記事
Toshihiko Minamoto · 2021年5月18日 12m read

インデックス処理

第2部: インデックス処理

クラスにどのようなインデックスが必要であるのか、それをどのように定義するのかについて理解できたので、次に、どのように処理するのかについて確認しましょう。

クエリプラン

（**注意**: クラスに変更を適用する場合と同様に、ライブシステムにインデックスを追加する場合にもリスクが伴います。インデックスが入力されているときに、ユーザーがデータにアクセスしたり更新したりすると、クエリ結果が空になったり誤った結果が生じることがあります。また、構築中のインデックスが破損する場合もあります。ライブシステムでインデックスを定義したり使用したりするには追加の手順があり、それについてはこのセクションで触れていますが、詳細はドキュメントに記載されています。）

新しいインデックスの準備ができたら、SQLオプティマイザが、クエリを実行する上で最も効率的に読み取れるインデックスであると判断するかどうかを確認できます。
プランを確認するために実際にクエリを実行する必要はありません。
クエリがあれば、プランをプログラムで確認することができます。

```
Set query = 1

Set query(1) = “ SELECT SSN,Name FROM Sample.Person WHERE OfficeState = 'MA' ”

        D $system.SQL.ShowPlan(.query)
```

また、システムエクスプローラー ->
SQLより、システム管理ポータルのインターフェースに従って確認することもできます。

ここから、どのインデックスが、テーブルデータ（または「マスターマップ」）をロードする前に使用されているのかを確認できます。 プランの新しいクエリは期待どおりに動作しているのか、
プランのロジックが合理的であるのかを検討しましょう。

SQLオプティマイザが作成したインデックスを使用していることを確認できたら、これらのインデックスが適切に機能しているかどうかを確認できます。

インデックスの構築

（現時点では計画しているだけの段階であり、データが存在しない場合は、ここに説明されている手順は必要ありません。）

インデックスを定義しても、テーブルのデータが自動的に入力または「構築」されるわけではありません。 まだ

構築されていない新しいインデックスをクエリプランに使用すると、クエリ結果が誤りとなったり空になったりするリスクがありますが、 マップの選択可能性を0に設定すると、インデックスの使用準備が整う前にそのインデックスを「無効」にすることができます。0に設定するというのは、基本的に、SQLオプティマイザに、クエリの実行にはそのインデックスを使用できないことを指示していることになります。

write $SYSTEM.SQL.SetMapSelectability("Sample.Person","QuickSearchIDX",0) ; Set selectability of index QuickSearchIDX false

上記の呼び出しは、新しいインデックスを追加する前でも使用できることに注意してください。 SQLオプティマイザは、この新しいインデックスの名前と、それが非アクティブであることを認識するため、クエリには使用しません。

インデックスの入力は、 %BuildIndices
メソッド（##class(<class>).%BuildIndices($lb("MyIDX"))
）を使用するか、システム管理ポータルのSQLページ（アクションドロップダウン）で行うことができます。

インデックスの構築にかかる時間は、テーブル内の行数とインデックスの種類によって異なります。通常、bitsliceインデックスの作成には時間が掛かります。

このプロセスが完了したら、もう一度SetMapSelectivityメソッドを使用して、新たに入力されたインデックスを有効にできます（この場合は 1 に設定）。

インデックスを作成するというのは、基本的に、KILLコマンドとSETコマンドを発行して、インデックスを入力することです。 そのため、このプロセスを実施する間は、ディスク領域がいっぱいにならないように、ジャーナリングを無効にすることを検討してください。

新しいインデックスと既存のインデックスを作成する詳細な手順について、特にライブシステムでの手順については、以下にリンクされているドキュメントをご覧ください。

https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPTindices#GSQLOPTindicesbuildreadwrite

インデックスの管理

ようやく、実際にインデックスを使用できるようになりました。 クエリがどれほど効率的に実行するのか、ある特定のインデックスがどれくらいの頻度で使用されるのか、そしてインデックスの一貫性が崩れた場合にどのように対処するのかといったことを考慮する必要があります。

パフォーマンス

まず、このインデックスがクエリのパフォーマンスにどのような影響を与えたのかを検討できます。 SQLオプティマイザがクエリに新しいインデックスを使用していることがわかっている場合は、クエリを実行して、そのパフォーマンスの統計（グローバル参照数、読み取られた行数、クエリの準備時間と実行時間、およびディスクで費やされた時間）を収集することができます。

