

# Oracle9i

データベース・パフォーマンス・プランニング

リリース 2 (9.2)

2002 年 7 月

部品番号 : J06251-01

ORACLE®

---

Oracle9i データベース・パフォーマンス・プランニング, リリース 2 (9.2)

部品番号 : J06251-01

原本名 : Oracle9i Database Performance Planning, Release 2 (9.2)

原本部品番号 : A96532-01

原本著者 : Lenore Luscher

原本協力者 : Andrew Holdsworth, Jorn Bartels, Maria Colgan, Michele Cyran, Bjorn Engsig, Cecilia Gervasio, Connie Dialeris Green, Mattias Jankowitz, Peter Kilpatrick, Anjo Kolk, JP Polk, Virag Saksena, Sabrina Whitehouse, Graham Wood, Valarie Moore

Copyright © 2001, 2002 Oracle Corporation. All rights reserved.

Printed in Japan.

制限付権利の説明

プログラム (ソフトウェアおよびドキュメントを含む) の使用、複製または開示は、オラクル社との契約に記された制約条件に従うものとします。著作権、特許権およびその他の知的財産権に関する法律により保護されています。

当プログラムのリバース・エンジニアリング等は禁止されています。

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

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

危険な用途への使用について

オラクル社製品は、原子力、航空産業、大量輸送、医療あるいはその他の危険が伴うアプリケーションを用途として開発されておりません。オラクル社製品を上述のようなアプリケーションに使用することについての安全確保は、顧客各位の責任と費用により行ってください。万一かかる用途での使用によりクレームや損害が発生いたしましても、日本オラクル株式会社と開発元である Oracle Corporation (米国オラクル) およびその関連会社は一切責任を負いかねます。当プログラムを米国国防総省の米国政府機関に提供する際には、『Restricted Rights』と共に提供してください。この場合次の Notice が適用されます。

Restricted Rights Notice

Programs delivered subject to the DOD FAR Supplement are "commercial computer software" and use, duplication, and disclosure of the Programs, including documentation, shall be subject to the licensing restrictions set forth in the applicable Oracle license agreement. Otherwise, Programs delivered subject to the Federal Acquisition Regulations are "restricted computer software" and use, duplication, and disclosure of the Programs shall be subject to the restrictions in FAR 52.227-19, Commercial Computer Software - Restricted Rights (June, 1987). Oracle Corporation, 500 Oracle Parkway, Redwood City, CA 94065.

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

---

---

# 目次

はじめに .....	iii
対象読者 .....	iv
このマニュアルの構成 .....	iv
関連文書 .....	iv
表記規則 .....	v
1 パフォーマンスを考慮した設計と開発 .....	
オラクル社の新しい方法論 .....	1-2
投資対効果 .....	1-2
拡張性の理解 .....	1-3
拡張性 .....	1-3
インターネットにおける拡張性 .....	1-4
拡張性を妨げる要因 .....	1-6
システム・アーキテクチャ .....	1-7
ハードウェア・コンポーネントとソフトウェア・コンポーネント .....	1-7
要件に合った正しいシステム・アーキテクチャの構成 .....	1-10
アプリケーション設計の原理 .....	1-13
アプリケーション設計の簡潔さ .....	1-13
データ・モデル化 .....	1-14
表および索引の設計 .....	1-14
ビューの使用 .....	1-17
SQL の実行効率 .....	1-17
アプリケーションの実装 .....	1-19
アプリケーション開発の傾向 .....	1-21

ワークロードのテスト、モデル化および実装 .....	1-22
データのサイズ設定 .....	1-22
ワークロードの見積り .....	1-22
アプリケーションのモデル化 .....	1-23
設計のテスト、デバッグおよび検証 .....	1-24
新しいアプリケーションの配置 .....	1-25
ロールアウトの方法 .....	1-25
パフォーマンス・チェックリスト .....	1-26

## 2 アプリケーションのパフォーマンスの監視と改善

統計情報の重要性 .....	2-2
統計情報収集ツール .....	2-6
履歴データとベースラインの重要性 .....	2-8
パフォーマンスに対する直観的判断 .....	2-8
Oracle のパフォーマンス改善方法 .....	2-9
パフォーマンス改善の概要 .....	2-9
Oracle のパフォーマンス改善方法の手順 .....	2-11
オペレーティング・システムのチェック方法 .....	2-12
パフォーマンスを概念的にモデル化する際の意志決定プロセスの例 .....	2-12
Oracle システムにおける誤りの上位 10 項目 .....	2-13
ハードウェア構成のパフォーマンス特性 .....	2-15

## 3 パフォーマンスの緊急事態に対処する方法

パフォーマンスの緊急事態に対処する方法の概要 .....	3-2
パフォーマンスの緊急事態に対処する方法の手順 .....	3-2

## 索引

---

# はじめに

このマニュアルでは、アプリケーションを適切に設計し、統計情報を利用してアプリケーションのパフォーマンスを監視することによって **Oracle** のパフォーマンスを改善する方法を説明します。**Oracle** のパフォーマンスを改善する方法の他に、パフォーマンス上の問題に対処するために、パフォーマンスの緊急事態に対処する方法も説明します。

ここでは、次の項目について説明します。

- [対象読者](#)
- [このマニュアルの構成](#)
- [関連文書](#)
- [表記規則](#)

## 対象読者

『Oracle9i データベース・パフォーマンス・プランニング』は、Oracle の操作、メンテナンスおよびパフォーマンスの責任者を支援するための高度なメソッドです。このマニュアルは、データベース管理者、アプリケーション設計者、プログラマまたはマネージャを対象にしています。読者は、Oracle9i、オペレーティング・システムおよびアプリケーション設計をよく理解している必要があります。

## このマニュアルの構成

このマニュアルの構成は、次のとおりです。

### 第 1 章「パフォーマンスを考慮した設計と開発」

Oracle アプリケーションの設計時に考慮するパフォーマンスの問題について説明します。

### 第 2 章「アプリケーションのパフォーマンスの監視と改善」

Oracle のパフォーマンスを改善する方法、およびアプリケーションのパフォーマンス改善に必要な統計情報の重要性について説明します。

### 第 3 章「パフォーマンスの緊急事態に対処する方法」

パフォーマンスに関する緊急事態に対処する方法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、『Oracle9i データベース概要』、『Oracle9i アプリケーション開発者ガイド - 基礎編』および『Oracle9i 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については、『Oracle Enterprise Manager 概説』および『Oracle Enterprise Manager 管理者ガイド』を参照してください。

Oracle Application Server のチューニングについては、『Oracle Application Server パフォーマンス・チューニング・ガイド』を参照してください。

この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、『Oracle9i 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](#)
- [コード例の表記規則](#)

## 本文の表記規則

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

規則	意味	例
太字	太字は、本文中に定義されている用語または用語集に含まれている用語、あるいはその両方を示します。	この句を指定する場合は、 <b>索引構成表</b> を作成します。
イタリック	イタリックは、構文の句またはプレースホルダを示します。	<i>parallel_clause</i> を指定できます。  <code>Uold_release.SQL</code> を実行します。 <i>old_release</i> はアップグレード前にインストールしたリリースを指します。
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、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 列を問い合わせます。  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。

規則	意味	例
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

コード例では、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、コード例の表記規則と使用例を示しています。

規則	意味	例
[ ]	大カッコで囲まれている項目は、1 つ以上のオプション項目を示します。大カッコ自体は入力しないでください。	DECIMAL (digits [ , precision ])
{ }	中カッコで囲まれている項目は、そのうちの 1 つのみが必要であることを示します。中カッコ自体は入力しないでください。	{ENABLE   DISABLE}
	縦線は、大カッコまたは中カッコ内の複数の選択肢を区切るために使用します。オプションのうち 1 つを入力します。縦線自体は入力しないでください。	{ENABLE   DISABLE} [COMPRESS   NOCOMPRESS]
...	水平の省略記号は、次のいずれかを示します。 <ul style="list-style-type: none"><li>■ 例に直接関係のないコード部分が省略されていること。</li><li>■ コードの一部が繰り返し可能なこと。</li></ul>	CREATE TABLE ...AS subquery;  SELECT col1, col2, ..., coln FROM employees;

規則	意味	例
. . . . . .	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 ----- /fsl/dbs/tbs_01.dbf /fsl/dbs/tbs_02.dbf . . . . . . /fsl/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がある変数を示します。	CONNECT SYSTEM/system_password
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レベ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。	SELECT last_name, employee_id FROM employees;  sqlplus hr/hr



---

# パフォーマンスを考慮した設計と開発

システムの高いパフォーマンスは設計段階で決まり、その効果は使用しているシステムが稼働している間持続されます。パフォーマンスの問題については、最初の設計段階での綿密な検討によって、本番でのシステムのチューニングが容易になります。

この章は、次の項で構成されています。

- [オラクル社の新しい方法論](#)
- [投資対効果](#)
- [拡張性の理解](#)
- [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](#)
- [新しいアプリケーションの配置](#)

## オラクル社の新しい方法論

コンピュータ・システムの規模が拡大して複雑になり、ビジネス用アプリケーションでのインターネットの役割が重要になるに従い、システムのパフォーマンスはますます重要になっています。このような状況に対応するために、オラクル社はパフォーマンスに関する新しい方法論を構築しました。この方法論は、設計やパフォーマンスに関するオラクル社の豊富な経験に基づいており、システムのパフォーマンスを大幅に向上させる明確で簡潔なアクティビティを紹介しています。

パフォーマンスの方針によってその効果は異なり、システム（業務システムや意思決定支援システムなど）の目的に応じたパフォーマンス手法が求められます。このマニュアルでは、データベース設計者、管理者またはパフォーマンス担当者が考慮する必要がある内容について説明します。

システムのパフォーマンスとは、設計してシステムに組み込むものです。偶然にパフォーマンスがよくなるわけではありません。パフォーマンスに関する問題は、通常、システム・リソースの競合または消耗が原因で発生します。システム・リソースが消耗されると、そのシステムは高いレベルのパフォーマンスに追随することができません。新しいパフォーマンス方法論は、データベースに関する慎重な計画と設計に基づいており、システム・リソースの消耗がシステム停止の原因になることを防止します。システムは、リソースの競合を除去することによって、ビジネス要件を満たすレベルにまで拡張できます。

**関連項目：**『Oracle9i 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』

## 投資対効果

高性能なプロセッサ、メモリーおよびディスク・ドライブが比較的安価で入手できることから、安易なシステム・リソースの追加購入によって、パフォーマンスを改善しようとする動きがあります。多くの場合、新しい CPU、メモリーまたはディスク・ドライブの増設によってパフォーマンスは一時的には確かに改善されます。しかし、ハードウェア増設によるパフォーマンスの改善は、一時的な問題の解決と考えてください。アプリケーションに対する需要率とロード率が増加し続けると、近い将来、同様の問題に直面する可能性が高くなります。

ハードウェアの増設によって、システムのパフォーマンスがまったく改善されない場合もあります。システム設計が不適切な場合、追加のハードウェアを割り当ててもパフォーマンスは改善しません。ハードウェアを追加購入する前には、アプリケーション内でシリアル化やシングル・スレッド化が行われていないことを確認してください。長期的に見ると、各ビジネス・トランザクションで使用する物理リソースの数の点では、一般的に、使用しているアプリケーションの効率を上げるほうが効果的です。

## 拡張性の理解

拡張性という用語は、開発環境での様々な状況で使用されます。次の項では、アプリケーション設計者とパフォーマンス担当者にとって重要な拡張性について説明します。

## 拡張性

拡張性とは、システム・リソースの使用率増加に従って、より高いワークロードを処理するためのシステム能力です。つまり、拡張性があるシステムでは、ワークロードが2倍になるとシステムで使用するシステム・リソースも2倍になります。これは当たり前のようですが、システム内で競合が発生すると、最初のワークロードの場合に比べてリソース使用率が2倍以上になる場合があります。

リソースの競合によって拡張性が低くなる例を次に示します。

- ユーザー数の増加に伴い、アプリケーションでかなりの同時実行性管理が要求される場合
- ロック・アクティビティが増加した場合
- データ整合性に関するワークロードが増加した場合
- オペレーティング・システムのワークロードが増加した場合
- データ量の増加に伴い、トランザクションでのデータ・アクセスの増加が要求される場合
- SQL と索引の不適切な設計が原因で、同じ戻り行数に対する論理 I/O 数が増加した場合
- データベース・オブジェクトでのメンテナンス時間が長くなったため、可用性が低下した場合

アプリケーションのワークロードが増加し、それ以上のスループットは不可能という点までシステム・リソースを消耗した場合、そのアプリケーションには拡張性がないと言えます。このようなアプリケーションでは、スループットが固定化し、応答時間が長くなります。

リソースの消耗の例を次に示します。

- ハードウェアの消耗
- 大量トランザクションでの表スキャンによって、ディスク I/O の不足が発生する場合
- 過剰なネットワーク要求によって、ネットワークとスケジューリングにボトルネックが発生する場合
- メモリ割当てによって、ページングとスワッピングが発生する場合
- プロセスやスレッドの過剰な割当てによって、オペレーティング・システムのスラッシングが発生する場合

以上のことから、アプリケーション設計者は、ユーザー数やデータ量に関係なく同じリソースを使用し、システム・リソースの制約を超える負荷を与えないように設計する必要があります。

## インターネットにおける拡張性

インターネット経由でアクセスできるアプリケーションには、さらに複雑なパフォーマンスや可用性の要件があります。インターネット専用に設計および記述されたアプリケーションもありますが、一般会計アプリケーションなどの典型的なバックオフィス・アプリケーションでさえ、その一部またはすべてのデータがオンラインで使用できるように要求される場合があります。

