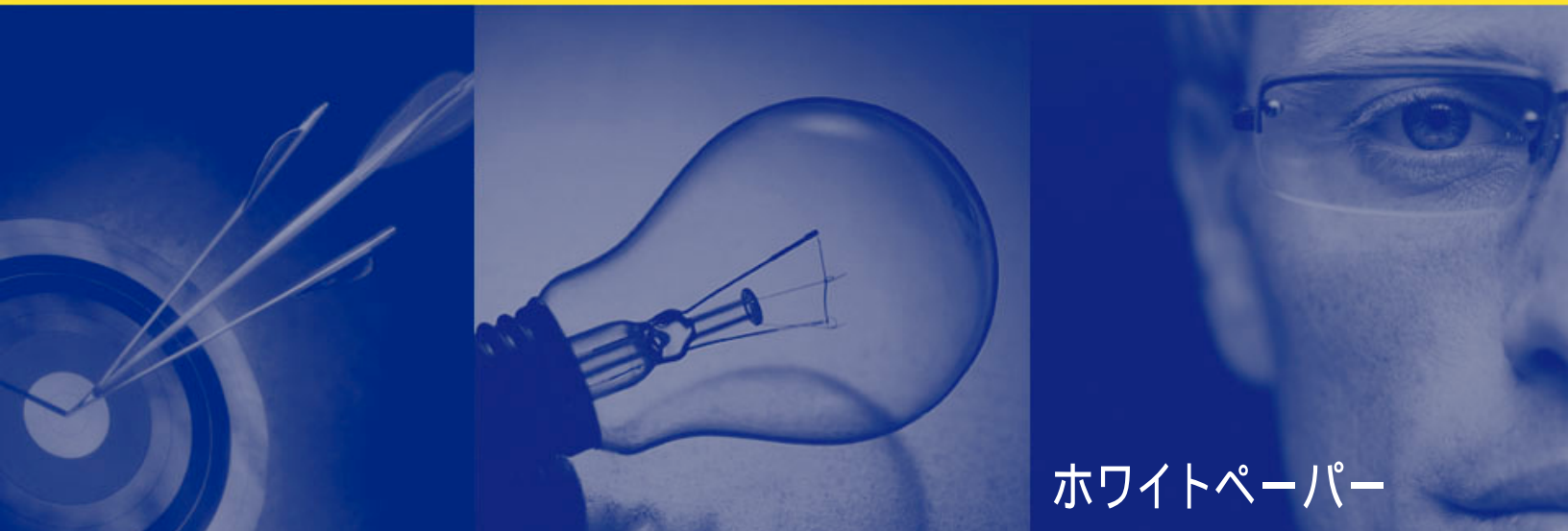


# SQL Serverに関してよくある間違いトップ10

著者

Kevin Kline

Quest Software, Inc.



ホワイトペーパー

© Copyright Quest<sup>®</sup> Software, Inc. 2007. All rights reserved.

このガイドには著作権で保護されている機密情報が含まれています。このガイドで説明されているソフトウェアは、ソフトウェアライセンスまたは守秘義務契約の下に提供されています。このソフトウェアは、適用される契約の条項に準拠している場合のみ使用またはコピーできます。このマニュアルの一部または全部を複製または転送することは、購入者による個人使用以外の目的では、Quest Software, Inc.の書面による許可がない場合、複写および記録を含む電子的または機械的ないかなる手段でも禁じられています。

## 保証

この文書に含まれている情報は、予告なく変更されることがあります。クエスト・ソフトウェアは、この情報に関するいかなる保証もいたしません。クエスト・ソフトウェアは、商品性および特定目的への適合性に関する黙示的な保証については一切免責されるものとします。クエスト・ソフトウェアは、この情報の提供または使用に起因して申し立てられた、直接的、間接的、付随的、または結果的ないかなる損害についても責任を負わないものとします。

## 商標

このガイドで使用されているすべての商標および登録商標は、その所有者の財産です。

ワールドワイド本社  
5 Polaris Way  
Aliso Viejo, CA 92656  
[www.quest.com](http://www.quest.com)  
電子メール: [info@quest.com](mailto:info@quest.com)

各地域および海外の所在地情報については、本社サイトを参照してください。

最終更新日 - 2007年6月14日

# 目次

はじめに.....	1
よくある間違いトップ 10 .....	2
10. スケーラビリティの要件が不明 .....	2
9. 不適切に正規化されたテーブル .....	4
8. インデックスの問題 .....	5
7. ベースラインまたはベンチマークがない.....	6
6. ディスク領域について考慮しても IO が考慮されていない.....	7
5. エラーの通知.....	9
4. バックアップ .....	10
3. クエリのチューニング処理 .....	11
2. 内部構造に関する知識不足 .....	12
1. カーソル .....	13
結論とまとめ .....	14
著者について .....	15

## はじめに

Microsoft SQL Serverは当初、ずっと以前の1995年に、Sybase社からライセンスを受けたコードベースを使用するWindows NT製品として市場に投入されました。当時と現在の間の長年にわたる進歩にもかかわらず、データベース管理者(DBA)、アプリケーション開発者、データベース設計者は、同じ種類の間違いを何度も繰り返しているように思われます。

一方では、データベースアプリケーションの設計、構築、保守を行ってきた業界のベテランは、後輩たちが、簡単に回避することができる過去の間違いを繰り返すので失望する状況になりがちです。これらは、アプリケーションのパフォーマンスの問題や保守の容易さについて簡単な対処法がある、「収穫しやすい果実」のような間違いなのです。他方で、IT業界は人材の絶え間ない移動に直面しており、専門知識(そしてさらには過去の間違い)を学び始めたばかりの新しい人材が常に存在することになります。

