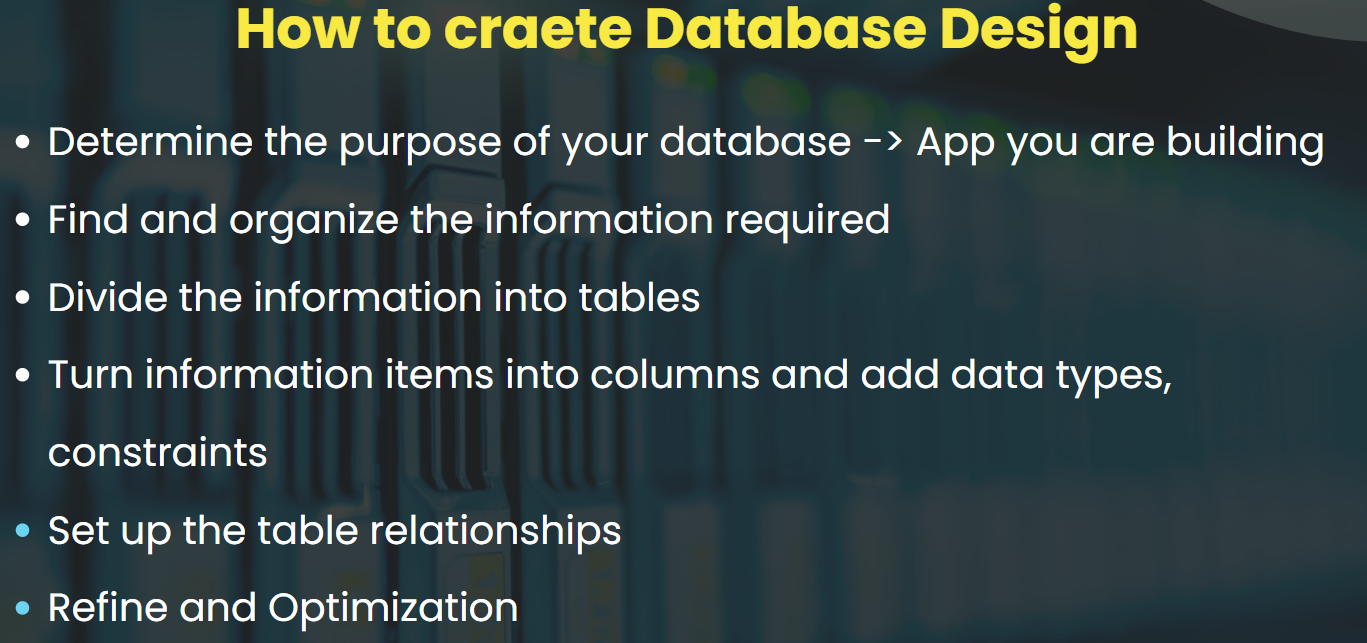
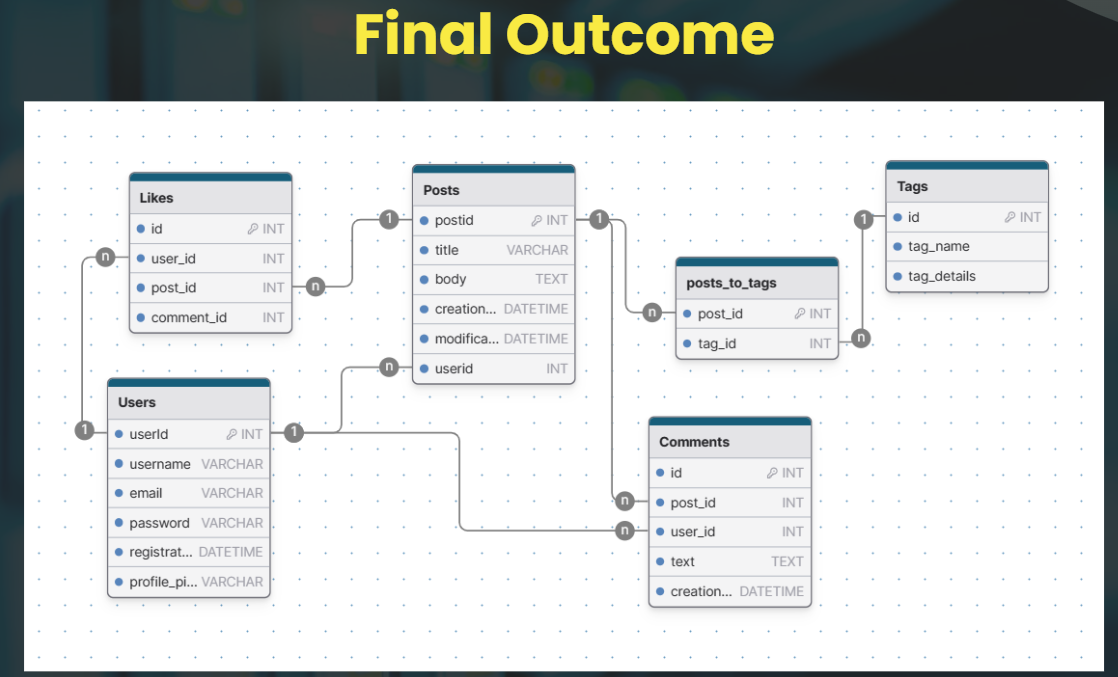


