Experiment No.8

Title: Execution of OLAP Operations

Batch:SY-IT(B3) **Roll No.:16010423076** **Experiment No.:8**

Aim: Execution of OLAP operations

Resources needed: MySQL, Postgres

Theory

OLAP:

In computing, online analytical processing, or OLAP is an approach to answering multi-dimensional analytical (MDA) queries. OLAP is part of the broader category of business intelligence, which also encompasses relational database report writing and data mining. Typical applications of OLAP include business reporting for sales, marketing, management reporting, business process management (BPM), budgeting and forecasting, financial reporting and similar areas, with new applications coming up, such as agriculture. The term OLAP was created as a slight modification of the traditional database term OLTP (Online Transaction Processing).

OLAP tools enable users to analyze multidimensional data interactively from multiple perspectives. OLAP consists of three basic analytical operations: consolidation (roll-up), drill-down, and slicing and dicing. Consolidation involves the aggregation of data that can be accumulated and computed in one or more dimensions. For example, all sales offices are rolled up to the sales department or sales division to anticipate sales trends. By contrast, the drill-down is a technique that allows users to navigate through the details. For instance, users can view the sales by individual products that make up a region's sales. Slicing and dicing is a feature whereby users can take out (slicing) a specific set of data of the OLAP cube and view (dicing) the slices from different viewpoints.

OLAP queries can be implemented by using analytical SQL functions
Oracle has extensions to ANSI SQL to allow to quickly computing aggregations and rollups.
These new statements include:

rollup

cube

grouping

These simple SQL operators allow creating easy aggregations directly inside the SQL.

Creating tabular aggregates with ROLLUP:
ROLLUP enables an SQL statement to calculate multiple levels of subtotals across a specified group of dimensions. It also calculates a grand total. ROLLUP is a simple extension to the GROUP BY clause, so its syntax is extremely easy to use. Create cross-tabular reports with CUBE:
In multidimensional jargon, a "cube" is a cross-tabulated summary of detail rows. CUBE enables a SELECT statement to calculate subtotals for all possible combinations of a group of dimensions. It also calculates a grand total.
This is the set of information typically needed for all cross-tabular reports, so CUBE can calculate a cross-tabular report with a single select statement

---

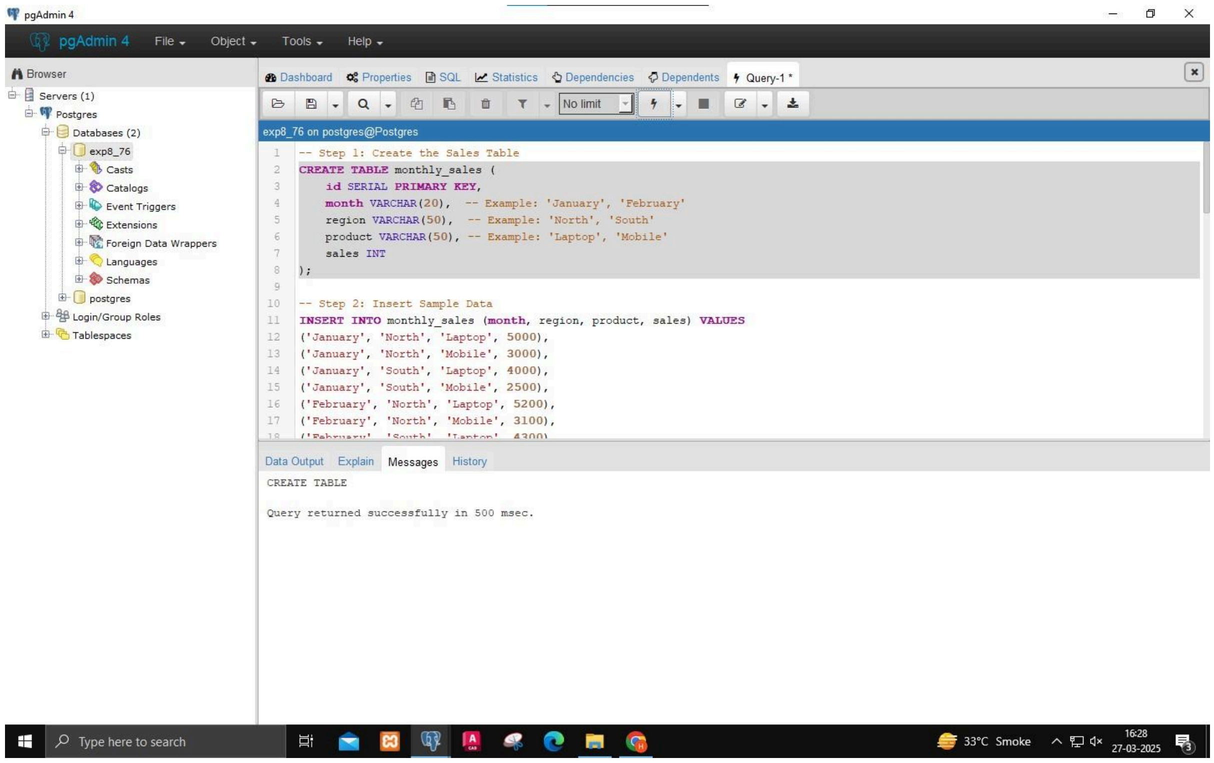
Activities:

