Cal Poly Pomona

Safa Alasady

Database Design and Implementation of Microsoft SQL

3/31/2024

CIS 3050.05

Professor Ahmed Azam

Spring 2024

Table of Contents

Statement of Academic Honesty

My name is: Safa Alasady, 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: Safa Alasady Date: 3/31/2024

Signature: 

Introduction

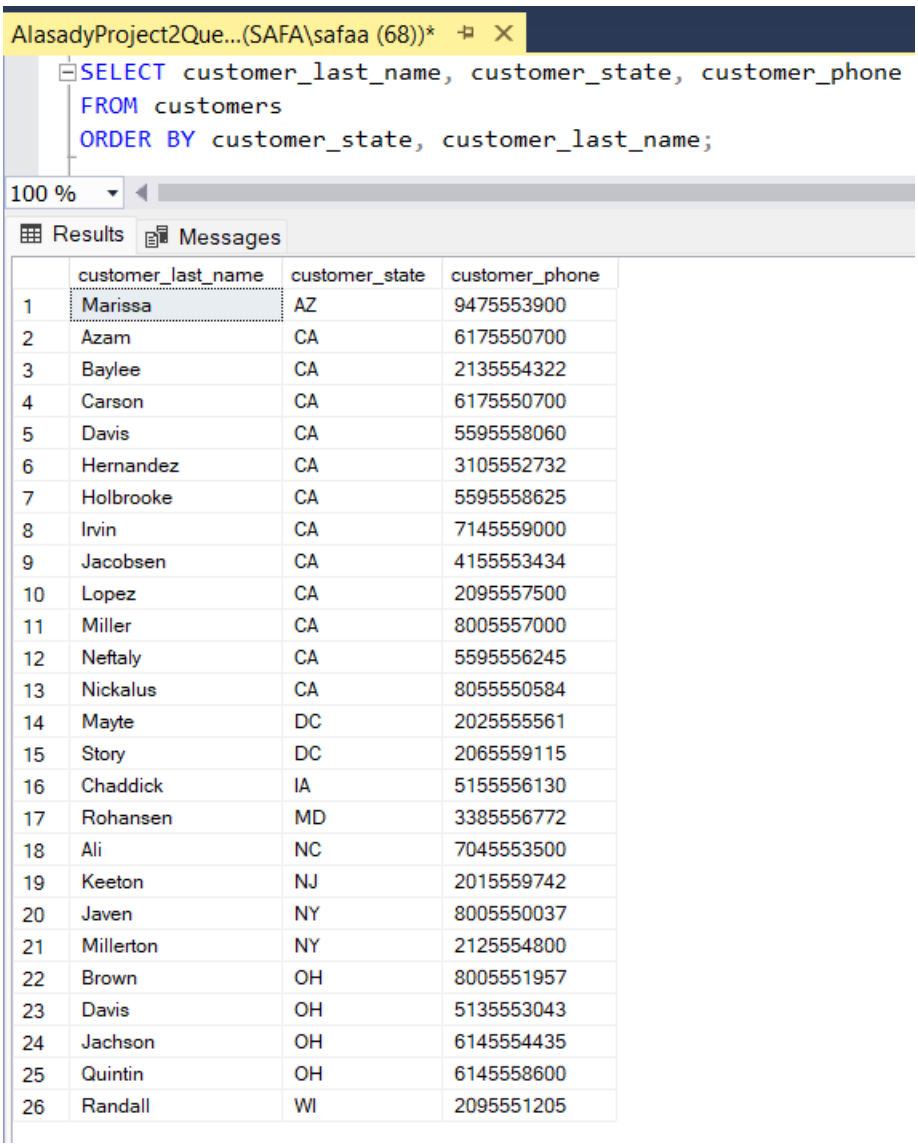
SQL is the standard language that is particularly important for relational database management systems. Tools in SQL allow people to create a database and be able to manipulate data within these databases. SQL is recognized as an international standard by the International Organization for Standardization and by the American National Standards Institute, and it has been accepted as a U.S. standard. Multiple versions of SQL have been released beginning in 1986 and most recently, in 2016. SQL is part of both mainframe and personal computers. The purpose of this project is to learn the important concepts of logical data modeling, physical data modeling, and the process of designing databases. It is through this project that we learn how to design, develop, and show how databases work with business rules/specifications that we create. Being able to create significant reports from tables in the database is a result of using SQL.

Project Description

Based on the tables provided in the project instructions, students are expected to implement business specifications and produce a report of all queries to demonstrate the capability and functionality of the database. I, as a student, am responsible for designing, developing, and showing how the database works. I will also be able to transform conceptual entity relationship diagrams into a real-life example of the database on Microsoft SQL Server. To be able to show and complete the queries, students have to use important SQL clauses like SELECT, FROM, WHERE, and many more. These clauses also all together show how to retrieve specific data from tables in a database. Through this project, students will be able to learn database design and its implementations.

1. Query #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.



AlasadyProject2Que...(SAFA\safaa (68))*

```
SELECT customer_last_name, customer_state, customer_phone
FROM customers
ORDER BY customer_state, customer_last_name;
```

100 %

Results | Messages

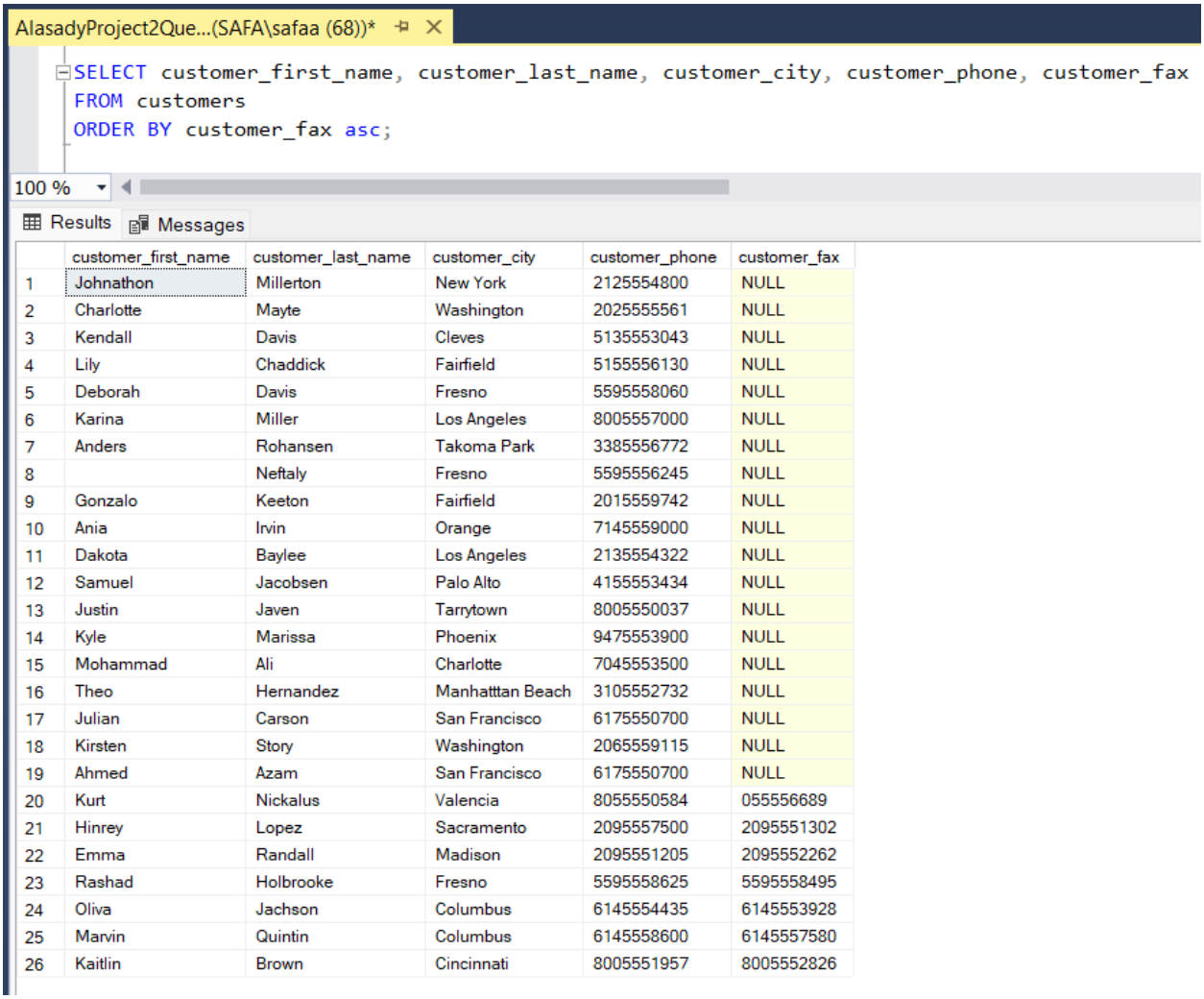
	customer_last_name	customer_state	customer_phone
1	Marissa	AZ	9475553900
2	Azam	CA	6175550700
3	Baylee	CA	2135554322
4	Carson	CA	6175550700
5	Davis	CA	5595558060
6	Hernandez	CA	3105552732
7	Holbrooke	CA	5595558625
8	Irvin	CA	7145559000
9	Jacobsen	CA	4155553434
10	Lopez	CA	2095557500
11	Miller	CA	8005557000
12	Neftaly	CA	5595556245
13	Nickalus	CA	8055550584
14	Mayte	DC	2025555561
15	Story	DC	2065559115
16	Chaddick	IA	5155556130
17	Rohansen	MD	3385556772
18	Ali	NC	7045553500
19	Keeton	NJ	2015559742
20	Javen	NY	8005550037
21	Millerton	NY	2125554800
22	Brown	OH	8005551957
23	Davis	OH	5135553043
24	Jachson	OH	6145554435
25	Quintin	OH	6145558600
26	Randall	WI	2095551205

This query selects customer_last_name, customer_state, and customer_phone from the customers table and sorts the results in ascending order by customer_state and customer_phone.

1. Query #2: 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;

3.



