Database Advance Job Sheet-8: Pivoting dan Grouping Sets

From:

AL AZHAR RIZQI RIFA'I FIRDAUS

Class:

2 I

Absence:

01

Student Number Identity:

2241720263

Department:

Information Technology

Study Program:

Informatics Engineering

Practicum 1

1. From the Sales.CustGroups view that has been created, create a SELECT query to display the custid, custgroup, and country columns.

minggu8 > query > num1.sql

```
1  -- CREATE VIEW Sales.CustGroups
2  -- AS
3  --     SELECT custid, CHOOSE(custid % 3 + 1, N'A', N'B', N'C')
   AS custgroup, country
4  --     FROM Sales.Customers;
5
6  SELECT * FROM Sales.CustGroups;
```

RESULTS

	custid	custgroup	country
1	1	B	Germany
2	2	C	Mexico
3	3	A	Mexico
4	4	B	UK
5	5	C	Sweden
6	6	A	Germany
7	7	B	France
8	8	C	Spain
9	9	A	France
10	10	B	Canada

2. Modify the T-SQL code from step 3 above to display the country column, then using the PIVOT operator, add 3 additional columns containing the number of customers in each group (A, B, & C).

minggu8 > query > num2.sql

```sql
SELECT country,[A],[B],[C]
FROM (
    SELECT custid, custgroup, country
    FROM Sales.CustGroups
) AS D
PIVOT (
    COUNT(custid)
    FOR custgroup IN ([A],[B],[C])
) AS pvt;
```

RESULTS

	country	A	B	C
1	Argentina	2	1	0
2	Austria	0	0	2
3	Belgium	0	1	1
4	Brazil	3	5	1
5	Canada	2	1	0
6	Denmark	0	1	1
7	Finland	2	0	0
8	France	4	3	4
9	Germany	3	4	4
10	Ireland	0	1	0

Practicum 2

3. Copy the SELECT statement from Problem 2 above, then run it again. Is the result of of this query the same as the result in Practicum Part 1 no 4 above? Is the number of rows generated exactly the same?

```
minggu8 > query > num3.sql
1   -- ALTER VIEW Sales.CustGroups
2   -- AS
3   --      SELECT custid, CHOOSE(custid % 3 + 1, N'A', N'B', N'C')
    AS custgroup, country, city, contactname
4   --      FROM Sales.Customers;
5
6   SELECT country,[A],[B],[C]
7   FROM (
8       SELECT custid, custgroup, country
9       FROM Sales.CustGroups
10  ) AS D
11  PIVOT (
12      COUNT(custid)
13      FOR custgroup IN ([A],[B],[C])
14  ) AS pvt;
```

RESULTS

	country	A	B	C
1	Argentina	2	1	0
2	Austria	0	0	2
3	Belgium	0	1	1
4	Brazil	3	5	1
5	Canada	2	1	0
6	Denmark	0	1	1
7	Finland	2	0	0
8	France	4	3	4
9	Germany	3	4	4
10	Ireland	0	1	0

- The result between both is same.

4. Modify the SELECT statement to add city and contactname columns!

minggu8 > query > num4.sql

```sql
SELECT country, city, contactname, [A],[B],[C]
FROM Sales.CustGroups
PIVOT (
    COUNT(custid)
    FOR custgroup IN ([A],[B],[C])
) AS pvt;
```

RESULTS

	country	city	contactname	A	B	C
1	Argentina	Buenos Aires	Gaffney, Lawrie	0	1	0
2	Argentina	Buenos Aires	Ray, Mike	1	0	0
3	Argentina	Buenos Aires	Tiano, Mike	1	0	0
4	Austria	Graz	Kane, John	0	0	1
5	Austria	Salzburg	Meston, Tosh	0	0	1
6	Belgium	Bruxelles	Mace, Donald	0	0	1
7	Belgium	Charleroi	Gulbis, Katrin	0	1	0
8	Brazil	Campinas	Cheng, Yao-Qi...	0	1	0
9	Brazil	Resende	Li, Yan	0	1	0
10	Brazil	Rio de Janeiro	Cohen, Shy	0	1	0

Practicum 3

5. Create a CTE called PivotCustGroups that gets the custid, country, and custgroup columns from the Sales.CustGroups view. Then, create a SELECT query against the CTE and use the PIVOT operator, just like in the SELECT query in the previous Practicum Section before.

minggu8 > query > num5.sql

```sql
WITH PivotCustGroups AS (
    SELECT custid, country, custgroup
    FROM Sales.CustGroups
)

SELECT country, [A],[B],[C]
FROM (
    SELECT custid, custgroup, country
    FROM PivotCustGroups
) AS D

PIVOT (
    COUNT(custid)
    FOR custgroup IN ([A],[B],[C])
) AS pvt;
```

RESULTS

	country	A	B	C
1	Argentina	2	1	0
2	Austria	0	0	2
3	Belgium	0	1	1
4	Brazil	3	5	1
5	Canada	2	1	0
6	Denmark	0	1	1
7	Finland	2	0	0
8	France	4	3	4
9	Germany	3	4	4
10	Ireland	0	1	0

6. Are the results exactly the same as those in Practicum Part 1? Why is that?

- The result is same, but the method that used is different.