1. Create a dataset in PostgreSQL and MySQL
2. Apply rollup and cube operations to the same

Result (With Output) :

```
-- Step 1: Create the Sales Table
CREATE TABLE monthly_sales (
    id SERIAL PRIMARY KEY,
    month VARCHAR(20),  -- Example: 'January', 'February'
    region VARCHAR(50),  -- Example: 'North', 'South'
    product VARCHAR(50), -- Example: 'Laptop', 'Mobile'
    sales INT
);
```

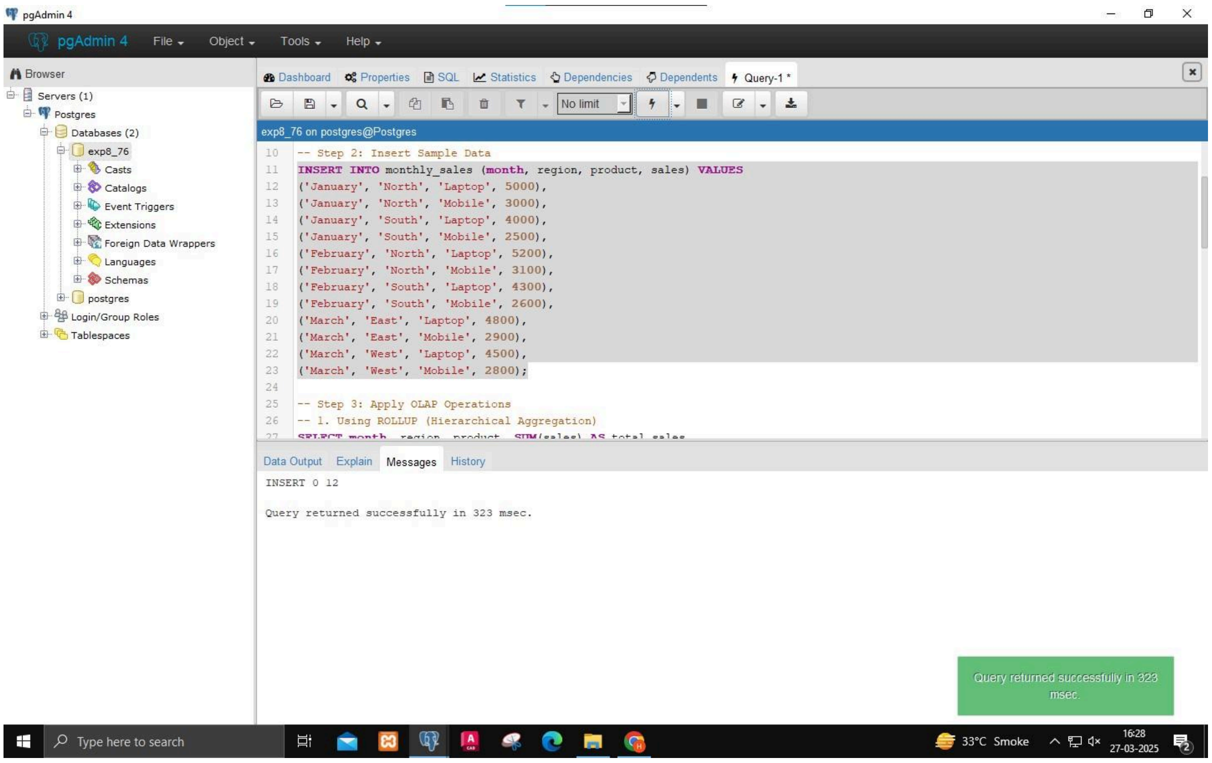
Creates a table monthly_sales with columns for month, region, product, and sales amount.



```
-- Step 1: Create the Sales Table
CREATE TABLE monthly_sales (
    id SERIAL PRIMARY KEY,
    month VARCHAR(20),  -- Example: 'January', 'February'
    region VARCHAR(50),  -- Example: 'North', 'South'
    product VARCHAR(50), -- Example: 'Laptop', 'Mobile'
    sales INT
);

-- Step 2: Insert Sample Data
INSERT INTO monthly_sales (month, region, product, sales) VALUES
('January', 'North', 'Laptop', 5000),
('January', 'North', 'Mobile', 3000),
('January', 'South', 'Laptop', 4000),
('January', 'South', 'Mobile', 2500),
('February', 'North', 'Laptop', 5200),
('February', 'North', 'Mobile', 3100),
('February', 'South', 'Laptop', 4300),
```

```
CREATE TABLE

Query returned successfully in 500 msec.
```

```
-- Step 2: Insert Sample Data
INSERT INTO monthly_sales (month, region, product, sales) VALUES
('January', 'North', 'Laptop', 5000),
('January', 'North', 'Mobile', 3000),
('January', 'South', 'Laptop', 4000),
('January', 'South', 'Mobile', 2500),
('February', 'North', 'Laptop', 5200),
('February', 'North', 'Mobile', 3100),
('February', 'South', 'Laptop', 4300),
('February', 'South', 'Mobile', 2600),
('March', 'East', 'Laptop', 4800),
('March', 'East', 'Mobile', 2900),
('March', 'West', 'Laptop', 4500),
('March', 'West', 'Mobile', 2800);
```



```
-- Step 2: Insert Sample Data
INSERT INTO monthly_sales (month, region, product, sales) VALUES
('January', 'North', 'Laptop', 5000),
('January', 'North', 'Mobile', 3000),
('January', 'South', 'Laptop', 4000),
('January', 'South', 'Mobile', 2500),
('February', 'North', 'Laptop', 5200),
('February', 'North', 'Mobile', 3100),
('February', 'South', 'Laptop', 4300),
('February', 'South', 'Mobile', 2600),
('March', 'East', 'Laptop', 4800),
('March', 'East', 'Mobile', 2900),
('March', 'West', 'Laptop', 4500),
('March', 'West', 'Mobile', 2800);

-- Step 3: Apply OLAP Operations
-- 1. Using ROLLUP (Hierarchical Aggregation)
SELECT month, region, product, SUM(sales) AS total_sales
```

