Assignment 1

Exercise: Windows function and partition by

Ques 1

Create a query with the following columns:

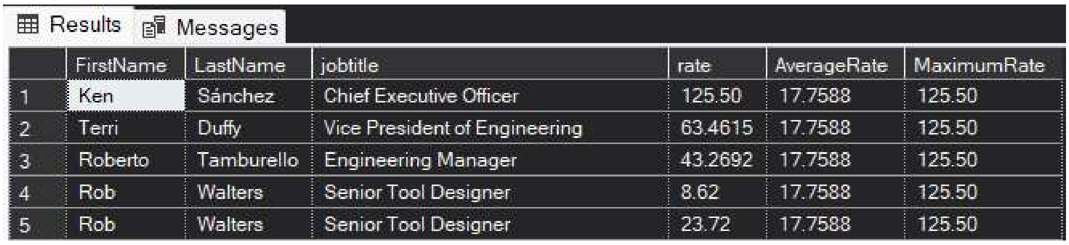
- FirstName and LastName, from the Person.Person table**
- JobTitle, from the HumanResources.Employee table**
- Rate, from the HumanResources.EmployeePayHistory table**
- A derived column called "AverageRate" that returns the average of all values in the "Rate" column, in each row
- Note: ***All the above tables can be joined on BusinessEntityID***

```
select FirstName,LastName,jobtitle,rate,avg(rate) over() as AverageRate
from person.Person pp
inner join HumanResources.Employee hre
on hre.BusinessEntityID = pp.BusinessEntityID
inner join HumanResources.EmployeePayHistory hrep
on hrep.BusinessEntityID = pp.BusinessEntityID
```

Ques 2

Enhance your query from Exercise 1 by adding a derived column called "MaximumRate" that returns the largest of all values in the "Rate" column, in each row.

```
select FirstName,LastName,jobtitle,rate,avg(rate) over() as AverageRate,
max(rate) over() as MaximumRate -- maximum rate column added
from person.Person pp
inner join person.businessentity pb
on pp.BusinessEntityID = pb.BusinessEntityID
inner join HumanResources.Employee hre
on hre.BusinessEntityID = pb.BusinessEntityID
inner join HumanResources.EmployeePayHistory hrep
on hrep.BusinessEntityID=pb.BusinessEntityID
```



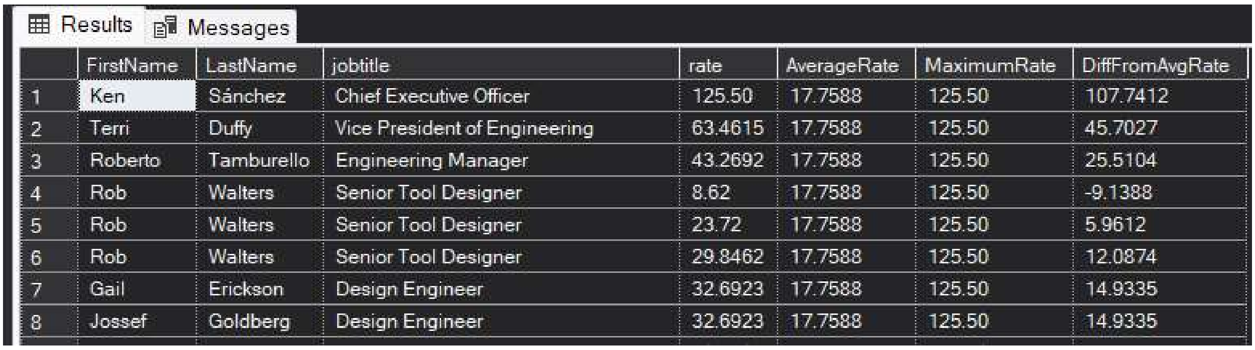
	FirstName	LastName	jobtitle	rate	AverageRate	MaximumRate
1	Ken	Sánchez	Chief Executive Officer	125.50	17.7588	125.50
2	Terri	Duffy	Vice President of Engineering	63.4615	17.7588	125.50
3	Roberto	Tamburello	Engineering Manager	43.2692	17.7588	125.50
4	Rob	Walters	Senior Tool Designer	8.62	17.7588	125.50
5	Rob	Walters	Senior Tool Designer	23.72	17.7588	125.50

Ques 3

Enhance your query from ques 2 by adding a derived column called "DiffFromAvgRate" that returns the result of the following calculation: An employees's pay rate, MINUS the average of all values in the "Rate" column.

```
Select *,(rate-averagerate) as DiffFromAvgRate
from
```

```
(
select FirstName,LastName,jobtitle,rate,avg(rate) over() as AverageRate,
max(rate) over() as MaximumRate -- maximum rate column added
from person.Person pp
inner join HumanResources.Employee hre
on hre.BusinessEntityID = pp.BusinessEntityID
inner join HumanResources.EmployeePayHistory hrep
on hrep.BusinessEntityID = pp.BusinessEntityID
)temp
```

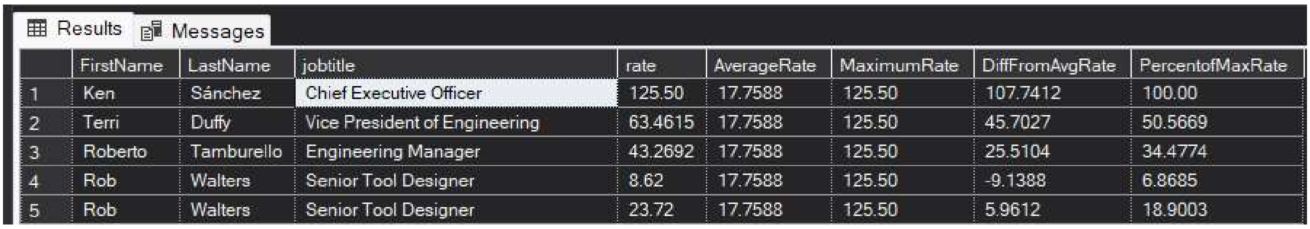


	FirstName	LastName	jobtitle	rate	AverageRate	MaximumRate	DiffFromAvgRate
1	Ken	Sánchez	Chief Executive Officer	125.50	17.7588	125.50	107.7412
2	Terri	Duffy	Vice President of Engineering	63.4615	17.7588	125.50	45.7027
3	Roberto	Tamburello	Engineering Manager	43.2692	17.7588	125.50	25.5104
4	Rob	Walters	Senior Tool Designer	8.62	17.7588	125.50	-9.1388
5	Rob	Walters	Senior Tool Designer	23.72	17.7588	125.50	5.9612
6	Rob	Walters	Senior Tool Designer	29.8462	17.7588	125.50	12.0874
7	Gail	Erickson	Design Engineer	32.6923	17.7588	125.50	14.9335
8	Jossef	Goldberg	Design Engineer	32.6923	17.7588	125.50	14.9335

Ques 4

Enhance your query from Ques 3 by adding a derived column called "PercentofMaxRate" that returns the result of the following calculation: An employees's pay rate, DIVIDED BY the maximum of all values in the "Rate" column, times 100.

```
Select *,(rate-averagerate) as DiffFromAvgRate,
(rate*100/MaximumRate)as PercentofMaxRate  -- % of maximum column added
from
(
select FirstName,LastName,jobtitle,rate,avg(rate) over() as AverageRate,
max(rate) over() as MaximumRate -- maximum rate column added
from person.Person pp
inner join HumanResources.Employee hre
on hre.BusinessEntityID = pp.BusinessEntityID
inner join HumanResources.EmployeePayHistory hrep
on hrep.BusinessEntityID = pp.BusinessEntityID
)temp
```



	FirstName	LastName	jobtitle	rate	AverageRate	MaximumRate	DiffFromAvgRate	PercentofMaxRate
1	Ken	Sánchez	Chief Executive Officer	125.50	17.7588	125.50	107.7412	100.00
2	Terri	Duffy	Vice President of Engineering	63.4615	17.7588	125.50	45.7027	50.5669
3	Roberto	Tamburello	Engineering Manager	43.2692	17.7588	125.50	25.5104	34.4774
4	Rob	Walters	Senior Tool Designer	8.62	17.7588	125.50	-9.1388	6.8685
5	Rob	Walters	Senior Tool Designer	23.72	17.7588	125.50	5.9612	18.9003

Ques 5

Create a query with the following columns:

- “Name” from the Production.Product table, which can be alised as “ProductName”
- “ListPrice” from the Production.Product table
- “Name” from the Production. ProductSubcategory table, which can be alised as “ProductSubcategory”*

- “Name” from the Production.ProductCategory table, which can be alised as “ProductCategory”**

Join Production.ProductSubcategory to Production.Product on “ProductSubcategoryID”*

Join Production.ProductCategory to ProductSubcategory on “ProductCategoryID”

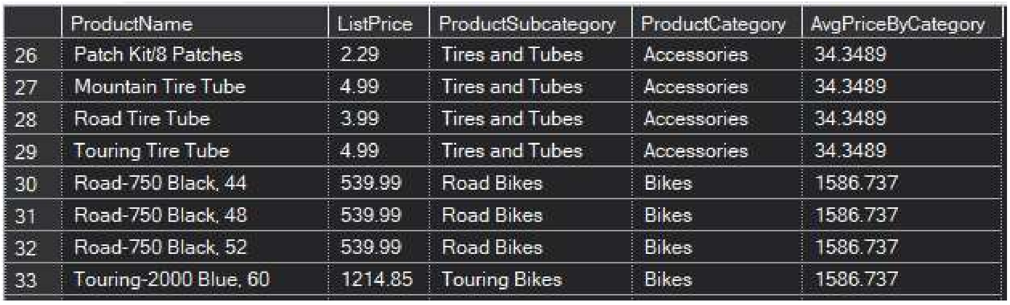
All the tables can be inner joined, and you do not need to apply any criteria.

```
select pp.name as ProductName,ListPrice,
pps.Name as ProductSubcategory,
ppc.name as ProductCategory
from Production.Product as pp
inner join Production.ProductSubcategory as pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```

Ques 6

Enhance your query from ques 5 by adding a derived column called "AvgPriceByCategory " that returns the average ListPrice *for the product category in each given row*.

```
select pp.name as ProductName,ListPrice,
pps.Name as ProductSubcategory,
ppc.name as ProductCategory,
avg(ListPrice) over(partition by ppc.name) as AvgPriceByCategory
from Production.Product as pp
inner join Production.ProductSubcategory as pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



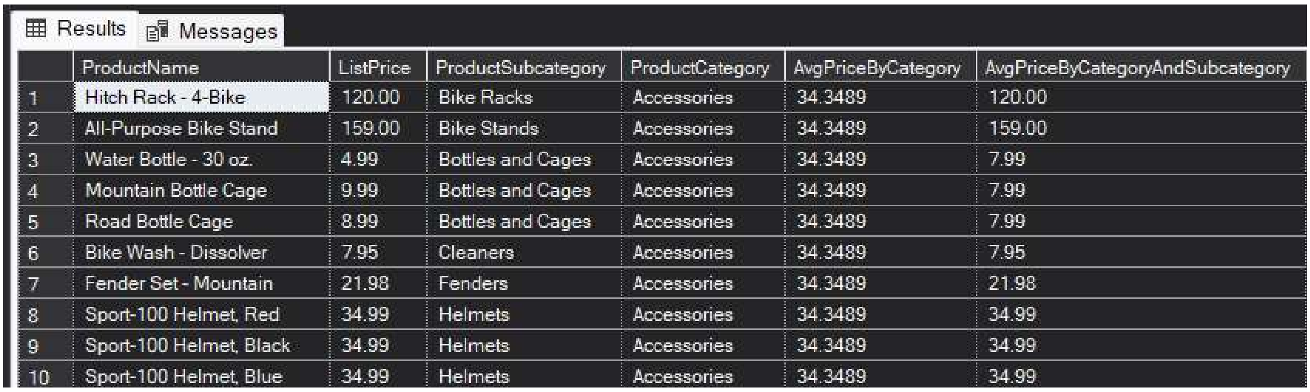
	ProductName	ListPrice	ProductSubcategory	ProductCategory	AvgPriceByCategory
26	Patch Kit/8 Patches	2.29	Tires and Tubes	Accessories	34.3489
27	Mountain Tire Tube	4.99	Tires and Tubes	Accessories	34.3489
28	Road Tire Tube	3.99	Tires and Tubes	Accessories	34.3489
29	Touring Tire Tube	4.99	Tires and Tubes	Accessories	34.3489
30	Road-750 Black, 44	539.99	Road Bikes	Bikes	1586.737
31	Road-750 Black, 48	539.99	Road Bikes	Bikes	1586.737
32	Road-750 Black, 52	539.99	Road Bikes	Bikes	1586.737
33	Touring-2000 Blue, 60	1214.85	Touring Bikes	Bikes	1586.737

Ques 7

Enhance your query from Ques 6 by adding a derived column called "AvgPriceByCategoryAndSubcategory" that returns the average ListPrice *for the product category AND subcategory in each given row.*

```
select pp.name as ProductName,ListPrice,
pps.Name as ProductSubcategory,
ppc.name as ProductCategory,
avg(ListPrice) over(partition by ppc.name) as AvgPriceByCategory,
avg(listprice) over(partition by ppc.name,pps.name)as AvgPriceByCategoryAndSubcategory
from Production.Product as pp
inner join Production.ProductSubcategory as pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
```

```
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```

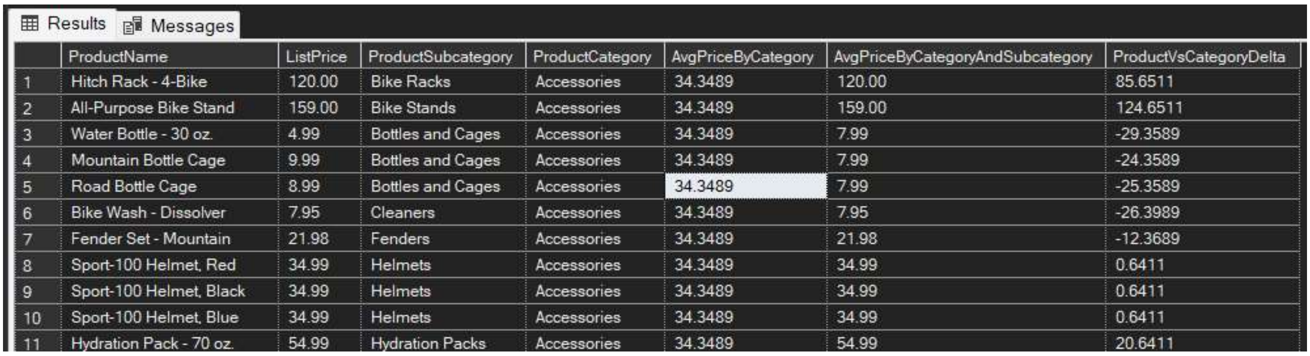


	ProductName	ListPrice	ProductSubcategory	ProductCategory	AvgPriceByCategory	AvgPriceByCategoryAndSubcategory
1	Hitch Rack - 4-Bike	120.00	Bike Racks	Accessories	34.3489	120.00
2	All-Purpose Bike Stand	159.00	Bike Stands	Accessories	34.3489	159.00
3	Water Bottle - 30 oz.	4.99	Bottles and Cages	Accessories	34.3489	7.99
4	Mountain Bottle Cage	9.99	Bottles and Cages	Accessories	34.3489	7.99
5	Road Bottle Cage	8.99	Bottles and Cages	Accessories	34.3489	7.99
6	Bike Wash - Dissolver	7.95	Cleaners	Accessories	34.3489	7.95
7	Fender Set - Mountain	21.98	Fenders	Accessories	34.3489	21.98
8	Sport-100 Helmet, Red	34.99	Helmets	Accessories	34.3489	34.99
9	Sport-100 Helmet, Black	34.99	Helmets	Accessories	34.3489	34.99
10	Sport-100 Helmet, Blue	34.99	Helmets	Accessories	34.3489	34.99

Ques 8

Enhance your query from ques 7 by adding a derived column called "ProductVsCategoryDelta" that returns the result of the following calculation: A product's list price, MINUS the average ListPrice for that product's category.

