ondia

Relational DB & SQL

Session 9

Window Functions

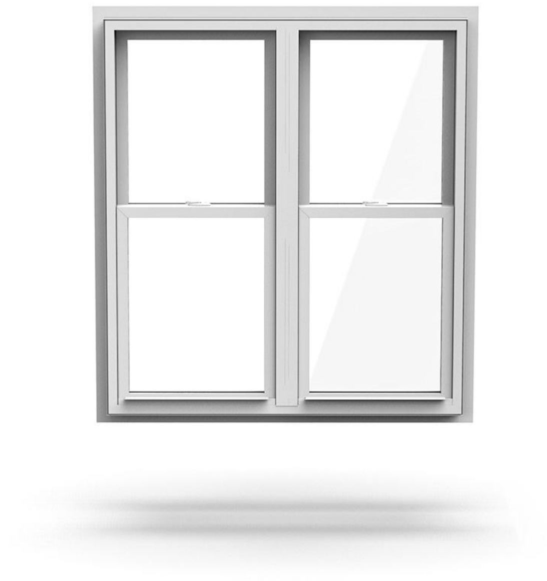
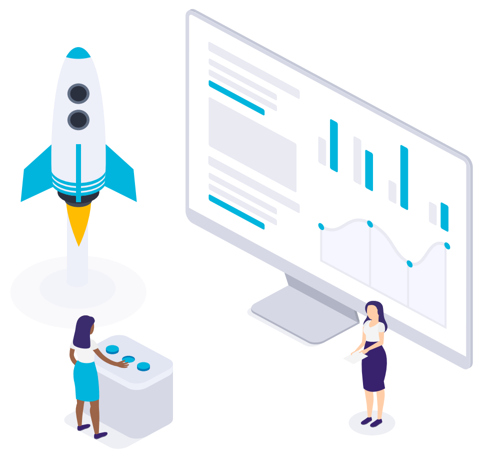
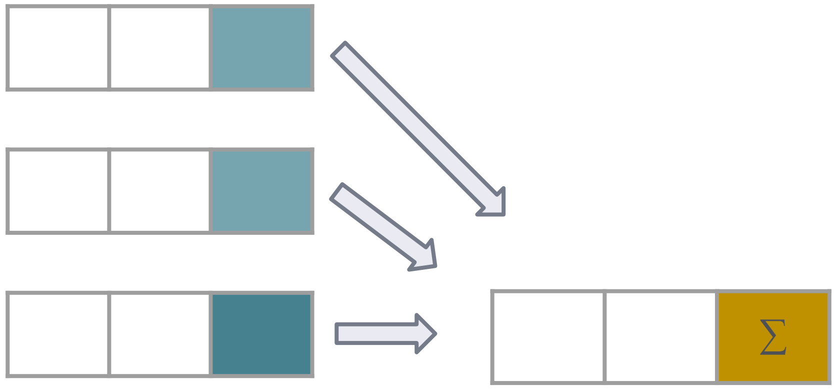


Table of Contents

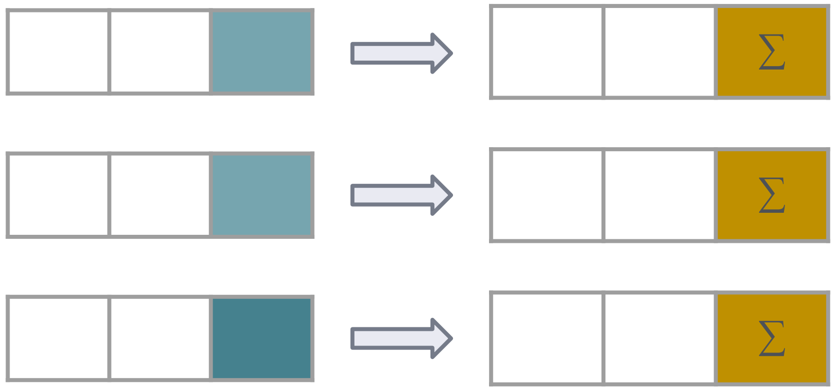
Window Functions (WF) vs. GROUP BY



Aggregate with GROUP BY



Aggregate with WF



WF vs. GROUP BY

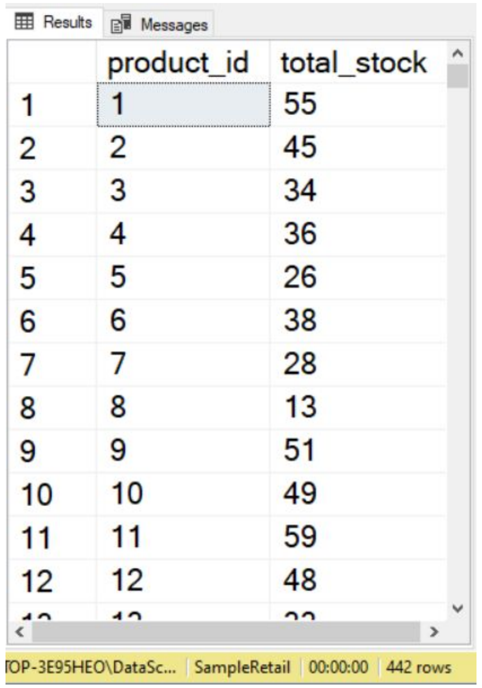
	GROUP BY	Window Functions
Distinct	necessity	optional
Aggregating	necessity	optional
Ordering	invalid	valid
Performance	slower	faster
Dependency on Selected Fields	dependent	independent

Query Time

Question: Write a query that shows the total stock amount of each product in the stock table.

(Use both of Group by and WF)

Expected Output:



	product_id	total_stock
1	1	55
2	2	45
3	3	34
4	4	36
5	5	26
6	6	38
7	7	28
8	8	13
9	9	51
10	10	49
11	11	59
12	12	48

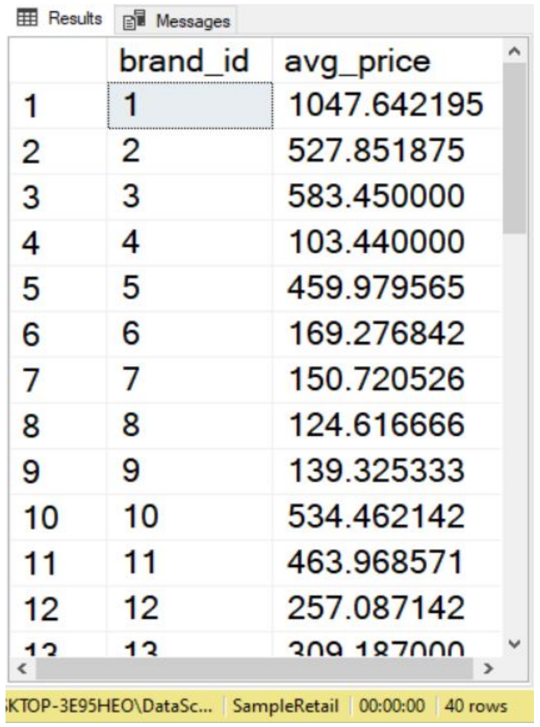
```
TOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 442 rows
```

Query Time For You

Question: Write a query that returns average product prices of brands.

(Use both of Group by and WF)

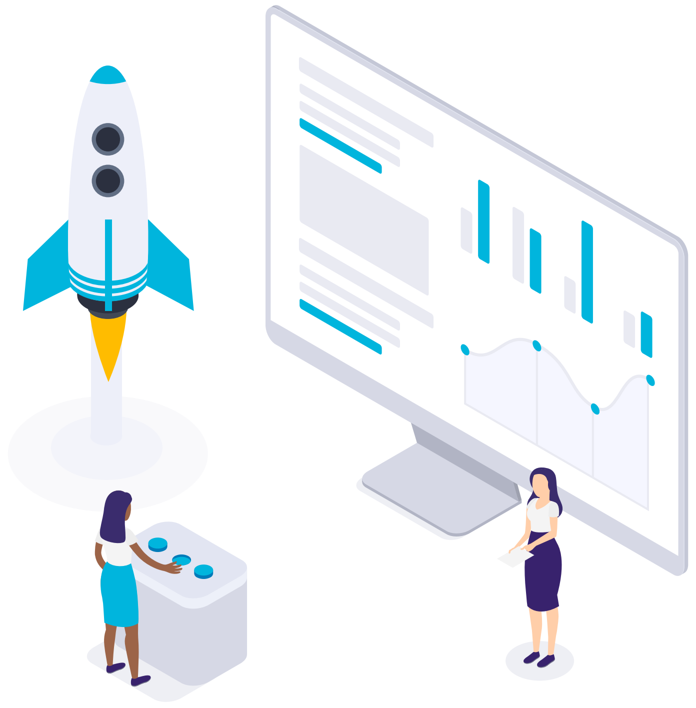
Expected Output:



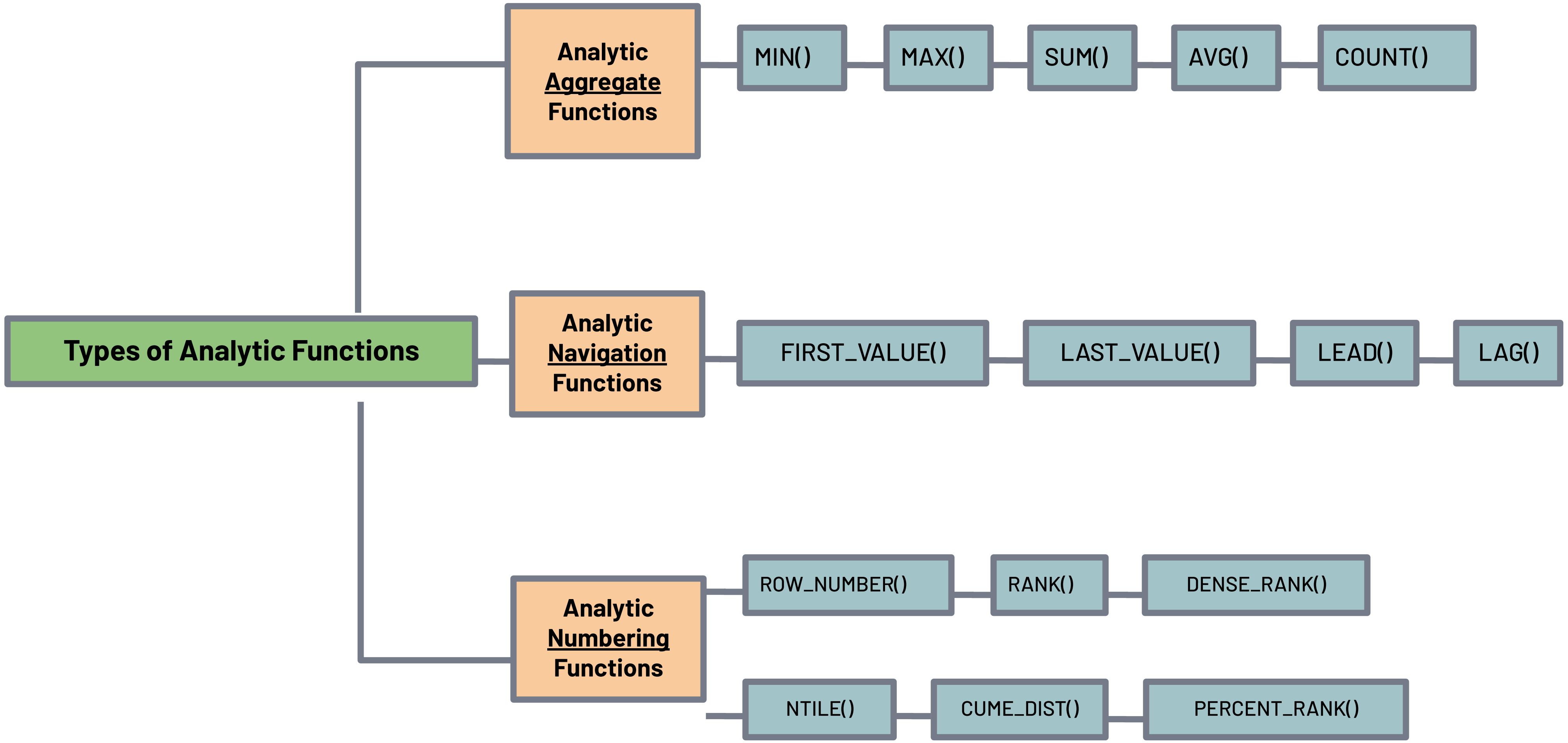
	brand_id	avg_price
1	1	1047.642195
2	2	527.851875
3	3	583.450000
4	4	103.440000
5	5	459.979565
6	6	169.276842
7	7	150.720526
8	8	124.616666
9	9	139.325333
10	10	534.462142
11	11	463.968571
12	12	257.087142
13	13	309.187000

KTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 40 rows

Types of WF



Types of WF


Types of Analytic Functions
Analytic Aggregate Functions
MIN()
MAX()
SUM()
AVG()
COUNT()
Analytic Navigation Functions
FIRST_VALUE()
LAST_VALUE()
LEAD()
LAG()
Analytic Numbering Functions
ROW_NUMBER()
RANK()
DENSE_RANK()
NTILE()
CUME_DIST()
PERCENT_RANK()

WF Syntax and Keywords

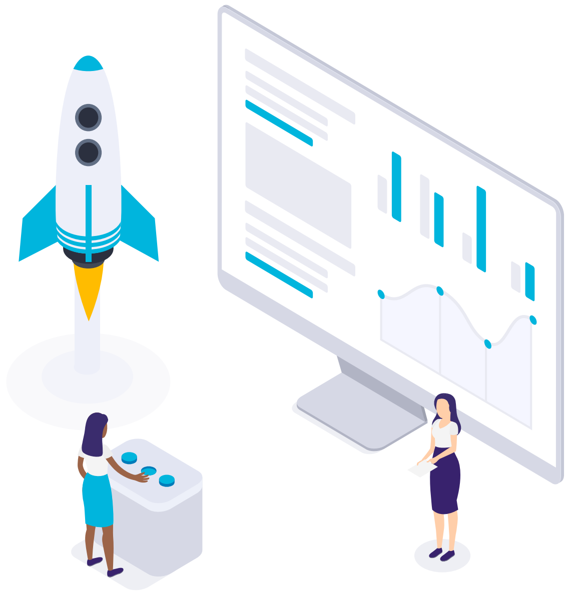
Syntax and Keywords

```
SELECT {columns},
FUNCTION() OVER ( PARTITION BY ... ORDER BY ... WINDOW FRAME )
FROM table1;
```

id	date	time
1	2019-07-05	22
1	2019-04-15	26
2	2019-02-06	28
1	2019-01-02	30
2	2019-08-30	20
2	2019-03-09	22

```
SELECT *,
AVG(time) OVER (
                PARTITION BY  id ORDER BY date
                ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
              ) as avg_time
FROM time_of_sales
```

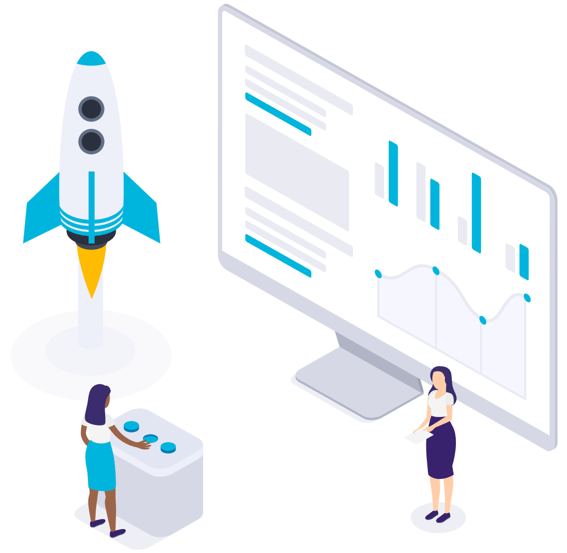
How to Apply WF



Analytic Aggregate Functions

You know how to use
MIN(), MAX(), AVG(), COUNT() or SUM()
functions.

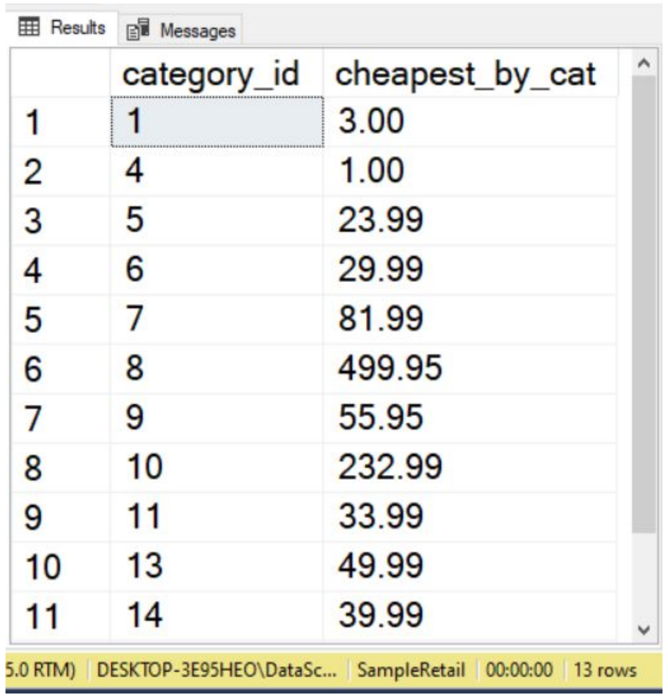
So let’s move on to SQL Server to do some examples.



Query Time

What is the cheapest product price for each category?

Expected Output:



Results | Messages

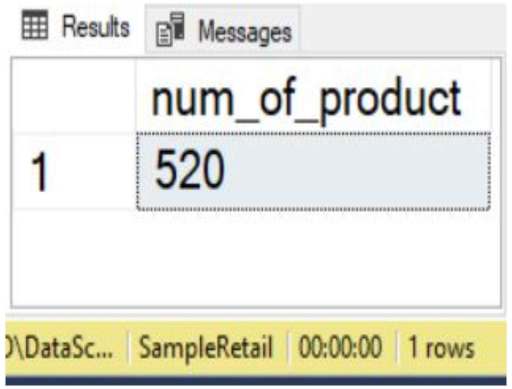
	category_id	cheapest_by_cat
1	1	3.00
2	4	1.00
3	5	23.99
4	6	29.99
5	7	81.99
6	8	499.95
7	9	55.95
8	10	232.99
9	11	33.99
10	13	49.99
11	14	39.99

5.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 13 rows

Query Time

How many different product in the product table?

Expected Output:



Results | Messages

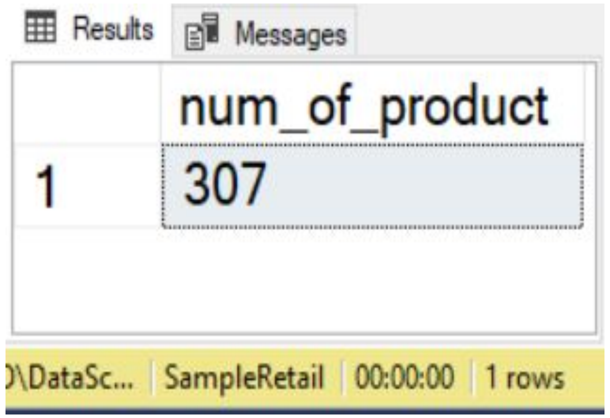
	num_of_product
1	520

```
)\DataSc... | SampleRetail | 00:00:00 | 1 rows
```

Query Time

How many different product in the order_item table?

Expected Output:



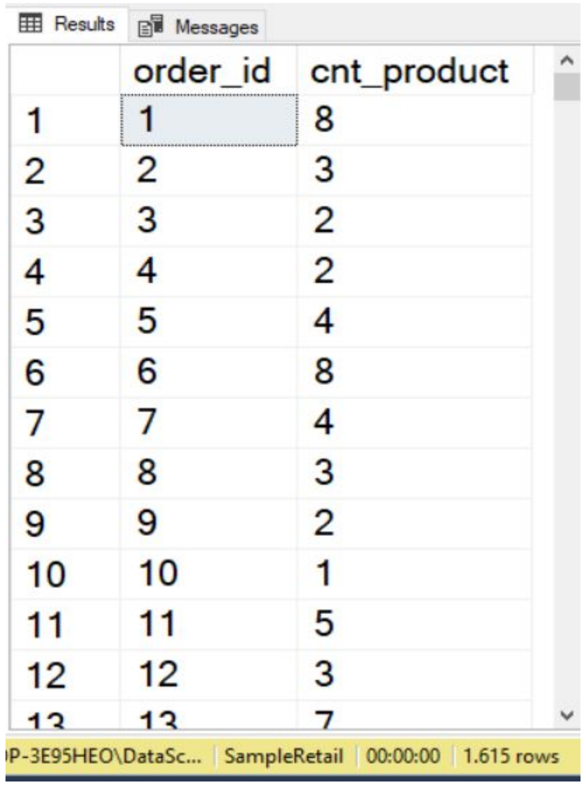
```
Results   Messages
     num_of_product
1    307

)\DataSc... | SampleRetail | 00:00:00 | 1 rows
```

Query Time

Write a query that returns how many products are in each order?