前の例に戻りましょう。

SELECT SSN,Name,DOB FROM Sample.Person WHERE Name %STARTSWITH 'Smith,J'

このクエリのパフォーマンスを支援するために、次のインデックスがあります。

```
Index QuickSearchIDX On Name [ Data = (SSN, DOB, Name) ];
```

NameIDXインデックスは、すでにNameプロパティにあります。

直感的には、クエリはQuickSearchIDXを使って実行していることがわかります。NameIDXは、Nameプロパティに基づいていますが、SSNまたはDOBのデータ値が含まれていないため、2番目の選択肢である可能性があります。

QuickSearchIDXを使ったこのクエリのパフォーマンスは、実行するだけで簡単に確認できます。

（余談ですが、パフォーマンスの違いを分かりやすくするために、クエリを実行する前に、クエリキャッシュを消去しています。 SQLクエリが実行されると、次回実行時のパフォーマンスを改善できるように、実行に使用されたプランが保存されます。この記事の目的から外れてしまうため詳しい説明はしませんが、SQLパフォーマンスに関するその他の考慮事項として、この記事の最後に資料を記載しておきます。）

Query Plan

Relative cost = 1856

- Read index map Sample.Person.QuickSearchIDX, looping on %SQLUPPER(Name) (with a %STARTSWITH range condition) and ID.
- For each row:

    Output the row.

Row count: **31** Performance: **0.003** seconds **154** global references **3264** lines executed **1** disk read latency (ms)

(率直に)クエリの実行に必要なのは、QuickSearchIDXのみです。

QuickSearchIDXではなくNameIDXを使ったパフォーマンスを比較して見ましょう。これは、クエリキーワードである%IGNOREINDEXを追加して行います（SQLオプティマイザが特定のインデックスを選択できないようにします）。

クエリを次のように記述します。

```
SELECT SSN,Name,DOB FROM %IGNOREINDEX QuickSearchIDX Sample.Person WHERE Name %STARTSWITH 'Smith,J'
```

クエリプランはNameIDXを使用するようになったので、インデックスを介して見つかった関連する行IDを使って、Sample.Personのデータグローバル（または「マスターマップ」）から読み取る必要があることがわかります。

Row count: **31** Performance: **0.020** seconds **137** global references **3792** lines executed **17** disk read latency (ms)

実行に必要な時間、実行される行数、およびディスクのレイテンシが増加しているのがわかります。

次に、インデックスを全く使用せずに、このクエリを実行してみましょう。 クエリを次のように調整します。

```
SELECT SSN,Name,DOB FROM %IGNOREINDEX * Sample.Person WHERE Name %STARTSWITH 'Smith,J'
```

インデックスを使用しない場合、条件を満たすかどうか、データの行を確認する必要があります。

Row count: **31** Performance: **0.765** seconds **149999** global references **1202681** lines executed **517** disk read latency (ms)

特殊なインデックスであるQuickSearchIDXが、インデックスを使用しない場合よりも100倍以上速く、また一般的なNameIDXを使用した場合よりもほぼ10倍速く、クエリを実行できています。さらに、NameIDXを使用した場合は、インデックスを全く使用しない場合よりも30倍以上速く実行しています。

この特定の例では、QuickSearchIDXとNameIDXのパフォーマンスの差はわずかですが、数百万行に対して実行するクエリを1日に数百回実行するのであれば、貴重な時間を節約できることがわかるでしょう。

SQLUtilitiesを使った既存のインデックスの分析

%SYS.PTools.SQLUtilitiesには、IndexUsage、JoinIndices、TablesScans、TempIndicesなどのプロシージャが含まれています。 これらは、特定のネームスペースにある既存のクエリを分析し、特定のインデックスが使用される頻度、テーブルの各行で反復を選択しているクエリ、そしてインデックスをシミュレートする一時ファイルを生成しているクエリに関する情報を報告します。

これらのプロシージャを使用することで、インデックスが対処できる可能性があるギャップと、使用されていないまたは非効率であるために削除することを検討した方がよいインデックスを特定することができます。