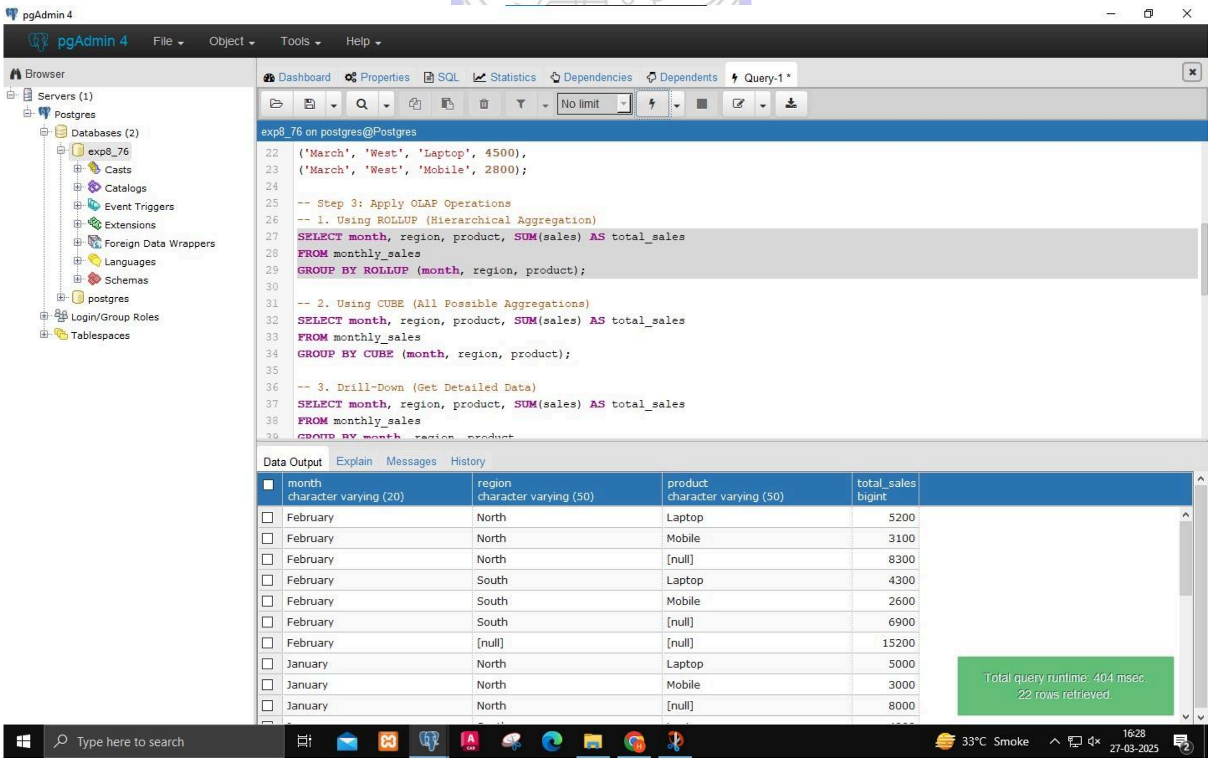
```
INSERT 0 12

Query returned successfully in 323 msec.
```

Query returned successfully in 323 msec.

