Assignment 3: Data Fragmentation

CSE 511: Data Processing at Scale - Spring 2023

Available: 03/01/2023 **Due Date**: 03/15/2023 11:59pm **Points:** 100

Introduction

The task in this assignment is to simulate data partitioning approaches using PostgreSQL. Each student must generate a set of Python functions that load the input data into a relational table, and partition the table using two different horizontal fragmentation approaches. The fragmentation approaches that will be taken in this assignment are range and round-robin partitioning, some more details to the same can be found here:

Problem Statement

We will be using the `subreddits.csv` file and the `created_utc` column to create partitioned tables, this will be maintained for the grading as well. To make it easier to follow, we have divided this problem statement into a series of steps needed in order to get everything up and running. High-level information about each function below has also been made available in the `assignment3.py` file as docstrings:

Step 1: Loading the data (20 pts)

Implement a python function `load_data()`

The very first thing we will do is learn to load data into a postgres table using python.

1. The `load_data()` function takes a total of **4** inputs. The name of table to be created, the path of the csv file, the header file path containing the column header information for the table and the database connection object.
2. To start you off, we have provided the code that creates a table for you. You need to implement loading the data into this table provided.

Hints:

1. Loading rows can take a time when using insert. To make this faster, we can use different methods. This is a good blog reference to make this loading process much faster. The summary section of the same shows the comparison between different methods.

***Note**: In this assignment, you are graded only based on whether data is loaded, **not** on how fast it is loaded.*

Step 2: Range Partition (40 pts)

Implement a Python function `range_partition()`
Now that we have a table with data loaded into it. We will be able to use it to insert data into our new table which has partitions.

1. The `range_partition()` function takes a total of **6** inputs, the name of the table that currently holds the data (from step 1), the name of the table to be partitioned, the number of partitions to create, the path to the header file that contains column headers and their data types, the column based on which we are creating the partition, the database connection object.
2. PostgreSQL has an inbuilt range partition method that we will be taking advantage of for this step. Here are some references on how to perform range partition in postgres:
   a. dbi Blog (dbi-services.com)
   b. PostgreSQL: Documentation: Table Partitioning (Declarative Partitioning)
3. Your code should generate N horizontal fragments for the column `created_utc` of the `subreddits` table and store them in PostgreSQL. The algorithm should partition the table based on N uniform ranges of the attribute `created_utc`

Hints:
Do refer the references for details. Some high level steps to implement this step:

1. Create table that is about to be partitioned.
   a. You can refer code to create tables in the `load_table()` function and implement it here with required modifications.
2. Get the min and max values of `created_utc` . Decide on the ranges.
   a. The range should be uniform. In case the range is coming out to be float - round it up to the next number. So, if the created_utc is within [1, 102], and the number of partitions is 5 - the range is ((102 - 1) + 1)/5 = 20.4, which will be considered as 21 - the ranges for each partition will be, [1, 22), [22, 43), [43, 64), [64, 85), [85-106).
   b. Data inserted during testing will always be within the range of values currently in the table.
3. Create the partition tables
   a. In case the partition_table_name is “range_part”, and there are 5 partitions, then the name of the parent table will be range_part with the name of the child (partitions) tables as range_part0, range_part1, range_part2, range_part3, and range_part4.
4. insert data from the table created in Step 1 to these partitions now.

Step 3: Round Robin Partition (40pts)

Implement a Python function `round_robin_partition()`
In the earlier function, we saw how to create range partitions using inbuilt postgres function. Range partition is available in postgres, however, round robin partition is not available. In this step, we will use other operations to simulate the same in postgres.

