Distributed Database Systems

Assignment 1

Report by - Niraj Sonje

List and Range Partitioning

Introduction

Partitioning is a database design technique that involves dividing a large table into smaller, more manageable pieces known as partitions. These partitions can be created based on specific criteria, such as a column's values or ranges of values. In this report, we will discuss list and range partitioning in the context of a PostgreSQL database using a sample dataset.

Database Creation using Postgres and Python

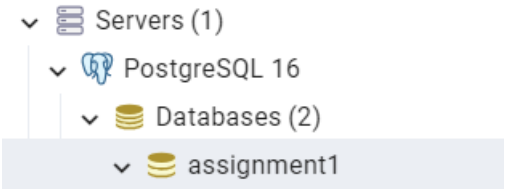
Before partitioning, we create a PostgreSQL database using Python. Here are the steps to achieve this:

- Install the psycopg2 library in Python if not already installed.
- Define a Python function to connect to the PostgreSQL server and create a new database if it does not exist. This function establishes a connection using Python's '**psycopg2**' library.

Terminal:

```
PS C:\E\ASU SEM 1\DDS> & "C:/Users/Niraj Sonje/AppData/Local/Microsoft/Windows
Do you want to create new database?
Press 1 for Yes or 0 for No: 1
Connecting to Default Database to create new database 'assignment1'
Database 'assignment1' created successfully.
PostgreSQL connection is closed.
Select the operation you want to perform:
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
```

PostgreSQL:



List Partitioning for Sales Regions

List partitioning is a method of dividing a table into partitions based on specific values in a chosen column. In our example, we use the `SALES_REGION_TABLE` to demonstrate list partitioning for sales regions.

Steps:

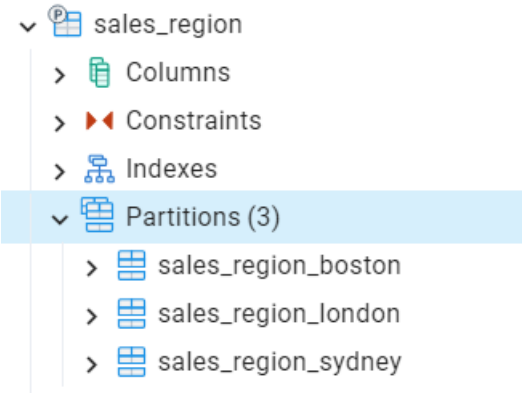
1. **Table Creation**: We start by creating the main table `SALES_REGION_TABLE` with columns like `id`, `region`, and `product`. This table serves as the parent table for our partitions.

   Terminal:

```
Select the operation you want to perform:
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
1
List partitioning created successfully.
```

2. **Partition Tables**: Next, we create separate partition tables for each sales region, such as `sales_region_boston`, `sales_region_sydney`, and `sales_region_london`. These partition tables inherit the structure of the main table.

   PostgreSQL:



3. **Partitioning Criteria**: Importantly, we specify the criteria for each partition table using the `PARTITION BY LIST` clause during table creation. For example, `sales_region_boston` is designated for rows where the `region` column equals 'Boston'.
4. **Data Insertion**: As data is inserted into these tables, PostgreSQL automatically routes the data to the appropriate partition based on the `region` value.

Terminal:

```
List partitioning created successfully.
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
2
Data inserted into SALES_REGION_TABLE successfully.
```

5. **Querying**: When querying data from `SALES_REGION_TABLE`, PostgreSQL accesses the relevant partition efficiently based on the `region` value, optimizing query performance.