インターネット世代のアプリケーションの特長は次のとおりです。

- 1日24時間、年365日使用可能
- 同時ユーザーの数が予測不可能で正確な把握が困難
- 容量の計画が困難
- あらゆるタイプの間合せが使用可能
- 複数層アーキテクチャ
- ステートレスなミドルウェア
- 開発サイクルの短縮
- 最小時間でのテストの実行

図 1-1 インターネットのワークロード増加曲線

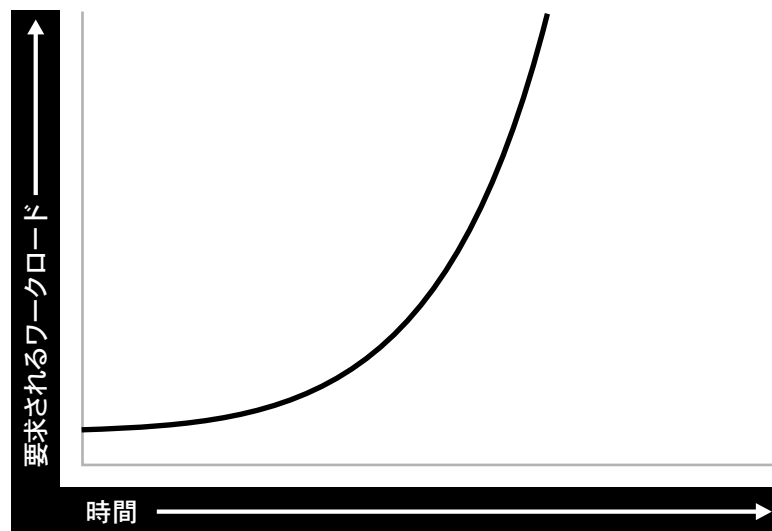


図 1-1 は、需要率の増加に伴う典型的なインターネット /E-Business の需要増加曲線です。アプリケーションは、ワークロードが増加した場合、および需要の増加をサポートするためにハードウェアが増設された場合、それにあわせて拡張できることが必要です。設計に失敗すると、ハードウェア・リソースの増設や再設計に関係なく、実装が限界に達する可能性があります。

インターネット・アプリケーションは非常に短い期間で開発することが要求され、テストや評価の時間も限定されています。しかし、不適切な設計は、将来、システムの再構成や再実装を招くことになります。アーキテクチャや実装に既知の制限があるアプリケーションをインターネット上に配置する場合、およびワークロードが需要予測を超えている場合は、将来、確実に障害が発生します。ビジネスの観点からは、低いパフォーマンスによる顧客の喪失になりかねません。Web ユーザーは、7 秒以内に応答がないと、二度と興味を持つことはありません。

多くの場合、新規の実装に移行するためにシステムを停止する間のコストも含めて、システムを再設計するコストは、最初からシステムを適切に構築した場合のコストより多くかかります。方針は単純です。開発の当初から拡張性に留意して設計と実装を行うことです。

## 拡張性を妨げる要因

アプリケーションの作成時に、設計者とアーキテクチャ担当者は、可能な限り完全な拡張性に近づけることを目指してください。この完全な拡張性は線形拡張性とも呼ばれ、システムのスループットが CPU の数に正比例します。

線形拡張性は、設計者の制御を超える部分があるため、実際には不可能です。しかし、アプリケーション設計や実装に可能な限り拡張性を持たせることは、ハードウェア・コンポーネントの拡張や CPU テクノロジーの発展によって、現在と将来にわたるパフォーマンス目標が達成できることを保証することになります。

### 線形拡張性を妨げる要因

#### 1. 不適切なアプリケーションの設計、実装および構成

アプリケーションは拡張性に最も大きく影響します。次に例を示します。

- 不適切なスキーマ設計によって、SQL にコストがかかり拡張性がなくなります。
- 不適切なトランザクション設計によって、ロックやシリアライズ化の問題が発生します。
- 不適切な接続管理によって、応答時間が長くなりシステムの信頼性が低くなります。

ただし、設計のみが問題ではありません。アプリケーションの物理的な実装が弱点になる場合があります。次に例を示します。

- システムが誤った I/O 方針のまま本番環境で使用される場合があります。
- テスト時に作成された実行計画とは異なる実行計画が本番環境で使用される場合があります。
- 実行時の空きメモリーを十分に考慮せずに大量のメモリー割当てを行うメモリー集中型のアプリケーションによって、過剰な量のメモリーが使用される場合があります。
- 非効率的なメモリー使用やメモリー・リークによって、動作中の仮想メモリー・サブシステムに高いストレスがかかります。このようなストレスは、パフォーマンスや可用性に影響を与えます。

#### 2. ハードウェア・コンポーネントの誤ったサイズ指定

すべてのハードウェア・コンポーネントの誤った容量計画による問題は、関連するハードウェア価格の下落によって軽減されつつあります。ただし、容量が大きすぎると、システム上でのワークロードが増加するにつれて、拡張性の問題がわかりにくくなる場合があります。

### 3. ソフトウェア・コンポーネントの制限

すべてのソフトウェア・コンポーネントには拡張性があり、リソース使用に関する制限があります。このことは、アプリケーション・サーバー、データベース・サーバーおよびオペレーティング・システムでも同様です。アプリケーションの設計では、ソフトウェアの処理能力を超えた要求をしないでください。

### 4. ハードウェア・コンポーネントの制限

ハードウェアの拡張性は完全ではありません。ほとんどのマルチプロセッサ・マシンでは、限られた数の CPU で線形のスケーリングに近づくことができますが、ある特定の数を超えると、CPU を追加しても、その数に正比例したパフォーマンスの改善は期待できません。CPU を追加してもパフォーマンスは向上せず、むしろ低下する場合さえあります。このような状態は、ワークロードとオペレーティング・システムの設定と深く関連しています。

---

**注意：** これらの要因は、オラクル社のサーバー・パフォーマンス・グループが拡張性のないシステムをチューニングする際に得た経験に基づいています。

---

## システム・アーキテクチャ

システムのアーキテクチャには、次の主要な部分があります。

- ハードウェア・コンポーネントとソフトウェア・コンポーネント
- 要件に合った正しいシステム・アーキテクチャの構成

## ハードウェア・コンポーネントとソフトウェア・コンポーネント

### ハードウェア・コンポーネント

設計者とアーキテクチャ担当者は、複数層環境の各層でハードウェアのサイズ指定と容量計画を行う必要があります。アーキテクチャ担当者は、バランスの取れた設計を行うことが必要です。これは、橋の設計に似ています。橋の設計者は、橋に関する様々なペイロードや構造的要件をすべて考慮する必要があります。橋の強度は、最も強度が弱いコンポーネントによって決まります。したがって、橋は、すべてのコンポーネントがその設計上の限度に同時に達するように、バランスよく設計されています。

ハードウェアには、主に次のコンポーネントが含まれています。

- CPU
- メモリー

- I/O サブシステム
- ネットワーク

**CPU** 1 つ以上の CPU が存在します。CPU の処理能力は、ハンドヘルド・デバイスの単純な CPU から高性能サーバーの CPU まで様々です。その他のハードウェア・コンポーネントのサイズは、通常、システム上の CPU の倍数で指定されます。

**メモリー** データベース・サーバーとアプリケーション・サーバーには、データをキャッシュしたり、時間のかかるディスク・アクセスを避けるために十分な量のメモリーが必要です。

**I/O サブシステム** I/O サブシステムは、クライアント PC のハード・ディスクから高性能なディスク・アレイまで様々です。ディスク・アレイは、1 秒間に数千回の I/O を実行でき、複数の I/O パスとホット・プラグ可能なミラー化ディスクに関する冗長性によって可用性を提供します。

**ネットワーク** システム内のすべてのコンピュータは、モデム回線から高速内部 LAN に至るネットワークに接続しています。ネットワーク仕様に関する主な考慮点は、帯域幅（ボリューム）と待機時間（スピード）です。

**関連項目：** これらのリソースのチューニングについては、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

## ソフトウェア・コンポーネント

コンピュータに共通のハードウェア・コンポーネントがあるように、アプリケーションにも共通の機能コンポーネントがあります。ソフトウェアの開発を機能コンポーネントごとに分割することで、アプリケーションの設計やアーキテクチャが理解しやすくなります。システムの一部のコンポーネントは、アプリケーションの実装を促進したり共通コンポーネントの再開発を回避するために購入した既存のソフトウェアによって実行されます。

ソフトウェア・コンポーネントとハードウェア・コンポーネントの相違は、ハードウェア・コンポーネントが 1 つのタスクのみ実行するのに対して、1 つのソフトウェア・コンポーネントは様々なソフトウェア・コンポーネントの役割を実行できる点にあります。たとえば、ディスク・ドライブはデータの格納や取出しのみを行います、クライアント・プログラムはユーザー・インタフェースを管理し、ビジネス・ロジックを実行できます。

ほとんどのアプリケーションには、次のコンポーネントが含まれています。

- ユーザー・インタフェースの管理
- ビジネス・ロジックの実装
- ユーザー要求とリソース割当ての管理
- データとトランザクションの管理

**ユーザー・インタフェースの管理** これは、アプリケーション・ユーザーに最も認識されやすいコンポーネントです。このコンポーネントは、次の機能を備えています。

- ユーザーが使用するスクリーンの描画
- ユーザー・データの収集とビジネス・ロジックへの転送
- データ入力 of 検証
- アプリケーションの各レベルまたは状態へのナビゲート

**ビジネス・ロジックの実装** このコンポーネントは、アプリケーション機能の中心となる主要なビジネス・ルールを実装します。このコンポーネントでのエラーは、修復に相当なコストを要する場合があります。このコンポーネントは、宣言型アプローチとプロシージャ型アプローチの組合せで実装されます。一意キーと外部キーの定義は、宣言アクティビティの一例です。また、値引計画の実装は、プロシージャ・ベースのロジックの一例です。

このコンポーネントの共通機能は次のとおりです。

- データ・モデルのリレーショナル表構造への移行
- リレーショナル表構造での制約の定義
- ビジネス・ルールを実装するためのプロシージャ型ロジックのコーディング

**ユーザー要求とリソース割当ての管理** このコンポーネントは、すべてのソフトウェアで実装されます。ただし、アプリケーション設計による影響を受ける要求やリソースがある一方で、影響を受けない要求やリソースもあります。

マルチユーザー・アプリケーションでは、ほとんどのユーザー要求によるリソース割当ては、データベース・サーバーまたはオペレーティング・システムによって処理されます。ただし、ユーザー数や使用パターンが不明または急速に増加している大規模アプリケーションの場合、システムのアーキテクチャ担当者は、どのソフトウェア・コンポーネントもオーバーロードになったり不安定にならないように事前に対処する必要があります。

このコンポーネントの共通機能は次のとおりです。

- データベースとの接続管理
- 効率的な SQL の実行（カーソルと SQL の共有）
- クライアントの状態に関する情報の管理
- ハードウェア・リソース間でのユーザー要求のロードの均衡化
- ハードウェア / ソフトウェア・コンポーネントに対する操作上のターゲットの設定
- タスクの非同期実行に対する永続キューイング

**データとトランザクションの管理** このコンポーネントは、主にデータベース・サーバーとオペレーティング・システムに対する機能を備えています。

このコンポーネントの共通機能は次のとおりです。

- ロックとトランザクション・セマンティクスを使用した、データへの同時アクセス
- 索引およびメモリー・キャッシュを使用した、データへの最適化されたアクセス
- ハードウェア障害時のデータ変更の記録
- データに対して定義された規則の適用

## 要件に合った正しいシステム・アーキテクチャの構成

初期のシステム・アーキテクチャの構成作業は、反復的なプロセスです。アーキテクチャ担当者は、予算やスケジュールの制約内でシステム要件を満たす必要があります。インタラクティブ・ユーザーがビジネス取引を行い、データベースの内容に基づいて意思決定を行うことが必要なシステムは、ユーザー要件が主体のアーキテクチャになります。システム上にインタラクティブ・ユーザーがほとんど存在しない場合は、プロセス主体のアーキテクチャになります。

インタラクティブ・ユーザー・アプリケーションの例を次に示します。

- 経理アプリケーションや会計帳簿アプリケーション
- 注文入力システム
- 電子メール・サーバー
- Web ベースの小売販売アプリケーション
- 取引システム

プロセス主体のアプリケーションの例を次に示します。

- 公共料金支払システム
- 不正検出システム
- ダイレクト・メール

多くの場合、プロセス主体のアプリケーションは、ユーザー・インタフェース要素がないため、マルチユーザー・アプリケーションより設計が簡単です。しかし、対象がプロセス指向であるため、大量のデータや様々な成功要因の処理に慣れていないアーキテクチャ担当者は混乱する場合があります。プロセス主体のアプリケーションでは、ユーザーベースのアプリケーションとデータ・ウェアハウスの両方で使用する一連のスキルが必要になります。したがって、このマニュアルでは、インタラクティブ・ユーザー向けのシステム・アーキテクチャ関連について重点的に説明します。

---

**注意：** システム・アーキテクチャの生成は、決定的なプロセスではありません。ビジネス要件、テクノロジーの選択、既存のインフラストラクチャやシステム、予算や労働力などの実際の物理リソースを慎重に考慮する必要があります。

---

次の質問は、システム・アーキテクチャに関する完全なガイドではありませんが、アーキテクチャに対する考え方の参考になります。これらの質問では、ビジネス要件がアーキテクチャ、実装の容易さ、およびシステム全体のパフォーマンスや可用性に対して与える影響を示しています。たとえば、次のような質問があります。

■ システムでサポートするユーザーは何名ですか？

ほとんどのアプリケーションは、次のいずれかのカテゴリに該当します。

- ごく少数のユーザーが使用量の少ないマシンまたは排他的にマシンを使用する場合  
このタイプのアプリケーションでは、通常、ユーザーは1名です。アプリケーション設計の重点は、応答時間を短くし、しかもアプリケーションによる管理を最小限に抑えることで、1名のユーザーの生産性をできるだけ高くすることにあります。このようなアプリケーションのユーザーは、相互に介入することはほとんどなく、リソースの競合も最小限になります。