```
select pp.name as ProductName,ListPrice,
pps.Name as ProductSubcategory,
ppc.name as ProductCategory,
avg(ListPrice) over(partition by ppc.name) as AvgPriceByCategory,
avg(listprice) over(partition by ppc.name,pps.name)as AvgPriceByCategoryAndSubcategory,
(listprice-avg(ListPrice) over(partition by ppc.name)) as ProductVsCategoryDelta
from Production.Product as pp
inner join Production.ProductSubcategory as pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



	ProductName	ListPrice	ProductSubcategory	ProductCategory	AvgPriceByCategory	AvgPriceByCategoryAndSubcategory	ProductVsCategoryDelta
1	Hitch Rack - 4-Bike	120.00	Bike Racks	Accessories	34.3489	120.00	85.6511
2	All-Purpose Bike Stand	159.00	Bike Stands	Accessories	34.3489	159.00	124.6511
3	Water Bottle - 30 oz.	4.99	Bottles and Cages	Accessories	34.3489	7.99	-29.3589
4	Mountain Bottle Cage	9.99	Bottles and Cages	Accessories	34.3489	7.99	-24.3589
5	Road Bottle Cage	8.99	Bottles and Cages	Accessories	34.3489	7.99	-25.3589
6	Bike Wash - Dissolver	7.95	Cleaners	Accessories	34.3489	7.95	-26.3989
7	Fender Set - Mountain	21.98	Fenders	Accessories	34.3489	21.98	-12.3689
8	Sport-100 Helmet, Red	34.99	Helmets	Accessories	34.3489	34.99	0.6411
9	Sport-100 Helmet, Black	34.99	Helmets	Accessories	34.3489	34.99	0.6411
10	Sport-100 Helmet, Blue	34.99	Helmets	Accessories	34.3489	34.99	0.6411
11	Hydration Pack - 70 oz.	54.99	Hydration Packs	Accessories	34.3489	54.99	20.6411

Exercise: ROW_NUMBER()

Ques 1

Create a query with the following columns (feel free to borrow your code from Exercise 1 of the PARTITION BY exercises):

- “Name” from the Production.Product table, which can be alised as “ProductName”
- “ListPrice” from the Production.Product table
- “Name” from the Production. ProductSubcategory table, which can be alised as “ProductSubcategory”

- “Name” from the Production.ProductCategory table, which can be alised as “ProductCategory”

Join Production.ProductSubcategory to Production.Product on “ProductSubcategoryID”*

Join Production.ProductCategory to ProductSubcategory on “ProductCategoryID”

All the tables can be inner joined, and you do not need to apply any criteria.

