Ira A. Fulton Schools of Engineering
Arizona State University

CSE 511: Data Processing at Scale

Assignment 3: Data Fragmentation

Purpose

The required task is to simulate data partitioning approaches on top of an open source relational database management system (i.e., PostgreSQL).

Objectives

Learners will be able to:

- Link a database in a Python file and then write queries in the Python file to perform certain operations.
- Write code in Python files to observe how round-robin and range partitions function in practice.
- Reproduce how to use virtual machine software.

Technology Requirements

- Python 3.5
- PostgreSQL 9.5

Assignment Description

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.

Please review the **videos and additional details regarding Assignment 3** before beginning. These are located in your course *Welcome and Start Here* section.

- Assignment 3: Data Fragmentation Introduction Video
- Assignment 3: Data Fragmentation Submission and Feedback Video
- Metadata Table in Assignments 3 and 4

Note: Project details in the Overview Document may have been updated since the recording of the videos, so some directions or items may not match perfectly. Please follow the Overview Document's directions to complete your work correctly.

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

Directions

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 stores 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 3.5.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 Directions for Assignment Deliverables

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/1EBlmGZmqDQ8XGTuiHPoqP7XE2ZykAXkX/view?usp=sharing

You will use the following files within your assignment (attached in Coursera's Assignment Overivew page). 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".
2. testHelper.py: Put this one together with tester.py
3. test_data.txt: Some test data

4. Interface.py: Implement the interface in Interface.py

When you are ready to submit:

1. Go to “**Programming Assignment: Assignment 3: Data Fragmentation**”.
2. Click on the "**My submissions**" tab (located at the top of the page under the deadline date).
3. Click on “**Create submission**.”
4. Upload one file for the assignment and click “**Submit**.”
    a. If there is an error in your assignment, you may make corrections and resubmit.

Evaluation

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.**

The test cases are executed in a simultaneous manner. If you pass any 2 of the 5 test cases, you would receive 40% of the total marks.

Common Errors:

1. Error in roundrobinpartition(): Range partitioning not done properly.
2. Error in roundrobininsert(): Round robin insert failed! - Couldnt find (100, 1, 3) tuple in rrobin_part0 table.
3. Error in loadratings(): relation 'ratings' does not exist.
4. Error in rangepartition(): Range partitioning not done properly.
5. Error in rangeinsert(): Range insert failed! Couldnt find (100, 2, 3) tuple in range_part2 table.
6. Exceptions in your Interface.py with following error messages: expected an indented block, No module named ‘pandas’ (Python Errors)
7. Errors in SQL Statements.
8. Incomplete assignment, missing code blocks and poorly written queries.

Learner Checklist

Prior to submitting, read through the Learner Checklist to ensure you are ready to submit your best work.

- ☐ Did you title your file correctly and convert it into a single **Interface.py file**?
- ☐ Did you answer all of the questions to the best of your ability?
- ☐ Did you make sure your answers directly address the prompt(s) in an organized manner that is easy to follow?
- ☐ Did you proofread your work?