AlasadyProject2Que...(SAFA\safaa (68))*

```
SELECT customer_first_name, customer_last_name, customer_city, customer_phone, customer_fax
FROM customers
ORDER BY customer_fax asc;
```

100 %

Results | Messages

	customer_first_name	customer_last_name	customer_city	customer_phone	customer_fax
1	Johnathon	Millerton	New York	2125554800	NULL
2	Charlotte	Mayte	Washington	2025555561	NULL
3	Kendall	Davis	Cleves	5135553043	NULL
4	Lily	Chaddick	Fairfield	5155556130	NULL
5	Deborah	Davis	Fresno	5595558060	NULL
6	Karina	Miller	Los Angeles	8005557000	NULL
7	Anders	Rohansen	Takoma Park	3385556772	NULL
8		Neftaly	Fresno	5595556245	NULL
9	Gonzalo	Keeton	Fairfield	2015559742	NULL
10	Ania	Irvin	Orange	7145559000	NULL
11	Dakota	Baylee	Los Angeles	2135554322	NULL
12	Samuel	Jacobsen	Palo Alto	4155553434	NULL
13	Justin	Javen	Tarrytown	8005550037	NULL
14	Kyle	Marissa	Phoenix	9475553900	NULL
15	Mohammad	Ali	Charlotte	7045553500	NULL
16	Theo	Hernandez	Manhatttan Beach	3105552732	NULL
17	Julian	Carson	San Francisco	6175550700	NULL
18	Kirsten	Story	Washington	2065559115	NULL
19	Ahmed	Azam	San Francisco	6175550700	NULL
20	Kurt	Nickalus	Valencia	8055550584	055556689
21	Hinrey	Lopez	Sacramento	2095557500	2095551302
22	Emma	Randall	Madison	2095551205	2095552262
23	Rashad	Holbrooke	Fresno	5595558625	5595558495
24	Oliva	Jachson	Columbus	6145554435	6145553928
25	Marvin	Quintin	Columbus	6145558600	6145557580
26	Kaitlin	Brown	Cincinnati	8005551957	8005552826

This query selects customer_first_name, customer_last_name, customer_city, customer_phone, and customer_fax to organize with five columns from the customers table and order the results by customer_fax in ascending order.

1. Query #3: Write a query that displays all the customers from New York or New Jersey in the “Customers” table.

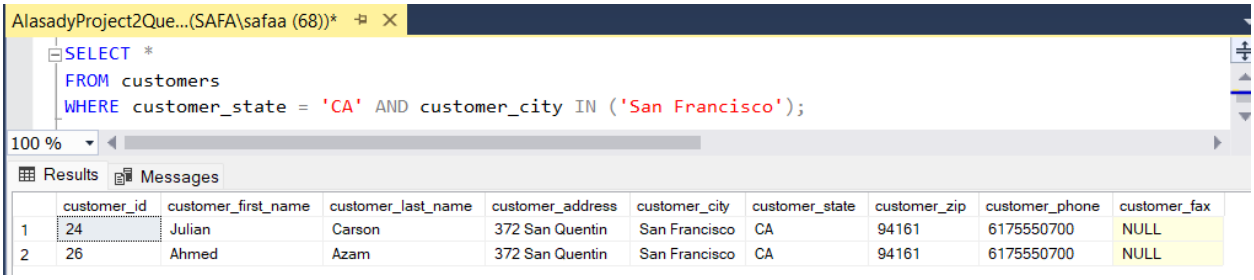
2. SELECT *
   FROM customers
   WHERE customer_city IN ('New York', 'New Jersey');

3. 

This query selects all columns from the customers table with nine columns to only show customers that live in customer_city of “New York” and “New Jersey”.

1. Query #4: 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 IN ('San Francisco');

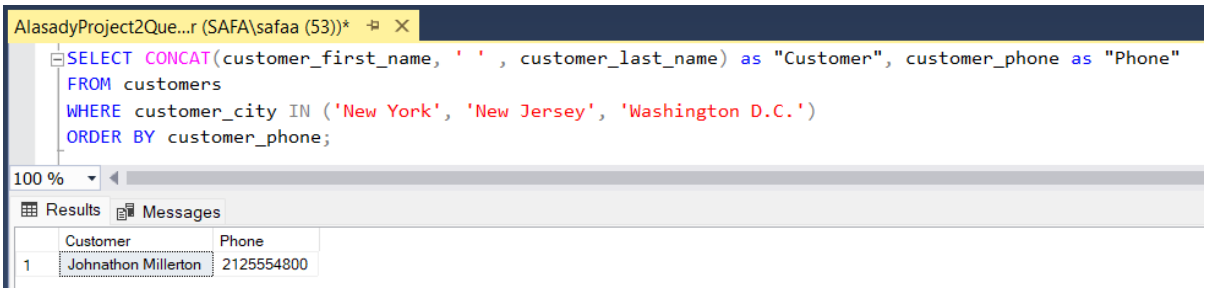
3. 

	customer_id	customer_first_name	customer_last_name	customer_address	customer_city	customer_state	customer_zip	customer_phone	customer_fax
1	24	Julian	Carson	372 San Quentin	San Francisco	CA	94161	6175550700	NULL
2	26	Ahmed	Azam	372 San Quentin	San Francisco	CA	94161	6175550700	NULL

This query selects all columns from the customers table to show information about customers that only live in the state of “CA” and the city of “San Francisco”.

1. Query #5: 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_city IN ('New York', 'New Jersey', 'Washington D.C.')
   ORDER BY customer_phone;

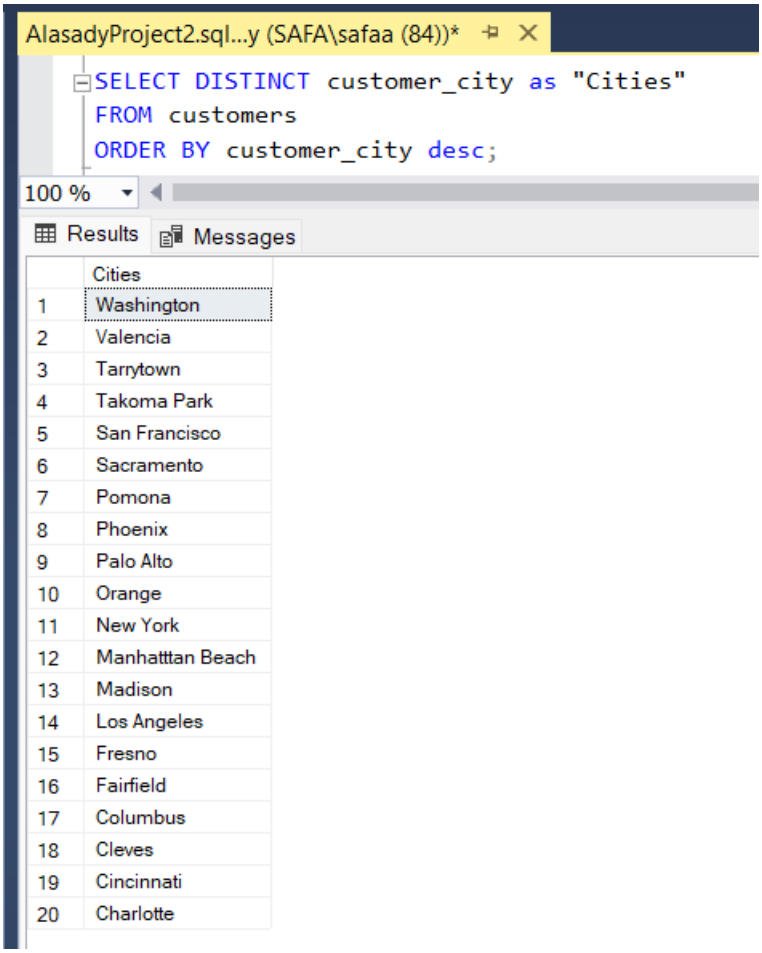
3. 

This query selects customer_first_name and customer_last_name and concatenates both columns from the customers table to show the customer’s first and last name together only as one column. It also selects customer_phone and only chooses customers that could live in “New York”, “New Jersey” or “‘Washington D.C.”. The table is sorted by customer_phone. The column customer_phone is labeled as “Phone”.

1. Query #6: 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;

3.



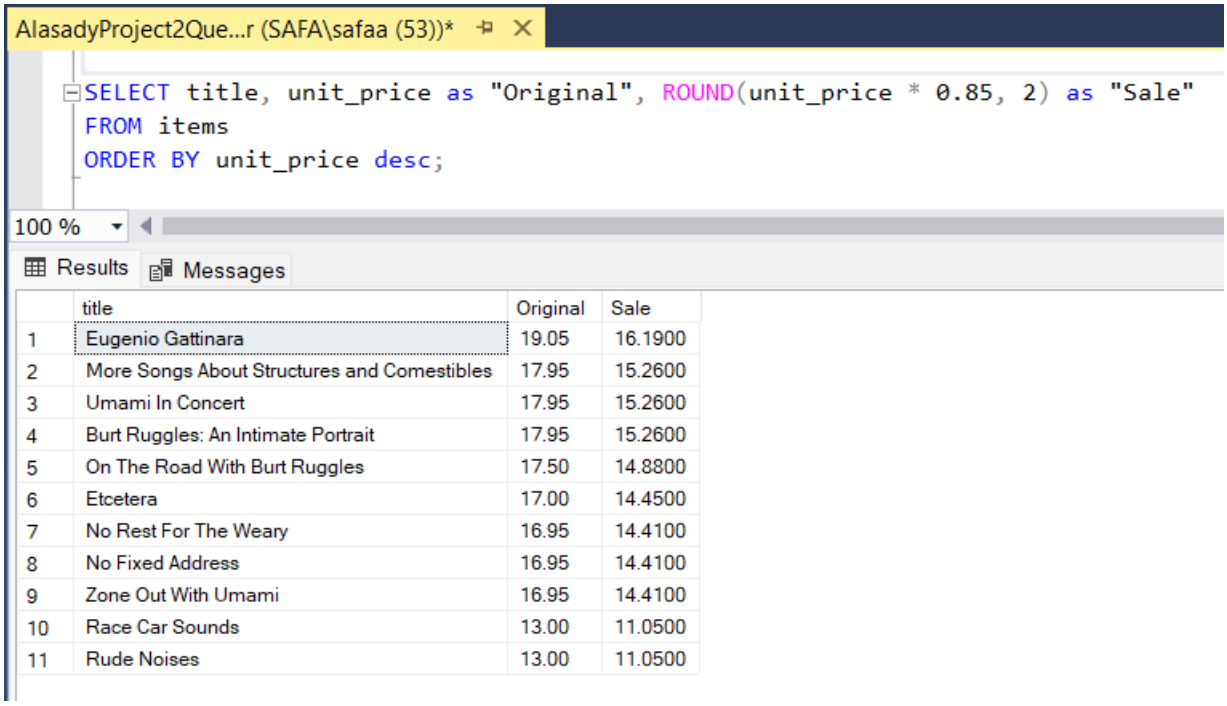
	Cities
1	Washington
2	Valencia
3	Tarrytown
4	Takoma Park
5	San Francisco
6	Sacramento
7	Pomona
8	Phoenix
9	Palo Alto
10	Orange
11	New York
12	Manhatttan Beach
13	Madison
14	Los Angeles
15	Fresno
16	Fairfield
17	Columbus
18	Cleves
19	Cincinnati
20	Charlotte

This query only selects customer_city from the customers table and makes an alias for the same column as “Cities”. The results are sorted by customer_city in descending order.

