PIZZACO

DATABASE PROJECT

TEAM BETA

BY: ARYAN MERCHANT

&

YVAN NGAH

Table Of Contents

Content PG#

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 main 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)

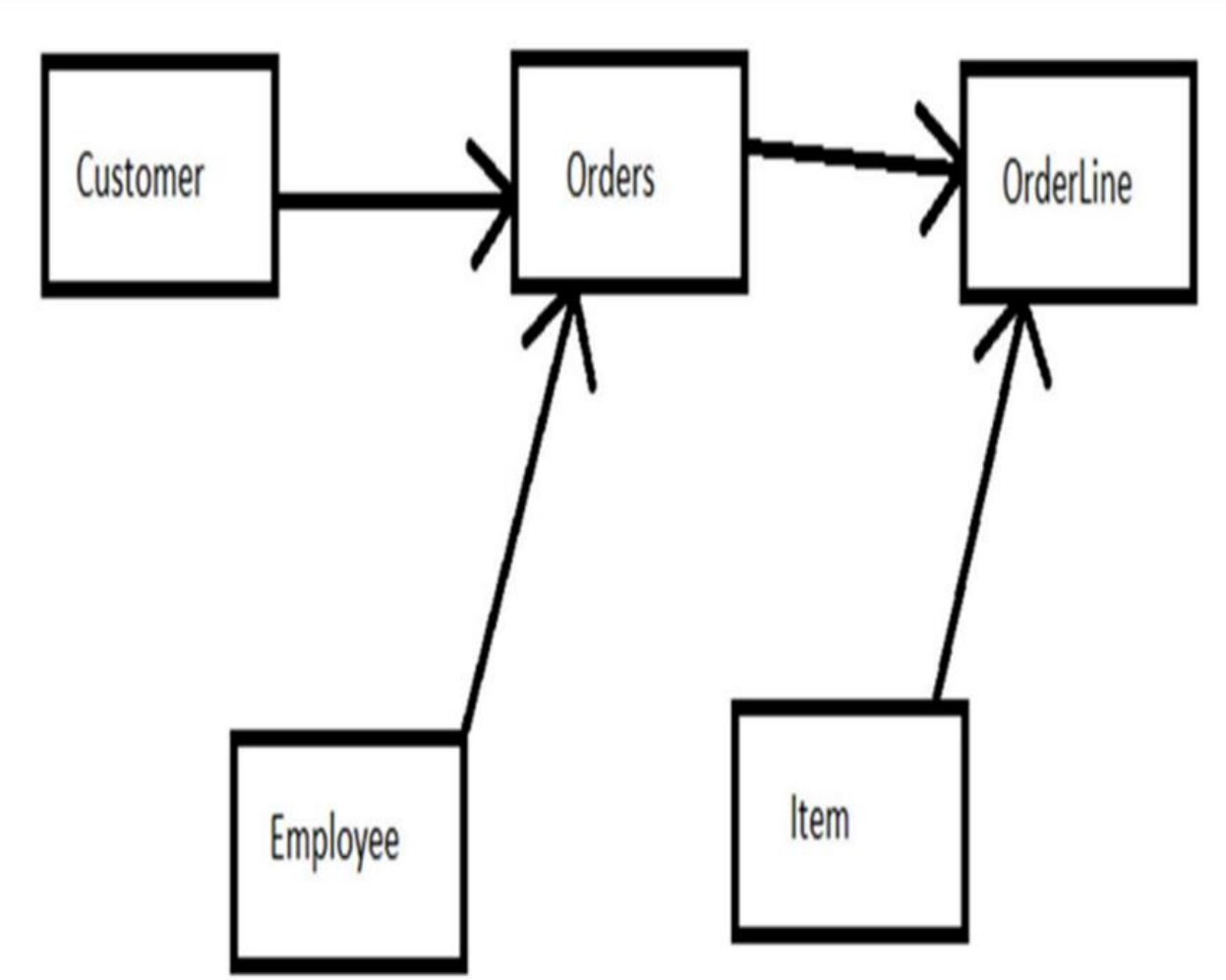
FK: OrderID → Orders

ItemID → Item

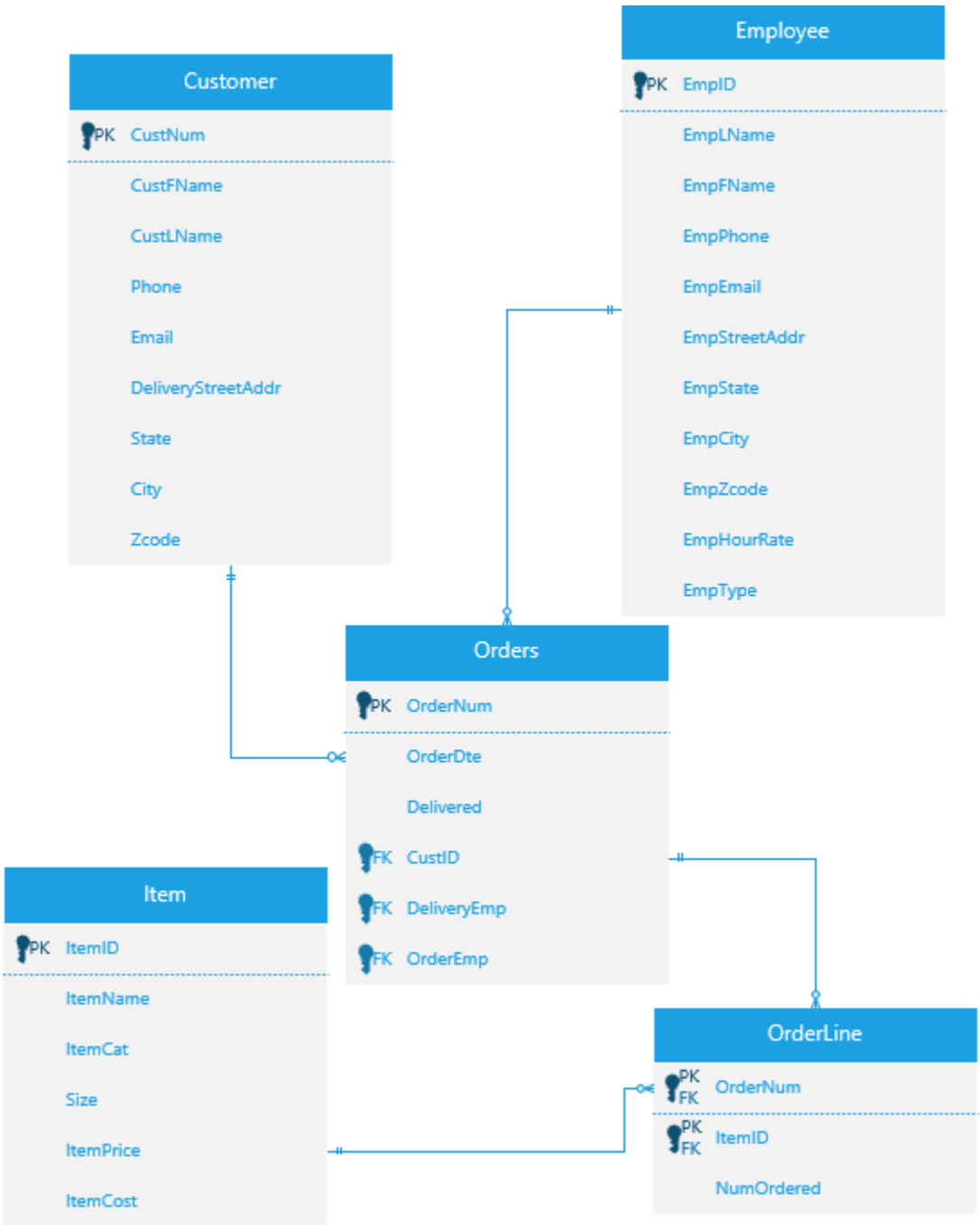
PIZZACO

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

PIZZACO



PIZZACO


Customer
PK CustNum
CustFName
CustLName
Phone
Email
DeliveryStreetAddr
State
City
Zcode
Employee
PK EmpID
EmpLName
EmpFName
EmpPhone
EmpEmail
EmpStreetAddr
EmpState
EmpCity
EmpZcode
EmpHourRate
EmpType
Orders
PK OrderNum
OrderDte
Delivered
FK CustID
FK DeliveryEmp
FK OrderEmp
Item
PK ItemID
ItemName
ItemCat
Size
ItemPrice
ItemCost
OrderLine
PK FK OrderNum
PK FK ItemID
NumOrdered

PIZZACO

```
CREATE
DATABASE
PIZZA;
          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),
            EMPHOURRATE DECIMAL(4,2),
            EMPTYPE 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
('30505', '10505', 1);
```

```
USE
USERNAME


            DROP TABLE CUSTOMER;

            DROP TABLE EMPLOYEE;

            DROP TABLE ITEM;

            DROP TABLE ORDERS;

            DROP TABLE ORDERLINE;
```

Top 10 SQL

Top 10 SQL Cont.

Aryan Merchant Contributions

- User's View

- Half DDL

- DBSD

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;
```

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

SUPPLEMENT