**Joins in MySQL**

In MySQL, a join is a database operation that combines rows from two or more tables based on a related column between them.

The primary purpose of a join is to retrieve and display data from multiple tables in a single result set, allowing you to access and analyze data that is distributed across different tables within a database.

Here are the key points to understand about joins in MySQL:

1. **Combining Data:**

Joins allow you to combine data from multiple tables into a single result set.

This is particularly useful when data is spread across different tables, and you need to consolidate it for analysis or reporting.

1. **Related Columns:**

To perform a join, you specify one or more columns from each table that have a relationship.

This relationship is typically established through primary keys and foreign keys, where a column in one table matches a column in another table.

1. **Types of Joins:**

MySQL supports various types of joins, including INNER JOIN, LEFT JOIN, RIGHT JOIN, and FULL JOIN.

Each type of join combines rows differently and includes or excludes rows based on the match condition.

1. **Syntax:**

The basic syntax of a join involves specifying the tables to be joined and the join condition using the **ON** clause.

You can also use additional clauses like **WHERE** to further filter the result set.

1. **Result Set:**

The result of a join is a new virtual table that combines columns from both tables based on the specified condition.

This virtual table can be used like any other table in SQL queries.

**Types of Joins**

In MySQL, there are several types of joins that allow you to combine data from multiple tables based on a related column between them.

The main types of joins are as follows:

1. **Inner Join(or Join)**
2. **Outer Join**
   1. **LEFT JOIN (or LEFT OUTER JOIN)**
   2. **RIGHT JOIN (or RIGHT OUTER JOIN)**
   3. **FULL JOIN (or FULL OUTER JOIN)**
3. **SELF JOIN**

**Inner Join**

An INNER JOIN in MySQL is a type of SQL join operation that combines rows from two or more tables based on a related column between them.

The result of an INNER JOIN includes only the rows that have matching values in both tables being joined.

If there's no match between the tables based on the specified join condition, those rows are excluded from the result set.

Here's a more detailed explanation of an INNER JOIN:

* **Combining Data**:

An INNER JOIN is used to retrieve data from multiple tables by matching rows based on a specified condition.

It combines rows from two or more tables into a single result set.

* **Related Columns**:

To perform an INNER JOIN, you specify one or more columns from each table that have a relationship.

This relationship is typically established through primary keys and foreign keys, where a column in one table matches a column in another table.

* **Matching Rows**:

When you perform an INNER JOIN, MySQL looks for rows where the values in the specified columns match between the tables.

Rows that meet this condition are included in the result set, while rows that don't have matching values are excluded.

**Here's the basic syntax of an INNER JOIN**

SELECT

columns

FROM table1

**INNER JOIN** table2 **ON table1.column\_name = table2.column\_name;**

In this syntax:

* **SELECT columns**:

Specifies the columns you want to retrieve from the combined result of the join.

You can select columns from either or both of the tables being joined.

* **table1** and **table2**:

Are the names of the tables you want to join.

**table1** represents the first table, and **table2** represents the second table.

* **ON table1.column\_name = table2.column\_name**:

Defines the join condition by specifying which columns should be compared for equality between the two tables.

Rows where these columns have matching values are included in the result set.

**Customers and Orders**:

-- Create the Customers table

**CREATE TABLE Customers (**

**customer\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**first\_name VARCHAR(50),**

**last\_name VARCHAR(50),**

**email VARCHAR(100),**

**phone\_number VARCHAR(15)**

**);**

-- Create the Orders table

**CREATE TABLE Orders (**

**order\_id INT AUTO\_INCREMENT PRIMARY KEY,**

**customer\_id INT,**

**order\_date DATE,**

**total\_amount DECIMAL(10, 2),**

**FOREIGN KEY (customer\_id) REFERENCES Customers(customer\_id)**

**);**

-- Insert 30 rows into the 'Customers' table

INSERT INTO Customers (first\_name, last\_name, email, phone\_number)

