Article
Sergey Kamenev · May 30, 2017 6m read

Globals Are Magic Swords For Managing Data. Part 1

Globals, these magic swords for storing data, have been around for a while, but not many people can use them efficiently or know about this super-weapon altogether.

If you use globals for tasks where they truly shine, the results may be amazing, either in terms of increased performance or dramatic simplification of the overall solution (1, 2).

Globals offer a special way of storing and processing data, which is completely different from SQL tables. They were first introduced in 1966 in the M(UMPS) programming language, which was initially used in medical databases. It is still used in the same way, but has also been adopted by some other industries where reliability and high performance are top priorities: finance, trading, etc.

Later M(UMPS) evolved into Caché ObjectScript (COS). COS was developed by InterSystems as a superset of M.

The original language is still accepted by developers' community and alive in a few implementations. There are several signs of activity around the web: MUMPS Google group, Mumps User's group), effective ISO Standard, etc.

Modern global based DBMS supports transactions, journaling, replication, partitioning. It means that they can be used for building modern, reliable and fast distributed systems.

Globals do not restrict you to the boundaries of the relational model. They give you the freedom of creating data structures optimized for particular tasks. For many applications reasonable use of globals can be a real silver bullet offering speeds that developers of conventional relational applications can only dream of.

Globals as a method of storing data can be used in many modern programming languages, both high- and low-level. Therefore, this article will focus specifically on globals and not the language they once came from.

2. How globals work

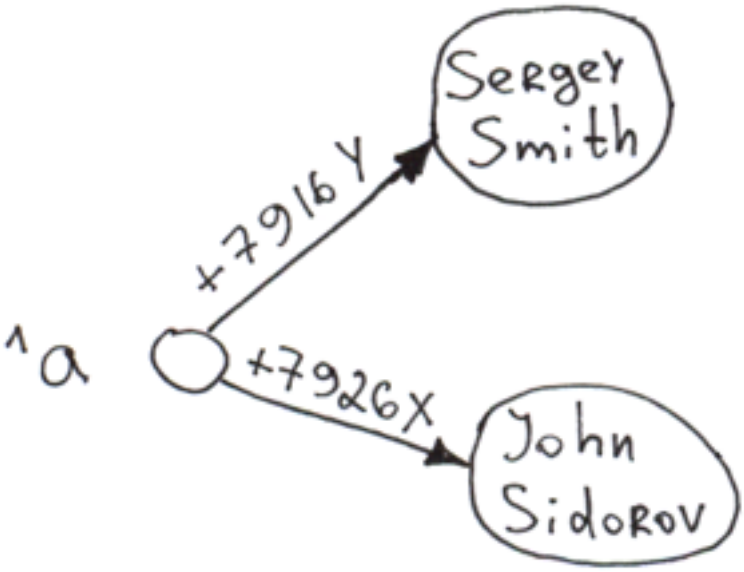
Let's first understand how do globals work and what are their advantages. Globals can be viewed from different perspectives. In this part of the article we will look at them as trees or as hierarchical data storages.

Simply put, a global is a persistent array. An array that is automatically saved to the disk.

It's hard to imagine anything simpler for storing data. In the program code (written in the COS/M language), the only difference from a regular associative array is the **^** symbol that stands before their names.

You don't need to know SQL to save data to a global since all the necessary commands are really easy and can be learned in an hour.

Let's start with the simplest example, a single-level tree with two branches. Examples are written in COS.




```
Set ^a("+7926X") = "John Sidorov"
Set ^a("+7916Y") = "Sergey Smith"
```

When data is inserted into a global (the Set command), 3 things happen automatically:

1. **Saving** data to the disk.
2. **Indexing**. What's in the brackets is a subscript, what's to the right of the equal sign is a node value.
3. **Sorting**. Data is sorted by a key. The next traversal will put « Sergey Smith » into the first position followed by « John Sidorov » . When obtaining a list of users from a global, the database does not spend time on sorting. You can actually request a sorted list starting from any key, even a non-existing one (output will start from the first real key following this one).

