Cal Poly Pomona

Deshol John

MS SQL Query Database

March 28, 2024

CIS 3050.05

Section 5

Spring 2024

# Table of Contents

[Table of Contents 1](#_Toc845970538)

[Students' Academic Dishonesty 2](#_Toc613157683)

[Introduction/ Project Description and Requirements 3](#_Toc1340104584)

[Query #1 4](#_Toc1328144097)

[Query #2 5](#_Toc330052501)

[Query #3 6](#_Toc1617982954)

[Query #4 7](#_Toc650486866)

[Query #5 8](#_Toc1102244877)

[Query #6 9](#_Toc390240436)

[Query #7 10](#_Toc900920025)

[Query #8 11](#_Toc1603427146)

[Query #9 12](#_Toc1069786042)

[Query #10 13](#_Toc637447847)

[Query #11 14](#_Toc1954668615)

[Query #12 15](#_Toc1189889335)

[Query #13 16](#_Toc858206923)

[Query #14 17](#_Toc1281651346)

[Query #15 18](#_Toc233117220)

[Query #16 19](#_Toc391921082)

[Query #17 20](#_Toc1276881142)

[Query #18 21](#_Toc1271553563)

[Query #19 23](#_Toc175694526)

[Query #20 24](#_Toc1999711348)

[Query #21 25](#_Toc767556420)

[Query #22 26](#_Toc625974102)

[Query #23 27](#_Toc1767493646)

[Query #24 28](#_Toc943000377)

[Query #25 29](#_Toc2084501034)

[Query #26 30](#_Toc504322462)

[Query #27 31](#_Toc1468334449)

[Query #28 32](#_Toc493923825)

[Results and Discussion 33](#_Toc1622178202)

[Lessons Learned 34](#_Toc2120961898)

[Conclusion 34](#_Toc322663018)

[References 34](#_Toc617085426)

# Students' Academic Dishonesty

Statement of Academic Honesty

My name is: \_\_\_\_Deshol John\_\_\_\_\_\_, I declare that, except where fully referenced, no aspect of this project has been copied from any other source. I understand that any act of Academic Dishonesty such as plagiarism or collusion may result in serious offense and punishments. I promise not to lie about my academic work, to cheat, or to steal the words or ideas of others, nor will I help fellow students to violate the Code of Academic Honesty.

Name: \_\_\_\_\_Deshol John\_\_\_\_\_\_

Date: \_\_\_\_\_\_3/28/2024\_\_\_\_\_\_\_\_\_

Signature: \_\_\_\_\_Deshol John\_\_\_\_\_\_\_\_\_

# Introduction/ Project Description and Requirements

Introduction

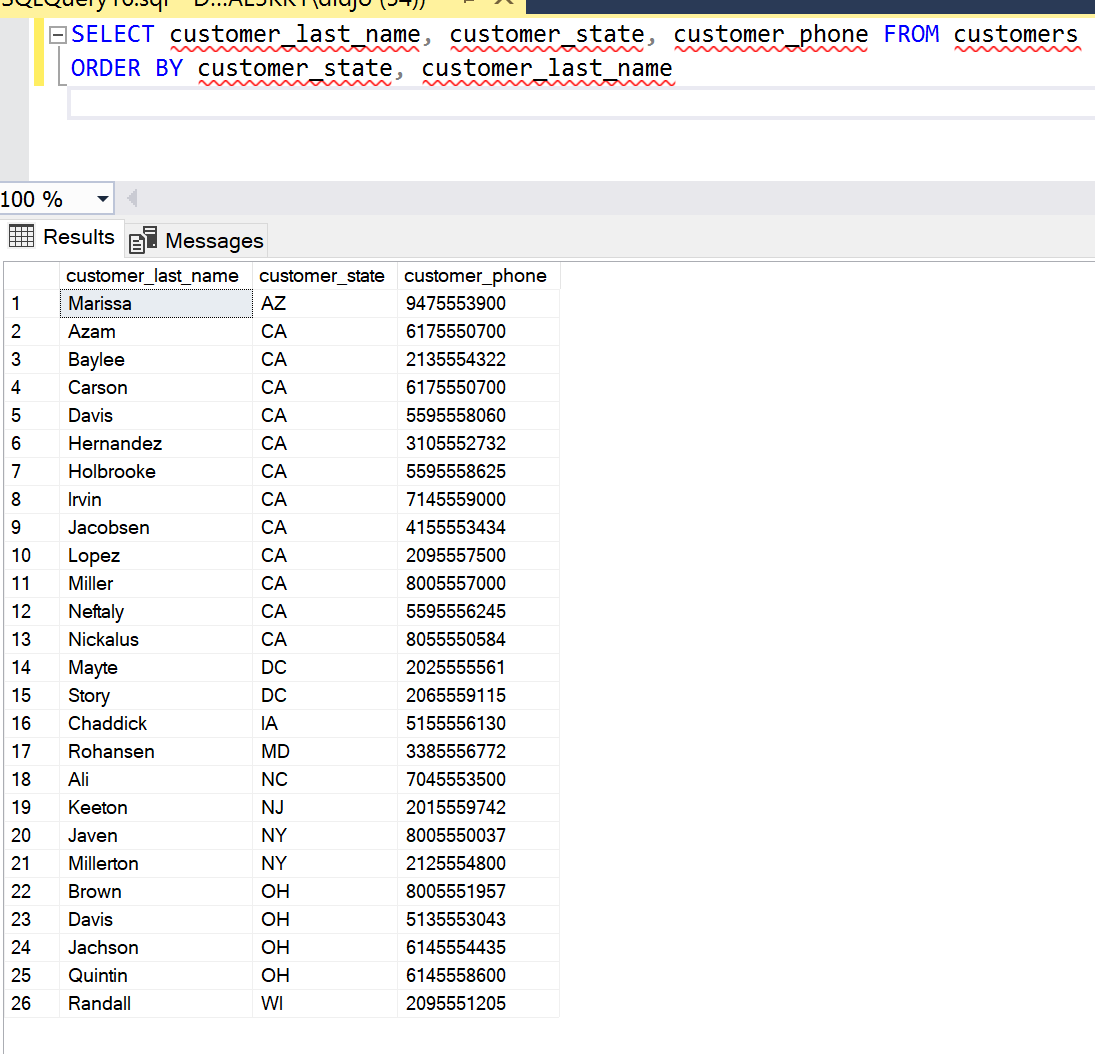
The following project is for Microsoft SQL and SQL Server Management studios. Project Two was assigned to showcase my understanding of Microsoft SQL. To complete this, I had to develop a database that included logical and physical designs while demonstrating the ability to deliver meaningful reports. Making the database involves creating queries and using SQL select statements. The purpose will be for designing, developing, and demonstrating the functionality of a database created based on a set of business specifications that I create. Specific data will be given to me, and it will be my job to develop queries in a database that showcases the data in an organized matter. This will be through multiple queries and creating tables represented by the Entity Relationship Diagram (ERD) provided, select statements, insert statements, and a few more statements which you will see in the following pages. The script file provided me with tables and information I must add to help achieve the MS SQL objective. The following pages will also show the queries extracted from the database that I had created.

Project Description

This project is designed to touch all aspects of the fundamental concepts of database design and logical/physical data modeling covered in chapters 5 and 6. You are responsible for designing, developing, and demonstrating the functionality of a database created based on a set of business specifications that you create. Each student should submit a report that includes the logical and physical design of the database and demonstrates the database's ability to deliver meaningful reports.

The MS SQL project will introduce the various aspects of the SQL SELECT statement and the methods of retrieving data from the database tables. This project will also introduce the  
fundamentals of updating and deleting records. The project will utilize a set of tables that are  
represented by the ERD and are created and populated by the script file.

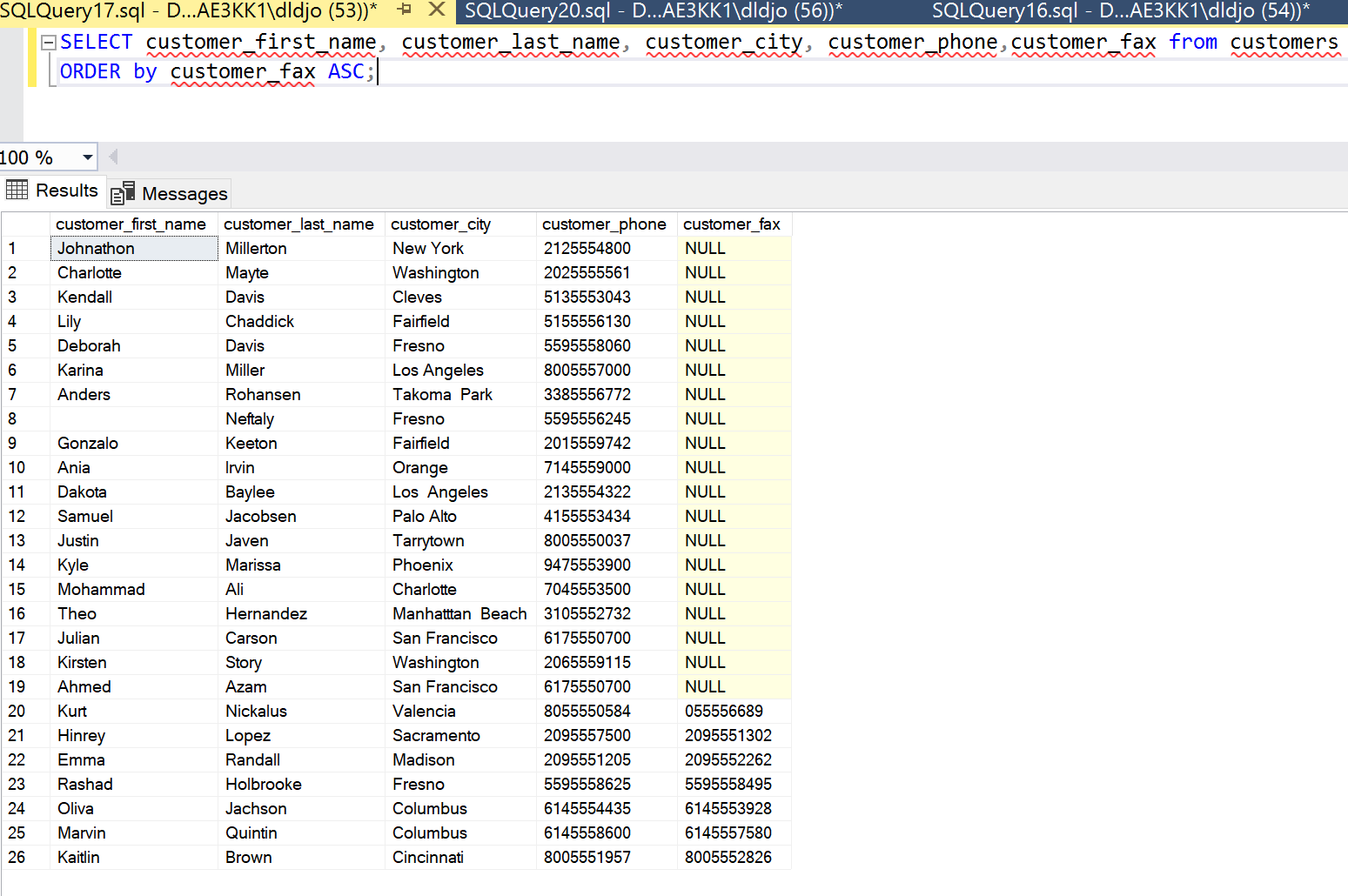
# Query #1

1. Write a query that displays a list of all customers showing the customer’s **last name**,  
   customer **state**, and **phone number**. Sort the results by customer state, then customer  
   last name.
2. SELECT customer\_last\_name, customer\_state, customer\_phone FROM customers ORDER BY customer\_state, customer\_last\_name
3. 
4. Once I created the tables in the database, I used the SELECT statement to display the list of customers. I started by selecting the customer’s last name and phone number then sorted it so that the last name became before the phone number.

