INFORMATION ANYWHERE

iAnywhere SOLUTIONS™
A SYBASE COMPANY

Technical Guide

Adaptive Server® Anywhere
パフォーマンス向上のための手引き

目次

はじめに

Adaptive Server Anywhereをインストールし運用を開始すると、次に課題となるのがそのパフォーマンスです。より高くそしてより優れたパフォーマンスは、製品の運用効率を上げ、さらには収益性をも高めることでしょう。

Adaptive Server Anywhereは、ユーザの手を煩わせることなく、そのままの状態でも優れたパフォーマンスを発揮できるようデザインされています。しかし、パフォーマンスを向上させるためのデータベース改善の余地は常にあります。

ここでは、Adaptive Server Anywhereのパフォーマンスをどのように向上できるかを見ていきましょう。

パフォーマンス分析ツール

パフォーマンス向上のための作業に着手する前に、まずは、現在のパフォーマンスを分析してみましょう。

SQL Anywhere Studio製品パッケージには、Adaptive Server Anywhereデータベースのパフォーマンスを分析するための数多くの分析ツールが用意されています。これらの分析ツールにはリリースごとに新しい機能が追加され、より強力かつより完成されたツールへと改善されています。

ここでは、以下の5つのパフォーマンス分析ツールについて説明します。使用方法の詳細については、製品の『オンライン・マニュアル』を参照してください。

- 要求レベルのロギング
- プロシージャ・プロファイリング
- グラフィカルなプラン
- パフォーマンス・モニタ
- タイミング・ユーティリティ

要求レベルのロギング

要求レベルのロギングは、サーバもしくはクライアントのどちらに問題があるのか明らかでない場合の特定のアプリケーションのパフォーマンス分析に活用できます。また、問題の原因となったサーバへの要求を特定するのにも有用です。

要求レベルのロギングは、アプリケーションから受信した要求とアプリケーションに送信した応答を個別にログにとります。サーバがアプリケーションに何を要求されているのかを特定するのに特に役立つツールです。

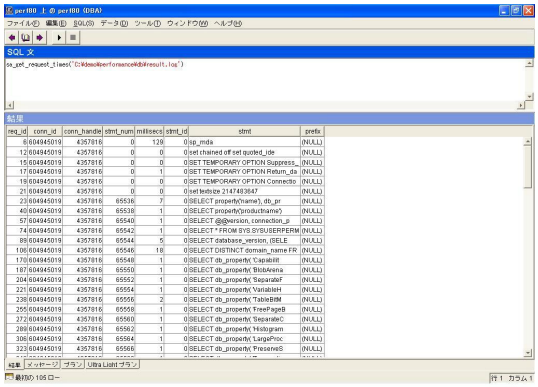
ログをとる情報には、例えば、タイム・スタンプ、コネクションID、要求タイプなどがあります。データベース・サーバの-zrコマンド・ライン・オプションを使用して、ログをとる情報を指定することもできます。また、-zoオプションを使用すると、ログをファイル出力し、情報をさらに分析することもできます。

sa_get_request_times ([request log filename [, connection id]]) ストアド・プロシージャは、要求レベルのログを読み出し、グローバル・テンポラリ・テーブルsatmp_request_timeに、ログからの文およびその実行時間と共に要約します。時間は、INSERT/UPDATE/DELETE文に関しては、そのままのものが記録されます。クエリに関しては、PREPAREからDROPまで(describe/open/fetch/close)のトータルの経過時間が記録されます。したがって、オープン・カーソルには注意する必要があります。

```
sa_get_request_times( [ request log filename [, connection id ] ] )
```

satmp_request_timeをよく分析しましょう。低コストであるが頻繁に実行される文は、パフォーマンス低下の原因となっている可能性があります。

satmp_request_time



sa_get_request_profile([request_log_filename [, connection id]])を使用すると、sa_get_request_times()を呼び出して、その結果となるsatmp_request_timeをもう1つのグローバル・テンポラリ・テーブルsatmp_request_profileに要約することができます。この処理もまた、文をまとめ、実行回数や実行時間などを記録します。

```
sa_get_request_profile( [ request_log_filename [, connection id ] ] )
```

プロシージャ・プロファイリング

プロシージャ・プロファイリングは、ストアド・プロシージャ、関数、イベント、トリガの実行時間を表示します。また、プロシージャの行ごとの実行時間を表示することもできます。データベースのプロファイリング情報を使用すると、データベースのパフォーマンスを向上させるにはどのプロシージャをチューニングすればよいかを特定することができます。

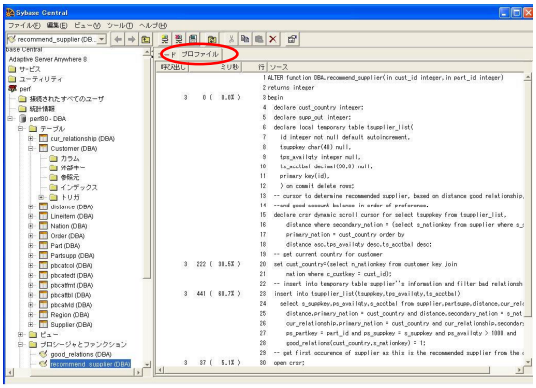
プロシージャ・プロファイリングは、上記要求レベルのロギングにより高コストであることが判明したデータベースの特定のプロシージャ（ストアド・プロシージャ、関数、イベント、トリガなど）を分析するのに利用できます。また、プロシージャ・プロファイリングは、トリガ、イベント、ネスト・ストアド・プロシージャ・コールなどの隠れている高コストなプロシージャを見つけ出したり、さらには、プロシージャ自体の潜在的な問題領域を正確に特定するのにも役立ちます。

