

Table Of Contents

Content	PG#
Users View.....	2
DDL.....	3
Data Dictionary.....	4
DBSD.....	5
ERD.....	6
Project Script.....	7
Team Top 10 SQL's.....	15
Aryan Merchant Contributions.....	2
Yvan Ngah Contributions.....	2

PIZZACO

This project is about creating a database system for a store named PIZZACO, owned by John, which offers pizza to customers in delivery format. We are creating this project from the perspective of the owner, John. The focus of the database system is to manage customer orders. The purpose of this project is to create a centralized and secure database system which will ultimately result in cost savings and increased profitability. Currently, the store is managing all its operations manually on paper without the aid of any database system.

This project is needed to improve the store's efficiency and productivity by reducing manual efforts and maintaining accurate records of customer orders. It will also help the store to calculate specific expressions like total order and average income per day. It will store all customer orders, making it easier for the store to access and manage data. The store staff, including the owner, managers, and employees, will utilize this project. This project will help them to manage their daily operations more efficiently and accurately. Types of reports, forms, and queries: total cost per day, total sales per day, number of customers, customer order forms and number of items sold on a specific day.

For example:

1. For if Mr. John wants to check on the profits for a specific date.

Mr. John would be able to use the database to get a report on the profits on that specific date.

2. If an employee has not shown up for their shift.

Mr. John would be able to use the database to find the phone number of that employee or call another employee in.

3. If a delivery employee needs to know the address of a customer. They would be able to check the database to find that customer's address

PIZZACO

Customer(CustNum, CustFName, CustLName, Phone, Email, DeliveryStreetAddr, State, City, ZCode)

Employee(EmpID, EmpLName, EmpFName, EmpPhone, EmpEmail, EmpStreetAddr, EmpState, EmpCity, EmpZCode, EmpType, EmpHourRate)

Orders(OrderNum, OrderDte, Delivered, CustID, DeliveryEmp, OrderEmp)

FK: CustID → Customer

DeliveryEmp → Employee

OrderEmp → Employee

Item(ItemID, ItemName, ItemCat, Size, ItemCost, ItemPrice)

OrderLine(OrderNum, ItemID, NumOrdered)

