trunc(MAX_MEM_USED/1024) "MAX MEM",
       NUMBER_PASSES PASS,
       trunc(TEMPSEG_SIZE/1024) TSIZE
  FROM V$SQL_WORKAREA_ACTIVE
 ORDER BY 1,2;

The output of this query might look like the following:
SID         OPERATION     ESIZE       MEM   MAX MEM  PASS   TSIZE
--- ----------------- --------- --------- --------- ----- -------
  8   GROUP BY (SORT)       315       280       904     0
  8         HASH-JOIN      2995      2377      2430     1   20000
  9   GROUP BY (SORT)     34300     22688     22688     0
 11         HASH-JOIN     18044     54482     54482     0
 12         HASH-JOIN     18044     11406     21406     1  120000
```

この出力は、作業領域がワン・パス・モードで動作しているハッシュ結合（PASS 列）を、セッション 12（列 SID）が実行していることを示しています。この作業領域は現在、11406KB のメモリー（MEM 列）を使用しており、過去に最大 21406KB の PGA メモリー（MAX MEM 列）を使用しました。また、サイズ 120000KB の一時セグメントにも流用されています。最終的に、列 ESIZE には、PGA メモリー・マネージャが予測する、このハッシュ結合での最大メモリー使用量が示されます。この最大値は、PGA メモリー・マネージャがワークロードに基づいて動的に計算します。

作業領域を割当て解除したとき、すなわち、関連する SQL 演算子の実行が完了したときに、作業領域は V$SQL_WORKAREA_ACTIVE ビューから自動的に削除されます。

V$SQL_WORKAREA 実行計画が 1 つ以上の作業領域を使用するカーソルがロードされるたびに、累積された作業領域の統計がメンテナンスされます。作業領域が割当て解除されるたびに、V$SQL_WORKAREA 表がその作業領域の実行統計で更新されます。

V$SQL_WORKAREA を V$SQL と結合して、作業領域をカーソルに関連付けることができます。V$SQL_PLAN とも結合でき、計画のどの演算子が作業領域を使用しているかを正確に判断できます。

例 14-7 に、V$SQL_WORKAREA 動的ビューでの代表的な問合せを 3 つ示します。

例 14-7 V$SQL_WORKAREA への問合せ

次の問合せでは、最もキャッシュ・メモリーを必要とする上位 10 個の作業領域を検索します。

```
SELECT *
  FROM
     ( SELECT workarea_address, operation_type, policy, estimated_optimal_size
         FROM V$SQL_WORKAREA
        ORDER BY estimated_optimal_size )
 WHERE ROWNUM <= 10;
```

次の問合せでは、ワン・パスまたはマルチ・パスで実行された 1 つ以上の作業領域を持つカーソルを検索します。

```
col sql_text format A80 wrap
SELECT sql_text, sum(ONEPASS_EXECUTIONS) onepass_cnt,
       sum(MULTIPASSES_EXECUTIONS) mpass_cnt
FROM V$SQL s, V$SQL_WORKAREA wa
WHERE s.address = wa.address
GROUP BY sql_text
HAVING sum(ONEPASS_EXECUTIONS+MULTIPASSES_EXECUTIONS)>0;
```

特定のカーソルのハッシュ値とアドレスを使用することにより、関連する作業領域に関する情報を含むカーソル実行計画が次の問合せで表示されます。

```
col "O/1/M" format a10
col name format a20
SELECT operation, options, object_name name,
       trunc(bytes/1024/1024) "input(MB)",
       trunc(last_memory_used/1024) last_mem,
       trunc(estimated_optimal_size/1024) optimal_mem,
       trunc(estimated_onepass_size/1024) onepass_mem,
       decode(optimal_executions, null, null,
              optimal_executions||'/'||onepass_executions||'/'||
              multipasses_executions) "O/1/M"
  FROM V$SQL_PLAN p, V$SQL_WORKAREA w
 WHERE p.address=w.address(+)
   AND p.hash_value=w.hash_value(+)
   AND p.id=w.operation_id(+)
   AND p.address='88BB460C'
   AND p.hash_value=3738161960;

OPERATION    OPTIONS  NAME     input(MB) LAST_MEM OPTIMAL_ME ONEPASS_ME O/1/M
------------ -------- -------- --------- -------- ---------- ---------- ------
SELECT STATE
SORT         GROUP BY              4582        8         16         16 16/0/0
HASH JOIN    SEMI                  4582     5976       5194       2187 16/0/0
TABLE ACCESS FULL     ORDERS         51
TABLE ACCESS FUL      LINEITEM     1000
```

アドレスおよびハッシュ値は、問合せでパターンを指定することにより V$SQL ビューから取得できます。その例を次に示します。

```
SELECT address, hash_value
FROM V$SQL
WHERE sql_text LIKE '%my_pattern%';
```

PGA_AGGREGATE_TARGET のチューニング

初期化パラメータ PGA_AGGREGATE_TARGET のチューニングを容易にするため、次の 2 種類の PGA アドバイス・パフォーマンス・ビューが提供されています。

- V$PGA_TARGET_ADVICE
- V$PGA_TARGET_ADVICE_HISTOGRAM

これらのビューを調べれば、経験的な方法で PGA_AGGREGATE_TARGET の値をチューニングする必要がありません。このビューの内容を使用すると、PGA_AGGREGATE_TARGET の値を変更したときに、主な PGA 統計がどのような影響を受けるかを調べることができます。

どちらのビューでも、可能性のある高い値および低い値を評価するため、予測に使用する PGA_AGGREGATE_TARGET の値は、そのパラメータの現在の値の分数または倍数から導出されます。予測に使用される値の範囲は、10MB ～最大 256GB です。

Oracle は、ワークロードの履歴を記録し、その履歴を異なる値の PGA_AGGREGATE_TARGET でシミュレートすることによって PGA アドバイス・パフォーマンス・ビューを生成します。このシミュレーション・プロセスはバックグラウンドで実行され、ワークロードの履歴を継続的に更新してシミュレーション結果を生成します。その結果は V$PGA_TARGET_ADVICE または V$PGA_TARGET_ADVICE_HISTOGRAM に問い合せることによって任意の時点で表示できます。

PGA アドバイス・パフォーマンス・ビューの自動生成を有効化するには、次のパラメータが設定されていることを確認してください。

- PGA_AGGREGATE_TARGET。自動 PGA メモリー管理が有効化されます。初期値の設定は、14-49 ページの「PGA_AGGREGATE_TARGET の初期設定」を参照してください。
- STATISTICS_LEVEL。TYPICAL（デフォルト）または ALL に設定します。このパラメータを BASIC に設定すると、PGA パフォーマンス・アドバイス・ビューの生成がオフになります。

これらの PGA アドバイス・パフォーマンス・ビューの内容は、インスタンス起動時または PGA_AGGREGATE_TARGET の変更時にリセットされます。

注意： シミュレーションに実際の実行のすべての要素を含めることはできません。したがって、導出された統計が、実際のパフォーマンス統計に完全には一致しない場合があります。PGA_AGGREGATE_TARGET を変更した後は、常にシステムを監視して、新しいパフォーマンスが予測どおりであるかどうかを確認してください。

V$PGA_TARGET_ADVICE このビューでは、初期化パラメータ PGA_AGGREGATE_TARGET の値を変更した場合に、V$PGASTAT の統計 cache hit percentage および over allocation count がどのような影響を受けるかを予測します。例 14-8 に、このビューの代表的な問合せを示します。

例 14-8 V$PGA_TARGET_ADVICE への問合せ

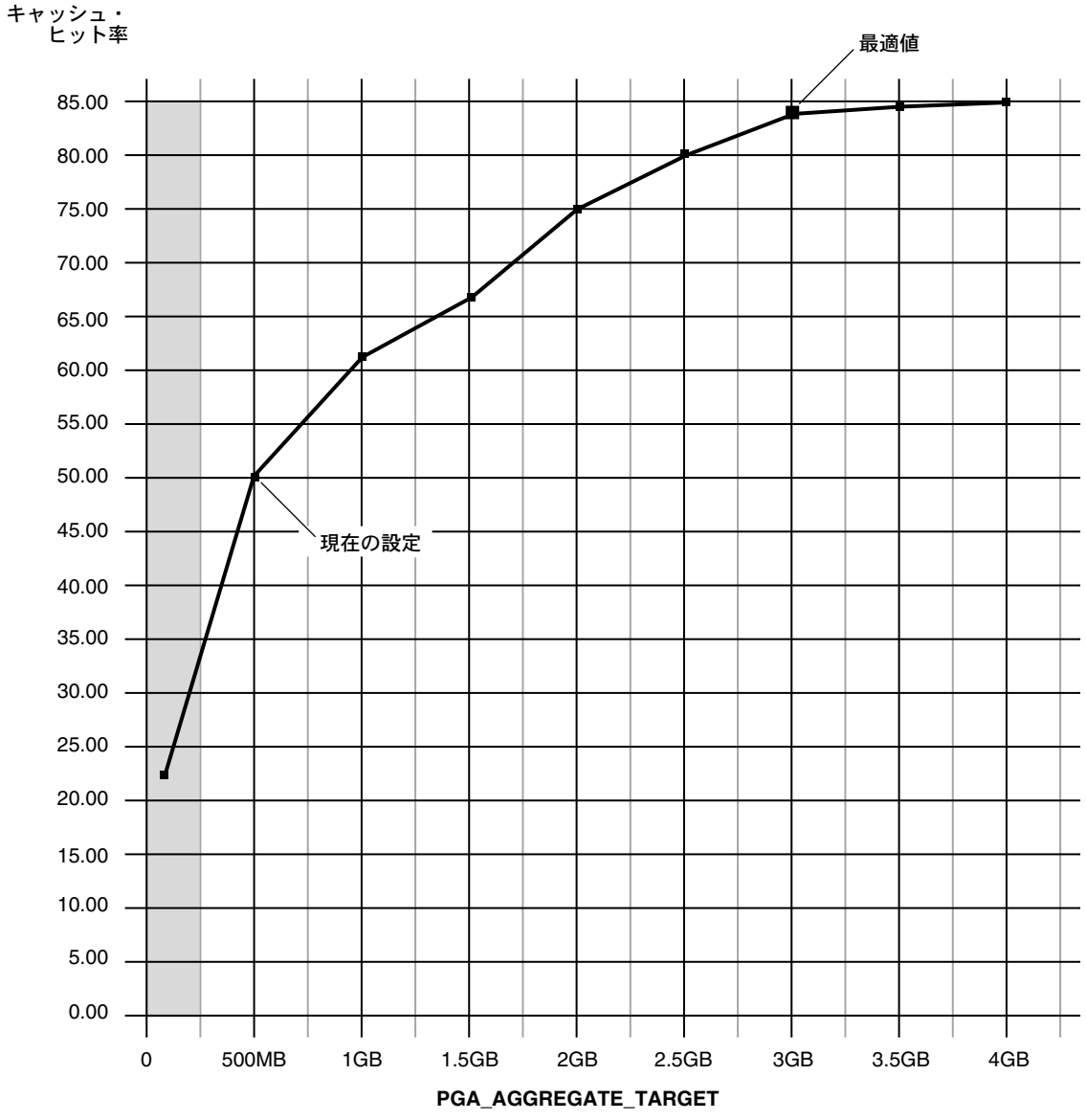
```
SELECT round(PGA_TARGET_FOR_ESTIMATE/1024/1024) target_mb,
       ESTD_PGA_CACHE_HIT_PERCENTAGE cache_hit_perc,
       ESTD_OVERALLOC_COUNT
  FROM V$PGA_TARGET_ADVICE;
```

この問合せの出力例を次に示します。

```
 TARGET_MB  CACHE_HIT_PERC  ESTD_OVERALLOC_COUNT
----------  --------------  --------------------
        63              23                   367
       125              24                    30
       250              30                     3
       375              39                     0
       500              58                     0
       600              59                     0
       700              59                     0
       800              60                     0
       900              60                     0
      1000              61                     0
      1500              67                     0
      2000              76                     0
      3000              83                     0
      4000              85                     0
```

この問合せの結果は例 14-3 に示すように図示できます。

図 14-3 V$PGA_TARGET_ADVICE の図示



この曲線は、PGA_AGGREGATE_TARGET の増加による PGA cache hit percentage の向上状況を示しています。グラフ内の影付きの部分は over allocation ゾーンで、列 ESTD_OVERALLOCATION_COUNT の値が 0（ゼロ）以外になります。これは PGA メモリーの最低所要量にも達しないほど PGA_AGGREGATE_TARGET が小さいことを示します。PGA_AGGREGATE_TARGET を over allocation ゾーンに設定すると、メモリー・マネージャによってメモリーが過剰割当てされ、実際に消費された PGA メモリーが設定された制限を超過します。したがって、PGA_AGGREGATE_TARGET の値をそのゾーンに設定しても意味がありません。この例では、PGA_AGGREGATE_TARGET は最低 375MB に設定する必要があります。

注意： PGA cache hit percentage の理論的な最大値が 100% の場合でも、作業領域の実際の最大サイズには制限があります。したがって、PGA_AGGREGATE_TARGET の値をさらに増やしても、理論的な最大値に達しない場合があります。これが発生するのは、最適メモリー要件が大きく、cache hit percentage の値が低いヒット率（90% など）に下がる可能性のある大規模 DSS システムのみです。

---

over allocation を超えると、PGA cache hit percentage が急速に増加します。これは最適、またはワン・パスで実行される作業領域の数が増加し、マルチ・パス実行の数が減少するためです。この例では、500MB 付近で曲線の変化が発生していますが、これはほとんどの（場合によってはすべての）作業領域が最適、または少なくともワン・パスで実行可能になるポイントに対応しています。その変化以降も cache hit percentage は低いペースで増加し、先細りが始まり、PGA_AGGREGATE_TARGET の増加によるわずかな向上しか見られないポイントに達します。図 14-3 に、PGA_AGGREGATE_TARGET が 3GB に達したときのその状態を示します。この時点の cache hit percentage は 83% で、PGA メモリーを 1GB 増やしてもわずか（2%）しか向上しません。この例では、PGA_AGGREGATE_TARGET の最適値は 3GB と考えられます。

PGA_AGGREGATE_TARGET は、最適値に設定するか、少なくとも over allocation ゾーンより上の領域の可能な最大値に設定するのが理想的です。一般的には、PGA cache hit percentage は 60% 以上に設定します。60% の時点でシステムは、理想的な状況で実際に処理する必要のあるバイト数のほぼ 2 倍の量を処理するためです。この例では、PGA_AGGREGATE_TARGET を最低 500MB から 3GB にできるだけ近く設定するのが理想的です。ただし、PGA_AGGREGATE_TARGET パラメータの正しい設定は、実際には PGA コンポーネントにどれだけのメモリーを使用できるかによって異なります。一般的に、PGA メモリーを追加する場合は、共有プールまたはバッファ・キャッシュの場合と同様に、一部の SGA コンポーネントのメモリーを減らす必要があります。これは Oracle インスタンスに使用する全メモリーが、システムで使用できる物理メモリー量の制約を受ける場合があるためです。したがって、システム内で使用可能なメモリーと、様々な SGA コンポーネントのパフォーマンス（共有プールのアドバイザの統計およびバッファ・キャッシュ・アドバイザの統計で監視）という、より大きな視点で、PGA メモリーの増量の決定を下す必要があります。メモリーを SGA から取得できない場合は、システムへの物理メモリーの追加を検討します。

関連項目：

- 14-29 ページ「共有プールのアドバイザ統計」
- 14-6 ページ「バッファ・キャッシュ・アドバイザの統計」

PGA_AGGREGATE_TARGET のチューニング方法 PGA_AGGREGATE_TARGET をチューニングする場合は、チューニング・ガイドラインとして次の手順に従います。

1. メモリーが過剰割当てされないよう PGA_AGGREGATE_TARGET を設定します。過剰割当てゾーンに設定しないでください。例 14-8 では、PGA_AGGREGATE_TARGET は最低 375MB に設定する必要があります。
2. 過剰割当てを解消した後、応答時間の要件およびメモリーの制約に基づいて PGA cache hit percentage を極力最大化します。例 14-8 では、PGA に割当て可能なメモリーに制限 *X* があると仮定しています。
   - この制限 *X* が最適値を超過している場合は、PGA_AGGREGATE_TARGET を最適値に設定します。この時点では、PGA_AGGREGATE_TARGET へのメモリー割当ての増加に伴う有益性はごくわずかです。例 14-8 で 10GB を PGA 専用とする場合は、PGA_AGGREGATE_TARGET を最適値である 3GB に設定します。残りの 7GB は SGA 専用となります。
   - 制限 *X* が最適値より小さい場合は、PGA_AGGREGATE_TARGET を *X* に設定します。例 14-8 で 2GB のみを PGA 専用とする場合は、PGA_AGGREGATE_TARGET を 2GB に設定し、cache hit percentage 75% を受け入れます。

さらに、Oracle で収集され、インスタンスの起動時から累積されるほとんどの統計と同様に、時間間隔の初めと終わりにおけるビューのスナップショットを取得できます。その時間間隔の予測値は次のように導出できます。

```
        estd_overalloc_count = (difference in estd_overalloc_count between the two snapshots)

                                        (difference in bytes_processed between the two snapshots)
estd_pga_cache_hit_percentage = ------------------------------------------------------------------
                                (difference in bytes_processed + extra_bytes_rw  between the two snapshots )
```

V$PGA_TARGET_ADVICE_HISTOGRAM このビューは、初期化パラメータ PGA_AGGREGATE_TARGET の値を変更したときに、パフォーマンス・ビュー V$SQL_WORKAREA_HISTOGRAM に表示される統計がどのように影響を受けるかを予測します。動的ビュー V$PGA_TARGET_ADVICE_HISTOGRAM を使用すると、予測に使用する一連の PGA_AGGREGATE_TARGET 値における、最適、ワン・パス、マルチ・パスの各作業領域の予測実行回数に関する詳細情報を表示できます。

V$PGA_TARGET_ADVICE_HISTOGRAM ビューは V$SQL_WORKAREA_HISTOGRAM ビューと同じであり、予測に使用する PGA_AGGREGATE_TARGET 値を示す 2 つの追加列があります。したがって、PGA_AGGREGATE_TARGET の希望値を選択するための追加の述語を使用し、V$SQL_WORKAREA_HISTOGRAM ビューに対して実行される任意の問合せがこのビューで使用できます。

例 14-9 V$PGA_TARGET_ADVICE_HISTOGRAM への問合せ

次の問合せでは、初期化パラメータ PGA_AGGREGATE_TARGET の値を現在の値の 2 倍に設定したときの、V$SQL_WORKAREA_HISTOGRAM の予測内容が表示されます。

```
SELECT LOW_OPTIMAL_SIZE/1024 low_kb, (HIGH_OPTIMAL_SIZE+1)/1024 high_kb,
       estd_optimal_executions estd_opt_cnt,
       estd_onepass_executions estd_onepass_cnt,
       estd_multipasses_executions estd_mpass_cnt
  FROM v$pga_target_advice_histogram
 WHERE pga_target_factor = 2
   AND estd_total_executions != 0
 ORDER BY 1;
```

この問合せの出力例を次に示します。

```
LOW_KB   HIGH_KB   ESTD_OPTIMAL_CNT   ESTD_ONEPASS_CNT   ESTD_MPASS_CNT
------   -------   ----------------   ----------------   --------------
     8        16             156107                  0                0
    16        32                148                  0                0
    32        64                 89                  0                0
    64       128                 13                  0                0
   128       256                 58                  0                0
   256       512                 10                  0                0
   512      1024                653                  0                0
  1024      2048                530                  0                0
  2048      4096                509                  0                0
  4096      8192                227                  0                0
  8192     16384                176                  0                0
 16384     32768                133                 16                0
 32768     65536                 66                103                0
 65536    131072                 15                 47                0
131072    262144                  0                 48                0
262144    524288                  0                 23                0
```

この出力は、PGA_AGGREGATE_TARGET を 2 のべき乗で増加させると、16MB 未満のすべての作業領域を最適モードで実行できることを示しています。

関連項目：『Oracle9*i* データベース・リファレンス』

V$SYSSTAT および V$SESSTAT

V$SYSSTAT ビューと V$SESSTAT ビューの統計は、最適なメモリー・サイズ、ワン・パス・メモリー・サイズおよびマルチ・パス・メモリー・サイズで実行された作業領域の総数を示します。これらの統計は、インスタンスまたはセッションが開始された後から累積されます。

次の問合せは、インスタンスの開始後にこれらのモードで作業領域が実行された回数の総数とパーセンテージを示します。

```
SELECT name profile, cnt, decode(total, 0, 0, round(cnt*100/total)) percentage
    FROM (SELECT name, value cnt, (sum(value) over ()) total
    FROM V$SYSSTAT
    WHERE name like 'workarea exec%');
```

この問合せの出力例を次に示します。

```
PROFILE                                    CNT PERCENTAGE
---------------------------------- ---------- ----------
workarea executions - optimal             5395         95
workarea executions - onepass              284          5
workarea executions - multipass              0          0
```

SORT_AREA_SIZE の構成

SORT_AREA_SIZE を使用したソート操作のチューニングは、Oracle 共有サーバー・オプションを実行している構成か、自動メモリー管理モードで実行していない構成にのみ関連します。後者の場合は、自動メモリー管理モードへの切替えをお薦めします。そのほうが管理しやすく、手動でチューニングされるシステムより優れている場合が多いためです。

この項では、次の項目について説明します。

- ソートの基礎
- メモリー・ソートとディスク・ソートの認識
- アプリケーションの特性
- SORT_AREA_SIZE の考慮事項
- SORT_AREA_RETAINED_SIZE の考慮事項
- NOSORT の使用によるソートを行わない索引作成
- GROUP BY NOSORT の使用

ソートの基礎

ソートは、リクエスタにデータを戻す前に、一定の基準に従ってデータの順序付けを行う操作です。

ソートを行う操作は、次のとおりです。

- CREATE INDEX
- SELECT .... ORDER BY
- SELECT DISTINCT
- SELECT .... GROUP BY
- SELECT ... CONNECT BY

- SELECT ... CONNECT BY ROLLUP
- ソート / マージ結合

> **関連項目：** ソートを実行する SQL 文のリストは、『Oracle9*i* データベース概要』を参照してください。

WORKAREA_SIZE_POLICY パラメータを MANUAL に設定すると、ソートに割り当てられる最大メモリー量はパラメータ SORT_AREA_SIZE で定義されます。ソート操作が SORT_AREA_SIZE メモリーに完全には収まらない場合、ソートはいくつかのフェーズに分割されます。各フェーズの一時出力は、ディスク上の一時セグメントに格納されます。これらのソート・セグメントが作成される表領域は、ユーザーの一時表領域です。

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くない場合は、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。

> **注意：** ディスク・ソート操作の一部として実行する読込みと書込みは、バッファ・キャッシュをバイパスします。

メモリー・ソートとディスク・ソートの認識

ソート・アクティビティを反映する統計を収集し、V$SQLAREA や V$SYSSTAT などの動的パフォーマンス・ビューに統計を格納します。

表 14-4 は、ソート動作を決定する V$SYSSTAT からの統計をリストしたものです。

表 14-4 ソート動作を反映する V$SYSSTAT 統計

統計	説明
sorts (memory)	十分に小さいため、ディスク上の一時ソート・セグメントへの I/O を行わずにメモリーで完全に実行できるソートの数。
sorts (disk)	大きすぎるためにメモリーで完全に実行できずに、ディスク上の一時ソート・セグメントへの I/O を必要とするソートの数。

たとえば、次の問合せでは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)');
```

この問合せの出力例を次に示します。

```
NAME                                                                  VALUE
---------------------------------------------------------------- ----------
sorts (memory)                                                    430418566
sorts (disk)                                                          33255
```

また、ソートを実行する個々の SQL 文を検出するには、V$SQLAREA ビューを問い合せます。SORTS で行の順序付けを行い、大半のソートを行う SQL 文を指定します。たとえば、次のようにします。

```
SELECT HASH_VALUE, SQL_TEXT, SORTS, EXECUTIONS
  FROM V$SQLAREA
 ORDER BY SORTS;
```

OLTP 環境における最良の解決方法は、ソート・アクティビティを回避するように SQL 文をチューニングできるかどうかを調べることです。

アプリケーションの特性

OLTP システムでのパフォーマンスを最高にするには、大半のソートがメモリー内のみで発生するようにする必要があります。ディスクに書き込まれたソートは、パフォーマンスに影響を与える可能性があります。ソート領域サイズに収まらないソートを OLTP アプリケーションが頻繁に実行し、不要なソートを回避するようにそのアプリケーションをチューニングされている場合、インスタンス全体の SORT_AREA_SIZE パラメータを大きくすることを考慮してください。

ディスクへのソートを行う平均のソートより大きいソートを実行するプログラムがわずかしかない場合は、そのワークロードまたはアプリケーション（たとえば、索引の作成）についてのみセッション・レベルで SORT_AREA_SIZE を変更できます。

通常、DSS アプリケーションは大量のデータにアクセスします。これらのタイプのアプリケーションは、アプリケーションの性質と扱われるデータ量を考慮すると、ディスクへのソートを実行すると予測されます。DSS アプリケーションでは、ディスク・ソートを最も効果的に行うように最適な SORT_AREA_SIZE を指定することが重要です。メモリーに割り当てるソートを増やしても、必ずしもソートは高速になりません。

SORT_AREA_SIZE の考慮事項

SORT_AREA_SIZE を選択する際の主な考慮事項は、メモリー使用量とソート・パフォーマンスとのバランスをとることです。

Oracle8 リリース 8.0 以上では、ソートの開始時の 1 回のメモリー割当てで SORT_AREA_SIZE 全体が割り当てられません。メモリーは、必要に応じて最大 SORT_AREA_SIZE まで DB_BLOCK_SIZE チャンク内で割り当てられます。

つまり、SORT_AREA_SIZE メモリーを増やすことは、システム上の大半のプロセスがソートを実行し、そのために最大の割当てを使用する場合の考慮事項です。このような状況で

は、インスタンス全体に対して SORT_AREA_SIZE を増やすと、オペレーティング・システムからさらに多くのメモリーが割り当てられます（専用接続の場合、すなわち共有サーバー環境を使用しない場合）。これは、システムに使用可能な空きメモリーがある場合には必ずしも問題ではありません。ただし、十分な空きメモリーがない場合、ページングまたはスワッピングが発生します。

共有サーバー環境を使用すると、追加のメモリーは共有プールか、またはラージ・プールが構成されている場合（LARGE_POOL_SIZE 初期化パラメータが指定されている場合）は、ラージ・プールから割り当てられます。共有プールで使用される実際のメモリー量は、SORT_AREA_SIZE、SORT_AREA_RETAINED_SIZE およびソートで使用される実際の割当てのうち、より少ない量です。

SORT_AREA_SIZE が小さすぎる場合、ソートはあまり効率的には実行されません。これは、メモリー内ソートがディスク・ソートになっているか、あるいはソートの処理に必要なソート・ランの回数が必要以上に多い可能性があるということです。このような状況はいずれも、パフォーマンスを非常に低下させる可能性があります。

SORT_AREA_SIZE を大きくした後は、パフォーマンスは必ずしも改善されないことに注意してください。

SORT_AREA_SIZE の増加 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、SORT_AREA_SIZE の値を大きくすることで向上することがあります。そのかわり、DSS 環境では、SORT_AREA_SIZE を大きくしても、ソートをメモリー専用ソートにできない可能性があります。ただし、現在の値と選択した新しい値によっては、ソートをさらに高速にできます。

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。SORT_AREA_SIZE をどの程度大きくすることがシステムにとって有効かを判断する必要があります。

SORT_AREA_RETAINED_SIZE の考慮事項

SORT_AREA_RETAINED_SIZE パラメータは、ユーザーまたは問合せの次の部分へのソート済みデータの送信をソートが開始した後に、Oracle がソート領域のサイズを削減する際のメモリーの下限を決定します。

専用接続では、解放されたメモリーはオペレーティング・システムには解放されず、セッションでの再利用に割り当てられます。

ただし、接続が共有サーバーを介して行われる場合、SORT_AREA_RETAINED_SIZE を設定することでメモリー上の利点が生まれる可能性があります。ソートが完了した後でこのパラメータを設定すると、ソート済みデータは SGA に格納されます。SGA で使用されるメモリー量は、実際の使用量または SORT_AREA_RETAINED_SIZE（設定した場合）よりも少ない量です。設定されていない場合は、SORT_AREA_SIZE です。共有サーバー環境で SORT_AREA_RETAINED_SIZE を設定することが有益なのはこのためです。

注意： 共有サーバーを介してデータベースに対して行われた接続では、通常、大きいソートは実行しないでください。

共有サーバーを使用するとメモリーが節約される場合がありますが、SORT_AREA_RETAINED_SIZE を設定すると、追加の I/O がディスク上の一時セグメントに対してデータの読込みおよび書込みを行います（SORT_AREA_RETAINED_SIZE バイトより多くのバイトがソートに必要な場合）。

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。ただし、ソートを行わずに索引を作成できます。表の中の行が昇順でロードされていると、ソートを実行させずに、索引をより高速に作成できます。

NOSORT 句 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 句を使用してください。たとえば、次の CREATE INDEX 文は、employees 表で行のソートが発生しないように、employees 表の last_name 列に対して索引 my_emp_name_ix を作成します。

```
CREATE INDEX my_emp_name_ix
   ON employees(last_name)
   NOSORT;
```

この SQL 例では、表内の行は索引付きの列値の昇順にロードされるものと仮定します。

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 句を使用する場合 データを事前にソートし、その順序でロードすることが、表をロードする最も高速な方法ではない場合があります。

- 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 句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 句で索引を作成してください。

GROUP BY NOSORT の使用

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取り出した場合や、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。

15

I/O 構成および設計

I/O サブシステムは、Oracle データベースに不可欠なコンポーネントです。この章では、基本的な I/O の概念を紹介し、データベースの様々な部分の I/O 要件について説明し、I/O サブシステムの設計のための構成例を示します。

この章には次の項があります。

I/O の理解

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 I/O によって制限されます。CPU タイムの大部分を I/O アクティビティが完了するまでの待機に使用するアプリケーションは I/O バウンドと呼ばれます。

Oracle は、適切に作成されたアプリケーションのパフォーマンスが、I/O で制限されないように設計されています。I/O システムが最大限またはそれに近い状態で動作しており、許容時間内に I/O リクエストに対応できない場合は、I/O のチューニングを行うと、アプリケーションのパフォーマンスを向上できます。ただし、アプリケーションが I/O バウンドではない場合（たとえば、CPU が制限要因である場合）、I/O をチューニングしてもパフォーマンスを改善できません。

I/O レイアウトの設計

I/O システムを設計するときは、次のデータベース要件を考慮してください。

1. ディスクの最小バイト数などの記憶域
2. 継続的（24 時間× 7 日）または営業時間のみなどの可用性
3. I/O スループットやアプリケーション応答時間などのパフォーマンス

多くの I/O 設計では、パフォーマンスが問題とならないと仮定して記憶域要件および可用性要件の計画を立てています。ただし、これに該当しない場合もあります。構成するディスクおよびコントローラの数は I/O スループットおよび冗長性の要件で判断することが最適です。この場合、ディスクのサイズは記憶域要件で判断できます。

ディスクのパフォーマンスおよび信頼性

データベースにおいて、I/O サブシステムは、システムの可用性、パフォーマンスおよびデータの整合性にとって重要です。これらの領域の弱点は、データベース・システムの安定性、拡張性または信頼性を欠く可能性があることです。

すべての I/O サブシステムは、磁気ディスク・ドライブを使用します。従来の磁気ディスク・ドライブには移動部品があります。これらの移動部品には設計上の寿命があるため、これら部品の信頼性およびパフォーマンスのばらつきは、製造上の許容誤差に影響されます。理論的に同一のすべてのドライブが同様に作動するとは限らず、時間の経過とともに正常に動作しなくなる可能性があります。大型ディスク構成をアセンブルする場合は、ディスク・ドライブの平均障害間隔時間（MTBF）およびディスク数から、ディスク障害が日常的に発生しているかどうかを確認する必要があります。システム（データ）の中核機能はディスク上に常駐するため、ディスク障害が頻繁に発生するのは望ましくありません。

ディスク・テクノロジ

I/O サブシステムの主要なコンポーネントであるディスク・ドライブは、近年ほとんど変化していません。唯一変化したのは、ドライブの容量が 1GB 以下から 50GB 以上へ増えたこと、ディスク・アクセス時間（rpm）が多少改善されたこと、そしてこれによりスループットが向上したことです。その一方で、CPU は、改良によって 18 か月ごとにパフォーマンスが 2 倍に向上し続けてきました。

ディスク容量に必要なディスク数を設定する場合、時間の経過につれてディスクのサイズが増えるため、必要なディスク数は減るという、ディスク・ドライブのパラドックスがあります。ただし、パフォーマンスでディスク数を設定する場合、18 か月ごとに CPU 当たりのディスク数を倍増する必要があります。このパラドックスに対応することは、多くのシステム構成、特に使用するディスク数を減らしてシステムのコストを低くしようとするシステム構成では問題でした。

製造される実際のディスクのサイズの他に、ディスク・サブシステムの接続方法が変化しました。小さいシステムでは、一般に、ディスク・ドライブは多数のディスク・コントローラのうちの 1 つを介して小型コンピュータ・システム・インタフェース（SCSI）で個々にホスト・マシンに接続されます。ハイエンド・システムの場合、ディスクのミラー化およびストライプ化はパフォーマンスおよび可用性に不可欠です。これらの要件により、多数のディスクが複雑な配線の構成で接続されます。

ただし、ディスクのテクノロジ自体は大幅に変化していませんが、I/O サブシステムは前述の多くの問題を克服するディスク・アレイに発展しました。システムはディスク・ミラー化を行い、ディスクのホット・スワッピングを提供し、多くの場合はファイバ・インタフェースによりホストへの接続が簡単になりました。さらに進化したディスク・アレイは、実際にはそれ自体が小型のコンピュータで、専用の CPU、高性能なリジリエント書込みを実現するバッテリ付きのメモリー・キャッシュ、ダイアル・ホーム診断機能およびプロキシ・バックアップを備えているため、ホスト・オペレーティング・システムを経由せずにバックアップができます。

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。ほとんどのディスクには、アクセス回数（毎秒の I/O 操作）と毎秒のデータ転送量（I/O データ・レート、つまり**スループット**）の両方に制限があります。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。

ロード・バランシングおよびストライプ化

パフォーマンス・エンジニアの目標は、使用可能なデバイス間に I/O 負荷を均等に分散することです。これをロード・バランシングまたは I/O の分散と呼びます。以前は、ロード・バランシングは手動で行う必要がありました。データベース管理者は、各データ・ファイルの I/O 要件および特性を判断し、1 つのディスク上にまとめて配置できるファイルに基づき、I/O レイアウトを設計し、すべてのディスクにアクティビティを均等に分散します。

特定の表または索引が I/O 集中型であった場合は、I/O 負荷をさらに分散するために、データを手動で分散する（ストライプ化する）オプションも DBA にありました。これは、個別のデータ・ファイル上にオブジェクトのエクステントを分割し、次にデバイス上にデータ・ファイルを分散することによって実現されました。

基本的に、ストライプ化によって、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データの異なる部分に同時にアクセスできます。オペレーティング・システム、ハードウェア・ベンダーおよびサード・パーティのソフトウェア・ベンダーは、多数の物理デバイス間で頻繁に使用されるファイルをストライプ化するツールを提供しているため、I/O をバランス化する作業は DBA にとってきわめて簡単です。

ストライプ化は、OLTP 環境で多数の行を持つ表へのランダム・アクセスを最適化する場合、および DSS 環境でのパラレル操作で大量のデータを素早くスキャンできるようにする場合に便利です。ストライプ化の手法は、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有効です。ファイルをストライプ化することで、Recovery Manager のバックアップおよびリストアのパフォーマンスを向上させることもできます。

使用可能なハードウェアにとって同時におこる負荷が過剰な場合、ストライプ化では問題が軽減されません。

ストライプ化と RAID

各特定システムのリカバリ可能要件も考慮する必要があります。Redundant arrays of inexpensive disks（RAID）構成では、ストライプ化のオプションが提供されると同時にデータの信頼性が改善されます。選択する RAID レベルは、パフォーマンスおよびコストにより異なります。様々な RAID レベルは、その I/O 特性に従って、異なるタイプのアプリケーションに適合します。

次に、RAID 構成に最適なアプリケーションとともに、データベース・ファイルに使用する最も一般的な RAID 構成の概要を示します。この概要の説明はきわめて一般的なものです。特定の構成の実装の詳細は、ハードウェアおよびソフトウェアのベンダーに問い合せてください。

RAID 0: ストライプ化

ファイルは多数の物理ディスク間でストライプ化されます。この構成では読込みおよび書込みのパフォーマンスが提供されますが、信頼性は提供されません。

注意： RAID 0 は読込みおよび書込みともに最高のパフォーマンスを提供しますが、冗長性がないため、実際には RAID システムではありません。Oracle では、RAID 0 システムに実働データベース・ファイルを配置しないようにお薦めします。

RAID 1: ミラー化

物理ディスクを使用して、同時にメンテナンスされるファイルの特定の個数 *n* のコピーを格納します。必要なディスク数は $n \times m$ です。*m* は、元のファイルを格納するのに必要なディスク数です。RAID 1 では、高い信頼性および優れた読込み発生率を提供します。*n* 個のコピーをメンテナンスするには *n* 回の書込みが必要なため、書込みのコストが高くなる場合があります。

RAID 0+1: ストライプ化およびミラー化

このレベルでは、RAID 0 と RAID 1 の技術を組み合せます。高い信頼性とともに、RAID 1 よりも優れた読込みおよび書込みのパフォーマンスを実現するため、このレベルは幅広く使用されています。

注意： ミラー化により、I/O ボトルネックが発生する場合があります。一般に、各ミラーへの書込みプロセス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 の問題を回避するには、プライマリ・データベースの場合と同じディスク数を使用してコピーをストライプ化します。

RAID 5: ストライプ化および冗長性

RAID 5 のストライプ化は、RAID 0 のストライプ化と似ています。ディスク障害のために消失したデータをリカバリできるようにするには、グループ内のディスク全体で一様に、すべてのディスクに関するパリティ・データをストライプ内に格納します。RAID 1 と比較した場合、その利点はディスク・コストの節約にあります。RAID 5 は優れた信頼性を実現します。順次読込みは最も有利ですが、書込みパフォーマンスは低下する可能性があります。この構成は、書込み中心のアプリケーションに対しては理想的ではない場合があります。

最近の RAID 5 の実装では、バッテリ付きのメモリー（NVRAM）を多量にインストールすることで、従来の多数の RAID 5 の制限を回避しています。キャッシュは、次の 2 つの方法によってパフォーマンスのペナルティをオフセットします。

- 物理 I/O 操作の遅れを助長します。小さな書込みペナルティは、それを開始した個々の I/O のボトルネックにならないためです。
- ディスク・アレイが小さな書込みを大きなバッチにグループ化するのを支援し、小さな書込みペナルティを発生させないようにします。

短所は次のとおりです。

- キャッシュが高価なため、RAID 1 に対する RAID 5 の（記憶域 1 バイト当たりの）コストの優位性が損なわれる可能性があります。
- I/O スループットの要件がキャッシュ容量を超過する場合があります。相当な速度で I/O コールが継続すると、キャッシュがいっぱいになり、アレイと物理ディスクが再同期化されるまで、アレイへのすべての I/O の停止の原因となる可能性があります。

予算、パフォーマンスおよび可用性のバランス化

I/O サブシステムの選択およびレイアウトの設計では、予算、パフォーマンスおよび可用性のバランスが課題です。スループットの要件および実行する I/O の数の増加に伴い、I/O サブシステムを拡張できるようにすることが不可欠です。拡張可能な I/O システムを実現するには、選択したシステムの停止時間を最小にすると同時に、時間の経過とともに発展可能なシステムである必要があります。一般に、95% の最適なパフォーマンスを達成するようにシステムを構成し、構成の簡略化および最終的なアップグレードのために 5% のパフォーマンスを放棄します。

I/O の基本構成

この項では、収集する基本情報およびシステムの I/O 構成を定義するときの決定事項について説明します。必要な可用性、リカバリ能力およびパフォーマンスは維持しながら、できるだけ単純な構成を維持します。構成が複雑になるに従って、管理、メンテナンスおよびチューニングが困難になります。

アプリケーション I/O 特性の決定

この項では、決定する必要がある基本的な I/O 特性および要件について説明します。I/O 特性は、必要な技術のタイプおよびその技術の構成方法の決定に影響を与えます。I/O の要件には、サイト固有のパフォーマンス、領域およびリカバリ能力が含まれています。

効率的な I/O サブ・システムを設計するには、次の情報が必要です。

- I/O 率 : 読込み率および書込み率
- I/O 並行性
- I/O サイズ

- 可用性
- 記憶域サイズ

システムの I/O 構成を定義する場合は、パラレル索引作成やパラレル・スキャンなどのパラレル操作によって生成される最大負荷を考慮する必要があります。ディスク構成の機会は 1 回であるため、十分な負荷分散が組み込まれない場合はパラレル操作を測定できません。このことは、ログ・ディスクおよびアーカイブ・ディスクにとって特に重要です。

I/O 率 : 読込み率および書込み率

読込み率とは、1 秒当たりの読込み回数です。書込み率とは、1 秒当たりの書込み回数です。読込み率と書込み率の合計が I/O 率（1 秒当たりの I/O 操作回数）です。パフォーマンスのよいシステムの場合、アプリケーションの I/O 率は必要なディスクおよびコントローラの絶対最小数を決定するときの重要な要因です。

書込み率が高いシステムは、次の構成オプションからも利点が得られます。

- RAW デバイスまたはディスクへのダイレクト読込みが可能なサード・パーティ・ソフトウェアを使用して、UNIX バッファ・キャッシュへの読み書きを回避する。
- キャッシュを一杯にすることなく使用済ブロックのディステージを維持するのに十分な大きさのメモリー・キャッシュを持つ I/O ディスク・アレイを使用する。
- 書込み中心のアプリケーションでは RAID 5 構成を避ける。

> **関連項目：** パフォーマンスを損なわずに、高い書込み率を維持できるかどうかは、ベンダー固有のマニュアルを参照してください。

I/O 並行性

I/O 並行性は、I/O サブ・システムへの I/O リクエストを同時に行う個別のプロセスの個数を測定します。Oracle にとっての I/O 並行性は、I/O リクエストを同時に発行しているプロセス数とみなされます。I/O 並行性が高いということは、I/O リクエストを同時に発行している個別のプロセスが多数あることを意味します。並行性が低いということは、I/O リクエストを同時に発行しているプロセスが少ないことを意味します。

注意： I/O 率の高いシステムは、高い並行性と小さなブロック要求が特徴です。データ・レートの高いシステムは、並行性の低さと大きなブロック要求が特徴です。

I/O サイズ

I/O サイズは、Oracle から見た I/O リクエストのサイズです。最小 I/O サイズはオペレーティング・システムのブロック・サイズであり、最大値は一般的に Oracle ブロック・サイズにマルチブロック読込みカウントを乗じた因数となります。

I/O サイズは操作のタイプによって異なる可能性がありますが、アプリケーションの性質に応じていくつかの合理的で一般的な見積りを行うことができます。

- DSS システムでは、I/O の大部分がほぼ *n* * DB_BLOCK_SIZE で大きくなるのが普通です。
- OLTP システムでは、I/O が DB_BLOCK_SIZE のサイズになる場合がほとんどです。

I/O サイズおよび並行性に影響を与える要因 Oracle システムでは、各種ファイルにディスク・サブシステムの様々な要件があります。これらの要件は、データの読込みおよび書込み率とこれらの操作の並行性によって区別されます。並行性が低いときは、速い速度がディスク・サブシステムによって比較的容易に維持されます。ディスク・サブシステムの設計では、次の要因を考慮することが重要です。

- 操作パラメータ。これらは、Oracle、オペレーティング・システム、ハードウェアおよび論理ボリューム・マネージャ（LVM）などのサード・パーティ・ソフトウェアの構成可能および変更不能な操作パラメータです。

  関連項目： 15-13 ページの表 15-2

- 各ファイルの一般的な I/O サイズおよび並行性の程度。これらは同一タイプのファイルであっても、サイトごとに大きく異なります。たとえば、多数の同時ディスク・ソートを実行するサイトは、主に小規模のメモリー内ソートを実行するサイトとは異なる TEMP 表領域の特性を持っています。

  表 15-1 は、理論的なインストレーションでの読込み率、書込み率および並行性を示したものです。この例では、主としてメモリー内ソートを実行し、索引がバッファ・キャッシュにキャッシュされたままになる、問合せと更新の多い OLTP アプリケーションについて説明しています。

表 15-1 読込み率、書込み率および並行性の程度

コンポーネント	読込み率	書込み率	並行性
アーカイブ・ログ	高	高	低
REDO ログ	高	高	低
UNDO セグメント表領域 :	低	高	高
TEMP 表領域	低	低	高
索引表領域	低	中	高

表 15-1 読込み率、書込み率および並行性の程度（続き）

コンポーネント	読込み率	書込み率	並行性
データ表領域	高	中	高
アプリケーション・ログ・ファイルおよび出力ファイル	低	中	高
バイナリ（共有）	低	低	高

可用性

サイト固有の可用性要件では、ディスク記憶域の技術およびレイアウトに追加の仕様を必要とします。一般的な考慮事項には、REDO ログ、アーカイブログおよび制御ファイルのミラー化など、リカバリ能力要件および Oracle 固有の安全対策に適合する RAID 技術が含まれます。

専用ディスクを REDO ログ・ファイルのミラー化専用にすることは、重要な安全対策です。この対策によって、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。

> **関連項目：** リカバリ能力の詳細は、『Oracle9*i* ユーザー管理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。

記憶域サイズ

現在使用可能な大型ディスクを使用している場合、パフォーマンスおよび可用性の要件が満たされていれば、記憶域の要求はすでに満たされていることがほとんどです。ただし、システムが大量のデータをオンラインで格納する場合、データについては高い並行性やスループットの要件はありませんが、記憶域要件が不足する場合があります。そのような場合に記憶領域を最大にする次の方法を考慮する必要があります。

- 別の RAID 構成を使用します。
- ディスクの数を増やします。
- ディスクのサイズを増やします。

I/O 構成の決定

I/O レイアウト・フェーズでは、使用可能なハードウェアおよびソフトウェアに応じて、次のシステム特性を調べる必要があります。

- データベース・ファイルに使用できるディスク数
- RAID レベル
- ストライプ深度およびストライプ幅
- RAW デバイスとファイル・システムのどちらを使用するか
- システムが非同期 I/O を実行できるかどうか

パフォーマンスを最大にするためのシステム構成の目標は、使用可能なディスク上でできるだけ均等にデータベースの I/O 負荷を分散することです。

I/O の分散以外の考慮事項には、予想されるシステムの拡張および必要なリカバリ能力が含まれます。

I/O システム

I/O システムの機能を認識する必要があります。ハードウェアおよびソフトウェアのマニュアルを検討したり（必要に応じて）、サイトでテストを行うことで、この情報を得られます。テストには、次の要因を変更して含めることができます。

- I/O サイズ
- ファイル・システムと RAW デバイスの比較
- 読込みおよび書込みパフォーマンス
- 順次アクセスおよびランダム・アクセス
- 非同期 I/O と同期 I/O の比較
- I/O 向上のためのシステム機能の構成

通常このようなタイプの調査を行うと、最終設計に使用できる潜在的な構成も検討できます。

I/O 要件と I/O システムの適合

アプリケーションにより I/O システムにかかる負荷をシミュレートするベンチマークがある場合、または既存の本番システムがある場合、Oracle 統計を参照し、ファイル当たりの I/O 率およびデータベース全体の I/O 率を判断してください。

> **関連項目：** オペレーティング・システムの I/O 統計の詳細は、『Oracle9*i* データベース・パフォーマンス・プランニング』を参照してください。

Oracle ファイルの I/O 統計を判断するには、次のことを調べます。

- 物理読込み数（V$FILESTAT.PHYRDS）
- 物理書込み数（V$FILESTAT.PHYWRTS）
- 平均読込み時間
- 物理 I/O 数。この数値は物理読込みと物理書込みの合計です。

Oracle バッファ・キャッシュが適切にサイズ設定されていると仮定すると、物理読込みおよび物理書込みの統計が役に立ちます。

重要なもう一つの統計は、1 秒当たりの平均 I/O 数です。この比率を算出するには、ある間隔の V$FILESTAT データをサンプリングし、物理読込みおよび物理書込みを加算して、その合計を秒単位の経過時間で除算します。一般の計算式は次のようになります。

```
Average I/O per second = (physical reads + physical writes) / elapsed seconds
```

> **関連項目：** V$FILESTAT の差分情報の計算方法の詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

I/O 要件を見積るには、I/O 統計を新しいシステムの予測ワークロードと比べます。比べたデータをディスクの能力と比較すると、新しいアプリケーションの I/O 要件と I/O システムの機能との間に不整合があるかどうかを確認できます。

また、一般的な負荷の一部ではなく、I/O 率が平均よりはるかに高い I/O 中心の操作を識別します。システムがこれらの率を維持できるかどうかを確認してください。索引作成、データのロード、バッチ処理などの操作は、このカテゴリに該当します。

> **関連項目：** V$FILESTAT の差分情報の計算方法の詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

FILESYSTEMIO_OPTIONS

Oracle9*i* リリース 2（9.2）では、FILESYSTEMIO_OPTIONS 初期化パラメータを使用して、ファイル・システムのファイルについて非同期 I/O あるいはダイレクト I/O を有効化することも無効化することもできます。このパラメータは、プラットフォーム固有であり、それぞれのプラットフォームに最適なデフォルト値が設定されています。このパラメータは、動的に変更して、デフォルト設定を更新できます。

関連項目： 16-4 ページ「FILESYSTEMIO_OPTIONS 初期化パラメータ」

オペレーティング・システムまたはハードウェアのストライプ化を使用したファイルのレイアウト

オペレーティング・システムに LVM ソフトウェアまたはハードウェア・ベースのストライプ化がある場合、これらのツールを使用して I/O を分散できます。LVM またはハードウェア・ストライプ化を使用するときの決定事項には、**ストライプ深度**および**ストライプ幅**が含まれます。

- ストライプ深度は、ストライプのサイズで、ストライプ単位とも呼ばれます。
- ストライプ幅は、ストライプ深度とストライプ・セットを構成するドライブ数の積です。

これらの値を適切に選択して、システムが必要なスループットを維持できるようにします。Oracle データベースにおける適切なストライプ深度は、256KB ～ 4MB です。ストライプ深度によって、得られるアプリケーションの利点の種類が異なります。最適なストライプ深度およびストライプ幅は、次の項目により異なります。

- 要求された I/O サイズ
- I/O リクエストの並行性
- 物理ストライプ境界とブロック・サイズ境界との位置合せ
- 可変転送レート
- 提案されたシステムの管理可能性

要求された I/O サイズ

表 15-2 に、I/O サイズの設定で使用できる Oracle およびオペレーティング・システムのパラメータを示します。

表 15-2 Oracle およびオペレーティング・システム操作パラメータ

パラメータ	説明
DB_BLOCK_SIZE	単一ブロック I/O リクエストのサイズ。また、このパラメータをマルチブロック・パラメータと組み合せて使用して、マルチブロック I/O リクエスト・サイズを決定します。
OS ブロック・サイズ	REDO ログおよびアーカイブ・ログ操作の I/O サイズを決定します。
最大 OS I/O サイズ	単一 I/O リクエストのサイズに上限を設けます。
DB_FILE_MULTIBLOCK_READ_COUNT	全表スキャンの最大 I/O サイズは、このパラメータに DB_BLOCK_SIZE を乗算して計算されます（上限値は、オペレーティング・システムの制限を受けます）。
SORT_AREA_SIZE	ソート操作のための I/O サイズおよび並行性を決定します。
HASH_AREA_SIZE	ハッシュ操作のための I/O サイズを決定します。

I/O サイズの他に、並行性の程度も理想的なストライプ深度を調べる上で役立ちます。ストライプ幅およびストライプ深度を選択する場合は、次の点を考慮してください。

- 並行性の低い（順次）システムでは、単一 I/O が同じディスクに 2 回アクセスしないようにします。たとえば、ストライプ幅が 4 つのディスクであり、ストライプ深度が 32k であると仮定します。1MB の単一 I/O リクエスト（たとえば、全表スキャンの場合）が Oracle サーバー・プロセスで発行された場合、ストライプ内の各ディスクは要求されたデータを戻すために 8 回 I/O を実行する必要があります。このような状況を回避するために、平均 I/O のサイズは、ストライプ深度とストライプ幅の積より小さいサイズにしてください。そうでない場合は、オペレーティング・システムに対して Oracle が単一 I/O リクエストを行うと、同じディスクに対して複数の物理 I/O リクエストが発生します。
- 並行性が高い（ランダム）システムでは、単一 I/O リクエストが複数の物理 I/O コールに分解されないようにしてください。そうでなければ、システムで実行される物理 I/O リクエスト数が何倍にもなり、I/O 応答時間が大幅に下がります。

I/O リクエストの並行性

従来の OLTP 環境などの高度な小さい同時 I/O リクエストが存在するシステムでは、ストライプ深度を大きく保つことが有効です。I/O サイズより大きいストライプ深度を使用することは粗密なストライプ化と呼ばれます。並行性の高いシステムのストライプ深度は次のようになります。

```
n * DB_BLOCK_SIZE
```

n は 1 より大

粗密なストライプ化ではアレイ内の 1 ディスクで複数の I/O リクエストに対応できます。この方法では、一連のストライプ化ディスクで多数の同時 I/O リクエストに対応でき、I/O セットアップ・コストも最小限で済みます。粗密なストライプ化では全体的な I/O スループットが最大化されます。全表スキャンの場合も同様に、ストライプ深度が大きく、かつ 1 つのドライブで対応可能な場合は、マルチブロック読込みが有益です。DSS 環境のパラレル問合せも、粗密なストライプ化の候補です。これは、それぞれ個別の I/O を発行する個々のプロセスが多数存在するためです。粗密なストライプ化は、同時要求が少ないシステムで使用されると、ホット・スポットが生成される可能性があります。

従来の DSS 環境や並行性の低い OLTP システムなどの大きい I/O リクエストがほとんど存在しないシステムでは、ストライプ深度を小さく保つことが有益です。これはファイングレイン・ストライプ化と呼ばれます。そのようなシステムのストライプ深度は次のように表されます。

```
n * DB_BLOCK_SIZE
```

n はマルチブロック読込みパラメータ（DB_FILE_MULTIBLOCK_READ_COUNT など）よりも小さくなります。

ファイングレイン・ストライプ化では、複数のディスクで単一 I/O リクエストを処理できます。ファイングレイン・ストライプ化では、個々の I/O リクエストまたは応答時間のパフォーマンスが最大化されます。

物理ストライプ境界とブロック・サイズ境界との位置合せ

一部の Oracle ポートでは、Oracle ブロック境界がストライプと揃わない可能性があります。ストライプ深度が Oracle ブロックのサイズと同じである場合、Oracle から発行された単一 I/O によって 2 つの物理 I/O 操作が発生する場合があります。

これは、OLTP 環境では最適ではありません。1 つの論理 I/O で物理 I/O が 1 つのみ発生する確率が高くなるようにするには、最小のストライプ深度が Oracle ブロック・サイズの少なくとも 2 倍である必要があります。表 15-3 に、ランダム・アクセスおよび順次読込みでお薦めする最小ストライプ深度を示します。

表 15-3 最小ストライプ深度

ディスク・アクセス	最小ストライプ深度
ランダム読込みおよび書込み	最小ストライプ深度は、Oracle ブロック・サイズの 2 倍です。
順次読込み	最小ストライプ深度は、Oracle ブロック・サイズを乗算した DB_FILE_MULTIBLOCK_READ_COUNT の値の 2 倍です。

関連項目： プラットフォームの固有のマニュアル

可変転送レート

ディスク・ドライブの転送レートはディスクのすべての部分で同一であるわけではありません。ディスク・ヘッドは、ディスク・ドライブの内側のセクターより外側のセクターでの動きが速いため、外側のセクターの転送レートの方が速くなります。

ディスク・ドライブの外側の部分には、内側の部分よりも多くのデータが格納されます。

ディスク・ドライブの転送レートは、ドライブの最も内側と最も外側で、2 のべき乗の違いがあります。したがって、頻繁にアクセスするデータは、ディスク・ドライブの最も外側に近い方に配置するのが有効です。

提案されたシステムの管理可能性

LVM の場合、最も管理の簡単な構成は、使用可能なすべてのディスク上に単一ストライプ化ボリュームを構成したものです。この場合、ストライプ幅はすべての使用可能なディスクを包含します。すべてのデータベース・ファイルはそのボリューム内に常駐し、効果的に負荷を均等分散します。この単一ボリューム・レイアウトは、ほとんどの状況で適切なパフォーマンスを実現します。

単一ボリューム構成は、RAID 1 などの簡単なリカバリを可能にする RAID 技術と併用する場合のみ有効です。それ以外の場合、単一のディスクを失うことはすべてのファイルを同時に失うことであり、完全なデータベースのリストアおよびリカバリを実行する必要があることを意味します。

パフォーマンスの他に、管理可能性の問題があります。システムの設計では、ディスクを簡単に追加できるようにして、データベースの拡張を可能にする必要があります。課題は、負荷のバランスを維持しながら拡張を行うことです。

たとえば、初期構成で、64 個の 16GB のディスク上に単一ストライプ化ボリュームを作成する必要があるとします。これはプライマリ・データの 1 テラバイト（TB）の合計ディスク領域になります。場合によっては、システムが動作した後に、将来のデータベース拡張を可能にするためにさらに 80GB（すなわち、5 つのディスク）を追加する必要があります。

この領域をデータベースで使用できるようにするオプションには、5 つの新しいディスクを含む第 2 のボリュームの作成があります。ただし、これらの新しいディスクがその上に配置されたファイルに必要な I/O スループットを保持できない場合、I/O ボトルネックが発生する可能性があります。

もう 1 つのオプションは、元のボリュームのサイズを増やすことです。LVM は高度になりつつあり、ストライプ幅の動的な再構成を可能にするので、システムがオンライン中にディスクを追加できます。このため、本番環境で単一ストライプ化ボリューム上にすべてのファイルを配置できるようになってきました。

