

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



Estimated time needed: 30 minutes

Objectives

In this lab you will learn how to create:

- Grouping sets
- Rollup
- Cube
- Materialized Query Tables (MQT)

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

```
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 grouping sets(year,category);
```

Copied!

The partial output can be seen in the image below.

The screenshot shows the pgAdmin 4 interface. On the left is the 'Browser' pane showing the database structure. The 'production1' database is expanded, showing 'Schemas (1)' with 'public' schema. Under 'public', 'Tables (3)' are listed: 'DimCustomer', 'DimMonth', and 'FactBilling'. The 'Query Editor' pane in the center contains the following SQL query:

```
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 grouping sets(year,category);
8
```

Below the query editor is the 'Data Output' pane, which displays the results of the query in a table format:

	year	category	totalbilledamount
	integer	character varying (10)	bigint
1	2015	[null]	119808719
2	2011	[null]	119427469
3	2014	[null]	119239283
4	2010	[null]	119484658
5	2017	[null]	119526654
6	2019	[null]	120820495
7	2016	[null]	120433289
8	2012	[null]	120761543
9	2018	[null]	119595980

A green status bar at the bottom right of the Data Output pane indicates: 'Successfully run. Total query runtime: 947 msec. 13 rows affected.'

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.

The screenshot shows the pgAdmin interface with a SQL query editor and its output. The query is as follows:

```
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;
```

The output table shows the following data:

	year	category	totalbilledamount
1	2009	Company	59048255
2	2009	Individual	61215072
3	2009	[null]	120263327
4	2010	Company	58725739
5	2010	Individual	60758919
6	2010	[null]	119484658
7	2011	Company	58559675
8	2011	Individual	60867794
9	2011	[null]	119427469

A green message box at the bottom right states: "Successfully run. Total query runtime: 694 msec. 34 rows affected."

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. 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 cube(year,category)
8. order by year, category;
```

Copied!

The partial output can be seen in the image below.

The screenshot shows the pgAdmin interface. On the left is the 'Browser' pane showing a tree view of the database structure: postgres > Databases (2) > Product > FactBilling. The 'Query Editor' is open, displaying the following SQL query:

```

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 cube(year,category)
8 order by year, category;

```

The 'Data Output' tab shows the results of the query:

	year integer	category character varying (10)	totalbilledamount bigint
1	2009	Company	59048255
2	2009	Individual	61215072
3	2009	[null]	120263327
4	2010	Company	58725739
5	2010	Individual	60758919
6	2010	[null]	119484658
7	2011	Company	58559675
8	2011	Individual	60867794
9	2011	[null]	119427469

A green message box at the bottom right states: 'Successfully run. Total query runtime: 751 msec. 36 rows affected.'

Exercise 5 - Create a Materialized Query Table(MQT)

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

Step 1: Create the MQT.

Execute the sql statement below to create an MQT named countrystats.

```

1. 1
2. 2
3. 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 MQT named countrystats that has 3 columns.

- Country
- Year
- totalbilledamount

The MQT 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 MQT.

Execute the sql statement below to populate the MQT countrystats.

```

1. 1

1. REFRESH MATERIALIZED VIEW countrystats;

```

Copied!

The command above populates the MQT with relevant data.

Step 3: Query the MQT.

Once an MQT is refreshed, you can query it.

Execute the sql statement below to query the MQT 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 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 rollup(year, quartername)
8. order by year, quartername;
```

Copied!

Problem 3: Create a cube for the columns year,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

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 cube(year,quartername);
```

Copied!

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

- Click here for Hint
- ▼ Click here for Solution

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

1. CREATE MATERIALIZED VIEW average_billamount (year,quarter,category,country, average_bill_amount) AS
2. (select year,quarter,category,country, avg(billedamount) as average_bill_amount
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 year,quarter,category,country
9. );
```

Copied!

```
1. 1
1. refresh MATERIALIZED VIEW average_billamount;
```

Copied!

Congratulations! You have successfully finished the Populating a Data Warehouse lab.

Author

Amrutha Rao

Change Log

Date (YYYY-MM-DD)	Version	Changed By	Change Description
2022-04-14	0.2	Amrutha Rao	converted initial version to pgAdmin workaround.
2021-09-29	0.1	Ramesh Sannareddy	Created initial version of the lab

© IBM Corporation 2022. All rights reserved.