Expected Output:



Results | Messages

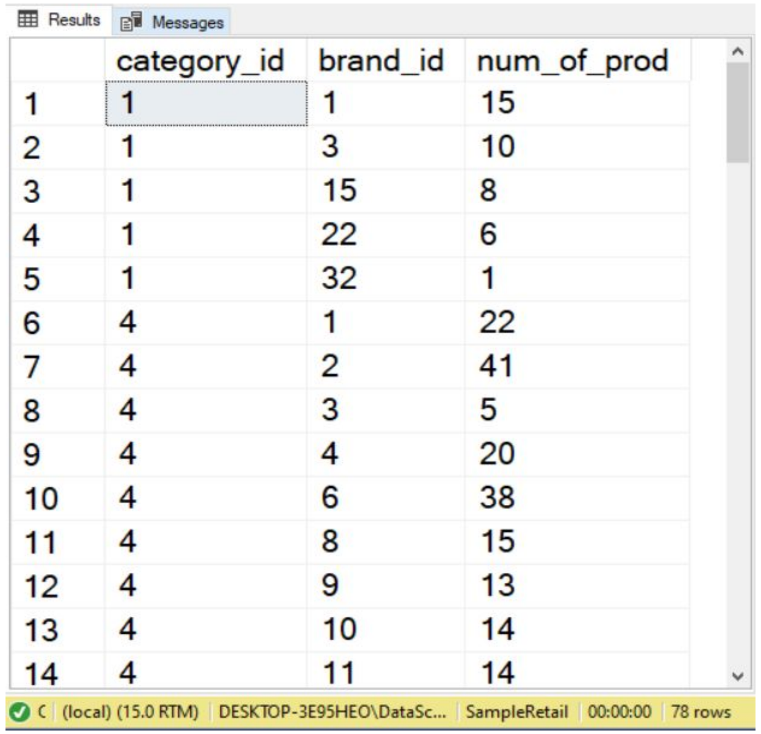
	order_id	cnt_product
1	1	8
2	2	3
3	3	2
4	4	2
5	5	4
6	6	8
7	7	4
8	8	3
9	9	2
10	10	1
11	11	5
12	12	3
13	13	7

P-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows

Query Time

Write a query that returns the number of products in each category of brands.

Expected Output:

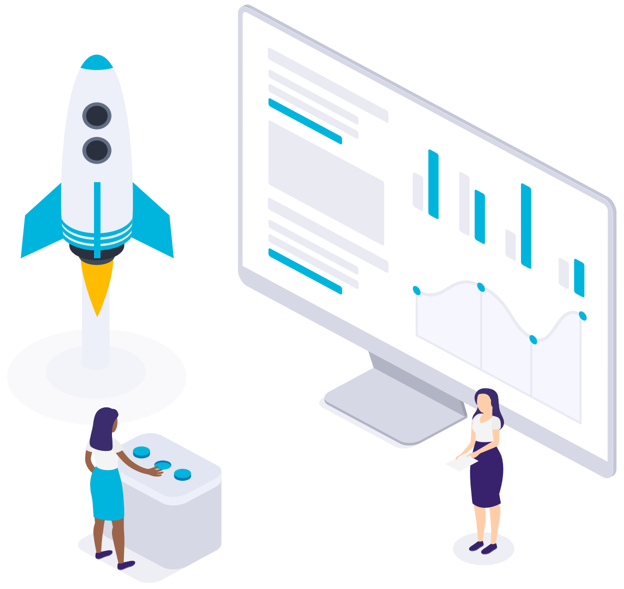


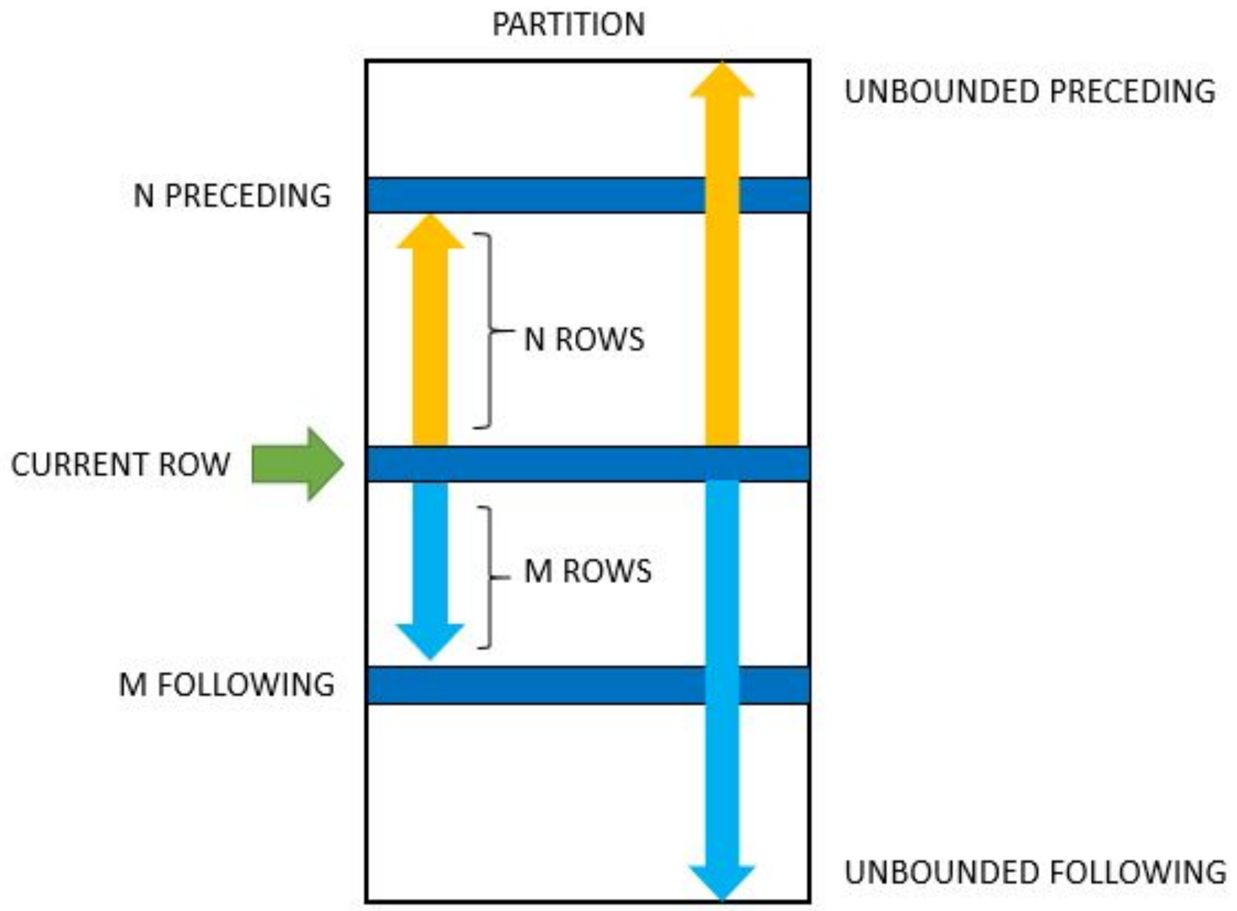
Results | Messages

	category_id	brand_id	num_of_prod
1	1	1	15
2	1	3	10
3	1	15	8
4	1	22	6
5	1	32	1
6	4	1	22
7	4	2	41
8	4	3	5
9	4	4	20
10	4	6	38
11	4	8	15
12	4	9	13
13	4	10	14
14	4	11	14

```
C (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 78 rows
```

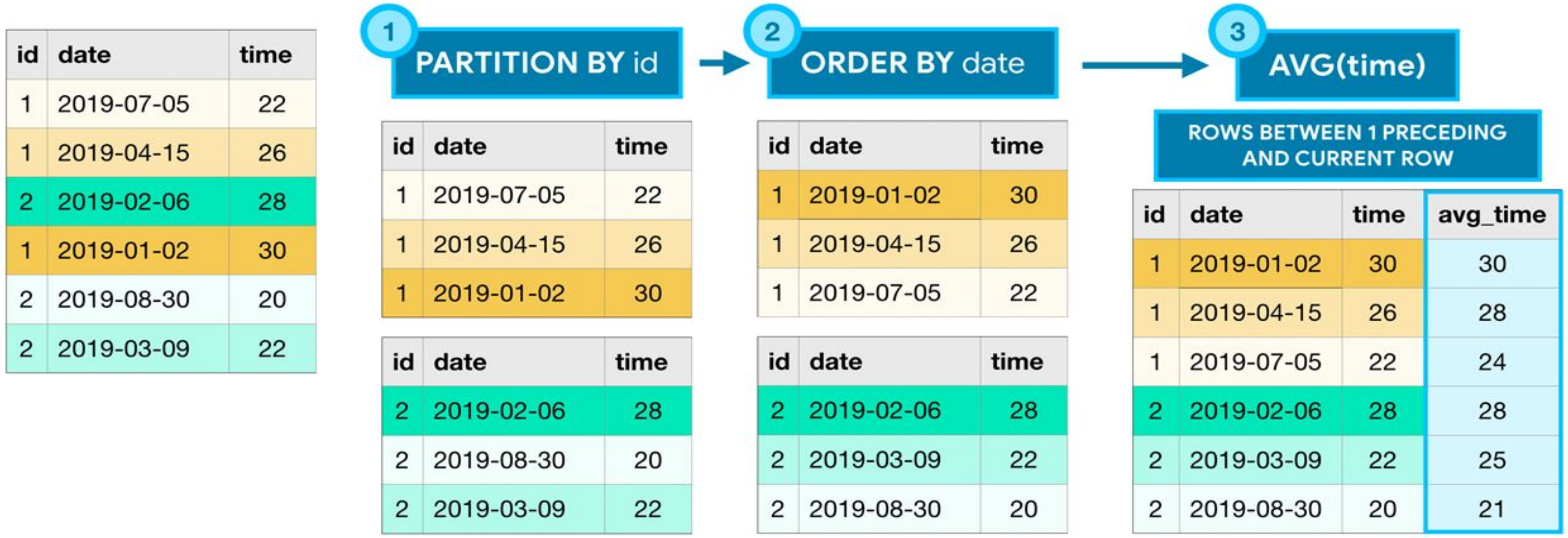
Window Frames





★ **Default:** **UNBOUNDED PRECEDING AND CURRENT ROW**

```
SELECT *,
AVG(time) OVER (
                    PARTITION BY  id ORDER BY date
                    ROWS BETWEEN 1 PRECEDING AND CURRENT ROW
                  ) as avg_time
FROM time_of_sales
```