Inserts sales data for different months, regions, and products into the monthly_sales table.

```
-- Step 3: Apply OLAP Operations
-- 1. Using ROLLUP (Hierarchical Aggregation)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY ROLLUP (month, region, product);
```

Generates total sales per product, per region, per month, and provides subtotals for each hierarchy, including a grand total.



```
('March', 'West', 'Laptop', 4500),
('March', 'West', 'Mobile', 2800);

-- Step 3: Apply OLAP Operations
-- 1. Using ROLLUP (Hierarchical Aggregation)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY ROLLUP (month, region, product);

-- 2. Using CUBE (All Possible Aggregations)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY CUBE (month, region, product);

-- 3. Drill-Down (Get Detailed Data)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY month, region, product
```

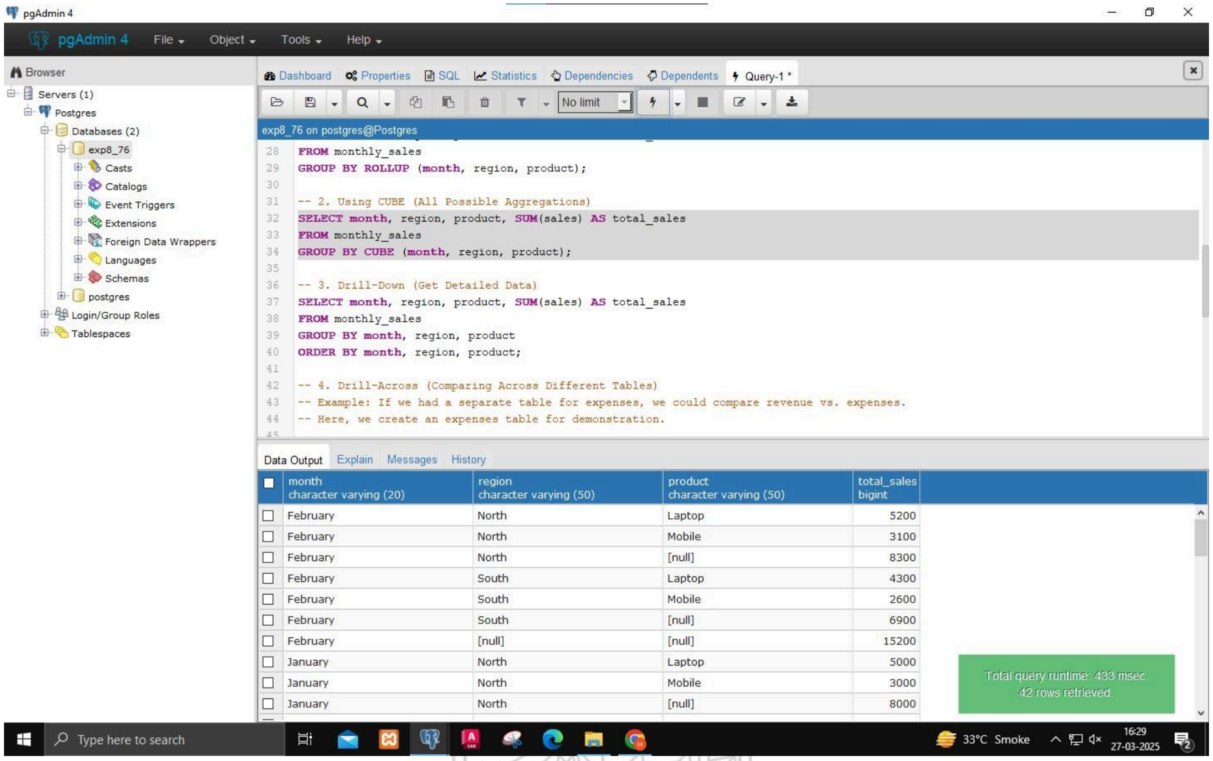
month character varying (20)	region character varying (50)	product character varying (50)	total_sales bigint
February	North	Laptop	5200
February	North	Mobile	3100
February	North	[null]	8300
February	South	Laptop	4300
February	South	Mobile	2600
February	South	[null]	6900
February	[null]	[null]	15200
January	North	Laptop	5000
January	North	Mobile	3000
January	North	[null]	8000

Total query runtime: 404 msec.
22 rows retrieved.

```
-- 2. Using CUBE (All Possible Aggregations)
SELECT month, region, product, SUM(sales) AS total_sales
```

```
FROM monthly_sales
GROUP BY CUBE (month, region, product);
```

Generates all possible subtotal combinations, including totals by month, region, product, and grand total.



exp8_76 on postgres@Postgres

```
FROM monthly_sales
GROUP BY ROLLUP (month, region, product);

-- 2. Using CUBE (All Possible Aggregations)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY CUBE (month, region, product);

-- 3. Drill-Down (Get Detailed Data)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY month, region, product
ORDER BY month, region, product;

-- 4. Drill-Across (Comparing Across Different Tables)
-- Example: If we had a separate table for expenses, we could compare revenue vs. expenses.
-- Here, we create an expenses table for demonstration.
```

