WINDOWING/ANALYTIC FUNCTIONS

WINDOWING/ANALYTIC FUNCTIONS

THIS IS A SUITE OF FUNCTIONS
THAT ARE SYNTACTIC SUGAR

WINDOWING/ANALYTIC FUNCTIONS

THEY MAKE CERTAIN COMPLEX OPERATIONS REALLY, REALLY SIMPLE TO EXPRESS

WINDOWING/ANALYTIC FUNCTIONS

THESE OPERATIONS WOULD OTHERWISE REQUIRE SEVERAL STEPS WITH MULTIPLE INTERMEDIATE TABLES

WINDOWING FUNCTIONS

A WINDOWING FUNCTION PERFORMS A CALCULATION ACROSS A SET OF TABLE ROWS THAT ARE SOMEHOW RELATED TO THE CURRENT ROW

WINDOWING FUNCTIONS

A WINDOWING FUNCTION PERFORMS A CALCULATION ACROSS A SET OF TABLE ROWS THAT ARE SOMEHOW RELATED TO THE CURRENT ROW

THIS SET OF TABLE ROWS IS CALLED A WINDOW

WINDOWING FUNCTIONS

THIS IS SIMILAR TO AGGREGATE FUNCTIONS THAT ARE USED WITH (GROUP BY)

BUT UNLIKE AGGREGATE FUNCTIONS, THE USE OF WINDOWING FUNCTIONS DOES NOT CAUSE ROWS TO BE GROUPED INTO A SINGLE OUTPUT ROW

WINDOWING FUNCTIONS

AN EXAMPLE

CONSIDER THE FOLLOWING SALES TABLE

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89
13	1	3	2016-01-18	7455.67
14	1	1	2016-01-18	8236.33
15	2	4	2016-01-18	8988.64
16	2	1	2016-01-18	9456.01

AN EXAMPLE

LET US CALCULATE A RUNNING TOTAL OF REVENUE

A RUNNING TOTAL IS THE SUMMATION OF A SEQUENCE OF NUMBERS WHICH IS UPDATED EACH TIME A NEW NUMBER IS ADDED TO THE SEQUENCE

WINDOWING FUNCTIONS

RUNNING TOTAL OF REVENUE

THE QUERY RESULT SHOULD HAVE A COLUMN WITH THE RUNNING TOTAL

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8
13	1	3	2016-01-18	7455.67	63795.47
14	1	1	2016-01-18	8236.33	72031.8
15	2	4	2016-01-18	8988.64	81020.44
16	2	1	2016-01-18	9456.01	90476.45

WINDOWING FUNCTIONS

RUNNING TOTAL OF REVENUE

FIRST MAKE SURE THE TABLE IS SORTED BY ORDER ID

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8
13	1	3	2016-01-18	7455.67	63795.47
14	1	1	2016-01-18	8236.33	72031.8
15	2	4	2016-01-18	8988.64	81020.44
16	2	1	2016-01-18	9456.01	90476.45

1299.45 IS THE 1ST ELEMENT OF THE REVENUE COLUMN

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	**1299.45**	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8

3641.78 IS THE SUM OF 1ST TWO ELEMENTS OF THE REVENUE COLUMN

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	**1299.45**	1299.45
2	1	1	2016-01-17	**2342.33**	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8

8185.76 IS THE SUM OF 1ST THREE ELEMENTS OF THE REVENUE COLUMN

AND SO ON...

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	**1299.45**	1299.45
2	1	1	2016-01-17	**2342.33**	3641.78
3	1	2	2016-01-17	**4543.98**	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8

FOR EACH ROW,

CALCULATE THE SUM
OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

OrderID	StoreID	ProductID	OrderDate	Revenue	Running Total
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	44944.82
10	1	2	2016-01-18	2433.76	47378.58
11	2	3	2016-01-18	3644.33	51022.91
12	1	4	2016-01-18	5316.89	56339.8
13	1	3	2016-01-18	7455.67	63795.47

FOR EACH ROW,

CALCULATE THE SUM
OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

THIS IS EXACTLY HOW WINDOWING FUNCTIONS ARE USED

FOR EACH ROW,

CALCULATE THE SUM
OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

A RESULT IS CALCULATED FOR
EACH ROW

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

THE RESULT IS COMPUTED USING A WINDOW OF ROWS RELATIVE TO THE CURRENT ROW

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

THE RESULT IS COMPUTED BY APPLYING THE SPECIFIED FUNCTION OVER THE WINDOW

FOR EACH ROW,

CALCULATE THE SUM OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

FOR EACH ROW,

CALCULATE THE SUM
OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

THIS IS WHERE THE WINDOWING FUNCTION IS SPECIFIED

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

WE START WITH AN AGGREGATE FUNCTION

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

THERE ARE MANY DIFFERENT AGGREGATE FUNCTIONS YOU CAN USE

THE USUAL FUNCTIONS LIKE AVG, MAX, MIN, COUNT ETC

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

THERE ARE ALSO SPECIAL FUNCTIONS
LIKE FIRST_VALUE, LAG, LEAD, RANK ETC

WE'LL GO OVER THEM IN DETAIL LATER...