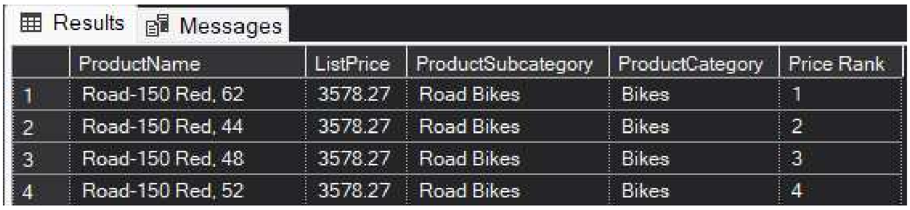
```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory
from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```

Ques 2

Enhance your query from Exercise 1 by adding a derived column called

"Price Rank " that ranks all records in the dataset by ListPrice, in descending order. That is to say, the product with the most expensive price should have a rank of 1, and the product with the least expensive price should have a rank equal to the number of records in the dataset.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank'
from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank
1	Road-150 Red, 62	3578.27	Road Bikes	Bikes	1
2	Road-150 Red, 44	3578.27	Road Bikes	Bikes	2
3	Road-150 Red, 48	3578.27	Road Bikes	Bikes	3
4	Road-150 Red, 52	3578.27	Road Bikes	Bikes	4

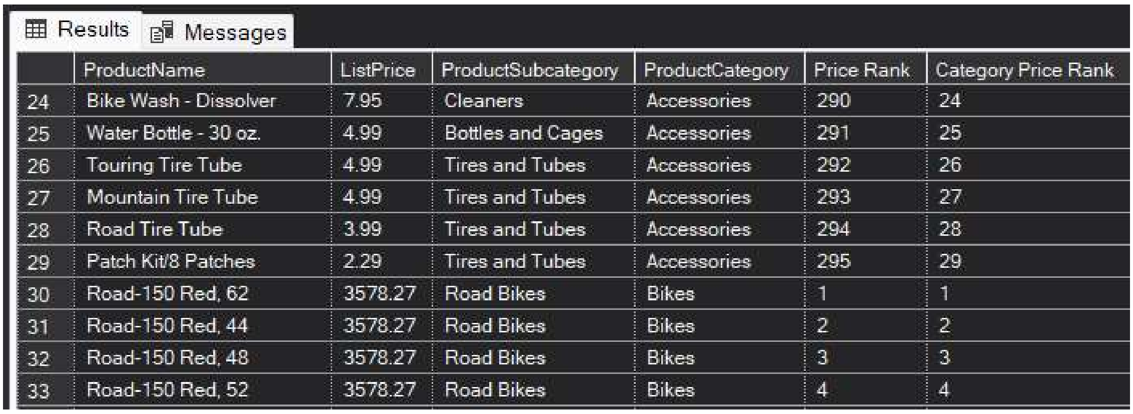
Ques 3

Enhance your query from Exercise 2 by adding a derived column called

"Category Price Rank" that ranks all products by ListPrice – *within each category* - in descending order. In other words, every product within a given category should be ranked relative to other products in the same category.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank',
row_number() over (partition by ppc.name order by listprice DESC)as 'Category Price Rank'
from Production.Product pp
inner join Production.ProductSubcategory pps
```

```
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



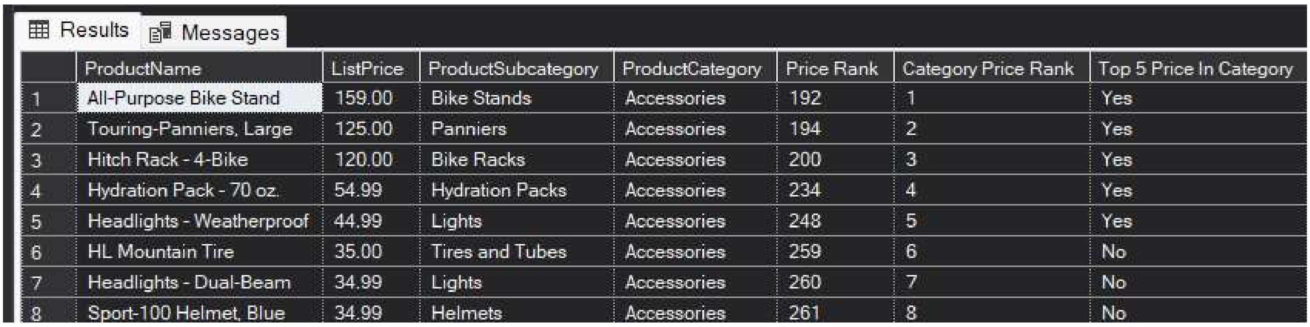
	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank	Category Price Rank
24	Bike Wash - Dissolver	7.95	Cleaners	Accessories	290	24
25	Water Bottle - 30 oz.	4.99	Bottles and Cages	Accessories	291	25
26	Touring Tire Tube	4.99	Tires and Tubes	Accessories	292	26
27	Mountain Tire Tube	4.99	Tires and Tubes	Accessories	293	27
28	Road Tire Tube	3.99	Tires and Tubes	Accessories	294	28
29	Patch Kit/8 Patches	2.29	Tires and Tubes	Accessories	295	29
30	Road-150 Red, 62	3578.27	Road Bikes	Bikes	1	1
31	Road-150 Red, 44	3578.27	Road Bikes	Bikes	2	2
32	Road-150 Red, 48	3578.27	Road Bikes	Bikes	3	3
33	Road-150 Red, 52	3578.27	Road Bikes	Bikes	4	4

Ques 4

Enhance your query from Exercise 3 by adding a derived column called

"Top 5 Price In Category" that returns the string “Yes” if a product has one of the top 5 list prices in its product category, and “No” if it does not. You can try incorporating your logic from Exercise 3 into a CASE statement to make this work.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank',
row_number() over (partition by ppc.name order by listprice DESC)as 'Category Price Rank',
case when row_number() over (partition by ppc.name order by listprice DESC) <6
then 'Yes' else 'No' end as  'Top 5 Price In Category'
from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



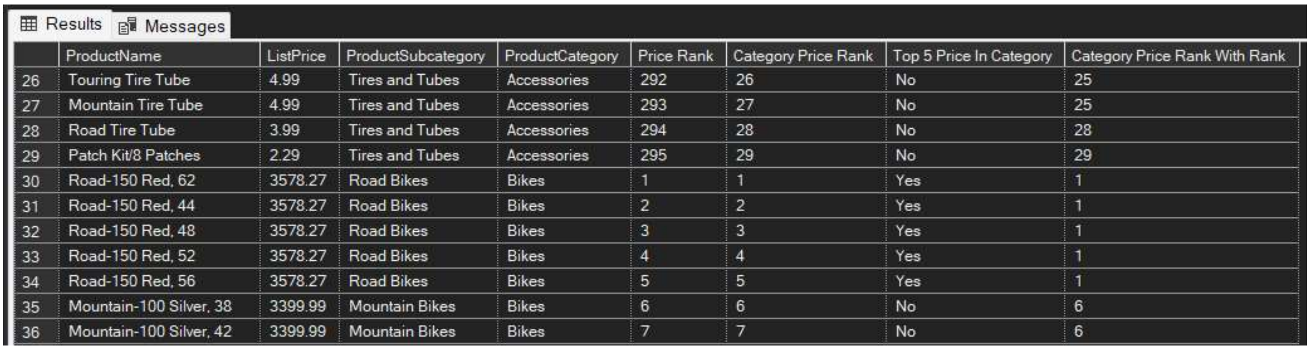
	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank	Category Price Rank	Top 5 Price In Category
1	All-Purpose Bike Stand	159.00	Bike Stands	Accessories	192	1	Yes
2	Touring-Panniers, Large	125.00	Panniers	Accessories	194	2	Yes
3	Hitch Rack - 4-Bike	120.00	Bike Racks	Accessories	200	3	Yes
4	Hydration Pack - 70 oz.	54.99	Hydration Packs	Accessories	234	4	Yes
5	Headlights - Weatherproof	44.99	Lights	Accessories	248	5	Yes
6	HL Mountain Tire	35.00	Tires and Tubes	Accessories	259	6	No
7	Headlights - Dual-Beam	34.99	Lights	Accessories	260	7	No
8	Sport-100 Helmet, Blue	34.99	Helmets	Accessories	261	8	No

Exercise: Rank and Dense_rank

Ques 1

Using your solution query to Exercise 4 from the ROW_NUMBER exercises as a staring point, add a derived column called “Category Price Rank With Rank” that uses the RANK function to rank all products by ListPrice – *within each category* - in descending order. Observe the differences between the “Category Price Rank” and “Category Price Rank With Rank” fields.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank',
row_number() over (partition by ppc.name order by listprice DESC)as 'Category Price Rank',
case when row_number() over (partition by ppc.name order by listprice DESC) <6
then 'Yes' else 'No' end as  'Top 5 Price In Category',
rank() over (partition by ppc.name order by listprice DESC) as 'Category Price Rank With Rank'
from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



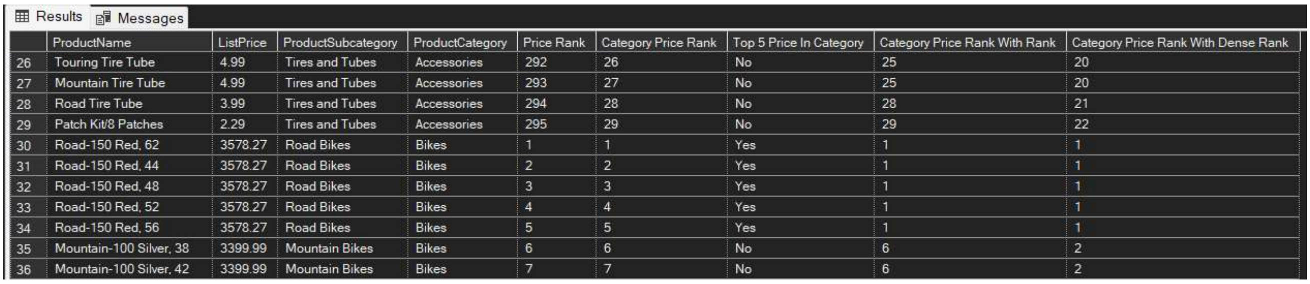
	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank	Category Price Rank	Top 5 Price In Category	Category Price Rank With Rank
26	Touring Tire Tube	4.99	Tires and Tubes	Accessories	292	26	No	25
27	Mountain Tire Tube	4.99	Tires and Tubes	Accessories	293	27	No	25
28	Road Tire Tube	3.99	Tires and Tubes	Accessories	294	28	No	28
29	Patch Kit/8 Patches	2.29	Tires and Tubes	Accessories	295	29	No	29
30	Road-150 Red, 62	3578.27	Road Bikes	Bikes	1	1	Yes	1
31	Road-150 Red, 44	3578.27	Road Bikes	Bikes	2	2	Yes	1
32	Road-150 Red, 48	3578.27	Road Bikes	Bikes	3	3	Yes	1
33	Road-150 Red, 52	3578.27	Road Bikes	Bikes	4	4	Yes	1
34	Road-150 Red, 56	3578.27	Road Bikes	Bikes	5	5	Yes	1
35	Mountain-100 Silver, 38	3399.99	Mountain Bikes	Bikes	6	6	No	6
36	Mountain-100 Silver, 42	3399.99	Mountain Bikes	Bikes	7	7	No	6

Ques 2

Modify your query from Exercise 2 by adding a derived column called "Category Price Rank With Dense Rank" that that uses the DENSE_RANK function to rank all products by ListPrice – *within each category* - in descending order. Observe the differences among the “Category Price Rank”, “Category Price Rank With Rank”, and “Category Price Rank With Dense Rank” fields.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank',
row_number() over (partition by ppc.name order by listprice DESC)as 'Category Price Rank',
case when row_number() over (partition by ppc.name order by listprice DESC) <6
then 'Yes' else 'No' end as  'Top 5 Price In Category',
rank() over (partition by ppc.name order by listprice DESC) as 'Category Price Rank With Rank',
dense_rank() over (partition by ppc.name order by listprice DESC) as 'Category Price Rank With Dense
Rank'

from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



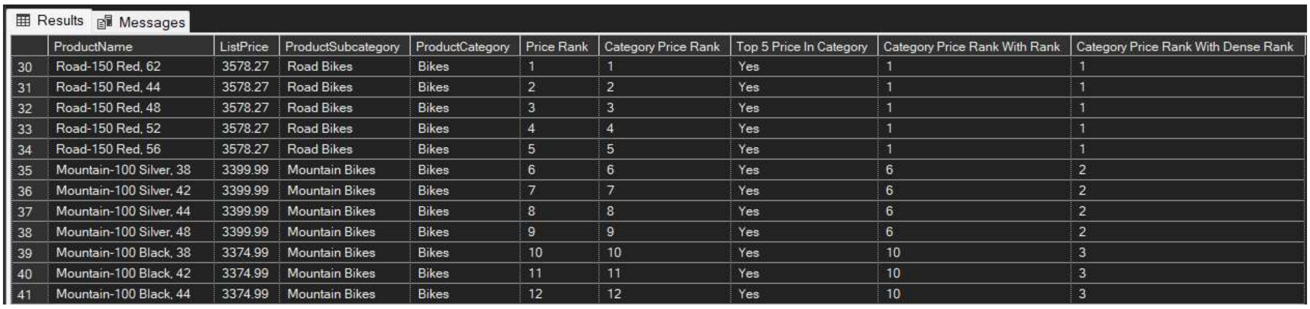
	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank	Category Price Rank	Top 5 Price In Category	Category Price Rank With Rank	Category Price Rank With Dense Rank
26	Touring Tire Tube	4.99	Tires and Tubes	Accessories	292	26	No	25	20
27	Mountain Tire Tube	4.99	Tires and Tubes	Accessories	293	27	No	25	20
28	Road Tire Tube	3.99	Tires and Tubes	Accessories	294	28	No	28	21
29	Patch Kit/8 Patches	2.29	Tires and Tubes	Accessories	295	29	No	29	22
30	Road-150 Red, 62	3578.27	Road Bikes	Bikes	1	1	Yes	1	1
31	Road-150 Red, 44	3578.27	Road Bikes	Bikes	2	2	Yes	1	1
32	Road-150 Red, 48	3578.27	Road Bikes	Bikes	3	3	Yes	1	1
33	Road-150 Red, 52	3578.27	Road Bikes	Bikes	4	4	Yes	1	1
34	Road-150 Red, 56	3578.27	Road Bikes	Bikes	5	5	Yes	1	1
35	Mountain-100 Silver, 38	3399.99	Mountain Bikes	Bikes	6	6	No	6	2
36	Mountain-100 Silver, 42	3399.99	Mountain Bikes	Bikes	7	7	No	6	2

Ques 3

Examine the code you wrote to define the “Top 5 Price In Category” field back in the ROW_NUMBER exercises. Now that you understand the differences among ROW_NUMBER, RANK, and DENSE_RANK, consider which of these functions would be most appropriate to return a true top 5 products by price, assuming we want to see the top 5 **distinct** prices AND we want “ties” (by price) to all share the same rank.