7. What are the advantages of using CTE when creating a query that uses the PIVOT operator?

- Using Common Table Expressions (CTEs) when creating queries with the PIVOT operator provides benefits such as cleaner code, improved performance, reusability, easier debugging,

and better code management. CTEs allow you to separate data processing logic from the main PIVOT query, making your code more readable and efficient.

Practicum 4

8. Create a SELECT query that displays the total sales amount data for each product category, for each customer. Display each product category in its own column, as shown below.

minggu8 > query > num8.sql

```
1  WITH SalesByCategory AS (
2      SELECT o.custid, SUM(od.unitprice * od.qty) AS SalesValue, categoryname
3      FROM Sales.Orders AS o
4      JOIN Sales.OrderDetails AS od ON o.orderid = od.orderid
5      JOIN Production.Products AS p ON od.productid = p.productid
6      JOIN Production.Categories AS c ON p.categoryid = c.categoryid
7      WHERE YEAR(o.orderdate) = 2008
8      GROUP BY o.custid, categoryname
9  )
10
11 SELECT custid, [Beverages], [Condiments], [Confections], [Dairy Products],
   [Grains/Cereals], [Meat/Poultry], [Produce], [Seafood]
12 FROM (
13     SELECT * FROM SalesByCategory
14 ) AS p
15
16 PIVOT (
17     SUM(SalesValue)
18     FOR categoryname IN ([Beverages], [Condiments], [Confections], [Dairy
       Products], [Grains/Cereals], [Meat/Poultry], [Produce], [Seafood])
19 ) AS pvt;
```

RESULTS

	custid	Beverages	Condiments	Confections	Dairy Products	Grains/Cereals	Meat/Poultry	Produce	Seafood
1	1	NULL	426.00	NULL	1255.00	NULL	NULL	91.20	530.00
2	2	NULL	NULL	64.40	390.00	NULL	NULL	NULL	60.00
3	3	380.00	NULL	NULL	NULL	280.00	NULL	NULL	NULL
4	4	282.00	NULL	4440.00	812.50	NULL	NULL	NULL	304.00
5	5	850.50	300.00	2202.55	NULL	NULL	1237.90	1368.00	2151.60
6	6	NULL	114.00	283.00	714.00	NULL	NULL	424.00	625.00
7	7	NULL	NULL	NULL	437.50	292.50	NULL	NULL	NULL
8	8	NULL	NULL	NULL	NULL	280.00	NULL	NULL	NULL
9	9	533.00	1750.00	1515.10	556.80	665.00	624.00	705.00	837.00
10	10	1706.50	1290.10	4518.30	992.50	684.00	234.00	1872.00	930.00
11	11	1380.00	NULL	NULL	220.00	441.00	NULL	120.00	270.00
12	12	1037.00	NULL	NULL	25.00	NULL	NULL	364.80	150.00
13	14	570.00	1843.80	591.60	NULL	1216.00	NULL	NULL	208.00
14	15	NULL	405.75	NULL	NULL	NULL	NULL	NULL	108.00

Practicum 5

9. Construct a SELECT query that returns the country column, column A, column B, and column C from the Sales.PivotCustGroups view that was created.

minggu8 > query > num9.sql

```sql
-- CREATE VIEW Sales.PivotCustGroups AS WITH PivotCustGroups AS
(
--     SELECT custid, country, custgroup
--     FROM Sales.CustGroups
-- )

-- SELECT country, p.A, p.B, p.C
-- FROM PivotCustGroups
-- PIVOT (
--     COUNT(custid)
--     FOR custgroup IN (
--         A, B, C
--     )
-- ) AS p;

SELECT country, A, B, C
FROM Sales.PivotCustGroups;
```

RESULTS

	country	A	B	C
1	Argentina	2	1	0
2	Austria	0	0	2
3	Belgium	0	1	1
4	Brazil	3	5	1
5	Canada	2	1	0
6	Denmark	0	1	1
7	Finland	2	0	0
8	France	4	3	4
9	Germany	3	4	4
10	Ireland	0	1	0

Practicum 6

10. Create a SELECT query against the Sales.PivotCustGroups view that generates data like this produces data like the following view:

minggu8 > query > num10.sql

```sql
SELECT custgroup, country, SUM(numberofcustomers) AS
numberofcustomers
FROM Sales.PivotCustGroups
UNPIVOT (
    numberofcustomers
    FOR custgroup IN (
        A, B, C
    )
) AS unpivoted
GROUP BY custgroup, country
ORDER BY country;
```

RESULTS

	custgroup	country	numberofcustomers
1	A	Argentina	2
2	B	Argentina	1
3	C	Argentina	0
4	A	Austria	0
5	B	Austria	0
6	C	Austria	2
7	A	Belgium	0
8	B	Belgium	1
9	C	Belgium	1
10	A	Brazil	3

