ORACLE®

ORACLE
ハイパフォーマンスチューニングSQL編

ORACLE High-Performance SQL Tuning

Donald K. Burleson／著　平松徹／訳　日本オラクル株式会社／監修

SE
SHOEISHA

本書内容に関するお問合せについて

このたびは翔泳社の書籍をお買い上げいただき、誠にありがとうございます。弊社では、読者の皆様からのお問合せに適切に対応させていただくため、以下のガイドラインへのご協力をお願い致しております。下記項目をお読みいただき、手順に従ってお問い合わせください。

●ご質問される前に

弊社Webサイトの「Q&Aコーナー」（http://www.shoeisha.com/info/help.asp）をご参照ください。これまで受けたご質問への回答（FAQ）や、的確なご質問方法に関する情報を掲示しています。

●ご質問方法

弊社Webサイトの質問専用フォーム（http://www.shoeisha.com/book/qa/）をご利用ください。記載漏れや独自の用紙等によるご質問、お電話や電子メールによるお問合せ、本書にはさみ込まれたアンケートはがきに記入されたご質問等は、お受けしておりません。

※質問専用シートのお取り寄せについて

Webサイトにアクセスする手段をお持ちでない方は、ご氏名、ご送付先（ご住所／郵便番号／電話番号またはFAX番号／電子メールアドレス）および「質問専用シート送付希望」と明記のうえ、電子メール（qaform@shoeisha.com）、FAX、郵便（80円切手をご同封願います）のいずれかにて“編集部読者サポート係”までお申し込みください。お申し込まれた手段によって、折り返し質問シートをお送りいたします。シートに必要事項を漏れなく記入し、“編集部読者サポート係”までFAXまたは郵便にてご返送ください。

●ご回答について

ご回答は、ご質問いただいた手段によってご返事申し上げます。ご質問の内容によっては、回答に数日ないしはそれ以上の期間を要する場合があります。

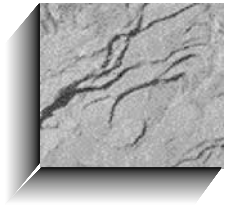
●ご質問に際してのご注意

本書の対象を越えるもの、記述個所を特定されないもの、また読者固有の環境に起因するご質問等にはお答えできませんので、予めご了承ください。

●郵便物送付先およびFAX番号

送付先住所　：〒160-0006 東京都新宿区舟町5
FAX番号　：03-5362-3818
宛先　：（株）翔泳社出版局 編集部読者サポート係

CONTENTS



著者紹介

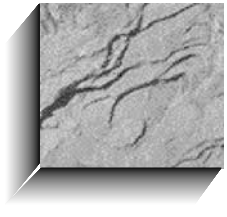
Donald Keith Burleson

20年あまりの現場経験を持つ、世界でも屈指のOracleデータベースエキスパート。大規模オンラインデータベースのアーキテクチャ設計を専門として、有数の強力かつ複雑なシステムを構築。これまでに執筆した書籍は12冊、雑誌記事は60本を超える。また、Oracleデータベースの牽引役とも呼べる『Oracle Internals』誌の編集主任も務めている。

また、DonaldはOracleデータベースチューニング、Oracleデータウェアハウジング、Oracle Webデータベースの設計に特化したリモートDBAサポートとコンサルティング業務も行っており、http://www.dba-oracle.com/やhttp://www.remote-dba.net/をはじめ、数々のWebサイトを手がけている。以前は大学の助教授としても活躍し、コンピュータの学士コースで100を超える講座を担当した。講師としても有名で、Oracle Openworldをはじめとする国際的なデータベースカンファレンスでもたびたび講演している。

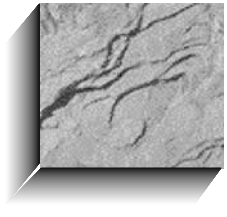
TechRepublic.comではコラム「Oracle Answers」の執筆を担当し、Oracleのテクニカルヒントを週刊で電子メール配信している。

コンサルタント業務以外では、視覚障害者を支援する慈善プログラムも精力的に手がけている。Donaldはミニチュアポニーをガイドホース(盲導馬)として調教する手法を開拓し、The Guide Horse Foundationという非営利団体を運営(http://www.guidehorse.org/またはhttp://www.guidehorse.com/)。同団体では、視覚障害のある人々に盲導馬を無償で提供している。



謝辞

専門書を作るという作業には、必ずチームワークがあります。本書が生まれたのも、ひとえにさまざまな人々が惜しまず協力してくれたおかげです。まず、本書の基本的な構想と肉付けを熱心に支援してくれたLisa McClainに、そして、本書の編集とデザインで多大な貢献をしてくれたRoss Doll、Carolyn Welch、Paulina Pobochaに感謝します。それから、本書の索引作りですばらしい活躍をしてくれたJennifer Burlsonにも感謝の言葉を。



まえがき

はじめに

Oracle SQLのチューニングは、Oracleチューニングでも最も重要な分野です。そして、SQL文をうまくチューニングすることができれば、Oracleデータベース全体のパフォーマンスを劇的に改善することができます。

しかし残念ながらOracleの専門家にも、SQL文を最適な形でチューニングする方法を知らない人が多いのです。

本書は、SQL文を記述しなければならない人々にとって大きな助けとなる、包括的なガイドブックとして役立つよう執筆したものです。本書では読者のみなさんにSQLチューニングの手順をすべて紹介し、あらゆるSQLをチューニングできるよう、実践的なヒントとテクニックを提供します。

本書では、Oracleのオプティマイザモードについて、またSQLの実行計画を表示するテクニックについて解説し、あらゆるSQL文のパフォーマンスを改善する方法を豊富に用意しています。また、非常に複雑で専門的なタスクを取り上げ、SQLチューニングの手順を1つ1つ、わかりやすく説明しています。

Oracle SQLは「宣言型の」言語なので、ユーザーは使用するデータ列、データが格納されている表、そしてデータに対する制約を指定するだけです。SQLは、生まれながらにすばらしい力が備わっています。それゆえ、同じ処理を実行するSQL文をさまざまな形で記述することができますが、そのパフォーマンスは個々の文によって大きく異なります。本書ではSQL文の使い分け方や

内部処理の実行計画を表示する方法、そして実行計画を変更して文のパフォーマンスを上げる方法を取り上げます。あらゆるOracleデータベースのパフォーマンスを大幅に向上できる、胸が躍るようなトピックです。

本書には問題の原因となるSQL文を識別し、それらをチューニングして最適なパフォーマンスを実現する、お墨付きのテクニックを盛り込んでいます。どのテクニックにもOracle標準のユーティリティを利用しているため、SQL文をチューニングするために特別な製品は必要ありません。また、本書が包括的なガイドブックとなるよう、本書では単純なselect文から複雑な非相関副問合せまで、SQLチューニングのあらゆる側面を取り上げています。さらに、筆者がSQLのパフォーマンスを劇的に向上するべく編み出した、数々の実践的なヒントとテクニックも紹介しています。

Oracle SQLが進化を続け、いっそう複雑になっていくとともに、本書も進化していきます。今後の改訂に向けたご意見やアイデアがあれば、遠慮なくお知らせください。有益な情報があれば、don@burleson.ccまで電子メールをお願いします。

PART I

予備知識編

CHAPTER 1

SQL入門

「SQL」という語は、Structured Query Language（構造化問い合わせ言語）を略したものです。しかし、SQLは「構造化」されていません。また、「問い合わせ」だけに利用するものでもなく、さらには「言語」でもありません。SQLそのものはCやCOBOLなど、他の言語に組み込まれたものだからです。このような間違った名称で呼ばれているものの、SQLはリレーショナルデータベースへのアクセス手段としては最も一般的です。

