CSE 512 – Assignment 1

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.

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 as comma separated text files where all ratings are contained 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

The lines within this file are ordered first by UserID, then, within user, by MovieID. 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. Download PostgreSQL (http://www.postgresql.org)
2. Download rating.dat file from the MovieLens website (http://files.grouplens.org/datasets/movielens/ml-10m.zip)
3. Implement a Python function `Load_Ratings()` that takes a file system path that contains the rating.dat file as input. `Load_Ratings()` then load the rating.dat content into a table (saved in PostgreSQL) named `Ratings` that has the following schema

   UserID - MovieID - Rating

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) `ItemID`, (4) `Rating`. `RoundRobin_Insert()` then inserts a new tuple in the right fragment (of the partitioned ratings table) 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) `ItemID`, (4) `Rating`. `Range_Insert()` then inserts a new tuple in the correct fragment (of the partitioned ratings table) based upon the `Rating` value.
8. Implement a Python function `Delete_Partitions()` that deletes all generated partitions as well as any metadata related to the partitioning scheme.

Deadline. Wednesday, Sept 16th 2015 (11:59 pm)