- 中規模から大規模の社内ユーザーが共有アプリケーションを使用する場合  
このタイプのアプリケーションでは、ユーザー数は社内ですべてのシステムを実際に使用してビジネスを行う従業員数に限定されます。したがって、ユーザー数は予測可能です。ただし、信頼性のあるサービスを配布することはビジネスにとって必須です。ユーザーは共有リソースを使用するため、アプリケーション設計では、大量のシステム・ロード下での応答時間、各セッションで使用されるリソースの増大、および将来的な拡張のための余地に重点を置きます。

- 無数のユーザーがインターネット上に存在する場合  
このタイプのアプリケーションでは、すべてのシステム・コンポーネントがそれぞれの上限を超えないように設計する必要があります。これによって、システムが停止したり不安定になるボトルネックが発生します。このようなアプリケーションでは、複合的なロード・バランシング、ステートレスなアプリケーション・サーバー、および効率的なデータベース接続管理が必要です。さらに、統計およびガバナを使用して、システムのオーバーロードのためにユーザー要求が満たされない場合にユーザーがフィードバックを受け取るようにする必要があります。

■ ユーザーと対話するにはどのような方法がありますか？

単純な Web ブラウザからカスタム・クライアント・プログラムに至る、幅広い選択肢からユーザー・インタフェースを選択できます。

- ユーザーはどこに位置しますか？

ユーザー間の距離は、ネットワーク待機時間に対処するためのアプリケーションの設計方法に影響を与えます。また、ユーザーの位置は、1 日の中でビジーな時間帯、つまりバッチ機能やシステム・メンテナンス機能を実行できない時間帯にも影響を与えます。

- ネットワーク・スピードとは何ですか？

ネットワーク・スピードは、データ量に影響を与え、アプリケーション・サーバーやデータベース・サーバーとのユーザー・インタフェースの対話特性にも影響を与えます。対話主体のユーザー・インタフェースでは、キー・ストロークごとまたはフィールド・レベルの妥当性チェックごとにバックエンド・サーバーと通信できます。対話が少ないインタフェースは、画面ごとに送受信を行うモデルで機能します。通信速度が遅いネットワークでは、対話主体のユーザー・インタフェースを使用しても満足なデータ入力スピードは得られません。

- ユーザーがアクセスするデータ量、および読取り専用のデータが占める割合はどの位ですか？

オンラインでの問合せデータ量は、表や索引の設計からプレゼンテーション・レイヤーの設計に至るあらゆる局面に影響を与えます。データベースのサイズによってユーザーの応答時間が影響されないように設計する必要があります。アプリケーションが主として読取り専用の場合、レプリケーションおよびアプリケーション・サーバーのローカル・キャッシュへのデータ配分は、有効な選択肢になります。また、この選択によって、主なトランザクション・サーバーでのワークロードが削減されます。

- ユーザーの応答時間に関する要件とは何ですか？

ユーザー・タイプに対する考慮が重要です。正しい意思決定を行うために正確な情報が必要な役員がユーザーの場合、ユーザーの応答時間は重要です。データ入力を行うユーザーなど他のタイプのユーザーの場合、高レベルのパフォーマンスは必要ありません。

- ユーザーは 1 日 24 時間のサービスを望んでいますか？

取引が 1 日 24 時間行われている今日のインターネット・アプリケーションの場合、24 時間稼働は必須です。ただし、1 つのタイム・ゾーンでのみ稼働している企業システムの場合は、終業後にシステムを停止できます。終業後のシステム停止時間を利用して、バッチ処理やシステム管理を実行できます。このような処理は、使用可能な全システムを稼働させずに実行すると、さらに経済的です。

- すべての変更はリアルタイムで行う必要がありますか？

ユーザーの応答時間内にトランザクションを実行する必要があるか、またはキューに入れて非同期で実行できるかを判断することが重要です。

次の質問は二次的な質問です。これらは設計にも関連していますが、予算や実装の容易さとの関連が強い質問です。たとえば、次のような質問があります。

- データベースの規模はどの位ですか？  
データベースの規模は、データベース・サーバー・マシンのサイズ指定に影響を与えます。大規模データベースを持つシステムでは、ワークロードから判断されるマシンよりも大型のマシンを用意することが必要になる場合があります。これは、大規模データベースに伴う管理オーバーヘッドは、主としてデータベース・サイズに関連しているためです。表および索引が大きくなると、制限時間内に表を再構成したり索引を作成する必要があるため、必要な CPU の数もそれに比例して増加します。
- ビジネス・トランザクションに必要なスループットは？
- 可用性に関する要件は何ですか？
- このアプリケーションを作成して管理するためのスキルはありますか？
- 予算上の制約から妥協が求められるのは何ですか？

## アプリケーション設計の原理

この項では、アプリケーション作成時に設計に関して決定する内容を説明します。

### アプリケーション設計の簡潔さ

アプリケーションには、他の製品と同様に設計と開発が必要です。適切に設計された構造、マシンおよびツールは、通常、信頼性があり、使用方法やメンテナンスが容易で概念的にも簡潔です。多くの場合、設計が適切に見える場合は、実際に適切に設計されています。アプリケーションの作成では、この原理を常に覚えておいてください。

設計に関して次の点を考慮してください。

- 表の設計が複雑で誰も完全に理解できない場合、その表の設計は不適切と言えます。
- SQL 文が非常に長く、オブティマイザがリアルタイムに効率よく最適化できない場合は、その SQL 文、基礎となるトランザクションまたは表の設計は不適切と言えます。
- 表に索引があり、同じ列を繰返し参照する場合は、索引の設計が不適切と言えます。
- オンライン・ユーザーに対する迅速な応答で、適切な資格がないまま問合せが発行される場合は、ユーザー・インタフェースまたはトランザクションの設計が不適切と言えます。
- ソフトウェアの多くの層で、データベースのコールがアプリケーション・ロジックに沿っていない場合は、ソフトウェアの開発方法が不適切と言えます。

## データ・モデル化

データ・モデル化は、リレーショナル・アプリケーションを適切に設計するために重要です。データ・モデル化は、ビジネス慣習を即座に表現する方法で行います。正しいデータ・モデル化については、様々な議論があります。重要な点は、最も頻繁に行われるビジネス・トランザクションによって影響を受けるエンティティをモデル化することです。モデル化フェーズでは、あまり重要でないデータ要素のモデル化に時間を取られると、開発リード・タイムが延長される結果になります。モデル化ツールを使用すると、スキーマ定義を簡単に生成できます。短期間でプロトタイプを作成する必要がある場合に便利です。

## 表および索引の設計

表の設計では、主要なトランザクションの柔軟性とパフォーマンスの間でバランスを取ることが重要です。データベースの柔軟性を保持しながら予想外のワークロードに対処する点で、表の設計はデータ・モデルとよく似ており、最低でも第3正規形に正規化しておく必要があります。ただし、ユーザーが要求する特定のトランザクションは、パフォーマンス向上のために非正規化することができます。

この技法の例には、事前に結合した表の格納、導出列の追加および集計値があります。Oracle には、クラスタ化による集計データと事前結合データの格納、およびマテリアライズド・ビュー機能に関する多くのオプションが用意されています。これらの機能によって、より簡潔な表の設計を最初から採用できます。

ここでも、ビジネス上重要な表に重点的にリソースを使用すると、パフォーマンスが向上します。重要でない表の場合は、アプリケーション開発を迅速に進めるためにも、設計にあまり時間をかけないことも可能です。ただし、プロトタイプおよびテストの結果、重要でない表でパフォーマンスの問題が発生する場合は、ただちに設計を修正する必要があります。

索引の設計も主として、アプリケーション設計者が生成した SQL に基づく反復的なプロセスです。ただし、主キー制約を適用した索引、および個人名など既知のアクセス・パターン上の索引を作成することから開始することも可能です。アプリケーションの開発が進み、実サイズのデータ上でテストを実行すると、索引を改善することによって、特定の問合せのパフォーマンスを改善する必要があります。新規索引の作成時に、索引の設計に関して考慮する点を次に示します。

- [索引への列の追加または索引構成表の使用](#)
- [異なる索引タイプの使用](#)
- [索引のコストに関する考察](#)
- [索引内のシリアライズ化](#)
- [索引内の列の順序付け](#)

## 索引への列の追加または索引構成表の使用

問合せをスピードアップする簡単な方法の1つは、実行計画から表アクセスを排除して論理 I/O の数を削減することです。この方法は、問合せによって参照されるすべての列を索引に追加して実行します。この列は選択リスト列、および結合またはソートが必要な列です。この方法は、時間がかかる I/O を削減して、オンライン・アプリケーションの応答時間を短縮する場合に特に役立ちます。その効果は、適切なサイズのデータを使用して最初にアプリケーションをテストしたときに明らかになります。

この技法の積極的な使用法は、索引構成表 (IOT) を作成することです。ただし、IOT のリーフ・サイズの増加が I/O の削減を妨げないように注意する必要があります。

## 異なる索引タイプの使用

いくつかのタイプの索引を使用でき、それぞれ状況に応じて利点があります。次に、各タイプの索引に関連したパフォーマンスの問題について説明します。

**B ツリー索引** 標準的な索引タイプです。主キー索引および選択的な索引に最も適しています。B ツリー索引を連結索引として使用すると、索引列順にソートされたデータを取り出すことができます。

**ビットマップ索引** この索引は、カーディナリティが低いデータに適しています。この索引は、圧縮技法によって最小限の I/O で多数の ROWID を生成できます。選択列以外の列でビットマップ索引を組み合わせることによって、最小限の I/O と多数の ROWID で AND 操作と OR 操作を効率よく実行できます。ビットマップ索引は索引内で問合せを満たすことができるため、COUNT() を使用した問合せで特に効果的です。

**ファンクション・ベース索引** この索引によって、ベース・データ上の関数から導出された値に B ツリーからアクセスできます。ファンクション・ベース索引には NULL の使用に制限があり、コストベースのオプティマイザを使用可能にしておく必要があります。

ファンクション・ベース索引は、複合列に対する問合せを行って、導出された結果を生成したり、データがデータベースに格納されている方法によって制限を受けないようにする場合に特に役立ちます。このような問合せの例として、(販売価格 - 値引) × 数量の計算から導出された値を超える受注の明細項目の問合せがあります。明細項目は表内の列です。別の例では、UPPER 関数をデータに適用して大 / 小文字を区別しない検索ができます。

**関連項目：**『Oracle9i 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』

**パーティション索引** グローバル索引のパーティション化によって、パーティション・プランニングが索引アクセス内で発生し、I/O を削減できます。レンジ・パーティション化またはリスト・パーティション化を適切に定義すると、正しい索引パーティションを高速で索引スキャンするため、問合せ時間がかなり短縮できます。

**逆キー索引** この索引は、挿入アプリケーションでの索引のホット・スポットを除去するように設計されています。この索引は、挿入パフォーマンスに優れていますが、索引レンジ・スキャンには使用できません。

### 索引のコストに関する考察

索引構造の作成とメンテナンスにはコストがかかり、ディスク領域、CPU および I/O 容量などのリソースを消費します。設計者は、索引のメンテナンスにかかるコストより、索引の使用による利点が上回るように設計する必要があります。

索引のメンテナンスにかかるコストは、次の簡単な見積りを使用してください。索引キーの INSERT、DELETE または UPDATE でメンテナンスされる各索引には、実際の表に対する DML 操作で使用する 3 倍のリソースが必要です。つまり、3 つの索引がある表に INSERT 操作を行うと、索引がない表に INSERT 操作を行う場合に比較して約 10 倍遅くなります。DML、特に INSERT 主体のアプリケーションの場合は、問合せと INSERT 操作のパフォーマンスとの間のバランスを取る必要があるため、索引の設計は十分に検討する必要があります。

### 索引内のシリアル化

順序またはタイムスタンプを使用して索引付きキー値を生成すると、データベースのホット・スポット問題が生じて、応答時間やスループットに影響を与える場合があります。これは、通常、索引の適切な増加の原因となる単調なキーの増加による結果です。この問題を回避するには、索引の全範囲にわたって挿入するキーを生成します。これによって、スケラブルで領域を効率よく使用するバランスの取れた索引になります。この索引は、逆キー索引を使用するか、接頭辞および順序値へのサイクル順序を使用して生成します。

### 索引内の列の順序付け

設計者は、索引作成に関する規則を柔軟に定義する必要があります。状況に応じて、次のいずれかの方法で索引内のキーに順序を付けます。

1. 選択頻度の高い列から順序を付けます。これによって、最小限の I/O で必要な ROWID に最も速くアクセスできるため、通常はこの方法を使用します。この技法は、主として主キーや選択頻度の高いレンジ・スキャンに使用します。
2. 列に順序を付け、データをクラスタ化またはソートして I/O を削減します。大規模なレンジ・スキャンでの I/O スキャンは、通常、選択頻度の低い順に列を順序を付けるか、取り出す順にデータをソートして削減できます。

## ビューの使用

ビューを使用すると、アプリケーションの設計が高速化かつ簡素化されます。簡単なビュー定義によって、データの取出し、表示、収集および格納処理を優先するプログラムをデータ・モデルの複雑さから解放できます。

ただし、ビューは、正しいプログラミング・インタフェースを提供する一方で、部分的に最適化された、リソース集約的な問合せの原因になる場合があります。ビューの最も不適切な使用例は、ビューが他の複数のビューを参照し、その複数のビューが問合せ内で結合されている場合です。多くの場合、開発者はビューを使用せずに表から直接問合せを満たすことができます。ビューが持つ本来の性質によって、通常は、オプティマイザによる最適な実行計画の生成が困難となります。