id	date	time
1	2019-07-05	22
1	2019-04-15	26
2	2019-02-06	28
1	2019-01-02	30
2	2019-08-30	20
2	2019-03-09	22

id	date	time
1	2019-07-05	22
1	2019-04-15	26
1	2019-01-02	30

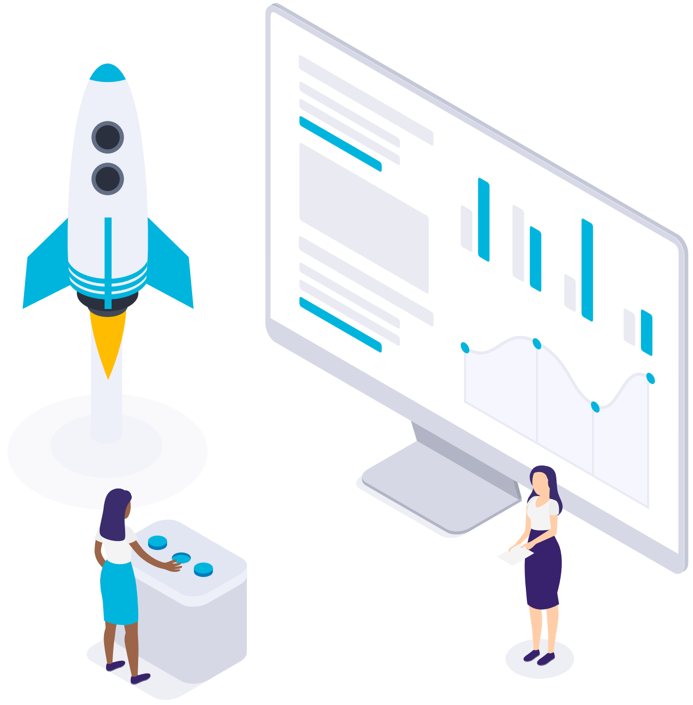
id	date	time
2	2019-02-06	28
2	2019-08-30	20
2	2019-03-09	22

id	date	time
1	2019-01-02	30
1	2019-04-15	26
1	2019-07-05	22

id	date	time
2	2019-02-06	28
2	2019-03-09	22
2	2019-08-30	20

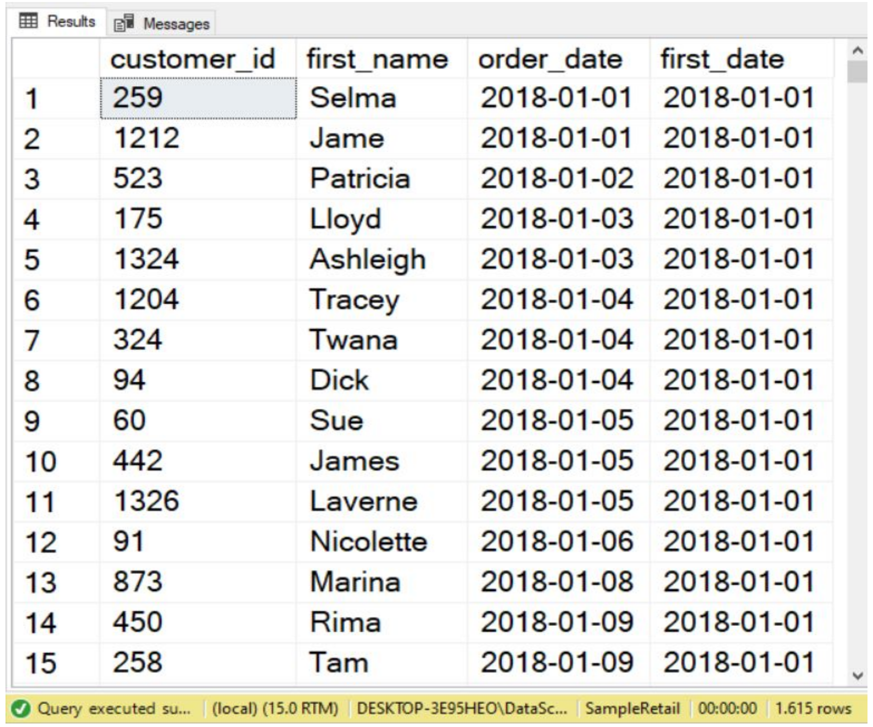
id	date	time	avg_time
1	2019-01-02	30	30
1	2019-04-15	26	28
1	2019-07-05	22	24
2	2019-02-06	28	28
2	2019-03-09	22	25
2	2019-08-30	20	21

Analytic Navigation Functions



First_Value Function

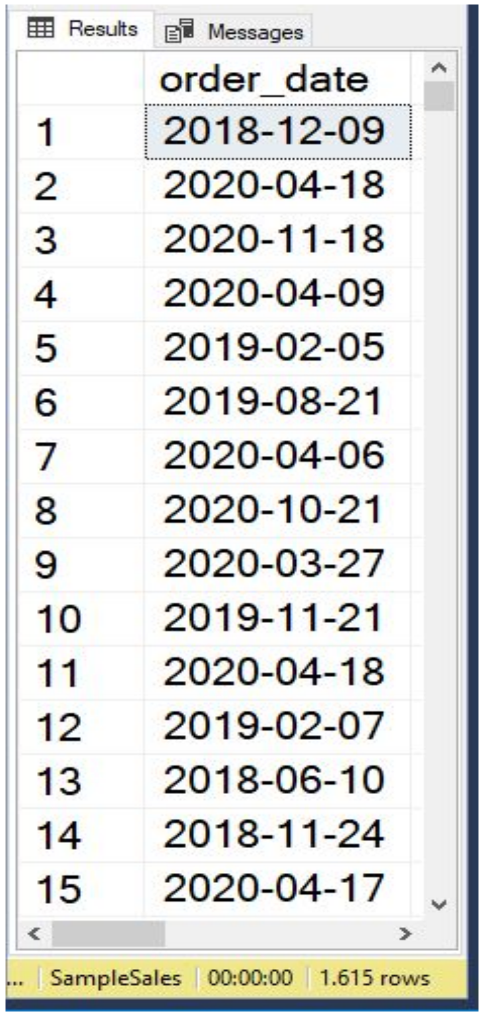
```
SELECT   A.customer_id, A.first_name, B.order_date,
         first_value (order_date) OVER (ORDER BY B.ORDER_DATE) first_date
FROM     sale.customer A, sale.orders B
WHERE    A.customer_id = B.customer_id
```



	customer_id	first_name	order_date	first_date
1	259	Selma	2018-01-01	2018-01-01
2	1212	Jame	2018-01-01	2018-01-01
3	523	Patricia	2018-01-02	2018-01-01
4	175	Lloyd	2018-01-03	2018-01-01
5	1324	Ashleigh	2018-01-03	2018-01-01
6	1204	Tracey	2018-01-04	2018-01-01
7	324	Twana	2018-01-04	2018-01-01
8	94	Dick	2018-01-04	2018-01-01
9	60	Sue	2018-01-05	2018-01-01
10	442	James	2018-01-05	2018-01-01
11	1326	Laverne	2018-01-05	2018-01-01
12	91	Nicolette	2018-01-06	2018-01-01
13	873	Marina	2018-01-08	2018-01-01
14	450	Rima	2018-01-09	2018-01-01
15	258	Tam	2018-01-09	2018-01-01

```
Query executed su... | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows
```

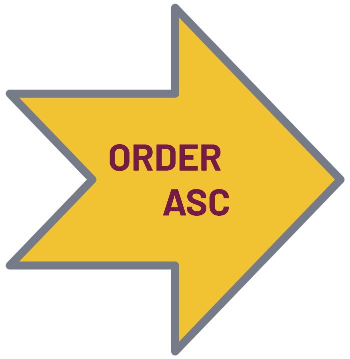
First_Value Function

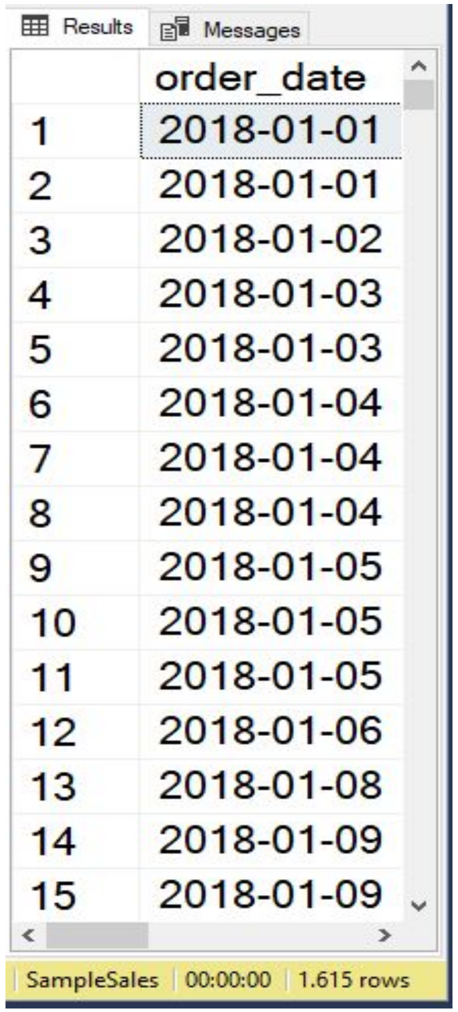


Results | Messages

	order_date
1	2018-12-09
2	2020-04-18
3	2020-11-18
4	2020-04-09
5	2019-02-05
6	2019-08-21
7	2020-04-06
8	2020-10-21
9	2020-03-27
10	2019-11-21
11	2020-04-18
12	2019-02-07
13	2018-06-10
14	2018-11-24
15	2020-04-17