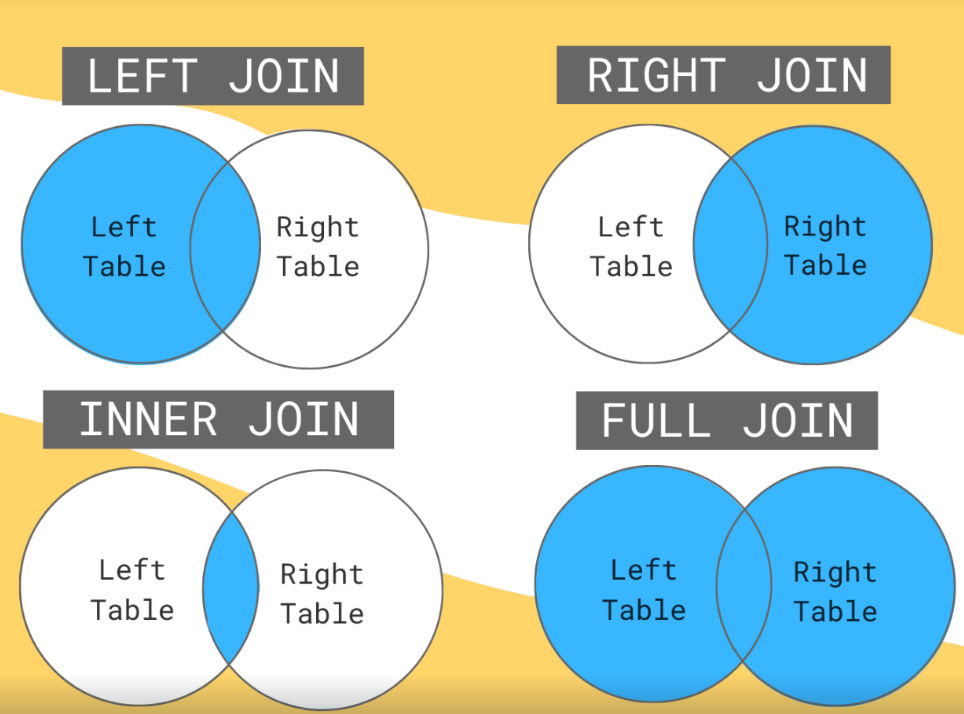
<https://drawsql.app/>

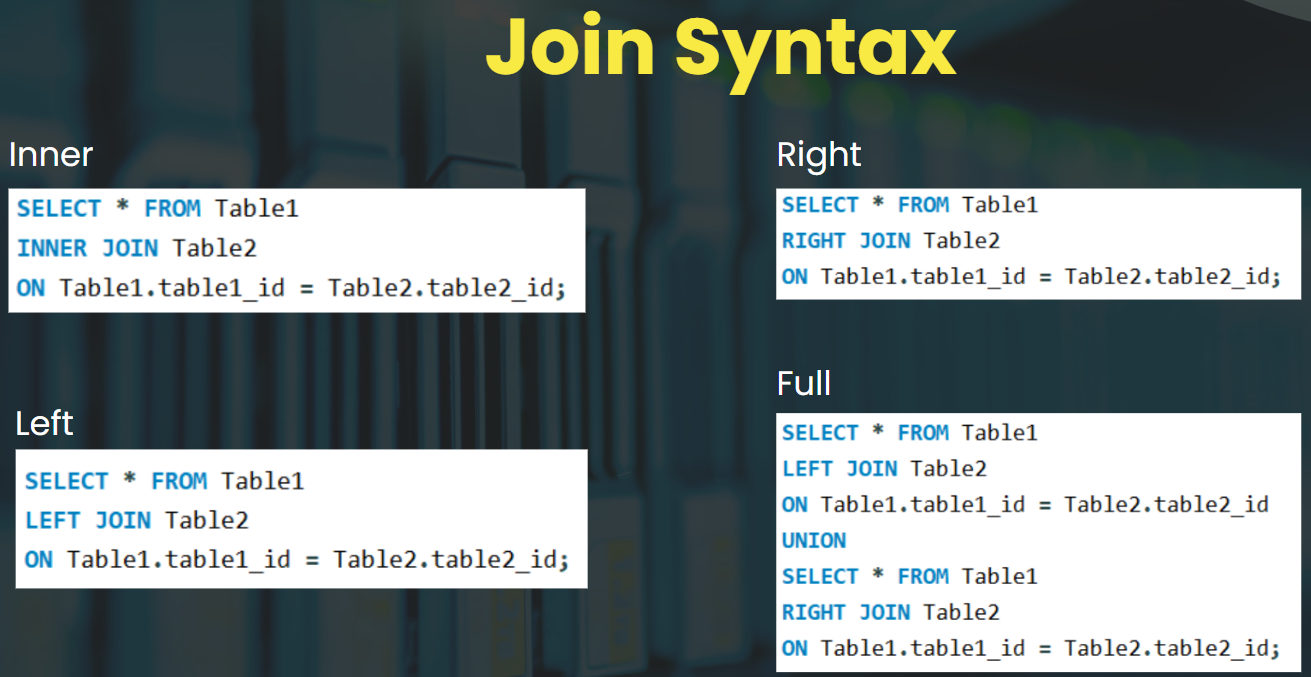
<https://dbdiagram.io/d>



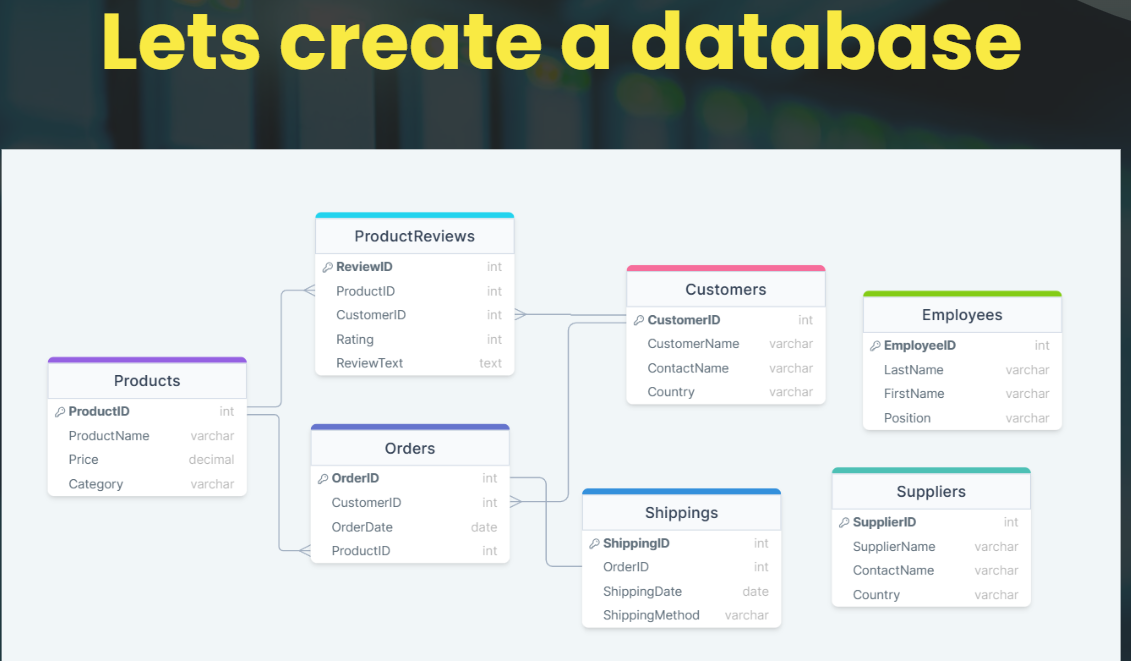












<https://www.hackerrank.com/domains/sql>

# **Step by Step process of Database Design :**

### **The Design Process**

* Determine the purpose of your database (This helps prepare you for the remaining steps)
* Find and organize the information required
* Divide the information into tables
* Turn information items into columns and adding data types , constrains
* Set up the table relationships
* Refine your design

# **Blog Application Database Design :**

Now let's build a full fledged blog application database design with all the knowledge that we have learned

