

CSE 511: Data Processing at Scale

Data Fragmentation

Introduction:

The required task is to simulate data partitioning approaches on top of an open source relational database management system (i.e., PostgreSQL). You must generate a set of Python functions that load the input data into a relational table, partition the table using different horizontal fragmentation approaches, and insert new tuples into the right fragment.

Input Data

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

The ratings.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

Requirements:

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

1. Install PostgreSQL 9.5 if you have not already installed it.
2. Download ratings.dat file from the MovieLens website:
<http://files.grouplens.org/datasets/movielens/ml-10m.zip>

You can use partial data from ratings.dat for testing. You do not need to test the entire dataset.

3. Implement a Python function `Load_Ratings()` that takes a file system path that contains the ratings.dat file as input. `Load_Ratings()` then load the ratings.dat content into a table (saved in PostgreSQL) named Ratings that has the following schema:

UserID(int) – MovieID(int) – Rating(float)

4. Implement a Python function `Range_Partition()` that takes as input: (1) the Ratings table stored in PostgreSQL and (2) an integer value N; that represents the number of partitions. `Range_Partition()` then 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.
5. Implement a Python function `RoundRobin_Partition()` 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).
6. Implement a Python function `RoundRobin_Insert()` that takes as input: (1) Ratings table stored in PostgreSQL, (2) UserID, (3) MovieID, (4) Rating. `RoundRobin_Insert()` then inserts a new tuple to the Ratings table and the right fragment based on the round robin approach.
7. Implement a Python function `Range_Insert()` that takes as input: (1) Ratings table stored in Postgresql (2) UserID, (3) MovieID, (4) Rating. `Range_Insert()` then inserts a new tuple to the Ratings table and the correct fragment (of the partitioned ratings table) based upon the Rating value.
8. Implement function `Delete_Partitions()` for your testing convenience. It will not be graded.

Frequently Asked Questions:

- Partition numbers start from 0, if there are 3 partitions then `range_part0`, `range_part1`, `range_part2` are partition table names for range partitions and `rrobin_part0`, `rrobin_part1`, `rrobin_part2` are partition table names for round robin partitions.
- Do not change partition table names prefix given in `assignment_tester.py`
- Do not hard code input filename.
- Do not hard code database name.
- Table schema should be equivalent to what has been described in point 3.
- Use Python 2.7.x version.

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:

1. Do not use global variables in your implementation. Metadata table in the database is allowed.
2. You are not allowed to modify the data file on disk.
3. Two insert functions can be called many times at any time. They are designed for maintaining the tables in the database when insertions happen.
4. Any print function you add to your Interface.py will be suppressed in the grader feedback.

Submission:

Only submit the Interface.py file. Do not change the file name. Do not put it into a folder or upload a zip.

We provide a virtual machine that has the testing environment and an installed PostgreSQL:

OS username: user

OS password: user

Postgres username: postgres

Postgres password: 1234

You can download it and use any VM software such as [VirtualBox](#) to import it:

<https://drive.google.com/file/d/1EBImGZmqDQ8XGTuiHPoqP7XE2ZykAXkX/view?usp=sharing>

You will use the following files within your assignment. These files are used to test your Interface.py

1. tester.py: Test your interface.py using this tester. Run it using "python tester.py".
[tester.py](#)
2. testHelper.py: Put this one together with tester.py
[testHelper.py](#)
3. test_data.txt: Some test data
[test_data.txt](#)
4. Interface.py: Implement the interface in Interface.py
[Interface.py](#)

Feedback:

There are a total of five test cases, and the total points are 20, so each test case is 4 points. We will test your "loadratings()", "rangepartition()", "rangeinsert()", "roundrobinpartition()", and "roundrobininsert()" sequentially, and **if one of the functions fails, you will see the corresponding .sql error logs that indicate where the error occurred.**

Test cases are not executed in parallel. You have to unlock all previous test cases to execute the current test case. For example, to execute test case 2, your code must be able to successfully execute test case 1. The test case number in the feedback will provide you an indication of which part of the submission caused the error.