

Benjamin (Ben) Chen

(Bech1695)

Took 8 hours to complete this Lab #3, originally took 4, but Professor Knox advised some changes

Takeaways from OH:

- 1 correct answer set per problem (which is if it follows the criteria), but many different ways to write the SQL to get there
- “Clean code” just means not something overly complicated and not properly formatted
- Professor will not be grading this one
- 10 employees in the employee table is normal (even if instruction PDF has 9), Prof. Knox made changes since last semester

1. List in alphabetical order without duplicates the Title of Courtesy for employees. (5)

The screenshot shows the MySQL Workbench interface. On the left, the 'Navigator' pane displays the 'SCHEMAS' tree with 'bech1695' and 'Northwind' databases. Under 'Northwind', the 'Tables' folder is expanded, showing various tables including 'Employees'. The main 'Query Editor' pane contains the following SQL query:

```
1 • SELECT DISTINCT TitleOfCourtesy
2 FROM Employees
3 ORDER BY TitleOfCourtesy;
4
```

Below the query editor, the 'Result Grid' is displayed, showing the results of the query:

TitleOfCourtesy
Dr.
Mr.
Mrs.
Ms.
Prime Time

At the bottom, the 'Output' pane shows the 'Action Output' table with the following row:

#	Time	Action	Message	Duration / Fetch
1	00:17:58	SELECT DISTINCT TitleOfCourtesy FROM Em...	5 row(s) returned	0.032 sec / 0.000 sec

SELECT DISTINCT TitleOfCourtesy

- Ensures that there are no duplicate titles.
- Without this, it will repeat a bit of “Mr.” and “Mrs.”, etc.

- I tested this by trying no "DISTINCT"

FROM Employees

- Retrieves the titles from the Employees table.

ORDER BY TitleOfCourtesy

- Sorts the results alphabetically.

This query should return 5 unique TitleOfCourtesy values in alphabetical order

2. List only the name of the company(s) that reside in London and a field called Contact that contains the name and title of the contact person for that company. (6)

Query 1 x

SQLAdditions

My Snippets

```

1 • SELECT CompanyName,
2     CONCAT(ContactName, ' - ', ContactTitle) AS Contact
3 FROM Customers
4 WHERE City = 'London';

```

Result Grid

CompanyName	Contact
Around the Horn	Thomas Hardy - Sales Representative
B's Beverages	Victoria Ashworth - Sales Representative
Consolidated Holdings	Elizabeth Brown - Sales Representative
Eastern Connection	Ann Devon - Sales Agent
North/South	Simon Crowther - Sales Associate
Seven Seas Imports	Hari Kumar - Sales Manager

Result 10 x

Read Only Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	22:45:35	SELECT CONCAT(FirstName, ' ', LastName) AS Name, ...	9 row(s) returned	0.031 sec / 0.000 sec
✓ 2	23:07:26	SELECT OrderID, OrderDate, RequiredDate, ShippedD...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 3	23:20:22	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 4	23:23:16	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 5	23:38:43	SELECT OrderID, OrderDate, RequiredDate, ...	37 row(s) returned	0.047 sec / 0.000 sec
✓ 6	23:39:08	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 7	23:46:11	SELECT CONCAT(Employees.FirstName, ' ', Employees....	1 row(s) returned	0.047 sec / 0.000 sec
✓ 8	23:53:55	SELECT CONCAT(FirstName, ' ', LastName) AS Name, ...	3 row(s) returned	0.031 sec / 0.000 sec
✓ 9	23:58:11	SELECT ProductID, ProductName, UnitPrice, UnitsInSt...	4 row(s) returned	0.032 sec / 0.000 sec
✓ 10	00:01:52	SELECT CompanyName, CONCAT(ContactName, ...	6 row(s) returned	0.047 sec / 0.000 sec

SELECT CompanyName

- Retrieves company names

CONCAT(ContactName, ' - ', ContactTitle) AS Contact

- Adds together the person's name and their employee title into a single field named "Contact".
- Added the dash mark to show clear separation between title and name, just to make the rows easier to read

FROM Customers

- Accesses the data from the Customers table.

WHERE City = 'London'

- Filters results to only include companies located in London.