LVM がストライプへのディスクの動的な追加をサポートできない場合は、より小さく管理しやすいストライプ幅を選択する必要があります。そのようにすると、新しいディスクを追加する場合、ストライプ幅だけシステムを拡張できます。

前述の例で、8 個のディスクがより管理しやすいストライプ幅といえます。これは、8 個のディスクで 1 秒間に必要な数の I/O を維持できる場合のみ可能です。したがって、追加のディスク領域が必要なときは、別の 8 ディスク・ストライプを追加して、ボリューム間で I/O のバランスを維持することができます。

注意： ストライプ幅が小さくなるほど、ボリューム上にファイルを分散する時間の必要性が高くなり、このプロシージャは手動による I/O の分散に近づきます。

手動による I/O の分散

システムに LVM またはハードウェアのストライプ化がない場合、各ファイルの I/O 要件に従ってファイルを分散することにより、使用可能なディスク間で I/O を手動でバランス化する必要があります。ファイルの配置に関する決定を行うには、データベース・ファイルの I/O 要件および I/O システムの機能についてよく理解している必要があります。このようなデータに慣れておらず、解析対象の代表的なワークロードを取得できない場合は、まず推定を行い、次に使用量がわかったときにこのレイアウトをチューニングします。

ディスクを手動でストライプ化するには、ファイルの記憶域要件を I/O 要件と関連付ける必要があります。

1. ファイルおよびディスクのサイズをチェックして、データベースのディスク記憶域要件を評価します。
2. 1 ファイル当たりの予測 I/O スループットを識別します。最高の I/O 率を持つファイルおよび多数の I/O を持たないファイルを判断します。I/O 率を均等にするために、すべての使用可能なディスク上にファイルをレイアウトします。

手動 I/O 分散の一般的な方法として、頻繁に使用される表をその索引から分離することが挙げられます。これは正しくありません。一連のトランザクション中は、索引が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。データ・ファイルには索引や表データが含まれているため、これを単純に分離するだけでは十分ではありません。ファイルを分離する決定は、そのファイルの I/O 率がデータベースのパフォーマンスに影響を与える場合にのみ行ってください。

ファイルを分割する場合

オペレーティング・システムのストライプ化または手動 I/O 分散を使用するかどうかに関係なく、I/O システムまたは I/O レイアウトが要求された I/O 率をサポートできない場合は、I/O 率の高いファイルをそれ以外のファイルから分離する必要があります。このようなファイルは計画段階かシステムの本稼働後に確認できます。

ファイルを分離する決定は、I/O 率、リカバリ能力の問題、管理可能性の問題によってのみ影響を受けます（たとえば、LVM がストライプ幅の動的な再構成をサポートしない場合、同一構成の新しいストライプを作成して *n* 個のディスクを追加できるように、さらに小さいストライプ幅を作成する必要がある場合があります）。

ファイルを分離する前に、ボトルネックが実際に I/O の問題であるかどうかを検証します。ボトルネックの調査から生成されたデータでは、最高の I/O 率を持つファイルを識別します。

関連項目： 6-3 ページ「多くのリソースを消費する SQL の識別およびデータ収集」

表、索引および TEMP 表領域

I/O の多いファイルが表および索引を含む表領域に属するデータ・ファイルである場合は、これらのファイルの I/O を SQL のチューニングまたはアプリケーション・コードのいずれかで削減できるかどうかを識別します。

I/O の多いファイルが TEMP 表領域に属するデータ・ファイルである場合は、このアクティビティを回避するためにディスク・ソートを実行する SQL 文をチューニングするか、あるいはソートをチューニングするかを調べます。

不要な I/O を回避するようにアプリケーションをチューニングした後、I/O レイアウトが引き続き必要なスループットを維持できない場合は、I/O の多いファイルの分離を考慮してください。

関連項目： 6-3 ページ「多くのリソースを消費する SQL の識別およびデータ収集」

REDO ログ・ファイル

I/O の多いファイルが REDO ログ・ファイルである場合は、REDO ログ・ファイルをその他のファイルから分離することを考慮してください。可能な構成には、次のことが含まれています。

- すべての REDO ログを、他のファイルのない 1 つのディスクに置きます。可用性も考慮します。すなわち、リカバリ能力の目的で、同じグループのメンバーは異なる物理ディスクおよびコントローラ上にあることが必要です。
- 他のファイルを格納しない個別のディスク上に各 REDO ログ・グループを置きます。

- 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、複数のディスクにまたがって REDO ログ・ファイルをストライプ化します（この場合、手動ストライプ化は不可能です）。
- REDO ログに RAID 5 を使用しないでください。

REDO ログ・ファイルは、ログ・ライター（LGWR）プロセスで順次書き込まれます。同じディスクに対する同時実行アクティビティが存在しない場合、この操作はさらに高速にできます。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。システムが非同期 I/O をサポートせず、この機能が現在構成されていない場合、この機能を使用することが有効かどうかを確認します。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。

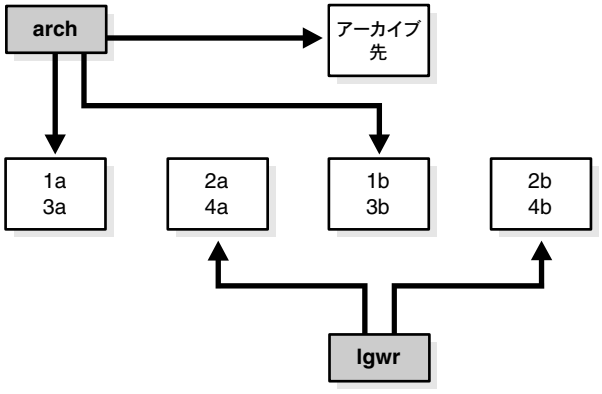
アーカイブ REDO ログ

アーカイバが遅い場合、アーカイバ読込みと LGWR 書込みが分離されるようにして、アーカイバ・プロセスと LGWR の間の I/O 競合を防止することが賢明です。これは、ログを代替ドライブに置くことで達成されます。

たとえば、システムに 4 つの REDO ログ・グループが存在し、各グループが 2 つのメンバーを持っているとします。個別ディスク・アクセスを作成するには、8 つのログ・ファイルにそれぞれ 1a、1b、2a、2b、3a、3b、4a、4b のラベルを付けてください。この場合、最低でも 4 つのディスクと、アーカイブ・ファイル用に 1 つのディスクが必要です。

図 15-1 は、競合を最小限にするために、ディスク間で REDO メンバーを分散する方法を示しています。

図 15-1 ディスク間での REDO メンバーの分散



この例では、LGWR はログ・グループ 1（メンバー 1a と 1b）から切り替えられ、ログ・グループ 2（2a と 2b）に書込みを行います。同時に、アーカイバ・プロセスはグループ 1 から読込みをして、アーカイブ先に書込みを行います。REDO ログ・ファイルがどのようにして競合から分離されているかに注意してください。

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。したがって、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。

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一般にヘッドの移動はわずかです。このため、ログ書込みのスピードが大幅に向上します。

3 つの構成サンプル

この項では、I/O システムを構成する高水準の例を 3 つ示します。これらの例には、ディスク・トポロジやストライプ深度などを定義する計算例が含まれています。

すべてのディスクにまたがったすべての内容のストライプ化

I/O 構成の最も簡単なアプローチは、すべての使用可能ディスクにまたがってストライプ化された、1 つの大きなボリュームを作成することです。リカバリ能力を考慮して、ボリュームがミラー化されます（RAID 1）。ディスク当たりのストライプ化の単位は、頻繁な I/O 操作のための最大 I/O サイズより大きくする必要があります。そうすると、ほとんどの場合は十分なパフォーマンスが得られます。

異なるディスクへのアーカイブ・ログの移動

アーカイブ・ログが他のファイルと同じディスク・セット上でストライプ化されている場合、REDO ログがアーカイブされる際、これらのディスク上の I/O リクエストが影響を受けるおそれがあります。個別のディスクにアーカイブ・ログを移動する場合の利点は次のとおりです。

- 非常に高速なアーカイブを実行できます（順次 I/O を使用）。
- アーカイブ先のディスク上で応答時間が低下しても、その影響を受けるものは他にありません。

アーカイブ・ログ用のディスク数は、アーカイブ・ログ生成頻度およびアーカイブ記憶域の必要量により決定されます。

個別のディスクへの REDO ログの移動

更新の多い OLTP システムでは、REDO ログは書込み中心です。他のディスクおよびアーカイブ REDO ログ・ファイルから分離されたディスクに REDO ログ・ファイルを移動すると、次の利点があります。

- REDO ログの書込みは、可能なかぎり高速で行われます。したがって、トランザクション処理のパフォーマンスは最高になります。
- REDO ログの書込みが他の I/O で損なわれることはありません。

REDO ログ用のディスク数は、ほとんどの場合に、現行技術のディスク・サイズと比較して一般に小さい REDO ログ・サイズで決定されます。一般に、2 つのディスクを持つ構成（フォルト・トレランスのために 4 つのディスクにミラー化など）で十分です。特に、2 つのディスク上で REDO ログ・ファイルを交互に使用すると、1 つのファイルに REDO ログ情報を書き込む場合、アーカイブの完了した REDO ログの読込みが妨げられません。

Oracle Managed Files

ファイル・システムを使用してすべての Oracle データを取り込むシステムの場合、データベース管理は Oracle Managed Files を使用して簡単に行えます。表領域、一時ファイル、オンライン・ログおよび制御ファイルについては、Oracle は内部的に標準ファイル・システム・インタフェースを使用して、必要に応じてファイルを作成および削除します。管理者は、特定のタイプのファイルに使用するファイル・システム・ディレクトリのみを指定します。データ・ファイルについてはデフォルトの場所を 1 つ、制御ファイルおよびオンライン REDO ログ・ファイルについては複数の場所を最大 5 つ指定できます。

Oracle では、一意のファイルが作成された後、そのファイルが不要になると削除されます。このため、管理者が誤ったファイルを指定したことにより発生する破損、および廃止されたファイルで消費される無駄なディスク領域が減り、テスト・データベースおよび開発データベースの作成が簡素化されます。また、オペレーティング・システム固有のファイル名を SQL スクリプトに設定する必要がないため、ポータブルなサード・パーティのツールの開発を容易にします。

新しいファイルは管理ファイルとして作成できますが、古いファイルは古い方法で管理されます。したがって、データベースには Oracle Managed Files と手動管理ファイルの両方を配置できます。

注意： Oracle Managed Files は、RAW デバイスでは使用できません。

Oracle Managed Files のチューニング

Oracle Managed Files をチューニングする場合はいくつかの点に注意する必要があります。

- Oracle Managed Files ではファイル・システムを使用する必要があるため、DBA はデータのレイアウト方法は管理しません。したがって、ファイル・システムを正しく構成することが重要です。
- Oracle Managed Files システムは、ストライプ化をサポートする LVM 上に構築する必要があります。ロード・バランシングおよびスループットの向上のために、Oracle Managed Files システム内のディスクをストライプ化する必要があります。
- Oracle Managed Files は、動的に拡張可能な論理ボリュームをサポートする LVM 上で使用するときに最高の機能を発揮します。それ以外の場合は、論理ボリュームをできるだけ大きく構成する必要があります。
- Oracle Managed Files は、ファイル・システムに大きな拡張可能なファイルがある場合、最高の機能を発揮します。

> **関連項目：** Oracle Managed Files の使用方法の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

データ・ブロック・サイズの選択

この項では、最適なパフォーマンスを得るためにデータベース・ブロック・サイズを選択するときの考慮事項を示します。

読込み データのサイズとは関係なく、目標は必要なデータを取り出すために必要な読込み回数を最小にすることです。

- 行が小さく、アクセスがきわめてランダムな場合は、小さなブロック・サイズを選択します。
- 行が小さく、アクセスがきわめて順次である場合は、大きなブロック・サイズを選択します。
- 行が小さく、アクセスがランダムかつ順次である場合は、大きなブロック・サイズを選択するのが有効です。
- 行が大きい（たとえば、ラージ・オブジェクト（LOB）データが含まれている）場合は、大きなブロック・サイズを選択します。

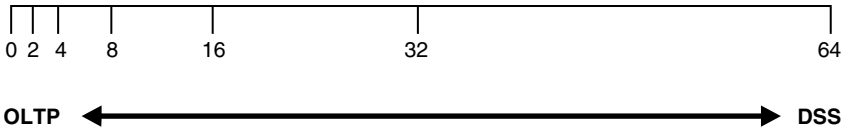
書込み 並行性の高い OLTP システムで、大きなブロック・サイズを使用する場合は、`INITRANS`、`MAXTRANS` および `FREELISTS` の適切な値について検討します。これらのパラメータは、ブロック内で許可されている更新の並行性の程度に影響を与えます。ただし、自動セグメント領域管理を使用する場合は、`FREELISTS` の値を指定する必要はありません。

選択すべきブロック・サイズが不明な場合、多数のトランザクションを処理するシステムでは、8KB のデータベース・ブロック・サイズを試行してください。これは優れた妥協案であ

り、通常は有効です。8KB を超えるサイズが必要なのは LOB データを処理するシステムのみです。

図 15-2 に、OLTP アプリケーションまたは DSS アプリケーションに適した様々なブロック・サイズを示します。

図 15-2 ブロック・サイズ（KB 単位）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。

ブロック・サイズの長所と短所

表 15-4 は、様々なブロック・サイズの長所と短所を説明します。

表 15-4 ブロック・サイズの長所と短所

ブロック・サイズ	長所	短所
小さい	行数が少なく大量のランダム・アクセスに適しています。 ブロック競合が低減されます。	メタデータ（すなわち、ブロック・ヘッダー）による比較的大きな領域のオーバーヘッドがあります。 行が大きい場合にはお薦めできません。1 行がブロックに収まらない場合、1 ブロック当たりの格納行数がわずかになったり、さらには行の連鎖が発生する可能性があります。
大きい	オーバーヘッドが少ないため、データを格納する空間が多くなります。 1 回の I/O でバッファ・キャッシュに多数の行を読み込めます（行サイズおよびブロック・サイズにより異なります）。 順次アクセスまたは非常に大きな行（LOB データなど）に適しています。	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消費されます。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 バイトが消費されます。 OLTP 環境で使用される索引ブロックには適しません。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。

16

オペレーティング・システム・リソース

この章では、Oracle データベース・サーバー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。

この章には次の項があります。

- 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・システムの問題の解決
- CPU について
- システムの CPU 使用率の調査

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・システムの統計の重要性については、『Oracle9*i* データベース・パフォーマンス・プランニング』を参照してください。

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 buffer busy waits 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 buffer busy waits を削減すると、システム・コールの数が減少します。

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・システムのキャッシュの使用

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にならないですし、これらの構造ではリソースが消費されます。このことは、UNIX ファイル・システムにデータベース・ファイルがある UNIX システムで最も顕著です。デフォルトでは、すべてのデータベース I/O はファイル・システム・キャッシュ内を通過します。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・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。

この問題は NT では発生しません。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。

Oracle バッファ・キャッシュのバッファ・ブロックの存在により、オペレーティング・システムのキャッシュは冗長になる場合がありますが、Oracle が Oracle バッファ・キャッシュを使用しないこともよくあります。このような場合に、UNIX またはオペレーティング・システムのキャッシュがバイパスされる直接 I/O、またはオペレーティング・システムのキャッシュが使用されない RAW デバイスを使用すると、オペレーティング・システムのバッファリングを使用したときより、パフォーマンスが劣化することがあります。その場合の例をいくつか示します。

- 一時表領域での読込みまたは書込み
- NOCACHE LOB に格納されるデータ
- パラレル問合せスレーブで読み込まれるデータ

オペレーティング・システム・レベルでキャッシュするファイルと、キャッシュしないファイルを混在させる必要がある場合があります。

ハードウェア・キャッシュ

一部の基礎となる I/O サブシステムでは、ディスク読込みおよび書込みのハードウェア・レベルのキャッシュを実装して、I/O リクエストに対する応答時間を短縮しています。そのようなサブシステムは、書き込まれたデータが安全であることが保証された後でのみ、書込み要求を承認するよう構成することが重要です。

たとえば、サブシステムが、RAM キャッシュへのデータ書込み直後に書込みを承認する RAM キャッシュを実装したと仮定します。Oracle データベース・サーバーはそのデータを安全であると判断します。電源障害が発生した場合、そのデータは失われます。その場合、Oracle サーバーは書込みが失われたかどうかを判断する手段を持たないため、破損につながる可能性があります。この問題を解消するため、ほとんどの I/O サブシステムは、RAM キャッシュのデータが電源障害によって失われないようにするためのメカニズムを備えています。これを保証できないサブシステムは、データが単に RAM キャッシュに書き込まれた後ではなく、ディスクに書き込まれた後でのみ、書込みを承認するよう構成するのが一般的にはベストです。

非同期 I/O

同期 I/O では、I/O リクエストがオペレーティング・システムに発行されても、書込みの完了が確認されない限り書込みプロセスによってブロックされます。処理はその後で継続できます。

非同期 I/O では、プロセスは I/O リクエストを発行し、さらに処理を継続できます。I/O の結果のチェックは後からでも可能です。複数の I/O リクエストを発行し、それらの要求のステータスを後から収集することもできます。これにより、オペレーティング・システムは、任意の I/O 操作を必要な時点でパラレル化できます。パラレル処理を行うと、操作完了までの総時間を短縮できます。

極端な例で、4 つのデータ・ブロックを 4 種類のファイルに書き出す必要があると仮定します。同期 I/O では、ブロック 1 の発行、待機、ブロック 2 の発行、待機、ブロック 3 の発行、待機、ブロック 4 の発行、待機が必要です。非同期 I/O では、ブロック 1、2、3 および 4 を発行し、4 つのブロックすべてが完了するのを待機することができます。オペレーティング・システムは、4 つの I/O リクエストを一度に受け取るので、パラレルですべての要求を処理できます。応答時間の合計は、4 つの I/O 時間の合計ではなく、4 つのうちの最も長い I/O の時間となります。

プラットフォームには、デフォルトで非同期 I/O をサポートしているもの、特殊な構成が必要なもの、基礎となる一定のファイル・システム・タイプでのみ非同期 I/O をサポートしているものがあります。

FILESYSTEMIO_OPTIONS 初期化パラメータ

Oracle9*i* リリース 2（9.2）では、FILESYSTEMIO_OPTIONS 初期化パラメータを使用して、ファイル・システムのファイルについて非同期 I/O あるいはダイレクト I/O を有効化することも無効化することもできます。このパラメータは、プラットフォーム固有であり、それぞれのプラットフォームに最適なデフォルト値が設定されています。このパラメータは、動的に変更して、デフォルト設定を更新できます。

FILESYTEMIO_OPTIONS は、次のいずれかの値に設定できます。

- ASYNCH: ファイル・システム・ファイル上の非同期 I/O を有効化します。
- DIRECTIO: ファイル・システム・ファイル上の直接 I/O を有効化します。
- SETALL: ファイル・システム・ファイル上の非同期および直接 I/O を有効化します。
- NONE: ファイル・システム・ファイル上の非同期および直接 I/O を無効化します。

> **関連項目：** 詳細は使用しているプラットフォーム固有のマニュアルを参照してください。

メモリー使用量

メモリー使用量は、バッファ・キャッシュの制限と初期化パラメータの両方の影響を受けます。

バッファ・キャッシュの制限

UNIX バッファ・キャッシュはオペレーティング・システムのメモリー・リソースを消費します。あるバージョンの UNIX では、UNIX バッファ・キャッシュに一定量のメモリーを割り当てることができますが、現在ではより複雑なメモリー管理メカニズムが使用されるのが普通です。通常これらの管理メカニズムでは、I/O をキャッシュする際に空きメモリー・ページを使用できます。そのようなシステムでは、オペレーティング・システムのレポート・ツールは空きメモリーがないことを示すのが普通で、通常問題にはなりません。プロセスがより多くのメモリーを必要とする場合、通常は I/O データをキャッシュするメモリーが開放されて、そのプロセス・メモリーを割り当てることができます。

メモリー使用量に影響を与えるパラメータ

Oracle セッションから要求されるメモリーは多数の要因に依存します。一般的に、大きな要因には次のものがあります。

- オープン・カーソルの数
- PL/SQL で使用されるメモリー（PL/SQL 表など）
- SORT_AREA_SIZE 初期化パラメータ

Oracle9*i* では、PGA_AGGREGATE_TARGET 初期化パラメータを使用することにより、セッションのメモリー使用量をより大きく制御することができます。

プロセス・スケジューラの使用

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可能性があります。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、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。

また、Oracleのバックグラウンド・プロセスをCPUにバインドしないでください。これは、バインドされたプロセスでCPU不足を発生する場合があります。これは、特にバインド元プロセスがオペレーティング・システムのスレッドを生成する場合に発生します。この場合、親プロセスとそのスレッドすべてがCPUにバインドされます。

オペレーティング・システムのリソース・マネージャの使用

一部のプラットフォームではオペレーティング・システム・リソース・マネージャが提供されます。これらは、CPUのリソース割当てに優先順位を付けることによってピーク負荷使用パターンの影響を少なくするように設計されています。これらは通常、ユーザーがアクセス可能なリソースと各ユーザーがこれらのリソースのどの程度を消費可能かを統括する、管理方針を実装します。

オペレーティング・システム・リソース・マネージャは、ドメインや類似のファシリティとは異なります。ドメインは、1つのシステム内で1つ、あるいは複数のまったく別の環境を提供します。ディスク、CPU、メモリーおよびその他すべてのリソースがドメイン専用となっており、他のドメインからアクセスできません。他の類似のファシリティは、異なる領域、通常個別のCPUまたはメモリー領域へのシステム・リソースの一部として完全に分離されています。ドメインと同様、個別のリソース領域はその領域に割り当てられた処理専用となっています。プロセスは境界を超えて移行できません。ドメインとは異なり、その他すべてのリソース（通常はディスク）は、システムのすべてのパーティションからアクセスできます。

Oracleはドメイン内で実行される他、パーティション化されたメモリー（RAM）リソースの割当てが動的でなく、固定されている場合は、その他の不完全なパーティション構造体内で実行されます。

注意： Oracle は、メモリーが動的に割り当てられるリソース・パーティション環境ではサポートされません。

オペレーティング・システム・リソース・マネージャは、リソースのグローバル・プール内、通常はドメインあるいはシステム全体内でのリソース割当てに優先順位を設定します。プロセスはグループに割り当てられ、グループはリソース・プール内の任意の場所にあるリソースに割り当てられます。

注意： オペレーティング・システム・リソース・マネージャのメモリー管理および割当てファシリティと Oracle との併用はサポートされません。Oracle インスタンス内でリソース割当て機能を提供する Oracle Database Resource Manager は、オペレーティング・システムのリソース・マネージャと併用することはできません。

注意： オペレーティング・システム・リソース・マネージャで実行されている場合、Oracle は各インスタンスが専用オペレーティング・システム・リソース・マネージャ・グループあるいは管理エンティティに割り当てられている場合にのみサポートされます。また、すべてのインスタンスのプロセスを実行している専用エンティティは、1 つの優先順位（またはリソース使用）レベルで実行される必要があります。異なる優先順位レベルの個別の Oracle プロセスの管理は、サポートされません。インスタンスの破壊などの重大な結果が発生します。

関連項目：

- オペレーティング・システムのリソース・マネージャ、および Oracle と Oracle Database Resource Manager と併用して動作するリソース割当て / 割当て解除機能の全リストについては、システム・ベンダーおよび Oracle の代理店にお問い合せください。Oracle は特定リリース・レベルとこれらのシステム機能との互換性を保証しません。
- Oracle Database Resource Manager の詳細は、『Oracle9*i* データベース概要』および『Oracle9*i* データベース管理者ガイド』を参照してください。

オペレーティング・システムの問題の解決

この項では、様々なシステムでのチューニングのヒントを示します。次の項目について説明します。

- UNIX ベースのシステムのパフォーマンスに関するヒント
- NT システムのパフォーマンスに関するヒント
- ミッドレンジおよびメインフレーム・コンピュータのパフォーマンスに関するヒント

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。

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 ベースのシステムのパフォーマンスに関するヒント

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（ユーザー・モードと呼ばれる場合もあります）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、潜在する問題の徴候にすぎず、次の項目に関係している可能性があります。

- ページングまたはスワッピング
- 実行しているオペレーティング・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 システムのパフォーマンスに関するヒント

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してください。NT では、Performance Monitor で多数の因数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、これらの領域でのボトルネックを容易に回避できるように同じグラフ上に表示されます。

ミッドレンジおよびメインフレーム・コンピュータのパフォーマンスに関するヒント

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 または 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 サーバーの実行中は、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。

CPU 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断し、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 リソースの量を判断します。

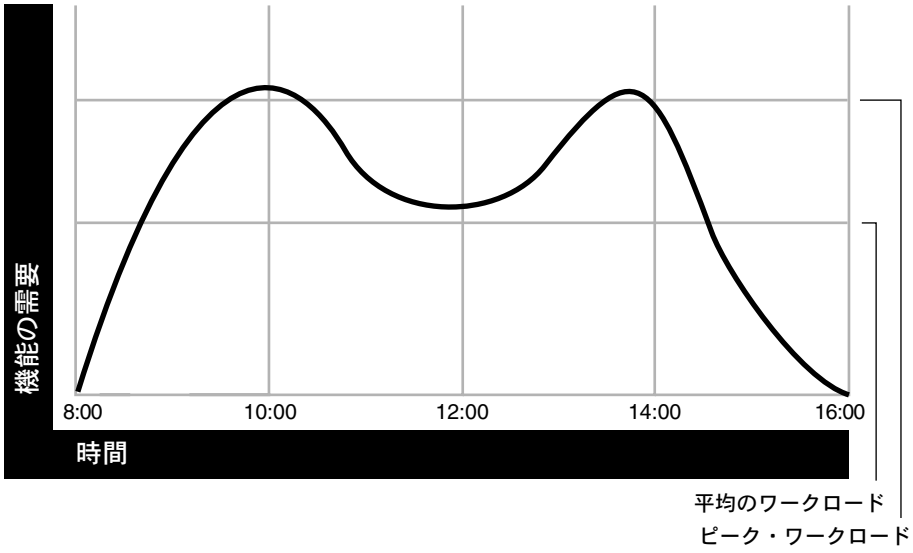
- システムがアイドル状態（Oracle のアクティビティがほとんど存在しないか、Oracle 以外のアクティビティが存在する）の場合
- システムが平均ワークロードの場合
- システムがピーク・ワークロードの場合

Statspack または `UTLBSTAT/UTLESTAT` ユーティリティを使用して、様々なワークロードのスナップショットを獲得できます。UNIX の `vmstat`、`sar` や `iostat` および NT の Performance Monitor などのオペレーティング・システム・ツールは、`UTLBSTAT/UTLESTAT` などと同じ間隔で実行し、統計全体の補完ビューを提供する必要があります。

注意： Statspack および `UTLBSTAT/UTLESTAT` の詳細は、第 21 章「Statspack の使用方法」を参照してください。

システムの CPU 使用率のレベルを評価するときには、ワークロードが重要な要因となります。ピーク・ワークロード時には、CPU 使用率が 90% の場合アイドルは 10% であり、待機時間が発生するのは受容できます。低ワークロード時に使用率が 30% となるのも理解できます。しかし、システムが標準のワークロードで高い使用率を示している場合は、ピーク・ワークロードに耐える余裕がありません。次に、図 16-1 において、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ワークロードの時間に伴う変化の例を示します。

図 16-1 平均のワークロードおよびピーク・ワークロード



この例のアプリケーションでは、1 日に 8 時間作業するユーザーが 100 人います。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 時間当たり 1,200 のトランザクションをサポートする必要があり、1 分当たり平均 20 トランザクションをサポートする必要があります。需要率が一定の場合は、この平均ワークロード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 分当たり 20 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。達成する必要があるピークの割合が 1 分当たり 120 トランザクションの場合は、このピーク・ワークロード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、ワークロードがピークのときに Oracle が CPU リソースの 90% を使用するとします。その場合、ワークロードが平均の期間では、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 tpm / 120 tpm * 90% = 15% of available CPU resource
```

ここで、tpm は 1 分当たりのトランザクション数を表します。

20 tpm 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、次のような問題があります。CPU リソースの 90% を使用しても 1 分当たり 120 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 20 tpm 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、（線状の拡張性を前提とすると）CPU リソースの 90% を使用して 1 分当たり 120 トランザクションを処理できます。

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、ワークロードがかつてのピーク・レベルにまで上昇する可能性があります。そのときには、実際には以前よりも高くなった新しいピークの割合に対応できる CPU 能力はありません。

CPU 能力の問題は、次の場合にも発生します。

- チューニング（過剰消費となっている CPU の問題を検出し、解決するプロセス）する場合。

  関連項目： 16-12 ページ「システムの CPU 使用率の調査」

- システム・アーキテクチャの変更などのハードウェア容量を増加する場合。

  関連項目： システム・アーキテクチャの向上については、『Oracle9*i* データベース・パフォーマンス・プランニング』を参照してください。

- CPU のリソース割当てに優先順位を付けることにより、ピーク負荷使用パターンの影響を少なくする場合。Oracle Database Resource Manager、データベース・ユーザーとアプリケーション間で CPU リソースを割当て、管理することによって、これを行います。

  関連項目： Oracle Database Resource Manager の詳細は、『Oracle9*i* データベース概要』および『Oracle9*i* データベース管理者ガイド』を参照してください。

コンテキストのスイッチング

Oracle では、この項で説明する複数のコンテキストのスイッチング機能があります。

ポスト・ウェイト・ドライバ

Oracle プロセスは、別の Oracle プロセスをポスト（メッセージの送信）することができ、さらにポストされるのを待機できる必要があります。

たとえば、フォアグラウンド・プロセスが LGWR をポストして、指定の時点までのブロックをすべて書き出してコミットを承認するよう、LGWR に通知する必要があります。

このポスト・ウェイト・メカニズムは、UNIX のセマフォを使用して実装するのが普通ですが、これらのセマフォがリソース集中型である場合があります。したがって、一部のプラットフォームでは、ポスト・ウェイト・ドライバを提供しています。通常は、ポスト・ウェイト・インタフェースを簡単に実装できるカーネル・デバイス・ドライバが提供されます。

メモリーマップ済みシステム・タイマー

Oracle では、システム時間を問い合せてタイミング情報を得る必要が生じる場合があります。その場合、オペレーティング・システム・コールが実行され、比較的コストのかかるコンテキストのスイッチングが発生することがあります。一部のプラットフォームでは、メモリーマップ済みタイマーを実装し、プロセス仮想アドレス空間のアドレスに現在のタイミング情報を収集します。このメモリーマップ済みタイマーから時間を読み込む方が、システム・コールのコンテキストのスイッチングのオーバーヘッドよりも経済的です。

1 回のコールで複数の非同期 I/O を発行するリスト I/O インタフェース

リスト I/O とは、1 回のシステム・コールで複数の非同期 I/O リクエストを発行できる Application Program Interface です。個別のシステム・コールで複数の I/O リクエストを発行する必要がありません。この機能の主な利点は、コンテキストのスイッチングの所要回数が減少することです。

システムの CPU 使用率の調査

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。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。

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ー、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。ワークロード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% 未満である）場合は、CPU の問題が存在します。

NT では、Performance Monitor 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 Performance Monitor は Performance Manager とは異なります。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 システムでの 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。

メモリー管理のチェック

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

UNIX の `sar` あるいは `vmstat` などのツール、または NT の Performance Monitor 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。

大きすぎるページ表

UNIX では、処理領域が大きくなりすぎると、ページ表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 管理のチェック

スラッシングは I/O 管理の課題です。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、ワークロード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・スライスを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。

関連項目： 第 15 章「I/O 構成および設計」

ネットワーク管理のチェック

クライアント / サーバーのラウンドトリップをチェックします。メッセージを処理する際にはオーバーヘッドが発生します。アプリケーションで多数のメッセージを生成し、ネットワークを介して送信する必要がある場合、メッセージ送信の待機時間によって CPU のオーバーロードが発生することがあります。この問題を軽減するには、多数のラウンドトリップを実行せずに、複数のメッセージをまとめます。たとえば、配列挿入、配列フェッチなどを使用できます。

プロセス管理のチェック

この項で説明するいくつかのプロセス管理の項目をチェックする必要があります。

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 の場合には不経済です。インスタンス当たりのプロセスが 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表を共有します。

オペレーティング・システムの新規プロセスの開始

オペレーティング・システムの新規プロセスを開始する際には高いコストがかかります。プログラマは、単一目的のプロセスを生成し、そのプロセスを終了後に、また新規のプロセスを生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生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、そのたびにページ表を構築する必要があるからです。共有メモリー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する必要があるため、問題が増大します。

たとえば、1GB の SGA がある場合は、4KB ごとにページ表のエントリがあり、1 つのページ表のエントリは 8 バイトになります。エントリのサイズの合計は (1G / 4KB) * 8 バイトになります。ページ表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。

17

インスタンス・リカバリ・パフォーマンスの構成

この章では、インスタンス・リカバリを行う時間を構成するガイドラインを示します。

この章には次の項があります。

インスタンス・リカバリについて

インスタンス・リカバリおよびクラッシュ・リカバリとは、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。通常の操作では、インスタンスが異常終了せず、SHUTDOWN IMMEDIATE 文を使用して正常にシャットダウンされると、まだディスク上のデータ・ファイルに書き込まれていないメモリー内の変更内容は、シャットダウン時に実行されたチェックポイントの一部としてディスクに書き込まれます。

ただし、シングル・インスタンス・データベースがクラッシュした場合、または Oracle Real Application Cluster 構成の全インスタンスがクラッシュした場合には、Oracle は次回の起動時にクラッシュ・リカバリを実行します。Oracle Real Application Cluster 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インスタンス・リカバリを自動的に実行します。インスタンスおよびクラッシュ・リカバリは 2 つのステップ、すなわち、最初はキャッシュ・リカバリ、次にトランザクション・リカバリで行われます。

キャッシュ・リカバリ（ロールフォワード）キャッシュ・リカバリ・ステップで、REDO ログ・ファイル中のコミット済みおよび未コミットの変更内容が、影響を受けたデータ・ブロックに適用されます。キャッシュ・リカバリ処理に必要な作業は、データベースに対する変更率（1 秒当たりの更新トランザクション）と、各チェックポイント間の時間に比例します。

トランザクション・リカバリ（ロールバック）データベースの一貫性を保つには、クラッシュ時点でコミットされなかった変更を取り消す（つまり、ロールバックする）必要があります。トランザクション・リカバリ・ステップで、ロールバック・セグメントを適用してコミットされていない変更を取り消します。トランザクション・リカバリ処理を実行する必要のある作業は、システム障害が発生したときにまだコミットされていなかったトランザクション数に比例します。

チェックポイント処理およびキャッシュ・リカバリ

Oracle は、定期的に**チェックポイント**を記録します。チェックポイントは最大のシステム変更番号（SCN）なので、それ以下のデータ・ブロックまたはその SCN のデータ・ブロックはすべてデータ・ファイルに書き出されることがわかっています。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 を含む REDO レコードの変更内容のみがリカバリ時に適用される必要があります。キャッシュ・リカバリ処理の所要時間は、2 つの要因、すなわち、チェックポイントの SCN より高い SCN に変更内容を持つデータ・ブロックの個数、およびこれらの変更内容を検出するために読み込む必要のあるログ・ブロックの個数で決定されます。

チェックポイントがパフォーマンスに与える影響

頻繁なチェックポイント処理では、データ・ファイルに使用済みバッファを頻繁に書き込むので、インスタンス障害発生時のキャッシュ・リカバリ時間を短縮します。チェックポイント処理が頻繁に行われる場合、現在のチェックポイント位置とログの終わりまでの REDO ログにある REDO レコードを適用すると、処理するデータ・ブロック数が比較的少なくてすみます。このため、リカバリのキャッシュ・リカバリ・フェーズはかなり短くなります。

ただし、更新が多いシステムでは、チェックポイント処理により DBW*n* プロセスが書込みを行うので、頻繁なチェックポイント処理によりランタイム・パフォーマンスが低下することがあります。

高速なインスタンス・リカバリのトレードオフ

インスタンス・リカバリの所要時間を最小にするには、Oracle が頻繁にチェックポイント処理をするように強制して、リカバリ時に適用する REDO ログ・レコード数を最小限に保つ必要があります。ただし、更新の多いシステムでは、頻繁にチェックポイント処理を行うと、通常のデータベース操作のオーバーヘッドが増加します。

日常的な操作効率がリカバリ時間の最短化よりも重要な場合は、チェックポイントによるデータ・ファイルへの書込み回数を減らします。これにより、操作効率は向上しますが、インスタンス・リカバリ時間も増えます。

関連項目：

- ランタイム・パフォーマンスを最大化する方法については、17-4 ページの「ランタイム・パフォーマンスを最適化するためのチェックポイント回数の削減」を参照してください。
- インスタンス・リカバリ時間を最小化する方法については、17-5 ページの「キャッシュ・リカバリ所要時間の構成」を参照してください。

ランタイム・パフォーマンスを最適化するためのチェックポイント回数の削減

チェックポイントの回数を減らし、ランタイム・パフォーマンスを最適化するためには、次のことを行います。

- システムが生成する REDO の量に従って、オンライン REDO ログ・ファイルをサイズ設定します。およその目安として、多くても 20 分ごとに 1 回ログを切り替えます。小さなログ・ファイルは、チェックポイント・アクティビティを増加し、パフォーマンスを低下させることがあります。ログはすべて同じサイズにしてください。
- LOG_CHECKPOINT_INTERVAL 初期化パラメータの値（物理ブロック・サイズの倍数）を 0（ゼロ）に設定します。この値によって、定期的なチェックポイントがなくなります。
- LOG_CHECKPOINT_TIMEOUT 初期化パラメータの値を 0（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。
- FAST_START_MTTR_TARGET（および FAST_START_IO_TARGET）の値を 0（ゼロ）に設定し、ファスト・スタート・チェックポイント機能を無効にします。

関連項目： チェックポイントの詳細は、『Oracle9*i* データベース概要』を参照してください。

キャッシュ・リカバリ所要時間の構成

キャッシュのリカバリをチューニングして、リカバリ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。これらの方法には次のものがあります。

- キャッシュ・リカバリ時間に影響を与える初期化パラメータ
- ファスト・スタート・チェックポイント処理を使用したインスタンス・リカバリ時間の制限
- REDO の量に影響を与える LOG_CHECKPOINT_TIMEOUT の設定
- REDO の量に影響を与える LOG_CHECKPOINT_INTERVAL の設定
- REDO アプリケーションを高速化するためのパラレル・リカバリの使用

キャッシュ・リカバリ時間に影響を与える初期化パラメータ

例 17-1 の初期化パラメータは、キャッシュ・リカバリ時間に影響を与えます。

表 17-1 キャッシュ・リカバリに影響を与える初期化パラメータ

パラメータ	用途
FAST_START_MTTR_TARGET	予測される「平均リカバリ時間」(MTTR)、すなわち、リカバリを実行してインスタンスを起動するのに要する予想時間を秒単位で指定できます。
FAST_START_IO_TARGET	この初期化パラメータは、FAST_START_MTTR_TARGET の方が適切であるために使用されなくなりました。このパラメータは、使用済みバッファ数の上限を指定します。
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 ブロック数を制限します。
RECOVERY_PARALLELISM	インスタンスまたはクラッシュ・リカバリで使用する同時リカバリ・プロセス数を指定します。
LOG_PARALLELISM	Oracle の REDO 割当ての並行性のレベルを指定します。

注意： インスタンス障害の後に起動時間を制御するには、FAST_START_MTTR_TARGET 初期化パラメータを使用することをお薦めします。ファスト・スタート・チェックポイント処理は、Enterprise Edition でのみ使用できます。

FAST_START_IO_TARGET 初期化パラメータは、FAST_START_MTTR_TARGET の方が適切であるために使用されなくなりました。

DB_BLOCK_MAX_DIRTY_TARGET 初期化パラメータは削除されました。

ファスト・スタート・チェックポイント処理を使用したインスタンス・リカバリ時間の制限

Oracle Enterprise Edition の機能には、インスタンス・リカバリを制御するファスト・スタート・フォルト・リカバリ機能が含まれています。この機能を使用すると、キャッシュ・リカバリに必要な時間が短縮され、使用済みバッファ数および最新の REDO レコードと最後のチェックポイントの間で生成された REDO レコード数を制限することにより、リカバリの境界を設定および予測できます。

ファスト・スタート・リカバリ機能の基盤はファスト・スタート・チェックポイント・アーキテクチャです。一括書込みを行う従来のイベント・ドリブン（すなわち、ログ・スイッチング）チェックポイント処理のかわりに、ファスト・スタート・チェックポイント処理が段階的に行われます。各 DBWn プロセスは、チェックポイントの位置を進めるためにバッファをディスクに定期的に書き込みます。最も古い変更済みブロックが最初に書き込まれて、書込みごとにチェックポイントを進めるようにします。ファスト・スタート・チェックポイント機能によって、一括書込みおよび従来のチェックポイント処理で発生していた I/O スパイクをなくします。

管理者は FAST_START_MTTR_TARGET 初期化パラメータで、リカバリのキャッシュ・リカバリ・フェーズを完了させる目標（制限した）時刻を指定すると、Oracle は自動的にその目標にあわせて段階的なチェックポイント書込みを変更します。FAST_START_MTTR_TARGET 初期化パラメータを使用して、予想される平均リカバリ時間（MTTR）を秒単位で指定できます。これらは、Oracle が単一インスタンスのクラッシュ・リカバリまたはインスタンス・リカバリを実行する予想時間です。

FAST_START_MTTR_TARGET

FAST_START_MTTR_TARGET 初期化パラメータは、インスタンスまたはシステム障害からのリカバリ時間の構成を簡素化します。このパラメータで、クラッシュ・リカバリまたはインスタンス・リカバリに要する予想時間の秒数を指定できます。FAST_START_MTTR_TARGET は、リカバリ時間ができるだけこの予想時間に近づくように、Oracle の操作を変更する一連のパラメータに内部的に変換されます。

注意： FAST_START_MTTR_TARGET を使用する場合は、FAST_START_IO_TARGET、LOG_CHECKPOINT_INTERVAL および LOG_CHECKPOINT_TIMEOUT 初期化パラメータを無効にするか、削除する必要があります。これらのパラメータをアクティブな値に設定すると FAST_START_MTTR_TARGET を妨げるので、V$INSTANCE_RECOVERY ビューの TARGET_MTTR 列が期待値とは異なる値になります。

FAST_START_MTTR_TARGET の最大値は 3600 または 1 時間です。値を 3600 より大きい値に設定すると、その値は 3600 に丸められます。FAST_START_MTTR_TARGET の最小値はありません。ただし、このことはリカバリ時間の目標を必要なだけ低い時間に設定できるということではありません。クラッシュ・リカバリを行う時間は、目標の使用済みバッファ数の下限（1000）と、初期化およびファイルのオープンに要する時間などの要因によって制限されます。

FAST_START_MTTR_TARGET の値を低すぎる値に設定すると、有効な平均リカバリ時間（MTTR）の目標は、システムが達成できる最良の MTTR になります。最も不適切な状況のリカバリでもそのように長い時間がかからないような高い値に FAST_START_MTTR_TARGET の値を設定すると、有効な MTTR の目標は、バッファ・キャッシュ全体が使用済みという最も不適切な状況のシナリオの予測 MTTR になります。有効な MTTR を調べるには、V$INSTANCE_RECOVERY ビュー内の TARGET_MTTR 列を使用してください。

注意： V$INSTANCE_RECOVERY 内の TARGET_MTTR 列は、V$INSTANCE_RECOVERY の設定値が低すぎるか高すぎる場合には FAST_START_MTTR_TARGET とは異なるものになります。定期的に V$INSTANCE_RECOVERY ビュー内の MTTR_TARGET 列をチェックし、パラメータの設定と比較してください。パラメータの設定が一貫して目標の値とは異なる場合、パラメータの設定を調整してください。

関連項目： FAST_START_MTTR_TARGET 設定の詳細は、17-11 ページの「予測 MTTR の監視 : シナリオ例」を参照してください。

REDO の量に影響を与える LOG_CHECKPOINT_TIMEOUT の設定

初期化パラメータ LOG_CHECKPOINT_TIMEOUT を整数値 *n* 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 REDO ログへの最後の書込みブロックとの間で、最大 *n* 秒のログ・アクティビティが発生する可能性があることを意味します。これによって、チェックポイント位置が最新 REDO ブロックと一定の時間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、すなわち、30 分です。

REDO の量に影響を与える LOG_CHECKPOINT_INTERVAL の設定

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、ログがいっぱいになり、ログ・スイッチが試行される前に現在のログにチェックポイントが実行されるようにしています。

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。V$INSTANCE_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。

関連項目：

- 17-13 ページ「パフォーマンスのオーバーヘッドの計算」
- チューニング・チェックポイントの詳細は、第 15 章「I/O 構成および設計」を参照してください。

REDO アプリケーションを高速化するためのパラレル・リカバリの使用

リカバリのキャッシュ・リカバリ・フェーズをチューニングするには、パラレル・リカバリを使用します。パラレル・リカバリは作業分割アプローチを使用して、リカバリのキャッシュ・リカバリ・フェーズで様々なプロセスを様々なデータ・ブロックに割り当てます。

たとえば、リカバリの間に、REDO ログが読み込まれ、REDO 適用を必要とするブロックが解析されます。これらのブロックは、以降すべてのリカバリ・プロセスに均等に分散されバッファ・キャッシュに読み込まれます。データ・ファイルが様々なディスク・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リカバリは、パラレル・リカバリに適しています。

RECOVERY_PARALLELISM 初期化パラメータを使用して、インスタンスまたはクラッシュ・リカバリでの同時リカバリ・プロセス数を指定します。パラレル・プロセッシングを使用するには、RECOVERY_PARALLELISM パラメータの値が 1 より大きい必要がありますが、PARALLEL_MAX_SERVERS 初期化パラメータの値を超えることはできません。

LOG_PARALLELISM 初期化パラメータは REDO のパラレル生成を可能にし、更新中心の特定のワークロードのスループットを改善することができます。16 を超えるプロセッサを有するハイエンド・サーバー上で Oracle を使用中に、REDO 割当てラッチで競合が非常に多い場合は、並行性 REDO を有効にすることを検討する必要があります。

注意： RECOVERY_PARALLELISM 初期化パラメータを使用して、インスタンスまたはクラッシュ・リカバリでの同時リカバリ・プロセス数を指定します。

メディア・リカバリはこのパラメータの影響を受けません。メディア・リカバリについては、RECOVER DATABASE 文の PARALLEL 句を使用します。

リカバリは、通常はデータ・ブロックの読込みで 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 が高速化する場合にのみ、リカバリのパフォーマンスに役立ちます。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・リカバリを使用しても、リカバリが CPU バインドでない限り大幅には改善されません。

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

キャッシュ・リカバリの監視

V$INSTANCE_RECOVERY ビューを使用してリカバリ・パラメータの現在の設定を確認します。このビューの統計を使用して、チェックポイントに最も大きな影響を持つパラメータを計算して求めることもできます。V$INSTANCE_RECOVERY には表 17-2 に示す列が含まれます。

注意： V$INSTANCE_RECOVERY の最後の 3 つのフィールドは Oracle9*i* で新規で、最も重要です。FAST_START_MTTR_TARGET 初期化パラメータを使用すると、V$INSTANCE_RECOVERY のその他の 7 つのフィールドは有用性が低くなります。

表 17-2 V$INSTANCE_RECOVERY ビュー

列	説明
RECOVERY_ESTIMATED_IOS	バッファ・キャッシュ内の使用済みバッファ数が含まれています（Standard Edition では、このフィールドの値は常に NULL です）。
ACTUAL_REDO_BLKS	リカバリのための読込みに必要な REDO ブロックの現在の数。
TARGET_REDO_BLKS	リカバリ時に処理される REDO ブロックの最大数の目標。この値は、3 つの列（LOG_FILE_SIZE_REDO_BLKS、LOG_CHKPT_TIMEOUT_REDO_BLKS および LOG_CHKPT_INTERVAL_REDO_BLKS）の最小値です。
LOG_FILE_SIZE_REDO_BLKS	リカバリ時に処理する REDO ブロック数で、最小のログ・ファイルのサイズの 90% に対応します。
LOG_CHKPT_TIMEOUT_REDO_BLKS	LOG_CHECKPOINT_TIMEOUT を満たす、リカバリ時に処理される必要がある REDO ブロック数。
LOG_CHKPT_INTERVAL_REDO_BLKS	LOG_CHECKPOINT_INTERVAL を満たす、リカバリ時に処理される必要がある REDO ブロック数。
FAST_START_IO_TARGET_REDO_BLKS	このフィールドは廃止されました。このフィールドは、下位互換性のために残されています。このフィールドの値は常に NULL です。

表 17-2 V$INSTANCE_RECOVERY ビュー（続き）

列	説明
TARGET_MTTR	秒単位で示される有効平均リカバリ時間（MTTR）の目標。この目標値は、通常は FAST_START_MTTR_TARGET 初期化パラメータの値と同じになります。FAST_START_MTTR_TARGET に設定した値が小さすぎるため、時間内にリカバリを行えない場合、TARGET_MTTR フィールドには FAST_START_MTTR_TARGET より大きい有効な MTTR の目標値が含まれています。最悪の場合（バッファ・キャッシュ全体が使用済みである）のリカバリでも、それほど長い時間がかからないような高い値に FAST_START_MTTR_TARGET を設定すると、TARGET_MTTR フィールドには最悪の場合のシナリオの予測 MTTR が入ります。FAST_START_MTTR_TARGET を指定しない場合、このフィールドは 0（ゼロ）です。
ESTIMATED_MTTR	使用済みバッファとログ・ブロックの個数に基づいた秒数で示される現行の予測平均リカバリ時間（MTTR）（FAST_START_MTTR_TARGET を指定しない場合でも、現行の予測 MTTR を示します）。
CKPT_BLOCK_WRITES	チェックポイントを使用禁止にしていれば回避されていたディスクへの書込みの回数。

関連項目： V$INSTANCE_RECOVERY ビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

予測 MTTR の監視 : シナリオ例

V$INSTANCE_RECOVERY の TARGET_MTTR フィールドには、有効な MTTR の目標が含まれています。V$INSTANCE_RECOVERY の ESTIMATED_MTTR フィールドには、クラッシュがただちに発生した場合の予測 MTTR が含まれています。システムが指定された MTTR の目標に追随できるかどうかを調べるために、これらの 2 つのフィールドへの問合せを行います。

例として、初期化パラメータの設定が次のとおりであるとします。

```
FAST_START_MTTR_TARGET = 6      # seconds
```

データベースをオープンした後に次の問合せを行います。

```
SELECT TARGET_MTTR, ESTIMATED_MTTR, CKPT_BLOCK_WRITES
FROM V$INSTANCE_RECOVERY;
```

Oracle から次の結果が戻されます。

```
TARGET_MTTR ESTIMATED_MTTR CKPT_BLOCK_WRITES
18          15             0
```

TARGET_MTTR は 18 秒です。この値は、指定された FAST_START_MTTR_TARGET の値（6 秒）より大きい値です。つまり、6 秒以内にデータベースのリカバリを行うのは不可能です。18 秒は、システムが達成できる MTTR の最小目標です。

最小値の 18 秒は、使用済みバッファ数の目標の明確な下限値の 1000 ブロックに基づいて計算されます（使用されなくなった FAST_START_IO_TARGET 初期化パラメータはこの下限に従います。すなわち、FAST_START_IO_TARGET を 1000 未満の値には設定できません）。ESTIMATED_MTTR フィールドには予測平均リカバリ時間が入ります。データベースはオープンしたばかりであるため、システムには使用済みバッファがほとんどありません。そのため、ESTIMATED_MTTR は最小の TARGET_MTTR より低くなる可能性があります。

ここで、次の文を使用して FAST_START_MTTR_TARGET を修正するとします。

```
ALTER SYSTEM SET FAST_START_MTTR_TARGET = 30;
```

V$INSTANCE_RECOVERY に問合せを再発行すると、Oracle が次の結果を戻します。

```
TARGET_MTTR ESTIMATED_MTTR CKPT_BLOCK_WRITES
30          15             0
```

ESTIMATED_MTTR フィールドは依然 15 秒です。つまり、現時点での予測 MTTR（クラッシュがただちに発生した場合）も 15 秒です。これは、新しい REDO が書き込まれず、また使用済みになったデータ・ブロックもないためです。

集中的な更新アクティビティがデータベースで発生し、その直後に V$INSTANCE_RECOVERY に問い合せたと仮定します。Oracle から次の結果が戻されます。

```
TARGET_MTTR ESTIMATED_MTTR CKPT_BLOCK_WRITES
30          36             54367
```

有効な MTTR 目標が 30 秒であることを確認します。現時点の予想 MTTR（クラッシュがただちに発生した場合）は 36 秒です。これは問題ありません。これは、まだ終了していないチェックポイントの書込みが存在する可能性があるため、バッファ・キャッシュには目標より多くの使用済みバッファが含まれていることを意味します。

ここで、1 分間待ち、V$INSTANCE_RECOVERY に問合せを再発行するとします。Oracle から次の結果が戻されます。

```
TARGET_MTTR ESTIMATED_MTTR CKPT_BLOCK_WRITES
30          31             55230
```

この時点の予測 MTTR は 31 秒に低下しました。これは、この期間にさらに多くの使用済みバッファが書き出されたためです。このことは、V$INSTANCE_RECOVERY の CKPT_BLOCK_WRITES フィールドの増加でわかります。

注意： physical writes の個数から physical writes non checkpoint（V$SYSSTAT から）の個数を引いた値は、V$INSTANCE_RECOVERY 内の CKPT_BLOCK_WRITES フィールドに等しくなります。

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',
               '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
```

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込み回数が表示されます。

このデータを使用して、FAST_START_MTTR_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て、ファスト・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))] x 100% = EIO
```

変数は表 17-3 で説明します。

表 17-3 変数の定義

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です。
*_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です。
PWNC	physical writes non checkpoint
PW	physical writes
PR	physical reads
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

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_MTTR_TARGET の値を増やします。

FAST_START_MTTR_TARGET を 0（ゼロ）以外の値に設定して生じる追加書込み数は、アプリケーションにより異なります。キャッシュ・サイズには依存しません。

パフォーマンスのオーバーヘッドの計算 : シナリオ例

例として、初期化パラメータの設定が次のとおりであるとします。

```
FAST_START_MTTR_TARGET = 90  # 90 seconds
```

統計が安定してから、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
```

physical write checkpoint 統計は、V$INSTANCE_RECOVERY ビューの CKPT_BLOCK_WRITES フィールドでも見ることができます。その例を次に示します。

```
SELECT CKPT_BLOCK_WRITES
FROM V$INSTANCE_RECOVERY;
```

Oracle から次の結果が戻されます。

```
CKPT_BLOCK_WRITES    3767
```

その結果は、V$SYSSTAT: 3767 = 14932 - 11165 からの結果に一致します。

更新を数時間行った後、問合せを再発行します。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
```

SELECT 文から戻された値を 17-13 ページの計算式に代入して、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判別します。

```
[((17467 - 14932) - (13231 - 11165)) / ((3011 - 2376) + (17467 - 14932))] x 100% = 14.8%
```

この結果が示すように、ファスト・スタート・チェックポイントを使用可能にすることで、ファスト・スタート・チェックポイントを使用可能にし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、リカバリ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。

リカバリ時間を短縮するには、FAST_START_MTTR_TARGET パラメータの値を 60 に減らします。バッファ・キャッシュの項目が除去された後に、第 2 の区間全体で V$SYSSTAT 統計を計算して新しいパフォーマンス・オーバーヘッドを算出します。V$SYSSTAT を問い合せます。

```
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
```

更新を行った後に、問合せを再発行します。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
```

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))] x 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。

MTTR の調整

FAST_START_MTTR_TARGET 初期化パラメータを使用すると、REDO ログの長さとデータ・キャッシュ内の使用済みデータ・バッファ数を制限するために内部のシステム・トリガー値が計算されます。この計算では、REDO ブロックの読み込みとデータ・ブロックの読込みおよび書込みの予測時間を使用します。

最初は、内部デフォルトが使用されます。これらのデフォルトは、システム操作時に見積り実行時間と置き換えられます。ただし、最適値は障害からの実際のリカバリから取られた測定値から取得されます。

注意： FAST_START_MTTR_TARGET を効果的に位置合せするには、いくつかのインスタンス・リカバリを実行して、REDO ブロックを読み込む時間とデータ・ブロックの読込みおよび書込みを行う時間が正確に記録されるようにします。

インスタンス・リカバリを行って FAST_START_MTTR_TARGET を調整する前に、データベース・クラッシュまたはハードウェア・クラッシュが発生した場合に FAST_START_MTTR_TARGET を調整するかどうかを決定します。これは、データベース・ファイルがファイル・システムに格納される場合、または I/O サブシステムにメモリー・キャッシュがある場合に考慮する必要があります。ファイルがキャッシュされるかどうかにより、ディスクに対する読込みおよび書込みにかなりの違いがあるからです。インスタンス・リカバリ時に実行されるワークロードは、生成された REDO レコードの量が同程度かどうかを確認する上で、システム上の平均ワークロードをきわめて適切に表現している必要があります。

MTTR アドバイザ

Oracle9*i* リリース 2（9.2）から、MTTR アドバイザが使用できます。追加の物理書込みに関して、異なる MTTR の設定がシステム・パフォーマンスに与える影響を評価する際に役立ちます。

MTTR アドバイザの動作

MTTR アドバイザが使用可能な場合に、しばらくの間通常のワークロードでシステムを実行して V$MTTR_TARGET_ADVICE を問い合せると、他の MTTR 設定におけるキャッシュ書込み回数の見積りに対する、現在の MTTR 設定におけるキャッシュ書込み回数の比率がわかります。たとえば、比率が 1.2 の場合、キャッシュ書込み回数が 20%多いことを示します。

異なる MTTR の設定と、それに対応するキャッシュ書込み比率を確認することで、リカバリおよびパフォーマンス要件に適合する MTTR の値を判別できます。V$MTTR_TARGET_ADVICE では、総物理書込み（ダイレクト書込みを含む）の比率と、総 I/O（読込みを含む）の比率が提供されます。