```
select pp.name as ProductName,pp.ListPrice,
pps.name as ProductSubcategory,
ppc.name as ProductCategory,
row_number() over(order by listprice DESC) as 'Price Rank',
row_number() over (partition by ppc.name order by listprice DESC)as 'Category Price Rank',
case when dense_rank() over (partition by ppc.name order by listprice DESC) <6
then 'Yes' else 'No' end as  'Top 5 Price In Category',
rank() over (partition by ppc.name order by listprice DESC) as 'Category Price Rank With Rank',
dense_rank() over (partition by ppc.name order by listprice DESC) as 'Category Price Rank With Dense
Rank'

from Production.Product pp
inner join Production.ProductSubcategory pps
on pp.ProductSubcategoryID = pps.ProductSubcategoryID
inner join Production.ProductCategory ppc
on ppc.ProductCategoryID = pps.ProductCategoryID
```



	ProductName	ListPrice	ProductSubcategory	ProductCategory	Price Rank	Category Price Rank	Top 5 Price In Category	Category Price Rank With Rank	Category Price Rank With Dense Rank
30	Road-150 Red, 62	3578.27	Road Bikes	Bikes	1	1	Yes	1	1
31	Road-150 Red, 44	3578.27	Road Bikes	Bikes	2	2	Yes	1	1
32	Road-150 Red, 48	3578.27	Road Bikes	Bikes	3	3	Yes	1	1
33	Road-150 Red, 52	3578.27	Road Bikes	Bikes	4	4	Yes	1	1
34	Road-150 Red, 56	3578.27	Road Bikes	Bikes	5	5	Yes	1	1
35	Mountain-100 Silver, 38	3399.99	Mountain Bikes	Bikes	6	6	Yes	6	2
36	Mountain-100 Silver, 42	3399.99	Mountain Bikes	Bikes	7	7	Yes	6	2
37	Mountain-100 Silver, 44	3399.99	Mountain Bikes	Bikes	8	8	Yes	6	2
38	Mountain-100 Silver, 48	3399.99	Mountain Bikes	Bikes	9	9	Yes	6	2
39	Mountain-100 Black, 38	3374.99	Mountain Bikes	Bikes	10	10	Yes	10	3
40	Mountain-100 Black, 42	3374.99	Mountain Bikes	Bikes	11	11	Yes	10	3
41	Mountain-100 Black, 44	3374.99	Mountain Bikes	Bikes	12	12	Yes	10	3

Exercise: lead and lag

Ques 1

Create a query with the following columns:

- “PurchaseOrderID” from the Purchasing.PurchaseOrderHeader table
- “OrderDate” from the Purchasing.PurchaseOrderHeader table
- “TotalDue” from the Purchasing.PurchaseOrderHeader table
- “Name” from the Purchasing.Vendor table, which can be aliased as “VendorName”*****

**Join Purchasing.Vendor to Purchasing.PurchaseOrderHeader on BusinessEntityID = VendorID*

Apply the following criteria to the query:

- Order must have taken place on or after 2013
- TotalDue must be greater than $500

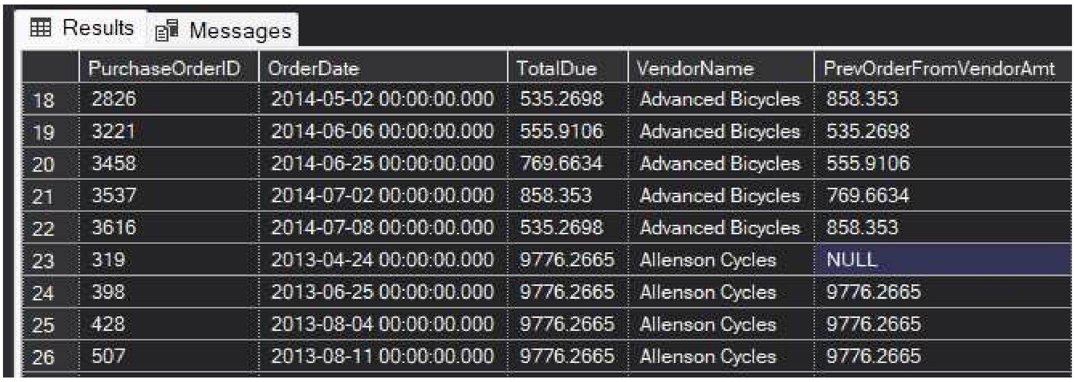
```
select
    PurchaseOrderID,
    OrderDate,TotalDue,
    pv.Name as VendorName
from
    Purchasing.PurchaseOrderHeader pph
inner join
    Purchasing.Vendor pv
    on pv.BusinessEntityID = pph.VendorID
where
    TotalDue > 500 and OrderDate >= '2013-01-01'
```

Ques 2

Modify your query from Exercise 1 by adding a derived column called

"PrevOrderFromVendorAmt", that returns the “previous” TotalDue value (relative to the current row) *within the group of all orders with the same vendor ID.* We are defining “previous” based on order date.

```
select
    PurchaseOrderID,
    OrderDate,
    TotalDue,
    pv.Name as VendorName,
    lag(TotalDue,1) over(partition by pv.name order by orderdate ASC) as PrevOrderFromVendorAmt
from
    Purchasing.PurchaseOrderHeader pph
inner join
    Purchasing.Vendor pv
    on pv.BusinessEntityID = pph.VendorID
where
    TotalDue > 500 and OrderDate >= '2013-01-01'
order by
    pv.name,orderdate ASC
```



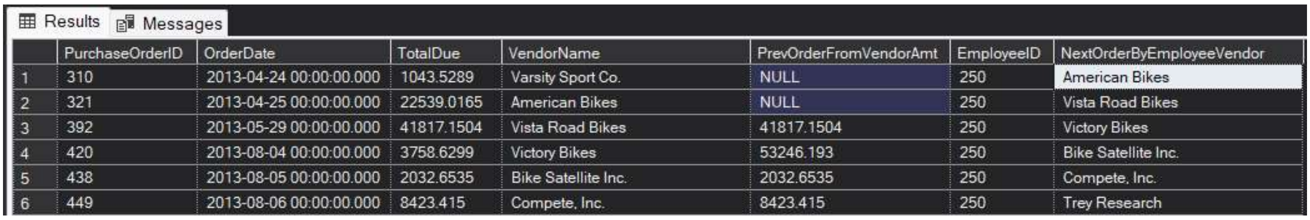
	PurchaseOrderID	OrderDate	TotalDue	VendorName	PrevOrderFromVendorAmt
18	2826	2014-05-02 00:00:00.000	535.2698	Advanced Bicycles	858.353
19	3221	2014-06-06 00:00:00.000	555.9106	Advanced Bicycles	535.2698
20	3458	2014-06-25 00:00:00.000	769.6634	Advanced Bicycles	555.9106
21	3537	2014-07-02 00:00:00.000	858.353	Advanced Bicycles	769.6634
22	3616	2014-07-08 00:00:00.000	535.2698	Advanced Bicycles	858.353
23	319	2013-04-24 00:00:00.000	9776.2665	Allenson Cycles	NULL
24	398	2013-06-25 00:00:00.000	9776.2665	Allenson Cycles	9776.2665
25	428	2013-08-04 00:00:00.000	9776.2665	Allenson Cycles	9776.2665
26	507	2013-08-11 00:00:00.000	9776.2665	Allenson Cycles	9776.2665

Ques 3

Modify your query from Exercise 2 by adding a derived column called

"NextOrderByEmployeeVendor", that returns the “next” vendor name (the “name” field from Purchasing.Vendor) *within the group of all orders that have the same EmployeeID value in* Purchasing.PurchaseOrderHeader*.* Similar to the last exercise, we are defining “next” based on order date.