この章ではOracle SQLの基本的な特徴を紹介し、本書の全体を通じて利用するテクニックの基礎を押さえておきましょう。ここでは次のトピックを取り上げます。

- **SQLの基本的な特徴**――ここではSQLと他のナビゲーショナルデータベース問い合わせ言語を比較します。
- **SQLのはじまり**――ここではSQLが進化し、事実上の業界標準データベースアクセス手段となるまでの道のりを紹介します。
- **SQLオプティマイザ**――ここではSQLを最適化する基本的な手順を簡単に紹介します。
- **SQLチューニングの目標**――ここではSQLのチューニング全般でめざすべき目標を取り上げます。
- **Oracleチューニングの一環としてのSQLチューニング**――ここではOracle全体のチューニングに合わせてSQLをチューニングする方法を説明します。
- **SQLチューニングの障害となる問題**――ここではOracle SQLをチューニングする際に発生する問題について説明します。
- **SQLのチューニング手順**――ここでは個別のSQL文をチューニングする際の一般的な目標について説明します。
- **SQLチューニングツールキット**――ここでは本書の全体を通じて利用する、SQL文のチューニングテスト用ツールキットを紹介します。

SQLの基本的な特徴

SQL標準規格は、もともとは既存データベースの煩雑なナビゲーショナル言語に代わる規格として提唱されたものです。1960年代は、IBMのIMSデータベースが唯一の大規模商用データベース管理システムでした。IMSはリレーショナルデータベースではなく階層データベースであり、内部ポインタ構造体を使ってデータベースのレコード間を移動していました。

ナビゲーショナルデータベースのアクセスツールでは、プログラマはデータ構造体の間を移動するのにポインタ追跡という手法を使う必要がありました。ここで、IDMSデータベースとして一般的だった初期CODASYLネットワークにおける問い合わせの例を見てみましょう。

```
MOVE 'JONES' TO CUST-DESC.
   OBTAIN CALC CUSTOMER.
   MOVE CUSTOMER-ADDRESS TO OUT-REC.
   WRITE OUT-REC.
   FIND FIRST ORDER WITHIN CUSTOMER-ORDER.
       PERFORM ORDER-LOOP UNTIL END-OF-SET.
************
 ORDER-LOOP.
************
     OBTAIN FIRST ORDER-LINE-REC WITHIN ORDER-LINE.
     PERFORM ORDER-LINE-LOOP UNTIL END-OF-SET.
     FIND NEXT ORDER WITHIN CUSTOMER-ORDER.
 ************
 ORDER-LINE-LOOP.
 ************
     OBTAIN NEXT ORDER-LINE-REC WITHIN ORDER-LINE.
     MOVE QUANTITY-ORDERED TO OUT-REC.
     WRITE OUT-REC.
     OBTAIN OWNER WITHIN ORDER-LINE-PRODUCT
     MOVE PRODUCT-NAME TO OUT-REC.
     WRITE OUT-REC.
```

この例では、問い合わせによってデータレコード間を移動し、レコードにアクセスし、ポインタを検出してから、そのポインタの値に基づいてポインタ間を移動しています(**図1-1**)。このタイプのデータベースにおける問い合わせでは「データを抽出するにはデータベースの内部構造を知る必要がある」という点がポイントとなります。

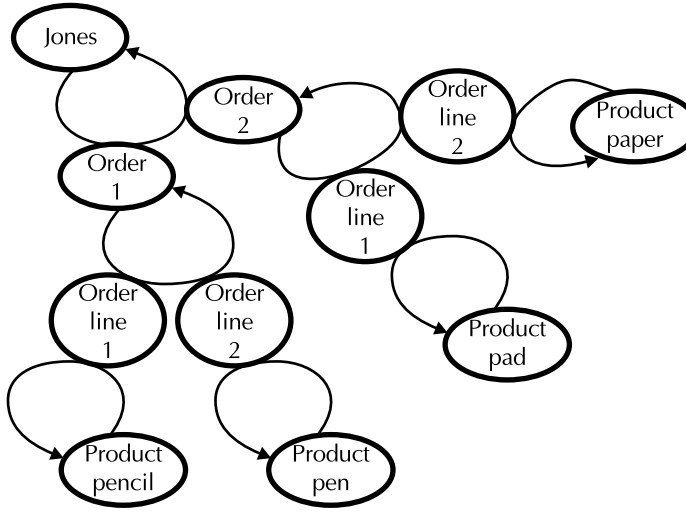


図 1-1　ナビゲーショナルデータベースの問い合わせ

これと同等の文をSQLで表現すると、構文も機能もまったく異なったものになります。ナビゲーショナルデータベースアクセス言語とは違い、SQLは表示したい列や、データを格納した表、表の結合条件といったものだけを指定するだけですむように設計されています。

```
select
   customer_address,
   product_name,
   quantity_ordered
from
   customer c,
   order o,
   order_line l,
   product P
where
   customer_name = 'JONES'
and
   c.cust_nbr = o.cust_nbr
and
   o.order_nbr = l.order.nbr
and
   l.product_nbr = p.product_nbr
;
```

SQLの基本構造については、この章の後半で詳しく見ていきます。

一般に、SQLといえばリレーショナルデータベースが連想されますが、SQLが非リレーショナルデータベースの世界でも一般的であることを覚えておくのは重要なことです。IDMSネットワークデータベースの開発者は、SQLエンジンを作成したあと、自分たちの製品をIDMS/Rと改名しました。また、いくつかのオブジェクト指向データベースではSQLフロントエンドを提供しているため、リレーショナルデータベースのような外観になっています。

SQLのはじまり

1970年代、IBMのDr. Edgar CoddとChris J. Dateは、データストレージ用にリレーショナルモデルを開発しました。このモデルでは、データは「リレーション(関係)」あるいは「表(テーブル)」という単純な線形構造に格納されます。これまでのデータベースモデルに比べて最も改善された点は、その「単純さ」です。リレーショナルモデルでは、ユーザーは膨大な数のナビゲーショナルデータ処理言語(DML)のコマンドを覚える必要はなく、代わりにSQLという宣言型言語を導入して、データへのアクセスやデータ処理を簡略化しています。

CoddとDateのモデルでは、表は「行」と「列」という2次元の配列で表現されます。行は「タプル」(「カップル」の韻を踏んだもの)、そして列は「属性」と呼ばれました。1つの表には必ずその表の「主キー」となる1つ以上のフィールドがあります。階層モデルやネットワークモデル(表がそれぞれポインタに接続されている)とは異なり、CoddとDateのリレーショナルデータベースモデルで

は、それぞれの表が独立しています。

既存の階層／ネットワークデータベースに比べ、リレーショナルデータベースモデルでは次の点が改善されています。

- **単純さ**——行と列で構成される「表」という概念は非常にシンプルで、簡単に理解できます。エンドユーザーには、単純なデータモデルが1つあればすみます。階層データベースやネットワークデータベースで利用する複雑なダイアグラムは、リレーショナルデータベースでは必要ありません。
- **データ独立性**——データ独立性とは、既存のプログラムに影響を与えることなくデータ構造（この場合は表）を変更できることを指します。これは、「それぞれの表が互いにハードリンクされていない」ということが大きな理由です。表には列を追加し、データベースには表を追加することができるので、新しい関係を追加する際にも、表の構造を変える必要はほとんど（あるいはまったく）ありません。リレーショナルデータベースでは、階層データベースやネットワークデータベースよりもはるかに高度なデータ独立性が実現されています。
- **宣言型データアクセス**——SQLでは、ユーザーは自分が必要なデータを指定すれば、埋め込みSQLという手続き型言語がデータの取得方法を決定してくれます。リレーショナルデータベースへのアクセスでは、ユーザーはシステムに対してデータの取得条件を指定します。すると、システムはSQL文で指定した選択条件を満たすデータを取得します。CODASYL DML言語（組み込み関数はアクセスパスの詳細を知っておく必要がある）とは違って、データベースのナビゲーションがエンドユーザーや組み込み関数の目に触れることもありません。

市場では、リレーショナルデータベースの内部ストレージ構成よりも、その宣言型データアクセス機能が興味を引きました。そして、「SQL」という語は「リレーショナルモデル」と同じ意味を持つようになったのです。

