記事
Toshihiko Minamoto · 2022年4月1日 7m read

InterSystems IRIS の新しい埋め込み SQL

Benjamin De Boe
がユニバーサルキャッシュクエリ
に関する素晴らしい記事を書いていますが、ユニバーサルキャッシュクエリ（UCQ）とは一体何でしょうか。また、昔ながらの埋め込み SQL を書いている場合になぜそれを気にする必要があるのでしょうか。 Caché と Ensemble において、キャッシュクエリは xDBC と動的 SQL を解決するために生成されるでしょうが、InterSystems IRIS の埋め込み SQL は、キャッシュクエリを使用するように改善されました。そこで、名前に「ユニバーサル」が追加されているのです。 現在では、IRIS で実行されるすべての SQL は、UCQ クラスから実行されるようになっています。

InterSystems はなぜこのように変更したのでしょうか？ 良い質問です！
ここでの大きなメリットは、ライブ環境での柔軟性にあります。
以前は、インデックスを追加したり、TuneTable を実行したりすると、キャッシュクエリの SQL はこの新しい情報をすぐに利用していたにもかかわらず、埋め込み SQL はクラスまたはルーチンが手動でコンパイルされるまで変更されないままになっていました。
アプリケーションがデプロイされたクラスを使用していた場合、または OBJ コードのみを出荷していたのであれば、顧客のシステムで再コンパイルすることなど不可能です。
現在では、システム上のすべての SQL ステートメントが最新の def. と利用可能な最新のチューニングデータを使用できるようになっています。 将来的には、InterSystems IRIS には、本番環境システムを毎晩監視して調整し、テーブルがどのようにクエリされているかに応じて SQL プランをカスタマイズする、オプションのツールが用意される予定です。
このツールセットが拡充するにつれ、ユニバーサルキャッシュクエリの力も大きくなるでしょう。

埋め込み SQL は以前より遅くなっていませんか？ その通りとも、そうでないとも言えます。 別のルーチンでタグを呼び出すと、同じルーチンで呼び出すよりも少しコストがかかるため、速度は遅くなりますが、UCQ コード生成は埋め込みとは異なり、これらの変更を使用することで、別のルーチンを呼び出すコスト以上のメリットがあります。 UCQ コードの処理が遅くなっているケースはあるか、と訊かれれば、「はい」と答えることになりますが、全体的にはパフォーマンスが向上するはずです。
私はずいぶん前から埋め込み SQL を専門としているため、埋め込み SQL は動的 SQL よりも高速だということを常に指摘していますが、確かに高速と言えども、オブジェクトを高速化するために行われたすべての作業を考えると、この 2 つの手法の差は小さいため、動的 SQL を使用していることをからかったりはしません。

エラーはどのようにしてチェックするようになりましたか？ 埋め込み SQL のエラー処理は変更されていません。 エラーに遭遇した場合、SQLCODE が負の数に設定され、%msg がそのエラーの説明に設定されます。 変更されたのは、発生するエラーの種類です。
デフォルトの動作では、SQL は初めてクエリが実行されるまでコンパイルしないようになりました。 つまり、ルーチンのフィールドまたはテーブルのスペルを間違えても、そのルーチンをコンパイルするときにはエラーは報告されません。このエラーは、動的 SQL と同じように、初めて実行したときに報告されます。 SQLCODE はすべての SQL コマンドに設定されていますが、私のように面倒くさがり屋の方は、FETCH の後にのみ SQLCODE をチェックするでしょう。 OPEN でもチェックするのもお勧めです。

```
 &SQL(DECLARE cur CURSOR FOR
 SELECT Name,junk
 into :var1, :var2
 FROM Sample.Person)
 &SQL(OPEN cur)
 write !,"Open Status: ",SQLCODE,?20,$G(%msg)
 for  {
 &SQL(FETCH cur)
 write !,"Fecth Status: ",SQLCODE,?20,$G(%msg)
 QUIT:SQLCODE'=0
```

```
 w !,var1
 }
 &SQL(CLOSE cur)
 write !,"Close Status: ",SQLCODE,?20,$G(%msg)
 QUIT
```

