記事
Minoru Horita · 2020年12月14日 14m read

知っておくと便利なクエリパフォーマンスのコツ - Tune Table

優れた執筆者は、題名で読者を引き込み、答えを記事のどこかに隠すべきだと考えられています。 だとすると、私は出来の悪い執筆者ということになってしまいます。私の自尊心は見知らぬインターネットユーザーの皆さんのご意見にかかっていますので、お手柔らかにお願いします。

私の同僚 Brendan が Developer Community に記載されている情報をレビューしていたとき、Tune Table について書き忘れがあったことに気が付いたのです！ これはクエリパフォーマンスにおいて
2番目に重要なツール (1 番はインデックス、インデックスがないと処理が遅い)
なので、時間をかけて理解しておく価値があります。
今回使用する例の多くは完全なものではありませんが、詳細は簡単に入力できます。 すべての例において、特に明記が無い限り、フィールドはすべて個別にインデックスが作成されているものと想定してください。

それでは、以下のクエリがあるとしましょう。

```
SELECT *
FROM People
WHERE HomeState  = 'MA'
 AND PersonId = '123-45-6789'
```

どのインデックスを使うべきでしょうか？ 冗談なしで、一旦読むのをやめて、答えを聞かせてください。
大声で答えてください。 同僚の皆さんに変な人だなんて思われませんよ、大丈夫です。

正解は、もちろん、PersonId のインデックスです。 なぜこれが正解なのか？ 各 State (州)
には何百万もの人々が生活している可能性がある中で、PersonId はほぼ一意の数値 (的な)
識別子であることが分かります。 ここで理解する必要があるのは、この答えが分かるのは、列の識別子を見るとスキーマに関する情報がある程度は分かるためである、という点です。
コンピューターにとって、これらの名前は何の意味もなしません。
この点を分かりやすく説明するために、こちらのクエリを見てみましょう。

```
SELECT *
FROM TheTable
WHERE Field1 = 32
 AND Field2 = 0
```

使うのはインデックスは Field1？それとも Field2？
上のクエリは、クエリオプティマイザによって最適化されるようなクエリです。
この場合はどのようにインデックスを選べばいいのでしょう？
適切な決断をするにはデータについて知る必要があります。 これを解決するのが Tune Table です。

Tune Table は、テーブル内のデータサンプルを調べ、テーブルに関する統計を保管することで、クエリオプティマイザが適切な決断をできるようにします。 今まで Tune Table
を実行したことが無い場合は、クエリがランダムでも効果的にインデックスを選択しているということです

(厳密には違いますが、ほぼそういうことです)。
実際、クエリが遅いことについてサポートに電話されることがあれば、まず最初に「Tune Table は実行されましたか？」と聞かれると思います。 ‘ はい ’ と言えるようにしておいてください。

Tune Table の実行方法

Tune Table を実行するには、ターミナルに移動して、次を実行します。

d $SYSTEM.SQL.TuneTable(<Table>)

これに渡せるフラグは沢山ありますが、私のおすすめはこちらです。

d $SYSTEM.SQL.TuneTable(<Table> ,1,1,.msg,1)

これらのフラグにより、クラス定義が更新され、新しい値が表示されるほか、最後のフラグによりクラスが最新の状態に維持されます。 クエリが Tune Table の新しい情報を利用できるようにするには、すべての埋め込みクエリをコンパイルし、キャッシュされているクエリを消去する必要があります。
フラグに関する詳細は、こちらをお読みください。

https://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?...

Tune Table の機能

Tune Table の実行ステップを紹介します。 まず最初に、SELECT COUNT(*) を実行してテーブル内の行数を計算します。 その行数を基に、サンプルとして使用する行数を決定します (漸近的には行数の平方根に似ています)。 そして、テーブルで定義されている各 ID に対し、加重サンプルを適切に判断した上で、その行をサンプルに含めるかどうかを決定します。
このサンプルを使って統計を算出し、それを使ってクエリを最適化します。

ここには重要なポイントが 2 つあります。 1 つ目はカウントを得る必要があること。 2 つ目は、テーブル内の各 ID を取得できる必要があることです。 この結果、エクステントインデックスがあることにより、Tune Table の実行が大幅に改善されるのです。 テーブルがビットマップに対応している (つまり正の整数 IDKEY がある) 場合は、ビットマップのインデックスがあれば、自動的にビットマップのエクステントを得ることができます。

Tune Table はどのような指標を測定するのか？ 年代順に古い方から説明していきます。

2 つの OG (旧世代の指標)

次に紹介する 2 つの指標は Tune Table の最も古い指標です。 どれだけ古いと思いますか？
私が高校に入学する前 (1999年) に追加されたものです。 Y2K 以前の Caché を実行している方がおられましたら、私までメールをください。是非、お話しさせていただきたいです。

Extent Size は、テーブル内の行数を測定します。

これは一番理解しやすい指標で、JOIN を実行する順序を決定するのに便利です。

では、下のクエリがあるとしましょう。

```
SELECT *
FROM Table1 T1
 JOIN Table2 T2
 ON T1.Field1 = T2.Field1
```