## SQL の実行効率

システム開発の設計およびアーキテクチャ・フェーズでは、アプリケーション開発者は SQL の実行効率を理解することが重要です。このため、開発環境では次の特性をサポートしている必要があります。

- 適切なデータベース接続管理

データベースへの接続は、かなりコストがかかる拡張性のない操作です。このため、データベースへの同時接続数はできるだけ少なくする必要があります。アプリケーションの初期化時には、1 名のユーザーが接続している単純なシステムが理想的です。しかし、Web ベースまたは複数層アプリケーションでは、複数のアプリケーション・サーバーが使用され、ユーザーへのデータベース接続が多重化しているため、接続数を少なくするのは困難です。このようなタイプのアプリケーションでは、データベース接続をプールし、ユーザー要求ごとに接続が再確立されないように設計する必要があります。

- 適切なカーソルの使用方法と管理

ユーザー接続のメンテナンスと同様に、システムでの解析アクティビティを最小にすることが重要です。解析とは SQL 文を解析し、SQL 文の実行計画を作成するプロセスです。このプロセスには、構文検査、セキュリティ検査、実行計画の生成、共有構造の共有プールへのロードなど、多くのフェーズがあります。解析操作には次の 2 種類があります。

- **ハード解析。**最初に SQL 文が発行されたときは、共有プール内に一致するものはありません。ハード解析は、解析に含まれているすべての操作を実行するため、最もリソース集約的であり、拡張性がありません。
- **ソフト解析。**最初に SQL 文が発行されたとき、共有プール内には一致するものがあります。別のユーザーが以前に実行した結果が一致する場合があります。SQL 文は共有されるため、パフォーマンスが向上します。ただし、ソフト解析は、システム・リソースを消費する構文検査やセキュリティ検査が必要であるため理想的とは言えません。

解析はできるだけ最小限にする必要があるため、アプリケーション開発者は、アプリケーションで SQL 文を 1 回解析し、その SQL 文を繰返し実行するように設計してください。これは、カーソルを使用して行います。経験がある SQL プログラマであれば、カーソルのオープンと再実行の概念を理解しています。

アプリケーション開発者は、SQL 文が共有プール内で共有されていることも確認する必要があります。これを行うには、実行ごとに変化する問合せの部分を変数として表現します。これを行わないと、SQL 文は 1 回解析され、以降は他のユーザーに再使用されることはありません。SQL の共有を確実にするには、変数を使用し、SQL 文で文字列リテラルは使用しないでください。たとえば、次のようにします。

文字列リテラルがある文は次のとおりです。

```
SELECT * FROM emp
WHERE ename
LIKE 'KING';
```

変数がある文は次のとおりです。

```
SELECT * FROM emp
WHERE ename
LIKE :1;
```

次の例は、単純な OLTP アプリケーションでのテスト結果です。

Test	#Users Supported
No Parsing all statements	270
Soft Parsing all statements	150
Hard Parsing all statements	60
Re-Connecting for each Transaction	30

このテストは、4 台の CPU マシンで実行されました。システム上の CPU の数が増えると、差異も大きくなります。

## アプリケーションの実装

開発環境とプログラミング言語の選択は、開発チームのスキルと、アプリケーションの指定時に決定したアーキテクチャに関連しています。しかし、アプリケーションをスケーラブルで高パフォーマンスにできる、いくつかの簡単なパフォーマンス管理規則があります。

1. ソフトウェア・コンポーネントに適した開発環境を選択し、その環境によってパフォーマンスに関する設計が限定されないようにしてください。設計が限定される場合は、選択した言語または環境が不適切と言えます。

- ユーザー・インタフェース

プログラミング・モデルには、HTML 生成からウィンドウ・システムの直接コールまで様々なモデルがあります。開発方法では、ユーザー・インタフェース・コードの応答時間に重点を置く必要があります。HTML または Java をネットワークで送信する場合は、ネットワーク・ボリュームや対話を最小限にしてください。

- ビジネス・ロジック

Java や PL/SQL などの解析言語は、ビジネス・ロジックのコード化に理想的です。これらの言語は完全に移植可能であるため、ロジックを比較的簡単にアップグレードできます。いずれの言語も構文が豊富で、コードは読んだり解析するのが簡単です。ビジネス・ロジックで複雑な数学関数が必要な場合は、コンパイル済みバイナリ言語が必要になる場合があります。ビジネス・ロジック・コードは、クライアント・マシン、アプリケーション・サーバーおよびデータベース・サーバーに配置できます。ただし、アプリケーション・サーバーに配置するのが最も一般的です。

- ユーザー要求とリソース割当て

プログラミング言語によって影響を受けることはほとんどありませんが、データベース接続やカーソル管理をマスクするツールおよび第四代言語では、非効率的なメカニズムが使用される場合があります。このようなツールや環境を評価するときは、データベース接続モデルおよびカーソルやバインド変数の使用方法についてチェックしてください。

- データ管理とトランザクション

プログラミング言語による影響はほとんどありません。

2. ソフトウェア・コンポーネントを実装するときは、そのコンポーネントの機能を実装し、他のコンポーネントに対応付けられた機能は実装しないでください。他のコンポーネントの機能を実装すると、設計や実装が部分的に最適化されてしまいます。これはすべてのコンポーネントに該当します。
3. 機能のギャップを放置したり、検討中のソフトウェア・コンポーネントを設計、実装またはテストで使用しないでください。多くの場合、ギャップは、アプリケーションを展開するか、実際のボリュームでテストするまで検出されません。このギャップは、通常、アーキテクチャまたは当初のシステム仕様が不適切であることを示します。データ・アーカイブ / パージ・モジュールは、初期のシステム設計、作成および実装時には無視されがちです。

4. プロシージャ型ロジックを実装するときは、C、Java、PL/SQL などの手続き型言語で実装してください。データ・アクセス（問合せ）またはデータ変更（DML）を実装するときは、SQL を使用します。これは、プロシージャ・コードとデータ・アクセス（非プロシージャ型 SQL）コードが混合しているビジネス・ロジック・モジュールに固有の規則です。プロシージャ型ロジックを SQL アクセスに適用することはお薦めできません。これは、リソース集約的で不適切な SQL になる恐れがあります。DECODE ケース文を持つ SQL 文は、多くの場合、最適化が必要です。これは、この文には、大量の OR 述語または UNION や MINUS などの集合演算子が含まれているためです。
5. 頻繁にアクセスし、変更がほとんどなく、繰り返し取り出すのにコストがかかるデータはキャッシュします。ただし、キャッシュ・メカニズムは使い易くし、前述の方法でデータにアクセスするよりもコストが低くなることを確認してください。この方法は、リモート・データ・ストアまたはコストがかかるデータ・ストアから繰り返しデータを取り出すよりも、頻繁に使用するデータ値をキャッシュしてローカルに格納した方がコストが低くなるすべてのモジュールに適用できます。

ローカルにキャッシュするデータの例を次に示します。

- 本日の日付。SELECT SYSDATE FROM DUAL は、データベースにかかるワークロードの 60% 以上を占める場合があります。
- 現行ユーザー名。
- 税率、値引率、位置情報など、繰り返し使用するアプリケーション変数および定数。
- ローカルでのデータ・キャッシュは、さらにアプリケーション・サーバーの中間層へのローカル・データ・キャッシュの構成に発展させることができます。この構成によって、中央のデータベース・サーバーのロードが減ります。ただし、ローカル・キャッシュを構成するときは、キャッシュの複雑さによってパフォーマンスに影響がでないように注意してください。
- ローカル順序の生成。

キャッシュを使用する場合は、設計との密接な関係を考慮してください。たとえば、ユーザーが午前 0 時に接続していて、日付がキャッシュされている場合、その日付値は午前 0 時を過ぎると無効になります。

6. コンポーネント間のインタフェースを最適化し、すべてのコンポーネントが最もスケーラブルな構成で使用されていることを確認してください。この規則では最小限の説明が必要で、すべてのモジュールとそのインタフェースに適用されます。
7. 外部キー参照を使用します。アプリケーションから参照整合性を適用するのはコストがかかります。親から子の列値を選択し、その存在を確認することで、外部キー参照をメンテナンスできます。Oracle が提供する外部キー制約規定（SQL は使用しません）は高速で、しかも簡単に宣言でき、ネットワークの通信量は発生しません。

## アプリケーション開発の傾向

今日のアプリケーション開発には、2つの大きな特長があります。1つは、コンパイル済み C または C++ アプリケーションにかわって Java の使用が増えたこと、もう 1 つは、スキーマの設計に影響を与えるオブジェクト指向の設計が増えていることです。

Java はコードの移植性に優れ、プログラマに優れた可用性を提供します。ただし、Java にはパフォーマンスと密接に関係した事項が多数あります。Java はインタプリタ型言語であるため、C 言語などのコンパイル型言語に比べてロジックの実行速度が遅くなります。その結果、クライアント・マシンでのリソース使用率が増加します。このため、強力な CPU をクライアント・マシンや中間層マシンに設置する必要があり、プログラマは効率的なコードを作成する必要があります。

Java はオブジェクト指向言語であるため、データ・アクセスはビジネス・ロジックを実行しないクラスに隔離することをお勧めします。この結果、プログラマは、使用しているデータ・アクセス方法の効率に関する知識がなくてもメソッドを呼び出すことができます。この方法では、データベース・アクセスが最小限になり、データベースへのインタフェースには最も簡潔で単純なインタフェースが使用されます。

このタイプのソフトウェア設計では、常にすべての WHERE 述語が問合せに効率的に組み込まれるとは限らず、行のフィルタ処理は Java プログラムで実行されます。これは、かなり非効率的です。さらに、DML 操作、特に INSERT 操作では、1 つの INSERT 操作が実行されると配列インタフェースは使用できなくなります。プロシージャ・コールでも非効率的になる場合があります。データベースとのデータのやりとりでは、実際のデータベース・コールよりも多くのリソースが使用されます。

一般的に、最適なトランザクション設計の実現には、データ・アクセス・コールをビジネス・ロジックの次に配置するのが最適です。

プログラミング・レベルでのオブジェクト指向の採用は、オブジェクト指向のデータベースを Oracle サーバー内に作成することにつながります。このことは、オブジェクト構造を BLOB に格納して、データベースを索引付きカード・ファイルとして効率的に使用する場合があります。Oracle オブジェクト・リレーショナル機能を使用する場合に至る、多くの場合で明らかです。

オブジェクト指向のアプローチをスキーマ設計に採用する場合は、リレーショナル格納モデルの柔軟性が失われないように注意してください。多くの場合、スキーマ設計に対するオブジェクト指向のアプローチによって、データ構造にかなりの非正規化が生じ、相当なメンテナンスとオブジェクトに関連付けられた REF ポインタが必要になります。多くの場合、このような設計は、リレーショナル格納方法に置き換わる前の階層データベースやネットワーク・データベースの設計に戻ることにになります。

要約すると、データベースにデータを長期間格納し、同じスキーマでの非定型問合せまたはアプリケーション開発を計画する場合は、リレーショナル格納方法によって、最適なパフォーマンスと柔軟性が得られます。

## ワークロードのテスト、モデル化および実装

### データのサイズ設定

不適切なサンプル・セットで作業を行うと、可変長データの処理時にサイズの見積りを誤る場合があります。また、データ量の増加に従ってキー長も大幅に長くなり、列サイズの見積りの変更が生じます。

システムが稼働し始めると、データベース規模の拡大、特に索引の増加は予想が困難となります。表は時間とともに拡大し、索引は、キー生成、挿入パターンおよび行の削除に関するアプリケーションの個々の動作によって影響を受けます。最も不適切なケースは、昇順キーを使用して行を挿入してから、一部の行を残してほとんどの行を表の左側から削除した場合です。これによって、ギャップと不要な領域が残ります。索引をこのように使用している場合は、オンラインの索引再作成機能の使用方法を調べてください。

優れた DBA は、オブジェクトごとに領域割当てを監視し、拡大して制御できなくなる可能性があるオブジェクトを監視しています。アプリケーションを十分理解することで、急速に拡大したり予測不能になる可能性があるオブジェクトを発見することができます。これは、あらゆるシステムにおいて、パフォーマンスと可用性計画の両方に必要なことです。本番データベースを実装する場合は、インタラクティブ・ユーザーがアプリケーションを使用しているときに領域管理が最小限になるように設計する必要があります。この設計方法は、データ、一時およびロールバック用のすべてのセグメントに適用されます。

### ワークロードの見積り

容量計画やテスト用にワークロードを見積ることは、しばしば魔術にたとえられます。見積りに関連する変数の数を見ても、正確な見積りがほとんど不可能であることが容易に理解できます。それでも、設計者は、マシンと CPU の数、メモリーおよびディスク・ドライブの数を指定し、最終的にはアプリケーションを展開する必要があります。サイズ設定で使用する技法はいくつかあり、それぞれにメリットがあります。サイズを設定するときは、少なくとも 2 つの方法を使用して意思決定プロセスを検証し、サポート用のドキュメントを準備することをお勧めします。

### 類似システムからの推定

これは経験的なアプローチで、類似の特性を持ち、パフォーマンスを把握している既存のシステムを基礎システムとして使用します。既知の相違点に基づいて、このシステムの仕様をサイズの担当者が変更します。このアプローチでは、既存のシステムと関連する部分は参考になりますが、関連しない部分の処理ではほとんど参考になりません。

このアプローチは、大規模な建物、船舶、橋梁、石油掘削装置などのエンジニアリング・プロジェクトのコストを見積るときに、ほとんどすべてのエンジニアリング分野で使用されています。参照システムがサイズの点で計画しているシステムと大きな差異がある場合は、コンポーネントの一部が設計の上限を超えてしまう可能性があります。

