**Implementing OLAP Operations in Oracle 10g Express Edition**

**PIVOT operation:** The PIVOT operator takes data in separate rows, aggregates it and converts it into columns. To see the PIVOT operator in action we need to create a test table.

Ref: http://oracle-base.com/articles/11g/pivot-and-unpivot-operators-11gr1.php

1. Creating Table:

CREATE TABLE sales (

item\_name varchar2(5) NOT NULL,

color varchar2(6) NOT NULL,

cloths\_size varchar2(6) NOT NULL,

quantity NUMBER(5) NOT NULL

);

2. Inserting Data:

INSERT ALL

INTO sales VALUES ('skirt', 'dark', 'small', 2)

INTO sales VALUES ('skirt', 'dark', 'medium', 5)

INTO sales VALUES ('skirt', 'dark', 'large', 1)

INTO sales VALUES ('skirt', 'pastel', 'small', 11)

INTO sales VALUES ('skirt', 'pastel', 'medium', 9)

INTO sales VALUES ('skirt', 'pastel', 'large', 15)

INTO sales VALUES ('skirt', 'white', 'small', 2)

INTO sales VALUES ('skirt', 'white', 'medium', 5)

INTO sales VALUES ('skirt', 'white', 'large', 3)

INTO sales VALUES ('dress', 'dark', 'small', 2)

INTO sales VALUES ('dress', 'dark', 'medium', 6)

INTO sales VALUES ('dress', 'dark', 'large', 12)

INTO sales VALUES ('dress', 'pastel', 'small', 4)

INTO sales VALUES ('dress', 'pastel', 'medium', 3)

INTO sales VALUES ('dress', 'pastel', 'large', 3)

INTO sales VALUES ('dress', 'white', 'small', 2)

INTO sales VALUES ('dress', 'white', 'medium', 3)

INTO sales VALUES ('dress', 'white', 'large', 0)

INTO sales VALUES ('shirt', 'dark', 'small', 2)

INTO sales VALUES ('shirt', 'dark', 'medium', 6)

INTO sales VALUES ('shirt', 'dark', 'large', 6)

INTO sales VALUES ('shirt', 'pastel', 'small', 4)

INTO sales VALUES ('shirt', 'pastel', 'medium', 1)

INTO sales VALUES ('shirt', 'pastel', 'large', 2)

INTO sales VALUES ('shirt', 'white', 'small', 17)

INTO sales VALUES ('shirt', 'white', 'medium', 1)

INTO sales VALUES ('shirt', 'white', 'large', 10)

INTO sales VALUES ('pants', 'dark', 'small', 14)

INTO sales VALUES ('pants', 'dark', 'medium', 6)

INTO sales VALUES ('pants', 'dark', 'large', 0)

INTO sales VALUES ('pants', 'pastel', 'small', 1)

INTO sales VALUES ('pants', 'pastel', 'medium', 0)

INTO sales VALUES ('pants', 'pastel', 'large', 1)

INTO sales VALUES ('pants', 'white', 'small', 3)

INTO sales VALUES ('pants', 'white', 'medium', 0)

INTO sales VALUES ('pants', 'white', 'large', 2)

SELECT \* FROM dual;

**ROLLUP**

In addition to the regular aggregation results we expect from the GROUP BY clause, the ROLLUP extension produces group subtotals from right to left and a grand total. If "n" is the number of columns listed in the ROLLUP, there will be n+1 levels of subtotals.

Ref: <http://oracle-base.com/articles/misc/rollup-cube-grouping-functions-and-grouping-sets.php>

**Only group by**

SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY item\_name, cloths\_size, color

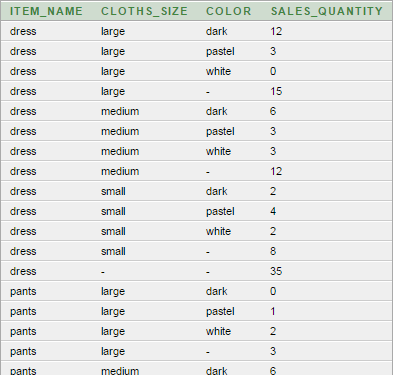
**Rollup**

SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY rollup(item\_name, cloths\_size, color)

ORDER BY item\_name, cloths\_size, color;

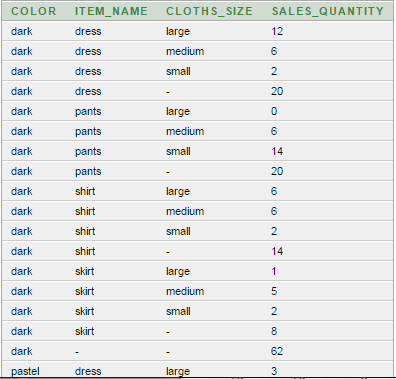


SELECT color, item\_name, cloths\_size, SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY rollup(color, item\_name, cloths\_size)

