Hands-on Lab: Querying the Data Warehouse (Cubes, Rollups, Grouping Sets and Materialized Views)

Estimated time needed: 30 minutes

Purpose of the Lab:

The purpose of this lab is to provide hands-on experience in advanced SQL query techniques using PostgreSQL in a Cloud IDE environment. The lab focuses on teaching how to create grouping sets, rollups, and cubes for data aggregation and summarization, as well as how to implement and utilize Materialized Query Tables (Materialized views) for efficient data querying. These skills are essential for managing and analyzing large datasets, particularly in data warehousing and business intelligence contexts.

Benefits of Learning the Lab:

By completing this lab, you will gain valuable insights into the practical application of complex SQL queries and data manipulation techniques. The knowledge of grouping sets, rollups, and cubes will enable learners to effectively summarize and analyze data, which is crucial in making informed business decisions. Understanding and implementing Materialized views provides an efficient way to handle large-scale data by reducing the computational load during frequent query executions. These skills are highly beneficial for careers in data analysis, database administration, and business intelligence, enhancing your ability to manage and analyze data in real-world scenarios.

Objectives

In this lab you will learn how to create:

- Grouping sets
- Rollup
- Cube
- Materialized views

Exercise 1 - Launch a PostgreSQL server instance on Cloud IDE and open up the pgAdmin Graphical User Interface

This lab requires that you complete the previous lab Populate a Data Warehouse.

If you have not finished the Populate a Data Warehouse Lab yet, please finish it before you continue.

GROUPING SETS, CUBE, and ROLLUP allow us to easily create subtotals and grand totals in a variety of ways. All these operators are used along with the GROUP BY operator.

GROUPING SETS operator allows us to group data in a number of different ways in a single SELECT statement.

The **ROLLUP** operator is used to create subtotals and grand totals for a set of columns. The summarized totals are created based on the columns passed to the ROLLUP operator.

The **CUBE** operator produces subtotals and grand totals. In addition, it produces subtotals and grand totals for every permutation of the columns provided to the CUBE operator.

Exercise 2 - Write a query using grouping sets

After you launch a PostgreSQL server instance on Cloud IDE and open up the pgAdmin Graphical User Interface run the below query.