1. Query #7: 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 * 0.85, 2) as "Sale"
   FROM items
   ORDER BY unit_price desc;

3.

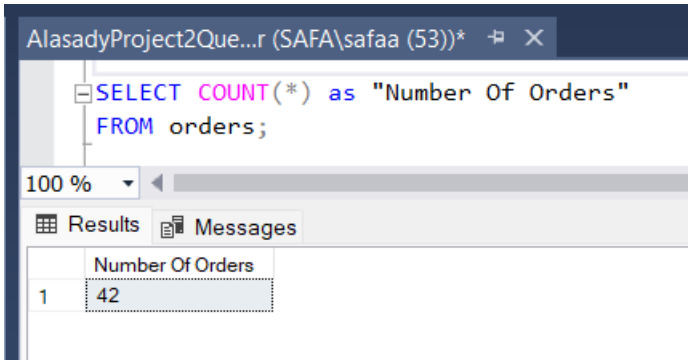


	title	Original	Sale
1	Eugenio Gattinara	19.05	16.1900
2	More Songs About Structures and Comestibles	17.95	15.2600
3	Umami In Concert	17.95	15.2600
4	Burt Ruggles: An Intimate Portrait	17.95	15.2600
5	On The Road With Burt Ruggles	17.50	14.8800
6	Etcetera	17.00	14.4500
7	No Rest For The Weary	16.95	14.4100
8	No Fixed Address	16.95	14.4100
9	Zone Out With Umami	16.95	14.4100
10	Race Car Sounds	13.00	11.0500
11	Rude Noises	13.00	11.0500

This query selects two columns from the items table, title, and unit_price, and also adds a new column where the unit price is given a 15 percent discount by the ROUND function with an alias of “Sale”. The results are sorted by unit_price in descending order.

1. Query #8: Write a query that displays the number of orders.

2. SELECT COUNT(*) as "Number Of Orders"
   FROM orders;

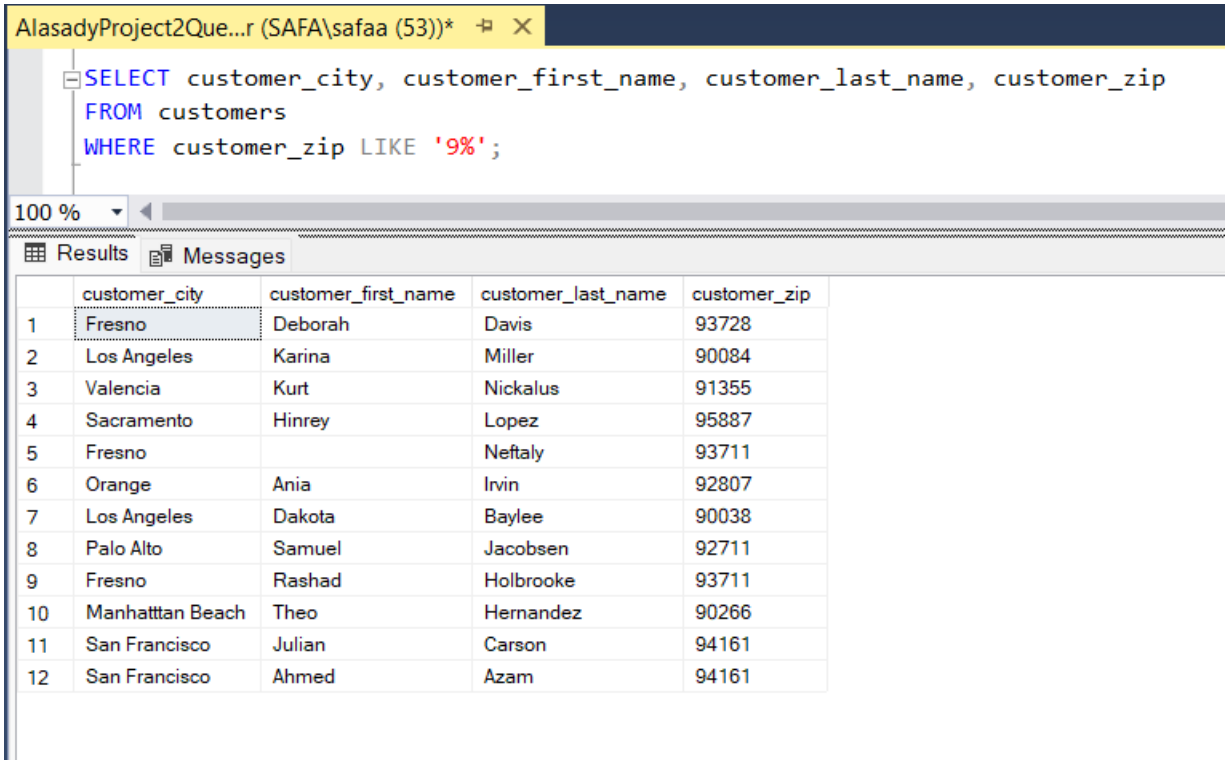
3. 

This query used the COUNT function to count the total number of orders from the orders table, which is also using an alias of “Number Of Orders”. It is also noted that the asterisk is used in the COUNT function to select all orders.

1. Query #9: 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%';

3.

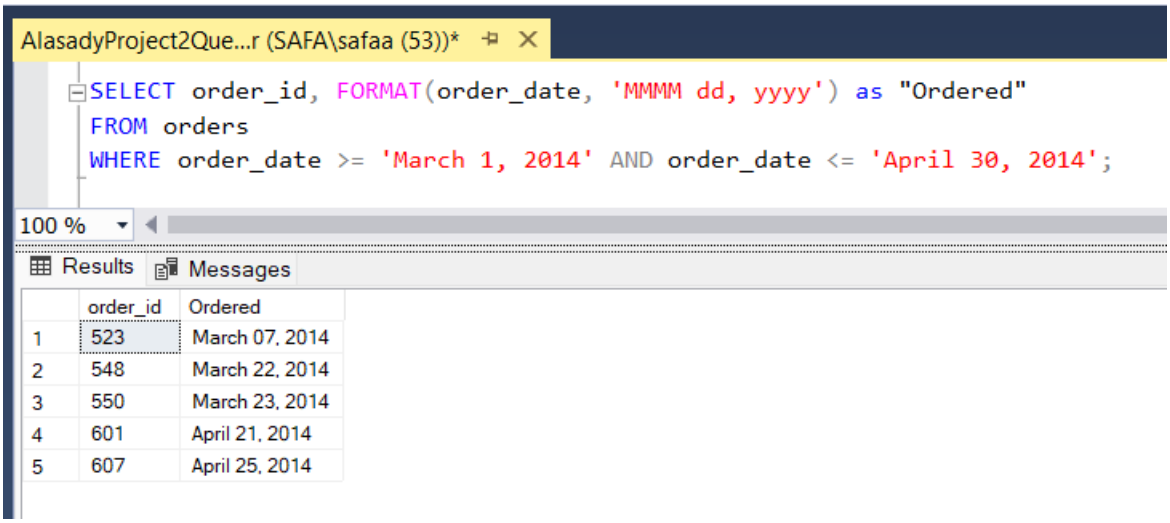


	customer_city	customer_first_name	customer_last_name	customer_zip
1	Fresno	Deborah	Davis	93728
2	Los Angeles	Karina	Miller	90084
3	Valencia	Kurt	Nickalus	91355
4	Sacramento	Hinrey	Lopez	95887
5	Fresno		Neftaly	93711
6	Orange	Ania	Irvin	92807
7	Los Angeles	Dakota	Baylee	90038
8	Palo Alto	Samuel	Jacobsen	92711
9	Fresno	Rashad	Holbrooke	93711
10	Manhatttan Beach	Theo	Hernandez	90266
11	San Francisco	Julian	Carson	94161
12	San Francisco	Ahmed	Azam	94161

This query selects customer_city, customer_first_name, customer_last_name, customer_zip from the customers table to only show rows that start with the number nine in their zip codes.

1. Query #10: 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”.

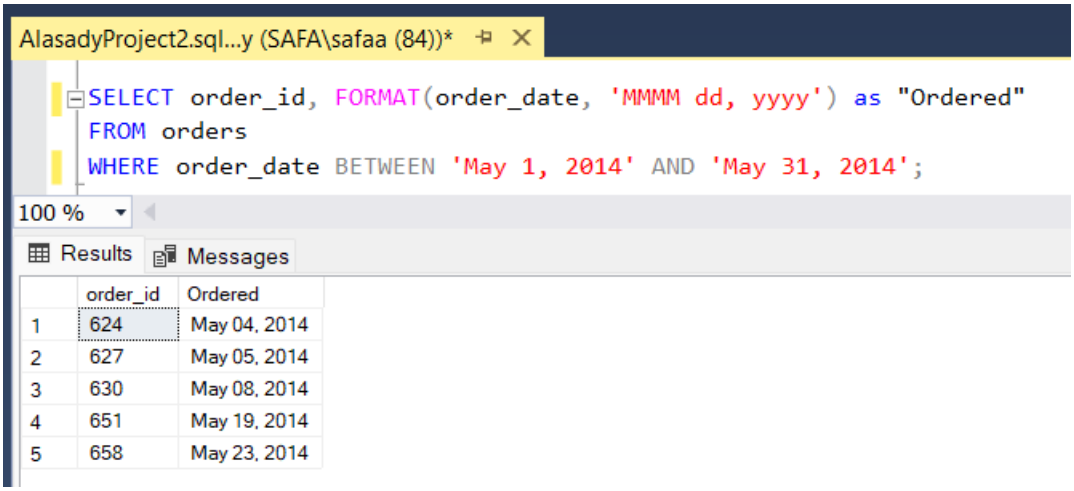
2. SELECT order_id, FORMAT(order_date, 'MMMM dd, yyyy') as "Ordered"
   FROM orders
   WHERE order_date >= 'March 1, 2014' AND order_date <= 'April 30, 2014';

3. 

