Article
Robert Cemper · Sep 10, 2021 4m read

Storage Considerations on large data sets

I'd like to share with you some storage features that also exist in Caché but are almost unknown and mostly not used. They are of course available in IRIS and gain importance with large and distributed storage architectures.

When you define a persistent class in your EDI or a table using DDL then you would typically select a meaningful name that relates to the content of your table in your context.
You know that data and indices are stored in Globals but you don't have to care for it.
The definition is done by the compiler based on the name of the class or the table.
Except for class names longer than 29 characters you can predict Standard Global Names quite easily as described in the documentation.

```
Class dc.MyCompany.EmployeeRegister Extends %Persistent
Storage Default
<DataLocation>^dc.MyCompany.EmployeeRegisterD</DataLocation>
<IdLocation>^dc.MyCompany.EmployeeRegisterD</IdLocation>
<IndexLocation>^dc.MyCompany.EmployeeRegisterI</IndexLocation>
<StreamLocation>^dc.MyCompany.EmployeeRegisterS</StreamLocation>
```

This is quite comfortable and allows you to concentrate on the logic around your solution.
It has almost no impact to performance as the excellent storage methods *inside* the databases use high efficient compression algorithms.

Outside the database we have various parts that suffer from long global names:
First the Journal on disk storage. Next any feature that transports Global names over some networks: pure ECP, Mirroring, Sharding.
Since the ranking in speed hasn't changed with all technology improvement:
Memory > Storage > *Network* > User.
So with intensive network traffic (Sharding, ECP) or Journal+Network (Mirror) there is some disadvantage.

To get control of Global names there is a parameter **DEFAULTGLOBAL**
applying this to the previous example we see this result:

```
/// Set the root of the global names
Parameter DEFAULTGLOBAL = "^dc.er";
Storage Default
<DataLocation>^dc.erD</DataLocation>
<IdLocation>^dc.erD</IdLocation>
<IndexLocation>^dc.erI</IndexLocation>
<StreamLocation>^dc.erS</StreamLocation>
```

This is much better but not yet the best you can achieve.
As you know with this approach every index is stored at the first subscript level of <IndexLocation>.

If your class/table has a lot of indices this first level could be significant.
This mix is not attractive if you have a mix of
- pure key indices
- collated indices
- indices with data
- bitmap indices & bitslice indices
it would be more efficient to have each index in its own global

To split indices on individual globals parameter USEEXTENTSET is the solution
Extending the previous example we see this result:

```
Parameter DEFAULTGLOBAL = "^dc.er";
Parameter USEEXTENTSET = 1;
Storage Default
<DataLocation>^dc.er.1</DataLocation>
<ExtentLocation>^dc.er</ExtentLocation>
<IdLocation>^dc.er.1</IdLocation>
<Index name="IDKEY"><Location>^dc.er.1</Location></Index>
<Index name="xDOB"><Location>^dc.er.2</Location></Index>
<Index name="xName"><Location>^dc.er.3</Location></Index>
<IndexLocation>^dc.er.I</IndexLocation>
<StreamLocation>^dc.er.S</StreamLocation>
```

Now each index has a homogeneous storage structure.
If you omit parameter DEFAULTGLOBAL you get generated Global names.
this is the default setup if you run Sharding. In this case:

```
Parameter USEEXTENTSET = 1;
Storage Default
<DataLocation>^BLWQ.C069.1</DataLocation>
<ExtentLocation>^BLWQ.C069</ExtentLocation>
<IdLocation>^BLWQ.C069.1</IdLocation>
<Index name="IDKEY"><Location>^BLWQ.C069.1</Location></Index>
<Index name="xDOB"><Location>^BLWQ.C069.2</Location></Index>
<Index name="xName"><Location>^BLWQ.C069.3</Location></Index>
<IndexLocation>^BLWQ.C069.I</IndexLocation>
<StreamLocation>^BLWQ.C069.S</StreamLocation>
```

Final remarks:
The effects of the 2 parameters will be recognized only with rather large data sets
and are especially seen in any Sharding installation.

You find more details in these documentations
Hashed Global Names , Index Global Names , Table Definition Optimization

#Globals #Object Data Model #Performance #Relational Tables #Caché #Ensemble #HealthShare #InterSystems IRIS #InterSystems IRIS Analytics (DeepSee) #InterSystems IRIS for Health

Source URL: https://community.intersystems.com/post/storage-considerations-large-data-sets