


Article

[Robert Cemper](#) · Feb 16, 2018  5m read

[Open Exchange](#)

## The adopted Bitmap

No doubt bitmap indexing, if used with a suitable property, performs just impressive!  
But there is a major limit: ID key has to be a positive integer.  
For modern class definitions working with CacheStorage this is a default.

BUT: There are hundreds (thousands ?) old applications out in the field that are still using composite ID keys.

Or - to name the origin - work on Globals with 2 subscript levels (or more).  
They are by construction excluded from our "Bitmap Wonderland".

Of course, using the feature of multiple storage maps could possibly allow to escape - somehow.  
But in those cases where you have many many GB of data stored in that way and where you work with millions of lines of old - often poorly documented - code.  
You wouldn't think seriously more than 5 min. if you change the storage structure or not.  
So the bitmap is not with you.

In most cases, the reason for this structure was to separate from each other data for companies / clients / departments / product groups / years of archive / similar bulk data.  
With the option in background to split data by global mapping to several databases.  
[It's not Sharding but somehow related]

A typical class might look like this (strongly simplified):

```
Class DC.RefOld Extends %Persistent [ Final ]
{
Property Grp As %String;
Property Number As %Integer(MINVAL = 1);
Property CountryCode As %String(VALUELIST =
",AT,US,UK,IT,DE,RU,CH,FR");
Property Name As %Name;
Property City As %String;
Index idx On (Grp, Number) [ IdKey ];
Index xCC On CountryCode;
Storage Default
{
<Data name="RefOldDefaultData">
  <Value name="1">
    <Value>CountryCode</Value>
  </Value>
  <Value name="2">
    <Value>Name</Value>
  </Value>
</Data>
}
```

```

    </Value>
    <Value name="3">
        <Value>City</Value>
    </Value>
</Data>
<DataLocation>^DC.RefOldD</DataLocation>
<DefaultData>RefOldDefaultData</DefaultData>
<IdLocation>^DC.RefOldD</IdLocation>
<IndexLocation>^DC.RefOldI</IndexLocation>
<StreamLocation>^DC.RefOldS</StreamLocation>
<Type>%Library.CacheStorage</Type>
}

```

Analysis of several installations showed me that variation of Group ( 1st subscript) is quite often small with just one dozen or two of distinct values and that queries across Groups are very rare. In addition, I observed this structure often with reference data with low change / insert rates.

The plan of this simple query :

```
select CountryCode,count(*) from DC.RefOld where grp=3 group by CountryCode
```

contains this key information:

Read master map `DC.RefOld.idx`, using the given Grp, and looping on Number.  
which means:

We do a table scan reading ALL subscripts of this branch (with all data) without any index

OK. let's be creative:

Based on the original class we create a ghost with a modified storage definition that fulfills the requirement: IDKEY is a positive integer.

```

Class DC.RefBit Extends %Persistent [ Final ]
{
    /// make it static
    /// Property Grp As %String;
    Parameter MANAGEDEXTENT = 0;
    Property Number As %Integer(MINVAL = 1);
    Property CountryCode As %String(VALUELIST =
    ",AT,US,UK,IT,DE,RU,CH,FR");
    Property Name As %Name;
    Property City As %String;
    Index idx On Number [ IdKey ];
    Index bxCC On CountryCode [ Type = bitmap ];
    /// Static method to pass along GRP
    ClassMethod SetGrp(GRP As %String) As %Integer [
SqlProc ]
    {
        set ^DC($username)=GRP
        quit $$$OK }
    Storage Default

```

```

{
<Data name="RefBitDefaultData">
  <Value name="1">
    <Value>CountryCode</Value>
  </Value>
  <Value name="2">
    <Value>Name</Value>
  </Value>
  <Value name="3">
    <Value>City</Value>
  </Value>
</Data>
<DataLocation>^DC.RefOldD(^DC($username))</DataLocation>
<DefaultData>RefBitDefaultData</DefaultData>
<IdLocation>^DC.RefOldD</IdLocation>
<IndexLocation>^DC.RefBitI
(^DC($username))</IndexLocation>
<StreamLocation>^DC.RefOldS</StreamLocation>
<Type>%Library.CacheStorage</Type>
}
}

```

The trick is to pass the value for the first subscript in our storage into the queries. And you have to pass it in a way that is accessible also for detached background jobs. For object access this is simple. For SQL queries there is a Stored Procedure SetGrp to pass along the required information. As it is independent of row content it is executed only once in your query.

This is the new query :

```

select CountryCode,count(*) from DC.RefBit
where DC.RefBit_SetGrp(3)=1 group by CountryCode

```

The query plan honors this by

Read bitmap index DC.RefBit.bxCC, looping on %SQLUPPER(CountryCode) and ID. and that was the target.

There are some limits to be aware of:

- maintenance of the bitmap index requires some extra code to be triggered. Not shown here.
- passing the Group parameter was a compromise and more sophisticated solutions may exist

## Advantages:

- speed up time critical queries on static data significantly
- existing data structure was not changed
- old code could continue to work without touching it
- bitmap aware queries are well separated from traditional queries
- it could be achieved just by additional code added with reasonable effort.

This is a coding example working on Caché 2018.1.3 and IRIS 2020.2  
It will not be kept in sync with new versions  
It is also **NOT** serviced by InterSystems **Support** !

[#Indexing](#) [#Performance](#) [#SQL](#) [#Caché](#) [#InterSystems](#) [IRIS](#)  
[Check the related application on InterSystems Open Exchange](#)

Source URL: <https://community.intersystems.com/post/adopted-bitmap>