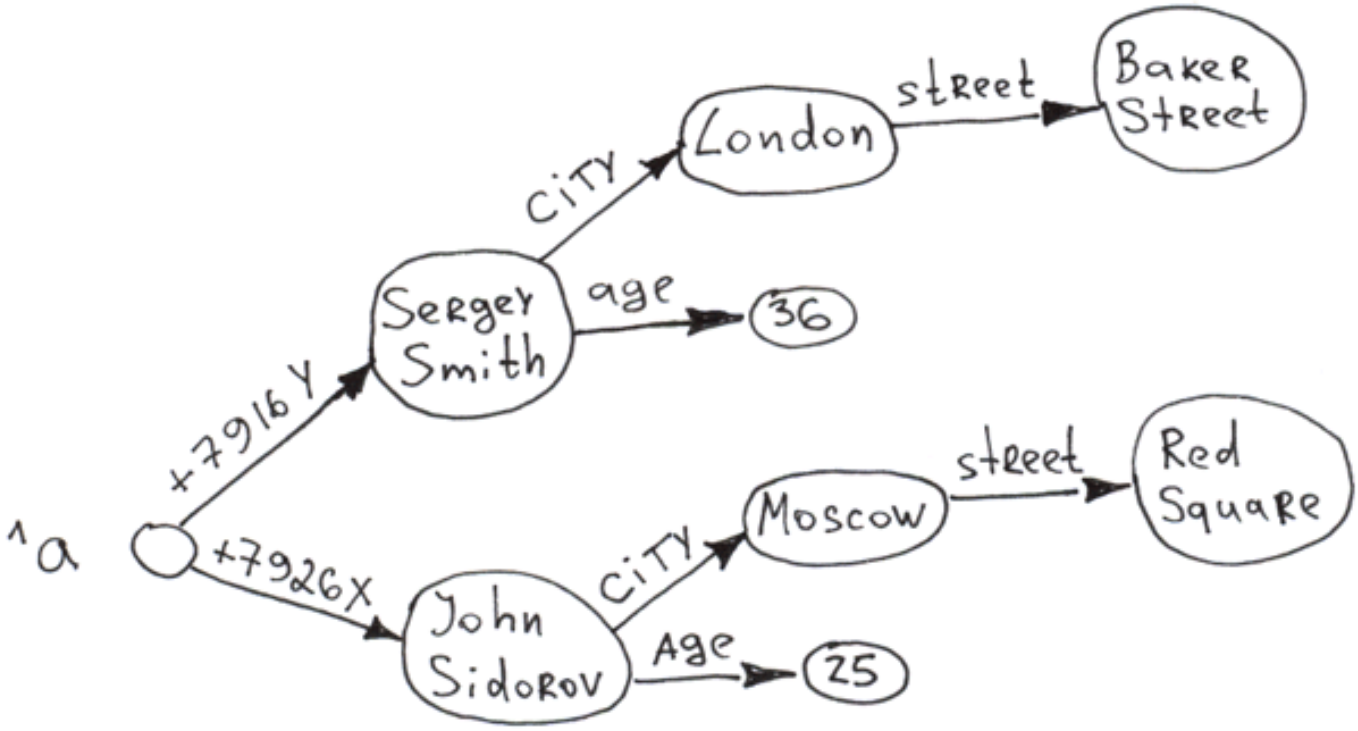
All of these operations are performed at an incredible speed. On my home system (i5-3340, 16GB, HDD WD 1TB Blue), I managed to reach 1,050,000 inserts/sec in a single process. On multi-core systems, speeds can reach dozens of millions of inserts/sec.

Of course, record insertion speed itself doesn't say much. We can, for instance, write data to text files – that's how processing works at Visa, according to rumors. However, with globals, we get a structured and indexed storage that you can work with enjoying its high speed and ease of use.

- **The biggest strength of globals is the speed of inserting new nodes into them.**
- **Data is always indexed in a global. One-level and in-depth tree traversals are always very fast.**

Let's add a few second- and third-level branches to the global.