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID ROWS BETWEEN
UNBOUNDED PRECEDING and CURRENT ROW) as Running_Total;
```

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID
ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as
Running_Total;
```

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS **FROM THE TOP ROW** TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID
ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as
Running_Total;
```

FOR EACH ROW,

CALCULATE THE SUM

OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID
ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as
Running_Total;
```

FOR EACH ROW,

THIS IS IMPORTANT

CALCULATE THE SUM
OVER ALL ROWS FROM THE TOP ROW TILL THIS ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID
ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as
Running_Total;
```

WE NEED TO MAKE SURE THAT THE ROWS ARE ARRANGED IN ASCENDING ORDER OF TRANSACTION IDS

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID
ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT ROW) as
Running_Total;
```

THIS IS THE DEFAULT ROWS SPECIFICATION

WHEN YOU USE ORDER BY IN THE WINDOW SPECIFICATION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(order by OrderID) as Running_Total;
```

THIS WORKS EXACTLY THE SAME AS THE PREVIOUS QUERY

LET'S ADD A LITTLE TWIST TO OUR RUNNING TOTAL CALCULATION

AT THE BEGINNING OF A NEW DAY, THE RUNNING TOTAL ON REVENUE COLUMN SHOULD RESET TO ZERO

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89
13	1	3	2016-01-18	7455.67

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58
15	2	4	2016-01-18	8988.64	37697.20
16	2	1	2016-01-18	9456.01	47153.21

RUNNING TOTAL FOR '2016-01-17'

AT THE BEGINNING OF A NEW DAY, THE RUNNING TOTAL ON REVENUE COLUMN SHOULD RESET TO ZERO

RUNNING TOTAL FOR '2016-01-18'

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
					1.78
					5.76
					7.81
					0.82
					5.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58
10	1	2	2016-01-18	2433.76	4055.34
11	2	3	2016-01-18	3644.33	7699.67
12	1	4	2016-01-18	5316.89	13016.56
13	1	3	2016-01-18	7455.67	20472.23
14	1	1	2016-01-18	8236.33	28708.56
15	2	4	2016-01-18	8988.64	37697.20
16	2	1	2016-01-18	9456.01	47153.21

AT THE BEGINNING OF A NEW DAY, THE RUNNING TOTAL ON REVENUE COLUMN SHOULD RESET TO ZERO

THE QUERY THAT WILL ACCOMPLISH WHAT WE WANT IS

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order by OrderID) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58
10	1	2	2016-01-18	2433.76	4055.34
11	2	3	2016-01-18	3644.33	7699.67

THIS IS OUR WINDOW FUNCTION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order by OrderID)
 as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58

THIS IS THE AGGREGATE FUNCTION THAT SPECIFIES THE COMPUTATION TO BE PERFORMED ON THE WINDOW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate
order by OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58

'OVER' DESIGNATES IT AS A WINDOW FUNCTION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order by
 OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58

PARTITION BY NARROWS THE WINDOW TO A SUBSET OF THE ENTIRE DATASET

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order
by OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58

THE SUBSET IS CREATED BASED ON ORDERDATE

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order
by OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58

THESE SUBSETS ARE CALLED PARTITIONS

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order
by OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24

THE WINDOW FOR THE ROW WILL BE CONSTRUCTED WITHIN THE RELEVANT PARTITION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over(partition by OrderDate order
by OrderID) as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
3	1	2	2016-01-17	4543.98	8185.76
4	2	4	2016-01-17	5102.05	13287.81
5	1	4	2016-01-17	5673.01	18960.82
6	1	3	2016-01-17	6345.10	25305.92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24

THE WINDOW FOR THE ROW WILL BE CONSTRUCTED WITHIN THE RELEVANT PARTITION

OrderID	StoreID	ProductID	OrderDate	Revenue	RUNNING_TOTAL
1	2	2	2016-01-17	1299.45	1299.45
2	1	1	2016-01-17	2342.33	3641.78
					76
					81
					82
					92
7	2	1	2016-01-17	8902.65	34208.57
8	2	3	2016-01-17	9114.67	43323.24
9	2	2	2016-01-18	1621.58	1621.58
10	1	2	2016-01-18	2433.76	4055.34
					6
					3
					6
15	2	4	2016-01-18	8988.64	37697.20
16	2	1	2016-01-18	9456.01	47153.21

PARTITION FOR ALL ROWS WITH ORDERDATE = '2016-01-17'

PARTITION FOR ALL ROWS WITH ORDERDATE = '2016-01-18'

THE WINDOW FOR THE ROW WILL BE CONSTRUCTED WITHIN THE RELEVANT PARTITION

PARTITIONS ARE CREATED FOR EACH DISTINCT VALUE OF ORDERDATE

EVERY ROW BELONGS TO ONE SPECIFIC PARTITION

LET'S CONSIDER 1 ROW AND CALCULATE IT'S RUNNING TOTAL

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)over(
partition by
OrderDate order
by OrderID) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89
13	1	3	2016-01-18	7455.67
14	1	1	2016-01-18	8236.33
15	2	4	2016-01-18	8988.64
16	2	1	2016-01-18	9456.01

COLLECT ALL THE ROWS WITH THE SAME ORDER DATE

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)over(
partition by
OrderDate
order
by OrderID) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89
13	1	3	2016-01-18	7455.67
14	1	1	2016-01-18	8236.33
15	2	4	2016-01-18	8988.64
16	2	1	2016-01-18	9456.01

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)over(
partition by
OrderDate order
by OrderID)
as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

WE NEED TO MAKE SURE THAT THE ROWS ARE ARRANGED IN ASCENDING ORDER OF TRANSACTION IDS

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)over(
partition by
OrderDate order
by OrderID)
as Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

WHEN AN ORDER BY CLAUSE IS PRESENT THE DEFAULT ROWS SPECIFICATION IS

