CSE 512 Assignment 1

Maximum points Possible – 10

The required task is to simulate data partitioning approaches on-top of an open source relational database management system (i.e., PostgreSQL) and build a simplified query processor that access data from the generated partitions. Each student must generate a set of Python functions that load the input data into a relational table, partition the table using different horizontal fragmentation approaches, insert new tuples into the right fragment, and perform query on various fragments.

Input Data: The input data is a Movie Rating data set collected from the MovieLens web site (http://movielens.org). The raw data is available in the file ratings.dat.

The rating.dat file contains 10 million ratings and 100,000 tag applications applied to 10,000 movies by 72,000 users. Each line of this file represents one rating of one movie by one user, and has the following format:

UserID::MovieID::Rating::Timestamp

Ratings are made on a 5-star scale, with half-star increments. Timestamps represent seconds since midnight Coordinated Universal Time (UTC) of January 1, 1970. A sample of the file contents is given below:

1::122::5::838985046

1::185::5::838983525

1::231::5::838983392

Required Task: Below are the steps you need to follow to fulfill this assignment requirements:

1. Install PostgreSQL.

2. Install Python3.x if it is not installed.

3. Install module psycopg2 for python3.x

4. Download rating.dat file from the MovieLens website: http://files.grouplens.org/datasets/movielens/ml-10m.zip

You can use partial data for testing.

5. Implement a Python function loadRatings() that takes a file system path that contains the rating file as input. loadRatings() then load all ratings into a table (saved in PostgreSQL) named ratings that has the following schema

userid(int) – movieid(int) – rating(float)

For your testing, we provide test_data.txt which provides a small fraction of rating.dat file. Be noted that we will use a larger dataset during evaluation. Also note that we don't load timestamps of ratings.

6. Implement a Python function *rangePartition()* that takes as input: (1) the Ratings table stored in PostgreSQL and (2) an integer value N; that represents the number of partitions. Then, *rangePartition()* generates N horizontal fragments of the ratings table and store them in PostgreSQL. The algorithm should partition the ratings table based on N uniform ranges of the rating attribute.

7. Implement a Python function *roundRobinPartition()* that takes as input: (1) the ratings table stored in PostgreSQL and (2) an integer value N; that represents the number of partitions. The function then generates N horizontal fragments of the ratings table and stores them in PostgreSQL. The algorithm should partition the ratings table using the round robin partitioning approach (explained in class).

8. Implement a Python function *roundRobinInsert()* that takes as input: (1) ratings table stored in PostgreSQL, (2) userid, (3) itemid, (4) rating. Then, *roundRobinInsert()* inserts a new tuple to the ratings table and the right fragment based on the round robin approach.

9. Implement a Python function *rangeInsert()* that takes as input: (1) ratings table stored in PostgreSQL (2) userid, (3) itemid, (4) rating. Then, *rangeInsert()* inserts a new tuple to the ratings table and the correct fragment (of the partitioned ratings table) based upon the rating value.

10. Implement a Python function rangeQuery() that takes as input: (1) *RatingMinValue* (2) *RatingMaxValue* (3) *openconnection (4) outputPath*. Please note that the *rangeQuery* would not use ratings table but it would use the range and round robin partitions of the ratings table. Then, *rangeQuery*() returns all tuples for which the *rating* value is larger than or equal to *RatingMinValue* and less than or equal to *RatingMaxValue*. The returned tuples should be stored in outputPath. Each line represents a tuple that has the following format such that *PartitionName* represents the full name of the partition i.e. *RangeRatingsPart1* or *RoundRobinRatingsPart4* etc. in which this tuple resides.

PartitionName, UserID, MovieID, Rating

Example:

range_ratings_part0,1,377,0.5
round_robin_ratings_part1,1,377,0.5

Note: Please use ‘,’ (COMMA, no space character) as delimiter between *PartitionName*, *UserID*, *MovieID* and *Rating*.

10. Implement a Python function *pointQuery* that takes as input: (1) *RatingValue*. (2) *openconnection (3) outputPath*. Please note that the *pointQuery* would not use ratings table but it would use the range and round robin partitions of the ratings table. Then, *pointQuery*() returns all tuples for which the *rating* value is equal to *RatingValue*. The returned tuples should be stored in outputPath. Each line represents a tuple that has the following format such that *PartitionName* represents the full name of the partition i.e. *RangeRatingsPart1* or *RoundRobinRatingsPart4* etc. in which this tuple resides.

PartitionName, UserID, MovieID, Rating

Example

range_ratings_part3,23,459,3.5
round_robin_ratings_part4,31,221,0

Note: Please use ‘,’ (COMMA, no space character) as delimiter between *PartitionName*, *UserID*, *MovieID* and *Rating*.

Partitioning Questions:

The number of partitions here refer to the number of tables to be created. For rating values in [0, 0.5, 1, 1.5, 2, 2.5, 3, 3.5, 4, 4.5, 5]

Case N = 1, One table containing all the values.

Case N = 2, Two tables

Partition 0 has values [0,2.5]

Partition 1 has values (2.5,5]

Case N = 3, Three tables

Partition 0 has values [0, 1.67]

Partition 1 has values (1.67, 3.34]

Partition 2 has values (3.34, 5]

Uniform ranges means a region is divided uniformly, I hope the example gives a clear picture.

Assignment Tips!

- Partition numbers start from 0, if there are 3 partitions then *range_ratings_part0*, *range_ratings_part1*, *range_ratings_part2* are partition table names for range partitions and *round_robin_ratings_part0*, *round_robin_ratings_part1*, *round_robin_ratings_part2* are partition table names for round robin partitions.
- Do not change partition table names prefix given in tester1.py
- Do not hard code input filename.
- Please use the same database name, table name, user name and password as provided in the assignment to keep it consistent.
- Do not import anything from testHelper1.py and tester1 files. These files are provided only for your testing purpose. If you import anything from these files, your submission will raise error in our test environment which will result in 0 point.

- Please make sure to run the provided tester and make sure there is no indentation error in your submission. In case of **any compilation error, 0 marks will be given.**
- For any case of doubt in the assignment, PLEASE USE Discussion Boards, Individual mails would not be entertained.
- The output file should have the exactly same format with the sample. The grading is based on string comparison. One **mismatch** will cause 1 point loss, two mismatch will cause 2 points loss, etc.
- Pay attention to the **outputPath**. If you output the result to a different path, the grading script will not be able to locate your file. You will get 0 point.
- Do not use global variables in your implementation. Global variable is allowed only for storing two prefixes: RANGE_TABLE_PREFIX and RROBIN_TABLE_PREFIX provided in tester1.py file. You can declare these two global variables in Interface1.py file also. No other global variable is allowed. Meta-data table in the database is allowed.
- You are not allowed to modify the data file on disk.
- Two insert functions can be called many times at any time. They are designed for maintaining the tables in the database when insertions happen.
- Test cases provided in tester1.py file are for your testing only. We will use more test cases during evaluation. So, think about critical test cases during your implementation.
- Use Python 3.x version.

Submission Instructions:

- Only submit the Interface1.py file. Do not change the file name. Do not put it into a folder or upload a zip.
- Multiple submissions are allowed. Only the latest submission will be graded. No late submission is accepted.

Note:-

Failure to follow the instructions provided in the document will result in the loss of the points.