SQLのモデル

最初のSQLのモデルは、3つのカテゴリに属する機能を持つとされていました。それは「定義」「操作」「認証」です。

- **定義**——これはデータ定義言語（DDL）を指します。DDLはオブジェクトの作成（create）、削除（drop）、変更（alter）の機能を実行します。
- **操作**——これはデータ操作言語（DML）を指します。DMLは選択（select）、挿入（insert）、更新（update）、削除（delete）の機能を実行します。
- **認証**——これは権限の付与と取り消しを制御する機能を指します。

選択操作（select）

選択操作(select)では、表にある不必要な行がフィルタで除外されるので、表の行数が少なくなります。where句で条件を指定すると、不必要な行を結果セットから除外することができます(**図1-2**)。つまり、selectは問い合わせ結果の縦方向の長さを縮めるものです。

表示操作（project）

selectが行の数を減らす操作であるのと同じく、表示操作(project)では列の数が減らされます。SQLのselectで指定した列の名前は、表示対象となる列を決定します(**図1-3**)。つまり、projectは問い合わせ結果の横方向の長さを縮めるものです。

結合操作（join）

結合操作(join、**図1-4**)は、1つの列を共有する2つ以上の独立した表を関連付けるのに利用します。結合操作では2つ以上の独立した表が、共通の列値に従ってマージされます。

このシンプルな枠組みを見れば、SQLの問い合わせが「状態空間型」の問い合わせであることがわかります。つまり、問い合わせの作成者は、データにアクセスするためのナビゲーションパスのことを考える必要はありません。データへのナビゲーションパスは、SQLオプティマイザが内部的に処理してくれます。

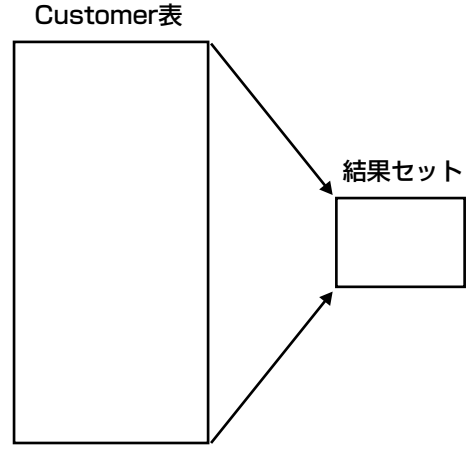


図1-2　選択操作（select）によるフィルタリングで、戻り行の数を減らす

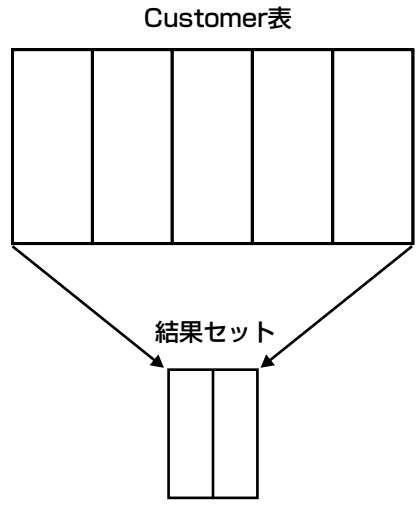


図 1-3　表示操作（project）で、戻り列の数を減らす

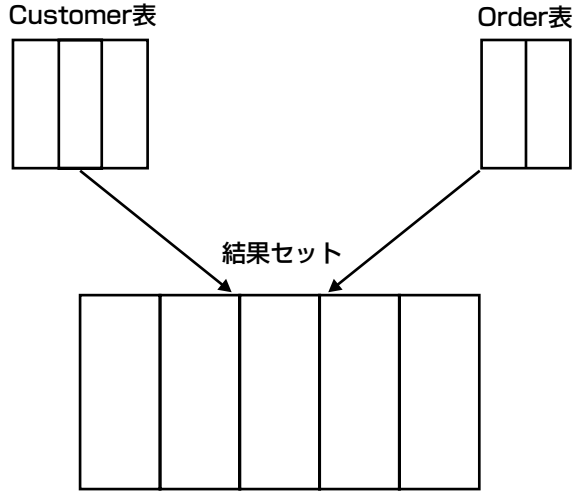


図 1-4　結合操作

SQLオプティマイザ

理想的には、SQLオプティマイザはデータ要求に応える最も効率的な手段を生み出すべきものです。問い合わせの対象となるすべての表に関する情報（たとえば行の数、1つの列にある値の分布など）をSQLオプティマイザが持っているならば、オプティマイザは必ず最適な実行計画を選択してデータにアクセスすべきです。

ところが現実には、SQLオプティマイザは人間の介入を必要とします。つまり、オブジェクトの統計情報すべてをSQLオプティマイザで確実に利用可能な状態にしておくのはDBAの役割なのです。そして、SQLの最適化という作業は途方もなく複雑なため、最適なデータアクセスのパスを確実に選び取るオプティマイザを用意するのは、どんなベンダーにとっても非常に難しいことです。したがって、「オプティマイザにどう手を加えれば最適なパスを選択できるようになるか」を理解するには、本書のような参考書が必要となります。

SQLチューニングの目標

SQLのチューニングは、Oracleのチューニングに関するあらゆる作業のうち最も時間がかかり、最も困難です。SGAのチューニングとは異なり、SQLのチューニングには終わりがありません。すべてのSQL文を個別にチューニングする必要があり、そのチューニングをいつまでも持続させるにはヒントを使ったり、Oracle8*i*の新機能であるオプティマイザのプランスタビリティを利用しなければなりません。オプティマイザのプランスタビリティに関する詳細は、Chapter 13を参照してください。

また、SQLをチューニングすると、Oracleのパフォーマンスにとって好都合な影響が迅速に現れます。Oracleオプティマイザが選択した実行計画が最適とはいえない場合でも、人間の手でチューニングすれば問い合わせの速度が3倍も高速になるというケースが多いのです。これが1日に何千回も実行される問い合わせならば、結果的にはデータベースのスループットが劇的に向上し、データベースサーバーに利用するハードウェアの寿命も延ばすことができます。SQLチューニングの目標としては、次のような項目があります。

- **すべてのSQLで、絶大なパフォーマンスを確保する**――ここでは、「SQLを記述する組み込み関数にとっての目標と、SQLチューニングの責任者にとっての目標は大きく異なる」ということに注意してください。組み込み関数にとっては、「正確な結果をなるべく迅速に返す問い合わせ」を得ることが重要ですが、この場合はパフォーマンスの視点からすると最適とはいえないSQLができてしまうこともよくあります。
- **すべてのSQLにとって効果のあるOracle統計情報と索引を作成する**――Oracle SQLオプティマイザは単独で完全に機能するものではありません。そこで、DBAにとっては「どのSQLも必ず最適な形で実行されるよう、適切な索引とオブジェクト分析テクニックを生み出す」ことがSQLチューニングの目標となります。

■ **すべてのSQLに適した実行計画を確立する**——索引を新たに追加すると、たくさんのSQL文に変更が加わる場合があります。SQLチューニングの主要な目標は、チューニングの変更を永続的なものにすることです。どんなSQL文であっても、最適な実行計画は1つしかありません。そして、選ぶべき実行計画が判明したあとは、DBAは問い合わせにヒントを追加したり、オプティマイザのプランスタビリティを使ってその実行計画を永続化することが目標となります。

本書ではこれらの目標を中心的なテーマとし、本書全体を通じて説明していきます。

Oracleチューニングの一環としてのSQLチューニング

SQLのチューニングは、Oracleのチューニングとは切っても切れない関係にあり、パフォーマンス面で即座に影響が現れます。しかし、Oracleであらかじめ済ませておくべきチューニング処理を行わないままSQLのチューニングに取りかかりたいという衝動も、非常によく起こります。

Oracleのチューニングには、厳密な順序があります。Oracleデータベースはどれも階層で構成され、グローバルコンポーネントはサブコンポーネントに影響を与えます(**図1-5**)。

この階層は非常に重要です。というのは、高レベルで行ったチューニングに何らかの不手際があると、その下にあるレベルのアクティビティがすべて影響を受けるためです。Oracleチューニ