```
ROWS BETWEEN
UNBOUNDED PRECEDING
and CURRENT ROW
```

```
from Sales_Dat
select
OrderID,storeI
ductID,OrderDa
venue,
sum(Revenue)
over( partition by
OrderDate order
by OrderID) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

8185.76

THE SUM IS COMPUTED OVER THIS WINDOW

```
from Sales_Dat
select
OrderID,storeI
ductID,OrderDa
venue,
sum(Revenue)over( p
artition by
OrderDate order
by OrderID) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

WHAT IF WE DROPPED THE ORDER BY CLAUSE?

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)
over(partiti
on by
OrderDate) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

THE DEFAULT ROWS SPECIFICATION CHANGES!

ROWS BETWEEN UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING

```
from Sales_Data
select
OrderID,storeID,P
roductID,OrderDat
e,Revenue,
sum(Revenue)
over(partiti
on by
OrderDate) as
Running_Total;
```

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67

ROWS BETWEEN UNBOUNDED PRECEDING and UNBOUNDED FOLLOWING

THE SUM IS COMPUTED OVER THE ENTIRE PARTITION

SUMMARISING

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over( partition by OrderDate order
by OrderID ROWS BETWEEN UNBOUNDED PRECEDING
and CURRENT ROW) as Running_Total;
```

WE'VE SEEN A COUPLE OF DIFFERENT EXAMPLES OF WINDOWING FUNCTIONS

SUMMARISING

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
sum(Revenue)over( partition by OrderDate order
by OrderID ROWS BETWEEN UNBOUNDED PRECEDING and CURRENT
ROW) as Running_Total;
```

LET'S LOOK AT THE GENERAL SYNTAX

SUMMARISING

```
sum(Revenue)
```

AGGREGATE FUNCTION

```
over
```

DESIGNATES THIS AS A WINDOWING FUNCTION

```
(partition by OrderDate
```

DETERMINES PARTITIONS/SUBSETS

```
order by OrderID
```

DECIDE ORDERING INSIDE THE PARTITION

```
ROWS BETWEEN UNBOUNDED
PRECEDING and
CURRENT ROW)
```

THE SET OF ROWS RELATIVE TO THE CURRENT ROW WITHIN THE PARTITION

LET'S CALCULATE A MOVING AVERAGE OF THE REVENUE

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
avg(Revenue)over(order by OrderID
ROWS BETWEEN 3 PRECEDING and CURRENT ROW) as
Moving_Revenue_Average;
```

