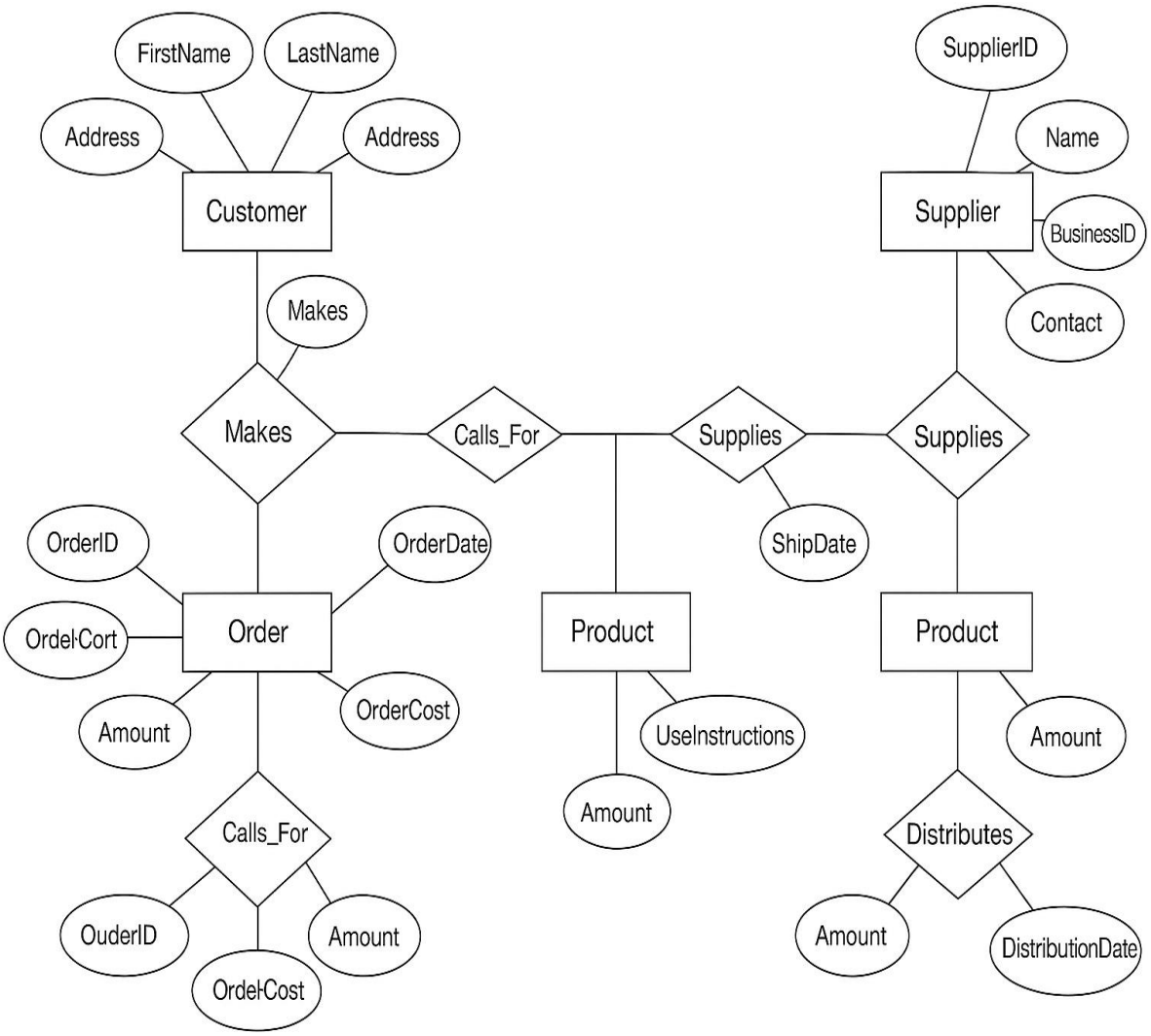
ER MODEL FOR AMAZON WEB SERVICES

Project Summary: Amazon-Style Supply Chain Database Model

This project presents an Entity-Relationship (ER) model and its corresponding SQL implementation for simulating a simplified Amazon-style e-commerce supply chain system. The goal is to capture the interactions between core business entities such as **Customers**, **Suppliers**, **Shippers**, **Products**, and **Orders**.

The system is designed to:

- Track **customer orders and billing details product inventory and supplier shipments**
- Record **distribution logistics** by shippers
- Enable data analysis through **aggregate SQL functions** to derive business insights.