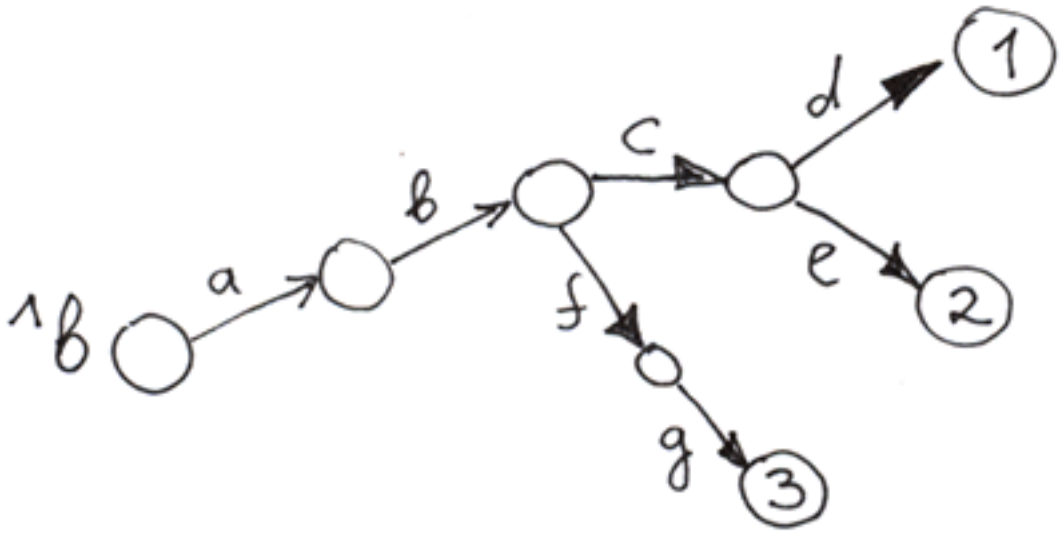
```
Set ^a("+7926X", "city") = "Moscow"
Set ^a("+7926X", "city", "street") = "Req Square"
Set ^a("+7926X", "age") = 25
Set ^a("+7916Y", "city") = "London"
Set ^a("+7916Y", "city", "street") = "Baker Street"
Set ^a("+7916Y", "age") = 36
```



Apparently, you can build multi-level trees using globals. Access to any node is nearly instantaneous thanks to auto-indexing after every insert. Tree branches on any level are sorted by a key.

As you can see data can be stored in keys and values both. The combined length of a key (the sum of lengths of all indexes) can reach 511 bytes, and values can be up to 3.6 MB in size in Caché. The number of levels in a tree (number of dimensions) is capped at 31.