MTTR アドバイザの有効化

MTTR アドバイザを有効化するには次の 2 つの初期化パラメータを設定する必要があります。

- STATISTICS_LEVEL
- FAST_START_MTTR_TARGET

STATISTICS_LEVEL

STATISTICS_LEVEL は、TYPICAL または ALL に設定されていることを確認してください。

FAST_START_MTTR_TARGET

MTTR アドバイザを有効化するには、初期化パラメータ FAST_START_MTTR_TARGET を 0（ゼロ）以外の値に設定します。FAST_START_MTTR_TARGET が指定されていない場合、MTTR アドバイザは OFF になります。

MTTR アドバイザが ON の場合、次の 5 種類の MTTR 設定でチェックポイントのキュー動作がシミュレートされます。

- 現在の FAST_START_MTTR_TARGET 設定
- 現在の設定に 0.1、0.5、1.5 および 2 を乗算した値

注意： FAST_START_MTTR_TARGET を 0（ゼロ）以外の値に設定する場合、および MTTR アドバイザが ON の場合は、次のパラメータを無効化（0（ゼロ）に設定）することをお薦めします。

- LOG_CHECKPOINT_TIMEOUT
- LOG_CHECKPOINT_INTERVAL
- FAST_START_IO_TARGET

これらの初期化パラメータは、FAST_START_MTTR_TARGET を上書きしたり、FAST_START_MTTR_TARGET よりもっと活発にチェックポイントを動作させる可能性を持っているため、シミュレーションの妨げとなることがあります。

MTTR アドバイザの表示

Oracle9*i* リリース 2（9.2）には、MTTR アドバイザで収集した統計またはアドバイザを表示するための動的パフォーマンス・ビューが提供されています。

V$MTTR_TARGET_ADVICE

MTTR アドバイザが ON になっていると、V$MTTR_TARGET_ADVICE に収集済みのアドバイザ情報が表示されます。通常このビューには 5 行が表示され、それぞれ現在の MTTR、現在の MTTR に 0.1 を乗算した値、0.5 を乗算した値、1.5 を乗算した値、および 2 を乗算した値に対応しています。ただし、5 つの値のうちの 1 つ以上が、システムで維持可能な最小 MTTR 目標よりも小さい場合、それに対応する行は、システムで維持可能な最小 MTTR 目標に対応する単一行で置き換えられます。同様に、5 つの値のうちの 1 つ以上が、システムで維持可能な最悪の MTTR 目標よりも大きい場合、それに対応する行は、システムで維持可能な最悪の MTTR 目標に対応する単一行で置き換えられます。

MTTR アドバイザが OFF の間は、MTTR アドバイザを最後にオンにしたときに収集された情報が表示されます。

関連項目： これらのビューの列の詳細は、24-22 ページの「V$MTTR_TARGET_ADVICE」を参照してください。

トランザクション・リカバリのチューニング

インスタンス・リカバリの 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、ファスト・スタート・オン・デマンド・ロールバックとファスト・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、このリカバリ・フェーズの効率を上げます。

注意： これらの機能はファスト・スタート・フォルト・リカバリに含まれ、Oracle9*i* Enterprise Edition でのみ使用可能です。

この項の項目は次のとおりです。

- ファスト・スタート・オン・デマンド・ロールバックの使用
- ファスト・スタート・パラレル・ロールバックの使用

ファスト・スタート・オン・デマンド・ロールバックの使用

ファスト・スタート・オン・デマンド・ロールバック機能を使用すると、データベースのオープン直後（通常はキャッシュ・リカバリ終了の直後）から新しいトランザクションを自動的に開始できます。終了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と、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のみ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、ロング・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に、パラメータの設定や文の発行を行う必要はありません。

ファスト・スタート・パラレル・ロールバックの使用

ファスト・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、ファスト・スタート・パラレル・ロールバックのトランザクション・リカバリに対する関係は、パラレル・リカバリのキャッシュ・リカバリに対する関係と同じです。

ファスト・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、いつパラレル・ロールバックを開始し、複数のパラレル・プロセス間に作業を分散させるかを自動的に決定します。プロセス 1 では 1 つのトランザクションをロールバックし、プロセス 2 では次のトランザクションをロールバックする、というようにロールバックします。

ファスト・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リカバリです。トランザクション内リカバリでは、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 つのパラレル処理を割り当ててリカバリ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、Oracle は複数プロセスにトランザクション 5 を分割して自動的にトランザクション内リカバリを開始します。つまり、プロセス 1 が 1 つのパート、プロセス 2 が別のパートというように分割します。

初期化パラメータ FAST_START_PARALLEL_ROLLBACK を表 17-4 にリストされている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・リカバリに関係するプロセス数を制御します。

表 17-4 FAST_START_PARALLEL_ROLLBACK パラメータ値

値	意味
FALSE	ファスト・スタート・パラレル・ロールバックをオフにします。
LOW	リカバリ・サーバー数が CPU_COUNT 初期化パラメータの値の 2 倍を超えないことを指定します。
HIGH	リカバリ・サーバー数が CPU_COUNT 初期化パラメータの値の 4 倍を超えないことを指定します。

Oracle Real Application Clusters の構成でのパラレル・ロールバック

Oracle Real Application Clusters では、ファスト・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リカバリ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・セグメントがオンラインになった後は、そのインスタンスのみ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。

ファスト・スタート・パラレル・ロールバックの進捗の監視

ファスト・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 ビューを調べます。V$FAST_START_SERVERS は、ファスト・スタート・パラレル・ロールバックを処理するすべてのリカバリについての情報を提供します。V$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：

- UNDO 領域管理の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。
- Oracle Real Application Clusters 環境でのファスト・スタート・パラレル・ロールバックの詳細は、『Oracle9*i* Real Application Clusters 配置およびパフォーマンス』を参照してください。
- 初期化パラメータの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- 透過的アプリケーション・フェイルオーバー（TAF）の詳細は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

18

UNDO セグメントと一時セグメントの構成

UNDO および一時セグメントを構成する場合にはパフォーマンスを考慮する必要があります。

この章には次の項があります。

UNDO セグメントの構成

Oracle では、UNDO データの管理を完全に自動化する、自動 UNDO 管理を提供しています。自動 UNDO 管理モードで動作するデータベースは、UNDO セグメントを透過的に作成および管理します。自動 UNDO 管理を使用することをお薦めするのは、データベース管理を大幅に簡素化し、UNDO（ロールバック）セグメントの手動チューニングの必要がないためです。ロールバック・セグメントを使用する手動 UNDO 管理がサポートされるのは、下位互換性のためです。

自動 UNDO 管理の構成

自動 UNDO を構成するには、次の初期化パラメータを追加するのみでできます。

```
UNDO_MANAGEMENT=AUTO
```

UNDO 表領域を作成し、その表領域に保持される UNDO データの最大保存時間を決定することもできます。

> **関連項目：** 自動 UNDO の構成方法の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

ロールバック・セグメントの構成

自動 UNDO 管理は、ロールバック領域を処理する望ましい方法です。自動 UNDO 管理では、静的に割り当てられたロールバック・セグメント・セットに UNDO 領域を配分するかわりに、単一の UNDO 表領域に UNDO 領域を割り当てることができます。UNDO セグメント間での領域の作成と割当ては、Oracle サーバーで自動的に行われます。

ロールバック・セグメントで、1 つ以上の表領域が作成されます。ロールバック・セグメントは、これらの表領域に手動で作成されます。ロールバック・セグメントの数とサイズは、DBA が決定する必要があります。

ロールバック・セグメントの数とサイズの決定

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な領域管理は避けてください。

表 18-1 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合せに対して適切なものです。

表 18-1 ロールバック・セグメント数の選択

現行トランザクション数（*n*）	ロールバック・セグメントの推奨数
n < 16	4
16 <= n < 32	8
32 <= n	n/4

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。

> **関連項目：** 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、および OPTIMAL パラメータの調整の詳細は、『Oracle9*i* データベース管理者ガイド』を参照してください。

長い問合せの場合 長い問合せが選択しているデータを同時実行で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込み一貫性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。

ロング・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ます。できるだけ避けてください。

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、エクステントが 2 つあり、サイズはおよそ 10KB 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。

一時表領域の構成

一時表領域を構成すると、ディスク・ソートのパフォーマンスの最適化に役立ちます。そのためには、適切な STORAGE 句と、ソートに使用する正しいタイプの表領域を選択することです。

ソート表領域のデフォルト STORAGE 句を選択するには、次のことを行います。

- `PCTINCREASE` を 0（ゼロ）にします。
- `INITIAL` および `NEXT` を同じサイズに設定し、`SORT_AREA_SIZE` の因数を設定します。

正しいタイプの表領域を選択すると、ディスク・ソートはさらに効率的になります。ディスク・ソートに使用できる様々な表領域は、次のとおりです。

- 一時表領域
- TEMPORARY タイプの表領域
- 永続的な表領域

一時表領域 一時表領域は、ディスク・ソートの最も効率的な表領域です。一時表領域の特性は、次のとおりです。

- 領域管理（エクステントの割当ておよび割当て解除）はローカルに管理されます。したがって、ST エンキューを使用することは避けられます。
- 各インスタンスに作成されたソート・セグメントは再利用されます（このセグメントは表領域が削除された場合のみ削除されます）。Oracle Real Application Clusters を使用する場合、1 つのインスタンス当たり 1 つのソート・セグメントが必要です。
- ソートを実行するすべてのプロセスでは、各ソートにセグメント（および潜在的な多数のエクステント）を割り当てるのではなく、ソート・セグメントの既存のソート・エクステントを再利用します。現在動作しているソートの数に対するエクステント数が不十分である場合は、必要なエクステントがインスタンスの起動ごとに 1 回追加されます。それ以降、これらのエクステントはリサイクルされます。

- 15-6 ページの「I/O の基本構成」の説明に従って、一時表領域をストライプ化できます。表領域が I/O ボトルネックを作成しているときのみ、その表領域を他の表領域と区別する必要があります。
- 一時表領域を作成するには、CREATE TEMPORARY TABLESPACE 文を使用します。

  関連項目：

  - 一時表領域の詳細は、『Oracle9*i* データベース概要』を参照してください。
  - CREATE TEMPORARY TABLESPACE 文の使用方法の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

TEMPORARY タイプの表領域 一時表領域に続いて、ソート操作に使用する次に最適な表領域はこれらのタイプの表領域です。TEMPORARY タイプの表領域の特性は次のとおりです。

- 領域管理はディクショナリ管理であるため、領域管理では ST エンキューを使用します。
- インスタンスのソート・セグメントは、インスタンスの起動時に削除され、最初のソートが実行されるときに再作成されます。その後、必要に応じてエクステントが割り当てられます。
- 領域管理はディクショナリ管理形式ですが、領域の割当ておよび割当て解除の回数は減ります。ソートを実行するすべてのプロセスでは、各ソートにエクステントとセグメントの割当ておよび割当て解除するのではなく、単一のソート・セグメントの既存のソート・エクステントを再利用します。Oracle Real Application Clusters を使用する場合、1 つのインスタンス当たり 1 つのソート・セグメントが必要です。ソート数に対するエクステント数が不十分である場合、必要なエクステントがインスタンスの起動ごとに 1 回追加され、それ以降はリサイクルされます。そのため、ST エンキューの負荷をかなり軽減します。
- TEMPORARY タイプの表領域を作成するには、TEMPORARY 句を使用する CREATE TABLESPACE 文または ALTER TABLESPACE 文を使用します。

  関連項目： TEMPORARY 句の使用方法の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

永続的な表領域 永続的な表領域（TEMPORARY タイプではない）は、ディスク・ソートのパフォーマンス的に見て最も非効率的です。この理由は、次のとおりです。

- ST エンキューは、ソート・セグメントに割り当てられた各エクステントの割当ておよび割当て解除に使用されます。
- ソート・セグメントは再利用されません。ディスク・ソートを実行する各プロセスでは、それぞれのソート・セグメントを作成した後に、それらを削除します。また、単一のソート操作では、多くのエクステントの割当ておよび割当て解除を必要とする場合があり、各エクステント割当てには ST エンキューが必要です。

最適なパフォーマンスを得るには、一時表領域を使用することが最適な選択です。一時表領域は、領域管理に ST エンキューを使用する必要性をなくし、ソート・エクステントを再利用できるという別の利点があります。

表 18-2 表領域のソート

表領域タイプ	領域管理	ソート・セグメントの再利用	ソート・エクステントの再利用	文の作成
一時 (一時ファイルを使用)	ローカル管理	可。ソート・セグメントは削除されません。	常時	CREATE TEMPORARY TABLESPACE
TEMPORARY タイプ (データ・ファイルを使用)	ディクショナリ管理	インスタンス起動時にソート・セグメントが再作成されます。	可。エクステントは、インスタンス起動時にソート・セグメントが削除されるまで再利用されます。	CREATE TABLESPACE ... TYPE TEMPORARY
永続的	ディクショナリ管理	不可。ソート・セグメントは再利用されません。	不可。ソート・セグメントは再利用されません。	CREATE TABLESPACE

一時表領域および TEMPORARY タイプの表領域には、表やロールバック・セグメントなどの永続オブジェクトを含めることはできません。

関連項目： CREATE TABLESPACE 文および ALTER TABLESPACE 文の構文の詳細は、『Oracle9*i* SQL リファレンス』を参照してください。

19

共有サーバーの構成

共有サーバーを適切に構成するとパフォーマンスが大幅に向上します。

この章には次の項があります。

共有サーバーのパフォーマンスの概要

共有サーバーを使用すると、プロセス数と、サーバー・マシンで消費されるメモリー量を減らすことができます。共有サーバーは、多数の OLTP ユーザーが断続的なトランザクションを実行するシステムに有効です。

また、データベースへの単位時間当たりの接続数が高いシステムでは、一般的に専用サーバーよりも共有サーバーを使用する方が適しています。共有サーバーでは、接続要求を受けるよりも前に同時接続要求を処理するためのディスパッチャが使用可能な状態になっています。一方、専用サーバーの場合は、接続固有の専用サーバーが、接続要求ごとに順次初期化されます。

共有サーバー・アーキテクチャを使用すると、特定のデータベース機能のパフォーマンスが改善することも、または多少低下することもあります。たとえば、パラレル実行がアクティブな場合、セッションが別の共有サーバーに移行できないことがあります。

クライアントからの要求が処理された後もセッションを移行できない場合があります。これはすべてのユーザー情報が、UGA に格納されているとは限らないためです。サーバーがクライアントからの要求の処理を待機していた場合、UGA に格納されていなかったユーザー状態に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。

共有サーバー数の構成

ある種の機能を使用する場合、あるサーバーがセッションに長期間バインドされる可能性があるため、追加の共有サーバーを構成する必要が生じる場合があります。

この項では、Oracle のアーキテクチャで使用するプロセスの競合を低減する方法について説明します。

- ディスパッチャ固有のビューを使用する競合の識別
- 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
- 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

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$DISPATCHER は、ディスパッチャ・プロセスの一般的な情報を提供します。
- V$DISPATCHER_RATE は、ディスパッチャ・プロセスの統計を提供します。

関連項目：

- これらのビューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- 統計を監視する Oracle Tuning Pack アプリケーションについては、『Oracle Enterprise Manager 概要』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、現在のサンプルの統計です。接頭辞 AVG_ が付いている統計は、コレクション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かつ最大値を下回る場合、共有サーバー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場合があります。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一般規則としてお薦めします。共有サーバー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$DISPATCHER_RATE 統計をポーリングすることによってワークロードをシミュレートすることもできます。これらの統計の正しい解釈方法は、プラットフォームごとに異なります。アプリケーションの種類が変わると、V$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・プロセスを追加する方法と接続プーリングを使用可能にする方法について説明します。

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 文の SET オプションを使用して DISPATCHERS 初期化パラメータの値を増やします。

ディスパッチャ・プロセスの総数は、MAX_DISPATCHERS 初期化パラメータの値で制限されます。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場合があります。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・プロセスの追加の詳細は、『Oracle9*i* データベース管理者ガイド』および『Oracle9*i* Net Services 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

システム負荷が増加し、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、接続プーリング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。

DISPATCHERS を使用すると、それぞれのディスパッチャに様々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。その例を次に示します。

```
DISPATCHERS = "(PROTOCOL=TCP)(POOL=ON)"
```

オプション属性 POOL は、Oracle Net 接続プーリング機能を使用可能にします。TICK は、ネットワーク TICK のサイズ（秒数）です。TICK のデフォルトは 15 秒です。

> **関連項目：** DISPATCHERS パラメータとそのオプションの詳細は、『Oracle9*i* データベース・リファレンス』および『Oracle9*i* Net Services 管理者ガイド』を参照してください。

セッション多重化の有効化

多重化は、Connection Manager プロセスで、複数ユーザーから個別のディスパッチャへのネットワーク・セッションを確立およびメンテナンスする場合に使用されます。たとえば、いくつかのユーザー・プロセスが Connection Manager プロセスからの 1 つの接続を経由して 1 つのディスパッチャに接続できます。

Connection Manager は、ユーザーとディスパッチャの通信を共有接続経由で管理します。Connection Manager 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（ゼロ）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、ディスパッチャ・プロセスが最大限に使用されるため、セッションの多重化は有効です。

多重化は、ディスパッチャ間のデータベース・リンク・セッションを多重化する場合にも役立ちます。各ディスパッチャのセッション数の制限はプラットフォームによって異なります。その例を次に示します。

```
DISPATCHERS="(PROTOCOL=TCP)(MULTIPLEX=ON)"
```

共有サーバー・プロセスの競合の低減

この項では、共有サーバーの競合を識別する方法と共有サーバー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 V$QUEUE を使用します。このビューには、共有サーバー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のみがこのビューを利用できます。表 19-1 は、要求の待機時間とキュー内の要求の数を示す列をリストしたものです。

表 19-1 V$QUEUE 内の待機時間と要求列

列	説明
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 V$QUEUE
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の数が判断できます。

```
SELECT COUNT(*) "Shared Server Processes"
    FROM V$SHARED_SERVER
    WHERE STATUS != 'QUIT';
```

この問合せの結果を次に示します。

```
Shared Server Processes
-----------------------
10
```

共有サーバーでのリソース競合を検出した場合は、まず、共有プールとラージ・プールを調査し、これがメモリー競合でないことを確認します。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してください。これを行うには、次の項の説明に従ってオプションのサーバー・プロセス・パラメータを変更します。

共有サーバー・プロセスの設定および変更

この項では、共有サーバー・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。

- MAX_DISPATCHERS
- MAX_SHARED_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。

- DISPATCHERS
- SHARED_SERVERS

MAX_DISPATCHERS および MAX_SHARED_SERVERS 初期化パラメータの値は、インスタンス上で実行するディスパッチャとサーバーの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開始後は変更できません。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： MAX_DISPATCHERS を設定すると、すべての DISPATCHERS のディスパッチャ値に対するディスパッチャ数が制限されます。

また、DISPATCHERS パラメータの DISPATCHER 属性および SHARED_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 文の 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。これらのパラメータに静的パラメータによって設定された制限を超える値を入力した場合は、静的パラメータの値が使用されます。

MAX_SHARED_SERVERS のデフォルト値は、SHARED_SERVERS の値によって決まります。SHARED_SERVERS が 10 より小さいか、等しい場合、MAX_SHARED_SERVERS は 20 がデフォルト値となります。SHARED_SERVERS が 10 以上の場合、MAX_SHARED_SERVERS のデフォルト値は SHARED_SERVERS の 2 倍の値となります。

共有サーバー・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、SHARED_SERVERS は作成された共有サーバー数となります。Oracle では共有サーバーの数をこの最小値未満にすることはできません。処理の間に、Oracle が共通キューの要求のアクティビティを基準にした負荷が追加共有サーバーを必要としていると判断した場合、Oracle は自動的に MAX_SHARED_SERVERS によって定義された限界値まで共有サーバーを追加します。そのため、明示的に共有サーバー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場合があります。

共有サーバー・プロセスの数が初期化パラメータ MAX_SHARED_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、MAX_SHARED_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、SHARED_SERVERS の値を変更して共有プロセスを追加できます。このパラメータの値は、初期化パラメータ・ファイルで変更することも、ALTER SYSTEM 文の SHARED_SERVERS パラメータを使用して変更することもできます。この制限を使用して施行し、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。

最大共有サーバー数を増やす

共有サーバーは、データ・アクセスを実行し、この情報をディスパッチャに渡すプロセスです。

次にディスパッチャはデータをクライアント・プロセスに送ります。すべての要求を処理できる十分な数の共有サーバーがなければキューに入れられ（V$QUEUE）、要求の処理に時間がかかります。ただし、V$QUEUE 統計を調べる前に、まず共有サーバーが不足していないかを調べるのが得策です。

システムの使用可能な RAM 容量を調べます。ps を調べるか、または他のオペレーティング・システム・ユーティリティを使用して、共有サーバーが使用するメモリー容量を調べます。使用可能な RAM 容量を共有サーバーのサイズで割ります。これにより、システムに追加できる共有サーバーの最大数がわかります。

最善の方法は、スワップを開始するまで MAX_SHARED_SERVERS パラメータを徐々に増やしていくことです。共有サーバーが原因でスワッピングが発生した場合、スワッピングが停止するまで、サーバー数を減らすか、物理 RAM の容量を増やします。オペレーティング・システムおよびアプリケーションはそれぞれ異なるため、MAX_SHARED_SERVERS の理想的な設定は試行錯誤で見つけるほかありません。

MAX_SHARED_SERVERS を変更するには、まず初期化パラメータ・ファイルを編集します。ファイルを保存し、インスタンスを再起動します。SHARED_SERVERS を MAX_SHARED_SERVERS に設定するのは、共有サーバー・プロセスの数を固定する場合に限られます。次の規則に留意してください。

- SHARED_SERVERS は、データベースが平均的負荷の場合に必要とされる予想共有サーバー数より多少大きめに設定してください。
- MAX_SHARED_SERVERS は、データベースがピーク負荷の場合に必要とされる予想共有サーバー数より多少大きめに設定してください。

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、SHARED_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。DISPATCHERS を指定した場合の SHARED_SERVERS のデフォルト設定は 1 です。

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は時間の経過につれて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伴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少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の割合を高く設定してください。

最初は、共有サーバーを少なめに割り当てるのが賢明です。必要に応じて、追加の共有サーバーは自動的に始動し、共有サーバーのアイドル状態が長すぎる場合は自動的に割当て解除されます。ただし、初期サーバーはアイドル状態であっても、常に割り当てられたままです。

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

MAX_DISPATCHERS および DISPATCHERS では、最大同時セッション数を 1 ディスパッチャ当たりの接続数で除算した値と等しい値、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、1 ディスパッチャ当たりの接続数 1,000 であれば良好なパフォーマンスが得られます。

同時共有サーバー使用による共有サーバーの追加使用の禁止

ALTER SYSTEM 文の SET オプションを使用して、アクティブな共有サーバー数を変更できます。追加のユーザーが共有サーバーにアクセスしないようにするには、SHARED_SERVERS を 0（ゼロ）に設定します。これは、共有サーバーの追加使用を一時的に禁止します。SHARED_SERVERS を正の値にリセットすると、すべての現行ユーザーに対し共有サーバーが使用可能になります。

ディスパッチャの使用を禁止するには、文 ALTER SYSTEM SHUTDOWN [IMMEDIATE] を発行します。ディスパッチャを再度オンラインにするには、DISPATCHERS パラメータで ALTER SYSTEM SET コマンドを使用します。

関連項目：

- ディスパッチャについては、『Oracle9*i* データベース・リファレンス』、特に V$DISPATCHER ビューと V$DISPATCHER_RATE ビューの説明を参照してください。
- ALTER SYSTEM 文の詳細は、『Oracle9*i* 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、『Oracle9*i* データベース管理者ガイド』を参照してください。

第 IV 部

システム関連のパフォーマンス・ツール

第 IV 部では、Oracle のシステム関連のパフォーマンス・ツールに関する情報を示します。

第 IV 部には次の章が含まれます。

- 第 20 章「データベース統計収集用の Oracle のツール製品」
- 第 21 章「Statspack の使用方法」

20

データベース統計収集用の Oracle のツール製品

この章では、パフォーマンス・データの収集が重要である理由と、用意されているツールの使用方法について説明します。

この章には次の項があります。

Oracle のツール製品の概要

効果的なデータの収集と解析は、システム・パフォーマンスの問題を識別して修正するために不可欠です。Oracle では、パフォーマンス・エンジニアがインスタンスとデータベースのパフォーマンスに関する情報を収集できるように多数のツールを提供しています。

- Oracle Enterprise Manager Diagnostics Pack はグラフィカル・ユーザー・インタフェースを備えた最も機能が豊富なパフォーマンス・ツールです。このツールは、オペレーティング・システム統計のデータ解析と収集を行います。
- Statspack および BSTAT/ESTAT は、インスタンス関連のパフォーマンス・データを収集するコマンドライン・インタフェース・ツールです。Statspack は BSTAT/ESTAT の後継で、BSTAT/ESTAT ツールに比べて機能が大幅に向上しています。
- V$ ビューは、SQL で問い合せることができます。このツールには、Oracle インスタンスのパフォーマンスに関連する動的パフォーマンス・データが含まれています。このデータは、インスタンスがシャットダウンされると失われます。

データ収集の原理

パフォーマンスの問題を効果的に診断するには、システムがうまく動作しないときに、後で比較するために確立したパフォーマンス・ベースラインを持つことが重要です。ベースライン・データ・ポイントがないと、新しい問題を識別することが非常に困難になる可能性があります。たとえば、システム上のトランザクション量の増加、トランザクション・プロファイルやアプリケーションの変更、またはユーザー数の増加などです。

Oracle Enterprise Manager、Statspack および BSTAT/ESTAT、V$ の各ビューは異なるインタフェース（GUI、コマンドライン、SQL）を持ち、それらのインタフェースで収集および報告されるデータの大部分は V$ ビューから抽出されます。V$ ビューはメモリー常駐データに基づいているため、インスタンスがシャットダウンされると、インスタンスに関連するデータが失われます。

ある日から翌日までのデータの分析を行うには、V$ ビューから表示できるデータを保存する必要があります。各インスタンスを起動すると、メモリー常駐 V$ ビューが再度初期化されます。したがって、特定の期間内に変更された内容を判別するには、パフォーマンス・データの差を計算する必要があります。これは、期間の終わりの統計値から、期間の始めの統計値を減算することで行われます。このようにすると、その期間中のインスタンスのアクティビティ、すなわち差分が得られます。さらに、各統計の差分を（たとえば、秒またはトランザクションごとに）正規化できます。

応答時間およびインスタンスで実行された操作がシステム上の代表的な負荷（すなわち、バッチ、オンラインまたはその両方）とすると、ある期間のすべての統計の差分はベースラインと考えることができます。

Oracle が提供する各ツール（V$ ビュー自体を除く）には、このデータを保存し、差分を判断するためのメカニズムがあります。

オペレーティング・システムおよびネットワークの統計を収集することも重要です。次に、これらの統計を Oracle パフォーマンス・データと相互に関連付けできます。Statspack、BSTAT/ESTAT または独自のツールのいずれかを使用する場合は、オペレーティング・システム統計情報を収集するメカニズムを考案する必要があります。Oracle Enterprise Manager には、この機能が組み込まれています。

統計の解釈

最初にパフォーマンス・データを調査するときは、統計を調べることによって潜在的な理論を構築できます。統計の解釈が正しいかどうかを確認する 1 つの方法は、他のデータとのクロスチェックを行うことです。これにより、統計またはイベントが実際に対象のものであるかどうかがわかります。

ここではいくつかの陥りやすい誤りについて説明します。

- ヒット率

  チューニングを行う場合は、問題があるかどうかを容易に判断するために、比率を計算することが一般的です。そのような率には、バッファ・キャッシュ・ヒット率、ソフト解析率およびラッチ・ヒット率があります。これらの率を、パフォーマンス・ボトルネックがあるかどうかを厳密に判断する識別子として使用しないでください。むしろ、インジケータとして使用してください。ボトルネックがあるかどうかを識別するには、他の関連する証拠を調べる必要があります。

- 時間統計のある待機イベント

  インスタンス・レベルで TIMED_STATISTICS を TRUE に設定すると、Oracle サーバーはすでに使用できる待機カウントの他にイベントの待機時間を収集します。このデータは、イベントの合計待機時間をパフォーマンス・データ収集間の総経過時間と比較する場合に役立ちます。たとえば、待機イベントが 2 時間の期間のうちのわずか 30 秒であれば、このイベントが待機時間別に順序付けされるときに最高ランクの待機イベントであっても、このイベントを調べて得られるものはほとんどありません。ただし、イベントが 45 分間のうちの 30 分であれば、そのイベントは調べる価値があります。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

- Oracle 統計とその他の要因の比較

  統計を調べる場合、統計に価値があるかどうかに影響を与える他の要因を考慮することが重要です。そのような要因には、ユーザー負荷やハードウェア能力があります。待機時間が 45 分間のスナップショットのうちの 30 分間であったイベントでも、システム上に 2000 人のユーザーがいて、ホスト・ハードウェアが 64 ノード・マシンであった場合は問題とはみなされないことがあります。

- 時間統計のない待機イベント

  TIMED_STATISTICS が正しくない場合、イベントの待機時間は使用できません。したがって、各イベントを待機した時間数で待機イベントを順序付けることのみ可能です。最大待機回数を持つイベントが潜在的なボトルネックを示すことがありますが、主要なボトルネックとは言えません。これはイベントを長時間待つ場合に発生する可能性がありますが、そのイベントを待つ合計待機時間は短時間です。その逆も成り立ちます。待機回数が少ないイベントは、その待機時間が合計待機時間の大きな割合を占めている場合に問題になることがあります。比較のために使用する待機時間がなければ、待機イベントが実際に重要かどうかの判断は困難です。

- アイドル待機イベント

  Oracle では、いくつかの待機イベントを使用して、Oracle サーバー・プロセスがアイドル状態であるかどうかを示します。一般に、これらのイベントはパフォーマンスの問題を調べるときは有効でなく、待機イベントを調べるときは無視する必要があります。

- 計算済み統計

  計算済み統計（割合、トランザクションごとに正規化された統計、比率など）を解釈する場合は、計算済み統計を実際の統計カウントと相互検査することが重要です。その検査で、導出レートが実際に重要かどうかが確認されます。小さい統計カウントは通常、異常な比率を割り引くことができます。たとえば、最初の検査で、ソフト解析率 50% は、一般に潜在的なチューニング領域を示します。ただし、データ収集期間にハード解析が 1 つとソフト解析が 1 つのみあった場合、この領域が重要な領域でないことを統計カウントが示していても、ソフト解析率は 50% になります。この場合、統計カウントが低いため、比率は重要ではありません。

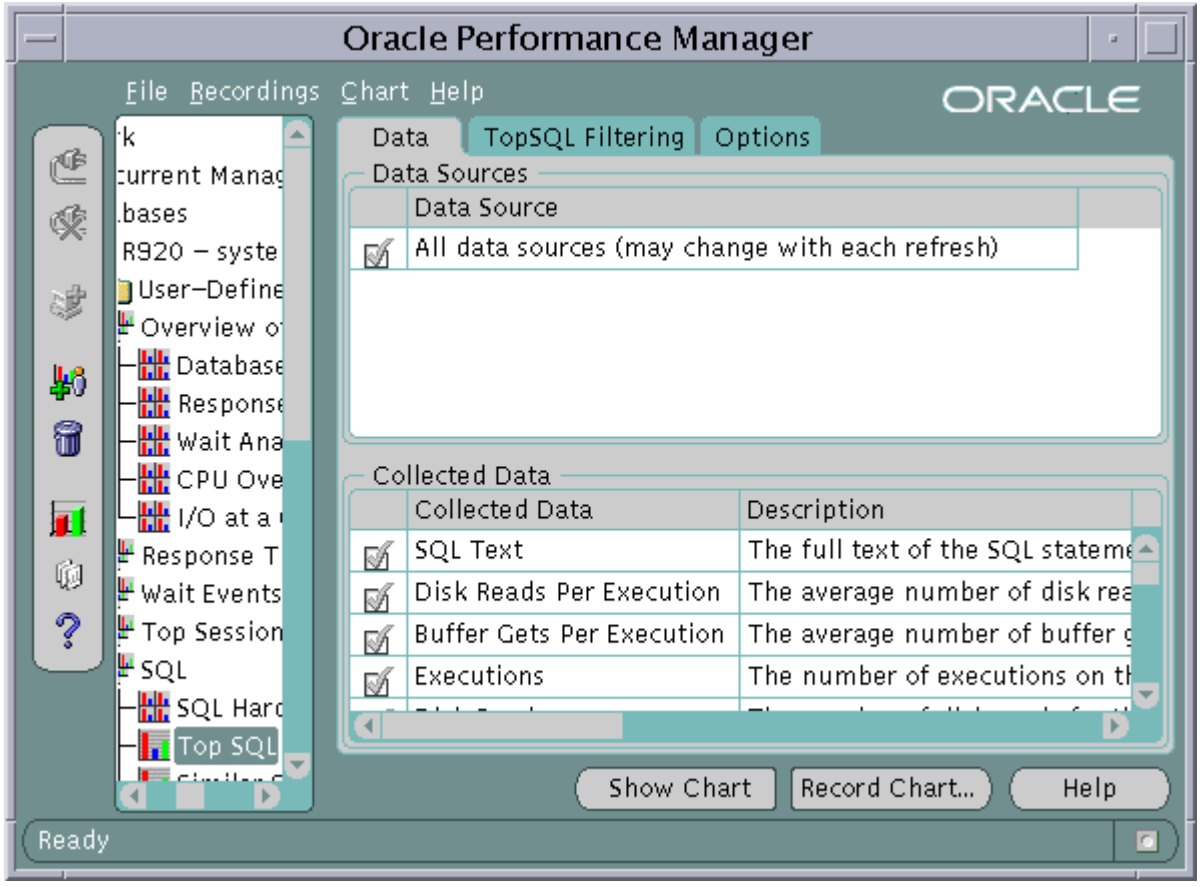
  関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

Oracle Enterprise Manager Diagnostics Pack

Oracle Enterprise Manager (EM) Diagnostics Pack は、インスタンスのパフォーマンス・データの他に関連するオペレーティング・システム、中間層およびアプリケーションのパフォーマンス・データを収集するので、徹底的な診断が可能です。

この Diagnostics Pack は、このパフォーマンス・データを自動的に分析し、グラフィカル・インタフェースに表示し、アラートを使用してただちにパフォーマンスの問題に注意を向かせます。問題が検出されたときは、電子メールまたはページで自動的に通知できます。Oracle Enterprise Manager には、ガイド付きドリルダウンとエキスパート・アドバイスを使用して、素早くパフォーマンスの問題を解決するための統合的な診断手法も含まれています。図 20-1 は、Oracle Performance Manager の画面です。

図 20-1 Oracle Performance Manager



EM を使用して、獲得したデータを個別のパフォーマンス・リポジトリ・データベース内に格納することもできます。複数のデータベースのパフォーマンス・データを同じリポジトリに格納できます。

EM Intelligent Agent データ収集サービスは、スケジュール・ベースでパフォーマンス・データを収集します。単一エージェントは、すべての Oracle データベースとターゲット・ノードのオペレーティング・システムのデータ収集を管理できます。データは、パフォーマンス報告のために履歴データ・リポジトリに自動的に格納されます。リポジトリに格納されたデータを使用して、データベース負荷、キャッシュの割当てと効率性、リソースの競合、影響の強い SQL など、データベース・パフォーマンスの複数の側面を分析できます。

履歴パフォーマンス・データの HTML レポートは、EM コンソールから生成できます。これらのレポートは、データベース・システムの使用状況とパフォーマンスの総合的な分析を提供します。この分析には、ブラウザから簡単にアクセスおよびナビゲートできます。EM は、線グラフや棒グラフなどを使用して、これらのパフォーマンス測定値のサブセットを監視するためのグラフィカル・リアルタイム Performance Overview も提供します。

Performance Overview チャートを使用すると、パフォーマンス・データにドリルインしてパフォーマンス・ボトルネックの原因を探し出すことによって、存在するパフォーマンスの問題をトラブルシューティングできます。たとえば、大量のソート・アクティビティを受け持つセッションおよび対応する SQL にドリルダウンして、ただちにメモリー・ソートのパーセントの低下を調べることができます。問題のコンテキストで SQL 診断ツールを起動することによって、このプロセスで発見されたインパクトの強い SQL 文をさらに調べることができます。

関連項目：

- 『Oracle Enterprise Manager 概要』
- 『Oracle Diagnostics Pack スタート・ガイド』

Statspack

Statspack は、より多くの情報を収集する点、また後で報告と分析に使用できるパフォーマンス統計を Oracle の表に永続的に格納する点において、よく知られている UTLBSTAT/UTLESTAT チューニング・スクリプトとは基本的に異なります。収集したデータは提供されたレポートを使用して分析できます。このレポートには、「instance health and load」サマリー・ページ、リソースを多く使用する SQL 文および従来の待機イベントと初期化パラメータが含まれています。

関連項目： 第 21 章「Statspack の使用方法」

V$ パフォーマンス・ビュー

V$ ビューは、すべての Oracle パフォーマンス・チューニング・ツールで使用されるパフォーマンス情報ソースです。V$ ビューは、インスタンスの起動時に初期化されたメモリー構造に基づいています。メモリー構造および構造を表すビューは、インスタンスが存続する間、Oracle により自動的にメンテナンスされます。

> **関連項目：** 第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」

Oracle ツールを使用してパフォーマンス・データを収集しない場合は、独自のツールを開発する必要があります。必要なパフォーマンス・ビューからデータをディスク上に保存して、そのデータを分析し、収集した他のデータと比較できるようにする必要があります。複数の収集があるため、各収集を識別するためのキーが必要です。この方法は、Statspack で使用される方法に非常に類似しています。

次に、パフォーマンス・データを単一 V$ ビューから Oracle 表に保存する方法の例を示します。独自の収集ツールを実装するには、すべての不可欠な V$ ビューに対し、類似する収集メカニズムを実行する必要があります。

注意： Oracle Enterprise Manager Diagnostics Pack または Statspack を使用してパフォーマンス・データを収集することをお薦めします。これらのツールは、パフォーマンスの分析に必要なすべてのデータを収集するように設計されています。

例 ― ファイル I/O データの保存

次の例では、収集データを格納するテーブルを作成します。このテーブルには、すべての V$FILESTAT 列があり、収集 ID 列と収集日付列もあります。最初の 2 つの収集について、この表で差分を報告するサンプル SQL 文も含まれています。

表を作成し、最初のデータ収集を挿入します。

```
CREATE TABLE coll_filestat AS
  SELECT 1       coll_id       -- collection number
       , sysdate coll_date     -- collection date
       , fs.*
   FROM V$FILESTAT fs;

ALTER TABLE coll_filestat add
   (CONSTRAINT coll_filestat PRIMARY KEY (coll_id, file#));
```

この間隔の終わりに、第 2 の収集を挿入します。

```
INSERT INTO coll_filestat
SELECT 2                        -- collection number
      , sysdate                 -- collection date
      , fs.*
 FROM V$FILESTAT fs;
```

> **注意：** 他の収集を挿入するには、収集 ID キーを一意に保つ必要があります。これは、順序番号で行うことができます（順序番号には、1 つの収集で獲得されたすべての V$ ビューの値が必要です）。

I/O の多い表領域の問合せを行うには、次のようにします。

```
SELECT t.tablespace_name
      ,   SUM(fs2.phyrds-fs1.phyrds)
        / MAX(86400*(fs2.coll_date-fs1.coll_date)) "Rd/sec"
      ,   SUM(fs2.phyblkrd-fs1.phyblkrd)
        / MAX(86400*(fs2.coll_date-fs1.coll_date)) "Blk/sec"
      ,   SUM(fs2.phywrts-fs1.phywrts)
        / MAX(86400*(fs2.coll_date-fs1.coll_date)) "Wr/sec"
      ,   SUM(fs2.phyblkwrt-fs1.phyblkwrt)
        / MAX(86400*(fs2.coll_date-fs1.coll_date)) "Blk/sec"
   FROM coll_filestat fs1, coll_filestat fs2, dba_data_files t
  WHERE fs2.file#   = fs1.file#
    AND fs2.coll_id = fs1.coll_id + 1
    AND t.file_id   = fs2.file#
  GROUP BY t.tablespace_name
  ORDER BY sum(fs2.phyblkrd+fs2.phyblkwrt-fs1.phyblkrd-fs1.phyblkwrt) DESC;
```

前述の SELECT 文の出力例を次に示します。

```
TABLESPACE_N Rd/sec Blk/sec Wr/sec Blk/sec
------------ ------ ------- ------ -------
AP_T_02       287.1  2245.7     .0      .0
PO_T_01       313.5   650.6     .2      .2
RECEIVABLE_T  401.0   613.8    2.4     2.4
INV_T_01      154.3   155.3     .0      .0
APPLSYS_T_01   63.3   139.6     .4      .4
PA_T_01       102.3   102.3     .0      .0
SO_I_01        63.4    63.4   34.5    34.5
TEMP            2.3    45.0    1.9    47.0
RECEIVABLE_I   73.0    73.0     .1      .1
AP_T_03        69.3    69.3     .0      .0
RECEIVABLE_I   65.1    65.1    1.9     1.9
SO_T_01        54.0    57.8    2.9     2.9
```

```
SYSTEM          45.2    59.0     .3      .3
PER_T_01        48.0    58.7     .0      .0
AP_T_01         12.9    51.0     .2      .2
SO_T_03         43.0    43.0    1.2     1.2
PER_I_01        30.8    30.8     .0      .0
FA_T_01         22.3    22.3     .0      .0
INV_I_01        20.7    20.7     .7      .7
PO_I_01         19.5    19.5     .7      .7
GSR_T_01        19.2    19.2     .4      .4
INV_I_03        18.3    18.3     .0      .0
ROLL_01          1.4     1.4   14.7    14.7
PA_I_01         14.3    14.3     .2      .2
```

21

Statspack の使用方法

この章では、Statspack のインストール、構成、使用方法について説明します。

この章には次の項があります。

注意： Statspack は、8.1.5 以下のリリースではサポートされていません。1 つの PERFSTAT ユーザー・アカウントで、複数のデータベースからデータを格納する操作は、現在サポートされていません。

関連項目： Oracle データベースとともにインストールされた SPDOC.TXT ファイルについては、21-29 ページの「Statspack マニュアル」を参照してください。

Statspack の概要

データベースをチューニングする際には、後でシステムの動作状態が悪い場合に比較するためのパフォーマンス・ベースラインを確立しておくことが重要です。ベースライン・データ・ポイントは、新しいパフォーマンス上の問題を診断するときにチェックする要因を識別する際に役立ちます。チェックする要因は次のとおりです。

- システム上のトランザクション量が増えましたか？
- トランザクション・プロファイルまたはアプリケーションが変更されましたか？
- ユーザー数が増えましたか？

Statspack パッケージは、パフォーマンス・データの収集、自動化、格納および表示ができる SQL、PL/SQL および SQL*Plus スクリプトのセットです。Statspack は、パフォーマンス統計を Oracle 表に永続的に格納し、後でレポート作成および分析用に使用できます。収集したデータは提供された Statspack レポートを使用して分析できます。このレポートには、instance health and load サマリー・ページ、リソースを多く使用する SQL 文および従来の待機イベントと初期化パラメータが含まれています。

> **関連項目：** Oracle では、統計の収集および分析を行う GUI ツールを含む診断用パックを提供しています。20-5 ページの「Oracle Enterprise Manager Diagnostics Pack」を参照してください。

Statspack と BSTAT/ESTAT の比較

Statspack は、既存の UTLBSTAT/UTLESTAT パフォーマンス・スクリプトとは次の点で異なります。

- Statspack は高度なリソース SQL などより多くのデータを収集します。
- Statspack は、キャッシュ・ヒット率、割合、トランザクション統計など、パフォーマンス・チューニングを行うときに役立つ多数の比率をあらかじめ計算します これらの比率の多くは、BSTAT/ESTAT を使用するときに手動計算を行う必要があります。
- Statspack では、パフォーマンス統計は、PERFSTAT のユーザーが所有する永続的な表を使用して格納されます。そのつど表を作成および削除せずに、データが既存の表に挿入されるため、履歴データの比較がさらに容易になります。
- Statspack は、レポート生成からデータの収集を分離します。データはスナップショットが取られるときに収集されます。その際、パフォーマンス・エンジニアはパフォーマンス・レポートを実行し、収集されたデータを表示します。

> **注意：** ここで使用する**スナップショット**という用語は、1 回で収集され、一意の ID で識別されるパフォーマンス統計のセットを示します。この統計には、スナップショット番号（SNAP_ID）が含まれています。この種類のスナップショットは、Oracle のスナップショット・レプリケーション・テクノロジとは関係がありません。

- Statspack では、DBMS_JOB またはオペレーティング・システム・ユーティリティを使用して収集タスクをスケジュールし、データ収集を簡単に自動化できます。
- Statspack は、COMMIT または ROLLBACK でトランザクションを終了することを考えるため、トランザクション数を user commits + user rollbacks として計算します。BSTAT/ESTAT は COMMIT のみでトランザクションを終了することを想定するため、transaction = user commits であると仮定します。このため、Statspack と BSTAT/ESTAT とで 1 トランザクション当たりの統計を比較すると、比率が大きく異なる可能性があります。

注意： BSTAT/ESTAT を Statspack とともに実行する場合、その両方を同じユーザーとして実行しないでください。STATS$WAITSTAT 表と表名の競合があります。

Statspack の動作

Statspack インストール・スクリプトを実行すると、ユーザー PERFSTAT が自動的に作成されます。PERFSTAT は、Statspack パッケージに必要なオブジェクトをすべて所有していますが、パフォーマンス・チューニングに必要な V$ ビューにおける問合せのみの権限が付与されます。

Statspack ユーザーは、**スナップショット**の概念、すなわち、単一のパフォーマンス・データ収集が理解できるようになります。作成された各スナップショットは、そのスナップショットが作成された時点で生成された一意の番号のスナップショット ID で識別されます。新しい収集が行われるたびに、新しい SNAP_ID が生成されます。

SNAP_ID は、データベース識別子（DBID) およびインスタンス番号（INSTANCE_NUMBER）とともに、スナップショットの一意のキーを導出します。この一意の組合せを使用すると、Oracle Real Application Clusters データベースの複数のインスタンスを同じ表に格納できます。

スナップショットが作成された後、パフォーマンス・レポートを実行できます。このレポートは、BSTAT/ESTAT レポートの場合とほぼ同様に、開始スナップショットおよび終了スナップショットの ID の入力を要求し、この 2 つのスナップショット期間のインスタンスに関するアクティビティを計算します。比較すると、指定された第 1 の SNAP_ID は BSTAT の実行と同等であるとみなすことができます。さらに、指定された第 2 の SNAP_ID は ESTAT と同等であるとみなすことができます。もともと 2 つの静的データ・ポイントのみを比較できる BSTAT/ESTAT と異なり、このレポートでは指定された 2 つのスナップショットを比較できます。

Statspack のデータベース領域要件の構成

デフォルトの初期エクステント・サイズおよび次のエクステント・サイズは、すべての Statspack の表と索引について 100K、1MB、または 5MB のいずれかです。Statspack をインストールするには、約 64MB が必要です。

Statspack パッケージで必要なデータベース領域の量は、スナップショットの回数、データベースとインスタンスのサイズおよび収集された構成可能なデータ量によって異なります。したがって、各サイトで正確な一般的な STORAGE 句と領域利用率の予測を行うことは困難です。

- ディクショナリ管理の表領域内にパッケージをインストールする場合は、作成されたオブジェクトで使用される領域を監視し、必要に応じてセグメントの STORAGE 句を調整してください。
- ローカル管理の表領域内にパッケージをインストールする場合、記憶特性は自動的に管理されているため STORAGE 句は必要ありません。

Statspack のインストール

Statspack をインストールする方法は、次の 2 つです。

- 対話型での Statspack のインストール
- バッチ・モードでの Statspack のインストール

バッチ・モードは、PERFSTAT ユーザーのパスワード、デフォルト表領域および一時表領域のプロンプトを必要としないときに便利です。

対話型での Statspack のインストール

インストールの第 1 ステップでは、PERFSTAT ユーザーを作成します。このユーザーは、Statspack 表、制約および Statspack パッケージを含む作成されたすべての PL/SQL コードおよびデータベース・オブジェクトを所有します。インストール中に、PERFSTAT ユーザーのパスワード、デフォルト表領域および一時表領域をプロンプトされます。デフォルト表領域は、表や索引などのすべての Statspack オブジェクトの作成に使用します。一時表領域は、ソート型のアクティビティに使用します。

> **関連項目：** 一時表領域の詳細は、『Oracle9*i* データベース概要』を参照してください。

注意：

- パスワードは必須で、機密扱いの必要があります。
- PERFSTAT ユーザーの DEFAULT 表領域または TEMPORARY 表領域に SYSTEM 表領域を指定しないでください。SYSTEM を指定すると、インストールは問題を指定するエラーで異常終了します。SYSTEM 表領域を使用して、統計データの格納やソートを行うことはお薦めしません。データを格納する場合は TOOLS 表領域を使用し、ソートにはインスタンスの TEMP 表領域を使用してください。このエラーをリカバリするには、スクリプトの削除（SPDROP.SQL）を実行し、次にインストールを再実行します。
- インストール中は、DBMS_SHARED_POOL パッケージと DBMS_JOB PL/SQL パッケージが作成されます。DBMS_SHARED_POOL は、共有プール内に Statspack パッケージを確保します。DBMS_JOB は、DBMS_JOB を使用して、定期的なスナップショットを自動的にスケジュールすると仮定して作成されます。

Statspack パッケージをインストールするには、ORACLE_HOME/rdbms/admin ディレクトリに切り替えるか、インストール・スクリプト SPCREATE.SQL を呼び出すときに、ORACLE_HOME/rdbms/admin ディレクトリを完全に指定します。

Statspack をインストールするには、次の手順を実行します。

- SQL*Plus を起動します。
- SYSDBA 権限を持つユーザーとして接続します。次に例を示します。

```
SQL>  CONNECT / AS SYSDBA
```

- SPCREATE.SQL スクリプトを実行します。
  - UNIX プラットフォームの場合は、次のように入力します。

```
SQL>  @?/rdbms/admin/spcreate
```

  - Windows プラットフォームの場合は、次のように入力します。

```
SQL>  @%ORACLE_HOME%¥rdbms¥admin¥spcreate
```

- PERFSTAT ユーザーのパスワード、デフォルト表領域および一時表領域がプロンプトされた場合、適切な情報を入力します。

SPCREATE.SQL インストール・スクリプトは、次のスクリプトを自動的に実行します。

- SPCUSR.SQL: ユーザーを作成し、権限を付与します。
- SPCTAB.SQL: 表を作成します。
- SPCPKG.SQL: パッケージを作成します。

インストール中にエラーが発生しなかったかどうかを確認するために、SPCUSR.LIS、SPCTAB.LIS および SPCPKG.LIS の出力ファイルをチェックします。次に例を示します。

```
ORACLE_HOME/bin/spcusr.lis
ORACLE_HOME/bin/spctab.lis
ORACLE_HOME/bin/spcpkg.lis
```

バッチ・モードでの Statspack のインストール

Statspack をバッチ・モードでインストールするには、SPCREATE.SQL を実行する前に、デフォルトおよび一時表領域を指定する SQL*Plus 変数に値を割り当てる必要があります。これらの変数は次のとおりです。

- DEFAULT_TABLESPACE: デフォルト表領域の場合
- TEMPORARY_TABLESPACE: 一時表領域の場合
- PERFSTAT_PASSWORD: PERFSTAT ユーザーの場合

たとえば、UNIX では、次のようになります。

```
SQL>  CONNECT / AS SYSDBA
SQL>  define default_tablespace='TOOLS'
SQL>  define temporary_tablespace='TEMP'
SQL>  define perfstat_password='my_perfstat_password'
SQL>  @?/rdbms/admin/spcreate
```

SPCREATE.SQL を実行する場合は、変数によって入力される情報をプロンプトしません。

注意： セットアップが完了した後、セキュリティのために PERFSTAT ユーザーのパスワードを変更します。

Statspack の使用方法

この項では、次の項目について説明します。

- Statspack スナップショットの作成
- 自動統計収集
- Statspack パフォーマンス・レポートの実行
- Statspack 内に収集されるデータの量の構成
- 待機イベントに使用される時間単位
- イベントのタイミング
- Statspack パフォーマンス・データの管理および共有
- Statspack 使用時の Oracle Real Application Clusters の考慮事項

Statspack スナップショットの作成

スナップショットを作成する最も簡単な対話型の方法は、PERFSTAT ユーザーとして SQL*Plus にログインし、プロシージャ STATSPACK.SNAP を実行します。その例を次に示します。

```
SQL>  CONNECT perfstat/my_perfstat_password
SQL>  EXECUTE statspack.snap;
```

注意： Oracle Real Application Clusters 環境では、データを収集するインスタンスに接続する必要があります。

このようなスナップショットを取得すると、Statspack 表にパフォーマンス統計の現在の値が格納されます。このスナップショットは、後から取得した別のスナップショットと比較するためのベースラインとして使用できます。

パフォーマンス分析を向上させるために、TIMED_STATISTICS 初期化パラメータを TRUE に設定します。これにより、収集するデータの中に重要なタイミング情報が組み込まれます。TIMED_STATISTICS パラメータは、ALTER SYSTEM 文を使用して動的に変更できます。タイミング・データは重要であり、通常は、Oracle サポートがパフォーマンスの問題を診断する場合に要求されます。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM や ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

一般に、収集フェーズと報告段階（ベンチマーク実行時など）を自動化するには、作成されたスナップショットの snap_id を知る必要が生じる場合があります。スナップショットを作成して snap_id を表示するには、STATSPACK.SNAP ファンクションを呼び出します。

例 21-1 SQL*Plus での snap ファンクションのコール

次の無名 PL/SQL ブロックを使用します。

```
SQL> variable snap number;
SQL> begin   :snap := statspack.snap;   end;
  2  /
PL/SQL procedure successfully completed.
SQL> print snap
      SNAP
----------
        12
```

自動統計収集

ある日、週、年と、次の日、週、年とのパフォーマンスの比較を行うには、一定期間にわたって作成された複数のスナップショットが必要です。スナップショットを収集する最良の方法は、一定の時間間隔での収集を自動化することです。次のオプションがあります。

- データベース内で、Oracle の DBMS_JOB プロシージャを使用してスナップショットのスケジューリングを行います。
- スナップショットなどのスケジューリングを行うには、UNIX の cron などのオペレーティング・システムのユーティリティを使用します。Windows プラットフォームの場合は、Windows NT の at ユーティリティまたは Windows 2000 の「システム ツール」>「タスク」を使用できます。

DBMS_JOB を使用した統計の収集

Oracle で自動化された方法で統計収集などの別のタスクをスケジューリングしたり、実行するには、DBMS_JOB パッケージを使用します。これを行う方法に関するスクリプト例は、毎時スナップショットのスケジューリングを行う SPAUTO.SQL に定時に示されます。

システムの OLTP またはバッチのピーク負荷を反映するように、標準的な時間間隔でスナップショットのスケジューリングをする場合があります。たとえば、OLTP 負荷について午前 9 時、午前 10 時、午前 11 時、正午および午後 6 時にスナップショットを作成し、バッチ・ウィンドウについては深夜 0 時および午前 6 時にスナップショットを作成するとします。

DBMS_JOB を使用してスナップショットのスケジューリングを行う場合、ジョブが自動的に実行されるようにするには、初期化ファイルで JOB_QUEUE_PROCESSES 初期化パラメータを 0（ゼロ）より大きい値に設定する必要があります。

注意： Oracle Real Application Clusters 環境で SPAUTO.SQL を使用する場合、SPAUTO.SQL スクリプトをクラスタ内の各インスタンスで 1 回実行する必要があります。同様に、各インスタンスに JOB_QUEUE_PROCESSES パラメータを設定する必要があります。

統計収集の時間間隔の変更

統計収集の時間間隔を変更するには、DBMS_JOB.INTERVAL プロシージャを使用します。次に例を示します。

```
EXECUTE DBMS_JOB.INTERVAL(job_number,'SYSDATE+(1/48)');
```

'SYSDATE+(1/48)' により、統計が 1/48 時間ごと、つまり 30 分おきに収集され、*job_number* が実行する特定のジョブを参照します。

ジョブが即時に実行されるように強制するには、次のようにします。

```
EXECUTE DBMS_JOB.RUN(job_number);
```

指定したジョブを削除するには、次のようにします。

```
EXECUTE DBMS_JOB.REMOVE(job_number);
```

関連項目： DBMS_JOB パッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

Statspack パフォーマンス・レポートの実行

スナップショットが作成された後、パフォーマンス・レポートを生成できます。レポートを生成する SQL スクリプトは、開始スナップショット ID、終了スナップショット ID、およびレポート名をプロンプトします。Statspack パッケージには 2 つのレポートが含まれています。

- まず、Statspack レポート SPREPORT.SQL を実行します。これはインスタンスのパフォーマンスのすべての側面を網羅する、インスタンスの一般的な状況レポートです。このレポートは、BSTAT/ESTAT レポートと同様に、2 つのスナップショットのすべての統計の比率および差異を計算し出力します。
- このインスタンス・レポートを調べた後、単一 SQL 文（ハッシュ値で識別可能）で SQL レポート SPREPSQL.SQL を実行します。SQL レポートでは、単一 SQL 文に関連するデータに関するレポートのみを行います。

注意： 開始スナップショットと終了スナップショットが異なるインスタンス・スタートアップから作成された場合、開始スナップショットと終了スナップショットを指定することは正しくありません。つまり、開始スナップショットが作成されたときから終了スナップショットが作成されたときまでインスタンスをシャットダウンしないでください。

これが必要なのは、Statspack がデータを収集するために問い合せる、データベースの動的パフォーマンス表がメモリーに常駐しているためです。したがって、データベースをシャットダウンすると、パフォーマンス表の値が 0（ゼロ）にリセットされます。Statspack は終了スナップショット統計から開始スナップショット統計を差し引くため、出力結果は無効です。シャットダウンの間で作成された開始スナップショットと終了スナップショットをレポートで指定すると、レポートは対応するエラーを表示します。