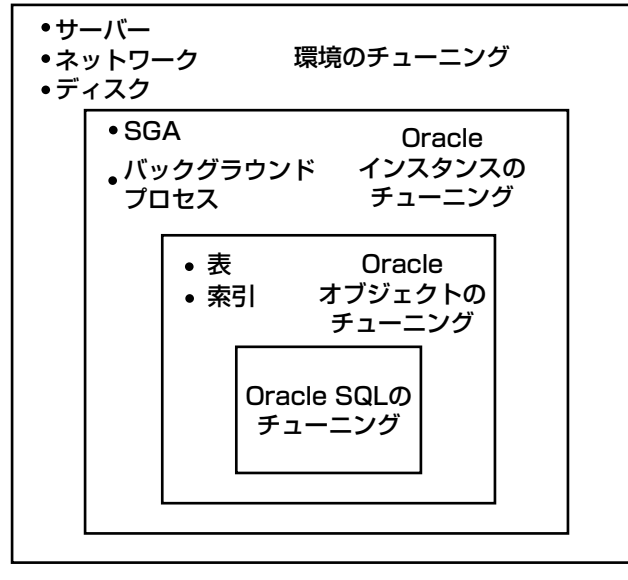


図1-5　Oracleチューニングの階層

ングの階層を詳しく分類すると、次のようになります。

- **環境のチューニング**——データベースサーバー、ネットワーク、ディスクI/Oサブシステムなどのチューニングです。これらの要素をチューニングしていないうちは、Oracleデータベースをチューニングすべきではありません。
- **データベースサーバーのチューニング**——Oracleデータベースサーバーのチューニングは、Oracleチューニングの出発点です。データベースサーバーのRAMやCPU速度が不足するような事態になると、Oracleをどれだけチューニングしても問題は解決しません。
- **ネットワークのチューニング**——ネットワークの基本部分は、ネットワークプロトコルのレベルでパケット転送の問題が発生しないようにチューニングしておかなくてはなりません。Oracleネットワーク通信のチューニングについては、拙書『Oracle9*i* ハイパフォーマンスチューニング STATSPACK編』(翔泳社より刊行予定)のChapter 7を参照してください。
- **ディスクのチューニング**——ディスクI/Oサブシステムのチューニングは、Oracleをチューニングする前に必ずやっておくべき作業です。ディスクI/Oのボトルネックやディスクアクセスの問題に対処していないうちにOracleのチューニングに取りかかっても、意味がありません。
- **インスタンスのチューニング**——外部環境のチューニングが完了した次の段階は、Oracleインスタンスのチューニングです。ここにはSGAメモリやOracleバックグラウンドプロセスの挙動に関するチューニングが含まれます。SQLのチューニングでは、この段階でデータベース内のすべてのSQLに利用する、デフォルトの*optimizer_mode*を設定します。Oracleの*optimizer_mode*パラメータに関する詳細は、Chapter 14および15を参照してください。
- **オブジェクトのチューニング**——インスタンスのチューニングが完了したあとは、Oracleオブジェクトを1つ1つチューニングし、最適のパフォーマンスを引き出す必要があります。この段階では、記憶領域パラメータ(特に、I/Oに影響のあるもの)をすべて適切に設定します。*pctfree*、*pctused*、*freelists*の設定内容はどれも、SQLのパフォーマンスに大きく影響します。
- **SQLのチューニング**——全般的なチューニングがすべて完了したら、ようやくSQLのチューニングです。この段階では頻繁に使われるSQL文を見分け、それらの文をチューニングし、最適な実行計画を確実に保つ必要があります。

データベースの設計とSQLのパフォーマンス

ここまでは、SQLの速度に関する最も重要な側面について、意図的に説明を省いてきました。それは、最初のデータ構造の設計です。**図1-6**は、その概要を示しています。Oracleの表におけるデータ正規化と冗長化の度合いをどう計画するかによって、問い合わせの速度は大きく左右されます。本書でも説明しますが、冗長な列を表に追加(冗長化)すれば、SQLで効率の悪い結合を実行する必要もなく、パフォーマンスを向上することができます。

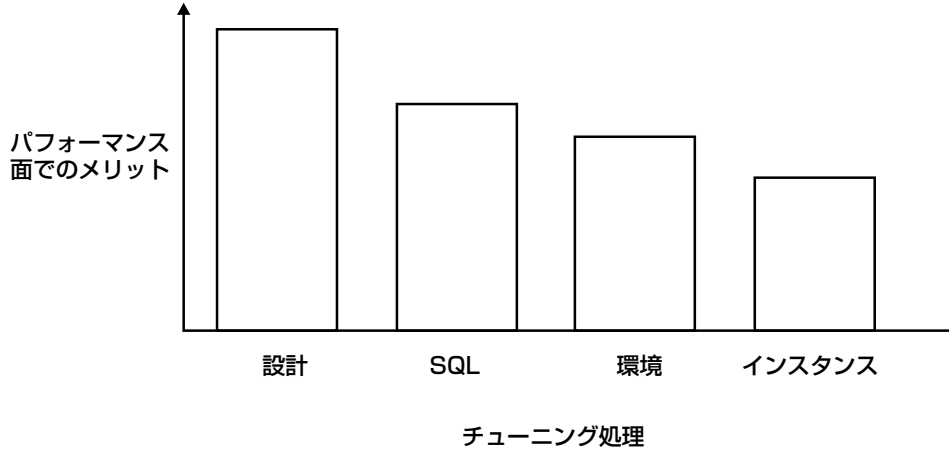


図1-6　チューニングの処理別に見た、パフォーマンス面での相対的なメリット

このことに関して説明を省いてきた理由は、アプリケーションがひとたび本番環境として稼動したあとでは、DBAが表の設計を変更する可能性がきわめて低いためです。しかし、別の章では列の複製というテクニックについて取り上げ、ある表から別の表にデータ列を複製(**図1-7**)してSQLの表結合を避ける方法と、複製したデータ項目をトリガーによって最新に保つ方法を説明します。

次は、SQLのチューニングに取りかかる際にDBAが直面する問題について見てみましょう。

冗長化した表

Product_ID	Product_Name	Order_NBR	Customer_ID
661	Pen	123	Jones
427	Pencil	123	Jones
873	Paper	123	Jones
661	Pen	456	Smith

図1-7　列の複製によるパフォーマンスの向上

SQLチューニングの障害となる問題

Oracle SQLのチューニングは、Oracleのチューニングにおいて最も時間がかかり、フラストレーションがつのり、いらいらする部分です。SQLのチューニングが非常にわずらわしい作業となる要因を、いくつか挙げてみましょう。

- **問題の原因となっているSQL文の特定**——本書の後半では、Oracleライブラリキャッシュからを探し出す方法を説明します。ただし、Oracle8の初期リリース(オプティマイザのプランスタビリティをサポートしていないもの)では、チューニング内容を永続化するためにSQLのソースコードの場所を見つける必要があります。ご存知のとおり、SQLのソースコードはPL/SQLやCプログラム、クライアントサイドのVisual Basicコードなど、さまざまな場所に存在しています。

- **経営陣からの反発**——SQLのチューニングには時間と費用がかかり、そういった時間への投資を経営陣が渋ることも珍しくはありません。たいていの場合、DBAは費用便益分析を行って、データベースのSQLをすべてチューニングすればハードウェアリソースの節約につながり、費用に見合う作業となることを示さなければなりません。

- **アドホック(不定形)SQLジェネレータのチューニング**——SAPアプリケーションなどの製品は、SAP ABAPプログラムの内部でOracle SQLを動的に生成します。そのため、それらSQLのソースコードを変更できないという場合がよくあります。

- **SQLプログラマからの反発**——自分の作成したSQL文が最適ではなかったと、すんなり認めてくれるプログラマはあまりいません。この場合、DBAにとっては、特定の文を書き換えてパフォーマンスを向上させるという決断を下すのが非常に困難になります。

