

Міністерство науки і освіти, молоді та спорту України
Національний технічний університет України
«Київський політехнічний інститут»
Кафедра АСОІУ

ЗВІТ
Про виконання лабораторної роботи №3
З дисципліни
«OLAP та сховища даних»

Прийняв:
Олійник Ю.О.

Виконав:
студент 3-го курсу
Гр. ІІІ-52 ФІОТ
Набоков Едуард

Київ 2018

Запити:

1. Показати ранг кожного товару у групі (відповідно до зменшення ціни в групі). Запит реалізувати через функції rank() та dense_rank(). Порівняти результати виконання.

```
SELECT p.id_product, p.id_product_type, p.price,  
       rank() OVER(PARTITION by p.id_product_type ORDER by p.price desc)  
rank_ex,  
       dense_rank() OVER(PARTITION by p.id_product_type ORDER by p.price  
desc) dense_rank_ex  
FROM product p;
```

The screenshot shows a database query editor with a 'Query Builder' tab. The SQL query is displayed in the editor, and the 'Query Result' tab shows the results of the query. The results are displayed in a table with 6 columns: ID_PRODUCT, ID_PRODUCT_TYPE, PRICE, RANK_EX, and DENSE_RANK_EX. The table contains 21 rows of data.

ID_PRODUCT	ID_PRODUCT_TYPE	PRICE	RANK_EX	DENSE_RANK_EX
6	13	3	15	1
7	16	3	15	1
8	11	3	12	4
9	9	3	12	4
10	10	4	65	1
11	20	5	98	1
12	21	5	95	2
13	15	5	30	3
14	14	5	25	4
15	19	6	170	1
16	18	6	120	2
17	17	6	100	3
18	23	7	7	1
19	22	7	4	2
20	4	7	3	3
21	24	7	3	3

2. Вивести список із двох найдешевших товарів у кожній групі.

```
SELECT p_type, prod, p_price  
FROM  
(  
    SELECT p.id_product_type p_type, p.id_product prod, p.price p_price,  
           dense_rank() OVER(PARTITION by p.id_product_type ORDER by p.price)  
rank_prod  
    FROM product p  
)  
WHERE rank_prod <= 2;
```

Worksheet Query Builder

```

FROM product p;

-- 2 Вивести список із двох найдешевших товарів у кожній групі.
SELECT p_type, prod, p_price
FROM
(
    SELECT p.id_product_type p_type, p.id_product prod, p.price p_price,
           dense_rank() OVER(PARTITION by p.id_product_type ORDER by p.price) rank_prod
    FROM product p
)
WHERE rank_prod <= 2;

```

Query Result x

SQL | All Rows Fetched: 15 in 0.008 seconds

	P_TYPE	PROD	P_PRICE
1	1	2	10
2	1	6	55
3	3	9	12
4	3	11	12
5	3	12	15
6	3	16	15
7	3	13	15
8	4	10	65
9	5	14	25
10	5	15	30
11	6	17	100
12	6	18	120
13	7	24	3
14	7	4	3
15	7	22	4

| Line 11 Column 5 | Insert | Modified | Unix/Mac: LF

3. Показати які товари по кожній групі мають найбільші продажі .

```

SELECT t.id_product_type, t.id_product FROM
(
    SELECT p.id_product_type, p.id_product, COUNT(*) max_sold_by_product,
           MAX(COUNT(*)) OVER(PARTITION by p.id_product_type) max_sold_per_type
    FROM invoice i
    JOIN invoice_detail invd
    ON i.id_invoice = invd.id_invoice
    JOIN product p
    ON invd.id_product = p.id_product
    GROUP by p.id_product_type, p.id_product
    ORDER by p.id_product_type, p.id_product
) t WHERE max_sold_by_product = max_sold_per_type;