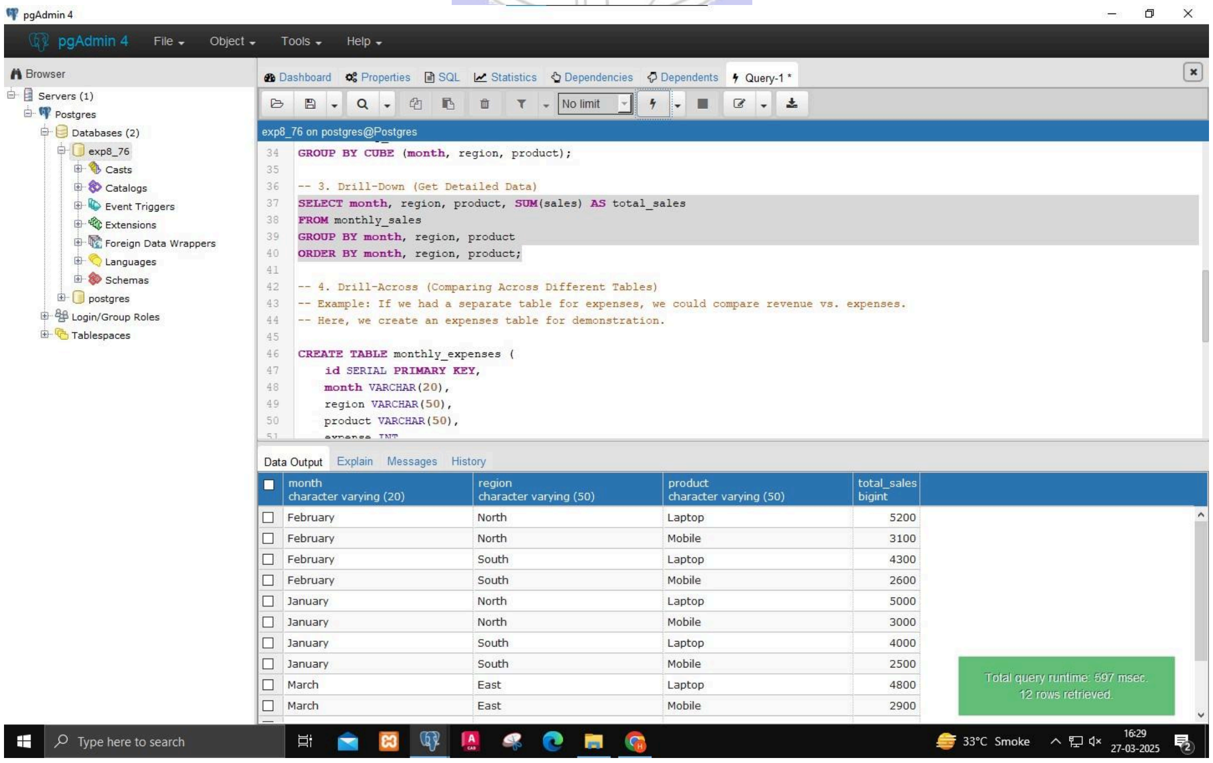
month character varying (20)	region character varying (50)	product character varying (50)	total_sales bigint
February	North	Laptop	5200
February	North	Mobile	3100
February	North	[null]	8300
February	South	Laptop	4300
February	South	Mobile	2600
February	South	[null]	6900
February	[null]	[null]	15200
January	North	Laptop	5000
January	North	Mobile	3000
January	North	[null]	8000

Total query runtime: 433 msec.
42 rows retrieved.

-- 3. Drill-Down (Get Detailed Data)

```
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY month, region, product
ORDER BY month, region, product;
```

Displays detailed sales data for each month, region, and product without aggregation levels.



exp8_76 on postgres@Postgres

```
GROUP BY CUBE (month, region, product);

-- 3. Drill-Down (Get Detailed Data)
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY month, region, product
ORDER BY month, region, product;

-- 4. Drill-Across (Comparing Across Different Tables)
-- Example: If we had a separate table for expenses, we could compare revenue vs. expenses.
-- Here, we create an expenses table for demonstration.

CREATE TABLE monthly_expenses (
    id SERIAL PRIMARY KEY,
    month VARCHAR(20),
    region VARCHAR(50),
    product VARCHAR(50),
```

