Oct 6, 2020

***The project is progressing fairly well.*** The prototype is working. It is difficult to use and clumsy but working and actually working well sometimes.

The theme is to take the data out of one database, summarize it and put the summary into another database. The source database has 7000 to 10000 lines per day. The summary is by day. The source database, the 7000 per day, is a flat file. It is really just “UNIX datetime stamp” and “number”. The number is a mixture of different fields and the fields are distinctive enough that they can be separated by an algorithm.

We are used to going to the site and, with the mouse, making decisions. Just click on what you want. Then click “go” and the software will take you where you want to be. (this is a very global image; bear with me). The Python is like that, but the decisions must be put into the Python script manually.

* Database name
* Starting date
* Targets … There are 1 to six targets

Also adjustable are these parameters. I mostly just leave them alone.

* Window for the algorithm (example; we want 1834000 + or – [ 1200] where 1200 is the window)
* Search number … The algorithm goes day by day and then picks the number of values for that day before going on to the next day. I mostly use a ‘search number’ of 5000.
* The daily search starts at a time of the day. I use 02:00:00, but this could be anything.
* Within the databases the tables need to be specified. The data is always in the same table, first\_table. I mostly do not have to worry about them.

The big summary database has 14 columns and is pretty cool. Data, (six columns), python script, number of rows summarized for that day, Then number per field for each date within that row (six columns). Finally, the contributing database name.

I like a window of 1200 for the pendulums and 140000 for the rotating pendulums. I like a search number of 5000 and a start time of 02:00:00. The search number should show up in column\_eight. As it goes below the search number, the window looks too small. 90% should be about right. The program looks at 5000 rows a day and column\_eight should show something less than 5000.

I have about 30 databases. I have not seen anything of any interest. How to deal with this mess? These are really two experiments;

***Is gravity constant:*** There are four mechanical pendulums. They have a period of about 1.83 millionths of a second or about 1.8 seconds. Three of the pendulums are “pendulum drives”. These are electronic and run on “D” batteries. The fourth pendulum is a regular grandfather clock.

***Is mass constant:*** The handle is through rotational momentum or rotational inertia. I have two rotational things; One is electronic and the other is a regular “400-day anniversary clock”. The electronic versions are giving me garbage and the brass anniversary clock shows promise. I am in the process of acquiring another anniversary clock.

The gravity experiment has been running for 3 years. The mass experiment is much newer, having been started in February 8, 2020 and the brass anniversary clock was started in July 2020.

These are completely separate questions and should not be mixed together. The reason for this structure is easy: the same platform works for both.

***The Python/sqlite3 platform has some code that may be of broad interest.***

Datetime Python and sqlite3. Sqlite3 asks the developer to define the column fields. One of the definitions is DATE. I use Python to add the datetime and then retrieve it as a datetime. But, for whatever reason, the python always returns a string from the datetime (this is column\_one). The algorithm wants to move from day to day and it is a trick. The datetime object accepts the string as a datetime, because it is perfect. So: take the string, make it into a datetime, increment it by one day and turn it back into a string. It takes two lines and is pretty cool.

But this weird: the dattime string can be used for comparison. Effectively this works {is this string mathematically greater than that string}

The sqlite3 columns are REAL, but the come back as a FLOAT.

I have been passing the results into an entirely different database. This requires an object for each database. But, with carefully attention to details it is no great trick.

For each day (about 8000 rows):

* And number is read and put in a category based upon its size.
* Within the category the counter is incremented and the value is added to the sum.

At the end of the day, the average is calculated and row is added to the summary database:

* Averages for the six fields
* Number of total rows read (this should approach the search \_number)
* Python code name
* Number of rows used for each of the six fields
* Name of the contributing database

At the end of the day target is moved to the new average. So, if the averages are wandering around the algorithm will follow it. Processing the next database involves starting the program again and it is good to reset the targets, but after that it is automatically reset.

And at the end of the day, the day is incremented by one and the whole starts again.

The first databases were sorted into the database. This was changed for unimportant reasons to just put all data into one column. But the older databases required, essentially making the data bac into one flat file and then sorted. Hence, the {for I range (1,5) stuff. And there is a bit of code to ensure that no strings are passed into a comparison.

The code works well.