ストアド・プロシージャにより、サーバが収集したプロシージャ・プロファイリング情報を表示させることができます。sa_procedure_profile_summaryストアド・プロシージャは、データベース内のすべてのプロシージャに関する情報を提供します。この機能を利用して、ストアド・プロシージャ、関数、イベント、トリガのプロファイリング用データを同じ結果セット内に表示することができます。Sybase Centralを使用すると、こうした情報をより快適に検査することができます。

```
sa_procedure_profile_summary
```

プロファイリングは、動的に有効もしくは無効にすることができます。プロファイリングにより生成された情報は、一時的なものであり、サーバのメモリに保存されます。Sybase Centralの「プロファイル」タブを使用すると、この情報を表示することができます。プロファイリングを一度有効にすると、データベースは、プロファイリングが無効にされるまで、もしくは、サーバがシャットダウンされるまで、プロファイリング用の情報を収集し、1 ms単位で蓄積します。

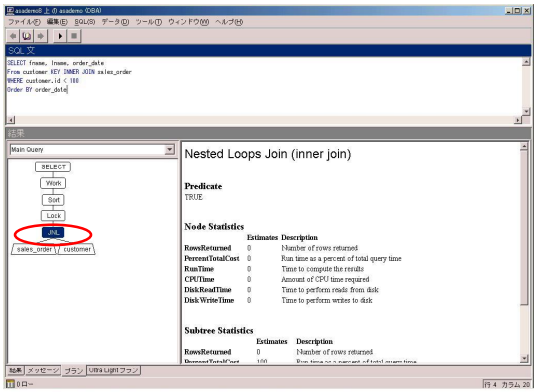
プロファイル情報



グラフィカルなプラン

Interactive SQLのグラフィカル・プラン機能は、クエリの実行プランを表示します。特定のクエリに関するパフォーマンス問題を診断するのに使用することができます。プラン内の情報から、データベースのどこにインデックスを追加するかなどを決定することができます。

統計情報付きのグラフィカルなプラン



グラフィカルなプランは、短いプランや長いプランに比べ、はるかに多くの情報を提供します。グラフィカルなプランを統計情報付きで表示するか、統計情報なしで表示するか、選択することもできますが、どちらを選択しても、プランのどの部分が最も高コストであると推定されているのかはすぐに明らかになります。統計情報付きのグラフィカルなプランは、（表示するのにサーバに負荷がかかりますが）クエリの実行時にサーバがモニタしている実際のクエリ実行に関する統計情報を提供します。これにより、クエリ・オプティマイザがアクセス・プラン構築に使用する推定とクエリ実行中にモニタされた実際の統計情報とを直接比較することができるようになります。ただし、オプティマイザはクエリのコストを正確に推定できない場合がありますので、この2つには多少の違いがあることを忘れないでください。グラフィカルなプランは、アクセス・プランのデフォルトのフォーマットです。

グラフィカルなプラン図中のノードをクリックすると、そのノードに関する詳細な情報を見ることができます。統計情報付きのグラフィカルなプランは、そのプランに関わるすべての推定だけでなく、文を実行したときの実際の実行時間のコストも表示します。コストを表示するには、文を実際に実行する必要があります。これは、高コストなクエリの場合、プランへのアクセスに遅延が生じるかもしれないということを意味します。また、変更を取り消すためにRollbackを実行することはできますが、DeleteやUpdateといったクエリが部分的に実行されるということになります。

パフォーマンス上の問題を抱えている場合、また、推定したロー・カウントやラン・タイムが予測したものと異なる場合には、統計情報付きのグラフィカルなプランを使用してください。統計情報付きのグ

ラフィカルなプランは、双方の比較を行えるよう、推定と実際の統計情報を提供します。この2つが大きく異なる場合、オプティマイザには正しく推定するための情報が不足していると考えられます。

以下に、キーとなる統計情報と問題があった場合の対策を示します。これらの統計情報は、統計情報付きのグラフィカルなプランでチェックすることができます。

- ロー・カウントは、結果セット内のローを測定します。推定したロー・カウントが実際のロー・カウントと大きく異なる場合は、述部（Predicate）の選択性が正しくない可能性があります。

- 正確な選択性の推定は、クエリ・オプティマイザが適切に動作するのに不可欠です。例えば、オプティマイザが、誤ってある述部の選択性は高い（例：5%）と推定したが、実際の選択性は低い（例：50%）場合などには、パフォーマンスは低下してしまいます。総じて、推定が正確ではないという場合もありますが、大きな誤差は問題の原因となります。述部がヒストグラムが存在しないベース・カラムにかかる場合は、CREATE STATISTICS文を実行し、ヒストグラムを作成してください。これで問題が解決するかもしれません。選択性のエラーが依然問題となる場合は、最後の手段として、クエリ・テキスト内の述語と共に選択性のユーザ推定を指定するといった方法も考えられます。