ORDER BY color, item\_name, cloths\_size;



SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY rollup(item\_name), rollup(cloths\_size, color)

ORDER BY item\_name, cloths\_size, color;

## CUBE

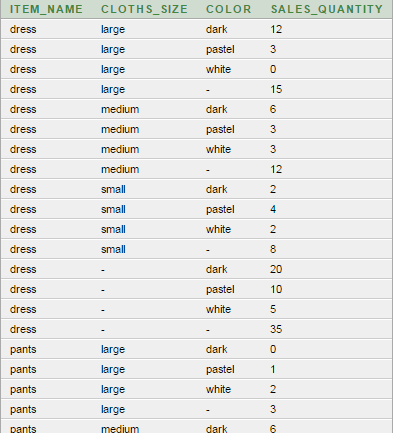
In addition to the subtotals generated by the ROLLUP extension, the CUBE extension will generate subtotals for all combinations of the dimensions specified. If "n" is the number of columns listed in the CUBE, there will be 2n subtotal combinations.

SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY cube(item\_name, cloths\_size, color)

ORDER BY item\_name, cloths\_size, color;



SELECT decode(grouping(item\_name),1, 'all', item\_name) as item\_name,

decode(grouping(cloths\_size),1, 'all', cloths\_size) as cloths\_size,

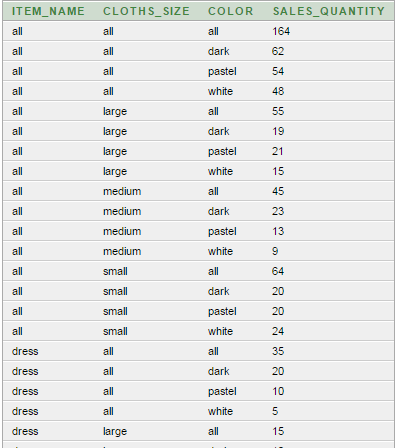
decode(grouping(color),1, 'all', color) as color,

SUM(quantity) AS sales\_quantity

FROM sales

GROUP BY cube(item\_name, cloths\_size, color)

ORDER BY item\_name, cloths\_size, color;



## GROUPING Functions

### GROUPING

It can be quite easy to visually identify subtotals generated by rollups and cubes, but to do it programatically you really need something more accurate than the presence of null values in the grouping columns. This is where the GROUPING function comes in. It accepts a single column as a parameter and returns "1" if the column contains a null value generated as part of a subtotal by a ROLLUP or CUBE operation or "0" for any other value, including stored null values.

The following query is a repeat of a previous cube, but the GROUPING function has been added for each of the dimensions in the cube.

SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity,

GROUPING(item\_name) AS item\_name\_flag,

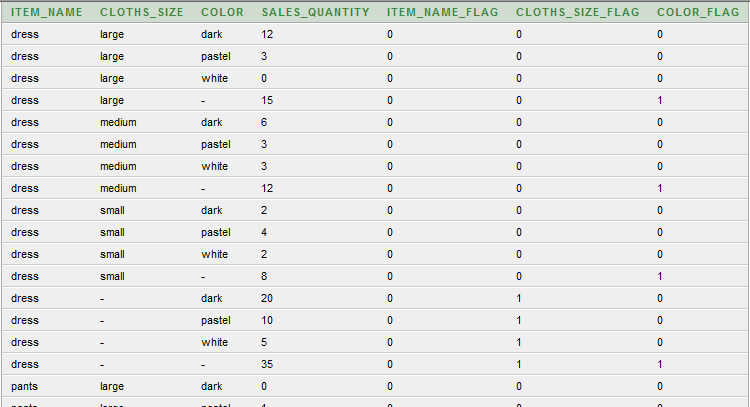
GROUPING(cloths\_size) AS cloths\_size\_flag,

GROUPING(color) AS color\_flag

FROM sales

GROUP BY CUBE (item\_name, cloths\_size, color)

ORDER BY item\_name, cloths\_size, color;



SELECT item\_name, cloths\_size, color, SUM(quantity) AS sales\_quantity,

GROUPING(item\_name) AS f1g,

GROUPING(cloths\_size) AS f2g,

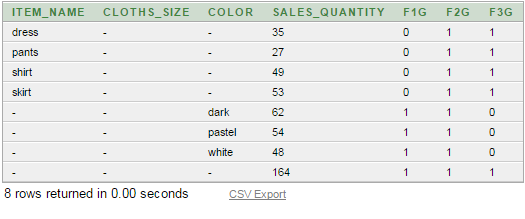
GROUPING(color) AS f3g

FROM sales

GROUP BY CUBE (item\_name, cloths\_size, color)

having (GROUPING(color) = '1' and GROUPING(cloths\_size) = '1') or (GROUPING(item\_name) = '1' and GROUPING(cloths\_size) = '1')

ORDER BY item\_name, cloths\_size, color;