... SampleSales | 00:00:00 | 1.615 rows

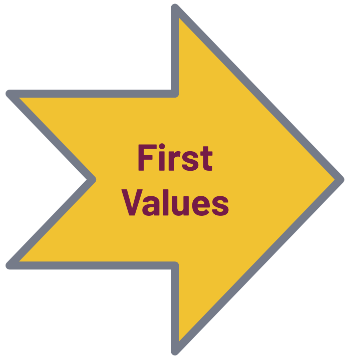


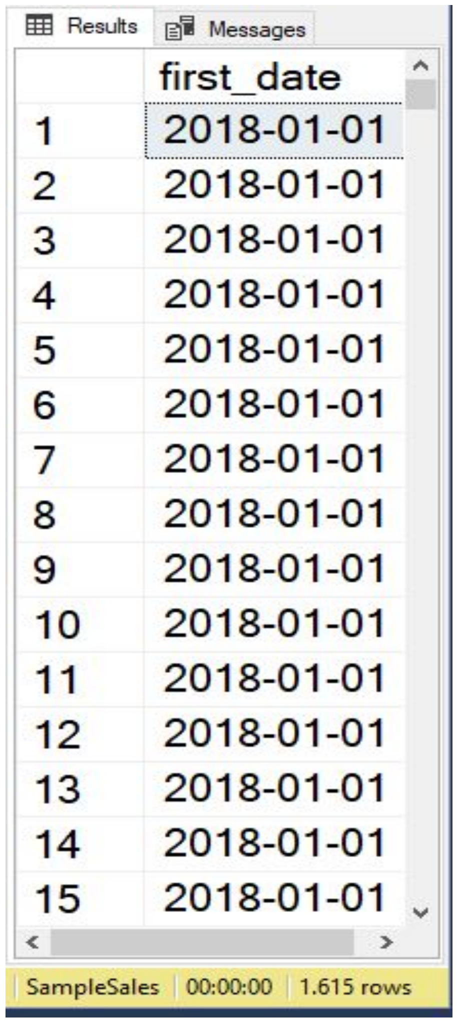


Results | Messages

	order_date
1	2018-01-01
2	2018-01-01
3	2018-01-02
4	2018-01-03
5	2018-01-03
6	2018-01-04
7	2018-01-04
8	2018-01-04
9	2018-01-05
10	2018-01-05
11	2018-01-05
12	2018-01-06
13	2018-01-08
14	2018-01-09
15	2018-01-09

SampleSales | 00:00:00 | 1.615 rows





Results | Messages

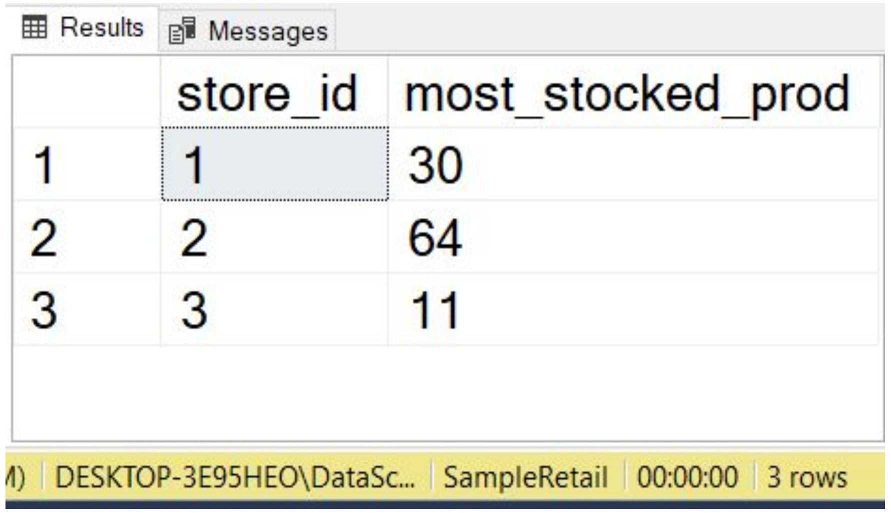
	first_date
1	2018-01-01
2	2018-01-01
3	2018-01-01
4	2018-01-01
5	2018-01-01
6	2018-01-01
7	2018-01-01
8	2018-01-01
9	2018-01-01
10	2018-01-01
11	2018-01-01
12	2018-01-01
13	2018-01-01
14	2018-01-01
15	2018-01-01

SampleSales | 00:00:00 | 1.615 rows

Query Time

Write a query that returns one of the most stocked product in each store.

Expected Output:



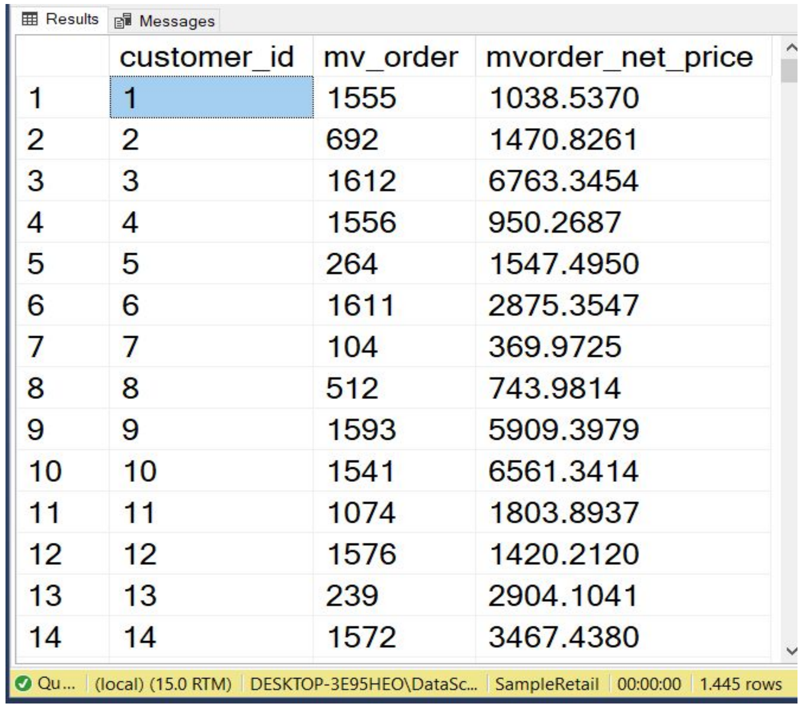
	store_id	most_stocked_prod
1	1	30
2	2	64
3	3	11

```
M) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 3 rows
```

Query Time

Write a query that returns customers and their most valuable order with total amount of it.

Expected Output:



Results | Messages

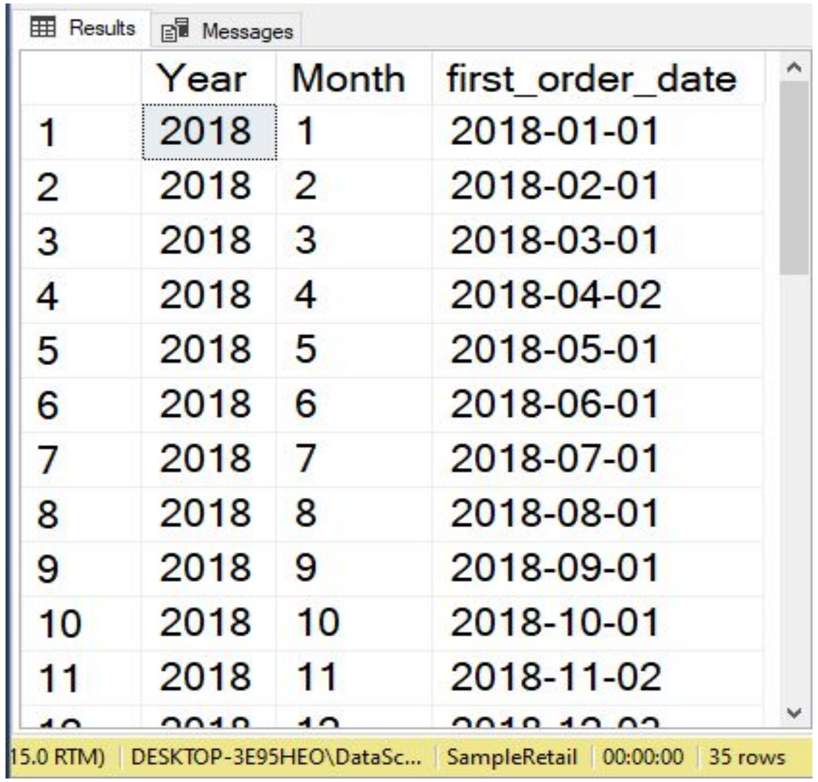
	customer_id	mv_order	mvorder_net_price
1	1	1555	1038.5370
2	2	692	1470.8261
3	3	1612	6763.3454
4	4	1556	950.2687
5	5	264	1547.4950
6	6	1611	2875.3547
7	7	104	369.9725
8	8	512	743.9814
9	9	1593	5909.3979
10	10	1541	6561.3414
11	11	1074	1803.8937
12	12	1576	1420.2120
13	13	239	2904.1041
14	14	1572	3467.4380

Qu... | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.445 rows

Query Time

2. Write a query that returns first order date by month.

Expected Output:



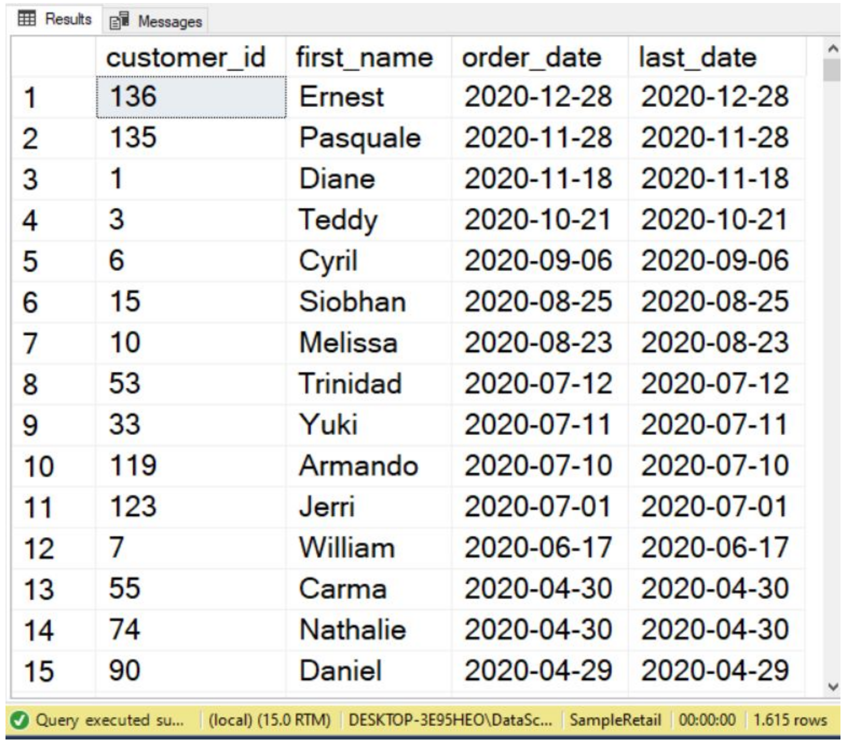
Results | Messages

	Year	Month	first_order_date
1	2018	1	2018-01-01
2	2018	2	2018-02-01
3	2018	3	2018-03-01
4	2018	4	2018-04-02
5	2018	5	2018-05-01
6	2018	6	2018-06-01
7	2018	7	2018-07-01
8	2018	8	2018-08-01
9	2018	9	2018-09-01
10	2018	10	2018-10-01
11	2018	11	2018-11-02

```
15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 35 rows
```

