Lecture 06 - Global Variables Considered Harmful

In the world of computer programming the term “considered harmful” really relates to a letter to the ACM in the late 1960’s referring to “GOTO’s Considered Harmful”. This letter by Dr Edward Dijkstra, one of the worlds top computer language researchers set in motion the development of block structured languages and a host of other developments that we take for granted today.

An examle of a GOTO being bad is a chunk of code written in Fortran 1966 that I worked on. IT was 3621 lines of code with 18 subroutines - all stuck together with computed GOTos.

```
          if(j)15,22,331
          if(i.ne.0)331,22,18
          ...

  15      if(b.gt.4096)51,22,211
```

Code was a nightmare to debug!

I have talked to a number of language theorists and they indite that global variables are just as bad as GOTOs.

Most people avoid these globals like the plage! That is a good thing.

Then there was the database….

So we store globals in a database. This means that the global state data is not just global during the run of the program but is global across multiple runs of the program. Not just multiple runs but global across years and many program in different languages and it is still there.

We can have 1960s’ Cobol in a bank accessing the same set of data in the data warehouse as a new web application that is running on an iPhone!

Often is is “state” data. An example is login data.

Now we collect the data and model the data and backup the data and love the data. It is still global-state-data. In huge volumes.

Volume turns out to be a problem. SQL performance is determined by how much data there is. A full table scan with 1000 rows in a test database will not perform the same way on 100,000,000 rows in

production.

So testing requires production scale data. But production scale data means making expensive copies of data. If you have a $100,000.00 system for production and 3 development groups that want to use the same data… $300k for development hardware?

How do you get a production copy? Backup and restore of an entire database can take time.

What about Security - do the developers get access to all the customers private data? What about HIPPA? *Example*.

What about business continuity -and- $^{24}/_{7}$ availability. What are the performance impacts.

Having a regular backup and restore practice in a database is critical to survivability. If you don't restore a backup at least twice a year you have no-clue if it still works. *Example*.

Some Solutions - BackupMan - for PostgreSQL - is the solution for backups. Easy to setup and maintain. Full database clones via btrfs - at the hardware level.

Avoid: Docker - like the plage. *Example*.

Avoid: running any significant database on Windows.

Hosted solutions (AWS) are expensive for what you get. *Example*.

Do understand the costs and triad offs. The higher the level of availability the more it is going to cost. A system that can be down for 5 minutes a year costs a lot less than one that has to be running 24x7 a year. *Example*.