



Getting Started with Oracle 12c

Grouping and Aggregating data using SQL

Description:

BISP is committed to provide BEST learning material to the beginners and advance learners. In the same series, we have prepared a complete end-to end Hands-on Beginner's Guide for Oracle Analytic Functions. The document focuses Grouping and Aggregating data using SQL

Join our professional training program and learn from experts.

History:

Version	Description	Change
0.1	Initial	Draft
0.1	Review	#1

Author
Pawan Madanan

Publish Date

Contents:

ROLLUP Operator:.....	3
CUBE Operator :.....	5
GROUPING FUNCTION:.....	9
GROUPING_ID.....	12
GROUPING SETS:.....	13
COMPOSITE COLUMNS.....	14



ROLLUP Operator:

About **ROLLUP** Operator:

- ROLLUP is an extension to Group By clause.
- ROLLUP enables a SELECT statement to calculate multiple levels of subtotals across a specified group of dimensions.
- Can be used by report makers to extract statistics and summary information from the result to use in charts ,graphs , reports.
- ROLLUP creates grouping by moving Right to Left along the list of columns in GROUP BY clause.
- Syntax of ROLLUP :
SELECT ... GROUP BY ROLLUP(grouping_column_reference_list).

To execute queries based on ROLLUP Operator, I am going to use table

Tables used : order_details

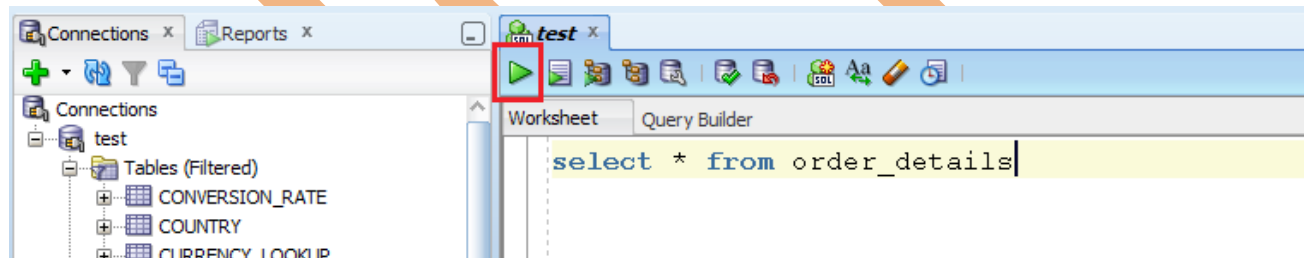
Order_Details basically contains details of different orders.

Before using ROLLUP operator, lets see whole of the data present in **order_details** table.

For we will be executing the following query :

Select * from order_details;

Then click on “run” button specified in below image to execute the query .



Output:

A screenshot of a query result window. The window title is 'Query Result x'. It shows a table with 12 rows and 10 columns. The columns are: ORDER_DETAIL_CODE, ORDER_NUMBER, SHIP_DATE, PRODUCT_NUMBER, PROMOTION_CODE, QUANTITY, UNIT_COST, UNIT_PRICE, and UNIT_SALE_PRICE. The data is as follows:

ORDER_DETAIL_CODE	ORDER_NUMBER	SHIP_DATE	PRODUCT_NUMBER	PROMOTION_CODE	QUANTITY	UNIT_COST	UNIT_PRICE	UNIT_SALE_PRICE
1	1012861	100754 21-JUL-04	94110	0	3717	1.85	5	4.65
2	1012872	100755 21-JUL-04	38110	10210	220	20	31.55	25.24
3	1012873	100755 21-JUL-04	27110	0	53	238.88	460.52	428.28
4	1012874	100755 21-JUL-04	28110	0	371	16	33.59	32.58
5	1012875	100755 21-JUL-04	2110	0	391	6.62	12.53	12.15
6	1012886	100756 21-JUL-04	111110	0	129	91.8	180.63	167.99
7	1012887	100756 21-JUL-04	106110	0	23	536.09	1161.46	1091.77
8	1012888	100756 21-JUL-04	107110	0	24	660.65	1359.72	1278.14
9	1012889	100756 21-JUL-04	8110	0	112	75	151.77	141.15
10	1012900	100757 20-JUL-04	84110	0	148	78.55	116.73	108.56
11	1012901	100757 20-JUL-04	29110	0	187	41.18	73.5	68.36
12	1012902	100757 20-JUL-04	17110	0	261	60	90.09	83.78

