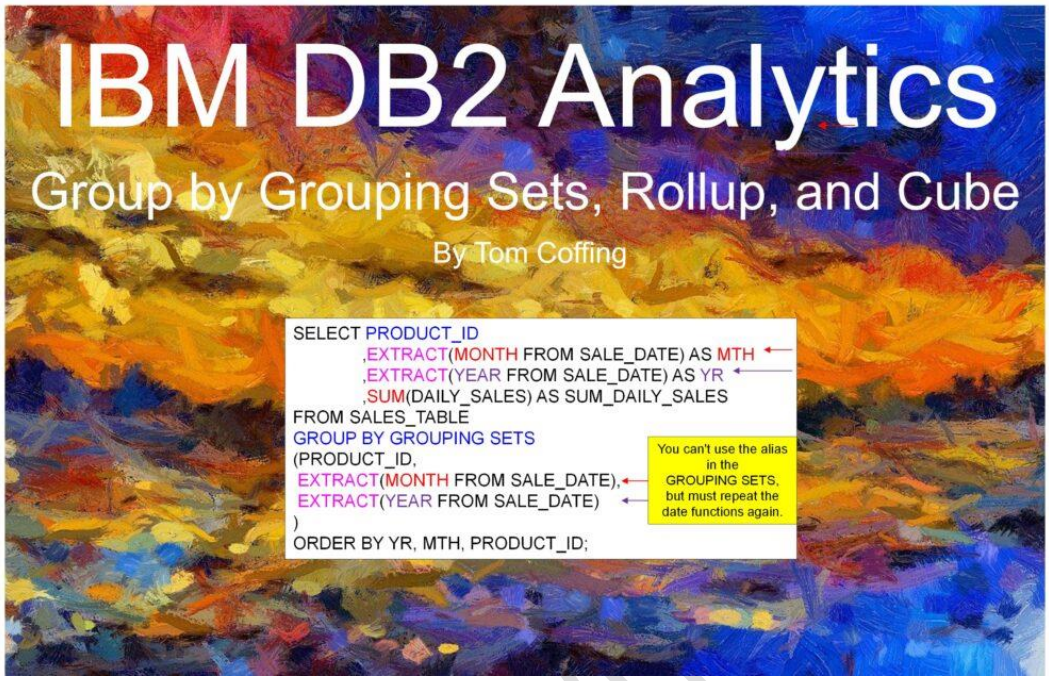
Group By Grouping Sets, Rollup, and Cube

IBM DB2 Analytics – Group By Grouping Sets, Rollup, and Cube

Ref : https://coffingdw.com/ibm-db2-analytics-group-by-grouping-sets-rollup-and-cube/



GROUP BY GROUPING SETS, ROLLUP, and CUBE are instrumental when aggregating data by multiple dimensions in a single query. It avoids the need for numerous separate queries to compute totals for different levels of aggregation.

The **GROUP BY GROUPING SETS** provides basic dimensions, which are taken further by the GROUP BY ROLLUP and then taken to the limit by GROUP BY CUBE.

Normal Group By Statement

Normally, we must perform separate GROUP BY statements to analyze data from different perspectives. Below are two separate queries on the same table to see the data calculated by sum per product_id and month. Watch next as we use advanced techniques to combine groupings.

Group By Grouping Sets, Rollup, and Cube

```
SELECT PRODUCT_ID
      ,SUM(DAILY_SALES) AS SUM_DAILY_SALES
FROM SALES_TABLE
GROUP BY PRODUCT_ID ORDER BY PRODUCT_ID ;
```

PRODUCT_ID	SUM_DAILY_SALES
1000	331204.72
2000	306611.81
3000	224587.82

```
SELECT EXTRACT(MONTH FROM SALE_DATE) AS MTH
      ,SUM(DAILY_SALES) AS SUM_DAILY_SALES
FROM SALES_TABLE
GROUP BY EXTRACT(MONTH FROM SALE_DATE)
ORDER BY MTH;
```

MTH	SUM_DAILY_SALES
9	418769.36
10	443634.99

Group by Grouping Sets

Using **GROUP BY GROUPING SETS** in a single query is advantageous over running three separate queries because it enhances efficiency, simplifies code maintenance, potentially improves performance through optimized execution, and offers flexibility in dynamic grouping. It's a powerful SQL feature for efficiently and effectively summarizing data across multiple dimensions.

