Entity-attribute-value model in relational databases. Should globals be emulated on tables? Part 2.

Article
Sergey Kamenev · May 28, 2020 7m read

Open Exchange

Entity-attribute-value model in relational databases. Should globals be emulated on tables? Part 2.

A More Industrial-Looking Global Storage Scheme

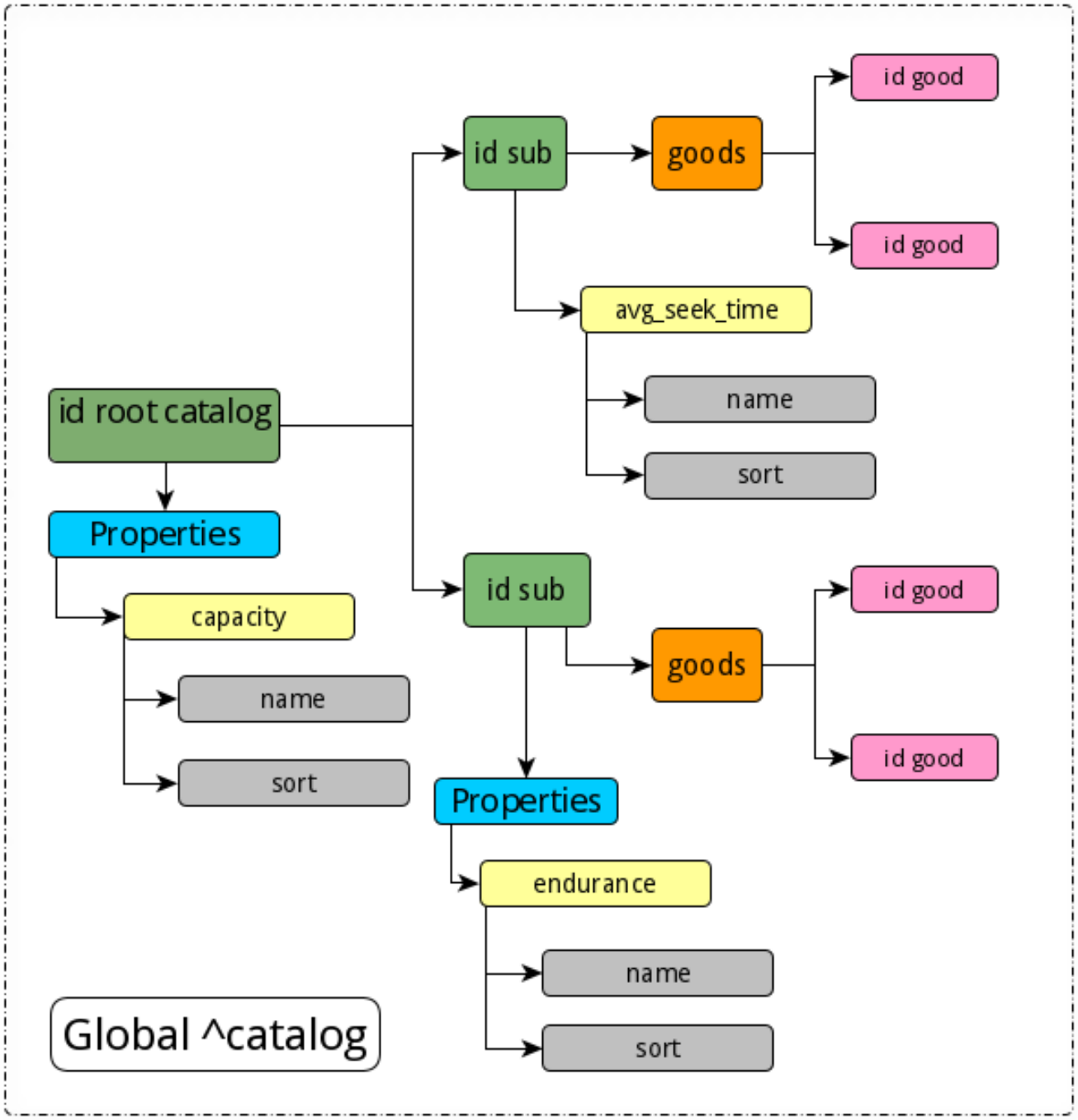
In the first article in this series, we looked at the entity–attribute–value (EAV) model in relational databases, and took a look at the pros and cons of storing those entities, attributes and values in tables. We learned that, despite the benefits of this approach in terms of flexibility, there are some real disadvantages, in particular a basic mismatch between the logical structure of the data and its physical storage, which causes various difficulties.