これらのプロシージャと使用例についての詳細は、以下のクラスに関するドキュメントをご覧ください。

https://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPToptqueryindexanalysis

インデックスに関する問題？

インデックスを検証することで、インデックスが存在し、クラスの各行で正しく定義されているのかを確認できます。 インデックスが破損した状態になるクラスはありませんが、クエリが空の結果セットや誤った結果セットを返しているようであれば、クラスの既存のインデックスが現在有効であるかを確認することをお勧めします。

インデックスは、プログラムで次のように検証できます。

```
Set status = ##class(<class>).%ValidateIndices(indices,autoCorrect,lockOption,multiProcess)
```

ここで、インデックスパラメーターはデフォルトで空の文字列です。つまり、すべてのインデックスかインデック

スの名前を含む$listbuildオブジェクトを検証します。

autoCorrectは、デフォルトで0になることに注意してください。
1である場合、検証プロセスで発生するすべてのエラーは修正されます。 機能的にはインデックスを再構築することに変わりませんが、ValidateIndicesのパフォーマンスは比較的に遅くなります。

詳細は、%Library.Storageクラスのドキュメントをご覧ください。

https://docs.intersystems.com/latest/csp/documatic/%25CSP.Documatic.cls?PAGE=CLASS&LIBRARY=%25SYS&CLASSNAME=%25Library.Storage#%ValidateIndices

インデックスの削除

あるインデックスが不要になった場合、またはテーブルに大規模な変更を行って、後で関連するインデックスを構築する際にパフォーマンスへの影響がないようにする場合は、Studioでインデックスの定義をクラスから取り除き、該当するインデックスグローバルノードを削除することができます。 または、DDLを介してDROP INDEXコマンドを実行することも
できます。このコマンドでも、インデックスの定義とデータを消去することができます。 その後、キャッシュ済みクエリをパージすることで、確実に、削除されたインデックスが既存のプランによって使用されなくなります。

さて今度は？

インデックスは、SQLパフォーマンスの一部にしかすぎません。
この流れに並行し、インデックスのパフォーマンスと使用状況を監視するオプションは他にもあります。
SQLパフォーマンスを理解するには、次のことについても学ぶことをお勧めします。

Tune Tables - テーブルに代表的なデータが入力されてから、またはデータの分布が大幅に変化した場合に実行するユーティリティです。 このユーティリティは、クラスの定義に、フィールドの長さや1つフィールドに含まれる一意の値の数など、SQLオプティマイザが効率的に実行するためにクエリプランを選択しやすくするメタデータを提供します。

Kyle
Baxterが、これに関する記事を執筆しています（https://community.intersystems.com/post/one-query-performance-trick-you-need-know-tune-table）。

クエリプラン - 基盤のコードがSQLクエリをどのように実行するか論理的に表現したプランです。 クエリが遅い場合、どのクエリプランが生成されているのか、クエリに適しているのか、このクエリをさらに最適化するために何を行えるのかを検討することができます。

キャッシュドクエリ - 準備済みの動的SQLステートメント -
キャッシュドクエリは、基本的にクエリプランの下にあるコードです。

参考文献

インデックスの定義と構築に関するドキュメント。
読み取り/書き込みのライ
ブシステムで検討する必要のある追加手順が含ま
れています。 https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLOPTindices

ISC SQLコマンド – DDLを介したインデックス処理の構文関連資料について、CREATE INDEXとDROP INDEXを参照してください。
これらのコマンドを実行するための
適切なユーザー権限が含まれています。
https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=RSQLCOMMANDS

ISCクラスでのSQL照合に関する詳細。 デフォルトでは、文字列の値は、インデックスグローバルにSQLUPPER(“ STRING ”) として格納されます。

https://docs.intersystems.com/irislatest/csp/docbook/DocBook.UI.Page.cls?KEY=GSQLcollation

［最終更新日: 2020年5月6日 - インデックス構築パフォーマンスとDROP INDEXコマンドの修正。］

#SQL #インデックス付け #パフォーマンス #ベストプラクティス #Caché #InterSystems IRIS

ソースURL:
https://jp.community.intersystems.com/post/%E3%82%A4%E3%83%B3%E3%83%87%E3%83%83%E3%82%AF%E3%82%B9%E5%87%A6%E7%90%86