VALUES

('John', 'Smith', 'john.smith@email.com', '123-456-7890'),

('Jane', 'Doe', 'jane.doe@email.com', '987-654-3210'),

('Michael', 'Johnson', 'michael.johnson@email.com', '555-555-5555'),

('Emily', 'Brown', 'emily.brown@email.com', '111-222-3333'),

('David', 'Wilson', 'david.wilson@email.com', '999-888-7777'),

('Sarah', 'Davis', 'sarah.davis@email.com', '444-444-4444'),

('Robert', 'Lee', 'robert.lee@email.com', '777-777-7777'),

('Jennifer', 'Moore', 'jennifer.moore@email.com', '888-888-8888'),

('William', 'Taylor', 'william.taylor@email.com', '555-123-4567'),

('Mary', 'Anderson', 'mary.anderson@email.com', '123-987-6543'),

('Richard', 'White', 'richard.white@email.com', '555-555-5555'),

('Laura', 'Turner', 'laura.turner@email.com', '444-123-7890'),

('Charles', 'Harris', 'charles.harris@email.com', '555-444-3333'),

('Patricia', 'Martinez', 'patricia.martinez@email.com', '123-987-5555'),

('Thomas', 'Clark', 'thomas.clark@email.com', '555-123-7890'),

('Nancy', 'Scott', 'nancy.scott@email.com', '555-555-1234'),

('Daniel', 'Lewis', 'daniel.lewis@email.com', '987-123-4567'),

('Karen', 'Allen', 'karen.allen@email.com', '555-555-5555'),

('Joseph', 'Hall', 'joseph.hall@email.com', '123-123-1234'),

('Linda', 'Turner', 'linda.turner@email.com', '555-987-6543'),

('Steven', 'Williams', 'steven.williams@email.com', '555-555-5555'),

('Susan', 'Jones', 'susan.jones@email.com', '444-987-1234'),

('Matthew', 'Davis', 'matthew.davis@email.com', '555-333-3333'),

('Lisa', 'Smith', 'lisa.smith@email.com', '123-456-7890'),

('James', 'Brown', 'james.brown@email.com', '555-444-5555'),

('Michelle', 'Johnson', 'michelle.johnson@email.com', '555-555-1234'),

('Brian', 'Lee', 'brian.lee@email.com', '777-888-7777'),

('Angela', 'Taylor', 'angela.taylor@email.com', '555-555-5555'),

('Robert', 'Martin', 'robert.martin@email.com', '555-123-9876');

**-- Inserting rows to the Orders table**

-- Customer 1

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(1, '2023-06-01', 120.50),

(1, '2023-06-10', 80.75),

(1, '2023-06-15', 95.25),

(1, '2023-06-20', 110.00),

(1, '2023-06-25', 75.50);

-- Customer 2

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(2, '2023-06-05', 200.00),

(2, '2023-06-12', 55.00),

(2, '2023-06-20', 75.25),

(2, '2023-06-25', 90.00),

(2, '2023-07-02', 130.50);

-- Customer 3

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(3, '2023-06-01', 120.50),

(3, '2023-06-10', 80.75),

(3, '2023-06-15', 95.25),

(3, '2023-06-20', 110.00),

(3, '2023-06-25', 75.50);

-- Customer 4

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(4, '2023-06-01', 120.50),

(4, '2023-06-10', 80.75),

(4, '2023-06-15', 95.25),

(4, '2023-06-20', 110.00),

(4, '2023-06-25', 75.50);

-- Customer 5

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(5, '2023-06-01', 120.50),

(5, '2023-06-10', 80.75),

(5, '2023-06-15', 95.25),

(5, '2023-06-20', 110.00),

(5, '2023-06-25', 75.50);

-- Customer 6

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(6, '2023-06-01', 120.50),

(6, '2023-06-10', 80.75),

(6, '2023-06-15', 95.25),

(6, '2023-06-20', 110.00),

(6, '2023-06-25', 75.50);

