SQL Server – Aggregate Functions with OVER Clause

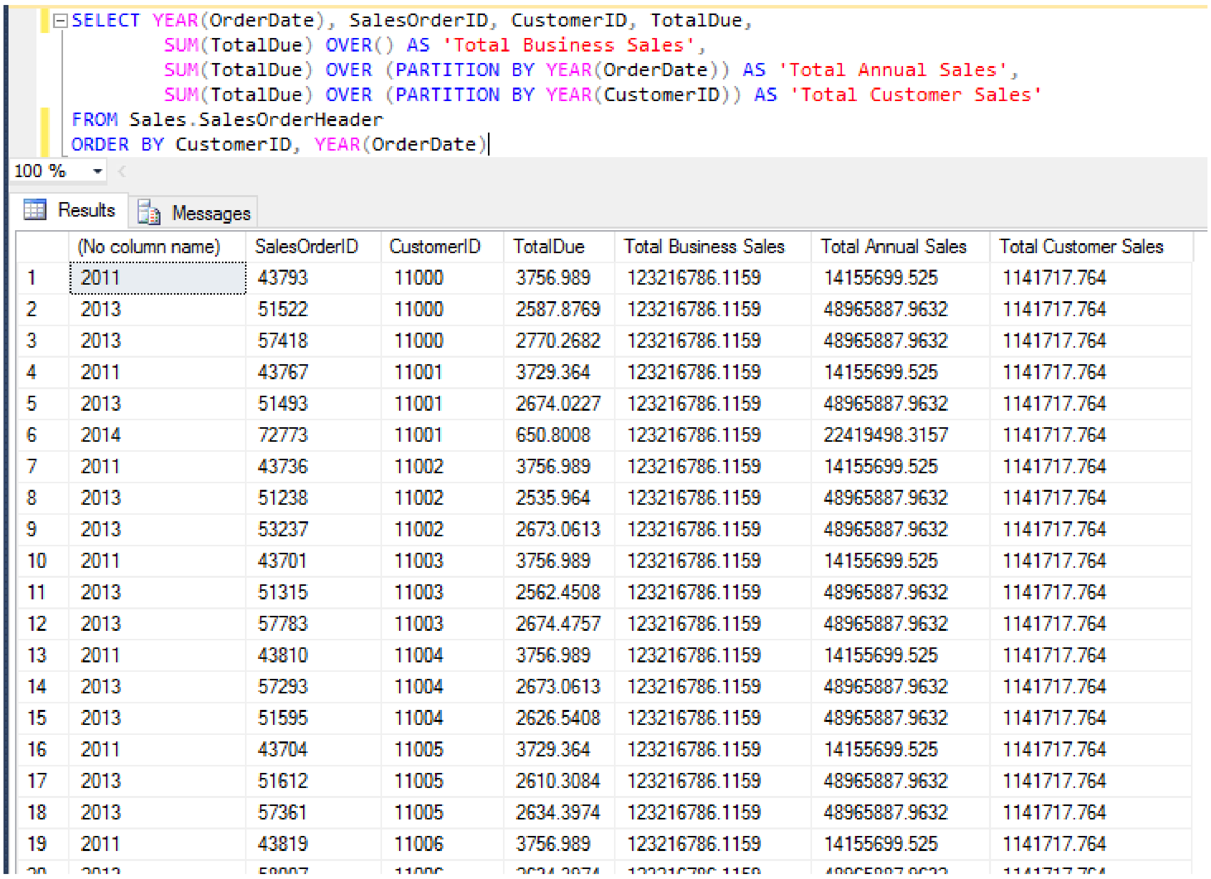
Did you know that you can use the SQL Server aggregate functions SUM, COUNT, MAX, MIN and AVG with an OVER Clause now?

Using an OVER clause you can produce individual record values along with aggregate values to different levels, without using a GROUP BY clause.

- Aggregate to different Levels
- Generate a Running Total

Aggregate To Different Levels with OVER (PARTITION BY ….)