- ラン・タイムは、クエリを実行する際の所要時間を測定します。テーブル・スキャンやインデックス・スキャンのラン・タイムが間違っている場合は、REORGANIZE TABLE文を実行することでパフォーマンスを向上させることができます。sa_table_fragmentationやsa_index_density機能を使用すると、テーブルまたはインデックスが断片化しているかどうか調べることができます。

- 推定ソースがGuessである場合、オプティマイザは必要な情報を取得することができず、これが問題の原因となることがあります。推定ソースがIndexで、選択性の推定が間違っている場合は、インデックスの“歪み”が原因であると考えられます。このような場合には、REORGANIZE INDEX文により、インデックスのデフラグを行ってください。

- キャッシュのリード数とキャッシュのヒット数がまったく同じである状態というのは、データベース全体がキャッシュされているという理想的な状態です。リード数がヒット数を上回っているのは、サーバはキャッシュに行こうとするが失敗しディスクから情報を読み出さなくてはならないということです。ハッシュ・ジョインなどのケースで、こうした状況は予想されます。しかし、ネスト・ループ・ジョインなどの他のケースでは、低いキャッシュのヒット率は、パフォーマンスに問題があるということを示している場合があります。このような場合には、キャッシュ・サイズを増やしてください。

パフォーマンス・モニタ

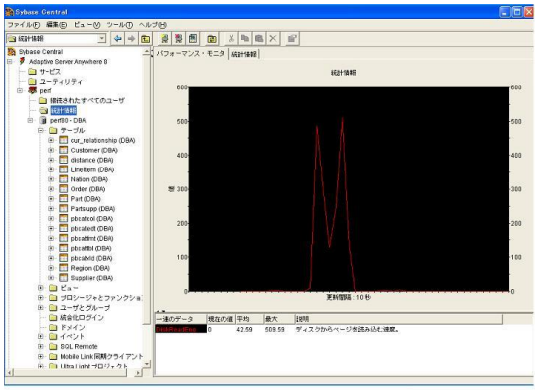
パフォーマンス・モニタは、ディスクへのアクセスやメモリへのアクセスなど、データベース・サーバのアクションに関する詳細な情報をトラッキングするのに便利なツールです。

Sybase Centralのパフォーマンス・モニタを使用すると、接続したAdaptive Server Anywhereデータベース・サーバの多様な統計情報を、Sybase Central内でグラフ化することができます。Sybase Central内のすべての統計情報は、「統計情報」フォルダに表示されます。

パフォーマンス・モニタの特長を以下に示します。

- リアルタイムな更新（更新間隔は調整可能）
- サイズ変更可能な色分けされた凡例
- 設定可能な表示プロパティ

パフォーマンス・モニタ



Sybase Centralのパフォーマンス・モニタは、統計情報を集めるために、サーバに対して実際のクエリを使用します。したがって、こうしたモニタ自体が、キャッシュのリード数/秒などいくつかの統計情報に影響を与えます。Sybase Centralのパフォーマンス・モニタの代わりに、Windowsのパフォーマンス・モニタを使用して、サーバの統計情報をグラフ表示することもできます。

Windowsのパフォーマンス・モニタの利点として、以下の2点が挙げられます。

- より多くのパフォーマンス統計情報を提供する（おもにネットワーク通信に関する統計情報）
- 共有メモリを使用してモニタする（サーバに対して実際のクエリを使用しないため統計情報への影響はなし）

複数のバージョンのAdaptive Server Anywhereを同時に起動している場合には、複数のバージョンのパフォーマンス・モニタを同時に起動することも可能です。

タイミング・ユーティリティ

fetchtst、instest、trantestなどのパフォーマンス・テスト用のユーティリティは、Adaptive Server Anywhere のインストール先ディレクトリ配下の、/samples/asa/ディレクトリにあります。また、これらユーティリティに関するドキュメントは、同じフォルダにあるReadme.txtファイルに収容されています。これらのツールにより、統計情報付きのグラフィカルなプランに比べて、より正確なタイミングでのパフォーマンス分析が可能になります。また、サーバやデータベース構成などの条件を考慮した、達成可能なベスト・パフォーマンス（例えばスループット）の指標を得ることもできます。

Fetchtst	任意のクエリのフェッチ・レートを測定します。
Instest	テーブルにローを挿入するのにかかる時間を割り出します。
Trantest	決められた条件下で（サーバ構成、データベース設計、トランザクション数）処理可能な負荷を計算します。

同時性について

データベース・サーバは、トランザクション処理時、テーブルの1つまたは複数のローをロックすることができます。データベース・サーバは、このロックにより、他のトランザクションによる同時アクセスを防ぎ、データベース内の情報の信頼性を確保します。また、更新中の情報を識別することで、結果クエリの精度を高めます。

データベース・サーバは、明示的な指示がなくても、こうしたロックを自動で設定します。そして、トランザクションが得たすべてのロックを、そのトランザクションが完了するまで保持します。ローにアクセスしているトランザクションがロックを保持しているともいえます。他のトランザクションは、ロックの種類に応じて、ローに制限付きでアクセスできたり、まったくアクセスできなかったりします。

1つまたは複数のローが多くのユーザによって同時に頻繁にアクセスされると、パフォーマンスの低下が起こりえます。ロックに問題があると考えられる場合には、sa_locksプロシージャを使用して、データベースのロックに関する情報を入手することもできます。ロック問題が特定された場合には、関連する接続プロセスに関する情報をAppInfoの接続プロパティから見ることができます。

```
sa_locks
```