To solve these issues, we decided to see whether using globals — which are optimized for storing hierarchical information — for tasks the EAV approach typically handles would work.

In Part 1, we created a catalog for an online store, first using tables, then using just one global. Now, let's try to implement the same structure for several globals.

In the first global, ^catalog, we'll store the directory structure. In the second global, ^good, we'll store goods. And in the global ^index, we'll store indexes. Since our properties are tied to a hierarchical catalog, we won't create a separate global for them.

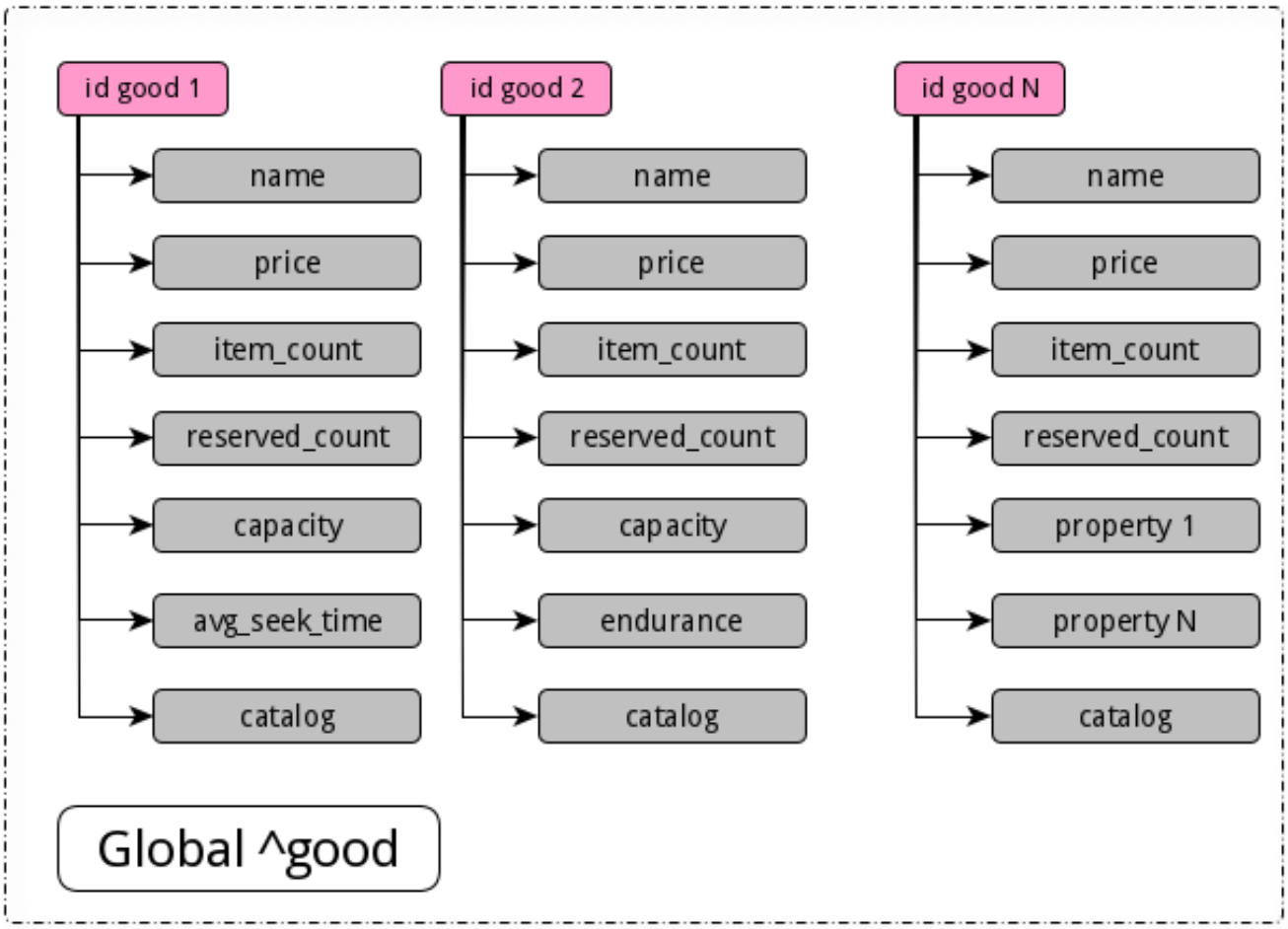
With this approach, for each entity (except for properties), we have a separate global, which is good from the point of view of logic. Here's the global catalog structure:




```
Set ^?atalog(root_id, "Properties", "capacity", "name") = "Capacity, GB"
Set ^?atalog(root_id, "Properties", "capacity", "sort") = 1

Set ^?atalog(root_id, sub1_id, "Properties", "endurance", "name") = "Endurance, TBW"
Set ^?atalog(root_id, sub1_id, "Properties", "endurance", "sort") = 2

Set ^?atalog(root_id, sub1_id, "goods", id_good1) = 1
Set ^?atalog(root_id, sub1_id, "goods", id_good2) = 1

Set ^?atalog(root_id, sub2_id, "Properties", "avg_seek_time", "name") = "Rotate speed
, ms"
Set ^?atalog(root_id, sub2_id, "Properties", "avg_seek_time", "sort") = 3
```