-- Customer 7

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(7, '2023-06-01', 120.50),

(7, '2023-06-10', 80.75),

(7, '2023-06-15', 95.25),

(7, '2023-06-20', 110.00),

(7, '2023-06-25', 75.50);

-- Customer 8

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(8, '2023-06-01', 120.50),

(8, '2023-06-10', 80.75),

(8, '2023-06-15', 95.25),

(8, '2023-06-20', 110.00),

(8, '2023-06-25', 75.50);

-- Customer 9

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(9, '2023-06-01', 120.50),

(9, '2023-06-10', 80.75),

(9, '2023-06-15', 95.25),

(9, '2023-06-20', 110.00),

(9, '2023-06-25', 75.50);

-- Customer 10

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(10, '2023-06-01', 120.50),

(10, '2023-06-10', 80.75),

(10, '2023-06-15', 95.25),

(10, '2023-06-20', 110.00),

(10, '2023-06-25', 75.50);

-- Customer 11

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(11, '2023-06-01', 120.50),

(11, '2023-06-10', 80.75),

(11, '2023-06-15', 95.25),

(11, '2023-06-20', 110.00),

(11, '2023-06-25', 75.50);

-- Customer 12

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(12, '2023-06-05', 200.00),

(12, '2023-06-12', 55.00),

(12, '2023-06-20', 75.25),

(12, '2023-06-25', 90.00),

(12, '2023-07-02', 130.50);

-- Customer 13

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(13, '2023-06-01', 120.50),

(13, '2023-06-10', 80.75),

(13, '2023-06-15', 95.25),

(13, '2023-06-20', 110.00),

(13, '2023-06-25', 75.50);

-- Customer 14

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(14, '2023-06-01', 120.50),

(14, '2023-06-10', 80.75),

(14, '2023-06-15', 95.25),

(14, '2023-06-20', 110.00),

(14, '2023-06-25', 75.50);

-- Customer 15

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(15, '2023-06-01', 120.50),

(15, '2023-06-10', 80.75),

(15, '2023-06-15', 95.25),

(15, '2023-06-20', 110.00),

(15, '2023-06-25', 75.50);

-- Customer 16

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(16, '2023-06-01', 120.50),

(16, '2023-06-10', 80.75),

(16, '2023-06-15', 95.25),

(16, '2023-06-20', 110.00),

(16, '2023-06-25', 75.50);

-- Customer 17

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(17, '2023-06-01', 120.50),

(17, '2023-06-10', 80.75),

(17, '2023-06-15', 95.25),

(17, '2023-06-20', 110.00),

(17, '2023-06-25', 75.50);

-- Customer 18

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(18, '2023-06-01', 120.50),

(18, '2023-06-10', 80.75),

(18, '2023-06-15', 95.25),

(18, '2023-06-20', 110.00),

(18, '2023-06-25', 75.50);

-- Customer 19

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(19, '2023-06-01', 120.50),

(19, '2023-06-10', 80.75),

(19, '2023-06-15', 95.25),

(19, '2023-06-20', 110.00),

(19, '2023-06-25', 75.50);

-- Customer 20

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(20, '2023-06-01', 120.50),

(20, '2023-06-10', 80.75),

(20, '2023-06-15', 95.25),

(20, '2023-06-20', 110.00),

(20, '2023-06-25', 75.50);

-- Customer 21

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(21, '2023-06-01', 120.50),

(21, '2023-06-10', 80.75),

(21, '2023-06-15', 95.25),

(21, '2023-06-20', 110.00),

(21, '2023-06-25', 75.50);

-- Customer 22

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(22, '2023-06-05', 200.00),

(22, '2023-06-12', 55.00),

(22, '2023-06-20', 75.25),

(22, '2023-06-25', 90.00),

(22, '2023-07-02', 130.50);