パフォーマンス向上のための21のヒント

データベースの現在の状況を把握したら、次にシステムを最大限活用するための調整をどのように行うかを決めます。

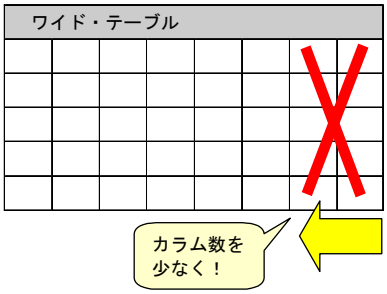
パフォーマンスを向上させる最も基本的な手段として、優れたスキーマ設計を採用するということが挙げられます。データベースのパフォーマンスの優劣は、最終的にはその“骨組み”に大きく依存します。データベース・スキーマは、まさにデータベースの“骨格”となるものであり、これには、テーブル、ビュー、トリガおよびこれらの関係を定義する定義情報が含まれます。

以下に挙げる点に注意しながら、データベース・スキーマをもう一度見直してみましょう。ちょっとした変更がデータベースのパフォーマンスを大きく向上させることになるかもしれません。

【ヒント1】 テーブルのカラム数を減らす

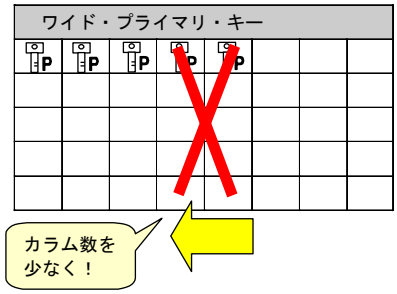
テーブル内のカラムの数が多すぎて、個別のローのサイズがデータベースのページ・サイズを超えると、各ローは複数のページにまたがって分割されてしまいます。ローが複数のページに渡ると、その分各ローを読み出す時間も長くなります。

カラムの数の多いテーブルは、ワイド・テーブルと呼ばれています。ワイド・テーブルを使用していることが分かっていて、パフォーマンスの低下に気づいた場合は、テーブルを正常なサイズに戻し、カラムの数を減らすことを考えてください。カラムの数を減らすことができない場合には、対策としてデータベースのページ・サイズを大きくすることもできます。これは、テーブルのほとんどがワイド・テーブルである場合、特に有効です。



【ヒント2】 プライマリ・キーのカラム数を減らす

複数のカラムからなるプライマリ・キーは、ワイド・テーブルと同様、ワイド・プライマリ・キーと呼ばれます。プライマリ・キーに多くのカラムがあるほど、サーバへの要求も増えることになります。したがって、プライマリ・キーのカラムの数を減らすことにより、パフォーマンスを向上させることができます。

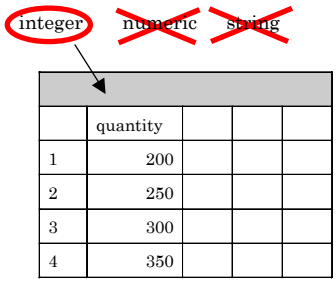


【ヒント3】 制約を宣言する

異なるテーブルのカラムの値同士に暗黙的な関係があると、それらのテーブル間に宣言されていないプライマリ・キーと外部キーの関係があるということになります。関係を宣言しないとインデックスの保守にかかる時間を削減することができますが、逆に、関係を宣言するとジョインが行われるときのクエリのパフォーマンスを向上させることができます。これは、コスト・モデルが推定という作業をより正確に行えるためです。

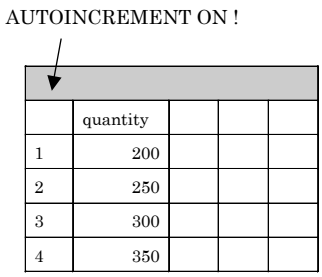
【ヒント4】 適切なデータ型を採用する

データ型は、値の範囲、それらの値で実行可能な操作、値のメモリへの保存方法など、特定のデータ・セットに関する情報を格納するものです。使用するデータに適したデータ型を採用することでパフォーマンスを向上することができます。例えば、数字のみを含む値に対して、charやstringといったデータ型を割り当てないようにします。また、numericやstringなどの高コストなデータ型に優先して、低コストなデータ型を可能な限り選択するようにします。



【ヒント5】 AUTOINCREMENTを適用する

プライマリ・キーの値はユニークである必要があります。プライマリ・キーにユニークな値を設定する方法は各種ありますが、最も効率的なのは、デフォルトのカラム値をAUTOINCREMENT ONに設定する方法です。ユニークな値を維持したいどのカラムでもこうしたデフォルトの設定を行うことができます。AUTOINCREMENT機能を使用するこの方法では、サーバが値を生成します。そのため、他の方法よりプライマリ・キーの値を速く生成することができます。

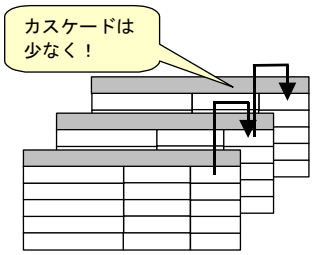


【ヒント6】 トリガの使用を見直す

使用しているトリガをサーバで利用可能な機能に置き換えることができないか見直してください。例えば、最新の更新時間とユーザ情報でカラムをアップデートするというトリガは、これに対応するサーバの特別値で置き換えることができます。また、すでにあるトリガをデフォルト設定で使用するということもパフォーマンス向上につながります。