Below, we have grouped our data by **product_id, month, and year**. Notice in the top row that we made a yearly total of **862404.35**. The yearly calculation has null values in the **product_id** and **month**.

If you were to add up both months below, you would see that in **September, we made 418769.36**, and in **October, it was 443634.99.** When you **combine** both months, we will be back at **862404.35**. The **month** calculation has **null** values in the **product_id and year.**

If you add up what we made for **product_id 1000, 2000, and 3000,** they **add up** to **862404.35.**

The **product_id** calculations have **null values** in the **month** and **year**.

GROUP BY GROUPING SETS is great when aggregating data by multiple dimensions in a single query.

Group By Grouping Sets, Rollup, and Cube

```
SELECT PRODUCT_ID
      ,EXTRACT(MONTH FROM SALE_DATE) AS MTH
      ,EXTRACT(YEAR FROM SALE_DATE) AS YR
      ,SUM(DAILY_SALES) AS SUM_DAILY_SALES
FROM SALES_TABLE
GROUP BY GROUPING SETS
(PRODUCT_ID,
 EXTRACT(MONTH FROM SALE_DATE),
 EXTRACT(YEAR FROM SALE_DATE)
)
ORDER BY YR, MTH, PRODUCT_ID;
```

You can't use the alias in the GROUPING SETS, but must repeat the date functions again.

PRODUCT_ID	MTH	YR	SUM_DAILY_SALES
?	?	2024	862404.35
?	9	?	418769.36
?	10	?	443634.99
1000	?	?	331204.72
2000	?	?	306611.81
3000	?	?	224587.82

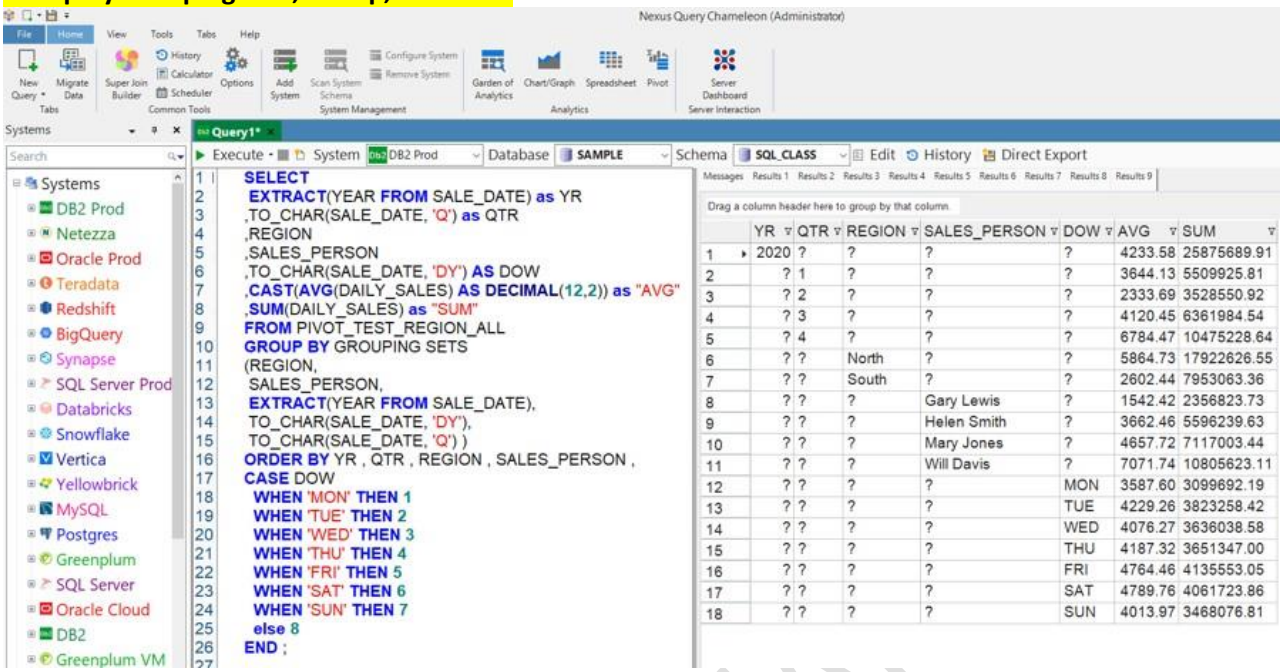
Supercharged GROUP BY GROUPING SETS Example

