Where Can I Add a Window Function?

May appear in

- SELECT list
- ORDER BY clause

Cannot be found in

- FROM clause
- WHERE clause
- GROUP BY clause
- HAVING clause

New Ways to Solve Problems

$\left.\begin{array}{l}\text{Apply row numbers, rank,}\\ \text{or divide rows}\\ \text{Aggregate with no GROUP BY}\end{array}\right\}$ 2005

Moving aggregates

Analytic functions

The Parts of the OVER Clause

ORDER BY | PARTITION BY | Framing

ORDER BY

May be one or more columns or expressions -- even a subquery

Required when the order is important to the function

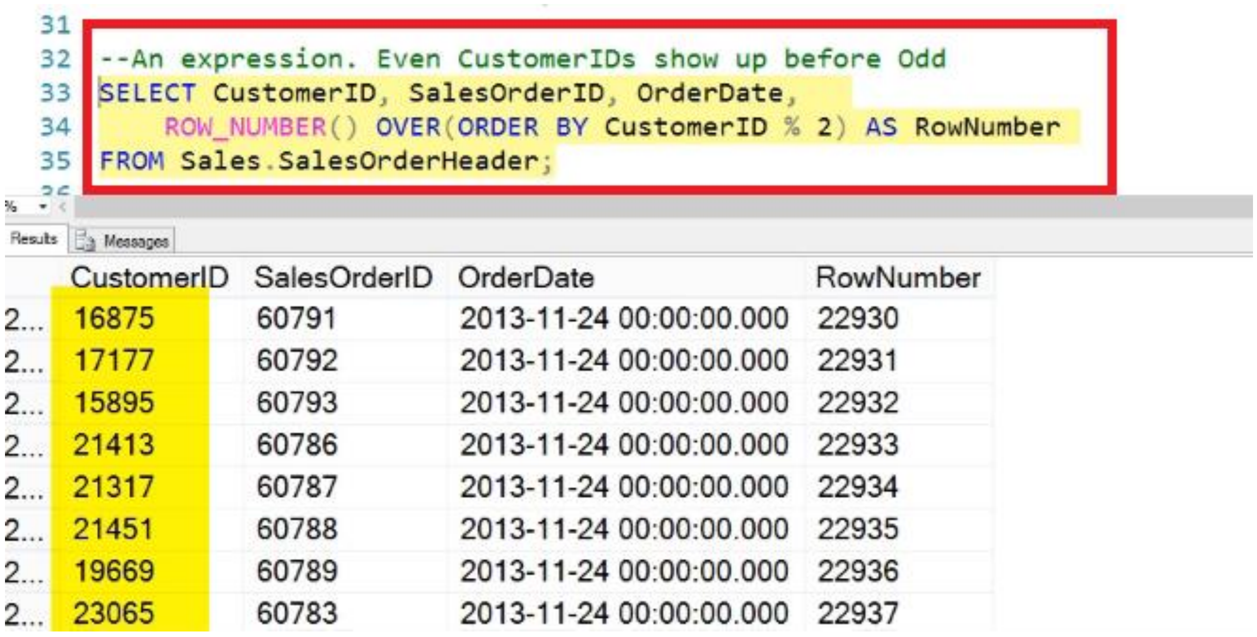
PARTITION BY

May be one or more olumns or expressions – even a subquery

Always supported and always optional

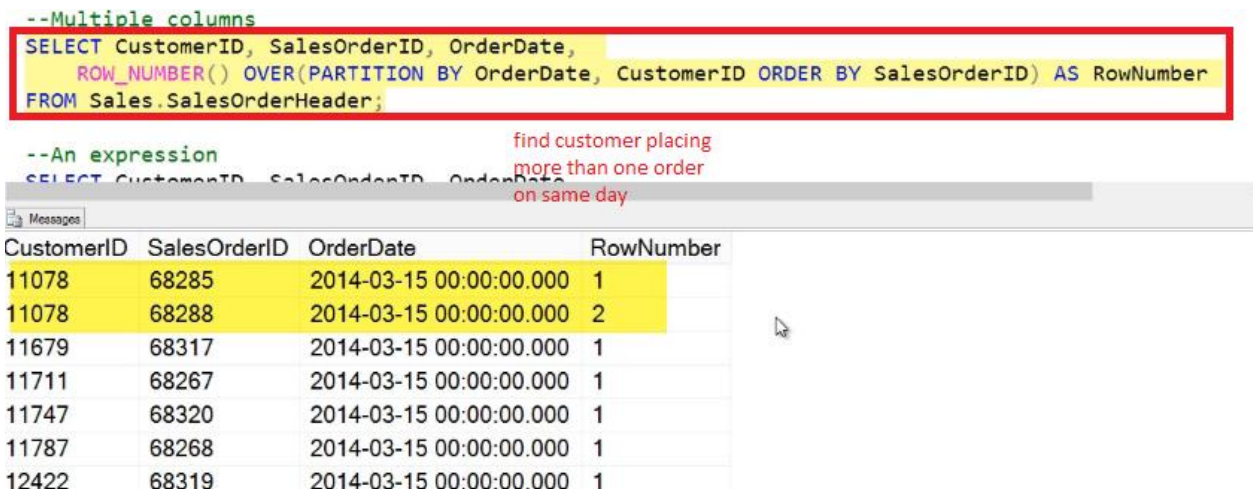
Diff b/w partition and group by

Partition by aggregates and display result for each row while group by gives a single row and other details are lost



```
--An expression. Even CustomerIDs show up before Odd
SELECT CustomerID, SalesOrderID, OrderDate,
    ROW_NUMBER() OVER(ORDER BY CustomerID % 2) AS RowNumber
FROM Sales.SalesOrderHeader;
```

```
Results  Messages
     CustomerID  SalesOrderID  OrderDate                RowNumber
2... 16875       60791         2013-11-24 00:00:00.000  22930
2... 17177       60792         2013-11-24 00:00:00.000  22931
2... 15895       60793         2013-11-24 00:00:00.000  22932
2... 21413       60786         2013-11-24 00:00:00.000  22933
2... 21317       60787         2013-11-24 00:00:00.000  22934
2... 21451       60788         2013-11-24 00:00:00.000  22935
2... 19669       60789         2013-11-24 00:00:00.000  22936
2... 23065       60783         2013-11-24 00:00:00.000  22937
```



```
--Multiple columns
SELECT CustomerID, SalesOrderID, OrderDate,
    ROW_NUMBER() OVER(PARTITION BY OrderDate, CustomerID ORDER BY SalesOrderID) AS RowNumber
FROM Sales.SalesOrderHeader;

--An expression
SELECT CustomerID, SalesOrderID, OrderDate
```

