Article
Kyle Baxter · Aug 29, 2016 6m read

Data Storage - Information You Must Know to Make Good Decisions When Developing

This post is the direct result of working with an InterSystems customer who came to me with the following problem:

```
SELECT COUNT(*) FROM MyCustomTable
```

Takes 0.005 seconds, total 2300 rows. However:

```
SELECT * FROM MyCustomTable
```

Took minutes. The reason for this is subtle and interesting enough for me to write a post about. This post is lengthy, but if you scroll to the bottom I'll write a quick summary, so if you've gotten this far and think you've already read enough, scroll to the end to get the main point. Check for the sentence in **bold**.

There is consideration to take when creating your classes when it comes to storage. As many of you know, all data in Caché is stored in Globals.

<Digression>

If you don't know this then I think this post is going to be a bit much. I recommend checking out this excellent tutorial in our docs:

http://docs.intersystems.com/latest/csp/docbook/DocBook.UI.Page.cls?KEY=...

If you have never used Caché/Ensemble/HealthShare the above tutorial is very valuable, and even if you have, it is worth some of your day to check it out!

</Digression>

Now, because all the data is stored in Globals, it is important to understand how your class definitions map to globals. Let's build an application together! We'll go over some common pitfalls and discuss how your class development effects your storage strategies with a special look at SQL performance.

Let's pretend we're the US Census Bureau and we want to have a database to store information for all the people in the USA. So we build a class like so:

```
Class USA.Person extends %Persistent

{


 Property Name as %String;
&nbsp;Property SSN as %String;
```

```
 Property Address as %String;


 Property DateOfBirth as %Date;


}
```

SSN stands for "Social Security Number" which, although not originally intended to be a persons ID number, is their de facto ID number. However, we're traditionalists so we will not use that for the ID. That said, we clearly want that indexed as it's a great way to look someone up. We know that sometimes we're going to have to look people up by name, so we'll want a name index, too. And because our boss loves his reports based on age buckets, we think a DOB index might be good, too. So let's add those to our class

```
Class USA.Person extends %Persistent

{


 Property Name as %String;


 Property SSN as %String;


 Property Address as %String;


 Property DateOfBirth as %Date;


 Index NameIDX On Name;


 Index SSNIDX On SSN [Unique];


 Index DOBIDX on DateOfBirth;


}
```

Alright. So let's add a row and see what our globals look like. Our INSERT statement looks like so:

```
INSERT INTO USA.Person (Name,SSN,Address,DateOfBirth) VALUES
   ('Baxter, Kyle','111-11-1111','1 Memorial Drive, Cambridge, MA 02142','1985-07-20'
)
```

And the global:

```
USER>zw ^USA.PersonD


^USA.PersonD=1



^USA.PersonD(1)=<b>$lb(</b>"","Baxter, Kyle","111-11-1111","1 Memorial Drive, Cambrid
ge, MA 02142",52796<b>)</b>
```

The default storage for a class stores your data in ^Package.ClassD. If the class name is too long it can be hashed, and you can find it in the Storage Definition at the bottom of your class definition. What do the indexes look like?

```
USER>zw ^USA.PersonI


^USA.PersonI("DOBIDX",52796,1)=""



^USA.PersonI("NameIDX"," BAXTER, KYLE",1)=""



^USA.PersonI("SSNIDX"," 111-11-1111",1)=""
```

Excellent, our storage is looking pretty good so far. So we add our 320 million people and we can get people pretty quickly. But now we have a problem, because we want to treat the president and all ex-presidents with some special consideration. So we add a special class for the president:

```
Class USA.President extends USA.Person


{



Property PresNumber as %Integer;



Index PresNumberIDX on PresNumber;
}
```

Nice. Due to inheritance we get all the properties from USA.Person, and we add one to let us know which number president he was. Since I do want to get a little political, I'm going to INSERT our NEXT president. Here's the statement:

```
INSERT INTO USA.President (Name,SSN,DateOfBirth,Address,PresNumber) VALUES ('McDonald
,Ronald','221-18-7518','01-01-1963','1600 Pennsylvania Ave NW, Washington, DC 20006',
45)
```

Note: His SSN spells 'Burger'. Sorry if it is your SSN.