   Terminal (sales_region):

```
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
3

Data from sales_region:

(1, 'Boston', 219)
(2, 'Boston', 529)
(4, 'Boston', 500)
(5, 'Boston', 370)
(6, 'Boston', 435)
(9, 'Boston', 389)
(10, 'Boston', 672)
(12, 'Boston', 668)
(15, 'Boston', 942)
(17, 'Boston', 174)
(18, 'Boston', 905)
(20, 'Boston', 982)
(23, 'Boston', 125)
(25, 'Boston', 256)
(27, 'Boston', 506)
(29, 'Boston', 469)
(30, 'Boston', 617)
(32, 'Boston', 853)
(35, 'Boston', 855)
(37, 'Boston', 935)
(43, 'Boston', 447)
(3, 'London', 417)
(16, 'London', 988)
(26, 'London', 122)
(33, 'London', 500)
(34, 'London', 581)
(36, 'London', 969)
(39, 'London', 653)
(40, 'London', 744)
(46, 'London', 512)
(47, 'London', 683)
(48, 'London', 197)
(7, 'Sydney', 828)
(8, 'Sydney', 191)
(11, 'Sydney', 896)
```

Terminal (sales_region_boston):

```
Data from sales_region_boston:

(1, 'Boston', 219)
(2, 'Boston', 529)
(4, 'Boston', 500)
(5, 'Boston', 370)
(6, 'Boston', 435)
(9, 'Boston', 389)
(10, 'Boston', 672)
(12, 'Boston', 668)
(15, 'Boston', 942)
(17, 'Boston', 174)
(18, 'Boston', 905)
(20, 'Boston', 982)
(23, 'Boston', 125)
(25, 'Boston', 256)
(27, 'Boston', 506)
(29, 'Boston', 469)
(30, 'Boston', 617)
(32, 'Boston', 853)
(35, 'Boston', 855)
(37, 'Boston', 935)
(43, 'Boston', 447)
```

Terminal (sales_region_sydney):

```
Data from sales_region_sydney:

(7, 'Sydney', 828)
(8, 'Sydney', 191)
(11, 'Sydney', 896)
(13, 'Sydney', 169)
(14, 'Sydney', 538)
(19, 'Sydney', 420)
(21, 'Sydney', 388)
(22, 'Sydney', 206)
(24, 'Sydney', 208)
(28, 'Sydney', 176)
(31, 'Sydney', 482)
(38, 'Sydney', 749)
(41, 'Sydney', 433)
(42, 'Sydney', 478)
(44, 'Sydney', 602)
(45, 'Sydney', 971)
(49, 'Sydney', 656)
(50, 'Sydney', 208)
```

Terminal (sales_region_london):

```
Data from sales_region_london:

(3, 'London', 417)
(16, 'London', 988)
(26, 'London', 122)
(33, 'London', 500)
(34, 'London', 581)
(36, 'London', 969)
(39, 'London', 653)
(40, 'London', 744)
(46, 'London', 512)
(47, 'London', 683)
(48, 'London', 197)
```

Range Partitioning for Sales Dates

Range partitioning involves dividing a table into partitions based on a specific range of values in a chosen column. In our example, we use the `SALES_TABLE` to demonstrate range partitioning for sales dates.

Steps:

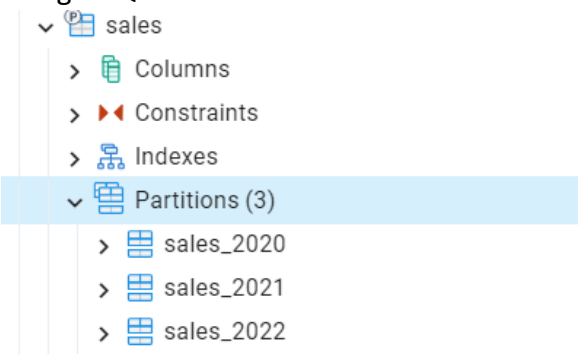
1. **Table Creation**: Similar to list partitioning, we create the main table `SALES_TABLE` with columns like `id`, `sale_date`, `amount `, and `product`.

   Terminal:

```
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
4
Range partitioning created successfully.
```

2. **Partition Tables**: We create separate partition tables, such as `sales_2020`, `sales_2021`, and `sales_2022`, corresponding to specific years. These partition tables inherit the structure of the main table.

   PostgreSQL:



3. **Data Insertion**: Data is inserted into these tables, and PostgreSQL automatically routes the data to the appropriate partition based on the `sale_date`.