データ収集はレポート生成とは別に行われるため、任意のデータ・ポイントのレポートが作成できるという柔軟性があります。たとえば DBA が、提供された自動化スクリプトを使用して、1 時間ごとの正時のデータ収集を自動化する必要があるとします。3 時間分のデータ・ウィンドウを見て調査した方がよいと考えられるパフォーマンスの問題が後から発生した場合、必要な開始点と終了点を指定してレポートを実行するだけで済みます。

Statspack レポートの実行

2 つの期間のインスタンス全体の統計の変化を調べるには、PERFSTAT ユーザーの接続中に SPREPORT.SQL スクリプトを実行します。SPREPORT.SQL スクリプトは、Oracle ホームの rdbms/admin ディレクトリに配置されています。

> **注意：** Oracle Real Application Clusters 環境では、レポート対象のインスタンスに接続する必要があります。

レポートを実行すると、次の内容をプロンプトします。

- 開始スナップショット ID
- 終了スナップショット ID
- 作成するレポート・テキスト・ファイルの名前

> **注意：** シリアル番号と session_id（SID）は、開始スナップショットと終了スナップショットで同一にする必要があります。スナップショット ID 行の間の空白行は、インスタンスが再起動（シャットダウン / 起動）されている間に、シリアル番号が変更されたことを意味します。したがって、空白行により、Statspack レポートを実行する場合に同時に使用できない開始スナップショットと終了スナップショットを識別できます。

例 21-2 は、レポートを実行する SQL コマンドとレポート出力例の一部を示したものです。

例 21-2 プロンプトによる Statspack レポートの作成

```
SQL>  connect perfstat/my_perfstat_password
SQL>  @?/rdbms/admin/spreport
```

Windows プラットフォームの場合、レポートを実行するコマンドは次のとおりです。

```
SQL>  @%ORACLE_HOME%¥rdbms¥admin¥spreport
```

出力例 :

```
SQL>  connect perfstat/my_perfstat_password
Connected.
SQL>  @?/rdbms/admin/spreport

DB Id       DB Name      Inst Num Instance
----------- ------------ -------- ------------
2618106428  PRD1                1 prd1
Completed Snapshots
                           Snap                      Snap
Instance     DB Name         Id   Snap Started    Level Comment
------------ ------------ ----- ----------------- ----- ----------------
prd1         PRD1             1 11 May 2000 12:07     5
                              2 11 May 2000 12:08     5

Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap: 1
Begin Snapshot Id specified: 1

Enter value for end_snap: 2
End   Snapshot Id specified: 2

Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is sp_1_2  To use this name, press <return> to
continue, otherwise enter an alternative. Enter value for report_name: <press return
or enter a new name>

Using the report name sp_1_2
```

レポートは前にスクロールし、指定されたファイルにも書き込まれます。次に例を示します。

```
ORACLE_HOME/bin/sp_1_2.lis
```

プロンプトを表示せずにレポートを実行するには、SPREPORT を実行する前に開始スナップ ID、終了スナップ ID およびレポート名を指定する SQL*Plus 変数に値を割り当てます。

これらの変数は次のとおりです。

- BEGIN_SNAP: 開始スナップショット ID を指定します。
- END_SNAP: 終了スナップショット ID を指定します。
- REPORT_NAME: レポート出力名を指定します。

例 21-3 プロンプトを使用しない場合の Statspack レポートの作成（UNIX）

```
SQL>  connect perfstat/my_perfstat_password
SQL>  define begin_snap=1
SQL>  define end_snap=2
SQL>  define report_name=batch_run
SQL>  @?/rdbms/admin/spreport
```

SPREPORT.SQL を実行中の場合は、変数によって入力される情報をプロンプトしません。

SQL レポートの実行

インスタンス・レポートを調べると、さらに詳しく調べる必要のある高負荷の SQL 文が見つかることがあります。SQL レポート SPREPSQL.SQL は、統計、完全な SQL テキストおよびその文に関連する SQL プランに関する情報（レベル 6 のスナップショットが作成された場合）を表示します。

> **関連項目：** レベルについては、21-17 ページの「スナップショット・レベル」を参照してください。

報告される SQL 文は、文の SQL テキストが数値で表現されているハッシュ値で識別されます。各文のハッシュ値は、インスタンス・レポートの SQL セクションに文ごとに表示されます。

PERFSTAT ユーザーとして接続している間に SPREPSQL.SQL スクリプトを実行します。このレポートは、Oracle ホームの rdbms/admin ディレクトリ内にあります。

> **注意：** Oracle Real Application Clusters 環境では、レポート対象のインスタンスに接続する必要があります。

SPREPSQL.SQL レポートは、次の項目をプロンプトします。

- 開始スナップショット ID
- 終了スナップショット ID
- SQL 文のハッシュ値
- 作成するレポート・テキスト・ファイルの名前

例 21-4 SPREPORT.SQL の出力例

```
SQL>  connect perfstat/my_perfstat_password
Connected.
SQL>  @?/rdbms/admin/sprepsql

   DB Id    DB Name      Inst Num Instance
----------- ------------ -------- ---------
 2618106428 PRD1                1 prd1

Completed Snapshots

                           Snap                     Snap
Instance     DB Name         Id   Snap Started    Level Comment
------------ ------------ ----- ----------------- ----- -------
prd1         PRD1            37 02 Mar 2001 11:01     6
                             38 02 Mar 2001 12:01     6

                             39 08 Mar 2001 09:01     5
                             40 08 Mar 2001 10:02     5

Specify the Begin and End Snapshot Ids
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
Enter value for begin_snap: 39
Begin Snapshot Id specified: 39

Enter value for end_snap: 40
End   Snapshot Id specified: 40

Specify the Hash Value
~~~~~~~~~~~~~~~~~~~~~~
Enter value for hash_value: 1988538571
Hash Value specified is: 1988538571

Specify the Report Name
~~~~~~~~~~~~~~~~~~~~~~~
The default report file name is sp_39_40_1988538571.  To use this name,
press <return> to continue, otherwise enter an alternative.
Enter value for report_name:

Using the report name sp_39_40_1988538571
```

レポートは前にスクロールし、指定されたファイルにも書き込まれます。次に例を示します。

```
sp_39_40_1988538571.lis
```

SPREPSQL.SQL スクリプトは、SQL レポートをバッチ・モードで実行できます。プロンプトを表示せずにレポートを実行するには、開始スナップ ID、終了スナップ ID、ハッシュ値、およびレポート名を指定する SQL*Plus 変数に値を割り当てた上で SPREPSQL.SQL スクリプトを実行します。これらの変数は次のとおりです。

- BEGIN_SNAP: 開始スナップショット ID を指定します。
- END_SNAP: 終了スナップショット ID を指定します。
- HASH_VALUE: ハッシュ値を指定します。
- REPORT_NAME: レポート出力名を指定します。

例 21-5 バッチ・モードでの SPREPORT.SQL の実行

```
SQL>  connect perfstat/my_perfstat_password
SQL>  define begin_snap=39
SQL>  define end_snap=40
SQL   define hash_value=1988538571
SQL>  define report_name=batch_sql_run
SQL>  @?/rdbms/admin/sprepsql
```

SPREPSQL.SQL を実行中の場合は、変数によって入力される情報をプロンプトしません。

PERFSTAT スキーマに関するオプティマイザ統計の収集

パフォーマンス・レポート実行中のパフォーマンスを最大にするには、PERFSTAT で所有されている表と索引に関するオプティマイザ統計を収集します。この収集は、PERFSTAT の表のデータ・ボリュームに大きな変更がある場合に行う必要があります。

PERFSTAT スキーマのオプティマイザ統計を収集するには、DBMS_STATS または DBMS_UTILITY を使用し、PERFSTAT ユーザーを指定します。その例を次に示します。

```
EXECUTE DBMS_STATS.GATHER_SCHEMA_STATS(OWNNAME=>'PERFSTAT',CASCADE=>TRUE);
```

または

```
EXECUTE DBMS_UTILITY.ANALYZE_SCHEMA('PERFSTAT','COMPUTE');
```

Statspack 内に収集されるデータの量の構成

スナップショット・レベルおよび指定されたしきい値は、Statspack で獲得されるデータの量に影響を与えます。情報の収集量を変更する場合は、別のスナップショット・レベルを指定します。スナップショット・レベルが高いほど、多くのデータが収集されます。インストール時に設定されるデフォルトのレベルはレベル 5 です。

代表的な用途として、レベル 5 はほとんどのサイトで効果的です。レベル 6 のスナップショットを使用すると有益な場合もあります。次のような例があります。

- 最初のベースラインを取得する場合
- 新しいアプリケーションまたはアプリケーションの変更内容がインストールされて、新しいベースラインを取得する場合
- オプティマイザ統計を収集した後

関連項目： 21-17 ページ「スナップショット・レベル」

スナップショット SQL のしきい値

スナップショット・レベルの他にも構成可能なその他のパラメータがあります。これらのパラメータは、SQL 文でデータを収集する時のしきい値として使用します。データは、指定されたしきい値に達した SQL 文で収集されます。パッケージで使用されるスナップショット・レベルおよびしきい値の情報は、STATS$STATSPACK_PARAMETER 表に格納されます。

スナップショット・レベルおよび SQL しきい値のデフォルト値の変更

スナップショットの作成に使用するデフォルト・パラメータは、インスタンスのワークロードに合うように変更できます。Statspack MODIFY_STATSPACK_PARAMETER または SNAP プロシージャで、単純に適切なパラメータと新規の値を使用します。次に例を示します。

```
SQL>  EXECUTE STATSPACK.SNAP(i_ucomment=>'this is a temporary commment');
SQL>  EXECUTE STATSPACK.MODIFY_STATSPACK_PARAMETER(i_ucomment=>'this is a commment
that is saved');
```

MODIFY_STATSPACK_PARAMETER プロシージャおよび SNAP プロシージャに渡すことのできるパラメータは、21-20 ページの表 21-1 にリストされています。

新しい値の一時的な使用 インスタンスのデフォルト・スナップショット値とは異なるスナップショット・レベルまたはしきい値を一時的に使用するには、必要なしきい値またはスナップショット・レベルを指定してスナップショットを作成します。この値は、作成する即時スナップショットにのみ使用します。新しい値はデフォルトとして保存されません。

単一のレベル 6 のスナップショットを作成する場合の例を示します。

```
SQL>  EXECUTE STATSPACK.SNAP(i_snap_level=>6);
```

新しいデフォルトの保存 次の 2 つの方法のいずれかで、新しい値をインスタンスのデフォルトとして保存できます。

- STATSPACK.SNAP および I_MODIFY_PARAMETER 入力変数を使用して、スナップショットを作成し、データベースに保存する新しいデフォルトを指定します。

  ```
  SQL>  EXECUTE STATSPACK.SNAP(i_snap_level=>10, i_modify_parameter=>'true');
  ```

  I_MODIFY_PARAMETER 値を TRUE に設定すると、STATS$STATSPACK_PARAMETER 表に新しいしきい値が保存されます。これらのしきい値は、以降のすべてのスナップショットに使用します。

  I_MODIFY_PARAMETER を FALSE に設定するか省略した場合、新しいパラメータ値は保存されません。その時点で作成されたスナップショットのみが、指定された値を使用します。以降のすべてのスナップショットは、STATS$STATSPACK_PARAMETER 表で既存の値を使用します。

- STATSPACK.MODIFY_STATSPACK_PARAMETER プロシージャを使用して、スナップショットを作成せずにただちにデフォルトを変更します。たとえば、次の文ではスナップショット・レベルが 10 に変更され、BUFFER_GETS および DISK_READS の SQL しきい値が変更されます。

  ```
  SQL>  EXECUTE STATSPACK.MODIFY_STATSPACK_PARAMETER
        (i_snap_level=>10, i_buffer_gets_th=>10000, i_disk_reads_th=>1000);
  ```

  このプロシージャは値を永続的に変更しますが、スナップショットは作成しません。

スナップショット・レベル

この項では、スナップショット・レベルを説明します。

レベル >= 0 一般的なパフォーマンス統計 0（ゼロ）より大きいレベルでは、待機統計、システム・イベント、システム統計、ロールバック・セグメント・データ、行キャッシュ、SGA、バックグラウンド・イベント、セッション・イベント、ロック統計、バッファ・プール統計、親ラッチ統計などの一般的なパフォーマンス統計が収集されます。

レベル >= 5 追加データ : SQL 文 このレベルには、下位レベルで収集されるすべての統計と、リソース使用率の高い SQL 文に関するパフォーマンス・データが含まれます。レベル 5（またはそれ以上の）スナップショットでは、スナップショットの作成に必要な時間は、SHARED_POOL_SIZE、およびスナップショット作成時の共有プール内の SQL 文の数によって異なります。共有プールが大きいほど、スナップショット作成にかかる時間が長くなります。

Statspack で収集される SQL 文は、6 つの事前定義のしきい値パラメータのうちの 1 つを超える SQL 文です。

- SQL 文の実行回数。デフォルトは 100。
- SQL 文で実行されるディスク読込み回数。デフォルトは 1,000。
- SQL 文で実行される解析コール回数。デフォルトは 1,000。
- SQL 文で実行されるバッファ取得回数。デフォルトは 10,000。
- SQL 文で使用される共有可能メモリーのサイズ。デフォルトは 1 MB。
- SQL 文のバージョン・カウント。デフォルトは 20。

これらのしきい値パラメータのそれぞれの値は、収集する SQL 文を決定するときに使用します。SQL 文のリソース使用量が前述のしきい値のうちの 1 つを超えている場合、そのリソース使用量はスナップショットの実行中に取得されます。

使用する SQL しきい値のレベルは、STATS$STATSPACK_PARAMETER 表に格納されているか、スナップショットの作成時に指定されるしきい値で格納された SQL しきい値のレベルです。

レベル >= 6 追加データ : SQL 計画および SQL 計画使用状況 このレベルには、下位レベルで収集されたすべての統計と、リソース使用量の多い取得済み SQL 文のそれぞれの SQL 計画および SQL 計画使用状況データが含まれます。

レベル 6 スナップショットは、SQL 文に使用する実行計画が変更されたかどうかを判別するための貴重な情報を収集します。したがって、計画が変更された可能性がある場合はレベル 6 のスナップショットを使用してください。

SQL 文の計画を収集するには、スナップショットを作成する時点で SQL 文が共有プール内にあることが必要であり、SQL しきい値のうちの 1 つを超えている必要があります。共有プール内のすべての文の計画を収集するには、これらのスナップショットの実行しきい値を 0（ゼロ）に指定します。

> **関連項目：** この確認を行う方法の詳細は、21-16 ページの「スナップショット・レベルおよび SQL しきい値のデフォルト値の変更」を参照してください。

レベル >= 7 追加データ : セグメント・レベルの統計 このレベルには、下位レベルで収集されたすべての統計が含まれ、さらに使用頻度の高いセグメントに関するパフォーマンス・データが収集されます。RAC 固有のセグメント・レベルの統計もレベル 7 で取得されます。

レベル 7 のスナップショットは、どのセグメントが頻繁にアクセスされ、競合が起きやすいかを判断する情報を収集します。この情報で、セグメントの一部またはそのセグメントが格納されている表領域の物理的配置を変更できます。たとえば、セグメントの I/O 負荷を適切に拡散するには、頻繁にアクセスするセグメントを格納する表領域に、別のディスク上のファイルを追加したり、セグメントをパーティション化できます。この情報は、PCTFREE または INITRANS などのセグメント属性値を変更する際に役立ちます。RAC 環境でこの情報

を使用すると、インスタンス間の通信の多くの原因となるセグメントを簡単に見つけられます。

レベル 7 には次のセグメント統計が含まれます。

- logical reads
- db block changes
- physical reads
- physical writes
- physical reads direct
- physical writes direct
- global cache cr blocks served（RAC 専用）
- global cache current blocks served（RAC 専用）
- buffer busy waits
- ITL waits
- row lock waits

Statspack はすべてのセグメント統計を取得しますが、定義済みのしきい値パラメータの 1 つを超える次の統計のみをレポートします。

- セグメント上の論理読込み数。デフォルトは 10,000 です。
- セグメント上の物理読込み数。デフォルトは 1,000 です。
- セグメント上の buffer busy waits の数。デフォルトは 100 です。
- セグメント上の row lock waits の数。デフォルトは 100 です。
- セグメント上の ITL waits の数。デフォルトは 100 です。
- global cache cr blocks served の数（RAC 専用）。デフォルトは 1,000 です。
- global cache current blocks served の数（RAC 専用）。デフォルトは 1,000 です。

これらのしきい値パラメータのそれぞれの値は、収集する SQL 文を決定するときに使用します。セグメントの統計がしきい値を超えている場合、このセグメントに関するすべての統計がスナップショットの実行中に取得されます。使用する SQL しきい値のレベルは、STATS$STATSPACK_PARAMETER 表に格納されているか、スナップショットの作成時に指定されるしきい値で格納された SQL しきい値のレベルです。

レベル >= 10 追加統計 : 親ラッチおよび子ラッチ このレベルには、下位レベルで収集されたすべての統計と、親ラッチおよび子ラッチの情報が含まれます。このレベルで収集されるデータは、スナップショット作成の時間が長くなる原因になることがあります。このレベルは多くのリソースを消費する可能性があり、Oracle 担当者からアドバイスがあったときのみ使用してください。

セッション ID の指定

特定のセッションに関するセッション統計と待機イベントを収集する（インスタンス統計と待機イベントに加えて）場合は、Statspack へのコールでセッション ID を指定します。セッションに関して収集された統計には、セッション統計、セッション・イベントおよびロック・アクティビティが含まれています。デフォルト動作では、セッション・レベル統計を収集しません。

たとえば、次のようにします。

```
SQL>  EXECUTE STATSPACK.SNAP(i_session_id=>3);
```

SNAP および MODIFY_STATSPACK_PARAMETER プロシージャのパラメータ

STATSPACK.SNAP プロシージャおよび STATSPACK.MODIFY_STATSPACK_PARAMETER プロシージャに渡すことのできるパラメータは、次のとおりです。

表 21-1 SNAP プロシージャおよび MODIFY_STATSPACK_PARAMETER プロシージャのパラメータ

パラメータ名	有効な値の範囲	デフォルト値	意味
i_snap_level	0, 5, 6, 7, 10	5	スナップショット・レベル
i_ucomment	テキスト	空白	スナップショットで格納されるコメント
i_executions_th	整数 >=0	100	SQL しきい値 : 文の実行回数
i_disk_reads_th	整数 >=0	1000	SQL しきい値 : 文によって実行されたディスク読込みの回数
i_parse_calls_th	整数 >=0	1000	SQL しきい値 : 文によって実行された解析コールの回数
i_buffer_gets_th	整数 >=0	10000	SQL しきい値 : 文によって実行されたバッファ取得の回数
i_sharable_mem_th	整数 >=0	1048576	SQL しきい値 : 共有可能メモリーの量
i_version_count_th	整数 >=0	20	SQL しきい値 : SQL 文のバージョン番号
i_seg_phy_reads_th	整数 >=0	1000	セグメント統計のしきい値 : セグメント上の物理読込み数
i_seg_log_reads_th	整数 >=0	10000	セグメント統計のしきい値 : セグメント上の論理読込み数
i_seg_buff_busy_th	整数 >=0	100	セグメント統計のしきい値 : セグメントに対する buffer busy waits の数

表 21-1 SNAP プロシージャおよび MODIFY_STATSPACK_PARAMETER プロシージャのパラメータ（続き）

パラメータ名	有効な値の範囲	デフォルト値	意味
i_seg_rowlock_w_th	整数 >=0	100	セグメント統計のしきい値 : セグメントに対する row lock waits の数
i_seg_itl_waits_th	整数 >=0	100	セグメント統計のしきい値 : セグメントに対する ITL waits の数
i_seg_cr_bks_sd_th	整数 >=0	1000	セグメント統計のしきい値 : セグメントに対する global cache cr blocks served の数（RAC）
i_seg_cu_bks_sd_th	整数 >=0	1000	セグメント統計のしきい値 : セグメントに対する global cache current blocks served の数（RAC）
i_session_id	V$SESSION の有効な SID	0（セッションなし）	セッションのグラニュル統計を取得する Oracle セッションのセッション ID
i_modify_parameter	TRUE、FALSE	FALSE	今後のスナップショットで指定するパラメータを使用するかどうかを判断

待機イベントに使用される時間単位

Oracle では、マイクロ秒の微小単位でパフォーマンス・データの収集をサポートします。マイクロ秒単位のビューには、次の列が含まれています。

- V$SYSTEM_EVENT、V$SESSION_EVENT（TIME_WAITED_MICRO 列）
- V$SQL（CPU_TIME、ELAPSED_TIME 列）
- V$LATCH（WAIT_TIME 列）
- V$SQL_WORKAREA、V$SQL_WORKAREA_ACTIVE（ACTIVE_TIME 列）

注意： 他のビューの既存の列では、引き続きセンチ秒時間を使用しています。

マイクロ秒の時間単位は、ロールアップ・データに適していない場合があるため、Statspack はほとんどの時間を秒単位で表示します。平均時間は、時間をミリ秒単位で報告することの多いオペレーティング・システムの監視ユーティリティとの比較を容易にするため、ミリ秒単位で表示されます。

明確にするために、使用する時間単位は Statspack レポートの各時間列の列ヘッダーで指定されます。使用される規則は次のとおりです。

- （s）－ 秒
- (cs) － センチ秒（1/100 秒）
- (ms) － ミリ秒（1/1000 秒）
- (us) － マイクロ秒（1/1,000,000 秒）

イベントのタイミング

タイミングが使用できる場合、Statspack レポートは時間で待機イベントを順序付けします。これらのイベントは、トップ 5、およびバックグラウンドとフォアグラウンドの待機イベント・セクションにリストされています。

インスタンスの TIMED_STATISTICS が FALSE でも、ユーザーまたはプログラムのサブセットが TIMED_STATISTICS を動的に TRUE に設定した場合の Statspack レポートの出力では、（個々のプログラムまたはユーザーが待機していた）タイミングを持つイベントとそうでないイベントがあり、不整合のように見えることがあります。このような状況では、トップ 5 のセクションも異常に見えます。

パフォーマンスの問題の診断を容易にするには、インスタンス・レベルで TIMED_STATISTICS を true に設定するのが適切です。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

Statspack パフォーマンス・データの管理および共有

この項では、次の項目について説明します。

エクスポートによるデータの共有

データを他のサイトと共有する場合（たとえば、Oracle Support が RAW 統計を必要とする場合）、PERFSTAT ユーザーをエクスポートできます。このために、エクスポート・パラメータ・ファイル（SPUEXP.PAR）が提供されています。このファイルを使用するには、エクスポート・パラメータ・ファイル名の他に、ユーザー ID パラメータを持つエクスポート・コマンドを指定します。その例を次に示します。

```
exp userid=perfstat/my_perfstat_password parfile=spuexp.par
```

このコマンドで、SPUEXP.DMP と呼ばれるファイルと SPUEXP.LOG ログ・ファイルが作成されます。別のデータベースにデータをロードする場合は、インポート・コマンドを使用します。

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

不要なデータの削除

SPPURGE.SQL スクリプトを使用して、PERFSTAT スキーマから不要なデータをパージします。これにより、指定した開始スナップショット ID と終了スナップショット ID の間のスナップショットが削除されます。

注意： このスクリプトを実行する前に、独自のエクスポート・パラメータまたは SPUEXP.PAR に用意されているパラメータを使用して、スキーマをバックアップとしてエクスポートする必要があります。

パージで大きいロールバック・セグメントを使用する必要があるのは、パージされる各スナップショット ID に関連するすべてのデータが削除されるためです。ロールバック・セグメント拡張エラーは、次の 2 つの方法のいずれかで回避できます。

- パージ対象のスナップショット ID の指定範囲を狭める。
- SPPURGE.SQL スクリプトを実行する前に、SET TRANSACTION USE ROLLBACK SEGMENT 文を実行して大きいロールバック・セグメントを明示的に使用する。

> **関連項目：** 『Oracle9*i* SQL リファレンス』

SPPURGE.SQL を実行すると、接続先のインスタンスと選択可能なスナップショットが表示されます。その後低いスナップ ID と高いスナップ ID が要求されます。この範囲に入るすべてのスナップショットがパージされます。

例 21-6 SPPURGE.SQL の実行例

```
SQL>  CONNECT perfstat/my_perfstat_password
SQL>  SET TRANSACTION USE ROLLBACK SEGMENT rbig;
SQL>  @?/rdbms/admin/sppurge

   Database Instance currently connected to
   ========================================
                                Instance
      DB Id    DB Name    Inst Num Name
   ----------- ---------- -------- ----------
     720559826 PERF              1 perf

   Snapshots for this database instance
   ====================================
               Snap
      Snap Id Level Snapshot Started      Host            Comment
   ---------- ----- --------------------- --------------- --------------------
            1     5  30 Feb 2000 10:00:01 perfhost
            2     5  30 Feb 2000 12:00:06 perfhost
            3     5  01 Mar 2000 02:00:01 perfhost
            4     5  01 Mar 2000 06:00:01 perfhost
```

注意： SPPURGE.SQL は、接続先のデータベース・インスタンスに指定された下限と上限のスナップショット ID の間のすべてのスナップショットを削除します。処理を続ける前に、このデータをエクスポートしてください。

```
Specify the Low Snap ID and High Snap ID range to purge
   ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
   Enter value for losnapid: 1
   Using 1 for lower bound.

   Enter value for hisnapid: 2
   Using 2 for upper bound.

   Deleting snapshots 1 - 2
```

```
Purge of specified snapshot range complete. If you want to rollback the purge, it is
still possible to do so. Exiting from SQL*Plus automatically commits the purge.

SQL> -- end of example output
```

バッチ・モードでパージするには、パージ対象となる低いスナップショット ID と高いスナップショット ID を、SQL*Plus 変数の値に割り当てる必要があります。これらの変数は次のとおりです。

- LOSNAPID: 開始スナップショット ID
- HISNAPID: 終了スナップショット ID

例 21-7 バッチ・モードでの SPPURGE.SQL の実行

```
SQL>  CONNECT perfstat/my_perfstat_password
SQL>  DEFINE losnapid=1
SQL>  DEFINE hisnapid=2
SQL>  @?/rdbms/admin/sppurge
```

SPPURGE.SQL を実行中の場合は、変数によって入力される情報をプロンプトしません。

すべての Statspack データの切捨て

すべてのパフォーマンス・データを無差別に切り捨てるには、SPTRUNC.SQL を使用します。このスクリプトは、収集されたすべての統計データを切り捨てます。

> **注意：** このスクリプトを実行する前に、独自のエクスポート・パラメータまたは SPUEXP.PAR. に用意されているエクスポート・パラメータを使用して、スキーマをバックアップとしてエクスポートすることをお薦めします。

例 21-8 SPTRUNC.SQL の実行例

```
SQL>  CONNECT perfstat/my_perfstat_password
SQL>  @?/rdbms/admin/sptrunc
~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
```

> **注意：** SPTRUNC.SQL を実行すると、Statspack 表からすべてのデータが削除されます。処理を続ける前に、このデータをエクスポートしてください。

```
If you would like to continue, enter any string, followed by <return>.
Enter value for anystring:
entered - starting truncate operation
Table truncated.
<etc>
Truncate operation complete.
```

Statspack 使用時の Oracle Real Application Clusters の考慮事項

Statspack で使用されるデータベース・インスタンスの一意の識別子は、DBID と INSTANCE_NUMBER です。Oracle Real Application Clusters を使用すると、INSTANCE_NUMBER は次の起動時に変更される可能性があります（INSTANCE_NUMBER パラメータが初期化ファイルで設定されているか、インスタンスが異なる順序で起動されるため）。

Statspack は INSTANCE_NUMBER と DBID を使用してインスタンスのスナップショット・プリファレンスを識別するため、インスタンスのスナップショットを作成する場合、別のレベルまたはしきい値のセットが使用される可能性があります。これは次の条件下でのみ発生します。

- 起動と起動の間でインスタンス番号が切り替えられた。
- 少なくとも 1 つのインスタンスに使用するデフォルトの Statspack パラメータが DBA によって変更された。
- 使用するパラメータ（しきい値やスナップショット・レベル）がすべてのインスタンス上で同じになっていない。

パラメータを変更するのは、指定値を保存するか、MODIFY_STATSPACK_PARAMETER プロシージャを使用して、インストール後に DBA がパラメータを明示的に変更した場合に限ります。インスタンスごとに Statspack スナップショット・パラメータが異なるかどうかをチェックするには、STATS$STATSPACK_PARAMETER 表に問い合せます。

注意： デフォルトの Statspack パラメータを変更した場合、Oracle Real Application Clusters データベース内のインスタンスごとに初期化パラメータ・ファイル内で INSTANCE_NUMBER をハードコード化し、この問題の発生を回避できます。

関連項目： INSTANCE_NUMBER 初期化パラメータ設定の推奨事項および問題については、『Oracle9*i* Real Application Clusters 管理』を参照してください。

Statspack の削除

Statspack を削除するには、SYSDBA 権限を持つユーザーとして接続し、SQL*Plus から次の SPDROP スクリプトを実行します。その例を次に示します。

```
SQL>  CONNECT / AS SYSDBA
SQL>  @?/rdbms/admin/spdrop
```

SPDROP.SQL スクリプトは次のスクリプトを呼び出します。

- SPDTAB.SQL ー 表とパブリック・シノニムを削除します。
- SPDUSR.SQL ー ユーザーを削除します。

作成された 2 つの出力ファイル（SPDTAB.LIS および SPDUSR.LIS）のそれぞれをチェックして、パッケージが完全に削除されたことを確認します。

Statspack が提供するスクリプトおよびマニュアル

この項では、次の項目について説明します。

- Statspack のインストールおよび削除のスクリプト
- Statspack のレポートおよび自動化のスクリプト
- Statspack をアップグレードするためのスクリプト
- Statspack パフォーマンス・データ・メンテナンスのためのスクリプト
- Statspack マニュアル

Statspack のインストールおよび削除のスクリプト

SYSDBA 権限を持つユーザーとして、Statspack のインストールおよび削除のスクリプトを実行する必要があります。

- SPCREATE.SQL: 次のスクリプトを呼び出して Statspack 環境全体を作成します。
  - SPCUSR.SQL: Statspack ユーザーを作成します（PERFSTAT）。
  - SPCTAB.SQL: Statspack 表を作成します。
  - SPCPKG.SQL: Statspack パッケージを作成します。
- SPDROP.SQL: 次のスクリプトを呼び出して Statspack 環境全体を削除します。
  - SPDTAB.SQL: Statspack 表を削除します。
  - SPDUSR.SQL: Statspack ユーザーを削除します（PERFSTAT）。

Statspack のレポートおよび自動化のスクリプト

Statspack レポートおよび自動化のスクリプトは、PERFSTAT ユーザーとして実行する必要があります。

- SPREPORT.SQL: Statspack レポートを生成します。
- SPREPSQL.SQL: 指定された特定の SQL ハッシュ値に関する Statspack SQL レポートを生成します。
- SPREPINS.SQL: 指定されたデータベースとインスタンスに関する Statspack レポートを生成します。
- SPAUTO.SQL: Statspack 統計収集を自動化します（DBMS_JOB を使用）。

Statspack をアップグレードするためのスクリプト

Statspack のアップグレード用スクリプトは、SYSDBA 権限を持つユーザーとして実行する必要があります。

- SPUP90.SQL: データを 9.0 スキーマから 9.2 スキーマに変換します。アップグレードを実行する前に既存のスキーマのバックアップを取ります。
- SPUP817.SQL: Statspack 8.1.7 からアップグレードする場合は、まずこのスクリプトを実行する必要があります。
- SPUP816.SQL: Statspack 8.1.6 からアップグレードする場合は、まずこのスクリプトを実行し、次に SPUP817.SQL を実行する必要があります。

Statspack パフォーマンス・データ・メンテナンスのためのスクリプト

Statspack のデータ・メンテナンス用スクリプトは、PERFSTAT ユーザーとして実行する必要があります。

- SPPURGE.SQL: 特定のデータベース・インスタンスの限られた範囲のスナップショット ID をパージします。
- SPTRUNC.SQL: Statspack 表中のすべてのパフォーマンス・データを切り捨てます。

> **注意：** 使用しているスキーマ内のデータをすべて削除する場合以外、このスクリプトは使用しないでください。このスクリプトを使用する前に、そのデータをバックアップとしてエクスポートできます。

- SPUEXP.PAR: PERFSTAT ユーザー全体をエクスポートするために提供されているエクスポート・パラメータ・ファイル。

Statspack マニュアル

ORACLE_HOME/rdbms/admin ディレクトリにある SPDOC.TXT ファイルには、Statspack パッケージに関する指示とマニュアルが含まれています。

第 V 部

インスタンスのパフォーマンスの最適化

第 V 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・システムの様々な要素をチューニングする方法について説明します。

第 V 部には次の章が含まれます。

- 第 22 章「インスタンスのチューニング」
- 第 23 章「ネットワークのチューニング」

22

インスタンスのチューニング

データベースの初期構成後は、インスタンスのチューニングがパフォーマンスのボトルネックを解消するために重要になります。

この章には次の項があります。

パフォーマンス・チューニングの原理

パフォーマンス・チューニングでは、システムの初期構成に対して異なる（ただし関連性がある）方法を必要とします。システムの構成では、初期システムの構成が機能的なものになるように整理された手順に従ってリソースを割り当てます。

チューニングを開始するには、最も影響のあるボトルネックを識別し、適切な変更を行ってそのボトルネックの影響を低減するかまたは排除します。チューニングは通常、システムが本番開始以前か、または稼働状態になった後で事後的に行われます。

注意： このマニュアルを読む前に、『Oracle9*i* データベース・パフォーマンス・プランニング』を読んでおくことをお薦めします。オラクル社では、長年にわたる設計およびパフォーマンス経験に基づき、新しいパフォーマンス手法を設計しました。このマニュアルでは、システム・パフォーマンスを大幅に向上させる明瞭で単純なアクティビティについて説明します。次の項目について説明しています。

- 投資対効果
- 拡張性
- システム・アーキテクチャ
- アプリケーション設計の原理
- ワークロードのテスト、モデル化および実装
- 新規アプリケーションの配布

ベースライン

最も効率的なチューニングの方法は、パフォーマンスの問題が生じた場合に比較に使用できるパフォーマンス・ベースラインを確立することです。DBA の多くは、自分のシステムを熟知し、ピークの使用期間を簡単に識別できます。たとえば、ピーク期間は 10.00am ～ 12.00pm である場合、また 1.30pm ～ 3.00pm である場合もあります。これには、深夜の 12.00am から 6.00am までのバッチ・ウィンドウが含まれることがあります。

サイトでこのような高負荷の時間を識別し、このような時間のパフォーマンス・データを収集する監視ツールをインストールすることが重要です。アプリケーションが QA サイクル中の初期のトライアル段階にある時点から、データ収集を構成することが最適です。それ以外の場合は、システムが最初に稼働したときに、このデータ収集を構成する必要があります。

注意： 拡張機能リストのために、システムの監視とチューニングに Oracle Enterprise Manager（EM）Diagnostics Pack を使用することをお薦めします。ただし、サイトに EM がない場合、Statspack を使用して Oracle インスタンス統計を収集できます。

説明のために、Statspack レポート出力と V$ ビューからの直接問合せの組合せが例の中で使用されるのは、これらがすべてのインストールで使用できるからです。

関連項目： Oracle インスタンスのパフォーマンス・チューニング・ツールの詳細は、第 20 章「データベース統計収集用の Oracle のツール製品」を参照してください。

収集されたベースライン・データには、次の内容が含まれていることが理想的です。

- アプリケーション統計（トランザクション・ボリューム、応答時間）
- データベース統計
- オペレーティング・システム統計
- ディスク I/O 統計
- ネットワーク統計

症状および問題点

パフォーマンス・チューニングの一般的な誤りは、ある問題の症状を現実の問題自体であると思い違いをすることです。多くのパフォーマンス統計はこの症状を示すこと、およびこの症状を識別することが修正を実施するために十分なデータではないことを認識することが重要です。その例を次に示します。

- 低速な物理 I/O

  一般に、この原因はディスクの構成が適切ではないことにあります。しかし、チューニングが適切ではない SQL から発行された、これらのディスク上の大量の不要な物理 I/O が原因になっている可能性もあります。

- ラッチの競合

  インスタンスを再構成して、ラッチの競合をチューニングできることはほとんどありません。むしろ、ラッチの競合は通常、アプリケーションの変更により解決されます。

- 過剰な CPU 使用率

  過剰な CPU 使用率は通常、システム上にアイドル状態の CPU がほとんどないことを意味します。この原因として、システムのサイズ設定が不適切であること、SQL 文がチューニングされていないこと、またはアプリケーション・プログラムが不十分である可能性があります。

関連項目： 22-18 ページの表 22-1「待機イベントおよび潜在的な原因」

チューニングの時期

チューニングには次の 2 種類があります。プロアクティブな監視およびボトルネックの解消です。

プロアクティブな監視

プロアクティブな監視は通常、定期的にスケジュールされた間隔で行われます。この場合、システム動作とリソースの使用量が変化したかどうかを識別するために多数のパフォーマンス統計が調べられます。プロアクティブな監視は、プロアクティブなチューニングとも呼ばれます。

通常は、進行中の重大な問題が監視により明らかにならないかぎり、監視によりシステムの構成が変化することはありません。状況によっては、経験豊富なパフォーマンス・エンジニアが統計のみで潜在的な問題点を識別できますが、通常はパフォーマンスの低下を伴います。

明らかなパフォーマンスの低下がないときに、事前のアクションとしてシステムを微調整することは危険なアクティビティであり、不必要にパフォーマンスを低下させる可能性があります。システムを微調整することは事後チューニングと考えられ、事後チューニングの手順に従う必要があります。

監視は通常、大きな容量計画の調査の一環です。この場合、リソース使用を調べて、アプリケーションが使用されている方法、およびアプリケーションがデータベース・リソースとホスト・リソースを使用している方法の変化を調べます。

ボトルネックの解消 : チューニング

チューニングは通常、パフォーマンスの問題の修正を意味します。ただし、チューニングは、分析、設計、コーディング、生産およびメンテナンスの各段階を通じて、アプリケーションのライフサイクルの一部である必要があります。多くの場合、チューニング段階はシステムが本番に入るまで残されます。このとき、チューニングは事後対応処理になります。この場合、最も重要なボトルネックが識別され、修正されます。

チューニングの目的は通常、リソース使用量を減らしたり、操作を完了する経過時間を減らすことにあります。いずれの場合も、目標は特定のリソースの有効利用を向上することにあります。一般に、パフォーマンスの問題は特定のリソースの過剰使用によって発生します。そのリソースは、システム内のボトルネックとなります。ボトルネックと潜在的な修正を識別するには、様々な多くの段階があります。それらについては次の項で説明します。

競合の様々な形態は症状であり、次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。

- アプリケーションまたはアプリケーションの使用方法の変更
- Oracle の変更
- ホスト・ハードウェア構成の変更

多くの場合、ボトルネックを解決する最も効果的な方法は、アプリケーションを変更することです。

パフォーマンス・チューニングの手順

次に、Oracle パフォーマンス・メソッドの主な手順を示します。

1. パフォーマンス問題の範囲についてユーザーから候補フィードバックを取得します。このステップでは、問題の定義を行います。
2. オペレーティング・システム、データベースおよびアプリケーション統計一式を取得します。次に、証拠を探すためにホスト・システムの検査と Oracle 統計の調査を行います。
3. 一般的なパフォーマンス・エラーのリストを検討して、収集されたデータが問題に影響を与えていることを示しているかどうかを確認します。
4. 収集されたパフォーマンス・データを使用して、何がシステムで起こっているかを示す概念モデルを構築します。
5. 行う変更および変更を実装した場合に予測される結果を提示します。次に、アプリケーション・パフォーマンスで変更の実装および測定を行います。
6. 手順 1 で定義したパフォーマンスの目的が達成されたかどうかを判断します。達成されていない場合は、パフォーマンスの目標が達成されるまで手順 5 と 6 を繰り返します。

> **関連項目：** 共通エラーのリストおよびこのパフォーマンス・メソッドの理論的な説明については、『Oracle9*i* データベース・パフォーマンス・プランニング』を参照してください。

この章の後半では、Oracle パフォーマンス・メソッドの各手順について詳しく説明します。

問題の定義

ソリューションの実装を試みる前に、チューニング調査の目的と問題の性質をよく理解しておくことが不可欠です。これについて理解していないと、事実上、効果的な変更は実装できません。この段階で収集されたデータを使用して、次に行うこと、および調査する事象を簡単に決定できます。

次のデータを収集します。

1. パフォーマンスの目的を識別します。

   許容できるパフォーマンスの測定尺度は何ですか？1 時間、または 1 秒間当たり何件のトランザクションで、応答時間が必要なパフォーマンス・レベルを満たしますか？

2. 問題の範囲を識別します。

   スローダウンで何が影響を受けますか？たとえば、インスタンス全体は低速ですか？それは、特定のアプリケーション、プログラム、特定の操作またはシングル・ユーザーですか？

3. 問題が発生したときの時間帯を識別します。

   その問題はピーク時間のみ明白ですか？パフォーマンスはその日の経過にともなって低下しますか？スローダウンは徐々に（月または週の単位で）発生しましたか、または突然発生しましたか？

4. スローダウンを検証します。

   これは、問題の範囲の識別に役立ち、問題の修復のために実装された変更により実際に改善されたかどうかを判断するときの比較結果の測定基準としての役割を果たします。一貫して再生可能な応答時間またはジョブ実行時間の測定値を検索します。プログラムの動作が正常だったときよりタイミングがどのくらい悪化していますか？

5. 変更を識別します。

   パフォーマンスが許容可能になった後に変化した内容を識別します。これにより、潜在的な原因を素早くつきとめることができます。たとえば、オペレーティング・システムのソフトウェア、ハードウェア、アプリケーション・ソフトウェアまたは Oracle リリースがアップグレードされましたか？さらに多くのデータがシステムにロードされたか、データ・ボリュームまたはユーザー人口が増加しましたか？

このフェーズの終わりまでに、症状についてよく理解しておく必要があります。症状をプログラムまたはプログラム・セットにローカルなものとして識別できる場合、その問題はインスタンス全体のパフォーマンスの問題とは異なる方法で処理されます。

> **関連項目：** アプリケーションまたはユーザーに固有なパフォーマンスの問題の解決については、第 6 章「SQL 文の最適化」を参照してください。

ホスト・システムの検査

データベース・サーバーに対する負荷とデータベース・インスタンスを調べてください。オペレーティング・システム、I/O サブシステムおよびネットワーク統計を検討してください。これらの領域を調べると、調査する価値のあるものは何かが容易にわかります。多層のシステムでは、アプリケーション・サーバーの中間層ホストも調べてください。

ホスト・ハードウェアを調べると、システム内のボトルネックがよくわかります。このため、相互参照と以降の診断に役立つ Oracle パフォーマンス・データを判断できます。

調べるデータには、次のものがあります。

CPU の使用率

アイドル状態の CPU が大量にある場合、I/O、アプリケーションまたはデータベースのボトルネックが存在する可能性があります ただし、wait I/O はアイドル状態の CPU とみなす必要があります。

CPU 使用率が高い場合は、CPU が効果的に使用されているかどうかを判断してください。CPU 使用率の大部分は、CPU 使用率の高い少数のプログラムによるものですか、または均等に分散されたワークロードで CPU が消費されていますか？

CPU が使用頻度の高い少量のプログラムで使用されている場合は、プログラムを調べて原因を判断してください。

Oracle 以外のプロセス プログラムが Oracle のプログラムではない場合は、それらのプログラムがそのような量の CPU を必要としているかどうかを識別してください。必要としている場合は、プログラムの実行をピーク以外の時間に遅らせることができますか？

Oracle プロセス 少量の Oracle プロセスがほとんどの CPU リソースを使用する場合は、`SQL_TRACE` と `TKPROF` を使用して SQL 文または PL/SQL 文を識別し、特定の問合せまたは PL/SQL のプログラム・ユニットをチューニングできるかどうかを確認します。たとえば、CPU を多く使用するようなキャッシュ内の多数のデータ読込み（論理読込み）に関連した `SELECT` 文があった場合、その文の SQL の最適化により CPU の集中的な使用を回避できます。

Oracle CPU 統計 Oracle CPU 統計は、3 つの `V$` ビューで使用できます。

- `V$SYSSTAT` は、すべてのセッションにおける Oracle の CPU 使用率を示します。統計「CPU used by this session」は、すべてのセッションで使用されている CPU の合計を示します。
- `V$SESSTAT` は、各セッションにおける 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します。
- Oracle Database Resource Manager を実行している場合、`V$RSRC_CONSUMER_GROUP` は、各コンシューマ・グループの CPU 使用率の統計を示します。

CPU 統計の解釈 CPU タイムと実時間が異なることを認識することが重要です。8 つの CPU を使用する場合、実時間の所定の時間に、8 分の CPU タイムが利用できます。NT および UNIX では、これはユーザー時間またはシステム時間（NT では特権モード）となります。したがって、システム上のすべてのプロセス（スレッド）で使用される平均 CPU タイムは、1 分の実時間間隔当たり 1 分を超える可能性があります。

ある時点での 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 分を使用している場合は、総 CPU タイムの 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。CPU タイムを使用しているプロセスを識別し、原因を解明し、それらのプロセスのチューニングを試行してください。

> **関連項目：** 第 10 章「SQL トレースおよび TKPROF の使用」

CPU の使用率が Oracle サーバー・プロセス全体に均等に分散されている場合は、Statspack レポートを調べて他の証拠を確認してください。

I/O の問題の検出

過度にアクティブな I/O システムは、2 より大きいディスク・キューの長さ、すなわち、20 ～ 30 ミリ秒を超えるディスク・サービス時間でわかります。I/O システムが過度にアクティブである場合、さらに多くのディスク間に I/O を分散させることで利益を得られる潜在的なホット・スポットの有無をチェックします。また、これらのリソースを使用して、プログラムのリソース要件を少なくして負荷を減らせるかどうかも識別します。

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ります。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減らしてください。Oracle 以外のファイルへのアクセスは、V$ 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。

多数の UNIX システム上の `sar -d`（または `iostat`）や Windows 2000 システム上の Performance Monitor などのツールは、システム全体の I/O 統計を調べます。

> **関連項目：** プラットフォームで使用可能なツールのオペレーティング・システムのマニュアル

`V$SYSTEM_EVENT` の Oracle 待機イベント・データをチェックして、トップの待機イベントが I/O 関連かどうかを確認します。I/O 関連イベントには、`db file sequential read`、`db file scattered read`、`db file single write` および `db file parallel write` が含まれます。これらはすべて、データ・ファイル・ヘッダー、制御ファイルあるいはデータ・ファイルに対して実行される I/O に該当するイベントです。これらの待機イベントのうちのいずれかが高い平均時間に該当する場合は、I/O の競合を調べる必要があります。

Statspack レポート内の I/O セクションでホスト I/O システム・データを相互参照し、ホット・データ・ファイルおよび表領域を識別します。さらに、オペレーティング・システムから報告された I/O 時間と、Oracle から報告された時間とを比較して、それらに一貫性があるかどうかを確認します。

I/O システムを再構成する必要があるかどうかを調べる前に、I/O システム上の負荷を減らせるかどうかを判断します。Oracle I/O 負荷を減らすには、V$SQLAREA ビューを問い合せるか、または Statspack レポートの 'SQL ordered by physical reads' セクションを確認して、多数の物理読込みを実行する SQL 文を調べます。これらの文を調べて、I/O の回数を減らすようにこれらの SQL 文をチューニングする方法を調べます。

SQL 文で発生した 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てください。Oracle サーバーが使用可能な I/O リソースを使用していない場合、I/O をすべて使用しているプロセスを識別します。プロセスが I/O をすべて使用している理由を判断し、次にこのプロセスをチューニングします。

関連項目：

- 第 6 章「SQL 文の最適化」
- 24-56 ページ「V$SQLAREA」
- 第 15 章「I/O 構成および設計」
- 散布読込みと順次読込みの違いと、それが I/O に与える影響については、22-29 ページの「db file scattered read」および 22-31 ページの「db file sequential read」を参照してください。

ネットワーク

オペレーティング・システムのユーティリティを使用して、ネットワーク・ラウンドトリップの ping 時間と衝突数を調べます。ネットワークで応答時間の大幅な遅延が発生している場合は、考えられる原因を調べてください。

ネットワーク負荷を減らすには、大きいデータ転送をピーク時間外へスケジュールするか、リモート・ホストに対して要求当たり 1 回ずつ（またはそれ以上）アクセスせずに、要求をバッチ処理するようにアプリケーションをコーディングします。

関連項目： 重要なオペレーティング・システム統計の説明については、『Oracle9*i* データベース・パフォーマンス・プランニング』を参照してください。

Oracle 統計の調査

問題の診断の一貫性が保たれるようにするには、Oracle 統計を調べオペレーティング・システムの統計と相互参照してください。ただし、目標が Oracle インスタンスをチューニングすることにあれば、修正アクションを実装する前に Oracle 統計を見て Oracle の観点からリソース・ボトルネックを識別します。

関連項目： 22-15 ページ「Oracle 統計の解釈」

統計収集のレベルの設定

Oracle9*i* リリース 2（9.2）では、初期化パラメータ STATISTICS_LEVEL が提供され、すべての主要統計収集またはアドバイザをデータベースで制御します。このパラメータでは、データベースに統計収集レベルを設定します。

STATISTICS_LEVEL の設定に応じて、次のように一定のアドバイザ機能および統計が収集されます。

BASIC: アドバイザ機能も統計も収集されません。

TYPICAL: 次のアドバイザ機能または統計が収集されます。

- バッファ・キャッシュ・アドバイザ機能
- MTTR アドバイザ機能
- 共有プール・サイズ設定アドバイザ機能
- セグメント・レベル統計
- PGA ターゲット・アドバイザ機能
- 時間統計

ALL: 前述のすべてのアドバイザ機能または統計以外に、次の統計が収集されます。

- オペレーティング・システム時間統計
- 行ソース実行統計

デフォルト・レベルは TYPICAL です。STATISTICS_LEVEL は動的パラメータであり、システムまたはセッション・レベルで変更される可能性があります。

前述のリストのすべてのアドバイザ機能または統計は、ALTER SYSTEM で変更すると、STATISTICS_LEVEL の新しい値に応じて動的にオンまたはオフになります。

次のアドバイザ機能または統計に限り、ALTER SESSION で修正すると、ローカル・セッションでのみオンまたはオフになります。システム全体での状態は変更されません。

- 時間統計
- オペレーティング・システム時間統計
- 行ソース実行統計

V$STATISTICS_LEVEL このビューには、STATISTICS_LEVEL で制御される統計またはアドバイザ機能のステータスがリストされます。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： このビューの列の詳細は、24-60 ページの「V$STATISTICS_LEVEL」を参照してください。

チューニング中に使用する一般的な Oracle データ・ソースを次に示します。これらのソースは、待機イベントとシステム統計という 2 種類の統計に分けることができます。

待機イベント

待機イベントは、処理を継続する前にイベントが完了するまで待機する必要があることを示すために、サーバー・プロセス / スレッドによって増やされる統計です。待機イベント・データは、ラッチの競合、バッファの競合、I/O の競合などのパフォーマンスに影響を与えると思われる様々な問題の症状を表します。ただし、これらの問題は実際の原因でなく問題の症状にすぎないことに注意してください。

サーバー・プロセスは、次の症状に対して待機します。

- リソースが使用可能になるまで（バッファやラッチなど）
- アクションが完了するまで（I/O など）
- 実行する追加作業（クライアントが次に実行する SQL 文を提供するまで待機する場合など）サーバー・プロセスが追加作業を待機していることを識別するイベントをアイドル・イベントと呼びます。

待機イベント統計には、イベントを待った回数や、イベントが完了するまでの待機時間があります。待機イベント統計について、V$SESSION_WAIT、V$SESSION_EVENT および V$SYSTEM_EVENT の各イベントの問合せを行うことができます。TIMED_STATISTICS 構成パラメータを true に設定すると、各リソースを待機した時間もわかります。ユーザー応答時間をできるだけ少なくするには、イベントが完了するまでサーバー・プロセスが待機する時間を減らします。すべての待機イベントが同じ待機時間を持っているとはかぎりません。したがって、発生回数の多い待機イベントより、最大の合計時間を持つイベントを調べるほうが重要です。通常は、少なくともパフォーマンスの監視中に TIMED_STATISTICS 動的パラメータを true に設定することが最善です。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

事後パフォーマンス・チューニングを実行するときに、待機イベントと関連するタイミング・データを調査します。最大時間がリストされるイベントは、多くの場合、パフォーマンス・ボトルネックを顕著に示しています。たとえば、V$SYSTEM_EVENT を参照することで、多くの buffer busy waits が発生していると気づくことがあります。おそらく、多数のプロセスが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。

関連項目： V$SESSION_WAIT、V$SESSION_EVENT および V$SYSTEM_EVENT の各ビューの差異については、22-23 ページの「待機イベント」を参照してください。

システム統計

システム統計は通常、パフォーマンスの問題の原因をさらに示すものを見つけるために、待機イベント・データとともに使用されます。

たとえば、最大の待機イベント（待機時間の点で）が buffer busy waits イベントであることを V$SYSTEM_EVENT が示している場合、V$WAITSTAT ビューで使用できる特定のバッファ待機統計を調べて、どのブロック・タイプが最大の待機カウントと最大の待機時間を持っているかを識別します。ブロック・タイプを識別した後、問題が発生している間にリアルタイムで V$SESSION_WAIT を調べ、表示されたファイル番号とブロック番号を使用して競合されたオブジェクトも識別します。このデータの組合せは、適切な修正アクションを示しています。

統計は、多数の V$ ビューで使用できます。次の内容を含む共通ビューもあります。

V$SYSSTAT これには、ロールバック、論理および物理 I/O、解析データなど、Oracle の様々な部分の統計全体が含まれています。バッファ・キャッシュ・ヒット率などの比率を計算するには、V$SYSSTAT からのデータを使用します。

V$FILESTAT これには、ファイル当たりの I/O 回数や平均読込み時間など、ファイルごとの詳細なファイル I/O 統計が含まれています。

V$ROLLSTAT これには、詳細なロールバック・セグメントおよび UNDO セグメント統計が含まれています。

V$ENQUEUE_STAT これには、エンキューが要求された回数やエンキューを待機した回数、待機時間など、各エンキューの詳細なエンキュー統計が含まれています。

V$LATCH これには、各ラッチが要求された回数やラッチを待機した回数など、各ラッチの詳細なラッチ使用統計が含まれています。

> **関連項目：** チューニングに使用する V$ ビューの詳細説明は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

セグメント・レベルの統計

Oracle9*i* リリース 2（9.2）以上では、個別のセグメントに関連するパフォーマンス問題を確認するために役立つ、セグメント・レベルの統計を収集できます。セグメント・レベルの統計を収集して表示することは、インスタンスで競合度の高い表あるいは索引を効果的に識別するための優れた方法です。

パフォーマンスの問題を識別するために待機イベントまたはシステム統計を表示した後、セグメント・レベルの統計を使用して問題の原因となっている特定の表または索引を検索できます。バッファ・ビジー待機が、大半の待機時間の原因になっていることを V$SYSTEM_EVENT が示している例を考えます。buffer busy waits の原因になっているトップ・セグメントを V$SEGMENT_STATISTICS から選択できます。これにより、それらのセグメントの問題の解決に集中できます。

セグメント・レベルの統計は、次の動的ビューを使用して問い合せます。

- V$SEGSTAT_NAME このビューには収集するセグメント統計と、各種統計（たとえばサンプル統計など）のプロパティがリストされます。
- V$SEGSTAT これは非常に効率的で、リアルタイム監視が可能なこのビューには、統計値、統計名およびその他の基本情報が表示されます。

- V$SEGMENT_STATISTICS ユーザーが扱いやすい統計値のビューです。V$SEGSTAT のすべての列の他、ここにはセグメント所有者や表領域名に関する情報があります。統計の理解は容易になりますが、コストがより高くなります。

  関連項目： 列情報の詳細は、24-32 ページから始まる第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

変更の実装および測定

チューニングを実施した後、多くの場合、問題を軽減できると思われる 2 つまたは 3 つの変更を識別できます。どの変更が最高の利益を提供するかを識別するには、一度に 1 回の変更を実装することをお薦めします。変更の効果は、問題定義段階で見られたベースライン・データ測定と対照して測定する必要があります。

一般に、パフォーマンスの問題を持つ大半のサイトでは、一度に重複した変更を実装するので、どの変更が利益を実現したかを識別できません。これはすぐに問題になることはありませんが、どの変更が最も効果をあげ、どのような作業を優先するべきかを知ることは不可能なので、今後同様の問題が発生した場合に大きな障害になります。

個別に変更を実装できない場合は、異なる変更の効果の測定を試みてください。たとえば、変更された問合せのパフォーマンスを向上するために新しい索引を作成する効果とは別に、REDO の生成を最適化するために初期化変更を行う効果を測定します。SQL をチューニングし、オペレーティング・システムのディスク・レイアウトを変更し、初期化パラメータも同時に変更する場合は、オペレーティング・システムのアップグレードを行う利益を測定できません。

パフォーマンス・チューニングは反復的なプロセスです。インスタンス・ワイドのパフォーマンスの問題を解決する万全策が見つかることはほとんどありません。ほとんどの場合、あるボトルネックを解決しても別の（ときにはさらに悪い）問題が発生するため、優れたパフォーマンスにはパフォーマンス・チューニング段階を反復する必要があります。

いつチューニングを停止するかを知ることも重要です。パフォーマンスの最も優れた測定は、統計が理想的な値にどの程度近いかではなく、ユーザーの理解力です。

Oracle 統計の解釈

インスタンスにパフォーマンスの問題があった時を示す統計を収集します。比較のためのベースライン・データをすでに収集してある場合は、問題のワークロードを最も代表するベースラインからのデータと、現行のデータを比較できます。

2 つのレポートを比較する場合、それらのレポートが、システムを比較できるようなワークロードか確認してください。

関連項目： 20-2 ページ「データ収集の原理」

負荷の検査

待機イベントは通常、最初に検査するデータです。ただし、ベースライン・レポートがある場合は、負荷が変化したかどうかをチェックします。ベースラインがあるかどうかにかかわらず、リソースの使用率が高いかどうかを確認すると便利です。