**PIVOT**

PIVOT operation using DECODE function in Oracle 10g. (In 11g direct PIVOT operation exist)

SELECT item\_name, cloths\_size,

SUM(DECODE(color, 'dark', quantity, 0)) AS dark\_sum\_quantity,

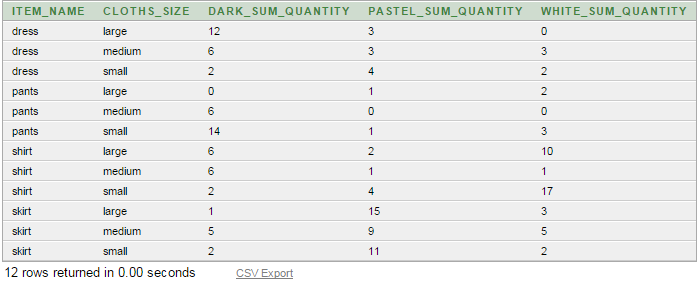
SUM(DECODE(color, 'pastel', quantity, 0)) AS pastel\_sum\_quantity,

SUM(DECODE(color, 'white', quantity, 0)) AS white\_sum\_quantity

FROM sales

GROUP BY item\_name, cloths\_size

ORDER BY item\_name, cloths\_size;



SELECT item\_name,

SUM(DECODE(color, 'dark', quantity, 0)) AS dark\_sum\_quantity,

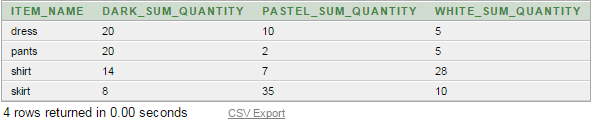
SUM(DECODE(color, 'pastel', quantity, 0)) AS pastel\_sum\_quantity,

SUM(DECODE(color, 'white', quantity, 0)) AS white\_sum\_quantity

FROM sales

GROUP BY item\_name

ORDER BY item\_name;



SELECT cloths\_size,

SUM(DECODE(color, 'dark', quantity, 0)) AS dark\_sum\_quantity,

SUM(DECODE(color, 'pastel', quantity, 0)) AS pastel\_sum\_quantity,

SUM(DECODE(color, 'white', quantity, 0)) AS white\_sum\_quantity

FROM sales

GROUP BY cloths\_size

ORDER BY cloths\_size;



SELECT item\_name, color,

SUM(DECODE(cloths\_size, 'small', quantity, 0)) AS small\_sum\_quantity,

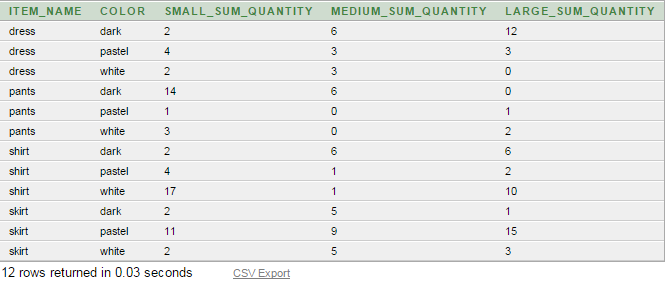
SUM(DECODE(cloths\_size, 'medium', quantity, 0)) AS medium\_sum\_quantity,

SUM(DECODE(cloths\_size, 'large', quantity, 0)) AS large\_sum\_quantity

FROM sales

GROUP BY item\_name, color

ORDER BY item\_name, color;



**UNPIVOT Operation:** The UNPIVOT operator converts column-based data into separate rows. To see the UNPIVOT operator in action we need to create a test table.

1. Creating Table:

CREATE TABLE unpivot\_test (

id NUMBER,

customer\_id NUMBER,

product\_code\_a NUMBER,

product\_code\_b NUMBER,

product\_code\_c NUMBER,

product\_code\_d NUMBER

);

2. Inserting Data:

INSERT INTO unpivot\_test VALUES (1, 101, 10, 20, 30, NULL);

INSERT INTO unpivot\_test VALUES (2, 102, 40, NULL, 50, NULL);

INSERT INTO unpivot\_test VALUES (3, 103, 60, 70, 80, 90);

INSERT INTO unpivot\_test VALUES (4, 104, 100, NULL, NULL, NULL);

COMMIT;

3. Prior to 11g, we can get the same result using the DECODE function and a pivot table with the correct number of rows. In the following example we use the CONNECT BY clause in a query from dual to generate the correct number of rows for the unpivot operation.

SELECT id,

customer\_id,

DECODE(unpivot\_row, 1, 'A',

2, 'B',

3, 'C',

4, 'D',

'N/A') AS product\_code,

DECODE(unpivot\_row, 1, product\_code\_a,

2, product\_code\_b,

3, product\_code\_c,

4, product\_code\_d,

'N/A') AS quantity

FROM unpivot\_test,

(SELECT level AS unpivot\_row FROM dual CONNECT BY level <= 4)

ORDER BY 1,2,3;