# Query #2

1. Write a query that displays a list of all customers showing the customer’s **first name**, **last name**, **city**, **phone number** and **fax**. Sort the results by customer fax number in ascending order
2. SELECT customer\_first\_name, customer\_last\_name, customer\_city, customer\_phone, customer\_fax from customers

ORDER by customer\_fax ASC;

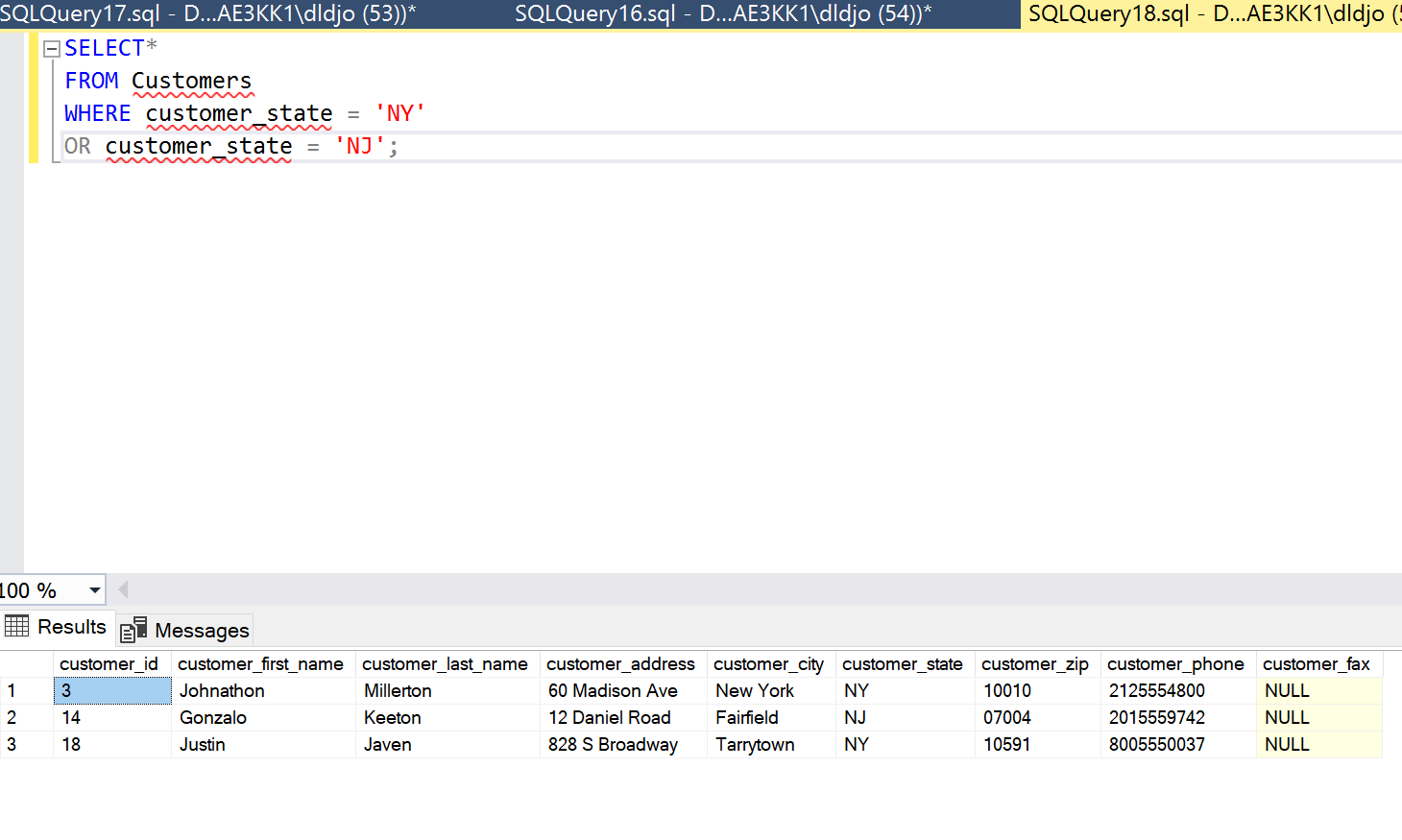
1. 
2. Using the SELECT statement I selected the customer’s first and last name, phone number and fax. After I ordered the data by the fax number in ascending order with the ORDER BY statement.

# Query #3

1. Write a query that displays all the customers from New York or New Jersey in the  
   “Customers” table.
2. SELECT\* FROM Customers

WHERE customer\_state = 'NY'

OR customer\_state = 'NJ';

1. 
2. Using SELECT \* FROM I selected all the information from the customer table. Then I made sure to filter using the WHERE statement to only show me customers from Los Angeles or San Francisco.

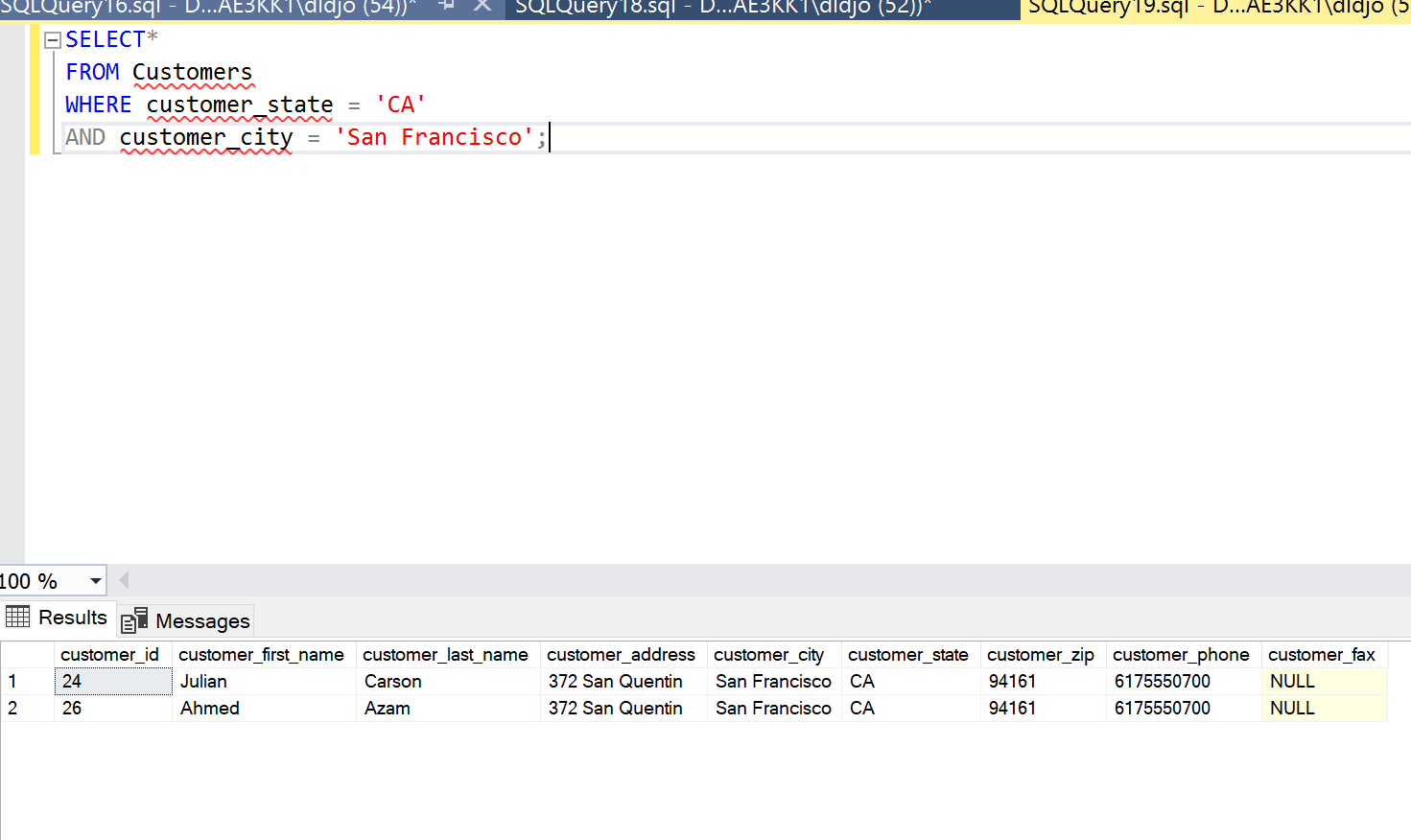
# Query #4

1. Write a query that displays all the customers from the state of California and live in San  
   Francisco.
2. SELECT\*

FROM Customers

WHERE customer\_state = 'CA'

AND customer\_city = 'San Francisco';

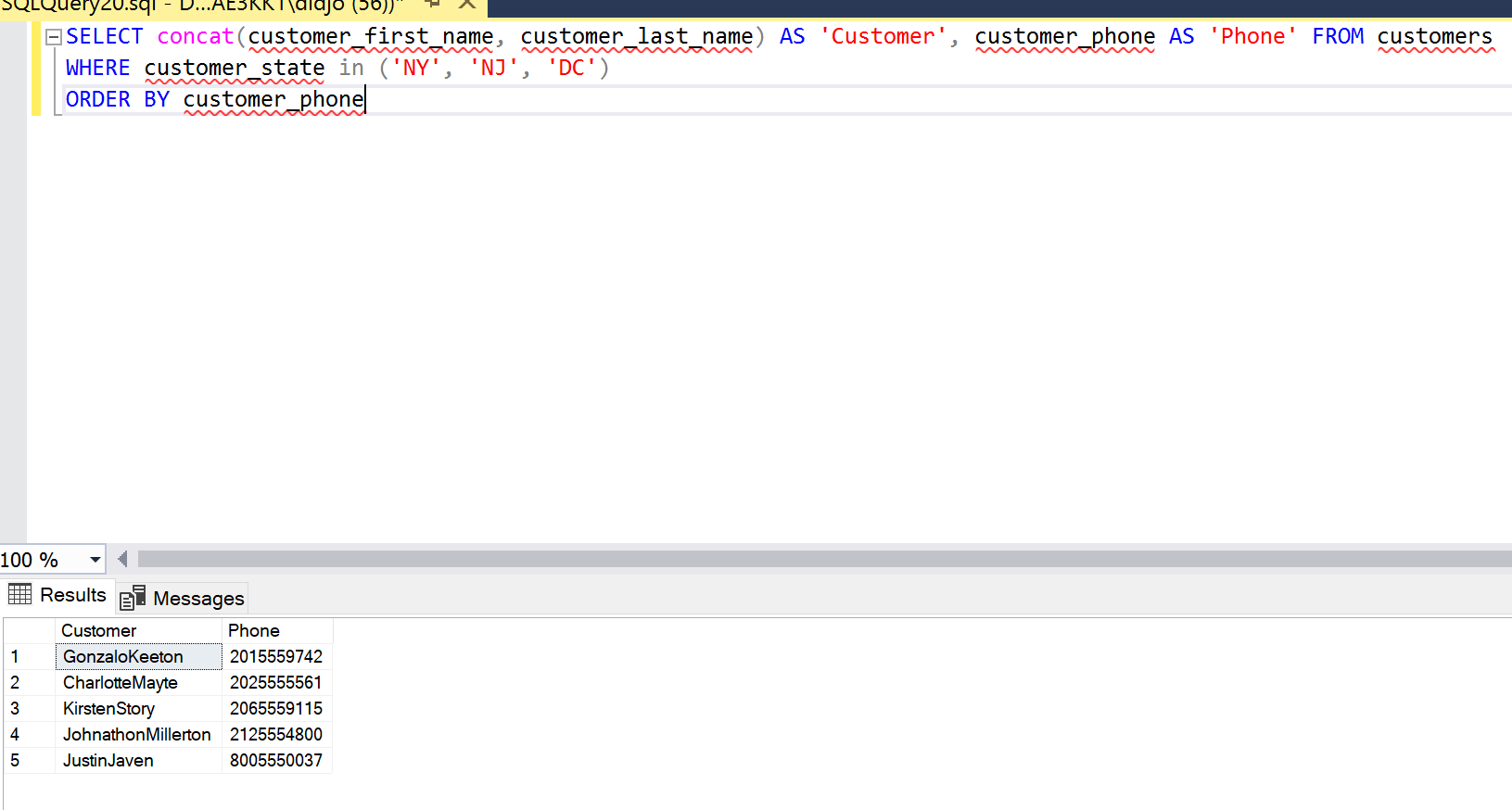
1. 
2. I selected all the information from the customers, then commanded the database to only give me the information from customers that live in San Francisco, California

# Query #5

1. Write a query that displays each customer name as a single field in the format “firstname  
   lastname” with a heading of Customer, along with their phone number with a heading of  
   Phone. Use the IN operator to only display customers in New York, New Jersey, or  
   Washington D.C. Sort the results by phone number.
2. SELECT concat(customer\_first\_name, customer\_last\_name) AS 'Customer', customer\_phone AS 'Phone' FROM customers

WHERE customer\_state in ('NY', 'NJ', 'DC')

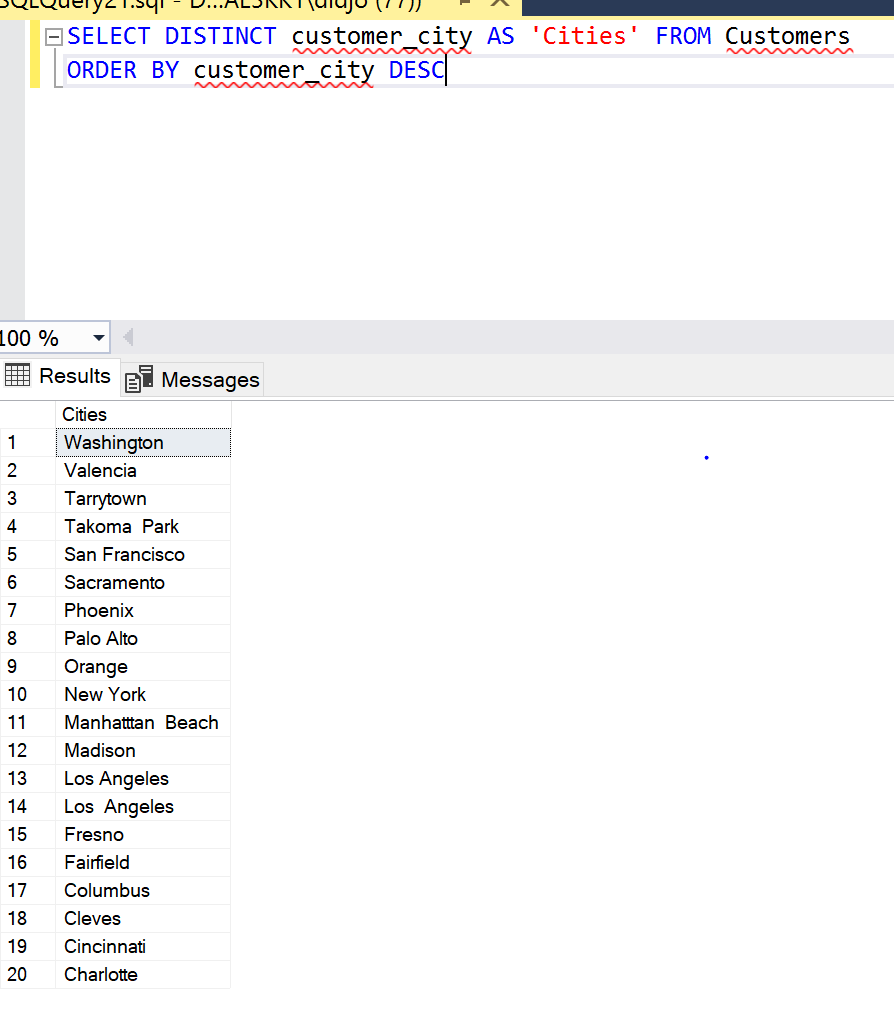
ORDER BY customer\_phone

1. 
2. Using the concat command I placed both the customer first and last name in one field to have them both display as a full name which I named “Customer”. Then I renamed the heading of all the phone numbers to “Phone”. After I filtered the customers by only showing me the ones that live in New York, New Jersey, or Washington DC. At the end using ORDER BY I sorted the customers by phone number.

