Lab report #6

Sadovskaya Veronika

GitHub: https://github.com/sdveronika/DataMola22

Task 1 - Create Ad Hoc SQL FIRST_VALUE, LAST_VALUE

1) Create Ad Hoc SQL FIRST_VALUE

For an example of how this function works, we used a query that displays the highest profit for each country for the month (the data turned out to be the same, since a cross join was used when generating orders):

```
SELECT order_date, country_r,
       FIRST_VALUE(PROFIT) OVER (PARTITION BY order_date, country_r ORDER BY profit DESC
       RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS highest_profit_in_the_country_in_a_month
FROM(
SELECT TRUNC(order_date,'MM') AS order_date, country_r, SUM(price) AS profit
FROM dw_cl.cl_t_transaction
GROUP BY TRUNC(order_date,'MM'),country_r)
WHERE country_r IN (SELECT DISTINCT country_r FROM dw_cl.cl_t_transaction)
ORDER BY 1;
```

Query Result | SQL | Fetched 50 rows in 0,327 seconds

	ORDER_DATE	COUNTRY_R	HIGHEST_PROFIT_IN_THE_COUNTRY_IN_A_MONTH
1	01.01.21	Belarus	314650
2	01.01.21	Poland	314650
3	01.01.21	Russia	314650
4	01.01.21	Ukraine	314650
5	01.01.21	USA	314650
6	01.02.21	Belarus	284200
7	01.02.21	Poland	284200
8	01.02.21	Russia	284200
9	01.02.21	Ukraine	284200
10	01.02.21	USA	284200
11	01.03.21	Belarus	314650
12	01.03.21	Poland	314650

2) Create Ad Hoc SQL LAST_VALUE

For an example of how this function works, we used a query that displays the lowest profit for each country for the month (the data turned out to be the same, since a cross join was used when generating orders):

```
SELECT order_date, country_r,
       LAST_VALUE(PROFIT) OVER (PARTITION BY order_date, country_r ORDER BY profit DESC
       RANGE BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) AS lowest_profit_in_the_country_in_a_month
FROM(
SELECT TRUNC(order_date,'MM') AS order_date, country_r, SUM(price) AS profit
FROM dw_cl.cl_t_transaction
GROUP BY TRUNC(order_date,'MM'),country_r)
WHERE country_r IN (SELECT DISTINCT country_r FROM dw_cl.cl_t_transaction)
ORDER BY 1;
```

Query Result | SQL | Fetched 50 rows in 0,354 seconds

	ORDER_DATE	COUNTRY_R	LOWEST_PROFIT_IN_THE_COUNTRY_IN_A_MONTH
1	01.01.21	Belarus	314650
2	01.01.21	Poland	314650
3	01.01.21	Russia	314650
4	01.01.21	Ukraine	314650
5	01.01.21	USA	314650
6	01.02.21	Belarus	284200
7	01.02.21	Poland	284200
8	01.02.21	Russia	284200
9	01.02.21	Ukraine	284200
10	01.02.21	USA	284200
11	01.03.21	Belarus	314650
12	01.03.21	Poland	314650

Task 2 - Create Ad Hoc SQL RANK, DENSE_RANK, ROWNUM

1) Create Ad Hoc SQL RANK

Let's show an example of how the rank function works on a table containing information about dishes. Select from table dw_cl.cl_t_dish:

```
1  SELECT * FROM dw_cl.cl_t_dish;
```

Script Output | Query Result | SQL | All Rows Fetched: 5 in 0,023 seconds

	DISH_NAME	DISH_CATEGORY	PRICE	COMPOSITION	WEIGHT	DISH_STATUS
1	chebupelli	hot	61	chebupelli ingredients	512	Y
2	pasta	hot	87	pasta ingredients	959	Y
3	soup	hot	25	soup ingredients	915	Y
4	pizza	hot	25	pizza ingredients	486	Y
5	greek salad	appetizer	82	greek salad ingredients	797	Y

```
6  SELECT dish_name, dish_category, price, rank() OVER ( ORDER BY price ASC) price_rank
7  FROM dw_cl.cl_t_dish
8  ORDER BY price_rank;
```

Query Result | SQL | All Rows Fetched: 5 in 0,034 seconds

	DISH_NAME	DISH_CATEGORY	PRICE	PRICE_RANK
1	soup	hot	25	1
2	pizza	hot	25	1
3	chebupelli	hot	61	3
4	greek salad	appetizer	82	4
5	pasta	hot	87	5

The Oracle/PLSQL RANK function returns the rank of a value in a group of values. It is very similar to the DENSE_RANK function. However, the rank function can cause non-consecutive rankings if the tested values are the same. Whereas, the DENSE_RANK function will always result in consecutive rankings.

2) Create Ad Hoc SQL DENSE_RANK

Execute the previous query, only use the dense_rank function:

```
10 SELECT dish_name, dish_category, price, DENSE_rank() OVER ( ORDER BY price ASC) price_rank
11 FROM dw_cl.cl_t_dish
12 ORDER BY price_rank;
```

Query Result | SQL | All Rows Fetched: 5 in 0,021 seconds

	DISH_NAME	DISH_CATEGORY	PRICE	PRICE_RANK
1	soup	hot	25	1
2	pizza	hot	25	1
3	chebupelli	hot	61	2
4	greek salad	appetizer	82	3
5	pasta	hot	87	4

When using the rank function, the value 2 was missed, but when using dense_rank, all values were displayed sequentially, without missing values.

Unlike the RANK() function, the DENSE_RANK() function returns rank values as consecutive integers. It does not skip rank in case of ties. Rows with the same values for the rank criteria will receive the same rank values.