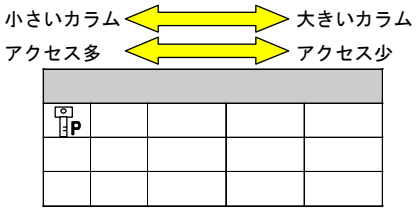
【ヒント7】 カスケードを削減する

関連したアクションのカスケーディングは、トランザクションごとに複数のテーブルの更新が必要となるため、パフォーマンスという観点でみると、高コストなものになります。例えば、employeeテーブルからdepartmentテーブルへの外部キーがON UPDATE CASCADEと定義されていて、department IDを更新すると、employeeテーブルも自動的に更新されることになります。関連したアクションのカスケーディングは便利な機能ですが、アプリケーションのロジックでこれを実践する方が効率的である場合もあります。



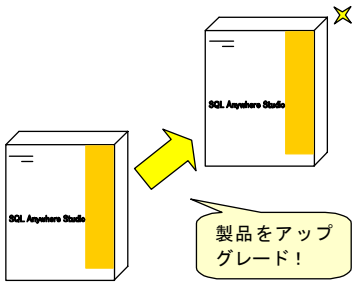
【ヒント8】 カラムの配置を考える

ロー内のカラムは、作成された順番に、順次アクセスされます。例えば、ローの最後にあるカラムにアクセスするためには、Adaptive Server Anywhereは、ロー内の手前のカラムをすべてスキップする必要があります。プライマリ・キーのカラムは、常にローの最初に保存されます。パフォーマンスを向上させるには、小さいカラムや頻繁にアクセスされるカラムを、アクセスのほとんどないカラムの前に配置するようにテーブルを作成することが重要になります。



【ヒント9】 製品をアップグレードする

Adaptive Server Anywhereは、パフォーマンスや操作性といった観点から常に評価され、また強化され続けています。Adaptive Server Anywhereのユーザは、製品のリリースごとに新しい機能や動作の改善点を利用し、パフォーマンスを最適化するのに役立てることができます。



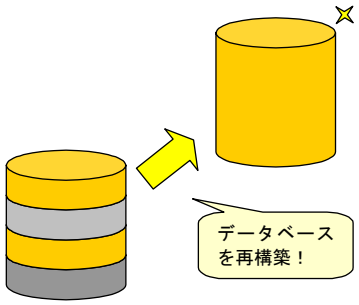
Adaptive Server Anywhereの旧バージョンで作成したデータベースを単に新しいリリース上で動作させることもできますが、新しい機能の多くは、データベースをアップグレードして初めて利用可能となります。Upgradeユーティリティを起動すると、システム・テーブル、システム・プロシージャ、データベース・オプションを追加、変更でき、データベースをAdaptive Server Anywhereの新しいバージョンのものにアップグレードすることができます。

【ヒント10】 データベースを再構築する

データベースの再構築というのは、データベース全体をアンロードし、さらにリロードするというプロセスです。データベース・ファイル・フォーマットのアップグレードといった呼び方をする場合もあります。

データベースの再構築では、データやスキーマなどすべての情報はいったん取り除かれ、また、一様な

方法でデータベースへと戻されます。こうして、ディスク・ドライブのデフラグと同じように、スペースを埋めてゆき、パフォーマンスを向上させます。再構築は、特定の設定を変更する機会でもあります。ただし、速度の面では、Upgradeユーティリティの方が再構築よりも優れています。また、保存ファイルへの影響もありません。



再構築によりデータベースをアップグレードした場合、ソフトウェア最新バージョンのすべての新機能およびパフォーマンス強化へのアクセスも可能となります。

【ヒント11】 コンフィグレーションを確認する

データベースおよびサーバのコンフィグレーションを見直し、ハードウェアまたはファイル・スペースが使用に適したものであることを確認します。

【ヒント12】 適切なページ・サイズを選択する

ページ・サイズもデータベースのパフォーマンスに影響を及ぼします。Adaptive Server Anywhereは、1024、2048（デフォルト）、4096、8192、16384、32768バイトのページ・サイズをサポートしています。それぞれのページ・サイズに利点と欠点があります。

小さめのページは、より少ない情報を保持します。またスペースの使用もより非効率なものとなります。しかし、小さいページ・サイズを使用すると、同じサイズのキャッシュにより多くのページを保管できるため、より少ないリソースでAdaptive Server Anywhereを運用することができます。小さめのページは、データベースをメモリ容量の少ない小型のマシンで運用する場合に特に有効です。また、データベースをさまざまなロケーションからの細かな情報の取り出しに主に使用するような場合にも有効です。

これに対し、大きめのページ･サイズは、どちらかといえばシーケンシャルなテーブル・スキャンを実行するクエリや大きいデータベースに有効です。ディスクの物理的な設計にもよりますが、大きめのページ･サイズを選択した場合は、多数の小ブロックのかわりに少数の大ブロックを検索することで、検索の効率が高まることがよくあります。大きめのページ･サイズの利点として、インデックスのファン・アウトを改善しインデックス・レベルの数を減らすことができる、カラム数の多いテーブルを作成することができるといった点が他に挙げられます。