```

Worksheet Query Builder

```

SELECT p.id_product_type, p.id_product, COUNT(*) max_sold_by_product,
       MAX(COUNT(*)) OVER(PARTITION by p.id_product_type) max_sold_per_type
FROM invoice i
JOIN invoice_detail invd
ON i.id_invoice = invd.id_invoice
JOIN product p
ON invd.id_product = p.id_product
GROUP by p.id_product_type, p.id_product
ORDER by p.id_product_type, p.id_product
) t WHERE max_sold_by_product = max_sold_per_type;

```

Query Result x

SQL | All Rows Fetched: 6 in 0.006 seconds

	ID_PRODUCT_TYPE	ID_PRODUCT
1	1	7
2	3	9
3	4	10
4	5	14
5	6	18
6	7	4

perform "Go to Declaration" | Line 32 Column 1 | Insert | Modified | Unix/Mac: LF

4. Вивести список товарів з найбільшою вартістю, які найдовше зберігаються на полицях складу.

```

SELECT * FROM(
    SELECT s.shelf, s.id_product, p.price, s.date_oper,
        MAX(SYSDATE - s.date_oper) OVER (PARTITION by s.shelf) max_diff,
        MAX(p.price) OVER (PARTITION by s.shelf) max_price
    FROM product p
    JOIN store s
    ON p.id_product = s.id_product
) t
WHERE t.max_price = t.price AND t.max_diff = (SYSDATE - t.date_oper);

```

[illegible]

5. Побудувати запит з використанням функцій LEAD. Той же запит реалізувати через функцію LAG. Порівняти результати їх виконання.

```
SELECT p.id_product_type, p.id_product, p.price,
       LEAD(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by
p.price) price_next
FROM product p;
```

```
SELECT p.id_product_type, p.id_product, p.price,
       LAG(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by
p.price) price_next
FROM product p;
```

Worksheet Query Builder

```

) t
WHERE t.max_price = t.price AND t.max_diff = (SYSDATE - t.date_oper);

-- 5 Побудувати запит з використанням функцій LEAD. Той же запит реалізувати через функцію LAG. Порівняти результати їх виконання.
SELECT p.id_product_type, p.id_product, p.price,
       LEAD(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by p.price) price_next
FROM product p;

SELECT p.id_product_type, p.id_product, p.price,
       LAG(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by p.price) price_next
FROM product p;

```

Query Result x

SQL | All Rows Fetched: 21 in 0.001 seconds

ID_PRODUCT_TYPE	ID_PRODUCT	PRICE	PRICE_NEXT
1	1	2	10
2	1	6	55
3	1	8	60
4	1	7	78
5	3	9	12
6	3	11	12
7	3	12	15
8	3	16	15
9	3	13	15
10	4	10	65
11	5	14	25
12	5	15	30
13	5	21	95
14	5	20	98
15	6	17	100
16	6	18	120

ask3_scripts.sql | Line 45 Column 8 | Insert | Modified | Unix/Mac: LF

Worksheet Query Builder

```

) t
WHERE t.max_price = t.price AND t.max_diff = (SYSDATE - t.date_oper);

-- 5 Побудувати запит з використанням функцій LEAD. Той же запит реалізувати через функцію LAG. Порівняти результати їх виконання.
SELECT p.id_product_type, p.id_product, p.price,
       LEAD(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by p.price) price_next
FROM product p;

SELECT p.id_product_type, p.id_product, p.price,
       LAG(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by p.price) price_next
FROM product p;

```

Query Result x

SQL | All Rows Fetched: 21 in 0.005 seconds

ID_PRODUCT_TYPE	ID_PRODUCT	PRICE	PRICE_NEXT
1	1	2	10
2	1	6	55
3	1	8	60
4	1	7	78
5	3	9	12
6	3	11	12
7	3	12	15
8	3	16	15
9	3	13	15
10	4	10	65
11	5	14	25
12	5	15	30
13	5	21	95
14	5	20	98
15	6	17	100
16	6	18	120

ask3_scripts.sql | Line 49 Column 7 | Insert | Modified | Unix/Mac: LF

6. Побудувати аналітичні запити, які б використовували наступні функції (один запит - одна функція):