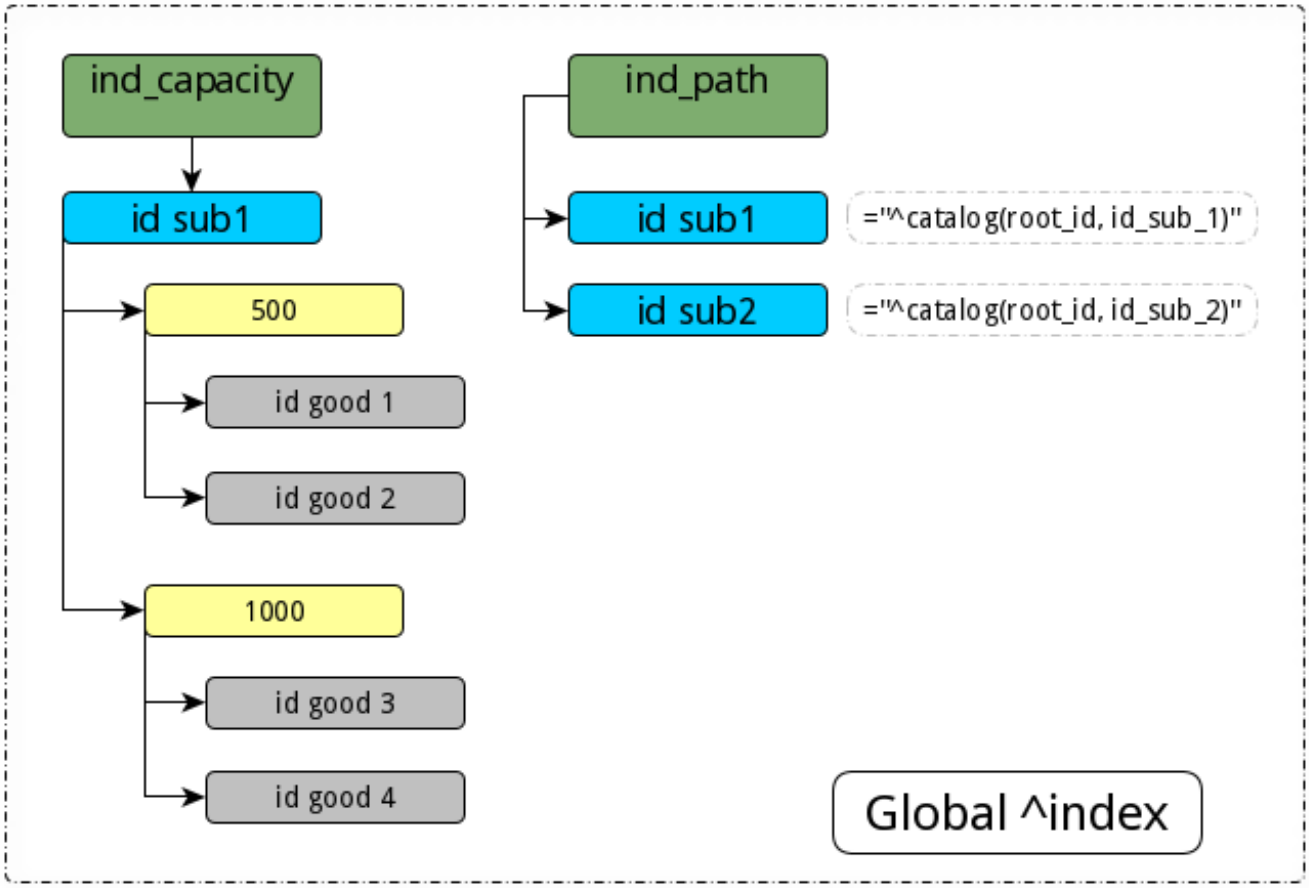
```
Set ^?atalog(root_id, sub2_id, "goods", id_good3) = 1
Set ^?atalog(root_id, sub2_id, "goods", id_good4) = 1
```