大きめのページ･サイズにはより多くのメモリが必要になります。また、1枚のページに格納できるローの最大数は255であるため、ローの数が少ないテーブルは各ページを埋めることができず、結果としてスペースを非効率に使用することになります。

NOTE

特に大きなページ・サイズ(16 kbまたは32 kb) は、大容量のデータベース・サーバのキャッシュが常時確実に利用できる場合を除き、ほとんどのアプリケーションで推奨しません。16 kbや32 kbのページ・サイズを使用する前には、メモリやディスク・スペースの増加がパフォーマンス特性に与える影響を十分に調査してください。

【ヒント13】 ファイルの断片化を解消する

オペレーティング・システム・ファイルの断片化を解消するため、定期的にデフラグ用ユーティリティを起動し、ディスクのデフラグを実行してください。ファイルの断片化は、パフォーマンスに弊害をもたらします。

Windows NT/2000/XP上でデータベースを起動すると、データベース・サーバは、データベース内のファイル・フラグメントの数を特定し、その数が複数である場合には以下の情報をメッセージ・ウィンドウに表示します。

```
データベース・ファイル"mydatabase.db" はnnn個のディスク・フラグメントで構成されています
```

データベース内のファイル・フラグメントの数は、DBFileFragmentsデータベース・プロパティを使用しても知ることができます。

【ヒント14】 データベースの断片化を抑制する

データベースの断片化は、データベースに変更を加えると自然に起きてしまいます。断片化が進むと、パフォーマンスにも影響を及ぼすようになり、これは、データベースのサイズが大きくなるとより顕著になります。

データベースの断片化が大きく進んでしまった場合は、断片化を削減しパフォーマンスを向上させるため、テーブルを再編成してみてください。多くのテーブルでdelete/update/insertの動作を頻繁に繰り返し行った場合など、いくつかのケースでは、データベースの再構築も考慮する必要があります。

インデックスの断片化

インデックスは、特定のカラムの検索をスピードアップするためのものです。しかし、インデックス付きのテーブルで数多くのDELETE操作が行われると、インデックスは断片化することがあります。このような場合に、インデックスが頻繁にアクセスされ、また、キャッシュ容量が小さくキャッシュにすべてのインデックスを保持することができないと、パフォーマンスの低下が起こります。

sa_index_densityストアド・プロシージャを使用すると、データベースのインデックスの断片化の進み具合に関する情報を得ることができます。このプロシージャを実行するには、DBA権限が必要となります。sa_index_densityストアド・プロシージャは、以下の文により呼び出します。

```
CALL sa_index_density (['table_name'[,'owner_name']])
```

インデックスの断片化が極端に進んでしまった場合は、REORGANIZE TABLEを起動してください。他に、インデックスを削除し、また作成し直すといったことも可能です。この時、削除するインデックスがプライマリ・キーである場合には、外部キー・インデックスも削除し作成し直す必要があります。

また、クラスタード・インデックスをテーブル上に作成し、パフォーマンスの向上を図ることもできます。クラスタード・インデックスは、テーブルのローをインデックスとほぼ同じ順番に格納するという働きをします。

テーブルの断片化

テーブルの断片化は、ローが隣接した状態で保管されていない場合、すなわちローが複数のページに分割されている場合に起こります。余分にページにアクセスする必要があるため、このような場合パフォーマンスは低下します。

Adaptive Server Anywhereでは、多少ローが増えても大丈夫なように、各ページに余分な領域を設けています。しかし、更新が加えられローがそのページの余分な領域を超えてしまうと、ローは分割されてしまいます。そして、そのローの元の場所にはローの続きが保管されている他のページへのポインタができます。 例えば、UPDATE文により空白のローを埋めたり、テーブルに新しいカラムを挿入したりすると、問題となるローの分割が引き起こされることがあります。ローが他のページにあると、これらのページにアクセスするための時間が必要となります。

sa_table_fragmentationストアド・プロシージャを使用すると、データベースのテーブルの断片化の進み具合に関する情報を得ることができます。sa_table_fragmentationストアド・プロシージャは、以下の文により呼び出します。

```
CALL sa_table_fragmentation (['table_name'[,'owner_name']])
```

テーブルの断片化を最小限に抑えるには、以下の3つの方法があります。

- テーブルのページの余分な領域の割合を指定し、将来の更新に備える。
  この PCTFREE 指定は、CREATE TABLE、ALTER TABLE、DECLARE LOCAL TEMPORARY TABLE、LOAD TABLEにより行うことができます。

- 特定のテーブルを再編成する。
  REORGANIZE TABLEを実行すると、特定のテーブルまたはテーブルの1部をデフラグすることができます。テーブルの再編成は、データベースへのアクセスに影響を与えません。

- データベース全体を再構築する。
  再構築は、システム・テーブルを含むすべてのテーブルをデフラグするという点で、より包括的な方法になります。再構築は、さらに、テーブルのローをクラスタード・インデックスやプライマリ・キーで指定された順に並べ替えます。

【ヒント15】 適切なハードウェアを使用する

PC上でAdaptive Server Anywhereを動作させる場合には、サーバがCPU、メモリ、ディスクなどの必要条件を満たしているか確認します。快適な動作環境は、パフォーマンスを向上させます。

- Windows Windows 95、Windows 98、Windows Me、Windows NT (v4.0以上)、Windows 2000、Windows XP