– FIRST_VALUE (або LAST_VALUE)

```
SELECT id_product_type, id_product, low_price FROM (
    SELECT id_product_type, id_product, price,
        FIRST_VALUE(price) OVER (PARTITION by id_product_type ORDER BY price)
    low_price
    FROM product
) WHERE price = low_price;
```

The screenshot shows a SQL Query Builder window with the following query:

```
LAG(p.price, 1, 0) OVER(PARTITION by p.id_product_type ORDER by p.price) price_next
FROM product p;

-- 6 Побудувати аналітичні запити, які б використовували наступні функції (один запит – одна функція):
-- FIRST_VALUE (або LAST_VALUE)
SELECT id_product_type, id_product, low_price FROM (
    SELECT id_product_type, id_product, price,
        FIRST_VALUE(price) OVER (PARTITION by id_product_type ORDER BY price) low_price
    FROM product
) WHERE price = low_price;
```

The Query Result window shows the following data:

ID_PRODUCT_TYPE	ID_PRODUCT	LOW_PRICE
1	1	2
2	3	9
3	3	11
4	4	10
5	5	14
6	6	17
7	7	4
8	7	24

– MAX (або MIN)

```
SELECT id_product_type, id_product, low_price FROM (
    SELECT id_product_type, id_product, price,
        MIN(price) OVER(PARTITION by id_product_type ORDER by price) low_price
    FROM product p
) WHERE price = low_price;
```

The screenshot shows a SQL Query Builder window with the following query:

```
) WHERE price = low_price;

-- MAX (or MIN)
SELECT id_product_type, id_product, low_price FROM (
    SELECT id_product_type, id_product, price,
        MIN(price) OVER(PARTITION by id_product_type ORDER by price) low_price
    FROM product p
) WHERE price = low_price;

SELECT s.shelf,
    sum(s.quantity) OVER(PARTITION by s.shelf) sumn
```

The Query Result window shows the following data:

ID_PRODUCT_TYPE	ID_PRODUCT	LOW_PRICE
1	1	2
2	3	9
3	3	11
4	4	10
5	5	14
6	6	17
7	7	4
8	7	24

```

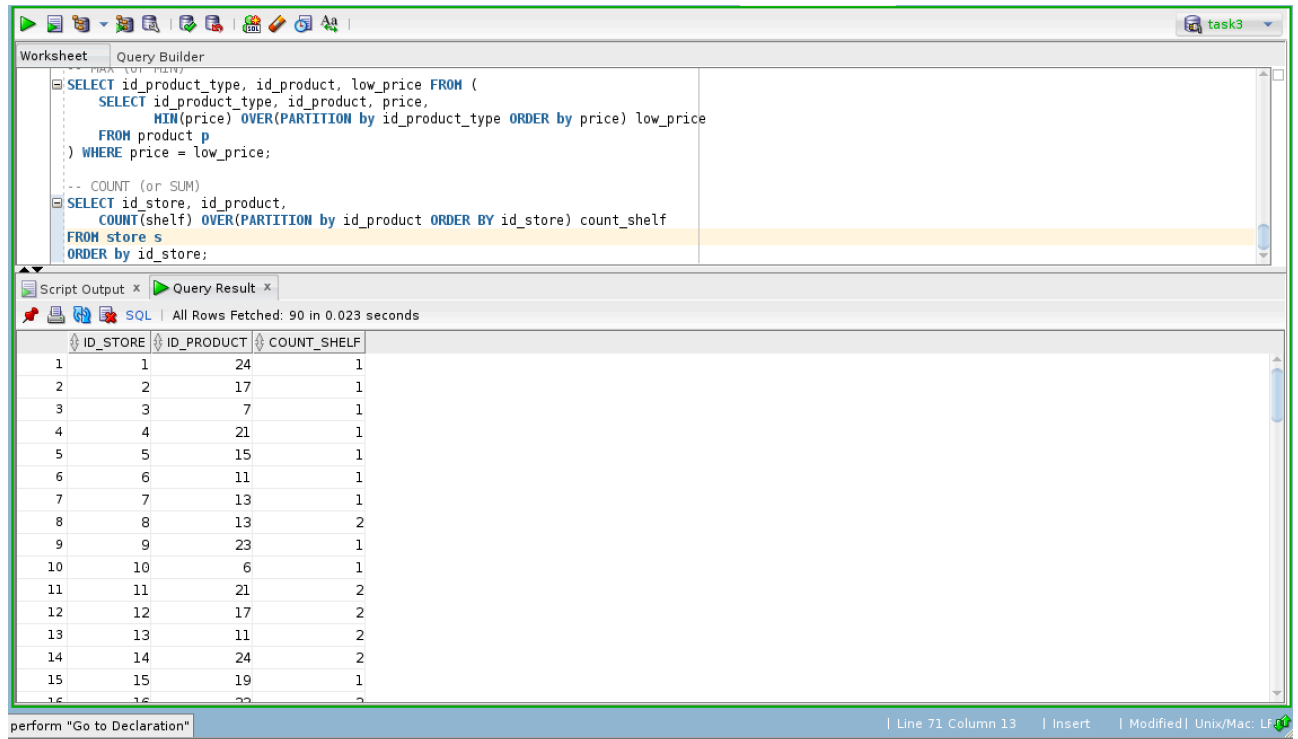
-- COUNT( або SUM)
SELECT id_store, id_product,

        COUNT(shelf) OVER(PARTITION by id_product ORDER BY id_store) count_shelf

FROM store s

ORDER by id_store;