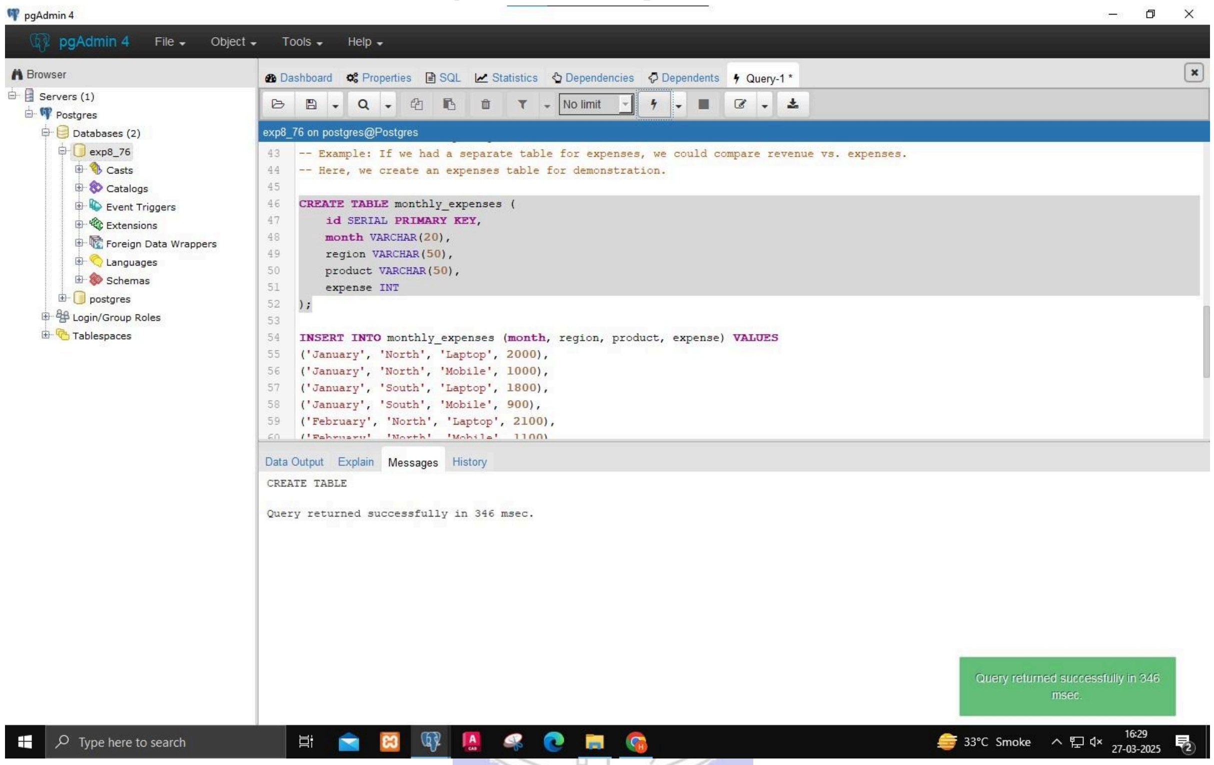
month character varying (20)	region character varying (50)	product character varying (50)	total_sales bigint
February	North	Laptop	5200
February	North	Mobile	3100
February	South	Laptop	4300
February	South	Mobile	2600
January	North	Laptop	5000
January	North	Mobile	3000
January	South	Laptop	4000
January	South	Mobile	2500
March	East	Laptop	4800
March	East	Mobile	2900

Total query runtime: 597 msec.
12 rows retrieved.

-- 4. Drill-Across (Comparing Across Different Tables)

```
-- Example: If we had a separate table for expenses, we could
     compare revenue vs. expenses.
-- Here, we create an expenses table for demonstration.

CREATE TABLE monthly_expenses (
    id SERIAL PRIMARY KEY,
    month VARCHAR(20),
    region VARCHAR(50),
    product VARCHAR(50),
    expense INT
);
```

Creates a table monthly_expenses to store expenses data for different months, regions, and products.



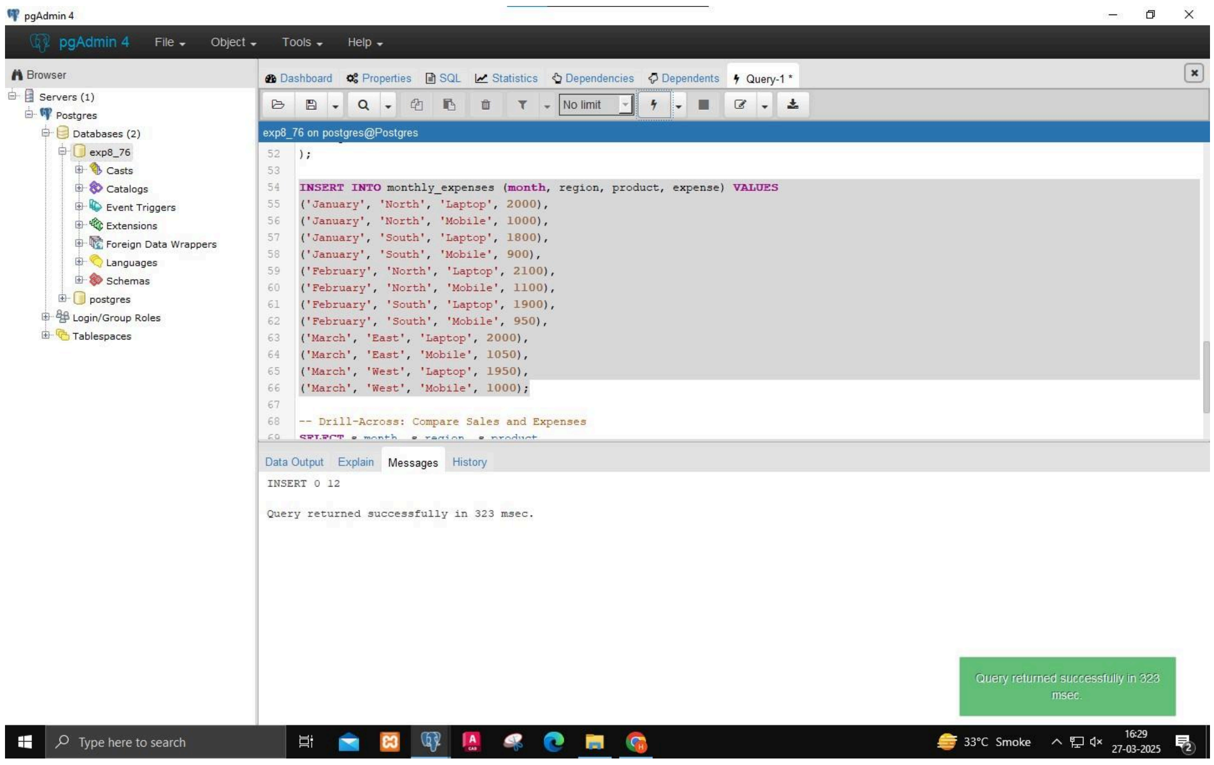
```
-- Example: If we had a separate table for expenses, we could compare revenue vs. expenses.
-- Here, we create an expenses table for demonstration.

CREATE TABLE monthly_expenses (
    id SERIAL PRIMARY KEY,
    month VARCHAR(20),
    region VARCHAR(50),
    product VARCHAR(50),
    expense INT
);

INSERT INTO monthly_expenses (month, region, product, expense) VALUES
('January', 'North', 'Laptop', 2000),
('January', 'North', 'Mobile', 1000),
('January', 'South', 'Laptop', 1800),
('January', 'South', 'Mobile', 900),
('February', 'North', 'Laptop', 2100),
```

```
CREATE TABLE

Query returned successfully in 346 msec.
```