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
avg(Revenue)over(order by OrderID
ROWS BETWEEN 3 PRECEDING and CURRENT ROW) as
Moving_Revenue_Average;
```

WE USE AVG AS THE AGGREGATING FUNCTION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
avg(Revenue)over(order by OrderID
ROWS BETWEEN 3 PRECEDING and CURRENT ROW) as
Moving_Revenue_Average;
```

MAKE SURE THAT THE ROWS ARE IN ASCENDING ORDER

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
avg(Revenue)over(order by OrderID
ROWS BETWEEN 3 PRECEDING and CURRENT ROW) as
Moving_Revenue_Average;
```

THE AVERAGE IS COMPUTED OVER THE (CURRENT ROW-3 : CURRENT ROW)

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	2	3	1/17/2016	9114.67	21.0
7	2	1	1/17/2016	8902.65	20.5
6	1	3	1/17/2016	6345.1	14.6
5	1	4	1/17/2016	5673.01	13.1
4	2	4	1/17/2016	5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4

PARTITION FOR '2016-01-18'

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
16	2	1	1/18/2016	9456.01	20.1
15	2	4	1/18/2016	8988.64	19.1
14	1	1	1/18/2016	8236.33	17.5
13	1	3	1/18/2016	7455.67	15.8
12	1	4	1/18/2016	5316.89	11.3
11	2	3	1/18/2016	3644.33	7.7
10	1	2	1/18/2016	2433.76	5.2
9	2	2	1/18/2016	1621.58	3.4

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	2	3	1/17/2016	9114.67	21.0
7	2	1	1/17/2016	8902.65	20.5
					14.6
					13.1
					11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4

TOTAL REVENUE FOR 17TH JAN IS 43,323

PARTITION FOR '2016-01-18'

16	2	1	1/18/2016	9456.01	20.1
					19.1
					17.5
					15.8
12	1	4	1/18/2016	5316.89	11.3
11	2	3	1/18/2016	3644.33	7.7
10	1	2	1/18/2016	2433.76	5.2
9	2	2	1/18/2016	1621.58	3.4

TOTAL REVENUE FOR 18TH JAN IS 47,153

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	**2**	**2**	**1/17/2016**	**1299.45**	**3.0**
8	2	3	1/17/2016	9114.67	21.0
7	2	1	1/17/2016	8902.65	20.5
6	1	3	1/17/2016	6345.1	14.6
5	1	4	1/17/2016	5673.01	13.1
4	2	4	1/17/2016	5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4

TOTAL REVENUE FOR 17TH JAN IS 43,323

FOR THE HIGHLIGHTED ROW, REVENUE/TOTAL REVENUE FOR THE DAY IS 1299/43,323 IS 0.03 WHICH IS MULTIPLIED BY 100

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	**2**	**3**	**1/17/2016**	**9114.67**	**21.0**
7	2	1	1/17/2016	8902.65	20.5
6	1	3	1/17/2016	6345.1	14.6
5	1	4	1/17/2016	5673.01	13.1
4	2	4	1/17/2016	5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4

TOTAL REVENUE FOR 17TH JAN IS 43,323

FOR THE HIGHLIGHTED ROW, REVENUE/TOTAL REVENUE FOR THE DAY IS 9114.67/43,323 IS 0.21 WHICH IS MULTIPLIED BY 100

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	2	3	1/17/2016	9114.67	21.0
7	2	1	1/17/2016	8902.65	20.5
[illegible]	[illegible]	[illegible]	[illegible]	6345.1	14.6
[illegible]	[illegible]	[illegible]	[illegible]	5673.01	13.1
[illegible]	[illegible]	[illegible]	[illegible]	5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4
16	2	1	1/18/2016	9456.01	20.1
15	2	4	1/18/2016	8988.64	19.1
14	1	1	1/18/2016	8236.33	17.5
[illegible]	[illegible]	[illegible]	[illegible]	7455.67	15.8
[illegible]	[illegible]	[illegible]	[illegible]	5316.89	11.3
11	2	3	1/18/2016	3644.33	7.7
10	1	2	1/18/2016	2433.76	5.2
9	2	2	1/18/2016	1621.58	3.4

THEIR SUM IS 100.

THEIR SUM IS 100.

LET US WRITE A QUERY THAT SHOWS THE REVENUE OF EACH ORDERID AS A PERCENTAGE OF TOTAL REVENUE FOR A GIVEN DATE

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage Revenue Day Level
1	2	2	1/17/2016	1299.45	3.0
8	**2**	**3**	**1/17/2016**	**9114.67**	**21.0**
7	2	1	1/17/2016	8902.65	20.5
6	1	3	1/17/2016	6345.1	14.6
5	1	4	1/17/2016	5673.01	13.1
4	2	4	1/17/2016	5102.05	11.8
3	1	2	1/17/2016	4543.98	10.5
2	1	1	1/17/2016	2342.33	5.4

TO CALCULATE THE FIELD FOR EVERY ROW, WE NEED

$$\frac{\text{REVENUE} \times 100}{\text{TOTAL REVENUE FOR THE DAY}}$$

TO CALCULATE THE FIELD FOR EVERY ROW, WE NEED

$$\frac{\text{REVENUE} \times 100}{\text{TOTAL REVENUE FOR THE DAY}}$$

$$= \frac{\text{REVENUE} \times 100}{\text{SUM(REVENUE) OVER (PARTITION BY ORDERDATE)}}$$

TO CALCULATE THE FIELD FOR EVERY ROW, WE NEED

$$\frac{\text{REVENUE X 100}}{\text{SUM(REVENUE) OVER (PARTITION BY ORDERDATE)}}$$

RECALL: WHEN YOU ONLY USE PARTITION BY CLAUSE WITHOUT AN ORDER BY CLAUSE

THE SUM IS COMPUTED OVER THE ENTIRE PARTITION

TO CALCULATE THE FIELD FOR EVERY ROW, WE NEED

= REVENUE X 100

SUM(REVENUE) OVER (PARTITION BY ORDERDATE)

THE QUERY THAT WILL ACCOMPLISH WHAT WE WANT IS

THE QUERY THAT WILL ACCOMPLISH WHAT WE WANT IS

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,

(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

THIS IS OUR WINDOWING FUNCTION WHICH CALCULATES THE SUM OF REVENUE FOR A DAY

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,

(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

THIS WINDOWING FUNCTION IS CALCULATED FOR THE PARTITION TO WHICH ROW THE BELONGS

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,

(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	Percentage_Revenue_Day_Level
1	2	2	1/17/2016	1299.45	3.0
8	**2**	**3**	**1/17/2016**	**9114.67**	**21.0**
7	2	1	1/17/2016	8902.65	20.5
6	1	3	1/17/2016	6345.1	14.6
5	1	4	1/17/2016	5673.01	13.1
4	2	4	1/17/2016	5102.05	11.8
3	1	2	1/17/2016	543.93	10.5
2	1	1	1/17/2016	2342.33	5.4

TOTAL REVENUE FOR 17TH JAN IS 43,323

THE VALUE OF THIS FUNCTION WILL BE CONSTANT FOR ALL THE ROWS THAT BELONG TO SAME PARTITION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,

(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

REVENUE IS YOUR VANILLA SELECT QUERY (JUST A FIELD HERE) AND WILL CHANGE WITH VERY ROW

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,

(Revenue)*100/
(SUM(REVENUE)over(partition by OrderDate)) as
Percentage_Revenue_Day_Level;
```