find customer placing more than one order on same day

```
Messages
CustomerID  SalesOrderID  OrderDate                RowNumber
11078       68285         2014-03-15 00:00:00.000  1
11078       68288         2014-03-15 00:00:00.000  2
11679       68317         2014-03-15 00:00:00.000  1
11711       68267         2014-03-15 00:00:00.000  1
11747       68320         2014-03-15 00:00:00.000  1
11787       68268         2014-03-15 00:00:00.000  1
12422       68319         2014-03-15 00:00:00.000  1
```

Ranking function

ROW NUMBER A RANDOM NUMBER WITH INCREASING VALUE WITHIN PARTITITON

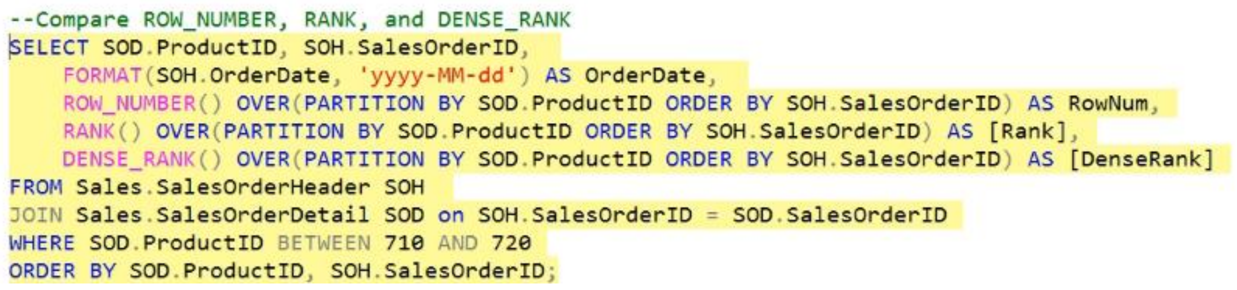
RANK GIVES RANK BASED ON ORDER BY CLAUSE BUT IF TWO RANKS ARE SAME IT SKIPS THE NEXT NO

DENSE RANK SIMILAR TO RANK GIVES RANK BASED ON ORDER BY CLAUSE BUT IF TWO RANKS ARE SAME IT DOES NOT SKIPS THE NEXT NO

Syntax

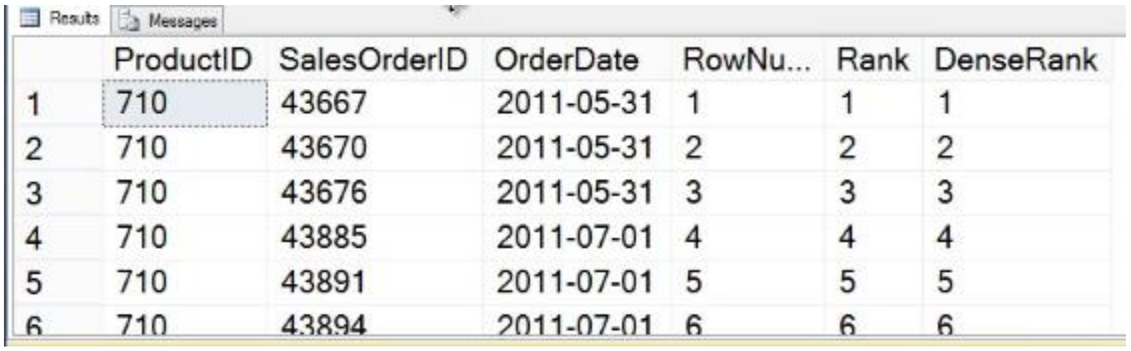
```
ROW_NUMBER|RANK|DENSE_RANK()
OVER([PARTITION BY <expression>] ORDER BY <expression>)

NTILE(<number of buckets>)
OVER([PARTITION BY <expression>] ORDER BY <expression>)
```

