The required task is to simulate data partitioning approaches on-top of an open source relational database management system (i.e., PostgreSQL). 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, and insert new tuples into the right fragment. A detailed explanation about round-robin partitioning and range partitioning can be found here: [https://www.ibm.com/support/knowledgecenter/en/SSZJPZ\_11.7.0/com.ibm.swg.im.iis.ds.parjob.dev.doc/topics/partitioning.html (Links to an external site.)](https://www.ibm.com/support/knowledgecenter/en/SSZJPZ_11.7.0/com.ibm.swg.im.iis.ds.parjob.dev.doc/topics/partitioning.html)

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:

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. rangePartition() 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.

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. roundRobinInsert() then 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. rangeInsert() then inserts a new tuple to the ratings table and the correct fragment (of the partitioned ratings table) based upon the rating value.

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 tester1.py
* Do not hard code input filename.
* Do not hard code database name.
* Do not import anything from testHelper1.py file. It is provided only for your testing purpose. If you import anything from this file, your submission will raise error in our test environment which will result in 0 point.
* Table schema should be equivalent to what has been described in point 3.
* Use Python 3.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. 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.

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. 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.

Required Files

You will use the following files for your assignment.

1. Interface1.py: You only need to modify it. Complete the empty methods: loadRatings(), rangePartition(), roundRobinPartition(), roundRobinInsert(), and rangeInsert(). Don't change any other method.

* [Interface1.pyPreview the document](https://asu.instructure.com/courses/46555/files/12450536/download?wrap=1)

2. tester1.py: It contains some test cases for testing Interface1.py. You can add additional test cases if you want. Run tester1.py file to test Interface1.py file.

* [tester1.pyPreview the document](https://asu.instructure.com/courses/46555/files/12450538/download?wrap=1)

3. testHelper1.py: It contains some methods used by tester1.py file. Don't modify anything in this file. Don't import anything from this file into Interface1.py file.

* [testHelper1.pyPreview the document](https://asu.instructure.com/courses/46555/files/12450539/download?wrap=1)

4. test\_data1.txt: Sample test data

* [test\_data1.txtPreview the document](https://asu.instructure.com/courses/46555/files/12450537/download?wrap=1)

Put all these files under same directory during your evaluation.

Submission:

Only submit the Interface1.py file. Do not change the file name. Do not put it into a folder or upload a zip. Make sure that Interface1.py file can be run in Python3.x environment.