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()

ROW_NUMBER()

IT DISPLAYS THE SERIAL NUMBER OF THE ROW IN A PARTITION ACCORDING TO THE **ORDER BY** PART OF THE WINDOWING FUNCTION

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
row_number()over(partition by OrderDate order by OrderID)
as order_number_in_a_day;
```

ROW_NUMBER()

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
row_number()over(partition by OrderDate order by
OrderID) as order_number_in_a_day;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	Order_number_in_a_day
1	2	2	2016-01-17	1299.45	1
2	1	1	2016-01-17	2342.33	2
3	1	2	2016-01-17	4543.98	3
4	2	4	2016-01-17	5102.05	4
5	1	4	2016-01-17	5673.01	5
6	1	3	2016-01-17	6345.10	6
7	2	1	2016-01-17	8902.65	7
8	2	3	2016-01-17	9114.67	8
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	2
11	2	3	2016-01-18	3644.33	3
12	1	4	2016-01-18	5316.89	4
13	1	3	2016-01-18	7455.67	5
14	1	1	2016-01-18	8236.33	6
15	2	4	2016-01-18	8988.64	7
16	2	1	2016-01-18	9456.01	8

PARTITION FOR '2016-01-17'

PARTITION FOR '2016-01-18'

ROW_NUMBER()

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
row_number()over(partition by OrderDate order by
OrderID) as order_number_in_a_day;
```

