Climb Inc.
Growing to Meet Your Needs

データベース・パフォーマンス管理
タイム・ベース分析の重要性
Igniteについての考察

株式会社クライム

データベース・パフォーマンスのモニターが必要な時に考慮すべき次の 3 つの要素があります。

- 何を計測するか？
- それをどこまで深く計測するか?
- どんなコンテキストをその測定に適用するか？

何を計測するか？

パフォーマンス問題に関してデータベースをモニターする従来の手法はデータベース内で起こったことをモニターすることです。これはデータベース・ベンダーが通常提供するユティリティで、また多くのサード・パーティ製品が使用する手法です。このアプローチはデータベース内で起こったことをすべてカウントする手法です。例えば特定の時間内での物理リード、ライト数、ロック統計などです。

しかし隔離環境ではこれらの計測は何の意味もありません。例えば 19.2m リードは大きいでしょうか？それともたいしたことがないでしょうか？もちろんこの情報をトレンド分析に適応させ、この時間内でのリード数と他の時間内での数と比較することができます。しかしこれはどのくらい役に立ちますか? ユーザが異常に高いリード数があったとしてもそれがパフォーマンスに影響するかどうか分からないし、そうであってもパフォーマンスを落とす最も大きな要因かどうかわかりません。さらに逆にパフォーマンスに影響するのがデイスク・リード数であっても、何がデイスク・リードの増加の原因か分かりません。それは他のシステムの障害や、SQL コーディングの品質の悪さからの問題かもしれません。

単にオペレーションの数を計測してもユーザは何もわかりません。より望ましいアプローチはオペレーション数を計測するのではなく、そのオペレーションに係った時間です。パフォーマンスはユーザがクエリをサブミット、またはレスポンスを受け取った時間を測定することが基本になります。言い換えればエンドユーザのパフォーマンスは時間が基本になります。

よい例えとして非常に長いハシゴがあります。それらの横木が一定の間隔でなければ、1 つの横木からその次への時間はその距離に比例します。トータルな時間は、単に横木の数を数えることは意味がなく、足止めさせる各ステップでの計測が重要で、その合計になります。

タイム・ベースの計測に対する引数はデータベースに起こる負荷に関連します。(時間周期を計測することは単なる集計より複雑です。) しかしデータベース・パフォーマンス阻害する過度のオーバーヘッドがないということで、タイム・ベースのアプローチの方がより良い手法と考えます。

どこまで深く計測するか?

それはできるだけ最階層レベルで計測することです。ベータベース・モニタリングの最大の目的はそのボトルネックの根本を解明することです。バケツのデータを統合し、バケツの底にドリルダウンできないなら、バケツの底に眠る問題の根本原因を解明することはできません。

このアプローチの長所は根本原因の分析に限られません。問題の根本原因を結果として本当に解明することは、問題があれば開発者がデータベースを責め、DB 管理者が開発者を責める責任追及を解消します。ボトルネックがどこなのかを解明することでそれによりそれぞれが責め合う問題はなくなります。そしてメンバーは問題解決するための生産的な業務に関わることができます。

どんなコンテキストをその測定に適用するか？

驚くことに多くのツールは収集したデータにコンテキストを適応させていません。それらは単にオペレーションの数を照合するのみです。しかし最終的に最適化が必要なパフォーマンスはユーザが経験しているパフォーマンスです。それゆえユーザがデータベースに対する要求で計測することに意味があります。各ユーザ・リクエストでの最初から達成するまでのトータルのターンアラウンドになります。

要約

最終的にパフォーマンスはパフォーマンスに関するすべてのエンドユーザの経験からの総計になります。それゆえすべてのユーザのデータベース・リクエストを計測し、これらのリクエストをできるだけ小さなコンポーネントに分割化し、それらの小さな各コンポーネント・タスクの時間を計測することが必要です。これでユーザのリクエストのどの部分が最も時間が係り、最も大きな遅延が起きている場所が分かるのでアクティビティに優先度を決め、最も詳細なレベルで根本原因分析を行うことができます。