### **Entities**

Users

* UserID (PK) → int
* Username → varchar
* Email - varchar
* PasswordHash - varchar
* RegistrationDate → datetime
* ProfilePicture → varchar

Posts

* PostID (PK) → int
* Title → varchar
* Body → text
* CreationDate → datetime
* ModificationDate → datetime
* UserID (FK) → int

Comments

* CommentID (PK) → int
* PostID (FK) → int
* UserID (FK) → int
* Text → text
* CreationDate → datetime

Likes

* LikeID (PK) → int
* UserID (FK) → int
* PostID (FK, nullable) → int
* CommentID (FK, nullable) → int

Tags

* TagID (PK) → int
* Name → varchar

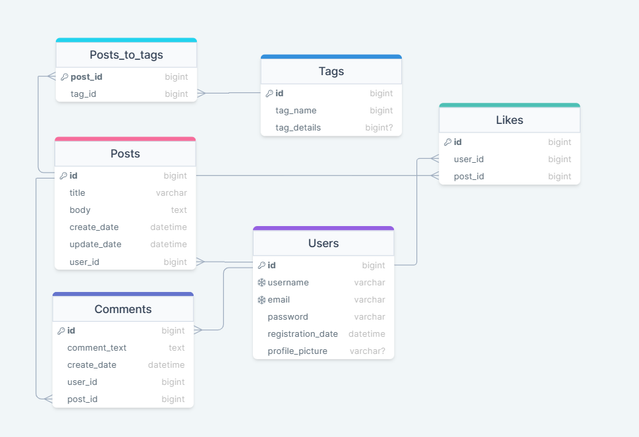
PostToTags