Query returned successfully in 346 msec.

```
INSERT INTO monthly_expenses (month, region, product, expense)
     VALUES
('January', 'North', 'Laptop', 2000),
('January', 'North', 'Mobile', 1000),
('January', 'South', 'Laptop', 1800),
('January', 'South', 'Mobile', 900),
('February', 'North', 'Laptop', 2100),
('February', 'North', 'Mobile', 1100),
('February', 'South', 'Laptop', 1900),
('February', 'South', 'Mobile', 950),
('March', 'East', 'Laptop', 2000),
('March', 'East', 'Mobile', 1050),
('March', 'West', 'Laptop', 1950),
('March', 'West', 'Mobile', 1000);
```

Inserts expense data for the same months, regions, and products, allowing us to compare revenue vs. expenses.

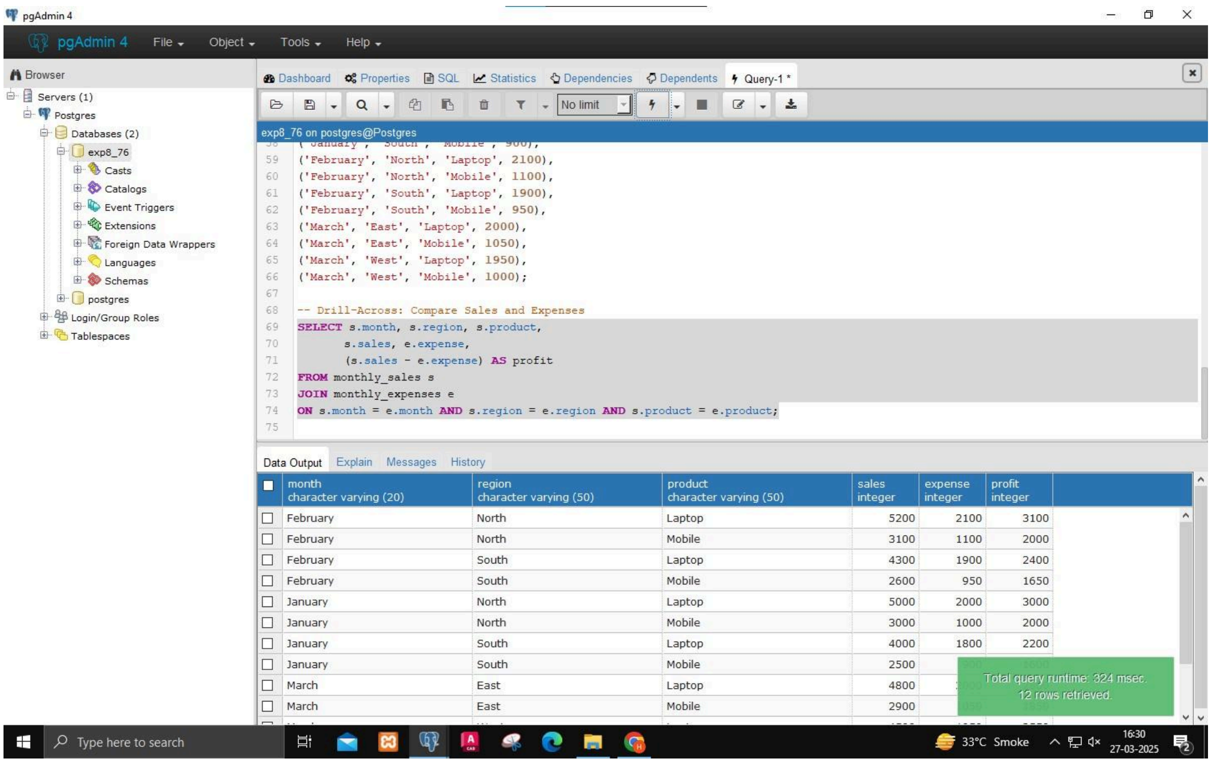


```
);

INSERT INTO monthly_expenses (month, region, product, expense) VALUES
('January', 'North', 'Laptop', 2000),
('January', 'North', 'Mobile', 1000),
('January', 'South', 'Laptop', 1800),
('January', 'South', 'Mobile', 900),
('February', 'North', 'Laptop', 2100),
('February', 'North', 'Mobile', 1100),
('February', 'South', 'Laptop', 1900),
('February', 'South', 'Mobile', 950),
('March', 'East', 'Laptop', 2000),
('March', 'East', 'Mobile', 1050),
('March', 'West', 'Laptop', 1950),
('March', 'West', 'Mobile', 1000);

-- Drill-Across: Compare Sales and Expenses
```

```
INSERT 0 12

Query returned successfully in 323 msec.
```

```
-- Drill-Across: Compare Sales and Expenses
SELECT s.month, s.region, s.product,
       s.sales, e.expense,
       (s.sales - e.expense) AS profit
FROM monthly_sales s
JOIN monthly_expenses e
ON s.month = e.month AND s.region = e.region AND s.product =
     e.product;
```