A global with goods will look something like this:




```
Set ^good(id_good, property1) = value1
Set ^good(id_good, property2) = value2
Set ^good(id_good, property3) = value3
Set ^good(id_good, "catalog") = catalog_id
```

Of course, we need indexes so that for any section of the catalog with goods, we can sort by the properties we need. The index global will have a structure something like this:




```
Set ^index(id_catalog, property1, id_good) = 1
; To quickly get the full path to concrete sub-catalog
Set ^index("path", id_catalog) = "^catalog(root_id, sub1_id)"
```

Thus, in any section of the catalog, we can get a sorted list. An index global is optional. It's useful only if the number of products in this section of the catalog is large.

ObjectScript Code for Working with Demo Data

Now we'll use ObjectScript to work with our data. To start, let's get the properties of a specific good. We have the ID of a particular good and we need to display its properties in the order given by the sort value. Here's the code for that:

```
get_sorted_properties(path, boolTable)
{
  ; remember all the properties in the temporary global
  While $QLENGTH(@path) > 0 {
    if ($DATA(@path("Properties"))) {
      set ln=""
      for {
     Set ln = $order(@path("Properties", ln))
     Quit: ln = ""

        IF boolTable & @path("Properties", ln, "table_view") = 1 {
```

```
         Set ^tmp(@path("Properties", ln, "sort"), ln) = @path("Properties", ln, "nam
e")
     }
   ELSE {
       Set ^tmp(@path("Properties", ln, "sort"), ln) = @path("Properties", ln, "name"
)
   }
    }
  }
}

print_sorted_properties_of_good(id_good)
{
  Set id_catalog = ^good(id_good, "catalog")
  Set path = ^index("path", id_catalog)

  Do get_sorted_properties(path, 0)

  set ln =""
  for {
   Set ln = $order(^tmp(ln))
   Quit: ln = ""
   Set fn = ""
   for {
  Set fn = $order(^tmp(ln, fn))
  Quit: fn = ""
  Write ^tmp(ln, fn), " ", ^good(id_good, fn),!
   }
  }
}
```

Next, we want to get products from the catalog section in the form of a table, based on id_catalog:

```
print_goods_table_of_catalog(id_catalog)
{
  Set path = ^index("path", id_catalog)
  Do get_sorted_properties(path, 1)

  set id=""
  for {
    Set id = $order(@path("goods"), id)
    Quit: id = ""

    Write id," ", ^good(id, "price"), " "

    set ln =""
    for {
      Set ln = $order(^tmp(ln))
      Quit: ln = ""
      Set fn = ""
      for {
      Set fn = $order(^tmp(ln, fn))
      Quit: fn = ""
```

```
      Write ^tmp(ln, fn), " ", ^good(id, fn)
      }
      Write !
    }
  }
}
```

Readability: EAV SQL Versus Globals

Now let’s compare the use of EAV and SQL against using globals. With regard to code clarity, it’s clear that this is a subjective parameter. But let's look, for example, at creating a new product.

We’ll start with the EAV approach, using SQL. First, we need to get a list of object properties. This is a separate task and quite time-consuming. Assume we already know the IDs of these three properties: capacity, weight, and endurance.

```
START TRANSACTION
INSERT INTO good (name, price, item_count, catalog_id) VALUES ('F320 3.2TB AIC SSD',
700, 10, 15);

SET @last_id = LAST_INSERT_ID ();

INSERT INTO NumberValues ??Values??(@last_id, @id_capacity, 3200);
INSERT INTO NumberValues ??Values??(@last_id, @id_weight, 0.4);
INSERT INTO NumberValues ??Values??(@last_id, @id_endurance, 29000);
COMMIT
```