Lets say we want to know the details :

Requirement:

- a) Total quantity for every order on each corresponding product shipping date.
- b) Total quantity for every order irrespective of the corresponding product shipping dates
- c) Grand total of quantities.

With a single query execution, we get all of the above information. Otherwise we would have to use multiple select statements with UNION ALL, that would take multiple table access unlike in ROLL UP operator, only single access to the base table is required.

Query :

```
select order_number, ship_date, sum( quantity) as total from order_details
group by rollup( order_number, ship_date)
```

In the above query, three groups were created by ROLLUP Operator, i.e. (Order_number, ship_date), (order_number), ()

Output :

	ORDER_NUMBER	SHIP_DATE	TOTAL
1	800000	08-MAR-06	66
2	800000		66
3	810000	18-AUG-06	334
4	810000		334
5	820000	21-APR-06	135
6	820000		135
7	830000	18-MAY-06	199
8	830000		199
9	100100	19-JAN-04	9799
10	100100		9799
11	100200	16-FEB-04	2498
12	100200		2498
13	100300	15-MAR-04	7931
14	100300		7931
15	100400	19-APR-04	3500
16	100400		3500

.....

114919	834934		55
114920	834935	12-MAR-07	2084
114921	834935		2084
114922	834936	16-MAR-06	1629
114923	834936		1629
114924			89237091

3

Line labelled as “1” indicates a group totaled by both ship_date and order_number and is also answer to our first requirement but for a particular order number and ship date. You can get other total values from other rows, wherever ordernumber and ship date is not null.

Line labelled as “2” indicates a group totaled by order_number and also answer to second requirement but for single order .

So how do we know ,that any particular row contains total for order_number irrespective of ship_date ?

The answer to this is , In rows where you find ship_date in null , then you directly say that total has been calculated by order_number only(the null values shown here are generated by roll up operator, they are not actually present in database ,so how to differentiate between null values generated by roll up operator and stored null values has also been covered in this documentation, but the NULL 's here are replaced by Blank,so originally you will be seeing null's when you will executing the above query NOT blanks , I did so , so you don't get confused).

Other rows havent been labelled but can be seen and you can get the total value for other order_number's ,wherever shipdate is NULL .

Line labelled as “3” indicates Grand total and also answer to our third requirement.

Note: The NULL 's in the result are replaced by Blank,so originally you will be seeing null's when you will executing the above query NOT blanks , I did so , so you don't get confused.

Note: if you want to the end of the query result(as Scrolling down will be cumbersome) ,click anywhere on the result set , and then press **Ctrl + End**.

CUBE Operator :

About CUBE Operator :

- CUBE is an extension of the GROUP BY clause.
- CUBE produces subtotals for all possible combinations of groupings specified .
- Used for cross-tabular reports
- Syntax of CUBE operator:

SELECT ... GROUP BY CUBE (grouping_column_reference_list)

To execute queries based on CUBE Operator, I am going to use table

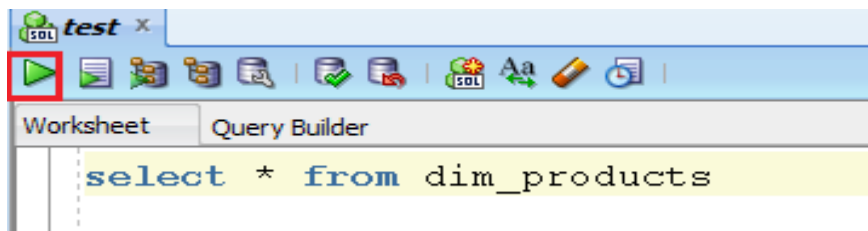
Tables used : dim_products

Dim_products basically contains data about different products

Before using **CUBE** operator, lets see whole of the data present in **dim_products** table.