This should return 6 companies in London along with their contact person's name and title

3. List the ProductID, ProductName, UnitPrice, and UnitsInStock for all products that have a value between 19 and 19.5 for the price. Make sure results are sorted by the price. (4)

The screenshot shows the SQL Server Enterprise Manager interface. The 'Query 1' window displays the following SQL query:

```

1 • SELECT ProductID, ProductName, UnitPrice, UnitsInStock
2 FROM Products
3 WHERE UnitPrice BETWEEN 19 AND 19.5
4 ORDER BY UnitPrice;

```

The 'Result Grid' shows the following data:

	ProductID	ProductName	UnitPrice	UnitsInStock
▶	2	Chang	19.0000	17
	36	Inlagd Sill	19.0000	112
	44	Gula Malacca	19.4500	27
	57	Ravioli Angelo	19.5000	36
*	NULL	NULL	NULL	NULL

The 'Output' window shows the 'Action Output' table:

#	Time	Action	Message	Duration / Fetch
✓ 1	22:55:38	SELECT table_schema, table_name, table_row...	13 row(s) returned	0.047 sec / 0.000 sec
✓ 2	00:17:09	SELECT DISTINCT TitleOfCourtesy FROM E...	5 row(s) returned	0.031 sec / 0.000 sec
✓ 3	00:22:05	SELECT TitleOfCourtesy FROM Employees O...	10 row(s) returned	0.031 sec / 0.000 sec
✓ 4	00:22:11	SELECT DISTINCT TitleOfCourtesy FROM E...	5 row(s) returned	0.031 sec / 0.000 sec
✓ 5	00:25:03	SELECT CompanyName, CONCAT(Conta...	6 row(s) returned	0.031 sec / 0.000 sec
✓ 6	00:33:39	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec

SELECT ProductID, ProductName, UnitPrice, UnitsInStock

- Retrieves the required product details (based on the question).

FROM Products

- Pulls data from the Products table.

WHERE UnitPrice BETWEEN 19 AND 19.5

- Filters products where the unit price is within the specified range (inclusive).

ORDER BY UnitPrice

- Sorts the results by price in ascending order by default.

This will return four products within the specified price range, sorted in increasing order.

4. List the ProductID, ProductName, UnitPrice, and UnitsInStock by the number of units in stock. Only show products that have a value less than 14.0 for the price and have at least 100 units available. (2)

The screenshot shows the Microsoft Access interface. At the top, a tab labeled 'Query1' is active. The SQL View pane contains the following query:

```

1 • SELECT ProductID, ProductName, UnitPrice, UnitsInStock
2 FROM Products
3 WHERE UnitPrice < 14.0
4 AND UnitsInStock >= 100
5 ORDER BY UnitsInStock;

```

Below the SQL pane, the 'Result Grid' tab is selected, displaying the following data:

	ProductID	ProductName	UnitPrice	UnitsInStock
▶	33	Geitost	2.5000	112
	75	Rhnbru Klosterbier	7.7500	125
*	NULL	NULL	NULL	NULL

At the bottom of the interface, the 'Output' pane shows the 'Action Output' table:

#	Time	Action	Message	Duration / Fetch
✓ 6	00:33:39	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
✓ 7	00:37:51	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.032 sec / 0.000 sec
✓ 8	00:40:38	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
✓ 9	00:41:26	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
✓ 10	00:46:56	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.031 sec / 0.000 sec
✓ 11	00:47:07	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.031 sec / 0.000 sec

SELECT ProductID, ProductName, UnitPrice, UnitsInStock

- Retrieves the required product details (required by question).

FROM Products

- Pulls data from the Products table.

WHERE UnitPrice < 14.0

- Filters products with a price lower than 14.0.

AND UnitsInStock >= 100

- Ensures that only products with at least 100 units in stock are included.

ORDER BY UnitsInStock

- Sorts the results by the number of units in stock in ascending order.

The query should return two products that meet the price and stock requirements.



