Article
Vitaliy Serdtsev · Jun 29, 2017 6m read

SQL index for array property elements

Sometimes, it comes in very handy (especially for the EAV model) to use array properties in a class and be able to qickly search by their elements: both the key and the value.

Let ' s take a look at a simple example:

```
Class User.eav Extends %Persistent
{

Index idx1 On attributes(ELEMENTS) [ Data = entity ];

Index idx2 On (attributes(KEYS), attributes(ELEMENTS)) [ Data = entity ];

Property entity;

Property attributes As array Of %String(SQLTABLENAME = "attributes") [
SqlFieldName = attr ];

/// d ##class(User.eav).RepopulateAll()
ClassMethod RepopulateAll()
{
  d ..%DeleteExtent()

  s name=$TR("Sibe^rian pi^ne ce^dar","^",$c(769))

  s obj=..%New()
  s obj.entity="Human"
  d obj.attributes.SetAt(22,"Age")
  d obj.attributes.SetAt(186,"Height")
  d obj.attributes.SetAt("Jack","Name")
  d obj.%Save()

  s obj=..%New()
  s obj.entity="Tree"
  d obj.attributes.SetAt(186,"Age")
  d obj.attributes.SetAt(22,"Height")
  d obj.attributes.SetAt("Pines","Family")
  d obj.attributes.SetAt(name,"Name")
  d obj.%Save()

  /*

  ;or

  &sql(insert into eav(entity) select 'Human' union select 'Tree')
  &sql(insert into attributes(eav,element_key,attr)
  select 1,'Age',22 union
  select 1,'Height',186 union
  select 1,'Name','Jack' union
```

```
  select 2,'Age',186 union
  select 2,'Height',22 union
  select 2,'Family','Pines' union
  select 2,'Name',:name)
  */

  d ..Reindex()
}

/// d ##class(User.eav).Reindex()
ClassMethod Reindex()
{
  d ..%BuildIndices(,1)

  d $system.SQL.TuneTable("SQLUser.eav",1)
  d $system.SQL.TuneTable("SQLUser.attributes",1)
  d $system.OBJ.Compile($classname(),"cu/multicompile=1")
}

}
```

After population

```
USER>d ##class(User.eav).RepopulateAll()
```

, the following data will appear in our tables:

ID	entity
1	Human
2	Tree

eav	ID	attr	elementkey
1	1¦¦Age	22	Age
1	1¦¦Name	Jack	Name
1	1¦¦Height	186	Height
2	2¦¦Age	186	Age
2	2¦¦Height	22	Height
2	2¦¦Name	Siberian pine cedar	Name
2	2¦¦Family	Pines	Family

A global with data:

```
USER>zw ^User.eavD
^User.eavD=2
^User.eavD(1)=$lb("","Human")
^User.eavD(1,"attributes","Age")=22
^User.eavD(1,"attributes","Height")=186
^User.eavD(1,"attributes","Name")="Jack"
^User.eavD(2)=$lb("","Tree")
^User.eavD(2,"attributes","Age")=186
^User.eavD(2,"attributes","Family")="Pines"
^User.eavD(2,"attributes","Height")=22
^User.eavD(2,"attributes","Name")="Sibe?rian pi?ne ce?dar"
```

A global with indexes:

```
USER>zw ^User.eavI
^User.eavI("idx1"," 186",1)=$lb("","Human")
^User.eavI("idx1"," 186",2)=$lb("","Tree")
^User.eavI("idx1"," 22",1)=$lb("","Human")
^User.eavI("idx1"," 22",2)=$lb("","Tree")
^User.eavI("idx1"," JACK",1)=$lb("","Human")
^User.eavI("idx1"," PINES",2)=$lb("","Tree")
^User.eavI("idx1"," SIBE?RIAN PI?NE CE?DAR",2)=$lb("","Tree")
^User.eavI("idx2","Age"," 186",2)=$lb("","Tree")
^User.eavI("idx2","Age"," 22",1)=$lb("","Human")
^User.eavI("idx2","Family"," PINES",2)=$lb("","Tree")
^User.eavI("idx2","Height"," 186",1)=$lb("","Human")
^User.eavI("idx2","Height"," 22",2)=$lb("","Tree")
^User.eavI("idx2","Name"," JACK",1)=$lb("","Human")
^User.eavI("idx2","Name"," SIBE?RIAN PI?NE CE?DAR",2)=$lb("","Tree")
```

Let ' s run the following query now:

entity
Human
Tree

The query runs, but uses full scanning and not our indexes. If we look at our tables in the SMP (System Management Portal), we won't find idx1 and idx2 there, although we know for sure that the data was generated.

This happens because the SQL engine " sees " only those indexes for array properties that are based exclusively on the fields of the subtable array and contain a key, i.e. propArray(KEY). Both of our indexes contain the " entity " field, which is missing in the " attributes " subtable.

You will also not see the **Index idx3 On attributes(ELEMENTS);**, since it doesn't contain attributes(KEYS), but the following indexes:

- Index idx4 On (attributes(KEYS), attributes(ELEMENTS));
- Index idx5 On (attributes(ELEMENTS), attributes(KEYS));

will be visible and will be taken into account in queries. However, they are not optimal for all types of queries.

So what is the most effortless method of unhiding indexes for the elements of an array property from the SQL engine?

Caché 2015.1 allows you to project a collection as a table field, if this collection projects into a subtable using the SetCollectionProjection/GetCollectionProjection methods.

This functionality is disabled by default.

Earlier versions of Caché do not have these methods, but you can try to enable this feature manually:

```
%SYS>s ^%SYS("sql","sys","collection projection")=1
```

After you make this change, make sure to recompile the classes.

So, let's turn this parameter on and see what it does.

We can now see our indexes in the SMP, and there is a hidden collection-field called “ attr ” in the “ eav ” table. However, our query still doesn't see the idx1/idx2 indexes.

To fix the situation, let's use the already familiar predicate FOR SOME %ELEMENT:

entity
Human
Tree

The idx1 index is now used in the query. Let's change it a bit:

entity
Human

entity
Tree

The last two examples use the idx2 index instead of idx1.

UPD: now the same can be done using SQLPROJECTION, namely:

```
Property attributes As array Of %String(SQLPROJECTION = "table/column",
SQLTABLENAME = "attributes") [ SqlFieldName = attr ];
```

This is a translation of the following article. Thanks [@Evgeny Shvarov] for the help in translation.

This post is also available on Habrahabr[ru].

Inspired by 17383689[ru].

Special thanks to [@Alexander Koblov] for the tip in the framework of WRC.

#Indexing #ObjectScript #SQL #Caché

Source URL:https://community.intersystems.com/post/sql-index-array-property-elements