The following example shows individual order records along with the grand total of all sales, the annual revenue for the year of the order and the total customer revenue for the customer that placed the order.



```sql
SELECT YEAR(OrderDate), SalesOrderID, CustomerID, TotalDue,
        SUM(TotalDue) OVER() AS 'Total Business Sales',
        SUM(TotalDue) OVER (PARTITION BY YEAR(OrderDate)) AS 'Total Annual Sales',
        SUM(TotalDue) OVER (PARTITION BY YEAR(CustomerID)) AS 'Total Customer Sales'
FROM Sales.SalesOrderHeader
ORDER BY CustomerID, YEAR(OrderDate)
```

	(No column name)	SalesOrderID	CustomerID	TotalDue	Total Business Sales	Total Annual Sales	Total Customer Sales
1	2011	43793	11000	3756.989	123216786.1159	14155699.525	1141717.764
2	2013	51522	11000	2587.8769	123216786.1159	48965887.9632	1141717.764
3	2013	57418	11000	2770.2682	123216786.1159	48965887.9632	1141717.764
4	2011	43767	11001	3729.364	123216786.1159	14155699.525	1141717.764
5	2013	51493	11001	2674.0227	123216786.1159	48965887.9632	1141717.764
6	2014	72773	11001	650.8008	123216786.1159	22419498.3157	1141717.764
7	2011	43736	11002	3756.989	123216786.1159	14155699.525	1141717.764
8	2013	51238	11002	2535.964	123216786.1159	48965887.9632	1141717.764
9	2013	53237	11002	2673.0613	123216786.1159	48965887.9632	1141717.764
10	2011	43701	11003	3756.989	123216786.1159	14155699.525	1141717.764
11	2013	51315	11003	2562.4508	123216786.1159	48965887.9632	1141717.764
12	2013	57783	11003	2674.4757	123216786.1159	48965887.9632	1141717.764
13	2011	43810	11004	3756.989	123216786.1159	14155699.525	1141717.764
14	2013	57293	11004	2673.0613	123216786.1159	48965887.9632	1141717.764
15	2013	51595	11004	2626.5408	123216786.1159	48965887.9632	1141717.764
16	2011	43704	11005	3729.364	123216786.1159	14155699.525	1141717.764
17	2013	51612	11005	2610.3084	123216786.1159	48965887.9632	1141717.764
18	2013	57361	11005	2634.3974	123216786.1159	48965887.9632	1141717.764
19	2011	43819	11006	3756.989	123216786.1159	14155699.525	1141717.764
20	2013	[illegible]	11006	2634.3974	123216786.1159	48965887.9632	1141717.764

```
SUM(TotalDue) OVER() AS 'Total Business Sales'
```

This expression produces a grand total across the whole data set. There is no partitioning of the data. This is why every record shows the same value for the “Total Business Sales” column.

```
SUM(TotalDue) OVER (PARTITION BY YEAR(OrderDate)) AS 'Total Annual Sales'
```

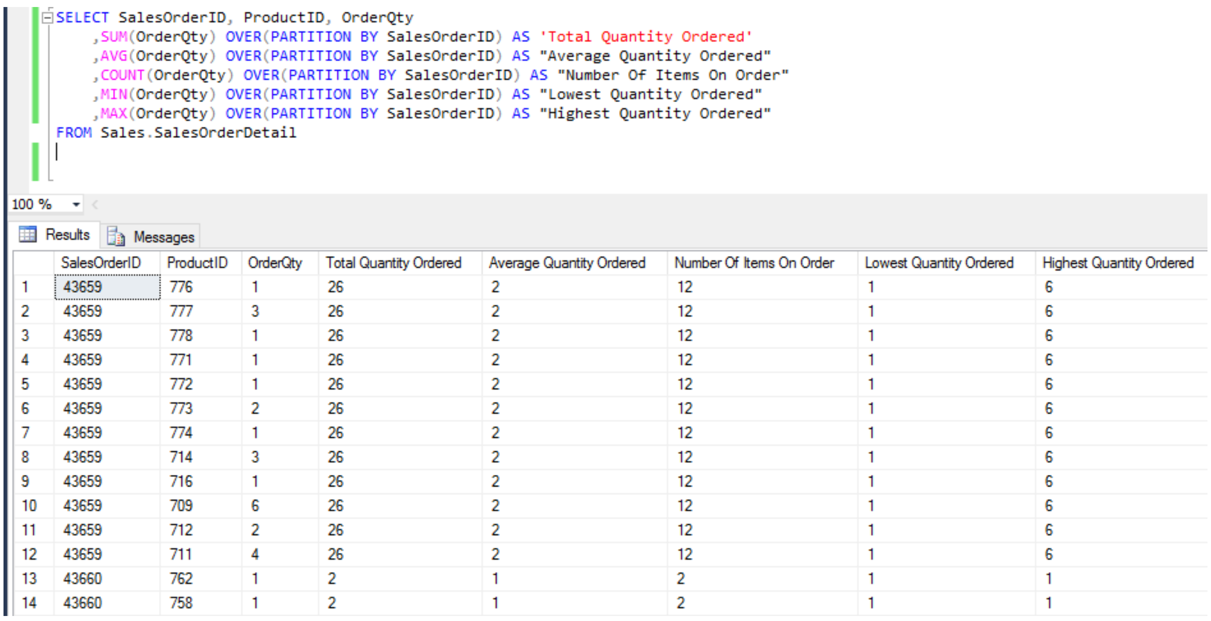
This expression instructs SQL Server to group (partition) the data by the YEAR of the orderdate and produce an annual sales total. You will see that this value is the same for each common year.

```
SUM(TotalDue) OVER (PARTITION BY CustomerID) AS 'Total Customer Sales'
```