This query selects order_id and order_date from the orders table. The column order_date, which is also labeled as “Ordered”, is formatted in a way that it starts with month, day, and year. Using the WHERE clause, the column order_date only shows orders from March 1, 2014, to April 30, 2014.

1. Query #11: 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”.

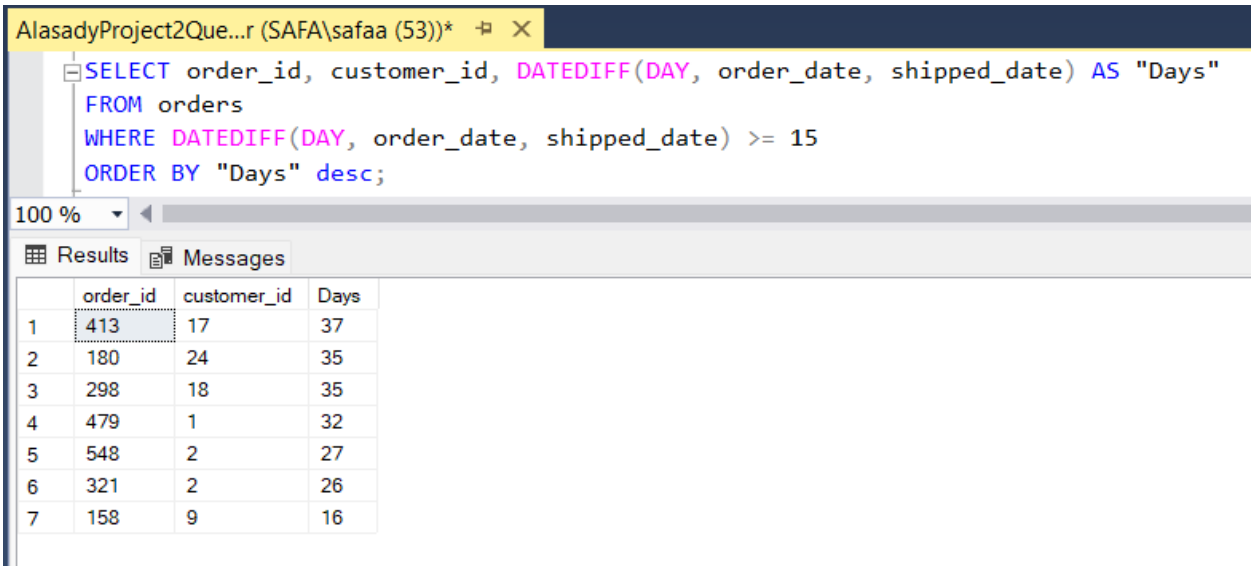
2. SELECT order_id, FORMAT(order_date, 'MMMM dd, yyyy') as "Ordered"
   FROM orders
   WHERE order_date BETWEEN 'May 1, 2014' AND 'May 30, 2014';

3. 

From the orders table, order_id and order_date are selected. The column order_date is labeled as “Ordered” and formatted as month, day, and year. To get orders from May, the BETWEEN clause is used in the query.

1. Query #12: 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 "Days" desc;

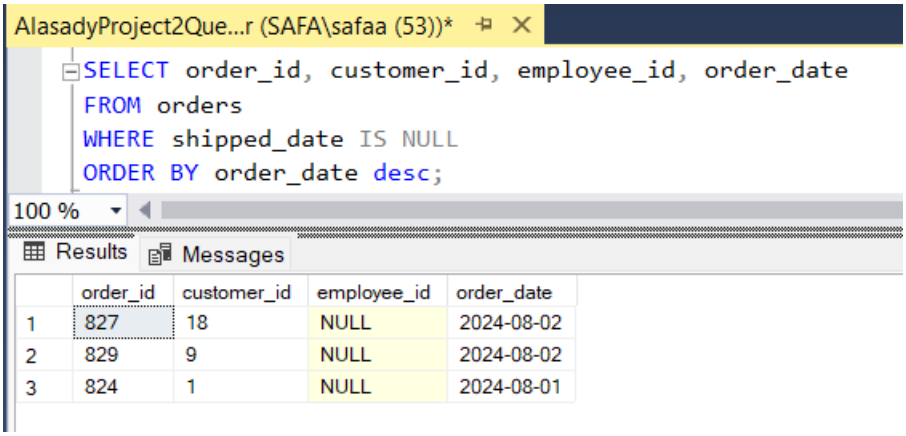
3. 

	order_id	customer_id	Days
1	413	17	37
2	180	24	35
3	298	18	35
4	479	1	32
5	548	2	27
6	321	2	26
7	158	9	16

From the orders table, order_id, customer_id, and order_date are selected in the query. The column, order_date, is labeled as “Days”. The DATEDIFF function is used to display the number of days from the order_date and the ship_date. Orders that have fifteen or more days are shown. The results are also sorted by order_date, or “Days” in descending order.

1. Query #13: 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 order_id, customer_id, employee_id, order_date
   FROM orders
   WHERE shipped_date IS NULL
   ORDER BY order_date desc;

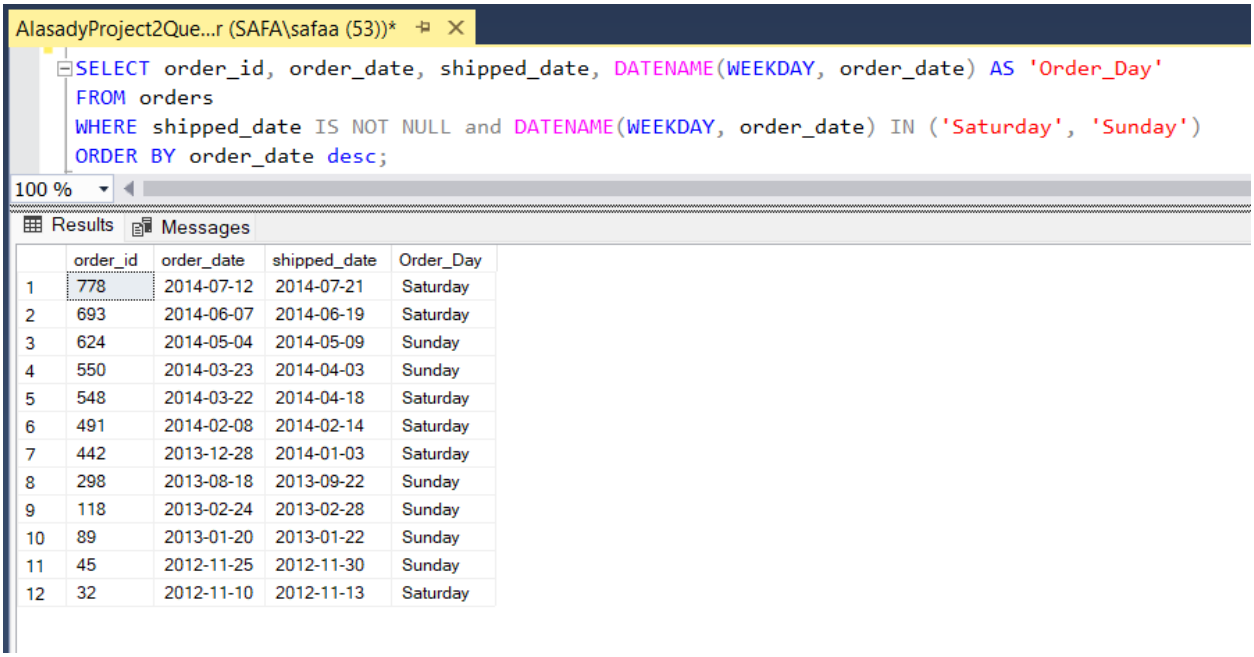
3. 

	order_id	customer_id	employee_id	order_date
1	827	18	NULL	2024-08-02
2	829	9	NULL	2024-08-02
3	824	1	NULL	2024-08-01

The columns order_id, customer_id, employee_id, and order_date are selected from the table, orders, to show orders that have not been shipped. The results are sorted by order_date in descending order.

1. Query #14: 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 shipped_date IS NOT NULL and DATENAME(WEEKDAY, order_date) IN ('Saturday', 'Sunday')
   ORDER BY order_date desc;

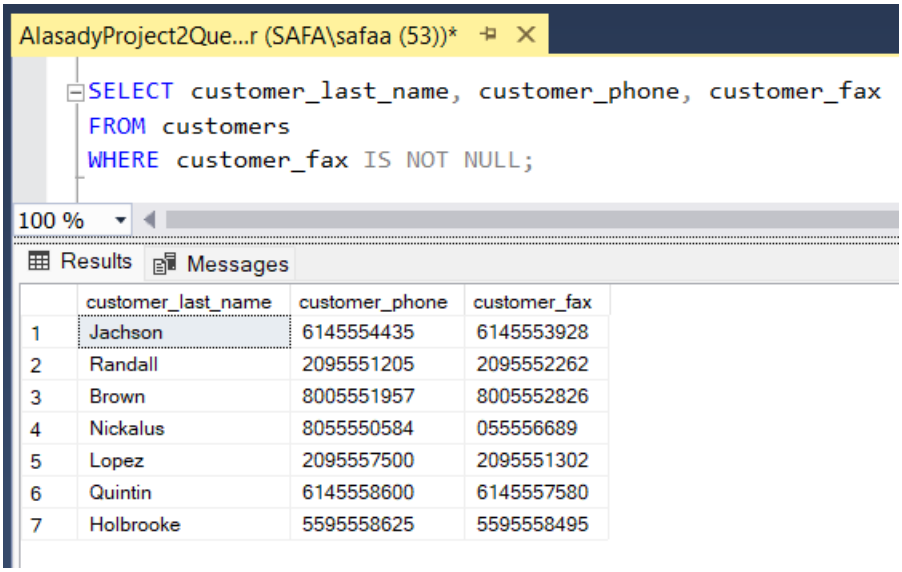
3. 

	order_id	order_date	shipped_date	Order_Day
1	778	2014-07-12	2014-07-21	Saturday
2	693	2014-06-07	2014-06-19	Saturday
3	624	2014-05-04	2014-05-09	Sunday
4	550	2014-03-23	2014-04-03	Sunday
5	548	2014-03-22	2014-04-18	Saturday
6	491	2014-02-08	2014-02-14	Saturday
7	442	2013-12-28	2014-01-03	Saturday
8	298	2013-08-18	2013-09-22	Sunday
9	118	2013-02-24	2013-02-28	Sunday
10	89	2013-01-20	2013-01-22	Sunday
11	45	2012-11-25	2012-11-30	Sunday
12	32	2012-11-10	2012-11-13	Saturday

This query selects order_id, order_date, shipped_date, and a formatted version of order_date, from the orders table. The formatted version of order_date is formatted by the weekday the order was made on and is labeled as "Order_Day". The results are filtered to only show orders that have been shipped and were placed on Saturday and Sunday. Results are also sorted by order_date in descending order.