# Query #6

1. Write a query that will list all the cities that have customers with a heading of Cities. Only list each city once (no duplicates) and sort in descending alphabetical order.
2. SELECT DISTINCT customer\_city AS 'Cities' FROM Customers

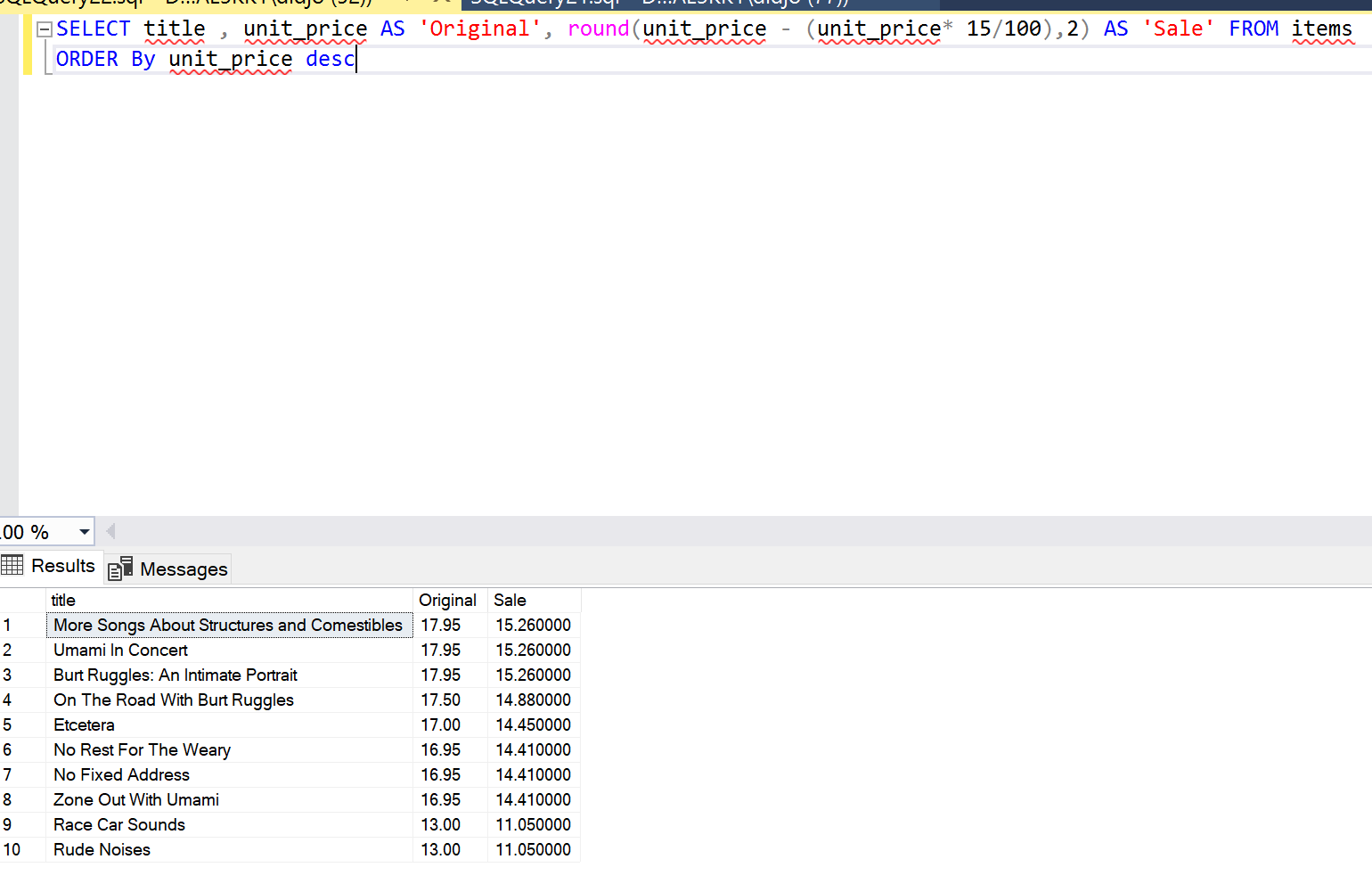
ORDER BY customer\_city DESC

1. 
2. To start off I used the SELECT DESTINCT statement to view all the cities without seeing duplicates. In addition I named the field to say “Cities” when displaying. At the end I sorted the list in alphabetical order in a descending format using ORDER BY.

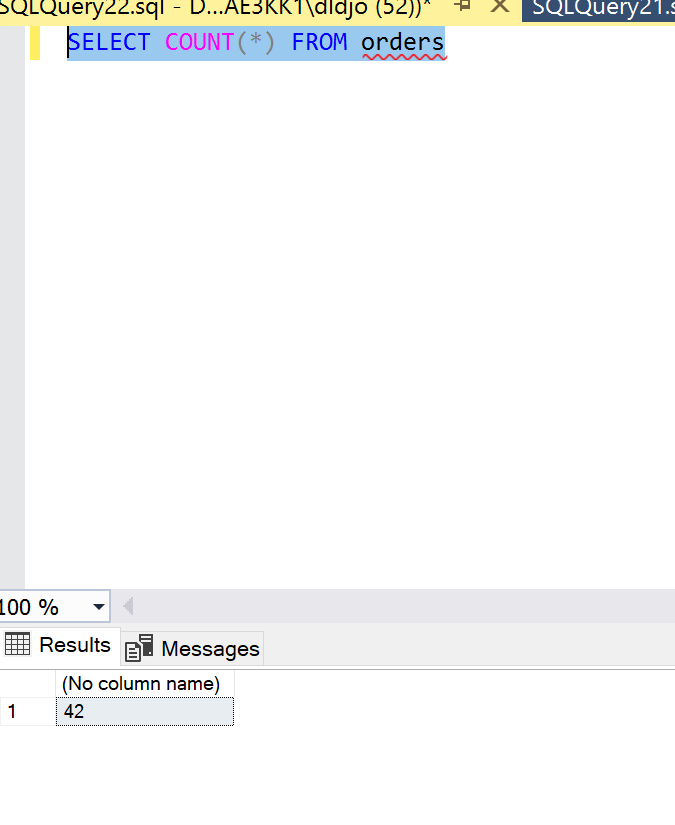
# Query #7

1. Write a query that displays the title of each item along with the price (with a heading of  
   Original) and a calculated field reflecting the price with a 15% discount (with a heading of Sale). Display the sale price with two decimal places using the ROUND function. Sort by price from highest to lowest.
2. SELECT title , unit\_price AS 'Original', round(unit\_price - (unit\_price\* 15/100),2) AS 'Sale' FROM items

ORDER By unit\_price desc

1. 
2. Using SELECT to select the item titles and prices and renamed the item price to display as “original” in the heading. Then I added a round function that calculates the 15% discount while also rounding it to the second decimal place and renamed the discounted price to “Sale”. After I sorted the items by unit price from highest to lowest.

# Query #8

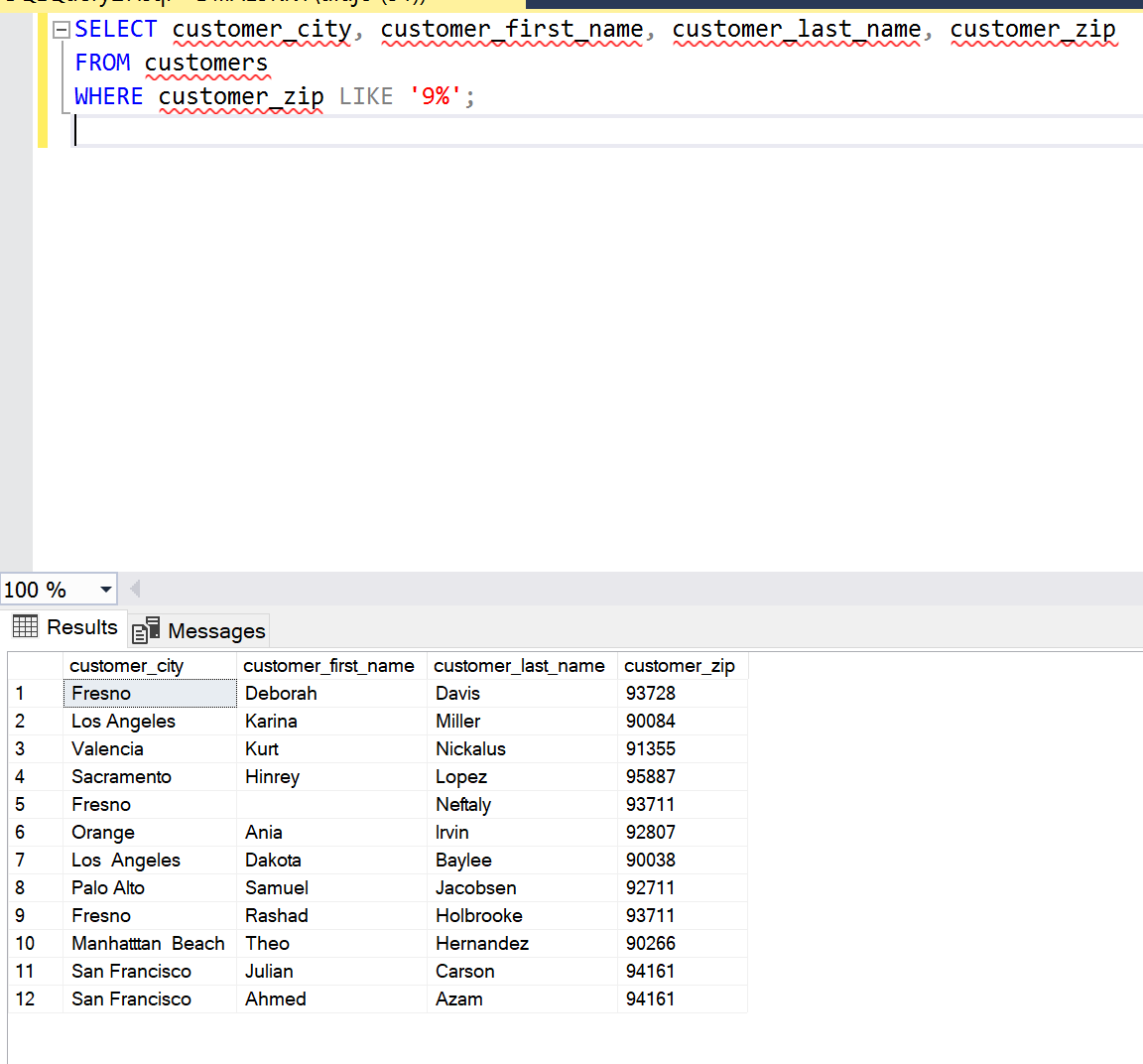
1. Write a query that displays the number of orders.
2. SELECT COUNT(\*) FROM orders
3. 
4. I used the SELECT COUNT statement to display the total number of orders from the orders table.