- Intel 486以上のCPU、またはこれに相当するCPU

- Adaptive Server Anywhereは、最小4 Mbのメモリで動作することができます。しかし、データベースにJavaを使用する場合は8 Mb以上のメモリ、管理ツールを使用する場合は

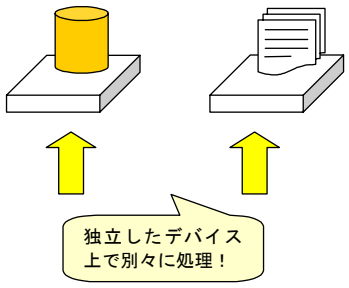
32 Mb 以上のRAMが必要になります。オペレーティング・システムに必要なメモリ容量に加えて、これらのメモリ容量が必要となります。

- データベースとログファイルの保存に十分なディスク容量
- 最低限のハードウェア必要条件しか満たしておらず、パフォーマンスが不十分である場合には、ハードウェアのアップグレードを検討してください。通常の場合は、ハードウェア構成が、サーバにかかる作業負荷に十分であるかを検討してください。

【ヒント16】 ファイル配置を変更する

ディスク・ドライブの動作は、最新のプロセッサやRAMに比較すると遅いものです。ディスクがページを読み出したり、ページに書き込んだりするのをただ単に待っていることがデータベース・サーバの動作が遅い原因となっていることがよくあります。物理的に異なるデータベース・ファイルを物理的に異なるデバイスに配置してみると、ほとんどの場合、データベースのパフォーマンスは向上します。例えば、1台のディスク・ドライブがキャッシュとのデータベース・ページのやりとりでビジー状態にあっても、もう1方のデバイスはログファイルの書き込みを行っているといった状況が想定できます。この方法によりパフォーマンスの向上を実現するには、それぞれのデバイスが独立している必要があります。パーティションで切られたいくつかの論理ドライブを持つ単一のディスクでは、こうしたパフォーマンスの向上はあまり期待できません。

トランザクション・ログのミラーリングを行っている場合、使用しているトランザクション・ログ・ミラー・ファイルを物理的に分かれたドライブに配置すると、Adaptive Server Anywhereは、ログ・ファイルやログ・ミラー・ファイルへの書き出しを効率的に行うことができるため、より早く動作するようになります。また、ディスク障害に対する保護もより強固なものになります。トランザクション・ログ・ファイルおよびトランザクション・ログ・ミラー・ファイルの配置場所は、dblogコマンド・ライン・ユーティリティもしくはSybase Centralの「ログ・ファイル設定の変更」で指定することができます。



Adaptive Server Anywhere結合の形成やソートといった操作を行うときには、キャッシュ内の利用可能なスペースが足りなくなることがあります。こうした場合、Adaptive Server Anywhereは、そのスペースを集中的に使うようになります。このような場合、データベースの全体的なパフォーマンスの優劣は、テンポラリ・ファイルのあるデバイスのスピードに大きく依存することになります。テンポラリ・ファイルが高速デバイス上にあり、そのデバイスがデータベース・ファイルのあるデバイスから物理的に分離していると、Adaptive Server Anywhereの動作は速くなります。 これは、テンポラリ・ファイルを使用す必要がある操作の多くはまた、データベースからも大量の情報を入手する必要があるためです。情報を2台の別々のディスクに配置することにより、このような操作を同時に行うことができるようになります。

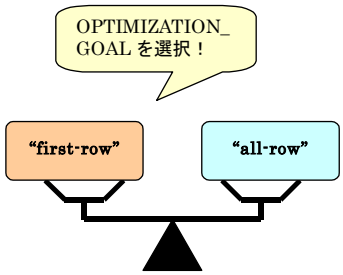
テンポラリ･ファイルの配置場所は、よく注意して選択してください。Windows上では、Adaptive Server Anywhereは、環境変数ASTMP、TMP、TMPDIR、TEMPを順に調べ、テンポラリ・ファイルを置くディレクトリを決定します。どれも定義されていない場合、Adaptive Server Anywhereは、自身のテンポラリ・ファイルをカレント・ディレクトリに配置します。これは、最高のパフォーマンスを導き出すのに

決していい場所とは言えません。

パフォーマンスを向上させるには、データベースを、別々のデバイスにある複数のDB領域に分割することもできます。このような場合は、同じジョイン操作で使われるテーブルを別々のDB領域に分割し、それぞれのファイルから情報が読み出されるようにしてください。

上と似た手法に、テンポラリ・ファイルやデータベース・ファイルをRAIDデバイスやWindows NTのストライプ・セットに配置するといったものもあります。これらのデバイスは、論理ドライブとして動作しますが、ファイルを多くの物理ドライブに分散し、複数のヘッドで情報にアクセスするため、パフォーマンスを飛躍的に向上させます。

【ヒント17】“first-row”と“all-rows”を使い分ける



OPTIMIZATION_GOALオプションは、Adaptive Server AnywhereがSQL文を、応答時間で最適化するか（first-row）、リソースの総消費量で最適化するか（all-rows）をコントロールします。言い換えると、クエリ処理を、最初のローを素早く返すよう最適化するか、完全な結果セットを返すためのコストを最小限にするよう最適化するかを選択することができます。