調査する負荷関連の統計には、`redo size`、`session logical reads`、`db block changes`、`physical reads`、`physical writes`、`parse count (total)`、`parse count (hard)` および `user calls` があります。このデータは、`V$SYSSTAT` から問合せが行われます。秒ごとおよびトランザクションごとに、このデータを正規化することが最も有効です。

Statspack レポートの「ロード・プロファイル」の項を参照してください。データは、トランザクションおよび秒ごとに正規化されています。

負荷の変更

秒ごとの負荷プロファイル統計は、スループットの変化（すなわち、インスタンスの作業実行量が毎秒ごとに増えているかどうか）を示します。トランザクションごとの統計は、アプリケーション特性の変化をベースライン・レポートからの対応する統計と比較することで識別します。

高いアクティビティ率

アクティビティ率が非常に高いかどうかを識別するには、秒ごとに正規化した統計を調べます。包括的に高い値を推薦することが難しいのは、しきい値が各サイトで異なり、アプリケーション特性、CPU の個数と速度、オペレーティング・システム、I/O システムおよび Oracle リリースで異なるからです。

次に、いくつかの一般化された例を示します（許容値は各サイトで異なります）。

- 秒当たり 100 を超えるハード解析率は、システム上に非常に大量なハード解析があることを示します。高いハード解析率は重大なパフォーマンスの問題を発生させるので、調査する必要があります。通常は、高いハード解析率に共有プール上のラッチの競合とライブラリ・キャッシュ・ラッチが伴います。latch free の待機が、トップ 5 の待機イベントに現れているかどうかをチェックし、現れていれば、Statspack レポートのラッチ・セクションを調べます。
- 高いソフト解析率は、秒当たり 300 以上の率になる可能性があります。不必要なソフト解析もアプリケーションの拡張性を制限します。最適な方法として、SQL 文をセッション当たり 1 回ソフト解析し、何回も実行します。

待機イベント統計を使用したボトルネックへのドリルダウン

Oracle が何かの待機を処理する場合は必ず、定義済待機イベント・セットの 1 つを使用し、待機を記録します。（待機イベントのすべてのリストについては、V$EVENT_NAME を参照してください。）待機イベントのいくつかがアイドル・イベントと呼ばれているのは、プロセスがアイドル状態であり、作業が行われるまで待機するためです。アイドル状態でないイベントはリソースあるいはアクションが完了するまでの非生産的な待機時間を示します。

注意： すべての症状が待機イベントによって証明されるわけではありません。チェックできる統計については、22-19 ページの「追加された統計情報」を参照してください。

待機イベント・データを使用する最も効率的な方法は、待機時間別にイベントを順序付けすることです。この方法は、TIMED_STATISTICS が true に設定されているときのみ可能です。設定しない場合は、、待機イベントを待機数別に順位付けします。これは、一般的に問題を最もよく表す順序付けではありません。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

どこで時間が消費されているかが判明してから、次の手順を実行してください。

1. V$SYSTEM_EVENT のデータ収集を調べます。対象のイベントは、待機時間別に順位付けする必要があります。

   待機時間の最も大きいパーセンテージを持つ待機イベントを識別します。待機時間のパーセンテージを決定するには、アイドル・イベント（Null event、SQL*Net message from client、SQL*Net message to client、SQL*Net more data from client など）を除くすべての待機イベントの合計待機時間を追加します。各イベントの待機時間をすべてのイベントの総待機時間で除算し、5 つの最も重要なイベントの相対的なパーセンテージを計算します。

   > **関連項目：** アイドル・イベントの全リストは、22-48 ページの「アイドル待機イベント」を参照してください。

   別の方法として、Statspack レポートのフロント・ページの「トップ 5 待機イベント」の項を参照してください。この項では、待機イベント（アイドル・イベントを除く）を自動的に順序付けし、相対的なパーセンテージを計算します。

```
Top 5 Wait Events
~~~~~~~~~~~~~~~~~                                             Wait     % Total
Event                                            Waits   Time (cs)  Wt Time
-------------------------------------------- ------------ ------------ -------
latch free                                        217,224       65,056   63.55
db file sequential read                            39,836       31,844   31.11
db file scattered read                              3,679        2,846    2.78
SQL*Net message from dblink                         1,186          870     .85
log file sync                                         830          775     .76
```

   前述の例で、最高順位の待機イベントは latch free イベントです。状況によっては、同程度のパーセンテージを持つイベントがいくつか存在する場合があります。このため、すべてのイベントが同じタイプのリソース要求（たとえば、すべてが I/O 関連イベント）に関連している場合に、追加の証拠を提供できます。

2. これらのイベントの待機数と平均待機時間を見てください。たとえば、I/O 関連イベントの場合、平均時間が I/O システムが低速であるかどうかを識別するのに役立つ場合もあります。次に、Statspack レポートの「Wait Event」の項から引用した、このデータの例を示します。

```
                                                          Avg
                                                 Total Wait   wait  Waits
Event                            Waits   Timeouts  Time (s)    (ms)   /txn
-------------------------- ------------ ---------- ---------- ------ ------
latch free                  5,560,989  2,705,969      26,117      5  827.7
db file sequential read       137,027          0       2,129     16   20.4
SQL*Net break/reset to cli      1,566          0       1,707   1091    0.2
```

3. トップの待機イベントは、次に調査する場所を識別します。表 22-1 に、一般的な待機イベントを示します。前述の例では、チェックする適切なデータがラッチ関連です。（高負荷 SQL を見ることもお薦めします。）
4. 待機イベントで指示される関連データを調べて、このデータから得られる他の情報を確認します。この情報が待機イベント・データとの一貫性を持っているかどうかを判断します。ほとんどの場合、パフォーマンス・ボトルネックの潜在的な原因に関する理論の展開を開始するためのデータは十分にあります。
5. この理論が有効であるかどうかを判断するには、利用可能な他の統計ですでに調べたデータの一貫性をクロスチェックします（適切な統計は問題により異なりますが、通常は V$SYSSTAT やオペレーティング・システム統計などにある、負荷のプロファイル関連のデータが含まれています）。他のデータとのクロスチェックを行って、展開中の理論を肯定または否定します。

待機イベントおよび潜在的な原因の表

表 22-1 に、待機イベントと考えられる原因との関連付けの他、次に検討するのに最も有益と思われる Oracle データの概要を示します。

表 22-1 待機イベントおよび潜在的な原因

待機イベント	一般的な領域	考えられる原因	検索 / 調査
buffer busy waits	バッファ・キャッシュ、DBWR	バッファ・タイプによって異なります。たとえば、索引ブロックの待機は、昇順に基づく主キーが原因である場合があります。	問題が発生している間に V$SESSION_WAIT を調べ、競合したブロックのタイプを判別します。
free buffer waits	バッファ・キャッシュ、DBWR、I/O	低速な DBWR（おそらく I/O に起因） 小さすぎるキャッシュ	オペレーティング・システム統計を使用して書込み時間を調べます。キャッシュが小さすぎることの証拠があるかどうかについてバッファ・キャッシュ統計をチェックします。
db file scattered read	I/O、SQL 文のチューニング	チューニングが適切ではない SQL 低速な I/O システム	V$SQLAREA を調べて、多数のディスク読込みを実行する SQL 文があるかどうかを確認します。I/O システムと V$FILESTAT をクロスチェックして、長い読込み時間をチェックします。
db file sequential read	I/O、SQL 文のチューニング	チューニングが適切ではない SQL 低速な I/O システム	V$SQLAREA を調べて、多数のディスク読込みを実行する SQL 文があるかどうかを確認します。I/O システムと V$FILESTAT をクロスチェックして、長い読込み時間をチェックします。
enqueue	ロック	エンキューのタイプにより異なる	V$ENQUEUE_STAT を参照します。

表 22-1 待機イベントおよび潜在的な原因（続き）

待機イベント	一般的な領域	考えられる原因	検索 / 調査
latch free	ラッチの競合	ラッチにより異なる	V$LATCH をチェックします。
log buffer space	ログ・バッファの I/O	小さいログ・バッファ 低速な I/O システム	V$SYSSTAT の統計 redo buffer allocation retries をチェックします。メモリーの構成の章の、ログ・バッファの構成の項をチェックしてください。オンライン REDO ログを格納するディスクをチェックして、リソースの競合の有無をチェックします。
log file sync	I/O、コミット過剰	オンライン・ログを格納する低速なディスク バッチされないコミット	オンライン REDO ログを格納するディスクをチェックして、リソースの競合の有無をチェックします。V$SYSSTAT から毎秒のトランザクション数（コミット数＋ロールバック数）をチェックします。

関連項目：

- 表 22-1 にリストした各イベントの詳細およびクロスチェックするその他の情報は、22-23 ページの「待機イベント」を参照してください。
- V$ ビューの問合せの詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。

追加された統計情報

対応する待機イベントを持たないパフォーマンスの問題を示すことのできる統計は多数あります。

REDO ログ領域要求統計

V$SYSSTAT 統計の redo log space requests は、サーバー・プロセスが REDO ログ・バッファ内の領域を待機するのではなく、オンライン REDO ログ内の領域を待機する必要があった回数を示します。この統計と待機イベントの大きい値は、LGWR ではなく、チェックポイント、DBWR、アーカイバ・アクティビティをチューニングする必要があることの標識として使用する必要があります。ログ・バッファのサイズを増やしても効果がありません。

読込み一貫性

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。

- 多数の小さいトランザクションがあり、変化が起こっている同じ表のバックグラウンド内でアクティブな長時間実行問合せが動作している場合、表の一貫読込みイメージを取得するために、問合せはこれらの変化を頻繁にロールバックする必要がある場合があります。次の V$SYSSTAT 統計を比較して、変化が発生しているかどうかを判断します。
  - consistent changes 統計は、そのブロック上で読込み一貫性を実行するために、データベース・ブロックがロールバック・エントリを適用した回数を示します。多数の consistent changes を生成するワークロードは、多数のリソースを使用する可能性があります。
  - consistent gets 統計は、一貫したモードでの論理読込み数をカウントします。
- 大きいロールバック・セグメントがほとんどない場合、システムはトランザクションがどの SCN にコミットされたかを正確に知るために、遅延ブロックのクリーンアウト時に長時間かけてトランザクション表をロールバックする可能性があります。次の V$SYSSTAT 統計の比率は、1 に近い値であることが必要です。

  ```
  ratio = transaction tables consistent reads undo records applied /
          transaction tables consistent read rollbacks
  ```

  解決策は、さらに多くの小さいロールバック・セグメントを作成するか、自動 UNDO 管理を使用することです。
- ロールバック・セグメントが十分にない場合、ロールバック・セグメント（ヘッダーまたはブロック）の競合が発生します。この問題は、次のようにすると明らかになります。
  - WAITS 数を V$ROLLSTAT 内の GETS 数と比較する方法。GETS に対する WAITS の比率は小さい値である必要があります。
  - V$WAITSTAT を調べて、CLASS 'undo header' のバッファに対して多数の WAITS があるかどうかを確認する方法。

  解決策は、さらに多くのロールバック・セグメントを作成することです。

継続行による表フェッチ

V$SYSSTAT 内の table fetch continued row 統計数をチェックして、移行行または連鎖行を検出できます。少数の連鎖行（1% 以下）は、システム・パフォーマンスに影響を与える可能性はほとんどありません。ただし、連鎖行のパーセンテージが大きいと、パフォーマンスに影響を与える可能性があります。

ブロック・サイズより大きい行の連鎖は避けられません。そのようなデータについては、ブロック・サイズのより大きい表領域の使用を考慮してください。

ただし、小さい行の場合は、適切な領域パラメータとアプリケーション設計を使用することで連鎖を回避できます。たとえば、キー値が入力され、かつその他のほとんどの列が NULL である行を、実際のデータで更新しないでください。その行のサイズが大きくなります。その場合は初めからデータが入力された行を挿入します。

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。これを行の移行と呼びます。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。これを行の連鎖と呼びます。行は挿入時にも連鎖される可能性があります。

移行と連鎖は、特に次の場合のパフォーマンスに影響があります。

- 移行と連鎖の原因となる UPDATE 文のパフォーマンスはよくありません。
- 移行行または連鎖行が追加入出力を実行するため、これらの行を選択する問合せをします。

LIST CHAINED ROWS 句を指定した 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。

注意： オプティマイザ統計を収集する場合は、ANALYZE ではなく DBMS_STATS パッケージを使用することをお薦めします。このパッケージでは、統計のパラレル収集、パーティション・オブジェクトのグローバル統計の収集およびその他の方法による統計の改善ができます。また、コストベース・オプティマイザは、最終的には DBMS_STATS で収集された統計のみ使用します。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、次のような場合、コストベース・オプティマイザに関連しない統計収集では、DBMS_STATS ではなく ANALYZE 文を使用する必要があります。

- VALIDATE 句または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックについての情報を収集する場合

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 UTLCHN1.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よって異なります。出力表は、CHAINED_ROWS 表と同じ列名、データ型およびサイズである必要があります。

移行行を回避するには、PCTFREE を増やします。ブロック内に使用可能な空き領域を多く残しておくと、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。頻繁に行が削除される表の場合は、データブロックに部分的に空き領域が生じることがあります。行を挿入し後から拡張する場合、行の削除されたブロックにその行が挿入されることがありますが、拡張の余地はありません。表を再編成すると主な空き領域を完全に空のブロックにできます。

注意： PCTUSED は、PCTFREE の反対の意味ではありません。

関連項目：

- PCTUSED の詳細は、『Oracle9*i* データベース概要』を参照してください。
- 表の再編成については、『Oracle9*i* データベース管理者ガイド』を参照してください。

解析関連の統計

アプリケーションの解析が長くなるほど、競合の可能性が高くなり、システムの待機時間が長くなります。parse time CPU（CPU 解析時間）が CPU タイムの大半を占める場合、SQL 文の実行ではなく解析に時間が消費されています。この場合には、アプリケーションはリテラル SQL を使用しているために SQL を共有できないか、または共有プールの構成が適切でないことがあります。

関連項目： 第 14 章「メモリーの構成と使用方法」

Oracle による解析に費やす時間の範囲を識別するために、多数の統計が利用できます。V$SYSSTAT から解析関連の統計の問合せを行います。その例を次に示します。

```
SELECT NAME, VALUE
  FROM V$SYSSTAT
 WHERE NAME IN (  'parse time cpu', 'parse time elapsed'
                , 'parse count (hard)', 'CPU used by this session' );
```

解析が問題となるかどうかの判断を助けるために計算される、様々な比率があります。

- parse time CPU / parse time elapsed

  この比率は、解析に費やされる CPU タイムのうちのどのくらいが、ラッチなどのリソースに対する待機ではなく、解析操作自体によるものかを示します。比率 1 は良好であり、経過時間が競合率の高いリソースを待機するために費やされなかったことを示します。

- parse time CPU / CPU used by this session

  この比率は、Oracle サーバー・プロセスで使用される CPU 全体のうちのどのくらいが解析関係の操作で費やされたかを示します。0（ゼロ）に近い値ほど良好であり、CPU の大部分が解析に費やされないことを示します。

待機イベント

V$SESSION_WAIT、V$SESSION_EVENT および V$SYSTEM_EVENT の各ビューは、どのようなリソースを待機したかに関する情報を表示し、また、構成パラメータ TIMED_STATISTICS が true に設定されている場合は、各リソースを待機した時間に関する情報も表示します。

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM か ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

パフォーマンス・チューニングを実行するときに、待機イベントと関連するタイミング・データを調査します。最大時間がリストされるイベントは、多くの場合、パフォーマンス・ボトルネックを顕著に示しています。

これらの 3 つのビューには、同じデータの関連する（ただし、異なる）ビューが含まれています。

- V$SESSION_WAIT は、現在の状態ビューです。このビューには、現在待機されているイベントか、各セッションで最後に待機されたイベントがリストされます。
- V$SESSION_EVENT は、各セッションで待機されるイベントの累積履歴をリストします。セッションが終了した後、そのセッションに対する待機イベント統計はこのビューから削除されます。
- V$SYSTEM_EVENT は、インスタンスの起動以降にインスタンス全体（すなわち、ロールアップされた、すべてのセッション待機イベント・データ）により待機されるイベントと回数をリストします。

V$SESSION_WAIT は現在の状態ビューであるため、V$SESSION_EVENT または V$SYSTEM_EVENT よりも細かい単位の情報も含まれています。このビューには、P1、P2 および P3 の 3 つのパラメータ列で現行イベントの追加識別データが含まれています。

たとえば、V$SESSION_EVENT はセッション 124（SID=124）が db file scattered read イベントで多く待機していたことを示すことはできますが、どのファイルとブロック番号かは示しません。ただし、V$SESSION_WAIT は P1 内のファイル番号、P2 内に読み込まれたブロック番号および P3 内に読み込まれたブロック数を示します（P1 と P2 により待機イベントがどのセグメントに対して発生するかを判断できます）。

この章では、V$SESSION_WAIT の使用例を中心に説明します。ただし、時間間隔のパフォーマンス・データの収集と、パフォーマンスおよび容量分析のためにこのデータを保存することをお薦めします。この形式のロールアップ・データの問合せは、Oracle Enterprise Manager Diagnostics Pack や Statspack などのツールで V$SYSTEM_EVENT ビューから行います。

最も一般的に発生するイベントについては、この章で、大 / 小文字を区別するアルファベット順にリストして説明します。調べる対象の他のイベント関連データも含まれています。各イベント名に使用する大 / 小文字区別は、V$SYSTEM_EVENT ビューでの表示と同一です。

> **関連項目：** 待機イベントの全リストは、『Oracle9*i* データベース・リファレンス』を参照してください。

SQL*Net

次のイベントは、データベース・プロセスがデータベース・リンクまたはクライアント・プロセスからの確認を待機していることを示します。

- SQL*Net break/reset to client
- SQL*Net break/reset to dblink
- SQL*Net message from client
- SQL*Net message from dblink
- SQL*Net message to client
- SQL*Net message to dblink
- SQL*Net more data from client
- SQL*Net more data from dblink
- SQL*Net more data to client
- SQL*Net more data to dblink

これらの待機がシステム上で、または応答時間の問題に対処しているユーザーに対して、待機時間の大部分を占めている場合、ネットワークまたは中間層がボトルネックになる可能性があります。

クライアント関連のイベントは、`SQL*Net message from client` イベントで説明したとおりに診断する必要があります。dblink 関連のイベントは、`SQL*Net message from dblink` イベントで説明したとおりに診断する必要があります。

SQL*Net message from client

これはアイドル・イベントですが、このイベントを診断に使用できるときに何が問題でないかを説明することが重要です。このイベントは、サーバー・プロセスがクライアント・プロセスからの処理を待機していることを示します。ただし、このイベントが、長い応答時間を経験しているユーザーの待機時間の大半を生成している状況がいくつかあります。その原因は、ネットワーク・ボトルネックまたはクライアント・プロセスでのリソース・ボトルネックのいずれかである可能性があります。

ネットワーク・ボトルネック ネットワーク・ボトルネックは、アプリケーションによってサーバーとクライアントとの間で多量の通信が発生し、ネットワークの待機時間（ラウンドトリップの時間）が長い場合に発生する可能性があります。症状には次のものがあります。

- このイベントに対する多数の待機
- データベースとクライアント・プロセスは、時間のほとんどがアイドル状態（ネットワーク通信待ち状態）です。

ネットワーク・ボトルネックを軽減するには、次のことを試行します。

- アプリケーションをチューニングしてラウンドトリップを軽減します。
- 待機時間を削減するためのオプション（たとえば、VSAT リンクとは反対の地上回線）を探索します。
- 通信量の大きいコンポーネントを待機時間の少ないリンクに移動するように、システム構成を変更します。

関連項目：『Oracle9*i* データベース・パフォーマンス・プランニング』

クライアント・プロセス上のリソース・ボトルネック クライアント・プロセスがリソースの大半を使用している場合、データベース内で実行できることはありません。症状には次のものがあります。

- 待機数は多くなくても、待機時間は長い。
- クライアント・プロセスは、リソースを多く使用している。

場合によっては、クライアント・プロセスで使用される CPU の量により、待機しているユーザーのトラッキングのための待機時間がわかります。この場合のクライアントという用語は、n 層アーキテクチャにおける、データベース・プロセス（中間層、デスクトップ・クライアント）以外の任意のプロセスを指します。

SQL*Net message from dblink

このイベントは、セッションがリモート・ノードに問合せを送り、データベース・リンクからの応答を待機している状態であることを意味します。この時間は、次の理由で増える可能性があります。

- ネットワーク・ボトルネック

  詳細は、22-25 ページの「SQL*Net message from client」を参照してください。

- リモート・ノードで問合せを実行するのに要する時間

  リモート・ノードで実行されている問合せを確認することが有益です。リモート・データベースにログインし、データベース・リンクで作成されたセッションを検索し、そのセッションで実行される SQL 文を調べます。

buffer busy waits

この待機は、複数のプロセスが同時にアクセスしようとしているバッファ・キャッシュ内に、いくつかのバッファがあることを示しています。バッファのクラスごとに、待機統計について V$WAITSTAT を問い合せます。buffer busy waits を持つ共通のバッファ・クラスとして、data block、segment header、undo header および undo block などがあります。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ー ファイル ID
- P2 ー ブロック ID

原因

考えられる原因を判別するには、V$WAITSTAT を問い合せて、競合対象のクラスのタイプを識別します。

```
SELECT class, count
  FROM V$WAITSTAT
 WHERE count > 0
  ORDER BY count DESC;
```

出力例:

```
CLASS                   COUNT
------------------ ----------
data block              43383
undo header             10680
undo block               5237
segment header            785
```

競合対象のセグメントおよびセグメント・タイプを識別するには、V$SESSION_WAIT から戻されたファイル ID の値とブロック ID の値（p1 および p2）を使用して DBA_EXTENTS を問い合せます。

```
SELECT segment_owner, segment_name
  FROM DBA_EXTENTS
 WHERE file_id = <&p1>
   AND <&p2> BETWEEN block_id AND block_id + blocks - 1;
```

アクション

必要なアクションは、競合対象のクラスと実際のセグメントにより異なります。

セグメント・ヘッダー 競合がセグメント・ヘッダー上にある場合、これは最も起こりうる空きリストの競合です。

ローカルに管理されている表領域で自動セグメント領域管理を行えば、PCTUSED、FREELISTS および FREELIST GROUPS の各パラメータを指定する必要はありません。可能であれば、手動領域管理から自動セグメント領域管理に切り替えます。

自動セグメント領域管理を使用できない（たとえば、表領域でディクショナリ領域管理を使用している）場合は、次の情報が関係します。

空きリストは、セグメントの様々なエクステント内に存在するブロックを通常含む空きデータ・ブロックのリストです。この空きリストのブロックには、PCTFREE より大きい領域があります。これは、既存の行への更新を予約するためのブロックのパーセントです。一般に、データベース・オブジェクトのプロセス空きリストに含まれるブロックは、PCTFREE と PCTUSED の制約を満たす必要があります。FREELISTS パラメータでプロセスの空きリスト数を指定します。FREELISTS のデフォルト値は 1 です。最大値はデータ・ブロック・サイズによって決まります。

そのセグメントの空きリストに対する現在の設定を見つけるには、次のことを実行します。

```
SELECT SEGMENT_NAME, FREELISTS
  FROM DBA_SEGMENTS
 WHERE SEGMENT_NAME = segment name
   AND SEGMENT_TYPE = segment type;
```

空きリストを設定します。または、空きリスト数を増やします。さらに空きリストを追加しても問題が軽減されない場合は、空きリスト・グループを使用します（このようにすると、単一のインスタンス内でも違いが出る可能性があります）。Oracle Real Application Clusters を使用する場合は、各インスタンスがそれ自体の空きリスト・グループを持っているかどうかを確認します。

関連項目：

- 自動セグメント領域管理、空きリスト、PCTFREE および PCTUSED については、『Oracle9*i* データベース概要』を参照してください。
- Oracle Real Application Clusters 環境での空きリスト・グループの使用については、『Oracle9*i* Real Application Clusters セットアップおよび構成』を参照してください。

データ・ブロック 表または索引（セグメント・ヘッダーではない）に対する競合がある場合、次のようにします。

- 選択性のない索引を使用した SQL 文をチェックしてください。
- 昇順インデックス（すなわち、多数のプロセスによって同じ点に挿入される索引。たとえば、キー値に順序番号ジェネレータを使用する索引）をチェックしてください。
- 自動セグメント領域管理を使用するか、空きリストを増やして複数のプロセスが同じブロック内に挿入されないようにすることを検討してください。

UNDO ヘッダー ロールバック・セグメント・ヘッダーに対する競合の場合

- 自動 UNDO 管理を使用しない場合は、さらにロールバック・セグメントを追加してください。

undo block ロールバック・セグメント・ブロックに対する競合の場合

- 自動 UNDO 管理を使用しない場合は、ロールバック・セグメント・サイズを大きくすることを検討してください。

db file scattered read

このイベントは、ユーザー・プロセスが SGA 内のバッファ・キャッシュにバッファを読み込み、物理 I/O コールが戻るまで待機することを意味します。db file scattered read は、データを複数の不連続メモリー位置に読み込むために散布読込みを発行します。散布読込みは通常、マルチブロック読込みです。全体スキャンの他、（索引の）高速全体スキャンでも行うことができます。

db file scattered read 待機イベントは、全表スキャンが発生していることを識別します。バッファ・キャッシュへの全表スキャンを実行すると、読み取られたブロックは物理的に相互に接近していないメモリー位置に読み込まれます。そのような読込みが散布読込みコールと呼ばれるのは、ブロックがメモリー全体に分散されているからです。対応する待機イベントが「db file scattered read（DB ファイル散布読込み）」と呼ばれるのは、このためです。バッファ・キャッシュへの全表スキャンのためのマルチブロック（最大で DB_FILE_MULTIBLOCK_READ_COUNT ブロック）読込みは、'db file scattered read' に対する待機として現れます。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― 絶対ファイル番号
- P2 ― 読み込まれるブロック
- P3 ― ブロック数（1 より大きい値である必要がある）

アクション

健全なシステムでは、物理読込み待機がアイドル待機後の最大待機である必要があります。ただし、小さい索引付きアクセスを行う必要のある運用（OLTP）システム上に、ダイレクト読込み待機（パラレル問合せを持つ全表スキャンを意味する）または db file scattered read 待機があるかどうかも確認してください。

システム上の過剰な I/O 負荷を示す他の要素として、次のものがあります。

- 低いバッファ・キャッシュ・ヒット率
- ユーザーへの応答時間を長くしている待機時間のほとんどを発生させている待機イベント

過剰な I/O の管理

過剰な I/O 待機を処理する方法はいくつかあります。有効性の順に示すと、これらの方法は次のとおりです。

1. SQL チューニングで I/O アクティビティを削減します。
2. ワークロードを管理することによって I/O を行う必要性を減らします。
3. さらに多くのディスクを追加して、各ディスクの I/O 数を削減します。
4. 既存のディスク間に I/O を再配分して I/O ホット・スポットを削減します。

関連項目： 第 15 章「I/O 構成および設計」

最初に行うことは、I/O を削減するためのチャンスを見つけることです。これらのイベントを待機するセッションを実行して SQL 文の調査を開始し、V$SQLAREA から多数の物理 I/O を実行する文を調べます。過剰な I/O を実行し、実行計画にマイナスの影響を与える可能性のある要因として、次のものがあります。

- 不適切に最適化された SQL
- 索引の欠落
- 表の高度な並列度（スキャン方向へオプティマイザを偏らせる）
- オプティマイザの正確な統計がないこと

不十分な I/O 分散

I/O を削減する他、ディスク間のファイルの I/O 分散も調べます。I/O はディスク間に均等に分散されていますか、あるいは、いくつかのディスク上にホット・スポットがありますか？データベースの I/O リクエストを満たすだけの十分な個数のディスクがありますか？

データベースによる I/O 操作（読込みと書込み）の総数を調べ、それと使用したディスク数を比較します。必ず、LGWR プロセスと ARCH プロセスの I/O アクティビティを含めてください。

I/O を待機しているセッションで実行された SQL 文の検索

次の問合せを使用して、ある時点でどのセッションが I/O を待機しているかを確認します。

```
SELECT s.sql_address, s.sql_hash_value
  FROM V$SESSION s, V$SESSION_WAIT w
 WHERE w.event LIKE 'db file%read'
   AND w.sid = s.sid ;
```

I/O を必要とするオブジェクトの検索

アクセスされるオブジェクトを検索するには、次の問合せを使用します。

```
SELECT segment_owner, segment_name
  FROM DBA_EXTENTS
 WHERE file_id = &p1
   AND &p2 between block_id AND block_id + blocks - 1 ;
```

db file sequential read

このイベントは、ユーザー・プロセスが SGA 内のバッファ・キャッシュにバッファを読み込み、物理 I/O コールが戻るまで待機することを意味します。このコールが散布読込みと異なるのは、順次読取りでは、データを連続メモリー領域に読み込むからです。順次読取りは通常、単一ブロック読込みです。

単一ブロック I/O は通常、索引を使用した結果です。エクステント境界のため、すなわち、バッファがすでにバッファ・キャッシュ内に存在するために全表スキャン・コールが単一ブロック・コールに切り捨てられることはほとんどありません。これらの待機は、'db file sequential read' としても表示されます。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― 絶対ファイル番号
- P2 ― 読み込まれるブロック
- P3 ― ブロック数（1 とする必要がある）

> **関連項目：** 過剰 I/O の管理、不十分な I/O 分散および I/O を発生させる SQL と I/O が実行されるセグメントの検索については、22-29 ページの「db file scattered read」を参照してください。

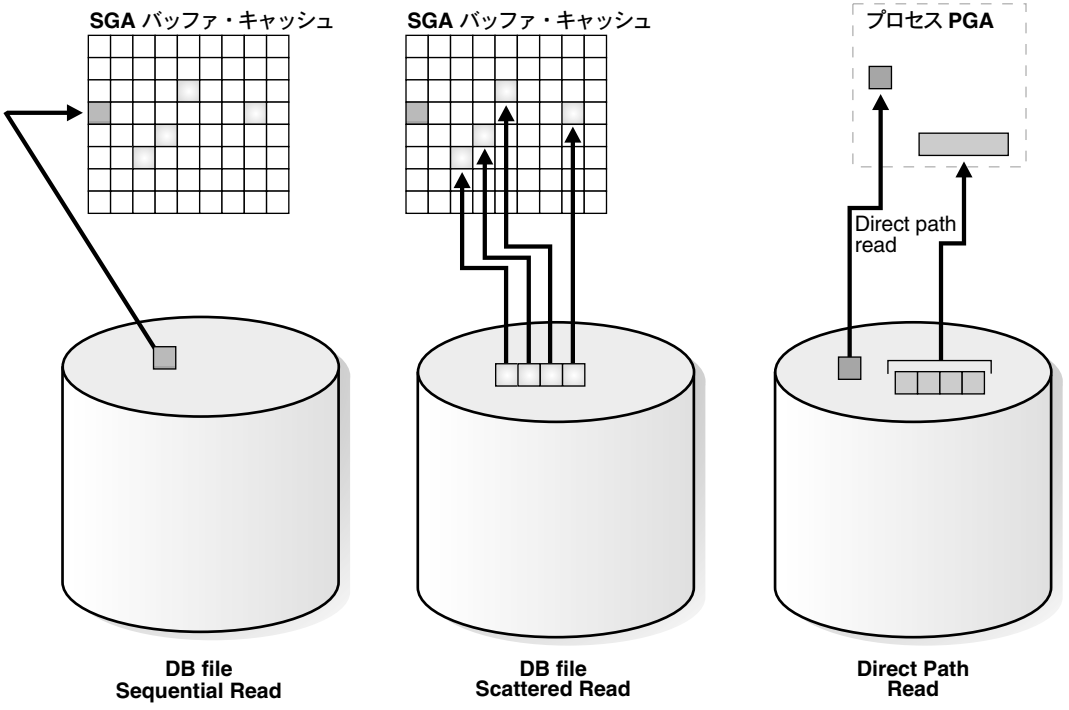
アクション

健全なシステムでは、物理読込み待機がアイドル待機後の最大待機である必要があります。ただし、パラレル問合せを持つ、全表スキャンに近いスキャンの必要な大きいデータ・ウェアハウス上に、db file sequential reads があるかどうかも確認してください。

図 22-1 は、次の待機イベント間の違いを示したものです。

- db file sequential read（1 つの SGA バッファに読み込まれる単一ブロック）
- db file scattered read（多数の不連続 SGA バッファに読み込まれるマルチブロック）
- direct read（PGA への単一またはマルチブロック読込み、SGA のバイパス）

図 22-1 scattered read、sequential read および direct read



direct path read および direct path read（LOB）

あるセッションがバッファをディスクから PGA に直接読み込む（SGA 内のバッファ・キャッシュとは反対）場合、このイベント上で待機します。I/O サブシステムが非同期 I/O をサポートしない場合、各待機は物理読込み要求に対応します。

I/O サブシステムが非同期 I/O をサポートする場合、このプロセスでは読込み要求の発行を、PGA にすでに存在するブロックの処理に重複させることができます。プロセスがディスクからまだ読み込まれていない PGA 内のブロックにアクセスしようとする場合、待機コールを発行し、このイベントの統計を更新します。したがって、待機数は必ずしも読込み要求数と同じではありません（'db file scattered read' および 'db files sequential read' とは異なります）。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― 読込みコールの File_id
- P2 ― 読込みコールの Start block_id
- P3 ― 読込みコール内のブロック数

原因

これは次の状況で発生します。

- ソートが大きすぎ、メモリー内に収まらないため、ディスクに移動する。ソートがメモリーに収まらない場合、そのソートの一部が直接ディスクに書き出されます。そのデータは、ダイレクト読込みで後から再度読み込まれます。
- パラレル・スレーブが、データのスキャンに使用される。
- サーバー・プロセスが、I/O システムがバッファを戻すより早くバッファを処理する。これは過負荷の I/O システムを示します。

アクション

file_id は、読込みが TEMP 表領域内のオブジェクトに対するものか、パラレル・スレーブによる全表スキャンに対するものかを示します。これは、大きいデータ・ウェアハウス・サイトに対する最大待機です。ただし、ワークロードが DSS ワークロードではない場合は、これが発生している理由を調べます。

ディスクでのソート 待機が発生しているセッションで、現在実行されている SQL 文を調べて、ソートの原因を確認します。ソートを生成する SQL 文を検索するために、V$TEMPSEG_USAGE を問い合せます。また、ソートのサイズを決定するセッションに関する V$SESSTAT からの統計を問い合せます。SQL 文をチューニングしてソートを削減できるかどうかを確認します。WORKAREA_SIZE_POLICY が MANUAL である場合、システム（ソートが大きすぎない場合）または個々のプロセスの SORT_AREA_SIZE を大きくすることを検討します。WORKAREA_SIZE_POLICY が AUTO である場合、PGA_AGGREGATE_TARGET を大きくするかどうかを調べます。

関連項目： 14-46 ページ「PGA 作業メモリーの構成」

全表スキャン 高い並列度で表を定義すると、全表スキャンをパラレル・スレーブで検索するようにオプティマイザを偏らせることができます。ダイレクト・パス読込みを使用して読み込むオブジェクトと、問合せコーディネータで実行される SQL 文をチェックします。全表スキャンがワークロードの有効な部分である場合は、I/O サブシステムが並列度に対して適したサイズに設定されているかどうかを確認します。

ハッシュ領域サイズ ハッシュ結合を呼び出す問合せ計画の場合、過剰な I/O は HASH_AREA_SIZE が小さすぎることから発生する可能性があります。WORKAREA_SIZE_POLICY が MANUAL である場合、システムまたは個々のプロセスの HASH_AREA_SIZE を大きくすることを検討してください。WORKAREA_SIZE_POLICY が AUTO である場合、PGA_AGGREGATE_TARGET を大きくするかどうかを調べます。

関連項目：

- 22-30 ページ「過剰な I/O の管理」
- 14-46 ページ「PGA 作業メモリーの構成」

direct path write

プロセスがバッファを PGA から直接書き込む（バッファ・キャッシュからバッファを書き込む DBWR とは反対）場合、プロセスは書込みコールが完了するまでこのイベント上で待機します。ダイレクト・パス書込みを実行できる操作には、ソートがディスクに移動するとき、パラレル DML 操作時、ダイレクト・パス INSERT、パラレル作成表の選択時、およびいくつかの LOB 操作があります。

ダイレクト・パス読込みと同様に、I/O サブシステムが非同期書込みをサポートする場合、待機数は発行された書込みコール数と同じではありません。セッションが PGA 内のすべてのバッファを処理し、I/O リクエストが完了するまで作業を継続できない場合、セッションは待機します。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― 書込みコールの File_id
- P2 ― 書込みコールの Start block_id
- P3 ― 書込みコール内のブロック数

原因

これは次の状況で発生します。

- ソートが大きすぎ、メモリー内に収まらないため、ディスクに移動する。
- オブジェクトを作成 / 移入するために、パラレル DML が発行される。

アクション

大きいソートについては、22-33 ページの「ディスクでのソート」を参照してください。

パラレル DML については、ディスク間の I/O 分散をチェックし、I/O サブシステムが並列度の大きさに対して適したサイズに設定されているかどうかを確認してください。

enqueue

エンキューは、データベース・リソースへのアクセスをシリアル化するロックです。このイベントは、セッションが別のセッションで保持されているロックを待機していることを示します。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― ロックの TYPE（または名前）および MODE
- P2 ― ロックのリソース識別子 ID1
- P3 ― ロックのリソース識別子 ID2

V$LOCK 列との比較をチェックします。

- V$LOCK.ID1 = P2
- V$LOCK.ID2 = P3

P1 列の次の SQL 変換を実行すると、同じ値が V$LOCK.TYPE 内に表示されます。

```
V$LOCK.TYPE = chr(bitand(P1,-16777216)/16777215)|| chr(bitand(P1,16711680)/65535)
```

エンキューが要求されているモードを取得するには、次の文を発行します。

```
request = mod(P1, 65536);
```

ロックおよびロック・ホルダーの検索

ロックを保持するセッションを見つけるには、V$LOCK の問合せを行います。イベント・エンキューを待機するすべてのセッションでは、REQUEST <> 0 を持つ V$LOCK 内に行があります。したがって、次の 2 つの問合せのいずれかを使用して、ロックを保持しているセッションとロックを待機しているセッションを検索します。

エンキュー待機がある場合は、次の文を使用することによりこれらを確認できます。

```
SELECT * FROM V$LOCK WHERE request > 0:
```

待機中のロックのホルダーおよびウェイタのみを表示するには、次の文を使用します。

```
SELECT DECODE(request,0,'Holder: ','Waiter: ')|| sid sess, id1, id2, lmode, request,
type
   FROM V$LOCK
 WHERE (id1, id2, type) IN (SELECT id1, id2, type FROM V$LOCK WHERE request>0)
   ORDER BY id1, request;
```

関連項目：

- V$LOCK の使用の詳細は、第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」を参照してください。
- エンキューの詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。

アクション

適切なアクションは、エンキューのタイプにより異なります。

ST エンキュー 競合されたエンキューが ST エンキューである場合、問題は動的な領域割当てにある可能性が最も高いと言えます。セグメントにそれ以上空き領域がない場合、Oracle はセグメントにエクステントを動的に割り当てます。このエンキューは、ディクショナリ管理表領域にのみ使用します。

このリソースに対する競合を解決するには、次のようにします。

- 一時（すなわち、ソート）表領域が一時ファイルを使用するかどうかを確認します。使用しない場合は、一時ファイルを使用するように切り替えます。
- 動的に拡張するセグメントが表領域に含まれている場合は、ローカル管理表領域を使用するように切り替えます。

  関連項目： TEMPFILE およびローカル管理表領域の詳細は、『Oracle9*i* データベース概要』を参照してください。

- ローカル管理表領域に切り替えることができない場合は、拡張するオブジェクトの次のエクステント・サイズを、一定の領域割当てを回避できる十分な大きさに変更することによって、ST エンキュー・リソースの使用量を減らすことができます。どのセグメントが常に拡張するかを判断するには、すべての SEGMENT_NAME について DBA_SEGMENTS ビューの EXTENTS 列を継続的に監視して、どのセグメントがどの程度の速さで拡張するかを識別します。
- セグメント内の領域を事前に割り当てます（たとえば、ALTER TABLE ALLOCATE EXTENT SQL 文でエクステントを割り当てる方法）。

HW エンキュー HW エンキューは、セグメントの最高水位標を超える領域の割当てをシリアル化する場合に使用します。

- V$SESSION_WAIT.P2 / V$LOCK.ID1 は表領域番号です。
- V$SESSION_WAIT.P2 / V$LOCK.ID2 は、領域が割り当てられるオブジェクトのセグメント・ヘッダーの相対 DBA です。

これがオブジェクトの競合点である場合、エクステントの手動割当てにより問題が解決されます。

TM エンキュー TM ロック上の待機の最も一般的な理由は、制約される列が索引付けされない場合の外部キー制約に関係している傾向があります。この問題を回避するには、外部キー列を索引付けします。

TX エンキュー これらのロックは、トランザクションがその最初の変更を開始するときに排他的に取得され、トランザクションが COMMIT または ROLLBACK を行うまで保持されます。

- モード 6 の TX の待機は、あるセッションが別のセッションですでに保持されている行レベル・ロックを待機しているときに発生します。これは、あるユーザーがある行を更新または削除し、別のセッションがその行を更新または削除する場合に発生します。

  解決方法は、ロックをすでに保持している最初のセッションに COMMIT または ROLLBACK を実行させることです。

- モード 4 の TX の待機は、セッションがブロック内で ITL（必要なトランザクション・リスト）スロットを待機している場合に発生する可能性があります。これは、セッションがそのブロック内の行をロックしても、1 つ以上の他のセッションの行が同じブロックでロックされ、そのブロック内に空き ITL スロットがない場合に発生します。通常は、Oracle が別の ITL スロットを動的に追加します。この操作は、ITL を追加するためのブロック内の空き領域が十分にない場合には実行できません。空き領域が十分にある場合、セッションはモード 4 で TX エンキューを持つスロットを待機します。

  解決方法は、表の INITTRANS または MAXTRANS を変更する（ALTER 文を使用するか、それより高い値で表を再作成する）ことによって使用可能な ITL の個数を増やすことです。

- モード 4 の TX の待機は、セッションが UNIQUE 索引内の潜在的な重複のためにセッションが待機している場合にも発生します。2 つのセッションが同じキー値を挿入しようとする場合、第 2 のセッションは ORA-0001 が発生したかどうかを確認するまで、待機する必要があります。

  解決方法は、ロックをすでに保持している最初のセッションに COMMIT または ROLLBACK を実行させることです。

- モード 4 の TX の待機は、共有ビットマップ・インデックス・フラグメントのためにセッションが待機している場合にも可能です。ビットマップ索引は、キー値と ROWID の範囲を索引付けします。ビットマップ索引内の各エントリは、実際の表中の多数の行をカバーできます。2 つのセッションが同じビットマップ・インデックス・フラグメントでカバーされる行を更新する場合、第 2 のセッションは、モード 4 で TX ロックを待機して、第 1 のトランザクションの `COMMIT` または `ROLLBACK` を待機します。
- モード 4 の TX の待機は、`PREPARED` トランザクションを待機している場合にも発生する可能性があります。

関連項目： 参照整合性およびデータの明示的ロックの詳細は、『Oracle9*i* アプリケーション開発者ガイド - 基礎編』を参照してください。

free buffer waits

この待機イベントは、サーバー・プロセスが空きバッファを検索できず、使用済みバッファを書き出すことによって空きバッファを作成するためにデータベース・ライターを転送したことを示します。使用済みバッファは、内容が変更されたバッファです。使用済みバッファは、DBWR がブロックをディスクに書き込み終えると、再利用するために解放されます。

原因

DBWR は、次の状況では使用済みバッファの書込みを継続できない場合があります。

- I/O システムが低速である。
- I/O システムが待機しているラッチなどのリソースがある。
- バッファ・キャッシュが小さすぎるため、DBWR はサーバー・プロセスのバッファのクリーニングに大部分の時間を費やす。
- バッファ・キャッシュが大きすぎるため、DBWR プロセスは要求を満たすだけのキャッシュ内のバッファを十分に解放できない。

アクション

このイベントが頻繁に発生する場合は、DBWR に対するセッション待機を調べて、DBWR を遅らせる原因があるかどうかを確認してください。

書込み セッションが書込みを待機している場合は、書込みを遅らせている原因を解明し、修正してください。次の点をチェックします。

- `V$FILESTAT` を調べて、書込みの大半が発生する場所を確認してください。
- I/O システムのホスト・オペレーティング・システム統計を調べてください。書込み時間は許容できるものですか？

I/O が低速である場合は、次のようにしてください。

- さらに高速な I/O 手段を使用して書込み時間を高速化することを検討してください。
- 多数のスピンドル（ディスク）とコントローラの間に I/O アクティビティを拡散してください。

> **関連項目：** I/O のバランス化については、第 15 章「I/O 構成および設計」を参照してください。

キャッシュが小さすぎる場合 キャッシュが小さすぎるために DBWR が非常にアクティブである可能性があります。バッファ・キャッシュ・ヒット率が低いかどうかを確認して、これが考えられる原因であるかどうかを調べます。また、V$DB_CACHE_ADVICE ビューを使用して、それより大きいキャッシュ・サイズが有利かどうかを判断します。

> **関連項目：** 14-6 ページ「バッファ・キャッシュのサイズ設定」

キャッシュが 1 つの DBWR に対して大きすぎる場合 キャッシュ・サイズが適切であり、I/O がすでに均等に分散されている場合は、非同期 I/O を使用するか、複数のデータベース・ライターを使用して、DBWR の動作を修正できます。

複数のデータベース・ライター（DBWR）・プロセスまたは I/O スレーブの検討

複数のデータベース・ライター・プロセスを構成したり、I/O スレーブを使用するのは、トランザクション・レートが高い場合や、バッファ・キャッシュ・サイズが大きすぎて単一の DBW*n* プロセスが負荷に耐えられない場合に役立ちます。

DB_WRITER_PROCESSES 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 までと、DBWa から DBWj）を構成できます。複数の DBWR プロセスを構成すると、書き込まれるバッファの識別に必要な作業が分散され、また、これらのプロセス間に I/O 負荷もが分散されます。複数のデータベース・ライター・プロセスは、複数の CPU を持つシステム（CPU 8 つにつき最低 1 つのデータベース・ライター）や、複数のプロセッサ・グループを持つシステム（最低でプロセス・グループと同数のデータベース・ライター）にお薦めします。

CPU の数またはプロセッサ・グループの数に基づいて、Oracle は、適切な DB_WRITER_PROCESSES のデフォルト設定を選択するか、ユーザー定義の設定を調整します。

DBWR_IO_SLAVES 複数の DBWR プロセスを使用することが実用的ではない場合、Oracle には複数のスレーブ・プロセス間に I/O 負荷を分散できる機能があります。DBWR プロセスは、ブロックを書き出すためにバッファ・キャッシュ LRU リストをスキャンする唯一のプロセスです。ただし、これらのブロックの I/O は I/O スレーブで実行されます。I/O スレーブの個数は、DBWR_IO_SLAVES パラメータで決定されます。

DBWR_IO_SLAVES は、（CPU が 1 つの場合など）複数の DB_WRITER_PROCESSES を使用できない場合を想定しています。I/O スレーブは、非同期 I/O が使用できないときにも役立ちます。書き込まれるキャッシュ内のブロックの識別を継続するために DBWR を解放することによって、複数の I/O スレーブが非ブロッキング要求をシミュレートするからです。一般的に、非同期 I/O がある場合、オペレーティング・システム・レベルの非同期 I/O をお薦めします。

DBWR の I/O スレーブは、初回の I/O リクエストが実行されるときに、データベースが開いた直後に割り当てられます。DBWR は、I/O の実行とは別に、DBWR 関連のすべての作業の実行を継続します。I/O スレーブは単に、DBWR のために I/O を実行します。バッチの書込みは I/O スレーブ間でパラレル化されます。

注意： DBWR_IO_SLAVES を実装するには、余分な共有メモリーを I/O バッファと要求キューに割り当てる必要があります。複数の DBWR プロセスを I/O スレーブと併用することはできません。I/O スレーブを構成すると、1 つの DBWR プロセスのみ起動するように設定されます。

複数の DBWR プロセスと I/O スレーブの選択 複数の DBWR プロセスを構成すると、単一の DBWR プロセスが、必要なワークロードに耐えられないときのパフォーマンスに有益です。ただし、複数の DBWR プロセスを構成する前に、非同期 I/O が使用でき、システム上に構成されるかどうかを確認します。システムが非同期 I/O をサポートしても現在使用されていない場合は、複数の DBWR プロセスを構成すると問題が軽減されるかどうかを確認するために非同期 I/O を使用可能にします。システムが非同期 I/O をサポートしない場合、または非同期 I/O がすでに構成され、DBWR ボトルネックが依然として存在する場合は、複数の DBWR プロセスを構成します。

注意： プラットフォーム上で非同期 I/O を使用できない場合は、DISK_ASYNCH_IO 初期化パラメータを FALSE に設定して非同期 I/O を無効にできます。

複数の DBWR を使用すると、バッファの収集と書込みがパラレル化されます。そのため、複数の DBW*n* プロセスは同じ数の I/O スレーブを使用する 1 つの DBWR プロセスよりもスループットを向上します。このため、複数の DBWR プロセスを優先して、I/O スレーブは使用されなくなりました。I/O スレーブは、複数の DBWR プロセスを構成できない場合のみ使用してください。

関連項目： チューニング・チェックポイントの詳細は、第 17 章「インスタンス・リカバリ・パフォーマンスの構成」を参照してください。

latch free

ラッチは、メモリー構造を保護するために Oracle で使用される下位レベルの内部ロックです。ラッチ解放イベントは、サーバー・プロセスがラッチを取得しようとしたときに更新され、ラッチは最初の試行で使用できなくなります。

> **関連項目：** ラッチおよび内部ロックの詳細は、『Oracle9*i* データベース概要』を参照してください。

アクション

このイベントは、ラッチ待機がシステム上の待機時間全体の重要な部分である場合か、または問題が発生している個々のユーザーに対してのみかを考慮してください。

- この待機イベントの原因を判断するには、競合するラッチを識別します。異なる目的に対し、様々なタイプのラッチが使用されます。たとえば、共有プール・ラッチは共有プール内のアクションを保護し、キャッシュ・バッファ LRU 連鎖はバッファ・キャッシュ内のアクションを保護します。
- 関連するリソースのリソース使用量を調べます。たとえば、ライブラリ・キャッシュ・ラッチの競合が大きい場合は、ハードとソフトの解析率を調べます。
- ラッチ競合が発生しているセッションの SQL 文を調べて、共通性があるかどうかを確認してください。

次の V$SESSION_WAIT パラメータ列をチェックします。

- P1 ― ラッチのアドレス
- P2 ― ラッチ番号
- P3 ― すでにスリープしたプロセスの回数

例 : 現在待機しているラッチの検索

```
SELECT n.name, SUM(w.p3) Sleeps
  FROM V$SESSION_WAIT w, V$LATCHNAME n
 WHERE w.event = 'latch free'
   AND w.p2 = n.latch#
 GROUP BY n.name;
```

表 22-2 ラッチ解放待機イベント

ラッチ	SGA 領域	考えられる原因	検索対象
共有プール、ライブラリ・キャッシュ	共有プール	文の再利用不足 バインド変数を使用しない文 アプリケーション・カーソル・キャッシュのサイズが不十分 各実行後に明示的にクローズされたカーソル 頻繁なログオン / ログオフ 基礎的なオブジェクト構造の変更（たとえば、切捨て） 小さすぎる共有プール	次の項目が高いセッション（V$SESSTAT 内） CPU 解析時間 所要解析時間 解析カウント（ハード）/ 実行カウントの比率 解析カウント（合計）/ 実行カウントの比率 次のカーソル（V$SQLAREA/V$SQL 内） PARSE_CALLS / EXECUTIONS の高い比率 EXECUTIONS = 1 WHERE 句のリテラルのみ異なる（すなわち、バインド変数を使用しない） 高い RELOADS 高い INVALIDATIONS 大きい（> 1MB）SHARABLE_MEM
キャッシュ・バッファ LRU 連鎖	バッファ・キャッシュ LRU リスト	過剰なバッファ・キャッシュ・スループット。たとえば、正しくない索引に繰り返しアクセスする非効率的な SQL（大きい索引レンジ・スキャン）、または多くの全表スキャン 実行中のワークロードに耐えられない DBWR。したがって、フォアグラウンド・プロセスは、空きバッファを検索するのにラッチを保持する時間が長くなります。 キャッシュが小さすぎる可能性がある	論理 I/O または物理 I/O が非常に多く、選択性のない索引が使用される文
キャッシュ・バッファ連鎖	バッファ・キャッシュ	**ホット・ブロック**と呼ばれる 1 つ（または少数）のブロックへのアクセスの繰返し	順序番号ジェネレータを使用せずに、番号を生成するために表の行を更新する順序番号生成コード きわめて類似している述語を使用して、選択性のない同一の索引をスキャンする非常に多くのプロセスから発生する索引リーフ・ブロックの競合 ホット・ブロックが属するセグメントの識別

共有プールとライブラリ・キャッシュ・ラッチの競合

共有プールまたはライブラリ・キャッシュ・ラッチの競合の主な原因は解析です。不要な解析および不要な解析の様々なタイプの識別に使用できる手法は多数あります。

非共有 SQL この方法では、リテラルがバインド変数と置換された場合に共有できる類似した SQL 文を識別します。その概念は次のいずれかです。

- 実行を 1 つのみ持つ SQL 文を手動で検査して、SQL 文が類似しているかどうかを確認します。

```
SELECT sql_text
  FROM V$SQLAREA
 WHERE executions < 4
 ORDER BY sql_text;
```

- 類似した文である可能性がある文をグループ化することによって、このプロセスを自動化します。これを行うには、同じものである可能性の高い SQL 文のバイト数を予測し、そのバイト数によって SQL 文をグループ化します。たとえば、次の例では、最初の 60 バイトまでが同一で、その後が異なる文をグループ化します。

```
SELECT SUBSTR(sql_text,1, 60), COUNT(*)
  FROM V$SQLAREA
 WHERE executions < 4
 GROUP BY SUBSTR(sql_text, 1, 60)
 HAVING COUNT(*) > 1;
```

再解析された共有可能な SQL V$SQLAREA ビューをチェックします。次の問合せを入力してください。

```
SELECT SQL_TEXT, PARSE_CALLS, EXECUTIONS
FROM V$SQLAREA
ORDER BY PARSE_CALLS;
```

PARSE_CALLS の値が指定の文の EXECUTIONS 値に近い場合は、その文の再解析を継続できます。解析コールが多い文をチューニングします。

セッション別 不要な解析コールが発生しているセッションを識別して、不要なコールを識別します。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 pa.sid, pa.value "Hard Parses", ex.value "Execute Count"
  FROM v$sesstat pa, v$sesstat ex
 WHERE pa.sid=ex.sid
   AND pa.statistic#=(select statistic#
       FROM v$statname where name='parse count (hard)')
   AND ex.statistic#=(select statistic#
```

```
      FROM v$statname where name='execute count')
AND pa.value>0;
```

結果としてすべてのセッションのリストとセッションで実行された解析の量が得られます。各システム識別子（SID）ごとに、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。

> **注意：** この問合せではインスタンス起動後のすべての解析コールがカウントされるため、解析率の高いセッションを検索することをお薦めします。たとえば、50 日間の接続が高い解析値を示すとしても、2 番目の 10 分間の接続の方がより速い速度で解析される場合があります。

出力は、次のようなものです。

```
   SID  Hard Parses  Execute Count
------  -----------  -------------
     7            1             20
     8            3          12690
     6           26            325
    11           84           1619
```

キャッシュ・バッファ LRU 連鎖 キャッシュ・バッファ LRU 連鎖のラッチは、キャッシュ内のバッファのリストを保護します。リストからバッファの追加、移動または削除を行う場合、ラッチを取得する必要があります。

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 に問い合せることによって検出できます。競合を回避するには、アプリケーションのチューニング、DSS ジョブのバッファ・キャッシュのバイパスまたはアプリケーションの再設計を検討してください。

キャッシュ・バッファ連鎖 cache buffers chains のラッチは、バッファ・キャッシュでバッファ・リストを保護する場合に使用します。これらのラッチは、バッファの検索、追加、およびバッファ・キャッシュからバッファを削除する場合に使用します。このラッチの競合は、競合度の高い（ホットな）ブロックが存在することを意味します。

頻繁にアクセスされるバッファ連鎖を識別し、競合するブロックを識別するには、V$LATCH_CHILDREN ビューを使用して cache buffers chains のラッチのラッチ統計を調べます。他の子ラッチと比較して、多くの GETS、MISSES および SLEEPS を持つ特定の cache buffers chains の子ラッチがある場合、これは子ラッチで競合します。

このラッチには、ADDR 列で識別されるメモリー・アドレスがあります。このラッチで保護されているブロックを識別するには、V$BH と結合された ADDR 列の値を使用します。たとえば、頻繁に競合されたラッチのアドレス（V$LATCH_CHILDREN.ADDR）を指定すると、ファイル番号とブロック番号の問合せが行われます。

```
SELECT file#, dbablk, class, state, TCH
  FROM X$BH
 WHERE HLADDR='address of latch';
```

X$BH.TCH は、バッファのタッチ・カウントです。X$BH.TCH の値が高い場合は、ホット・ブロックであることを示します。

各ラッチで保護されるブロックは多数あります。これらのバッファのうちの 1 つは、ホット・ブロックであると考えられます。TCH 値の高いブロックは、潜在的なホット・ブロックです。この問合せを複数回実行し、出力に一貫して表示されるブロックを識別します。ホット・ブロックを識別した後は、ファイル番号とブロック番号を使用して DBA_EXTENTS の問合せを行い、セグメントを識別します。

> **関連項目：** この手順については、22-31 ページの「I/O を必要とするオブジェクトの検索」を参照してください。

log buffer space

このイベントは、サーバー・プロセスがログ・バッファ内の空き領域を待機しているときに発生します。これは、LGWR が REDO を書き出すよりも、ユーザーがログ・バッファに REDO を書き込むほうが速いからです。

アクション

REDO ログ・バッファ・サイズを修正します。ログ・バッファのサイズがすでに適切なものである場合、オンライン REDO ログが存在するディスクが I/O 競合を受けないようにします。log buffer space 待機イベントは、REDO ログが存在するディスク上のディスク I/O 競合を示しているか、小さすぎるログ・バッファを示している可能性があります。REDO ログを含むディスクの I/O プロファイルをチェックして、I/O システムがボトルネックであるかどうかを調べます。I/O システムが問題ではない場合、REDO ログ・バッファが小さすぎる可能性があります。このイベントが問題にならなくなるまで、REDO ログ・バッファのサイズを大きくします。

log file switch

一般に発生する待機イベントは、次の 2 つです。

- Log file switch (archiving needed)
- Log file switch (checkpoint incomplete)

いずれのイベントでも、LGWR は次のオンライン REDO ログに切り替えることはできず、すべてのコミット要求はこのイベントを待機します。

アクション

`log file switch (archiving needed)` イベントの場合、アーカイバがタイムリにログをアーカイブできない理由を調べます。次の理由が考えられます。

- アーカイブ先に空き領域が不足している。
- アーカイバが REDO ログを十分高速に読み込めない（LGWR との競合）。
- アーカイバが十分高速に書込みを行えない（アーカイブ先での競合、または ARCH プロセスの数が不足している）。その他の可能性（ディスクが低速であったり、アーカイブ先がいっぱいなど）が除外された場合は、ARCn プロセス数の増加を検討してください。デフォルトは 2 です。
- 必須でリモートに転送されるアーカイブ・ログがある場合は、ネットワーク遅延によってこのプロセスが遅くなっていないか、またはエラーによって書込みが完了していないかをチェックしてください。

ボトルネックの性質により、I/O を再分散したり、アーカイブ先に領域を追加して問題を軽減することが必要な場合があります。`log file switch (checkpoint incomplete)` イベントの場合、次を実行してください。

- 過負荷または低速の I/O システムであるために DBWR が低速であるかどうかをチェックしてください。DBWR 書込み時間および I/O システムをチェックし、必要に応じて I/O を分散します。

  関連項目： 第 15 章「I/O 構成および設計」

- REDO ログが、少なすぎないか、または小さすぎないかをチェックしてください。REDO ログが少なすぎるか小さすぎる、あるいはその両方にあてはまる（たとえば、2 × 100k 個のログ）ために、DBWR がチェックポイントを完了する前に、すべてのログをサイクルする十分な REDO が生成される場合は、REDO ログのサイズまたは個数、あるいはその両方を増やします。

  関連項目： 13-5 ページ「REDO ログ・ファイルのサイズ指定」

log file sync

ユーザー・セッションがコミットする（またはロールバックする）場合、LGWR でセッションの REDO 情報を REDO ログ・ファイルにフラッシュする必要があります。COMMIT または ROLLBACK を実行するサーバー・プロセスは、REDO ログへの書込みが完了するまで、このイベントで待機します。

アクション

このイベントの待機がシステム上での長時間の待機か、応答時間の問題が発生しているユーザーまたはシステム上のユーザーによる長時間の待機を構成している場合は、平均待機時間を調べます。

平均待機時間は短いが、待機数が多い場合、アプリケーションは COMMIT をバッチ処理するのではなく、すべての INSERT 後にコミットできます。アプリケーションは、行ごとではなく 50 行後にコミットして待機を減らすことができます。

平均待機時間が多い場合は、LGWR に対するセッション待機を調べ、何を実行および待機するために多くの時間を費やしているかを調べます。待機が低速の I/O によるものである場合は、次のことを試行してください。

- REDO ログを含むディスク上の他の I/O アクティビティを削減するか、専用ディスクを使用します。
- 異なるディスク上に交互の REDO ログを設定して、LGWR に対するアーカイバの影響をできるだけ少なくします。
- REDO ログをさらに高速なディスクまたはさらに高速な I/O サブシステム（たとえば、RAID 5 から RAID 1 への切替え）に移動します。
- RAW デバイス（またはディスク・ベンダーが提供しているシミュレートされた RAW デバイス）を使用して書込みを高速化することを検討してください。
- アプリケーションのタイプにより異なりますが、1 行ごとではなく、*N* 行ごとにコミットして、COMMIT をバッチ処理することで、ログ・ファイルの同期が少なくて済みます。

rdbms ipc reply

このイベントは、バックグラウンド・プロセスのうちの 1 つから応答を待機する場合に使用します。

アイドル待機イベント

これらのイベントは、作業がないためにサーバー・プロセスが待機していることを示します。これは通常、ボトルネックが存在する場合にボトルネックがデータベース・リソースに対するものではないことを意味します。

チューニングの際、アイドル・イベントの大部分を無視することが必要なのは、アイドル・イベントがパフォーマンス・ボトルネックの性質を示さないためです。アイドル・イベントの中には、ボトルネックでないものを示す際に役立つものがあります。このタイプのイベントの例として、最も一般的に発生するアイドル待機イベントである SQL Net message from client があります。表 22-3 に、このアイドル・イベントと他のアイドル・イベント（およびそれらのカテゴリ）のリストを示します。

表 22-3 アイドル待機イベント

待機名	バックグラウンド・プロセス・アイドル・イベント	ユーザー・プロセス・アイドル・イベント	パラレル問合せアイドル・イベント	共有サーバー・アイドル・イベント	Oracle Real Application Clusters アイドル・イベント
dispatcher timer	.	.	.	×	.
lock manager wait for remote message	.	.	.	.	×
pipe get	.	×	.	.	.
pmon timer	×	.	.	.	.
PX Idle Wait	.	.	×	.	.
PX Deq Credit:need buffer	.	.	×	.	.
PX Deq Credit:send blkd	.	.	×	.	.
rdbms ipc message	×	.	.	.	.
smon timer	×	.	.	.	.
SQL*Net message from client	.	×	.	.	.
virtual circuit status	.	.	.	×	.