# Query #9

1. Write a query that displays the customer city, first name, last name, and zip code from the customer’s table. Use the LIKE operator to only display customers that reside in any zip  
   code beginning with 9.
2. SELECT customer\_city, customer\_first\_name, customer\_last\_name, customer\_zip

FROM customers

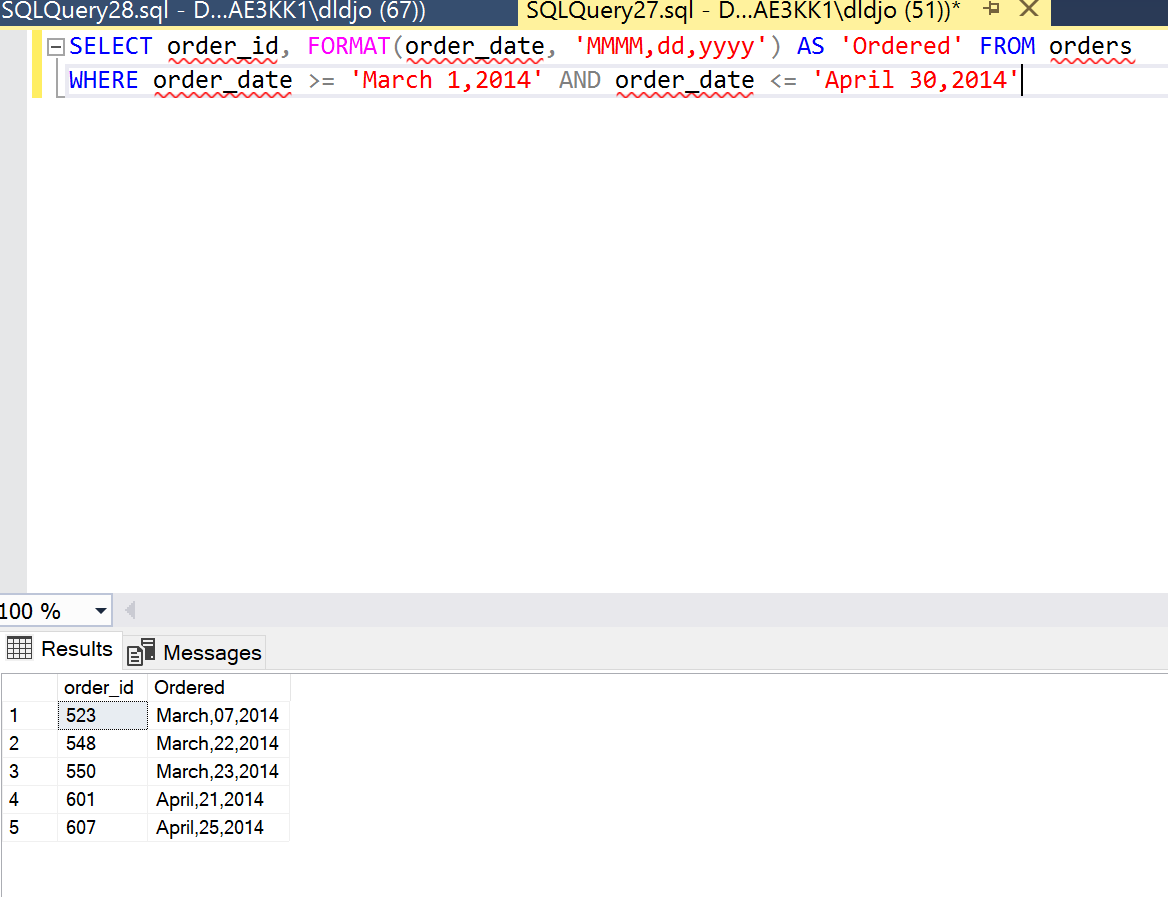
WHERE customer\_zip LIKE '9%';

1. 
2. From the customers table I selected the customer city, first and last name, and zip using both the SELECT and FROM statements. Once selected I used the LIKE statement to filter the results by displaying customer zips that only start with a 9.

# Query #10

1. Write a query that displays the order id and order date for any orders placed from March  
   1, 2014 through April 30, 2014. Do this WITHOUT using the BETWEEN clauses. Format the date field as Month dd, yyyy and use a heading of “Ordered”.
2. SELECT order\_id, FORMAT(order\_date, 'MMMM,dd,yyyy') AS 'Ordered' FROM orders

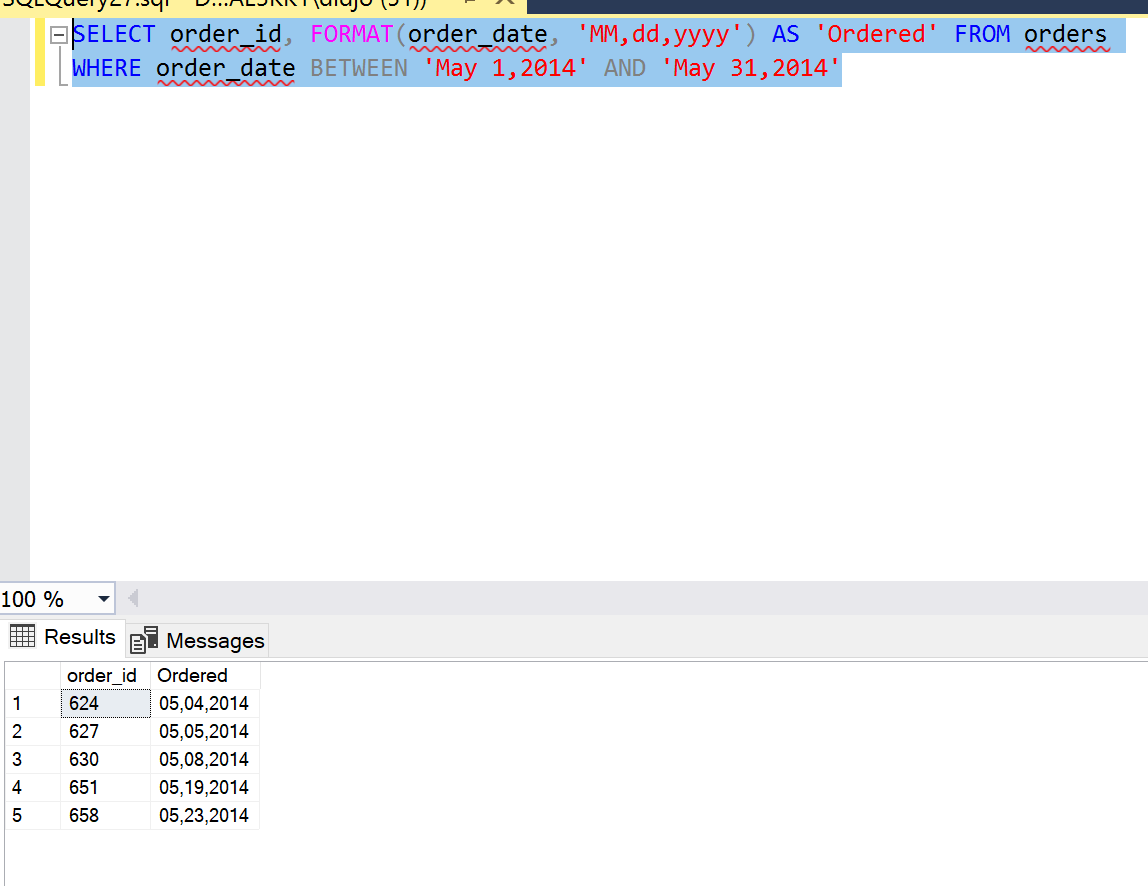
WHERE order\_date >= 'March 1,2014' AND order\_date <= 'April 30,2014'

1. 
2. From the order table I selected both the order id and date, to then format the order date as ‘MMMM, dd, yyyy’. In addition to formatting the date I changed the heading to say “Ordered”. I then used a function to show me orders that were > = “March 1st, 2014” and <= “April 30th, 2014”. Doing it this way allowed me to display the correct information needed without using a BETWEEN clause.

# Query #11

1. Write a query that displays the order id and order date for any orders placed during the  
   month of May 2014. Do this using the BETWEEN clauses. Format the date field as  
   mm/dd/yy and use a heading of “Ordered”.
2. SELECT order\_id, FORMAT(order\_date, 'MM,dd,yyyy') AS 'Ordered' FROM orders

WHERE order\_date BETWEEN 'May 1,2014' AND 'May 31,2014'

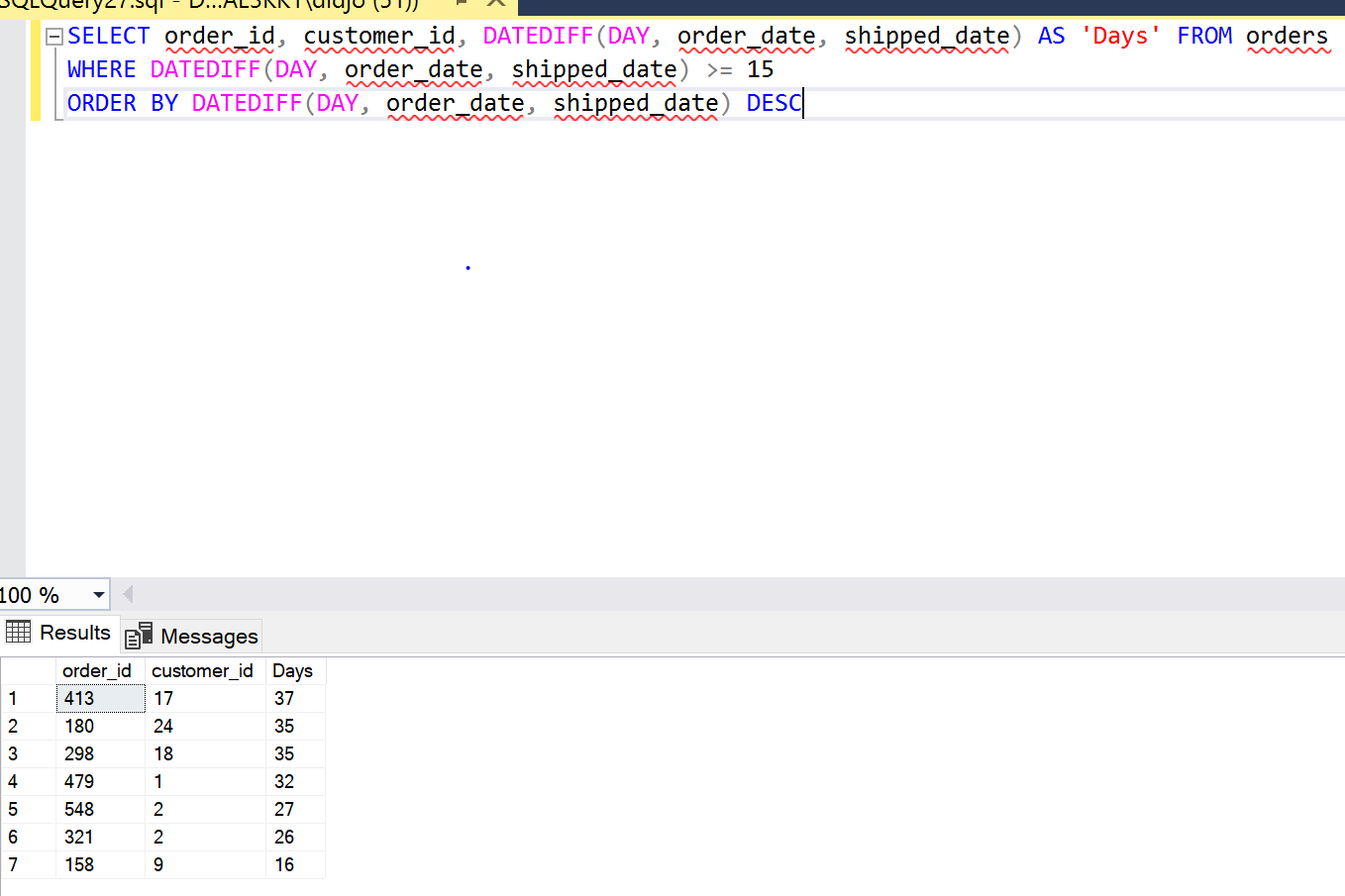
1. 
2. Using the SELECT function on order ids I formatted the orders into a “MM,dd,yy” format. Then with a BETWEEN clause I made so that the results displayed only orders from May 1st to May 31st.

# Query #12

1. Write a query which displays the order id, customer id, and the number of days between  
   the order date and the ship date (use the DATEDIFF function). Name this column “Days” and sort by highest to lowest number of days. Only display orders where this result is 15  
   days or more.
2. SELECT order\_id, customer\_id, DATEDIFF(DAY, order\_date, shipped\_date) AS 'Days' FROM orders

WHERE DATEDIFF(DAY, order\_date, shipped\_date) >= 15

ORDER BY DATEDIFF(DAY, order\_date, shipped\_date) DESC

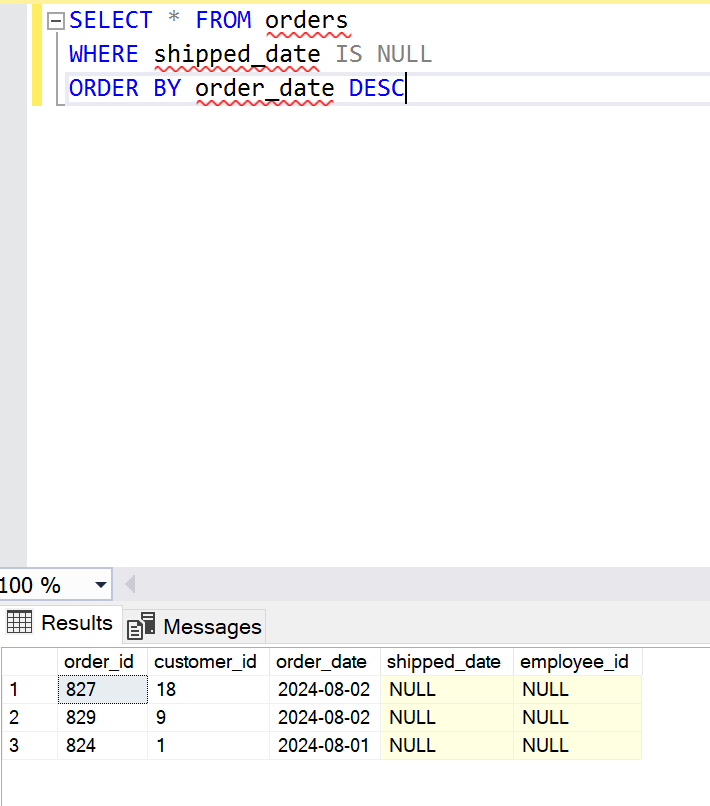
1. 
2. I started by selecting order id, customer id, and then used the DATEDIFF clause to make a function that would give me the number of days between the date ordered and shipped. Then I made that function into a column that would be titled “Days”. Also I made the column formatted to where it only shows me the number of days that were greater than or equal to 50. In addition, I formatted the column to display from highest to lowest.