```



The screenshot shows a database query editor with a 'Query Builder' tab. The query is as follows:

```

-- COUNT( або SUM)
SELECT id_store, id_product,
        COUNT(shelf) OVER(PARTITION by id_product ORDER BY id_store) count_shelf
FROM store s
ORDER by id_store;

```

Below the query, the 'Query Result' tab displays the results of the query. The results are shown in a table with the following columns: ID_STORE, ID_PRODUCT, and COUNT_SHELF. The table contains 16 rows of data.

ID_STORE	ID_PRODUCT	COUNT_SHELF
1	1	24
2	2	17
3	3	7
4	4	21
5	5	15
6	6	11
7	7	13
8	8	13
9	9	23
10	10	6
11	11	21
12	12	17
13	13	11
14	14	24
15	15	19
16	16	22

The status bar at the bottom indicates 'Line 71 Column 13 | Insert | Modified | Unix/Mac: LF'.

7. Вивести список товарів, які було придбано за певний період часу вивести вартість товарів по різних вікнам використовуючи наступну структуру

```

1. ...over( order by )
SELECT p.id_product_type, p.id_product, p.price, i.purchase_time datep,
        SUM(p.price) OVER(ORDER by i.purchase_time)

FROM invoice i

JOIN invoice_detail id

ON i.id_invoice = id.id_invoice

JOIN product p

ON id.id_product = p.id_product

WHERE i.purchase_time BETWEEN to_date('06.12.2011', 'dd.mm.yyyy') and
        to_date('10.02.2012', 'dd.mm.yyyy');

```

Worksheet Query Builder

```
-- 7
SELECT p.id_product_type, p.id_product, p.price, i.purchase_time datep,
       SUM(p.price) OVER(ORDER by i.purchase_time)
FROM invoice i
JOIN invoice_detail id
ON i.id_invoice = id.id_invoice
JOIN product p
ON id.id_product = p.id_product
WHERE i.purchase_time between to_date('06.12.2011', 'dd.mm.yyyy') and
                             to_date('10.02.2012', 'dd.mm.yyyy');
```

Script Output x Query Result x

SQL | All Rows Fetched: 7 in 0.003 seconds

ID_PRODUCT_TYPE	ID_PRODUCT	PRICE	DATEP	SUM(P.PRICE)OVER(ORDERBYI.PURCHASE_TIME)
1	7	4	3 06-DEC-11	3
2	6	17	10 13-DEC-11	103
3	7	23	7 14-DEC-11	110
4	1	7	78 14-JAN-12	188
5	1	7	78 20-JAN-12	266
6	3	12	15 05-FEB-12	281
7	5	14	25 07-FEB-12	306

perform "Go to Declaration" | Line 79 Column 32 | Insert | Modified | Unix/Mac: LF