DEMO



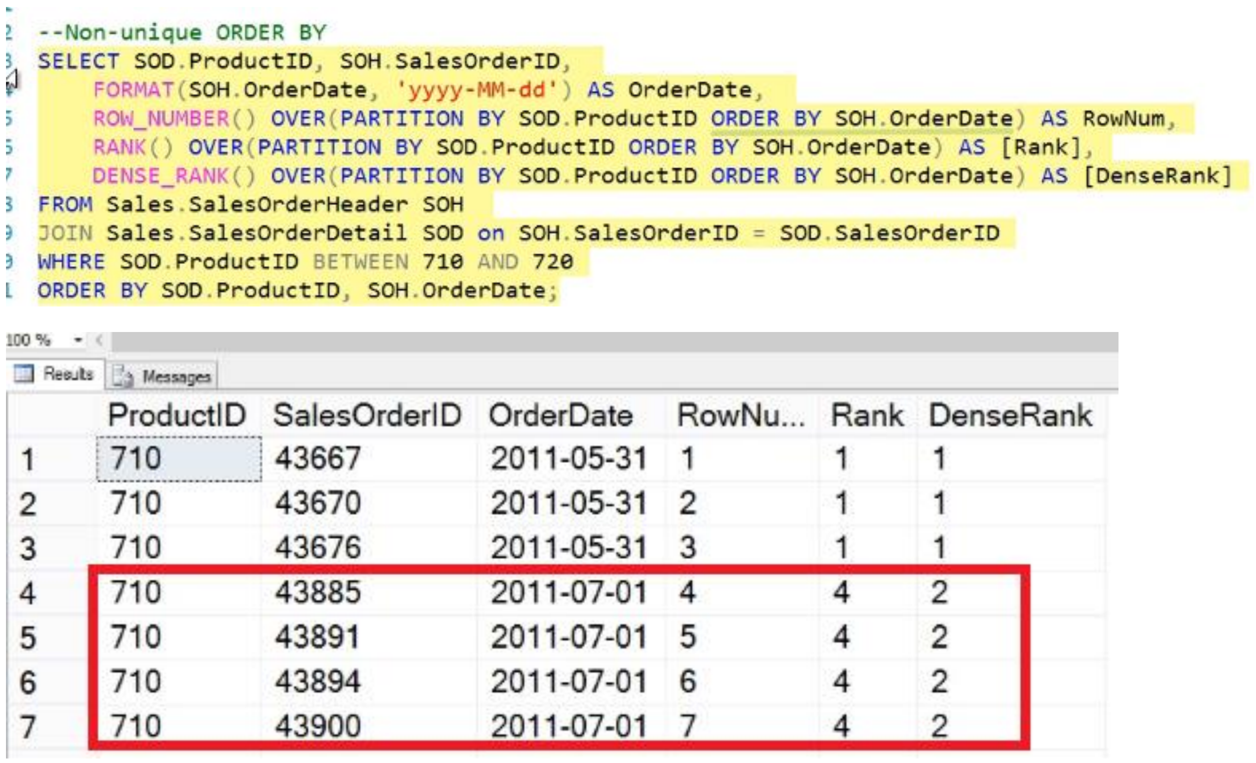
```
--Compare ROW_NUMBER, RANK, and DENSE_RANK
SELECT SOD.ProductID, SOH.SalesOrderID,
    FORMAT(SOH.OrderDate, 'yyyy-MM-dd') AS OrderDate,
    ROW_NUMBER() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.SalesOrderID) AS RowNum,
    RANK() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.SalesOrderID) AS [Rank],
    DENSE_RANK() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.SalesOrderID) AS [DenseRank]
FROM Sales.SalesOrderHeader SOH
JOIN Sales.SalesOrderDetail SOD on SOH.SalesOrderID = SOD.SalesOrderID
WHERE SOD.ProductID BETWEEN 710 AND 720
ORDER BY SOD.ProductID, SOH.SalesOrderID;
```

OPT



	ProductID	SalesOrderID	OrderDate	RowNu...	Rank	DenseRank
1	710	43667	2011-05-31	1	1	1
2	710	43670	2011-05-31	2	2	2
3	710	43676	2011-05-31	3	3	3
4	710	43885	2011-07-01	4	4	4
5	710	43891	2011-07-01	5	5	5
6	710	43894	2011-07-01	6	6	6

DEMO 2 WHEN SAME VALUE IS REPEATED



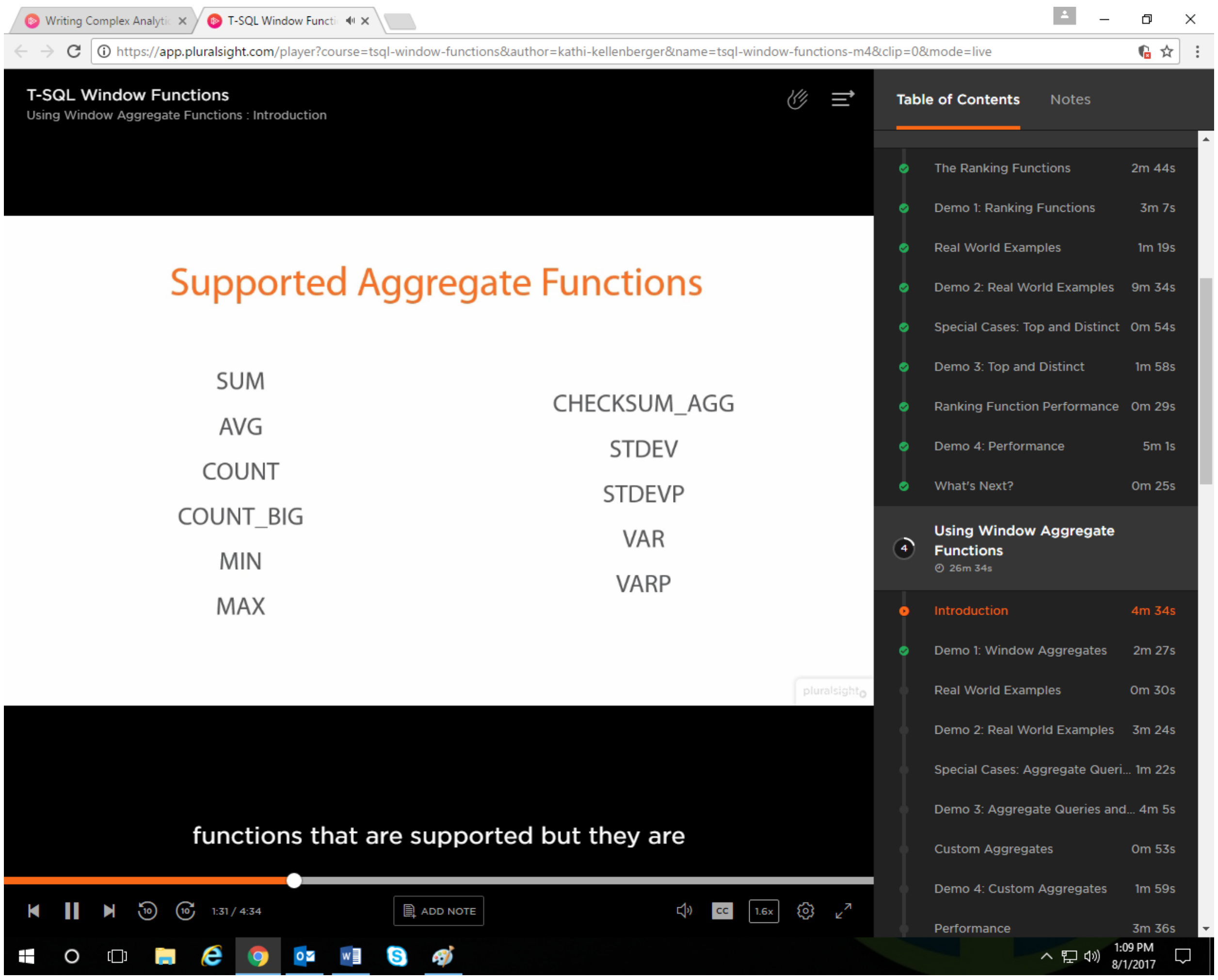
```sql
--Non-unique ORDER BY
SELECT SOD.ProductID, SOH.SalesOrderID,
    FORMAT(SOH.OrderDate, 'yyyy-MM-dd') AS OrderDate,
    ROW_NUMBER() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.OrderDate) AS RowNum,
    RANK() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.OrderDate) AS [Rank],
    DENSE_RANK() OVER(PARTITION BY SOD.ProductID ORDER BY SOH.OrderDate) AS [DenseRank]
FROM Sales.SalesOrderHeader SOH
JOIN Sales.SalesOrderDetail SOD on SOH.SalesOrderID = SOD.SalesOrderID
WHERE SOD.ProductID BETWEEN 710 AND 720
ORDER BY SOD.ProductID, SOH.OrderDate;
```