3) Create Ad Hoc SQL ROWNUM

An example of the operation of the row_number function will be shown in the previous query using the dense_rank function, adding a field calculated using the row_number function:

```
14  SELECT dish_name, dish_category, price,
15         DENSE_rank() OVER ( ORDER BY price ASC) price_rank,
16         ROW_NUMBER() OVER (ORDER BY price ASC) price_row_number
17  FROM dw_cl.cl_t_dish
18  ORDER BY price_rank;
```

Query Result

SQL | All Rows Fetched: 5 in 0,025 seconds

	DISH_NAME	DISH_CATEGORY	PRICE	PRICE_RANK	PRICE_ROW_NUMBER
1	soup	hot	25	1	1
2	pizza	hot	25	1	2
3	chebupelli	hot	61	2	3
4	greek salad	appetizer	82	3	4
5	pasta	hot	87	4	5

The row_number function assigns a unique number for each row in the ordered result set. If the partitioning clause is specified, then each row is assigned a number unique within a data partition, based upon its position in the sort order in that partition.

Task 3 - Create Ad Hoc SQL AGGREAGATE FUNCS

1) Create Ad Hoc SQL MAX()

An example of the operation of the MAX() function will be shown on a query that displays the maximum order amount in each city for a month:

```
1  SELECT TRUNC(order_date, 'MM') as order_date, country_r, city_r, MAX(total_cost) AS max_total_cost
2  FROM dw_cl.cl_t_transaction
3  GROUP BY TRUNC(order_date, 'MM'), country_r, city_r
4  ORDER BY 1, 2;
```

Query Result

SQL | All Rows Fetched: 60 in 0,293 seconds

	ORDER_DATE	COUNTRY_R	CITY_R	MAX_TOTAL_COST
1	01.01.21	Belarus	Minsk	1499
2	01.01.21	Poland	Warsaw	1499
3	01.01.21	Russia	Moscow	1499
4	01.01.21	Ukraine	Kiev	1499
5	01.01.21	USA	New York	1499
6	01.02.21	Belarus	Minsk	1499
7	01.02.21	Poland	Warsaw	1499
8	01.02.21	Russia	Moscow	1499
9	01.02.21	Ukraine	Kiev	1499
10	01.02.21	USA	New York	1499
11	01.03.21	Belarus	Minsk	1499
12	01.03.21	Poland	Warsaw	1499
13	01.03.21	Russia	Moscow	1499
14	01.03.21	Ukraine	Kiev	1499
15	01.03.21	USA	New York	1499
16	01.04.21	Belarus	Minsk	1499

2) Create Ad Hoc SQL MIN()

An example of the operation of the MIN() function will be shown on a query that displays the minimum order amount in each city for a month:

```
SELECT TRUNC(order_date, 'MM') as order_date, country_r, city_r, MIN(total_cost) AS avg_total_cost
FROM dw_cl.cl_t_transaction
GROUP BY TRUNC(order_date, 'MM'), country_r, city_r
ORDER BY 1, 2;
```

Query Result | SQL | Fetched 50 rows in 0,214 seconds

	ORDER_DATE	COUNTRY_R	CITY_R	AVG_TOTAL_COST
1	01.01.21	Belarus	Minsk	5
2	01.01.21	Poland	Warsaw	5
3	01.01.21	Russia	Moscow	5
4	01.01.21	Ukraine	Kiev	5
5	01.01.21	USA	New York	7
6	01.02.21	Belarus	Minsk	5
7	01.02.21	Poland	Warsaw	5
8	01.02.21	Russia	Moscow	5
9	01.02.21	Ukraine	Kiev	5
10	01.02.21	USA	New York	5
11	01.03.21	Belarus	Minsk	5
12	01.03.21	Poland	Warsaw	5
13	01.03.21	Russia	Moscow	5
14	01.03.21	Ukraine	Kiev	5

3) Create Ad Hoc SQL AVG()

An example of the operation of the AVG() function will be shown on a query that displays the average order amount in each city for a month:

```
SELECT TRUNC(order_date, 'MM') as order_date, country_r, city_r, AVG(total_cost) AS avg_total_cost
FROM dw_cl.cl_t_transaction
GROUP BY TRUNC(order_date, 'MM'), country_r, city_r
ORDER BY 1, 2;
```

Query Result | SQL | Fetched 50 rows in 0,199 seconds

	ORDER_DATE	COUNTRY_R	CITY_R	AVG_TOTAL_COST
1	01.01.21	Belarus	Minsk	752,739612903225806451612903225806451613
2	01.01.21	Poland	Warsaw	756,832516129032258064516129032258064516
3	01.01.21	Russia	Moscow	751,755354838709677419354838709677419355
4	01.01.21	Ukraine	Kiev	751,933419354838709677419354838709677419
5	01.01.21	USA	New York	749,474838709677419354838709677419354839
6	01.02.21	Belarus	Minsk	752,372142857142857142857142857142857143
7	01.02.21	Poland	Warsaw	746,372
8	01.02.21	Russia	Moscow	748,385142857142857142857142857142857143
9	01.02.21	Ukraine	Kiev	750,486428571428571428571428571428571429
10	01.02.21	USA	New York	748,220428571428571428571428571428571429
11	01.03.21	Belarus	Minsk	747,204774193548387096774193548387096774
12	01.03.21	Poland	Warsaw	745,504774193548387096774193548387096774
13	01.03.21	Russia	Moscow	748,962967741935483870967741935483870968
14	01.03.21	Ukraine	Kiev	753,638193548387096774193548387096774194