5. List the product ID, name, and price for all products that are in the "Seafood" category [Use a subquery matching the category name]. Show the most expensive items first. (12)

Query 1 x

SQLAdditions

My Snippets

```

1 • SELECT ProductID, ProductName, UnitPrice
2 FROM Products
3 WHERE CategoryID = (
4     SELECT CategoryID
5     FROM Categories
6     WHERE CategoryName = 'Seafood'
7 )
8 ORDER BY UnitPrice DESC;

```

Result Grid

ProductID	ProductName	UnitPrice
18	Carnarvon Tigers	62.5000
10	Ikura	31.0000
37	Gravad lax	26.0000
30	Nord-Ost Matjeshering	25.8900
36	Inlagd Sill	19.0000
40	Boston Crab Meat	18.4000
73	Rd Kaviar	15.0000
58	Escargots de Bourgogne	13.2500
46	Spegesild	12.0000
41	Jack's New England Clam Chowder	9.6500
45	Rogede sild	9.5000
13	Konbu	6.0000
NULL	NULL	NULL

Products 11 x

Apply Revert Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
✓ 1	22:45:35	SELECT CONCAT(FirstName, '', LastName) AS Name, ...	9 row(s) returned	0.031 sec / 0.000 sec
✓ 2	23:07:26	SELECT OrderID, OrderDate, RequiredDate, ShippedD...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 3	23:20:22	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 4	23:23:16	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 5	23:38:43	SELECT OrderID, OrderDate, RequiredDate, ...	37 row(s) returned	0.047 sec / 0.000 sec
✓ 6	23:39:08	SELECT OrderID, OrderDate, RequiredDate, ...	9 row(s) returned	0.047 sec / 0.000 sec
✓ 7	23:46:11	SELECT CONCAT(Employees.FirstName, '', Employees....	1 row(s) returned	0.047 sec / 0.000 sec
✓ 8	23:53:55	SELECT CONCAT(FirstName, '', LastName) AS Name, ...	3 row(s) returned	0.031 sec / 0.000 sec
✓ 9	23:58:11	SELECT ProductID, ProductName, UnitPrice, UnitsInSt...	4 row(s) returned	0.032 sec / 0.000 sec
✓ 10	00:01:52	SELECT CompanyName, CONCAT(ContactName, ...	6 row(s) returned	0.047 sec / 0.000 sec
✓ 11	00:07:32	SELECT ProductID, ProductName, UnitPrice FROM Pro...	12 row(s) returned	0.047 sec / 0.000 sec



SELECT ProductID, ProductName, UnitPrice

- Retrieves product details
- ProductID uniquely identifies the product
- ProductName is the name of product
- UnitPrice ensures product can be sorted by price

FROM Products

- Pulls data from the Products table.

WHERE CategoryID = (SELECT CategoryID FROM Categories WHERE CategoryName = 'Seafood')