	ProductID	SalesOrderID	OrderDate	RowNu...	Rank	DenseRank
1	710	43667	2011-05-31	1	1	1
2	710	43670	2011-05-31	2	1	1
3	710	43676	2011-05-31	3	1	1
4	710	43885	2011-07-01	4	4	2
5	710	43891	2011-07-01	5	4	2
6	710	43894	2011-07-01	6	4	2
7	710	43900	2011-07-01	7	4	2

NOTE

1.DON’T USE ORDER BY CLAUSE TWICE IN THE SAME QUERRY INSTEAD USE NESTED QUERRIES

2.DONT USE DISTINCT AAND ROW NUMBER IN THE SAME QUERRY INSTEAD USE NESTED QUERRIES


Writing Complex Analytic
T-SQL Window Functi
https://app.pluralsight.com/player?course=tsql-window-functions&author=kathi-kellenberger&name=tsql-window-functions-m4&clip=0&mode=live
T-SQL Window Functions
Using Window Aggregate Functions : Introduction
Table of Contents
Notes
ADD NOTE
1:31 / 4:34
1.6x
1:09 PM
8/1/2017

Supported Aggregate Functions

SUM
AVG
COUNT
COUNT_BIG
MIN
MAX

CHECKSUM_AGG
STDEV
STDEVP
VAR
VARP

functions that are supported but they are

Syntax

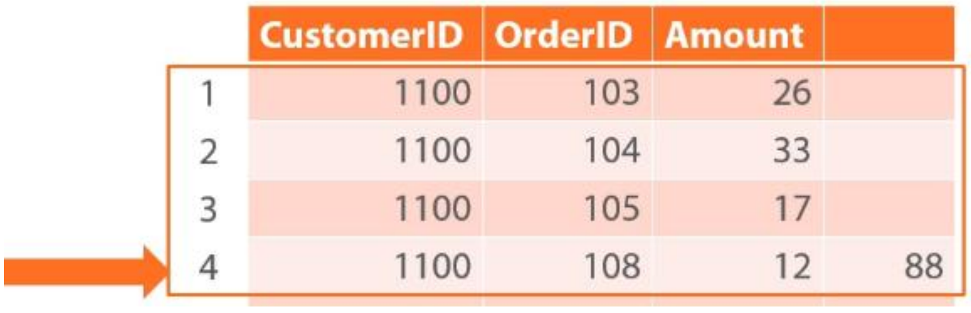
```
AggFunction(<expression>) OVER()
COUNT(*) OVER()

AggFunction(<expression>) OVER(PARTITION BY <expression>)
COUNT(*) OVER(PARTITION BY CustomerID)
```

~~AggFunction(<expression>) OVER(ORDER BY <expression>)~~

Framing Terms

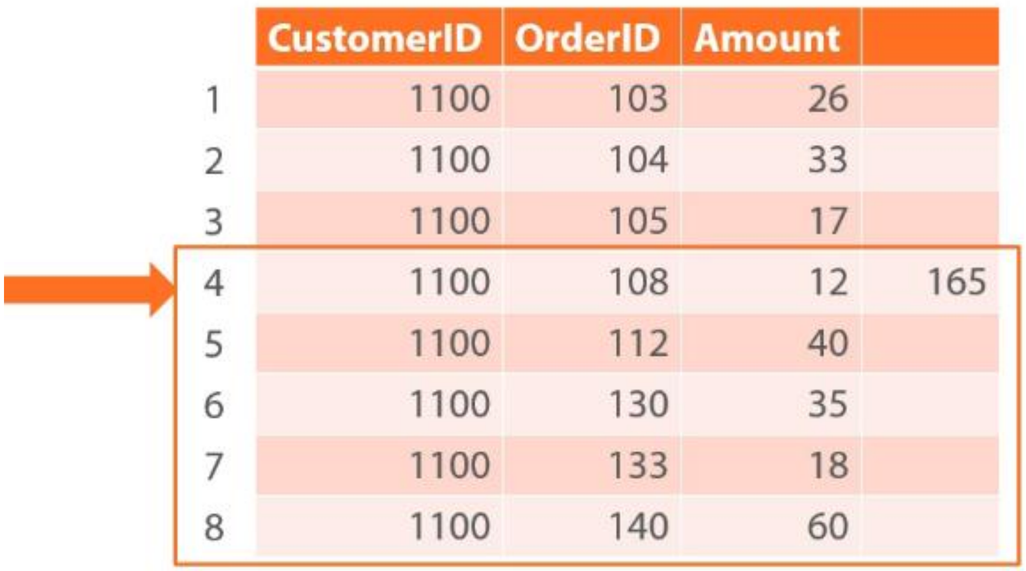
Current Row	Unbounded Preceding	Unbounded Following
Between	N Preceding	N Following

```
ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW
```