   Terminal:

```
1. List Partitioning
2. Insert List Data
3. Select List Data
4. Range Partitioning
5. Insert Range Data
6. Select Range Data
7. Exit
5
Data inserted into SALES_TABLE successfully.
```

4. **Querying**: When querying data from `SALES_TABLE`, PostgreSQL accesses the relevant partition based on the `sale_date` range, optimizing query performance for specific years.

   Termial (sales_table):

```
Data from sales:

(1, datetime.date(2020, 2, 14), 73, 'Product_A')
(10, datetime.date(2020, 3, 31), 23, 'Product_E')
(12, datetime.date(2020, 9, 3), 17, 'Product_B')
(13, datetime.date(2020, 11, 23), 4, 'Product_C')
(30, datetime.date(2020, 3, 10), 13, 'Product_E')
(33, datetime.date(2020, 8, 20), 55, 'Product_C')
(36, datetime.date(2020, 1, 11), 69, 'Product_A')
(39, datetime.date(2020, 4, 28), 87, 'Product_D')
(43, datetime.date(2020, 3, 30), 34, 'Product_C')
(45, datetime.date(2020, 12, 6), 75, 'Product_E')
(49, datetime.date(2020, 11, 27), 82, 'Product_D')
(2, datetime.date(2021, 11, 21), 89, 'Product_B')
(3, datetime.date(2021, 5, 19), 98, 'Product_C')
(4, datetime.date(2021, 6, 26), 72, 'Product_D')
(5, datetime.date(2021, 10, 26), 31, 'Product_E')
(6, datetime.date(2021, 2, 2), 17, 'Product_A')
(7, datetime.date(2021, 5, 21), 78, 'Product_B')
(9, datetime.date(2021, 10, 26), 11, 'Product_D')
(17, datetime.date(2021, 10, 7), 23, 'Product_B')
(18, datetime.date(2021, 3, 10), 52, 'Product_C')
(20, datetime.date(2021, 5, 26), 59, 'Product_E')
(21, datetime.date(2021, 5, 30), 13, 'Product_A')
(22, datetime.date(2021, 6, 13), 52, 'Product_B')
(24, datetime.date(2021, 12, 20), 43, 'Product_D')
(25, datetime.date(2021, 8, 8), 71, 'Product_E')
(26, datetime.date(2021, 8, 28), 52, 'Product_A')
(27, datetime.date(2021, 2, 27), 100, 'Product_B')
(35, datetime.date(2021, 5, 25), 19, 'Product_E')
(47, datetime.date(2021, 6, 15), 39, 'Product_B')
(8, datetime.date(2022, 2, 21), 10, 'Product_C')
(11, datetime.date(2022, 10, 24), 31, 'Product_A')
(14, datetime.date(2022, 7, 14), 14, 'Product_D')
(15, datetime.date(2022, 7, 6), 87, 'Product_E')
(16, datetime.date(2022, 1, 17), 95, 'Product_A')
(19, datetime.date(2022, 1, 29), 57, 'Product_D')
(23, datetime.date(2022, 11, 22), 5, 'Product_C')
(28, datetime.date(2022, 5, 11), 51, 'Product_C')
(29, datetime.date(2022, 12, 26), 13, 'Product_D')
(31, datetime.date(2022, 12, 25), 61, 'Product_A')
(32, datetime.date(2022, 9, 15), 74, 'Product_B')
(34, datetime.date(2022, 1, 28), 6, 'Product_D')
(37, datetime.date(2022, 9, 10), 75, 'Product_B')
(38, datetime.date(2022, 6, 1), 69, 'Product_C')
(40, datetime.date(2022, 9, 27), 74, 'Product_E')
```