2. ...over (partition ... order by...)

```
SELECT p.id_product_type, p.id_product, p.price,
       SUM(p.price) OVER(PARTITION by p.id_product_type ORDER by p.price rows
1 preceding) as asd
FROM product p;
```

Worksheet Query Builder

```
WHERE i.purchase_time between to_date('06.12.2011', 'dd.mm.yyyy') and
                             to_date('10.02.2012', 'dd.mm.yyyy');

SELECT p.id_product_type, p.id_product, p.price,
       SUM(p.price) OVER(PARTITION by p.id_product_type ORDER by p.price rows 1 preceding) as asd
FROM product p;

SELECT id_product_type, id_unit, price,
       MAX(price) OVER(PARTITION by id_product, id_unit) max_price_per_type_and_pro
FROM product
ORDER BY id_product_type, id_unit;
```

Script Output x Query Result x

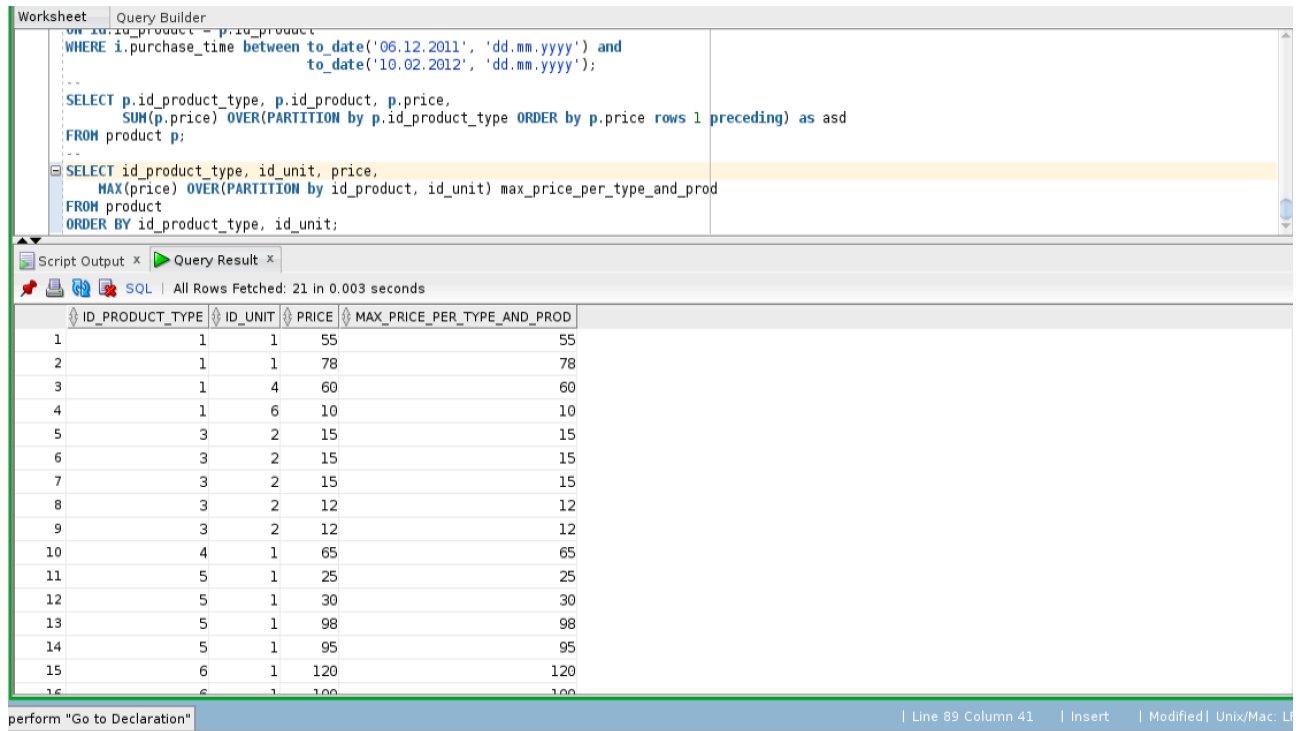
SQL | All Rows Fetched: 21 in 0.014 seconds