In the example below, we are grouping the dimensions of **year, quarter, region, salesperson**, and **day of the week.** We also use a case statement in the **ORDER BY** clause at the end of the query, so our data comes out exactly as we want. What a query!

Line two uses the **EXTRACT** command to pull the year from the sale date. In line three, we use the **TO_CHAR** command to get the quarter. In line six, we use the **TO_CHAR** command again to determine the day of the week. Notice that we use the **CAST** command on the daily sales average to get two decimal places.

The query was run on the greatest query tool in the World, Nexus.

Group By Grouping Sets, Rollup, and Cube



GROUP BY ROLLUP

A **GROUP BY ROLLUP** creates a hierarchical data rollup, generating subtotal rows from the most detailed level to the grouping specified. The rollup generates more rows in the result set due to the hierarchical nature, including subtotal rows for each level of aggregation up to the grand total.

The example below shows the **grandtotal** on the first row and the sum of sales for each product, plus product and month for all years, and product and month per year. We would have received more rows had there been multiple years of data.

Some databases allow you to use the alias in the rollup, but we had to repeat the date functions within the rollup.

Group By Grouping Sets, Rollup, and Cube

```
SELECT PRODUCT_ID
      ,EXTRACT(MONTH FROM SALE_DATE) AS MTH
      ,EXTRACT(YEAR FROM SALE_DATE) AS YR
      ,SUM(DAILY_SALES) AS SUM_DAILY_SALES
FROM SALES_TABLE
GROUP BY ROLLUP (PRODUCT_ID,
EXTRACT(MONTH FROM SALE_DATE),
EXTRACT(YEAR FROM SALE_DATE))
ORDER BY YR NULLS FIRST, MTH NULLS FIRST, PRODUCT_ID NULLS FIRST;
```

Extract Function to get the month and year

You can't use the alias in the ROLLUP, but must repeat the date functions again.