特定の問題に対してどれくらいのインパクトがあるかを知ることができるかはさらに考慮すべきです。例えば特定のアプリケーションに 2 つの大きなボトルネックのソースがあるとします。どの程度これらのパフォーマンス・ボトルネックにインパクトを与えることができますか？タイム・ベースまたはカウント・ベースのアプローチのどちらを使用しているかに関わらず特定の機能を何パーセント改善できるかを知ることは価値のあることです。例えば貧弱な SQL コーディングでボトルネックの一つのパフォーマンスを 25%の想定で改善できるとし、もう一つが 8%のみであれば後者より前者に焦点を合せることが合理的です。もしユーザのソリューションがこの種類の推定機能を提供できれば非常に有益なものとなります。

しかし、これがすべてではありません。以前に説明したように最初にどのソリューションもモニターするデータベースには最小限のインパクトでなければなりません。二番目にパフォーマンス問題の 70-80%はデータベースから想定されますが、それがボトルネックの原因のみとは限らず、アプリケーション・サーバやネットワーク経由のどちらかに起因する潜在的な問題かもしれません。それゆえデータベース・パフォーマンス・モニター・ツールがインフラストラクチャ、および/またはそれを行うサード・パーティ製品内でのこれらのレベルで比較機能を提供することは有益なことです。三つ目としてインテグレーションに関して、サードパーティ・サプライヤ、またはデータベース・ベンダーから提供される汎用的なデータベース管理ツールにインテグレートできることはまた有益なことです。最後に多重、異種データベースを持つ多くの環境で、1 つ製品とインストール・プロセスがすべてのデータベースのモニターに使用されることは非常に有益なことです。

ここではデータベース・パフォーマンス・モニターに関する違ったアプローチについて紹介を行い、タイム・ベースで、精度の高い、ユーザ・オリエンテッドなアプローチが望ましいソリューションであるとして説明を行いました。ここでそのソリューションとして Ignite を詳細に考えていきます。

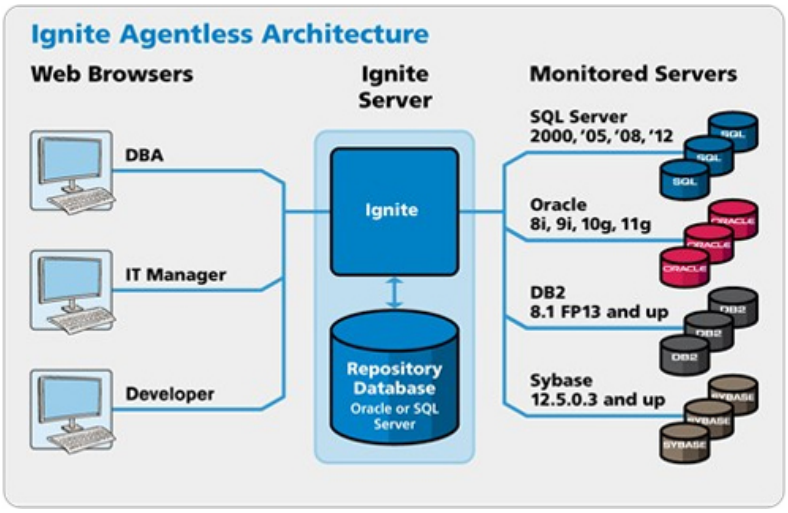

図 1: Ignite アーキテクチャ

アーキテクチャ

Ignite インプリメンテーションのアーキテクチャを図１に紹介し、そして３つの大きな特徴がります。Ignite は表示されているように各データベースをサポートし、同様に J2EE アプリケーション・サーバで、すべてがシングル・インストールです。