Last_Value Function

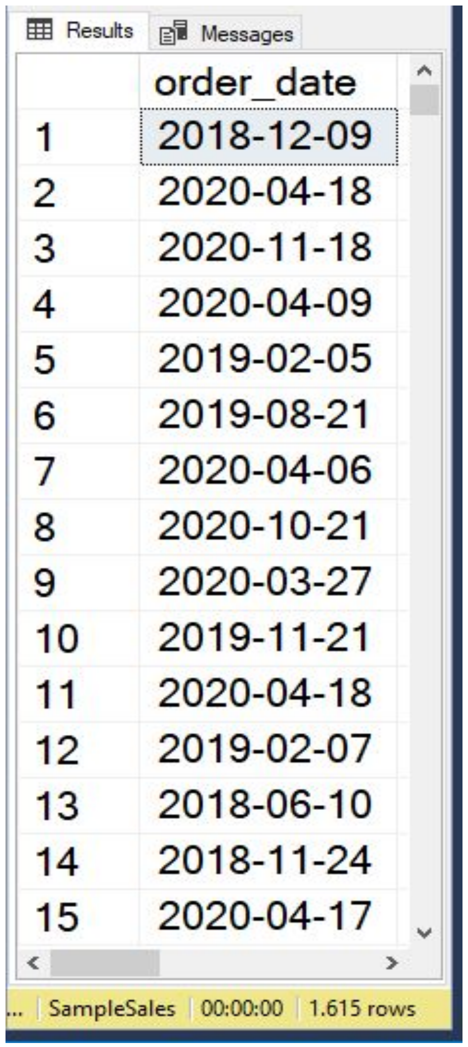
```
SELECT  A.customer_id, A.first_name, B.order_date,
        last_value (order_date) OVER (ORDER BY B.ORDER_DATE DESC) last_date
FROM    sale.customer A, sale.orders B
WHERE   A.customer_id = B.customer_id
```


Results | Messages

	customer_id	first_name	order_date	last_date
1	136	Ernest	2020-12-28	2020-12-28
2	135	Pasquale	2020-11-28	2020-11-28
3	1	Diane	2020-11-18	2020-11-18
4	3	Teddy	2020-10-21	2020-10-21
5	6	Cyril	2020-09-06	2020-09-06
6	15	Siobhan	2020-08-25	2020-08-25
7	10	Melissa	2020-08-23	2020-08-23
8	53	Trinidad	2020-07-12	2020-07-12
9	33	Yuki	2020-07-11	2020-07-11
10	119	Armando	2020-07-10	2020-07-10
11	123	Jerri	2020-07-01	2020-07-01
12	7	William	2020-06-17	2020-06-17
13	55	Carma	2020-04-30	2020-04-30
14	74	Nathalie	2020-04-30	2020-04-30
15	90	Daniel	2020-04-29	2020-04-29

```
Query executed su... | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows
```

Last_Value Function

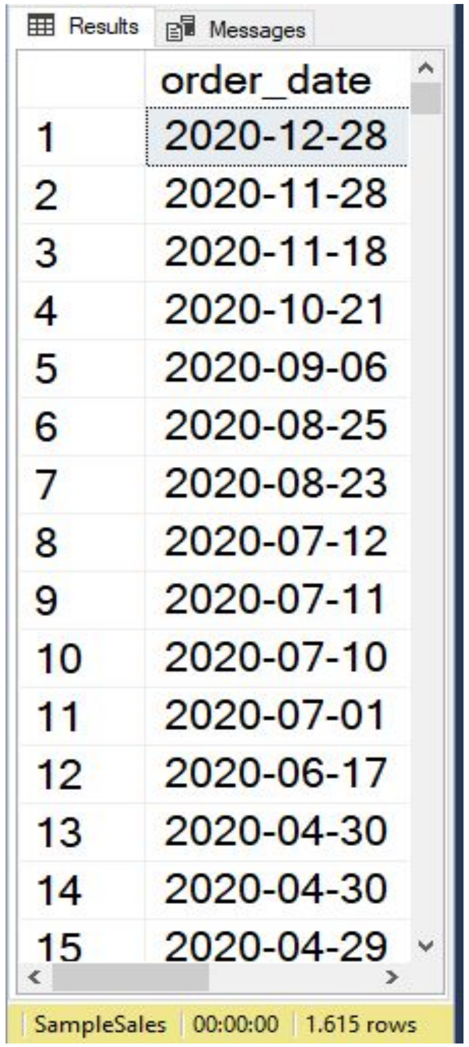


Results | Messages

	order_date
1	2018-12-09
2	2020-04-18
3	2020-11-18
4	2020-04-09
5	2019-02-05
6	2019-08-21
7	2020-04-06
8	2020-10-21
9	2020-03-27
10	2019-11-21
11	2020-04-18
12	2019-02-07
13	2018-06-10
14	2018-11-24
15	2020-04-17

... SampleSales | 00:00:00 | 1.615 rows

ORDER DESC

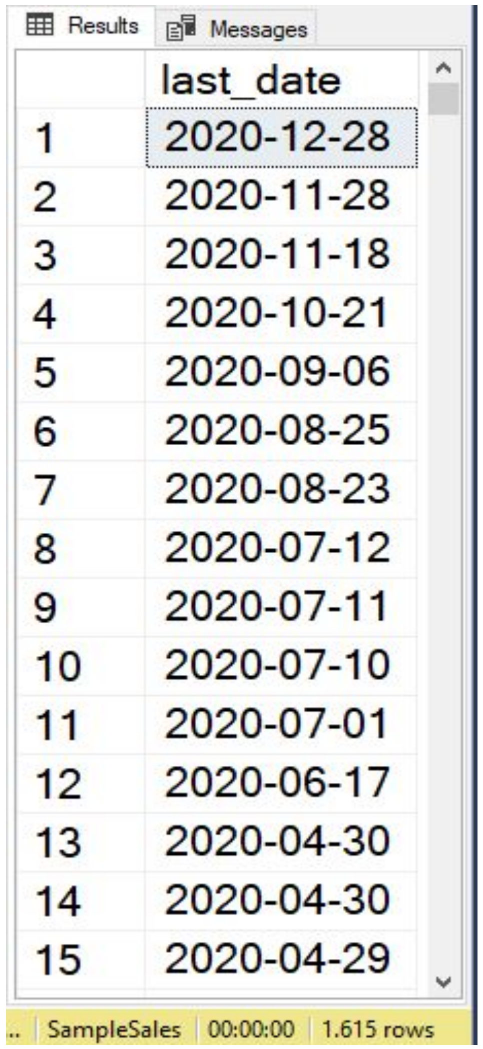


Results | Messages

	order_date
1	2020-12-28
2	2020-11-28
3	2020-11-18
4	2020-10-21
5	2020-09-06
6	2020-08-25
7	2020-08-23
8	2020-07-12
9	2020-07-11
10	2020-07-10
11	2020-07-01
12	2020-06-17
13	2020-04-30
14	2020-04-30
15	2020-04-29

SampleSales | 00:00:00 | 1.615 rows

Last Values



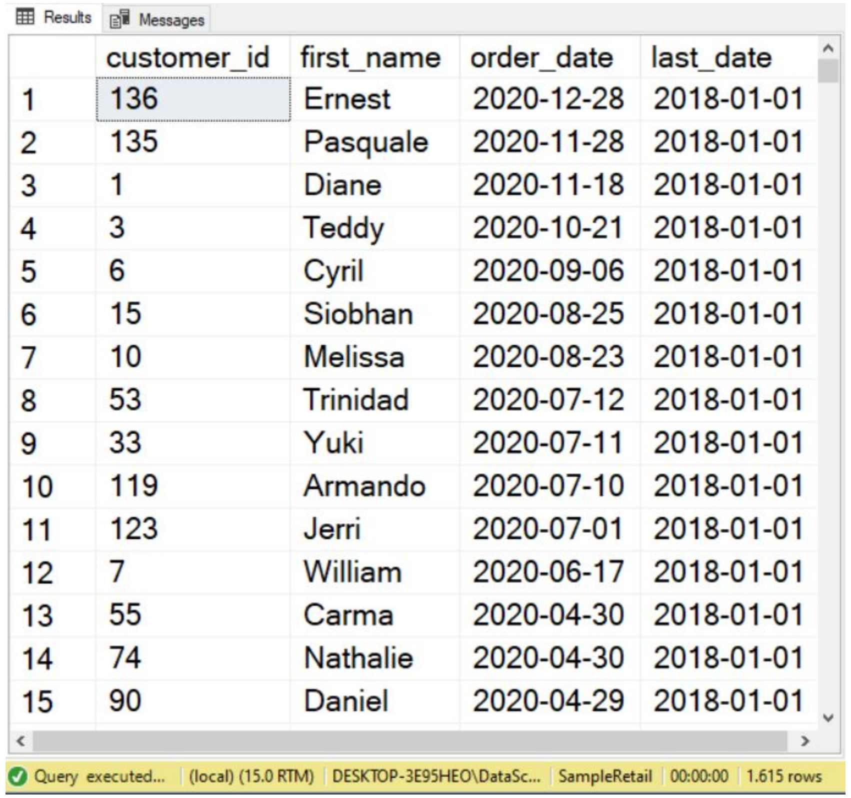
Results | Messages

	last_date
1	2020-12-28
2	2020-11-28
3	2020-11-18
4	2020-10-21
5	2020-09-06
6	2020-08-25
7	2020-08-23
8	2020-07-12
9	2020-07-11
10	2020-07-10
11	2020-07-01
12	2020-06-17
13	2020-04-30
14	2020-04-30
15	2020-04-29

.. SampleSales | 00:00:00 | 1.615 rows

Last_Value Function

```
SELECT   B.customer_id, A.first_name, B.order_date,
         last_value (order_date) OVER (ORDER BY B.order_date DESC ROWS BETWEEN UNBOUNDED PRECEDING AND UNBOUNDED FOLLOWING) last_date
FROM     sale.customer A, sale.orders B
WHERE    A.customer_id = B.customer_id
```