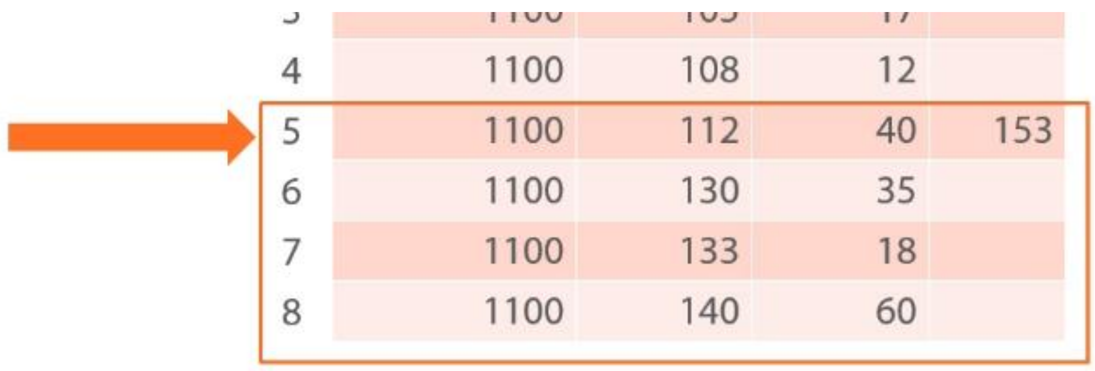


	CustomerID	OrderID	Amount	
1	1100	103	26	
2	1100	104	33	
3	1100	105	17	
4	1100	108	12	88

```
ROWS BETWEEN CURRENT ROW AND UNBOUNDED FOLLOWING
```

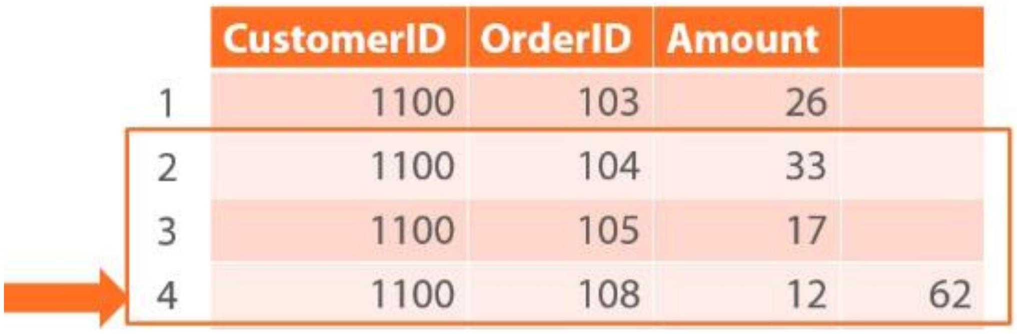


	CustomerID	OrderID	Amount	
1	1100	103	26	
2	1100	104	33	
3	1100	105	17	
4	1100	108	12	165
5	1100	112	40	
6	1100	130	35	
7	1100	133	18	
8	1100	140	60	



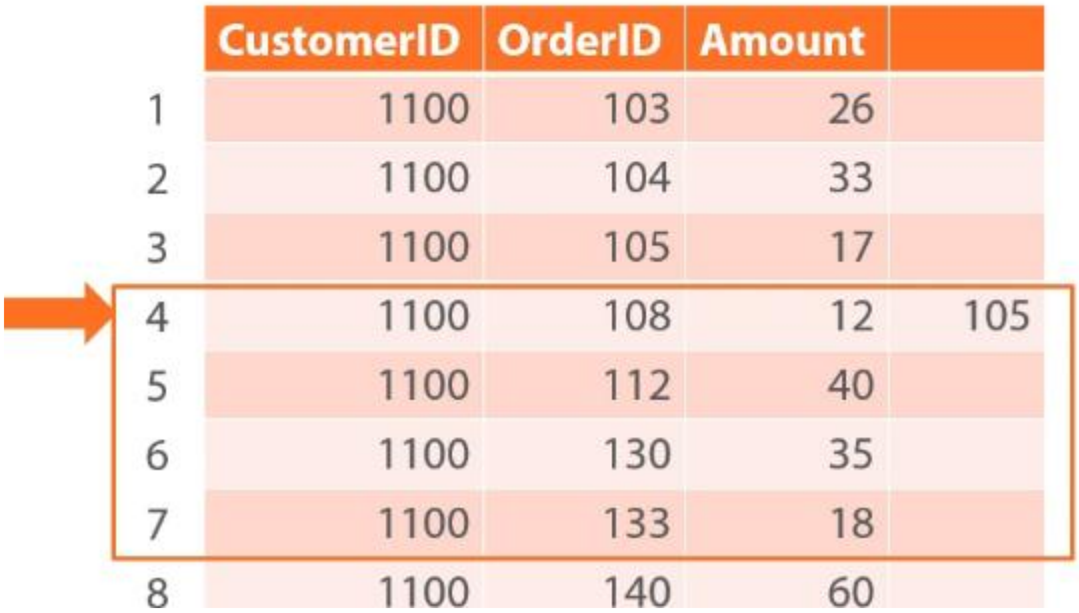
4	1100	108	12	
5	1100	112	40	153
6	1100	130	35	
7	1100	133	18	
8	1100	140	60	

```
ROWS BETWEEN 2 PRECEDING AND CURRENT ROW
```