さらに Ignite Repository サーバです。ここですべての待ち時間（ウェイト・タイム）の計算が実行され、整理されるので、この目的のために実システムにはオーバーヘッドはかかりません。実際には Ignite はメモリーに保存される情報をリードするために一定間隔で実システムをポールします。それゆえモニターしようとする使用中のシステムにはどんなソフトウェアもインストールされません。技術的に Ignite は「エージェントレス」システムです。このアプローチでは使用システムに対しては 1%以下のオーバーヘッドです。

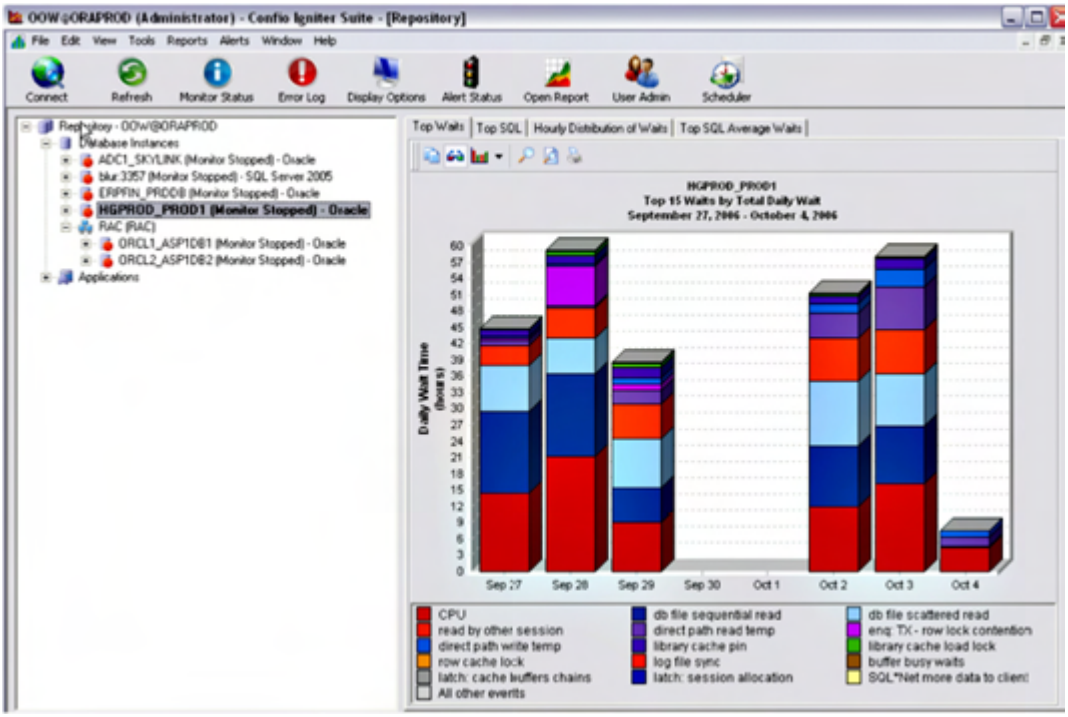

図 2: Ignite のハイレベル・レポート

Ignite の利用について

ユーザ・リクエストがデータベースに送られ、データベースはそれを個別のステップ群に分割し、それらのステップを実行します。データベースによってはこれらのステップ数は数百ステップ数になることがあります。Ignite は各ユーザ・リクエスト用に各ステップが取る時間の長さを計測します。ステップが完了するのに係る時間を待ち時間として参照されます。

しかしこれは Ignite が隠れて行っているので、ユーザはほとんど意識させません。実際に Ignite は実質的にはブラウザ・ベースのビジュアル・ツールです。図 2 ではハイレベルでのレポートで、ユーザはここから調査をスタートします。ほとんどのオプションは一目瞭然のもので、Scheduler はスケジュール・ベースでモニターを On/Off ができます。レポートは長期のトレンド分析を提供します。