* PostID (FK) → int
* TagID (FK) → int

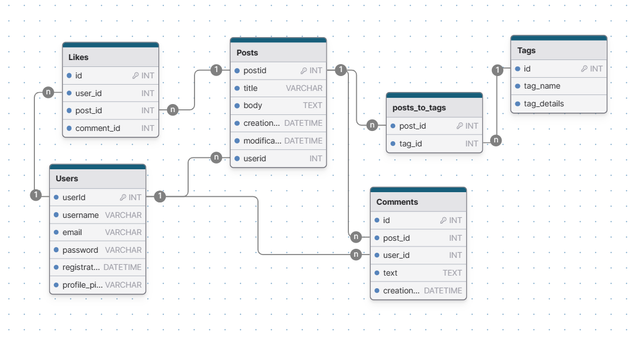
### **Relationships :**

* One-to-Many: Users to Posts, Posts to Comments, Users to Comments, Posts to Likes, Comments to Likes
* Many-to-Many: Posts to Tags (via PostToTags)

### **Diagram - using Drawsql**

****

### **Diagram - using DrawDB**

****

### **Expected Diagram features :**

* Primary and foreign keys are clearly denoted for each table.
* Relationships among tables are illustrated with appropriate cardinality indicators, such as one-to-many for user posts and many-to-many for post tags.
* The diagram is organized in a logical flow, making the data model of the blogging platform easily understandable.