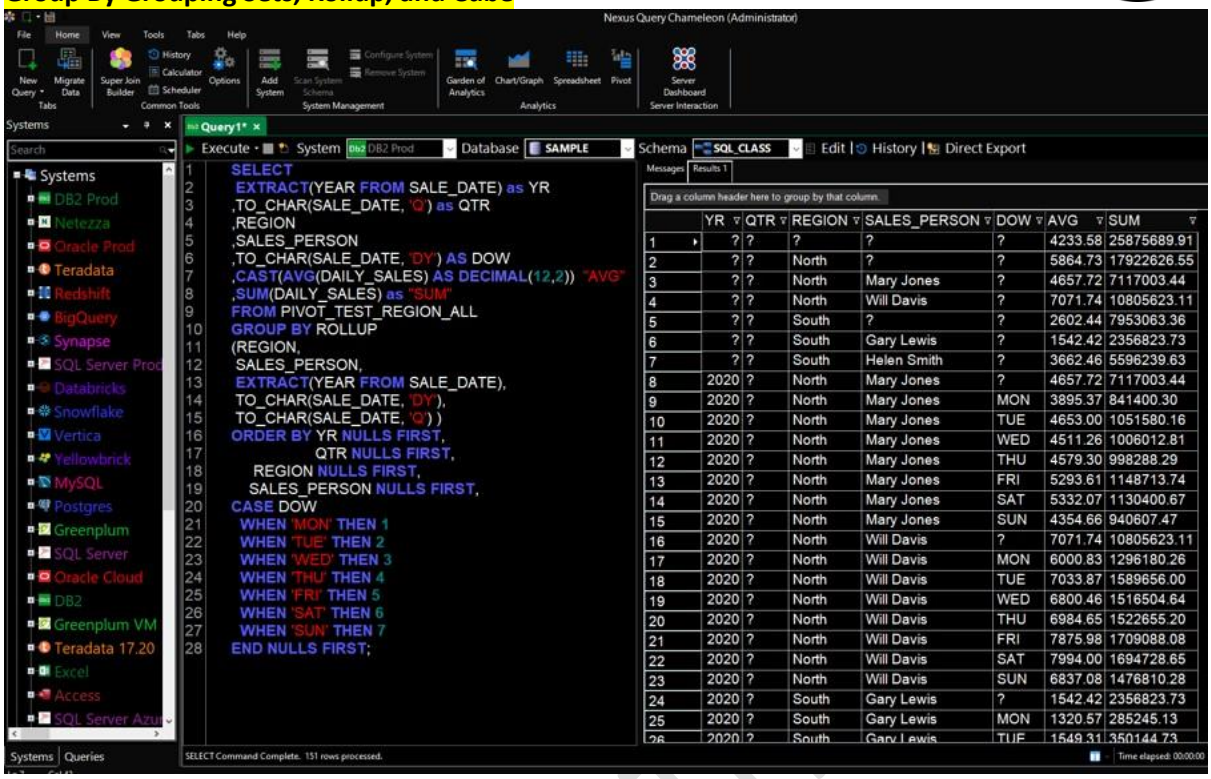
PRODUCT_ID	MTH	YR	SUM_DAILY_SALES
?	?	?	862404.35
1000	?	?	331204.72
2000	?	?	306611.81
3000	?	?	224587.82
1000	9	?	139350.69
2000	9	?	139738.91
3000	9	?	139679.76
1000	10	?	191854.03
2000	10	?	166872.90
3000	10	?	84908.06
1000	9	2024	139350.69
2000	9	2024	139738.91
3000	9	2024	139679.76
1000	10	2024	191854.03
2000	10	2024	166872.90
3000	10	2024	84908.06

Super Duper Charged GROUP BY ROLLUP Example

In the example below, we are grouping the dimensions of year, quarter, region, salesperson, and day of the week. Notice we are doing a **CAST** to change the data type for the average. We also use a case statement in the **ORDER BY** clause at the end of the query, so our data comes out exactly as we want.

Because the answer set returns **151 rows**, I will show it in a table below the example.

Group By Grouping Sets, Rollup, and Cube



Show **10202550100** entries

Showing **1 to 10 of 880 entries**

Show 10 entries Search:

YR	QTR	REGION	SALES_PERSON	DOW	AVG	SUM
					4233.58	25875689.91
				MON	3587.6	3099692.19
				TUE	4229.26	3823258.42
				WED	4076.27	3636038.58
				THU	4187.32	3651347
				FRI	4764.46	4135553.05
				SAT	4789.76	4061723.86
				SUN	4013.97	3468076.81
			Gary Lewis		1542.42	2356823.73
			Gary Lewis	MON	1320.57	285245.13

Showing 1 to 10 of 880 entries Previous Next

Group By Grouping Sets, Rollup, and Cube

Show 20 entries Search:

YR	QTR	REGION	SALES_PERSON	DOW	AVG	SUM
		South	Helen Smith		3662.46	5596239.63
		South	Helen Smith	MON	3133.64	676866.5
		South	Helen Smith	TUE	3680.87	831877.53
		South	Helen Smith	WED	3514.06	783636.43
		South	Helen Smith	THU	3657.64	797367.58
		South	Helen Smith	FRI	4163.51	903483.26
		South	Helen Smith	SAT	4096.3	868417.15
		South	Helen Smith	SUN	3400.88	734591.18
	1				3644.13	5509925.81
	1			MON	4100.21	852845.48
	1			TUE	5242.97	1153454.46
	1			WED	3585.12	788726.63
	1			THU	2076.74	431962.37
	1			FRI	3559.89	783177.1
	1			SAT	3872.05	836364.13
	1			SUN	3015.43	663395.64
	1		Gary Lewis		1310.98	495550.63
	1		Gary Lewis	MON	1486.59	77302.96
	1		Gary Lewis	TUE	1870.72	102889.79
	1		Gary Lewis	WED	1267.85	69732.13

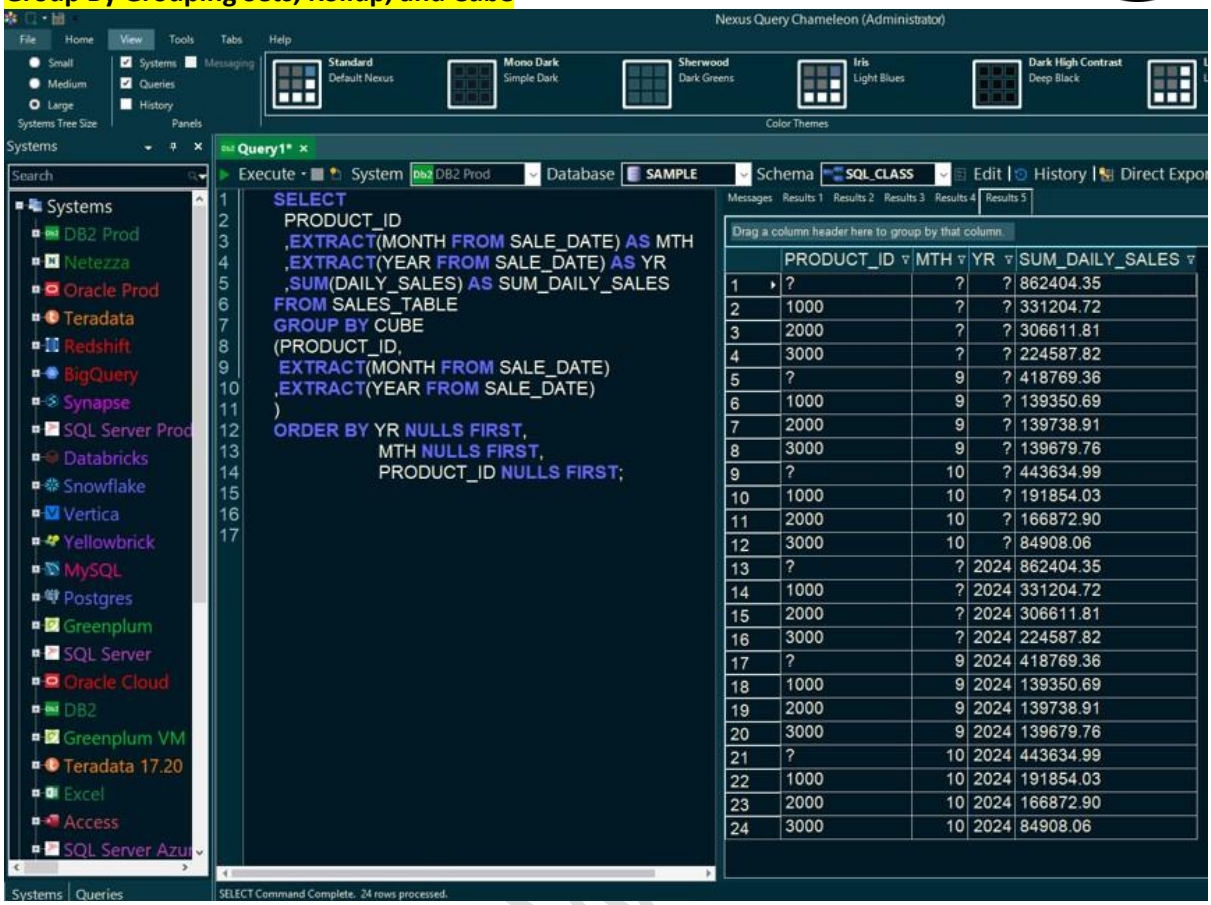
Showing 81 to 100 of 880 entries Previous Next