オプションを“first-row”に設定すると、Adaptive Server Anywhereは、クエリ結果の最初のローをフェッチする時間を短縮するアクセス・プランを選択します。この場合、おそらく検索にかかる合計の時間は長くなります。特に、Adaptive Server Anywhereのオプティマイザは、最初のローを返す時間を削減するために、結果のマテリアライズ（実体化）が必要となるアクセス・プランを可能であれば常に避けるようになります。この設定では、オプティマイザは、例えば、明示的なソート操作が必要になるプランよりも、クエリのORDER BY句を満たすインデックスを利用するアクセス・プランを優先します。

クエリのFROM句のFASTFIRSTROWテーブル・ヒントを使用すると、OPTIMIZATION_GOALの設定を変更することなく、特定のクエリのoptimization goal（最適化ゴール）を“first-row”に設定することができます。

オプションを“all-rows”（デフォルト）に設定すると、Adaptive Server Anywhereは、予測される総検索時間が一番短いアクセス・プランを選択するようクエリを最適化します。PowerBuilder DataWindowなど、結果セット全体を処理するようなアプリケーションには、OPTIMIZATION_GOAL を“all-rows”に設定するのが適切でしょう。

【ヒント18】 正しいカーソル・タイプを指定する

正しいカーソル・タイプを指定することでパフォーマンスは向上します。例えば、カーソルが読み取り専用である場合、そのカーソルは読み取り専用であると宣言することによって、最適化をより速くし、実行時間を短くすることができます。これは、制約をチェックしないなど実体化が少なくて済むためです。カーソルが更新可能である場合は、いくつかの書き換えを省略することができます。また、クエリが更新可能である場合、実行エンジンには、オプティマイザが選択した実行プランに応じて、キーセット駆動型のアプローチをする必要が生じます。ここでは、キーセット・カーソルは、より高コストであ

ることに注意してください。

【ヒント19】 スモール・テーブルの統計情報を有効に使う

Adaptive Server Anywhereは、それぞれの文を実行するのに最も効率的な方策を決めるため、統計情報を利用します。Adaptive Server Anywhereは、自動的にこうした統計情報を収集、更新し、データベースに恒久的に保管します。1つの文の処理中に収集された統計は、次の文を効率的に実行する方法を探し出すのに利用されます。

あまり有用でないと考えられる場合は統計を収集しない。こうした目的で、Adaptive Server Anywhereは、デフォルトでスモール・テーブルの統計を収集しないよう設定されています。スモール・テーブルは、min_table_size_for_histogramデータベース・オプションで定義されます。デフォルト値は、1000（ロー）です。

しかし、スモール・テーブルの統計収集がまったく役に立たない訳ではありません。スモール・テーブルの統計を作成するには、CREATE STATISTICS文を使用します。いったん作成された統計は、Adaptive Server Anywhereにより自動的に維持されます。Adaptive Server Anywhereが自動的にスモール・テーブルの統計を作成するよう設定することもできます。この設定は、min_table_size_for_histogramオプションを小さい値に設定することにより行いますが、その必要はなく、あまり推奨できません。

【ヒント20】 ユーザ推定を慎重に使用する

統計が不正確になる場合が時折あります。こうした状況は、大量のデータが追加、更新、削除された後クエリがあまり実行されていない時に最も起こりやすくなります。不正確な統計や統計が得られないという状況は、パフォーマンスにとっての障害となりえます。Adaptive Server Anywhereが統計を更新するのに長い時間がかかっている場合は、CREATE STATISTICSまたはDROP STATISTICSを実行し、統計をリフレッシュしてみてください。新しいバージョン（9.0以上）のAdaptive Server Anywhereは、LOAD TABLE実行時やクエリ実行中だけでなく、update DML文の実行時にも統計を更新します。

まれに、こうした統計による対策が効果的でない場合があります。条件の成功率がオプティマイザの推定とは異なるということがあらかじめ分かっている場合には、検索条件にユーザ推定を明示的に指定することもできます。

NOTE

ユーザ定義の推定がパフォーマンスを向上させることもありますが、継続して使用される文に明示的なユーザ定義の推定を使用するのは避けてください。データが変更されると、明示的な推定は、不正確になったり、オプティマイザが間違ったプランを選択するよう作用する恐れがあります。

ソフトウェアが選択するアクセス・プランが思わしくないために起こるパフォーマンス問題の次善策として不正確な選択性の推定を採用してしまった場合には、USER_ESTIMATESをOFFにし、その値を無視するよう設定することもできます。

【ヒント21】 無駄な関数を一掃する

繰り返し実行されるクエリの高コストなユーザ定義関数を削減すると、パフォーマンスを向上することができます。

結論

データベースのパフォーマンス向上に際して、常に必要となるのはやはり、あらかじめプランを持っておくことです。パフォーマンス向上のために、多大な労力や時間を集中して投入する必要はありません。まず、現在のデータベースのパフォーマンスを評価し、変更を加える前にすべてのオプションを検討してみてください。Adaptive Server Anywhereのパフォーマンス機能およびパフォーマンス分析ツールを使用してデータベースのスキーマを再評価することにより、パフォーマンスに関する問題は容易に診断することができます。そして、ここに記載した21のヒントを参考にしながら問題を解決し、データベースのパフォーマンスを最適な状態に保ちます。こうしてユーザは、新たな“課題”をクリアすることができるのです。

法的注意

＊本書は、米国iAnywhere Solutions社が作成・テストしたものを日本語に翻訳したものです。

アイエニウェア・ソリューションズ株式会社
http://www.ianywhere.jp/

1020369