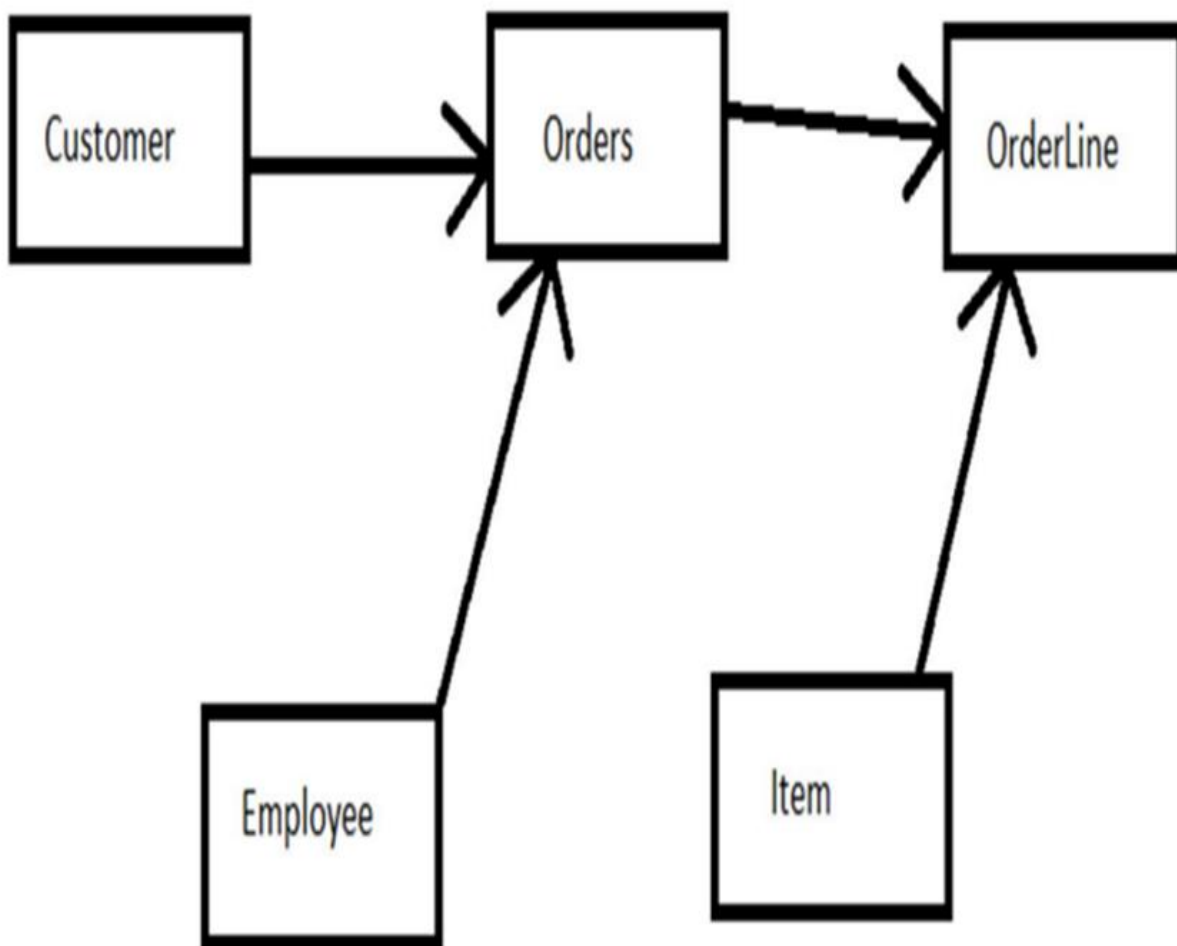
FK: OrderID → Orders

ItemID → Item

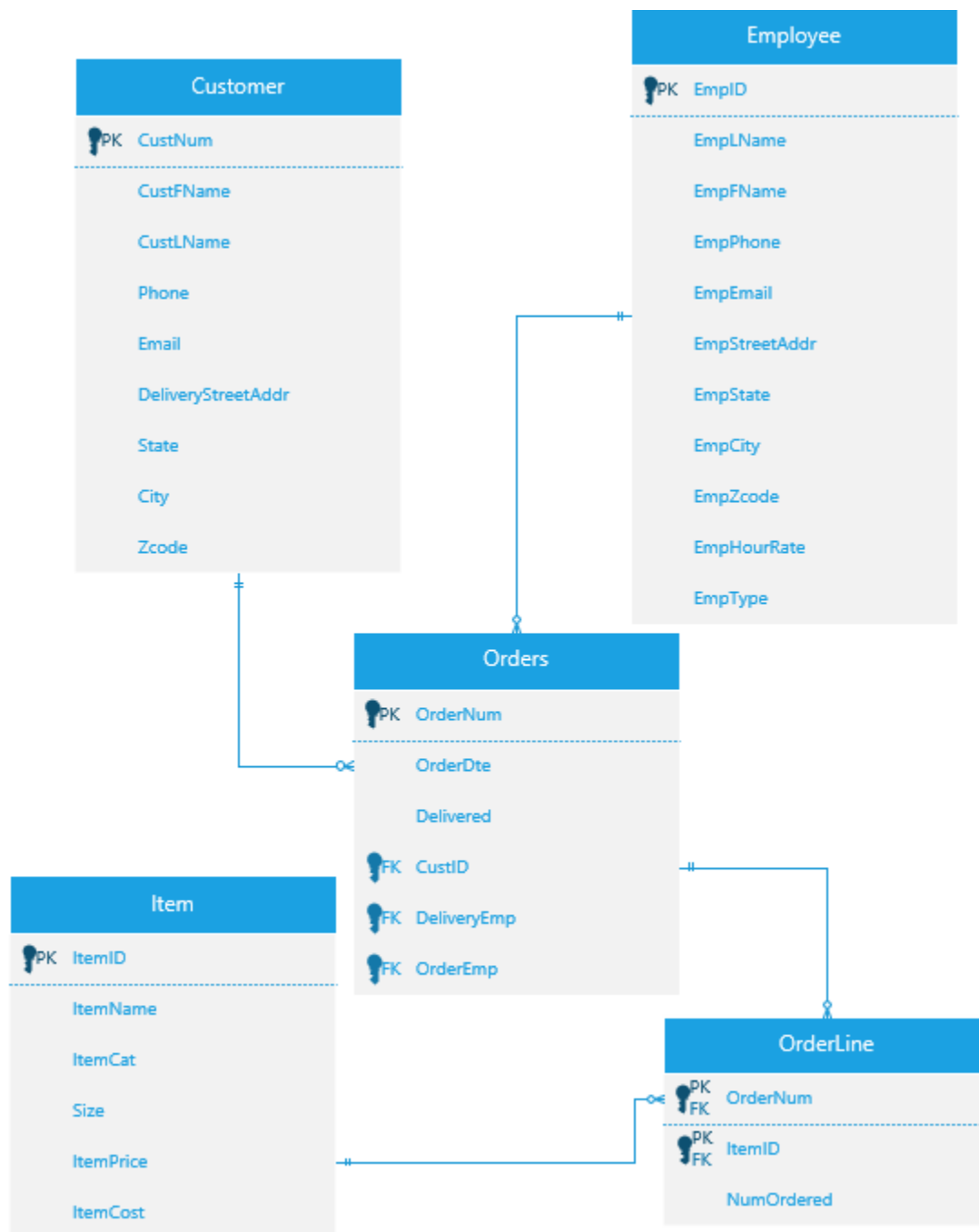
PIZZACO

Attribute Name	Contents	Type	Format	Range	Required	PK or FK	FK Referenced Table
Customer CustNum CustFName CustLName Phone Email DeliveryStre etAddr State City Zcode	Customer Customer Number Customer First Name Customer last Name Customer Phone Number Customer Email Adress Customer Delivery Street Address Customer State Customer City Customer Zip code	Customer CHAR(5) VARCHAR(50) VARCHAR(50) CHAR(10) VARCHAR(50) VARCHAR(50) VARCHAR(2) VARCHAR(50) CHAR(5)	Customer 99999 XXXXXXX XXXXXXX 999-999-9999 XXXXXXX XXXXXXX XX XXXXXXX 99999	Customer 10000-99999 10000-99999	Customer Y Y Y Y Y Y Y Y	Customer PK	Customer
Orders OrderNum OrderDate Delivered CustNum EmpID	Orders Order Number Order Date Order Status Customer Number Empomer Number	Orders CHAR(5) CHAR(8) VARCHAR(1) CHAR(5) CHAR(5)	Orders 99999 99-99-9999 X 99999 99999	Orders 10000-99999 Y/N 10000-99999 10000-99999	Orders Y Y Y Y Y	Orders PK FK FK	Orders Custom er Employ ee