GROUP BY CUBE

A **GROUP BY CUBE** creates a hierarchical data rollup, generating more detailed subtotal rows than the ROLLUP. The Cube generates more rows in the result set due to the hierarchical nature, including subtotal rows for each level of aggregation up to the grand total.

The example below shows the grandtotal on the first row and the sum of sales for each product for all years and each year, product and month for all years, product and month per year, and each month for all years. We would have received more rows had there been multiple years of data.

Group By Grouping Sets, Rollup, and Cube



DB2 Delicious GROUP BY CUBE Example

In the example below, we are grouping the dimensions of year, quarter, region, salesperson, and day of the week. Notice we are doing a CAST to change the data type for the average. We also use a case statement in the ORDER BY clause at the end of the query, so our data comes out exactly as we want.

Because the answer set returns a whopping 880 rows, I will show it in a table below the example.

Group By Grouping Sets, Rollup, and Cube

```
SELECT
 EXTRACT(YEAR FROM SALE_DATE) as YR
,TO_CHAR(SALE_DATE, 'Q') as QTR
,REGION
,SALES_PERSON
,TO_CHAR(SALE_DATE, 'DY') AS DOW
,CAST(AVG(DAILY_SALES) AS DECIMAL(12,2)) as "AVG"
,SUM(DAILY_SALES) as "SUM"
FROM PIVOT_TEST_REGION_ALL
GROUP BY CUBE
(REGION,
 SALES_PERSON,
 EXTRACT(YEAR FROM SALE_DATE),
 TO_CHAR(SALE_DATE, 'DY'),
 TO_CHAR(SALE_DATE, 'Q') )
ORDER BY YR NULLS FIRST,
                QTR NULLS FIRST,
                 REGION NULLS FIRST,
                 SALES_PERSON NULLS FIRST,
CASE DOW
  WHEN 'MON' THEN 1
  WHEN 'TUE' THEN 2
  WHEN 'WED' THEN 3
  WHEN 'THU' THEN 4
  WHEN 'FRI' THEN 5
  WHEN 'SAT' THEN 6
  WHEN 'SUN' THEN 7
END NULLS FIRST;
```

EXTRACT and TO_CHAR DATE Functions

You can't use the alias in the CUBE, but must repeat the date functions again.

Show **10202550100** entries

Search:

Group By Grouping Sets, Rollup, and Cube

Show 20 entries Search:

YR	QTR	REGION	SALES_PERSON	DOW	AVG	SUM
		South	Helen Smith		3662.46	5596239.63
		South	Helen Smith	MON	3133.64	676866.5
		South	Helen Smith	TUE	3680.87	831877.53
		South	Helen Smith	WED	3514.06	783636.43
		South	Helen Smith	THU	3657.64	797367.58
		South	Helen Smith	FRI	4163.51	903483.26
		South	Helen Smith	SAT	4096.3	868417.15
		South	Helen Smith	SUN	3400.88	734591.18
	1				3644.13	5509925.81
	1			MON	4100.21	852845.48
	1			TUE	5242.97	1153454.46
	1			WED	3585.12	788726.63
	1			THU	2076.74	431962.37
	1			FRI	3559.89	783177.1
	1			SAT	3872.05	836364.13
	1			SUN	3015.43	663395.64
	1		Gary Lewis		1310.98	495550.63
	1		Gary Lewis	MON	1486.59	77302.96
	1		Gary Lewis	TUE	1870.72	102889.79
	1		Gary Lewis	WED	1267.85	69732.13

Showing 81 to 100 of 880 entries Previous Next

Group By Grouping Sets, Rollup, and Cube

Showing 1 to 20 of 880 entries

PreviousNext