This expression instructs SQL Server to group (partition) the data by the CustomerID and produce a customer sales total. You will see that this value is identical where the CustomerID for an order is the same.

The OVER clause can be used with all of the aggregate functions.

The following example displays individual SalesOrderDetail records along with the total quantity ordered for the order, the average order quantity for the order, the number of items on the order, the lowest order quantity on the order and the highest quantity on the order.



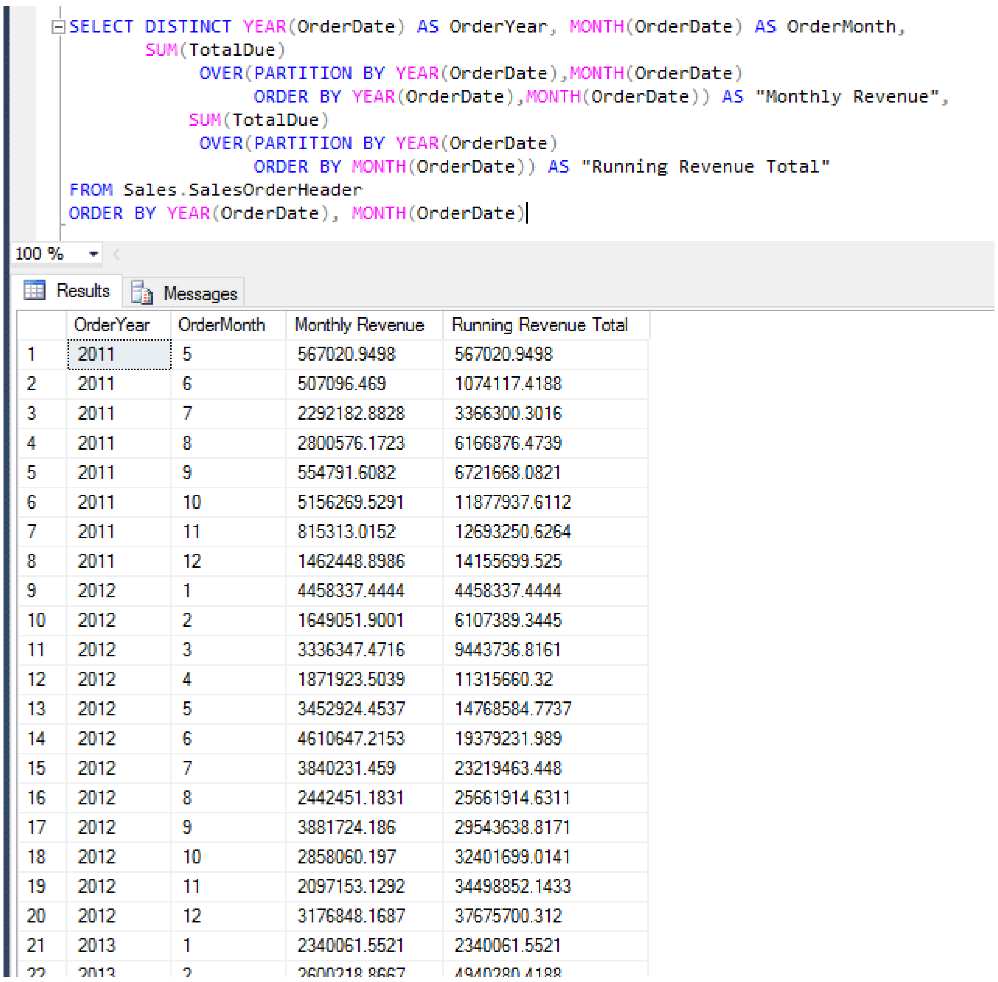
```
SELECT SalesOrderID, ProductID, OrderQty
      ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total Quantity Ordered'
      ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Average Quantity Ordered"
      ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Number Of Items On Order"
      ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Lowest Quantity Ordered"
      ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS "Highest Quantity Ordered"
FROM Sales.SalesOrderDetail
```