T1 を読み取ってから、T2 のフィールドで JOIN を実行するべきか、またはその逆にするべきか？ Extent Size は、行の数が少ない方のテーブルを示すので、どちらのテーブルから始めるべきなのかが分かります。
極端なケースとして、T1 には 100 億行あり、T2 には 100 行しかないと仮定します (T1 はこれまで病院に行った回数の合計を示すテーブルで、T2 は現在病院にいる患者の数を示すテーブルだと考えてください)。
この場合は、小さい方のテーブルから始め、大きな方で JOIN を実行すると良いでしょう。それは、JOIN を実行すると、読み取る必要のある行の数が制限されるためです。

Selectivity は、指定されたフィールドの値の一意性を測定します。

この指標を理解するにはもう少し説明が必要です。 考え方は 2 種類あります。

1. 1/x。x はフィールドに設定され得る値の数。
2. クエリ SELECT * FROM T WHERE field=? の実行結果として返されるテーブルの平均パーセンテージ

いずれの説明もこれだけでは分かりにくいので、いくつか例を挙げてはっきりさせたいと思います。

先ほどのクエリに、HomeState というフィールドがありました。 (アメリカの場合だと) そのフィールドの Selectivity は 2.0% になります。 なぜかと言いますと、 アメリカには 50 の州があるため、1/50 = .02 = 2% という計算になります。 これは簡単ですね。 しかし、分布はどうなのか？
明らかに、複数の州で人口が均等に分布しているはずがありません！ それは事実なのですが、Selectivity はそれを考慮しません。 均等な分布は、ほとんどのクエリにおいて適切な予測とされる上に、計算も簡単になるため、クエリはよりスピーディにコンパイルします。

もう 1 つの例として、PersonId について考えます。 人は亡くなったり、新しく生まれたりするので、 PersonId はたまに再利用する必要があるかもしれません。なので、その数字は完全に一意という訳でははありません。しかし、一意であることに変わりはありません。 そのため、.00001% の Selectivity が見られる場合があるかもしれません。 それは非常に適切な Selectivity と言えます！ クエリ SELECT * FROM People WHERE PersonId = ? は、 実行される度に 1 つまたは 2 つの行を返します。 一般的に、Selectivity は低いほど良いとされています。 しかし、これには例外があります…

一意のフィールドがあるとしたらどうでしょう。ID はいい例ですね。 一意のフィールドの Selectivity は 1 です。ちなみに、これまで述べた Selectivity はすべてパーセンテージです。 1 はパーセンテージではなく、行数です。したがい、クエリ SELECT * FROM People WHERE ID = ? は、 いつも 1 行を返します。

1 つのフィールドについて沢山の情報をカバーしましたが、おそらく一番重要な情報であると言えるでしょう。

原則として、パーセンテージは低い方が良く、1 が最適な値です。

「昔から」存在していた指標

次に紹介する統計は、いつも予測されてはいたものの、バージョン 2013.1
までは明示的に測定されていませんでした。

Block Count – 各マップ (インデックスとマスターマップ) がディスク上で占めるブロックの数を測定します。

Block Count は長年に渡って予測してきましたが、2013 年になって初めて明示的に測定する決断をしました。
Extent Size があるので不必要に思えるかもしれませんが、他にも実用性があります。 こちらのクエリ

SELECT COUNT(*) FROM MyTable
を説明すると理解に役立つでしょう。
各フィールドにインデックスが付いているとしたら、どれを読み取ればいいでしょうか？
読み取りたいのは、ディスク上で占めるスペースが一番少ないフィールドです。
この指標はその情報を明示的に示してくれます。
より複雑な状況でインデックスのかたまりを読み取る際のコストを予測するときにも表示されます。
通常、これはマップの幅を測定するのに優れている一方で、Extent Size
はその長さを測定するのに優れています。

Block Count は、親 / 子関係を使用する場合にも便利です。 この場合は、(デフォルトで)
両方のテーブルが同じグローバル変数に格納されます。しかし、Block Count がなければ、クエリオプティマイザは、潜在的に小さな親テーブルで読み取られたマップには膨大な数の子が含まれているために、実はとても大きなマップである可能性があるということを全く知ることができません。 実に、これは私たちが新規開発において親 / 子関係の使用を一切おすすめしていない理由の 1 つです
(どうしてもという場合は、サポートにお電話の上、ご相談ください！)。

Brendan からのメモ / トリビア: Extent Size、Selectivity、Block Count は FDBMS が使われていた時代、すなわち 1992 年には既に存在していたのですが、当時は手動で設定する必要がありました。

すべてを一変させた指標

バージョン 2014.1 では、フィールドの 1
つの値が過剰に大きな割合を占めていないかどうかを判断するための指標を追加しました。
これにより、Selectivity を計算する方法が大幅に変わりました。