- **再利用できないSQL文のチューニング**——サードパーティ製アプリケーションの多くで生成されるSQL文には、リテラル値(「select * from customer where name = 'JONES'」など)が埋め込まれています。こういった場合、ライブラリは再利用できない無数のSQL文で埋め尽くされ、構文が同じであるにもかかわらず、ホスト変数(select * from customer where name = :var1;」など)がないため再利用できないSQL文だらけになります。この場合は*cursor_sharing*を実装したり、場合によってはライブラリキャッシュのサイズを縮小したり、*alter system flush shared_pool*コマンドでフラッシュする必要があります。

- **限界収益点の低下**——DBAは利用頻度の高いSQL文を識別してチューニングします。それゆえに、チューニングの対象となるSQL文を見つけるのは、さらに難しくなります。たとえば、あまり実行されないSQL文の中にも、チューニングによって大きなメリットが得られるものは多いかもしれません。しかし、それらの文はライブラリキャッシュの内部ではあまり目立たないため、見つけ出すのが困難です。ということは、「チューニングによって得られるメリット(限界収益)が、文を見つけ出す労力に見合わなくなる点」が存在することになります(**図1-8**)。

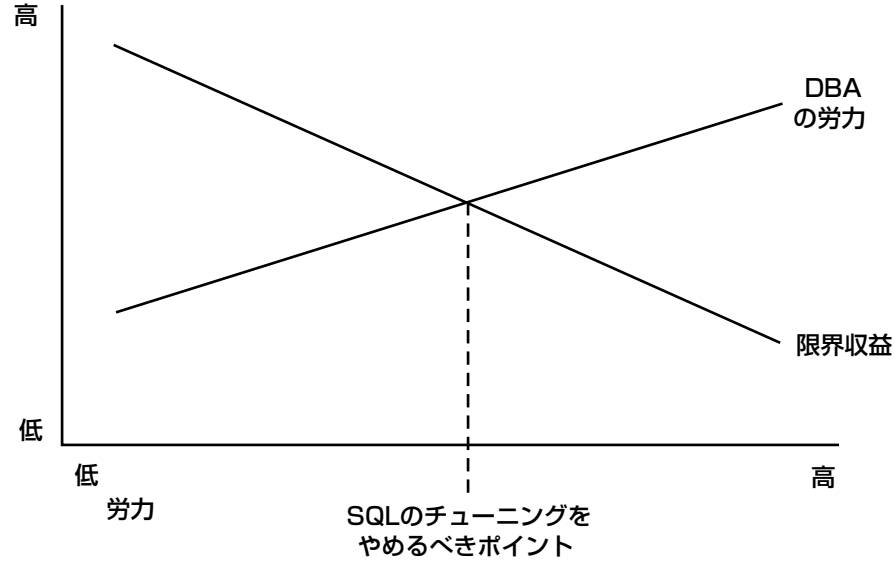


図1-8　SQLチューニングに関する限界収益点

こういった難題があるにもかかわらず、DBAはOracle SQLをチューニングすることによって、経営陣とプログラマのどちらからも尊敬される可能性があります。洗練された企業では、プログラマは自分たちのSQL文と現在の実行計画をDBAに送って検証してもらってから、そのSQLを本番環境に配置するようにしています。

次は、SQLのチューニングで必要となる手順を大まかに説明します。

SQLのチューニング手順

サーバー、ネットワーク、インスタンスやオブジェクトなど、前もって行う必須事項のチューニングが完了したら、いよいよSQLのチューニングです。このテーマについては以後も詳細に説明しますが、ここで一度、SQLのチューニング手順を簡単に述べておきます。

SQLのチューニングは、繰り返し行う作業であることを忘れないでください。DBAは、次のような課題に取り組まなければなりません。

- **頻繁に実行されるSQL文の特定**——SQLチューニングの第1歩は、利用頻度の高いSQL文を特定することです。この段階では、STATSPACKを使ったり、ライブラリキャッシュからSQL文を探し出します。
- **SQL文のチューニング**——SQL文のチューニングでは、実行計画の生成や、他の実行計画の評価を行います。そのためには、次のような方法があります。

- **索引の追加**——索引(特に、ビットマップ索引とファンクション索引)を追加すると、不必要なフルテーブルスキャンを取り除くことができます。
- **オプティマイザのモード変更**——オプティマイザのモードはrule、all_rows、first_rowsのいずれかに変更することができます。
- **ヒントの追加**——ヒントを追加すると、実行計画を強制的に変更することができます。

- **チューニング内容の永続化**——SQL文をチューニングしたあとは、それらの変更を永続化するためにSQLのソースコードを見つけて変更するか、オプティマイザのプランスタビリティを利用する必要があります。

では、これまでに挙げた手順を、それぞれ詳しく見ていきましょう。

頻繁に実行されるSQL文の特定

最初のステップでは、最もよく実行されるSQL文を特定します。SQLはさまざまなソース(Pro*CプログラムやVisual Basicのコードなど)からOracleに入力されるため、DBAは現在のライブラリキャッシュにあるSQLを取り出すためのアプローチを用意しておく必要があります。チューニングの対象となるSQLを見つけ出すアプローチは、次の2通りです。

- **STATSPACKの利用**——このアプローチでは、*stats$sql_summary*表を使ってSQL文を捕捉します。
- **ライブラリキャッシュの検索**——このアプローチでは、現在ライブラリキャッシュの中にあるすべてのSQLを解釈するユーティリティを使います。

では、それぞれのアプローチを詳しく見てみましょう。

STATSPACKによるSQLの捕捉

STATSPACKを使うと、STATSPACKに関連する表の中で、特に*stats$sql_summary*表にたくさんの行が格納されていることがわかります。アクティビティが活発でしきい値を非常に低く設定してあるデータベースでは、STATSPACKでスナップショットを1回要求するたびに、数百もの行が*stats$sql_summary*表に追加されることもあります。そのため、もうSQLのチューニングに利用しなくなった不要な行を、*stats$sql_summary*表から削除することが非常に重要です。ただし、*stats$sql_summary*表に格納されたSQLを、*stats$statspack_parameter*表に格納された次のしきい値を使ってフィルタをかけることもできます。

- ***executions_th***——SQL文の実行回数です(デフォルトは100)。
- ***disk_reads_th***——SQL文によって実行されるディスク読み込みの回数です(デフォルトは1,000)。

- ***parse_calls_th***——SQL文によって実行される解析コールの実行回数です（デフォルトは1,000）。
- ***buffer_gets_th***——SQL文によって実行されるバッファ取得回数です（デフォルトは10,000）。

SQL文は、いま挙げたどのしきい値を上回った場合であっても、*stats$sql_summary*表に組み込まれるということを覚えておいてください。多くのDBAは1時間単位でSTATSPACKのサンプルを取得するようスケジューリングし、スナップショットを収集した時点でライブラリキャッシュの内部に存在しているすべてのSQLのサンプルを取得できるようにしています。

STATPACKを使ってSQL文を収集する場合の大きな欠点は、SQLのチューニングが進むにつれてしきい値を変更する必要が生じることです。SQL文を識別してチューニングするときは、より使用頻度の低いSQL文も引き続きチューニングできるよう、しきい値を下げる必要があります。こうすれば、より多くの行が*stats$sql_summary*表に格納され、STATSPACK用の表領域がすばやく充填されるようになります。たいていのDBAは、それらのSQL文のチューニングが終わったあと、*stats$sql_summary*行を定期的に削除するようにしています。

ライブラリキャッシュの検索

SQLチューニングのアプローチとして一般的なもう1つの方法は、SQL文をライブラリキャッシュから取り出すというものです。このために利用するテクニックとしては、次のようなものがあります。

- **サードパーティのツール**——サードパーティ製のGUIツールを使えば、ライブラリキャッシュ内部のSQLをすばやく表示し、最も実行頻度が高く、リソースを消費する文を取り出すことができます。このようなツールとしては、OracleのEnterprise Manager Performance Packをはじめ、SQL*Lab、Q Diagnositc Center、SQL Expertなど、さまざまなものがあります。
- **SQL*Plusスクリプト**——本書ではaccess.sqlという、現在ライブラリキャッシュにあるすべてのSQLを取り出して説明するスクリプトについても説明します。