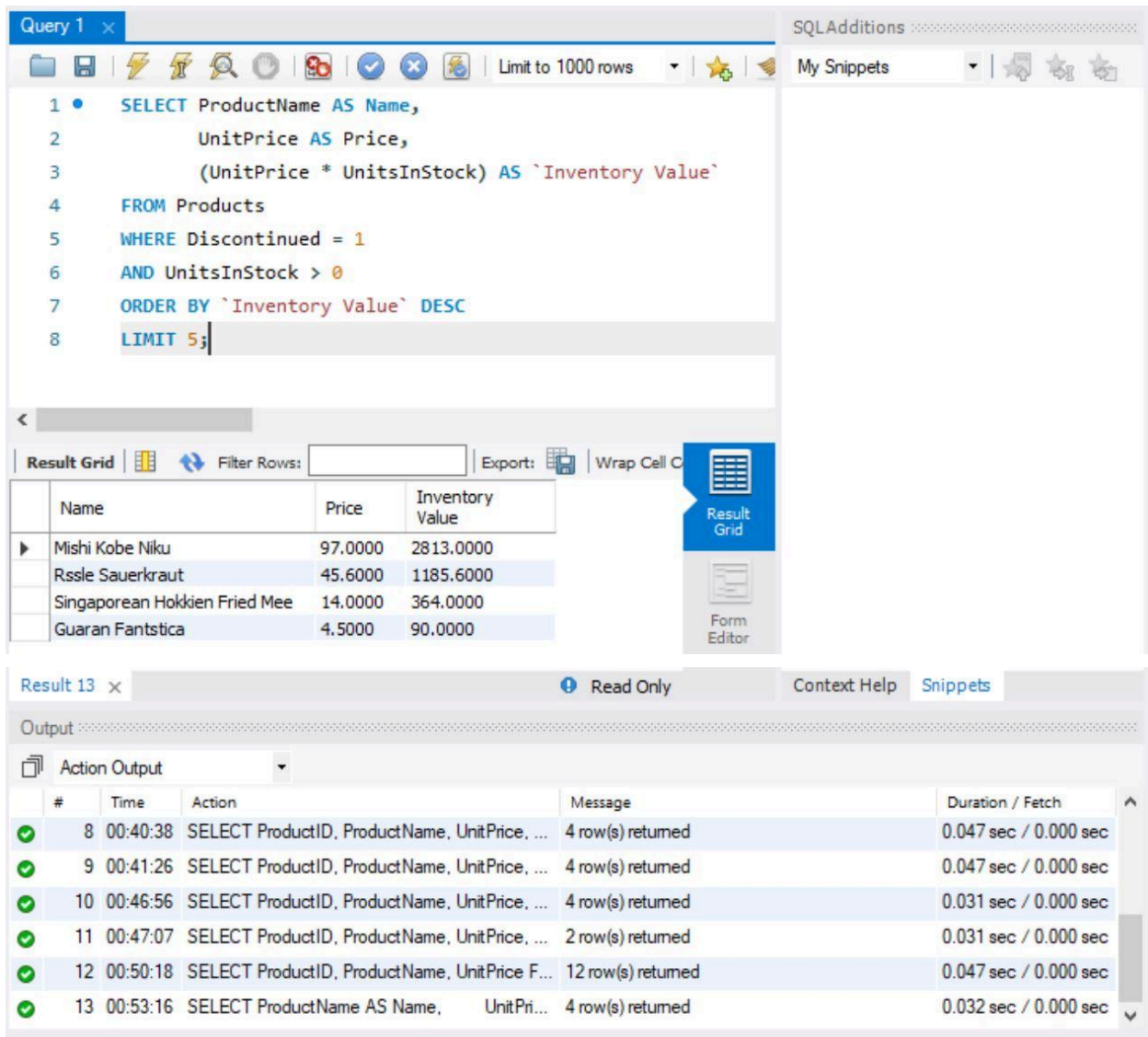
- Uses a subquery to find the CategoryID corresponding to "Seafood".
- Filters the Products table to include only those belonging to the "Seafood" category.

ORDER BY UnitPrice DESC

- Sorts results by UnitPrice in descending order, listing the most expensive items first.

This query should return 12 products that belong to the "Seafood" category, sorted from the highest to lowest price.

6. List the top 5 products in the company's inventory where the product is discontinued and they still have inventory. Display the product's name, unit price and total inventory value from most to least. Make the column headers "Name, Price, and Inventory Value".  
(4)



The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a query named 'Query 1' with the following SQL code:

```

1 SELECT ProductName AS Name,
2       UnitPrice AS Price,
3       (UnitPrice * UnitsInStock) AS `Inventory Value`
4 FROM Products
5 WHERE Discontinued = 1
6 AND UnitsInStock > 0
7 ORDER BY `Inventory Value` DESC
8 LIMIT 5;

```

The bottom pane shows the 'Result Grid' with the following data:

Name	Price	Inventory Value
Mishi Kobe Niku	97.0000	2813.0000
Rssle Sauerkraut	45.6000	1185.6000
Singaporean Hokkien Fried Mee	14.0000	364.0000
Guaran Fantstica	4.5000	90.0000

Below the result grid, there is a 'Result 13' tab showing a log of actions. The log includes the following entries:

#	Time	Action	Message	Duration / Fetch
8	00:40:38	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
9	00:41:26	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
10	00:46:56	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.031 sec / 0.000 sec
11	00:47:07	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.031 sec / 0.000 sec
12	00:50:18	SELECT ProductID, ProductName, UnitPrice F...	12 row(s) returned	0.047 sec / 0.000 sec
13	00:53:16	SELECT ProductName AS Name, UnitPri...	4 row(s) returned	0.032 sec / 0.000 sec

SELECT ProductName AS Name, UnitPrice AS Price, (UnitPrice \* UnitsInStock) AS Inventory Value

- Renames columns as "Name", "Price", and "Inventory Value" as required.
- Calculates total inventory value by multiplying UnitPrice \* UnitsInStock.