# Query #13

1. Write a query which displays the order id, customer id, employee id, and order date for all orders that have NOT been shipped, sorted by order date with the most recent order at  
   the top.
2. SELECT \* FROM orders

WHERE shipped\_date IS NULL

ORDER BY order\_date DESC

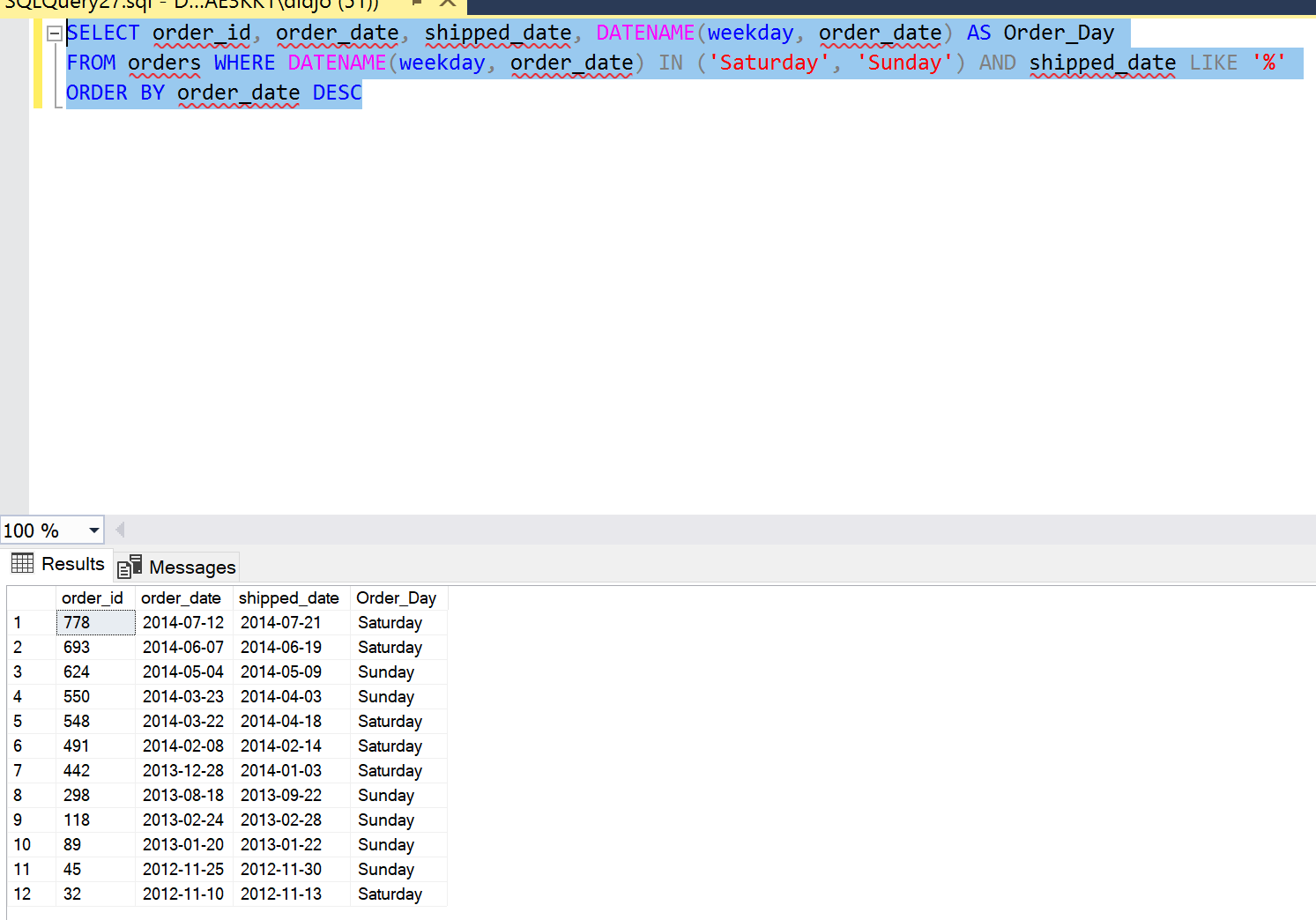
1. 
2. Using the SELECT \* FROM statement/clause I selected from the orders table and filtered the results to show only orders that weren’t shipped yet using the WHERE statement. Then with ORDER BY I sorted the dates into descending order.

# Query #14

1. The Marketing Department has requested a new report of shipped orders for which the  
   order was placed on either a Saturday or a Sunday. Write a query which displays the order id, order date, shipped date, along with a calculated column labeled “Order\_Day” showing the day of the week the order was placed (use the DAYNAME function). Only display orders that have shipped and were placed on a Saturday or Sunday. Sort by order date with most recent orders at the top.
2. SELECT order\_id, order\_date, shipped\_date, DATENAME(weekday, order\_date) AS Order\_Day

FROM orders WHERE DATENAME(weekday, order\_date) IN ('Saturday', 'Sunday')

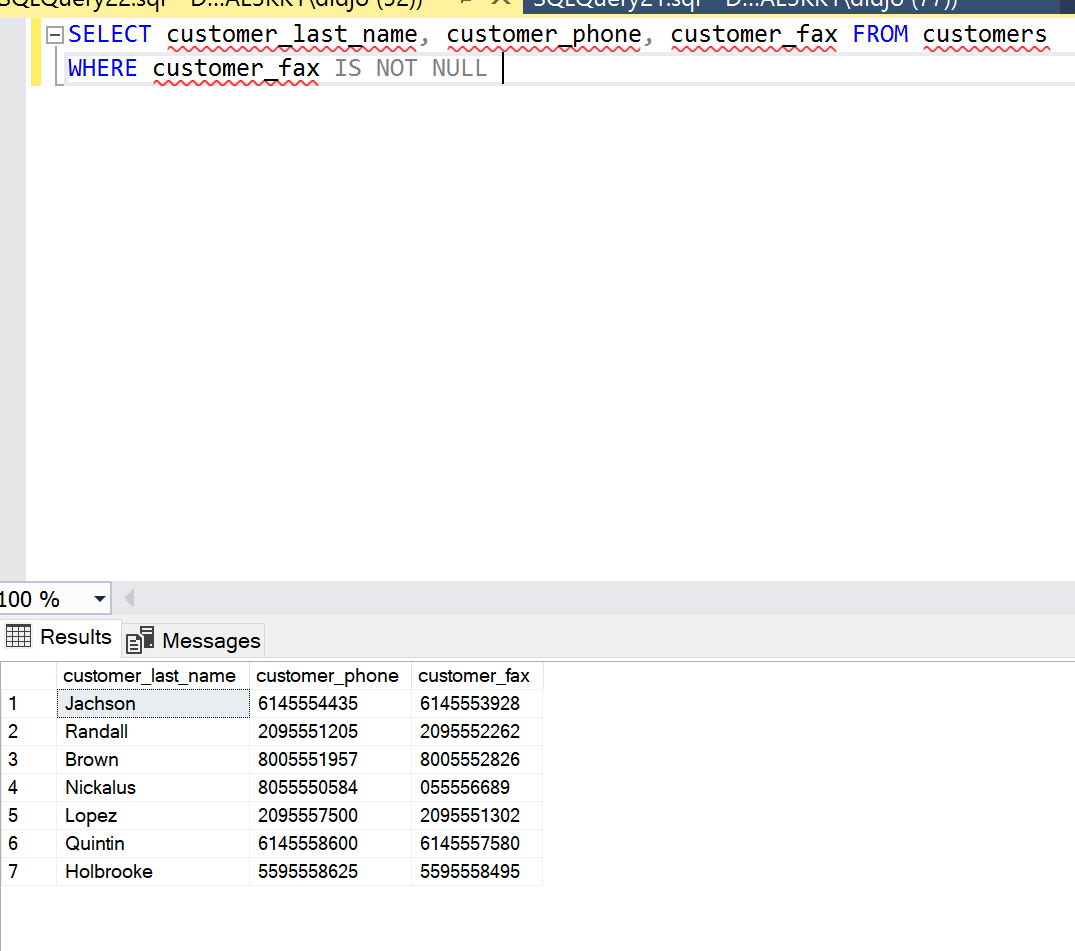
AND shipped\_date LIKE '%' ORDER BY order\_date DESC

1. 
2. Starting off I used the SELECT statement to select the order id, order date, and shipped date. Next using the DATENAME clause to get exact weekday of order shipments I used an AS clause to rename it “Order\_Day”. Then I displayed orders already shipped on Saturday and Sunday using the IN, AND, and LIKE clauses. Lastly, using ORDER BY I arranged the columns to sort the order date in descending order.

# Query #15

1. Write a query to display the customer’s last name, phone number, and fax number but  
   only display those customers that have a fax number.
2. SELECT customer\_last\_name, customer\_phone, customer\_fax, FROM customers

WHERE customer\_fax IS NOT NULL

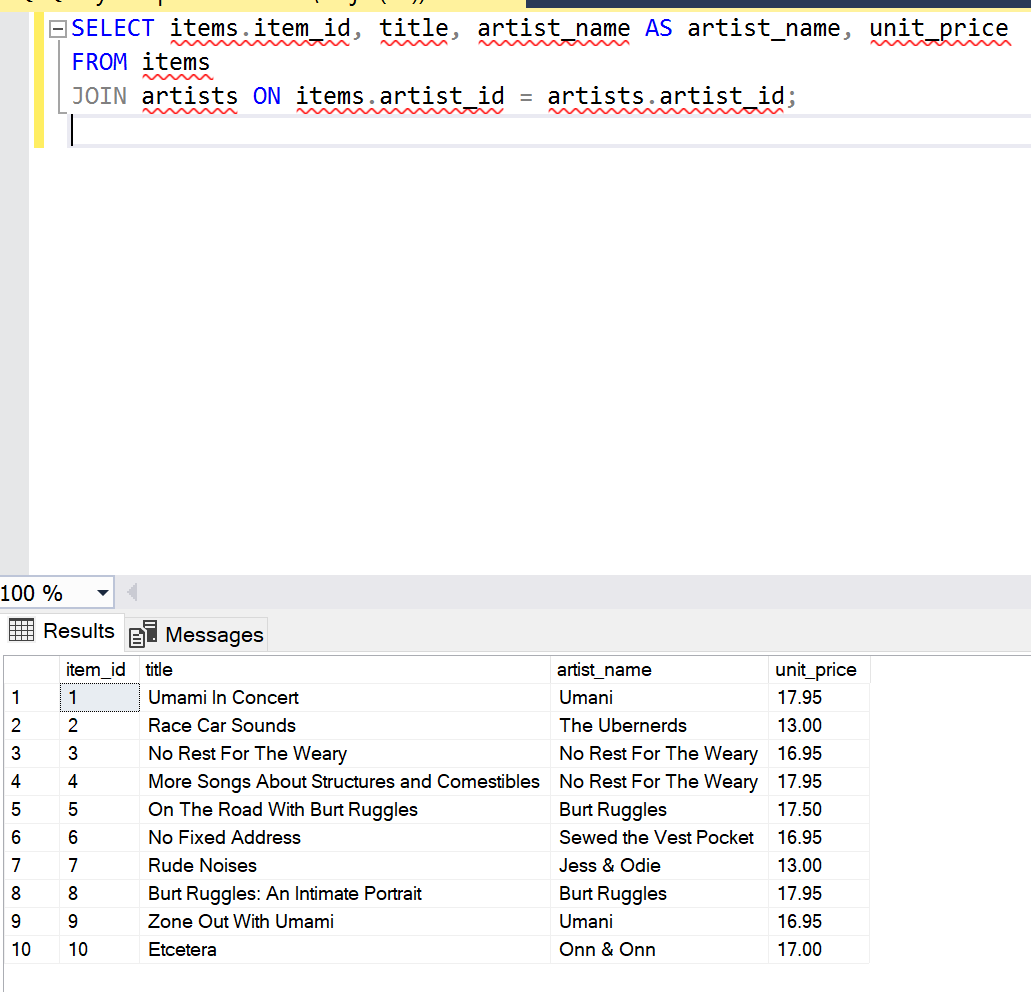
1. 
2. Using SELECT and FROM I selected from the customers table the customer last names, phone numbers, and fax numbers. I used the WHERE and NOT NULL to display only customers who have a fax number.

# Query #16

1. For each Item, retrieve the item id, title of the item, name of the artist, price of the item.  
   (use JOIN operation).
2. SELECT items.item\_id, title, artist\_name AS artist\_name, unit\_price

FROM items

JOIN artists ON items.artist\_id = artists.artist\_id;

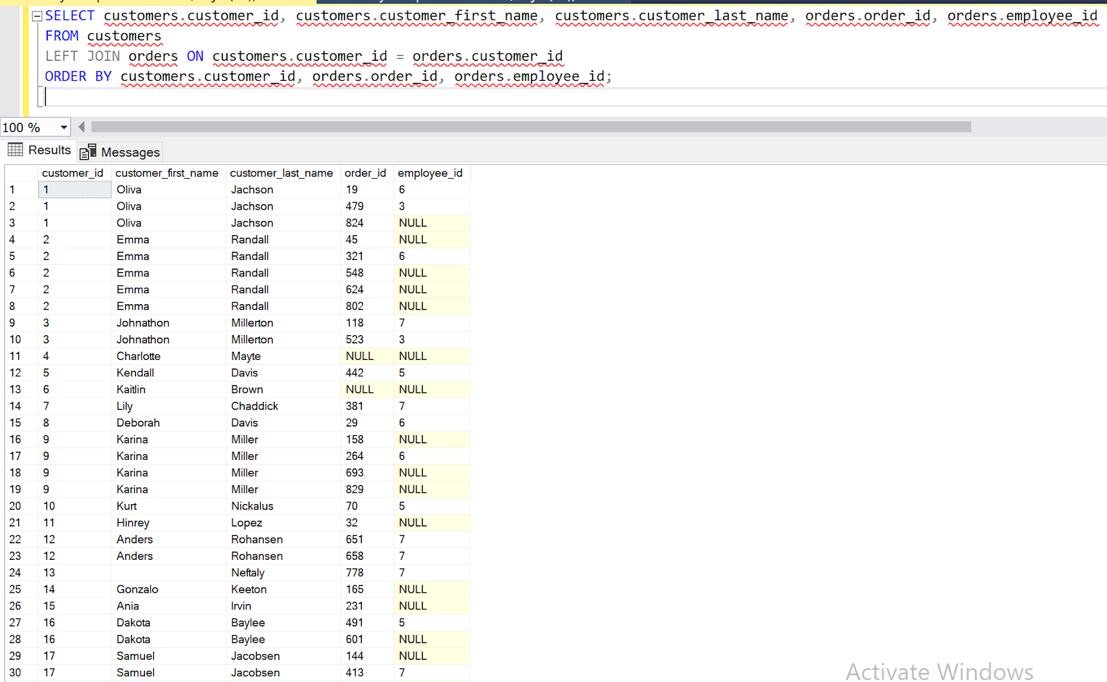
1. 
2. Using SELECT I selected items id, item titles, and artist name from the items table and displayed the columns artist name and price. I used the JOIN operator to help display the artist id from both the items and artist tables.

