06-window-functions-sqlite

June 3, 2024

1 Introduction

It's all about out-of-core processing with SQLite, especially with use of window functions.

As usual, the exercises require that you load a dataset and process it as required, saving the indicated files.

The data is provided as a CSV file with a header, representing a time series of traffic intensity measurements taken by particular detector in paricular time period, in the following form:

```
detector_id,shortname,name,starttime,endtime,count
1,CAUTC11FD101_D1,\N,2015-06-22 23:58:31,2015-06-23 00:00:01,1
1,CAUTC11FD101_D1,\N,2015-06-23 00:00:01,2015-06-23 00:01:31,1
1,CAUTC11FD101_D1,\N,2015-06-23 00:01:31,2015-06-23 00:03:01,0
1,CAUTC11FD101_D1,\N,2015-06-23 00:03:01,2015-06-23 00:04:31,3
```

The columns are as follows: the first one is a unique detector identifier, the second one is its short name, then a regular name (whatever it means), a start time of the measurement, an end time of the measurement, and a measurement itself which is a number of vehicles that passed the detector in the given time period from the start to the end.

The data is available at: http://awing.kis.agh.edu.pl:8080/detectors_names_traffic_s_small.csv.bz2 and it is Bzip2 compressed being roughly 179MB, or over 3GB uncompressed. The machine is in the AGH University network, so you need to use the VPN to access it from outside.

Program in your python script fetching, and importing the data into local SQLite database stored in `readings.sqlite` file. If you use Linux/MacOSX/BSD/Unix shell it could look like:

```
mkfifo fifo
curl awing.kis.agh.edu.pl:8080/detectors_names_traffic_s_small.csv.bz2 \
                                                        | bunzip2 > fifo
```

and in a separate terminal you could run the SQLite shell, which creates a table and imports the data from the named pipe (`fifo`):

```
echo "CREATE TABLE readings (detector_id INTEGER, shortname TEXT,\
                             name TEXT, starttime TIMESTAMP,\
                             endtime TIMESTAMP, \
                             count INTEGER);" | sqlite3 proj6_readings.sqlite
echo ".import --skip 1 --csv fifo readings" | sqlite3 proj6_readings.sqlite
```

After that you should have a ready to use SQLite database in `proj6_readings.sqlite` file.

Aternativelly, insead of using a named pipe (`mkfifo`) you can just download the file, uncompress, and import it – it would just take more disk space (mind that it is extra 3GB!).

2 Exercises

The scripts should expect the file `proj6_readings.sqlite`.

All query results should be **imported into DataFrames** and pickled into files as indicated.

First, check if you have all the records loaded, in. Just count them, like that:

```
con = sqlite3.connect("proj6_readings.sqlite")
cur = con.cursor()

result = cur.execute("SELECT count(*) from readings;").fetchall()

df = pd.DataFrame(result)

df
```

```
        0
0  147666
```

You can also do this directly from Pandas:

```
df = pd.read_sql("SELECT count(*) from readings;", con)
df
```

```
   count(*)
0    147666
```

For performance reasons, you might also want to index the database (it's not mandatory):

```
cur.execute("""
    CREATE INDEX detector_id ON readings (detector_id);
""").fetchall()
cur.execute("""
    CREATE INDEX starttime ON readings (starttime);
""").fetchall()
```

BTW, you may use the above code as a boilerplate to run SQL against SQLite.

2.1 Exercise 1: Basic counting

2 points

Find out how many distinct detectors there are. Each detector has a unique indentifier in the `detector_id` column. Pickle the result as a 1×1 DataFrame `proj6_ex01_detector_no.pkl`.

2.2 Exercise 2: Some stats for the detectors

4 points

Make sure that the data we have covers proper time span. Calculate minimum and maximum timestamps for each of the detectors, as well as how many measurements have been taken (a valid measurement is a not null value of `count`). Pickle it into `proj6_ex02_detector_stat.pkl` as a DataFrame with the following columns:

- detector ID,
- measurement count,
- minimum start time,
- maximum start time.

As you should be able to see, some of the detectors provide data for shorter time periods than the others.

The first three rows should look like:

detector_id	count(count)	min(starttime)	max(starttime)
1	357214	2015-06-22 23:58:31	2016-07-12 23:57:01
2	357201	2015-06-22 23:58:31	2016-07-12 23:57:01
3	356755	2015-06-22 23:58:31	2016-07-12 23:57:01

You can use any column names, but the column order should be as presented in the example.

2.3 Exercise 3: Moving Window

5 points

Calculate how much traffic intensity has changed compared to a previous measurement for detector 146. Use Window Functions of SQLite. Get just first 500 results - use SQL for this. Pickle it into `proj6_ex03_detector_146_lag.pkl` as a DataFrame with the following columns:

- detector ID,
- count,
- previous count.

The previous count is a value that was recorded by the same detector previously according to the `starttime` timestamp.

Mind that there is no previous value for the very first reading for a detector!

The first three records should look like:

detector_id	count	prev_count
146	1	NaN
146	0	1.0
146	0	0.0

You can use any column names, but the column order should be as presented in the example.

2.4 Exercise 4: Window

6 points

Calculate accumulated traffic intensity in a 10 record moving window (since a measurement is taken once in 1.5 minutes, the 10 record window would mean 15 minutes interval) for the detector 146. It is a sum that is calculated over all values of `count` over next 10 records at each record.

Pickle first 500 values into `proj6_ex04_detector_146_sum.pkl` as DataFrame with the following columns:

- detector id,
- count,
- window sum.

The first three records should look like:

detector_id	count	window_sum
146	1	2
146	0	2
146	0	2

You can use any column names, but the column order should be as presented in the example.

3 Submit your solution

As usual, commit your program to your GitLab project repository. Save it as `project06/project06.py`.