For we will be executing the following query shown in the image:

Then click on “run” button specified in below image to execute the query .



Output:

A screenshot of a query result window titled 'Query Result x'. It shows a table with 12 rows and 6 columns: PRODUCTNUMBER, INTRODUCTIONDATE, PRODUCTNAME, PRODUCTIONCOST, PICTURE, and PICTUREURL. The data is as follows:

PRODUCTNUMBER	INTRODUCTIONDATE	PRODUCTNAME	PRODUCTIONCOST	PICTURE	PICTUREURL
1	77 1995-10-26 00:00:00.0	Bear Survival Edge	45	P77PA3KV12	/cognos/san
2	78 1990-02-15 00:00:00.0	Seeker 35	79	P78PA3BN13	/cognos/san
3	79 1990-02-15 00:00:00.0	Seeker 50	92	P78PA3BN13	/cognos/san
4	80 1992-03-05 00:00:00.0	Seeker Extreme	94	P80PA3BN13	/cognos/san
5	81 1992-03-05 00:00:00.0	Seeker Mini	40	P81PA3BN13	/cognos/san
6	82 1992-03-05 00:00:00.0	Glacier Basic	20	P82PA3NV14	/cognos/san
7	83 1992-03-05 00:00:00.0	Glacier Deluxe	56	P83PA3NV14	/cognos/san
8	84 1992-03-05 00:00:00.0	Glacier GPS	78	P84PA3NV14	/cognos/san
9	85 1994-06-12 00:00:00.0	Glacier GPS Extreme	176	P85PA3NV14	/cognos/san
10	86 1990-02-15 00:00:00.0	BugShield Natural	2	P86OP4IR15	/cognos/san
11	87 1990-02-15 00:00:00.0	BugShield Spray	2	P86OP4IR15	/cognos/san
12	88 1990-02-15 00:00:00.0	BugShield Lotion Lite	2	P86OP4IR15	/cognos/san

Lets say we want to know the following details :

Requirement:

- average production cost for different products of different product line and of different product type
(ie average by product type , product line ,product name)
- average production cost for different products irrespective of product line and product type
(ie average by product name)
- average production cost for different product type irrespective of product line and product name.
(ie average by product type)
- average production cost for different products belonging to different product type irrespective of product line
(ie average by product type and product name)
- whole average .

With a single query execution ,we get all of the above information . otherwise we would have to use multiple select statements with UNION ALL . that would take multiple table access unlike in CUBE operator , only single access to the base table is required.

Query :

```
test x
[Icons]
Worksheet Query Builder
select productline, producttype, productname, round(avg(productioncost)) as average_cost
from dim_products
group by cube(productline, producttype, productname)
```

In the above query the cube operator created the following groups:

(productline, producttype, productname)

(productline)

(producttype)

(productname)

(productline, producttype)

(productline, productname)

(producttype, productname)

()

Total 6 sets are created by CUBE operator with three elements , from this you can infer a direct formula , ie

For “n” elements/columns , 2^n sets will be formed by the CUBE Operator.

Output:

	PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	AVERAGE_COST
1				90
2			Star Peg	1
3			Bear Edge	23
4			Firefly 2	16
5			Firefly 4	18
6			Polar Ice	73
7			Polar Sun	46
8			Seeker 35	79
9			Seeker 50	92
10			Star Dome	473
11			Star Lite	250
12			Hibernator	86

.....

113			Canyon Mule ...	166	
114			Lady Hailsto...	472	
115			Hibernator S...	78	
116			Lady Hailsto...	509	
117	Rope			212	3
118	Rope		Husky Rope 50	106	
119	Rope		Husky Rope 60	126	
120	Rope		Husky Rope 100	231	
121	Rope		Husky Rope 200	383	4
122	Irons			324	
123	Irons		Hailstorm St...	305	
124	Irons		Hailstorm Ti...	380	
125	Irons		Lady Hailsto...	277	
126	Irons		Lady Hailsto...	333	
127	Packs			124	
128	Packs		Canyon Mule ...	24	
129	Packs		Canyon Mule ...	41	

.....