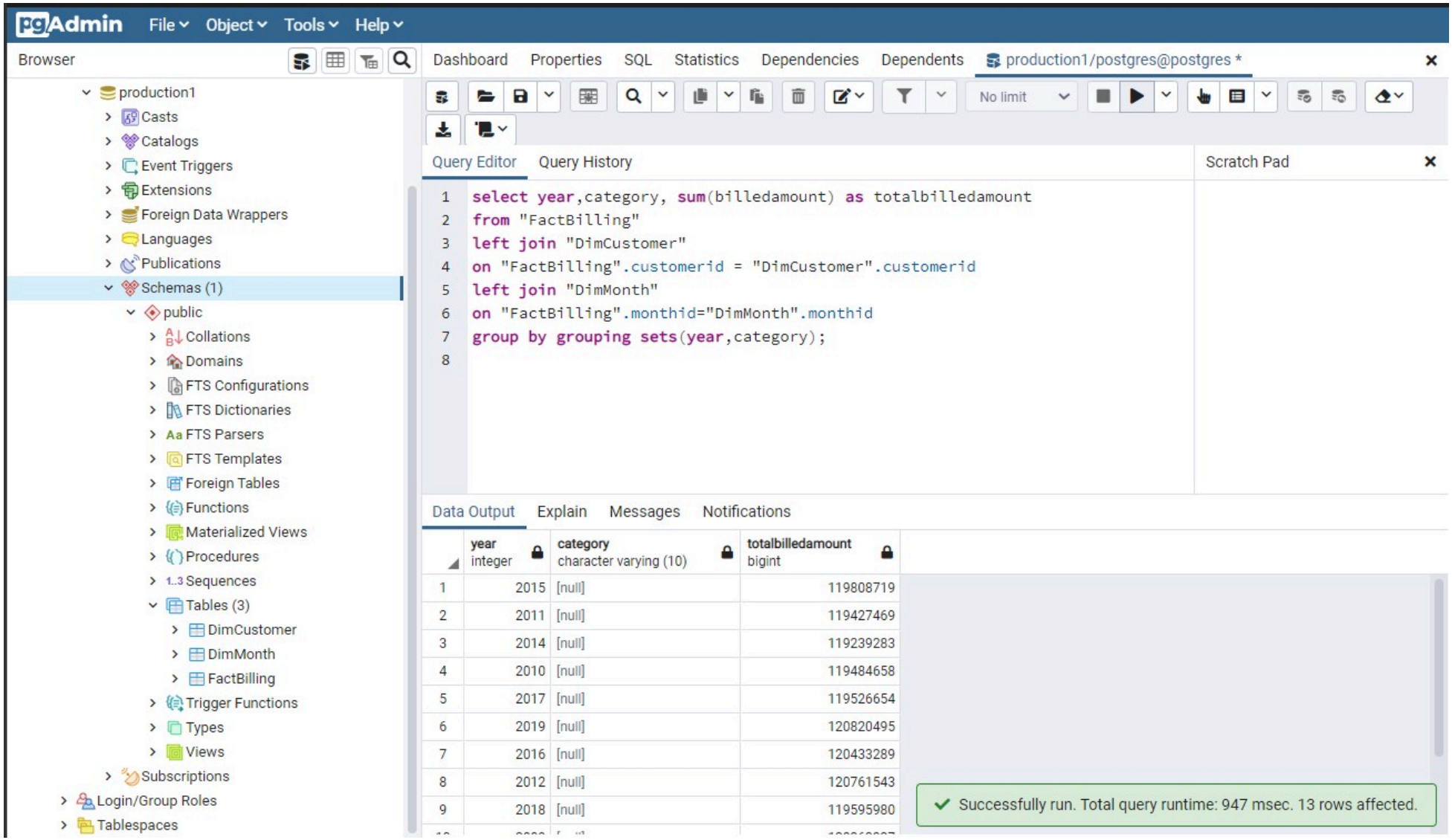
To create a grouping set for three columns labeled year, category, and sum of billedamount, run the sql statement below.

```
1
2
3
4
5
6
7
```

```
select year,category, sum(billedamount) as totalbilledamount
from "FactBilling"
left join "DimCustomer"
on "FactBilling".customerid = "DimCustomer".customerid
left join "DimMonth"
on "FactBilling".monthid="DimMonth".monthid
group by grouping sets(year,category);
```

Copied!

The partial output can be seen in the image below.