1. Query #15: 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;

3. 

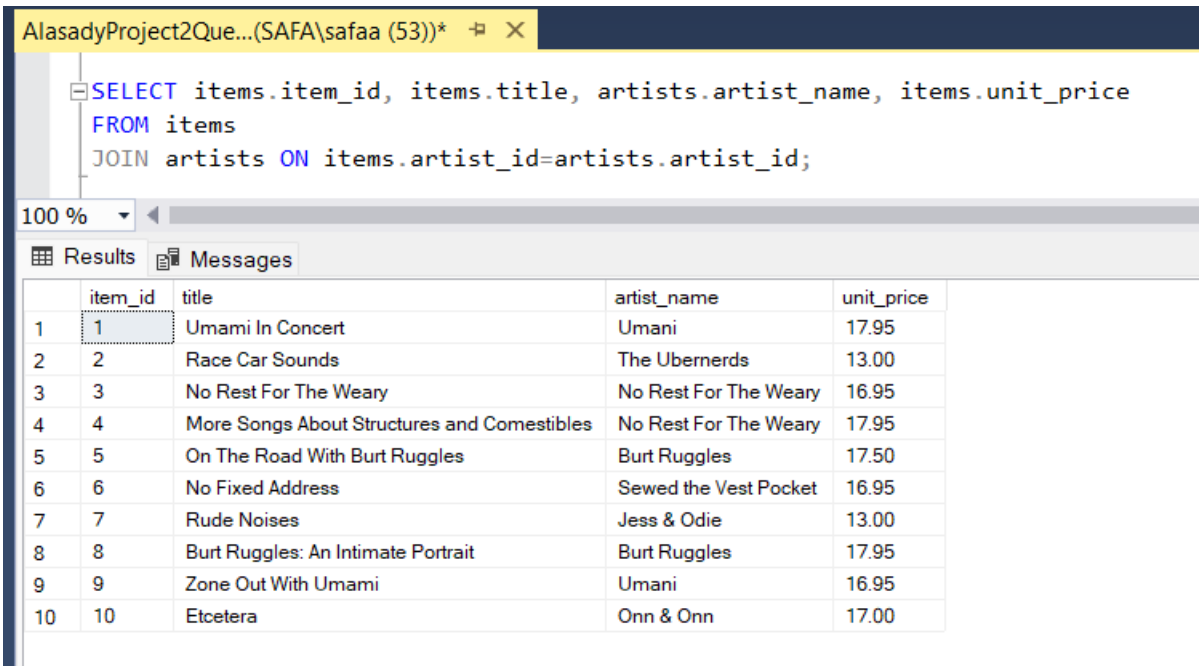
	customer_last_name	customer_phone	customer_fax
1	Jachson	6145554435	6145553928
2	Randall	2095551205	2095552262
3	Brown	8005551957	8005552826
4	Nickalus	8055550584	055556689
5	Lopez	2095557500	2095551302
6	Quintin	6145558600	6145557580
7	Holbrooke	5595558625	5595558495

The columns, customer_last_name, customer_phone, and customer_fax, from the customers table are selected. Results are filtered to only show customers that have a fax number.

1. Query #16: 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, items.title, artists.artist_name, items.unit_price
   FROM items
   JOIN artists ON items.artist_id=artists.artist_id;

3.

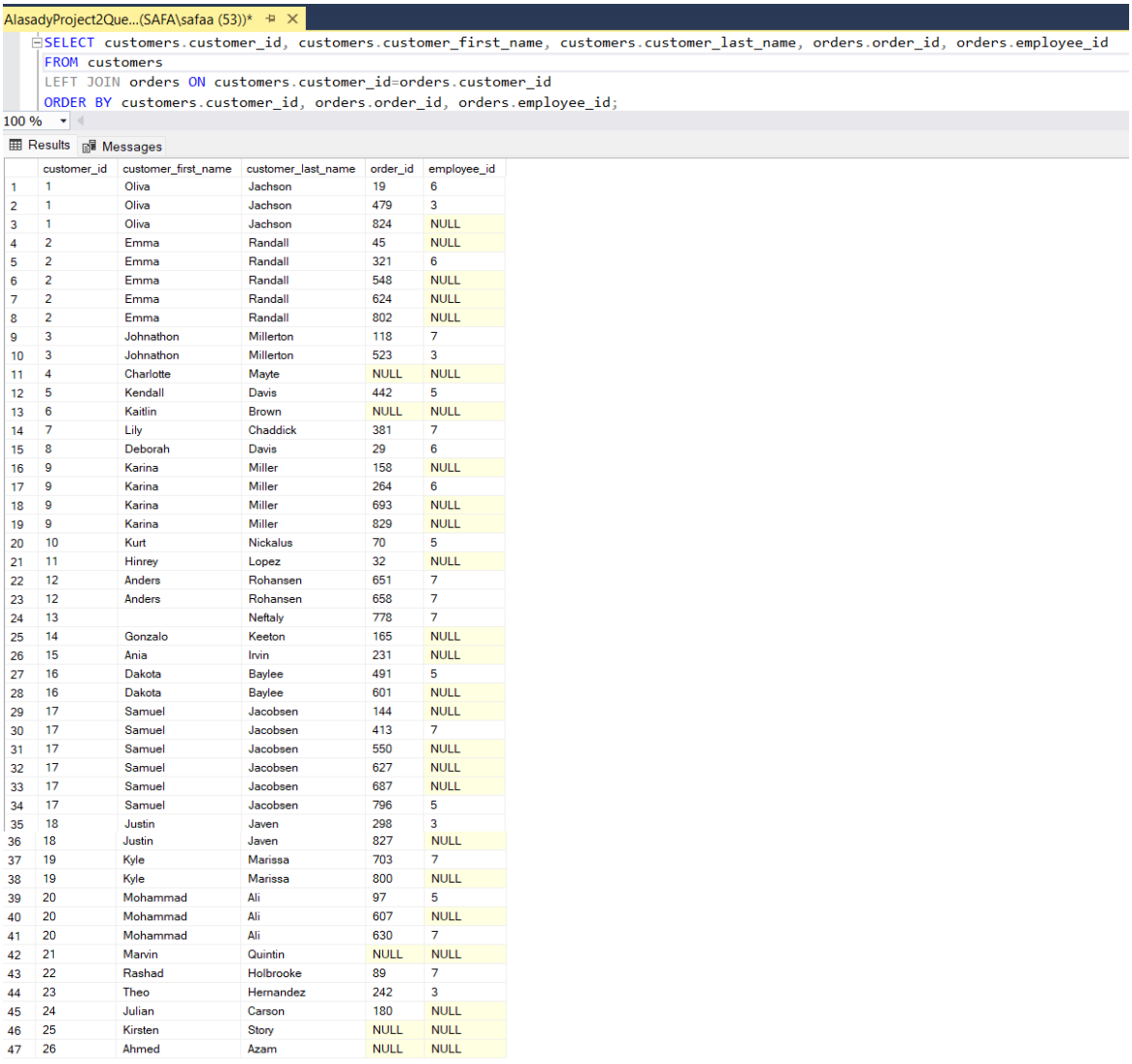


	item_id	title	artist_name	unit_price
1	1	Umami In Concert	Umani	17.95
2	2	Race Car Sounds	The Ubernerds	13.00
3	3	No Rest For The Weary	No Rest For The Weary	16.95
4	4	More Songs About Structures and Comestibles	No Rest For The Weary	17.95
5	5	On The Road With Burt Ruggles	Burt Ruggles	17.50
6	6	No Fixed Address	Sewed the Vest Pocket	16.95
7	7	Rude Noises	Jess & Odie	13.00
8	8	Burt Ruggles: An Intimate Portrait	Burt Ruggles	17.95
9	9	Zone Out With Umami	Umani	16.95
10	10	Etcetera	Onn & Onn	17.00

Columns item_id, title, artist_name, and unit_prices, are selected from the items table. The JOIN operation is used to get data from the items and artists table and gets data from both tables that match the artist_id column.

1. Query #17: 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;

3.



