Article
Irène Mykhailova · Avr 25, 2022 7m de lecture

Embedded SQL - nouveauté dans InterSystems IRIS

@Benjamin De Boe a écrit un excellent article sur les Universal Cached Queries, mais qu'est-ce qu'une Universal Cached Query (UCQ) et pourquoi devrais-je m'en préoccuper si j'écris du bon vieux Embedded SQL? Dans Caché et Ensemble, les Cached Queries seraient générées pour résoudre xDBC et Dynamic SQL. Maintenant, l'Embedded SQL d'InterSystems IRIS a été mis à jour pour utiliser les Cached Queries, d'où l'ajout du mot Universal au nom. Désormais, tout SQL exécuté sur IRIS le sera à partir d'une classe UCQ.

Pourquoi InterSystems a-t-il fait cela ? Bonne question ! Le grand avantage ici est la flexibilité dans un environnement opérationnel. Dans le passé, si vous ajoutez un index ou exécutez TuneTable, le SQL dans Cached Queries utilisait immédiatement ces nouvelles informations tandis que l'Embedded SQL restait inchangé jusqu'à ce que la classe ou la routine soit compilée manuellement. Si votre application utilisait des classes déployées ou n'expédiait que du code OBJ, la recompilation sur le système du client n'était pas envisageable. Désormais, toutes les SQL Statements sur un système utiliseront la dernière définition de classe et les dernières données de réglage disponibles. À l'avenir, InterSystems IRIS disposera d'outils optionnels permettant de surveiller et de régler vos systèmes de production chaque soir, en personnalisant les plans SQL en fonction de la manière dont les tableaux sont utilisés. La puissance de l'Universal Cached Query augmentera au fur et à mesure que ces outils se développeront.

Est-ce que mon Embedded SQL est plus lent maintenant ? Oui et non. Le fait de lancer une balise dans une routine différente est un peu plus coûteux que de lancer une balise dans la même routine, donc c'est plus lent, mais la génération du code UCQ était différente de celle de la version intégrée, et le fait de pouvoir utiliser ces changements compense largement la dépense liée à l'appel d'une routine différente. Y a-t-il des cas où le code UCQ est plus lent ? Oui, mais dans l'ensemble, les performances devraient être meilleures. Je suis un spécialiste de l'Embedded SQL depuis longtemps et j'aime toujours souligner que l'Embedded SQL est plus rapide que le Dynamic SQL. Il est toujours plus rapide, mais avec tout le travail qui a été fait pour rendre les objets plus rapides, la marge entre les deux styles est assez faible pour que je n'aille pas me moquer de vous si vous utilisez le SQL dynamique.

De quelle manière dois-je vérifier les erreurs maintenant ? Le traitement des erreurs pour Embedded SQL n'a pas changé. Le code SQLCODE sera défini par un nombre négatif si nous rencontrons une erreur et %msg sera défini par la description de cette erreur. Ce qui a changé, ce sont les types d'erreurs que vous pouvez obtenir. Par défaut, le SQL ne sera pas compilé avant la première exécution de la requête. Cela signifie que si vous avez mal orthographié un champ ou un tableau dans la routine, l'erreur ne sera pas signalée lors de la compilation de cette routine, elle sera signalée la première fois que vous exécuterez le SQL, comme pour le Dynamic SQL. Le code SQLCODE est défini pour chaque commande SQL, mais si vous êtes paresseux comme moi, vous ne vérifiez jamais le code SQLCODE qu'après un FETCH. Eh bien, maintenant, vous pourriez vouloir commencer à vérifier sur l'OPEN aussi.

```
&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
```

Dans le code ci-dessus, j'ai un champ invalide dans le SELECT. Comme nous ne compilons pas le SQL lorsque nous compilons la routine, cette erreur n'est pas signalée. Lorsque j'exécute le code, l'OPEN signale l'erreur de compilation tandis que le FETCH et le CLOSE signalent une erreur de curseur non ouvert. %msg n'est pas modifié, donc si vous le vérifiez à tout moment, vous obtiendrez des informations utiles :

```
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
```

Que faire si je ne veux pas modifier mon Embedded SQL ? Vous pouvez toujours le faire en utilisant les Frozen Query Plans. Une petite remarque secondaire, chaque mise à niveau majeure d'IRIS que vous faites bloque toutes les SQL Statements , donc rien ne changera si vous ne le laissez pas faire. Plus d'informations à ce sujet sont disponibles ici.

Revenons maintenant à la gestion de l'UCQ. Voici trois façons de bloquer les plans d'Embedded SQL dans votre application :

- Si vous envoyez un IRIS.DAT :
  a. Do $SYSTEM.OBJ.GenerateEmbedded() pour générer l'UTC pour l'Embedded SQL
  b. Bloquez les plans : do $SYSTEM.SQL.Statement.FreezeAll()
  c. Envoyez le IRIS.DAT
- Si vous utilisez des fichiers xml :
  a. Do $SYSTEM.OBJ.GenerateEmbedded() pour générer l'UTC pour l'Embedded SQL
  b. Bloquez les plans : do $SYSTEM.SQL.Statement.FreezeAll()
  c. Exporter les plans bloqués : do $SYSTEM.SQL.Statement.ExportAllFrozenPlans()
  d. Après avoir téléchargé votre application, téléchargez les plans bloqués : do $SYSTEM.SQL.Statement.ImportFrozenPlans()
- Bloquez les plans UTC sur le site du client :
  a. Chargez le code avec l'Embedded SQL sur le système du client
  b. Do $SYSTEM.OBJ.GenerateEmbedded() pour générer l'UTC pour l'Embedded SQL
  c. Bloquez tous les plans qui ont été générés : do $SYSTEM.SQL.Statement.FreezeAll()

Puis-je revenir au comportement précédent ? Non, la situation est telle qu'elle est aujourd'hui. Du point de vue du développeur, vous pouvez revenir à l'ancien comportement en ajoutant l'indicateur aux options de votre compilateur. Cela indiquera au compilateur de générer la classe UCQ lors de la compilation de la classe ou de la routine. S'il y a un problème avec le SQL, il sera signalé au moment de la compilation, comme c'était le cas auparavant.

```
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.
```

Si vous êtes inquiet de la surcharge liée à la génération des classes UCQ la première fois que l'Embedded SQL est exécuté, vous pouvez ajouter cette étape dans le cadre de l'installation de votre application pour les générer toutes à l'avance : do $SYSTEM.OBJ.GenerateEmbedded()

Il s'agit d'un aperçu très rapide de Universal Cached Queries et Embedded SQL. Je ne suis pas entré dans les détails de ce qui se passe à l'intérieur. J'ai simplement essayé de parler des problèmes que les gens peuvent rencontrer lorsqu'ils travaillent avec Embedded SQL dans IRIS. Globalement, le passage à UCQ devrait rendre les performances de SQL plus cohérentes pour tous les types de SQL et faciliter la mise à jour de SQL sur un système de production. Il y aura quelques rajustements. L'ajout de l'indicateur du compilateur me sera d'une grande aide. Maintenant, je dois juste m'habituer à chercher le code généré dans un nouvel endroit. Si vous avez des questions, des commentaires, des doutes à ce sujet, ou tout autre sujet lié à SQL dans InterSystems IRIS, faites-le moi savoir.

#SQL #InterSystems IRIS