注意： Statspack をインストールする場合、アイドル・イベントのリストを含む STATS$IDLE_EVENT 表を問い合せることもできます。

関連項目： 各アイドル待機イベントの説明は、『Oracle9*i* データベース・リファレンス』を参照してください。

23

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を与える様々な接続モデルについて説明し、ネットワーク上の問題点を紹介します。

この章には次の項があります。

接続モデルについて

問題の原因を判別する際に使用するテクニックは構成によって異なります。共有サーバー構成または専用サーバー構成を持つことができます。

- 共有サーバー構成を持っている場合、LSNRCTL サービスは dispatchers をリストします。
- 専用サーバー構成を持っている場合、LSNRCTL サービスは dedicated servers をリストします。

接続記述子にパラメータ（SERVER = DEDICATED）を入れると、共有サーバー用に構成したデータベースと専用サーバーを接続できます。

共有サーバー構成

この項では、共有サーバー構成の設定について説明します。

ディスパッチャの登録 LSNRCTL 制御ユーティリティの services 文はこのリストに登録されているすべてのディスパッチャを表示します。このリストには、ディスパッチャ・プロセス ID が含まれています。アラート・ログをチェックして、ディスパッチャが正常に起動されていることを確認できます。

> **注意：** PMON はディスパッチャをリスナーに登録するには 1 分ほどかかります。

```
LSNRCTL> services
Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=helios)(PORT=1521)))
Services Summary...
Service "sales.us.acme.com" has 1 instance(s).
  Instance "sales", status READY, has 3 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:0 refused:0 state:ready
         LOCAL SERVER
      "D000" established:0 refused:0 current:0 max:10000 state:ready
         DISPATCHER <machine: helios, pid: 1689>
         (ADDRESS=(PROTOCOL=tcp)(HOST=helios)(PORT=52414))
      "D001" established:0 refused:0 current:0 max:10000 state:ready
         DISPATCHER <machine: helios, pid: 1691>
         (ADDRESS=(PROTOCOL=tcp)(HOST=helios)(PORT=52415))
The command completed successfully.
```

> **関連項目：** 出力モードの設定については『Oracle9*i* Net Services 管理者ガイド』を参照してください。

初期化パラメータ・ファイルの構成

- DISPATCHERS 行が正しく設定されていることを確認します。たとえば、次のようにします。

```
DISPATCHERS = "(DESCRIPTION=(ADDRESS=(PROTOCOL=TCP)
               (HOST=hostname)(PORT=1492)(queuesize=32)))
          (DISPATCHERS = 1)
          (LISTENER = alias)
          (SERVICE = servicename)
          (SESSIONS = 1000)
          (CONNECTIONS = 1000)
          (MULTIPLEX = ON)
          (POOL = ON)
          (TICK = 5)"
```

  次の属性のうち、1 つの属性のみが必要です。

  - PROTOCOL
  - ADDRESS
  - DESCRIPTION

  ADDRESS と DESCRIPTION は、PROTOCOL で追加ネットワーク属性の指定をサポートします。前述の例で、DISPATCHERS 行全体を（PROTOCOL=TCP）とすることができます。属性の DISPATCHERS、LISTENER、SERVICE、SESSIONS、CONNECTIONS、MULTIPLEX、POOL および TICKS はすべてオプションです。

- オプションの MAX_DISPATCHERS 行が正しく設定されていることを確認します。たとえば、次のようにします。

```
MAX_DISPATCHERS = 4
```

  この行は起動する合計ディスパッチャ数を反映している必要があります。

- オプションの MAX_SHARED_SERVERS 行が正しく設定されていることを確認します。たとえば、次のようにします。

```
MAX_SHARED_SERVERS = 5
```

  この行は、システムのピーク負荷を基準に、PMON が作成可能な合計共有サーバーの上限を設定します。これは、すべての要求がサービスを受けられる程度に大きな値に設定しますが、この値に達するとシステムがスワップするほど大きな値にはしないでください。このパラメータの目的はサーバーのスワップを防止することです。次のスクリプトを実行し、どの最高水位標が実行されているサーバー数に適しているかを調べて、MAX_SHARED_SERVERS をその値以上に設定します。

```
SELECT maximum_connections "MAX CONN", servers_started "STARTED", servers_
terminated "TERMINATED", servers_highwater "HIGHWATER" FROM V$SHARED_SERVER_
MONITOR;
```

- オプションの SHARED_SERVERS 行が正しく設定されていることを確認します。たとえば、次のようにします。

  ```
  SHARED_SERVERS = 5
  ```

  これは、データベースの起動時に起動された共有サーバーの合計数です。PMON が保持しようとした共有サーバーの合計数も表しています。これは、データベースがアクティブなときに使用されることが予想される共有サーバーの合計数になります。MAX_SHARED_SERVERS は、ピーク負荷を処理するためのものです。

接続のチェック LSNRCTL 制御ユーティリティの services コマンドを使用して、過剰な接続拒否がないか調べます。これが接続の問題になっていないか、リスナーのログ・ファイルを調べます。その例を次に示します。

```
LSNRCTL> services
Connecting to
(DESCRIPTION=(ADDRESS=(PROTOCOL=tcp)(HOST=helios)(PORT=1521)))
Services Summary...
Service "sales.us.acme.com" has 1 instance(s).
  Instance "sales", status READY, has 2 handler(s) for this service...
    Handler(s):
      "DEDICATED" established:11 refused:0 state:ready
         LOCAL SERVER
      "D000" established:565 refused:4 current:155 max:10000 state:ready
         DISPATCHER <machine: helios, pid: 5673>
         (ADDRESS=(PROTOCOL=tcp)(HOST=helios)(PORT=38411))
The command completed successfully.
```

通常の条件では、拒否された数はゼロになります。リスナーをシャットダウンし、再起動して統計を消去します。リスナーの再起動後にも拒否回数が増加している場合は、その接続が拒否されています。拒否回数がゼロのままであれば、現在トラブルシューティングしている問題は、接続拒否とは関係ありません。

１秒当たりの接続速度のチェック 接続拒否は様々な理由で発生します。リスナー・ログを調べ、接続速度を確認してください。リスナー・ログ・アナライザ・スクリプトを実行してチェックします。

リスナーはキュー・ベースのプロセスです。低レベル・プロトコル・スタックからの接続要求を受け取ります。これには、オペレーティング・システムの最大値に構成可能な一定のキュー・スタックがあります。これは１つの接続しか一度に処理できないので、このプロセスが１秒当たりで処理可能な接続数には限界があります。

接続要求が到着する速度がこの限界値を超えると、要求はキュー処理されます。キュー・スタックにも限界がありますが、これは設定できます。より多くのリスナー・プロセスがある場合は、各プロセスに対して行われる要求数は少なくなり、迅速に処理されます。

リスナー・キューの増加は、listener.ora ファイルで行われます。listener.ora ファイルには、別々の名前によって多くのリスナーを含めることができます。リストされている中の 1 つのみに問題があると想定します。そうでない場合は、この方法がすべての適用可能なリスナーに適用されます。リスナー・キューを増加するには、(queuesize = *number*) を listener.ora ファイルに追加します。たとえば、次のようにします。

```
listener =
     (address =
          (protocol = tcp)
          (host = sales-pc)
          (port = 1521)
          (queuesize = 20)
     )
```

関連項目：『Oracle9*i* Net Services 管理者ガイド』

リスナーを停止し、再起動してこの新しいパラメータを初期化します。現在、共有サーバー構成を実行していない場合は、実行することを検討してください。リスナーにとっては、専用サーバー構成より共有サーバー構成でクライアント要求を処理するほうが、処理が速くなります。

注意： 共有サーバーのディスパッチャは、接続要求も受信するので、キュー・サイズをチューニングすることにより得られる利点もあります。

最大キュー・サイズは特定オペレーティング・システムの最大サイズによって決まります。

ネットワークの問題の検出

この項では、ローカル・エリア・ネットワーク（LAN）とワイド・エリア・ネットワーク（WAN）のトラブルシューティングの方法を説明します。

動的パフォーマンス・ビューを使用したネットワークのパフォーマンスの向上

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の削減を試してみてください。ネットワークによって加えられる遅延の測定は困難な場合があります。

ネットワーク遅延を測定するには、次の 3 つの動的パフォーマンス・ビューが役立ちます。

- V$SESSION_EVENT
- V$SESSION_WAIT
- V$SESSTAT

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、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。

待機時間および帯域幅

パフォーマンスに関わるネットワークでの最も重要な側面は、待機時間と帯域幅です。

- 待機時間という用語は、時間の遅延を意味します。たとえば、デバイスがネットワークへのアクセスを要求した時間と送信許可を受信した時間までのギャップなどです。
- 帯域幅とは、ネットワーク・メディアあるいはプロトコルのスループット容量です。ネットワーク信号のバリエーションによりネットワークの劣化が発生する場合があります。劣化の原因には、ケーブルが長すぎたり、ケーブルの種類が間違っていることなどがあります。エレベータや蛍光灯などの外部からのノイズ発生源も問題の原因となります。

共有ネットワーク・トポロジ

ローカル・エリア・ネットワーク・トポロジ

- イーサネット
- 高速イーサネット
- 1 ギガビット・イーサネット
- トークン・リング
- FDDI
- ATM

ワイド・エリア・ネットワーク・トポロジ

- DSL
- ISDN
- フレーム・リレー
- T-1、T-3、E-1、E-3
- ATM
- SONAT

表 23-1 は、各種トポロジの最も共通した速度を示します。

表 23-1 帯域幅レーティング

トポロジまたは キャリア	帯域幅
イーサネット	10 メガビット / 秒
高速イーサネット	100 メガビット / 秒
1 ギガビット・ イーサネット	1 ギガビット / 秒
トークン・リング	16 メガビット / 秒
FDDI	100 メガビット / 秒
ATM	155 メガビット / 秒（OC3）、622 メガビット / 秒（OC12）
T-1（米国のみ）	1.544 メガビット / 秒
T-3（米国のみ）	44.736 メガビット / 秒
E-1（米国以外）	2.048 メガビット / 秒
E-3（米国以外）	34.368 メガビット / 秒
フレーム・リレー	最高でキャリア速度までが可能なコミット情報速度、ただし通常はこの速度にはしません。
DSL	これは、最高でキャリア速度まで可能です。
ISDN	これは、最高でキャリア速度まで可能です。これは通常、低速モデムで使用されます。
ダイアル・アップ・ モデム	56 キロビット / 秒。迅速なスループットのため、通常はデータの圧縮が伴います。

ネットワークの問題の解決

この項では、パフォーマンスを改善する手法と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。

- ネットワークのボトルネックの検出
- ネットワークのボトルネックの分析
- 配列インタフェースの使用方法
- セッション・データ・ユニットのバッファ・サイズの調整
- TCP.NODELAY の使用方法
- Connection Manager の使用

関連項目：『Oracle9*i* Net Services 管理者ガイド』

ネットワークのボトルネックの検出

ネットワークの問題を解決する際の最初の手順は、トポロジ全体を理解することです。キーワードに関する情報を可能なかぎり収集します。このような情報は通常、ネットワーク図として表示されます。図には、ローカル・エリア・ネットワークとワイド・エリア・ネットワークで使用されているネットワーク・トポロジのタイプが含まれています。各種ネットワーク・セグメントのアドレスも含まれています。

この情報を検討してください。明確なネットワーク・ボトルネックとして次のものがあります。

- ダイアルアップ・モデム（通常のモデムあるいは ISDN）を使用して時間が重要なデータにアクセスしている場合。
- T-1 上でフレーム・リレー・リンクが実行されていても、9.6 キロビット程度の CIR であり、1 秒当たり最高で 9.6 キロビットの送信信頼性のため、帯域幅の残りを使用すると、データが消失する可能性がある場合。
- 高速ネットワーク・チャネルからのデータが低速ネットワークを経由している場合。
- ネットワーク・ホップが多すぎる場合。ルータ 1 台が 1 ホップになります。
- 1 つのウェブ・サイトに対して 10 メガビットのネットワークになっている場合。

パフォーマンスの低下の原因となる問題は数多くあります。次のチェックリストに従ってください。

- Network Sniffer のトレースを取得します。
- 次の点をチェックします。
  - ネットワーク、クライアントまたはサーバーで帯域幅が超過しているかどうか。
  - イーサネットの衝突。

  - トークン・リングまたは FDDI リング・ビーコン。
  - ラント・フレームが多数あるか。
  - WAN リンクの安定性。
- フレーム・リレーの帯域幅使用率チャートを取得し、CIR が超過していないかチェックします。
- サービス品質あるいはパケット優先設定が実行されているか。
- ファイアウォールが途中にあるか。

何も問題がなかった場合は、クライアントからデータ・サーバーへのネットワーク・ルートを探してください。ネットワークでの所要時間を知っておくと、トランザクションに必要な時間を推測できます。クライアント / サーバー間の通信には多数の小さなパケットが必要です。ネットワークでの待機時間が長いと、要求を送出して応答を受信するまでの間隔が長くなり、トランザクション処理が遅くなります。

クライアントからサーバーへのルート追跡（traceroute または同等のコマンド）を使用して、パス内の各デバイスのアドレス情報を取得します。たとえば、次のようにします。

```
tracert usmail05
Tracing route to usmail05.us.oracle.com [144.25.88.200]over a maximum of 30 hops:
  1   <10 ms   <10 ms    10 ms  whq1davis-rtr-749-f1-0-a.us.oracle.com
[144.25.216.1]
  2   <10 ms   <10 ms   <10 ms  whq4op3-rtr-723-f0-0.us.oracle.com [144.25.252.23]
  3   220 ms   210 ms   231 ms  usmail05.us.oracle.com [144.25.88.200]

Trace complete.
```

各デバイスを順に ping してタイミングをとります。大きなパケットを使用して最も遅い時間を調べます。ルータがパケットの分解と組立に時間を費やすことのないよう、必ず「don't fragment bit」を設定してください。パケット・サイズが 1472 であることに注意してください。これは、イーサネット用です。イーサネット・パケットのサイズは 1536 オクテット（実際には 8 ビット・バイト）です。ICPM パケット（これは ping 本来の用途）には 64 オクテットのヘッダーがあります。低速になっていると思われる場合はこの領域を確認します。たとえば、次のようにします。

```
ping -l 1472 -n 1 -f 144.25.216.1
Pinging 144.25.216.1 with 1472 bytes of data:
Reply from 144.25.216.1: bytes=1472 time<10ms TTL=255

ping -l 1472 -n 1 -f 144.25.252.23
Pinging 144.25.252.23 with 1472 bytes of data:
Reply from 144.25.252.23: bytes=1472 time=10ms TTL=254

ping -l 1472 -n 1 -f 144.25.88.200
Pinging 144.25.88.200 with 1472 bytes of data:
Reply from 144.25.88.200: bytes=1472 time=271ms TTL=253
```

前述の例では、追跡ルートを評価しています。ワークステーションから 144.25.216.1 に、144.25.216.1 から 144.25.252.23 に、144.25.252.23 から 144.25.88.200 に ping を実行するのが理想的です。これにより、移動した各セグメントの待機時間が正確に示されます。

ネットワークのボトルネックの分析

この項は、ネットワーク・ボトルネックの問題を特定する際に役立ちます。

問題が Oracle Net またはネットワークのいずれにあるかの判定

Oracle Net トレース機能は、エラーが Oracle 固有のものであるか、あるいはオペレーティング・システムが Transparent Network Substrate（Oracle TNS レイヤー）に渡す条件によるものかを明確にします。

解決しようとしている問題を抱えていることが疑わしい Oracle サーバー、リスナーおよびクライアントで Oracle Net トレース機能を有効にします。

サーバーでトレース機能を有効にするには、このサーバーの sqlnet.ora ファイルを探し、次の行を作成します。

```
TRACE_TIMESTAMP_SERVER = ON
TRACE_LEVEL_SERVER = 16
TRACE_UNIQUE_SERVER = ON
```

クライアントでトレース機能を有効にするには、このクライアントの sqlnet.ora ファイルを探し、ファイルに次の行を作成します。

```
TRACE_TIMESTAMP_CLIENT = ON
TRACE_LEVEL_CLIENT = 16
TRACE_UNIQUE_CLIENT = ON
```

リスナーでトレース機能を有効にするには、listener.ora ファイルを探し、ファイルに次の行を作成します。

```
TRACE_TIMESTAMP_listener_name = ON
TRACE_LEVEL_listener_name = 16
```

注意： TRACE_TIMESTAMP_*x* パラメータはオプションですが、デバッグを向上させるために組み込む必要があります。

クライアントとサーバーでトレースを生成するように、問題を再現します。ここで、生成されたトレースを分析します。

関連項目：

- Oracle Net トレース機能を有効にする詳細な手順は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。
- トレース・ファイルに示される Oracle Net エラーの定義については、『Oracle9*i* データベース・エラー・メッセージ』を参照してください。

問題はネットワークにあり、Oracle Net ではない場合、次の事項を調べる必要があります。

- 問題はローカル・ネットワークの 1 つのロケーションでのみ発生しているかどうか。
- 問題は WAN の 1 つの領域でのみ発生しているかどうか。

たとえば、データ・センターの存在するビル内では、システムは正常ですが、数マイル離れたビル内では低速になっていることがあります。

Oracle のエラー・コードすべてが、Oracle 固有のトラブルを表しているわけではありません。ORA-3113 は最もよく発生するエラーですが、基盤となっているネットワークに問題があることを示します。

注意： サーバー上でトレースを有効にすると、大量のトレース・ファイルを生成します。これを防ぐため、自身をトレースする個別の環境をセットアップします。この構成は、専用接続に有効です。まず、サーバーのオペレーティング・システムに Oracle のソフトウェア所有者としてログインします。一時ディレクトリを作成し、これから作成される構成ファイルとトレース・ファイルを保管します。sqlnet.ora、listener.ora および tnsnames.ora をそのディレクトリにコピーします。sqlnet.ora ファイルを編集し、トレース機能を有効にします。次の行を sqlnet.ora ファイルに追加します。

```
TRACE_DIRECTORY_SERVER = temporary_directory_just_created
```

次に listener.ora ファイルを修正して、リスナー・ポート（TCP、その他のプロトコルの場合は類似のテクニックを使用）を未使用ポートに変更します。このテストで使用する接続文字列用にクライアントの tnsnames.ora ファイルに同様の修正を行う必要があります。

TNS_ADMIN 環境をテンポラリ・ディレクトリをポイントするように設定します。リスナーを起動します。これで、新しいリスナーへのすべての接続がこのディレクトリに Server traces を送信します。問題を再現します。

Oracle エラー・メッセージを取得しながら、トレース・ファイルを調べエラーを探します。トラブルシューティング・バグの場合は、Oracle Net トレース分析が問題を完全に探し出すまでに一定の時間がかかります。ただし、高レベルの簡易トレース分析は簡単です。

問題がクライアント上またはサーバー上（Oracle Net 上）のいずれにあるかの判定

問題が Oracle Net にある場合、Oracle Net トレースを使用して問題が存在する場所を示します。トレース・ファイルにエラーがある場合は、エラーはクライアントのトレースにのみ現れるか、サーバーのトレースにのみ現れるかあるいは両方に現れるかを調べます。

クライアント・トレースのみのエラー

問題はクライアントにあります。ただし、ORA-3113 エラーまたは ORA-3114 エラーを取得している場合は、問題はサーバーにあります。

サーバー・トレースあるいはリスナー・トレースのみのエラー

問題はサーバーにあります。ただし、ORA-3113 エラーまたは ORA-3114 エラーを取得している場合は、問題はクライアントにあります。

すべてのエラー : クライアント、サーバーおよびリスナーのトレース

ORA-3113 エラーまたは ORA-3114 エラーを取得している場合は、問題はネットワークにあります。まず、サーバーのトラブルシューティングを行います。サーバーが正常であれば、クライアントに障害があります。

サーバーが共有サーバー用に構成されているかどうかのチェック

共有サーバー・アーキテクチャのトラブルシューティングは、さらに複雑である可能性があります。初期化パラメータ・ファイルに共有サーバーのパラメータが設定されているか確認します。また、共有サーバー・プロセスが存在していないかオペレーティング・システムを調べます。

ora_d000 や ora_d001 などの名前を探してディスパッチャをチェックします。たとえば、次のようにします。

```
ps -ef | grep ora_d
```

ora_s000 や ora_s001 などの名前を探して共有サーバーをチェックします。たとえば、次のようにします。

```
ps -ef | grep ora_s
```

関連項目：

- 共有サーバーのチューニングの詳細は、23-2 ページの「共有サーバー構成」を参照してください。
- 共有サーバーの概念とパラメータの詳細は、『Oracle9*i* データベース概要』および『Oracle9*i* Net Services 管理者ガイド』を参照してください。

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用してネットワーク・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、『Oracle Call Interface プログラマーズ・ガイド』を参照してください。

セッション・データ・ユニットのバッファ・サイズの調整

ネットワーク上にデータを送信する前に、Oracle Net はデータをセッション・データ・ユニット（SDU）にバッファリングします。SQL Net は、バッファがいっぱい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。大量のデータを取り出すときまたはパケット・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データの取出しを高速化できることがあります。

最適な SDU サイズは、標準転送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。

Oracle Net Configuration 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です。

関連項目： 『Oracle9*i* Net Services 管理者ガイド』

TCP.NODELAY の使用方法

セッションが確立すると、Oracle Net はデータをパッケージ化し、パケットを使用してサーバーとクライアント間でデータを送信します。デフォルトでは、さらに頻繁にネットワークにパケットをフラッシュさせる `TCP.NODELAY` パラメータが有効になります。Oracle Net 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： `TCP.NODELAY` 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。

Connection Manager の使用

Oracle Net では、Connection Manager を使用して多重化によりシステム・リソースを確保することができます。多重化とは、転送先サーバーへの単一のトランスポート接続を経由して、複数のクライアント・セッションを送り込むことです。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。これが適用されるのは、共有サーバー構成のみです。また、Connection Manager を使用して、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： Connection Manager の詳細は、『Oracle9*i* Net Services 管理者ガイド』を参照してください。

第 VI 部

パフォーマンス関連の参照情報

第 VI 部では、動的パフォーマンス・ビューおよび待機イベントに関する参照情報を示します。

第 VI 部には次の章が含まれます。

- 第 24 章「チューニングに使用する 動的パフォーマンス・ビュー」

24

チューニングに使用する動的パフォーマンス・ビュー

この章では、システムのチューニングとパフォーマンスの問題の調査に役立つ動的ビューに関する詳細な情報を示します。

この章には次の項があります。

関連項目： 動的パフォーマンス・ビューおよびその列の全リストは、『Oracle9*i* データベース・リファレンス』を参照してください。

動的パフォーマンス表

Oracle は稼動中に、現行のデータベース・アクティビティを記録する一連の仮想的な表をメンテナンスしています。これらの表は Oracle が作成したもので、動的パフォーマンス表と呼ばれます。

データベース管理者は動的パフォーマンス表のビューの問合せと作成を行い、他のユーザーにこれらのビューへのアクセス権を付与できます。これらのビューは、データベース管理者が変更または削除できないため、固定ビューと呼ばれることもあります。

SYS は、動的パフォーマンス表を所有します。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のみがこれらのビューを利用できます。ビュー名はすべて V_$ から始まります。ビューはこれらの表に対して作成され、次にビューのためにパブリック・シノニムが作成されます。シノニム名は V$ から始まります。

各ビューは、次のカテゴリのうちの 1 つに属します。

- 現在の状態ビュー
- カウンタ / アキュムレータ・ビュー
- 情報ビュー

現在の状態ビュー

表 24-1 にリストしたビューは、現在システム上で発生している内容を表示します。

表 24-1 現在の状態ビュー

固定ビュー	説明
V$LOCK	インスタンス上で現在保持 / 要求されているロック
V$LATCHHOLDER	ラッチを保持するセッション / プロセス
V$OPEN_CURSOR	インスタンス上のセッションでオープンしたカーソル
V$SESSION	インスタンスに現在接続されているセッション
V$SESSION_WAIT	セッションが現在待機中の異なるリソース

カウンタ / アキュムレータ・ビュー

これらのビューは、インスタンス / セッションの起動以降に特定のアクティビティが発生した回数を追跡します。ビューから直接選択して起動以降のアクティビティを確認します。

一定の時間間隔で発生しているアクティビティを対象とする場合は、時間間隔の前後にスナップショットを取得すると、それらの 2 つのスナップショット間の差分がその時間間隔のときのアクティビティを提供します。これは、sar、vmstat、iostat などのオペレーティング・システム・ユーティリティの機能に似ています。Statspack や BSTAT/ESTAT などの Oracle が提供するツールは、この差分を計算して一定の間隔でアクティビティのレポートを提供します。

注意： スナップショットはシステムの起動直後ではなく、安定状態のときに作成してください。システムの開始時は余分なオーバーヘッドが発生するため、安定状態のシステムのパフォーマンスを正確に反映しない場合があります。

表 24-2 セッション起動以降のサマリー

固定ビュー	説明
V$DB_OBJECT_CACHE	共有プール内のオブジェクト・レベル統計
V$FILESTAT	I/O アクティビティのファイル・レベル・サマリー
V$LATCH	ラッチ・アクティビティ・サマリー
V$LATCH_CHILDREN	子ラッチのラッチ・アクティビティ
V$LIBRARYCACHE	共有プールの名前空間レベル・サマリー
V$LIBRARY_CACHE_MEMORY	ライブラリ・キャッシュの現在のメモリー使用の要約（ライブラリ・キャッシュ・オブジェクト型別）
V$MYSTAT	独自セッションのリソース使用量サマリー
V$ROLLSTAT	ロールバック・セグメント・アクティビティ・サマリー
V$ROWCACHE	データ・ディクショナリ・アクティビティ・サマリー
V$SEGMENT_STATISTICS	セグメント・レベル統計をリアルタイム監視するための、ユーザーが扱いやすい DBA ビュー
V$SEGSTAT	セグメント・レベル統計をリアルタイム監視するための高効率性ビュー
V$SESSION_EVENT	現行セッションに対するすべての待機のセッション・レベル・サマリー

表 24-2 セッション起動以降のサマリー（続き）

固定ビュー	説明
V$SESSTAT	セッション起動以降のリソース使用量のセッション・レベル・サマリー
V$LIBRARY_CACHE_MEMORY	共有プールの LRU リスト・メカニズムのシミュレーション
V$SQL	V$SQLAREA の子カーソル詳細
V$SQLAREA	文 / 無名ブロックの共有プール詳細
V$SYSSTAT	リソース使用量のサマリー
V$SYSTEM_EVENT	待機リソースのインスタンス全体のサマリー
V$UNDOSTAT	UNDO 使用量のヒストグラム。各行は 10 分間隔を表します。
V$WAITSTAT	ブロック・クラス別バッファ待機の分析

情報ビュー

情報ビューでは、情報は現在の状態ビューほど動的ではありません。したがって、現在の状態のビューほど頻繁に情報の問合せを行う必要はありません。

表 24-3 情報ビュー

固定ビュー	説明
V$MTTR_TARGET_ADVICE	FAST_START_MTTR_TARGET を設定して MTTR アドバイザで収集したアドバイザ情報
V$PARAMETER および V$SYSTEM_PARAMETER	セッションのパラメータ値 インスタンス全体のパラメータ値
V$PROCESS	サーバー・プロセス（バックグラウンドおよびフォアグラウンド）
V$SEGSTAT_NAME	セグメント・レベル統計の統計プロパティ・ビュー
V$SQL_PLAN	最近実行されたカーソルの実行計画
V$SQL_PLAN_STATISTICS	実行計画の各操作の実行統計
V$SQL_PLAN_STATISTICS_ALL	V$SQL_PLAN の情報と、V$SQL_PLAN_STATISTICS および V$SQL_WORKAREA の実行統計が連結されます

表 24-3 情報ビュー（続き）

固定ビュー	説明
V$SQLTEXT	共有プール内の文の SQL テキスト
V$STATISTICS_LEVEL	STATISTICS_LEVEL 初期化パラメータで制御される統計またはアドバイザのステータス

動的パフォーマンス・ビューの説明

この項では、いくつかの動的パフォーマンス・ビューの詳細を説明します。

V$DB_OBJECT_CACHE

このビューは、ライブラリ・キャッシュ（共有プール）内のオブジェクトのオブジェクト・レベル統計を表示します。このビューは V$LIBRARYCACHE より多くの詳細を示し、共有プールでアクティブ・オブジェクトを検索するのに便利です。

V$DB_OBJECT_CACHE の有益な列

この表の列の大半は、現在の状態の情報を示します。

- OWNER: オブジェクトの所有者
- NAME: オブジェクト名（無名ブロック / カーソルのための SQL テキストの最初の 1000 文字）
- TYPE: オブジェクトのタイプ（たとえば、シーケンス、プロシージャ、ファンクション、パッケージ、パッケージ本体、トリガー）
- KEPT: オブジェクトが共有プール内に確保されているかどうか（YES、NO）
- SHARABLE_MEM: 使用する共有メモリーの量
- PINS: このオブジェクトを現在実行しているセッション
- LOCKS: このオブジェクトを現在ロックしているセッション

瞬間的な状態の列

次の列は、オブジェクトが最初にロードされた後のオブジェクトに関する統計を維持します。

- LOADS: このオブジェクトをロードする必要のあった回数
- INVALIDATIONS: このオブジェクトが無効にされた回数

例 24-1 共有プール実行およびメモリー使用量のサマリー

次の問合せでは、各種オブジェクト間の共有プール・メモリーの分布を示します。また、DBMS_SHARED_POOL.KEEP() プロシージャを使用して、オブジェクトが共有プール内に確保されたかどうかも示されます。

```
SELECT type, kept, COUNT(*), SUM(sharable_mem)
FROM V$DB_OBJECT_CACHE
GROUP BY type, kept;
```

例 24-2 大量の負荷を持つオブジェクトの検索

```
SELECT owner, name sharable_mem, kept, loads
FROM V$DB_OBJECT_CACHE
WHERE loads > 1
OR invalidations > 0
ORDER BY loads DESC;
```

例 24-3 確保されていない大きいオブジェクトの検索

次の問合せでは、大量のメモリーを使用して、すべてのオブジェクトを検索します。オブジェクトは、DBMS_SHARED_POOL.KEEP() を使用して確保できます。

```
SELECT owner, name, sharable_mem, kept
FROM V$DB_OBJECT_CACHE
WHERE sharable_mem > 102400
AND kept = 'NO'
ORDER BY sharable_mem DESC;
```

V$FILESTAT

このビューは、各ファイルの物理 I/O リクエストに関する情報を保持します。これは、ボトルネックが I/O 関連の場合、I/O アクティビティが発生している場所を特定する場合に有益です。V$FILESTAT は、データベース I/O（ログ・ファイル I/O ではなく）に関する次の情報を示します。

- 物理的な読込みと書込みの回数
- 読込みおよび書込みを行ったブロック数
- 読込みと書込みの I/O 時間の合計

これらの数字は、インスタンス起動以降のアクティビティを反映します。2 つのスナップショットが作成された場合、統計の差は時間間隔の I/O アクティビティを示します。

V$FILESTAT の有益な列

- FILE#: ファイルの番号
- PHYRDS: 実行された物理読込み回数
- PHYBLKRD: 読み込まれた物理ブロック数
- PHYWRTS: 実行された物理書込み回数
- PHYBLKWRT: 書き込まれた物理ブロック数

V$FILESTAT の注意

- 物理読込みとブロック読込みは、マルチブロック読込みコールのために異なる可能性があります。
- 物理書込みとブロック書込みは、プロセスによるダイレクト書込みのために異なる可能性があります。
- Sum（物理ブロック読込み）は、V$SYSSTAT からの physical reads と密接な相関関係があります。
- Sum（物理ブロック書込み）は、V$SYSSTAT からの physical writes と密接な相関関係があります。
- 読込み（ダイレクト読込みと、バッファ・キャッシュへ）は、サーバー・プロセスで行われます。バッファ・キャッシュからの書込みは、DBWR でのみ処理されます。ダイレクト書込みは、サーバー・プロセスで処理されます。

V$FILESTAT の結合列

表 24-4 は、V$FILESTAT の結合列をリストしたものです。

表 24-4 V$FILESTAT の結合列

列	ビュー	結合列
FILE#:	DBA_DATA_FILES	FILE_ID

例 24-4 Oracle データ・ファイル I/O のチェック

次の問合せでは、アプリケーションの実行中のある期間にわたって物理読込みと物理書込みの値を監視します。

```
SELECT NAME, PHYRDS, 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・ファイルの名前も取り出します。出力例は次のようになります。

```
NAME                                        PHYRDS    PHYWRTS
------------------------------------------- ---------- ----------
/oracle/ora81/dbs/ora_system.dbf                  7679       2735
/oracle/ora81/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、全体の I/O を統計が収集された時間間隔で割って、ディスクごとの I/O 発生率も求めてください。

注意： Oracle は正確に読込みおよび書込み回数を記録しますが、UFS（Unix file system）で実行しているデータベースは、実際のディスク・アクセスを反映しない場合があります。たとえば、読込み回数が、実際のディスク読込みではなく、UFS キャッシュ・ヒットを反映する場合があります。ただし、RAW デバイスに関しては読込みおよび書込み回数は正確になります。さらに、書込み回数はバッチ単位でしか記録されず、同じバッチ内のブロックはすべて書込み I/O の終了後の同じ回数が記録されます。

例 24-5 多数のマルチブロック読込みを持つファイルの検索

次の例は、多数のスキャンがヒットする可能性のある表領域の検索に便利です。

```
SELECT t.tablespace_name
      ,SUM(a.phyrds-b.phyrds)
       /MAX(86400*(a.snap_date-b.snap_date)) "Rd/sec"
      ,SUM(a.phyblkrd-b.phyblkrd)
       /greatest(SUM(a.phyrds-b.phyrds),1) "Blk/rd"
      ,SUM(a.phywrts-b.phywrts)
       /MAX(86400*(a.snap_date-b.snap_date)) "Wr/sec"
      ,SUM(a.phyblkwrt-b.phyblkwrt)
       /greatest(SUM(a.phywrts-b.phywrts),1) "Blk/wr"
  FROM snap_filestat a, snap_filestat b, dba_data_files t
 WHERE a.file# = b.file#
   AND a.snap_id = b.snap_id + 1
   AND t.file_id = a.file#
 GROUP BY t.tablespace_name
HAVING sum(a.phyblkrd-b.phyblkrd)
       /greatest(SUM(a.phyrds-b.phyrds),1) > 1.1
    OR SUM(a.phyblkwrt-b.phyblkwrt)
       /greatest(SUM(a.phywrts-b.phywrts),1) > 1.1
 ORDER BY 3 DESC, 5 DESC;
```

```
TABLESPACE_N Rd/sec Blk/rd Wr/sec Blk/wr
------------ ------ ------ ------ ------
TEMP            2.3   19.7    1.9   24.7
AP_T_02       287.1    7.8     .0    1.0
AP_T_01        12.9    4.0     .2    1.0
APPLSYS_T_01   63.3    2.2     .4    1.0
PO_T_01       313.5    2.1     .2    1.0
RECEIVABLE_T  401.0    1.5    2.4    1.0
SHARED_T_01     9.2    1.3     .4    1.0
SYSTEM         45.2    1.3     .3    1.0
PER_T_01       48.0    1.2     .0     .0
DBA_T_01         .2    1.0     .4    1.4
```

大規模のソートがディスクに移動するため、マルチブロック読込みおよび書込みのほとんどは TEMP 表領域に移動することがわかります。他の表領域は、全表スキャンによってマルチブロック読込みを取得します。

> **関連項目：** ファイル I/O データを収集する方法の例については、第 20 章「データベース統計収集用の Oracle のツール製品」を参照してください。

V$LATCH

このビューは、インスタンス起動以降の各ラッチ・タイプに関する統計のサマリーを維持します。このビューは、V$SESSION_WAIT でラッチ競合が確認されているときに問題が生じている SGA 内の領域を識別する場合に便利です。

V$LATCH の有益な列

- NAME: ラッチ名
- IMMEDIATE_GETS: 即時モードでのラッチに対する要求
- IMMEDIATE_MISSES: 失敗した IMMEDIATE_GETS
- GETS: 待機要求モードでのラッチ要求
- MISSES: 最初の試行でラッチを取得しなかった GETS
- SPIN_GETS: SPIN_GET 試行の中でラッチを取得したがスリープする必要がなかった GETS
- SLEEP1-SLEEP3: 1 ～ 3 回スリープした後のみ成功した GETS
- SLEEP4: 4 回以上スリープした後のみ成功した GETS
- WAIT_TIME: このラッチを待機する経過時間

V$LATCH の結合列

表 24-5 は、V$LATCH の結合列をリストしたものです。

表 24-5 V$LATCH の結合列

列	ビュー	結合列
NAME	V$LATCH_CHILDREN	NAME
	V$LATCHHOLDER	
	V$LATCHNAME	
NAME	V$LATCH_MISSES	PARENT_NAME
LATCH#	V$LATCH_CHILDREN	LATCH#
	V$LATCHNAME	

例 24-6 V$LATCH の問合せ

次の例では、V$LATCH から問い合せたデータを保持するための表が作成されます。

```
CREATE TABLE snap_latch as
SELECT 0 snap_id, sysdate snap_date, a.*
  FROM V$LATCH a;
 ALTER TABLE snap_latch add
     (constraint snap_filestat primary key (snap_id, name));
```

最初は、snap_id は 0（ゼロ）に設定されました。ある程度の時間間隔後、snap_id は 1 に設定されて、snap_latch 表は更新されます。

```
INSERT INTO snap_latch
SELECT 1, sysdate, a.*
  FROM V$LATCH a;
```

記録を挿入するために、前の SQL 文を使用するたびに snap_id を増分する必要があります。

連続した間隔で記録を挿入後、次の SELECT 文を使用して統計を表示します。ゼロで割る場合は、ゼロが代入されます。

```
SELECT SUBSTR(a.name,1,20) NAME, (a.gets-b.gets)/1000 "Gets(K)",
       (a.gets-b.gets)/(86400*(a.snap_date-b.snap_date)) "Get/s",
       DECODE ((a.gets-b.gets), 0, 0, (100*(a.misses-b.misses)/(a.gets-b.gets))) MISS,
       DECODE ((a.misses-b.misses), 0, 0,
              (100*(a.spin_gets-b.spin_gets)/(a.misses-b.misses))) SPIN,
       (a.immediate_gets-b.immediate_gets)/1000 "Iget(K)",
       (a.immediate_gets-b.immediate_gets)/ (86400*(a.snap_date-b.snap_date)) "IGet/s",
       DECODE ((a.immediate_gets-b.immediate_gets), 0, 0,
       (100*(a.immediate_misses-b.immediate_misses)/ (a.immediate_gets-b.immediate_gets))) IMISS
  FROM snap_latch a, snap_latch b
 WHERE a.name = b.name
   AND a.snap_id = b.snap_id + 1
   AND ( (a.misses-b.misses) > 0.001*(a.gets-b.gets)
       or (a.immediate_misses-b.immediate_misses) >
       0.001*(a.immediate_gets-b.immediate_gets))
ORDER BY 2 DESC;
```

前の SQL 文を実行する前に、次のように様々な表示の書式設定を指定します。

```
SET LIN 120
SET PAGES 60
SET NUMFORMAT 999999.9
```

例 24-7 サンプル・ラッチ統計

次の出力例では、V$FILESTAT 番号の場合と同じように、1 時間以上の差分によって得られたラッチ統計を示しています。ミスが取得の 0.1% より少なかったラッチはフィルタしています。

```
NAME                Gets(K)   Get/s  MISS   SPIN IGets(K)  IGet/s IMISS
------------------ -------- ------- ----- ------ -------- ------- -----
cache buffers chai  255,272  69,938   0.4   99.9    3,902   1,069   0.0
library cache       229,405  62,851   9.1   96.9   51,653  14,151   3.7
shared pool          24,206   6,632  14.1   72.1        0       0   0.0
latch wait list       1,828     501   0.4   99.9    1,836     503   0.5
row cache objects     1,703     467   0.7   98.9    1,509     413   0.2
redo allocation         984     270   0.2   99.7        0       0   0.0
messages                116      32   0.2  100.0        0       0   0.0
cache buffers lru        91      25   0.3   99.0    7,214   1,976   0.3
modify parameter v        2       0   0.1  100.0        0       0   0.0
redo copy                 0       0  92.3   99.3    1,460     400   0.0
```

ラッチ統計を調べる場合は、次の内容を調べます。

- ミス / 取得の比率はどうか。
- ミスのうちの何パーセントがスピンのみで取得されるか。

- ラッチは何回要求されたか。
- ラッチでスリープが何回発生したか。

92.3 パーセントのミス率を持つ REDO コピー・ラッチに対して多数の競合があると思われます。しかし、注意して見てください。REDO コピー・ラッチは、ほとんどの場合、即時モードで取得されます。即時取得の数は問題ないように見えます。即時取得は待機要求の取得の数倍の大きさです。したがって、REDO コピー・ラッチの競合はありません。

ただし、共有プール・ラッチとライブラリ・キャッシュ・ラッチの競合は存在するようです。実際に次の出力のような問題があるかどうかを確認するために、これらのラッチのスリープをチェックする問合せの実行を検討してください。

```
NAME                Gets(K)   Get/s  MISS   SPIN  SL01  SL02  SL03  SL04
------------------ -------- ------- ----- ------ ----- ----- ----- -----
cache buffers chai  255,272  69,938   0.4   99.9   0.1   0.0   0.0   0.0
library cache       229,405  62,851   9.1   96.9   3.0   0.1   0.0   0.0
shared pool          24,206   6,632  14.1   72.1  22.4   4.8   0.8   0.0
latch wait list       1,828     501   0.4   99.9   0.1   0.0   0.0   0.0
row cache objects     1,703     467   0.7   98.9   0.6   0.0   0.4   0.0
redo allocation         984     270   0.2   99.7   0.1   0.0   0.2   0.0
messages                116      32   0.2  100.0   0.0   0.0   0.0   0.0
cache buffers lru        91      25   0.3   99.0   1.0   0.0   0.0   0.0
modify parameter v        2       0   0.1  100.0   0.0   0.0   0.0   0.0
redo copy                 0       0  92.3   99.3   0.0   0.7   0.0   0.0
```

共有プール・ラッチ上のミス率が 14% であることを確認できます。ミスのうちの 72% がスピンによって CPU（1 回でもスリープする必要がある）を廃棄せずにラッチされました。何回もスリープする必要のあるミスはいくつかあります。

共有プール・ラッチが何回も要求される理由を調べてください。ラッチを保持または待っているセッションで実行される SQL とともに、システムのリソース使用の特性を調べてください。問題がなければ、それらをベースラインと比較します。

ラッチのチューニング

ラッチをチューニングしないでください。ラッチ競合が表示された場合、それは異常なリソース使用量が発生している SGA の一部の症状です。ラッチは、ある種の仮定（たとえば、カーソルが 1 回解析され、何回も実行されるなど）のもとにアクセスを制御します。問題を修復するには、競合が発生している SGA の各部分のリソース使用量を調べます。`V$LATCH` を調べるだけでは、問題は解決されません。

> **関連項目：** ラッチに関する詳細は、『Oracle9*i* データベース概要』を参照してください。

V$LATCH_CHILDREN

あるタイプのラッチの場合、データベースに複数のラッチがあります。V$LATCH は、各タイプのラッチの集計サマリーを提供します。個々のラッチを見るには、V$LATCH_CHILDREN ビューを問い合せます。

例 24-8 システム上の複数ラッチの個数の検出

```
SELECT name, count(*)
  FROM v$latch_children
 ORDER BY count(*) desc;

NAME                                     COUNT(*)
---------------------------------------- ----------
global tx hash mapping                         2888
global transaction                             2887
cache buffers chains                           2048
latch wait list                                  32
Token Manager                                    23
enqueue hash chains                              22
session idle bit                                 22
redo copy                                        22
process queue reference                          20
Checkpoint queue latch                           11
library cache                                    11
msg queue latch                                  11
session queue latch                              11
process queue                                    11
cache buffers lru chain                          11
done queue latch                                 11
channel operations parent latch                   4
session switching                                 4
message pool operations parent latch              4
ksfv messages                                     2
parallel query stats                              2
channel handle pool latch                         1
temp table ageout allocation latch                1
```

V$LATCHHOLDER

このビューは、ラッチを保持するセッションが変化しているかどうかを確認する際に便利です。ほとんどの場合は、実行される SQL 文またはラッチ・ホルダーが待機しているイベントを表示するために、他の表に結合できないような短い時間、ラッチが保持されます。

このラッチは、長時間にわたってラッチを保持している可能性のあるセッションを検索する場合に便利です。

V$LATCHHOLDER の結合列

表 24-6 は、V$LATCHHOLDER の結合列をリストしたものです。

表 24-6 V$LATCHHOLDER の結合列

列	ビュー	結合列
LADDR	V$LATCH_CHILDREN	ADDR
NAME	V$LATCH, V$LATCHNAME, V$LATCH_CHILDREN	NAME
PID	V$PROCESS	PID
SID	V$SESSION	SID

例 24-9 ラッチ・ホルダーで実行される SQL 文の検索

```
SELECT s.sql_hash_value, l.name
  FROM V$SESSION s, V$LATCHHOLDER l
WHERE s.sid = l.sid;

SQL_HASH_VALUE NAME
-------------- ----------------------------------------------
     299369270 library cache
    1052917712 library cache
    3198762001 library cache
SQL> /

SQL_HASH_VALUE NAME
-------------- ----------------------------------------------
     749899113 cache buffers chains
    1052917712 library cache
SQL> /

SQL_HASH_VALUE NAME
-------------- ----------------------------------------------
    1052917712 library cache
SQL> /
```

```
SQL_HASH_VALUE NAME
-------------- ------------------------------------------
     749899113 library cache
    1052917712 library cache
```

この例は、SQL 文 1052917712 が多数の解析リソースを使用していることを示します。次のステップでは、セッションで使用されるリソースを検索し、文を調べます。

V$LIBRARYCACHE

このビューには、インスタンス起動以降のライブラリ・キャッシュ内のオブジェクトに対する名前空間レベル・サマリーがあります。ライブラリ・キャッシュに関連するパフォーマンスの問題が発生している場合、このビューを使用すると、次のことを簡単に確認できます。

- ライブラリ・キャッシュ（共有プール）の特定の部分（名前空間）
- 問題の考えられる原因

次に、V$DB_OBJECT_CACHE、V$SQLAREA を使用してさらに詳細な情報を得ます。

V$LIBRARYCACHE の有益な列

- NAMESPACE: オブジェクト・クラス（SQL 領域、トリガーなど）
- GETS: この名前空間のオブジェクトに対する要求の処理
- GETHITS: キャッシュ内のハンドルを見つけた要求
- PINS: この名前空間のオブジェクトに対する PIN 要求
- PINHITS: 既存の PIN を再利用できる要求
- RELOADS: オブジェクトの一部がすでにキャッシュからフラッシュされているために、ライブラリ・キャッシュ内に格納されているオブジェクトをメモリーに再ロードする必要があった回数。大量の再ロードがある場合は、再利用可能情報がライブラリ・キャッシュからフラッシュされます。このため、オブジェクトに再度アクセスする前にオブジェクトの再ロード / 再構成が必要です。
- INVALIDATIONS: オブジェクトが無効にされた回数。たとえば、オブジェクトは実行することが安全でなくなったときに Oracle で自動的に無効にされます。たとえば、表のオプティマイザ統計が再計算される場合、発生した再計算が行われるときにライブラリ・キャッシュ内に現在存在しているすべての SQL 文は、その文の実行計画が最適なものでなくなる可能性があるために無効にされます。

GETHITRATIO（GETHITS/GETS）および PINHITRATIO（PINHITS/PINS）は、インスタンス起動以降にアクティビティを調べる場合にのみ使用できます。指定期間にわたるアクティビティを調べる場合、その期間の前後のスナップショットの違いから、アクティビティを計算するほうをお薦めします。

例 24-10 V$LIBRARYCACHE の問合せ

```
SELECT namespace, gets, 100*gethits/gets gethitratio,
       pins, 100* pinhits/pins getpinratio,
       reloads, invalidations
  FROM V$LIBRARYCACHE
 ORDER BY gets DESC
```

このビューへの問合せを行うときは、次のことを検索してください。

- 高い RELOADS または INVALIDATIONS
- 低い GETHITRATIO または PINHITRATIO

RELOADS が多いのは、次のことがあるためです。

- 無効にされるオブジェクト（多数の INVALIDATIONS）
- メモリーからスワップされるオブジェクト

低い GETHITRATIO は、オブジェクトがメモリーからスワップされることを示します。

低い PINHITRATIO は、次のことを示します。

- 同じカーソルを複数回実行しないセッション（たとえ様々なセッション間でカーソルが共有できる場合でも）
- 共有されるカーソルを検出しないセッション

次のステップでは、VDB_OBJECT_CACHE/VSQLAREA を問い合せて、問題がある種のオブジェクトに限定されているか、各種のオブジェクト間に広がっているかを確認します。無効にするものが多い場合、基礎となるオブジェクト（無効にされたオブジェクト）のうちのどれが変更されるかを調べると有効です。

V$LIBRARY_CACHE_MEMORY

この固定ビューは、ライブラリ・キャッシュの、ライブラリ・キャッシュ・オブジェクト型別の現在のメモリー使用を要約します。頻繁に問い合せてもライブラリ・キャッシュのラッチ競合が増加しません。列の説明は表 24-7 にリストされています。

表 24-7 V$LIBRARY_CACHE_MEMORY 列の説明

列名	データ型	説明
libcache_object_name	char(24)	ライブラリ・キャッシュのオブジェクト型の名前
libcache_pinned_memory	NUMBER	共有プール内で現在使用中のライブラリ・キャッシュ・メモリー・オブジェクトの数
libcache_pinned_count	NUMBER	ライブラリ・キャッシュで使用中のメモリー・オブジェクトの合計サイズ

表 24-7 V$LIBRARY_CACHE_MEMORY 列の説明（続き）

列名	データ型	説明
libcache_unpinned_memory	NUMBER	共有プール内で解放可能なライブラリ・キャッシュ・メモリー・オブジェクトの数
libcache_unpnned_count	NUMBER	ライブラリ・キャッシュで解放可能なメモリー・オブジェクトのサイズ

V$LOCK

このビューには、システム上で保持または要求されるすべてのロックに対する行があります。待機イベントのエンキューで待機するセッションを見つけた場合は、このビューを調べてください。ロックを待つセッションを検索する場合、イベントのシーケンスは次のようにすることができます。

1. ロックを保持するセッションを見つけるには、V$LOCK を使用します。
2. ロックを保持し、ロックを待つセッションで実行される SQL 文を検索するには、V$SESSION を使用します。
3. ロックを保持するセッションで何がブロックされるかを調べるには、V$SESSION_WAIT を使用します。
4. ロックを保持するプログラムとユーザーの詳細情報を取得するには、V$SESSION を使用します。

V$LOCK の有益な列

- SID: ロックを保持 / 要求するセッションの識別子
- TYPE: ロックのタイプ
- LMODE: セッションがロックを保持しているロック・モード
- REQUEST: セッションがロックを要求する際のロック・モード
- ID1, ID2: ロック・リソース識別子

共通のロック・タイプ

共通ロックのいくつかをこの項で説明します。

TX: 行トランザクション・ロック

- このロックは、データを変更する際に排他モード（モード 6）で要求されます。
- アクティブ・トランザクション当たり 1 つのロックが取得されます。このロックは、コミットまたはロールバックのためにトランザクションが終了するときに解放されます。
- 変更する行を含むブロックで ITL（interested transaction list）エントリが残されていない場合、セッションは共有モード（モード 4）でロックを要求します。このロックは、セッションがブロックの ITL エントリを取得したときに解放されます。
- 変更する行が別のセッションでロックされると、ロック・セッションのトランザクション・ロックが排他モードで要求されます。ロッキング・トランザクションが終了すると、この要求が終了し、行が基礎的なセッションの既存の TX ロックでカバーされます。
- ロックは、トランザクションのためのロールバック・セグメント・エントリとトランザクション・テーブル・エントリを指します。

このエンキュー上の競合を避けるには、次のことを行います。

- TX-6 エンキューでの競合を避けるには、アプリケーションを見直します。
- TX-4 エンキューでの競合を避けるには、オブジェクトの `INITRANS` の増加を検討します。

TM: DML ロック

- このロックは、データベース・オブジェクト上で排他モードで DDL 文を実行する際に要求されます。たとえば、排他モードの LOCK TABLE、ALTER TABLE または DROP TABLE があります。
- このロックは、`INSERT`、`UPDATE`、`DELETE` などの DML 文を実行する場合は共有モードでも取得されます。これにより、他のセッションは、同じオブジェクトで同時に DDL 文を実行できなくなります。
- データが変更されるすべてのオブジェクトについて、TM ロックが取得されます。
- このロックはオブジェクトを指します。