-- Customer 23

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(23, '2023-06-01', 120.50),

(23, '2023-06-10', 80.75),

(23, '2023-06-15', 95.25),

(23, '2023-06-20', 110.00),

(23, '2023-06-25', 75.50);

-- Customer 24

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(24, '2023-06-01', 120.50),

(24, '2023-06-10', 80.75),

(24, '2023-06-15', 95.25),

(24, '2023-06-20', 110.00),

(24, '2023-06-25', 75.50);

-- Customer 25

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(25, '2023-06-01', 120.50),

(25, '2023-06-10', 80.75),

(25, '2023-06-15', 95.25),

(25, '2023-06-20', 110.00),

(25, '2023-06-25', 75.50);

-- Customer 26

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(26, '2023-06-01', 120.50),

(26, '2023-06-10', 80.75),

(26, '2023-06-15', 95.25),

(26, '2023-06-20', 110.00),

(26, '2023-06-25', 75.50);

-- Customer 27

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(27, '2023-06-01', 120.50),

(27, '2023-06-10', 80.75),

(27, '2023-06-15', 95.25),

(27, '2023-06-20', 110.00),

(27, '2023-06-25', 75.50);

-- Customer 28

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(28, '2023-06-01', 120.50),

(28, '2023-06-10', 80.75),

(28, '2023-06-15', 95.25),

(28, '2023-06-20', 110.00),

(28, '2023-06-25', 75.50);

-- Customer 29

INSERT INTO Orders (customer\_id, order\_date, total\_amount)

VALUES

(29, '2023-06-01', 120.50),

(29, '2023-06-10', 80.75),

(29, '2023-06-15', 95.25),

(29, '2023-06-20', 110.00),

(29, '2023-06-25', 75.50);

**Some queries in INNER JOIN**

**1. Retrieve Customer Information with Order Details:**

**This query retrieves customer information along with their order details.**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_date,

Orders.total\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**2. Retrieve Orders Placed by a Specific Customer:**

**To retrieve orders placed by a specific customer (e.g., customer\_id = 1):**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_date,

Orders.total\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id

WHERE Customers.customer\_id = 1;

**3. Calculate Total Order Amount for Each Customer:**

**This query calculates the total order amount for each customer.**

SELECT

Customers.customer\_id,

Customers.first\_name,

Customers.last\_name,

SUM(Orders.total\_amount) AS total\_order\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY

Customers.customer\_id,

Customers.first\_name,

Customers.last\_name;

**4. Retrieve Orders Placed in a Specific Year:**

**To retrieve orders placed in a specific year (e.g., 2023):**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_date,

Orders.total\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id

WHERE YEAR(Orders.order\_date) = 2023;

**5. Retrieve Customers Who Placed Orders:**

**This query retrieves a list of customers who have placed orders:**

INSERT INTO Customers (first\_name, last\_name, email, phone\_number)

VALUES

('Alice', 'Johnson', 'alice.johnson@example.com', '+1-456-789-0123'),

('Michael', 'Williams', 'michael.williams@example.com', '+1-567-890-1234'),

('Sarah', 'Davis', 'sarah.davis@example.com', '+1-678-901-2345'),

('David', 'Brown', 'david.brown@example.com', '+1-789-012-3456'),

('Emily', 'Anderson', 'emily.anderson@example.com', '+1-890-123-4567');

select \* from customers;

SELECT

DISTINCT Customers.first\_name,

Customers.last\_name

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**6. Retrieve Customers Who Haven't Placed Orders:**

**To find customers who haven't placed any orders:**

SELECT

Customers.first\_name,

Customers.last\_name

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

WHERE Orders.order\_id IS NULL;

**7. Retrieve Orders and Customer Information Sorted by Order Date:**

**The following query retrieve orders and customer information sorted by order date:**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_date,

Orders.total\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id

ORDER BY Orders.order\_date;