Practicum 7

11. Create a SELECT query against the Sales.Customers table that consists of the contry column, city, and a calculated column that counts the number of customers called noofcustomers. Get the grouping set based on:

- country and city columns
- country column
- city column
- and a column with no groups

The correct result is shown in the following view:

```
minggu8 > query > num11.sql
1   SELECT country, city, COUNT(custid) AS noofcustomers
2   FROM Sales.Customers
3   GROUP BY GROUPING SETS (
4       (country, city), (country), (city), ()
5   );
```

country	city	noofcustomers
Germany	Aachen	1
NULL	Aachen	1
USA	Albuquerque	1
NULL	Albuquerque	1
USA	Anchorage	1
NULL	Anchorage	1
Spain	Barcelona	1
NULL	Barcelona	1

Practicum 8

12. Create a SELECT query against the Sales.OrderValues view that contains the columns:

- orderyear: year from the orderdate column
- ordermounth: month of the orderdate column
- orderday: day of the orderdate column
- salesvalue: total sales amount from column val

minggu8 > query > num12.sql

```
1  SELECT YEAR(orderdate) AS orderyear, MONTH(orderdate) AS
   ordermonth, DAY(orderdate) AS orderday, SUM(val) AS salesvalue
2  FROM Sales.OrderValues
3  GROUP BY CUBE (
4      YEAR(orderdate),
5      MONTH(orderdate),
6      DAY(orderdate)
7  );
```

RESULTS

	orderyear	ordermonth	orderday	salesvalue
1	2007	1	1	6931.60
2	2008	1	1	1738.00
3	*NULL*	1	1	8669.60
4	2007	4	1	851.20
5	2008	4	1	11549.89
6	*NULL*	4	1	12401.09
7	2007	5	1	5636.96
8	2008	5	1	5448.57
9	*NULL*	5	1	11085.53
10	2007	7	1	142.50

Practicum 9

13. Copy the query from Problem 12 above and change the CUBE subclause to ROLLUP, then run the query. The correct result is shown in the following display:

minggu8 > query > num13.sql

```sql
SELECT YEAR(orderdate) AS orderyear, MONTH(orderdate) AS
ordermonth, DAY(orderdate) AS orderday, SUM(val) AS salesvalue
FROM Sales.OrderValues
GROUP BY ROLLUP (
    YEAR(orderdate),
    MONTH(orderdate),
    DAY(orderdate)
);
```

RESULTS

	orderyear	ordermonth	orderday	salesvalue
1	2006	7	4	440.00
2	2006	7	5	1863.40
3	2006	7	8	2206.66
4	2006	7	9	3597.90
5	2006	7	10	1444.80
6	2006	7	11	556.62
7	2006	7	12	2490.50
8	2006	7	15	517.80
9	2006	7	16	1119.90
10	2006	7	17	1614.88

14. What is the difference between the ROLLUP and CUBE subclauses? Which one is more appropriately used for the query in step 1 above?

1. ROLLUP: ROLLUP creates a result set that represents a hierarchy, producing subtotals for each level of the specified grouping columns. It generates a result set with subtotals at various levels, moving from the most detailed level to the grand total level.
2. CUBE: CUBE, on the other hand, generates a result set with all possible combinations of the specified grouping columns, including subtotals and the grand total. It provides a more comprehensive summary of the data.

   ROLLUP is more suitable for the query as it generates subtotals at different levels of the date hierarchy, whereas CUBE would produce a more extensive set of combinations, which might not be needed for this specific case.

Practicum 10

15. Make a SELECT query against the Sales.OrderValues view and get the following columns:

- calculated column with alias groupid (use GROUPING_ID function with
- order year and order month as input parameters)
- orderyear: year from the orderdate field
- ordermonth: the month of the orderdate field
- salesvalue: total sales value from the val column
- since year and month are hierarchical, get all the grouping sets based on the orderyear and grouping sets based on the orderyear and ordermonth columns, then sort them by groupid, orderyear and month by groupid, orderyear, and ordermonth

The correct result is shown in the following display:

minggu8 > query > num15.sql

```
1  SELECT GROUPING_ID(YEAR(orderdate), MONTH(orderdate)) AS
   groupid, YEAR(orderdate) AS orderyear, MONTH(orderdate) AS
   ordermonth, SUM(val) AS salesvalue
2  FROM Sales.OrderValues
3  GROUP BY ROLLUP (
4      YEAR(orderdate),
5      MONTH(orderdate)
6  )
7  ORDER BY groupid, orderyear, ordermonth;
```

RESULTS

	groupid	orderyear	ordermonth	salesvalue
1	0	2006	7	27861.90
2	0	2006	8	25485.28
3	0	2006	9	26381.40
4	0	2006	10	37515.73
5	0	2006	11	45600.05
6	0	2006	12	45239.63
7	0	2007	1	61258.08
8	0	2007	2	38483.64
9	0	2007	3	38547.23
10	0	2007	4	53032.95