# **Quick recap of JOIN**

### **Defination**

SQL joins are used to combine rows from two or more tables, based on a related column between them. Joins allow us to retrieve data that exists across multiple tables and bring it together in a single result set.

### **Types of JOIN**

* INNER JOIN: Selects records that have matching values in both tables.
* LEFT JOIN: Selects all records from the left table, and the matched records from the right table. If there is no match, the result is NULL on the right side.
* RIGHT JOIN: Selects all records from the right table, and the matched records from the left table. If there is no match, the result is NULL on the left side.
* FULL JOIN: Selects all records when there is a match in either left or right table.
* SELF JOIN: More about this we will learn in - relationship -2 session.

### **Ven Diagram Links**

* INNER JOIN - [Diagram link](https://drive.google.com/file/d/1vdBGUMf15jdVvl-glZB2Ch3vFxuTZO1_/view?usp=sharing)
* LEFT JOIN - [Diagram Link](https://drive.google.com/file/d/1YQAz0buAY55d4giUU9I0-47vJMRI_69z/view?usp=sharing)
* RIGHT JOIN - [Diagram Link](https://drive.google.com/file/d/1Ffxgy3BfCzvD-zE52X_u59OjkZtOjpyC/view?usp=sharing)
* FULL JOIN - [Diagram Link](https://drive.google.com/file/d/1ohoQ0n2AJ5uFLoFEoFAgSUfjXCiT9BuA/view?usp=sharing)

# **SQL Self Join**

### **Description :**

* A self join in SQL is where you join a table with itself. It might sound complex, but it's a handy tool for finding relationships within the same data set.
* The key to a self join is to reference the table twice with different aliases, which act as separate "instances" or views of the same table.

### **Example Scenario :**

Consider a table named Employees

| **EmployeeID** | **EmployeeName** | **ManagerID** |
| --- | --- | --- |
| 1 | Alice | 3 |
| 2 | Bob | 3 |
| 3 | Carol | NULL |
| 4 | Dave | 2 |

To find all employees and their respective managers' names, you would use a self join:

SELECT

E1.EmployeeName AS Employee,

E2.EmployeeName AS Manager

FROM Employees E1

JOIN Employees E2 ON E1.ManagerID = E2.EmployeeID;

This query sets up two aliases for Employees (E1 and E2). It joins the table to itself on the condition that the ManagerID of the first instance matches the EmployeeID of the second instance. The output will be:

| **Employee** | **Manager** |
| --- | --- |
| Alice | Carol |
| Bob | Carol |
| Dave | Bob |

# **Problem Solving with JOIN**

#### **Sample Data**

* Compay Data - [Link](https://drive.google.com/file/d/1opNEku0eYQ0jHuRty2zCS1zcNjCxJ_8p/view?usp=sharing)
* Employees Data - [Link](https://drive.google.com/file/d/1aACdR3zwA4T9RZqarwEBRibaG0tvkc_p/view?usp=sharing)

#### **Print the name of the employee and the company he/she works for**

SELECT e.first\_name, e.last\_name, c.company

FROM employees e

JOIN company c ON e.companyID = c.id;

#### **Retrieve the names and countries of all companies along with the total number of employees in each company:**

SELECT c.company, AVG(e.salary) AS average\_salary

FROM company c

JOIN emplyees e ON c.id = e.companyID

GROUP BY c.company;

#### **Print the company name who has the highest number of employees whose name starts with the letter "S"**

SELECT c.company

FROM company c

JOIN emplyees e ON c.id = e.companyID

WHERE e.first\_name LIKE 'S%'

GROUP BY c.company

ORDER BY COUNT(e.id) DESC

LIMIT 1;

#### **Print the company name and the employee's full name**

SELECT c.company, CONCAT(e.first\_name, ' ', e.last\_name) AS full\_name

FROM company c

JOIN emplyees e ON c.id = e.companyID;

# **Multi Level Join**

### **Introduction:**

Joining multiple tables is a common scenario in database querying, allowing for a more detailed and comprehensive data extraction.