```
select
    PurchaseOrderID,
    OrderDate,
    TotalDue,
    pv.Name as VendorName,
    lag(TotalDue,1) over(partition by pv.name order by orderdate ASC) as PrevOrderFromVendorAmt,
    EmployeeID,
    lead(pv.name,1) over(partition by employeeid order by orderdate ASC)as NextOrderByEmployeeVendor
from
    Purchasing.PurchaseOrderHeader pph
inner join
    Purchasing.Vendor pv
    on pv.BusinessEntityID = pph.VendorID
where
    TotalDue > 500 and year(OrderDate) >= 2013
order by
    EmployeeID,orderdate ASC
```

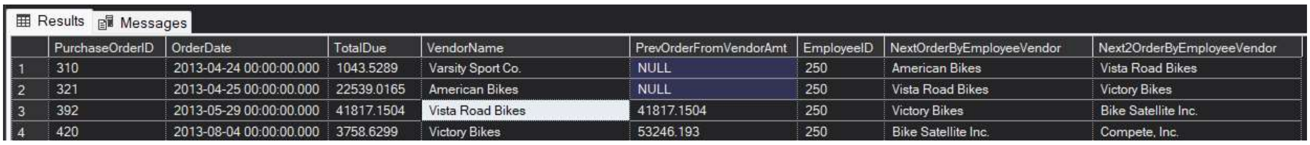


	PurchaseOrderID	OrderDate	TotalDue	VendorName	PrevOrderFromVendorAmt	EmployeeID	NextOrderByEmployeeVendor
1	310	2013-04-24 00:00:00.000	1043.5289	Varsity Sport Co.	NULL	250	American Bikes
2	321	2013-04-25 00:00:00.000	22539.0165	American Bikes	NULL	250	Vista Road Bikes
3	392	2013-05-29 00:00:00.000	41817.1504	Vista Road Bikes	41817.1504	250	Victory Bikes
4	420	2013-08-04 00:00:00.000	3758.6299	Victory Bikes	53246.193	250	Bike Satellite Inc.
5	438	2013-08-05 00:00:00.000	2032.6535	Bike Satellite Inc.	2032.6535	250	Compete, Inc.
6	449	2013-08-06 00:00:00.000	8423.415	Compete, Inc.	8423.415	250	Trey Research

Ques 4

Modify your query from Exercise 3 by adding a derived column called "Next2OrderByEmployeeVendor" that returns, within the group of all orders that have the same EmployeeID*, the vendor name offset TWO orders into the “future” relative to the order in the current row.* The code should be very similar to Exercise 3, but with an extra argument passed to the Window Function used.

```
select
    PurchaseOrderID,
    OrderDate,
    TotalDue,
    pv.Name as VendorName,
    lag(TotalDue,1) over(partition by pv.name order by orderdate ASC) as PrevOrderFromVendorAmt,
    EmployeeID,
    lead(pv.name,1) over(partition by employeeid order by orderdate ASC)as NextOrderByEmployeeVendor,
    lead(pv.name,2) over(partition by employeeid order by orderdate ASC)as Next2OrderByEmployeeVendor
from
    Purchasing.PurchaseOrderHeader pph
inner join
    Purchasing.Vendor pv
    on pv.BusinessEntityID = pph.VendorID
where
    TotalDue > 500 and year(OrderDate) >= 2013
order by
    EmployeeID,orderdate ASC
```



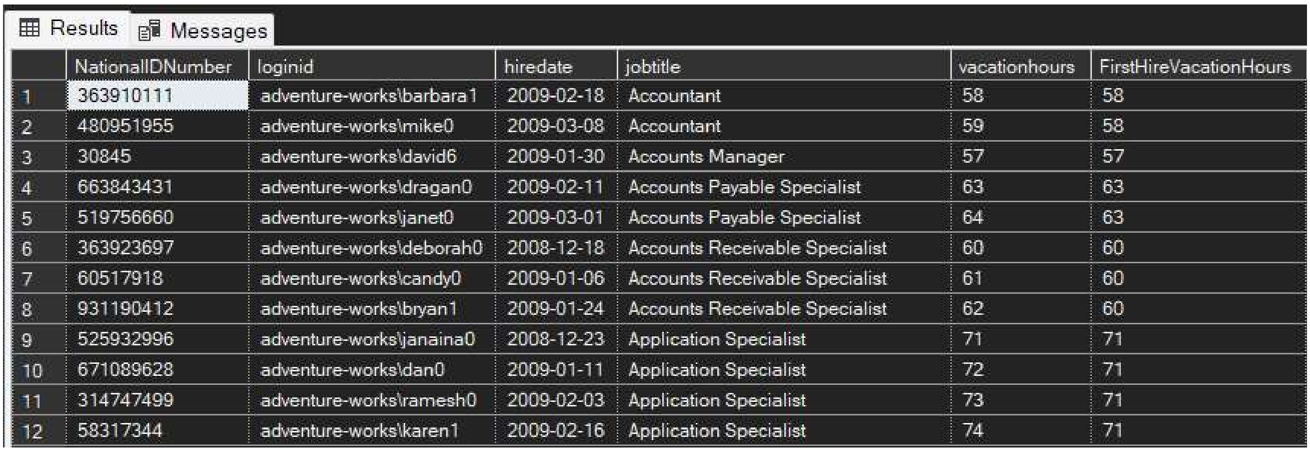
	PurchaseOrderID	OrderDate	TotalDue	VendorName	PrevOrderFromVendorAmt	EmployeeID	NextOrderByEmployeeVendor	Next2OrderByEmployeeVendor
1	310	2013-04-24 00:00:00.000	1043.5289	Varsity Sport Co.	NULL	250	American Bikes	Vista Road Bikes
2	321	2013-04-25 00:00:00.000	22539.0165	American Bikes	NULL	250	Vista Road Bikes	Victory Bikes
3	392	2013-05-29 00:00:00.000	41817.1504	Vista Road Bikes	41817.1504	250	Victory Bikes	Bike Satellite Inc.
4	420	2013-08-04 00:00:00.000	3758.6299	Victory Bikes	53246.193	250	Bike Satellite Inc.	Compete, Inc.

Exercise: First_Value()

Ques 1

- Create a query that returns all records - and the following columns - from the **HumanResources.Employee** table:
    - To make the effect of subsequent steps clearer, also sort the query output by "JobTitle" and HireDate, both in ascending order.
- Now add a derived column called "FirstHireVacationHours" that displays – for a given job title – the amount of vacation hours possessed by the *first* employee hired who has that same job title. For example, if 5 employees have the title "Data Guru", and the one of those 5 with the oldest hire date has 99 vacation hours, "FirstHireVacationHours" should display "99" for all 5 of those employees' corresponding records in the query.