## ベンチマーク

ベンチマーク・プロセスは、リソースと時間を多く消費し、必ずしも正確な結果が得られるわけではありません。開発初期またはプロトタイプ・フォームの段階でアプリケーションをベンチマークによってシミュレーションを行うと、実際の本番システムとは類似しない部分を測定する危険があります。それでも、長年にわたりデータベース開発組織で顧客アプリケーションのベンチマークを行ってきた結果、ベンチマーク・アプリケーションと実際の本番システムとの間に明確な相関関係を認めることができます。この差異は、主として開発プロセスに導入されたアプリケーションのいくつかの非効率な点によります。

しかし、ベンチマークは、許容できるレベルの精度でシステムのサイズを設定するために積極的に使用されています。特に、ベンチマークは、実際の I/O 要求数の判断やシステムが完全にロードされたときのリカバリ処理のテストに効果があります。

ベンチマークはその性質から、システム的全コンポーネントに対してそれぞれの上限までストレスをかけます。ベンチマーク中に全コンポーネントにストレスをかけ、アプリケーション設計と実装でのエラーをすべて識別します。また、ベンチマークでは、データベース、オペレーティング・システムおよびハードウェア・コンポーネントもテストされます。ほとんどのベンチマークは一度に実行されるため、システム・コンポーネントが失敗すると問題が発生する場合があります。ベンチマークはストレスが大きいアクティビティであるため、ベンチマーク作業から十分な結果を得るにはかなりの経験が必要です。

## アプリケーションのモデル化

アプリケーションのモデル化は、複雑な数学的モデルから典型的な単純計算を実行するモデルまで幅広く適用できます。いずれの方法にもメリットがあります。1 つは非常に正確に計算を行い、もう 1 つは大まかな見積りを計算します。両方の方法のデメリットは、実装のエラーや効率の悪さを識別できないことです。

見積りやサイズ設定は不正確なプロセスです。しかし、プロセスを精査すると的確な見積りを行うことができます。見積りプロセス全体では、誤った SQL の記述、不適切な索引の設計、または不適切なカーソル管理によるアプリケーションの効率の悪さは考慮されません。優れたエンジニアは、サイズを設定するときアプリケーションの効率の悪さも考慮します。また、効率の悪さを検出して、見積りを現実的なパフォーマンスにします。アプリケーションの効率の悪さを検出するプロセスの詳細は、「Oracle のパフォーマンス改善方法」を参照してください。

## 設計のテスト、デバッグおよび検証

テスト・プロセスは、主に機能とスタビリティのテストで構成されています。プロセスの途中で、パフォーマンス・テストが実行されます。

アプリケーションのパフォーマンス・テストを実行するときの簡単な規則を次に説明します。この情報は、アプリケーションの稼働後の本番アプリケーションと容量計画プロセスで重要になります。

- 現実的なデータ量とデータ配分によるテスト

すべてのテストは、完全に入力された表を使用して行う必要があります。テスト用データベースには、データ量や表のカーディナリティも含めて、本番システムと同様にデータを格納します。本番と同様の索引をすべて作成し、スキーマ統計を正確に入力します。

- 正しいオブティマイザ・モードの使用

すべてのテストは、本番で使用するオブティマイザ・モードで実行する必要があります。オラクル社ではコストベースのオブティマイザを重点的に研究開発してきました。したがって、コストベースのオブティマイザの使用をお勧めします。

- シングル・ユーザーのパフォーマンスのテスト

アイドル状態または使用量が少ないシステムでのシングル・ユーザーのパフォーマンスが許容範囲であるかをテストします。シングル・ユーザーのパフォーマンスが理想的な条件下でも許容範囲に達しない場合、リソースを共有するマルチ・ユーザーの状況下では満足なパフォーマンスは得られません。

- 全 SQL 文の計画の取得と文書化

各 SQL 文の実行計画を取得します。一部の SQL 文メトリックは、文を最低 1 回実行して取得する必要があります。このプロセスを使用して、オブティマイザが適切な実行計画を取得したことを検証し、SQL 文の関連コストが CPU タイムと物理 I/O 数の点から把握されていることを検証します。このプロセスは、チューニングやパフォーマンス作業が将来必要になる、使用量の多いトランザクションを識別するのに役立ちます。

- マルチユーザーのテスト

ユーザーのワークロードやプロファイルは完全に定量化されていないため、このプロセスを正確に実行するのは困難です。しかし、DML 文を実行するトランザクションをテストして、ロックの競合やシリアライズ化の問題がないことは確認する必要があります。

- 正しいハードウェア構成でのテスト

できるだけ本番システムに近い構成でテストすることが重要です。これは、ネットワーク待機時間、I/O サブシステム帯域幅、およびプロセッサのタイプとスピードについてテストする場合に特に重要です。このテストを正確に行わないと、潜在的なパフォーマンスの問題を正しく分析できません。

- 安定した状態でのパフォーマンスの測定

ベンチマークを行うときは、安定した状態でパフォーマンスを測定することが重要です。各ベンチマークの実行には開始フェーズが必要です。このフェーズでは、ユーザーがアプリケーションに接続し、アプリケーションでの作業を段階的に開始します。このプロセスによって、安定した状態になる前に、頻繁にキャッシュされるデータをキャッシュに初期化し、解析などの単純な実行操作を完了することができます。同様に、ベンチマークの実行後には終了フェーズが必要です。このフェーズでは、リソースをシステムから解放し、ユーザーは作業を終了して接続を切断します。

## 新しいアプリケーションの配置

この項では、アプリケーションの配置に関する設計上の配慮について説明します。

### ロールアウトの方法

新規のアプリケーションをロールアウトするときは、次の2つの方法が一般的です。

- ビッグ・バン・アプローチ全ユーザーが新しいシステムに一度に移行します。
- トリクル・アプローチユーザーは既存のシステムから新しいシステムに徐々に移行します。

いずれの方法にもメリットとデメリットがあります。ビッグ・バン・アプローチは、必要なスケールでアプリケーションを適切にテストする必要がありますが、単純にデータを切り替えるため、旧システムのデータ変換と同期は最小限で済みます。トリクル・アプローチでは、ワークロードの増加に伴う拡張性の問題をデバッグできますが、推移に沿って旧システムとの間でデータの移行を行う必要があります。

いずれの方法も、推移が発生したときシステムが停止する危険があるため、1つのアプローチのみを推奨することは困難です。トリクル・アプローチでは、実際のユーザーが新しいアプリケーションに移行するとそのユーザーをプロファイルし、移行したユーザーについてのみシステムを再構成できます。このアプローチは、採用した初期の段階では担当者の作業に影響を与えますが、サポート・サービスによるロードは軽減されます。これは、スケジュール外の停止が発生しても、影響を受けるユーザー数の割合は小さいことを意味します。

新規アプリケーションのロールアウト方法は、ビジネスごとに判断してください。使用するアプローチには、それぞれ固有の問題やストレスがあります。テストを重ねてそのテストから得られた知識が増えるほど、適切なロールアウト方法を判断できるようになります。

## パフォーマンス・チェックリスト

ロールアウト・プロセスを支援するため、タスク・リストを作成してください。これによって、正常に実行された場合は、本番でのパフォーマンスを改善する機会が増加します。また、問題が発生した場合は、アプリケーションをすばやくデバッグできます。次に例を示します。

1. 本番データベース用の制御ファイルを作成するときは、ロールアウトで予測される値以上の値を MAXINSTANCES、MAXDATAFILES、MAXLOGFILES、MAXLOGMEMBERS および MAXLOGHISTORY に設定して、データベースの拡張に備えます。これによって、ディスク領域の使用量が増え制御ファイルも大きくなりますが、緊急時に拡張が必要になったときに時間を節約できます。
2. ブロック・サイズとオプティマイザ・モードを、アプリケーションの開発時に使用した値に設定します。代表的なデータ量でテストを完了し、現行の SQL 実行計画が正しい場合は、スキーマ統計を開発 / テスト環境から本番データベースにエクスポートします。
3. 最小数の初期化パラメータを設定します。これらのパラメータは、SGA 内の様々なキャッシュのサイズを設定するための重要なパラメータです。アーカイブ・ダンプ先の動作を指定する追加パラメータを、バックアップ用とデバッグ用に設定する必要があります。他のパラメータは、基本的にデフォルトのままにしてください。さらにチューニングを行う必要がある場合は、システムのロード時に表示されます。

**関連項目：** 初期インスタンス構成での最小数のパラメータ設定については、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

4. データベース・オブジェクトの記憶領域オプションを設定して、ブロック競合の管理に備えます。INSERT/UPDATE/DELETE 操作が頻繁に発生する表と索引は、自動セグメント領域管理を使用するか、複数の空きリストと INITRANS の設定値を増やすことによって作成する必要があります。ロールバック・セグメントの競合を回避するには、自動 UNDO 管理を使用するか、複数のロールバック・セグメントを作成して必要なユーザー数をサポートする必要があります。

**関連項目：** 自動 UNDO 管理の使用方法、および自動セグメント領域管理による空き領域の管理の詳細は、『Oracle9i データベース管理者ガイド』を参照してください。

5. すべての SQL 文が最適であることを検証し、そのリソース使用率を把握します。
6. データベースに接続しているミドルウェアとプログラムの接続管理が効率的であること、およびログイン / ログオフが繰り返されていないことを確認します。
7. SQL 文がカーソルを効率的に使用していることを確認します。各 SQL 文は 1 回解析された後、繰り返し実行されます。これが正常に実行されない典型的な原因は、バインド変数が適切に使用されていないこと、および WHERE 句の述語が文字列リテラルとして送

信されていることです。プリコンパイラを使用してアプリケーションを開発している場合は、アプリケーションをプリコンパイルする前に、MAXOPENCURSORS、HOLD\_CURSOR および RELEASE\_CURSOR の各パラメータがデフォルト値から再設定されていることを確認します。

8. すべてのスキーマ・オブジェクトが開発環境から本番データベースに正常に移行していることを確認します。これには、表、索引、順序、トリガー、パッケージ、プロシージャ、ファンクション、Java オブジェクト、シノニム、権限付与およびビューが含まれます。テスト中に変更を行った場合は、本番システムでも変更が行われていることを確認します。
9. システムを展開した直後に、データベースとオペレーティング・システムの統計用ベースライン・セットを作成します。これを行うには、Oracle Enterprise Manager または Statspack を使用します。この最初の統計セットによって、設計プロセスとロールアウト・プロセスで行った予測を検証または訂正します。
10. 最初のボトルネック（常に 1 つです）を予測しながら開始し、Oracle のパフォーマンス改善方法に従ってパフォーマンスを改善します。



---

## アプリケーションのパフォーマンスの監視と改善

この章は、次の項で構成されています。

- [統計情報の重要性](#)
- [Oracle のパフォーマンス改善方法](#)

## 統計情報の重要性

問題に対処する前に、可能な限り統計情報を収集してアプリケーションの全体像を把握します。システム全体を把握するのは容易ではありません。しかし、データが収集されてアプリケーションに埋め込まれていれば、このプロセスはかなり容易になります。

医師が患者から症状を聞き出すように、可能な限り初期データを収集し、その統計情報から問題のアウトラインを明確にします。パフォーマンス分析プロセスの早い段階で症状に対処すると、通常は誤った分析になり、後で時間を浪費することになります。これは、最初の診察で胸の痛みを訴える患者に対して、医師が心臓の手術を指示することが非常に危険であるのと同じです。

### オペレーティング・システム統計情報

オペレーティング・システム統計情報では、システムの主要なハードウェア・コンポーネントの使用率やパフォーマンス、およびオペレーティング・システム自体のパフォーマンスに関する情報が提供されます。この情報は、CPU サイクルや物理メモリーなどのリソースが消耗する可能性を検出したり、ディスク・ドライブなどの周辺機器のパフォーマンスの低下を検出するために重要です。

オペレーティング・システム統計情報は、ハードウェアやオペレーティング・システムの稼働状態を示すのみです。システムのパフォーマンスを分析する担当者の多くは、ハードウェアの増設によってハードウェア資源の不足に対処します。これは、オペレーティング・システム統計情報で示される一連の症状に対する対処療法です。医師が診断を下すとき患者の体温、脈拍および身体的痛みを参照するのと同様に、オペレーティング・システム統計情報は最も適切な診断ツールと考えてください。ボトルネックを識別するには、パフォーマンス分析の対象となっているシステム内の全サーバーのオペレーティング・システム統計情報を収集してください。

オペレーティング・システム統計情報は次のとおりです。

- [CPU 統計情報](#)
- [仮想メモリー統計情報](#)
- [ディスク統計情報](#)
- [ネットワーク統計情報](#)

**CPU 統計情報** CPU の使用率は、チューニング・プロセス中で最も重要なオペレーティング・システム統計情報です。システム全体の CPU 使用率およびマルチ・プロセッサ環境内の各 CPU の CPU 使用率を取得してください。各 CPU の使用率によって、シングル・スレッドと拡張性の問題を検出できます。

ほとんどのオペレーティング・システムでは、CPU 使用情報を、ユーザーの領域またはモードでの経過時間か、カーネルの領域またはモードでの経過時間としてレポートしています。この追加統計情報を使用すると、CPU 上で実際に何が実行されているかをより詳細に分析できます。

Oracle データ・サーバー・システムでは通常、1 つのアプリケーションのみ実行され、サーバーはデータベース・アクティビティをユーザー領域で実行します。データベース要求のサービス（スケジューリング、同期、I/O、メモリー管理、プロセス / スレッドの作成と破棄など）に必要なアクティビティは、カーネル・モードで実行されます。すべての CPU が完全に使用されているシステムの場合、正常な Oracle システムの 65 ～ 95% はユーザー領域で実行されます。