TO CREATE AN ER MODEL , WE WILL WRITE THESE CODE IN SQL

-- CUSTOMER TABLE:-

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY AUTO_INCREMENT,
    FirstName VARCHAR(50),
    LastName VARCHAR(50),
    Name VARCHAR(100),
    Address TEXT,
    BillingInfo TEXT
);
```

-- Insert into Customer:-

```
INSERT INTO Customer (FirstName, LastName, Name, Address, BillingInfo)
VALUES
('John', 'Doe', 'John Doe', '123 Main St, NY', 'Visa 1234'),
('Jane', 'Smith', 'Jane Smith', '456 Elm St, CA', 'MasterCard 5678');
```

Result:- The Customer table stores customer details, including their name, address, and billing information.

-- Supplier Table:-

```
CREATE TABLE Supplier (
    SupplierID INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(100),
    BusinessID VARCHAR(50),
    Contact VARCHAR(100),
    Address TEXT
);
```

-- Insert into Supplier

```
INSERT INTO Supplier (Name, BusinessID, Contact, Address)
VALUES
('Global Supplies Co.', 'BUS123', 'supplier1@example.com', '789 Oak St, TX'),
('TechSource Inc.', 'BUS456', 'supplier2@example.com', '321 Pine St, FL');
```

Result:- The Supplier table stores supplier information, including the supplier's name, business ID, contact details, and address.

-- SHIPPER TABLE:-

```
CREATE TABLE Shipper (
    ShipperID INT PRIMARY KEY AUTO_INCREMENT,
    TransportationID VARCHAR(50),
    Contact VARCHAR(100)
);
```

-- Insert into Shipper

```
INSERT INTO Shipper (TransportationID, Contact)
VALUES
('TRANS001', 'shipper1@example.com'),
('TRANS002', 'shipper2@example.com');
```

Result:- The Shipper table records shipper details, including a transportation ID and contact information for each shipper.

-- PRODUCT TABLE:-

```
CREATE TABLE Product (
    ProductID INT PRIMARY KEY AUTO_INCREMENT,
    ItemID VARCHAR(50),
    UseInstructions TEXT
);
```

-- Insert into Product

```
INSERT INTO Product (ItemID, UseInstructions)
VALUES
('ITM001', 'Use with charger only'),
('ITM002', 'Handle with care – fragile');
```

Result:- The Product table stores product details, including an item ID and usage instructions for each product.

-- ORDER TABLE:-

```
CREATE TABLE `Order` (
    OrderID INT PRIMARY KEY AUTO_INCREMENT,
    TotalCost DECIMAL(10,2),
    CustomerID INT,
```

```
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID)
);
```

-- Insert into Order

```
INSERT INTO `Order` (TotalCost, CustomerID)
VALUES
(250.00, 1),
(150.00, 2);
```

Result:- The Order table records each order's total cost and associates it with a customer using a foreign key.

-- ORDER_PRODUCT RELATIONSHIP TABLE (Calls_For)

```
CREATE TABLE Order_Product (
    OrderID INT,
    ProductID INT,
    Amount INT,
    PRIMARY KEY (OrderID, ProductID),
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

-- Insert into Order_Product

```
INSERT INTO Order_Product (OrderID, ProductID, Amount)
VALUES
(1, 1, 2),
(1, 2, 1),
(2, 2, 3);
```

Result:- The Order_Product table links orders to products, specifying the quantity of each product in the order.

-- CUSTOMER_ORDER RELATIONSHIP (Makes)

```
CREATE TABLE Customer_Order (
    CustomerID INT,
    OrderID INT,
    Amount DECIMAL(10,2),
```

```
    OrderDate DATE,
    PRIMARY KEY (CustomerID, OrderID),
    FOREIGN KEY (CustomerID) REFERENCES Customer(CustomerID),
    FOREIGN KEY (OrderID) REFERENCES `Order`(OrderID)
);
```

-- Insert into Customer_Order

```
INSERT INTO Customer_Order (CustomerID, OrderID, Amount, OrderDate)
VALUES
(1, 1, 250.00, '2024-01-15'),
(2, 2, 150.00, '2024-02-05');
```

Result:- The Customer_Order table links customers to their orders, recording the amount and order date for each transaction.

-- SUPPLIER_PRODUCT RELATIONSHIP (Supplies)

```
CREATE TABLE Supplier_Product (
    SupplierID INT,
    ProductID INT,
    Amount INT,
    ShipDate DATE,
    PRIMARY KEY (SupplierID, ProductID),
    FOREIGN KEY (SupplierID) REFERENCES Supplier(SupplierID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
```

-- Insert into Supplier_Product

```
INSERT INTO Supplier_Product (SupplierID, ProductID, Amount, ShipDate)
VALUES
(1, 1, 100, '2024-01-10'),
(2, 2, 200, '2024-01-12');
```

Result:- The Supplier_Product table records which supplier supplies which product, along with the quantity and shipment date.

