Artículo
David Reche · 23 jul, 2019 Lectura de 5 min

Clases, tablas y globals - ¿Cómo funcionan?

¡Hola a tod@s!

Cuando hablo con alguien de perfil técnico por primera vez acerca de InterSystems IRIS, siempre comienzo hablando de que en el centro de todo InterSystems IRIS es una Base de Datos Multimodelo. En mi opinión, esta es la mayor ventaja (desde la visión de Sistemas de Bases de Datos), ya que:

- ¿Quieres obtener un resumen o partes específicas de tus datos? Usa SQL!
- ¿Necesitas trabajar de forma intensiva con un registro? Usa Objetos!
- ¿Quieres establecer un valor y conoces la clave? ~~Piensalo de nuevo.~~ Usa globals!

Y en todos los casos, el dato está almacenado de forma única. ¡Tú eliges la manera en la que quieres acceder al mismo!!

De un primer vistazo es una bonita historia - corta, concisa y con un mensaje; pero cuando se empieza a trabajar con InterSystems IRIS, comienzan a surgir preguntas: ¿Cómo están relacionados las clases, las tablas y los globals? ¿Qué son cada uno para el otro? ¿Cómo se almacenan realmente los datos?

En este artículo voy a tratar de responder estas preguntas y explicar qué está pasando realmente.

Primera parte. Orientación del Modelo.

Los que trabajan con datos tienden a tener un sesgo diferente dependiendo del modelo con el que usualmente trabajan.

Los desarrolladores piensan en objetos. Para ellos, las bases de datos y las tablas son cajas contra las que se interactua vía Guardar/Recuperar (preferiblemente sobre ORM - Object Relational Mapper). Pero la estructura para ellos siempre son objetos (por supuesto esto es principalmente cierto para desarrolladores en lenguajes orientados a objetos - la mayoría de nosotros).

Por otro lado los DBA suelen pensar en los datos como tablas - consecuencia de trabajar con bases de datos relacionales. Los objetos son solo la representación de una fila en este caso.

Como las clases persistentes de InterSystems IRIS también son una tabla que tienen los datos almacenados en globals, creo que es importante aclarar estos conceptos.

Segunda Parte. Ejemplo.

Digamos que creamos la clase persistente Point:

```
Class try.Point Extends %Persistent [DDLAllowed]
{

Property X;
Property Y;
}
```

Podríamos igualmente utilizar este DDL para crearla:

```
CREATE Table try.Point (
    X VARCHAR(50),
    Y VARCHAR(50))
```

Y se crearía la misma clase.

Después de la compilación, nuestra nueva clase debería tener una estructura Storage autogenerada que define el mapeo de globals a los datos en las columnas o propiedades:

```
Storage Default
{
<Data name="PointDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>X</Value>
</Value>
<Value name="3">
<Value>Y</Value>
</Value>
</Data>
<DataLocation>^try.PointD</DataLocation>
<DefaultData>PointDefaultData</DefaultData>
<IdLocation>^try.PointD</IdLocation>
<IndexLocation>^try.PointI</IndexLocation>
<StreamLocation>^try.PointS</StreamLocation>
<Type>%Library.CacheStorage</Type>
}
```

¿Qué está pasando aquí?

Desde abajo hacia arriba (**en negrita** está lo importante, puedes ignorar el resto por ahora):

- Type: tipo de almacenamiento generado. Een nuestro caso este es el tipo de almacenamiento por defecto para las clases persistentes
- StreamLocation - nombre del global donde se almacenan los Streams
- IndexLocation - nombre del global donde se almacenan los índices
- IdLocation - nombre del global donde almacenamos el contador del ID autoincremental
- **DefaultData** - nombre del elemento de almacenamiento XML que define el mapeo entre las columnas/propiedades y el global
- **DataLocation** - nombre del global donde almacenamos los datos

En este momento nuestro DefaulData es PointDefaultData, así que vamos a verlo. En esencia nos dice qué nodo del global almacena qué valor, con la siguiente estructura:

- 1 - %%CLASSNAME
- 2 - X
- 3 - Y

De forma que podemos esperar que nuestro global tenga esta pinta:

```
^try.PointD(id) = %%CLASSNAME, X, Y
```

Si sacamos por pantalla el valor del global, inicialmente estará vacío, porque no hemos añadido ningún dato:

```
zw ^try.PointD
```

Vamos a añadir un objeto:

```
set p = ##class(try.Point).%New()
set p.X = 1
set p.Y = 2
write p.%Save()
```

Revisamos el global de nuevo y ¿qué tenemos?

```
zw ^try.PointD
^try.PointD=1
^try.PointD(1)=$lb("",1,2)
```

Como puedes ver, nuestra estructura esperada (`%%CLASSNAME, X, Y`) está establecida con `$lb("",1,2)`, que corresponde a las propiedades X e Y de nuestro objeto (`%%CLASSNAME` es una propiedad del sistema, podemos ignorarla por ahora).

Podemos igualmente añadir una nueva fila mediante SQL:

```
INSERT INTO try.Point (X, Y) VALUES (3,4)
```

Ahora nuestro global tiene esta pinta:

```
zw ^try.PointD
^try.PointD=2
^try.PointD(1)=$lb("",1,2)
^try.PointD(2)=$lb("",3,4)
```

De esta forma, siempre que añadimos datos vía objetos o SQL, éstos son almacenados de acuerdo a la definición del Storage (eEs posible modificar manualmente la definición del Storage - puedes probar tú mismo y ver qué pasa).

Ahora bien, ¿qué pasa cuando queremos ejecutar una consulta SQL?

```
SELECT * FROM try.Point
```

Esta consulta se traduce en código ObjectScript, que itera sobre el global definido y puebla las columnas en base a la definición del Storage.

Ahora vamos a por las modificaciones. Borremos todos los datos de la tabla.

```
DELETE FROM try.Point
```

Veamos el global después de esto:

```
zw ^try.PointD
^try.PointD=2
```

Fijaos que solo el contador del ID se mantiene, de esta forma el siguiente nuevo objeto/fila tendrá el ID=3. Nuestra clase y tabla continúa existiendo.

Pero ¿qué pasa si hacemos?:

```
DROP TABLE try.Point
```

Esto destruye nuestra tabla, clase y borra todo el global.

```
zw ^try.PointD
```

Espero que siguiendo este ejemplo ahora entiendas mejor cómo los globals, las clases y las tablas se corresponden unos con otros.

#Globals #Mejores prácticas #Modelo de datos de objetos #Principiante #SQL #Tablas relacionales #InterSystems IRIS

URL de fuente:https://es.community.intersystems.com/post/clases-tablas-y-globals-%C2%BFc%C3%B3mo-funcionan