---

**注意：** UNIX システムの場合、CPU 統計情報では、I/O が導出されるまでの待機時間がアイドル時間として扱われています。

---

**仮想メモリー統計情報** 仮想メモリー統計情報は、主として、システム上でページング・アクティビティまたはスワッピング・アクティビティが発生していないことを検証するために使用します。ページングまたはスワッピングが発生すると、システムのパフォーマンスは急速かつ予想外に低下します。

個別のプロセス・メモリーの統計情報を使用すると、プログラミングの障害によるメモリー・リークを検出して、プロセス・ヒープから取得したメモリーの割当てを解除できます。また、この統計情報を使用すると、システムを起動して安定状態に達した後、メモリー使用量が増加していないことを検証できます。このメモリー使用量増加の問題は、中間層マシンの共有サーバー・アプリケーションでセッション状態が複数のユーザー対話の間持続され、完了時にメモリーの割当てが完全に解除されない状態の情報がある場合は、特に深刻です。

**ディスク統計情報** データベースは一連のディスクに常駐しているため、I/O サブシステムのパフォーマンスはデータベースのパフォーマンスにとって非常に重要です。ほとんどのオペレーティング・システムには、ディスクのパフォーマンスに関する広範囲な統計情報が用意されています。最も重要なディスク統計情報は、現在の応答時間とディスク・キューの長さです。この統計情報は、ディスクが最適に実行されているかどうか、ディスクがオーバーワークになっているかどうかを示します。ディスクの応答時間が 20 ミリ秒を超える場合は、ディスクが不適切に実行されているか、またはオーバーワークになっています。この状態がボトルネックになります。ディスク・キューが 3 以上になると、そのディスクはシステムのボトルネックになる可能性があります。

**ネットワーク統計情報** ネットワーク統計情報は、ディスク統計情報とほぼ同様に使用でき、ネットワークまたはネットワーク・インタフェースがオーバーロードかどうか、または最適に実行されているかどうかを判断できます。今日のネットワーク・アプリケーションでは、ユーザーの応答時間の大部分がネットワーク待機時間です。このため、この統計情報は重要なデバッグ・ツールです。

## データベース統計情報

データベース統計によって、データベースのロード・タイプ、およびデータベースで使用する内部リソースと外部リソースに関する情報が提供されます。データベース・リソースが消耗したときに、アプリケーション内のボトルネックを識別できます。

データベース統計情報は、SQL を使用したリレーショナル方式によってデータベースから直接取得できます。この統計情報は、INSERT INTO x AS SELECT ... 文または CREATE TABLE x AS SELECT ... 文を使用してデータベースに挿入しなおすことができます。この方法は、統計を長期間にわたって収集できるほとんどのスナップショット・メカニズムの基礎となっています。ほとんどの統計情報は、V\$ 表と呼ばれる（接頭辞が V\$ であるため）一連の仮想表またはビューに含まれています。これらは読取り専用で、SYS が所有しています。表の多くには、他の V\$ 表と結合できるように識別子とキーが含まれています。

意味のあるデータベース統計情報を取得するためには、データベース・インスタンスに対して TIMED\_STATISTICS パラメータを有効にする必要があります。TIMED\_STATISTICS パラメータを有効にした場合のパフォーマンスへの影響は、インスタンスのパフォーマンスと比較すると最小です。パフォーマンスの改善と完全な統計情報値のデバッグ処理にとって、このパラメータは、効率的なパフォーマンス分析の上で重要な位置を占めています。

主要なデータベース統計情報は次のとおりです。

- バッファ・キャッシュ
- 共有プール
- 待機イベント

**バッファ・キャッシュ** バッファ・キャッシュは、ディスクからメモリー内のバッファに読み込まれるブロックを管理します。また、直前に使用されたバッファおよび通常のデータベース処理で変更されたバッファに関する情報も保持しています。最適な問合せパフォーマンスを得るため、ユーザー問合せはバッファ・キャッシュ内の必要なすべてのデータ・ブロックにアクセスして、メモリーからの問合せを満たします。ただし、データベースのサイズはバッファ・キャッシュよりはるかに大きいため、常にこうなるわけではありません。この点に留意すると、バッファ・キャッシュには管理とチューニングが必要であることが理解できます。

バッファ・キャッシュをチューニングする目的は、キャッシュ内に必要なブロックをできるだけ多く持たせて、許容可能なユーザー問合せ時間を獲得することにあります。また、古いブロックをキャッシュからエージ・アウトするときに、シリアルライズ化ポイントまたはパフォーマンス・スパイクを引き起こさずに、時間がかかる I/O を削減することも目的です。このプロセスでは、バッファ・キャッシュ・メカニズム、データベース・ライターおよびチェックポイント・メカニズムに関する経験的な知識が必要です。ほとんどの情報は、V\$SYSSTAT 表から抽出できます。

**共有プール** 共有プールには、ユーザー・セッション、すべてのデータベース・セッションで使用する共有データ構造、およびディクショナリ・キャッシュに関する情報が含まれています。

共有プールを問い合わせることによって、データベース内で実行される SQL 文を分析できます。この問合せは、アプリケーション・ソース・コードに関する知識が不十分またはまったくない場合に特に重要です。実際の SQL 以外に、その SQL の実行回数、および SQL で実行される CPU とディスク I/O の量を決定できます。この情報は、V\$SQL 表から抽出できます。不明なアプリケーションをデバッグするときは、この情報を分析して対象のボトルネックを識別することが重要です。

**待機イベント** 通常のデータベース・サーバー操作のプロセスでは、リソースの共有や他のプロセスとの同期化が必要になる場合があります。たとえば、共有プール内でのメモリーの割当てやロックの待機などがあります。同様に、データベース・プロセスで、外部コードや制御外の他のプロセスを制御する場合があります。たとえば、I/O の実行や REDO ログを同期化するログ・ライターの待機などがあります。

このような場合、ユーザー・プロセスでは処理を停止して待機します。この待機時間は、最終的にユーザーの応答時間の一部になります。共有リソース上にキューイングされているマルチ・プロセスがある場合または同じ外部リソースを要求するマルチ・プロセスがある場合、データベースはシングル・スレッドを開始します。このため、拡張性に影響を与えます。パフォーマンス分析によって、データベースのリソースでキューイングが発生する原因を確認する必要があります。

V\$SYSTEM\_EVENT 表、V\$SESSION\_EVENT 表および V\$SESSION\_WAIT 表を使用すると、待機イベントの履歴やリアルタイムでの待機イベントを問い合わせることができます。V\$SESSION\_WAIT 表には、記録された待機イベントに基づいて他の V\$ 表に結合できる追加の列があります。V\$SESSION\_WAIT に指定されているこの追加の結合列を使用すると、待機イベントを重点的にドリルダウンして分析することができます。

**関連項目：** V\$ 表の詳細は、『Oracle9i データベース・リファレンス』を参照してください。

## アプリケーション統計情報

アプリケーション統計情報は、入手するのが最も困難ですが、システムに対して行われたパフォーマンス改善を測定するための重要な情報です。アプリケーション統計情報では、少なくとも、各作業時間に処理されたユーザー・トランザクションに関する日次サマリーが提供されます。さらに詳細な統計情報では、処理されたトランザクションの正確な詳細および各トランザクション・タイプの応答時間が提供されます。詳細な統計情報では、各トランザクションで要した時間をアプリケーション・サーバー、ネットワーク、データベースなどに分解した統計情報も提供されます。

最適な統計情報を入手するには、アプリケーションに関する計測手段がかなり必要です。計測手段を既存のアプリケーションに組み込むことは困難であるため、最初からアプリケーションに組み込むことをお勧めします。

## 統計情報収集ツール

### オペレーティング・システム・データ収集ツール

表 2-1 は、UNIX に関するオペレーティング統計情報を収集するためのツールです。

表 2-1 オペレーティング統計情報を収集するための UNIX ツール

UNIX	
CPU	sar、vmstat、mpstat、iostat
メモリー	sar、vmstat
ディスク	sar、iostat
ネットワーク	netstat

Windows NT/2000 の場合は、Performance Monitor ツールを使用します。

### データベース・データ収集ツール

Oracle には、3 種類のデータ収集ツールが用意されています。これらのツールのインストールおよび実行は、ますます複雑になっています。しかし、複雑になるにつれてレポート出力の内容も充実しています。次のツールがあります。

1. Statspack

Statspack は BSTAT/ESTAT スクリプトに基づいていますが、データベース・リポジトリ内のすべての統計情報を格納するようにデータ獲得が拡張されています。このため、詳細なベースライン設定とオフライン分析が可能です。Statspack レポートでは、BSTAT/ESTAT スクリプトよりも詳細な情報が、ボトルネックの検出に役立つような形式で提供されます。このメカニズムは、データベース統計情報を記録して収集するための最適な方法です。

2. Oracle Enterprise Manager (EM)

Oracle Enterprise Manager は、パフォーマンス・データの収集、格納およびレポートを行うためのグラフィカル・ユーザー・インタフェースを提供します。EM Intelligent Agent のデータ収集サービスによって、パフォーマンス・データをスケジュールに従って収集できます。1 つのエージェントで、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・システムに関するデータ収集を管理できます。データは、パフォーマンスをレポートするために、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は、データベース・パフォーマンスの多様な側面を分析するために使用できます。これらの側面には、データベース・ロード、キャッシュの割当てと効率、リソースの競合、影響が大きい SQL などがあります。

パフォーマンス・データ収集は、EM コンソールから直接起動するか、EM Diagnostics Pack - Capacity Planner アプリケーションを介して起動できます。パフォーマンスの履歴データに関する HTML レポートは、EM コンソールから生成できます。この HTML レポートによって、データベース・システムの使用率とパフォーマンスを包括的に分析することができます。このレポートには、ブラウザから簡単にアクセスしてナビゲートできます。また、EM では、グラフィカルな Performance Overview がリアルタイムで提供されるため、折れ線グラフや棒グラフなどを使用して、パフォーマンス・メトリックのサブセットを監視することができます。

Performance Overview のグラフでは、パフォーマンス・データをドリルダウンしてパフォーマンス・ボトルネックのソースを追跡することで、既存のパフォーマンスの問題をトラブルシュートできます。たとえば、大量のソート・アクティビティを実行するセッションとそれに対応する SQL をドリルダウンすることで、メモリー・ソート率の低下を即座に調査できます。このプロセスで検出された影響が大きい SQL 文は、問題のコンテキストで SQL 診断ツールを起動して、さらに詳細に調査できます。

### 3. BSTAT/ESTAT スクリプト

これらのスクリプトは、\$ORACLE\_HOME/rdbms/admin ディレクトリにある UTLBSTAT.SQL ファイルおよび UTLESTAT.SQL ファイルです。これらのツールは、2 つの時点の間で実行されたデータベース・アクティビティについて簡単なレポートを作成します。レポートは長期にわたってアーカイブできます。これらの統計情報は、収集する必要がある最低限の統計情報です。

---

---

**注意：** BSTAT/ESTAT は、8.0 以降に導入された多くの新機能に合せて更新されていないため、いずれサポート対象外となります。

Oracle 8 より前のリリースを実行していて、パフォーマンス・チューニングに関するヘルプが必要なユーザーは、BSTAT/ESTAT ではなく、Statspack の出力を用意してください。Statspack は、解析が容易な上に詳細な情報を提供するため、チューニングを迅速かつ効率的に行うことができます。

---

---

**関連項目：**『Oracle Enterprise Manager 概説』

## 履歴データとベースラインの重要性

パフォーマンス担当者の重要な仕事の 1 つは、正常に稼働していたアプリケーションにパフォーマンスの問題が生じた場合、その原因はシステムで何が変化したかを判断することです。しかし、現代の複雑なシステムでは、考えられる原因は広範囲にわたります。

パフォーマンスの履歴データは、できるだけ多くの変数を削減するために重要です。これは、アプリケーションが本番環境に展開されたその日から、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の統計情報を収集する必要があることを意味します。これは、満足なパフォーマンスが得られない場合にも該当します。アプリケーションが安定し、パフォーマンス特性に関する理解が進むと、一連の統計情報は今後の参照資料のベースラインになります。この統計情報は、満足なパフォーマンスが得られないときの統計情報と照合するために使用できます。また、将来の容量計画や拡大計画を立てる場合の基礎となります。

## パフォーマンスに対する直観的判断

データベースとオペレーティング・システムの統計情報によって、システムの実行状態が示されます。統計情報と実際のスループットを関連付けることによって、システムの実行状態を確認し、この先ボトルネックやリソース不足の可能性のある箇所を判断できます。これは、システムの監視および Oracle サーバーの作業の経験から得られたスキルです。

CPU の使用率は、理解したり監視するのが最も簡単なシステム使用率に関する統計情報です。この統計情報のある期間にわたって監視すると、稼働日 1 日および数週間にわたるシステムの使用状況を理解できます。ただし、この統計情報では、実行されたビジネス・トランザクションの数や各トランザクションに使用されたリソースは示されません。

実際に実行されたビジネス・トランザクション数を示す統計情報は、コミット数と生成された REDO の量です。これらの情報は、USER COMMITS および REDO SIZE の V\$SYSSTAT ビューにあります。これらの統計情報は、実際のトランザクション数とデータベース内で変更されたデータ量を示しています。アプリケーションやトランザクションのロジックは変更されていないのに、これらの数値が長期的に増加している場合は、より多数のビジネス・トランザクションが実行されていたことがわかります。論理ブロックの読取り数 (V\$SYSSTAT 統計の 'session logical reads') は、システムでの問合せワークロードも示します。この数値は慎重に解析してください。論理ブロックの読取り数の変化は、ワークロードの増加が原因ではなく、実行計画の変更が原因である場合があります。