-- SHIPPER_PRODUCT RELATIONSHIP (Distributes)

```
CREATE TABLE Shipper_Product (
    ShipperID INT,
```

```
    ProductID INT,
    Amount INT,
    DistributionDate DATE,
    PRIMARY KEY (ShipperID, ProductID),
    FOREIGN KEY (ShipperID) REFERENCES Shipper(ShipperID),
    FOREIGN KEY (ProductID) REFERENCES Product(ProductID)
);
-- Insert into Shipper_Product
INSERT INTO Shipper_Product (ShipperID, ProductID, Amount, DistributionDate)
VALUES
(1, 1, 50, '2024-01-17'),
(2, 2, 150, '2024-01-18');
```

Result:- The Shipper_Product table records which shipper distributes which product, along with the quantity and distribution date.

❖ We use SQL aggregate functions (SUM, AVG, COUNT, MAX, MIN) with GROUP BY to analyze customer behavior, order trends, supplier contributions, and product distribution to derive insights from our Amazon-like database.

➢ Total Amount of Products Ordered per Order

```
SELECT
    OrderID,
    SUM(Amount) AS TotalItemsOrdered
FROM Order_Product
GROUP BY OrderID;
```

- Result:- This query returns the total quantity of products ordered per OrderID by summing the Amount (which typically means quantity of each product in that order).

➢ Total Spend per Customer

```
SELECT
    C.CustomerID,
    C.Name,
```

```
    SUM(O.TotalCost) AS TotalSpend
FROM Customer C
JOIN `Order` O ON C.CustomerID = O.CustomerID
GROUP BY C.CustomerID, C.Name;
```


- Result:- This query returns the total amount each customer has spent, by summing the TotalCost of all their orders.

➢ Number of Orders per Customer

```
SELECT
    C.CustomerID,
    C.Name,
    COUNT(O.OrderID) AS NumberOfOrders
FROM Customer C
JOIN `Order` O ON C.CustomerID = O.CustomerID
GROUP BY C.CustomerID, C.Name;
```

- Result:- This query gives you the number of orders placed per customer, assuming each row in the Order table represents one order.

➢ Average Quantity Supplied per Supplier

```
SELECT
    S.SupplierID,
    S.Name,
    AVG(SP.Amount) AS AvgSuppliedAmount
FROM Supplier S
JOIN Supplier_Product SP ON S.SupplierID = SP.SupplierID
GROUP BY S.SupplierID, S.Name;
```

- Result:- This query joins the Supplier and Supplier_Product tables to compute the average quantity each supplier provided.

➢ Total Products Distributed per Product

```
SELECT
    ProductID,
    SUM(Amount) AS TotalDistributed
FROM Shipper_Product
```

```
GROUP BY ProductID;
```

- Result:- This query calculates the total amount of each product distributed by summing the Amount from the Shipper_Product table.

➢ Max Quantity Distributed by Shipper

```
SELECT
    ShipperID,
    MAX(Amount) AS MaxDistributedAmount
FROM Shipper_Product
GROUP BY ShipperID;
```

- Result:- This query calculates the maximum quantity of a product distributed by each shipper by finding the highest value of Amount from the Shipper_Product table.

We will us windows functions

1.) ROW_NUMBER()

```
SELECT
    CustomerID,
    OrderID,
    OrderDate,
    ROW_NUMBER() OVER (PARTITION BY CustomerID ORDER BY OrderDate) AS RowNum
FROM Customer_Order;
```

- Result:- Assigns a unique row number within each partition (customer), ordered by order date.

2.) RANK()

```
SELECT
    CustomerID,
    SUM(Amount) AS TotalAmount,
    RANK() OVER (ORDER BY SUM(Amount) DESC) AS CustomerRank
FROM Customer_Order
GROUP BY CustomerID;
```

- Result:- Ranks orders based on the total amount spent by customers, with gaps for tied values.

3.) DENSE_RANK()

```
SELECT
    CustomerID,
```

```
    SUM(Amount) AS TotalAmount,
    DENSE_RANK() OVER (ORDER BY SUM(Amount) DESC) AS DenseCustomerRank
FROM Customer_Order
GROUP BY CustomerID;
```


- Result:- Ranks customers like RANK(), but without gaps for tied values.

4.) LAG()

```
SELECT
    CustomerID,
    OrderID,
    Amount,
    LAG(Amount) OVER (ORDER BY OrderDate) AS PreviousOrderAmount
FROM Customer_Order;
```

- Result:- Shows the amount of the previous order for each customer.

Conclusion

This project presented a simplified ER model and SQL database for an Amazon-style e-commerce system. It effectively captures key entities and their interactions, enabling data analysis of customer behavior, order trends, and supply chain performance. The model provides a strong foundation for building more advanced, real-world solutions.