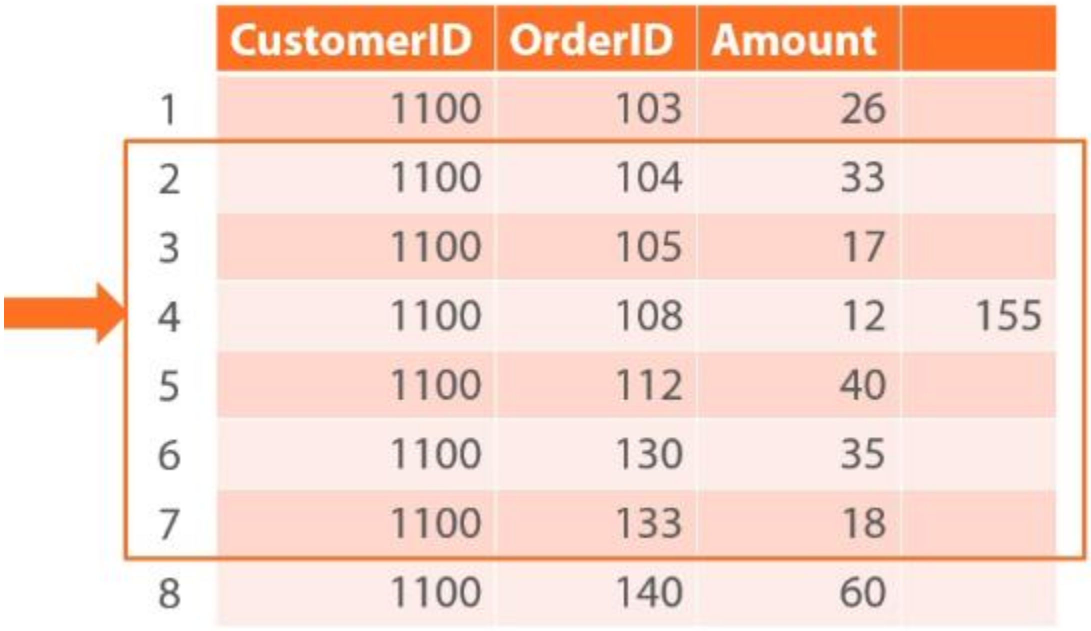
	CustomerID	OrderID	Amount	
1	1100	103	26	
2	1100	104	33	
3	1100	105	17	
4	1100	108	12	62

```
ROWS BETWEEN CURRENT ROW AND 3 FOLLOWING
```

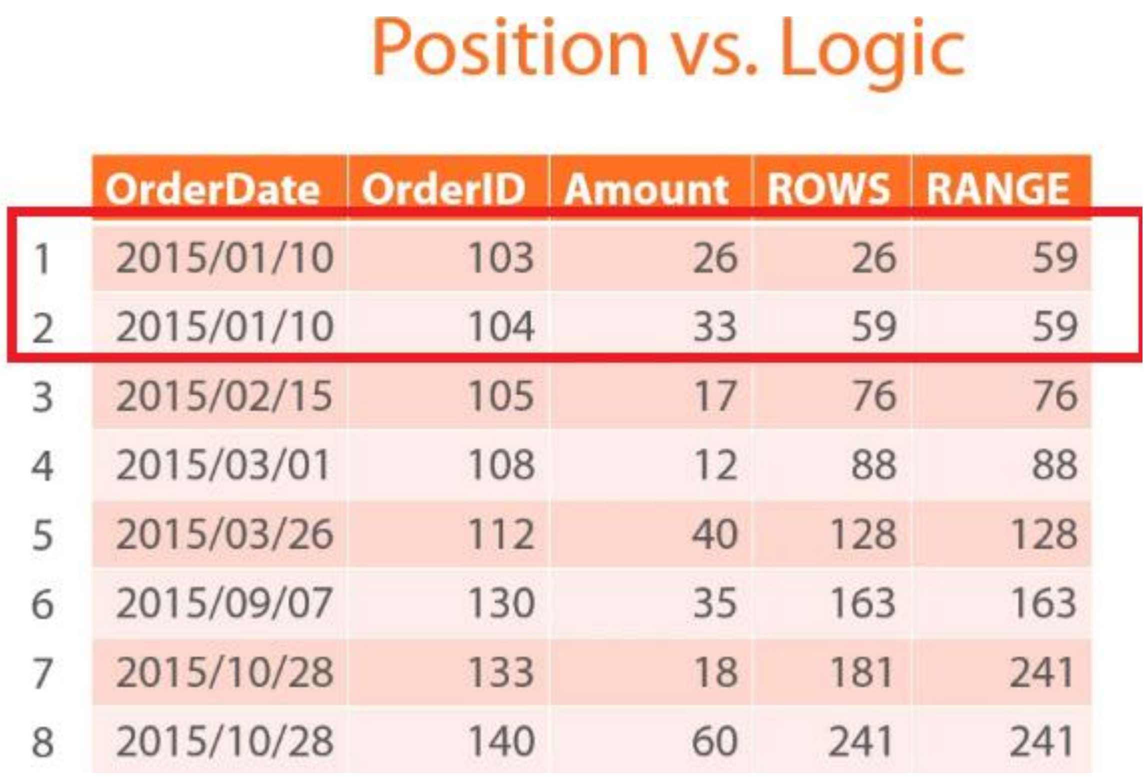


	CustomerID	OrderID	Amount	
1	1100	103	26	
2	1100	104	33	
3	1100	105	17	
4	1100	108	12	105
5	1100	112	40	
6	1100	130	35	
7	1100	133	18	
8	1100	140	60	

```
ROWS BETWEEN 2 PRECEDING AND 3 FOLLOWING
```



	CustomerID	OrderID	Amount	
1	1100	103	26	
2	1100	104	33	
3	1100	105	17	
4	1100	108	12	155
5	1100	112	40	
6	1100	130	35	
7	1100	133	18	
8	1100	140	60	