### **Why is it useful?**

It enables the integration of data across several tables, making it possible to analyze relationships and patterns across different segments of data within the database.

### **Table Details**

Customers

* CustomerID (int): A unique identifier for each customer.
* CustomerName (varchar): The name of the customer.
* ContactName (varchar): The contact name of the customer.
* Country (varchar): The country where the customer resides.

Orders

* OrderID (int): A unique identifier for each order.
* CustomerID (int): The identifier for the customer who placed the order. This is a foreign key that references CustomerID in the Customers table.
* OrderDate (date): The date on which the order was placed.
* ProductID (int): The identifier for the product ordered. This is a foreign key that references ProductID in the Products table.

Products

* ProductID (int): A unique identifier for each product.
* ProductName (varchar): The name of the product.
* Price (decimal): The price of the product.
* Category (varchar): The category to which the product belongs.

Shipping

* ShippingID (int): A unique identifier for each shipping entry.
* OrderID (int): The identifier for the order being shipped. This is a foreign key that references OrderID in the Orders table.
* ShippingDate (date): The date the order was shipped.
* ShippingMethod (varchar): The method used for shipping.

Suppliers

* SupplierID (int): A unique identifier for each supplier.
* SupplierName (varchar): The name of the supplier.
* ContactName (varchar): The contact name of the supplier.
* Country (varchar): The country where the supplier is located.

ProductReviews

* ReviewID (int): A unique identifier for the review
* ProductID (int): The product being reviewed. Foreign key to Products table.
* CustomerID (int): The customer who left the review. Foreign key to Customers table.
* Rating (int): The rating given to the product.
* ReviewText (varchar): The text of the review.

Employee

* EmployeeID (int): A unique identifier for each employee.
* LastName (varchar): The last name of the employee.
* FirstName (varchar): The first name of the employee.
* Position (varchar): The position of the employee within the company.

Create Table Query :

-- Customers Table

CREATE TABLE Customers (

CustomerID int PRIMARY KEY,

CustomerName varchar(255),

ContactName varchar(255),

Country varchar(255)

);

-- Orders Table

CREATE TABLE Orders (

OrderID int PRIMARY KEY,

CustomerID int,

OrderDate date,

ProductID int,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Products Table

CREATE TABLE Products (

ProductID int PRIMARY KEY,

ProductName varchar(255),

Price decimal(10,2),

Category varchar(255)

);

-- Shipping Table

CREATE TABLE Shipping (

ShippingID int PRIMARY KEY,

OrderID int,

ShippingDate date,

ShippingMethod varchar(255),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

-- Suppliers Table

CREATE TABLE Suppliers (

SupplierID int PRIMARY KEY,

SupplierName varchar(255),

ContactName varchar(255),

Country varchar(255)

);

-- ProductReviews Table

CREATE TABLE ProductReviews (

ReviewID int PRIMARY KEY,

ProductID int,

CustomerID int,

Rating int,

ReviewText varchar(255),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Employee Table

CREATE TABLE Employee (

EmployeeID int PRIMARY KEY,

LastName varchar(255),

FirstName varchar(255),

Position varchar(255)

);

Insert Sample Data

-- Insert into Customers

INSERT INTO Customers VALUES (1, 'Example Inc.', 'John Doe', 'USA');

INSERT INTO Customers VALUES (2, 'Demo Company', 'Jane Doe', 'Canada');

-- Insert into Products

INSERT INTO Products VALUES (1, 'Laptop', 1200.00, 'Electronics');

INSERT INTO Products VALUES (2, 'Smartphone', 800.00, 'Electronics');

-- Insert into Orders

INSERT INTO Orders VALUES (1, 1, '2023-01-01', 1);