Exercise 3 - Write a query using rollup

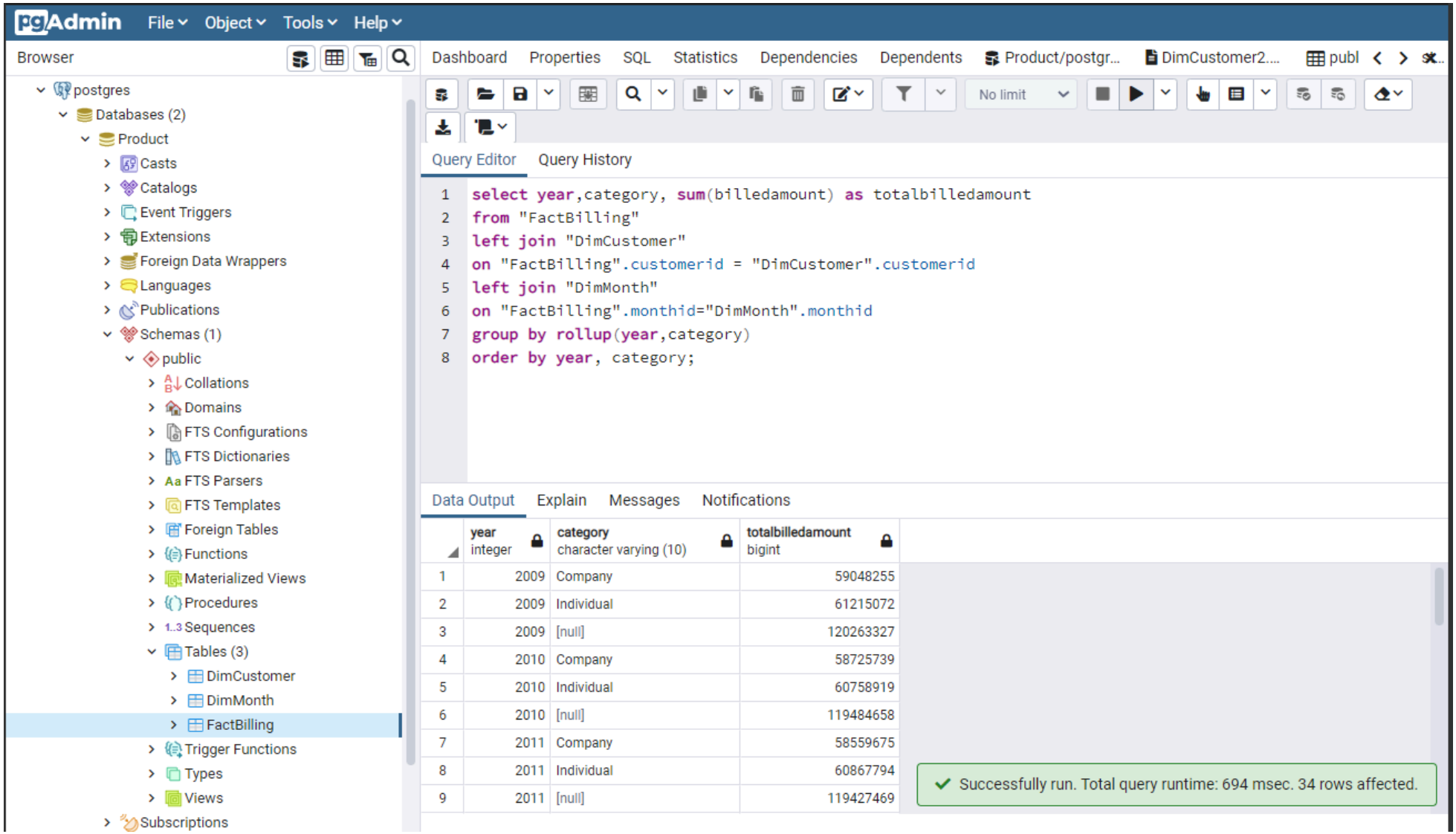
To create a rollup using the three columns year, category and sum of billedamount, run the sql statement below.

1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8

```
1. select year,category, sum(billedamount) as totalbilledamount
2. from "FactBilling"
3. left join "DimCustomer"
4. on "FactBilling".customerid = "DimCustomer".customerid
5. left join "DimMonth"
6. on "FactBilling".monthid="DimMonth".monthid
7. group by rollup(year,category)
8. order by year, category;
```

Copied!

The partial output can be seen in the image below.