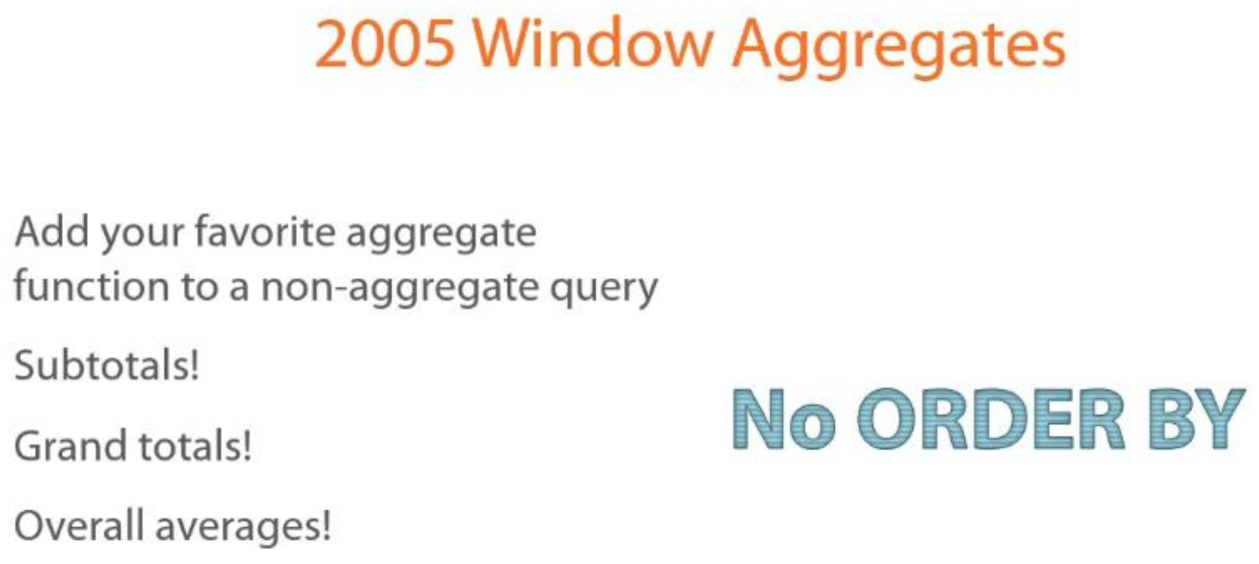


Position vs. Logic

	OrderDate	OrderID	Amount	ROWS	RANGE
1	2015/01/10	103	26	26	59
2	2015/01/10	104	33	59	59
3	2015/02/15	105	17	76	76
4	2015/03/01	108	12	88	88
5	2015/03/26	112	40	128	128
6	2015/09/07	130	35	163	163
7	2015/10/28	133	18	181	241
8	2015/10/28	140	60	241	241

Rows give the actual positional value but range gives logical value see from the above example order by order date gives the same range as the date is same but row give the actual sum

2.rows give higher performance



2005 Window Aggregates

Add your favorite aggregate function to a non-aggregate query

Subtotals!

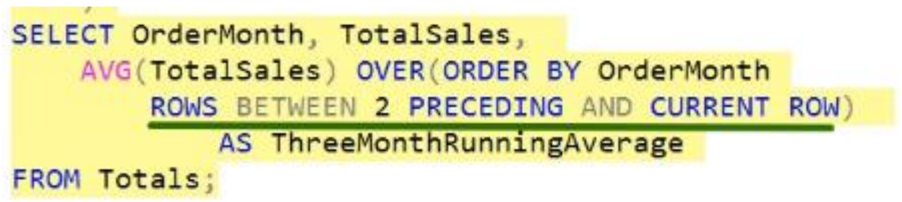
Grand totals!

Overall averages!

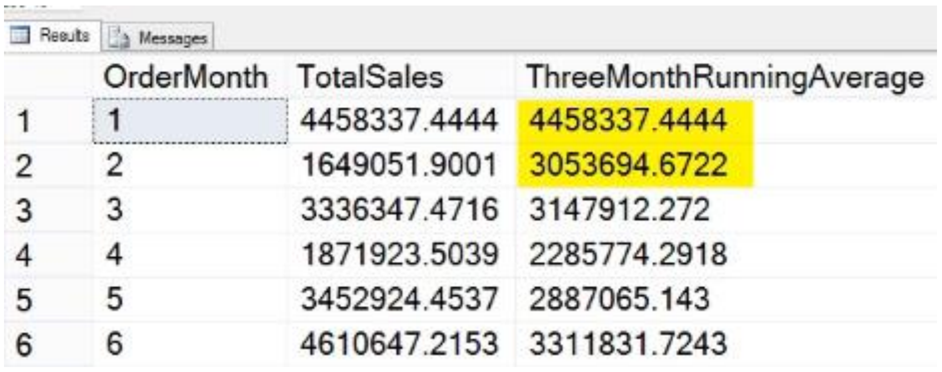
No ORDER BY

Syntax

```
AggregateFunction(<expression>)
OVER([PARTITION BY <expression>] ORDER BY<expression>
[ROWS|RANGE <expression>])

DEFAULT FRAME: RANGE BETWEEN UNBOUNDED PRECEDING AND
CURRENT ROW
```



```
SELECT OrderMonth, TotalSales,
    AVG(TotalSales) OVER(ORDER BY OrderMonth
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
            AS ThreeMonthRunningAverage
FROM Totals;
```



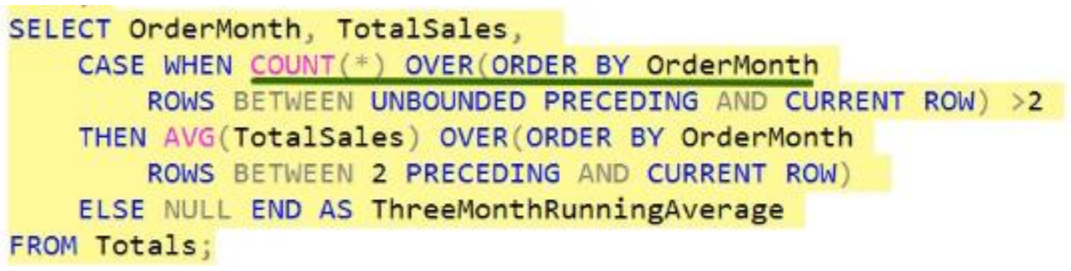
Results
Messages

	OrderMonth	TotalSales	ThreeMonthRunningAverage
1	1	4458337.4444	4458337.4444
2	2	1649051.9001	3053694.6722
3	3	3336347.4716	3147912.272
4	4	1871923.5039	2285774.2918
5	5	3452924.4537	2887065.143
6	6	4610647.2153	3311831.7243