Terminal (sales_table_2020):

```
Data from sales_2020:

(1, datetime.date(2020, 2, 14), 73, 'Product_A')
(10, datetime.date(2020, 3, 31), 23, 'Product_E')
(12, datetime.date(2020, 9, 3), 17, 'Product_B')
(13, datetime.date(2020, 11, 23), 4, 'Product_C')
(30, datetime.date(2020, 3, 10), 13, 'Product_E')
(33, datetime.date(2020, 8, 20), 55, 'Product_C')
(36, datetime.date(2020, 1, 11), 69, 'Product_A')
(39, datetime.date(2020, 4, 28), 87, 'Product_D')
(43, datetime.date(2020, 3, 30), 34, 'Product_C')
(45, datetime.date(2020, 12, 6), 75, 'Product_E')
(49, datetime.date(2020, 11, 27), 82, 'Product_D')
```

Terminal (sales_table_2021):

```
Data from sales_2021:

(2, datetime.date(2021, 11, 21), 89, 'Product_B')
(3, datetime.date(2021, 5, 19), 98, 'Product_C')
(4, datetime.date(2021, 6, 26), 72, 'Product_D')
(5, datetime.date(2021, 10, 26), 31, 'Product_E')
(6, datetime.date(2021, 2, 2), 17, 'Product_A')
(7, datetime.date(2021, 5, 21), 78, 'Product_B')
(9, datetime.date(2021, 10, 26), 11, 'Product_D')
(17, datetime.date(2021, 10, 7), 23, 'Product_B')
(18, datetime.date(2021, 3, 10), 52, 'Product_C')
(20, datetime.date(2021, 5, 26), 59, 'Product_E')
(21, datetime.date(2021, 5, 30), 13, 'Product_A')
(22, datetime.date(2021, 6, 13), 52, 'Product_B')
(24, datetime.date(2021, 12, 20), 43, 'Product_D')
(25, datetime.date(2021, 8, 8), 71, 'Product_E')
(26, datetime.date(2021, 8, 28), 52, 'Product_A')
(27, datetime.date(2021, 2, 27), 100, 'Product_B')
(35, datetime.date(2021, 5, 25), 19, 'Product_E')
(47, datetime.date(2021, 6, 15), 39, 'Product_B')
```

Terminal (sales_tables_2022):

```
Data from sales_2022:

(8, datetime.date(2022, 2, 21), 10, 'Product_C')
(11, datetime.date(2022, 10, 24), 31, 'Product_A')
(14, datetime.date(2022, 7, 14), 14, 'Product_D')
(15, datetime.date(2022, 7, 6), 87, 'Product_E')
(16, datetime.date(2022, 1, 17), 95, 'Product_A')
(19, datetime.date(2022, 1, 29), 57, 'Product_D')
(23, datetime.date(2022, 11, 22), 5, 'Product_C')
(28, datetime.date(2022, 5, 11), 51, 'Product_C')
(29, datetime.date(2022, 12, 26), 13, 'Product_D')
(31, datetime.date(2022, 12, 25), 61, 'Product_A')
(32, datetime.date(2022, 9, 15), 74, 'Product_B')
(34, datetime.date(2022, 1, 28), 6, 'Product_D')
(37, datetime.date(2022, 9, 10), 75, 'Product_B')
(38, datetime.date(2022, 6, 1), 69, 'Product_C')
(40, datetime.date(2022, 9, 27), 74, 'Product_E')
(41, datetime.date(2022, 3, 20), 98, 'Product_A')
(42, datetime.date(2022, 11, 20), 99, 'Product_B')
(44, datetime.date(2022, 1, 18), 41, 'Product_D')
(46, datetime.date(2022, 11, 20), 66, 'Product_A')
(48, datetime.date(2022, 6, 17), 11, 'Product_C')
(50, datetime.date(2022, 3, 20), 64, 'Product_E')
```

Conclusion

List and range partitioning are essential techniques for optimizing the performance of databases, particularly when dealing with large datasets. List partitioning allows us to divide data based on specific values, while range partitioning divides data based on ranges of values. By doing so, databases can efficiently manage and query data, leading to improved performance and scalability.

In the assignment, we demonstrated how list partitioning can be used to partition sales data by region and how range partitioning can be applied to partition sales data by date. These techniques can be adapted to various use cases to enhance database performance and organization.