ユーザは日付で選択した「トップ・ウェイト（ウェイトが高い）」を確認することができます。また他のタブの日付でのデータを同じように確認することができます。スクリーン・ショットでは時間遅延が起きている大きな理由が CPU、データベースのシーケンシャルなリード、データベースの分散リード、他のセッションでのリードの順序であることを示しています。しかし特に 9 月 28 日に行ロック競合が発生しています。Ignite はマウスをブロックに重ねることで、データのどのブロックの詳細を提供するツールチップをサポートします。

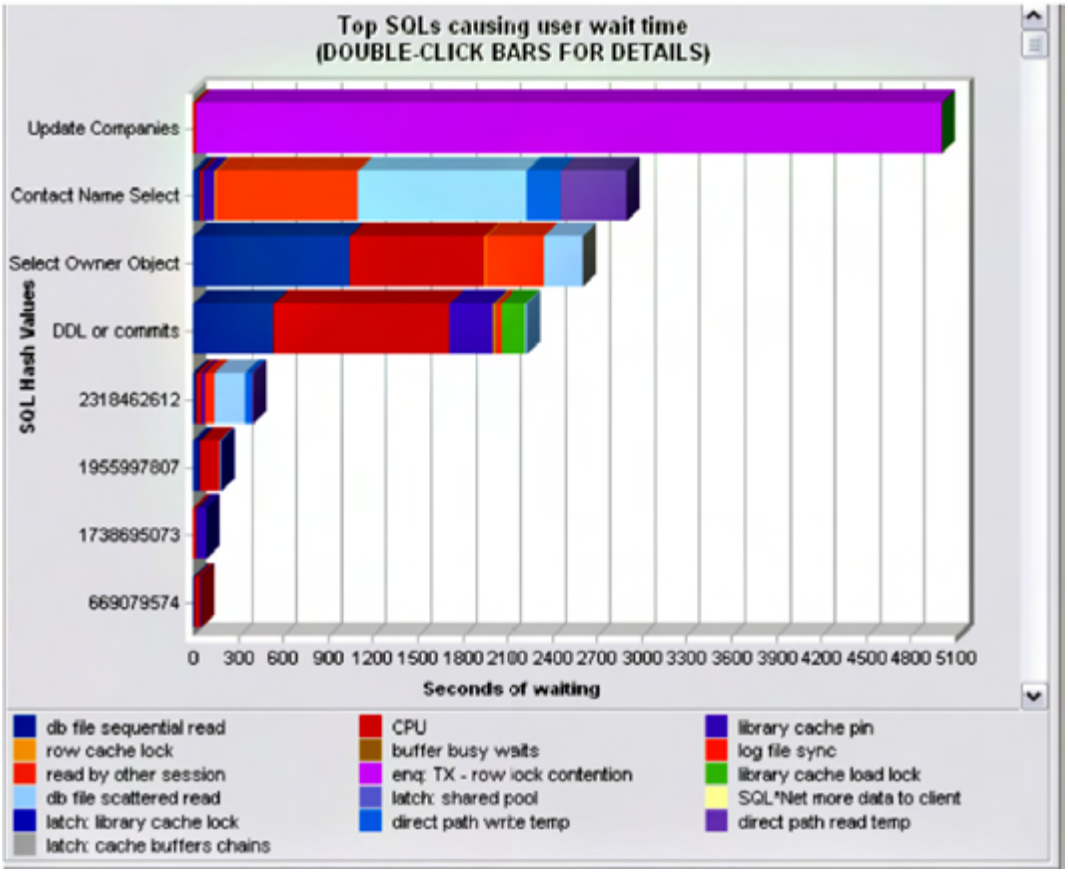

図 3: データへのドリルダウン

次に 1 時間毎、またはユーザが指定した時間間隔での情報を確認するためにドリルダウンします。そしてデータベースに転送された関連する SQL ステートメントを確認することで問題を特定することができます。事実、図 3 では高レベルでの詳細（トップ SQL ウェイト・タイム）を示しています。この画面のルック・アンド・フィールドはデータにドリルダウンした場合と同じです。