# Query #17

1. Write a query that displays the customer id, customer name, order id, and  
   employee id. Sort the results by customer id, order id, employee id. Use LEFT JOIN  
   operator.
2. SELECT customers.customer\_id, customers.customer\_first\_name, customers.customer\_last\_name, orders.order\_id, orders.employee\_id FROM customers

LEFT JOIN orders ON customers.customer\_id = orders.customer\_id

ORDER BY customers.customer\_id, orders.order\_id, orders.employee\_id;

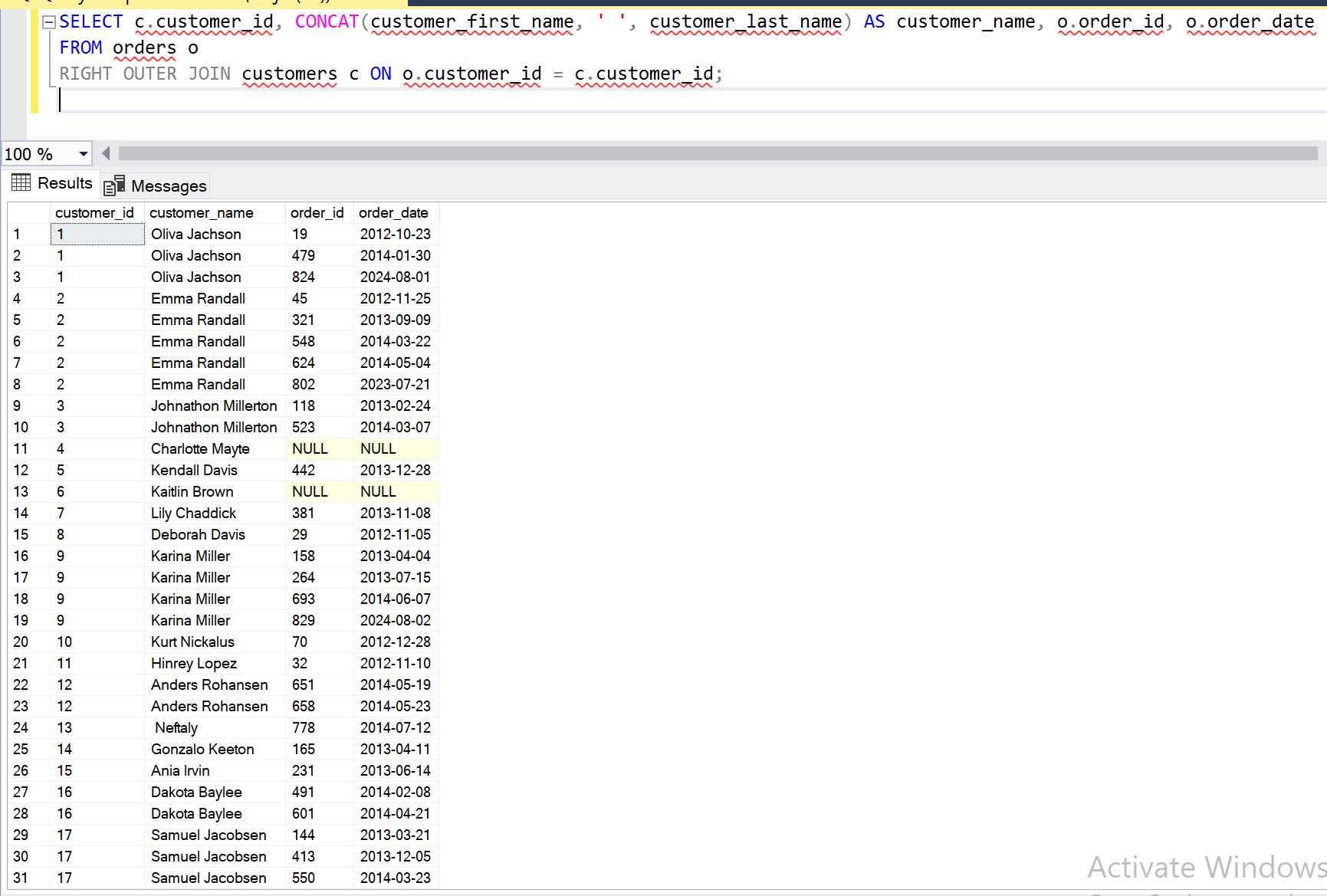
1. 
2. Using SELECT I selected certain items from the customer and order tables. Then I used a LEFT JOIN operator to ensure that all customers are included in the result, regardless of whether they have placed any orders. Doing so joined all the customers and orders on the customer id column. Then I sorted the results by customer id, order id, and employee id using ORDER BY.

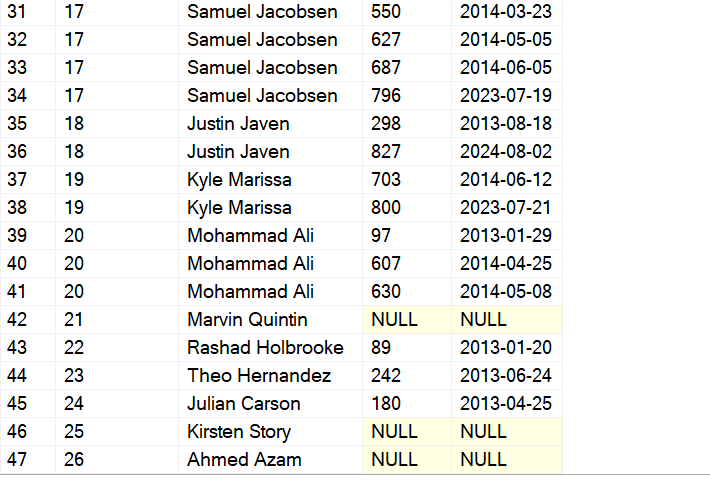
# Query #18

1. List customer identification number, customer name, order number and order date  
   for all orders listed in the order table. Include the order number, even if there is no  
   customer name, and identification number available. (RIGHT OUTER JOIN)
2. SELECT c.customer\_id, CONCAT(customer\_first\_name, ' ', customer\_last\_name) AS customer\_name, o.order\_id, o.order\_date

FROM orders o

RIGHT OUTER JOIN customers c ON o.customer\_id = c.customer\_id;

1. 



1. Selecting from the order table I obtained the customer id, customer name, order number and order date. I also used the CONCAT function to display the customer first and last name as one full name using the AS naming it “customer name”. I then used RIGHT OUTER JOIN to make sure that all orders from the orders table were included in the result, even if there is no corresponding customer information available in the customers table.

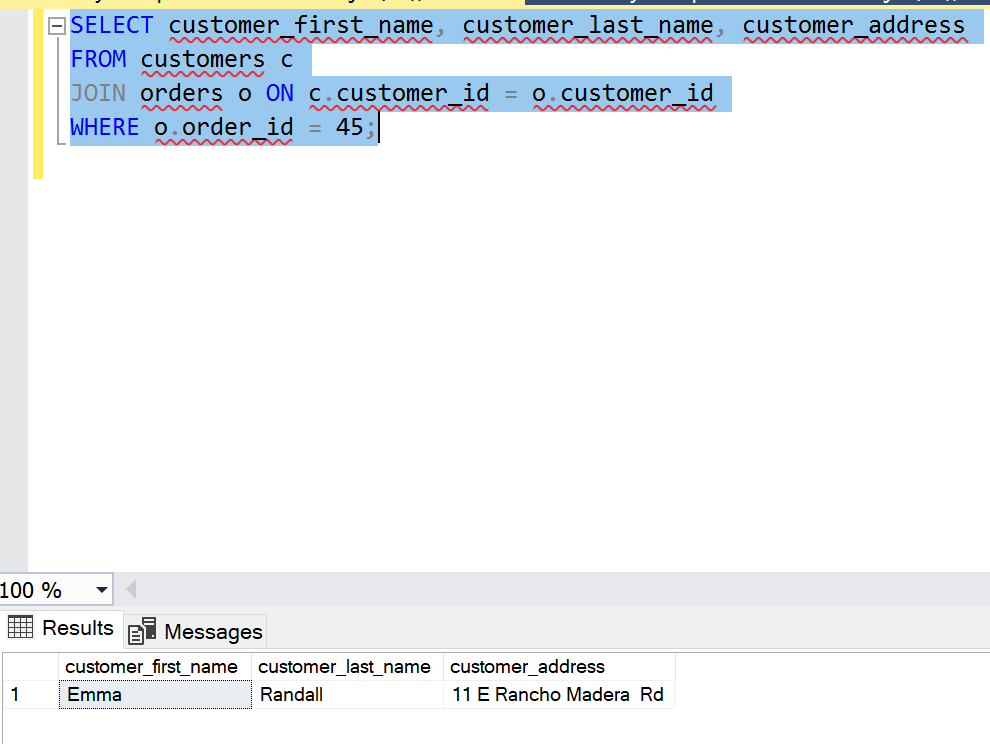
# Query #19

1. Write the name and address of the customer who placed order number 45.
2. SELECT customer\_first\_name, customer\_last\_name, customer\_address

FROM customers c

JOIN orders o ON c.customer\_id = o.customer\_id

WHERE o.order\_id = 45;

1. 
2. I selected the customer's first and last name as well as address from the customer table using SELECT and FROM. Then I used the JOIN operator to join the customer and order tables based on customer id to link the customer with the order. Lastly, I used WHERE to filter the order and display the details of the customer who placed order #45.

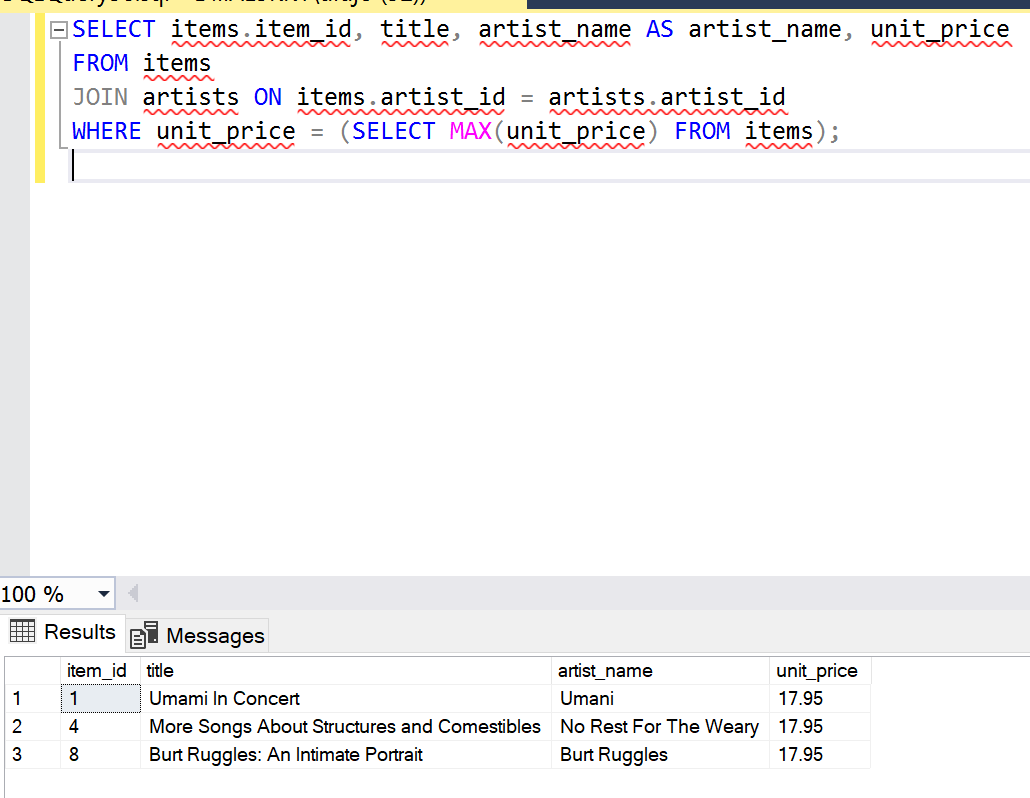
# Query #20

1. List the details about the item with the highest standard price.
2. SELECT items.item\_id, title, artist\_name AS artist\_name, unit\_price

FROM items

JOIN artists ON items.artist\_id = artists.artist\_id

WHERE unit\_price = (SELECT MAX(unit\_price) FROM items);

1. 
2. I selected the item id, title, and price from the items table, and name from the artists table. Then I joined the items and artists tables using JOIN based on the artist id column. Next I used a subquery to find the maximum price from the items table using MAX(price). The WHERE clause ensured that only items with the highest standard price were selected.