経験を重ねると、データベース統計情報とアプリケーション・ワークロードとを簡単に関連付けることができます。パフォーマンス担当の DBA は、データベース統計情報とアプリケーション・プロファイルを使用して、システムのワークロード特性を直観的に判断できるようになります。DBA は、頻繁に実行するトランザクションのパフォーマンスも予測する必要があります。アプリケーション内の主要な SQL 文を理解することは、パフォーマンスを診断する上で重要です。このようなアクティビティの多くは、非公式に行うことができます。

たとえば、ある主要なビジネス・トランザクションは、0.1 秒以内の応答時間で実行することが要求されているとします。トランザクションの初期調査では、このトランザクションの Logical Reads（論理読取り）は 200 で、その中の 40 は常にディスクから取得しています。ディスクの応答時間は 20 ミリ秒かかるため、I/O 時間は  $40 \times 0.02 = 0.8$  秒となり、応答時間の目標を達成しません。DBA は、論理 I/O 数を 80 に減らし、ディスクから取得する数は平均で 5 にするようにトランザクションの再設定をします。

パフォーマンス不足を回避するには、このような計算を本番環境を展開する前に実行することをお勧めします。システムが本番環境に展開された後、データ量は増加し、トランザクション統計情報も変化する可能性があるため、このプロセスは繰返し行う必要があります。

## Oracle のパフォーマンス改善方法

### パフォーマンス改善の概要

Oracle のパフォーマンス方法論は、使用している Oracle システムでのパフォーマンス問題を特定するのに役立ちます。この方法論には、ボトルネックの識別とその修正も含まれています。システムの変更は、ボトルネックの存在を確認した後に行うことをお勧めします。

パフォーマンスの改善は、本質的に反復的なプロセスです。つまり、最初のボトルネックを取り除いても別のボトルネックが現れるため、パフォーマンスがすぐに改善されない場合があります。また、シリアライズ・ポイントが効率の悪い共有メカニズムに移動したことによって、パフォーマンスが低下する場合があります。経験を重ね、ボトルネックを除去する方法に厳密に従うことで、アプリケーションをデバッグしてスケーラブルにすることができます。

パフォーマンスの問題は、通常、スループットの不足または許容範囲を超えたユーザー / ジョブ応答時間（あるいはその両方）が原因で発生します。問題は、アプリケーション・モジュール間に点在している場合や、システム全体にわたる場合があります。

データベースやオペレーティング・システムの統計情報を調べる前に、システムで最も重要なコンポーネントからのフィードバックを取得することが重要です。つまり、システムのユーザーとアプリケーションにかかる費用を最終的に負担する顧客からのフィードバックです。ユーザーからの典型的なフィードバックには、次のような報告が含まれます。

- オンラインのパフォーマンスが低いため、スタッフの仕事が停滞しています。
- 請求処理の実行時間が長すぎます。
- Web トラフィックが多くなると応答時間が長くなるため、このままでは顧客を失いかねません。
- 現在、1 日に 5000 件の取引を行っていますが、システムが限界に達しています。来月は、すべてのユーザーにロールアウトを行うため、取引数は 4 倍になる見込みです。

このような率直なフィードバックから、パフォーマンスの改善作業には厳しい成功要因を設定する理由が容易に理解できます。パフォーマンス目標とパフォーマンス担当者の最終基準を決定することで、すべてのレベルでのパフォーマンス・プロセスの管理が簡素化され、効率化されます。これらの重要な成功要因は、システム統計情報ではなく現実的なビジネス目標の観点から定義した方が効果的です。

前述のユーザーからの典型的なフィードバックに対して、現実的なビジネス目標を次のように設定できます。

- 請求処理では、3 時間で 1,000,000 件を処理する。
- Web サイトのピーク時には、1 つのページ・リフレッシュに対する応答時間は 5 秒以内にする。
- システムでは、8 時間で 25,000 件の取引を処理可能にする。

最終的な満足度は、システム・パフォーマンスに関するユーザーの認識で判断されます。パフォーマンス担当者の役割は、パフォーマンスを低下させているボトルネックを取り除くことです。ボトルネックの原因は、限られた共有リソースの非効率的な使用またはシリアルライズ化の原因となる共有リソースの不適切な使用にあります。すべての共有リソースには制限があるため、パフォーマンス担当者の目標は、効率的に共有リソースを使用してビジネス処理の件数を最大にすることです。高いレベルでは、データベース・サーバー全体を共有リソースとみなすことができます。逆に低いレベルでは、1 台の CPU やディスクを共有リソースとみなすことができます。

Oracle のパフォーマンス改善方法は、パフォーマンス目標を達成するまで、またはそれ以上の改善は不可能と判断するまで適用できます。このプロセスは反復的なプロセスであるため、システムのパフォーマンスにほとんど影響を与えないような調査を行う必要があります。重要なボトルネックを正確かつ適時に特定できるスキルを養うには、時間と経験が必要です。ただし、利用できるデータや統計情報を軽視する技術者は、たとえ経験豊かでも問題外です。このような技術者は、思い込みと不正確な方法によってデータベースのチューニングを行おうとします。これはデータベースのチューニング方法として非常に危険でコストもかかり、成功するとは考えられません。

今日のシステムは、それぞれが異なり複雑であるため、パフォーマンス分析のための確実で手間のかからない規則は作成できません。Oracle のパフォーマンス改善方法では、本質的に 1 つの作業方法を定義しており、最終的な規則は定義していません。ボトルネックの検出に関する唯一の規則は、規則がないことです。優れたパフォーマンス担当者は、提供されたデータを利用し、様々な側面を検討してパフォーマンスの問題を判断します。

## Oracle のパフォーマンス改善方法の手順

1. ユーザーから率直なフィードバックを取得します。パフォーマンス・プロジェクトの適用範囲、最終的なパフォーマンス目標、および将来のパフォーマンス目標を決定します。このプロセスは、今後の容量計画を立てるために重要です。
2. パフォーマンスの良し悪しに関係なく、システムからオペレーティング・システム、データベースおよびアプリケーションの統計情報をすべて取得します。すべての統計情報を取得できない場合は、できるかぎり取得します。使用できる統計情報がないのは、犯罪捜査で証拠がないのと同じです。証拠がないと捜査は困難で時間もかかります。
3. ユーザー・パフォーマンスが関連している全マシンのオペレーティング・システムが、健全であることをチェックします。オペレーティング・システムが健全であることをチェックすることによって、完全に使用されているハードウェアまたはオペレーティング・システムのリソースを検出できます。過度に使用されているリソースを症状としてリストし、後で分析します。さらに、すべてのハードウェアでエラーや診断症状がないことをチェックします。
4. Oracle に関して最もよく見られる誤りの上位 10 項目をチェックし、それぞれが問題となる可能性がないかを判断します。問題となる可能性がある場合は、症状としてリストし、後で分析します。これらの症状は、問題となる可能性が高いという理由から、リストに含めます。

**関連項目：** 2-13 ページ「[Oracle システムにおける誤りの上位 10 項目](#)」

5. 症状を手掛かりにして、システム上で発生している状況を概念的にモデル化し、パフォーマンス問題の原因を把握します。

**関連項目：** 2-12 ページ「[パフォーマンスを概念的にモデル化する際の意志決定プロセスの例](#)」

6. 一連の修正処理とシステムに対して予想される動作を提示し、アプリケーションに対して最も有効な処理から順に適用します。パフォーマンス改善作業での重要な規則は、一度に 1 つの変更のみ行い、変更前後の差異を測定することです。ただし、システム停止時には、この調査方法を忠実に実行するのは現実的ではありません。一度に複数の変更を適用する場合は、それぞれの変更が分離されるようにしてください。
7. 変更によって予定通りにパフォーマンスが改善されたことを検証し、ユーザーがパフォーマンスの改善を認識したかどうかを確認します。ユーザーが認識しない場合は、さらにボトルネックを探し、アプリケーションをより正確に把握するまで、概念的なモデルの調整を続けます。
8. パフォーマンス目標を達成するまで、または他の制約によってそれ以上の改善は不可能になるまで、最後の 3 つの手順を繰り返します。

この方法によって最大のボトルネックを識別し、パフォーマンス改善の目標を定めて作業を行います。重要なことは、アプリケーションの効率を高め、リソース不足とボトルネックを除去することで、パフォーマンスを大幅に改善することです。このプロセスでは、インスタ

ンスのチューニングで最低限（10% 未満）のパフォーマンス向上が期待でき、アプリケーションの効率の悪さを分離することで大幅な（100% 以上）パフォーマンス向上が期待できます。

## オペレーティング・システムのチェック方法

オペレーティング・システムの諸症状をチェックする際は、次の内容を考慮してください。

- システム全体と各 CPU について、ユーザー領域とカーネル領域での CPU 使用率をチェックします。
- ページングまたはスワッピングがないことを確認します。
- マシン間のネットワーク待機時間が許容範囲内であることをチェックします。
- 応答時間が長い、キューが長いディスクを検索します。
- ハードウェア・エラーがないことを確認します。

## パフォーマンスを概念的にモデル化する際の意志決定プロセスの例

概念的なモデル化は、最終的な意思決定に近い段階です。ただし、パフォーマンスのチューニング経験を重ねるに従って、実際に準拠する規則はないことがわかります。様々な統計情報を解析して適切な意思決定を行うには、柔軟性のある辛抱強いアプローチが必要です。

この項では、パフォーマンス担当者がボトルネックを調べる方法を説明します。ここで説明する方法は、プロセスのガイドラインとしてのみ使用してください。経験に従って、パフォーマンス担当者は関連する手順を追加していきます。この分析では、オペレーティング・システムとデータベースの統計情報は収集済みと想定しています。

1. ロード量がないか少量のマシンでの、1 名のユーザーに対する応答時間 / バッチ実行時間は許容範囲内ですか？

許容範囲外の場合は、アプリケーションのコードまたは設計が不適切であると考えられます。システム・リソースを共有するマルチ・ユーザー状況では、許容範囲内になることはありません。このような場合は、アプリケーションの内部統計情報を取得し、SQL トレースと SQL の計画情報を取得します。開発者とともに、データ、索引およびトランザクション SQL 設計に関する問題、およびバッチ / バックグラウンド処理の遅延の可能性について調査します。

2. すべての CPU が使用されていますか？

カーネル使用率が 40% を超えた場合は、ネットワーク転送、ページング、スワッピングまたはプロセス・スラッシングについてオペレーティング・システムを調査します。それ以外の場合は、ユーザー領域内の CPU 使用率を調査します。バックアップ、ファイル変換、印刷キューなど、CPU を消費するデータベース以外のジョブがマシン上で CPU の共有リソース量を制限していないかをチェックします。データベースが CPU のほとんどを使用していることを確認した後、CPU 使用率が上位の SQL を調査します。これらの文は、今後の分析の基礎になります。SQL および SQL を発行するトランザク

ションが最適に実行されていることをチェックします。Oracle9i より前のリリースの Oracle サーバーでは、バッファ取得数を使用して CPU 使用率を測定します。Oracle9i の Oracle サーバーでは、V\$SQL に実際の CPU 統計情報が用意されています。

**関連項目：** V\$SQL の詳細は、『Oracle9i データベース・リファレンス』を参照してください。

アプリケーションが最適で SQL が効率的に実行されている場合は、一部の作業をオフピーク時に再スケジュールしたり、大型のマシンを使用することを検討してください。

3. この段階でも、システム・パフォーマンスが不十分で、CPU リソースが完全に使用されていない場合。

このような場合は、サーバー内にシリアル化化や拡張性のない動作があります。サーバーから WAIT\_EVENTS 統計情報を取得し、最大のシリアル化・ポイントを確認してください。シリアル化・ポイントがない場合、問題はデータベース外にある可能性があります。重点的に調査する必要があります。WAIT\_EVENTS の絞込みには、アプリケーション SQL の修正とデータベース・パラメータのチューニングが含まれます。このプロセスは反復的なプロセスで、WAIT\_EVENTS を系統的にドリルダウンしてシリアル化・ポイントを絞り込む技術が必要です。

## Oracle システムにおける誤りの上位 10 項目

この項では、Oracle システムで最もよく見られる誤りについて説明します。Oracle のパフォーマンス改善方法に従うことで、これらの誤りを回避できます。システム内で誤りを発見した場合は、パフォーマンスの改善努力が効果を上げるようにアプリケーションを再設計します。

**関連項目：** 待機イベントの詳細は、『Oracle9i データベース・パフォーマンス・チューニング・ガイドおよびリファレンス』を参照してください。

1. 不適切な接続管理

アプリケーションでは、データベースとの対話ごとに接続と切断が行われます。この問題は、アプリケーション・サーバーのステートレスなミドルウェアで多く発生します。この問題がパフォーマンスに与える影響は 3 倍以上となり、まったく拡張性がありません。

2. カーソルと共有プールの不適切な使用

カーソルを使用しないと、解析が繰返し行われる結果になります。また、バインド変数を使用しないと、すべての SQL 文のハード解析が行われます。この問題がパフォーマンスに与える影響は倍となり、まったく拡張性がありません。カーソルをオープンするバインド変数とともにカーソルを使用し、繰返し実行します。アプリケーションで動的 SQL が生成されているかどうかを確認してください。

### 3. 誤ったデータベース I/O の取得

多くのサイトでは、データベースが使用可能なディスクに適切にレイアウトされていません。他のサイトでは、ディスクを I/O 帯域幅ではなくディスク領域ごとに構成しているため、ディスクの数が誤って指定されています。

### 4. REDO ログ設定の問題

多くのサイトは、REDO ログがほとんどない状態で実行され、REDO ログのサイズも小規模です。REDO ログが小規模な場合、システム・チェックポイントでは、バッファ・キャッシュと I/O システムに対して高いロードを与え続けます。REDO ログがほとんどない場合は、アーカイブが間に合わないため、データベースはアーカイブ・プロセスが追いつくまで待機します。