Employee EmpID EmpLName EmpFName EmpPhone EmpEmail EmpStreetAddr EmpState EmpCity EmpZcode EmpHourRate EmpType	Employee Employee Number Employee Last Name Employee First Name Employee Phone Number Employee Email Adress Employee Street Address Employee State Adress Employee City Adress Employee Zip Code Employee Hour Pay Employee Type	Employee Employee CHAR(5) VARCHAR(50) VARCHAR(50) CHAR(10) VARCHAR(50) VARCHAR(50) VARCHAR(2) VARCHAR(50) CHAR(5) DECIMAL(2,2) VARCHAR(50)	Employee 99999 XXXXXXX XXXXXXX 999-999-9999 XXXXXXX XXXXXXX XX XXXXXXX 99999 99.99 XXXXXXX	Employee 10000-99999 10000-99999 .99-99.99	Employee Y Y Y Y Y Y Y Y Y	Employee PK	Employee
OrderLine OrderNum ItemID NumOrdered	OrderLine Order Number Item Number Number Ordered	OrderLine CHAR(5) CHAR(5) CHAR(2)	OrderLine 99999 99999 99	OrderLine 10000-99999 10000-99999 1-99	OrderLine 	OrderLine PK/FK PK/FK	OrderLine Orders Item
Item ItemID ItemName ItemCat Size ItemPrice ItemCost	Item Item Number Item Name Item Category Item Size Item Price Item Cost	Item CHAR(5) VARCHAR(50) VARCHAR(50) VARCHAR(50) DECIMAL(2,2) DECIMAL(2,2)	Item 99999 XXXXXXX XXXXXXX XXXXXXX 99.99 99.99	Item 10000-99999 .99-99.99 .99-99.99	Item Y Y Y Y Y Y	Item PK	Item