OrderID	StoreID	ProductID	OrderDate	Revenue	Order_number_in_a_day
1	2	2	2016-01-17	1299.45	1
2	1	1	2016-01-17	2342.33	2
3	1	2	2016-01-17	4543.98	3
4	2	4	2016-01-17	5102.05	4
5	1	4	2016-01-17	5673.21	5
6	1	3	2016-01-17	6345.10	6
7	2	1	2016-01-17	8902.65	7
8	2	3	2016-01-17	9114.67	8
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	2
11	2	3	2016-01-18	3644.33	3
12	1	4	2016-01-18	6314.89	4
13	1	3	2016-01-18	7455.67	5
14	1	1	2016-01-18	8236.33	6
15	2	4	2016-01-18	8988.64	7
16	2	1	2016-01-18	9456.01	8

PARTITION FOR 2016-01-17

PARTITION FOR 2016-01-18

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()

RANK()

IT IS SIMILAR TO ROW_NUMBER IN OUTPUT WITH 1 DIFFERENCE

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
rank()over(partition by OrderDate order by OrderID) as
order_number_in_a_day;
```

THE SYNTAX IS SIMILAR

RANK()

IT IS SIMILAR TO ROW_NUMBER IN OUTPUT WITH 1 DIFFERENCE

IF THERE ARE ROWS HAVING SAME ORDERID

RANK() WILL GIVE THESE ROWS SAME RANK

ROW_NUMBER() WILL GIVE THESE ROWS DIFFERENT NUMBERS

RANK()

CONSIDER A TABLE WHERE THE FIRST TWO ROWS HAVE SAME ORDERID

OrderID	StoreID	ProductID	OrderDate	Revenue
1	2	2	2016-01-17	1299.45
1	1	1	2016-01-17	2342.33
2	1	1	2016-01-17	2342.33
3	1	2	2016-01-17	4543.98
4	2	4	2016-01-17	5102.05
5	1	4	2016-01-17	5673.01
6	1	3	2016-01-17	6345.10
7	2	1	2016-01-17	8902.65
8	2	3	2016-01-17	9114.67
9	2	2	2016-01-18	1621.58
10	1	2	2016-01-18	2433.76
11	2	3	2016-01-18	3644.33
12	1	4	2016-01-18	5316.89
13	1	3	2016-01-18	7455.67
14	1	1	2016-01-18	8236.33
15	2	4	2016-01-18	8988.64
16	2	1	2016-01-18	9456.01

RANK()

OUTPUT FROM RANK()

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Order_number_in_a_day
1	2	2	2016-01-17	1299.45	1
1	1	1	2016-01-17	2342.33	1
2	1	1	2016-01-17	2342.33	3
3	1	2	2016-01-17	4543.98	4
4	2	4	2016-01-17	5102.05	5
5	1	4	2016-01-17	5673.01	6
6	1	3	2016-01-17	6345.10	7
7	2	1	2016-01-17	8902.65	8
8	2	3	2016-01-17	9114.67	9
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	2
11	2	3	2016-01-18	3644.33	3
12	1	4	2016-01-18	5316.89	4
13	1	3	2016-01-18	7455.67	5
14	1	1	2016-01-18	8236.33	6
15	2	4	2016-01-18	8988.64	7
16	2	1	2016-01-18	9456.01	8

PARTITION FOR '2016-01-18'

RANK()

ROWS WITH SAME ORDERID WERE GIVEN SAME RANK

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Order_number_in_a_day
1	2	2	2016-01-17	1299.45	1
1	1	1	2016-01-17	2342.33	1
2	1	1	2016-01-17	2342.33	3
3	1	2	2016-01-17	4543.98	4
4	2	4	2016-01-17	5102.05	5
5	1	4	2016-01-17	5673.01	6
6	1	3	2016-01-17	6345.10	7
7	2	1	2016-01-17	8902.65	8
8	2	3	2016-01-17	9114.67	9
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	2
11	2	3	2016-01-18	3644.33	3
12	1	4	2016-01-18	5316.89	4
13	1	3	2016-01-18	7455.67	5
14	1	1	2016-01-18	8236.33	6
15	2	4	2016-01-18	8988.64	7
16	2	1	2016-01-18	9456.01	8

PARTITION FOR '2016-01-18'

ROW_NUMBER()

ROWS WITH SAME ORDERID WERE GIVEN DIFFERENT NUMBERS

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Order_number_in_a_day
1	2	2	2016-01-17	1299.45	1
1	1	1	2016-01-17	2342.33	2
2	1	1	2016-01-17	2342.33	3
3	1	2	2016-01-17	4543.98	4
4	2	4	2016-01-17	5102.05	5
5	1	4	2016-01-17	5673.01	6
6	1	3	2016-01-17	6345.10	7
7	2	1	2016-01-17	8902.65	8
8	2	3	2016-01-17	9114.67	9
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	2
11	2	3	2016-01-18	3644.33	3
12	1	4	2016-01-18	5316.89	4
13	1	3	2016-01-18	7455.67	5
14	1	1	2016-01-18	8236.33	6
15	2	4	2016-01-18	8988.64	7
16	2	1	2016-01-18	9456.01	8

PARTITION FOR '2016-01-18'

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()

NTILE()

WILL TELL YOU WHAT PERCENTILE A GIVEN ROW FALLS INTO

THE SYNTAX IS NTILE(NUMBER OF BUCKETS)

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
NTILE(5)over(partition by OrderDate order by OrderID)
as order_percentile_rank_in_a_day;
```