In this example, we have only three properties, and therefore the example doesn’t look so scary. In the general case, we’d still have a few inserts into the text table inside the transaction:

```
INSERT INTO TextValues ??Values??(@last_id, @ id_text_prop1, 'Text value of property
1');
INSERT INTO TextValues ??Values??(@last_id, @ id_text_prop2, 'Text value of property
2');
...
INSERT INTO TextValues Values (@last_id, @id_text_propN, 'Text value of property N');
```

Of course, we could simplify the SQL version a little if we used text notation instead of ID properties, such as “capacity” instead of a number. But in the SQL world, this isn’t acceptable. It’s customary instead to use a numeric ID to enumerate entity instances. This results in faster indexes (you need to index fewer bytes), it’s easier to track uniqueness, and it’s easier to automatically create a new ID. In this case, the insert fragment would look like this:

```
INSERT INTO NumberValues ??Values??(@last_id, 'capacity', 3200);
INSERT INTO NumberValues ??Values??(@last_id, 'weight', 0.4);
```

```
INSERT INTO NumberValues ??Values??(@last_id, 'endurance', 29000);
```

Here's the same example using globals:

```
TSTART
Set ^good(id, "name") = "F320 3.2TB AIC SSD"
Set ^("price") = 700, ^("item_count") = 10, ^("reserved_count") = 0, ^("catalog") = i
d_catalog
Set ^("capacity") = 3200, ^("weight") = 0.4, ^("endurance") = 29000
TCOMMIT
```

Now let's delete a good using the EAV approach:

```
START TRANSACTION
DELETE FROM good WHERE id = @ good_id;
DELETE FROM NumberValues ??WHERE good_id = @ good_id;
DELETE FROM TextValues ??WHERE good_id = @ good_id;
COMMIT
```

And then do the same with globals:

```
Kill ^good(id_good)
```

We can also compare the two approaches in terms of code length. As you can see from the previous examples, when you use globals, the code is shorter. This is good. The shorter the code, the fewer the errors and the easier it is to understand and maintain.

Generally, shorter code is also faster. And, in this case, that's certainly true, since globals are a lower-level data structure than relational tables.

Scaling Data with EAV and Globals

Next, let's look at horizontal scaling. With the EAV approach we have to at least distribute the three largest tables to the servers: Good, NumberValues,??and TextValues. Tables with entities and attributes can simply be completely copied to all servers, since they have little information.

On each server, with horizontal scaling, different products would be stored in the Good, NumberValues, ??and TextValues ??tables. We'd have to allocate certain ID blocks for products on each server so that there's no duplication of IDs for different products.

For horizontal scaling with globals, we'd have to configure ID ranges in the global and assign a global range to each server.

The complexity is approximately the same for EAV and for globals, except that for the EAV approach we'd have to configure ID ranges for three tables. With globals, we'd configure IDs for just one global. That is, it's easier to organize horizontal scaling for globals.

Data Loss with EAV and Globals

Finally, let's consider the risk of data loss due to corrupted database files. Where is it easier to save all the data: in five tables or in three globals (including an index global)?

I think it's easier in three globals. With the EAV approach, the data for different goods is mixed in tables, while for globals the information is stored more holistically. The underlying branches are stored and sorted sequentially. Therefore, corruption of part of the global is less likely to lead to damage than corruption of any of the tables in the EAV approach, where data is stored like intertwined pasta.

Another headache in data recovery is the display of information. With the EAV approach, information is divided among several tables and special scripts are required to assemble it into a single whole. In the case of globals, you can simply use the ZWRITE command to display all the values and the underlying branches of the node.

InterSystems IRIS Globals: A Better Approach?

The EAV approach has emerged as a trick for storing hierarchical data. Tables weren't originally designed to store nested data. The de facto EAV approach is the emulation of globals in tables. Given that tables are a higher-level and slower data storage structure than globals, the EAV approach fails in comparison with globals.

In my opinion, for hierarchical data structures, globals are more convenient and more comprehensible in terms of programming, and they're faster.

If you've been planning an EAV approach for your project, I suggest you consider using InterSystem IRIS globals to store hierarchical data.

#Databases #Globals #Performance #Relational Tables #SQL #Tips & Tricks #Unstructured Data #Caché #InterSystems IRIS #InterSystems IRIS for Health
Check the related application on InterSystems Open Exchange

Related posts

- Entity-attribute-value model in relational databases. Should globals be emulated on tables? Part 1.
- Entity-attribute-value model in relational databases. Should globals be emulated on tables? Part 2.