TM エンキューでの競合を避けるには、オブジェクトの表ロックの無効化を検討します。表ロックを無効化すると、オブジェクトで DDL が実行されなくなります。

ST ー 領域トランザクション・ロック

- ロックは（インスタンスではなく）各データベースに 1 つずつ存在します。
- このロックは、ローカル管理表領域以外の領域管理アクティビティ（作成またはエクステントの削除）について排他モードで要求されます。
- オブジェクトの作成、削除、拡張、切捨てはすべて、このロック上でシリアル化されます。
- このロック上の競合の最も一般的な原因は、ディスクに対するソート（実際の一時表領域を使用しない）またはロールバック・セグメントの拡大および縮小です。

このエンキュー上の競合を避けるには、次のことを行います。

- 実際の一時表領域を使用して一時ファイルを活用します。一時セグメントは作成されず、ディスクへのソート後に毎回削除されます。
- ローカル管理表領域を使用します。
- 動的な拡大および縮小を回避するには、ロールバック・セグメントをサイズ設定するか、自動 UNDO 管理を使用します。
- データベース・オブジェクトを作成および削除するアプリケーション特性を回避します。

UL ー ユーザー定義ロック

ユーザーは、独自のロックを定義できます。

> **関連項目：** ロックの詳細は、『Oracle9*i* データベース概要』を参照してください。

REQUEST/LMODE の共通するモード

- 0: なし
- 2: 行共有 : 共有 DML ロックに使用
- 4: 共有 : ITL エントリを待つときに共有 TX に使用
- 6: 行レベルの DML ロックに排他的に使用

V$LOCK のどの行にも、LMODE=0（それが要求であることを示す）または REQUEST=0（それが保持されたロックであることを示す）のいずれかがあります。

リソース識別子 ID1

DML ロックの場合、ID1 は object_id です。

TX ロックの場合、ID1 はロールバック・セグメントおよびトランザクション表エントリを指します。

V$LOCK の結合列

表 24-8 は、V$LOCK の結合列をリストしたものです。

表 24-8 V$LOCK の結合列

列	ビュー	結合列
SID	V$SESSION	SID
ID1, ID2, TYPE	V$LOCK	ID1, ID2, TYPE
ID1	DBA_OBJECTS	OBJECT_ID
TRUNCID1/65536)	V$ROLLNAME	USN

1. これは、セッションがロックを待っている場合にロックを保持するセッションを検索するために使用します。
2. これを使用して、DML ロックのロック済みオブジェクト（タイプ = 'TM'）を検索できます。
3. これを使用して、行トランザクション・ロック（タイプ = 'TX'）に使用するロールバック・セグメントを検索できます。ただし、機密性の低い結合は V$TRANSACTION から実行されます。

例 24-11 ロックを保持するセッションの検索

ロックを待つセッションの（ID1、ID2、タイプ）を検索します（LMODE=0）。

その ID1、ID2、タイプのためのロック（REQUEST=0）を保持するセッションを検索します。

```
SELECT lpad(' ',DECODE(request,0,0,1))||sid sess, id1, id2, lmode, request, type
FROM V$LOCK
 WHERE id1 IN (SELECT id1 FROM V$LOCK WHERE lmode = 0)
 ORDER BY id1,request

SID          ID1        ID2      LMODE    REQUEST TY
------ ---------- ---------- ---------- ---------- --
1237       196705     200493          6          0 TX <- Lock Holder
 1256      196705     200493          0          6 TX <- Lock Waiter
 1176      196705     200493          0          6 TX <- Lock Waiter
938        589854     201352          6          0 TX <- Lock Holder
 1634      589854     201352          0          6 TX <- Lock Waiter
```

例 24-12 セッションで実行される文の検索

```
SELECT sid, sql_hash_value
FROM V$SESSION
WHERE SID IN (1237,1256,1176,938,1634);

SID  SQL_HASH_VALUE
-----  --------------
  938      2078523611 <-Holder
 1176      1646972797 <-Waiter
 1237      3735785744 <-Holder
 1256      1141994875 <-Waiter
 1634      2417993520 <-Waiter
```

例 24-13 SQL 文のテキストの検索

```
HASH_VALUE SQL_TEXT
---------- ------------------------------------------------------------------
1141994875 SELECT TO_CHAR(CURRENT_MAX_UNIQUE_IDENTIFIER + 1 )   FROM PO_UNI
           QUE_IDENTIFIER_CONTROL  WHERE TABLE_NAME = DECODE(:b1,'RFQ','PO_
           HEADERS_RFQ','QUOTATION','PO_HEADERS_QUOTE','PO_HEADERS') FOR UP
           DATE OF CURRENT_MAX_UNIQUE_IDENTIFIER
1646972797 SELECT TO_CHAR(CURRENT_MAX_UNIQUE_IDENTIFIER + 1 )   FROM PO_UNI
           QUE_IDENTIFIER_CONTROL  WHERE TABLE_NAME = 'PO_HEADERS'  FOR UPD
           ATE OF CURRENT_MAX_UNIQUE_IDENTIFIER
2078523611 select CODE_COMBINATION_ID,  enabled_flag,  nvl(to_char(start_da
           te_active, 'J'), -1),  nvl(to_char(end_date_active, 'J'), -1), S
           EGMENT2||'.'||SEGMENT1||'.'||||SEGMENT6,detail_posting_allowed_f
           lag,summary_flag  from GL_CODE_COMBINATIONS  where CHART_OF_ACCO
           UNTS_ID = 101  and SEGMENT2 in ('000','341','367','388','389','4
           52','476','593','729','N38','N40','Q21','Q31','U21')  order by S
           EGMENT2, SEGMENT1, SEGMENT6
2417993520 select 0 into :b0  from pa_projects where project_id=:b1 for upd
           ate
3735785744 begin :X0 := FND_ATTACHMENT_UTIL_PKG.GET_ATCHMT_EXISTS(:L_ENTITY
           _NAME, :L_PKEY1, :L_PKEY2, :L_PKEY3, :L_PKEY4, :L_PKEY5, :L_FUNC
           TION_NAME, :L_FUNCTION_TYPE); end;
```

ロックされたセッションの文は、セッション 1176 と 1256 がセッション 1237 で保持されている PO_UNIQUE_IDENTIFIER_CONTROL 上のロックを待っているのに対して、セッション 1634 はセッション 938 で保持されている PA_PROJECTS 上のロックを待機していることを示します。セッションとユーザーに関する詳細をさらに取得するには、V$SESSION_WAIT、V$SESSION および V$SESSION_EVENT への問合せを行います。その例を次に示します。

- 誰がロックを保持しているか。
- ロックを保持しているセッションはアクティブか、アイドルか。
- ロックを保持している間にセッションは長時間実行問合せを実行しているか。

V$MTTR_TARGET_ADVICE

V$MTTR_TARGET_ADVICE には、各行に対応する MTTR の物理 I/O の数を予測する行が含まれています。これらの行では、物理 I/O 係数、つまり現在の MTTR 設定で測定間隔に実際に実行される I/O の回数に対する、見積り I/O の回数の比率が計算されます。列の説明は表 24-9 にリストされています。

表 24-9 V$MTTR_TARGET_ADVICE 列の説明

列名	データ型	説明
MTTR_TARGET_FOR_ESTIMATE	NUMBER	シミュレートされる MTTR 設定。これがビューの最初の行である場合は、現在の MTTR 設定に等しくなる必要があります。
ADVICE_STATUS	VARCHAR2(5)	MTTR シミュレーションの現在のステータス（ON|READY|OFF）
DIRTY_LIMIT	NUMBER	シュミレート中の MTTR から導出された使用済みバッファ制限
ESTD_CACHE_WRITES	NUMBER	この MTTR でのキャッシュ物理書込みの見積り数
ESTD_CACHE_WRITE_FACTOR	NUMBER	この MTTR のキャッシュ物理書込みの見積り比率。現在の MTTR 設定でのキャッシュ書込み数に対する見積りキャッシュ書込み数の比率を示します。
ESTD_TOTAL_WRITES	NUMBER	この MTTR でのキャッシュ物理書込みの見積り総数
ESTD_TOTAL_WRITE_FACTOR	NUMBER	この MTTR でのキャッシュ物理書込みの総見積り比率。現在の MTTR 設定での物理書込みの総数に対する物理書込みの見積り総数の比率を示します。
ESTD_TOTAL_IOS	NUMBER	この MTTR での I/O の見積り総数
ESTD_TOTAL_IO_FACTOR	NUMBER	この MTTR での I/O の総見積り比率。現在の MTTR 設定での I/O の総数に対する I/O の見積り総数の比率を示します。

V$MYSTAT

このビューは、現行セッションの統計を戻す V$SESSTAT のサブセットです。トリガーを通じてセッションのリソース使用量を監査する場合は、V$MYSTAT を使用してリソース使用量を把握します。V$SESSTAT で行をスキャンするよりはるかにコストが低いためです。

V$OPEN_CURSOR

このビューには、セッションでオープンされたすべてのカーソルがリストされます。このビューを使用する方法はいくつかあります。たとえば、様々なセッションでオープンされたカーソルの個数を監視できます。

システム・リソース使用量を診断する場合、高価な SQL（処理量の多い論理 I/O または物理 I/O）について V$SQLAREA と V$SQL へ問い合せることが役に立ちます。そのような場合、次のステップではそのソースを検索します。ユーザーが同じ一般ユーザーとしてデータベースにログインする（および V$SQLAREA に同じ PARSING_USER_ID を持っている）アプリケーションでは、これは困難になる可能性があります。V$SQLAREA の統計は、文が実行を終了した（および V$SESSION.SQL_HASH_VALUE から消滅）後に更新されます。したがって、文を再実行しないかぎり、セッションを直接検索できません。ただし、カーソルがセッションに対してまだオープンしている場合、V$OPEN_CURSOR を使用して文を実行したセッションを検索します。

V$OPEN_CURSOR の結合列

表 24-10 は、V$OPEN_CURSOR の結合列をリストしたものです。

表 24-10 V$OPEN_CURSOR の結合列

列	ビュー	結合列
HASH_VALUE, ADDRESS	V$SQLAREA,V$SQL,V$SQLTEXT	HASH_VALUE, ADDRESS
SID	V$SESSION	SID

例 24-14 文を実行したセッションの検索

```
SELECT hash_value, buffer_gets, disk_reads
FROM V$SQLAREA
WHERE disk_reads > 1000000
ORDER BY buffer_gets DESC;

HASH_VALUE BUFFER_GETS DISK_READS
---------- ----------- ----------
1514306888   177649108    3897402
 478652562    63168944    2532721
 360282550    14158750    2482065
 226079402    40458060    1592621
2144648214     1493584    1478953
1655760468     1997868    1316010
 160130138     6609577    1212163
3000880481     2122483    1158608
```

```
8 rows selected.

SQL> SELECT sid FROM V$SESSION WHERE sql_hash_value = 1514306888 ;

no rows selected

SQL> SELECT sid FROM V$OPEN_CURSOR WHERE hash_Value = 1514306888 ;

  SID
-----
 1125
  233
  935
 1693
  531

5 rows selected.
```

例 24-15 400 個より多いカーソルがオープンになっているセッションの検索

```
SELECT sid, count(*)
FROM v$open_cursor
 GROUP BY sid
HAVING COUNT(*) > 400
 ORDER BY count(*) desc;

 SID   COUNT(*)
----- ----------
 2359        456
 1796        449
 1533        445
 1135        442
 1215        442
  810        437
 1232        429
   27        426
 1954        421
 2067        421
 1037        416
 1584        413
  416        407
  398        406
  307        405
 1545        403
```

V$PARAMETER および V$SYSTEM_PARAMETER

これらのビューには、名前別に各初期化パラメータがリストされ、そのパラメータの値が表示されます。V$PARAMETER ビューには、問合せを行うセッションの現在の値が示されます。V$SYSTEM_PARAMETER ビューには、パラメータのインスタンス全体の値が表示されます。

たとえば、次の問合せを実行すると、問合せを実行するセッションの SORT_AREA_SIZE パラメータが表示されます。

```
SELECT value
  FROM V$PARAMETER
 WHERE name = 'sort_area_size';
```

V$PARAMETER の有益な列

- NAME: パラメータの名前
- VALUE: このセッションの現在の値（セッション内で変更される場合）、それ以外はインスタンス全体の値
- ISDEFAULT: パラメータの値がデフォルト値であるかどうか
- ISSES_MODIFIABLE: このパラメータをセッション・レベルで変更できるかどうか
- ISSYS_MODIFIABLE: インスタンスが開始した後にこのパラメータをインスタンス全体のレベルで動的に修正できるかどうか
- ISMODIFIED: インスタンス起動後にこのパラメータが変更されたかどうか、そうであれば、セッション・レベルとインスタンス（システム）レベルのいずれでこのパラメータが修正されたかどうか
- ISADJUSTED: Oracle がユーザーによって指定された値を調整したかどうか
- DESCRIPTION: パラメータの簡単な説明
- UPDATE_COMMENT: コメントがこのパラメータの DBA で指定されたかどうかの設定

関連項目：

- 範囲列の値の詳細は、『Oracle9*i* データベース・リファレンス』を参照してください。
- サーバー・パラメータ・ファイルについては、『Oracle9*i* データベース管理者ガイド』を参照してください。

V$PARAMETER および V$SYSTEM_PARAMETER データの用途

V$PARAMETER は、パラメータの現行の設定を決定するためにパフォーマンス・チューニング時に問合せが行われます。たとえば、バッファ・キャッシュ・ヒット率が低い場合、DB_BLOCK_BUFFERS（または DB_CACHE_SIZE）の値への問合せを行って現行のバッファ・キャッシュ・サイズを決定できます。

SQL*Plus の SHOW PARAMETER 文は、V$PARAMETER からデータへの問合せを行います。

例 24-16 SQL*Plus 内からの SORT_AREA_SIZE の決定

```
column name             format a20
column value            format a10
column isdefault        format a5
column isses_modifiable format a5

SELECT name, value, isdefault, isses_modifiable, issys_modifiable, ismodified
  FROM V$PARAMETER
 WHERE name = 'sort_area_size';

NAME                 VALUE      ISDEF ISSES ISSYS_MOD ISMODIFIED
-------------------- ---------- ----- ----- --------- ----------
sort_area_size       1048576    TRUE  TRUE  DEFERRED  MODIFIED
```

前述の例は、SORT_AREA_SIZE 初期化パラメータがインスタンス起動時に初期化パラメータとして設定されなかったが、このセッションのセッション・レベル（MODIFIED の値を持つ ISMODIFIED 列で示される）で変更されたことを示しています。

注意： V$PARAMETER から問合せを行うときは注意してください。インスタンス全体のパラメータを見る場合は、V$PARAMETER のかわりに V$SYSTEM_PARAMETER ビューを使用します。

V$PROCESS

このビューには、システム上で動作するすべての Oracle プロセスに関する情報が含まれています。この情報は、サーバー・プロセスの Oracle またはオペレーティング・システム・プロセス ID をデータベース・セッションに関連付ける場合に使用します。これは、いくつかの状況で必要になります。

- データベース・サーバー上のボトルネックがオペレーティング・システムのリソース（たとえば、CPU やメモリー）に関連している場合、または上位のリソース・ユーザーがサーバー・プロセス内の小さなセット内でローカライズされている場合は次の手順を実行します。
  1. リソースを多く使用するプロセスを検索します。
  2. それらのセッションを検索します。プロセスをセッションに関連付ける必要があります。
  3. セッションが非常に多くのリソースを使用している理由を見つけます。
- SQL*Trace ファイル名は、サーバー・プロセスのオペレーティング・システム・プロセス ID に基づきます。あるセッションのトレース・ファイルを見つけるには、そのセッションをサーバー・プロセスに関連付ける必要があります。
- `rdbms ipc reply` などのイベントの中には、セッションが待っているプロセスの Oracle プロセス ID を識別するものがあります。これらのプロセスが何を実行しているかを知るには、それらのセッションを検索する必要があります。
- サーバー（DBWR、LGWR、PMON など）に表示されるバックグラウンド・プロセスはすべて、サーバー・プロセスです。データベースでプロセスが何を実行しているかを見るには、それらのセッションを検索する必要があります。

V$PROCESS の有益な列

- `PID`: プロセスの Oracle プロセス ID
- `SPID`: プロセスのオペレーティング・システム・プロセス ID

V$PROCESS の結合列

表 24-11 は、`V$PROCESS` の結合列をリストしたものです。

表 24-11 V$PROCESS の結合列

列	ビュー	結合列
ADDR	V$SESSION	PADDR

例 24-17 サーバー・プロセス 20143 のセッションの検索

```
SELECT ' sid, serial#, aud sid : '|| s.sid||' , '||s.serial#||' , '||
       s.audsid||chr(10)|| '     DB User / OS User : '||s.username||
       '   /   '||s.osuser||chr(10)|| '    Machine - Terminal : '||
       s.machine||'  -  '|| s.terminal||chr(10)||
       '        OS Process Ids : '|| s.process||' (Client)  '||
       p.spid||' - '||p.pid||' (Server)'|| chr(10)||
       '   Client Program Name : '||s.program "Session Info"
  FROM V$PROCESS P,V$SESSION s
 WHERE p.addr = s.paddr
   AND p.spid = '20143';

Session Info
------------------------------------------------------------------
 Sid, Serial#, Aud sid : 2204 , 5552 , 14478782
     DB User / OS User : APPS   /   sifapmgr
    Machine - Terminal : finprod3  -
        OS Process Ids : 9095 (Client)  20143 - 1404 (Server)
   Client Program Name : RGRARG@finprod3 (TNS V1-V3)
```

例 24-18 PMON のセッションの検索

```
SELECT ' sid, serial#, aud sid : '|| s.sid||' , '||s.serial#||' , '||
       s.audsid||chr(10)|| '     DB User / OS User : '||s.username||
       '   /   '||s.osuser||chr(10)|| '    Machine - Terminal : '||
       s.machine||'  -  '|| s.terminal||chr(10)||
       '        OS Process Ids : '|| s.process||' (Client)  '||
       p.spid||' - '||p.pid||' (Server)'|| chr(10)||
       '   Client Program Name : '||s.program "Session Info"
  FROM V$PROCESS p, V$SESSION s
 WHERE p.addr = s.paddr
   AND s.program LIKE '%PMON%'

Session Info
------------------------------------------------------------------
 Sid, Serial#, Aud sid : 1 , 1 , 0
     DB User / OS User :    /   oracle
    Machine - Terminal : finprod7  -  UNKNOWN
        OS Process Ids : 20178 (Client)  20178 - 2 (Server)
   Client Program Name : oracle@finprod7 (PMON)
```

クライアント・プロセスとサーバー・プロセスがバックグラウンド・プロセスに対して同じであり、そのためにクライアント・プログラム名を指定できることが確認できます。

V$ROLLSTAT

このビューは、起動以降の各ロールバック・セグメントの統計サマリーを保持します。

V$ROLLSTAT の有益な列

- USN: ロールバック・セグメントの番号
- RSSIZE: ロールバック・セグメントの現在のサイズ
- XACTS: アクティブ・トランザクション数

一定期間の差分を実行するために有益な列

- WRITES: ロールバック・セグメントに書き込まれたバイト数
- SHRINKS: ロールバック・セグメントが過去の OPTIMAL を超えたが、また元通り縮小した回数
- EXTENDS: 次のエクステントにアクティブ・トランザクションがあったため、ロールバック・セグメントを拡張する必要のあった回数
- WRAPS: ロールバック・セグメントがラップした回数
- GETS: ヘッダー取得数
- WAITS: ヘッダー待機数

V$ROLLSTAT の結合列

表 24-12 は、V$ROLLSTAT の結合列をリストしたものです。

表 24-12 V$ROLLSTAT の結合列

列	ビュー	結合列
USN	V$ROLLNAME	USN

例 24-19 V$ROLLSTAT の問合せ

経過時間をラップで除算して、ロールバック・セグメントをラップするのに必要な平均時間を算定できます。これは、長時間実行問合せが「スナップショットが古すぎます。」というエラーを避けるためにロールバック・セグメントをサイズ設定する際に便利です。

また、最適サイズを増加する必要があるかどうかを知るために拡張と縮小を監視します。

V$ROWCACHE

このビューには、ディクショナリ・キャッシュ（行キャッシュとも呼ばれる）の統計が表示されます。各行には、各種のディクショナリ・キャッシュ・データの統計が含まれています。ディクショナリ・キャッシュ内に階層があるため、同じキャッシュ名が何回も表示される可能性があることに注意してください。

V$ROWCACHE の有益な列

- PARAMETER: キャッシュの名前
- COUNT: このキャッシュに割り当てられたエントリ数
- USAGE: 現在の使用済みエントリ数
- GETS: 要求の合計回数
- GETMISSES: ディクショナリ・キャッシュ・ミスが発生した要求数
- SCANS: スキャン要求数
- SCANMISSES: スキャンで必要なデータを検出できなかった回数
- MODIFICATIONS: キャッシュ・エントリの追加、変更、削除の回数
- DLM_REQUESTS: Real Application Clusters が DLM を要求した数
- DLM_CONFLICTS: Real Application Clusters と DLM が競合した数
- DLM_RELEASES: Real Application Clusters が DLM をリリースした数

V$ROWCACHE データの用途

- ディクショナリ・キャッシュが十分にサイズ設定されているかどうかを判別します。共有プールが小さすぎる場合、ディクショナリ・キャッシュは必要な情報をキャッシュするだけの十分なサイズに拡張できません。

  関連項目： 共有プールのチューニングの詳細は、14-26 ページの「共有プールのサイズ設定」を参照してください。

- アプリケーションがキャッシュに効率よくアクセスしているかどうかを判別します。アプリケーション設計でキャッシュを非効率的に使用している場合（この場合、ディクショナリ・キャッシュが大きくてもパフォーマンスの問題は軽減されません）です。たとえば、大量の GETS がサンプル期間内の DC_USERS キャッシュに対して表示される場合は、データベース内で作成された個々のユーザーが多数いて、アプリケーションが頻繁にそれらのユーザーのログオンとログオフを行っている可能性があります。これを確認するには、ログオン率の他に、システム内のユーザー数もチェックします。解析率も高くなります。これが中間層を持つ OLTP システムである場合、中間層で個々のアカウントを管理し、中間層がシングル・ユーザー、すなわち、アプリケーション所有者とし

てログオンできるようにするほうが効率的であることがあります。接続をアクティブのままにしておくことによってログオン / ログオフ率を減らすことも有効です。

- 動的領域割当てが発生しているかどうかを判別します。DC_SEGMENTS、DC_USED_EXTENTS および DC_FREE_EXTENTS に対するほぼ同規模の大量の修正は、大量の動的領域の割当てを意味する場合があります。解決策としては、次のエクステントを適切にサイズ設定したり、ローカル管理表領域を使用することなどが考えられます。領域割当てが一時的な用途の表領域で発生している場合は、実際の一時表領域を使用します。
- 発生している大量の順序番号作成を確認します。このことは、dc_sequences に対する修正で示されます。順序番号当たりのキャッシュ・エントリ数がそのときの変更回数に対して十分であるかどうかを確認します。
- ハード解析の証拠を収集します。ハード解析は、DC_COLUMNS、DC_VIEWS、DC_OBJECTS の各キャッシュに対する多数の GETS によっても証明できます。

例 24-20 V$ROWCACHE データの問合せ

ディクショナリ・キャッシュ統計を表示する適切な方法として、キャッシュ名でデータをグループ化します。

```
SELECT parameter
     , sum("COUNT")
     , sum(usage)
     , sum(gets)
     , sum(getmisses)
     , sum(scans)
     , sum(scanmisses)
     , sum(modifications)
     , sum(dlm_requests)
     , sum(dlm_conflicts)
     , sum(dlm_releases)
  FROM V$ROWCACHE
 GROUP BY parameter;
```

V$SEGMENT_STATISTICS

Oracle9*i* リリース 2（9.2）以上で使用可能なユーザーが扱いやすいこのビューでは、セグメント・レベル統計のリアルタイム監視が可能なため、DBA は個々の表または索引に関連付けられたパフォーマンスの問題を識別できます。

表 24-13 V$SEGMENT_STATISTICS ビュー

列	データ型	説明
OWNER	VARCHAR2(30)	オブジェクトの所有者
OBJECT_NAME	VARCHAR2(30)	オブジェクトの名前
SUBOBJECT_NAME	VARCHAR2(30)	サブオブジェクトの名前
TABLESPACE_NAME	VARCHAR2(30)	オブジェクトが属する表領域の名前
TS#	NUMBER	表領域番号
OBJ#	NUMBER	ディクショナリ・オブジェクト番号
DATAOBJ#	NUMBER	データ・オブジェクト番号
OBJECT_TYPE	VARCHAR2(18)	オブジェクトのタイプ
STATISTIC_NAME	VARCHAR2(64)	統計の名前
STATISTIC#	NUMBER	統計番号
VALUE	NUMBER	統計値

V$SEGSTAT

これはセグメント・レベル統計をリアルタイム監視するための高効率性ビューで、Oracle9*i* リリース 2（9.2）以上で使用できます。

表 24-14 V$SEGSTAT ビュー

列	データ型	説明
TS#	NUMBER	表領域番号
OBJ#	NUMBER	ディクショナリ・オブジェクト番号
DATAOBJ#	NUMBER	データ・オブジェクト番号
STATISTIC_NAME	VARCHAR2(64)	統計の名前
STATISTIC#	NUMBER	統計番号
VALUE	NUMBER	統計値

V$SEGSTAT_NAME

これはセグメント・レベル統計の統計プロパティ・ビューで、Oracle9*i* リリース 2（9.2）以上で使用できます。

表 24-15 V$SEGSTAT_NAME ビュー

列	データ型	説明
STATISTIC#	NUMBER	統計番号
NAME	VARCHAR2(64)	統計名
SAMPLED	VARCHAR2(3)	サンプル統計であるかどうかを示します

V$SESSION

このビューには、データベース・インスタンスに接続されているすべてのセッションに対して 1 行あります。各セッションには、ユーザー・セッションの他、DBWR、LGWR、アーカイバなどのバックグラウンド・プロセスが含まれています。

関連項目：『Oracle9*i* データベース概要』

V$SESSION の有益な列

V$SESSION は、基本的にはユーザーの SID または SADDR の検索に使用する情報ビューです。ただし、このビューには動的に変化するいくつかの列があり、ユーザーを調べるのに役立ちます。その例を次に示します。

SQL_HASH_VALUE, SQL_ADDRESS: これらの列は、セッションで現在実行されている SQL 文を識別します。NULL または 0 である場合、セッションは SQL 文を実行していません。PREV_HASH_VALUE と PREV_ADDRESS は、セッションで実行される直前の文を識別します。

注意： SQL*Plus から選択する場合は、完全な番号を見るために、列に十分な幅（11 の数字幅）が定義されているかどうか確認します。

STATUS: この列では、セッションが次のステータスであるかどうかを確認します。

- Active: SQL 文の実行（リソースの待機 / 使用）
- Inactive: さらに多くの作業（すなわち、SQL 文）の待機
- Killed: 強制終了されるようにマークが付く

次の列はセッションに関する情報を示し、次の内容の組合せ（1 つ以上）がわかっているときにセッションを検索する際に使用できます。

セッション情報

- SID: 他の列への結合に使用されるセッション識別子。
- SERIAL#: SID が別のセッションで再利用されるたびに増分されるカウンタ（あるセッションが終了し、別のセッションが開始して同じ SID を使用する場合）。
- AUDSID: セッション ID を一意に監査すると、データベースの生存中のセッションが識別されます。この監査は、問合せコーディネータのパラレル問合せスレーブを検索する場合にも有益です（PQ の実行時にそれらのスレーブは同じ AUDSID を持っています）。
- USERNAME: 接続されたセッションの Oracle ユーザー名。

クライアント情報

データベース・セッションは、データベース・サーバーで動作するクライアント・プロセスか、中間層サーバーまたはデスクトップから、SQL*Net を介してデータベースに接続するクライアント・プロセスによって開始されます。次の各列は、このクライアント・プロセスの情報を示します。

- OSUSER: クライアント・プロセスのオペレーティング・システム・ユーザー名
- MACHINE: クライアント・プロセスを実行しているマシン
- TERMINAL: クライアント・プロセスを実行している端末（適用可能な場合）
- PROCESS: クライアント・プロセスのプロセス ID
- PROGRAM: クライアント・プロセスによって実行されるクライアント・プログラム

PC から接続するユーザーの場合に TERMINAL、OSUSER を表示するには、ORACLE.INI にキー TERMINAL、USERNAME を設定するか、それらのキーがデフォルトで表示されないときは PC に Windows レジストリを設定します。

アプリケーション情報

パッケージ DBMS_APPLICATION_INFO を呼び出して、ユーザーを識別する情報を設定します。次の列が表示されます。

- CLIENT_INFO: DBMS_APPLICATION_INFO に設定される
- ACTION: DBMS_APPLICATION_INFO に設定される
- MODULE: DBMS_APPLICATION_INFO に設定される

次の V$SESSION 列も役立ちます。

- ROW_WAIT_OBJ#
- ROW_WAIT_FILE#
- ROW_WAIT_BLOCK#
- ROW_WAIT_ROW#

V$SESSION の結合列

他の固定ビューとの結合に使用できるいくつかの列のリストを表 24-16 に示します。

表 24-16 V$SESSION の結合列

列	ビュー	結合列
SID	V$SESSION_WAIT, V$SESSTAT, V$LOCK, V$SESSION_EVENT, V$OPEN_CURSOR	SID
(SQL_HASH_VALUE, SQL_ADDRESS)	V$SQLTEXT, V$SQLAREA, V$SQL	(HASH_VALUE, ADDRESS)
(PREV_HASH_VALUE, PREV_SQL_ADDRESS)	V$SQLTEXT, V$SQLAREA, V$SQL	(HASH_VALUE, ADDRESS)
TADDR	V$TRANSACTION	ADDR
PADDR	V$PROCESS	ADDR

例 24-21 セッションの検索

```
SELECT SID, OSUSER, USERNAME, MACHINE, PROCESS
  FROM V$SESSION
 WHERE audsid = userenv('SESSIONID');

  SID OSUSER     USERNAME    MACHINE     PROCESS
----- ---------- ----------- ----------- --------
  398 amerora    PERFSTAT    rgmdbs1     26582
```

例 24-22 マシンがわかっているときのセッションの検索

```
SELECT SID, OSUSER, USERNAME, MACHINE, TERMINAL
  FROM V$SESSION
 WHERE terminal = 'pts/tl'
   AND machine  = 'rgmdbs1';

 SID OSUSER    USERNAME  MACHINE    TERMINAL
---- --------- --------- ---------- ----------
 398 amerora   PERFSTAT  rgmdbs1    pts/tl
```

例 24-23 セッションで現在実行されている SQL 文の検索

特定のセッションで現在実行されている SQL 文を検索することが一般的な要件です。あるセッションでボトルネックが発生しているか、またはボトルネックに対する責任がある場合、この文ではセッションが何を実行しているか説明します。

```
col hash_value form 99999999999
SELECT sql_hash_value hash_value
  FROM V$SESSION WHERE sid = 406;

HASH_VALUE
----------
4249174653
SQL> /

HASH_VALUE
----------
4249174653
SQL> /

HASH_VALUE
----------
4249174653
SQL> /

HASH_VALUE
----------
4249174653
```

この例では、5 秒間待機し、文を再実行し、アクションを数回繰り返しました。同じ hash_value が何度も表示され、文がセッションで実行されることを示します。次のステップとして、V$SQLTEXT ビューと V$SQLAREA からの文の統計を使用して、文のテキストを検索します。

V$SESSION_EVENT

このビューは、すべてのセッションの待機イベントを要約します。V$SESSION_WAIT が、あるセッションの現在の待機を示すのに対して、V$SESSION_EVENT はセッションが開始した後に待機したすべてのイベントのサマリーを示します。

$SESSION_EVENT の有益な列

- SID: セッションの識別子
- EVENT: 待機イベントの名前
- TOTAL_WAITS: このセッションによるこのイベントに対する待機の総数
- TIME_WAITED: このイベントの待機の総時間（1/100 秒単位）
- AVERAGE_WAIT: このセッションによるこのイベントに対する平均待機時間（1/100 秒単位）
- TOTAL_TIMEOUTS: 待機がタイムアウトになった時間数

V$SESSION_EVENT の結合列

表 24-17 は、V$SESSION_EVENT の結合列のリストです。

表 24-17 V$SESSION_EVENT の結合列

列	ビュー	結合列
SID	V$SESSION	SID

例 24-24 データベース・ライターに対する待機の検索

```
SELECT s.sid, bgp.name
    FROM V$SESSION s, V$BGPROCESS bgp
   WHERE bgp.name LIKE '%DBW%'
     AND bgp.paddr = s.paddr;

SELECT event, total_waits waits, total_timeouts timeouts,
       time_waited total_time, average_wait avg
  FROM v$session_event
 WHERE sid = 3
 ORDER BY time_waited DESC;

EVENT                          WAITS TIMEOUTS TOTAL_TIME   AVG
------------------------------ -------- -------- ---------- -----
rdbms ipc message             1684385   921495  284706709 169.03
db file parallel write         727326        0    3012982   4.14
latch free                        157      157        281   1.78
control file sequential read      123        0         61   0.49
```

```
file identify                  45        0         29   0.64
direct path read               41        0          5   0.12
file open                      49        0          2   0.04
db file sequential read         2        0          2   1.00
```

V$SESSION_WAIT

これは、ボトルネックを検索するためのキー・ビューです。このビューでは、データベース内のどのセッションが現在待機しているか（またはセッションが何も待機していない場合にそのセッションで待機された最後のイベント）を示します。このビューは、システムにパフォーマンスの問題が発生しているとき、どの方向に進むかを探るための開始点として使用できます。

V$SESSION_WAIT では、インスタンスに接続されているすべてのセッションにそれぞれ行があります。このビューは、セッションが次の場合のいずれかを示します。

- リソースを使用している
- リソースを待機している
- アイドル状態（アイドル・イベントのうちの 1 つで待機している）

V$SESSION_WAIT の有益な列

- SID: セッションのセッション識別子
- EVENT: セッションが現在待っているイベント、またはセッションが待機する必要のあった最後のイベント
- WAIT_TIME: セッションがイベントを待機した時間（1/100 秒単位）。WAIT_TIME が 0（ゼロ）の場合、セッションは現在イベントを待機しています。
- SEQ#: セッションの待機ごとに増加
- P1、P2、P3: 待機に関する待機イベント固有の詳細
- P1TEXT、P2TEXT、P3TEXT: 所定のイベントに対する P1、P2、P3 の説明

関連項目：『Oracle9*i* データベース・リファレンス』および 22-23 ページの「待機イベント」

表 24-18 待機時間の説明

WAIT_TIME	意味	待機中
>0	最後の待機の待機時間（10 ms クロック・チック単位）	不可
0	セッションがこのイベントを現在待機中	可
-1	最後の待機で待機した時間が 10 ミリ秒未満	不可
-2	タイミングが無効	不可

表 24-19 に、EVENT、SEQ# および WAIT_TIME が一定期間にどのように変化するかの例を示します。

表 24-19 継続的に変化するイベント

時間	Seq #	イベント	待機時間	P1	P2	P3	アクション	待機中
0	43	latch free	0	800043F8	31	1	LRU ラッチを取得	可
10	43	latch free	10	800043F8	31	1	空きバッファを取得	不可
20	44	db file sequential read	0	5	1345	1	読込みコールを発行	可
30	44	db file sequential read	10	5	1345	1	バッファを処理	不可
35	45	enqueue	0	1415053318	196631	6355	バッファをロック	可
1040	45	enqueue	1000	1415053318	196631	6355	バッファを変更	不可

この例では、セッションは 0 ～ 10 の間**ラッチ**を待機し、20 ～ 30 の間 **db file sequential read** を待機し、35 ～ 1040 の間**ロック**を待機しました。時間の間隔が説明のために強調されています。Event と Seq# は、セッションが再度待機する必要があるまで変化しません。Wait Time は、セッションが実際にリソースを待機中または使用中であるかを示します。

V$SESSION_WAIT の結合列

表 24-20 は、V$SESSION_WAIT の結合列のリストです。

表 24-20 V$SESSION_WAIT の結合列

列	ビュー	結合列
SID	V$SESSION	SID

例 24-25 システムの現在待機の検索

```
SELECT event,
       sum(decode(wait_time,0,1,0)) "Curr",
       sum(decode(wait_time,0,0,1)) "Prev",
       count(*)"Total"
  FROM v$session_wait
 GROUP BY event
 ORDER BY count(*);

EVENT                                          Prev  Curr   Tot
---------------------------------------------- ---- ----- -----
PL/SQL lock timer                                 0     1     1
SQL*Net more data from client                     0     1     1
smon timer                                        0     1     1
pmon timer                                        0     1     1
SQL*Net message to client                         2     0     2
db file scattered read                            2     0     2
rdbms ipc message                                 0     7     7
enqueue                                           0    12    12
pipe get                                          0    12    12
db file sequential read                           3    10    13
latch free                                        9     6    15
SQL*Net message from client                     835  1380  2215
```

イベント別および wait_time（0= 待機中、0 以外 = 待機中ではない）別にデータをグループ化するこの問合せは、次のことを示します。

- セッションの大半は、SQL*Net message from client、pipe get、PMON timer などのアイドル・イベントを待機しています。
- CPU を使用するセッション数は、待機していないセッションの数（prev）で近似できます。ただし、1 つの問題を除きます。すなわち、何も待機していない（積極的にリソースを使用している）状態で、その最後の待機が SQL*Net message from client である多数のセッションが存在した場合があることです。

次のステップでは、セッションがアクティブかどうかを確認するために V$SESSION をチェックする必要があります。セッションがアクティブである場合のみ、リソースを積極的に待機しているか使用しているものとしてセッションをカウントします。このカウントを完結するには、次の文を使用します。合計列ではすべてのセッションの合計をカウントしますが、現在待機している列とリソースを使用してすでに待機した列は、アクティブ・セッションのみカウントします。

```
SELECT event,
       sum(decode(wait_Time,0,0,DECODE(s.status,'ACTIVE',1,0))) "Prev",
       sum(decode(wait_Time,0,1,DECODE(s.status,'ACTIVE',1,0))) "Curr",
       count(*) "Tot"
  FROM v$session s, v$session_wait w
 WHERE s.sid = w.sid
 GROUP BY event
 ORDER BY count(*);

EVENT                                  Prev  Curr   Tot
-------------------------------------- ----- ----- -----
SQL*Net message to client                 1     1     1 <- idle event
buffer busy waits                         1     1     1
file open                                 1     1     1
pmon timer                                0     1     1 <- idle event
smon timer                                0     1     1 <- idle event
log file sync                             0     1     1
db file scattered read                    0     2     2
rdbms ipc message                         0     7     7 <- idle event
pipe get                                  0    12    12 <- idle event
enqueue                                   0    14    14
latch free                               10    17    20
db file sequential read                   7    22    23
SQL*Net message from client               0  1383  2240 <- idle event
```

セッションは現在、アクティブである場合のみリソースを積極的に待機しているか使用しているものとしてカウントされています。このため、次のことが強調されます。

- 合計 2324 個のセッションがあります。
- 20 個のセッション（アクティブ待機のないアクティブ・セッション）が積極的にリソースを使用しています。
- 1463 個のセッションが待機しています。
- これらのセッションのうちの 58 個のセッションがアイドル状態でないイベントを待っています。この場合のアイドル・イベントは、SQL*Net message from client、pipe get、rdbms ipc message、PMON timer、SMON timer および SQL*Net message to client です。

  関連項目： 22-23 ページ「待機イベント」

- 14 個のセッションがロックアウトされています（パフォーマンスが低下している可能性があります）。
- PMON および SMON がタイマー上でスリープしています。
- 24 個のセッションが I/O コールが戻るまで待機しています（db file%read）。

V$SESSTAT

V$SESSTAT は、ログインで開始してからログアウトで終了するまでのセッション固有のリソース使用量統計を格納します。

このビューは V$SYSSTAT と同様に、次のタイプの統計を格納します。

- アクションが発生した回数（user commits など）
- 生成、アクセスまたは処理されたデータの実行総量（redo size など）
- TIMED_STATISTICS が TRUE である場合は、CPU used by this session などのいくつかのアクションの累積実行時間

注意： 初期化パラメータ STATISTICS_LEVEL を TYPICAL または ALL に設定すると、データベースの時間統計が自動的に収集されます。STATISTICS_LEVEL を BASIC に設定した場合に時間統計の収集を有効化するには、TIMED_STATISTICS を TRUE に設定する必要があります。

DB_CACHE_ADVICE、TIMED_STATISTICS または TIMED_OS_STATISTICS を明示的に初期化パラメータ・ファイルに、または ALTER_SYSTEM や ALTER SESSION を使用して設定した場合、その値は STATISTICS_LEVEL から導出された値を上書きします。

関連項目： STATISTICS_LEVEL の設定については、22-10 ページの「統計収集のレベルの設定」を参照してください。

V$SYSSTAT と V$SESSTAT の相違点は次のとおりです。

- V$SESSTAT は各セッションのデータのみを格納するのに対し、V$SYSSTAT はすべてのセッションの累積値を格納します。
- V$SESSTAT は一時的であり、セッションがログアウトした後に失われます。V$SYSSTAT は累積的であり、インスタンスがシャットダウンされるときのみ失われます。
- V$SESSTAT には、統計の名前が含まれていません。統計の名前を検索するには、このビューを V$SYSSTAT または V$STATNAME のいずれかに結合する必要があります。

V$SESSTAT を使用して、次の内容を持つセッションを検索できます。

- 最大リソース使用量
- 最大平均リソース使用量（ログオン時間に対するリソース使用量の比率）
- 現在のリソース使用量（2 つのスナップショット間の差分）

V$SESSTAT の有益な統計

V$SESSTAT の最も参照される統計は V$SYSSTAT について記述されている統計のサブセットであり、session logical reads、CPU used by this session、db block changes、redo size、physical writes、parse count (hard)、parse count (total)、sorts (memory)、sorts (disk) などがあります。

V$SESSTAT の有益な列

- SID: セッション識別子
- STATISTIC#: リソース識別子
- VALUE: リソース使用量

V$SESSTAT の結合列

表 24-21 は、V$SESSTAT の結合列をリストしたものです。

表 24-21 V$SESSTAT の結合列

列	ビュー	結合列
STATISTIC#	V$STATNAME	STATISTIC#
SID	V$SESSION	SID

例 24-26 データベースに現在接続されている最高の論理および物理 I/O 率を持つトップ・セッションの検索

次の SQL 文は、データベースに接続されているすべてのアクティブ・セッションの（1 秒当たりの）論理および物理読込み率を示します。論理および物理 I/O 率は、ログオン以降の経過時間（V$SESSION.LOGON_TIME からの値）を使用して計算されます。この値は、特にログオン期間にデータベースに接続されているセッションに対して正確ではない場合がありますが、この例では十分な値です。

session logical reads 統計と physical reads 統計について STATISTIC# を決定するには、次のようにします。

```
SELECT name, statistic#
  FROM V$STATNAME
 WHERE name IN ('session logical reads','physical reads') ;

NAME                           STATISTIC#
------------------------------ ----------
session logical reads                   9
physical reads                         40
```

次の問合せでこれらの値を使用すると、リソース使用量でセッションの順序が指定されます。

```
SELECT ses.sid
     , DECODE(ses.action,NULL,'online','batch')          "User"
     , MAX(DECODE(sta.statistic#,9,sta.value,0))
       /greatest(3600*24*(sysdate-ses.logon_time),1)     "Log IO/s"
     , MAX(DECODE(sta.statistic#,40,sta.value,0))
       /greatest(3600*24*(sysdate-ses.logon_time),1)     "Phy IO/s"
     , 60*24*(sysdate-ses.logon_time)                    "Minutes"
 FROM V$SESSION ses
    , V$SESSTAT sta
WHERE ses.status     = 'ACTIVE'
  AND sta.sid        = ses.sid
  AND sta.statistic# IN (9,40)
GROUP BY ses.sid, ses.action, ses.logon_time
ORDER BY
        SUM( DECODE(sta.statistic#,40,100*sta.value,sta.value) )
      / greatest(3600*24*(sysdate-ses.logon_time),1)  DESC;

  SID User   Log IO/s Phy IO/s Minutes
----- ------ -------- -------- -------
 1951 batch       291    257.3       1
  470 online    6,161     62.9       0
  730 batch     7,568     43.2     197
 2153 online    1,482     98.9      10
 2386 batch     7,620     35.6      35
 1815 batch     7,503     35.5      26
 1965 online    4,879     42.9      19
 1668 online    4,318     44.5       1
 1142 online      955     69.2      35
 1855 batch       573     70.5       8
 1971 online    1,138     56.6       1
 1323 online    3,263     32.4       5
 1479 batch     2,857     35.1       3
  421 online    1,322     46.8      15
 2405 online      258     50.4       8
```

システム上の個々のセッションの影響を適切に表示するために、結果が 1 秒当たりの総リソース使用量で順序付けされています。リソース使用量は、`session logical reads` と（重み付けされた）`physical reads` を加算して計算されています。

物理読込みは未処理の値に 100 を乗じて重み付けされ、物理 I/O のほうがすでにキャッシュ内にあるバッファの読込みより非常に高価であることを示しています。

物理 I/O の重み付け係数を計算するために、次の仮定が行われています。

- 物理 I/O（PIO）の平均待機は 10 ミリ秒でした（db file sequential read イベントおよび db file scattered read イベントの V$SYSTEM_EVENT.AVERAGE_WAIT から問い合せられます）。
- 平均論理 I/O 率（LIO）は 13000/ 秒 /CPU でした（統計名 session logical reads の V$SYSSTAT から問い合せられます。この統計は秒単位の経過時間とシステム上の CPU 数で除算されました）。
- これは、10 ミリ秒当たり 130 回の論理読込みの比率を示し、また、この構成では 10 ミリ秒当たり 1 回の物理読込みという比率を示します。この比率は、約 100 に丸められました。

V$SHARED_POOL_ADVICE

V$SHARED_POOL_ADVICE には、共有プールのサイズを変更した場合の、解析時間の見積り短縮量に関する情報が表示されます。サイズの範囲は、同等の時間間隔で現在の共有プール・サイズの 50 ～ 200% です。時間間隔の値は、共有プールの現在のサイズによって異なります。

表 24-22 V$SHARED_POOL_ADVICE ビュー

列	データ型	説明
SHARED_POOL_SIZE_FOR_ESTIMATE	NUMBER	見積りの共有プール・サイズ（MB）
SHARED_POOL_SIZE_FACTOR	NUMBER	現在の共有プール・サイズに関するサイズ係数
ESTD_LC_SIZE	NUMBER	ライブラリ・キャッシュで使用される見積りメモリー（MB）
ESTD_LC_MEMORY_OBJECTS	NUMBER	指定のサイズの共有プールに存在するライブラリ・キャッシュ・メモリー・オブジェクトの見積り数
ESTD_LC_TIME_SAVED	NUMBER	指定のサイズの共有プールでライブラリ・キャッシュ・メモリー・オブジェクトが検出されたことによる経過解析時間の見積り短縮量（秒）
ESTD_LC_TIME_SAVED_FACTOR	NUMBER	現在の共有プール・サイズに関する解析時間の見積り短縮係数
ESTD_LC_MEMORY_OBJECT_HITS	NUMBER	指定のサイズの共有プールでライブラリ・キャッシュ・メモリー・オブジェクトが検出された見積り回数

V$SQL

SQL 文が複数のカーソルにマップできるのは、カーソルで参照されるオブジェクトがユーザーごとに異なる可能性があるからです。複数のカーソル（子カーソル）が存在する場合、V$SQLAREA はすべてのカーソルにわたって集計された情報を提供します。

個々のカーソルを見るためには、V$SQL が使用できます。このビューには、SQL のカーソル・レベルの詳細が含まれています。このビューは、カーソルの解析に責任を持つセッションまたは人を見つける際に使用できます。

PLAN_HASH_VALUE 列には、カーソルの SQL 計画を数値で表現したものが含まれ、計画の比較に使用できます。PLAN_HASH_VALUE を使用すると、2 つの計画を行ごとに比較しないで、同じであるかどうかを簡単に識別できます。

V$SQL_PLAN

このビューは、実行後キャッシュされているカーソルの実行計画を調べる方法を提供します。

通常、このビューの情報は、EXPLAIN PLAN 文の出力に非常に似ています。ただし、EXPLAIN PLAN は文を実行する場合に使用できる理論的な計画を示すのに対し、V$SQL_PLAN には使用された実際の計画が含まれています。EXPLAIN PLAN 文で取得された実行計画がカーソルの実行に使用する実行計画とは異なる可能性があるのは、カーソルがセッション・パラメータの異なる値（たとえば、HASH_AREA_SIZE）でコンパイルされた可能性があるからです。

V$SQL_PLAN データの用途

- 現行の実行計画の決定
- 表に索引を作成する影響の確認
- ある種のアクセス・パスを含むカーソルの検索（たとえば、全表スキャンまたは索引レンジ・スキャン）
- オプティマイザで選択される、または選択されない索引の確認
- 開発者が予想する特定の実行計画（たとえば、ネステッド・ループ結合）をオプティマイザが選択するかどうかの判定

このビューは、計画比較におけるキー・メカニズムとしても使用できます。計画の比較は、次のタイプの変更が発生した場合に役立つ可能性があります。

- 索引の削除または作成
- データベース・オブジェクト上の ANALYZE 文の実行
- 初期化パラメータ値の修正

- 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切替え
- アプリケーションまたはデータベースの新しいリリースへのアップグレード後

以前の計画が維持される場合（たとえば、V$SQL_PLAN から選択されて参照のために永続的な Oracle 表に格納される場合）、SQL 文のパフォーマンスの変化とその文の実行計画の変化の相関を確認できます。

注意： オプティマイザ統計を収集する場合は、ANALYZE ではなく DBMS_STATS パッケージを使用することをお薦めします。このパッケージでは、統計のパラレル収集、パーティション・オブジェクトのグローバル統計の収集およびその他の方法による統計の調整ができます。また、コストベース・オプティマイザは、最終的には DBMS_STATS で収集された統計のみ使用します。このパッケージの詳細は、『Oracle9*i* PL/SQL パッケージ・プロシージャおよびタイプ・リファレンス』を参照してください。

ただし、次のような場合、コストベース・オプティマイザに関連しない統計収集では、DBMS_STATS ではなく ANALYZE 文を使用する必要があります。

- VALIDATE 句または LIST CHAINED ROWS 句を使用する場合
- 空きリスト・ブロックについての情報を収集する場合

V$SQL_PLAN の有益な列

このビューには、新しい列の他、ほとんどすべての PLAN_TABLE 列が含まれています。PLAN_TABLE にも存在する列は同じ値を持っています。

- ADDRESS: このカーソルの親に対するハンドルのアドレス
- HASH_VALUE: ライブラリ・キャッシュ内に存在する親の文のハッシュ値を持つ

ADDRESS 列と HASH_VALUE 列を使用して、カーソル固有の情報を追加するために V$SQLAREA と結合できます。

- CHILD_NUMBER: この実行計画を使用する子カーソルの番号

ADDRESS 列、HASH_VALUE 列および CHILD_NUMBER 列を使用して、子カーソル固有の情報を追加するために V$SQL と結合できます。

- OPERATION: このステップで実行される内部操作の名前（たとえば、TABLE ACCESS など）
- OPTIONS: OPERATION 列に記述されている処理に関するバリエーション（たとえば、FULL）
- OBJECT_NODE: オブジェクト（表名またはビュー名）の参照に使用するデータベース・リンクの名前。パラレル実行を使用するローカル問合せの場合、この列は操作による出力が消費される順序を記述します。

- OBJECT#: 表または索引のオブジェクト番号
- 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
- OBJECT_NAME: 表または索引の名前
- OPTIMIZER: 計画における最初の行（文の行）のオプティマイザの現行モード。たとえば、CHOOSE。操作がデータベース・アクセス（たとえば、TABLE ACCESS）である場合、オブジェクトが解析されるかどうかを指示します。
- ID: 実行計画の各ステップに割り当てられた数値
- PARENT_ID: 現行ステップの出力で処理する次の実行ステップの ID
- DEPTH: ツリー内における操作の深さ（またはレベル）。すなわち、PLAN_TABLE からの行をインデントするのに一般に使用されるレベル情報を取得するために、CONNECT BY を実行する必要はありません。ルート操作（文）はレベル 0 です。
- POSITION: 同じ PARENT_ID を持っているすべての操作に対する処理順序
- COST: オプティマイザのコストベース・アプローチにより予測される操作のコスト。ルールベース・アプローチを使用する文の場合、この列は NULL です。
- CARDINALITY: 操作によって作成される行数のコストベース・オプティマイザによる見積り
- BYTES: 操作によって作成されるバイト数のコストベース・オプティマイザによる見積り
- OTHER_TAG: OTHER 列の内容を記述します（値については第 9 章「EXPLAIN PLAN の使用方法」を参照してください）。
- PARTITION_START: 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
- PARTITION_STOP: アクセスされるパーティション範囲の停止パーティション
- PARTITION_ID: PARTITION_START と PARTITION_STOP 列の値の対を計算したステップ
- OTHER: ユーザーにとって有益な実行ステップに関するその他の情報（値については第 9 章「EXPLAIN PLAN の使用方法」を参照してください）
- DISTRIBUTION: パラレル問合せの場合、行を生成側の問合せサーバーからコンシューマの問合せサーバーに配分するために使用する方法を格納します。
- CPU_COST: オプティマイザのコストベース・アプローチで予測される操作の CPU コスト。ルールベース・アプローチを使用する文の場合、この列は NULL です。
- IO_COST: オプティマイザのコストベース・アプローチで予測される操作の I/O コスト。ルールベース・アプローチを使用する文の場合、この列は NULL です。
- TEMP_SPACE: オプティマイザのコストベース・アプローチで予測される操作（ソートまたはハッシュ結合）の一時領域使用量。ルールベース・アプローチを使用する文の場合、この列は NULL です。

- ACCESS_PREDICATES: アクセス構造の行を特定する場合に使用する述語。索引レンジ・スキャンの開始や停止の述語など
- FILTER_PREDICATES: 行を生成する前に、行をフィルタにかけるために使用する述語

DEPTH 列は、CONNECT BY 演算子で生成された LEVEL 疑似列を置きかえるものです。LEVEL 疑似列は PLAN_TABLE データをインデントするために SQL スクリプトで使用されることがあります。

V$SQL_PLAN の結合列

ADDRESS 列および HASH_VALUE 列、CHILD_NUMBER 列は、V$SQL または V$SQLAREA と結合してカーソル固有の情報を取り出す場合に使用します。たとえば、BUFFER_GETS は V$SQLTEXT と結合して SQL 文の完全なテキストを戻す場合に使用します。

表 24-23 は、V$SQL_PLAN の結合列をリストしたものです。

表 24-23 V$SQL_PLAN の結合列

列	ビュー	結合列
ADDRESS, HASH_VALUE	V$SQLAREA	ADDRESS, HASH_VALUE
ADDRESS, HASH_VALUE, CHILD_NUMBER	V$SQL	ADDRESS, HASH_VALUE, CHILD_NUMBER
ADDRESS, HASH_VALUE	V$SQLTEXT	ADDRESS, HASH_VALUE

SQL 文のためのオプティマイザ計画の決定

次の文は、ある SQL 文における EXPLAIN PLAN を示します。SQL 文の計画を調べることは、SQL 文をチューニングする際の最初のステップの 1 つです。実行計画が示している元の SQL 文は、その文の HASH_VALUE とアドレスで識別されます。

例 24-27 に、V$SQL_PLAN からの問合せとサンプル出力（1 つの子カーソルのみを想定）を示します。

例 24-27 V$SQL_PLAN の問合せ

```
SELECT /* TAG */ count(*)
  FROM employees e, departments d
 WHERE e.department_id = d.department_id;

  COUNT(*)
----------
        14

column operation   format a20
column options     format a20
column object_name format a20
```

```
column cost        format a20
column cost        format 9999
SELECT sql_text, address, hash_value
  FROM v$sql
 WHERE sql_text like '%TAG%';

SQL_TEXT   ADDRESS  HASH_VALUE
--------  --------  ----------
          82117BEC   171077025


SELECT sql_text, address, hash_value
  FROM v$sql
 WHERE sql_text LIKE '%TAG%'


SELECT /* TAG */ count(*)
  FROM employees e, departments d
 WHERE e.department_id = d.department_id

SQL_TEXT   ADDRESS  HASH_VALUE
--------  --------  ----------
          82157784  1224822469

SELECT operation, options, object_name, cost
  FROM v$sql_plan
 WHERE address = '\ 82157784'
   AND hash_value = 1224822469;

OPERATION            OPTIONS       OBJECT_NAME        COST
-------------------- ------------- ------------------ ----
SELECT STATEMENT                                         5
  SORT
    AGGREGATE
      HASH JOIN                                          5
      TABLE ACCESS   FULL          DEPARTMENTS           2
      TABLE ACCESS   FULL          EMPLOYEES             2

6 rows selected.
```

V$SQL_PLAN は、SQL 文の計画ではなくカーソルの計画を示します。その違いは、SQL 文には複数のカーソルを関連付けることができ、各カーソルはさらに CHILD_NUMBER で識別されます。次に、SQL 文が複数のカーソルを発生させるいくつかの例を示します。

- 同じ表名が 2 つの個別の表に変換される場合

  User1: `SELECT * FROM EMPLOYEES;`

  User2: `SELECT * FROM EMPLOYEES;`

  User2 は独自の EMPLOYEE 表を持ち、User1 はパブリック・シノニムで参照された表を使用します。

- User1 の環境が User2 とは異なる場合。たとえば、User2 がログイン・スクリプトでオプティマイザ・プランに first rows（ALTER SESSION SET OPTIMIZER_GOAL = FIRST_ROWS）を指定し、User1 が指定しなかった場合です。

  HASH_VALUE と ADDRESS を使用し、V$SQL_PLAN に対する問合せの結果に複数の計画が表示された場合、この SQL 文に複数の子カーソルがあるためです。この場合、子カーソル（CHILD_NUMBER で識別される）ごとに、計画を調べてそれらの子カーソルの違いが大きいかどうかを識別します。