PIZZACO



PIZZACO



PIZZACO

```
USE USERNAME;
```

```
CREATE TABLE CUSTOMER
( CUSTNUM CHAR(5) PRIMARY KEY,
  CUSTFNAME VARCHAR(50),
  CUSTLNAME VARCHAR(50),
  PHONE CHAR(12) ,
  EMAIL VARCHAR(50),
  DELIVERYSTREETADDR VARCHAR(50),
  STATE VARCHAR(2),
  CITY VARCHAR(50),
  ZCODE CHAR(5) );
```

```
CREATE TABLE EMPLOYEE
( EMPID CHAR(5) PRIMARY KEY,
  EMPLNAME VARCHAR(50),
  EMPFNAME VARCHAR(50),
  EMPPHONE CHAR(12) ,
  EMPEMAIL VARCHAR(50),
  EMPSTREETADDR VARCHAR(50),
  EMPSTATE VARCHAR(2),
  EMPCITY VARCHAR(50),
  EMPZCODE CHAR(5),
  EMPHOURLRATE DECIMAL(4,2),
  EMPLOYEE TYPE VARCHAR(50) );
```

```
CREATE TABLE ITEM
( ITEMID CHAR(5) PRIMARY KEY,
  ITEMNAME VARCHAR(50),
  ITEMCAT VARCHAR(50),
  SIZE VARCHAR(50),
  ITEMPRICE DECIMAL(4,2),
  ITEMCOST DECIMAL(4,2) );
```

```
CREATE TABLE ORDERS
( ORDERNUM CHAR(5) PRIMARY KEY,
  ORDERDTE CHAR(10),
  DELIVERED VARCHAR(1),
  CUSTID CHAR(5),
  EMPID CHAR(5) );
```

```
CREATE TABLE ORDERLINE
( ORDERNUM CHAR(5),
  ITEMID CHAR(5),
  NUMORDERED CHAR(2) );
```

```
INSERT INTO CUSTOMER VALUES
('99710', 'JAMES', 'BROWN', '123-456-8888', 'JAMES@GMAIL.COM', '12 JUMP STREET',
'GA', 'MACON', '31610');
```

```
INSERT INTO CUSTOMER VALUES
('91810', 'LEE', 'SMITH', '423-456-8564', 'LEE@GMAIL.COM', '64 SAMSUNG STREET',
'GA', 'MIAMI', '17412');
```

```
INSERT INTO CUSTOMER VALUES
('93010', 'BILL', 'JAKE', '141-755-6438', 'BILL@GMAIL.COM', '85 CART STREET', 'GA',
'SAN DIEGO', '73522');
```

```
INSERT INTO CUSTOMER VALUES
('96410', 'AL', 'RAVEN', '643-123-9234', 'AL@GMAIL.COM', '67 CRYSTAL STREET', 'GA',
'BUFFALO', '31340');
```

```
INSERT INTO CUSTOMER VALUES
('94410', 'FULTON', 'CARL', '633-123-6151', 'FULTON@GMAIL.COM', '71 GROVE STREET',
'GA', 'NASHVILLE', '84324');
```

```
INSERT INTO EMPLOYEE VALUES
('51010', 'BRYCE', 'PRESTON', '233-473-4561', 'BRYCE@GMAIL.COM', '92 MARY STREET',
'GA', 'MACON', '31610', 31.00, 'ORDER');
```

```
INSERT INTO EMPLOYEE VALUES
```



```
('52020', 'MACY', 'TANNER', '546-123-6523', 'MACY@GMAIL.COM', '82 DELLA STREET',
'GA', 'MIAMI', '17412', 34.00, 'ORDER');
```

```
INSERT INTO EMPLOYEE VALUES
```

```
('53030', 'DANNA', 'VO', '413-766-1345', 'DANNA@GMAIL.COM', '17 KATE ROAD', 'GA',
'SAN DIEGO', '73522', 33.00, 'ORDER');
```

```
INSERT INTO EMPLOYEE VALUES
```

```
('54040', 'FROST', 'KERR', '623-412-6218', 'FROST@GMAIL.COM', '12 BOSTON STREET',
'GA', 'BUFFALO', '31340', 29.50, 'DELIVERY');
```

```
INSERT INTO EMPLOYEE VALUES
```

```
('55050', 'RAUL', 'YANG', '723-464-6534', 'RAUL@GMAIL.COM', '24 LENNON LANE', 'GA',
'NASHVILLE', '84324', 32.50, 'DELIVERY');
```

```
INSERT INTO ORDERS VALUES
```

```
('30101', '03-27-2023', 'N', '99710', '51010');
```

```
INSERT INTO ORDERS VALUES
```

```
('30202', '04-15-2023', 'N', '91810', '52020');
```

```
INSERT INTO ORDERS VALUES
```

```
('30303', '07-12-2023', 'N', '93010', '53030');
```

```
INSERT INTO ORDERS VALUES
```

```
('30404', '02-23-2023', 'Y', '96410', '54040');
```

```
INSERT INTO ORDERS VALUES
```

```
('30505', '12-01-2023', 'Y', '94410', '55050');
```

```
INSERT INTO ITEM VALUES
```

```
('10101', 'LOW FAT PIZZA', 'THIN CRUST', 'MEDIUM', 24.99, 05.99);
```

```
INSERT INTO ITEM VALUES  
('10202', 'CHEESE PIZZA', 'THICK CRUST', 'SMALL', 12.99, 03.99);
```

```
INSERT INTO ITEM VALUES  
('10303', 'VEGAN PIZZA', 'FLAT CRUST', 'LARGE', 32.99, 07.99);  
;
```

```
INSERT INTO ITEM VALUES  
('10404', 'MEAT PIZZA', 'CHEESE CRUST', 'MEDIUM', 24.99, 05.99);
```

```
INSERT INTO ITEM VALUES  
('10505', 'KETO PIZZA', 'THIN CRUST', 'MEDIUM', 24.99, 05.99);
```

```
INSERT INTO ORDERLINE VALUES  
('30101', '10101', 2);
```

```
INSERT INTO ORDERLINE VALUES  
('30202', '10202', 1);
```

```
INSERT INTO ORDERLINE VALUES  
('30303', '10303', 1);
```

```
INSERT INTO ORDERLINE VALUES  
('30404', '10404', 5);
```

```
INSERT INTO ORDERLINE VALUES
```

```
USE beta  
DROP TABLE Customer;  
DROP TABLE Employee;  
DROP TABLE Item;  
DROP TABLE Orders;  
DROP TABLE OrderLine;
```