INSERT INTO Orders VALUES (2, 2, '2023-01-02', 2);

-- Insert into Shipping

INSERT INTO Shipping VALUES (1, 1, '2023-01-03', 'FedEx');

INSERT INTO Shipping VALUES (2, 2, '2023-01-04', 'UPS');

-- Insert into Suppliers

INSERT INTO Suppliers VALUES (1, 'Tech Parts Ltd.', 'Alice Smith', 'China');

INSERT INTO Suppliers VALUES (2, 'Gadgets Inc.', 'Bob Johnson', 'Germany');

-- Insert into ProductReviews

INSERT INTO ProductReviews VALUES (1, 1, 1, 5, 'Great product!');

INSERT INTO ProductReviews VALUES (2, 2, 2, 4, 'Very good, but battery life could be better.');

-- Insert into Employee

INSERT INTO Employee VALUES (1, 'Smith', 'Adam', 'Sales Manager');

INSERT INTO Employee VALUES (2, 'Jones', 'Betty', 'Marketing Coordinator');

### **Problems :**

#### **To find out the details of orders placed by each customer including the product names and prices, you would perform a join across these three tables:**

SELECT

Customers.CustomerName,

Orders.OrderID,

Orders.OrderDate,

Products.ProductName,

Products.Price

FROM

Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID

JOIN Products ON Orders.ProductID = Products.ProductID;

#### **To find out the details of each order, including customer name, product name, price, shipping date, and shipping method.**

SELECT

Customers.CustomerName,

Products.ProductName,

Products.Price,

Orders.OrderDate,

Shipping.ShippingDate,

Shipping.ShippingMethod

FROM

Customers

JOIN Orders ON Customers.CustomerID = Orders.CustomerID

JOIN Products ON Orders.ProductID = Products.ProductID

JOIN Shipping ON Orders.OrderID = Shipping.OrderID;

## **Resources - Official Documentation and Other Resources**

* ER Diagram - <https://www.lucidchart.com/pages/er-diagrams>
* Data base design Microsoft - <https://support.microsoft.com/en-gb/office/database-design-basics-eb2159cf-1e30-401a-8084-bd4f9c9ca1f5>
* Database Design Lucidchart - <https://www.lucidchart.com/pages/database-diagram/database-design>
* Database Modeling by Datanamic - <https://www.datanamic.com/support/lt-dez005-introduction-db-modeling.html>
* DrawSQL Popular database Designs - <https://drawsql.app/templates/popular>
* Database Design Tool - Miro - <https://miro.com/>
* Database Design Tool - DrawSQL - <https://drawsql.app/>
* Database Design Tool - DBDiagram - <https://dbdiagram.io/>

-- Switch to database

USE db101;

-- Show tables

SHOW TABLES;

-- Create Customers Table

CREATE TABLE Customers (

CustomerID int PRIMARY KEY,

CustomerName varchar(255),

ContactName varchar(255),

Country varchar(255)

);

-- Create Products Table

CREATE TABLE Products (

ProductID int PRIMARY KEY,

ProductName varchar(255),

Price decimal(10,2),

Category varchar(255)

);

-- Create Orders Table

CREATE TABLE Orders (

OrderID int PRIMARY KEY,

CustomerID int,

OrderDate date,

ProductID int,

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID)

);

-- Create Shipping Table

CREATE TABLE Shipping (

ShippingID int PRIMARY KEY,

OrderID int,

ShippingDate date,

ShippingMethod varchar(255),

FOREIGN KEY (OrderID) REFERENCES Orders(OrderID)

);

-- Create Suppliers Table

CREATE TABLE Suppliers (

SupplierID int PRIMARY KEY,

SupplierName varchar(255),

ContactName varchar(255),

Country varchar(255)

);

-- Create ProductReviews Table

CREATE TABLE ProductReviews (

ReviewID int PRIMARY KEY,

ProductID int,

CustomerID int,

Rating int,

ReviewText varchar(255),

FOREIGN KEY (ProductID) REFERENCES Products(ProductID),

FOREIGN KEY (CustomerID) REFERENCES Customers(CustomerID)