FROM Products

- Fetches data from the Products table.

WHERE Discontinued = 1

- Filters only discontinued products.

AND UnitsInStock > 0

- Ensures only products that still have inventory are included.

ORDER BY Inventory Value DESC

- Sorts by total inventory value from most to least.

LIMIT 5

- Returns only the top 5 products. But there were only 4!

The query should return up to 5 products, but since the hint states 4 rows, it likely means only 4 discontinued products match the criteria. This was discussed during Office Hours. Not a trick, but a way to realize how LIMIT works.

7. List each of the Sales employees' total number of orders (# Orders) and First and Last name together (Name). List the highest counts first. (7)

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a SQL query for 'Query 1'. The query selects the concatenated first and last names of sales employees and the count of their orders, ordered by the count in descending order. The bottom pane shows the 'Result Grid' with 9 rows of data. The right sidebar contains 'SQLAdditions' and 'My Snippets'. The bottom status bar shows the query execution details.

```

1 • SELECT CONCAT(FirstName, ' ', LastName) AS Name,
2     COUNT(Orders.OrderID) AS `# Orders`
3 FROM Employees
4 JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID
5 WHERE Employees.Title LIKE '%Sales%'
6 GROUP BY Employees.EmployeeID
7 ORDER BY `# Orders` DESC

```

Name	# Orders
Margaret Peacock	156
Janet Leverling	127
Nancy Davolio	123
Laura Callahan	104
Andrew Fuller	96
Robert King	72
Michael Suyama	67
Anne Dodsworth	43
Steven Buchanan	42

Result 1 x Read Only Context Help Snippets

Output

#	Time	Action	Message	Duration / Fetch
1	22:45:35	SELECT CONCAT(FirstName, ' ', LastName) A...	9 row(s) returned	0.031 sec / 0.000 sec

CONCAT(FirstName, ' ', LastName) AS Name

- Combines the first and last names into a single column labeled "Name".
- Added space in quotes for legibility.

COUNT(Orders.OrderID) AS # Orders

- Counts the total number of orders assigned to each employee.

FROM Employees JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID

- Joins the Employees table with the Orders table based on EmployeeID.

WHERE Employees.Title LIKE '%Sales%'

- This line filters only employees with "Sales" in their job title using LIKE.
- Have to be careful of this in the real world because sometimes titles like "Sales Manager" exists
  - The keyword '%Sales%' would also pull from people with titles of "Sales Manager" or "Sales Representative" or "Sales Agent" which would make the number higher than 7
  - I could have made the judgment call to use "LIMIT 7" here (hard-coded) in order for it to align with what the question is asking for, but after discussing with the professor, the "correct" answer (although the question explicitly states 7 rows) is to show all 9 rows.

GROUP BY Employees.EmployeeID

- Groups by each Sales Employee to get their total order count.

ORDER BY # Orders DESC

- Sorts the results in descending order to show employees with the most orders first.

The query should return 9 sales employees, ordered from the highest to lowest number of orders.

8. List each of the Sales Representatives' total number of orders (# Orders) and First and Last name together (Name). List the highest counts first and only if the count is at least 100. (3)

The screenshot shows the SQL Server Enterprise Manager interface. The top pane displays a SQL query (Query 1) that selects the concatenated first and last names of sales representatives and the count of orders they have, ordered by the count in descending order, with a filter for counts greater than or equal to 100.

```

1 • SELECT CONCAT(FirstName, ' ', LastName) AS Name,
2       COUNT(Orders.OrderID) AS `# Orders`
3 FROM Employees
4 JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID
5 WHERE Employees.Title = 'Sales Representative'
6 GROUP BY Employees.EmployeeID
7 HAVING `# Orders` >= 100
8 ORDER BY `# Orders` DESC;
9

```

The bottom pane shows the results of the query in a grid format. The results are as follows:

Name	# Orders
Margaret Peacock	156
Janet Leverling	127
Nancy Davolio	123

Below the results grid, the 'Output' pane shows the execution log, which includes the duration and number of rows returned for each step of the query execution.