Team Top 10 SQL

1. Display the customer number, customer last name, the order status of each order. In addition, display the employee number and last name for each order.

```
MariaDB [yngah]> SELECT CUSTOMER.CUSTNUM, CUSTOMER.CUSTLNAME, DELIVERED,
-> EMPLOYEE.EMPID, EMPLOYEE.EMPLNAME
-> FROM EMPLOYEE, CUSTOMER, ORDERS
-> WHERE CUSTOMER.CUSTNUM = ORDERS.CUSTID
-> AND ORDERS.EMPID = EMPLOYEE.EMPID;
```

CUSTNUM	CUSTLNAME	DELIVERED	EMPID	EMPLNAME
99710	BROWN	N	51010	BRYCE
91810	SMITH	N	52020	MACY
93010	JAKE	N	53030	DANNA
96410	RAVEN	Y	54040	FROST
94410	CARL	Y	55050	RAUL

5 rows in set (0.000 sec)

2. Display the order numbers for each order which has at least one medium item.

```
MariaDB [yngah]> SELECT ORDERNUM
-> FROM ORDERS
-> WHERE ORDERNUM IN
-> (SELECT ORDERNUM
-> FROM ORDERLINE
-> WHERE ITEMID IN
-> (SELECT ITEMID
-> FROM ITEM
-> WHERE SIZE = 'MEDIUM'));
```

ORDERNUM
30101
30404
30505

3 rows in set (0.001 sec)

3. Create a view on the item table named item details which consist of the item name, cost and the size of the item. Then order the items by their item cost.

```
MariaDB [yngah]> CREATE VIEW ITEM_DETAILS AS
-> SELECT ITEMNAME, SIZE, ITEMPRICE
-> FROM ITEM
-> ORDER BY ITEM COST;
Query OK, 0 rows affected (0.002 sec)
```

```
MariaDB [yngah]> SELECT* FROM ITEM_DETAILS;
+-----+-----+-----+
| ITEMNAME | SIZE | ITEMPRICE |
+-----+-----+-----+
| CHEESE PIZZA | SMALL | 12.99 |
| LOW FAT PIZZA | MEDIUM | 24.99 |
| MEAT PIZZA | MEDIUM | 24.99 |
| KETO PIZZA | MEDIUM | 24.99 |
| VEGAN PIZZA | LARGE | 32.99 |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

```
MariaDB [yngah]> SELECT COUNT (ITEMID) AS NUM_OF_ITEMS FROM ITEM GROUP BY ITEM COST;
+-----+
| NUM_OF_ITEMS |
+-----+
| 1 |
| 3 |
| 1 |
+-----+
3 rows in set (0.000 sec)
```

4. Set the number order to 10 where the item size is small.

```
MariaDB [yngah]> UPDATE ORDERLINE
-> SET NUMORDERED = 10
-> WHERE 'SMALL' = (SELECT SIZE
-> FROM ITEM
-> WHERE ITEM.ITEMID = ORDERLINE.ITEMID);
Query OK, 1 row affected (0.001 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [yngah]> SELECT* FROM ORDERLINE;
+-----+-----+-----+
| ORDERNUM | ITEMID | NUMORDERED |
+-----+-----+-----+
| 30101    | 10101  | 2          |
| 30202    | 10202  | 10         |
| 30303    | 10303  | 1          |
| 30404    | 10404  | 5          |
| 30505    | 10505  | 1          |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

5. Update the first name of every customer which last name is carl to null and then display each customer first name that is not null.

```
MariaDB [yngah]> UPDATE CUSTOMER
-> SET CUSTFNAME = NULL
-> WHERE CUSTLNAME = 'CARL';
Query OK, 1 row affected (0.002 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [yngah]> SELECT* FROM CUSTOMER;
+-----+-----+-----+-----+-----+-----+-----+-----+
| CUSTNUM | CUSTFNAME | CUSTLNAME | PHONE      | EMAIL          | DELIVERYSTREETADDR | STATE | CITY      | ZCODE |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 91810   | LEE       | SMITH     | 423-456-8564 | LEE@GMAIL.COM  | 64 SAMSUNG STREET  | GA    | MIAMI     | 17412 |
| 93010   | BILL      | JAKE      | 141-755-6438 | BILL@GMAIL.COM | 85 CART STREET    | GA    | SAN DIEGO | 73522 |
| 94410   | NULL      | CARL      | 633-123-6151 | FULTON@GMAIL.COM | 71 GROVE STREET   | GA    | NASHVILLE | 84324 |
| 96410   | AL        | RAVEN     | 643-123-9234 | AL@GMAIL.COM   | 67 CRYSTAL STREET | GA    | BUFFALO   | 31340 |
| 99710   | JAMES     | BROWN     | 123-456-8888 | JAMES@GMAIL.COM | 12 JUMP STREET    | GA    | MACON     | 31610 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.000 sec)
```

```
MariaDB [yngah]> SELECT CUSTFNAME
-> FROM CUSTOMER
-> WHERE CUSTFNAME IS NOT NULL;
+-----+
| CUSTFNAME |
+-----+
| LEE       |
| BILL      |
| AL        |
| JAMES     |
+-----+
4 rows in set (0.000 sec)
```


Aryan Merchant Contributions

- User's View**
- Half DDL**
- DBSD**

Aryan Merchant Top 10 SQL

Rough Draft

1. Show Referential Integrity has been established:

Referential integrity is a database concept that guarantees the consistency of the data and the validity of relationships between tables. Attempting to remove a record from the ITEM table that is already referred to in the ORDERLINE table is one way to show referential integrity in the yngah database:

```
DELETE
FROM ITEM
WHERE ItemID = 10101;
```

The above SQL query should fail with an error message indicating that the record cannot be deleted due to referential integrity constraints.