485	Mounta...	Rope	Husky Rope 50	106	
486	Mounta...	Rope	Husky Rope 60	126	
487	Mounta...	Rope	Husky Rope 100	231	
488	Mounta...	Rope	Husky Rope 200	383	
489	Mounta...	Tools		37	
490	Mounta...	Tools	Granite Axe	20	
491	Mounta...	Tools	Granite Ice	40	
492	Mounta...	Tools	Granite Grip	10	5
493	Mounta...	Tools	Granite Hammer	57	
494	Mounta...	Tools	Granite Shovel	46	
495	Mounta...	Tools	Granite Extreme	47	
496	Mounta...	Safety		42	
497	Mounta...	Safety	Husky Harness	44	
498	Mounta...	Safety	Granite Sign...	16	
499	Mounta...	Safety	Husky Harnes...	55	
500	Mounta...	Safety	Granite Clim...	52	

Line labelled as “1” indicates a whole average and also answer to our 5th requirement(e)

Line labelled as “2” indicates a group average by product name and also answer to second requirement(for a particular product if you will) and also answer to our second requirement(b)

Line labelled as “3” indicates a group average by product type and also answer to our third requirement(c).

Line labelled as “4” indicates a group average by product type and product name also answer to our fourth requirement(d).

Line labelled as “5” indicates a group average by product line , product type and also productname and also answer to our first requirement(a).

GROUPING FUNCTION:

About grouping function :

- It is used with ROLLUP or CUBE operator.
- It is used to identify the columns in a group forming the subtotal.
- It is used to distinguish between Stored nulls and the nulls produced by ROLLUP or CUBE operator.
- It returns a value **0** or **1**
- Syntax:
GROUPING (Column)

Use of GROUPING Function :

a)To determine the columns involved in the aggregation(sum,avg,min)

b) identify whether a NULL value in a result set indicates :

- NULL value from the base table ie stored NULL value
- NULL value created by ROLLUP or CUBE.

Return Value Indicates :

If return value is “0”, then it indicates:

- the corresponding columns was used to calculate aggregate value
- NULL value in the expression column is a stored NULL value

If return value is “1”, then it indicates:

- the corresponding columns was NOT used to calculate aggregate value
- NULL value in the expression column is NOT a stored NULL value, rather it has been created by CUBE or ROLLUP Operator .

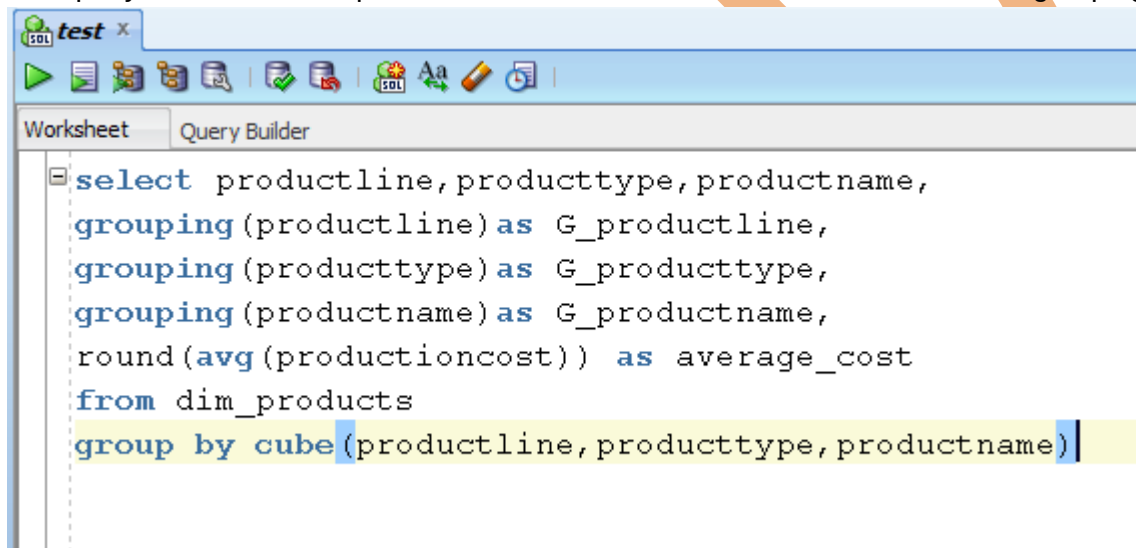
To execute queries based on GROUPING Operator, I am going to use table