チューニング対象のSQL文を特定できたあとは、それぞれのSQL文を個別にチューニングします。

SQL文のチューニング

SQL文のチューニングには、いくつかの処理が必要です。ここから、SQLのチューニングは全体的なレベルから個別の文レベルに移っていきます。SQLチューニングの最終目標は、実行計画がSQL文にとって最適になっているかどうかを確認することです。SQLの実行速度を検証するには、通常はSQL*Plusで*set timing on*コマンドを使い、実際に問い合わせの速度を計測します。

- **オプティマイザのモード変更**——オプティマイザのモードをrule、all_rows、first_rowsのいずれかに変更することにより、それぞれのモードに対応する実行計画が生成されます。生成された各実行計画の時間を測定し、最も実行時間が短い実行計画を決定します。
- **索引の追加**——索引(特に、ビットマップ索引とファンクション索引)を追加すると、不必要なフルテーブルスキャンを取り除くことができます。しかし、索引を1つ追加すると、他の多くのSQL文の実行計画が変更される可能性があることに注意してください。索引を1つ追加した結果、他のSQL文の実行速度が変わっただけだったということも、珍しくありません。
- **ヒントの追加**——変更を実行計画に強制的に適用するには、選択した文にヒントを追加します。OracleにはSQLの実行計画を変更するためのヒントがたくさん用意されています。ヒントをベースにSQLをチューニングする方法については、Chapter 12で詳しく説明します。

チューニング内容の永続化

チューニングが完了したあとは、そのチューニングによる変更を永続化することが非常に重要です。このためには、いくつかの方法があります。SQLチューニングの変更内容を永続化することは、*optimizer_mode*初期化パラメータを変更したり、索引を追加するといった全体にかかわる変更を加え、たくさんのSQL文の実行計画が変更される可能性のある環境では特に重要です。

オプティマイザのスタビリティ

Oracle8*i*ではOUTLINEという新しいパッケージが用意されています。このパッケージを使うと、どのSQL文にでもすぐに実行できる実行計画を格納することができます。OUTLINEには、次のような機能があります。

- Oracleはストアドアウトラインをすばやく取り出して実行するようになるため、場合によっては文の解析と実行時間が短縮されます。
- ソースコードを特定しなくても、SQL文のチューニングを簡単に永続化することができます。
- サードパーティ製品(SAPやPeoplesoftなど)のSQLであっても、ソースコードに手を入れずにチューニングすることができます。

オプティマイザのプランスタビリティを利用すれば、同じSQL文に対して、「表の再分析」「データの追加や削除」「表の列、制約、索引」「システム構成の変更」のような変更をデータベースに加た場合でも——さらには、オプティマイザを新しいバージョンにアップグレードした場合でも——同じ実行計画を維持することができます。

オプティマイザのプランスタビリティを使うには、$ORACLE_HOME/rdbms/adminディレクト

リからdbmsol.sqlというスクリプトを実行する必要があります。このスクリプトを実行すると、OUTLNというOracleユーザーが(DBA権限付きで)新規に作成され、OUTLN_PKGというパッケージがインストールされて、ストアドアウトラインの管理用プロシージャが用意されます。

Oracleには、ストアドアウトラインを作成するCREATE OUTLINEという文が用意されています。ストアドアウトラインは一連の属性で構成され、オプティマイザはこれを利用して実行計画を作成します。初期化パラメータを*create_stored_outlines=true*と設定すると、ストアドアウトラインを自動的に作成することもできます。ストアドアウトラインの使い方に関する詳細は、Oracle8*i*の製品マニュアルと本書のChapter 13を参照してください。

SQLソースの変更

Oracle8*i*よりも古い環境では、SQLのソースコードを特定しなければチューニングの変更を永続化することができません。SQLがクライアントサイドのアプリケーションで送出されるアプリケーションや、ODBCを使ってOracleと通信するシステム、そして動的SQLを生成するシステムでは、このことがやっかいな問題となる場合があります。

現実的には、アプリケーションからすべてのSQLを取り除くことを強くお勧めします。このためには通常、すべてのSQLをストアドプロシージャの中に配置し、そのストアドプロシージャをOracleのパッケージ内部に配置します。こうすることによって、SQLソースはOracleデータディクショナリに格納されるため、見つけやすくなります。また、リモートアプリケーションがどれもポータブルになる(可搬性を得られる)というメリットも生じます。これは、Oracleに対する呼び出しすべてが、関数やストアドプロシージャコールに格納されるためです。

SQLのチューニングについて本格的に解説する前に、本書で使うツールをいくつか概観します。誰の役にも立てるよう、本書ではSQLを分析する際に、Oracle標準のユーティリティとスクリプトを利用します。こうしておけば、高価なサードパーティ製品を使う必要もなく、本書を手にした誰もがすべてのSQLを分析し、チューニングすることができます。

SQLチューニングのガイドライン

SQLをチューニングする際の基本的なルールは、ごく簡単なものです。

1. どんなSQL文であっても、最適な実行計画は1つしかありません。その実行計画を見つけるのがDBAの仕事です。
2. SQLは外部プログラムからOracleデータベースに送られてくるため、DBAはライブラリキャッシュを継続的に監視し、チューニングされていないSQLがないか調べる必要があります。
3. うまくチューニングされたSQL文は、実行時間も最速となります。通常、このためには表のI/Oが最少となるような実行計画を作成します。

SQLチューニングの目標

Oracle SQLのチューニングというテーマは、さまざまな要素が複雑に絡み合っています。そこで、ここではSQLチューニングの目標を概説し、詳細についてはあとの章で説明します。

SQLチューニングの目標という言葉が出たところで、いくつかの目標について個別に説明していくことにしましょう。システムのパフォーマンスを向上させるためにDBAが守るべきガイドラインは、いくつかあります。SQLチューニングの目標は、難しいものではありません。

- **大きな表に対する不必要なフルテーブルスキャンをなくす**——不必要なフルテーブルスキャンが実行されると、余計なI/Oが大量に発生し、データベース全体のパフォーマンスが低下します。チューニングのエキスパートは、まず「問い合わせによって表の行がいくつ返されるか」という視点からSQLを検証していきます。ある問い合わせを発行した結果、順序付けで整理された表から返される行の割合が全体の40パーセント以下、あるいは順序付けされていない表から返される行の割合が全体の7パーセント以下ならば、その問い合わせはフルテーブルスキャンの代わりに索引を使うようチューニングする余地があります。不必要なフルテーブルスキャンを解消する最も一般的な対処法は、索引を追加することです。表には標準的なBツリー索引を追加することができ、ビットマップ索引とファンクション索引でもフルテーブルスキャンを解消することができます。フルテーブルスキャンをなくす場合は、索引スキャンとフルテーブルスキャンのI/Oコストを入念に比較調査して、マルチブロックREADとパラレル問い合わせの可能性について予測し、その結果に基づいて決断を下す必要があります。場合によっては、索引のヒントを1つSQL文に追加すれば、不必要なフルテーブルスキャンを索引スキャンに変換できることがあります。

- **小さな表のフルテーブルスキャンをキャッシュする**——フルテーブルスキャンが最も高速なアクセス手段である場合は、行を格納するための専用のデータバッファを確保する必要があります。Oracle7では「*alter table xxx cache*」という文を発行します。Oracle8以降では、小さな表はKEEPプールに強制的に格納すればキャッシュすることができます。

- **最適な索引の使用法を検証する**——問い合わせの速度を向上するには、この手順が特に重要です。Oracleではさまざまな索引があるため、チューニングの専門家は索引を1つ1つ調べ、適切な索引が使われているかどうかを確認しなくてはなりません。これにはビットマップ索引やファンクション索引も含まれます。

- **最適な結合テクニックを検証する**——問い合わせによっては、ネステッドループ結合を利用したほうが高速に実行できる場合もあれば、ハッシュ結合のほうが高速になる場合もあります。