```
-- I am not selecting everthing just for the sake of Screenshot
select
    NationalIDNumber,
    loginid,
    hiredate,
    jobtitle,
    vacationhours,
    first_value(VacationHours) over(partition by jobtitle order by hiredate ASC)as
FirstHireVacationHours
from HumanResources.Employee
order by JobTitle,HireDate ASC
```



	NationalIDNumber	loginid	hiredate	jobtitle	vacationhours	FirstHireVacationHours
1	363910111	adventure-works\barbara1	2009-02-18	Accountant	58	58
2	480951955	adventure-works\mike0	2009-03-08	Accountant	59	58
3	30845	adventure-works\david6	2009-01-30	Accounts Manager	57	57
4	663843431	adventure-works\dragan0	2009-02-11	Accounts Payable Specialist	63	63
5	519756660	adventure-works\janet0	2009-03-01	Accounts Payable Specialist	64	63
6	363923697	adventure-works\deborah0	2008-12-18	Accounts Receivable Specialist	60	60
7	60517918	adventure-works\candy0	2009-01-06	Accounts Receivable Specialist	61	60
8	931190412	adventure-works\bryan1	2009-01-24	Accounts Receivable Specialist	62	60
9	525932996	adventure-works\janaina0	2008-12-23	Application Specialist	71	71
10	671089628	adventure-works\dan0	2009-01-11	Application Specialist	72	71
11	314747499	adventure-works\ramesh0	2009-02-03	Application Specialist	73	71
12	58317344	adventure-works\karen1	2009-02-16	Application Specialist	74	71

Ques 2

- Create a query with the following columns:

  a. "ProductID" from the **Production.Product** table

  b. "Name" from the **Production.Product** table (alias this as "ProductName")

  c. "ListPrice" from the **Production.ProductListPriceHistory** table

d. “ModifiedDate” from the **Production.ProductListPriceHistory**

You can join the **Production.Product** and **Production.ProductListPriceHistory** tables on "ProductID".

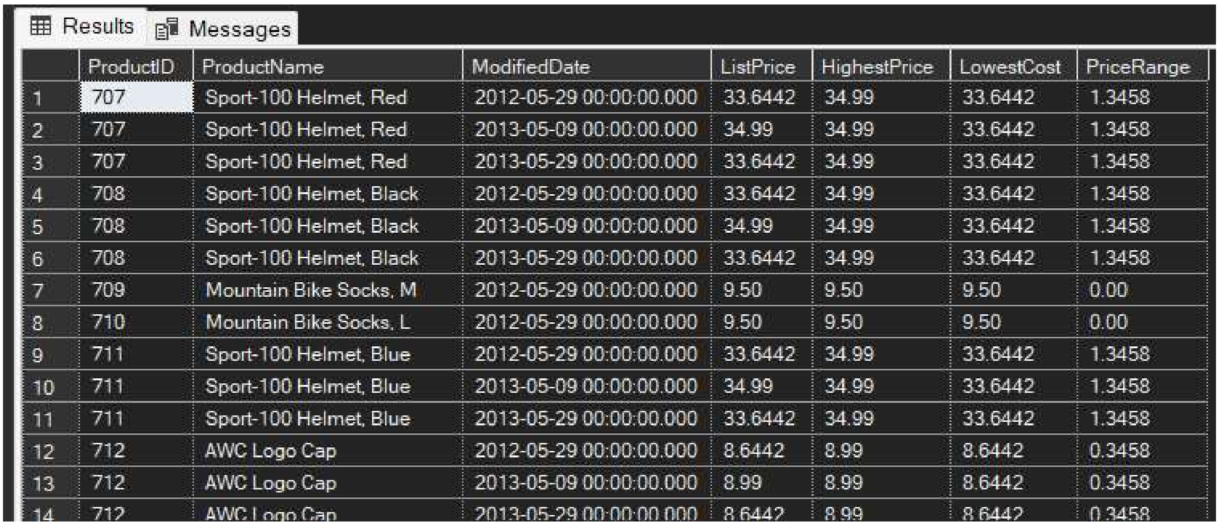
- i. Now add a derived column called “HighestPrice” that displays – for a given product – the highest price that product has been listed at. So even if there are 4 records for a given product, this column should only display the all-time highest list price for that product in each of those 4 rows.

```
select
    pp.ProductID,
    pp.name as ProductName,
    pplph.ListPrice,
    pplph.ModifiedDate,
    max(pplph.ListPrice) over(partition by pp.name)as HighestPrice
from Production.Product pp
inner join Production.ProductListPriceHistory pplph
on pp.ProductID = pplph.ProductID
order by pp.ProductID,ModifiedDate ASC
```

- Similarly, create another derived column called “LowestCost” that displays the all-time lowest price for a given product.

- Finally, create a third derived column called “PriceRange” that reflects, for a given product, the difference between its highest and lowest ever list prices.

```
select
    pp.ProductID,
    pp.name as ProductName,
    pplph.ModifiedDate,
    pplph.ListPrice,
    max(pplph.ListPrice) over(partition by pp.name)as HighestPrice,
    min(pplph.ListPrice) over(partition by pp.name)as LowestCost,
    MAX(pplph.ListPrice) OVER (PARTITION BY pp.ProductID) - MIN(pplph.ListPrice) OVER (PARTITION BY
pp.ProductID) AS PriceRange
from Production.Product pp
inner join Production.ProductListPriceHistory pplph
on pp.ProductID = pplph.ProductID
order by pp.ProductID,ModifiedDate ASC
```



	ProductID	ProductName	ModifiedDate	ListPrice	HighestPrice	LowestCost	PriceRange
1	707	Sport-100 Helmet, Red	2012-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
2	707	Sport-100 Helmet, Red	2013-05-09 00:00:00.000	34.99	34.99	33.6442	1.3458
3	707	Sport-100 Helmet, Red	2013-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
4	708	Sport-100 Helmet, Black	2012-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
5	708	Sport-100 Helmet, Black	2013-05-09 00:00:00.000	34.99	34.99	33.6442	1.3458
6	708	Sport-100 Helmet, Black	2013-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
7	709	Mountain Bike Socks, M	2012-05-29 00:00:00.000	9.50	9.50	9.50	0.00
8	710	Mountain Bike Socks, L	2012-05-29 00:00:00.000	9.50	9.50	9.50	0.00
9	711	Sport-100 Helmet, Blue	2012-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
10	711	Sport-100 Helmet, Blue	2013-05-09 00:00:00.000	34.99	34.99	33.6442	1.3458
11	711	Sport-100 Helmet, Blue	2013-05-29 00:00:00.000	33.6442	34.99	33.6442	1.3458
12	712	AWC Logo Cap	2012-05-29 00:00:00.000	8.6442	8.99	8.6442	0.3458
13	712	AWC Logo Cap	2013-05-09 00:00:00.000	8.99	8.99	8.6442	0.3458
14	712	AWC Logo Cap	2013-05-29 00:00:00.000	8.6442	8.99	8.6442	0.3458

Exercise: Subqueries

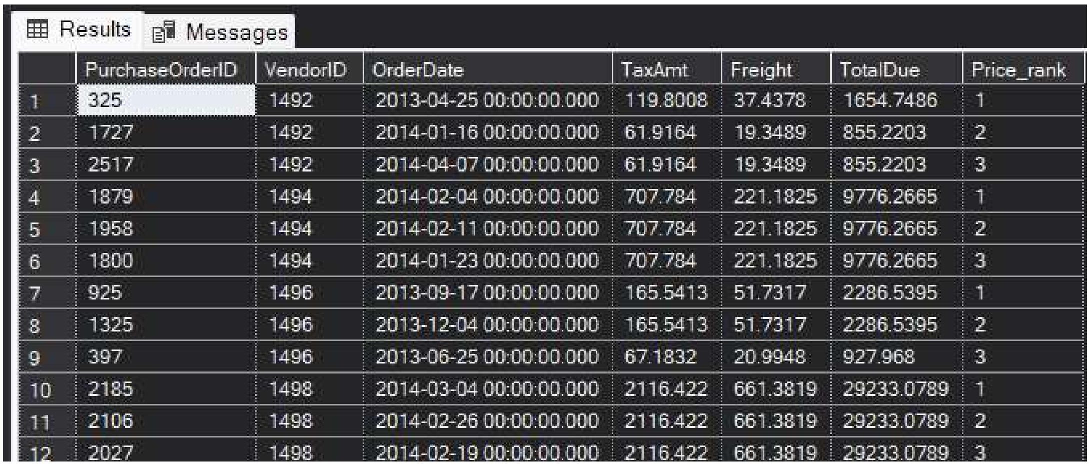
Ques 1

Write a query that displays the three most expensive orders, **per vendor ID**, from the Purchasing.PurchaseOrderHeader table. There should ONLY be three records per Vendor ID, even if some of the total amounts due are identical. "Most expensive" is defined by the amount in the "TotalDue" field.

Include the following fields in your output:

- PurchaseOrderID
- VendorID
- OrderDate
- TaxAmt
- Freight
- TotalDue