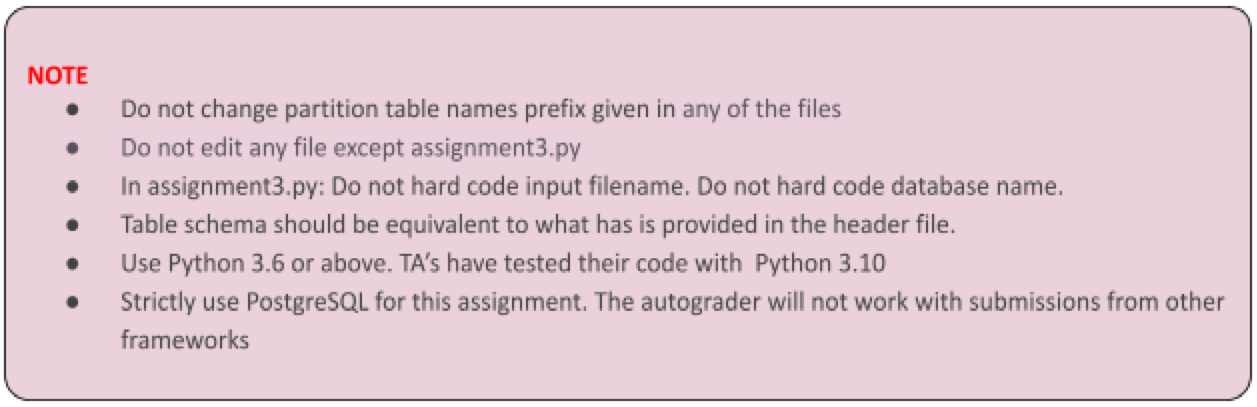
1. The `round_robin_partition()` takes a total of **5** inputs, the name of the table to be partitioned, the number of partitions to create, the path to the header file that contains column headers and their data types, the column based on which we are creating the partition, the database connection object.
2. PostgreSQL does not have built-in round robin partitioning. For this purpose, we will use Triggers in postgres to simulate this. A detailed walkthrough of this is present at:
    a. PostgreSQL: Documentation: Table Partitioning (Partition using Inhertance)
3. Your code should generate N horizontal fragments for the column `created_utc` of the `subreddits` table and store them in PostgreSQL. The algorithm should partition the table based on the round robin approach.

Hints:
The reference will cover all the necessary steps, however we might not need some of them. Some high level ideas to implement this step:

1. Create table that is about to be partitioned.
    a. You can refer to the code used to create tables in the load_table() function and implement it here with required modifications.
2. Create the partition tables
    a. In case partition_table_name is "rrobin_part", and the number of partitions is 3, then the name of the parent table will be "rrobin_part" and the child (partition) tables will be rrobin_part0, rrobin_part1, rrobin_part2.
3. Insert data from our step 1 table into these partitions.
    a. You can simply iterate over the data table rows and insert rows into one partition after the other.
4. Create a trigger function that will find partition where new data needs to go.
    a. The official postgres documentation will help a lot, specially to understand how triggers work. The high level steps that need to be done are
    b. Create a function that performs the following operations:
        i. Take the count of rows of each of the partition tables
        ii. Compare when there is a mismatch in rows to find and insert data into the correct table.

c. Create a trigger function that calls the above function on every insert. We will not be checking any other operation except insert, so, you do not have to worry about those.

***Note**: One of the tests is to insert data into partition tables. We have already provided a insert function to do this in the* `test_helper.py` *file. You can refer it to see how the insert operation works and design your partition table accordingly.*



NOTE

- Do not change partition table names prefix given in any of the files
- Do not edit any file except assignment3.py
- In assignment3.py: Do not hard code input filename. Do not hard code database name.
- Table schema should be equivalent to what has is provided in the header file.
- Use Python 3.6 or above. TA's have tested their code with Python 3.10
- Strictly use PostgreSQL for this assignment. The autograder will not work with submissions from other frameworks

How to work on the assignment and test your code

- You can work choose to work on your local setup, or to use the docker image.
- PostgreSQL 14 is already present (but not running) in the docker image. Find the configurations below
  - username: postgres
  - password: postgres
- Meta-data table in the database is allowed.
- You are not allowed to modify the data file on disk.
- Test your code using the tester.py file. Run it using `python3 tester.py`

Grading

- The assignment is **autograded**.
- You will use the following files within your assignment. These files are used to test your code
  a. `tester.py`: Test your code using this tester. Run it using "`python3 tester.py`".
  b. `test_helper.py`: put this one together with tester.py test_helper.py
  c. `assignment3.py`: Implement the interface in this file.
  DO NOT change any other file.
- Make sure that your code runs on the docker container provided! That environment will allow for a partial reproduction of the grading environment. ***In case your script fails to run in the grading environment, you will be graded 0.***

Submission Requirements & Guidelines

- This is an **individual** assignment
- Maintain your code on the **provided private GitHub repository for assignment-3** in the [SPRING-2023] [CSE511] Data Processing at Scale Organization). Make sure you don't use any other repository.
- **What to submit on canvas?**
  - One python file "`assignment3.py`" needs to be submitted on canvas.
  - You **MUST** name your python file as `assignment3.py`

Submission Policies

- Late submissions will *absolutely not* be graded (unless you have verifiable proof of emergency). It is much better to submit partial work on time and get partial credit than to submit late for no credit.
- Every student needs to *work independently* on this exercise. We encourage high-level discussions among students to help each other understand the concepts and principles. However, a code-level discussion is prohibited, and plagiarism will directly lead to the failure of this course. We will use anti-plagiarism tools to detect violations of this policy.