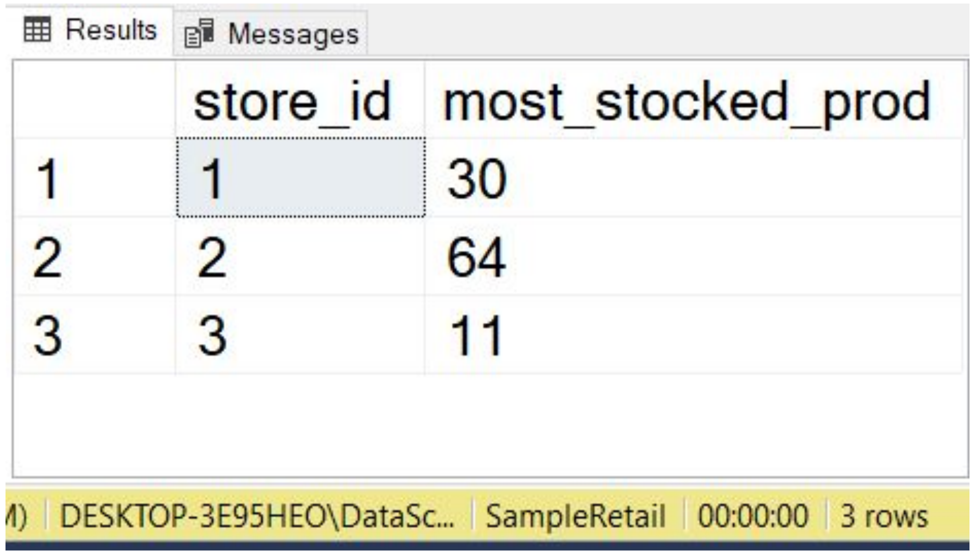
	customer_id	first_name	order_date	last_date
1	136	Ernest	2020-12-28	2018-01-01
2	135	Pasquale	2020-11-28	2018-01-01
3	1	Diane	2020-11-18	2018-01-01
4	3	Teddy	2020-10-21	2018-01-01
5	6	Cyril	2020-09-06	2018-01-01
6	15	Siobhan	2020-08-25	2018-01-01
7	10	Melissa	2020-08-23	2018-01-01
8	53	Trinidad	2020-07-12	2018-01-01
9	33	Yuki	2020-07-11	2018-01-01
10	119	Armando	2020-07-10	2018-01-01
11	123	Jerri	2020-07-01	2018-01-01
12	7	William	2020-06-17	2018-01-01
13	55	Carma	2020-04-30	2018-01-01
14	74	Nathalie	2020-04-30	2018-01-01
15	90	Daniel	2020-04-29	2018-01-01

```
Query executed... | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows
```

Query Time

Write a query that returns most stocked product in each store. (Use Last_Value)

Expected Output:



Results | Messages

	store_id	most_stocked_prod
1	1	30
2	2	64
3	3	11

M) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 3 rows

Lag() & Lead() Functions

LAG()

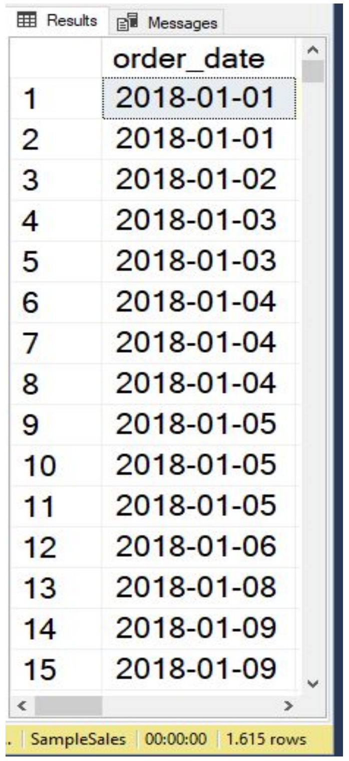
Returns the value in **<u>previous</u>** rows for each row of sorted column values.

LEAD()

Returns the value in **<u>next</u>** rows for each row of sorted column values.

Lag() Function

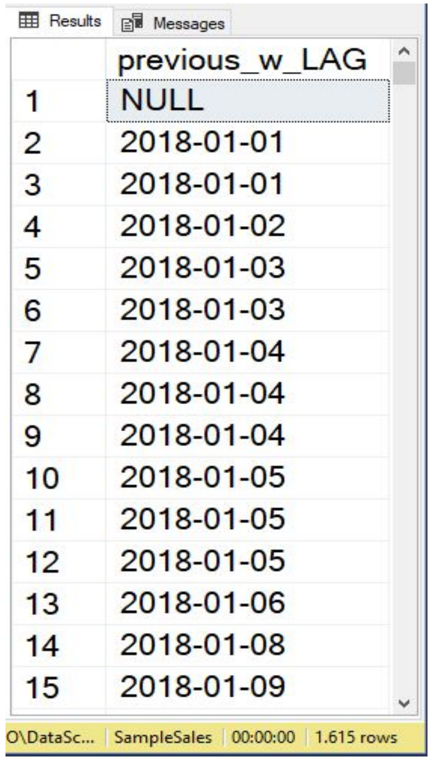
```
SELECT   order_date,
         LAG(order_date) OVER (ORDER BY order_date) previous_w_LAG
FROM     sale.orders
```



Results | Messages

	order_date
1	2018-01-01
2	2018-01-01
3	2018-01-02
4	2018-01-03
5	2018-01-03
6	2018-01-04
7	2018-01-04
8	2018-01-04
9	2018-01-05
10	2018-01-05
11	2018-01-05
12	2018-01-06
13	2018-01-08
14	2018-01-09
15	2018-01-09

. | SampleSales | 00:00:00 | 1.615 rows



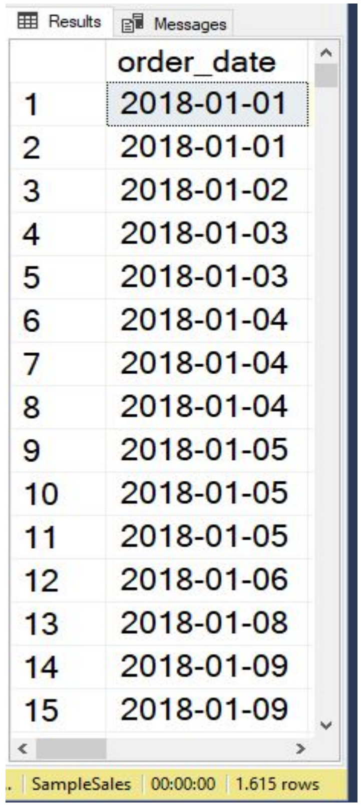
Results | Messages

	previous_w_LAG
1	NULL
2	2018-01-01
3	2018-01-01
4	2018-01-02
5	2018-01-03
6	2018-01-03
7	2018-01-04
8	2018-01-04
9	2018-01-04
10	2018-01-05
11	2018-01-05
12	2018-01-05
13	2018-01-06
14	2018-01-08
15	2018-01-09

O\DataSc... | SampleSales | 00:00:00 | 1.615 rows

Lead() Function

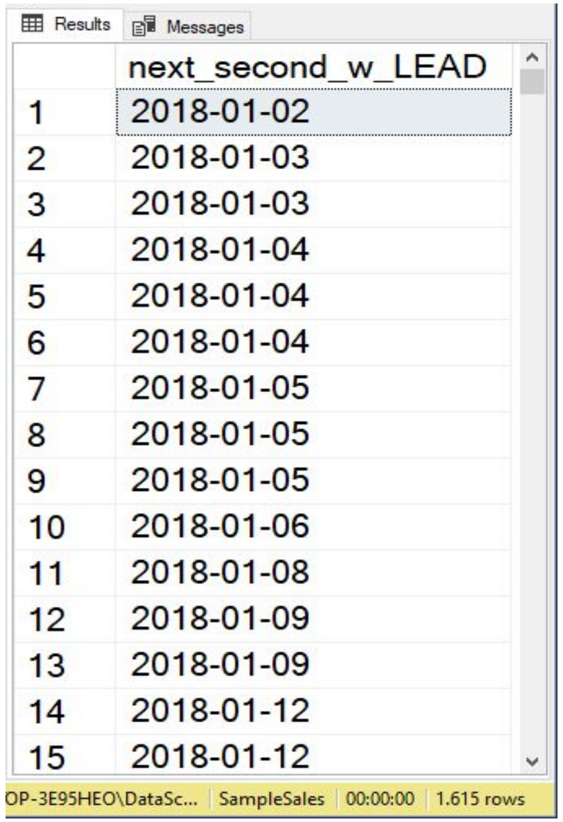
```
SELECT  order_date,
        LEAD(order_date, 2) OVER (ORDER BY order_date) next_second_w_LEAD
FROM    sale.orders
```



Results | Messages

	order_date
1	2018-01-01
2	2018-01-01
3	2018-01-02
4	2018-01-03
5	2018-01-03
6	2018-01-04
7	2018-01-04
8	2018-01-04
9	2018-01-05
10	2018-01-05
11	2018-01-05
12	2018-01-06
13	2018-01-08
14	2018-01-09
15	2018-01-09

. | SampleSales | 00:00:00 | 1.615 rows



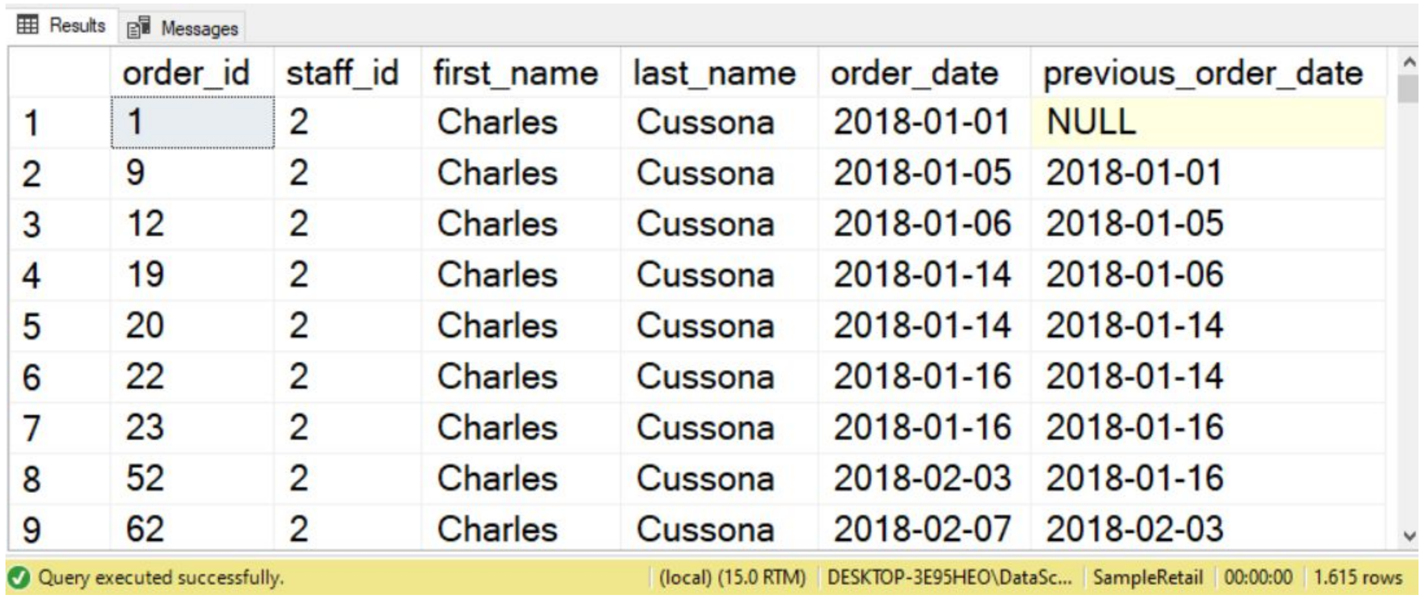
Results | Messages

	next_second_w_LEAD
1	2018-01-02
2	2018-01-03
3	2018-01-03
4	2018-01-04
5	2018-01-04
6	2018-01-04
7	2018-01-05
8	2018-01-05
9	2018-01-05
10	2018-01-06
11	2018-01-08
12	2018-01-09
13	2018-01-09
14	2018-01-12
15	2018-01-12