#	Time	Action	Message	Duration / Fetch
5	00:25:03	SELECT CompanyName, CONCAT(Cont...	6 row(s) returned	0.031 sec / 0.000 sec
6	00:33:39	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
7	00:37:51	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.032 sec / 0.000 sec
8	00:40:38	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
9	00:41:26	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
10	00:46:56	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.031 sec / 0.000 sec
11	00:47:07	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.031 sec / 0.000 sec
12	00:50:18	SELECT ProductID, ProductName, UnitPrice ...	12 row(s) returned	0.047 sec / 0.000 sec
13	00:53:16	SELECT ProductName AS Name, UnitPri...	4 row(s) returned	0.032 sec / 0.000 sec
14	00:56:21	SELECT CONCAT(FirstName, ' ', LastName) A...	9 row(s) returned	0.031 sec / 0.000 sec
15	00:58:39	SELECT CONCAT(FirstName, ' ', LastName) A...	3 row(s) returned	0.047 sec / 0.000 sec

CONCAT(FirstName, ' ', LastName) AS Name

- Merges first and last names into a single field named "Name".
- Added the space in quotations just for formatting purposes (legible).

COUNT(Orders.OrderID) AS # Orders

- Counts the number of orders assigned to each "Sales Representative". This time, it's not if the employee's title just has the word "Sales" in it.

FROM Employees JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID

- Joins the Employees table with the Orders table using EmployeeID.

WHERE Employees.Title = 'Sales Representative'

- Filters the results to include only employees with the exact title "Sales Representative".
- We can't use '%Sales%' here because there are "Sales Agents", "Sales Managers", etc.

GROUP BY Employees.EmployeeID

- Groups results by employee to compute order counts.

HAVING # Orders >= 100

- HAVING Filters results to only include employees with at least 100 orders.

ORDER BY # Orders DESC

- Sorts results in descending order by total order count.

The query should return only 3 employees who are Sales Representatives with at least 100 orders, sorted from highest to lowest order count.



9. List all Suppliers by country ascending if the supplier's region is not indicated. (20)

Query 1 x

```

1 • SELECT SupplierID, CompanyName, Country
2 FROM Suppliers
3 WHERE Region IS NULL
4 ORDER BY Country ASC;

```

SQLAdditions

My Snippets

Limit to 1000 rows

Result Grid

SupplierID	CompanyName	Country
10	Refrescos Americanas LTDA	Brazil
21	Lyngbysild	Denmark
23	Karkki Oy	Finland
28	Gai pturage	France
27	Escargots Nouveaux	France
18	Aux joyeux ecclsiastiques	France
11	Heli Swaren GmbH & Co. KG	Germany
12	Plutzer Lebensmittelgromrkte AG	Germany
13	Nord-Ost-Fisch Handelsgesellschaft mbH	Germany
26	Pasta Buttini s.r.l.	Italy
14	Formaggi Fortini s.r.l.	Italy
4	Tokyo Traders	Japan
6	Mayumi's	Japan
22	Zaanse Snoepfabriek	Netherl...
15	Norske Meierier	Norway
20	Leka Trading	Singapore
17	Svensk Sjfda AB	Sweden
9	PB Knckebrd AB	Sweden
8	Specialty Biscuits, Ltd.	UK
1	Exotic Liquids	UK
NULL	NULL	NULL

Suppliers 16 x

Apply Revert Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
6	00:33:39	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
7	00:37:51	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.032 sec / 0.000 sec
8	00:40:38	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
9	00:41:26	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
10	00:46:56	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.031 sec / 0.000 sec
11	00:47:07	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.031 sec / 0.000 sec
12	00:50:18	SELECT ProductID, ProductName, UnitPrice ...	12 row(s) returned	0.047 sec / 0.000 sec
13	00:53:16	SELECT ProductName AS Name, UnitPri...	4 row(s) returned	0.032 sec / 0.000 sec
14	00:56:21	SELECT CONCAT(FirstName, '', LastName) A...	9 row(s) returned	0.031 sec / 0.000 sec
15	00:58:39	SELECT CONCAT(FirstName, '', LastName) A...	3 row(s) returned	0.047 sec / 0.000 sec
16	01:01:09	SELECT SupplierID, CompanyName, Country ...	20 row(s) returned	0.047 sec / 0.000 sec

SELECT SupplierID, CompanyName, Country

- Retrieves the Supplier ID, Company Name, and Country.