ここで確認できるのは1つの特定のSQLステートメント(update companies)で行ロック競合の大きな問題があることを確認できます。

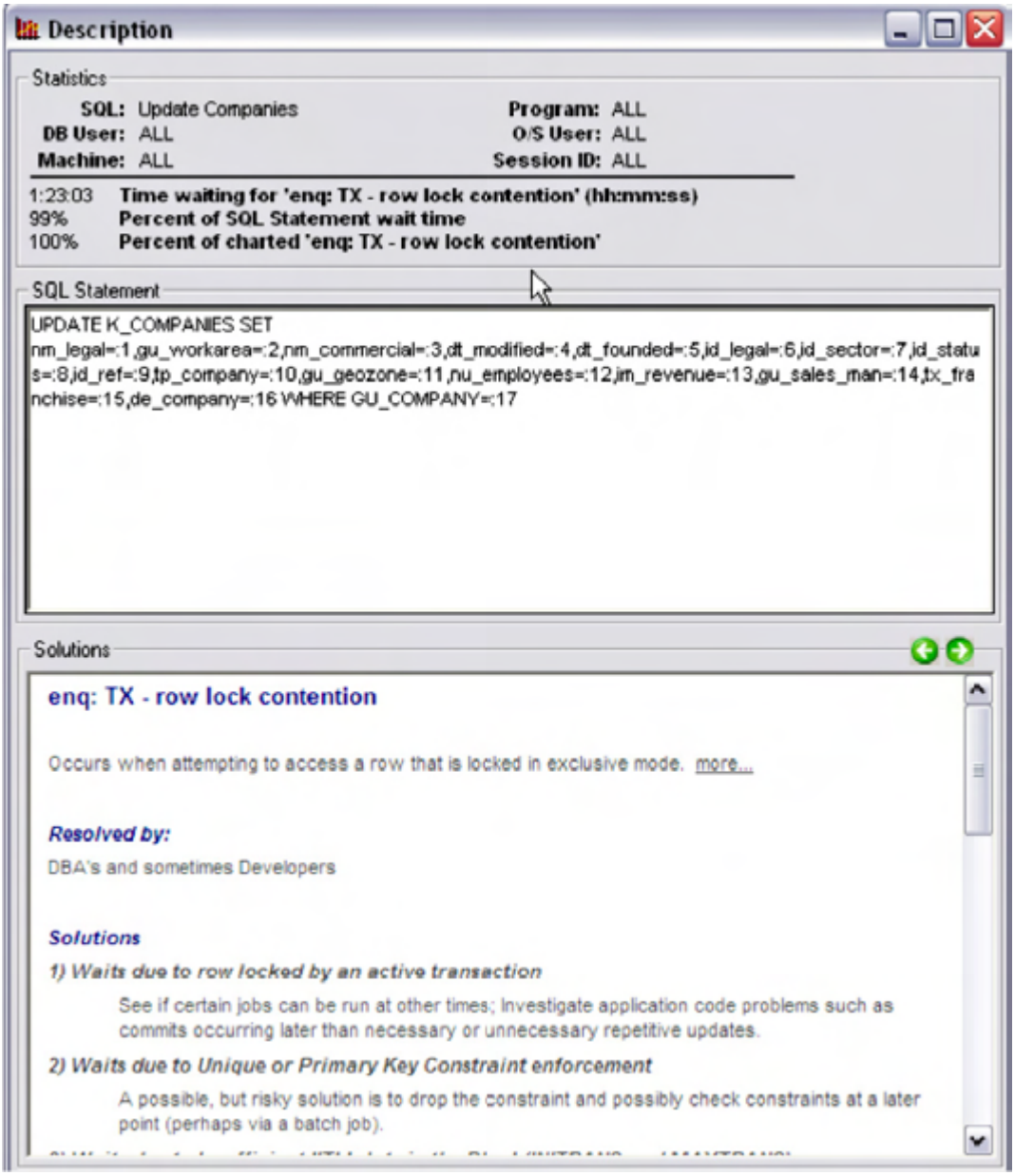

図 4: SQL コマンドへのドリルダウン

この場合に次に行うことは関連する実際の SQL へドリルダウンすることです。図 4 で確認できるようにIgniteは実際のSQL とともに推奨するソリューションを提供します。「99% Percent of SQL Statement wait time （SQL ステートメント・ウェイト・タイム）と記述されているラインにも注意してください。それはユーザが可能な解決策の潜在的な有効性を判断するための助けになるかもしれません。もしユーザが他とは対照的なこの問題の解決方法を考えていて、他の問題が 17%の問題パーセントとすると、救済の可能性は非常に低くなります。