So this is great! Let's look at your President global:

```
USER>zw ^USA.PresidentD
```

No data! And here we get to the meat of this post. Because we decided to inherit from USA.Person FIRST, we inherited not only its properties and indexes, but also its storage! So to locate President McDonald we need to look in ^USA.PersonD. And we can see the following:

```
^USA.PersonD(2)=$lb(
"~USA.President~","McDonald,Ronald","221-18-7518","1600 Pennsylvania Ave NW, Washingt
on, DC 20006",44560)


^USA.PersonD(2,"President")=<b>$lb(</b>45<b>)</b>
```

Two things to note here. First is that we can see that the (2) node has all the information already stored in USA.Person. While the (2,"President") node has just the information specific to the USA.President class.

What does this mean practically? Well if we want to do a: SELECT * FROM USA.President then we will NEED to go through the full person table. If we expect the person table to have 320,000,000 rows, and the President Table to have 45, then we are doing over 320,000,045 global references to get out 45 rows! Indeed, if we look at the query plan:

- `Read master map USA.President.IDKEY, looping on ID.`

- `For each row:`
- ` Output the row.`

We see what we expect. However, we have already seen that this means necessarily looking through ^USA.PersonD global. So this is going to be a 320,000,000+ Global Reference as we need to test EACH ^USA.PersonD to check if there is data in ^USA.PersonD(i,"President") as we don't know which people will be presidents. Well this is bad! Not what we wanted at all! Whatever can we do!? Well we have 2 options:

<u>Option 1</u>

Add an extent index. If we do that then we get a list of IDs so we know which people are presidents and we can use that information to read specific nodes of the ^USA.Person global. Because I have default storage I can use a bitmap index which will make this even faster. We add the index like so:

```
Index Extent [Type=Bitmap, Extent];
```

And when we look at our query plan for SELECT * FROM USA.President we can see:

- `Read extent bitmap USA.President.Extent, looping on ID.`
- `For each row:`
- `Read master map USA.President.IDKEY, using the given idkey value.`
  `Output the row.`
-

Ah, now this is going to be nice and fast. One global reference to read the Extent and then 45 more for the presidents. That's pretty efficient.

Downsides? Well joining into this table becomes a little more troublesome and might involve some more temp tables than you'd like.

Option 2

Change the class definition to:

```
Class USA.President extends (%Persistent, USA.Person)
```

Making %Persistent the first class extended will mean that USA.President gets its own storage definition. So the Presidents will be stored like:

```
USER>zw ^USA.PresidentD


^USA.PresidentD=1



^USA.PresidentD(1)=<b>$lb(</b>"","McDonald,Ronald","221-18-7518","1600 Pennsylvania A
ve NW, Washington, DC 20006",44560,45<b>)</b>
```

So this is good, because selecting from USA.President just means reading the 45 members of this global. Nice and easy and with a good clean design.

Downsides? Well now the Presidents are NOT in the Person table. So if you want information about Presidents AND non-Presidents you need to do *SELECT ... FROM USA.Person UNION ALL SELECT ... FROM USA.President*

If you stopped reading at the beginning, start again here!

When creating inheritance we have two options

Option 1: Inherit from the superclass first. This stores the data in the same global as the superclass. Useful if you want to have all the information together, and you can mitigate performance problems in the subclass by having an extent index.

Option 2: Inherit from %Persistent first. This stores the data in a new global. Useful if you are going to query the subclass a lot, however if you want to see data from both the superclass and subclass, you need to use a UNION query.

Which of these is better? Well, that depends on how you are going to use your application. If you are going to want to do a lot of queries on all the data together, then you probably want the first approach. However, if you don't think you'll ever query the data all together, then you are probably going to want the second approach. Both are totally fine, so long as you remember the extent index in Option 1.

Questions? Comments? Long rambling thoughts? Leave them below!

#Data Model #Globals #Object Data Model #ObjectScript #SQL #Tips & Tricks #Tutorial

Source URL:https://community.intersystems.com/post/data-storage-information-you-must-know-make-good-decisions-when-developing