残念ながら、経験豊富なSQL Server専門職にはよく知られている問題をすべて学習できるような、包括的なベストプラクティスのセットを提供する単独のWebサイトや知識ベースは存在しません。それどころか、SQL ServerはMicrosoft社自身のマーケティングで、生産的に利用するために必要なスキルと管理が少なくすむエンタープライズクラスのデータベースであると位置付けられたため、経験が浅い人たちにとっての状況が、ある意味ではいくぶん悪化したのです。この主張の大部分は正しいとはいえ、その逆もまた正しいのです。つまり、ハイエンド(ユーザ、トランザクション、またはデータボリュームの面で)の規模に合わせて調整する必要があるデータベースアプリケーションであれば、設計やアプリケーションの面ですまわずに適切に実行されるようにするために、優れたスキルと技巧が必要になります。

このホワイトペーパーで取り上げる、Microsoft SQL Serverに関してよくある間違いのトップ10のリストは、包括的なものではありません。しかし、さまざまな規模のIT企業で行われた、あらゆる経験レベルのITスタッフが携わったアプリケーションの開発時に頻繁に発生した大きな問題の中から、その一部を説明します。ここで説明する問題を理解することにより、Microsoft SQL Serverを使用する現在および将来のITプロジェクトで、類似の問題を識別できるようになり、他の現場で発生してきた多くの問題を回避できるようになります。その結果、仕事の能力が向上し、職場や顧客からの信頼度が増すこととなります。

## よくある間違いトップ 10

以下のトップ10リストには、IT業界内の、一般化したさまざまな分野が含まれています。そのため、このホワイトペーパーはDBAの問題またはアプリケーション開発者の問題を単独で掘り下げるのではなく、アプリケーション開発者、DBA、データベース設計者の全員に関係する一般的で(通常は)簡単に修正できる問題について考察しています。所属するIT企業では、これら3つの役割すべてを1人が担当する場合も、別の人がそれぞれの役割を担当することもあるでしょう。しかし、それぞれの役割には固有のわなや秘訣、落とし穴が確実に存在し、アプリケーションの構築、展開、保守、トラブルシューティングを行うときに悩みの種になる可能性があります。

このホワイトペーパーでは、トップ10の各項目について説明し、可能な場合には例を使って具体的に解説します。さらに、今後の業務で問題が発生しないように、現在の問題を解決するためのリソースと提案を紹介します。

### 10. スケーラビリティの要件が不明

多くのIT担当者は、アプリケーションの具体的なテクノロジーの要件のみを考慮しがちですが、この見方は近視眼的であり、アプリケーション開発プロジェクトに将来の問題を多数持ち込むこととなります。アプリケーション開発者、DBA、データベース設計者にとって特に重要なのは、アプリケーションのビジネス価値全体と、企業がそのアプリケーションを使用することになる状況を理解することです。この状況に関する情報がないと、IT担当者はおそらく、以下のようなさまざまな問題をアプリケーションに持ち込むこととなります。

1. **不十分な開発テスト:** これは、プロジェクトマネージャが、特定のスケーラビリティの目標に到達するために必要な期間について過度に楽観的な予測を行い、それを基に開発プロジェクト全体の予定を立ててしまうシナリオです。アプリケーション開発者には、通常、機能上の要件のみが示されます。たとえば、「エンドユーザは画面の各フィールドに値を入力し、次に[OK]をクリックして新しいレコードを認めるか、[キャンセル]をクリックして新しいレコードを中止します」のような要件が示されます。アプリケーション開発者が実行する必要のあるテストは、記述したコードが機能上の要件を満たしていることを確認するための、ユニットテストのみです。機能上の要件はシステムのスケーラビリティの要件を考慮していないため、アプリケーション開発者は、現在のコードユニットがエラーなしにコンパイルされ、適切な結果が返されれば、問題なくテストを中止して次のコードユニットのコーディングを開始することとなります。これらの種類の要件は、次の2つの方法で補強する必要があります。
  - a. 1 つ目に、「ユーザは[OK]または[キャンセル]をクリックしたら 1 秒以内にトランザクションが終了したことを通知されます」のように、パフォーマンスの目標を要件に含める必要があります。
  - b. 2 つ目に、要件にスケーラビリティの要件を含める必要があります。たとえば、次のように定義します。「40 人までのユーザが同時にシステムを使用している場合、ユーザは[OK]または[キャンセル]をクリックしたら 1 秒以内にトランザクションが終了したことを通知されます。41 人から 80 人のユーザが同時にシステムを使用している場合、フィードバック時間は 2 秒までとします。81 人以上のユーザがシステムを使用している場合、フィードバック時間は 5 秒を超えてはなりません。」

