The required task is to build a simplified query processor that accesses data from the partitioned ratings table.

Input Data: - Same as in Assignment 1 i.e. ratings.dat file.

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

- RangeQuery() – ○ Implement a Python function *RangeQuery* that takes as input: (1) *Ratings* table stored in PostgreSQL, (2) *RatingMinValue* (3) *RatingMaxValue* (4) *openconnection*
  - Please note that the *RangeQuery* would not use ratings table but it would use the range and round robin partitions of the ratings table.
  - *RangeQuery*() then 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 a text file, named ***RangeQueryOut.txt*** (in the same directory where Assignment2_Interface.py is present) such that 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:

    RangeRatingsPart0,1,377,0.5
    RoundRobinRatingsPart1,1,377,0.5

  - Note: Please use ‘,’ (COMMA) as delimiter between *PartitionName, UserID, MovieID* and *Rating*.

- PointQuery() – ○ Implement a Python function *PointQuery* that takes as input: (1) *Ratings* table stored in PostgreSQL, (2) *RatingValue*. (3) *openconnection*
  - Please note that the *PointQuery* would not use ratings table but it would use the range and round robin partitions of the ratings table.
  - *PointQuery*() then returns all tuples for which the *rating* value is equal to *RatingValue*. ○ The returned tuples should be stored in a text file, named ***PointQueryOut.txt*** (in the same directory where Assignment2_Interface.py is present) such that 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

    RangeRatingsPart3,23,459,3.5
    RoundRobinRatingsPart4,31,221,0

  - Note: Please use ‘,’ (COMMA) as delimiter between *PartitionName, UserID, MovieID* and *Rating*.

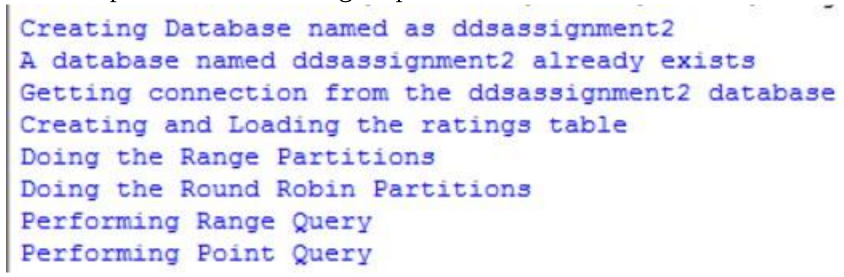
Please use the **function signature exactly same as mentioned in Assignment2_Interface.py**.
Naming Convention to be followed strictly:

- Database name – *ddsassignment2*
- Name of Rating table – *ratings*
- Postgres User name – *postgres*
- Postgres password – *1234*

- Name of the Range Query Output file – *RangeQueryOut.txt*
- Name of the Point Query Output file – *PointQueryOut.txt*

How to use tester.py:

- Open the Assignment2.zip file and dump all the contents in a folder.
- Also move *ratings.dat* file to this folder.
- As per the naming conventions provided above, setup your postgresql.
- Once the setup is done, test the tester.py by simply running it.
- It should provide the following output:



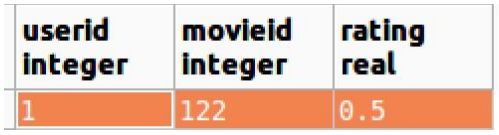
```
Creating Database named as ddsassignment2
A database named ddsassignment2 already exists
Getting connection from the ddsassignment2 database
Creating and Loading the ratings table
Doing the Range Partitions
Doing the Round Robin Partitions
Performing Range Query
Performing Point Query
```

- Now, you can implement your functions in Assignment2_Interface.py and once done, use the tester again to generate the output.
- **DONOT CHANGE** tester.py and Assignment1.pyc. Changing anyone of these would cause problems and the system will stop working, and may lead to **deduction of marks**.
- **PLEASE KEEP IN MIND**, this tester is just for your help. For grading purpose, an additional set of test cases would be used. It will try to break your code. So, please provide the functionalities accordingly, so that it handles all possible scenarios.

Information about the database created via tester.py file: -

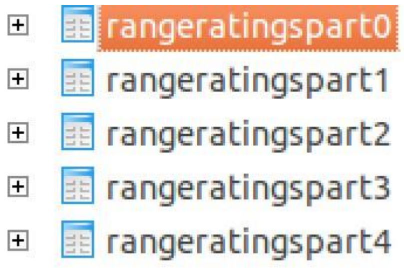
Tables and its structure: -

1. *Ratings*



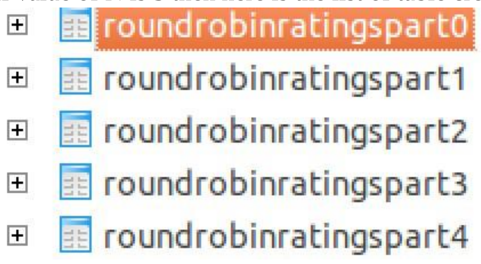
userid integer	movieid integer	rating real
1	122	0.5

2. *RangeRatingsPart0* to *RangeRatingsPartN* (where N = number of partition - 1)
   a. Same structure as *Ratings* table
   b. If value of N is 5 then here is the list of table created for Range Partitioning.



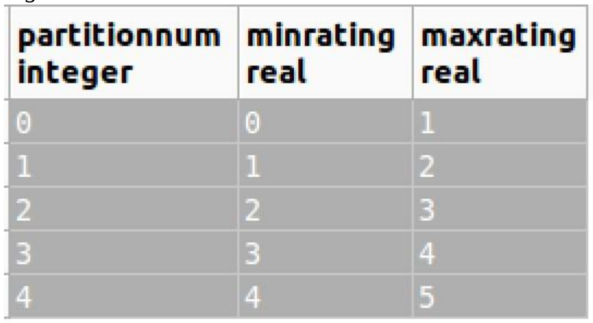
rangeratingspart0
rangeratingspart1
rangeratingspart2
rangeratingspart3
rangeratingspart4

3. *RoundRobinRatingsPart0* to *RoundRobinRatingsPartN* (where N = number of partition - 1)
   a. Same structure as *Ratings* table
   b. If value of N is 5 then here is the list of table created for Round Robin Partitioning



roundrobinratingspart0
roundrobinratingspart1
roundrobinratingspart2
roundrobinratingspart3
roundrobinratingspart4

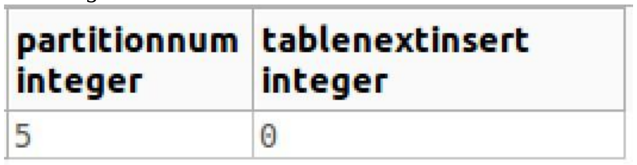
4. *RangeRatingsMetadata*



partitionnum integer	minrating real	maxrating real
0	0	1
1	1	2
2	2	3
3	3	4
4	4	5

- If *PartitionNum* is N, then it means table *RangeRatingsPartN* is being referred and this table contains records for rating values > minrating and <= maxrating.
- Example: - If N is 0, the *RangeRatingsPart0* is being referred and this partition contains all the record for which the rating values are between 0 and 1.

5. *RoundRobinRatingsMetadata*



partitionnum integer	tablenextinsert integer
5	0

- *Partitionnum* denotes total number of partition and *tablenextinsert* denotes that next partition the record should be inserted.