Day 9

ASSIGNMENT 1 : Write a SELECT query to retrieve all columns from a 'customers' table, and modify it to return only the customer name and email address for customers in a specific city.

A)

1. Original Query:

- Start with the original query that retrieves all columns from the 'products' table.

Sql

SELECT * FROM products;

2. Modification to Retrieve Specific Columns:

- Instead of selecting all columns (*),

specify the columns you want to retrieve.

In this case, you want the product name and price.

Sql

SELECT product_name, price

3. Filtering by Category:

- Add a WHERE clause to filter the results to only include products in a specific category. For example, if you want products in the category 'Electronics', you would specify category = 'Electronics'.

sql

WHERE category = 'Electronics'

4. Putting it Together:

- Combine the modified column selection and the WHERE clause to create the final query.

Sql

SELECT product_name, price

FROM products WHERE category = 'Electronics';

This query will return only the product name and price for products in the 'Electronics' category.

By following these steps, you can modify the original query to retrieve specific columns for products in a particular category.

Code :

```
Sql

SELECT product_name,

price FROM products WHERE category = 'Electronics';
```

OUTPUT : 'Electronics'

ASSIGNMENT 2 :

Q) Assignment 2: Craft a query using an INNER JOIN to combine 'orders' and 'customers' tables for customers in a specified region, and a LEFT JOIN to display all customers including those without orders.

A) :

1. Original Query:

- Start with the original query that combines the 'products' and 'orders' tables.

sql

```
SELECT * FROM products JOIN orders ON products.product_id = orders.product_id;
```

2. Filtering by Customer Join Date:

Add a WHERE clause to filter orders by customers who joined before a certain date.

For example, if you want orders from customers who joined before '2020-01-01',

you would specify customers.join_date < '2020-01-01'.

sql

```
WHERE customers.join_date < '2020-01-01'
```

3. INNER JOIN for Specified Join Date:

- Use INNER JOIN to combine the 'products' and 'orders' tables based on the specified join date.

sql

```
SELECT * FROM products INNER JOIN orders ON products.product_id = orders.product_id WHERE customers.join_date < '2020-01-01';
```

4. LEFT JOIN to Include All Products:

- Change the JOIN to LEFT JOIN to include all products, even those without orders.

sql

```
SELECT * FROM products LEFT JOIN orders ON products.product_id = orders.product_id WHERE customers.join_date < '2020-01-01';
```

By following these steps, you can craft a query that combines the 'products' and 'orders' tables to display all products, along with the orders they are associated with, for orders made by customers who joined before a certain date.

CODE :

sql

```
SELECT * FROM products LEFT JOIN orders ON products.product_id = orders.product_id WHERE customers.join_date < '2020-01-01';
```

OUTPUT : The output of executing this SQL query would be a table showing all products, along with any associated orders made by customers who joined before January 1, 2020. Each row would represent a product, with columns for product details and order details, where applicable.

ASSIGNMENT 3 :

Q) Utilize a subquery to find customers who have placed orders above the average order value, and write a UNION query to combine two SELECT statements with the same number of columns.

A):

Part 1:

Utilize a Subquery to Find Customers Who Have Placed Orders Above the Average Order Value First, we need to calculate the average order value. Then, we use this average order value in a subquery to find customers who have placed orders above this value.

Assume we have the following tables:

1. customers table:

- customer_id
- customer_name
- email

2. orders table:

- order_id
- order_date
- customer_id
- total_amount

SQL QUERY

```
SELECT c.customer_id, c.customer_name, c.email FROM customers c

WHERE c.customer_id IN (

SELECT o.customer_id FROM orders o

WHERE o.total_amount > (SELECT

AVG(total_amount) FROM orders))
```

In this query:

- The subquery (SELECT AVG(total_amount) FROM orders) calculates the average order value.
- The inner query SELECT o.customer_id FROM orders o WHERE o.total_amount > ... finds the customer IDs for orders with values above the average.
- The outer query retrieves the customer details for these customer IDs. Part 2: Write a UNION Query to Combine Two SELECT Statements with the Same Number of Columns

- Let's assume we have two different SELECT statements that retrieve customer information based on different criteria. We want to combine their results using the UNION operator.

Full Example:

• First SELECT statement retrieves customers from the customers table who are from a specific city .

• Second SELECT statement retrieves customers from the customers table who have placed orders after a certain date.

QUERY:

First SELECT statement: Customers from a specific city

```
SELECT customer_id, customer_name, email FROM customers

WHERE city = 'New York'

UNION

Second SELECT statement: Customers who have placed orders after a specific date

SELECT c.customer_id, c.customer_name, c.email FROM customers c

JOIN orders o ON c.customer_id = o.customer_i

WHERE o.order_date > '2023-01-01';
```