Tables used : dim_products

Requirement:

- a) average production cost for different products of different product line and of different product type
(ie average by product type , product line ,product name)
- b) average production cost for different products irrespective of product line and product type
(ie average by product name)
- c) average production cost for different product type irrespective of product line and product name.
(ie average by product type)
- d) average production cost for different products belonging to different product type irrespective of product line
(ie average by product type and product name)
- e) whole average .

Note : these are the same requirements used in the CUBE operator for the same table ie dim_products .
The query used in CUBE operator one . has been modified to show the use of grouping functions



```
select productline, producttype, productname,  
grouping (productline) as G_productline,  
grouping (producttype) as G_producttype,  
grouping (productname) as G_productname,  
round(avg (productioncost)) as average_cost  
from dim_products  
group by cube (productline, producttype, productname)
```

Output:

Query Result x | Fetched 150 rows in 0.013 seconds

PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	G_PRODUCTLINE	G_PRODUCTTYPE	G_PRODUCTNAME	AVERAGE_COST
1			1	1	1	90
2		Star Peg	1	1	0	1
3		Bear Edge	1	1	0	23
4		Firefly 2	1	1	0	16
5		Firefly 4	1	1	0	18
6		Polar Ice	1	1	0	73
7		Polar Sun	1	1	0	46
8		Seeker 35	1	1	0	79
9		Seeker 50	1	1	0	92
10		Star Dome	1	1	0	473
11		Star Lite	1	1	0	250
12		Hibernator	1	1	0	86

Output shown again:

Query Result x | Fetched 150 rows in 0.013 seconds

PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	G_PRODUCTLINE	G_PRODUCTTYPE	G_PRODUCTNAME	AVERAGE_COST
1			1	1	1	90
2		Star Peg	1	1	0	1
3		Bear Edge	1	1	0	23
4		Firefly 2	1	1	0	16
5		Firefly 4	1	1	0	18
6		Polar Ice	1	1	0	73
7		Polar Sun	1	1	0	46
8		Seeker 35	1	1	0	79
.....						
297	Campin...	Star Gazer 3	0	1	0	476
298	Campin...	Star Gazer 6	0	1	0	490
299	Campin...	EverGlow Lamp	0	1	0	17
300	Campin...	TrailChef Cup	0	1	0	5
301	Campin...	Hibernator Pad	0	1	0	29

Line labeled as "1", has "AVERAGE_COST" column has value 90 , which is average production cost , none of the columns are included in the aggregation ,thus grouping function returns 1,1,1 in G_PRODUCTLINE, G_PRODUCTTYPE, G_PRODUCTNAME .

Line labeled as "2", has "AVERAGE_COST" column has value 73 , to calculate this value ,only **productname** is included in the aggregation ,thus grouping function returns 0 for G_PRODUCTNAME and 1,1 in G_PRODUCTLINE, G_PRODUCTTYPE respectively.

Line labeled as “3”, has “AVERAGE_COST” column has value 470 , to calculate this value ,both **productline**, **productname** were included in the aggregation ,thus grouping function returns 0 for G_PRODUCTNAME , G_PRODUCTLINE and 1 for G_PRODUCTTYPE respectively.

(scroll up to the preceding page to see what 0 or 1 in grouping function indicates.)