AlasadyProject2Que...(SAFA\safaa (53))*

```
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;
```

100 %

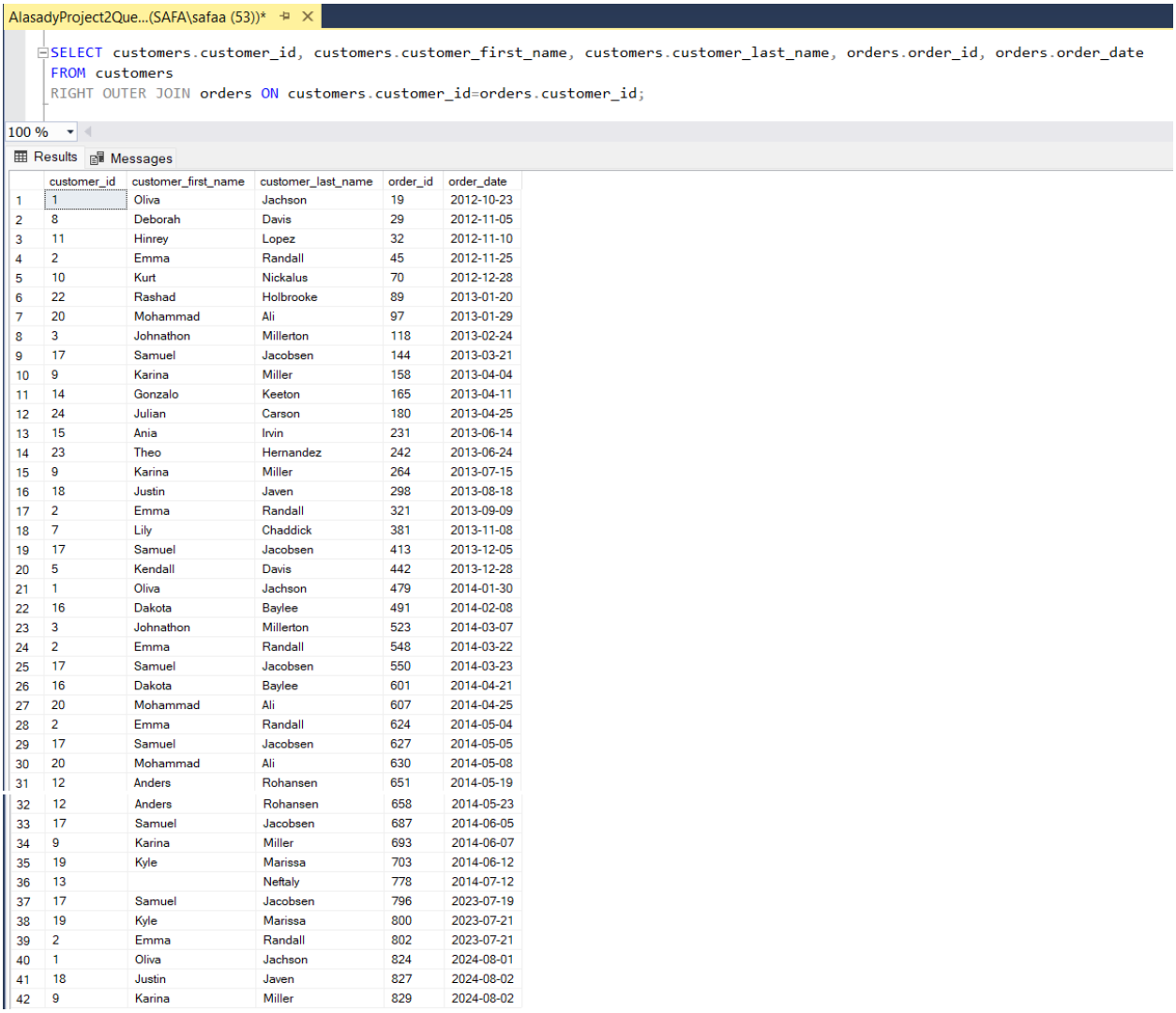
Results | Messages

	customer_id	customer_first_name	customer_last_name	order_id	employee_id
1	1	Oliva	Jachson	19	6
2	1	Oliva	Jachson	479	3
3	1	Oliva	Jachson	824	NULL
4	2	Emma	Randall	45	NULL
5	2	Emma	Randall	321	6
6	2	Emma	Randall	548	NULL
7	2	Emma	Randall	624	NULL
8	2	Emma	Randall	802	NULL
9	3	Johnathon	Millerton	118	7
10	3	Johnathon	Millerton	523	3
11	4	Charlotte	Mayte	NULL	NULL
12	5	Kendall	Davis	442	5
13	6	Kaitlin	Brown	NULL	NULL
14	7	Lily	Chaddick	381	7
15	8	Deborah	Davis	29	6
16	9	Karina	Miller	158	NULL
17	9	Karina	Miller	264	6
18	9	Karina	Miller	693	NULL
19	9	Karina	Miller	829	NULL
20	10	Kurt	Nickalus	70	5
21	11	Hinrey	Lopez	32	NULL
22	12	Anders	Rohansen	651	7
23	12	Anders	Rohansen	658	7
24	13		Neftaly	778	7
25	14	Gonzalo	Keeton	165	NULL
26	15	Ania	Irvin	231	NULL
27	16	Dakota	Baylee	491	5
28	16	Dakota	Baylee	601	NULL
29	17	Samuel	Jacobsen	144	NULL
30	17	Samuel	Jacobsen	413	7
31	17	Samuel	Jacobsen	550	NULL
32	17	Samuel	Jacobsen	627	NULL
33	17	Samuel	Jacobsen	687	NULL
34	17	Samuel	Jacobsen	796	5
35	18	Justin	Javen	298	3
36	18	Justin	Javen	827	NULL
37	19	Kyle	Marissa	703	7
38	19	Kyle	Marissa	800	NULL
39	20	Mohammad	Ali	97	5
40	20	Mohammad	Ali	607	NULL
41	20	Mohammad	Ali	630	7
42	21	Marvin	Quintin	NULL	NULL
43	22	Rashad	Holbrooke	89	7
44	23	Theo	Hernandez	242	3
45	24	Julian	Carson	180	NULL
46	25	Kirsten	Story	NULL	NULL
47	26	Ahmed	Azam	NULL	NULL

The columns, customer_id, customer_first_name, and customer_last_name from the customers table and order_id, and employee_id, are selected from the orders table. The LEFT JOIN operation is used to get data from the customers and items table, and gets data from both tables that match the customer_id column. The results are sorted by customers.customer_id, orders.order_id, orders.employee_id in ascending order.

1. Query #18: 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 customers.customer_id, customers.customer_first_name, customers.customer_last_name, orders.order_id, orders.order_date
   FROM customers
   RIGHT OUTER JOIN orders ON customers.customer_id=orders.customer_id;

3. 

```
SELECT customers.customer_id, customers.customer_first_name, customers.customer_last_name, orders.order_id, orders.order_date
FROM customers
RIGHT OUTER JOIN orders ON customers.customer_id=orders.customer_id;
```

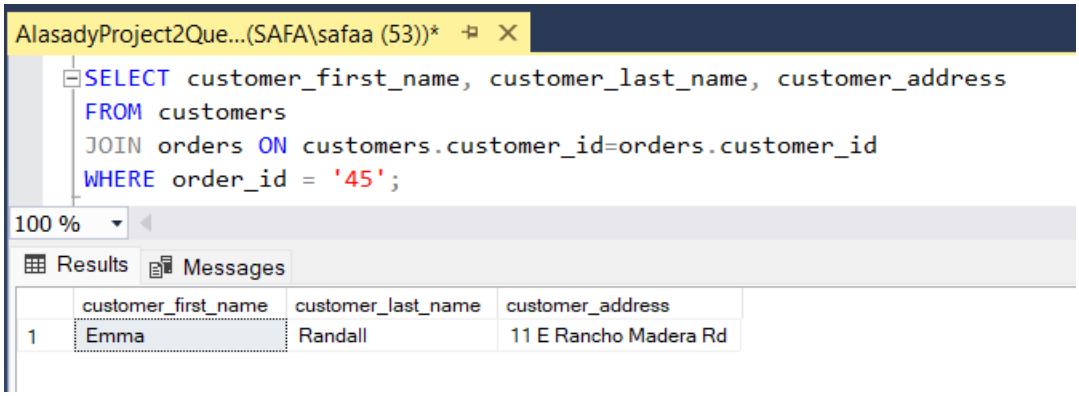
Results | Messages

	customer_id	customer_first_name	customer_last_name	order_id	order_date
1	1	Oliva	Jachson	19	2012-10-23
2	8	Deborah	Davis	29	2012-11-05
3	11	Hinrey	Lopez	32	2012-11-10
4	2	Emma	Randall	45	2012-11-25
5	10	Kurt	Nickalus	70	2012-12-28
6	22	Rashad	Holbrooke	89	2013-01-20
7	20	Mohammad	Ali	97	2013-01-29
8	3	Johnathon	Millerton	118	2013-02-24
9	17	Samuel	Jacobsen	144	2013-03-21
10	9	Karina	Miller	158	2013-04-04
11	14	Gonzalo	Keeton	165	2013-04-11
12	24	Julian	Carson	180	2013-04-25
13	15	Ania	Irvin	231	2013-06-14
14	23	Theo	Hernandez	242	2013-06-24
15	9	Karina	Miller	264	2013-07-15
16	18	Justin	Javen	298	2013-08-18
17	2	Emma	Randall	321	2013-09-09
18	7	Lily	Chaddick	381	2013-11-08
19	17	Samuel	Jacobsen	413	2013-12-05
20	5	Kendall	Davis	442	2013-12-28
21	1	Oliva	Jachson	479	2014-01-30
22	16	Dakota	Baylee	491	2014-02-08
23	3	Johnathon	Millerton	523	2014-03-07
24	2	Emma	Randall	548	2014-03-22
25	17	Samuel	Jacobsen	550	2014-03-23
26	16	Dakota	Baylee	601	2014-04-21
27	20	Mohammad	Ali	607	2014-04-25
28	2	Emma	Randall	624	2014-05-04
29	17	Samuel	Jacobsen	627	2014-05-05
30	20	Mohammad	Ali	630	2014-05-08
31	12	Anders	Rohansen	651	2014-05-19
32	12	Anders	Rohansen	658	2014-05-23
33	17	Samuel	Jacobsen	687	2014-06-05
34	9	Karina	Miller	693	2014-06-07
35	19	Kyle	Marissa	703	2014-06-12
36	13		Neftaly	778	2014-07-12
37	17	Samuel	Jacobsen	796	2023-07-19
38	19	Kyle	Marissa	800	2023-07-21
39	2	Emma	Randall	802	2023-07-21
40	1	Oliva	Jachson	824	2024-08-01
41	18	Justin	Javen	827	2024-08-02
42	9	Karina	Miller	829	2024-08-02

The columns, customer_id, customer_first_name, customer_last_name, from the customers table, and order_id and order_date from the orders table are selected. The RIGHT OUTER JOIN operation is used to join both the customers and orders table with the matching column, customer_id.