Exercise 4 - Write a query using cube

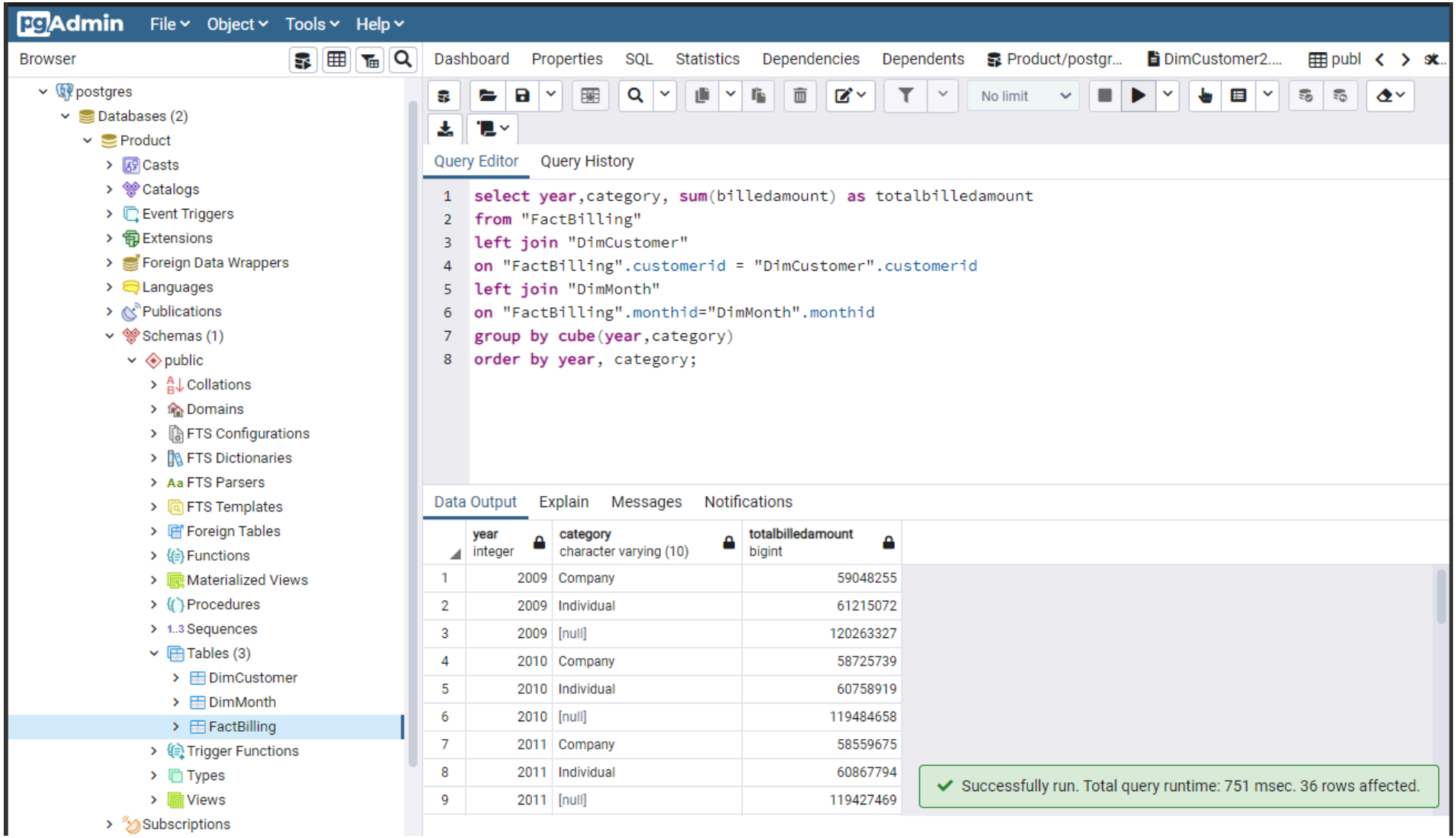
To create a cube using the three columns labeled year, category, and sum of billedamount, run the sql statement below.

```
1
2
3
4
5
6
7
8
```

```
select year,category, sum(billedamount) as totalbilledamount
from "FactBilling"
left join "DimCustomer"
on "FactBilling".customerid = "DimCustomer".customerid
left join "DimMonth"
on "FactBilling".monthid="DimMonth".monthid
group by cube(year,category)
order by year, category;
```

Copied!

The partial output can be seen in the image below.



Exercise 5 - Create a Materialized Query Table(Materialized views)

In pgAdmin we can implement materialized views using Materialized Query Tables.

Step 1: Create the Materialized views.

Execute the sql statement below to create an Materialized views named countrystats.

```
1
2
3
```

```
4. 4
5. 5
6. 6
7. 7
8. 8
```

```
1. CREATE MATERIALIZED VIEW countrystats (country, year, totalbilledamount) AS
2. (select country, year, sum(billedamount)
3. from "FactBilling"
4. left join "DimCustomer"
5. on "FactBilling".customerid = "DimCustomer".customerid
6. left join "DimMonth"
7. on "FactBilling".monthid="DimMonth".monthid
8. group by country,year);
```

Copied!