Combines sales and expenses data to calculate the profit for each month, region, and product.



exp8_76 on postgres@Postgres

```
('January', 'South', 'Mobile', 900),
('February', 'North', 'Laptop', 2100),
('February', 'North', 'Mobile', 1100),
('February', 'South', 'Laptop', 1900),
('February', 'South', 'Mobile', 950),
('March', 'East', 'Laptop', 2000),
('March', 'East', 'Mobile', 1050),
('March', 'West', 'Laptop', 1950),
('March', 'West', 'Mobile', 1000);

-- Drill-Across: Compare Sales and Expenses
SELECT s.month, s.region, s.product,
       s.sales, e.expense,
       (s.sales - e.expense) AS profit
FROM monthly_sales s
JOIN monthly_expenses e
ON s.month = e.month AND s.region = e.region AND s.product = e.product;
```

month character varying (20)	region character varying (50)	product character varying (50)	sales integer	expense integer	profit integer
February	North	Laptop	5200	2100	3100
February	North	Mobile	3100	1100	2000
February	South	Laptop	4300	1900	2400
February	South	Mobile	2600	950	1650
January	North	Laptop	5000	2000	3000
January	North	Mobile	3000	1000	2000
January	South	Laptop	4000	1800	2200
January	South	Mobile	2500	900	1600
March	East	Laptop	4800	2000	2800
March	East	Mobile	2900	1050	1850

Total query runtime: 324 msec.
12 rows retrieved.

Questions:

1. **Elaborate on the operations applied and results generated to your dataset**

In this experiment, we applied various OLAP operations on the monthly_sales dataset to analyze sales trends across different months, regions, and products.

ROLLUP: This operation generated hierarchical aggregation, providing subtotals at the product, region, and month levels, along with a grand total. This helps in trend analysis across different dimensions.

CUBE: Unlike ROLLUP, this operation computed all possible aggregations, allowing us to view sales summaries grouped by any combination of month, region, and product. This offers a comprehensive multidimensional view of the dataset.

Drill-down: By grouping only by month, region, and product, we obtained the detailed, unaggregated view of sales, which helps in granular analysis of trends.

Drill-across: We introduced a monthly_expenses table and joined it with monthly_sales to calculate the profit for each month, region, and product. This provided a comparative analysis of revenue versus cost.

2. **Explain if Drill-down, Drill-across can be applied in relational database, Justify with a query implementation.**

Yes, Drill-down and Drill-across can be applied in a relational database.

- Drill-down is performed by applying more specific GROUP BY conditions to analyze data at a finer level.
- Drill-across is applied when we need to combine multiple fact tables (e.g., Sales and Expenses) to derive insights across datasets.

Drill-down Example:

To analyze sales data at a more detailed level (Month → Region → Product), we use:

```
SELECT month, region, product, SUM(sales) AS total_sales
FROM monthly_sales
GROUP BY month, region, product
ORDER BY month, region, product;
```

This breaks down total sales per product within each region and month.

Drill-across Example:

To compare sales and expenses and compute profit, we use:

```
SELECT s.month, s.region, s.product,
       s.sales, e.expense,
       (s.sales - e.expense) AS profit
FROM monthly_sales s
JOIN monthly_expenses e
ON s.month = e.month AND s.region = e.region AND s.product
= e.product;
```

This joins two fact tables and enables cross-analysis of revenue vs. cost.

Thus, relational databases support Drill-down and Drill-across operations using GROUP BY, JOINs, and aggregations, making them effective for OLAP-style analysis.

Outcomes: CO4: Apply ETL processing, Multidimensional analysis and Online Analytical Processing on the Data Warehouse.

Conclusion: (Conclusion to be based on the outcomes achieved)
From this experiment, I learned how OLAP operations such as ROLLUP, CUBE, Drill-down, and Drill-across can be applied in a relational database to analyze multidimensional data. These operations enable different levels of aggregation and cross-analysis, helping in better decision-making and trend discovery within a dataset.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

- Paulraj Ponniah, “Data Warehousing: Fundamentals for IT Professionals”, Wiley India