# Query #21

1. Create a statement to insert a new record into the items table with the following values:

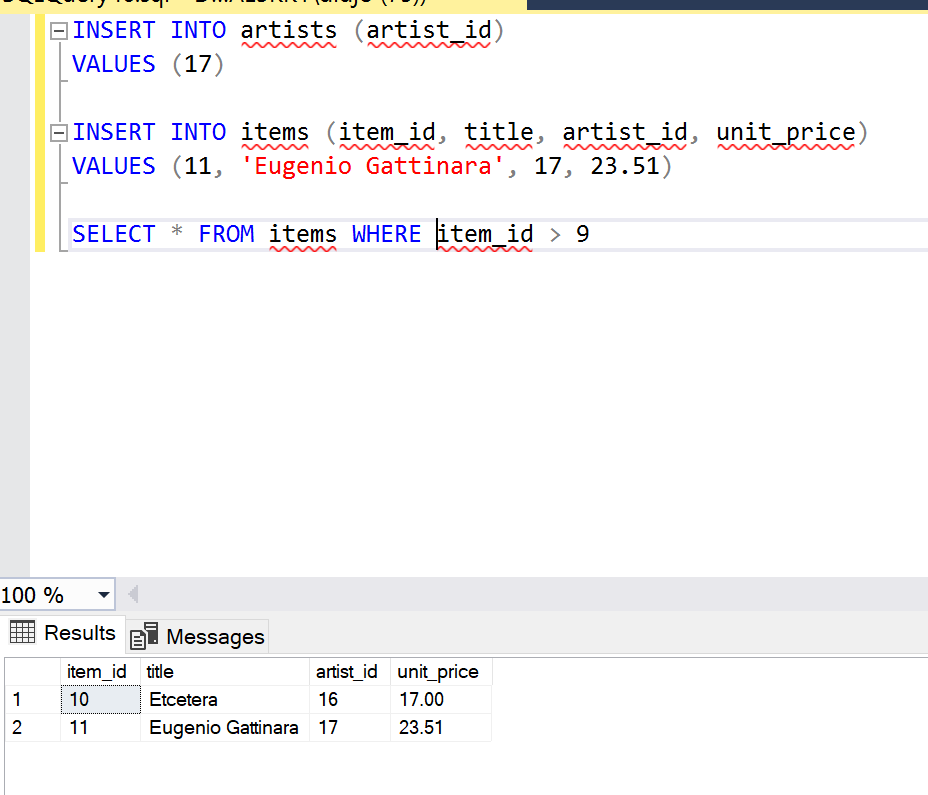
|  |  |
| --- | --- |
| item\_id: | 11 |
| title: | Eugenio Gattinara  https://www.quora.com/profile/Eugenio-Gattinara1 |
| Artist\_id: | 17 |
| unit\_price: | 23.5 |

Show your INSERT statement along with the results of the following SELECT query to verify  
that the insert worked correctly.  
select \* from items where item\_id > 9;

1. INSERT INTO items (item\_id, title, artist\_id, unit\_price)

VALUES (11, 'Eugenio Gattinara', 17, 23.51)

SELECT \* FROM items WHERE item\_id > 9

1. 
2. First, I started with using the INSERT statement 2 different times running each simultaneously. The reason for that was to match the same value from the items table to artists table, because artist id was a primary key. Then I used the SELECT FROM statement to prove that the INSERT statement worked.

# Query #22

1. Create a statement to update the record inserted in the previous step to change the unit  
   price of this item to $19.05.

|  |  |
| --- | --- |
| item\_id: | 11 |
| title: | Eugenio Gattinara |
| artist | 17 |
| unit\_price: | 19.05 |

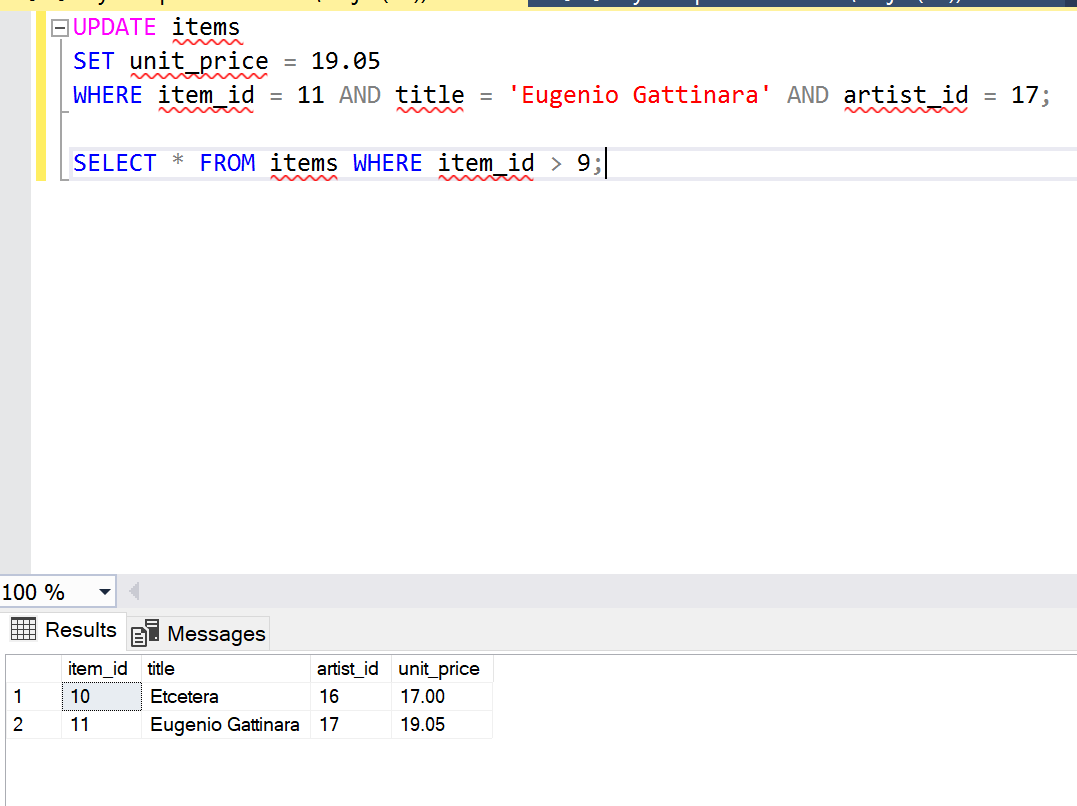
Show your UPDATE statement along with the results of the following SELECT query to verify  
that the insert worked correctly.  
select \* from items where item\_id > 9;

1. UPDATE items

SET unit\_price = 19.05

WHERE item\_id = 11 AND title = 'Eugenio Gattinara' AND artist\_id = 17;

SELECT \* FROM items WHERE item\_id > 9;

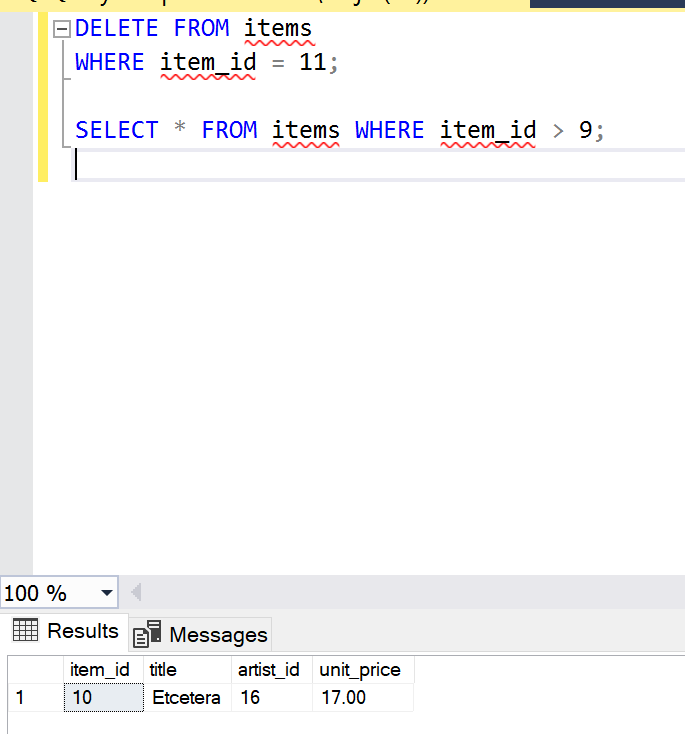
1. 
2. I began by updating the unit price to 19.05, but I also made sure to use WHERE and AND to ensure the only column updated was the one containing the title ‘Eugenio Gattinara’.

# Query #23

1. Create a statement to delete the entire record that was inserted and then updated in the  
   previous steps.  
   Show your DELETE statement along with the results of the following SELECT query to verify that the insert worked correctly.  
   select \* from items where item\_id > 9;
2. DELETE FROM items

WHERE item\_id = 11;

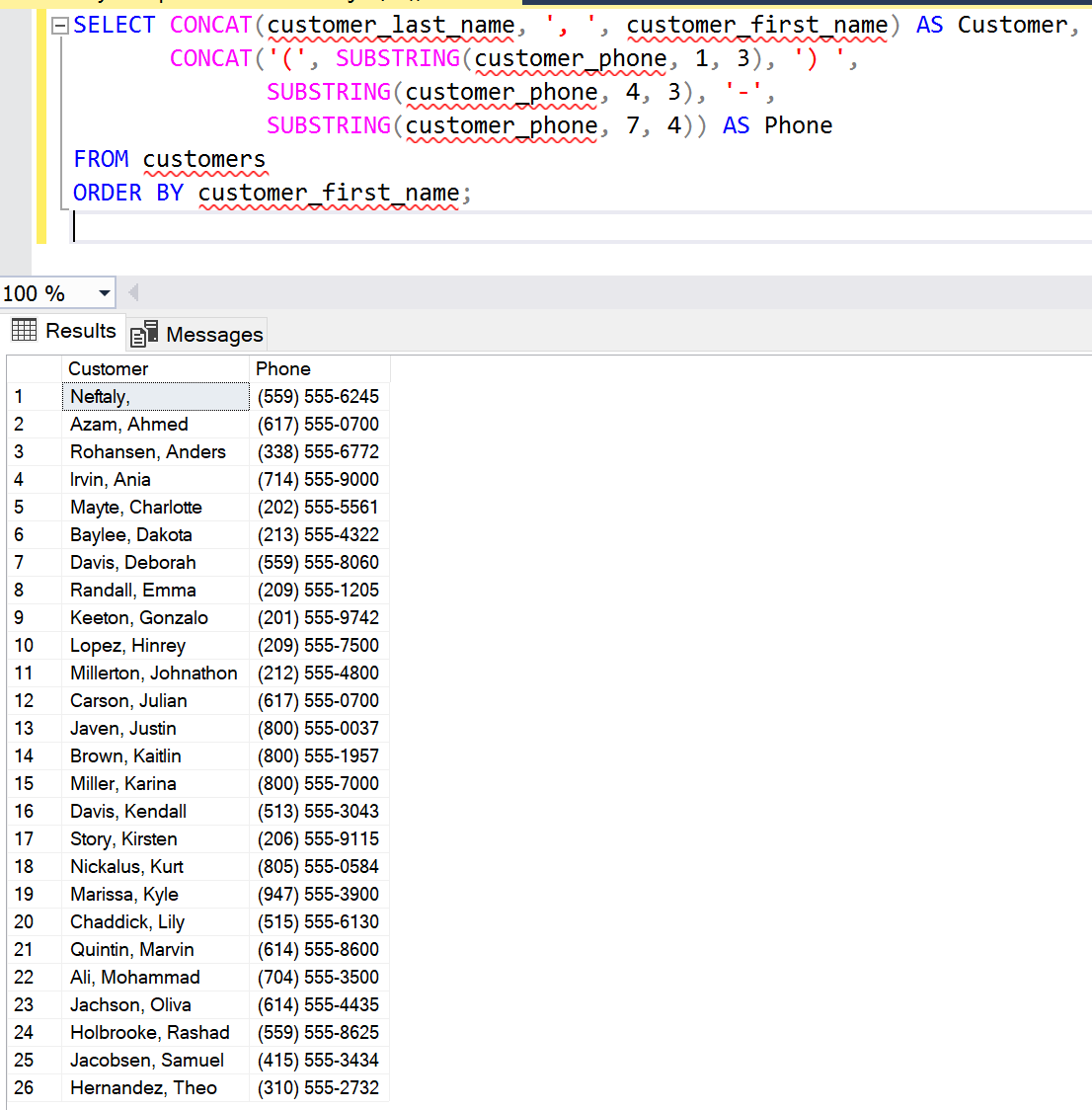
SELECT \* FROM items WHERE item\_id > 9;

1. 
2. I started with the DELETE clause to delete a column from the items table using WHERE to specify which column. The SELECT and FROM clauses were to pull up the results and ensure that the column or row was deleted.