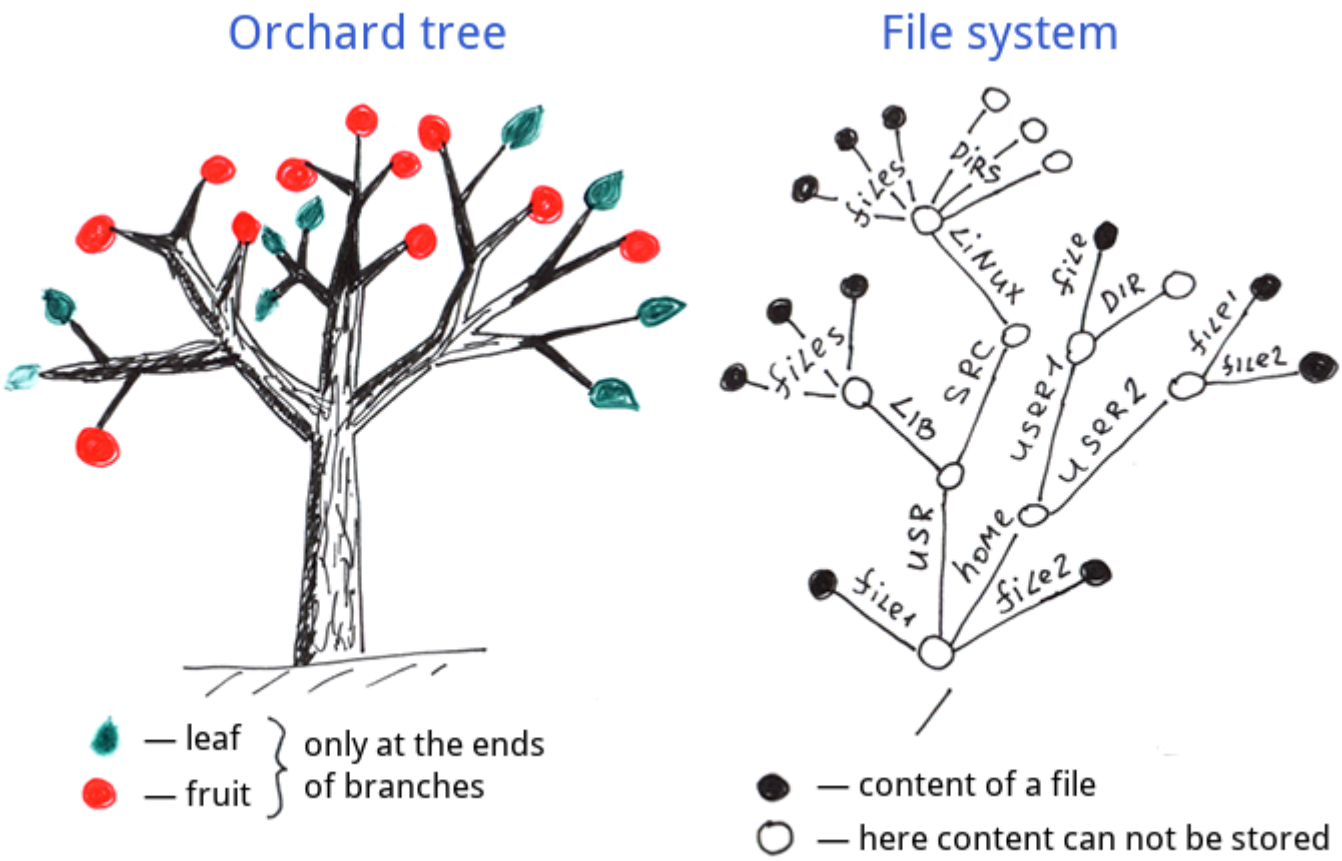
One more cool thing: you can build a tree without defining the values of top-level nodes.




```
Set ^b("a", "b", "c", "d") = 1
Set ^b("a", "b", "c", "e") = 2
Set ^b("a", "b", "f", "g") = 3
```

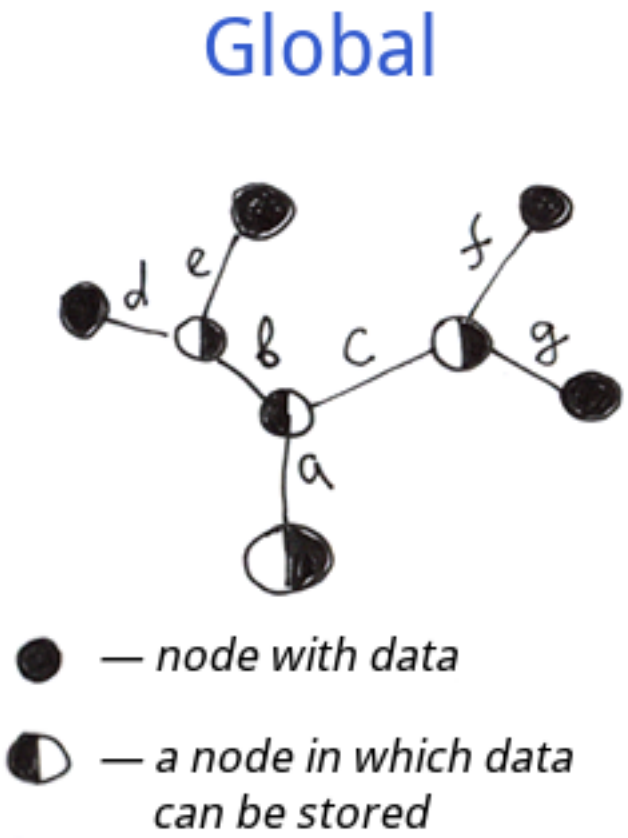
Empty circles are nodes without a value.
In order to better understand globals, let's compare them with other trees: garden trees and file system name trees.