2. **パフォーマンス不足:** 不十分な開発テストの当然の結果として、アプリケーションの展開後は通常、予期されるよりずっと早い時期にアプリケーションのパフォーマンス不足が確実に発生することになります。システムが 1 人のユーザ(開発者自身)についてのみテストされた場合、データバッファのメモリ負荷、プロセスバッファのメモリ負荷、ロックの問題、ブロックの問題などのさまざまなパフォーマンスの問題が見過ごされることになります。Gartner 社の最近の調査では、データベースアプリケーションのパフォーマンスの問題のうち、80%はパフォーマンスが低い SQL ステートメントが原因であることが明らかになりました。その結果として、データやデータ構造の操作と取得など、SQL が関係するすべての面でパフォーマンスのボトルネックが発生する可能性があります。
3. **追加の QA が必要:** 新しいアプリケーションを開発する際に、ほとんどの IT 企業は納期を設定することに関して常に楽観的です。そして、ほとんどの場合、開発プロセスの中で調整の対象となるのは品質管理(QA)のフェーズになります。わたしは、プロジェクトマネージャが、アプリケーションの範囲や機能を減らすのではなく、製品の QA フェーズに費やす時間を減らすことによって、遅れているプロジェクトの納期を満たそうとする場面を何度となく見てきました。これは、特に開発プロセスの開発フェーズにスケーラビリティのテストが含まれていないため、自滅的な対応です。結果として QA チームは、アプリケーションが想定どおり動作することを確認するだけでなく、さまざまなレベルのスケーラビリティで想定どおり動作することを確認する作業も担当することになります。皮肉なことに、無理な納期に合わせるために急いで作業することによって、QA チームでは十分なテストを行うためのより多くの時間が必要になっています。それに加えて、開発チームには、QA チームによって明らかにされた結果のすべてに対応するための時間が必要であるという事実を考慮しないで、プロジェクトマネージャがプロジェクトの予定を立てることがよくあります。このため、開発プロセス自体が原因となって、予期される負荷でアプリケーションが実際に機能することを保証できる人がプロジェクトにはいないため、アプリケーションは既に失敗する運命にあります。
4. **予測:** スケーラビリティに関するテストがない場合や不十分なテストしかない場合、その延長として、容量の計画や将来の要求に対する予測が存在しないという最後の問題がアプリケーション開発プロジェクトに持ち込まれます。現在のハードウェアと OS の構成でアプリケーションが想定したとおり動作可能だとは保証できないため、どの時点で、より多くのハードウェアや構成の変更が必要になるか、確信を持つことができません。この結果、多くの IT 企業はより多くのハードウェアを「投入する」という誤った対処を行うことが多いパフォーマンスの問題の根本原因について絶えず推測することになります。次のような事例を多く目撃しました。あるアプリケーションを展開後、想定したおりのパフォーマンスが出ませんでした。チームの最初の対処は、より多くのメモリ、ハードディスク、CPU などを追加することでしたが、原因はアプリケーションのロジックであり、ハードウェアの量では問題を完全には解決できませんでした。こうした状況では、多くの場合、アプリケーション開発チームはプロジェクトのスケーラビリティの要件を満たすために、アプリケーションの多くの部分を記述し直すという選択肢しかありません。

この問題を解決するためには、いくつかの手順があります。まず、アプリケーション開発者とデータベース設計者が、スケーラビリティとパフォーマンスの目標を含む機能要件を受取ることを要求する必要があります。少なくとも、アプリケーション開発者やデータベース設計者がその時々に取り組んでいる機能仕様に落とし込めるように、予想される1秒当たりの(またはその他の時間間隔での)同時ユーザ数とトランザクション数でアプリケーション全体を評価する必要があります。仕様を書くプロジェクトマネージャやシステムアナリストがこれらの質問に対する答えを把握していない場合は、経営者からの説明をもう一度受けてもらう必要があります。

次に、データベース管理者が、システムの負荷について全般的な種類(たとえばOLTPなのかOLAPなのか)について知っているだけでなく、月次レポートの処理サイクルなどの重要なビジネスサイクルを含めて、使用のピーク時間についても知っている必要があります。これが、システムハードウェアの適切な利用計画を作成し、インデックスの再構築のような予防的な保守タスクを計画する助けになります。

最後に、チームはプロジェクトで作業しているアプリケーション開発者が、コードのテストの規模を拡大する手段を持っていることを確認する必要があります。これには、データベースの運用バージョンやテストバージョンで発生するすべてのTransact-SQLステートメントを記録するなどの手順が含まれる場合があります。開発者は後でこの記録を何度も再生し、実行ごとに1人または複数の同時ユーザの利用状況をシミュレートすることができます。このようなスケーラビリティとパフォーマンスのテストを開発者の定例テストの一環とすることで、チームはアプリケーションにおいて多くの問題が回避され、アプリケーションのライフサイクルの初期に深刻なパフォーマンス上のボトルネックが生じないようにすることができます。

## 9. 不適切に正規化されたテーブル

スケーラビリティの要件が不明であることがアプリケーションの問題であるように、不適切に正規化されたテーブルが、データベース設計の問題点です。テーブルが適切に正規化されていると、新しいアプリケーションのデータベース設計が堅固な基礎に基づいていることとなります。これにより、冗長なデータが発生する確率が低下し、少ないミスで高品質のデータを取得でき、厳密に設計された論理的な構造を通して優れたパフォーマンスが実現されます。

正規化については、詳細な解説はこのホワイトペーパーで取り上げる範囲を超えていますが、順を追って簡単に説明します。正規化とは、具体的には、データの重複を減らし、データ整合性の点で一貫性が損なわれている状態に至る状況を減らすような「正規の形式」になるようにテーブルの分析と設計を行う方法を指します。テーブルは最も論理的な構造に縮小され、テーブル内の各レコードは「主キー」によって一意に識別されます。関連するテーブルは、「外部キー」のリレーションシップによって相互に相関関係が設定されます。

ただし正規化は、完全に厳密なデータベース設計を作成するときに必要な多くの手順のうちの一歩目です。したがって、特定のデータベースで行われた正規化の品質は、多くの場合、そのデータベースの設計全体の品質を予告するものとなります。このため不適切な正規化は、しばしば次のような他のさまざまな問題を示す場合があります。

- インデックスの設定が不適切であるか行われていない
- 外部キーのリレーションシップが不適切であるか存在しない
- 命名規則が不適切であるか存在しない
- テーブル内の列のデータ型が適切に選択されていない

適切に正規化できるかどうかは、単にトレーニングと経験の問題です。データベース設計で正規化が行われていない場合、そのデータベース自体の品質と経験が不足していることがわかります。そして当然、さまざまな問題が後で発生します。

## 8. インデックスの問題

データベースが論理的に設計され、適切な正規の形式でレイアウトされていても、実行する必要がある物理構造の設定が多数存在します。このプロセスでは、データベース内の各テーブルの列にインデックスを作成する必要があります。インデックスは、クエリオプティマイザに非常に選択的なオプションを提供し、システムにおいて全体としてのIO負荷を削減することによって、トランザクション処理を大幅に高速化します。(インデックスを適切に設計する方法に関する詳しい推奨事項については、わたしが書いた他のホワイトペーパーで確認してください。)