	SalesOrderID	ProductID	OrderQty	Total Quantity Ordered	Average Quantity Ordered	Number Of Items On Order	Lowest Quantity Ordered	Highest Quantity Ordered
1	43659	776	1	26	2	12	1	6
2	43659	777	3	26	2	12	1	6
3	43659	778	1	26	2	12	1	6
4	43659	771	1	26	2	12	1	6
5	43659	772	1	26	2	12	1	6
6	43659	773	2	26	2	12	1	6
7	43659	774	1	26	2	12	1	6
8	43659	714	3	26	2	12	1	6
9	43659	716	1	26	2	12	1	6
10	43659	709	6	26	2	12	1	6
11	43659	712	2	26	2	12	1	6
12	43659	711	4	26	2	12	1	6
13	43660	762	1	2	1	2	1	1
14	43660	758	1	2	1	2	1	1

Running Totals With The ORDER BY Sub Clause

The ORDER BY sub clause enables a running total to be generated.

The following example shows the monthly revenue for each month, along with the running total for the year.



```
SELECT DISTINCT YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
       SUM(TotalDue)
            OVER(PARTITION BY YEAR(OrderDate),MONTH(OrderDate)
                 ORDER BY YEAR(OrderDate),MONTH(OrderDate)) AS "Monthly Revenue",
           SUM(TotalDue)
            OVER(PARTITION BY YEAR(OrderDate)
                 ORDER BY MONTH(OrderDate)) AS "Running Revenue Total"
FROM Sales.SalesOrderHeader
ORDER BY YEAR(OrderDate), MONTH(OrderDate)
```

	OrderYear	OrderMonth	Monthly Revenue	Running Revenue Total
1	2011	5	567020.9498	567020.9498
2	2011	6	507096.469	1074117.4188
3	2011	7	2292182.8828	3366300.3016
4	2011	8	2800576.1723	6166876.4739
5	2011	9	554791.6082	6721668.0821
6	2011	10	5156269.5291	11877937.6112
7	2011	11	815313.0152	12693250.6264
8	2011	12	1462448.8986	14155699.525
9	2012	1	4458337.4444	4458337.4444
10	2012	2	1649051.9001	6107389.3445
11	2012	3	3336347.4716	9443736.8161
12	2012	4	1871923.5039	11315660.32
13	2012	5	3452924.4537	14768584.7737
14	2012	6	4610647.2153	19379231.989
15	2012	7	3840231.459	23219463.448
16	2012	8	2442451.1831	25661914.6311
17	2012	9	3881724.186	29543638.8171
18	2012	10	2858060.197	32401699.0141
19	2012	11	2097153.1292	34498852.1433
20	2012	12	3176848.1687	37675700.312
21	2013	1	2340061.5521	2340061.5521
22	2013	2	2600218.8667	4940280.4188

We see that the totals for each month are different and the fourth column shows the total growing through the months of the year, but the running total starts again when a new year starts.

If we look at December 2011, the value **14155699.525** is the total of all 12 months for 2011.