Outlier Selectivity (外れ値の選択性) - テーブル内で過剰な割合を占めるフィールドの選択性。

外れ値を伴う状況を理解するために、病院について考えてみましょう。
病院は一般的に地域のコミュニティにサービスを提供するものです。 HomeState
フィールドは、そのほとんどの値が病院のある州で同じになります。 では、MA (マサチューセッツ州)
の病院について考えます (私が住んでいる州なので)。 私の地元の病院は、患者の 90% が MA

の人たちです。それ以外は他の州から来た人たちや州を訪れている人たちです。 HomeState= ’ MA ’
を検索することは選択的でない一方で、それ以外の HomeState
を検索することは選択的である、ということを示す必要があります。 これを実現するのが Outlier Selectivity
です。

例えば、上述した HomeState フィールドだと、Tune Table はその Selectivity を0.04% として算出し、Outlier Selectivity を 90%、Outlier 値 (外れ値) を ‘ MA ’ として示します。 通常、Selectivity は低下します。 Selectivity の計算方法が変わり、人々をビックリさせたクエリプランもいくつか考案されたこともあり、これは大きな変化であったと言えます。 バージョン 2014.1 全体を更新する予定の方は、この点に注意してください。

最後の指標

この最後の指標は、一時ファイルのサイズをより正確に予測することを目的に、バージョン 2015.2
で追加したものです。

Average Field Size – フィールドの平均サイズ。

これも簡単に理解できる指標です。
サンプルを取る間に、テーブル内の各フィールドの平均サイズを計算してくれます。 これは一時ファイルのサイズを判断するのに便利です。また、これにより、オプティマイザは一時ファイルをメモリ内で構築できるのか (ローカル配列)、またはディスクベースの構造 (プライベートグローバルを処理する)
が必要なのかを判断できます。

実用的な検討事項

次に考えるのが「どれくらいの頻度で実行する必要があるのか？」という疑問です。
これといった答えはないのですが、 少なくとも (絶対に！！！)
一度は実行してください。でなければ、自分で値を用意しなくてはいけなくなります。
その後は、もう二度と実行しなくてもよいかもしれません。
現在のパフォーマンスに満足なら、実行する理由はありません、というのが一般的な答えでしょう。 実に、データベースの成熟度が高く、テーブル内のデータも安定したペースで成長しているのなら、今後もおそらく適切な値が得られるでしょう。

もし、テーブルの変化と共に SQL のパフォーマンスが劣化することがあれば、Tune Table
を再度実行することをおすすめします。
また、新しいデータが多く、テーブルが大幅に変化している場合は、Tune Table
を実行したほうが無難かもしれません。
しかし、これは「すべてのクエリが遅い」という状況にあれば予測できることです。
この状況に陥っている方は、WRC までお問い合わせください。システム設定を確認させていただきます。

Tune Table を実行する場合は、調整するテーブルに関連しているすべてのテーブルに対して実行してください。
そうしないと、テーブル間で Extent Sizes と Block Counts を比較てきなくなります
(これによりクエリのパフォーマンスが低下します)。

2016.2 よりも新しいバージョンをご利用の方は、Tune Table を実行する前にクエリプランをフリーズできる Frozen Query Plans をご利用いただけます。 このプラクティスは、Tune Table を実行したときに、新しいプランが役に立つと判断した場合にそれだけを利用できるという機能があるため、強くお奨めしています (%NOFPLAN を使ってクエリをテストすれば、新しいプランが役に立つのかどうかを確認できます)。

Frozen Query Plans についてはもう 1 つ大切なことがあり、このテクノロジーを使えば、Tune Table を実行するときに、システム上で既に実行したクエリをもう一度実行してしまうことを避けられます。
これは、現在のパフォーマンスに影響を与えることを心配せずに、Tune Table を実行するための手段です！
もう言い訳はできません！ Frozen Query Plans の詳細に興味がある方は、私が以前行ったウェビナーをご覧ください:

https://learning.intersystems.com/course/view.php?id=969

最後になりましたが、クエリパフォーマンスについては、Tune Table の統計を把握しておくことが大切です。
Tune Table は、クエリオプティマイザが適切な判断をするのに必要な情報を与えてくれます。
既にテーブルにインデックスを追加されている方は、次のステップとして Tune Table を実行すれば、クエリを超高速化できること間違いなしです。

今回の記事で使用しました SQL フォーマッタ (http://dpriver.com/pp/sqlformat.htm) を私に紹介してくれた Aaron Bentley に感謝の意を送りたいと思います。

また、私の記事を実際に理解できる内容に編集してくれる Brendan Bannon にも感謝いたします。

#SQL #パフォーマンス #Caché #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/%E7%9F%A5%E3%81%A3%E3%81%A6%E3%81%8A%E3%81%8F%E3%81%A8%E4%BE%BF%E5%88%A9%E3%81%AA%E3%82%AF%E3%82%A8%E3%83%AA%E3%83%91%E3%83%95%E3%82%A9%E3%83%BC%E3%83%9E%E3%83%B3%E3%82%B9%E3%81%AE%E3%82%B3%E3%83%84-tune-table