DBAと開発者は、データベースにインデックスを設定するジョブを綿密に実行する必要があります。そのようにしないと、以下の問題が発生します。

- **クラスタ化インデックスや主キーが存在しないか不適切であるためにIOが過剰:** クラスタ化インデックスは、SQL Serverデータベース内のすべてのテーブルにとって特に重要です。クラスタ化インデックスは、ディスクにレコードを書込む正確な物理的順序をSQL Serverに指示します。ただし、クラスタ化インデックスがないテーブルや、不適切なクラスタ化インデックスがあるテーブルを持つデータベース上に多くのアプリケーションが作成されます。これらは常にではなくても、通常は各テーブルの主キーに割当てられます。しかし、SQL Serverでは主キーがないテーブルを作成できるため、テーブルに主キーもクラスタ化インデックスもない場合があります。これは問題のある考え方ですが、一般的に見られる状態です。このようなアプリケーションのパフォーマンスが特に低いことは不思議ではありません。クラスタ化インデックスの設計についての詳細な説明はこのホワイトペーパーの範囲を超えますが、クエスト・ソフトウェアやインターネットからは、データベース内のテーブルすべてのために最適なクラスタ化インデックスを設計する助けになる他のホワイトペーパーを入手できます。
- **ページ分割が多すぎる:** クラスタ化インデックスはディスクにデータを書込む方法を定義します。クラスタ化インデックスが適切に定義されていない場合や存在しない場合(クラスタ化インデックスがないテーブルは「ヒープ」と呼ばれます)に、SQL Serverで不要な読取りや書込みが多数発生することがあります。また、テーブルを構成する8kのデータページが増えるときに、SQL Serverでは1つのテーブルを新しい2つのページに「分割」する必要が生じることがあり、さらに多くのIOオーバーヘッドが発生します。この問題は、インデックスが作成、再作成、デフラグされるときのFILL FACTORをテーブルに割当てることによって回避できます。
- **不適切な非クラスタ化インデックスのためにクエリが低速:** クラスタ化インデックスはすべてのテーブルに必要ですが、アプリケーションの最適なパフォーマンスのためには非キー列の非クラスタ化インデックスが必要です。非クラスタ化インデックスを設定するのに適した場所には、外部キーのフィールド、よく選択される日付フィールド、IDフィールド、SQLトランザクションのWHERE句でよく使用される任意の列が含まれます。また、特定の種類のクエリを大幅に高速化する「カバリングインデックス」と呼ばれるものを作成できます。ただし、テーブルに非クラスタ化インデックスが存在しないか、あってもその数が少ないデータベースを見かけることはよくあります。結果として、そのようなデータベース上に構築されたアプリケーションは優れたパフォーマンスを発揮できません。

DBAと開発者の多くが忘れていた別の問題は、インデックスに関する統計を管理する必要があることです。インデックスは、そのインデックスに関連付けられているテーブルにより多くのトランザクションが適用されるにつれて「古く」なります。結果として、データベースの統計を最新の状態に維持することが非常に重要になります。それには、SQL Serverの統計の自動更新機能を有効にすること(デフォルトで有効

になっています)と、統計を頻繁に(毎晩など)更新するDTS/SSISジョブを作成することの両方を実行します。

また、SQL Serverでは、FILL FACTORが自動管理されないことは広く知られていません。そのため、テーブルでクラスタ化インデックスを作成する適切な作業を終えて、それらのインデックスにFILL FACTORが割当てられている場合、時間が経過すると次第にそのFILL FACTORが劣化していきます。このため、それらのテーブルのFILL FACTORを定期的に再確立するDTS/SSISジョブを作成することが重要です。

## 7. ベースラインまたはベンチマークがない

アプリケーションのスケラビリティ要件を把握していないことにつきものなのは、アプリケーションがQAに渡された後、実際にどのようなパフォーマンスであるかを追跡して調査する作業を行わない、という失敗です。DBAと開発者の多くが犯す大きな間違いは、サーバの負荷がどれほどあるかを基本的に知らなかったり、ピーク負荷がどの程度になるかがわからないことです。ベースラインはアプリケーション全体(フロントエンドのコード、SQL Server、ハードウェア、OS構成)が「通常の」状況でどのように実行されるかを示すパフォーマンステストです。ベンチマークテストは、定義済みのワークロードでアプリケーション全体がどのように実行されるかを示す追加のパフォーマンステストです。たとえば、アプリケーションが30人の同時ユーザをサポートすると想定されている場合、30人の同時ユーザをシミュレートするベースラインを実行する必要があります。しかし次に、60人、120人、300人の同時ユーザで追加のベンチマークを実行する必要もあります。

ベースラインと、1つ以上のベンチマークを実行することで、アプリケーションの動作状況と、負荷がかかった状態でのパフォーマンスプロファイルの予測について適切な洞察を得ることになります。しかし、これらの知識がないと、おそらく以下のような問題が発生します。

- **問い合わせ件数のみでユーザの利用状況を管理する:** 通常時にアプリケーションがどのように動作するかを把握していないと、ユーザから連絡があって不満を伝えられたかどうかだけが、問題の有無を判断できる方法になります。これは多くの問題につながるため、大きな誤りです。
  - **常に問題が起こった後に対処している:** ユーザからの問い合わせに常に対応する必要がある立場にいる場合、「緊急が最優先」になって、問い合わせに答える以外の時間がまったくなくなります。
  - **ユーザがシステムを使用していないときのエラーを把握できない:** たとえば週末など、不満を言うユーザがいないと、問題が発生したことさえわからない場合があります。たとえば、問題が土曜日に発生した場合、月曜の朝に出勤して初めて問題を発見することになるため、問題が解決されるまですべてのユーザを待たせることになります。ユーザ体験を管理する適切なシステムを実装していれば、週末の間に、問題があったことをユーザが知る前に問題を検出して修正することもできました。
  - **パフォーマンスに影響しない、とらえにくいエラーを把握できない:** ユーザ体験だけがシステムの全体のパフォーマンスを測定する基準である場合、バックアップの失敗やスケジュールされたタスクのエラーなど、すぐにわからなくても同様に重要なエラーを把握できません。