この数値は最初のビジュアルな検証後における優先順位に役立ちます。

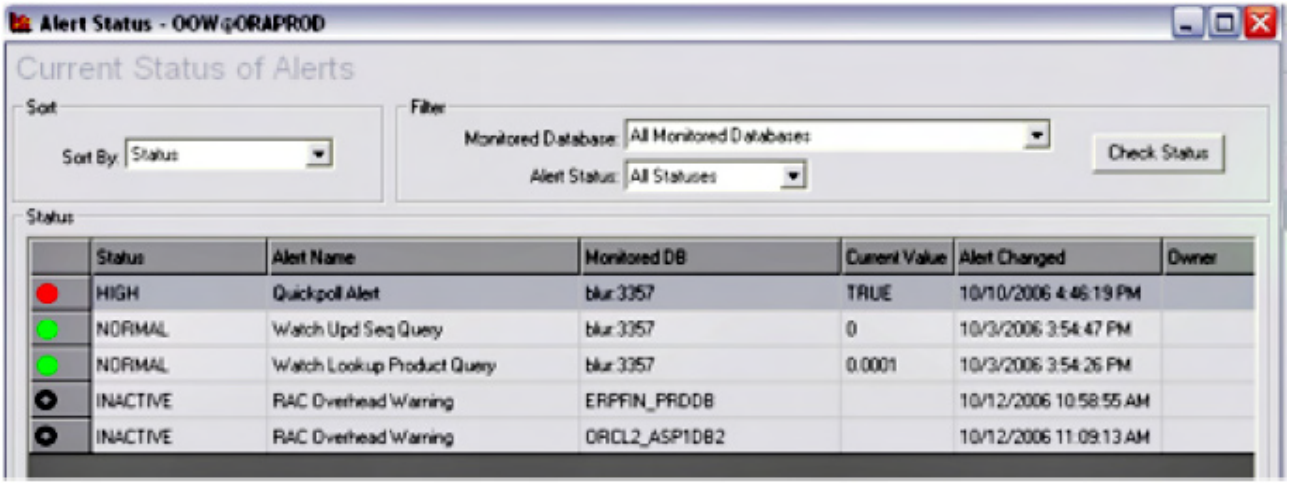

図 5: Ignite のアラート機能

図 5 に Ignite のアラート機能を示しています。これは問題が起きた時、さらに正確には問題が起きようとしたときに関係者に SMS メッセージか、E メールを送ることができます。

最後にこれまで SQL ステートメントとウェイト・タイムに関するデータの検証について紹介してきました。それはユーザによるパフォーマンスを検証することができることを理解することも重要なことで、特定のユーザが低パフォーマンスについて苦情があった時に役立ちます。さらにマシン・セッションによるデータにドリルダウンすることができます。