2. Join several tables together:

Using the JOIN keyword and the columns on which we want to join, we can combine multiple tables in the yngah database. For instance, the following SQL query can be used to join the ORDERS, ORDERLINE, and ITEM tables and retrieve the order number, customer name, item name, and quantity:

```
SELECT ORDERS.ORDERNUM, CUSTOMER.CUSTFNAME, CUSTOMER.CUSTLNAME,
ITEM.ITEMNAME, ORDERLINE.NUMORDERED
FROM ORDERS
JOIN CUSTOMER ON ORDERS.CUSTID = CUSTOMER.CUSTNUM
JOIN ORDERLINE ON ORDERS.ORDERNUM = ORDERLINE.ORDERNUM
JOIN ITEM ON ORDERLINE.ITEMID = ITEM.ITEMID;
```

3. Show several layers of Nested Query:

We can carry out intricate operations on the data in the yngah database by using nested queries. For instance, we can use a nested query like this to get the total number of orders for each customer:

```
SELECT CUSTFNAME, CUSTLNAME, (SELECT COUNT(*) FROM ORDERS WHERE
ORDERS.CUSTID = CUSTOMER.CUSTNUM) AS NUMORDERS
FROM CUSTOMER;
SELECT COUNT(*)
FROM Orders, Customer
WHERE Orders.CustNum = Customer.CustomerNum AND
CustFName, CustLName IN (SELECT CustFName, CustLName
FROM Customer);
```

This SQL query first pulls the customer's first and last name from the CUSTOMER table, and then it uses a nested query to count how many orders are related to that particular customer.

4. Make a View:

In a database, a view is a fictitious table that is created based on the output of a SQL query. We can use the following SQL query to create a view in the yngah database that displays the overall revenue generated by each item:

```
CREATE VIEW ITEM_REVENUE AS
SELECT ITEM.ITEMID, ITEM.ITEMNAME, SUM(ORDERLINE.NUMORDERED *
ITEM.ITEMPRICE) AS TOTAL_REVENUE
FROM ITEM
JOIN ORDERLINE ON ITEM.ITEMID = ORDERLINE.ITEMID
GROUP BY ITEM.ITEMID;
```

This SQL statement creates a view called ITEM_REVENUE that groups the results by item ID and computes the total revenue for each item by multiplying the quantity sold (from the ORDERLINE table) by the price (from the ITEM table).

5. An SQL with several compounded conditions:

We can use the following SQL query with a number of compound conditions to retrieve all orders that have been delivered and were placed by clients in either New York or California:

```
SELECT ORDERS.ORDERNUM, ORDERS.ORDERDTE, CUSTOMER.CUSTFNAME,
CUSTOMER.CUSTLNAME, CUSTOMER.STATE
FROM ORDERS
JOIN CUSTOMER ON ORDERS.CUSTID = CUSTOMER.CUSTNUM
WHERE ORDERS.DELIVERED = 'Y' AND (CUSTOMER.STATE = 'NY' OR CUSTOMER.STATE =
'CA');
```

Only orders that have been delivered and were placed by customers in either New York or California are returned by this SQL query because of the WHERE clause's filtering of the results.

6. Using the functions such as (Sum, Count, Avg, Max, Min) together with a Group by Clause:

The GROUP BY clause is used to group the result set based on one or more columns. When combined with aggregate functions like SUM, COUNT, AVG, MAX, and MIN, it allows us to calculate summary information about the groups. Here's an example:

```
SELECT category, SUM(price) AS total_sales
FROM products
GROUP BY category;
```

This query will calculate the total sales for each category of products.

7 Show the use of Having Clause:

The output of a GROUP BY query is filtered using the HAVING clause. Similar to the WHERE clause, the HAVING clause filters groups of rows based on aggregate values rather than individual rows as the WHERE clause does. Here's an illustration:

```
SELECT category, AVG(price) AS avg_price
FROM products
GROUP BY category
HAVING AVG(price) > 50;
```

The average price for each product category will be determined by this query, and any categories where the average price is less than or equal to 50 will be excluded.

8.Using Aliases:

In order to make a query easier to read and comprehend, aliases are used to give temporary names to tables or columns. Here's an illustration:

```
SELECT p.product_name, c.category_name
FROM products AS p
JOIN categories AS c ON p.category_id = c.category_id;
```

Assigning the aliases "p" and "c" to the tables "products" and "categories," respectively, using the AS keyword in this query. The remainder of the query can then make use of these tables' aliases to refer to them.