OP-3E95HEO\DataSc... | SampleSales | 00:00:00 | 1.615 rows

Query Time

1. Write a query that returns the order date of the one previous sale of each staff (use the LAG function)



Results | Messages

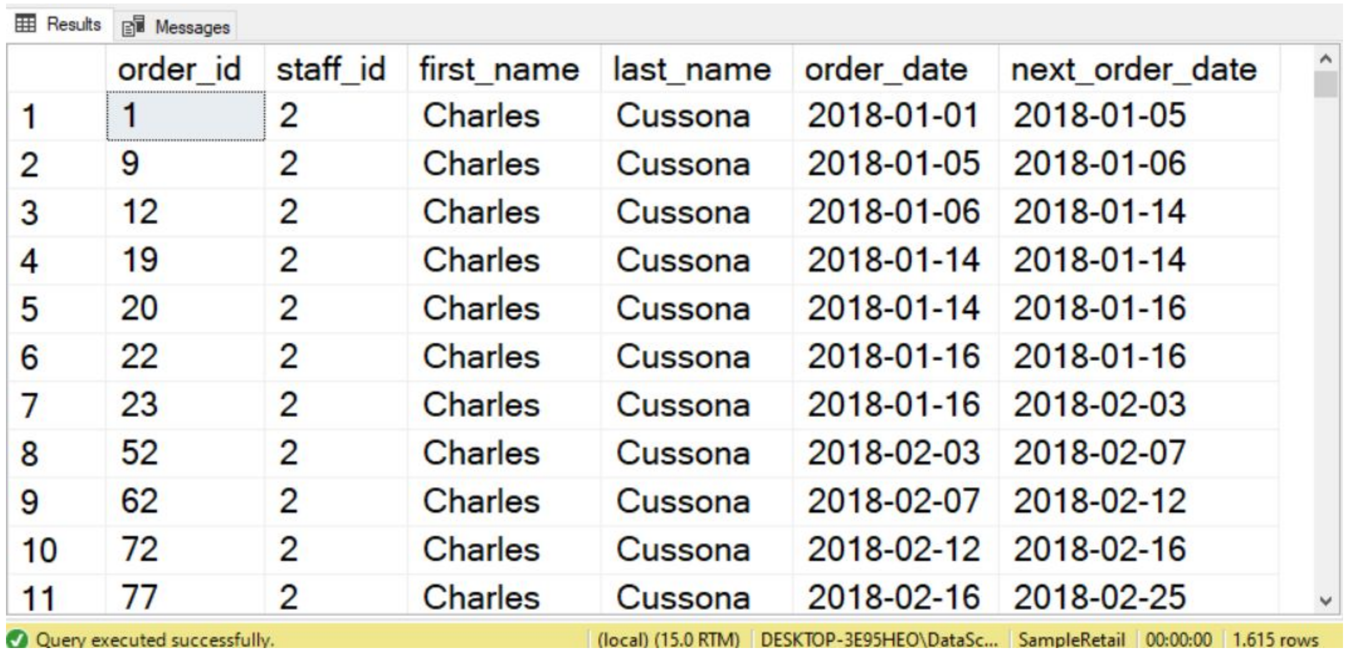
	order_id	staff_id	first_name	last_name	order_date	previous_order_date
1	1	2	Charles	Cussona	2018-01-01	NULL
2	9	2	Charles	Cussona	2018-01-05	2018-01-01
3	12	2	Charles	Cussona	2018-01-06	2018-01-05
4	19	2	Charles	Cussona	2018-01-14	2018-01-06
5	20	2	Charles	Cussona	2018-01-14	2018-01-14
6	22	2	Charles	Cussona	2018-01-16	2018-01-14
7	23	2	Charles	Cussona	2018-01-16	2018-01-16
8	52	2	Charles	Cussona	2018-02-03	2018-01-16
9	62	2	Charles	Cussona	2018-02-07	2018-02-03

Query executed successfully. | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows

Query Time For You

2. Write a query that returns the order date of the one next sale of each staff (use the LEAD function)

Expected Output:



Results | Messages

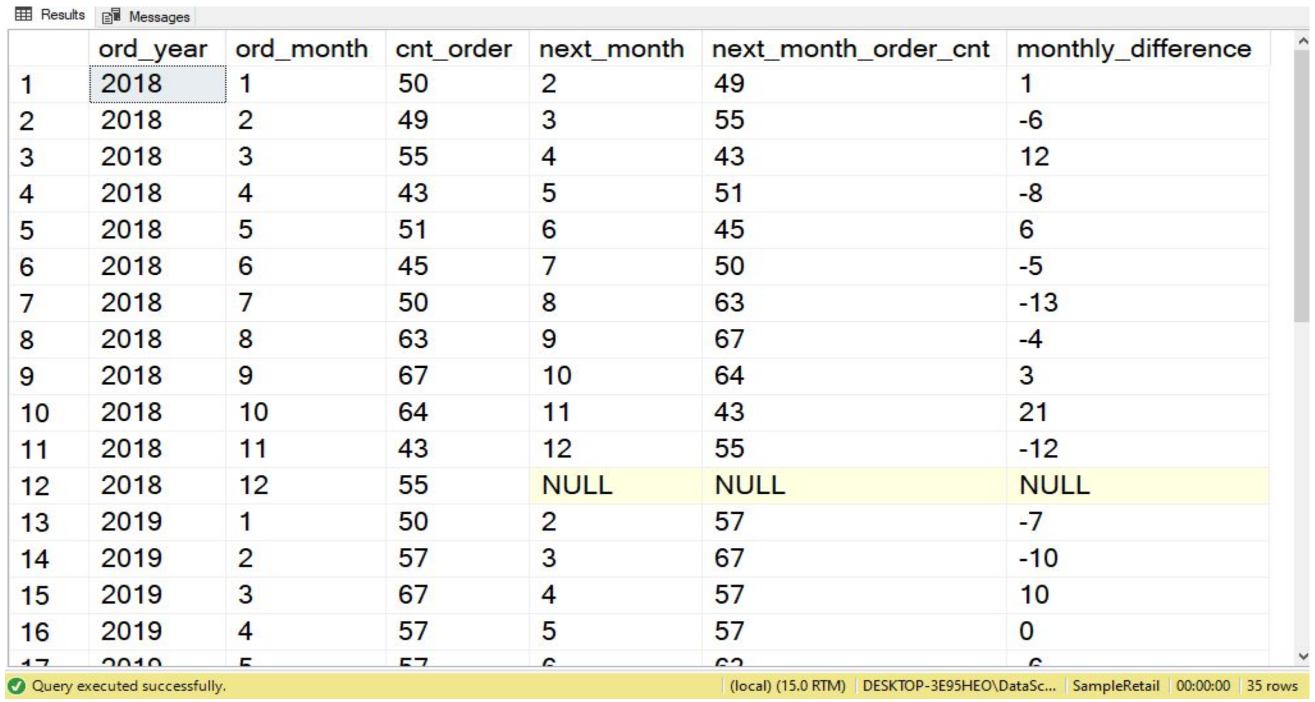
	order_id	staff_id	first_name	last_name	order_date	next_order_date
1	1	2	Charles	Cussona	2018-01-01	2018-01-05
2	9	2	Charles	Cussona	2018-01-05	2018-01-06
3	12	2	Charles	Cussona	2018-01-06	2018-01-14
4	19	2	Charles	Cussona	2018-01-14	2018-01-14
5	20	2	Charles	Cussona	2018-01-14	2018-01-16
6	22	2	Charles	Cussona	2018-01-16	2018-01-16
7	23	2	Charles	Cussona	2018-01-16	2018-02-03
8	52	2	Charles	Cussona	2018-02-03	2018-02-07
9	62	2	Charles	Cussona	2018-02-07	2018-02-12
10	72	2	Charles	Cussona	2018-02-12	2018-02-16
11	77	2	Charles	Cussona	2018-02-16	2018-02-25

Query executed successfully. | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 1.615 rows

Query Time For You

Write a query that returns the difference in the order count between the current month and the next month by year.

Expected Output:



Results | Messages

	ord_year	ord_month	cnt_order	next_month	next_month_order_cnt	monthly_difference
1	2018	1	50	2	49	1
2	2018	2	49	3	55	-6
3	2018	3	55	4	43	12
4	2018	4	43	5	51	-8
5	2018	5	51	6	45	6
6	2018	6	45	7	50	-5
7	2018	7	50	8	63	-13
8	2018	8	63	9	67	-4
9	2018	9	67	10	64	3
10	2018	10	64	11	43	21
11	2018	11	43	12	55	-12
12	2018	12	55	NULL	NULL	NULL
13	2019	1	50	2	57	-7
14	2019	2	57	3	67	-10
15	2019	3	67	4	57	10
16	2019	4	57	5	57	0
17	2019	5	57	6	63	-6

Query executed successfully. | (local) (15.0 RTM) | DESKTOP-3E95HEO\DataSc... | SampleRetail | 00:00:00 | 35 rows

THANKS!

Any questions?

Powered by Clarusway.de
Way to reinvent yourself