# Query #24

1. Using the SUBSTRING and CONCAT functions, write a query to display each customer  
   name as a single field in the format “Jones, Tom” with a heading of Customer along with  
   the customer\_phone field in a nicely formatted calculated column named Phone. For  
   example, a record containing the customer\_phone value 9095595443 would be output  
   with parentheses, spaces, and hyphens, like this: (909) 559-5443. Sort by first name.
2. SELECT CONCAT(customer\_last\_name, ', ', customer\_first\_name) AS Customer, CONCAT('(', SUBSTRING(customer\_phone, 1, 3), ') ', SUBSTRING(customer\_phone, 4, 3), '-', SUBSTRING(customer\_phone, 7, 4)) AS Phone FROM customers

ORDER BY customer\_first\_name;

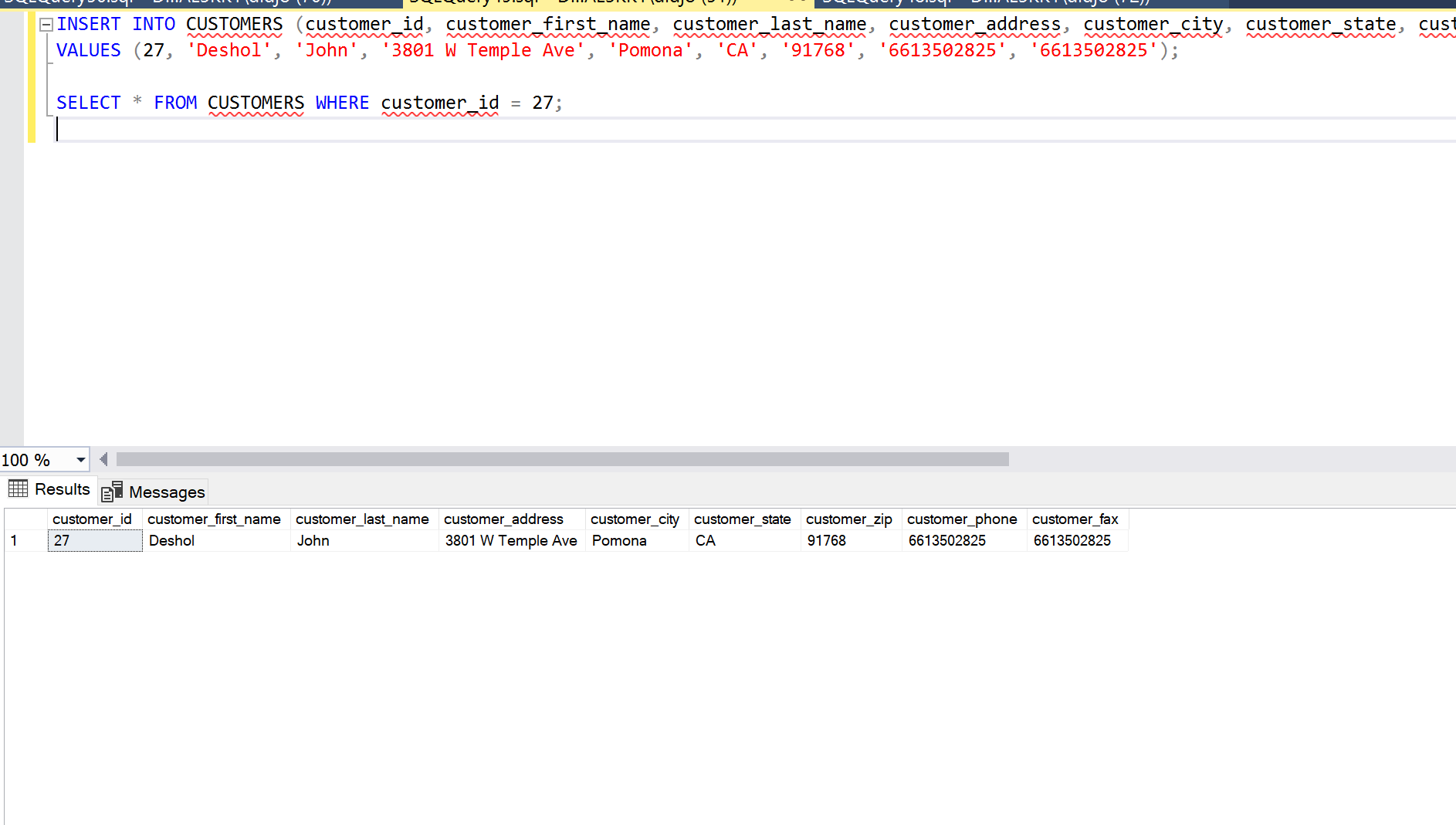
1. 
2. I started by using the CONCAT function to combine the customer’s first and last name into a column titled “customer”. In the same function I added a comma with quotation marks to have SQL display a comma in between the first and last name. Next I used SUBSTRING to extract substrings from the customer phone field to format it as (XXX) XXX-XXXX. Lastly, I used ORDER BY to sort the results by the first name of the customers.

# Query #25

1. Create a statement to insert a new record with your values: your customer id, first  
   name, last name, address, city, state, zip code and fax number. Show your INSERT statement along with the results of the following SELECT query to verify that the insert worked correctly.  
   select \* from Customer\_T where Customer ID = 27;
2. INSERT INTO CUSTOMERS (customer\_id, customer\_first\_name, customer\_last\_name, customer\_address, customer\_city, customer\_state, customer\_zip, customer\_phone, customer\_fax)

VALUES (27, 'Deshol', 'John', '3801 W Temple Ave', 'Pomona', 'CA', '91768', '6613502825', '6613502825');

SELECT \* FROM CUSTOMERS WHERE customer\_id = 27;

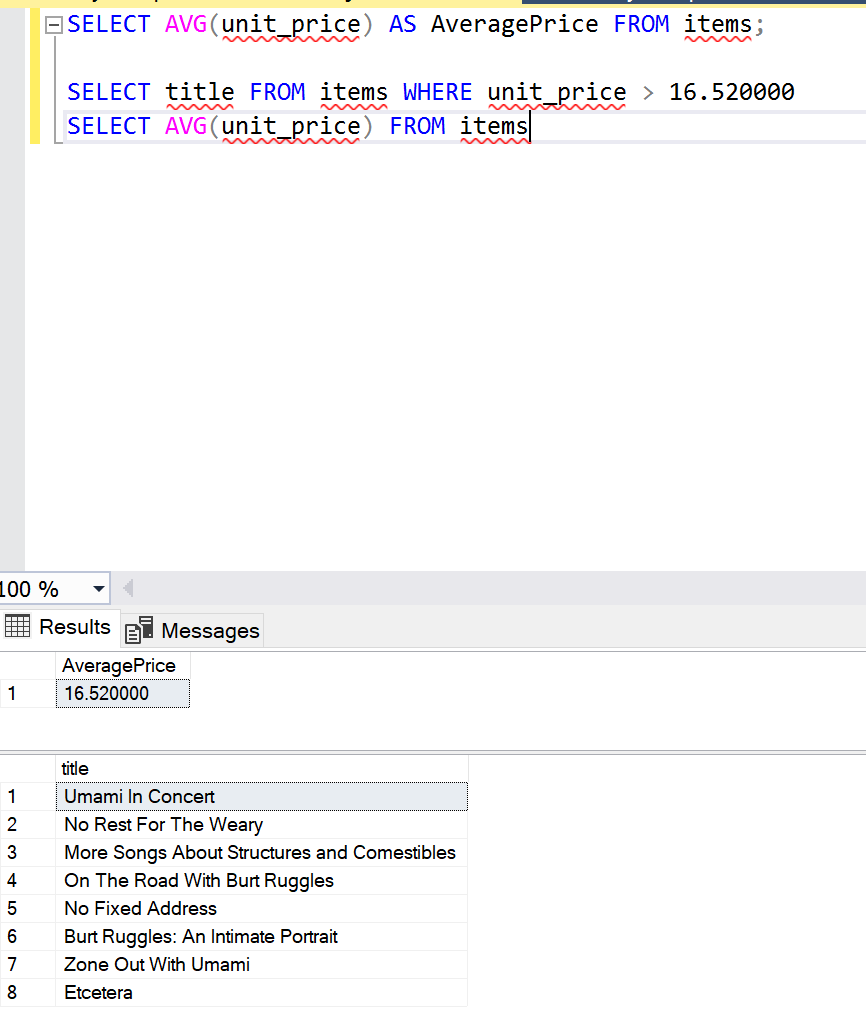
1. 
2. I used the INSERT clause to let SQL know that was inserting into the customer table. The VALUES clause was to put input my information into the table as ordered above. Then I used SELECT \* FROM to verify and display my information where the customer id = 27.

# Query #26

1. Creates a view that selects every title in the "item" table with a price higher  
   than the average price:
2. SELECT AVG(unit\_price) AS AveragePrice FROM items;

SELECT title FROM items WHERE unit\_price > 16.520000

SELECT AVG(unit\_price) FROM items

1. 
2. I used the SELECT AVG clause to identify the average unit or item price. Once identified I used the SELECT clause to get all the titles and used WHERE to only display titles that were above the average of 16.520000.

# Query #27

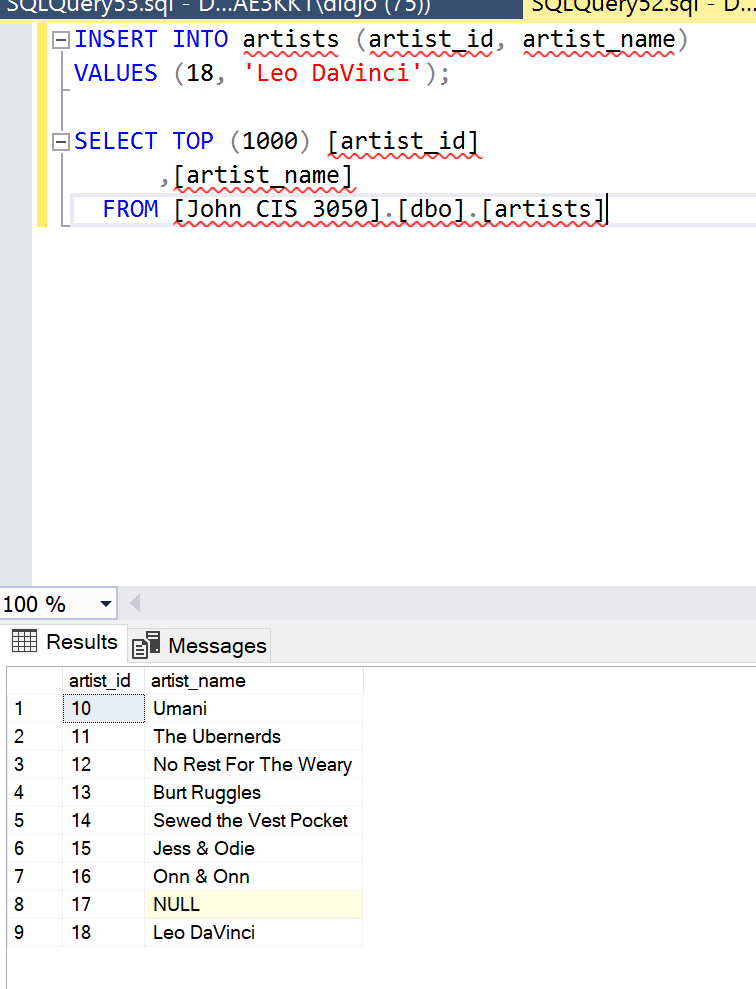
1. Explain the cardinality from the employees-to-employees table.

2. The term "cardinality" in the context of a database refers to the relationship between the rows of two tables. It describes how many rows in one table correspond to how many rows in another table through a relationship. Mentioning "employees-to-employees" as a table, it suggests a self-join scenario where you have an "employees" table and you're joining it to itself. In this case, the cardinality from the employees-to-employees table could vary based on the specific relationship you're establishing through the join. For example, since cardinality also states an employee id can have zero or one employee name, its possible for employees to share the same name but not the same employee id. This is because the employee id acts as the primary key.

# Query #28

1. Insert a new artist (artist id and artist name)
2. INSERT INTO artists (artist\_id, artist\_name)

VALUES (18, 'Leo DaVinci');

1. 
2. I used the INSERT clause/statement to indicate to SQL that I wanted to add a new artist row. The VALUES clause was to input the new artist id and name. The statemtents following were to ensure that the new artist was added into the table.

# Results and Discussion

After completing my report it is comprised of written portions and twenty-eight different queries. Each query contained functions and clauses that I had to use to complete it. In addition to completing the query and showing my reuslts I was tasked with 3 other parts given to me by my professor. The parts included showing the question or query instructions, typing in my SQL statement, and explaining said query performed. Obtaining the results of some queries were challenging due to the fact of unknown clauses and statements. However, after playing with SQL and reading the textbook material, I developed a better understanding.

# Lessons Learned

While completing this project I learned a couple of lessons. The first one is Microsoft SQL is not that hard once you get the hang of it. I was skeptical about learning how to use MS SQL, but after spending time on the software, I soon realized. Another being its harder to be very careful when inputting in clauses, statements, and or functions because the smallest error will cause the query to not be readable. Finally, it taught me a great deal of patience. When you’re inputting in queries it can be frustrating trying to figure out why it won’t execute. So being patient with the software and paying close attention to detail makes using Microsoft SQL ten times easier.

# Conclusion

In conclusion developing a database as stated above can be very challenging. Although challenging upon completion of the project overcoming the challenges felt rewarding. I personally think that I got a deeper understanding of Microsoft SQL. Developing a database that included logical and physical designs while delivering a meaningful report allowed me to deep dive into SQL concepts. There were many times throughout the project where I was learning a clause or function and the purpose of it began to relate to what I was learning in class. From each table and statement used I was able to piece together why certain clauses are important for certain statements or queries. Developing the database reminded me of the ERwin project and showed how databases are around us every day and how important they are for information.

# References

Hoffer, Jeffrey A., et al. *Modern Database Management*. Pearson Education Limited, 2016.

*SQL Constraints*, www.w3schools.com/sql/sql\_constraints.asp. Accessed 29 Mar. 2024.

*SQL Select Statement*, www.w3schools.com/sql/sql\_select.asp. Accessed 28 Mar. 2024.