**8. Retrieve Customers and Their Last Order Date:**

**Find customers and their most recent order date:**

SELECT

Customers.first\_name,

Customers.last\_name,

MAX(Orders.order\_date) AS last\_order\_date

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name;

**9. Retrieve Customers and Their Total Number of Orders:**

**Find customers and the total number of orders they have placed:**

SELECT

Customers.first\_name,

Customers.last\_name,

COUNT(Orders.order\_id) AS total\_orders

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name;

**10. Retrieve Customers Who Placed High-Value Orders:**

**To find customers who have placed orders with a total amount greater than a specific threshold (e.g., $500):**

SELECT

Customers.first\_name,

Customers.last\_name,

SUM(Orders.total\_amount) AS total\_order\_amount

FROM Customers

INNER JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name

HAVING total\_order\_amount > 500;

**Outer Join**

An outer join is a type of SQL join that retrieves records from two tables, even if there is no match between the columns being joined.

It includes unmatched rows from one or both tables in the result set, depending on the type of outer join used.

There are three main types of outer joins:

1. **LEFT OUTER JOIN (or LEFT JOIN):** This type of join returns all the rows from the left table and the matching rows from the right table. If there are no matching rows in the right table, it returns NULL values for the right table's columns.
2. **RIGHT OUTER JOIN (or RIGHT JOIN):** This type of join returns all the rows from the right table and the matching rows from the left table. If there are no matching rows in the left table, it returns NULL values for the left table's columns.
3. **FULL OUTER JOIN (or FULL JOIN):** This type of join returns all the rows when there is a match in either the left or right table. It includes all rows from both tables and returns NULL values for columns where there is no match.

Outer joins are useful when you want to retrieve data from one table and include any related data from another table but don't want to exclude rows from either table if there's no match.

They are particularly helpful for working with tables that have optional relationships or missing data in some cases.

**Examples on left join**

**1. Retrieve Customer Information with Their Orders (including customers with no orders):**

**This query retrieves customer information along with the details of their orders. Customers who have not placed any orders will also be included in the result with NULL values for order-related columns.**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**2. Count the Number of Orders for Each Customer (including customers with no orders):**

**This query counts the number of orders for each customer. It also includes customers who have not placed any orders, showing a count of 0 for them.**

SELECT

Customers.first\_name,

Customers.last\_name,

COUNT(Orders.order\_id) AS order\_count

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name;

**3. Retrieve Customers Who Placed Orders in January 2023 (including customers with no January orders):**

**This query retrieves customers who placed orders in January 2023 along with their order details. Customers who did not place orders in January will still be included in the result.**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

WHERE YEAR(Orders.order\_date) = 2023 AND MONTH(Orders.order\_date) = 1;

**Examples on right join**

**1. Retrieve Customer Information with Their Orders (including orders with no customers):**

**This query retrieves customer information along with the details of their orders. It also includes orders that do not have associated customers, showing NULL values for customer-related columns.**

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Customers

RIGHT JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**2. Count the Number of Orders for Each Customer (including customers with no orders):**

**This query counts the number of orders for each customer. It also includes customers who have not placed any orders, showing a count of 0 for them.**

SELECT

Customers.first\_name,

Customers.last\_name,

COUNT(Orders.order\_id) AS order\_count

FROM Orders

RIGHT JOIN Customers ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name;

**3. Retrieve Orders Placed by Customers with Specific Email Domains (including orders with no matching customers):**

**This query retrieves orders placed by customers whose email addresses have specific domains (e.g., "@appteknow.com"). It includes orders that do not have associated customers with matching email domains.**

-- updates first 15 rows

UPDATE Customers

SET email = CONCAT(SUBSTRING\_INDEX(email, '@', 1), '@appteknow.com')

LIMIT 15;

-- updates last 5 rows

UPDATE Customers

SET email = CONCAT(SUBSTRING\_INDEX(email, '@', 1), '@appteknow.com')