- **サービスレベル契約 (SLA) を満たせない:** 重要なアプリケーションはサービスレベル契約によって管理されることがよくあります。SLAは、アプリケーションのサービスが中断されてもかまわない時間を示します。長い経験を持つエンタープライズDBAからの重要なヒントがあります。満たすことができると100%自信がないSLAには決して同意しないようにします。完全なベースラインとベンチマークのセットがなければ、その確信は得られません。必ず必要なのは、SLAの要件を満たせることを示す量的な証拠に基づいてSLAのサポートを行うことです。
- **動作を予測できない:** アプリケーションが示す動作を将来にわたって予測できるようにすることが非常に重要です。長期 (今から数年間) と短期 (数週間) の両方について予測できる必要があります。特に、「週、月、四半期の終わりにビジー状態になるか」などを知る必要があります。また、「主な休日の後の状況」や「昼食後にビジー状態になるか」なども知る必要があります。周期的であっても通常のものであるビジネス周期にアプリケーションがどのように応答するかを知らないと、通常の状態であるのに、影響を与えることができない問題事例だと解釈する可能性があります。
- **問題の修正にどこから取り掛かるかがわからない:** ベースラインやベンチマークを実施していない場合、パフォーマンス上のボトルネックが最初に発生する場所の見当が付きません。パフォーマンスの問題が実際に発生したときには、どこから取り掛かるか適切な案がないこととなります。そのような場合、合理的であろうとする人は問題により多くのハードウェアを投入しようとしています。これらの人は、アプリケーションがどのように動作するかを把握していないため、ハードウェアの量では問題のあるコードは修正されることがわかりません。問題に対してハードウェアを投入する開発者やDBAは、最善の場合でも、パフォーマンス上のボトルネックを数カ月先延ばすだけになります。しかし、問題の原因は不適切なSQLコード、アプリケーションコード、データベース設計なので、遅かれ早かれ、ハードウェアでは問題を完全に解決できなくなるでしょう。

ベースラインとベンチマークを実行することにより、サーバで他のアプリケーションと共有することができる負荷を判断することもできます。たとえば、SQL Serverのハードウェアで他の多くのサービスを実行できます。SQL Serverのハードウェアに他のサービスを追加したい場合が多いのは、SQL ServerにはIISなど、その他のWindowsサービスが多数付属しており、使用されるからです。結果的に、ベースラインとベンチマークを併用する場合のみ、サーバでアプリケーションとサービスを組合わせた負荷を支えることができるかどうかを判断できます。

## 6. ディスク – 領域について考慮しても IO が考慮されていない

DBA、アプリケーション開発者、データベース設計者が犯す別の大きな間違いは、サーバディスクをデータ量の観点でのみ検討し、ディスクサブシステムで維持する必要があるIO量の観点では考えないことです。この知識がないと、以下の問題が発生します。

- **IOが不十分:** これまでに説明した間違いの多く (スケーラビリティの要件を把握していない、ベースラインやベンチマークを行わない) は、その結果として、DBAや開発者が不適切なディスクサブシステムを展開する原因になります。良好なパフォーマンスのためには、ディスクサブシステムでアプリケーションの要求をサポートできる必要があります。ほとんどのOLTPアプリケーションでは時間当たりのトランザクション数が多いことが必要で、OLAPアプリケーションでは時間当たりの転送速度が高いことが必要です。ディスクメーカーは、そのディスクで達成できる1秒当たりの最大トランザクション数と、最大転送速度 (MB/sec) の両方の情報を公開しています。ディスクIOを改善する方法は多数あります。

- 最初の最も重要な点は、データベースファイルとトランザクションログファイルは、別個の物理ディスクまたはRAIDアレイ上に配置する必要があります。
  - 2番目に、データベースファイルとトランザクションログファイルのサイズが十分大きく、通常の毎日の運用中に自動拡張が行われないことを確認してください。これは、終了するまでユーザを待たせることになる非常に低速な動作です。
  - 次に、パフォーマンスがまだ不十分である場合は、他のチューニングオプションを検討してください。たとえば、一時データベースを別の専用物理ディスクに配置したり、サーバ上に存在するCPUと同じ数だけ、サイズが等しいファイルが存在するようにデータベースやトランザクションログファイルを追加したりします。
  - 最後に、パーティション分割や、特定のファイルにインデックスを分離するなどのチューニング手法を使用します。これらの手法は、高い拡張性が必要なシステムにとって特に有用な場合があります。
- **フォールトトレランスが不適切:** フォールトトレランスはSQL Serverハードウェアの最も一般的な問題の1つです。ハードディスクが1台しかないサーバでは、ディスクの障害によってシステム全体が停止することになります。このため、ほとんどすべての運用データベースアプリケーションにはRAID (Redundant Array of Inexpensive/Independent Disk) が必要です。複数のハードディスクをRAIDに構成するには、いくつかの異なる方法があります。ただし、RAIDの主なメリットの1つは、通常は1台、場合によっては複数のディスクに障害が発生した場合でもSQL Serverの動作を継続できることです。
  - **RAIDの種類が不適切:** RAIDディスクサブシステムの選択により、SQL Serverのフォールトトレランスを大幅に改善できます。ただし、RAIDの種類が異なればIOのメリットとデメリットの種類も異なります。たとえば、RAID 5 (パリティ付きストライプディスク) は安価で、読取りIOの実行がかなり高速です。しかし、RAID 5の書込みIO操作は周知のように低速です。一方RAID 1 (ミラー)とRAID 10 (ストライプ化ミラー) は読取りと書込みのどちらも高速ですが、より多くのコストがかかります。
  - **ディスクスピンドル数が十分でない:** RAIDにディスクを追加するごとに、RAIDで支えることができるIOの数が増加します。アプリケーションのIOの要求に対して、RAID内に用意した別個の物理ディスク (つまりスピンドル数) の数が非効率であるのはよくある間違いです。
  - **コントローラやチャンネルの使い方が不適切:** 多くのディスクコントローラは2つのチャンネルと書込みキャッシュを備えています。標準的なサーバ用ハードディスクコントローラで両方のチャンネルを利用しないのはよくある間違いです。一方コントローラには、古いものの場合には特に、バッテリーバックアップがない書込みキャッシュを備えているものがあります。動作するバッテリーバックアップがないハードディスクコントローラで書込みキャッシュを有効にするのは大きな誤りです。
  - **SANを万能だと考える:** 多くのIT企業がSAN (Storage Area Network) を導入しています。SANの製造元は、ディスクIOのすべての要求に対する解決策としてSANを売込むことに成功しています。しかし、これはもちろん正しくありません。SANにはRAIDアレイにあるのと同じ問題がすべてあります。たとえば、多くのSAN管理者がたずねるのは必要なディスク領域のことだけで、必要なIOスループットについて検討し直すことはありません。また、SAN管理者は希望する任意の種類RAIDを使用してSAN上にドライブを定義することができます。したがって、アプリケーションにとって不適切な種類のRAIDを選択することがあ

