Materialized Views

Article
Robert Cemper · Apr 26, 2020 5m read

Materialized Views

A VIEW in SQL is basically a prepared SQL statement.
It has to be executed and assembled like any other SQL query.
MATERIALIZED VIEW means that the content is collected before hands and can be retrieved rather fast.
I saw the concept first with my favorite competitor named O* and they made a lot of noise about it.

```
{ favorite: because I could win every benchmark against them  }
```

Depending on the individual construction of this materialized views updates and maintenance might be required.

In Caché / IRIS this exists almost since ever and we take this as normal and a given fact.
Most developers are just not aware of it and with a little bit of polishing it can be presented as an excellent feature.
In addition, any update and maintenance happen as a built-in functionality with no extra effort.

See this example:
In our famous **Sample.Person** class in namespace **SAMPLES** I have extended and rebuilt his index.

```
/// Define an index for <property>Name</property>.
Index NameIDX On Name [ Data = (Name, Home.State, SSN) ];
```

And being experienced and comfortable with the query generator you know that

```
SELECT ID, Name, Home_State, SSN   from Sample.Person
```

will run fast across just using the index global **^Sample.PersonI("NameIDX")** and never touch you data global.
That's basically the functionality of a Materialized View and update is implicit.

Defined as VIEW (from MgmtPortal as Studio isn't so handy) you get this class.

```
Class Sample.Person.NameView [ ClassType = view
                             , CompileAfter = Sample.Person
                             , DdlAllowed
                             ,  Not ProcedureBlock
```

```
                                , SqlTableName = NameView
                                , ViewQuery = { SELECT ID
                                                     , Name

, Home_State
                                                     , SSN

from Sample.Person  } ]
       { Parameter READONLY = 1; }
```

But if you want a little bit more comfort like a backlink to your data you can map the index global itself .
So you can apply implicit JOIN syntax and have a fully functional table as here
SELECT Name, BaseClass->DOB, HomeState, SSN,"%CLASSNAME",BaseClass FROM Sample_Person.NameIDX

Name	DOB	HomeState	SSN	%CLASSNAME	BaseClass
Bachman,George E.	06/12/1977	AK	232-30-3200		67
Bachman,Hannah A.	05/22/1955	UT	537-37-2290		83
Bachman,Laura C.	11/24/2014	IN	899-59-6879	~Sample.Employee~	151
Basile,Lawrence N.	06/15/2008	NC	400-29-2516	~Sample.Employee~	171
Basile,Mark C.	12/18/1966	CT	275-41-4866		18

and here's the class definition and you have to design it manually

```
/// mapped index
/// Index NameIDX On Name [ Data = (Name, Home.State, SSN)
];
Class Sample.Person.NameIDX Extends %Persistent [ Final ]
{
Property IndexName As %String [
InitialExpression = "NameIDX", ReadOnly ];
Property SQLUPPERname As %String [ ReadOnly ];
Property BaseClass As Sample.Person [ ReadOnly ];

Index min On (IndexName, SQLUPPERname, BaseClass) [ IdKey ];

/// Classname of Index Source
Property %CLASSNAME As %String [ ReadOnly ];
/// Person's name.
Property Name As %String [ ReadOnly ];
/// Person's home address. This uses an embedded object.
Property HomeState As %String [ ReadOnly ];
/// Person's Social Security number. This is validated using
```

```
pattern match.
Property SSN As %String(PATTERN = "3N1""-""2N1""-""4N")
[ ReadOnly ];

Parameter READONLY = 1;
Parameter MANAGEDEXTENT As INTEGER = 0;

Storage Default
{
 <Data name="NameIDXDefaultData">
  <Value name="1">
    <Value>%CLASSNAME</Value>
  </Value>
  <Value name="2">
    <Value>Name</Value>
  </Value>
  <Value name="3">
    <Value>HomeState</Value>
  </Value>
  <Value name="4">
    <Value>SSN</Value>
  </Value>
 </Data>
 <DataLocation>^Sample.PersonI</DataLocation>
 <DefaultData>NameIDXDefaultData</DefaultData>
 <IdLocation>^Sample.Person.NameIDXD</IdLocation>
 <IndexLocation>^Sample.Person.NameIDXI</IndexLocation>
 <StreamLocation>^Sample.Person.NameIDXS</StreamLocation>
 <Type>%Library.CacheStorage</Type>
} }
```

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 #ObjectScript #SQL #Caché #Ensemble #InterSystems IRIS

120 2 1 3 370

Log in or sign up to continue

Add reply

Source URL: https://community.intersystems.com/post/materialized-views