いま挙げた目標は、一見すると簡単そうです。しかし、SQLチューニングの90パーセントはこういった作業で成り立っています。また、これらの作業を行うために、Oracle SQLの本質を徹底的に理解する必要はありません。次は、SQL文のチューニングに利用できるツールをひととおり紹介します。

SQL Tuning Toolkit

本書で使用するスクリプトは、http://www.shoeisha.com/down/ からダウンロードすることができます。この章では、次の基本的なスクリプトについて説明します。

- **access.sql**——このスクリプトはライブラリキャッシュのSQLをすべて挙げ、現在ライブラリキャッシュの中にある表のアクセスの種類や名前を示した一連のレポートを作成します。
- **access_report.sql**——SQLのアクティビティに関するさまざまなサマリーを表示するレポートのセットです。ここにはフルテーブルスキャン、全索引スキャン、索引レンジスキャン、そしてROWIDによるアクセスのレポートも含められます。
- **get_sql.sql**——ライブラリキャッシュ内の、条件に一致するすべてのSQLを一覧表示する簡単なスクリプトです。
- **plan.sql**——任意のSQL文の実行計画を表示する汎用スクリプトです。

これらのスクリプトは以後も本書でたびたび登場するため、ここでひととおり把握しておきましょう。

ライブラリキャッシュのSQLをレポートする

DBAはライブラリキャッシュの内部にSQLが新たに現れていないか、常に目を光らせていなければなりません。ここでは、ライブラリキャッシュにあるすべてのSQL文にOracleで*explain plan*文を実行し、実行計画をすべて分析して、すべての表と索引に対するアクセス方法のレポートを作成するテクニックを説明します。

このテクニックのどこが、また、レポートから提供される情報のどこが有益なのかは、一見しただけでは完全にわからないかもしれません。しかし、データベースのライブラリキャッシュが大きい場合は、表と索引の内部動作について重要な手がかりを得ることができます。また、レポートから得られた情報を利用すれば、どのデータベースオブジェクトを調整すべきかも明らかになります。このレポートは、次のようなデータベースアクティビティにとって非常に有益です。

- **利用頻度の高い表や索引を識別する**——データベースがどの表に最もよくアクセスしているかを調べます。
- **キャッシュ対象となる表を識別する**——KEEPプール（Oracle8）への配置やCACHEオプション（Oracle7）の利用対象となる、アクセス頻度の高い小さな表をすばやく見つけ出すことができます。このテクニックを拡張すれば、ブロック数やアクセス回数に関して一定の条件を満たす表を自動的にキャッシュすることもできます。筆者は、ブロック数が200未満の表が100回以上フルテーブルスキャンされた場合、それらをすべて自動的にキャッシュするようにしています。

■ **行の順序変更対象となる表を識別する**——頻繁に索引レンジスキャンが実行される大きな表を特定し、行の順序を変更してI/Oを減らすことができます。

■ **使われていない索引を削除する**——使われていない索引が占有している領域を再生することができます。さまざまな調査結果によれば、Oracleデータベースでは、使用可能な索引のうち、全体の4分の1以上が使われていないか、もともとの意図どおりには使われていないことがわかっています。

■ **索引を新規追加してフルテーブルスキャンを停止する**——フルテーブルスキャンをすばやく見つけることができるため、新しい索引を表に追加して速度を向上することができます。

access.sqlスクリプトの実行手順は、次のとおりです。

1. access.sql、access_report.sql、plan.sqlの各スクリプトをダウンロードします。
2. 表のスキーマ所有者に対し、次の文を発行します。

```
grant select on v_$sqltext to schema_owner;
grant select on v_$sqlarea to schema_owner;
grant select on v_$session to schema_owner;
grant select on v_$mystat to schema_owner;
```

3. SQL*Plusを起動してスキーマ所有者として接続し、access.sqlを実行します。

表の名前を限定していないSQL文を解釈するには、スキーマ所有者として接続する必要があります。また、ここで統計情報を得られるのは、所有者のライブラリキャッシュ内に存在するSQL文だけであることにも注意してください。アクティビティの激しいデータベースでは、このスクリプトを数回実行してもよいでしょう。大部分のOracleデータベースでは、10分以内で実行することができます。

access.sqlのレポート

さきに述べたとおり、access.sqlスクリプトはライブラリキャッシュ内のSQLをすべて取得し、それをsqltempという表に格納します。この表からそれらすべてのSQLが解釈され、plan_tableという単一の計画表に格納されます。そして、このplan_table表に向けて問い合わせが実行されます。

この時点で、次に挙げるリストと同じようなレポートが表示されます。まず、これまでのテクニックを使って得た出力に目を通し、それからレポートを作成する方法を解説します。わかりやすいよう、レポートは複数のセクションに分割します。最初のセクションにはライブラリキャッシュの中にあるSQL文の総数と、解釈できなかったSQL文の総数が示されています。表の所有者名が示されていない文は解釈できていません。解釈できなかった文の総数が多い場合は、スクリプトの実行時に適切なスキーマ所有者として接続していない可能性があります。

access.sqlで作成されたレポート

```
PL/SQL procedure successfully completed.

Mon Jan 29
page 1
                        Total SQL found in library cache        23907

Mon Jan 29
page 1
                     Total SQL that could not be explained      1065
```

フルテーブルスキャンのレポート

これはaccess_report.sqlで作成される最初の、そして最も重要なレポートです。次のリストにはフルテーブルスキャンを実行したすべてのSQL文と、その実行回数が表示されています。ここで、「C」と「K」の列にも注目してください。「C」はOracle7の表がキャッシュされているかどうかを示し、「K」はOracle8の表がKEEPプールに割り当てられているかどうかを示しています。さきに述べたとおり、フルテーブルスキャンの実行される小さな表はKEEPプールに格納したほうがよいでしょう。

```
Mon Jan 29
page 1
                          full table scans and counts


OWNER           NAME                          NUM_ROWS C K    BLOCKS  NBR_FTS
--------------- ------------------------- ------------ - - -------- --------
SYS             DUAL                                 N          2   97,237
SYSTEM          SQLPLUS_PRODUCT_PROFILE              N K        2   16,178
DONALD          PAGE                       3,450,209 N    932,120    9,999
DONALD          RWU_PAGE                         434 N          8    7,355
DONALD          PAGE_IMAGE                    18,067 N      1,104    5,368
DONALD          SUBSCRIPTION                     476 N K      192    2,087
DONALD          PRINT_PAGE_RANGE                  10 N K       32      874
ARSD            JANET_BOOKS                       20 N         8       64
PERFSTAT        STATS$TAB_STATS                      N         65       10
```

このレポートを見ると、フルテーブルスキャンを実行している非常に大きな表(DONALD.PAGEなど)がいくつかあることがわかります。ブロック数が200未満で適切なフルテーブルスキャンを実行している表は、KEEPプールに格納することができます。より大きな表のフルテーブルスキャンについても詳しく調べ、大きな表に対する適切なフルテーブルスキャンは*alter table parallel degree nn*コマンドでパラレル化する必要があります。

Oracleデータベースでは、問い合わせを索引で処理できない場合に大きな表のフルテーブルスキャンが実行されます。フルテーブルスキャンの頻度が高すぎる大きな表を識別できれば、適切に対応して索引を追加することができます。これが特に重要となるのは、Oracle7からOracle8に移

行する場合です。その理由は、Oracle8では組み込み関数付きの索引が用意されているためです。フルテーブルスキャンが実行されるもう1つの原因としては、索引レンジスキャンよりもフルテーブルスキャンのほうが高速であるとコストベースのオプティマイザが判断したケースが挙げられます。一般に、この場合のフルテーブルスキャンは小さな表で発生します(これらの表はOracle7ではキャッシュし、Oracle8ではKEEPプールに格納するのが理想的)。このフルテーブルスキャンのレポートは、次に挙げる2種類のSQLチューニングにとって不可欠です。