ります。このため、SANは他のディスクソリューションと同様に扱い、万能だと見なさないように注意してください。

## 5. エラーの通知

SQL Serverには、エラーが発生したときにDBAや他の主要スタッフに通知を行う簡単で容易な手段が用意されています。しかし、多くの企業がこの機能を有効にしたことがないのはなぜでしょうか。こうなった理由は正確にはわかりませんが、ITスタッフが単にこの機能のことを知らないことが理由の1つです。この結果次の問題が発生します。

- **エンドユーザに問題を通知してもらう必要がある:** 7で説明したように、問題の通知をエンドユーザに依存している場合は常に、その保守と管理の方法は不適切であると考えられます。最大の問題点は、ユーザがシステムを使用していないときに発生する問題を見逃す確率が非常に高いことです。結果として、問題が勤務時間外に発生した場合は、診断とトラブルシューティングの貴重な時間が無駄になります。重要なアプリケーションの担当者であれば、特にアプリケーションがミッションクリティカルなものである場合には、ユーザからのクレームに頭を悩ませることになるでしょう。したがって、SQL Serverに組み込まれている機能を活用し、エンドユーザがアプリケーションのパフォーマンスや動作について問題を見つけてしまう機会を最小限にする必要があります。
- **スケジュールした重要なジョブが介入なしに失敗する:** SQLエージェントを使用して多くの重要なジョブが作成され、自動化されます。SQLエージェントを使用すると、非常に複雑な操作を、自動で実行したり設定した任意のスケジュールで実行することができる複数ステップのジョブを作成できます。たとえば、夜間にバックアップと一定のDBCCチェックが実行されるように設定することはよくあります。ただし、エラー通知を有効にしていないと、自動化したジョブの状態を調査しに行くまで問題にまったく気付かない場合があります。わたしは、次のような場面に数え切れないほど遭遇してきました。DBAはデータベースの復旧が必要なのですが、バックアップをまったく、またはほとんどチェックしたことがないため、復旧時に初めて最後の正常なバックアップが1カ月以上前のものであることに気付くのです。これは解雇されてしまうような種類の誤りです。
- **管理対象のサーバをすべて手動でチェックする必要がある:** エラー通知を利用していないIT企業の多くは、社内にあるサーバが1台または数台です。このような構成は、理想的ではありませんが、多くの場合は管理可能です。しかし、IT企業が大きくなり、より多くのSQL Serverが展開されると、予防的なエラー通知システムを持っていない環境は管理不能になります。この場合、IT担当のDBAまたは開発者が、管理対象のサーバすべてを手動でチェックする必要があります。サーバで何もエラーが見つからない場合でも、各サーバをチェックするにはかなりの時間がかかる可能性があります。また、修正する必要があるエラーがあれば、何時間もかかってしまうことがあります。サーバが多ければ、必要な時間も長くなります。最終的には、サーバの数が、1日でチェック可能な数を超えたとき、この仕組みは破綻することになります。それよりも、SQL Serverでエラー通知システムを有効にして、エラーがあるときには通知されるようにしてください。こうすれば、サーバが自身を監視し、問題があれば管理者に通知してきます。
- **正確に事前に予防することは不可能:** サーバを注意深く観察しなければ、サーバの潜在的な問題に対する予期と準備を行うことがより困難になります。それどころか、破損したコード、失敗したSQLエージェントのジョブ、時間内に応答しないアプリケーションに対応すること

にすべての時間を費やすことになります。この問題に直面するよりも、ユーザが問題に気付く前に対応できるようにイベント通知をインストールしてください。

複数のサーバがある大企業で利用できる別の方法として、SQL Serverには「イベントの転送」と呼ばれる機能も用意されています。イベント転送は、1台以上のサーバでイベントとエラーをとらえ、1台の中央サーバにそれらを転送する組み込みの機能です。このサーバが次に、エラーが発生したことを開発者やDBAに通知します。DBAは非常に規模が大きいSQL Serverインフラストラクチャで、エラーの予防的な監視を非常に簡単に行えるようになります。

## 4. バックアップ

バックアップの重要性は誰もが知っています。しかし、IT企業が運用データベースを定期的、そして組織的にバックアップしていないことはよくあります。これは非常に厄介で、職への影響も避けられない可能性があります。

しかし、動作の確認をとっていない方法でバックアップを実施することは、バックアップをまったく行わないことと、ほとんど変わりません。最終的には同様の結果をもたらします。つまり、障害発生後にアプリケーションを復旧できません。しかも、IT企業とエンドユーザは障害が起きた場合でもすべて大丈夫であるという誤った安全意識を持っています。間違い4は、バックアップのテストと完全なテスト復旧を実行し、障害復旧の状況ですべてが正しく機能することを確認していないことです。この作業を行っていないと、次の問題が発生します。