  関連項目： チューニングする SQL 文を識別する方法については、6-3 ページの「多くのリソースを消費する SQL の識別およびデータ収集」を参照してください。

V$SQL_PLAN_STATISTICS

このビューでは、キャッシュされたカーソルそれぞれについて、実行計画内の各操作に対する実行統計が提供されます。

このビューで行ソースの統計を表示するには、DBA が STATISTICS_LEVEL パラメータを ALL に設定する必要があります。

表 24-24 V$SQL_PLAN_STATISTICS

列	データ型	説明
address	raw(4)	このカーソルの親に対するハンドルのアドレス
hash_value	NUMBER	ライブラリ・キャッシュ内に存在する親の文のハッシュ値。親カーソルの位置を特定するには、2 つの列（address および hash_value）を使用して、v$sqlarea と結合します。
child_number	NUMBER	この作業領域を使用する子カーソルの番号。この領域を使用する子カーソルの位置を特定するには、列（address、hash_value および child_number）を使用して、v$sql と結合します。
operation_id	NUMBER	実行計画の各ステップに割り当てられる数値
executions	NUMBER	このカーソルが実行された回数

表 24-24 V$SQL_PLAN_STATISTICS（続き）

列	データ型	説明
last_starts	NUMBER	[1] 最後の実行時にこの操作が開始された回数
starts	NUMBER	この操作が開始された回数であり、過去の実行に累積されます。
last_output_rows	NUMBER	最後の実行時に行ソースによって生成された行の数
output_rows	NUMBER	行ソースによって生成された行の数であり、過去の実行に累積されます。
last_cr_buffer_gets	NUMBER	最後の実行時に一貫モードで取り出されたバッファの数。問合せでは、バッファは通常一貫モードで取り出されます。
cr_buffer_gets	NUMBER	一貫モードで取り出されたバッファの数であり、過去の実行に累積されます。問合せでは、バッファは通常一貫モードで取り出されます。
last_cu_buffer_gets	NUMBER	最後の実行時に現行のモードで取り出されたバッファの数。INSERT、UPDATE、DELETE などの文では、バッファは現行モードで取り出されます。
cu_buffer_gets	NUMBER	現行のモードで取り出されたバッファの数であり、過去の実行に累積されます。INSERT、UPDATE、DELETE などの文では、バッファは現行モードで取り出されます。
last_disk_reads	NUMBER	最後の実行時に操作によって実行された物理ディスク読込みの回数
disk_reads	NUMBER	操作によって実行された物理ディスク読込みの回数であり、過去の実行に累積されます。
last_disk_writes	NUMBER	最後の実行時に操作によって実行された物理ディスク書込みの回数
disk_writes	NUMBER	操作によって実行された物理ディスク書込みの回数であり、過去の実行に累積されます。
last_elapsed_time	NUMBER	最後の実行時のこの操作に対応する経過時間（ミリ秒）
elapsed_time	NUMBER	この操作に対応する経過時間（ミリ秒）であり、過去の実行に累積されます。

[1] イテレータ（PARTITION または INLIST）の後の操作は複数回開始できます。開始の回数は実際には繰返しの回数を示します（パーティションの数またはリスト内の要素の数）

V$SQL_PLAN_STATISTICS_ALL

この表では、V$SQL_PLAN からの情報と、V$SQL_PLAN_STATISTICS および V$SQL_WORKAREA からの実行統計が連結されます。V$SQL_WORKAREA には、SQL メモリー（たとえば、ハッシュ結合やソート）を使用する行ソースのメモリー使用量統計が含まれています。

表 24-25 V$SQL_PLAN_STATISTICS_ALL

列	データ型	説明
address	raw(4)	このカーソルの親に対するハンドルのアドレス
hash_value	NUMBER	ライブラリ・キャッシュ内に存在する親の文のハッシュ値。カーソル固有の情報を追加するには、2 つの列（address および hash_value）を使用して、v$sqlarea と結合します。
child_number	NUMBER	この実行計画を使用する子カーソルの番号。子カーソル固有の情報を追加するには、列（address、hash_value および child_number）を使用して、v$sql と結合します。
operation	varchar2(30)	このステップで実行される内部操作の名前（たとえば、TABLE ACCESS など）
options	varchar2(30)	OPERATION 列に記述されている処理に関するバリエーション（たとえば、FULL）
object_node	varchar2(10)	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。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#	NUMBER	表または索引のオブジェクト番号
object_owner	varchar2(30)	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
object_name	varchar2(30)	表または索引の名前
optimizer	varchar2(20)	計画における最初の行（文の行）のオプティマイザの現行モード。たとえば、CHOOSE。操作がデータベース・アクセス（たとえば、TABLE ACCESS）である場合、オブジェクトが解析されるかどうかを示します。
id	NUMBER	実行計画の各ステップに割り当てられる数値
parent_id	NUMBER	現行ステップの出力で処理される次の実行ステップの ID
depth	NUMBER	ツリー内における操作の深さ（またはレベル）。plan_table の行をインデントする場合に一般的に使用されるレベル情報を取得する場合、CONNECT BY を実行する必要はありません。ルート操作（文）はレベル 0 です。
position	NUMBER	同じ PARENT_ID を持っているすべての操作の処理順序

表 24-25 V$SQL_PLAN_STATISTICS_ALL（続き）

列	データ型	説明
cost	NUMBER	オプティマイザのコストベース・アプローチによって見積られた操作コスト。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。
cardinality	NUMBER	操作によって生成される行数のコストベース・オプティマイザによる見積り
bytes	NUMBER	操作によって生成されるバイト数のコストベース・オプティマイザによる見積り
other_tag	varchar2(35)	OTHER 列の内容を記述します。値については表 9-2 を参照してください。
partition_start	varchar2(5)	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
partition_stop	varchar2(5)	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
partition_id	NUMBER	PARTITION_START 列および PARTITION_STOP 列の値のペアを計算したステップ
other	varchar2(4000)	ユーザーにとって有益な実行ステップに関するその他の情報。値については表 9-2 を参照してください。
distribution	varchar2(20)	生成側の問合せサーバーの行をコンシューマ側の問合せサーバーに分配する際に使用する手法。値については表 9-3 を参照してください。
cpu_cost	NUMBER	オプ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の CPU 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。
io_cost	NUMBER	オプティマイザのコストベース・アプローチによって見積られた操作の I/O 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。
temp_space	NUMBER	オプティマイザのコストベース・アプローチで見積もられる操作（ソートまたはハッシュ結合）の一時領域の使用量。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。
access_predicates	varchar2(4000)	アクセス構造に行を配置する場合に使用する述語。索引レンジ・スキャンの開始や停止の述語など
filter_predicates	varchar2(4000)	フィルタにかけた後で行を生成する場合に使用する述語
executions	NUMBER	このカーソルが実行された回数
last_starts	NUMBER	最後の実行時にこの操作が開始された回数
starts	NUMBER	この操作が開始された回数であり、過去の実行に累積されます。

表 24-25 V$SQL_PLAN_STATISTICS_ALL（続き）

列	データ型	説明
last_output_rows	NUMBER	最後の実行時に行ソースによって生成された行の数
output_rows	NUMBER	行ソースによって生成された行の数であり、過去の実行に累積されます。
last_cr_buffer_gets	NUMBER	最後の実行時に一貫モードで取り出されたバッファの数。問合せでは、バッファは通常一貫モードで取り出されます。
cr_buffer_gets	NUMBER	一貫モードで取り出されたバッファの数であり、過去の実行に累積されます。問合せでは、バッファは通常一貫モードで取り出されます。
last_cu_buffer_gets	NUMBER	最後の実行時に現行のモードで取り出されたバッファの数。INSERT、UPDATE、DELETE などの文では、バッファは現行モードで取り出されます。
cu_buffer_gets	NUMBER	現行のモードで取り出されたバッファの数であり、過去の実行に累積されます。INSERT、UPDATE、DELETE などの文では、バッファは現行モードで取り出されます。
last_disk_reads	NUMBER	最後の実行時に操作によって実行された物理ディスク読込みの回数
disk_reads	NUMBER	操作によって実行された物理ディスク読込みの回数であり、過去の実行に累積されます。
last_disk_writes	NUMBER	最後の実行時に操作によって実行された物理ディスク書込みの回数
disk_writes	NUMBER	操作によって実行された物理ディスク書込みの回数であり、過去の実行に累積されます。
last_elapsed_time	NUMBER	最後の実行時のこの操作に対応する経過時間（ミリ秒）
elapsed_time	NUMBER	この操作に対応する経過時間（ミリ秒）であり、過去の実行に累積されます。
policy	varchar2(10)	この作業領域のサイズ指定ポリシー。値は MANUAL と AUTO のいずれかです。
estimated_optimal_size	NUMBER	この作業領域が完全にメモリー内で操作の実行を完了する（最適な実行）場合に必要な見積りサイズ（KB）。オプティマイザ統計または過去の実行から導出されます。
estimated_onepass_size	NUMBER	この作業領域が 1 つのパスで操作を実行する場合に必要な見積りサイズ（KB）。オプティマイザ統計または過去の実行から導出されます。
last_memory_used	NUMBER	カーソルの最後の実行時にこの作業領域で使用されるメモリーのサイズ（KB）

表 24-25 V$SQL_PLAN_STATISTICS_ALL（続き）

列	データ型	説明
last_execution	varchar(10)	カーソルの最後の実行時に、この作業領域が OPTIMAL または ONE PASS を使用して実行されたか、あるいは ONE PASS メモリー要件（MULTI-PASS）で実行されたかを示します。
last_degree	NUMBER	カーソルの最後の実行時に使用された並列度
total_executions	NUMBER	この作業領域がアクティブにされた回数
optimal_executions	NUMBER	この作業領域が最適モードで実行された回数
onepass_executions	NUMBER	この作業領域がワン・パス・モードで実行された回数
multipasses_executions	NUMBER	この作業領域がワン・パス・メモリー要件で実行された回数
active_time	NUMBER	作業領域がアクティブになっている時間（ミリ秒）
max_tempseg_size	NUMBER	一時領域の最大使用量
tempseg_size	NUMBER	一時領域の使用量

V$SQLAREA

このビューは、共有プールに存在するすべての共有カーソルを追跡しています。このビューには、共有プールに存在するすべての SQL 文ごとに行が 1 行あります。このビューは、SQL 文のリソース使用量を検索するための非常に価値の高いビューです。

V$SQLAREA の情報列

- HASH_VALUE: SQL 文のハッシュ値
- ADDRESS: SQL 文の SGA アドレス

これらの 2 つの列は、SQL 文の識別に使用します。2 つの異なる文が同じ値にハッシュされる場合があります。そのような場合、hash_value とともにアドレスを使用する必要があります。

- PARSING_USER_ID: 文の最初のカーソルを解析したユーザー
- VERSION_COUNT: 文に対するカーソルの数
- KEPT_VERSIONS: DBMS_SHARED_POOL.KEEP() を使用して確保された文のカーソル
- SHARABLE_MEMORY: カーソルが使用する共有メモリーの総量
- PERSISTENT_MEMORY: カーソルが使用する永続的メモリーの総量
- RUNTIME_MEMORY: カーソルが使用するランタイム・メモリーの総量

- SQL_TEXT: SQL 文の最初の 1000 文字（最大で 1000 文字まで）
- MODULE, ACTION: DBMS_APPLICATION_INFO を使用して設定されたセッションによりカーソルが最初に解析された場合のセッション情報

V$SQLAREA の有益な他の列

これらの列は、文を実行するたびに増分されます。

- BUFFER_GETS: この文の論理読込み数
- DISK_READS: この文の物理読込み数
- SORTS: この文のソート数
- CPU_TIME: この文の解析および実行に使用される CPU タイム
- ELAPSED_TIME: この文の解析および実行の経過時間
- PARSE_CALLS: この文の解析コール（ハードおよびソフト）数
- EXECUTIONS: この文が実行された回数
- INVALIDATIONS: この文のカーソルが無効にされた回数
- LOADS: この文のロード（およびリロード）数
- ROWS_PROCESSED: この文から戻される行の総数

V$SQLAREA の結合列

表 24-26 は、V$SQLAREA の結合列をリストしたものです。

表 24-26 V$SQLAREA の結合列

列	ビュー	結合列
HASH_VALUE, ADDRESS	V$SESSION	SQL_HASH_VALUE, SQL_ADDRESS
HASH_VALUE, ADDRESS	V$SQLTEXT, V$SQL, V$OPEN_CURSOR	HASH_VALUE, ADDRESS
SQL_TEXT	V$DB_OBJECT_CACHE	NAME

例 24-28 リソース集中型 SQL の検索

使用できるコストは次のようにいくつかあります。

- 総論理 I/O（LIO）数、各実行の LIO
- 総物理 I/O（PIO）数、各実行の PIO

- PIO/LIO（低いキャッシュ・ヒット率）
- parse_calls、各実行の parse_calls

```
SELECT hash_value, executions, buffer_gets, disk_reads, parse_calls
  FROM V$SQLAREA
 WHERE buffer_gets > 10000000
    OR disk_reads > 1000000
 ORDER BY buffer_gets + 100*disk_reads DESC;

HASH_VALUE EXECUTIONS BUFFER_GETS DISK_READS PARSE_CALLS
---------- ---------- ----------- ---------- -----------
2676594883        126     7583140    6199113         126
4074144966        126     7264362    6195433          49
 228801498        136   236116544    2371187         136
 360282550       5467    21102603    4476317        2355
1559420740        201     8197831    4537591          39
3213702248   28039654   364516977         44         131
1547710012        865     7579025    3337735         865
3000880481       4481     3676546    2212658        2885
1398193708       4946    73018658    1515257        1418
1052917712    8342025   201246652      38240      327462
 371697988          7    74380777     862611           7
1514306888    3922461    29073852    1223482         268
1848522009          1     1492281    1483635           1
1478599096   28042103   140210513        594         164
 226079402      21473    22121577    1034787        4484
 478652562       4468    21669366    1020370        4438
2054874295      73520   118272694      29987       73520
```

注意： ある文がシステムで初めて実行されており、現在のリソース使用量の大部分を占める場合、この文ではその文を検索できません。BUFFER_GETS および DISK_READS 統計は、文の実行が終了するまで更新されないからです。

例 24-29 SQL 文で使用されているリソースの検索

```
SELECT hash_value, buffer_gets, disk_reads, executions, parse_calls
  FROM V$SQLAREA
 WHERE hash_Value = 228801498
   AND address = hextoraw('CBD8E4B0');

HASH_VALUE BUFFER_GETS DISK_READS EXECUTIONS PARSE_CALLS
---------- ----------- ---------- ---------- -----------
 228801498   236116544    2371187        136         136
```

V$SQLTEXT

このビューには、共有プール内の SQL 文に対する完全な SQL テキストが含まれています。

注意： V$SQLAREA には、最初の 1000 文字のみ含まれています。

V$SQLTEXT の有益な列

- HASH_VALUE: SQL 文のハッシュ値
- ADDRESS: SGA 内の SQL 文カーソルのアドレス
- SQL_TEXT: 64 文字チャンク内の文テキスト
- PIECE: SQL 文ピースのオーダー情報

V$SQLTEXT の結合列

表 24-27 は、V$SQLTEXT の結合列をリストしたものです。

表 24-27 V$SQLTEXT の結合列

列	ビュー	結合列
HASH_VALUE, ADDRESS	VSQL, VSESSION	HASH_VALUE, ADDRESS
HASH_VALUE, ADDRESS	V$SESSION	SQL_HASH_VALUE, SQL_ADDRESS

例 24-30 ハッシュ値が示す SQL 文の検索

```
SELECT sql_text
  FROM V$SQLTEXT
 WHERE hash_value = 228801498
 ORDER BY piece;

SQL_TEXT
----------------------------------------------------------------
select dbsu.primary_flag, i.site_use_code, i.rowid
from ra_customers dbc, ra_addresses dbad, ra_site_uses dbsu, ra_customers_interface
i
where (((((((i.orig_system_customer_ref=dbc.orig_system_reference and dbad.address_
id=dbsu.address_id) and i.site_use_code=dbsu.site_use_code) and dbsu.status='A') and
dbad.customer_id=dbc.customer_id) and i.request_id=:b0) and nvl(i.validated_
flag,'N')<>'Y') and ((i.primary_site_use_flag='Y' and dbsu.primary_flag='Y') or
dbsu.site_use_code in ('STMTS','DUN','LEGAL')))
group by dbsu.primary_flag,i.orig_system_customer_ref,i.site_use_code,i.insert_
update_flag,i.rowid
```

V$STATISTICS_LEVEL

V$STATISTICS_LEVEL には、STATISTICS_LEVEL 初期化パラメータで制御される統計またはアドバイザのステータスが示されます。V$STATISTICS_LEVEL の各行は、これらの統計またはアドバイザのいずれかを表します。

表 24-28 V$STATISTICS_LEVEL ビュー

列	データ型	説明
STATISTICS_NAME	VARCHAR2(64)	統計 / アドバイザの名前
DESCRIPTION	VARCHAR2(4000)	統計 / アドバイザの機能とその使用目的の説明
SESSION_STATUS	VARCHAR2(8)	ENABLED|DISABLED。このセッションの統計 / アドバイザのステータス
SYSTEM_STATUS	VARCHAR2(8)	ENABLED|DISABLED。システム全体の統計 / アドバイザのステータス
ACTIVATION_LEVEL	VARCHAR2(7)	BASIC|TYPICAL|ALL。この統計 / アドバイザが使用可能になる STATISTICS_LEVEL のレベル
STATISTICS_VIEW_NAME	VARCHAR2(64)	この統計 / アドバイザを外部化するビューが 1 つある場合はそのビューの名前。そのようなビューがない場合、この列は空になります。該当するビューが複数ある場合は、DESCRIPTION 列にそれらのビューの名前が示されます。
SESSION_SETTABLE	VARCHAR2(3)	YES|NO。この統計 / アドバイザをセッション・レベルで設定可能かどうか。

V$SYSSTAT

V$SYSSTAT は、インスタンスの開始時より累積された、インスタンス全体にわたるリソース使用量に関する統計を格納します。

このビューは V$SESSTAT と同様に、次のタイプの統計を格納します。

- アクションが発生した回数（user commits）
- 作成、アクセスまたは処理されたデータの実行総量（redo size）
- TIMED_STATISTICS が TRUE である場合は、いくつかのアクション（CPU used by this session）の実行に使用された累積時間

V$SYSSTAT の有益な列

- STATISTIC#: 統計の識別子
- NAME: 統計名
- VALUE: リソース使用量

各統計の値は、インスタンス起動以降におけるその統計のリソース使用量を格納します。次に、execute count 統計の列の値の例を示します。

表 24-29 V$SYSSTAT の有益な列

統計番号	名前	値
215	execute count	19,003,070

注意： 統計の STATISTIC# は、リリース間で変化する可能性があります。整合を保つために、STATISTIC# に依存しないでください。そのかわり、VALUE の問合せには統計の NAME 列を使用してください。

V$SYSSTAT の用途

このビューのデータは、システム・パフォーマンスの監視に使用します。バッファ・キャッシュ・ヒット率やソフト解析率などの導出された統計は、V$SYSSTAT データから計算されます。

このビューのデータは、システム・リソース使用量の監視、およびどのようにシステムのリソース使用量が時間経過につれて変化するかの監視にも使用します。ほとんどのパフォーマンス・データと同様に、ある一定期間にわたるシステムのリソース使用量を調べます。これを行うには、その期間の初めにビュー内のデータのスナップショットを作成し、その期間の終わりに別のスナップショットを作成します。各統計の値の差（終わりの値から始めの値を引いた差）は、その期間で使用されたリソースです。これは、Statspack や BSTAT/ESTAT などの Oracle ツールで使用される手法です。

ある期間のデータを別の期間のデータと比較するには、（各トランザクション、実行、秒またはログオンごとに）データを正規化します。両方のワークロードに関するデータを正規化すると、それらの 2 つのワークロードの間の偏差をさらに容易に識別できます。このような比較は特に、パッチ適用後、アプリケーションのアップグレード後、または、単に時間経過によって、ユーザー人口の増加やデータの増加がリソース使用量にどのような影響を与えるかを確認する場合に役立ちます。

また、V$SYSSTAT データを使用して、V$SYSTEM_EVENT ビューを問い合せて確認された競合対象のリソースのリソース使用量を調べることもできます。

V$SYSSTAT の有益な統計

この項では、チューニング時に最も役立つ V$SYSSTAT 統計のいくつかと、この統計の説明を示します。このリストはアルファベット順になっています。

> **関連項目：** 統計の全リストと統計の説明については、『Oracle9*i* データベース・リファレンス』を参照してください。

重要なデータベース使用量のインジケータ

- CPU used by this session: バックグラウンド・プロセスを除くすべてのセッションで使用された総 CPU 量。この統計の単位は 1/100 秒です。10 ミリ秒未満で完了するコールは、この単位まで丸められます。
- db block changes: 挿入、更新または削除操作の一部として、SGA 内のデータベース・ブロックに対して行われた変更の回数。この統計は、総データベース作業量のおよその指標です。各トランザクションのレベルでは、この統計はバッファが変更される比率を示します。
- execute count: SQL 文の実行（再帰的 SQL も含む）の総数。
- logons current: インスタンスに現在接続されているセッション。ある期間にまたがって 2 つのスナップショットを使用する場合は、平均値（差ではない）を使用する必要があります。
- logons cumulative: インスタンス起動以降の総ログオン数。特定の期間内のログオン数を決定するには、開始値から終了値を差し引きます。有益な導出された統計は、開始時間から終了時間までの接続回数を、その期間で費やした秒数で割ることです。これにより、ログオン率が算出されます。1 秒当たりのログオン数は 2 回以下とするのが最適です。これに対して、1 秒当たりのログオン 50 回は、非常に高いとみなされます。継続的にデータベースとの接続および切断を行う（たとえば、トランザクション当たり 1 回）アプリケーションは、良好な測定値を示しません。
- parse count (hard): 共有プールでミスになった解析コール数。SQL 文が実行され、その SQL 文が共有プール内にないか、共有プール内にあっても 2 つの SQL 文のメタデータの一部が異なるためにその SQL 文を共有できない場合に、ハード解析が行われます。これは、ある SQL 文が既存の SQL 文とテキストでは同一であるが、2 つの文で参照される表が物理的に異なる表に変換される場合に発生する可能性があります。ハー

ド解析が、CPU とリソース使用（たとえば、ラッチ）の点でコストが高い操作であるのは、共有プール内でメモリーを割り当て、その後文を実行する前に実行計画を決定するように Oracle に要求するからです。

- parse count (total): 解析コール（ハードとソフトの両方）の総数。セッションが SQL 文を実行し、その文がすでに共有プール内にあって使用できるときは、ソフト解析が行われます。使用する（すなわち、共有される）文の場合、既存の SQL 文に関連するすべてのデータ（オプティマイザ実行計画などのデータも含む）が発行されている現行の文に均等に適用できる必要があります。これらの 2 つの統計は、ソフト解析率の計算に使用します。
- parse time cpu: 解析に費やされた総 CPU タイム（1/100 秒単位）。これには、ハード解析とソフト解析の両方が含まれています。
- parse time elapsed: 実行する解析コールの総経過時間。
- physical reads: オペレーティング・システムから読み込まれたブロック数。このブロック数には、SGA バッファ・キャッシュへの物理読込み（バッファ・キャッシュ・ミス）と、PGA へのダイレクト物理読込み（たとえば、ダイレクト・ソート操作時）が含まれています。この統計は I/O リクエストの数ではありません。
- physical writes: DBWR により SGA バッファ・キャッシュからディスクに書き込まれたデータベース・ブロック、およびダイレクト書込みを実行するプロセスにより PGA から書き込まれたデータベース・ブロックの個数。
- redo log space requests: 一般的にログ・スイッチが必要であるために、サーバー・プロセスが REDO ログの領域を待機した回数。
- redo size: 生成された（したがって、ログ・バッファに書き込まれた）REDO の総量（バイト単位）。この統計（秒またはトランザクションで正規化）は、更新アクティビティを示すのに適切なインジケータです。
- session logical reads: バッファ・キャッシュ内または物理読込みによる論理読込み要求数。
- sorts (memory) および sorts (disk): sorts (memory) は、SORT_AREA_SIZE の内部に収まっている（したがって、ディスク上のソートを要求しなかった）ソート操作回数です。sorts (disk) は、SORT_AREA_SIZE より大きく、ディスク上の領域を使用してソートを行う必要のあったソート操作回数です。これらの 2 つの統計は、メモリー内ソート率の計算に使用します。
- sorts (rows): ソートされた行数の合計。この統計を 'sorts (total)' 統計で除算するとソート当たりの行数が算出されます。この行数は、データ・ボリュームとアプリケーション特性のインジケータです。
- table fetch by rowid: ROWID を使用して戻された（索引アクセスまたは「where rowid = &rowid」という形式の SQL 文が発行されたため）行数。
- table scans (rows gotten): 全表スキャンで処理された総行数。

- table scans (blocks gotten): 全表スキャンでスキャンされたブロック（分割行のブロックを除く）の個数。
- user commits + user rollbacks: これは、システム上の総トランザクション数を示します。この数値は、他の統計の各トランザクションの比率を計算するときに除数として使用します。たとえば、各トランザクションの論理読込み回数を計算するには、次の方程式を使用します。session logical reads / (user commits + user rollbacks)

物理 I/O に関する注意

Oracle から報告される物理読込みでは、実際の物理ディスク I/O 操作が発生しない場合があります。これは大半のオペレーティング・システムが、該当するブロックが存在する可能性のあるオペレーティング・システム・ファイルのシステム・キャッシュを持っているためです。あるいは、ブロックがディスクまたはコントローラ・レベル・キャッシュ内にも存在しうるため、この場合も実際の I/O が回避される可能性があります。Oracle から報告されるような物理読込みは単に、要求されたブロックがバッファ・キャッシュ内になかった（または、ダイレクト読込み操作の場合はプライベート・メモリーに読み込む必要があった）ことを示すにすぎません。

V$SYSSTAT 統計によるインスタンス効率

次に、V$SYSSTAT データから計算された代表的なインスタンス効率を示します。各比率の計算値はすべて、可能なかぎり 1 に近い値である必要があります。

バッファ・キャッシュ・ヒット率 : バッファ・キャッシュが小さすぎるかどうかを示す優れたインジケータです。

```
1 - ((physical reads - physical reads direct - physical reads direct (lob)) /
session logical reads)
```

ソフト解析率 : システム上に多数のハード解析が存在するかどうかを示します。この比率を生の統計と比較して、正確性を確認する必要があります。たとえば、ソフト解析率 0.2 は一般に高いハード解析率を示します。ただし、総解析数が少ない場合、その比率を無視する必要があります。

```
1 - ( parse count (hard) / parse count (total) )
```

メモリー内ソート率 : メモリー内で実行されるソートの比率を示します。運用（OLTP）システムでは、大半のソートが小さく、メモリー・ソートとしてのみ実行できることが理想的です。

```
sorts (memory) / ( sorts (memory) + sorts (disk) )
```

解析対実行の比率 : 操作環境では、SQL 文を 1 回解析し、何回も実行することが理想的です。

```
1 - (parse count/execute count)
```

解析 CPU 対合計 CPU の比率 : 使用した総 CPU タイムのうちのどのくらいの時間が、解析以外のアクティビティに費やされたかを示します。この比率が低いと、システムが実行している解析は多すぎます。

```
1 - (parse time cpu / CPU used by this session)
```

解析時間 CPU 対解析経過時間 : 多くの場合、ラッチの競合を示している可能性があります。この比率では、解析に費やす時間が CPU サイクル（すなわち、生産的な作業）に割り当てられているか、または解析に費やす時間が CPU サイクル上で費やされなかったかを計算します。CPU サイクル以外での解析に費やされた時間は通常、ラッチの競合のために時間がスリープに費やされたことを示します。

```
parse time cpu / parse time elapsed
```

V$SYSSTAT 統計からのロード・プロファイル・データ

システムのロード・プロファイルを決定するには、次の統計を秒およびトランザクションで正規化します。logons cumulative、parse count (total)、parse count (hard)、executes、physical reads、physical writes、block changes および redo size。

正規化されたデータを調べて使用率が高いかどうかを調べたり、そのデータを別のベースライン・データ・セットと比較してシステム・プロファイルが一定期間にどのように変化しているかを識別できます。たとえば、各トランザクションのブロックの変化は次のように計算されます。

```
db block changes / ( user commits + user rollbacks )
```

負荷を測定する追加の計算済み統計の内容は、次のとおりです。

- 各読込みでのブロックの変化 :

  ブロック読込みに対するブロック変化の比率です。この比率は、システムがほとんど読込みのみであるかどうか、またシステムが多数のデータ変更（挿入 / 更新 / 削除）を行うかどうかを示します。

  ```
  db block changes / session logical reads
  ```

- 各ソートの行数 :

  ```
  sorts (rows) / ( sorts (memory) + sorts (disk) )
  ```

V$SYSSTAT の結合列

表 24-30 は、V$SYSSTAT の結合列をリストしたものです。

表 24-30 V$SYSSTAT の結合列

列	ビュー	結合列
STATISTIC#	V$STATNAME	STATISTIC#

V$SYSTEM_EVENT

このビューは、インスタンスによるイベントの待機のサマリーです。V$SESSION_WAIT がシステムの現在の待機を示すのに対して、V$SYSTEM_EVENT はインスタンス起動後のインスタンスのすべてのイベント待機のサマリーを示します。このビューは、システム上の待機に関する履歴を表示する場合に役立ちます。2 つのスナップショットを作成して待機に対して差分をとると、一定の期間内のシステム上の待機を識別できます。

V$SYSTEM_EVENT の有益な列

- EVENT: 待機イベントの名前
- TOTAL_WAITS: このイベントに対する総待機数
- TIME_WAITED: このイベントの待機の総時間（1/100 秒単位）
- AVERAGE_WAIT: このセッションによるこのイベントに対する平均待機時間（1/100 秒単位）
- TOTAL_TIMEOUTS: 待機がタイムアウトになった時間数

例 24-31 システム上での総待機数の検索

```
SELECT event, total_waits waits, total_timeouts timeouts,
       time_waited total_time, average_wait avg
  FROM V$SYSTEM_EVENT
  ORDER BY 4 DESC;

EVENT                                  WAITS  TIMEOUTS TOTAL_TIME      AVG
---------------------------------- ---------- --------- ---------- --------
SQL*Net message from client         112079628         0 8622695365    76.93
virtual circuit status               83559794   1168000 4275791401    51.17
rdbms ipc message                   131463191 115900505 2865926648    21.80
dispatcher timer                    311975975 168152330 2296760866     7.36
PX Idle Wait                          7198490   7198559 1439690729   199.99
pmon timer                             939711    939639  287866277   306.33
smon timer                               9892      9114  287627013 29076.73
lock manager wait for remote mes     72001548  71967858  287526387     3.99
db file sequential read              29419894         0   32395392     1.10
PL/SQL lock timer                       19725     19688   29702609  1505.83
log file sync                         7055611        86    9550819     1.35
log file parallel write               7184801         4    8123534     1.13
SQL*Net more data from client          991402         0    3543149     3.57
db file parallel write                 727317         0    3012928     4.14
control file parallel write            950531         0    1975646     2.07
log file sequential read              1162465         0     813715     0.69
enqueue                                  9975      7692     423191    42.42
direct path read                       453873         0     298944     0.65
db file scattered read                 347172         0     292875     0.84
```

```
row cache lock                          472207        25     169365      0.35
direct path write                       124323         0     132075      1.06
buffer busy due to global cache         148122         0     122381      0.82
SQL*Net more data to client           17171954        52     101762      0.00
db file parallel read                    68849         0     100842      1.46
DFS lock handle                          18615      1080      97651      5.24
SQL*Net message to client             112079756         0      77604      0.00
control file sequential read             65793         0      62560      0.95
buffer busy waits                       132402        97      60351      0.45
latch free                               67675     57975      58365      0.86
log file switch completion                1449        24      34244     23.63
db file single write                     10868         0      25518      2.34
SQL*Net break/reset to client            19130         0       9387      0.49
LGWR wait for redo copy                 120199       356       8613      0.07
global cache lock busy                    4447         0       7574      1.70
undo segment extension                 5363841   5363828       6375      0.00
log file single write                     2143         0       6267      2.92
refresh controlfile command               2644         0       4837      1.82
library cache load lock                     49        10       3859     78.75
file open                                178566         0       2930      0.01
switch logfile command                     100         0       2468     24.68
library cache pin                         9261         1       1716      0.18
pipe get                                     9         3       1460    162.22
rdbms ipc reply                          10296         0        846      0.08
wait for gms registration                   32        32        672     21.00
process startup                             43         2        662     15.39
file identify                             5438         0        584      0.10
control file single write                  332         0        475      1.43
Null event                                  17        17        409     24.05
log buffer space                            18         0        209     11.61
wait for lock db to unfreeze                 1         1        199    199.00
local write wait                            11         0         44      4.00
LMON wait for LMD to inherit commu           1         1         10     10.00
wait for lock db to become frozen            2         2          3      1.50
instance state change                        2         0          0      0.00
global cache bg acks                         2         0          0      0.00
buffer deadlock                            141       141          0      0.00
```

ボトルネックを検索するには、次のようにします。

- Statspack では、最後にアイドル・イベントが示されます。
- 様々なイベントの待機に費やした時間を調べます。
- 各待機の平均時間も調べてください。いくつかの待機（`log file switch completion` など）は定期的に発生しますが、発生時には大きなパフォーマンス・ヒットが生じるためです。

V$UNDOSTAT

このビューでは、現行インスタンスで UNDO 領域とトランザクションがどのように実行されているかを監視します。インスタンス内の UNDO 領域の消費、トランザクションの並行性および問合せの長さに関する統計を使用できます。

V$UNDOSTAT で有益な列

- END_TIME: 10 分間隔ごとの終了時間
- UNDOBLKS: 消費された UNDO ブロックの総数
- MAXCONCURRENCY: 同時に実行されたトランザクションの最大数
- TXNCOUNT: 間隔内で実行された総トランザクション数
- MAXQUERYLEN: インスタンス内で実行された問合せの最大長（秒単位）
- UNXPSTEALCNT: その間隔内で UNDO エクステントをある UNDO セグメントから別の UNDO セグメントに転送する必要のある回数
- SSOLDERRCNT: その間隔内で発生した 'Snapshot Too Old' エラーの個数
- UNDOTSN: 各期間中にサービス状態にあった UNDO 表領域

ビューの最初の行は、現在の時間間隔に関する統計を示します。以降の各行は、10 分間隔を表します。24 時間のサイクルにわたって合計 144 行あります。

例 24-32 V$UNDOSTAT の問合せ

この例では、時刻 16:07 以前の過去 24 時間にシステム内で UNDO 領域がどのように消費されたかを示します。

```
SELECT * FROM V$UNDOSTAT;

END_TIME  UNDOBLKS  MAXCONCURRENCY  TXNCOUNT  MAXQUERYLEN  UNXPSTEALCNT  SSOLDERRCNT
--------  --------  --------------  --------  -----------  ------------  -----------
16:07          252              15      1511           25             2            0
16:00          752              16      1467          150             0            0
15:50          873              21      1954           45             4            0
15:40         1187              45      3210          633            20            1
15:30         1120              28      2498         1202             5            0
15:20          882              22      2002           55             0            0
```

収集された統計で、ピークの UNDO 消費が（15:30、15:40）の間隔で発生したことがわかります。10 分間に 1187 個の UNDO ブロック（すなわち、1 秒間に約 2 個のブロック）が消費されました。また、最高のトランザクション並行性は同時に 45 個のトランザクションを実行する同じ期間中に発生しました。最長の問合せ（1202 秒）は期間（15:20、15:30）内に実行（および終了）されました。問合せは実際に間隔（15:00、15:10）で開始され、約 15:20 まで続いたことに注意してください。

V$WAITSTAT

このビューは、インスタンス起動以降のすべてのバッファ待機のサマリーを保持します。このビューは、システム上に多数のバッファ・ビジーがある場合にクラス別に待機を分析する際に役立ちます。

V$WAITSTAT の有益な列

- `class`: ブロックのクラス（データ・セグメント・ヘッダー、UNDO セグメント・ヘッダー、データ・ブロック）
- `waits`: このクラスのブロックに対する待機数
- `time`: このクラスのブロックに対する総待機時間

待機の理由

次に、考えられる待機の理由を示します。

- UNDO セグメント・ヘッダー : 十分なロールバック・セグメントがない
- データ・セグメント・ヘッダー / 空きリスト : 空きリストの競合
- データ・ブロック
- バッファの CR クローンの大量発生
- 多数の削除を持つ索引に対するレンジ・スキャン
- 多数の削除済み行を持つ表での全表スキャン
- 並行性の高いブロック

> **関連項目：** 待機イベントの詳細は、第 22 章「インスタンスのチューニング」を参照してください。

A

パフォーマンスの例で使用されているスキーマ

この付録では、このマニュアルの様々な例で使用されている表を示します。統計は代表的なシステムからのものです。

この付録では、次の内容を説明します。

- PER_ALL_PEOPLE_F 表
- RA_CUSTOMERS 表
- SO_HEADERS_ALL 表および SO_HEADERS 表
- MTL_SYSTEM_ITEMS 表
- SO_LINES_ALL 表および SO_LINES 表

注意： これらのスキーマは、第 9 章「EXPLAIN PLAN の使用方法」など、各章内の例で使用されています。

PER_ALL_PEOPLE_F 表

この表には、システム上の社員に関するデータが格納されています。大企業の場合は、この表に 10,000 ～ 30,000 個の行があります。一意キーは連結索引ですが、person_id 自体にも非常に選択性があります。その他の選択性のある列は、employee_number と full_name です。

表の索引は次のとおりです。

```
Unique Index Name           Column Name
------ -------------------- ------------------------
NO     PER_PEOPLE_F_FK1     BUSINESS_GROUP_ID
NO     PER_PEOPLE_F_FK2     PERSON_TYPE_ID
NO     PER_PEOPLE_F_N50     LAST_NAME
NO     PER_PEOPLE_F_N51     EMPLOYEE_NUMBER
NO     PER_PEOPLE_F_N52     APPLICANT_NUMBER
NO     PER_PEOPLE_F_N53     NATIONAL_IDENTIFIER
NO     PER_PEOPLE_F_N54     FULL_NAME
YES    PER_PEOPLE_F_PK      PERSON_ID
                            EFFECTIVE_START_DATE
                            EFFECTIVE_END_DATE
```

RA_CUSTOMERS 表

この表には、システム内のすべての顧客に対して 1 行があります。大企業の場合は、この表には数十万個の行があります。主キーは customer_id です。次に、その他の選択性のある列を示します。

- Customer_id
- Customer_name
- Orig_system_reference（他のシステムからインポートされた顧客の顧客識別子を追跡する）

表の索引は次のとおりです。

```
Unique Index Name           Column Name
------ -------------------- ------------------------------
NO     RA_CUSTOMERS_N1      CUSTOMER_NAME
NO     RA_CUSTOMERS_N2      CREATION_DATE
NO     RA_CUSTOMERS_N3      CUSTOMER_KEY
NO     RA_CUSTOMERS_N4      JGZZ_FISCAL_CODE
YES    RA_CUSTOMERS_U1      CUSTOMER_ID
YES    RA_CUSTOMERS_U2      ORIG_SYSTEM_REFERENCE
YES    RA_CUSTOMERS_U3      CUSTOMER_NUMBER
```

SO_HEADERS_ALL 表および SO_HEADERS 表

この表には、システムの各オーダーのための行があります。一般的に大企業の場合は、この表に数百万個の行があります。主キーは header_id であり、(order_number、order_type_id) 上に別の一意キーがあります。次に、その他の選択性のある列を示します。

- customer_id (オーダーを発行する顧客)
- purchase_order_num (請求書発行のための発注書)
- original_system_reference (他のシステムからインポートされたオーダーのオーダー識別子を追跡する)

表の索引は次のとおりです。

Unique	Index Name	Column Name
NO	SO_HEADERS_N1	CUSTOMER_ID
NO	SO_HEADERS_N10	WH_UPDATE_DATE
NO	SO_HEADERS_N2	OPEN_FLAG
NO	SO_HEADERS_N3	PURCHASE_ORDER_NUM
NO	SO_HEADERS_N4	INVOICE_TO_SITE_USE_ID
NO	SO_HEADERS_N5	ORIGINAL_SYSTEM_REFERENCE
NO	SO_HEADERS_N6	S1
NO	SO_HEADERS_N7	S4
NO	SO_HEADERS_N8	S6
NO	SO_HEADERS_N9	ORIGINAL_SYSTEM_REFERENCE
		ORIGINAL_SYSTEM_SOURCE_CODE
YES	SO_HEADERS_U1	HEADER_ID
YES	SO_HEADERS_U2	ORDER_NUMBER
		ORDER_TYPE_ID

MTL_SYSTEM_ITEMS 表

この表は、so_lines_all の部品のマスターです。この表には、すべての組織のすべての部門に対応する行があります。主キーは、inventory_item_id および organization_id です。

表の索引は次のとおりです。

Unique	Index Name	Column Name
NO	MTL_SYSTEM_ITEMS_N1	ORGANIZATION_ID
		SEGMENT1
NO	MTL_SYSTEM_ITEMS_N2	ORGANIZATION_ID
		DESCRIPTION
NO	MTL_SYSTEM_ITEMS_N3	INVENTORY_ITEM_STATUS_CODE
NO	MTL_SYSTEM_ITEMS_N4	ORGANIZATION_ID
		AUTO_CREATED_CONFIG_FLAG
NO	MTL_SYSTEM_ITEMS_N5	WH_UPDATE_DATE
NO	MTL_SYSTEM_ITEMS_N6	ITEM_CATALOG_GROUP_ID
		CATALOG_STATUS_FLAG
NO	MTL_SYSTEM_ITEMS_N7	PRODUCT_FAMILY_ITEM_ID
		ORGANIZATION_ID
NO	MTL_SYSTEM_ITEMS_N8	SEGMENT1
		SEGMENT2
		SEGMENT3
YES	MTL_SYSTEM_ITEMS_U1	INVENTORY_ITEM_ID
		ORGANIZATION_ID

SO_LINES_ALL 表および SO_LINES 表

この表には、システム内のすべてのオーダー明細のための行があります。この行は、header_id を使用して so_headers_all 表に結合します。オーダーには 10 ～ 12 個の明細行があるので、この表は so_headers_all 内の行の 10 ～ 12 倍です。主キーは line_id です。次に、その他の選択性のある列を示します。

- header_id
- parent_line_id
- service_parent_line_id
- original_system_reference

表の索引は次のとおりです。

Unique	Index Name	Column Name
NO	SO_LINES_N1	HEADER_ID
NO	SO_LINES_N10	S5
NO	SO_LINES_N11	S6
NO	SO_LINES_N12	S8
NO	SO_LINES_N13	S9
NO	SO_LINES_N14	S28
NO	SO_LINES_N15	S29
NO	SO_LINES_N16	S30
NO	SO_LINES_N17	PARENT_LINE_ID
NO	SO_LINES_N18	SHIPMENT_SCHEDULE_LINE_ID
NO	SO_LINES_N19	ATO_LINE_ID
NO	SO_LINES_N2	LINK_TO_LINE_ID
NO	SO_LINES_N20	SERVICE_PARENT_LINE_ID
NO	SO_LINES_N21	SHIP_TO_SITE_USE_ID
NO	SO_LINES_N22	SOURCE_LINE_ID
NO	SO_LINES_N23	ORIGINAL_SYSTEM_LINE_REFERENCE
NO	SO_LINES_N24	RETURN_REFERENCE_ID
NO	SO_LINES_N25	S27
NO	SO_LINES_N26	CREDIT_INVOICE_LINE_ID
NO	SO_LINES_N27	S25
NO	SO_LINES_N28	WH_UPDATE_DATE
NO	SO_LINES_N29	DEMAND_STREAM_ID
NO	SO_LINES_N3	OPEN_FLAG
NO	SO_LINES_N4	COMMITMENT_ID
NO	SO_LINES_N5	INVENTORY_ITEM_ID
NO	SO_LINES_N6	REQUEST_ID
NO	SO_LINES_N7	S2
NO	SO_LINES_N8	S3
NO	SO_LINES_N9	S4
YES	SO_LINES_U1	LINE_ID

用語集

CBO

コストベース・オプティマイザ（Cost-based optimizer）。SQL 文の潜在的な実行計画セットの生成、各計画のコストの見積り、計画を生成するプラン・ジェネレータのコールを実行し、コストを比較して最も低コストの計画を選択します。この方法は、SQL 文でアクセスされる表の少なくとも 1 つに関する統計がデータ・ディクショナリに含まれている場合に使用されます。CBO は、問合せトランスフォーマ、エスティメータおよびプラン・ジェネレータで構成されます。

EXPLAIN PLAN

DML 文に対してオプティマイザが選択した実行計画を検証できる SQL 文。`EXPLAIN PLAN` を実行すると、オプティマイザは実行計画を選択し、計画を説明するデータをデータベース表に格納します。

LIO

論理 I/O。バッファ・キャッシュから読むことができる場合とできない場合のあるブロック読込み。

MTBF

平均障害間隔時間（Mean time between failures）。一般的なデータベース統計で、I/O のチューニングに重要です。

Oracle Trace

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 Server が使用する機能。Oracle Trace には、SQL スクリプトがいくつか用意されています。これらのスクリプトを使用すると、サーバーのイベント表にアクセスし、サーバーのイベント・データを収集してメモリーに格納します。また、収集しながらデータをフォーマットできます。

PGA

プログラム・グローバル領域。サーバー・プロセスのデータと制御情報が含まれる非共有のメモリー領域。サーバー・プロセスの開始時に作成されます。

PIO

物理 I/O。ブロックがバッファ・キャッシュに存在しなかったか、I/O がダイレクト I/O である（すなわちバッファ・キャッシュをバイパスする）ため、バッファ・キャッシュから読むことのできなかったブロック読込み。

RAID

Redundant Arrays of Inexpensive Disks。RAID 構成では、ストライプ化（手動によるデータの分散化）とともに、高いデータの信頼性を実現します。パフォーマンスとコストに基づいて様々な RAID 構成（レベル）が選択され、その I/O 特性に応じて、適応がふさわしいアプリケーションは異なります。

RBO

ルールベース・オプティマイザ（Rule-based optimizer）。使用可能なアクセス・パスとそのアクセス・パスのランクに基づいて、SQL 文の実行計画を選択します（複数の方法がある場合は、ランクの低い操作が使用されます）。RBO は、使用できる統計がない場合に使用されます。使用できる統計がある場合は CBO が使用されます。

SGA

システム・グローバル領域（System Global Area）。高速なアクセスのためにデータを格納するメイン・メモリー内のメモリー領域。Oracle では、共有 SQL および PL/SQL プロシージャ用の SGA メモリーの割当てに共有プールが使用されます。

SQL*Loader

入力ファイルを読み込み、解析します。大量のデータをロードする最も効率的な方法です。

SQL コンパイラ（SQL Compiler）

SQL 文を共有カーソルにコンパイルします。SQL コンパイラは、パーサー、オプティマイザおよび行ソース・ジェネレータで構成されます。

SQL トレース（SQL Trace）

基本的なパフォーマンス診断ツール。Oracle Server で実行されるアプリケーションのモニターとチューニングに役立ちます。SQL トレースを使用すると、アプリケーションで実行される SQL 文の効率性を評価し、各文の統計を生成できます。このツールで生成されるトレース・ファイルは、TKPROF の入力として使用されます。

SQL 文（同一）（SQL statements （identical））

テキストが同じ SQL 文は、すべての点で同一です。

SQL 文（類似）（SQL statements（similar））

類似する SQL 文は、リテラル値を変更するという点のみ異なります。リテラル値がバインド変数に置換された場合、SQL 文はテキストの点で同一になります。

Statspack

パフォーマンス・データの収集、自動化、格納および表示ができる SQL、PL/SQL および SQL*Plus スクリプトのセット。従来の UTLBSTAT/UTLESTAT チューニング・スクリプトに置き換わるものです。

TKPROF

診断ツール。Oracle Server で実行されるアプリケーションのモニターとチューニングに役立ちます。TKPROF は主に、SQL トレースの出力ファイルを処理して読取り可能な出力ファイルに変換し、トレース・ファイルに関してユーザー・レベルの文と再帰的 SQL コールのサマリーを提供します。また、SQL 文の効率性の評価、実行計画の生成、データベースに統計を格納する SQL スクリプトの作成が実行できます。

UGA

ユーザー・グローバル領域（User Global Area）。ユーザー・セッションに使用されるラージ・プール内のメモリー領域。

インスタンス・リカバリ（instance recovery）

クラッシュまたはシステム障害後に、コミットされていない Oracle データ・ブロックに REDO ログ・レコードを自動的に適用すること。

エスティメータ（estimator）

統計を使用して、選択性、カーディナリティおよび実行計画のコストを評価します。エスティメータの主な目標は、実行計画のコスト全体を算出することです。

エンキュー（enqueue）

これはロックの別の用語です。

オプティマイザ（optimizer）

SQL 文の式を評価し、より処理速度の速い等価の式に変換することによって、SQL 文を最も効率的に実行する方法を判断します。オプティマイザは実行計画セットを示し、SQL 文に対して最適な計画を選択します。**CBO** を参照してください。

解析（parse）

ハード解析は、SQL 文が実行され、かつ SQL 文が共有プール内にないか、共有プール内にあっても共有できないときに行われます。SQL 文は、2 つの SQL 文のメタデータが異なる場合に共有されません。これは、ある SQL 文が既存の SQL 文とテキストでは同一ですが、2 つの文で参照される表が物理的に異なる表に変換される場合、またはオプティマイザ環境が異なる場合に発生する可能性があります。

ソフト解析は、セッションが SQL 文を実行しようとし、かつ文がすでに共有プール内にあり、かつ文を使用できる（すなわち、共有できる）ときに行われます。共有される文の場合、既存の SQL 文に関連するすべてのデータ（オプティマイザ実行計画などのメタデータも含む）が発行対象の現行の文に同じように適用できる必要があります。

解析コール（parse call）

SQL 文実行の準備のために Oracle に行われるコール。このコールには、SQL 文の構文チェック、SQL 文の最適化、SQL 文の実行可能形式の作成（または指定）が含まれています。

外部結合（outer join）

結合する表のどれか 1 つの 1 つ以上の列で外部結合演算子（+）を使用する結合条件。Oracle は、この結合条件に一致する行をすべて戻します。また、外部結合演算子がついておらず外部結合演算子を持つ表に対応する行が存在しない表からは、すべての行を戻します。

キャッシュ（cache）

バッファ・キャッシュとも呼ばれています。すべてのバッファおよびバッファ・プール。

キャッシュ・リカバリ（cache recovery）

REDO ログ・ファイルのすべてのコミット済みおよびコミットされていない変更内容を、対象のデータ・ブロックに適用するインスタンス・リカバリの部分。インスタンス・リカバリのロールフォワード・フェーズとも呼ばれています。

競合（contention）

あるプロセスが別のプロセスで使用されているリソースを待機する必要のある場合。

行ソース・ジェネレータ（row source generator）

オプティマイザから最適な計画を受け取り、SQL 文の実行計画を出力します。行ソースは、1 組の行を反復方式で処理し、行セットを生成する反復制御構造です。

結合（join）

複数の表からデータを選択する問合せ。結合の特徴は、FROM 句内に複数の表が並んでいる点です。Oracle は、WHERE 句に指定された条件を使用して、それぞれの表の行を比較し、その結果作成された行を戻します。この条件は結合条件と呼ばれます。通常、結合されたすべての表の列はこの条件によって比較されます。

作業領域（work area）

メモリーのプライベート割当てであり、ソート、ハッシュ結合およびオンメモリー中心のその他の操作に使用されます。ソート演算子は、作業領域（ソート領域）を使用して一連の行のメモリー内ソートを実行します。同様に、ハッシュ結合演算子は作業領域（ハッシュ領域）を使用して、ハッシュ表を左側から入力して作成します。

自動トレース（Autotrace）

SQL オプティマイザが使用した実行パス、および文の実行統計に関するレポートを生成します。レポートは、DML 文のパフォーマンスの監視およびチューニングに役立ちます。

述語（predicate）

SQL 内の WHERE 条件。

ストライプ化（striping）

ディスク間でのデータのブロックの並列処理化。適切なストライプ化は I/O を減らし、パフォーマンスを向上します。

- ストライプ深度は、ストライプのサイズで、ストライプ単位とも呼ばれます。
- ストライプ幅は、ストライプ深度とストライプ・セットを構成するドライブ数の積です。

セグメント（segment）

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エクステントの集合です。

待機イベント（wait events）

処理を継続する際にあるイベントが完了するまで待機する必要があったことを示すために、サーバー・プロセス / スレッドによって増やされる統計。待機イベントは、事後パフォーマンス・チューニングの実行時に最初に検証する事項の 1 つです。

待機イベント（アイドル）（wait events（idle））

これらのイベントは、作業がないためにサーバー・プロセスが待機していることを示します。アイドル・イベントをチューニングのときに無視することが必要なのは、アイドル・イベントがパフォーマンスのボトルネックの性質を示さないからです。

単純な問合せ（simple query）

1 つの表のみを参照し、GROUP BY 関数の参照は行わない SELECT 文。

単純な文（simple statement）

単一表のみに関連する INSERT 文、UPDATE 文、DELETE 文または SELECT 文。

ディクショナリ・キャッシュ（dictionary cache）

データベース、その構造およびそのユーザーに関する参照情報を含むデータベース表とビューを集めたもの。Oracle は、SQL 文の解析時にデータ・ディクショナリに頻繁にアクセスします。ディクショナリ・データを保持するために、メモリー内に 2 つの特別な場所があります。1 つはデータ・ディクショナリ・キャッシュと呼ばれ、データをバッファ（データ・ブロック全体を保持する）のかわりに行として保持するために行キャッシュとも呼ばれます。もう 1 つの領域はライブラリ・キャッシュです。すべての Oracle ユーザー・プロセスは、データ・ディクショナリ情報にアクセスするためにこれらの 2 つのキャッシュを共有しています。

デカルト積（Cartesian product）

結合条件を使用しない結合はデカルト積（クロス積）に帰結します。デカルト積は、各表から 1 つずつ選んだ行の作成可能なすべての組合せです。つまり、2 つの表の結合において、1 つの表の各行がもう一方の表のすべての行とそれぞれ組み合わせられます。3 つ以上の表のデカルト積は、1 つの表の各行と残りの表のデカルト積のすべての行をそれぞれ組み合せた結果です。その他のすべての種類の結合は、デカルト積から、効率的に結合条件を満たさない行を絞り込んで作成されたデカルト積のサブセットです。

問合せトランスフォーマ（query transformer）

ユーザー問合せをリライトして、より効率的な問合せ計画を生成するかどうかを判断し、ビューをマージして副問合せのネスト解除を実行します。

等価結合（equijoin）

等価演算子が含まれている結合条件。

動的パフォーマンス・ビュー（dynamic performance views）

データベース管理者が動的パフォーマンス表（現在のデータベース・アクティビティを記録する仮想表）に作成するビュー。データベース管理者が変更または削除できないため、固定ビューと呼ばれます。

トランザクション・リカバリ（transaction recovery）

Oracle がコミットされない変更を元に戻すためにロールバック・セグメントを適用するインスタンス・リカバリの部分。インスタンス・リカバリのロールバック・フェーズとも呼ばれています。

パーサー（parser）

SQL 文の構文分析とセマンティック分析を実行し、（問合せで参照される）ビューを個別の問合せブロックに展開します。

バインド変数（bind variable）

SQL 文の中の変数。SQL 文を正常に実行するには、バインド変数を有効な値または値のアドレスに置換する必要があります。

バッファ（buffer）

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換します。

バッファ・プール（buffer pool）

バッファの集まり。

非等価結合（nonequijoin）

等価演算子以外の演算子が含まれている結合条件。

非同期 I/O（asynchronous I/O）

独立 I/O。転送に対する時間的な要件がなく、転送が終了する前に他のプロセスを開始できます。

複合問合せ（compound query）

2 つ以上の単純な文または複合文を組み合せるために集合演算子（UNION、UNION ALL、INTERSECT または MINUS）を使用する問合せ。複合問合せ内の単純な文または複合文それぞれは、コンポーネント・クエリーと呼ばれます。

プラン・ジェネレータ（plan generator）

CBO が最も低コストの計画を選択できるように、与えられた問合せに対して考えられる様々な計画を試行します。異なるアクセス・パス、結合方法および結合順序を試みることによって、問合せブロックに対する様々な計画を調査します。

ブロック（block）

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・アドレス空間の 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。

分散型の文（distributed statement）

分散データベースの 2 つ以上の個別ノード / インスタンスのデータにアクセスする文。リモート文は、分散データベースの 1 つのリモート・ノードのデータにアクセスします。

ページング（paging）

プログラムの作業メモリーのうちの使用頻度の低い部分を主メモリーから二次記憶媒体（通常はディスク）に移動することによって使用可能なメモリー領域を増やすための手法。転送単位はページと呼ばれます。

ボトルネック（bottleneck）

一般にはシステムの帯域幅がデータの処理される速度で中継されてくる量をサポートできないときのデータ伝送の遅延。ただし、システム内にボトルネックを生成する可能性のある要因は多数あります。

ミラー化（mirroring）

同じ内容のデータのコピーを 1 つ以上のディスクで保持すること。一般的にミラー化は、オペレーティング・システム・レベルで二重化されたハードディスクにおいて実現します。したがって、いずれかのディスクが使用不可能になっても、中断せずにもう一方のディスクが要求の処理を継続できます。

ライブラリ・キャッシュ（library cache）

共有されている SQL 領域と PL/SQL 領域を保持するメモリー構造。ライブラリ・キャッシュは共有プールの 3 つの部分のうちの 1 つです。

ラッチ（latch）

システム・グローバル領域で共有されているデータ構造を保護するための簡素な下位レベルのシリアル化メカニズム。

リテラル（literal）

コンパイル時に書き込まれ、実行時に読取り専用の定数の値。リテラルは高速にアクセスでき、また、修正が不要なときに使用します。

索引

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

Q

R

S

T

U

V

あ

い

し

す

せ

そ

た

ち

て

と

に

ね

は

ひ

ふ

へ

ほ

み

め

も

ゆ

よ

ら

り

る

れ

ろ

わ