```
Select *
from
(
select
    PurchaseOrderID,
    VendorID,
    OrderDate,
    TaxAmt,
    Freight,
    TotalDue,
    ROW_NUMBER() over(partition by vendorid order by totaldue DESC) as Price_rank
from
    Purchasing.PurchaseOrderHeader ppoh
)temp
where Price_rank <4
order by vendorid ASC,TotalDue DESC
```



	PurchaseOrderID	VendorID	OrderDate	TaxAmt	Freight	TotalDue	Price_rank
1	325	1492	2013-04-25 00:00:00.000	119.8008	37.4378	1654.7486	1
2	1727	1492	2014-01-16 00:00:00.000	61.9164	19.3489	855.2203	2
3	2517	1492	2014-04-07 00:00:00.000	61.9164	19.3489	855.2203	3
4	1879	1494	2014-02-04 00:00:00.000	707.784	221.1825	9776.2665	1
5	1958	1494	2014-02-11 00:00:00.000	707.784	221.1825	9776.2665	2
6	1800	1494	2014-01-23 00:00:00.000	707.784	221.1825	9776.2665	3
7	925	1496	2013-09-17 00:00:00.000	165.5413	51.7317	2286.5395	1
8	1325	1496	2013-12-04 00:00:00.000	165.5413	51.7317	2286.5395	2
9	397	1496	2013-06-25 00:00:00.000	67.1832	20.9948	927.968	3
10	2185	1498	2014-03-04 00:00:00.000	2116.422	661.3819	29233.0789	1
11	2106	1498	2014-02-26 00:00:00.000	2116.422	661.3819	29233.0789	2
12	2027	1498	2014-02-19 00:00:00.000	2116.422	661.3819	29233.0789	3

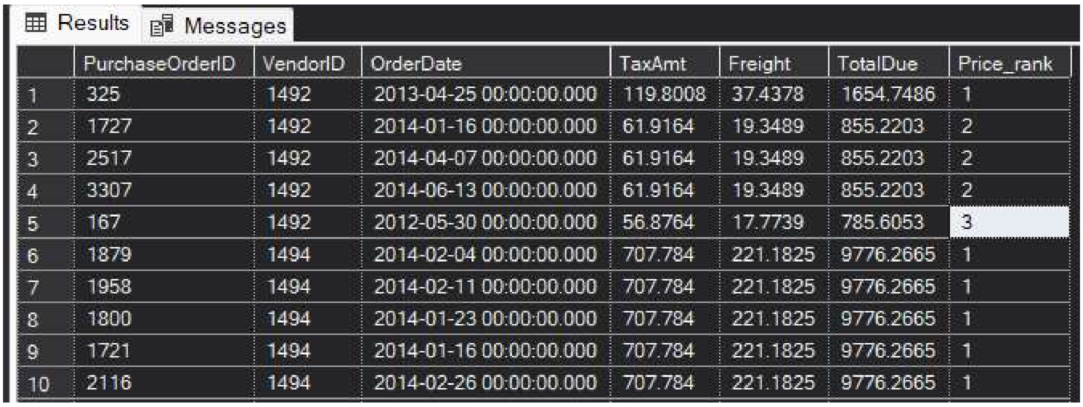
Ques 2

Modify your query from the first problem, such that the top three purchase order **amounts** are returned, regardless of how many records are returned per Vendor Id.

In other words, if there are multiple orders with the same total due amount, all should be returned as long as the total due amount for these orders is one of the top three.

Ultimately, you should see three distinct total due amounts (i.e., the top three) for each group of like Vendor Ids. However, there could be multiple records for each of these amounts.

```
Select *
from
(
select
    PurchaseOrderID,
    VendorID,
    OrderDate,
    TaxAmt,
    Freight,
    TotalDue,
    dense_rank() over(partition by vendorid order by totaldue DESC) as Price_rank
from
    Purchasing.PurchaseOrderHeader ppoh
)temp
where Price_rank <4
order by vendorid ASC,TotalDue DESC
```



	PurchaseOrderID	VendorID	OrderDate	TaxAmt	Freight	TotalDue	Price_rank
1	325	1492	2013-04-25 00:00:00.000	119.8008	37.4378	1654.7486	1
2	1727	1492	2014-01-16 00:00:00.000	61.9164	19.3489	855.2203	2
3	2517	1492	2014-04-07 00:00:00.000	61.9164	19.3489	855.2203	2
4	3307	1492	2014-06-13 00:00:00.000	61.9164	19.3489	855.2203	2
5	167	1492	2012-05-30 00:00:00.000	56.8764	17.7739	785.6053	3
6	1879	1494	2014-02-04 00:00:00.000	707.784	221.1825	9776.2665	1
7	1958	1494	2014-02-11 00:00:00.000	707.784	221.1825	9776.2665	1
8	1800	1494	2014-01-23 00:00:00.000	707.784	221.1825	9776.2665	1
9	1721	1494	2014-01-16 00:00:00.000	707.784	221.1825	9776.2665	1
10	2116	1494	2014-02-26 00:00:00.000	707.784	221.1825	9776.2665	1

ROW_BETWEEN

Ques 1

Create a query with the following columns:

- “OrderMonth”, a *derived* column (you’ll have to create this one yourself) featuring the month **number** corresponding with the Order *Date* in a given row

- “OrderYear”, a derived column featuring the **year** corresponding with the Order Date in a given row

- “SubTotal” from the **Purchasing.PurchaseOrderHeader** table

Your query should be an *aggregate* query – specifically, it should **sum** “SubTotal”, and **group by** the remaining fields

```
select
        orderdate,
        YEAR(orderdate) as OrderYear,
        MONTH(OrderDate) as OrderMonth,
        SubTotal,
        SUM(SubTotal) OVER (PARTITION BY YEAR(OrderDate), MONTH(OrderDate)) AS TotalSubTotal
from
        Purchasing.PurchaseOrderHeader ppoh
order by
        YEAR(orderdate),MONTH(OrderDate) ASC
```

Ques 2

Modify your query from Exercise 1 by adding a derived column called "Rolling3MonthTotal", that displays - for a given row - a running total of “SubTotal” for the prior three months (including the current row).

Ques 3

Modify your query from Exercise 3 by adding another derived column called "MovingAvg6Month", that calculates a rolling average of “SubTotal” for the previous 6 months, relative to the month in the “current” row. Note that this average should NOT include the current row.

Ques 4

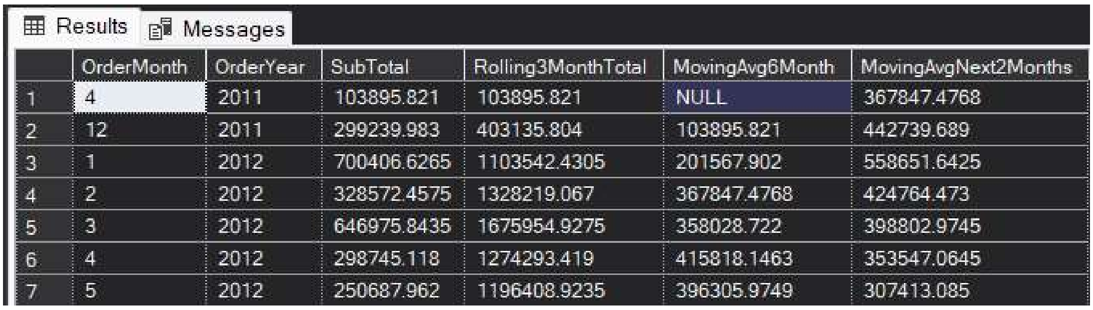
Modify your query from Exercise 3 by adding (yet) another derived column called “MovingAvgNext2Months” , that calculates a rolling average of “SubTotal” for the month in the current row **and** the next two months after that. This moving average will provide a kind of "forecast" for Subtotal by month.

```
SELECT
        OrderMonth,
        OrderYear,
        SubTotal,
        Rolling3MonthTotal = SUM(SubTotal) OVER(ORDER BY OrderYear, OrderMonth ROWS BETWEEN 2 PRECEDING
AND CURRENT ROW),
        MovingAvg6Month = AVG(SubTotal) OVER(ORDER BY OrderYear, OrderMonth ROWS BETWEEN 6 PRECEDING AND
1 PRECEDING),
        MovingAvgNext2Months = AVG(SubTotal) OVER(ORDER BY OrderYear, OrderMonth ROWS BETWEEN CURRENT
ROW AND 2 FOLLOWING)

FROM (
        SELECT
                OrderMonth = MONTH(OrderDate),
                OrderYear = YEAR(OrderDate),
                SubTotal = SUM(SubTotal)

        FROM Purchasing.PurchaseOrderHeader

        GROUP BY
                MONTH(OrderDate),
                YEAR(OrderDate)
) X
```



	OrderMonth	OrderYear	SubTotal	Rolling3MonthTotal	MovingAvg6Month	MovingAvgNext2Months
1	4	2011	103895.821	103895.821	NULL	367847.4768
2	12	2011	299239.983	403135.804	103895.821	442739.689
3	1	2012	700406.6265	1103542.4305	201567.902	558651.6425
4	2	2012	328572.4575	1328219.067	367847.4768	424764.473
5	3	2012	646975.8435	1675954.9275	358028.722	398802.9745
6	4	2012	298745.118	1274293.419	415818.1463	353547.0645
7	5	2012	250687.962	1196408.9235	396305.9749	307413.085