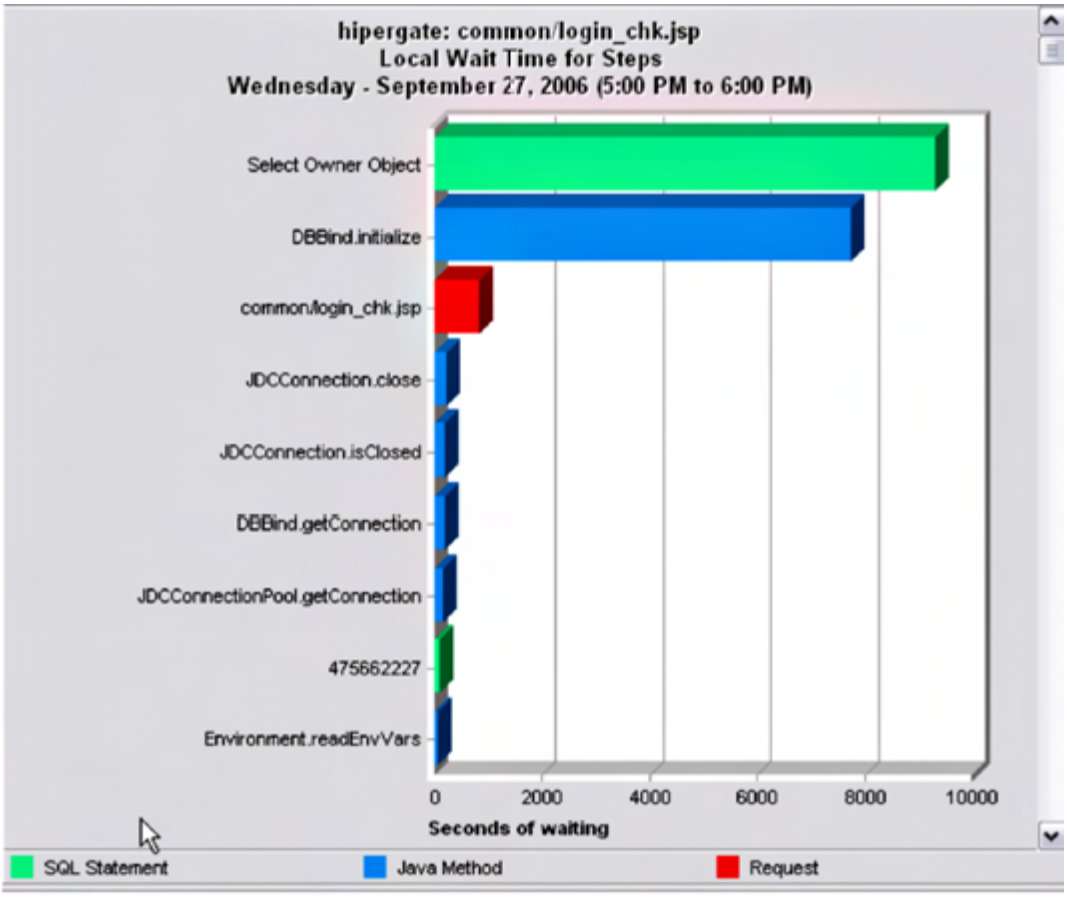

図 6 : SQL と Java の識別

また Ignite をもしデータベース・モニターと同様に使用すれば、アプリケーションでのパフォーマンスを確認することができます。しかしこのソリューションがアプリケーション・パフォーマンス・モニターというよりも主にアプリケーション・レベルで何が起こったかの可視化を必要とするデータベース関係者用であることに留意しておくべきです。 図 6 は SQL ステートメント、Java メソッド、Java リクエスト間を識別する例です。

Ignite はそれがデータベース・パフォーマンスをモニターする手法と似た方法で、アプリケーション・サーバ上でメソッドを監視します。

結論

コンピュータ・システムで最も重要な要素はそれを使用する人です。パフォーマンスはユーザとそのユーザに合わせることが必要なシステム（データベースに限定しない）をモニターする方法のみに関連します。これを現実的に行う唯一の方法は、それが行われる必要がある様々なことをするのに係る時間をモニターすることです。さらにこれをできるだけ詳細に行うことができる必要があります。それで問題の根本的な原因を突き止めることができます。Ignite はその機能を的確に提供し、データベースのパフォーマンス問題を抱える企業はその機能を確認すべきです。