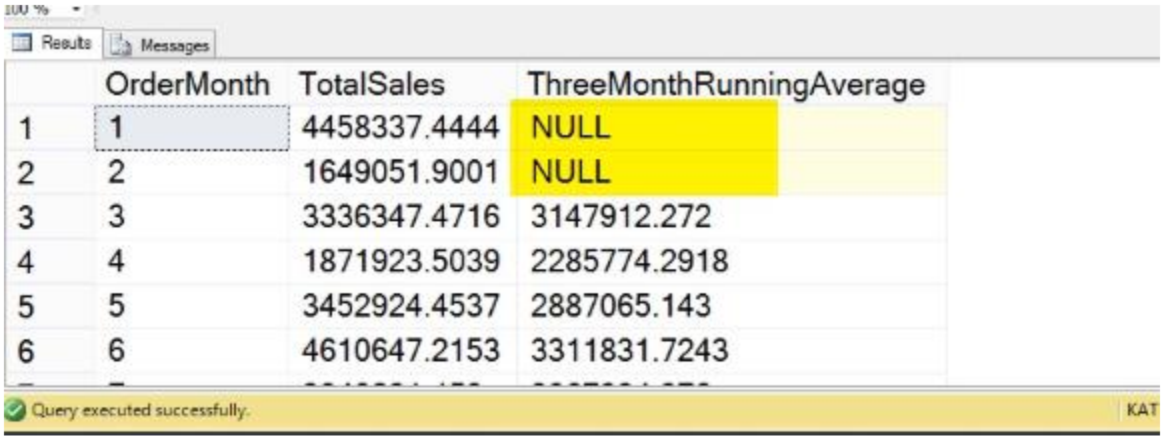
Output:

For the first two rows the output will be the no and the avg of 1 2 nos resp.

Instead we can use null for those rows using case statement



```
SELECT OrderMonth, TotalSales,
    CASE WHEN COUNT(*) OVER(ORDER BY OrderMonth
        ROWS BETWEEN UNBOUNDED PRECEDING AND CURRENT ROW) >2
    THEN AVG(TotalSales) OVER(ORDER BY OrderMonth
        ROWS BETWEEN 2 PRECEDING AND CURRENT ROW)
    ELSE NULL END AS ThreeMonthRunningAverage
FROM Totals;
```



100 %

Results | Messages

	OrderMonth	TotalSales	ThreeMonthRunningAverage
1	1	4458337.4444	NULL
2	2	1649051.9001	NULL
3	3	3336347.4716	3147912.272
4	4	1871923.5039	2285774.2918
5	5	3452924.4537	2887065.143
6	6	4610647.2153	3311831.7243

Query executed successfully.

KAT

Offset Functions

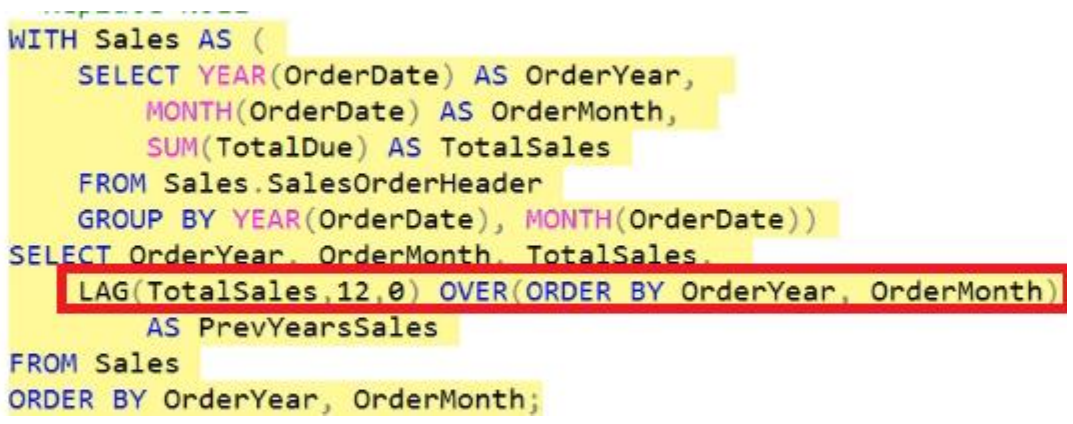
- LAG
- LEAD
- FIRST_VALUE
- LAST_VALUE

LAG and LEAD

- Include a column from another row
- LAG: Previous row
- LEAD: Next row
- ORDER BY required
- No frame

Syntax

```
LAG | LEAD(<expression>)
OVER([PARTITION BY <expression>] ORDER BY <expression>);


LAG | LEAD(<expression>[,<offset>][,<default>])
OVER([PARTITION BY <expression>] ORDER BY <expression>);
```



```
WITH Sales AS (
    SELECT YEAR(OrderDate) AS OrderYear,
        MONTH(OrderDate) AS OrderMonth,
        SUM(TotalDue) AS TotalSales
    FROM Sales.SalesOrderHeader
    GROUP BY YEAR(OrderDate), MONTH(OrderDate))
SELECT OrderYear, OrderMonth, TotalSales,
    LAG(TotalSales,12,0) OVER(ORDER BY OrderYear, OrderMonth)
        AS PrevYearsSales
FROM Sales
ORDER BY OrderYear, OrderMonth;
```

LAG (TotalSales,12,0) → 12 compare 12 row with 1 row and 0 means replace null values for first 11 rows with 0

FIRST_VALUE and LAST_VALUE

- First or last row of the partition
- FIRST_VALUE retrieves from the first row
- LAST_VALUE retrieves from the last row
- ORDER BY required
- Frame required

Syntax

```
FIRST_VALUE | LAST_VALUE(<expression>)
OVER([PARTITION BY <expression>] ORDER BY <expression>
<frame definition>);
```

SYNTAX

```
PERCENT_RANK | CUME_DIST()
OVER([PARTITION BY <expression>] ORDER BY <expression>)

Formula
PERCENT_RANK: (RANK – 1)/(N – 1)
CUME_DIST: RANK/N
```