小さな表のフルテーブルスキャンの場合は、*alter table xxx cache*コマンドを使って表をキャッシュします。これによって、最近使われた表の行はデータバッファの終端に格納されるため、その表に対するディスクI/Oが減少します(Oracle8では*alter table xxx storage* (*buffer_pool keep*)コマンドを発行し、キャッシュしたい表をKEEPプールに格納する)。

大きな表のフルテーブルスキャンの場合はSQL文を詳細に調査して、索引を使えばフルテーブルスキャンをなくせるかどうかを調べます。繰り返しますが、SQL文のオリジナルのソースは、すべてSQLTEMP表の中にあります。SQL文を個別に見つけ出し、解釈する方法については、この次のセクションで説明します。

次は、索引の使用状況に関するレポートです。索引のレポートは、Oracleチューニングにおける次の領域で不可欠です。

- **索引の使用状況**——索引のレポートを利用して、新しい索引がアプリケーションから実際に使われるよう保証します。こうしておけば、DBAは索引を新規に作成したあとも、それが実際に利用されているという実証的な証拠を得ることができます。
- **行の順序変更**——索引のレポートを利用して、行の順序を変更したほうがよさそうな表を見つけ出します。索引レンジスキャンの回数が多い表は、行の順序を変更して索引と同じ順序にすればメリットが得られます。行の長さによっては、順序変更を行えばパフォーマンスが10倍向上する可能性もあります。行の順序変更に関する詳細は、Chapter 10を参照してください。

では次に、索引レンジスキャンのレポートを見てみましょう。

索引レンジスキャンのレポート

次のリストは、索引レンジスキャンのレポートです。Oracleで最も一般的な索引アクセス方法は、索引レンジスキャンです。索引レンジスキャンはSQL文に制限句が含まれ、表の索引となる連続した範囲の値が要求されている場合に利用されます。

```
Mon Jan 29
page 1
                          Index range scans and counts

OWNER     TABLE_NAME           INDEX_NAME             TBL_BLOCKS    NBR_SCANS
--------- -------------------- -------------------- ------------ ------------
DONALD    ANNI_HIGHLIGHT       HL_PAGE_USER_IN_IDX            16        7,975
DONALD    ANNI_STICKY          ST_PAGE_USER_IN_IDX             8        7,296
```

```
DONALD    PAGEER                ISBN_SEQ_IDX                120        3,859
DONALD    TOC_ENTRY             ISBN_TOC_SEQ_IDX             40        2,830
DONALD    PRINT_HISTORY         PH_KEY_IDX                   32        1,836
DONALD    SUBSCRIPTION          SUBSC_ISBN_USER_IDX         192          210
ARSD      JANET_BOOK_RANGES     ROV_BK_RNG_BOOK_ID_I          8          170
PERFSTAT  STATS$SYSSTAT         STATS$SYSSTAT_PK            845           32
12 rows selected.
```

索引一意スキャンのレポート

次のレポートでは、索引一意スキャンを一覧表示しています。索引一意スキャンは、Oracleデータベースエンジンが索引を1つ使い、表から特定の行を取得する際に実行されます。一般に、Oracleデータベースでは、結合を実行するときにこのような「プローブ」アクセスを利用してもう一方の表を調べ、駆動表からの結合キーを検出します。このレポートは、表の中で(一定の範囲に含まれた行のフェッチに利用する索引ではなく)明確な行の識別に利用する索引を見つけ出す場合にも有効です。

```
Mon Jan 29
page 1
                          Index unique scans and counts

OWNER     TABLE_NAME            INDEX_NAME               NBR_SCANS
--------- -------------------- -------------------- ------------
DONALD    BOOK                  BOOK_ISBN                 44,606
DONALD    PAGEER                ISBN_SEQ_IDX              39,973
DONALD    BOOK                  BOOK_UNIQUE_ID             6,450
DONALD    ANNI_EAR              DE_PAGE_USER_IDX           5,339
DONALD    TOC_ENTRY             ISBN_TOC_SEQ_IDX           5,186
DONALD    PRINT_PERMISSIONS     PP_KEY_IDX                 1,836
DONALD    RDRUSER               USER_UNIQUE_ID_IDX         1,065
DONALD    CURRENT_LOGONS        USER_LOGONS_UNIQUE_I         637
ARSD      JANET_BOOKS           BOOKS_BOOK_ID_PK              54
DONALD    ERROR_MESSAGE         ERROR_MSG_IDX                 48
```

全索引スキャンのレポート

次のレポートには、全索引スキャンがすべて表示されています。Oracleオプティマイザは一時表領域での大規模ソートの代わりに全索引スキャンを実行することがあります。一般に、全索引スキャンはorder by句のあるSQLで発生します。

```
Mon Jan 29
page 1
                          Index full scans and counts

OWNER     TABLE_NAME            INDEX_NAME               NBR_SCANS
--------- -------------------- -------------------- ------------
DONALD    BOOK                  BOOK_ISBN                  2,295
DONALD    PAGER                 ISBN_SEQ_IDX                 744
```

通常は全索引スキャンのほうがディスクソートよりも高速ですが、いくつかの*init.ora*パラメータを使えば、全索引スキャンをより高速化することができます。たとえば、Oracle7の「*V77_plans_enabled*」というパラメータです(Oracle8では「*fast_full_scan_enabled*」に名前が変更されている)。高速全索引スキャンは、問い合わせに必要なすべての列が索引に格納されている場合に、フルテーブルスキャンに代わる手段として利用することができます。高速全索引スキャンでは*db_file_multiblock_read_count*パラメータで定義したマルチブロックI/Oを利用するため、通常の全索引スキャンよりも高速です。また、パラレル問い合わせを実行するためのパラレルヒントも、フルテーブルスキャンと同じ形で指定することができます。

一般に、Oracleデータベースエンジンでは全索引スキャンを使ってソートを回避しています。たとえば、*customer*という表にcust_nbr列の索引があるとします。この場合、データベースは「*select * from customer order by cust_nbr;;*」というSQLコマンドを、次の2通りの方法で処理することができます。

- **フルテーブルスキャンを実行し、結果セットをソート**——*db_file_multiblock_read_count*というinit.oraパラメータを設定するか、パラレルヒントを使って表アクセスをパラレル化すれば、フルテーブルスキャンは非常に高速に実行することができます。ただし、実行後に結果セットを一時表領域でソートする必要があります。
- **顧客番号順に行を取得**——この場合は索引を経由して行を読み込むため、ソートの必要はありません。

access.sqlで作成したレポートに関する制限

これらのレポートを作成するテクニックは、見かけほど完璧ではありません。実行計画を取得するには「生の」SQL文を解釈しなければならないのですが、「生の」SQL文だけでは表の所有者が誰なのかがわからない場合があります。生のSQLでは、表の名前が常に表の所有者名と関連付けられるわけではない点が問題となります。すべてのSQL文が完全に解釈できるよう、たいていのDBAはスキーマ所有者としてOracleに接続し、レポートを実行します。

ここから先の機能強化としては、どのOracleデータベースユーザーであってもレポートを実行できるよう、SQL文を解釈する直前に次の(マニュアルに記載されていない)コマンドを発行することが挙げられます。

```
Alter session set current_schema = 'table_owner';
```

こうすれば、SQL文を解釈する直前にスキーマの所有者を変更できるようになります。

これで、SQLのレポート機能に関する説明はひととおり説明しました。次は、個々のSQL文を抽出し、解釈する方法を見ていきます。

まとめ

この章では、Oracle SQLのチューニングによって得られるメリットと、チューニングの際に問題となる事柄を紹介しました。また、以後の章で利用する、SQL文を簡単にチューニングするツールについても概要を説明しました。ここから先は、データベースが最適なレベルで稼動する環境を確立するさまざまなテクニックを詳しく説明します。

Oracle SQLのチューニングは、Oracleデータベースチューニングの中でも、最も見返りの多い作業です。SQL文をチューニングして、以前は2時間かかっていた実行時間が20秒にまで短縮されたときの気分は、たとえようがありません。適切なアプローチと根気強さがあれば、DBAは誰からも賞賛され、データベース全体のパフォーマンスを劇的に改善することができます。