1. Query #19: 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
   JOIN orders ON customers.customer_id=orders.customer_id
   WHERE order_id = '45';

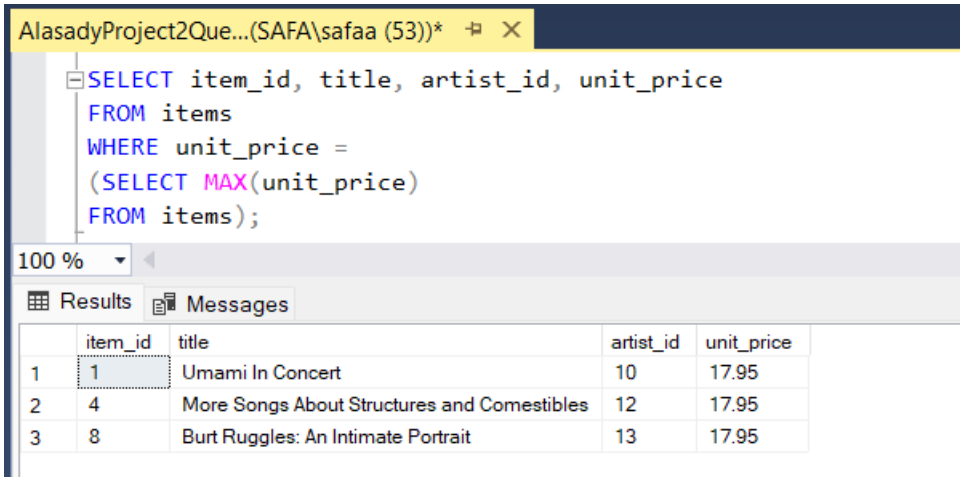
3. 

The query selects customer_first_name, customer_last_name, customer_address from the customers table. To get data on which customer was order number 45, a JOIN is used to show results where customer_id matches from both tables along with a WHERE clause to specify the order number.

1. Query #20: List the details about the item with the highest standard price.

2. SELECT item_id, title, artist_id, unit_price
   FROM items
   WHERE unit_price =
   (SELECT MAX(unit_price)
   FROM items);

3.



	item_id	title	artist_id	unit_price
1	1	Umami In Concert	10	17.95
2	4	More Songs About Structures and Comestibles	12	17.95
3	8	Burt Ruggles: An Intimate Portrait	13	17.95

The item_id, title, artist_id, unit_price columns are selected from the items table. To get details about the item with the highest standard price, we use a subquery in the WHERE clause to find the highest unit_price from the items table.

1. Query #21: Create a statement to insert a new record into the items table with the following values:

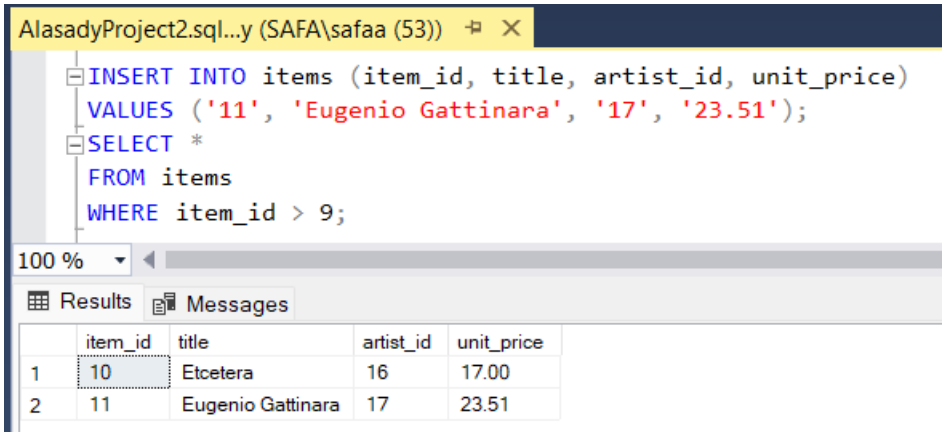
item_id:	11
title:	Eugenio Gattinara
Artist_id:	17
unit_price	23.51

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;

2. INSERT INTO items (item_id, title, artist_id, unit_price)
VALUES ('11', 'Eugenio Gattinara', '17', '23.51');
SELECT *
FROM items
WHERE item_id > 9;

3.



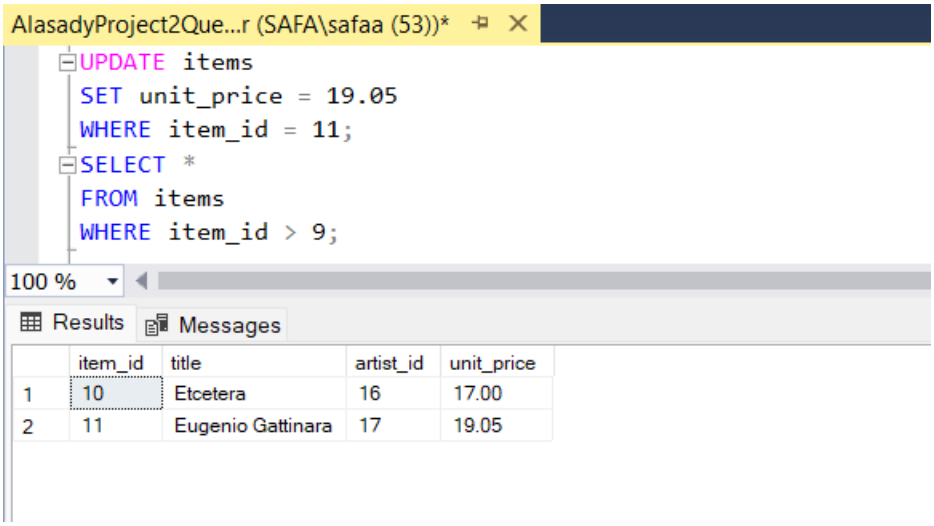
To insert a new record, the INSERT INTO clause is used with the details for item_id, title, artist_id, and unit_price. The SELECT and WHERE clauses are used to confirm that the INSERT INTO statement worked, showing rows where the item_id is greater than nine.

1. Query #22: 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_id:	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;

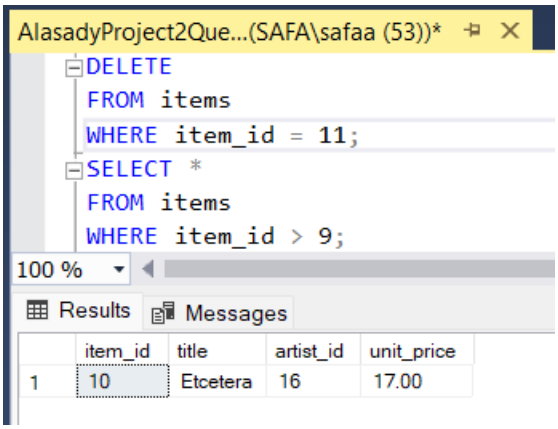
2. UPDATE items
SET unit_price = 19.05
WHERE item_id = 11;
SELECT *
FROM items
WHERE item_id > 9;

3. 

Instead of keeping the new record as is, the UPDATE statement is used to change the unit_price of the new record from 23.51 to 19.05. The SELECT and WHERE clause is again used to show the change, showing rows where the item_id is greater than nine.

1. Query #23: 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;

3. 

The DELETE statement is used to delete the new record that now shows the unit_price of 19.05. The SELECT and WHERE clause is again used to show the change, showing rows where item_id 11 before used to show up.

1. Query #24: 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;
3.

AlasadyProject2Que...(SAFA\safaa (53))*

```
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;
```

100 %

Results Messages

	Customer	Phone
1	Neftaly,	(559) 555-6245
2	Azam, Ahmed	(617) 555-0700
3	Rohansen, Anders	(338) 555-6772
4	Irvin, Ania	(714) 555-9000
5	Mayte, Charlotte	(202) 555-5561
6	Baylee, Dakota	(213) 555-4322
7	Davis, Deborah	(559) 555-8060
8	Randall, Emma	(209) 555-1205
9	Keeton, Gonzalo	(201) 555-9742
10	Lopez, Hinrey	(209) 555-7500
11	Millerton, Johnathon	(212) 555-4800
12	Carson, Julian	(617) 555-0700
13	Javen, Justin	(800) 555-0037
14	Brown, Kaitlin	(800) 555-1957
15	Miller, Karina	(800) 555-7000
16	Davis, Kendall	(513) 555-3043
17	Story, Kirsten	(206) 555-9115
18	Nickalus, Kurt	(805) 555-0584
19	Marissa, Kyle	(947) 555-3900
20	Chaddick, Lily	(515) 555-6130
21	Quintin, Marvin	(614) 555-8600
22	Ali, Mohammad	(704) 555-3500
23	Jachson, Oliva	(614) 555-4435
24	Holbrooke, Rashad	(559) 555-8625
25	Jacobsen, Samuel	(415) 555-3434
26	Hernandez, Theo	(310) 555-2732

This query selects customer_last_name, customer_first_name, customer_phone from the customers table. The CONCAT function is used to concatenate customer_last_name and customer_first_name to format it as customer last name, a comma, and then the customer first name in a single column. CONCAT is also used for customer_number where it is outputted into phone number format. The first column is labeled as “Customer” and the second column is labeled as “Phone”. Results are sorted by customer_first_name in ascending order.

1. Query #25: 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.

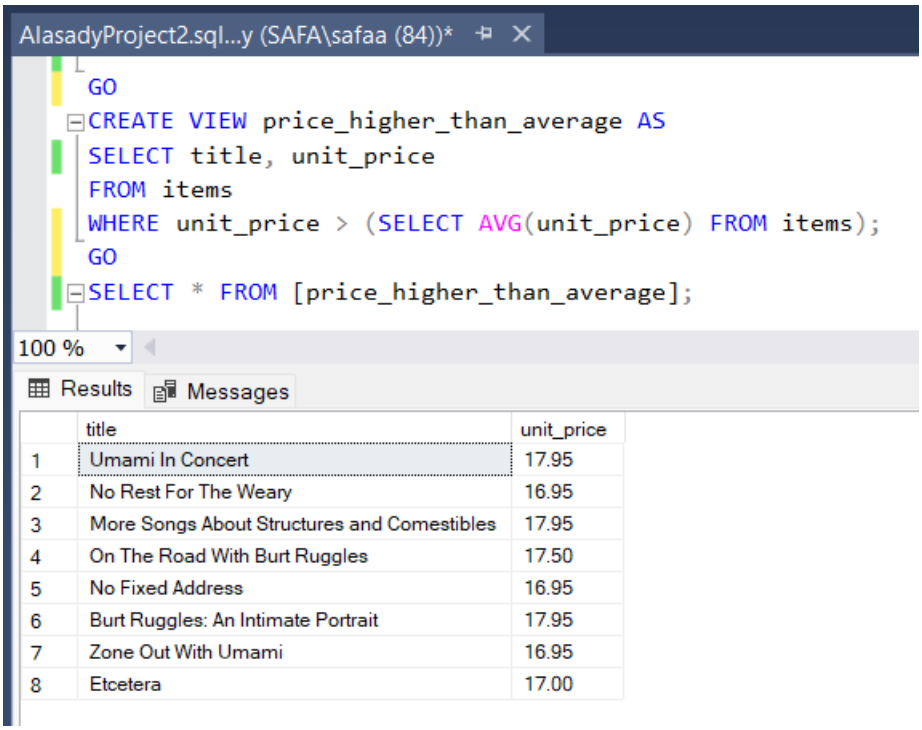
2. INSERT INTO customers (customer_id, customer_first_name, customer_last_name, customer_address, customer_city, customer_state, customer_zip, customer_phone, customer_fax)
   VALUES ('27', 'Safa', 'Alasady', '3801 W Temple Ave', 'Pomona', 'CA', '91768', '9096850815', '9096850815');
   SELECT *
   FROM customers
   WHERE customer_id = 27;

3.