上記のコードでは、SELECT に無効なフィールドが含まれています。 ルーチンのコンパイル時には SQL をコンパイルしないため、このエラーは報告されません。 コードを実行すると、OPEN がコンパイルエラーを報告し、FETCH と CLOSE は、カーソルが開かないエラーを報告します。 %msg は変更されないため、どのタイミングで確認しても、役立つ情報を得られます。

```
USER>d ^Embedded

Open Status: -29 Field 'JUNK' not found in the applicable tables^DECLARE cur CURSOR FOR SELECT Name , junk INTO compiling embedded cached query from Embedded.mac

Fetch Status: -102 Field 'JUNK' not found in the applicable tables^DECLARE cur CURSOR FOR SELECT Name , junk INTO compiling embedded cached query from Embedded.mac

Close Status: -102 Field 'JUNK' not found in the applicable tables^DECLARE cur CURSOR FOR SELECT Name , junk INTO compiling embedded cached query from Embedded.mac
```

埋め込み SQL を変更したくない場合はどうなりますか？ これは、Frozen Query Plans を使用することで達成できます。 簡単に言うと、IRIS がメジャーアップグレードされるたびに、すべての SQL ステートメントはフリーズされます。そのため、ユーザーが許可しない限り、何も変更されることはありません。これについては、こちらをお読みください。

さて、UCQ の処理の話に戻りましょう。 アプリケーションの埋め込み SQL プランをフリーズするには、以下の 3 つの方法があります。

IRIS.DAT を出荷する場合:

a. Do $SYSTEM.OBJ. GenerateEmbedded() で、埋め込み SQL の UTC を生成します。
b. Do $SYSTEM.SQL.Statement.FreezeAll() で、プランをフリーズします。
c. IRIS.DAT を出荷します。

xml ファイルを使用する場合:

a. Do $SYSTEM.OBJ. GenerateEmbedded() で、埋め込み SQL の UTC を生成します。
b. Do $SYSTEM.SQL.Statement.FreezeAll() で、プランをフリーズします。
c. Do $SYSTEM.SQL.Statement.ExportAllFrozenPlans() で、フリーズしたプランをエクスポートします。
d. Do $SYSTEM.SQL.Statement.ImportFrozenPlans()
   で、アプリケーションを読み込んだ後に、フリーズしたプランを読み込みます。

顧客サイトで UTC プランをフリーズする場合:

a. 埋め込み SQL のあるコードを顧客システムに読み込みます。
b. Do $SYSTEM.OBJ. GenerateEmbedded() で、埋め込み SQL の UTC を生成します。
c. Do $SYSTEM.SQL.Statement.FreezeAll() で、生成されたすべてのプランをフリーズします。

以前の動作に戻すことはできますか？ いいえ。これが現在の動作です。
開発者の観点からは、/compileembedded=1
というフラグをコンパイラのオプションに追加することで、以前の動作に戻すことができます。
これによって、クラスまたはルーチンをコンパイル中に UCQ
クラスを生成するようにコンパイラに指示されます。 SQL
に問題がある場合は、以前と同様にコンパイル時に報告されます。

```
Compiling routine : Embedded.mac
ERROR: Embedded.mac(5) : SQLCODE=-29 : Field 'JUNK' not found in the applicable tables^DECLARE cur
CURSOR FOR SELECT Name , junk INTO compiling embedded cached query from Embedded.mac
Detected 1 errors during compilation in 0.034s.
```

埋め込み SQL を初めて実行するときに UCQ クラスを生成するオーバーヘッドが気になる場合は、このステップをアプリケーションインストールに加えて、これらをすべて予め生成することが可能です: Do $SYSTEM.OBJ. GenerateEmbedded()

これはユニバーサルキャッシュクエリと埋め込み SQL を非常に全般的に見た概要です。
内部で何が行われているかについての実際の詳細には触れていません。 ユーザーが IRIS で埋め込み SQL を使用する際に遭遇する可能性のあることについて説明することにしました。 全体的に UCQ に移行すると、すべてのタイプの SQL で SQL のパフォーマンスの一貫性が高まり、本番システムでの SQL の更新がより簡単になります。 いくらかの調整が必要となるでしょうが、
コンパイラフラグの追加が大きく役立ちます。
今後は、生成されたコードを新しい場所で探すことに慣れる必要があります。 この件や InterSystems IRIS での SQL に関するその他についてのご質問、コメント、懸念があれば、私にお知らせください。

#SQL #InterSystems IRIS

ソースURL:https://jp.community.intersystems.com/post/intersystems-iris-%E3%81%AE%E6%96%B0%E3%81%97%E3%81%84%E5%9F%8B%E3%82%81%E8%BE%BC%E3%81%BF-sql