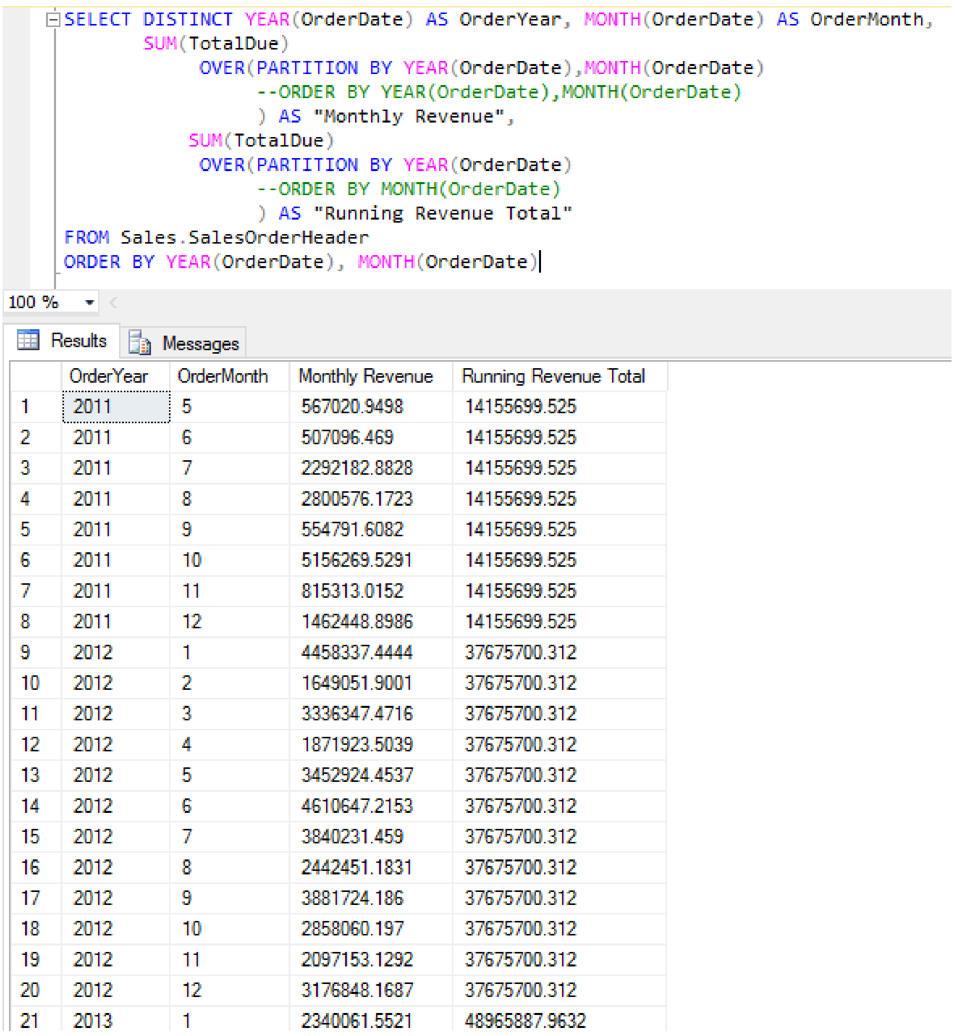
If we look at June 2011, the value **1074117.4188** is the total for May and June 2011.

```
SUM(TotalDue)
            OVER(PARTITION BY YEAR(OrderDate)
                   ORDER BY MONTH(OrderDate)) AS "Running Revenue Total"
```

The OVER clause for the fourth column, “Running Revenue Total” has both a PARTITION and ORDER BY sub clause.