### 5. 空きリスト、空きリスト・グループ、トランザクション・スロット (INITRANS)、またはロールバック・セグメントの不足による、バッファ・キャッシュ内のデータ・ブロックのシリアル化

この問題は、INSERT 操作が多いアプリケーション、ブロック・サイズが 8K または 16K に増えたアプリケーション、または多数のアクティブ・ユーザーがいてロールバック・セグメントがほとんどないアプリケーションで特に多い問題です。

### 6. 時間がかかる全表スキャン

大量またはインタラクティブなオンライン操作に対する全表スキャンに時間がかかる場合は、不適切なトランザクション設計、索引の欠落または不適切な SQL 最適化が原因です。時間がかかる全表スキャンは、本質的に I/O 集約型で拡張性がありません。

### 7. ディスク内ソート

オンライン操作に対するディスク内ソートは、不適切なトランザクション設計、索引の欠落または不適切な SQL 最適化が原因です。ディスク・ソートは、本質的に I/O 集約型で拡張性がありません。

### 8. 大量の再帰的 (sys) SQL

sys によって実行される大量の再帰的 SQL は、エクステンツ割当てなどのスペース・マネジメント・アクティビティがあることを示します。これは拡張性がなく、ユーザーの応答時間に影響を与えます。別のユーザー ID で実行される再帰的 SQL は、SQL および PL/SQL である場合が多いため、問題ではありません。

### 9. スキーマ・エラーとオプティマイザの問題

多くの場合、表を所有するスキーマが開発環境や古い実装から正常に移行されていないために、アプリケーションでは必要以上にリソースが使用されます。この例には、索引の欠落や不正確な統計情報などがあります。このようなエラーが、必ずしも最適ではない実行計画や低いインタラクティブ・ユーザー・パフォーマンスの原因になる場合があります。パフォーマンスが明確なアプリケーションを移行するときは、DBMS\_STATS パッケージを使用してスキーマ統計情報をエクスポートし、プラン・スタビリティをメンテナンスします。

同様に、初期化パラメータ・ファイルに設定したオブティマイザ・パラメータを、確認済みの最適な実行計画より優先させることができます。このために、スキーマ、スキーマ統計情報およびオブティマイザ設定をグループとしてまとめて管理し、パフォーマンスの一貫性を保証します。

#### 10. 非標準初期化パラメータの使用

このパラメータは、不適切なアドバイスや不正確な仮定に基づいて実装されている場合があります。特に、ラッチ上の `SPIN_COUNT` に対応付けられたパラメータおよびマニュアルに記載されていないオブティマイザ機能は、多くの問題の原因となり、このため、かなりの調査が必要となる可能性があります。

## ハードウェア構成のパフォーマンス特性

繰り返しますが、今日のシステムは、それぞれが異なり複雑であるため、パフォーマンス分析のための確実で手間のかからない規則は作成できません。ただし、検討する必要があるいくつかの数値について、この項で説明します。

#### 1. ディスク特性

- サイズは 512MB ～ 36GB
- シークは 5 ～ 10 ミリ秒
- 転送は 5 ～ 10 ミリ秒
- スループットは、各ディスクに対して 20 ～ 40 I/O 秒
- コントローラのスループットは、1 秒間に 1750 I/O

#### 2. メモリーからの読み込みスピードは 1 ～ 10 マイクロ秒

#### 3. ポイントツーポイントでのネットワーク待機時間は 1 ～ 25 ミリ秒

#### 4. ビジー・システム（最悪のケース）

- 業務系システム—ユーザー 60%、システム 40%
- 意思決定支援システム—ユーザー 90%、システム 10%



---

## パフォーマンスの緊急事態に対処する方法

この章は、次の項で構成されています。

- [パフォーマンスの緊急事態に対処する方法の概要](#)
- [パフォーマンスの緊急事態に対処する方法の手順](#)

## パフォーマンスの緊急事態に対処する方法の概要

この章では、パフォーマンスに関する緊急事態に対処する方法について説明します。このマニュアルの最初の2つの章では、アプリケーション・パフォーマンスの確立方法と改善方法を詳細に説明しました。しかし、緊急時にはシステム・コンポーネントの変化によって、信頼性の高い予測可能なシステムが、予測不可能でユーザー要求を満たすことができないシステムに変貌します。

このような場合のパフォーマンス担当者の役割は、変化した内容を迅速に特定して適切な処理を行い、通常のサービスにできるだけ早く戻すことにあります。多くの場合、緊急な処理が必要であり、パフォーマンス改善方法を厳密に実行するのは現実的ではありません。

パフォーマンス担当者は、パフォーマンスの問題をただちに特定して十分なデバッグ情報を収集し、その問題を正確に把握するか、それができない場合は、少なくとも同じ問題が再発しないことを確認する必要があります。

パフォーマンスの緊急事態に関する問題をデバッグする方法は、このマニュアルの前の章で説明したパフォーマンス改善方法と同じです。ただし、この問題の時間的な制約から、いくつかの段階で省略されています。デバッグ・プロセス時に見つかった事実を詳細にメモしたり記録しておくことは、後で行う修正作業の分析と正当化に対する基礎になります。これは、医師が患者の容態を詳細に記録して、将来に役立てることと同じです。

## パフォーマンスの緊急事態に対処する方法の手順

パフォーマンスの緊急事態に対処する方法は次のとおりです。

1. パフォーマンス上の問題を調査して、その症状を収集します。このプロセスには、次の2つの作業が含まれます。
  - システムのパフォーマンスが低下した状況について、ユーザーのフィードバックを収集します。問題がスループットか応答時間かを調べます。
  - ユーザーに、「パフォーマンスが正常だった時点から何か変化しましたか？」と質問します。この質問に対する答えは、問題を明確にする手掛りになる可能性があります。ただし、状況が悪化する中で先入観のない答えを得るのは困難な場合があります。
2. アプリケーション・システムの全コンポーネントのハードウェア使用率が健全であることをチェックします。CPU使用率が最も高いコンポーネントをチェックし、システム全体のコンポーネントについてディスク、メモリー使用量およびネットワーク・パフォーマンスをチェックします。このプロセスによって、問題の原因となっている層を迅速に識別できます。問題がアプリケーション内で発生している場合は、アプリケーション・デバッグを分析します。それ以外の場合は、データベース・サーバーを分析します。
3. データベース・サーバーがCPU上で制約を受けているか、待機イベントで待機し続けているのかを判断します。データベース・サーバーがCPU上で制約を受けている場合は、次の点を調査します。

- オペレーティング・システム・レベルで大量の CPU を消費しているセッション
- データベース・レベルで多数のバッファ取得を実行しているセッションまたは文 (V\$SESSTAT と V\$SQL をチェック)
- 最適ではない SQL の実行の原因となっている実行計画の変更 (特定するのは困難な場合があります)
- 初期化パラメータの誤った設定
- コード変更または全コンポーネントのアップグレードによるアルゴリズムの問題

データベース・セッションがイベント上で待機している場合は、V\$SESSION\_WAIT のリストにある待機イベントに従って、シリアライズ化の原因を判断します。ライブラリ・キャッシュの大規模な競合の場合は、ログインしたり SQL をデータベースに発行するのが不可能になる場合があります。このような場合は、履歴データを使用して、そのラッチで突然競合が発生した原因を判断します。ほとんどの待機が I/O に対する待機の場合は、すべての I/O を実行するセッションでサンプルの SQL を実行してみます。

4. 緊急時の処理を適用して、システムを安定化します。この安定化には、アプリケーションの一部をオフラインにしたり、システムに適用されている可能性があるワークロードを制限する処理を含めることができます。また、システムの再起動や処理中のジョブの終了も含めることができます。これらの処理は、その性質上、サービス・レベルと密接な関係があります。
5. システムが安定していることを確認します。システムを変更したり制限した場合は、システムが安定したことを確認し、データベースに関する統計情報を参照として収集します。次に、このマニュアルの前の章で説明したパフォーマンス改善方法に厳密に従い、すべての機能とユーザーをシステムに戻します。このプロセスでは、完璧を期すために、アプリケーションの大幅な再設計が必要になる場合があります。

**関連項目：** パフォーマンス改善方法と Oracle に関して最もよく見られる誤りについては、[第 2 章「アプリケーションのパフォーマンスの監視と改善」](#)を参照してください。



# 索引

## B

BSTAT/ESTAT スクリプト, 2-7  
B ツリー索引, 1-15

## C

CPU, 1-8  
統計情報, 2-2

## E

EM (Enterprise Manager), 2-6  
Enterprise Manager, 2-6

## I

IOT (索引構成表), 1-15

## O

Oracle のパフォーマンス改善方法, 2-9  
手順, 2-11

## S

Statspack, 2-6

## あ

アプリケーション  
開発の傾向, 1-21  
実装, 1-19  
設計の原理, 1-13  
配置, 1-25  
パフォーマンス, 2-8  
アプリケーションの配置, 1-25

## い

インターネットにおける拡張性, 1-4

## お

応答時間, 1-12  
オブジェクト指向, 1-21  
オペレーティング・システム  
症状の収集, 2-12  
チェック, 2-12  
オペレーティング・システム統計情報, 2-2

## か

概念的なモデル化, 2-12  
開発環境, 1-19  
拡張性, 1-3  
インターネット, 1-4  
妨げる要因, 1-6  
線形, 1-6  
仮想メモリー統計情報, 2-3

## き

---

逆キー索引, 1-16  
共有プール, 2-5  
緊急事態  
    パフォーマンス, 3-2

## さ

---

サービスの時間, 1-12  
索引  
    B ツリー, 1-15  
    逆キー, 1-16  
    コスト, 1-16  
    順序, 1-16  
    シリアライズ化, 1-16  
    設計, 1-14  
    パーティション化, 1-16  
    ビットマップ, 1-15  
    ファンクション・ベース, 1-15  
    列の順序付け, 1-16  
    列の追加, 1-15  
索引構成表, 1-15

## し

---

システム・アーキテクチャ, 1-7  
    構成, 1-10  
    ソフトウェア・コンポーネント, 1-8  
        データとトランザクション, 1-10  
        ビジネス・ロジックの実装, 1-9  
        ユーザー・インタフェースの管理, 1-9  
        ユーザー要求とリソース割当て, 1-9  
    ハードウェア・コンポーネント, 1-7  
        CPU, 1-8  
        I/O サブシステム, 1-8  
        ネットワーク, 1-8  
        メモリー, 1-8

## せ

---

設計  
    検証, 1-24  
    テスト, 1-24  
    デバッグ, 1-24  
設計の検証, 1-24  
設計の原理, 1-13  
設計のテスト, 1-24  
設計のデバッグ, 1-24  
線形拡張性, 1-6

## そ

---

ソフトウェア・コンポーネント, 1-8

## て

---

ディスク統計情報, 2-3  
データ  
    検索, 1-12  
    収集, 2-6  
    問合せ, 1-12  
    トランザクション, 1-10  
    ベースライン, 2-8  
    モデル化, 1-14  
    履歴, 2-8  
データの収集, 2-6  
データベース  
    サイズ, 1-13  
データベース統計情報, 2-4

## と

---

問合せ  
    データ, 1-12  
統計情報  
    アプリケーション・ワークロードとの関連, 2-8  
    オペレーティング・システム, 2-2  
        CPU 統計情報, 2-2  
        仮想メモリー統計情報, 2-3  
        ディスク統計情報, 2-3  
        ネットワーク統計情報, 2-3

収集ツール, 2-6  
  BSTAT/ESTAT スクリプト, 2-7  
  Oracle Enterprise Manager, 2-6  
  Statspack, 2-6  
  データベース・データ, 2-6  
データベース, 2-4  
  共有プール, 2-5  
  バッファ・キャッシュ, 2-4  
トランザクションとデータ, 1-10  
トリクル・ロールアウトの方法, 1-25

## ね

---

ネットワーク, 1-8  
ネットワーク・スピード, 1-12  
ネットワーク統計情報, 2-3

## は

---

パーティション索引, 1-16  
ハードウェア・コンポーネント, 1-7  
バッファ・キャッシュ, 2-4  
パフォーマンス改善方法, 2-9  
  手順, 2-11  
パフォーマンスの緊急事態, 3-2  
パフォーマンスの緊急事態に対処する方法, 3-2

## ひ

---

ビジネス・ロジック, 1-9, 1-19  
ビジネス・ロジックの実装, 1-9  
ビッグ・バン・ロールアウトの方法, 1-25  
ビットマップ索引, 1-15  
ビュー, 1-17  
表  
  設計, 1-14

## ふ

---

ファンクション・ベース索引, 1-15  
プログラミング言語, 1-19

## へ

---

ベースライン, 2-8

## ほ

---

ボトルネック  
  識別, 2-9  
  修正, 2-9

## め

---

メモリー, 1-8

## も

---

モデル化  
  概念的, 2-12  
  データ, 1-14  
  ワークロード, 1-23

## ゆ

---

ユーザー  
  位置, 1-12  
  インタフェース, 1-19  
  応答時間, 1-12  
  対話方法, 1-11  
  人数, 1-11  
  ネットワーク・スピード, 1-12  
  要求, 1-19  
ユーザー・インタフェースの管理, 1-9  
ユーザー要求, 1-9

## り

---

リソース割当て, 1-9, 1-19  
履歴データ, 2-8

## れ

---

列の順序付け, 1-16

## ろ

---

ロールアウトの方法  
  トリクル・アプローチ, 1-25  
  ビッグ・バン・アプローチ, 1-25

## わ

---

ワークロード

テスト, 1-24

見積り, 1-22

推定, 1-22

ベンチマーク, 1-23

モデル化, 1-23

ワークロードの推定, 1-22

ワークロードのベンチマーク, 1-23

ワークロードの見積り, 1-22

推定, 1-22

ベンチマーク, 1-23