GROUPING_ID

- **GROUPING_ID** function works as an extension of the GROUPING function
- Row filtering is easier with GROUPING_ID because the desired rows can be identified with a single condition of GROUPING_ID = n.
- It accepts one or more columns and returns the decimal equivalent of the GROUPING bit vector , which is just the combination of results from GROUPING function for each column specified.
- Syntax:
GROUPING_ID(columns)

To execute queries based on GROUPING_ID Operator, I am going to use table

Tables used : dim_products

The query used in CUBE operator one . has been modified to show the use of grouping_ID function:

```

select productline, producttype, productname, round(avg(productioncost)) as average_cost,
grouping(productline) as G_productline,
grouping(producttype) as G_producttype,
grouping(productname) as G_productname,
grouping_id(productline, producttype, productname) as gid_line_type_name
from dim_products
group by cube (productline, producttype, productname)
  
```

OUTPUT:

PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	AVERAGE_COST	G_PRODUCTLINE	G_PRODUCTTYPE	G_PRODUCTNAME	GID_LINE_TYPE_NAME
			90	1	1	1	7
		Star Peg	1	1	1	0	6
		Bear Edge	23	1	1	0	6
		Firefly 2	16	1	1	0	6
		Firefly 4	18	1	1	0	6
		Polar Ice	73	1	1	0	6
		Polar Sun	46	1	1	0	6

The GROUPING_ID function here is : GID_LINE_TYPE_NAME, which is actually concatenating the results of Grouping functions into a bit vector and then returning the decimal equivalent of that .

For ex in the First Row where AVERAGE_COST= 90, the grouping function values are 1 , 1 , 1 , converting it into bit vector 111 , and the decimal equivalent of this is 7

In next row we have GROUPING function values as 1,1,0 , converting it into bit vector : 110 and its decimal equivalent is : 6

NOTE: In the above output, I have used 3 grouping functions, you can avoid it directly use GROUPING_ID function.

So , GROUPING_ID avoids the use of multiple GROUPING function and also you can make the filtering conditions easier with the use single GROUPING_ID function , instead of multiple GROUPING functions,
by using only single condition ie GROUPING_ID = n

GROUPING SETS:

- Further extension of GROUP BY
- Used to define multiple groupings in the same query.

Why to use ? :

- Only one table access is required.
- No need to use complex union queries.

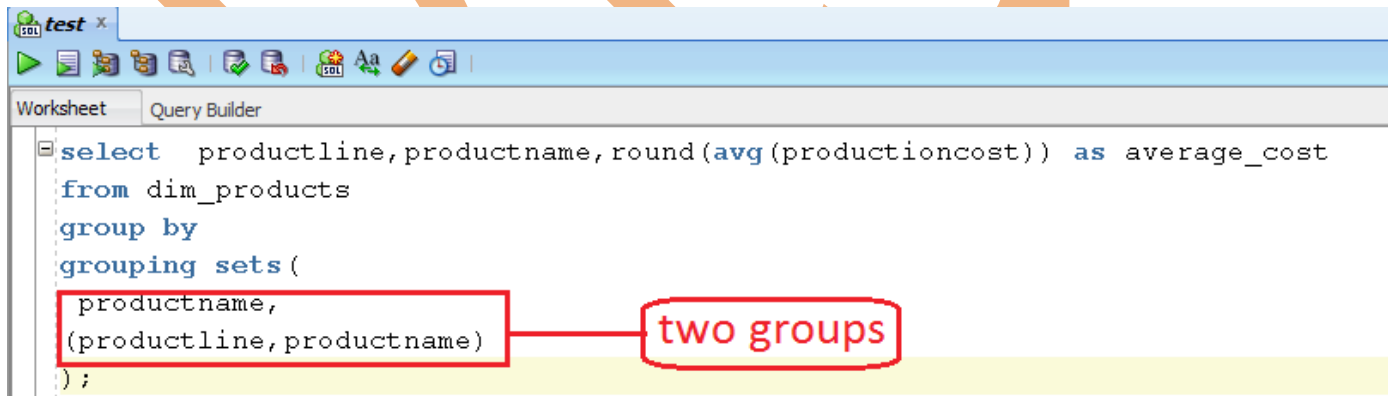
With the help of Grouping sets you can specify the groups on which the aggregation will be performed . Unlike in

ROLLUP or CUBE where if I say :

	Sets formed	Total sets
ROLLUP(a,b)	(a,b),a,()	3 sets
CUBE(a,b,c)	a,b,c,(a,b,c),(a,b),(a,c),(b,c),()	$2^3 = 8$ sets

So the above sets are automatically generated when you use CUBE or ROLLUP , but my interest might be in some of the groups ,this is where you can use GROUPINGSETS function to specify your own groups.

