Term: Fall 2023 **Subject:** Computer Science & Engineering (CSE) **Number:** 512

Course Title: Distributed **D**atabase **S**ystems (CSE 512)

Assignment 1

Due on Monday, October 2nd at 11:59 pm

This assignment is designed to help you understand the data partitioning approaches on top of the PostgreSQL database management system. The objective is that you can partition the table using different ***horizontal fragmentation*** approaches: ***list partitioning*** and ***range partitioning***, write the necessary set of Python functions, and insert new tuples into the right fragment.

Please submit a zip folder of your code and a pdf document with the results.

As data is inserted randomly, the results must be different for each student.

ASU Academic Integrity:
Students in this class must adhere to ASU's academic integrity policy, which can be found at https://provost.asu.edu/academic-integrity/policy. If you are caught cheating, you will be reported to the Fulton Schools of Engineering Academic Integrity Office (AIO). The AIO maintains a record of all violations and has access to academic integrity violations committed in all other ASU colleges/schools.

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

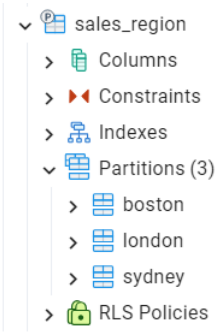
1. Install PostgreSQL.
   https://www.postgresql.org/download/
2. Download Python
3. Install psycopg2 in your project to connect with PostgreSQL.
   pip install psycopg2
4. Download the assignment1.py file and implement the required Python functions explained below.
   Do not close the connection inside the implemented function.
5. Write a report to explain how list and range partitioning works. Please add results and screenshots from the Python project and PostgreSQL program.

Part 1 (5 points): connect_postgres()
Implement a Python function **connect_postgres()** that connects the PostgreSQL. Please use the same password for installing the PostgreSQL to build a connection.

user = ‘postgres’
dbname = ‘postgres’’
host = ‘localhost’

Part 2 (5 points): create_database()
Implement a Python function **create_database()** that creates an ‘assignment1’ database in the PostgreSQL system. You need to connect to PostgreSQL using the **connect_postgres()** function.

Part 3 (15 points):list_partitioning()
Implement a Python function **list_partitioning()** that creates a **sales_region** table stored in PostgreSQL. This table has **id (int), amount (int), and region (text)** columns. **List partitioning** needs to be applied to the **region** field. Then generate three horizontal fragments of the sales_region table according to three regions: **London, Boston, and Sydney** and store them in PostgreSQL. After running this function, you can check PostgreSQL to see if the tables are created. You can see three partition tables, Boston, London, and Sydney, under the sales_region table like the image below. There will be no data in the tables at that point.



Part 4 (15 points): insert_list_data()
Implement a Python function **insert_list_data()** that generates and executes an INSERT statement for 50 rows in the sales_region table. The column values in the table will be assigned randomly. You need to import a random library. After generating column values, you need to execute the **INSERT** statement and add data to the **sales_region** table. Below are the criteria for the column data.

id: starts from 1 to 50. Increase one by one.
amount: random value between 100 and 1000. Use random.randint() function
region: random choice of REGIONS list. Use random.choice(REGIONS) function

---

Part 5 (10 points): select_list_data()
Implement a Python function **select_list_data()** that selects data from **sales_region, Boston, London, and Sydney** tables. Execute the **SELECT** query for each table and **print** their data to the console like the images below. Please take a **screenshot of the console** and add it to the report.

```
Sales Region Data:
(2, 138, 'Boston')
(4, 114, 'Boston')
(5, 406, 'Boston')
```

```
Boston Data:
(2, 138, 'Boston')
(4, 114, 'Boston')
(5, 406, 'Boston')
```

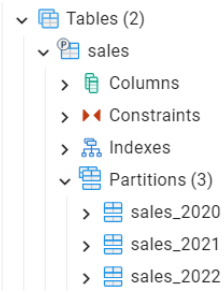
```
London Data:
(1, 582, 'London')
(3, 100, 'London')
(6, 994, 'London')
```

```
Sydney Data:
(11, 421, 'Sydney')
(14, 978, 'Sydney')
(15, 404, 'Sydney')
```

Part 6 (15 points): range_partitioning()
Implement a Python function **range_partitioning()** that creates a **sales** table and then generates three range partitioning of the table. The columns of the sales table are **id (int), product_name (text), amount (int), sale_date (date)**. Range partitioning needs to be applied to the **sale_date** column. There will be three range partitioning tables according to sale_date.

1. sales_2020 table: VALUES FROM ('2020-01-01') TO ('2020-12-31')
2. sales_2021 table: VALUES FROM ('2021-01-01') TO ('2021-12-31')
3. sales_2022 table: VALUES FROM ('2022-01-01') TO ('2022-12-31')

After running this function, you can check PostgreSQL to see if the tables are created. You can see three partition tables, sales_2020, sales_2021, and sales_2022, under the sales table like the image below. There will be no data in the tables at that point.



Part 7 (15 points): insert_range_data()
Implement a Python function **insert_range_data()** that generates and executes INSERT statements for 50 data to the sales table. The column values in the table will be assigned randomly. After generating column values, you need to execute the **INSERT** statement and add data to the **sales** table. Below are the criteria for the column data.

id: starts from 1 to 50. Increase one by one.
product_name: random choice of PRODUCT_NAMES list.
amount: random value between 1 and 100. Use random.randint() function
sale_date : generate random date between 2020-01-01 and 2022-12-31

Part 8 (10 points): select_range_data()
Implement a Python function **select_range_data()** that selects data from **sales, sales_2020, sales_2021 and sales_2022** tables. Execute the **SELECT** query for each table and **print** their data to the console like the images below. Please take a **screenshot of the console** and add it to the report.

```
Sales Data:
(3, 'Product_A', 56, datetime.date(2020, 11, 3))
(10, 'Product_A', 22, datetime.date(2020, 3, 15))
(16, 'Product_C', 19, datetime.date(2020, 6, 19))
```

```
Sales 2020 Data:
(3, 'Product_A', 56, datetime.date(2020, 11, 3))
(10, 'Product_A', 22, datetime.date(2020, 3, 15))
(16, 'Product_C', 19, datetime.date(2020, 6, 19))
```

```
Sales 2021 Data:
(1, 'Product_A', 68, datetime.date(2021, 7, 21))
(2, 'Product_B', 47, datetime.date(2021, 3, 1))
(4, 'Product_D', 30, datetime.date(2021, 4, 5))
```

```
Sales 2022 Data:
(7, 'Product_E', 99, datetime.date(2022, 4, 28))
(11, 'Product_A', 62, datetime.date(2022, 1, 1))
(14, 'Product_B', 66, datetime.date(2022, 9, 9))
```

Part 9 (10 points): Writing a report
Write a report to explain how list and range partitioning works. Please add results and screenshots from the Python project and PostgreSQL program.

----------The END----------