FROM Suppliers

- Pulls data from the Suppliers table.

WHERE Region IS NULL

- Ensures only suppliers where Region is NULL are included.
- In the Northwind database, Region may be NULL for suppliers who haven't provided that detail.

ORDER BY Country ASC

- Sorts results alphabetically by Country.

The query should return 20 suppliers who do not have a Region specified.

10. Using a RIGHT JOIN of Orders to Employees, find all employees that do not have any orders (without grouping the data). Print the Full Name (First, space, Last) and the salary of the employee. (1)

The screenshot shows the SQL Server Enterprise Manager interface. The 'Query 1' window displays the following SQL query:

```

1 • SELECT CONCAT(Employees.FirstName, ' ', Employees.LastName)
2     Employees.Salary
3 FROM Employees
4 LEFT JOIN Orders ON Employees.EmployeeID = Orders.EmployeeID
5 WHERE Orders.OrderID IS NULL;

```

The 'Result Grid' shows the following data:

Full Name	Salary
Coach Prime	458333

The 'Output' window shows the execution log with the following entries:

#	Time	Action	Message	Duration / Fetch
8	00:40:38	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
9	00:41:26	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.047 sec / 0.000 sec
10	00:46:56	SELECT ProductID, ProductName, UnitPrice, ...	4 row(s) returned	0.031 sec / 0.000 sec
11	00:47:07	SELECT ProductID, ProductName, UnitPrice, ...	2 row(s) returned	0.031 sec / 0.000 sec
12	00:50:18	SELECT ProductID, ProductName, UnitPrice F...	12 row(s) returned	0.047 sec / 0.000 sec
13	00:53:16	SELECT ProductName AS Name, UnitPric...	4 row(s) returned	0.032 sec / 0.000 sec
14	00:56:21	SELECT CONCAT(FirstName, ' ', LastName) AS...	9 row(s) returned	0.031 sec / 0.000 sec
15	00:58:39	SELECT CONCAT(FirstName, ' ', LastName) AS...	3 row(s) returned	0.047 sec / 0.000 sec
16	01:01:09	SELECT SupplierID, CompanyName, Country F...	20 row(s) returned	0.047 sec / 0.000 sec
17	01:02:42	SELECT CONCAT(Employees.FirstName, ' ', E...	0 row(s) returned	0.032 sec / 0.000 sec
18	01:04:25	SELECT CONCAT(Employees.FirstName, ' ', E...	1 row(s) returned	0.047 sec / 0.000 sec

CONCAT(Employees.FirstName, ' ', Employees.LastName) AS Full Name

- Combines first and last names into a single column named "Full Name".
- This is odd because "Coach" is not typically a first name, so maybe that's just a nickname we call Mr. Prime?

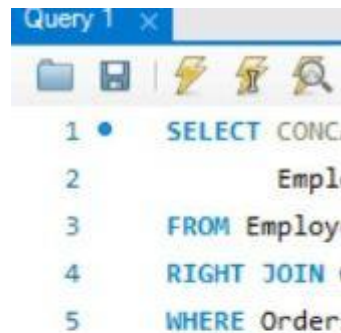
Employees.Salary

- Retrieves the salary of the employee.

WHERE Orders.OrderID IS NULL:

- Filters the results to show only employees who have no orders.

I originally used RIGHT JOIN instead of LEFT JOIN and ended up returning 0 rows.



```
Query 1 x
1 SELECT CONGRUITY
2     EmpID
3 FROM Employee
4 RIGHT JOIN Orders
5 WHERE OrderID
```



	Full Name	Salary
--	-----------	--------

Shown here:

I realized this was wrong because the requirement asks to find employees with no orders and this was not returning the expected amount of 1 row (returned 0 rows instead). The original intent was to perform a RIGHT JOIN to ensure all records from Orders are included even if there was no matching employee. This prioritizes Orders which means it only includes employees that match orders, not those without orders. So we must change RIGHT JOIN to LEFT JOIN to find employees with no orders (starting from Employees). We want to keep all employees and check if they're missing from Orders.

This now returns 1 row.