WHERE customer\_id IN (

SELECT customer\_id

FROM (

SELECT customer\_id

FROM Customers

ORDER BY customer\_id DESC

LIMIT 5

) AS subquery

);;

SELECT

Customers.first\_name,

Customers.last\_name,

Customers.email,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Orders

RIGHT JOIN Customers ON Customers.customer\_id = Orders.customer\_id

WHERE Customers.email LIKE '%@appteknow.com';

**Examples of full join**

In MySQL, a FULL JOIN is not directly supported. Instead, you can achieve a similar result by combining a LEFT JOIN and a RIGHT JOIN with a UNION statement. Here's how you can modify the queries to achieve a similar result:

**1. Retrieve All Customers and Their Orders (including unmatched rows from both tables):**

-- LEFT JOIN for customers with orders

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

UNION

-- RIGHT JOIN for orders without customers

SELECT

Customers.first\_name,

Customers.last\_name,

Orders.order\_id,

Orders.order\_date,

Orders.total\_amount

FROM Customers

RIGHT JOIN Orders ON Customers.customer\_id = Orders.customer\_id;

**2. Count the Number of Orders for Each Customer (including customers with no orders and orders with no customers):**

-- LEFT JOIN for customers with orders

SELECT

Customers.first\_name,

Customers.last\_name,

COUNT(Orders.order\_id) AS order\_count

FROM Customers

LEFT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name

UNION

-- RIGHT JOIN for orders without customers

SELECT

Customers.first\_name,

Customers.last\_name,

COUNT(Orders.order\_id) AS order\_count

FROM Customers

RIGHT JOIN Orders ON Customers.customer\_id = Orders.customer\_id

GROUP BY Customers.first\_name, Customers.last\_name;

**Joins on multiple tables**

The Northwind database is a sample database often used for educational purposes in database management systems.

It consists of several tables that represent different aspects of a fictional company's operations, such as customers, orders, products, and suppliers.

To demonstrate how to query this database using at least three tables in MySQL, I'll provide you with some examples along with explanations:

Assuming you have tables like **orders**, **order\_details**, **products**, and **categories**, here are some queries:

**1. Retrieve Product Information with Category Names:**

SELECT

p.ProductName,

p.UnitPrice,

c.CategoryName

FROM products p

JOIN categories c ON p.CategoryID = c.CategoryID;

This query retrieves a list of product names, their unit prices, and their corresponding category names by joining the **products** and **categories** tables on the **CategoryID** column.

**2. Get Customer Orders with Product Details:**

SELECT

o.OrderID,

c.ContactName,

od.Quantity,

p.ProductName

FROM orders o

JOIN customers c ON o.CustomerID = c.CustomerID

JOIN order\_details od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID;

This query retrieves a list of customer orders along with the product details for each order. It joins the **orders**, **customers**, **order\_details**, and **products** tables to achieve this.

**4. List Customers and Their Total Order Amounts:**

This query retrieves a list of customers along with the total order amount for each customer. It joins the customers, orders, and order\_details tables.

SELECT

c.CustomerID,

c.ContactName,

SUM(od.UnitPrice \* od.Quantity) AS TotalOrderAmount

FROM customers c

JOIN orders o ON c.CustomerID = o.CustomerID

JOIN order\_details od ON o.OrderID = od.OrderID

GROUP BY c.CustomerID;

**5. Find Employees and Their Territories:**

**This query retrieves a list of employees along with the territories they are responsible for. It involves joining the employees, employeeterritories, and territories tables**.