- **バックアップが適切であることが確実でない:** 障害復旧手順の中で簡単かつ重要な手順は、バックアップが常に適切であり、常に使用できるようにすることです。ほとんどのSQL Server DBAは、データベースの自動バックアップを複数の手順から成るジョブとして作成します。最初の手順では、データに対して最も重要なDBCCチェックを実行し、データの破損がないことを確認します。2番目の手順では、ローカルディスクサブシステム上の場所に検証を行ってデータベースをバックアップします。3番目の手順では、データベースのバックアップをネットワーク上のリモートの場所にコピーし、バックアップを長期間利用できるようにします。データベースは正常にバックアップされたのに、バックアップ処理の一環としてDBCCチェックが行われなかったため、バックアップ自体が壊れていた状況を見てきました。他の状況では、バックアップ自体は正常であったのに、クラッシュしたローカルディスクに置かれていて、復元不能でした。
- **重要なファイル入手する場所が明確でない:** 障害復旧のシナリオでは多くの場合、SQL Serverデータベースだけでなく、サーバ全体や、アプリケーション自身までもが復旧対象になる場合があります。サーバ全体の復旧がテストされていないため、DBAとサポートスタッフが、正しいバージョンのSQL ServerとそのService Packや修正プログラムはもちろん、DLL、MDACファイル、正しいOSとそのService Packや修正プログラムなども必要であることを認識していない状況に出会ったことがあります。データベースを最短時間で復旧しようとしているときに、重要なファイルが存在しないために、そのファイルを探し出す必要があると気付くときほど悲しい状況はありません。

復旧について覚えておく必要がある重要なことは、HIPAAやサーベンスオクスリ法などの新しい法的要件のため、長期的なアーカイブは以前にも増して重要である点です。結果として、何年か経過したアーカイブからSQL Serverデータベースを復旧するテストも実施する必要があります。数年経過したアーカイブはテープに格納されていて、テープドライブがもう存在しなくなっていることに気付く場合も珍しくあり

ません。すべてが正常に機能する状態にあって、完全な復旧が可能かどうかは、テストによってのみ明らかになります。

### 3. クエリのチューニング処理

前に触れたように、アプリケーションのパフォーマンスに関する問題の大多数は、不適切に記述されたSQLステートメントが原因です。しかし、適切なSQLコードが非常に重要であるのに、多くの開発者とDBAはSQL Serverツールキットのツールを使用してパフォーマンスをチューニングし、コードを最適化する方法を知りません。驚くほど多くのアプリケーション開発者とDBAはこのツールに精通していませんが、これに精通していないと、次の問題が発生します。

- **使用するツールがわからない:** SQL Server 2000やSQL Server 2005には、問題があるSQLステートメントを検出してステートメントのチューニングを行うことが明白な目的であるツールが多数用意されています。ホワイトペーパーでは各ツールの詳細な説明は行いませんが、SQLステートメントを記述するすべてのアプリケーション開発者とDBAは、次のツール使用方法を把握しておく必要があります。
  - Windowsパフォーマンスモニタ(PerfMon)
  - SQL Profiler
  - グラフィカルなプラン表示とSET SHOWPLAN
  - SET STATISTICS IO
- **クエリプランの読み方がわからない:** クエリプランは、旅行のときの案内付きの地図のようなもので、SQLステートメントにとって非常に重要なものです。クエリプランでは、SQL ServerがどのようにSQLステートメントを解決したかが正確に示されるので、ステートメントの各ステップの明確な内容が示されます。クエリプランの読み方を理解することは、アプリケーション開発者やDBAが処理内容を知るための唯一の手段です。データがキャッシュから取得されたか、ディスクから読取られたか、最適なインデックスが使用されたか、適切な結合方法が選択されたか、高速なインデックスシークの代わりに遅いテーブルスキャンが使われたかなどがわかります。
- **パフォーマンスを向上させる代替のプランを提案したり試したりできない:** クエリやSQLトランザクションのパフォーマンスを向上させるためには、同じ情報を要求する代替の方法を試して、より高速で優れたクエリプランが提供されるかどうかを確認する必要があります。しかし、クエリの処理について理解していない人は、WHERE IN句の代わりにWHERE EXISTS句を使用するなどの重要なテクニックや、COALESCE関数を追加するだけでクエリを大幅に高速化できることを知らないでしょう。
- **アプリケーションが数人を超えるユーザに対応できない:** 不適切なSQLによって生じる別の主な問題は、ロッキングとブロックの問題です。ユーザが操作のために1つ以上のレコードを確保すると、そのレコードはロックされ、そのレコードを要求している他のユーザはレコードを待機する必要があります。待機しているユーザは、レコードを確保しているユーザがレコードに対するロックを解除するまで「ブロック」されていると表現されます。ただし、不適切に記述されたSQLステートメントが、1つまたはいくつかのレコードだけを排他的にロックするのではなく、テーブル内のすべてのレコードをロックする場合があります。結果として、開発者が適

切なトランザクションプログラミング技法に従っていないため、このようなアプリケーションが1人または少数を超えるユーザに対応できることはまれです。クエリのチューニング処理について知り、自分のSQLステートメントに知識を適用することによってのみ、開発者やDBAはアプリケーションが多数の同時ユーザに対応可能だと確信することができます。

- **問題に際してさらに多くのハードウェアを投入する:** 前に触れたようにSQLコード内の問題を特定して診断する方法を知らない場合は、パフォーマンスが上がらない状態のサーバにより多くのハードウェアリソースを追加することになるでしょう。しかし、SQLコードが不適切な場合、これが問題の解決になるでしょうか。根本原因である不適切なSQLステートメントに焦点を当てて修正していないため、ほぼ確実に解決にはなりません。

これらの理由で、クエリのチューニング処理に関するすべてを学び、それを適用する方法と機会についても理解することが非常に重要です。

## 2. 内部構造に関する知識不足

クエリをチューニングする方法を知らないことから当然、ストレージエンジン、クエリエンジン、メモリ管理など、SQL Serverの内部構造に関する初歩的な原理について知識がないと推測されます。これらの知識がないと、おそらく以下のような問題が発生します。