ID_PRODUCT_TYPE	ID_PRODUCT	PRICE	ASD
1	1	2	10
2	1	6	55
3	1	8	60
4	1	7	78
5	3	9	12
6	3	11	12
7	3	12	15
8	3	16	15
9	3	13	15
10	4	10	65
11	5	14	25
12	5	15	30
13	5	21	95
14	5	20	98
15	6	17	100
16	6	10	100

perform "Go to Declaration" | Line 85 Column 6 | Insert | Modified | Unix/Mac: LF

3. ... over (partition col1, col2)

```
SELECT id_product_type, id_unit, price,  
       MAX(price) OVER(PARTITION by id_product, id_unit)  
max_price_per_type_and_prod  
FROM product  
ORDER BY id_product_type, id_unit;
```



The screenshot shows a database query editor with a 'Query Builder' tab. The SQL query is as follows:

```
SELECT id_product_type, id_unit, price,  
       MAX(price) OVER(PARTITION by id_product, id_unit)  
max_price_per_type_and_prod  
FROM product  
ORDER BY id_product_type, id_unit;
```

The query results are displayed in a table with 4 columns: ID_PRODUCT_TYPE, ID_UNIT, PRICE, and MAX_PRICE_PER_TYPE_AND_PROD. The results are sorted by ID_PRODUCT_TYPE and then by ID_UNIT.

ID_PRODUCT_TYPE	ID_UNIT	PRICE	MAX_PRICE_PER_TYPE_AND_PROD
1	1	55	55
2	1	78	78
3	1	60	60
4	1	10	10
5	3	15	15
6	3	15	15
7	3	15	15
8	3	12	12
9	3	12	12
10	4	65	65
11	5	25	25
12	5	30	30
13	5	98	98
14	5	95	95
15	6	120	120
16	6	100	100

The status bar at the bottom indicates 'Line 89 Column 41' and 'Insert' mode.

4. ... over (partition col1, col2... order by)

```
SELECT id_product_type, id_unit, price,  
  
       MAX(price) OVER(PARTITION by id_product, id_unit ORDER BY  
id_product_type) max_price_per_type_and_prod  
  
FROM product  
  
ORDER BY id_product_type, id_unit;
```

Worksheet

Query Builder

on i.id_product = p.id_product

WHERE i.purchase_time between to_date('06.12.2011', 'dd.mm.yyyy') and to_date('10.02.2012', 'dd.mm.yyyy');

--

SELECT p.id_product_type, p.id_product, p.price, SUM(p.price) OVER(PARTITION by p.id_product_type ORDER by p.price rows 1 preceding) as asd

FROM product p;

--

SELECT id_product_type, id_unit, price, MAX(price) OVER(PARTITION by id_product, id_unit ORDER BY id_product_type) max_price_per_type_and_prod

FROM product

ORDER BY id_product_type, id_unit;

Script Output x

Query Result x

SQL

All Rows Fetched: 21 in 0.007 seconds

	ID_PRODUCT_TYPE	ID_UNIT	PRICE	MAX_PRICE_PER_TYPE_AND_PROD
1	1	1	55	55
2	1	1	78	78
3	1	4	60	60
4	1	6	10	10
5	3	2	15	15
6	3	2	15	15
7	3	2	15	15
8	3	2	12	12
9	3	2	12	12
10	4	1	65	65
11	5	1	25	25
12	5	1	30	30
13	5	1	98	98
14	5	1	95	95
15	6	1	120	120
16	6	1	100	100

perform "Go to Declaration"

Line 92 Column 35 | Insert | Modified | Unix/Mac: L

perform "Go to Declaration"

| Line 92 Column 35 | Insert | Modified | Unix/Mac: LF