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

GROUP\_CONCAT(t.TerritoryDescription) AS Territories

FROM employees e

JOIN employeeterritories et ON e.EmployeeID = et.EmployeeID

JOIN territories t ON et.TerritoryID = t.TerritoryID

GROUP BY e.EmployeeID;

**6. Get Products with Suppliers and Categories:**

**-- This query retrieves a list of products along with their corresponding suppliers and categories. It joins the products, suppliers, and categories tables.**

SELECT

p.ProductName,

s.CompanyName AS Supplier,

c.CategoryName

FROM products p

JOIN suppliers s ON p.SupplierID = s.SupplierID

JOIN categories c ON p.CategoryID = c.CategoryID;

**7. Find Customers with No Orders:**

**-- This query identifies customers who have not placed any orders. It uses a LEFT JOIN to combine the customers and orders tables.**

SELECT

c.CustomerID,

c.ContactName

FROM customers c

LEFT JOIN orders o ON c.CustomerID = o.CustomerID

WHERE o.CustomerID IS NULL;

**8. Retrieve Product Sales by Month:**

**-- This query provides a monthly breakdown of product sales. It joins the orders, order\_details, and products tables while extracting the order date and calculating the total sales for each product.**

SELECT

MONTH(o.OrderDate) AS Month,

p.ProductName,

SUM(od.UnitPrice \* od.Quantity) AS TotalSales

FROM orders o

JOIN order\_details od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

GROUP BY Month, p.ProductName

ORDER BY Month, TotalSales DESC;

**9. List Products with Suppliers, Categories, Employees, and Customers:**

**-- This query retrieves a list of products along with their suppliers, categories, the employees who handled the orders, and the customers who placed those orders.**

SELECT

p.ProductName,

s.CompanyName AS Supplier,

c.CategoryName,

e.FirstName AS Employee,

cu.ContactName AS Customer

FROM products p

JOIN suppliers s ON p.SupplierID = s.SupplierID

JOIN categories c ON p.CategoryID = c.CategoryID

JOIN order\_details od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

JOIN employees e ON o.EmployeeID = e.EmployeeID

JOIN customers cu ON o.CustomerID = cu.CustomerID;

**10. Find Employees with Their Territories, Orders, and Shipped Products:**

**-- This query retrieves a list of employees along with the territories they are responsible for, the orders they handled, and the products that were shipped for those orders.**

SELECT

e.EmployeeID,

e.FirstName,

e.LastName,

GROUP\_CONCAT(t.TerritoryDescription) AS Territories,

o.OrderID,

o.OrderDate,

COUNT(od.ProductID) AS ShippedProducts

FROM employees e

JOIN employeeterritories et ON e.EmployeeID = et.EmployeeID

JOIN territories t ON et.TerritoryID = t.TerritoryID

JOIN orders o ON e.EmployeeID = o.EmployeeID

JOIN order\_details od ON o.OrderID = od.OrderID

GROUP BY e.EmployeeID, o.OrderID;

**11. Get Customers with Their Orders, Products, and Suppliers:**

**-- This query retrieves a list of customers along with the orders they placed, the products ordered, and the suppliers of those products.**

SELECT

cu.CustomerID,

cu.ContactName AS Customer,

o.OrderID,

p.ProductName,

s.CompanyName AS Supplier

FROM customers cu

JOIN orders o ON cu.CustomerID = o.CustomerID

JOIN order\_details od ON o.OrderID = od.OrderID

JOIN products p ON od.ProductID = p.ProductID

JOIN suppliers s ON p.SupplierID = s.SupplierID;

**12. List Products with Categories, Orders, Employees, and Shippers:**

**-- This query retrieves a list of products along with their categories, the orders they were part of, the employees who handled those orders, and the shippers who shipped them.**

SELECT

p.ProductName,

c.CategoryName,

o.OrderID,

e.FirstName AS Employee,

s.CompanyName as Shipper

FROM products p

JOIN categories c ON p.CategoryID = c.CategoryID

JOIN order\_details od ON p.ProductID = od.ProductID

JOIN orders o ON od.OrderID = o.OrderID

JOIN employees e ON o.EmployeeID = e.EmployeeID

JOIN shippers s ON o.ShipVia = s.ShipperID;