Query :



```

select productline, productname, round(avg(productioncost)) as average_cost
from dim_products
group by
grouping sets (
productname,
(productline, productname)
);
    
```

I have specified only 2 groups , so on those groups only , aggregation will be performed

Output:

	PRODUCTLINE	PRODUCTNAME	AVERAGE_COST	
1	Camping Equipment	Star Peg	1	1
2	(null)	Star Peg	1	
3	Personal Accessories	Bear Edge	23	
4	(null)	Bear Edge	23	
5	Camping Equipment	Firefly 2	16	
6	(null)	Firefly 2	16	
7	Camping Equipment	Firefly 4	18	
8	(null)	Firefly 4	18	2
9	Personal Accessories	Polar Ice	73	
10	(null)	Polar Ice	73	
11	Personal Accessories	Polar Sun	46	
12	(null)	Polar Sun	46	

Line labelled "1" indicates aggregation by both PRODUCTLINE and PRODUCTNAME , as we specified in the GROUPING SETS

Line labelled "2" indicates aggregation by PRODUCTNAME as we specified in the GROUPING SETS.

COMPOSITE COLUMNS

About composite columns :

- Collection of columns treated as single unit.
- used in GROUP BY clause.

How to define composite columns ?:

- Use parenthesis to group columns

Why to use ?

- To skip aggregation across certain levels.
- For ex
CUBE((a,b),c) | a ,b together is a composite column.

	Sets formed :
ROLLUP(a,(b,c))	(a,b,c), a,()
CUBE(a,(b,c))	(a,b,c),(),a,(b,c)

Query :

```
SQL Worksheet | History
Worksheet | Query Builder
select productline, producttype, productname, round(avg(productioncost)) as average_cost
from dim_products
group by cube(productline, (producttype, productname))
```

Both producttype and productname treated as a single unit

Output:

Explain Plan x | Script Output x | Query... x

SQL | All Rows Fetched: 236 in 0.058 seconds

PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	AVERAGE_COST
1 Mountaineering Equipment	Rope	Husky Rope 50	106
2 Mountaineering Equipment	Rope	Husky Rope 60	126
3 Mountaineering Equipment	Rope	Husky Rope 100	231
4 Mountaineering Equipment	Rope	Husky Rope 200	383
5 Golf Equipment	Irons	Hailstorm Steel Irons	305

Explain Plan x | Script Output x | Query Result x

SQL | All Rows Fetched: 236 in 0.058 seconds

PRODUCTLINE	PRODUCTTYPE	PRODUCTNAME	AVERAGE_COST
222	Insect...	BugShield Natural	2
223	Insect...	BugShield Lotion Lite	2
224	Climbi...	Granite Belay	35
225	Climbi...	Granite Pulley	19
226	Climbi...	Firefly Charger	35
227	Climbi...	Granite Carabiner	2
228	Climbi...	Granite Chalk Bag	9
229	Climbi...	Firefly Climbing Lamp	28
230	Climbi...	Firefly Rechargeable Battery	4
231 Camping Equipment			94
232 Golf Equipment			243
233 Mountaineering Equipment			65
234 Outdoor Protection			5
235 Personal Accessories			60
236			90

Only on four groups, aggregation was performed ,because we have specified composite columns ,which are treated as single unit,otherwise on $2^3 = 8$ groups aggregation would have been performed.,

Groups formed because of composite columns are :

Line labeled by "1" indicates aggregation by productline , producttype,productname

Line labeled by "2" indicates aggregation by producttype and productname

Line labeled by "3" indicates aggregation by productline

Line labeled by "4" indicates average cost of production_cost column in the in dim_products table

BISP