9.Changing the output titles produce (maybe use ||):

We can use the AS keyword to give the columns new names, changing the output column titles in a query. Here's an illustration:

```
SELECT product_name AS "Product Name", price || ' USD' AS "Price"
FROM products;
```

The "product_name" and "price" columns in this query have new titles thanks to the AS keyword. Also adding the string "USD" to the price column is the concatenation operator (||). "Product Name" and "Price (USD)" are the columns' names in the output.

10 All of the above are SQL techniques that can be used to manipulate and customize the output of a query.

1. You can divide your results into groups based on a particular column or set of columns and then perform aggregate functions on each group by using functions like Sum, Count, Avg, Max, and Min in conjunction with a Group By clause. This can be helpful for data analysis and summarization.
2. To filter groups based on a particular condition, the Having clause is used along with a Group By clause. While it operates similarly to the Where clause, it applies to groups rather than specific rows.

3. In order to make the output of a query easier to read and comprehend, aliases can be used to give columns or tables in the query custom names.
4. Using the AS keyword to assign unique column names or the || operator to concatenate strings will allow you to change the output titles.

Yvan Ngah Contributions

- Half DDL**
- ERD**
- Data Dictionary**
- Half Project Script**

Yvan Ngah Top 10 SQL

1. Display the customer number, customer last name, the order status of each order. In addition, display the employee number and last name for each order.

```
MariaDB [yngah]> SELECT CUSTOMER.CUSTNUM, CUSTOMER.CUSTLNAME, DELIVERED,
-> EMPLOYEE.EMPID, EMPLOYEE.EMPLNAME
-> FROM EMPLOYEE, CUSTOMER, ORDERS
-> WHERE CUSTOMER.CUSTNUM = ORDERS.CUSTID
-> AND ORDERS.EMPID = EMPLOYEE.EMPID;
```

CUSTNUM	CUSTLNAME	DELIVERED	EMPID	EMPLNAME
99710	BROWN	N	51010	BRYCE
91810	SMITH	N	52020	MACY
93010	JAKE	N	53030	DANNA
96410	RAVEN	Y	54040	FROST
94410	CARL	Y	55050	RAUL

```
5 rows in set (0.000 sec)
```

2. Display the order numbers for each order which has at least one medium item.

```
MariaDB [yngah]> SELECT ORDERNUM
-> FROM ORDERS
-> WHERE ORDERNUM IN
-> (SELECT ORDERNUM
-> FROM ORDERLINE
-> WHERE ITEMID IN
-> (SELECT ITEMID
-> FROM ITEM
-> WHERE SIZE = 'MEDIUM'));
```

ORDERNUM
30101
30404
30505

```
3 rows in set (0.001 sec)
```

3. Show referential integrity has been established in the order line table.

```
MariaDB [yngah]> ALTER TABLE ORDERLINE
-> ADD FOREIGN KEY (ITEMID
-> ) REFERENCES ITEM(ITEMID);
Query OK, 5 rows affected (0.031 sec)
Records: 5 Duplicates: 0 Warnings: 0

MariaDB [yngah]> SELECT* FROM ORDERLINE;
+-----+-----+-----+
| ORDERNUM | ITEMID | NUMORDERED |
+-----+-----+-----+
| 30101    | 10101  | 2          |
| 30202    | 10202  | 1          |
| 30303    | 10303  | 1          |
| 30404    | 10404  | 5          |
| 30505    | 10505  | 1          |
+-----+-----+-----+
```

4. Create a view on the item table named item details which consist of the item name, cost and the size of the item. Then order the items by their item cost.

```
MariaDB [yngah]> CREATE VIEW ITEM_DETAILS AS
-> SELECT ITEMNAME, SIZE, ITEMPRICE
-> FROM ITEM
-> ORDER BY ITEMPRICE;
Query OK, 0 rows affected (0.002 sec)

MariaDB [yngah]> SELECT* FROM ITEM_DETAILS;
+-----+-----+-----+
| ITEMNAME | SIZE | ITEMPRICE |
+-----+-----+-----+
| CHEESE PIZZA | SMALL | 12.99 |
| LOW FAT PIZZA | MEDIUM | 24.99 |
| MEAT PIZZA | MEDIUM | 24.99 |
| KETO PIZZA | MEDIUM | 24.99 |
| VEGAN PIZZA | LARGE | 32.99 |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

5. Display the item name, category, and price of each item which is a size medium and the price is greater than 5.