NTILE()

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
NTILE(5)over(partition by OrderDate order by OrderID) as
order_percentile_rank_in_a_day;
```

THIS FUNCTION DIVIDES EACH PARTITION INTO QUANTILES BASED ON (NUMBER OF BUCKETS)

AND GIVES EACH ROW A NUMBER THAT SPECIFIES IT'S QUANTILE

NTILE()

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
NTILE(5)over(partition by OrderDate order by OrderID) as
order_percentile_rank_in_a_day;
```

EACH PARTITION IS DIVIDED INTO 5 EQUAL BUCKETS AFTER SORTING THEM IN THE SPECIFIED ORDER

EACH ROW IS GIVEN A NUMBER BETWEEN 1 TO 5

NTILE()

THE SYNTAX IS NTILE(NUMBER OF BUCKETS)

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Order Percentile Rank in a Day
1	2	2	2016-01-17	1299.45	1
1	1	1	2016-01-17	2342.33	1
2	1	1	2016-01-17	2342.33	2
3	1	2	2016-01-17	4543.98	2
4	2	4	2016-01-17	5102.05	3
5	1	4	2016-01-17	5673.01	3
6	1	3	2016-01-17	6345.10	4
7	2	1	2016-01-17	8902.65	4
8	2	3	2016-01-17	9114.67	5
9	2	2	2016-01-18	1621.58	1
10	1	2	2016-01-18	2433.76	1
11	2	3	2016-01-18	3644.33	2
12	1	4	2016-01-18	5316.89	2
13	1	3	2016-01-18	7455.67	3
14	1	1	2016-01-18	8236.33	3
15	2	4	2016-01-18	8988.64	4
16	2	1	2016-01-18	9456.01	5

PARTITION FOR '2016-01-18'

NTILE()

THE SYNTAX IS NTILE(NUMBER OF BUCKETS)

PARTITION FOR '2016-01-17'

OrderID	StoreID	ProductID	OrderDate	Revenue	Order Percentile Rank in a Day
1	2	2	2016-01-17	1299.45	1
1	1	1	2016-01-17	2342.33	1
2	1	1	2016-01-17	2342.33	2
3	1	2	2016-01-17	4543.98	2
4	2	4	2016-01-17	5102.05	3
5	1	4	2016-01-17	5673.01	3
6	1	3	2016-01-17	6345.10	4
7	2	1	2016-01-17	8902.65	4
8	2	3	2016-01-17	9114.67	5

THERE ARE 5 BUCKETS AND EACH BUCKET HAS ON AVERAGE TWO ROWS

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()

LAG() AND LEAD() ARE VERY SIMILAR

LAG() HELPS YOU TO REFER TO FIELDS FROM PRECEDING ROWS

THE SYNTAX IS LAG(COLUMN_NAME,NUMBER OF ROWS AWAY)

LEAD() HELPS YOU TO REFER TO FIELDS FROM FOLLOWING ROWS

THE SYNTAX IS LEAD(COLUMN_NAME,NUMBER OF ROWS AWAY)

LAG()

HELPS YOU TO REFER TO FIELDS FROM PRECEDING ROWS

THE SYNTAX IS LAG(COLUMN_NAME,NUMBER OF ROWS AWAY)

THIS IS NORMALLY USED TO CALCULATE DIFFERENCES

LAG()

HELPS YOU TO REFER TO FIELDS FROM PRECEDING ROWS

THE SYNTAX IS LAG(COLUMN_NAME,NUMBER OF ROWS AWAY)

```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
Revenue-LAG(Revenue,1)over(partition by OrderDate order
by OrderID) as jump_in_revenue;
```

LAG()