The above command creates an Materialized views named countrystats that has 3 columns.

- Country
- Year
- totalbilledamount

The Materialized views is essentially the result of the below query, which gives you the year, quartername and the sum of billed amount grouped by year and quartername.

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. select year, quartername, sum(billedamount) as totalbilledamount
2. from "FactBilling"
3. left join "DimCustomer"
4. on "FactBilling".customerid = "DimCustomer".customerid
5. left join "DimMonth"
6. on "FactBilling".monthid="DimMonth".monthid
7. group by grouping sets(year, quartername);
```

Copied!

Step 2: Populate/refresh data into the Materialized views.

Execute the sql statement below to populate the Materialized views countrystats.

```
1. 1
```

```
1. REFRESH MATERIALIZED VIEW countrystats;
```

Copied!

The command above populates the Materialized views with relevant data.

Step 3: Query the Materialized views.

Once an Materialized views is refreshed, you can query it.

Execute the sql statement below to query the Materialized views countrystats.

```
1. 1
```

```
1. select * from countrystats;
```

Copied!

Practice exercises

Problem 1: Create a grouping set for the columns year, quartername, sum(billedamount).

▶ Click here for Hint
▼ Click here for Solution

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
```

```
1. select year, quartername, sum(billedamount) as totalbilledamount
2. from "FactBilling"
3. left join "DimCustomer"
4. on "FactBilling".customerid = "DimCustomer".customerid
5. left join "DimMonth"
6. on "FactBilling".monthid="DimMonth".monthid
7. group by grouping sets(year, quartername);
```

Copied!

Problem 2: Create a rollup for the columns country, category, sum(billedamount).

▶ Click here for Hint
▼ Click here for Solution

```
1. 1
2. 2
3. 3
4. 4
5. 5
6. 6
7. 7
8. 8
```

```
1. SELECT country, category, SUM(billedamount) AS totalbilledamount
2. FROM "FactBilling"
3. LEFT JOIN "DimCustomer"
4. ON "FactBilling".customerid = "DimCustomer".customerid
5. LEFT JOIN "DimMonth"
6. ON "FactBilling".monthid = "DimMonth".monthid
7. GROUP BY ROLLUP(country, category)
8. ORDER BY country, category;
```

Copied!

Problem 3: Create a cube for the columns year,country, category, sum(billedamount).

▶ Click here for Hint
▼ Click here for Solution

```
1
2
3
4
5
6
7
```

```
SELECT year, country, category, SUM(billedamount) AS totalbilledamount
FROM "FactBilling"
LEFT JOIN "DimCustomer"
ON "FactBilling".customerid = "DimCustomer".customerid
LEFT JOIN "DimMonth"
ON "FactBilling".monthid = "DimMonth".monthid
GROUP BY CUBE(year, country, category);
```

Copied!

Problem 4: Create an Materialized views named average_billamount with columns year, quarter, category, country, average_bill_amount.

▶ Click here for Hint
▼ Click here for Solution

```
1
2
3
4
5
6
7
8
9
```

```
CREATE MATERIALIZED VIEW average_billamount (year,quarter,category,country, average_bill_amount) AS
    (select   year,quarter,category,country, avg(billedamount) as average_bill_amount
    from "FactBilling"
    left join  "DimCustomer"
    on "FactBilling".customerid =  "DimCustomer".customerid
    left join "DimMonth"
    on "FactBilling".monthid="DimMonth".monthid
    group by year,quarter,category,country
    );
```

Copied!

```
1
```

```
refresh MATERIALIZED VIEW average_billamount;
```

Copied!

Congratulations! You have successfully finished the Querying the Data Warehouse (Cubes, Rollups, Grouping Sets and Materialized Views) lab.

Author

Amrutha Rao