Let's compare globals with the most familiar hierarchical structures: regular trees that grow in gardens and fields, as well as file systems.



As we can see leaves and fruits grow only on the ends of branches on regular trees.
File systems – information is also stored on the end of branches also known as full file names.

And here is the data structure of a global.



Differences:

1. **Internal nodes:** information in a global can be stored in every node, not on branch ends only.
2. **External nodes:** globals must have defined branch ends (ends with values) which is not mandatory for file system and garden trees.

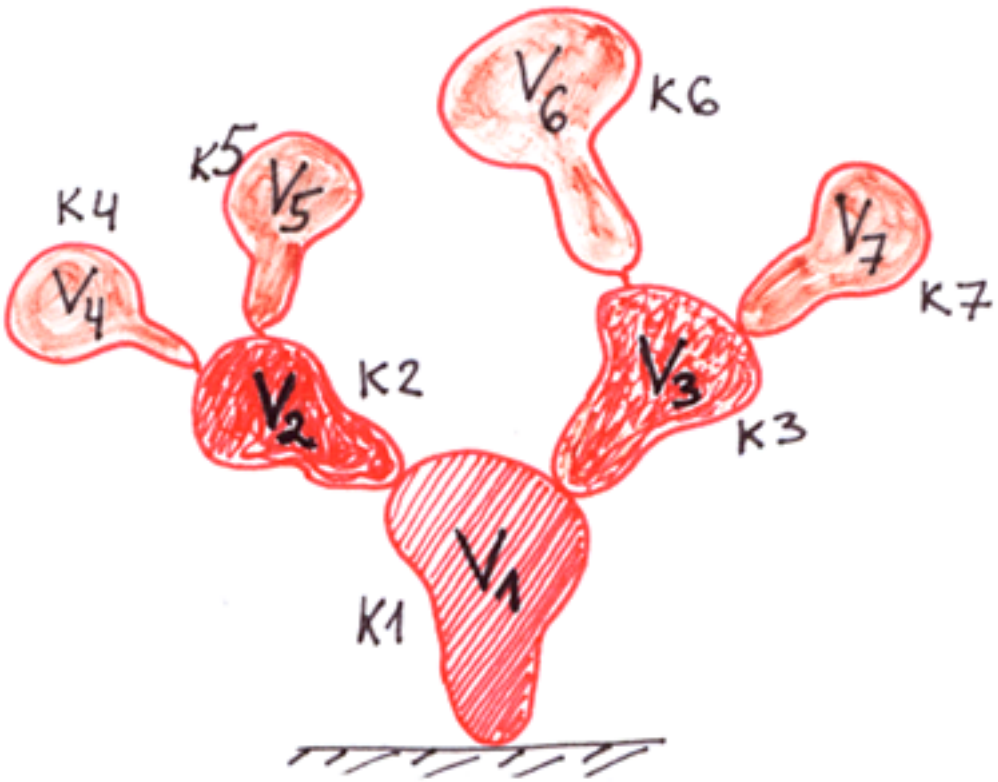
Regarding internal nodes we can treat the global's structure as a superset of file systems' name trees and garden trees structures. So global's structure is a more flexible one.

In general, a global is a **structured tree that supports data saving in each node.**

In order to better understand how do globals work let's imagine what would happen if the creators of a file systems used an approach identical to globals for storing information?

1. If the last file in a folder was deleted it would also delete the folder itself as well as all higher-level folders which contained just this deleted folder.

2. There would be no need in folders at all. There would be files with subfiles and files without subfiles. If you compare it with a regular tree each branch would become a fruit.



If the orchard tree was a global ...

