

Article

[Irene Mykhailova](#) · Mars 28, 2022 5m de lecture

Les classes, les tables et les globales - comment tout cela fonctionne?

Lorsque je décris InterSystems IRIS à des personnes plus orientées vers la technique, je commence toujours par dire qu'il s'agit d'un DBMS (système de gestion de base de données) multi-modèle.

À mon avis, c'est son principal avantage (du côté du DBMS). Et les données ne sont stockées qu'une seule fois. Vous choisissez simplement l'API d'accès que vous voulez utiliser.

- Voulez-vous une sorte de résumé pour vos données ? Utilisez SQL !
- Souhaitez-vous travailler en profondeur avec un seul enregistrement ? Utilisez des objets !
- Voulez-vous accéder ou définir une valeur et vous connaissez la clé ? Utilisez les globales !

À première vue, c'est une belle histoire - courte et concrète, elle fait passer le message, mais lorsque les gens commencent vraiment à travailler avec InterSystems IRIS, les questions apparaissent. Comment les classes, les tables et les globales sont-ils liés ? Que sont-ils les uns pour les autres ? Comment les données sont-elles réellement stockées ?

Dans cet article, je vais essayer de répondre à ces questions et d'expliquer ce qui se passe réellement.

Première partie. Le biais des modèles.

Les personnes qui travaillent avec des données ont souvent un biais en faveur du modèle avec lequel elles travaillent.

Les développeurs pensent en objets. Pour eux, les bases de données et les tableaux sont des boîtes avec lesquelles vous interagissez via CRUD (Créer-Lire-Mettre à jour-Supprimer, de préférence via ORM), mais le modèle conceptuel sous-jacent est constitué d'objets (bien sûr, c'est surtout vrai pour les développeurs utilisant des langages orientés objet - donc la plupart d'entre nous).

D'autre part, pour avoir passé beaucoup de temps dans des DBMS relationnels, les DBA considèrent souvent les données comme des tables. Dans ce cas, les objets ne sont que des enveloppes sur les lignes.

Et avec InterSystems IRIS, une classe persistante est aussi un table, qui stocke les données en globale, donc une clarification est nécessaire.

Deuxième partie. Un exemple.

Disons que vous avez créé une classe Point :

```
Class try.Point Extends %Persistent [DDLAllowed]
{
    Property X;
    Property Y;
}
```

Vous pouvez également créer la même classe avec DDL/SQL :

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

Après la compilation, notre nouvelle classe aurait généré automatiquement une structure de stockage qui fait correspondre les données qui sont nativement stockées dans les globaux aux colonnes (ou aux propriétés si vous êtes un penseur orienté objet) :

```
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>  
}
```

Qu'est-ce qui se passe ici ?

De bas en haut (les mots en gras sont importants, ignorez le reste) :

- Type : le type de stockage généré, dans notre cas le stockage par défaut pour les objets persistants
- StreamLocation - l'endroit où nous stockons les flux de données séquencé
- IndexLocation - la globale pour les indices
- IdLocation - la globale où nous stockons l'ID compteur autoincrémental
- DefaultData - l'élément XML de stockage qui fait correspondre la valeur globale aux colonnes/propriétés
- DataLocation - la globale dans lequel les données sont stockées

Maintenant notre "DefaultData" est PointDefaultData alors regardons de plus près sa structure. Essentiellement, cela dit que le noeud global a cette structure :

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

On peut donc s'attendre à ce que notre globale ressemble à ceci :

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

Mais si nous imprimons notre globale, il sera vide car nous n'avons pas ajouté de données :

```
zw ^try.PointD
```

Ajoutons un objet :

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

Et voici notre globale

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

Comme vous le voyez, notre structure attendue %%CLASSNAME, X, Y est définie avec \$lb("",1,2) qui correspond aux propriétés X et Y de notre objet (%%CLASSNAME est une propriété du système, ignorez-la).

Nous pouvons également ajouter une ligne via SQL :

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

Maintenant, notre globale ressemble à ceci :

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

Ainsi, les données que nous ajoutons par le biais d'objets ou de SQL sont stockées dans des globales en fonction des définitions de stockage (remarque : vous pouvez modifier manuellement la définition de stockage en remplaçant X et Y dans PointDefaultData - vérifiez ce qu'il arrive aux nouvelles données !)

Maintenant, que se passe-t-il lorsque nous voulons exécuter une requête SQL ?

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

Elle est traduite en code ObjectScript qui itère sur la globale ^try.PointD et remplit les colonnes en fonction de la définition du stockage - la partie PointDefaultData précisément.

Maintenant pour les modifications. Supprimons toutes les données du table :

```
DELETE FROM try.Point
```

Et voyons notre globale à ce stade :

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

Notez que seul le compteur d'ID est laissé, donc le nouvel objet/ligne aura un ID=3. De même, notre classe et notre table continuent d'exister.

Mais que se passe-t-il quand on lance :

```
DROP TABLE try.Point
```

Il détruirait le table, la classe et supprimerait la globale.

```
zw ^try.PointD
```

Si vous avez suivi cet exemple, j'espère que vous comprenez maintenant mieux comment les globales, les classes et les tables s'intègrent et se complètent. L'utilisation de la bonne API pour le travail à effectuer permet un développement plus rapide, plus agile et moins bogué.

[#Débutant](#) [#Globals](#) [#Object Data Model](#) [#SQL](#) [#Tables relationnelles](#) [#InterSystems IRIS](#)

URL de la source: <https://fr.community.intersystems.com/post/les-classes-les-tables-et-les-globales-comment-tout-cela-fonctionne>