The INSERT INTO statement is used to insert a new record with the student's information. The columns, customer_id, customer_first_name, customer_last_name, customer_address, customer_city, customer_state, customer_zip, customer_phone, and customer_fax, are selected form the customers table. The SELECT clause is used to check if the INSERT INTO worked and added a new record to the customers table. Just as a note, I left the customer fax as my phone number as well since I do not have a fax number.

1. Query #26: Creates a view that selects every title in the "item" table with a price higher than the average price.

2. GO
   CREATE VIEW price_higher_than_average AS
   SELECT title
   FROM items
   WHERE unit_price > (SELECT AVG(unit_price) FROM items);
   GO
   SELECT * FROM [price_higher_than_average];

3. 

	title	unit_price
1	Umami In Concert	17.95
2	No Rest For The Weary	16.95
3	More Songs About Structures and Comestibles	17.95
4	On The Road With Burt Ruggles	17.50
5	No Fixed Address	16.95
6	Burt Ruggles: An Intimate Portrait	17.95
7	Zone Out With Umami	16.95
8	Etcetera	17.00

The CREATE VIEW adds a view named “price_higher_than_average”. The columns selected are title and unit_price from the items table. The clauses after the CREATE VIEW get titles whose unit prices are specifically greater than the average price. Just to be sure, I had done the average of all unit prices myself, and it turned out to be 16.52. This means that the table displayed above correctly shows titles with unit prices higher than the average.

1. Query #27: Explain the cardinality from the employees-to-employees table.
2. /* The cardinality from the employees-to-employees table is a recursive foreign key between manager and employee. Managers can have multiple employees working under them as shown by table displayed below. For example, both Aria Moore and Paulo Locario have the same manager and same manager_id number, which is Mia Garcia. There is also a one to many relationship between the employee_id and manager_id. Each employee has only one manager while a manager can manage multiple employees.
   */

   SELECT employee_id, first_name, last_name, manager_id
   FROM employees
   ORDER BY employee_id

3.

AlasadyProject2.sql...y (SAFA\safaa (84))*

```
/* The cardinality from the employees-to-employees table is a recursive foreign key between
manager and employee. Managers can have multiple employees working under them as shown by
table displayed below. For example, both Aria Moore and Paulo Locario have the same manager
and same manager_id number, which is Mia Garcia. There is also a one to many relationship
between the employee_id and manager_id. Each employee has only one manager while a manager
can manage multiple employees.
*/

SELECT employee_id, first_name, last_name, manager_id
FROM employees
ORDER BY employee_id
```

100 %

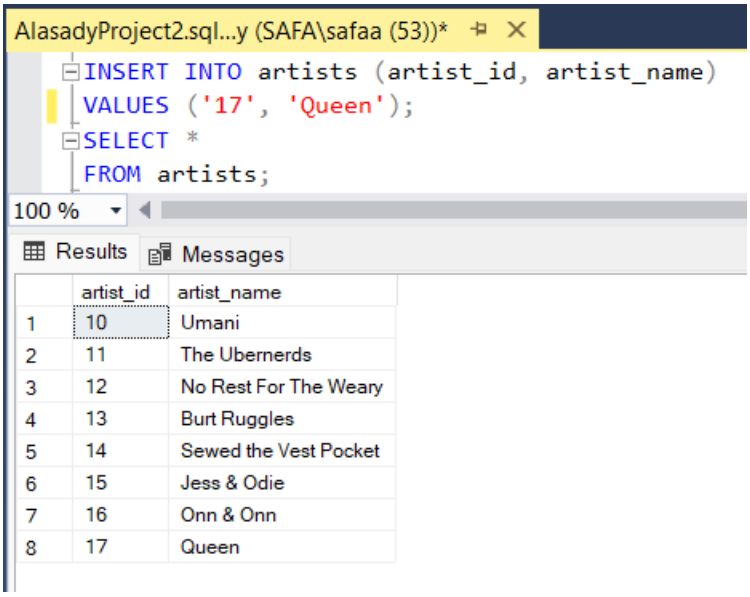
Results | Messages

	employee_id	first_name	last_name	manager_id
1	1	Mia	Garcia	NULL
2	2	Aria	Moore	1
3	3	Layla	Lee	2
4	4	Olivia	Hernandez	9
5	5	Robert	Aaronsen	4
6	6	Asher	White	8
7	7	Thomas	clark	2
8	8	Rhea	Leary	9
9	9	Paulo	Locario	1

Columns employee_id, first_name, last_name, and manager_id are selected from the employees table to show the recursive foreign key and the one to many relationship. From the table above, it is shown that employees have only one manager and managers manage multiple employees. The column for manager_id for Mia Garcia is NULL because Mia Garcia does not have a manager. Results are sorted by employee_id in ascending order.

1. Query #28: Insert a new artist (artist id and artist name)

2. INSERT INTO artists (artist_id, artist_name)
   VALUES ('17', ‘Queen’);
   SELECT *
   FROM artists;

3. 

	artist_id	artist_name
1	10	Umani
2	11	The Ubernerds
3	12	No Rest For The Weary
4	13	Burt Ruggles
5	14	Sewed the Vest Pocket
6	15	Jess & Odie
7	16	Onn & Onn
8	17	Queen

To add a new artist to the artists table, INSERT INTO is used with artitist_id and artist_name columns. To make sure that it was added, the SELECT and FROM clauses are used to show the artists table.

Report Analysis

For all twenty-eight queries, clauses like INSERT INTO, BETWEEN, OUTER JOIN, RIGHT JOIN, LEFT JOIN, DELETE, and many others were used to demonstrate how the database works and the implementation of these clauses to show tables with specific needs. For example, one of the queries asked to only show customers that live in the state of California and the city of San Francisco. I was able to complete this query by using the SELECT, FROM, and WHERE clause. One thing I wish I could change would be the order of the queries. While I do understand that as a student, I need to be able to problem solve and figure out any errors that may come up on Microsoft SQL, I had to spend a considerable amount of time doing queries 21 and 22 because both queries were not working until I had to complete query 28. The error that kept coming up when executing was that there was a conflict with the column artists_id and artists table.

Results

The queries did start easy then became more complicated as we got into JOINs. While learning the data models earlier in the semester, Microsoft SQL lets students be able to retrieve, organize, manipulate, define, and although the project did not require this, control databases. For most of the queries, it was mostly retrieval, organization, manipulation, and defining the database. All of the queries were, of course, mainly retrieval, but it was also organizing the data through the ASC, DESC, and ORDER BY which sorted result tables, the aggregate functions like AVG, COUNT, and ROUND, manipulating the data through functions like INSERT INTO, UPDATE and DELETE, and defining through functions like DROP.

Discussion

Most of the queries were about retrieving data and being able to change or filter data to what we need from tables. In real life, companies will need to use databases for their information, and through this project, I was able to simulate some aspects of that. Some challenges were learning how to create a view and get the table to show. At some point, the column unit_price was displaying all of the values rather than just the column unit_prices showing prices that were higher than the average price. I was able to figure out that CREATE VIEW, SELECT, FROM, and WHERE are executed once, and then we would use SELECT and FROM using the view name price_higher_than_average to see the result table. Just to verify that the results were correct, I calculated the average of all prices, and it was 16.52. The result table was correct. The solution to that was very simple, but it still took some time to figure out.

Lessons Learned

Many of the lessons learned were about the important SQL functions and statements, and how they are used to get data and being able to read it as a table. After completing each query, I learned important topics like how to use MS SQL and its server, how to use the three most used clauses, SELECT, FROM, and WHERE, how to do complex joins specifically outer, right, and left, and how to navigate and resolve errors. I also learned about the presence of a recursive foreign key and the one-to-many relationship between manager_id and employee_id. By the end of completing all queries, I am also able to see how databases are everywhere even if it is just simply putting in your username and password to log in to a website or application, that data is stored in a database. I was able to learn the fundamentals of Microsoft SQL Server and how it works with retrieving and manipulating data to fulfill important needs like filtering results.

Conclusion

In the end, this project felt a lot simpler than the previous project. This project was about using data that was provided and using it to complete queries to get results that can be analyzed. The benefit of this project is learning how to design, develop, and show how databases work with business rules and constraints. This project was very straightforward. I was looking forward to getting started on Microsoft SQL Server, especially as a Computer Information Systems major. My future career might consist of mostly using SQL. SQL is used everywhere for handling data and databases, so getting experience and a good understanding of SQL is important. The error that was a conflict between the column artists_id and artists table showed me that the order queries are done matter, and it should be organized in a logical order. In the future, I will make sure to analyze the error warning and understand what is preventing my queries from executing properly. The process of how I investigated the issue should also be used again when coming across errors similar to the one I faced. Because of this project, I was able to learn significant parts of how queries, views, joins, inserts, and much more all work together to retrieve important data and information to create and use tables.

References

- Hoffer, Jeffrey A, et al. Modern Database Management. 13th ed., Boston, Pearson Education, 2019.
- “SQL Joins.” W3schools.com, 2024, www.w3schools.com/sql/sql_join.asp. Accessed 29 Mar. 2024.
- “SQL Aggregate Functions.” W3schools.com, 2024, www.w3schools.com/sql/sql_aggregate_functions.asp. Accessed 29 Mar. 2024.
- Jones, Keil. “Quick Tutorial - Creating a View in SQL Server.” *YouTube*, 7 Feb. 2017, www.youtube.com/watch?v=bBncfoghbA8&t=1s&ab_channel=KeilJones. Accessed 29 Mar. 2024.