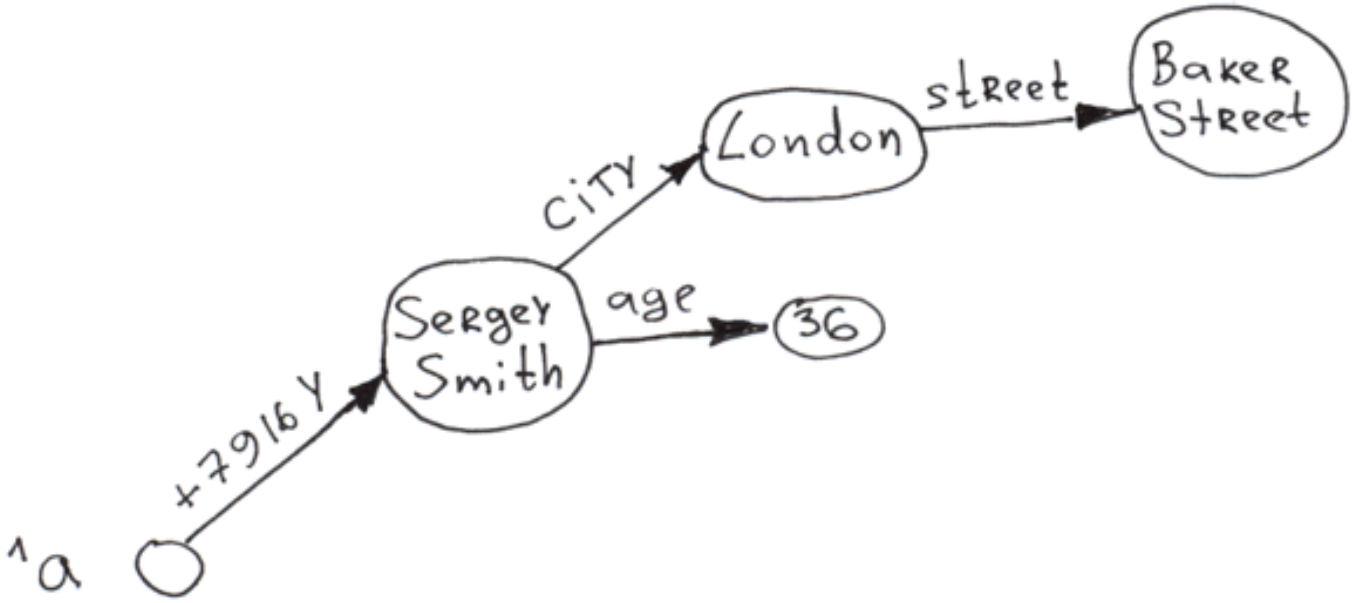
3. Things like README.txt would probably be no longer necessary. All you'd need to say about a folder's content could be written to the folder file itself. Generally filenames are indistinguishable from a folder names(e.g. /etc/readme can be either folder of file) that means that we would be fine operating files only.

4. Folders with subfolders and files could be deleted much faster. There have been articles on the net telling stories of how time-consuming and difficult it is to delete millions of small files (1, 2, 3). However, if you create a pseudo-file system based on a global, it will take seconds or even fractions of a second. When I was testing the removal of subtrees on my home PC I managed to delete 96-341 million nodes from a two-level tree on an HDD (not an SDD). And it worth to mention that we are talking about removing a part of a global tree, not the removing entire file containing a global.

Sub-trees removal is yet another advantage of globals: you don't need recursion for this. It's incredibly fast.

In our tree this could be done with one **Kill** command.

```
Kill ^a("+7926X")
```



Below is a small table with that can give a better understanding of the actions that you can perform on a global.

Key commands and functions related to globals in COS	
Set	Setting (initializing) branches up to a node (if undefined) and node value
Merge	Copying of a subtree
Kill	Removal of a subtree
ZKill	Deletion of the value of a particular node. The subtree stemming from the node is not affected
$Query	Full in-depth traversal of the tree
$Order	Returns the next subscript on the same level
$Data	Checking if a node is defined
$Increment	Atomic incrementation of a node's value in order to avoid reading and writing for ACID. The latest recommendation is to use $Sequence instead.

Thank you for your attention I'll be glad to answer your questions.

Disclaimer: *This article reflects author's private opinion and have no relation to the official position of InterSystems.*

Continuation of "Globals Are Magic Swords For Storing Data. Trees. Part 2". *You will learn what types of data can be displayed in globals and where they work best.*

#Beginner #Globals #Node.js #Performance #Relational Tables #Caché #InterSystems IRIS

Source URL: https://community.intersystems.com/post/globals-are-magic-swords-managing-data-part-1