HELPS YOU TO REFER TO FIELDS FROM PRECEDING ROWS

THE SYNTAX IS LAG(COLUMN_NAME,NUMBER OF ROWS

THIS IS THE REVENUE OF THE CURRENT ROW

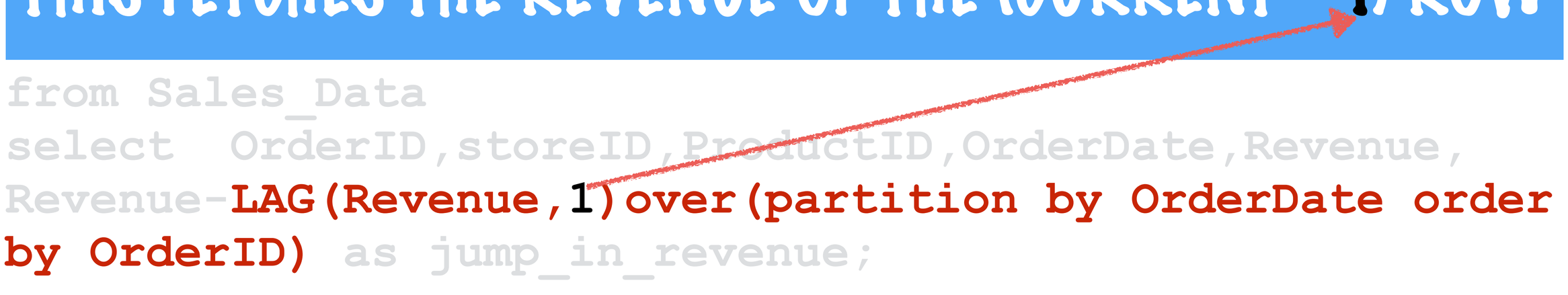
```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
Revenue-LAG(Revenue,1)over(partition by OrderDate order
by OrderID) as jump_in_revenue;
```

LAG()

HELPS YOU TO REFER FIELDS FROM PRECEDING ROWS

THE SYNTAX IS LAG(COLUMN_NAME,NUMBER OF ROWS

THIS FETCHES THE REVENUE OF THE (CURRENT - 1) ROW



```
from Sales_Data
select  OrderID,storeID,ProductID,OrderDate,Revenue,
Revenue-LAG(Revenue,1)over(partition by OrderDate order
by OrderID) as jump_in_revenue;
```

THE RESULT WILL BE NULL FOR THE FIRST ROW IN EACH PARTITION (IT HAS NO PRECEDING ROW)

OrderID	StoreID	ProductID	OrderDate	Revenue	jump_in_revenue
1	2	2	2016-01-17	1299.45	NULL
2	1	1	2016-01-17	2342.33	1042.88
3	1	2	2016-01-17	4543.98	2201.65
4	2	4	2016-01-17	5102.05	558.07
5	1	4	2016-01-17	5673.01	570.96
6	1	3	2016-01-17	6345.10	672.09
7	2	1	2016-01-17	8902.65	2557.55
8	2	3	2016-01-17	9114.67	212.02
9	2	2	2016-01-18	1621.58	NULL
10	1	2	2016-01-18	2433.76	812.18
11	2	3	2016-01-18	3644.33	1210.57
12	1	4	2016-01-18	5316.89	1672.56
13	1	3	2016-01-18	7455.67	2138.78
14	1	1	2016-01-18	8236.33	780.66
15	2	4	2016-01-18	8988.64	752.31
16	2	1	2016-01-18	9456.01	467.37

OTHERWISE THE RESULT IS THE DIFF IN REVENUE BETWEEN CURRENT AND PREVIOUS ROWS

OrderID	StoreID	ProductID	OrderDate	Revenue	jump_in_revenue
1	2	2	2016-01-17	1299.45	NULL
2	1	1	2016-01-17	2342.33	1042.88
3	1	2	2016-01-17	4543.98	2201.65
4	2	4	2016-01-17	5102.05	558.07
5	1	4	2016-01-17	5673.01	570.96
6	1	3	2016-01-17	6345.10	672.09
7	2	1	2016-01-17	8902.65	2557.55
8	2	3	2016-01-17	9114.67	212.02
9	2	2	2016-01-18	1621.58	NULL
10	1	2	2016-01-18	2433.76	812.18
11	2	3	2016-01-18	3644.33	1210.57
12	1	4	2016-01-18	5316.89	1672.56
13	1	3	2016-01-18	7455.67	2138.78
14	1	1	2016-01-18	8236.33	780.66
15	2	4	2016-01-18	8988.64	752.31
16	2	1	2016-01-18	9456.01	467.37

WE HAVE SPECIAL AGGREGATE FUNCTIONS AVAILABLE

ROW_NUMBER()

RANK()

NTILE()

LAG()

LEAD()