- **問題のあるコード:** システムの内部構造について知識がないアプリケーション開発者やDBAは、通常、品質の低いTransact-SQLプログラム、DTS/SSISジョブ、SQLステートメントを記述し、それらを運用環境に配置します。このようなコードには、IOが多い、ロックやデッドロックが多い、インデックスを利用できるときに利用しないなどの問題があります。
- **誤ったアラーム:** システムの内部構造について知識がないアプリケーション開発者やDBAは、チェックポイント処理やレイジーライタの影響を理解していません。チェックポイント処理とレイジーライタは規則的な間隔で処理が増加する傾向があるため、アクティブなときにはリソースの消費が急激に上昇します。これらの急上昇は実際には普通のことなので何も心配することはありません。しかし、経験が浅い人たちの多くは、PerfMonカウンタでこれらの急上昇を見ると必要以上に動揺します。
- **機会を逃す:** 経験が浅い人たちは、キャッシュから読み込む確率が最大になるコードを記述する機会を逃すことがよくあります。データはバッファキャッシュから読み込まれますが、多くの種類のTransact-SQLコードはプロシージャキャッシュから読み込むことができます。しかし、キャッシュからの読み取りを容易にするテクニックと、反対に、キャッシュが無視されるようにするという2つのテクニックがあります。たとえば、SQLデータ定義言語のステートメント(CREATE、ALTER、DROP)と、SQLデータ操作言語のステートメント(SELECT、INSERT、UPDATE、DELETE)を交互に実行することで、ステートメントが切替わるたびに(各実行内ではおそらく何度も)ストアプロシージャを強制的に再コンパイルできます。同様に、ハードディスクのIOは通常、キャッシュより大幅に低速であるにもかかわらず、ディスクIOを最適化しない場合があります。たとえば、経験が浅い人たちは、パーティション分割やファイルの適切な分散によってディスクIOを最小限にするための、内部構造についての知識を持っていません。

SQL Serverの内部構造については、優れた情報源が多数存在します。まずは、多数のWebキャストやホワイトペーパーが用意されている、MSDNとTechNetのWebサイトを参照してください。

## 1. カーソル

カーソルは、リレーショナルデータベースに備わる興味深いSQLの機能で、データベースがデータセット全体をレコードごとに処理できるようにする機能です。この機能によって開発者は、結果セットの操作時に大きなメリットを得ています。C#やVB.Netなどの得意なプログラミング言語で慣れているのと同じ、反復処理のスタイルで結果セットを操作できるためです。開発者はまた、カーソルのおかげで、優秀なデータベースプログラマに求められる、結果セットを使って考えるという思考様式の切替えをしないですみます。

残念なことに、SQL Server(とSybase Adaptive Server)のカーソルは、他の多くのデータベースプラットフォーム、特にOracleとは大きく異なる方法で実装されています。Oracleでは、カーソルと通常のSELECTステートメントの間にパフォーマンス上の違いはありません。これは、どのSELECTステートメントも内部ではカーソルとして実現されているからです。しかし、SQL Serverでのカーソルの処理は、通常のSELECTステートメントの処理とは大きく異なっています。SQL Serverでは、カーソルは独自のロックプロセスを持ち、独自のメモリの問題があります(ライフサイクルの最後の段階でカーソルのDEALLOCATEを実行する必要があります)。そして、カーソル独自のパフォーマンスの問題もあります。

カーソルは小さなデータセット、たとえば10,000件未満のレコードを操作する場合には何の問題もありません。しかし、カーソルをどのような場合に、どのような方法で使用するかは慎重に選択することをお勧めします。一般的には、バックグラウンドの管理用プロセスのためにのみカーソルを使用する必要があります。カーソルを標準的なプログラミング構成要素として、頻繁に呼出されるストアドプロシージャやユーザ定義関数内で使用しないでください。

## 結論とまとめ

SQL Serverアプリケーションではさまざまな問題を経験する可能性があります。このトップ10リストに示したものを含めて、ほとんどの問題は知識不足が原因です。データベース設計者、アプリケーション開発者、データベース管理者としてのキャリアを開始するときには、生涯、学習し続けることになると考えてください。学習するという概念の奥行きと幅広さは、最初は非常に困難に思えるかもしれませんが、すべては論理的かつ合理的であり、多くの場合には直感的でもあります。言換えれば、すべて筋が通っています。SQL Serverについてできるだけ多く学べば、SQL Serverに関してよくある間違いのトップ10を避けられるでしょう。

## 著者について

Kevin Klineはクエスト・ソフトウェアの、SQL Serverソリューションを専門とする技術戦略マネージャ (Technical Strategy Manager)です。クエスト・ソフトウェアは、SQL Serverプラットフォームでデータベースの管理とアプリケーションの監視を行うための、受賞歴があるツールを提供するリーダー企業です。Kevin Klineは国際的なSQL Server専門家の団体であるProfessional Association for SQL Server (PASS - [www.sqlpass.org](http://www.sqlpass.org))の代表者です。また、2004年以来Microsoft SQL Server MVPであり、『SQL in a Nutshell』(<http://www.oreilly.com/catalog/sqlnut2/>)の主著者、『Professional SQL Server 2005 Database Design & Optimization』(<http://www.apress.com/book/bookDisplay.html?bID=10005>)と『Database Benchmarking』([http://www.rampant-books.com/book\\_2007\\_1\\_database\\_benchmarking.htm](http://www.rampant-books.com/book_2007_1_database_benchmarking.htm))の共著者です。Klineは『Database Trends & Applications』誌と『SQL Server Magazine』誌のために毎月コラムを執筆し、SQLBlog.comとSQLMag.comでブログを公開しています。最高の評価を受けている話し手でもあり、Microsoft TechEd、PASS Community Summit、Microsoft IT Forum、DevTeach、SQL Connectionsなどの国際会議に参加しています。仕事で髪の毛をかきむっていないときは、4人の子供と過ごしたり、花と野菜の庭で過ごす時間を楽しんでいます。