);

-- Create Employee Table

CREATE TABLE Employee (

EmployeeID int PRIMARY KEY,

LastName varchar(255),

FirstName varchar(255),

Position varchar(255)

);

-- Insert into Customers

INSERT INTO Customers VALUES (1, 'Example Inc.', 'John Doe', 'USA');

INSERT INTO Customers VALUES (2, 'Demo Company', 'Jane Doe', 'Canada');

-- Insert into Products

INSERT INTO Products VALUES (1, 'Laptop', 1200.00, 'Electronics');

INSERT INTO Products VALUES (2, 'Smartphone', 800.00, 'Electronics');

-- Insert into Orders

INSERT INTO Orders VALUES (1, 1, '2023-01-01', 1);

INSERT INTO Orders VALUES (2, 2, '2023-01-02', 2);

-- Insert into Shipping

INSERT INTO Shipping VALUES (1, 1, '2023-01-03', 'FedEx');

INSERT INTO Shipping VALUES (2, 2, '2023-01-04', 'UPS');

-- Insert into Suppliers

INSERT INTO Suppliers VALUES (1, 'Tech Parts Ltd.', 'Alice Smith', 'China');

INSERT INTO Suppliers VALUES (2, 'Gadgets Inc.', 'Bob Johnson', 'Germany');

-- Insert into ProductReviews

INSERT INTO ProductReviews VALUES (1, 1, 1, 5, 'Great product!');

INSERT INTO ProductReviews VALUES (2, 2, 2, 4, 'Very good, but battery life could be better.');

-- Insert into Employee

INSERT INTO Employee VALUES (1, 'Smith', 'Adam', 'Sales Manager');

INSERT INTO Employee VALUES (2, 'Jones', 'Betty', 'Marketing Coordinator');

-- Display data from all tables

SELECT \* FROM Customers;

SELECT \* FROM Products;

SELECT \* FROM Orders;

SELECT \* FROM Shipping;

SELECT \* FROM Suppliers;

SELECT \* FROM ProductReviews;

SELECT \* FROM Employee;

-- Example Queries:

-- 1. Find details of orders placed by each customer including the product names and prices where category is 'Laptop'

SELECT s.CustomerName, o.OrderID, p.ProductName, p.Price

FROM Customers s

JOIN Orders o ON s.CustomerID = o.CustomerID

JOIN Products p ON p.ProductID = o.ProductID

WHERE p.Category = 'Electronics' AND p.ProductName = 'Laptop';

-- 2. Find details of each order, including customer name, product name, price, shipping date, and shipping method

SELECT s.CustomerName, p.ProductName, p.Price, sh.ShippingDate, sh.ShippingMethod

FROM Orders o

JOIN Customers s ON o.CustomerID = s.CustomerID

JOIN Products p ON o.ProductID = p.ProductID

JOIN Shipping sh ON o.OrderID = sh.OrderID;

-- 3. Find all products along with their reviews and customer names who reviewed them

SELECT p.ProductName, r.Rating, r.ReviewText, c.CustomerName

FROM ProductReviews r

JOIN Products p ON r.ProductID = p.ProductID

JOIN Customers c ON r.CustomerID = c.CustomerID;

-- 4. Find suppliers and the products they supply (assuming a relation exists in a real scenario)

-- Since no direct relation is defined, this example assumes a potential supplier-product table:

-- SELECT s.SupplierName, p.ProductName

-- FROM Suppliers s

-- JOIN SupplierProducts sp ON s.SupplierID = sp.SupplierID

-- JOIN Products p ON sp.ProductID = p.ProductID;

-- 5. Find employees and their corresponding orders handled (assuming a relation exists in a real scenario)

-- Assuming we have an Orders table with an EmployeeID:

-- SELECT e.FirstName, e.LastName, o.OrderID, o.OrderDate

-- FROM Employee e

-- JOIN Orders o ON e.EmployeeID = o.EmployeeID;

-- Additional joins and examples can be added based on specific requirements or additional table relations.