Discussion
Sylvain Guilbaud · Feb 21, 2022

How to efficiently store historical data, similar to current data, without mixing physical storage ?

Let's consider you would like to efficiently store your historical data in a similar structure than the one used for your current data, but without sharing the same physical storage (ie : not in the same global). What is the most efficient way to do it ?

Below a simple class of your current data :

```
Class data.current.person Extends (%Persistent, %Populate)
{

Parameter DEFAULTGLOBAL = "^on.person";

Property name As %String;

Property dob As %Date(FORMAT = 4);

Property activ As %Boolean [ InitialExpression = 1 ];

Property created As %TimeStamp [ InitialExpression = {$zdt($h,3)} ];

Storage Default
{
<Data name="personDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>name</Value>
</Value>
<Value name="3">
<Value>dob</Value>
</Value>
<Value name="4">
<Value>activ</Value>
</Value>
<Value name="5">
<Value>created</Value>
</Value>
</Data>
<DataLocation>^on.personD</DataLocation>
<DefaultData>personDefaultData</DefaultData>
<IdLocation>^on.personD</IdLocation>
<IndexLocation>^on.personI</IndexLocation>
<StreamLocation>^on.personS</StreamLocation>
<Type>%Storage.Persistent</Type>
}
}
```

NB : you can omit the *DEFAULTGLOBAL* parameter if you're happy with the one automatically generated (which is the case, most of the time) ; I use it in my example to emphasize the separation of globals in *current* and *archive* classes.

First approach

You could first think about creating 2 persistent classes and copy the same definition in both classes (except the Storage clause which will be recreated at the first compilation) :

```
Class data.current.person Extends %Persistent {}

Class data.archive.person Extends %Persistent {}
```

But this approach will rapidly reveal itself disastrous in terms of maintenance, for obvious reason of duplicating the code/definitions.

Second approach

Another idea could be to create a *common* class with *ClassType* = "" in order to not create the storage definition at its level. Then, you can make *archive* and *current* classes inherit from this common class and add *%Persistent* inheritance in both subclasses.

```
Class data.common.person Extends %Persistent [ ClassType = "" ] {}

Class data.current.person Extends (%Persistent,data.common.person) [ClassType = persi
stent ] {}

Class data.archive.person Extends (%Persistent,data.common.person) [ ClassType = pers
istent ] {}
```

This way, you solve the question and get separate globals for each persistent class with a unique class to store all the definition.

But, this is not really convenient to manage 3 classes instead of 2 and to have to maintain the definition in the common class ; neither to be obliged to open the *common* class to get the detail of the definition of the *current* class.

Third approach

Well, a third solution could be to return to a very simple approach : just create an archive class which inherits from the current data class...

```
Class data.current.person Extends %Persistent {}

Class data.archive.person Extends data.current.person {}
```

In that case, current and historical data will share the **same storage definition** and, as a result, their data will be stored in the **same globals**, which is not really convenient to physically separate both kind of data and thus not answering to our need.

A simpler approach

You can obtain a similar structure in *archive* class which inherits from *current*, **without** inheriting from its storage, by simply adding *%Persistent* inheritance **before** the class you want to inherit from.

In this case, even if the super class already contains a storage class with DataLocation (because of its persistent type), at compile time, the inheritance precedence will force the generation of a **new storage clause** based on the subclass definition, **with its own storage** for DataLocation, IdLocation, IndexLocation, StreamLocation.

```
Class data.archive.person Extends (%Persistent, data.current.person)
{
Parameter DEFAULTGLOBAL = "^off.person";

Storage Default
{
<Data name="personDefaultData">
<Value name="1">
<Value>%%CLASSNAME</Value>
</Value>
<Value name="2">
<Value>name</Value>
</Value>
<Value name="3">
<Value>dob</Value>
</Value>
<Value name="4">
<Value>activ</Value>
</Value>
<Value name="5">
<Value>created</Value>
</Value>
</Data>
<DataLocation>^off.personD</DataLocation>
<DefaultData>personDefaultData</DefaultData>
<IdLocation>^off.personD</IdLocation>
<IndexLocation>^off.personI</IndexLocation>
<StreamLocation>^off.personS</StreamLocation>
<Type>%Storage.Persistent</Type>
}
}
```

Other ideas ?

If you have other ways to address this question, please share your ideas and comments.

#Data Model #Globals #Caché #InterSystems IRIS