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;
```

ITEM_NAME	CLOTHS_SIZE	COLOR	SALES_QUANTITY
dress	large	dark	12
dress	large	pastel	3
dress	large	white	0
dress	large	-	15
dress	medium	dark	6
dress	medium	pastel	3
dress	medium	white	3
dress	medium	-	12
dress	small	dark	2
dress	small	pastel	4
dress	small	white	2
dress	small	-	8
dress	-	-	35
pants	large	dark	0
pants	large	pastel	1
pants	large	white	2
pants	large	-	3
pants	medium	dark	6

```
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;
```

COLOR	ITEM_NAME	CLOTHS_SIZE	SALES_QUANTITY
dark	dress	large	12
dark	dress	medium	6
dark	dress	small	2
dark	dress	-	20
dark	pants	large	0
dark	pants	medium	6
dark	pants	small	14
dark	pants	-	20
dark	shirt	large	6
dark	shirt	medium	6
dark	shirt	small	2
dark	shirt	-	14
dark	skirt	large	1
dark	skirt	medium	5
dark	skirt	small	2
dark	skirt	-	8
dark	-	-	62
pastel	dress	large	3

```
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 2^n 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;
```

ITEM_NAME	CLOTHS_SIZE	COLOR	SALES_QUANTITY
dress	large	dark	12
dress	large	pastel	3
dress	large	white	0
dress	large	-	15
dress	medium	dark	6
dress	medium	pastel	3
dress	medium	white	3
dress	medium	-	12
dress	small	dark	2
dress	small	pastel	4
dress	small	white	2
dress	small	-	8
dress	-	dark	20
dress	-	pastel	10
dress	-	white	5
dress	-	-	35
pants	large	dark	0
pants	large	pastel	1
pants	large	white	2
pants	large	-	3
pants	medium	dark	6

```
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;
```

ITEM_NAME	CLOTHS_SIZE	COLOR	SALES_QUANTITY
all	all	all	164
all	all	dark	62
all	all	pastel	54
all	all	white	48
all	large	all	55
all	large	dark	19
all	large	pastel	21
all	large	white	15
all	medium	all	45
all	medium	dark	23
all	medium	pastel	13
all	medium	white	9
all	small	all	64
all	small	dark	20
all	small	pastel	20
all	small	white	24
dress	all	all	35
dress	all	dark	20
dress	all	pastel	10
dress	all	white	5
dress	large	all	15

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;
```

ITEM_NAME	CLOTHS_SIZE	COLOR	SALES_QUANTITY	ITEM_NAME_FLAG	CLOTHS_SIZE_FLAG	COLOR_FLAG
dress	large	dark	12	0	0	0
dress	large	pastel	3	0	0	0
dress	large	white	0	0	0	0
dress	large	-	15	0	0	1
dress	medium	dark	6	0	0	0
dress	medium	pastel	3	0	0	0
dress	medium	white	3	0	0	0
dress	medium	-	12	0	0	1
dress	small	dark	2	0	0	0
dress	small	pastel	4	0	0	0
dress	small	white	2	0	0	0
dress	small	-	8	0	0	1
dress	-	dark	20	0	1	0
dress	-	pastel	10	0	1	0
dress	-	white	5	0	1	0
dress	-	-	35	0	1	1
pants	large	dark	0	0	0	0

```
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;
```

ITEM_NAME	CLOTHS_SIZE	COLOR	SALES_QUANTITY	F1G	F2G	F3G
dress	-	-	35	0	1	1
pants	-	-	27	0	1	1
shirt	-	-	49	0	1	1
skirt	-	-	53	0	1	1
-	-	dark	62	1	1	0
-	-	pastel	54	1	1	0
-	-	white	48	1	1	0
-	-	-	164	1	1	1

8 rows returned in 0.00 seconds CSV Export

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;
```

ITEM_NAME	CLOTHS_SIZE	DARK_SUM_QUANTITY	PASTEL_SUM_QUANTITY	WHITE_SUM_QUANTITY
dress	large	12	3	0
dress	medium	6	3	3
dress	small	2	4	2
pants	large	0	1	2
pants	medium	6	0	0
pants	small	14	1	3
shirt	large	6	2	10
shirt	medium	6	1	1
shirt	small	2	4	17
skirt	large	1	15	3
skirt	medium	5	9	5
skirt	small	2	11	2

12 rows returned in 0.00 seconds CSV Export

```
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;
```

ITEM_NAME	DARK_SUM_QUANTITY	PASTEL_SUM_QUANTITY	WHITE_SUM_QUANTITY
dress	20	10	5
pants	20	2	5
shirt	14	7	28
skirt	8	35	10

4 rows returned in 0.00 seconds CSV Export

```
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;

CLOTHS_SIZE	DARK_SUM_QUANTITY	PASTEL_SUM_QUANTITY	WHITE_SUM_QUANTITY
large	19	21	15
medium	23	13	9
small	20	20	24

3 rows returned in 0.00 seconds CSV Export

```
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;
```

ITEM_NAME	COLOR	SMALL_SUM_QUANTITY	MEDIUM_SUM_QUANTITY	LARGE_SUM_QUANTITY
dress	dark	2	6	12
dress	pastel	4	3	3
dress	white	2	3	0
pants	dark	14	6	0
pants	pastel	1	0	1
pants	white	3	0	2
shirt	dark	2	6	6
shirt	pastel	4	1	2
shirt	white	17	1	10
skirt	dark	2	5	1
skirt	pastel	11	9	15
skirt	white	2	5	3

12 rows returned in 0.03 seconds CSV Export

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;
```

ID	CUSTOMER_ID	PRODUCT_CODE	QUANTITY
1	101	A	10
1	101	B	20
1	101	C	30
1	101	D	-
2	102	A	40
2	102	B	-
2	102	C	50
2	102	D	-
3	103	A	60
3	103	B	70
3	103	C	80
3	103	D	90
4	104	A	100
4	104	B	-
4	104	C	-
4	104	D	-

16 rows returned in 0.00 seconds CSV Export