- The PARTITION BY defines the group to be the YEAR
- The ORDER BY defines that we evaluate the annual total after each month

Without the ORDER BY Clause the annual total is evaluated after each year (the partition determines the order). We can see this in the following example:



```
SELECT DISTINCT YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
       SUM(TotalDue)
            OVER(PARTITION BY YEAR(OrderDate),MONTH(OrderDate)
                 --ORDER BY YEAR(OrderDate),MONTH(OrderDate)
                 ) AS "Monthly Revenue",
           SUM(TotalDue)
            OVER(PARTITION BY YEAR(OrderDate)
                 --ORDER BY MONTH(OrderDate)
                 ) AS "Running Revenue Total"
FROM Sales.SalesOrderHeader
ORDER BY YEAR(OrderDate), MONTH(OrderDate)
```

100 %

Results | Messages

	OrderYear	OrderMonth	Monthly Revenue	Running Revenue Total
1	2011	5	567020.9498	14155699.525
2	2011	6	507096.469	14155699.525
3	2011	7	2292182.8828	14155699.525
4	2011	8	2800576.1723	14155699.525
5	2011	9	554791.6082	14155699.525
6	2011	10	5156269.5291	14155699.525
7	2011	11	815313.0152	14155699.525
8	2011	12	1462448.8986	14155699.525
9	2012	1	4458337.4444	37675700.312
10	2012	2	1649051.9001	37675700.312
11	2012	3	3336347.4716	37675700.312
12	2012	4	1871923.5039	37675700.312
13	2012	5	3452924.4537	37675700.312
14	2012	6	4610647.2153	37675700.312
15	2012	7	3840231.459	37675700.312
16	2012	8	2442451.1831	37675700.312
17	2012	9	3881724.186	37675700.312
18	2012	10	2858060.197	37675700.312
19	2012	11	2097153.1292	37675700.312
20	2012	12	3176848.1687	37675700.312
21	2013	1	2340061.5521	48965887.9632

The examples are based on the AdventureWorks2014 database. Here is the code for the queries used in this document:

```
--individual order records along with the grand total of all sales,
--the annual revenue for the year of the order and the
--total customer revenue for the customer that placed the order.
SELECT YEAR(OrderDate), SalesOrderID, CustomerID, TotalDue,
              SUM(TotalDue) OVER() AS 'Total Business Sales',
              SUM(TotalDue) OVER (PARTITION BY YEAR(OrderDate)) AS 'Total Annual Sales',
              SUM(TotalDue) OVER (PARTITION BY CustomerID) AS 'Total Customer Sales'
FROM Sales.SalesOrderHeader
ORDER BY CustomerID, YEAR(OrderDate)


--Individual SalesOrderDetail records along with the total quantity ordered for the order,
--the average order quantity for the order, the number of items on the order,
--the lowest order quantity on the order and the highest quantity on the order
SELECT SalesOrderID, ProductID, OrderQty
    ,SUM(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Total Quantity Ordered'
    ,AVG(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Average Quantity Ordered'
    ,COUNT(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Number Of Items On Order'
    ,MIN(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Lowest Quantity Ordered'
    ,MAX(OrderQty) OVER(PARTITION BY SalesOrderID) AS 'Highest Quantity Ordered'
FROM Sales.SalesOrderDetail


--Monthly Total & Running Total within each year

SELECT DISTINCT YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
          SUM(TotalDue)
                      OVER(PARTITION BY YEAR(OrderDate),MONTH(OrderDate)
                           ORDER BY YEAR(OrderDate),MONTH(OrderDate)) AS "Monthly Revenue",
                 SUM(TotalDue)
                      OVER(PARTITION BY YEAR(OrderDate)
                           ORDER BY MONTH(OrderDate)) AS "Running Revenue Total"
FROM Sales.SalesOrderHeader
ORDER BY YEAR(OrderDate), MONTH(OrderDate)

--Monthly Total & Annual Total
SELECT DISTINCT YEAR(OrderDate) AS OrderYear, MONTH(OrderDate) AS OrderMonth,
          SUM(TotalDue)
                      OVER(PARTITION BY YEAR(OrderDate),MONTH(OrderDate)
                           --ORDER BY YEAR(OrderDate),MONTH(OrderDate)
                              ) AS "Monthly Revenue",
                 SUM(TotalDue)
                      OVER(PARTITION BY YEAR(OrderDate)
                           --ORDER BY MONTH(OrderDate)
                              ) AS "Running Revenue Total"
FROM Sales.SalesOrderHeader
ORDER BY YEAR(OrderDate), MONTH(OrderDate)
```

If you are itching to learn more why not book on to our SQL Server Database Querying training courses? *This link will take you to the course outlines:*

http://ptr.co.uk/databases-business-intelligence-courses.