11. Find all orders that were shipped over a week (7 days) later than the required date.

(9) Also answer the following questions in your submission for this problem:

1. Which fields should be displayed and why?

OrderID

- This is necessary and helps us uniquely identify which order is delayed
- This is also our primary key in our database

OrderDate

- This one is more *optional* (and on the fence in my opinion), but I added it anyway because I wanted to provide context on how long a customer was waiting since the beginning of their purchase date.
- While this might not necessarily tell us much about how long the order was delayed, this *can still be* valuable business information as in “we

were only late by 8 days, but the customer waited an entire year to receive their order” (giving a more holistic picture of the customer experience)

- Screenshot shown here:

OrderDate	RequiredDate	ShippedDate
1997-12-15 00:00:00	1997-12-29 00:00:00	1998-01-21 00:00:00
1997-11-03 00:00:00	1997-11-17 00:00:00	1997-12-05 00:00:00
1997-01-23 00:00:00	1997-02-06 00:00:00	1997-02-24 00:00:00
1998-03-24 00:00:00	1998-04-07 00:00:00	1998-04-24 00:00:00
1997-04-23 00:00:00	1997-05-07 00:00:00	1997-05-23 00:00:00
1998-01-12 00:00:00	1998-01-26 00:00:00	1998-02-06 00:00:00
1997-09-08 00:00:00	1997-10-06 00:00:00	1997-10-15 00:00:00
1997-09-10 00:00:00	1997-09-24 00:00:00	1997-10-03 00:00:00
1998-01-13 00:00:00	1998-01-27 00:00:00	1998-02-04 00:00:00

RequiredDate

- This is necessary info and is half of the calculation because this shows when the order was expected to be shipped

ShippedDate

- This is required because it's the actual date the goods were shipped

## 2. How should the results be ordered?

ORDER BY (ShippedDate - RequiredDate) DESC;

- We can sort by delay duration by subtracting the RequiredDate from the ShippedDate and then order it in descending order to most delayed orders first. I initially wrote out this ShippedDate - RequiredDate DESC; as a prototype to 1) jot down the logic that I want to express and 2) write something that first works, and then we can improve on it by getting creative
- There's a cleaner way to do this in the form of DATEDIFF (which I did below in my actual query), and I replaced this whole subtraction nonsense

Another school of thought might be to order by shipped date.

ORDER BY ShippedDate DESC;

- We might want to do this because this sorts when the shipment actually happened, showing the most recent late shipments first
- This contrasts with the other strategy because this one reprioritizes recently late shipments. As in, maybe customers don't consider 1 day late as a big deal (salvageable customer experience), whereas being 29 days late and 30 days late doesn't make a difference because the shipment is extraordinarily late already.
  - If we're thinking about the context of food or something, something that is 1 day late might not expire but something that has been



The final judgment will have to depend on the business owner/leader. But for the sake of this homework, it's best to just go with the first option (reflected in the screenshot)

```
SELECT OrderID, OrderDate, RequiredDate, ShippedDate
```



- Retrieves an unique identifier for each other, the date when the customer placed the order, when customer expected the order, and when the order was actually shipped

FROM Orders

- Pulls data from the Orders table.

DATEDIFF(ShippedDate, RequiredDate) AS Days Late

- Calculates the difference between ShippedDate and RequiredDate as how many days the shipment was late and stores it in a new column called "Days Late"
- If positive, the order was shipped late.
- If negative or zero, the order was on time or early, but we won't see this because we made sure the interval is > 7 DAY
  - Some of the values are less than 7 days late, like 1 -> 6 days late, and they are not listed here.

Another thing to note: ORDER BY `Days Late` needs to happen before ShippedDate DESC; because if the orders were reversed, we would not be ranking in descending order according to the days late. The latest tardy will not be the first row.

We don't need JOINS here because all required fields exist in the Orders table. Orders table exists and contains OrderID, OrderDate, RequiredDate, ShippedDate The query should return 9 rows, listing all orders delayed by more than 7 days beyond the required date.

Note to self: We may want to check for NULL values for the ShippedDate because it's possible that orders are unshipped. We also want to be aware that some employees might not have orders, and how that might affect future queries.

## Grading Rubric

- Not optimal, clean code = format
- Professor said it's ok to comment after the screenshot, but future will need to comment inside the code