```
MariaDB [yngah]> SELECT ITEMNAME, ITEMCAT, ITEMPRICE
-> FROM ITEM
-> WHERE SIZE = 'MEDIUM' AND ITEMPRICE > 5;
+-----+-----+-----+
| ITEMNAME | ITEMCAT | ITEMPRICE |
+-----+-----+-----+
| LOW FAT PIZZA | THIN CRUST | 24.99 |
| MEAT PIZZA | CHEESE CRUST | 24.99 |
| KETO PIZZA | THIN CRUST | 24.99 |
+-----+-----+-----+
3 rows in set (0.000 sec)
```


6. Display the total amount of items in the database as the number of items, make sure to group the output by the item cost.

```
MariaDB [yngah]> SELECT COUNT(ITEMID) AS NUM_OF_ITEMS FROM ITEM GROUP BY ITEM COST;
+-----+
| NUM_OF_ITEMS |
+-----+
|          1 |
|          3 |
|          1 |
+-----+
3 rows in set (0.000 sec)
```

7. Display the item name, item price as price and item cost as cost where the price of the item price is greater than 10.

```
MariaDB [yngah]> SELECT ITEMNAME
-> , SUM(ITEMPRICE) AS PRICE,
-> SUM(ITEMCOST) AS COST
-> FROM ITEM
-> GROUP BY ITEMNAME
-> HAVING PRICE > 10;
+-----+-----+-----+
| ITEMNAME | PRICE | COST |
+-----+-----+-----+
| CHEESE PIZZA | 12.99 | 3.99 |
| KETO PIZZA | 24.99 | 5.99 |
| LOW FAT PIZZA | 24.99 | 5.99 |
| MEAT PIZZA | 24.99 | 5.99 |
| VEGAN PIZZA | 32.99 | 7.99 |
+-----+-----+-----+
5 rows in set (0.000 sec)
```

8. Display the yearly salary and title of all employees who work as order employees. Display this information as the salary and the position of each employee.

```
MariaDB [yngah]> SELECT EMPLOYEE AS POSITION,
-> EMPHOURLYRATE*40*52 AS SALARY
-> FROM EMPLOYEE
-> WHERE EMPLOYEE = 'ORDER';
+-----+-----+
| POSITION | SALARY |
+-----+-----+
| ORDER | 64480.00 |
| ORDER | 70720.00 |
| ORDER | 68640.00 |
+-----+-----+
3 rows in set (0.000 sec)
```

9. Update the first name of every customer which last name is carl to null and then display each customer's first name that is not null.

```
MariaDB [yngah]> UPDATE CUSTOMER
-> SET CUSTFNAME = NULL
-> WHERE CUSTLNAME = 'CARL';
Query OK, 1 row affected (0.002 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [yngah]> SELECT* FROM CUSTOMER;
+-----+-----+-----+-----+-----+-----+-----+-----+
| CUSTNUM | CUSTFNAME | CUSTLNAME | PHONE | EMAIL | DELIVERYSTREETADDR | STATE | CITY | ZCODE |
+-----+-----+-----+-----+-----+-----+-----+-----+
| 91810 | LEE | SMITH | 423-456-8564 | LEE@GMAIL.COM | 64 SAMSUNG STREET | GA | MIAMI | 17412 |
| 93010 | BILL | JAKE | 141-755-6438 | BILL@GMAIL.COM | 85 CART STREET | GA | SAN DIEGO | 73522 |
| 94410 | NULL | CARL | 633-123-6151 | FULTON@GMAIL.COM | 71 GROVE STREET | GA | NASHVILLE | 84324 |
| 96410 | AL | RAVEN | 643-123-9234 | AL@GMAIL.COM | 67 CRYSTAL STREET | GA | BUFFALO | 31340 |
| 99710 | JAMES | BROWN | 123-456-8888 | JAMES@GMAIL.COM | 12 JUMP STREET | GA | MACON | 31610 |
+-----+-----+-----+-----+-----+-----+-----+-----+
5 rows in set (0.000 sec)
```

```
MariaDB [yngah]> SELECT CUSTFNAME
-> FROM CUSTOMER
-> WHERE CUSTFNAME IS NOT NULL;
+-----+
| CUSTFNAME |
+-----+
| LEE |
| BILL |
| AL |
| JAMES |
+-----+
4 rows in set (0.000 sec)
```

10. Set the number order to 10 where the item size is small.

```
MariaDB [yngah]> UPDATE ORDERLINE
-> SET NUMORDERED = 10
-> WHERE 'SMALL' = (SELECT SIZE
-> FROM ITEM
-> WHERE ITEM.ITEMID = ORDERLINE.ITEMID);
Query OK, 1 row affected (0.001 sec)
Rows matched: 1  Changed: 1  Warnings: 0

MariaDB [yngah]> SELECT* FROM ORDERLINE;
+-----+-----+-----+
| ORDERNUM | ITEMID | NUMORDERED |
+-----+-----+-----+
| 30101 | 10101 | 2 |
| 30202 | 10202 | 10 |
| 30303 | 10303 | 1 |
| 30404 | 10404 | 5 |
| 30505 | 10505 | 1 |
+-----+-----+-----+
5 rows in set (0.000 sec)
```