Artículo
Alberto Fuentes · 18 ene, 2022 Lectura de 7 min

SQL embebido, ¿qué hay de nuevo en InterSystems IRIS?

Benjamin De Boe escribió este magnífico artículo sobre las Consultas universales en caché, pero ¿qué es una Consulta universal en caché (UCQ) y por qué deberían interesarme, si yo escribo en el antiguo y válido SQL embebido? En Caché y Ensemble, las consultas en caché o cacheadas se generaban para resolver xDBC y SQL dinámico. Ahora, en InterSystems IRIS, SQL embebido se ha sido actualizado para utilizar las consultas cacheadas (*Cached Queries*), de ahí que se añadiera la palabra "universal" en el nombre. Actualmente, cualquier SQL que se ejecute en IRIS lo hará desde una clase UCQ.

¿Por qué InterSystems ha implementado este cambio? ¡Esa es una buena pregunta! La gran ventaja aquí es la flexibilidad que se obtiene en un entorno real. En el pasado, si se añadía un índice o se ejecutaba TuneTable, el SQL de las Consultas en caché hacía uso de esta nueva información inmediatamente, mientras que SQL integrado permanece sin cambios hasta que la clase o rutina se compila manualmente. Si tu aplicación utilizaba clases implementadas o sólo enviaba código de tipo OBJ, compilar esto nuevamente en el sistema del cliente no era una opción. Actualmente, todas las sentencias SQL de un sistema utilizarán la última clase que se definió y los últimos datos de ajuste disponibles. En el futuro, InterSystems IRIS dispondrá de herramientas opcionales que podrán supervisar y ajustar tus sistemas de producción durante la noche, lo que permitirá personalizar los planes SQL en función de cómo se consulten las tablas. A medida que este conjunto de herramientas crezca, la capacidad de las Consultas universales en caché también lo hará.

¿Mi SQL embebido es más lento ahora? Sí y no. Llamar a una etiqueta en una rutina diferente es un poco más caro que llamar a una etiqueta en la misma rutina, así que es más lento, pero la generación de código UCQ era diferente de la embebida, y lograr que se utilicen esos cambios compensa y mucho el gasto de llamar a una rutina diferente.¿Hay casos en los que el código UCQ es más lento? Sí, pero en general se debería ver un mejor rendimiento. Yo utilizo SQL embebido desde hace mucho tiempo, y siempre me gusta señalar que SQL embebido es más rápido que SQL dinámico. Sigue siendo más rápido, pero con todo el trabajo que se ha hecho para que los objetos sean más rápidos, la diferencia entre los dos estilos es lo suficientemente pequeña como para que no me burle de los que usan SQL dinámico 

¿Cómo puedo comprobar los errores ahora? La gestión de errores para el SQL embebido no ha cambiado. SQLCODE será un número negativo si encontramos un error y %msg establecerá la descripción de ese error. Lo que ha cambiado son los tipos de errores que se pueden obtener. Ahora, el comportamiento predeterminado consiste en que SQL no será compilado hasta que se inicie la consulta por primera vez. Esto significa que si escribe mal un campo o una tabla en la rutina, el error no se reportará cuando se compile esa rutina, sino la primera vez que se ejecute el SQL, igual que en SQL dinámico. SQLCODE se establece para cada comando SQL, pero si eres tan perezoso como yo, sólo compruebas SQLCODE después de un FETCH. Bueno, quizás ahora también quieras comprobarlo con OPEN.

```
&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
```

En este código, tengo un campo no válido en el SELECT. Como no compilamos el SQL cuando compilamos la rutina, este error no se reporta. Cuando ejecuto el código, OPEN reporta un error en la compilación mientras que FETCH y CLOSE reportan un error de cursor no abierto. %msg no experimenta ningún cambio, por lo tanto, en cualquier momento que realices la comprobación, obtendrás información útil:

```
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
```

¿Qué pasa si no quiero realizar ninguna modificación en mi SQL embebido? Puedes hacerlo utilizando los Planes de consulta congelados. Una nota rápida, cada actualización importante en IRIS suspenderá todas las sentencias SQL, de modo que no se realizará ningún cambio si no se autoriza. Puedes leer más información sobre esto aquí.

Ahora, volvamos a hablar sobre el tema de las UCQ. Os muestro 3 formas de congelar los planes SQL embebidos en tu aplicación:

- Si envías un IRIS.DAT:
    a. do $SYSTEM.OBJ.GenerateEmbedded() para generar el UTC para SQL embebido
    b. Congelar los planes: do $SYSTEM.SQL.Statement.FreezeAll()
    c. Envía el IRIS.DAT
- Si utilizas archivos XML:
    a. do $SYSTEM.OBJ. GenerateEmbedded() para generar el UTC para SQL embebido
    b. Congelar los planes: do $SYSTEM.SQL.Statement.FreezeAll()
    c. Exporta los planes suspendidos: do $SYSTEM.SQL.Statement.ExportAllFrozenPlans()
    d. Después de cargar la aplicación, carga los planes suspendidos: do $SYSTEM.SQL.Statement.ImportFrozenPlans()
- Congelar los planes UTC en el sitio del cliente:
    a. Carga el código con SQL embebido en el sistema del cliente
    b. do $SYSTEM.OBJ.GenerateEmbedded() para generar el UTC para SQL embebido
    c. Congelar todos los planes que se generaron: do $SYSTEM.SQL.Statement.FreezeAll()

¿Puedo volver al comportamiento anterior? No, ahora funciona de esta manera. Desde el punto de vista del desarrollador, puedes recuperar el comportamiento anterior si añades la marca /compileembedded=1 en las opciones del compilador. Esto le dirá al compilador que genere la clase UCQ mientras compila la clase o la rutina. Si hay algún problema con el SQL, se informará de ello durante la compilación, como ocurría en el pasado.

```
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 te preocupa que ocurra una sobrecarga mientras se generan las clases UCQ la primera vez que se ejecute SQL embebido, puedes añadir este paso como parte de la instalación de tu aplicación para que todas las clases se generen por adelantado: do $SYSTEM.OBJ. GenerateEmbedded().

Este es un resumen de muy alto nivel acerca de las Consultas universales en caché y SQL embebido. Realmente no he entrado en detalles sobre lo que ocurre durante todo el proceso. Solo he tratado de hablar acerca de las cosas que la gente encontraría al trabajar con SQL embebido en IRIS. En general, cambiarse a UCQ debería hacer que el rendimiento de SQL sea más consistente en todos los tipos de SQL y facilitar la actualización de SQL en un sistema en producción. Desde luego, habrá algunas modificaciones. Añadir la marca del compilador será bastante útil para mí. Ahora, solo debo acostumbrarme a buscar el código que se ha generado en un nuevo lugar.

Si tienes alguna pregunta, comentario, o duda sobre SQL en InterSystems IRIS, no dudes en escribirla aquí.

#SQL #InterSystems IRIS