DBMS Project:Indian E-commerce

Indian e-commerce refers to the buying and selling of goods and services online within the Indian market. It has grown exponentially over the past decade due to increased internet penetration, smartphone usage, and the convenience of online shopping. The Indian e-commerce sector encompasses a wide range of industries, including retail, travel, food delivery, education, entertainment, and more.

Key Aspects of Indian E-commerce:

1. **Growth and Expansion**: The Indian e-commerce market has witnessed remarkable growth, with estimates suggesting that it could become one of the largest markets in the world. The sector is projected to reach over USD 350 billion by 2030. Factors driving this growth include improved digital infrastructure, government initiatives like Digital India, and the rise of new-age technology companies.
2. **Popular E-commerce Platforms**:
    - **Amazon India** and **Flipkart** dominate the retail segment, offering a wide range of products such as electronics, fashion, groceries, and more.
    - **Myntra**, **Ajio**, and **Nykaa** specialize in fashion and beauty.
    - **BigBasket** and **Grofers** focus on grocery delivery.
    - **Swiggy** and **Zomato** lead in food delivery services.
3. **Government Initiatives**: The Indian government has introduced initiatives like the **Digital India** campaign and policies to promote cashless transactions, boost MSMEs (Micro, Small, and Medium Enterprises), and regulate the e-commerce space. Schemes like **Startup India** have encouraged entrepreneurs to venture into the e-commerce domain.
4. **Challenges**:
    - **Logistics**: Despite rapid urbanization, India has a vast rural market, and logistics challenges remain in delivering to remote areas.
    - **Trust Issues**: Some consumers still have concerns about online payment security, product authenticity, and customer service.
    - **Competition**: The sector is highly competitive, with both global and domestic players vying for market share.
5. **Trends**:
    - **Mobile Commerce**: The majority of e-commerce transactions in India happen via mobile devices due to the affordability and widespread use of smartphones.
    - **Payment Systems**: The rise of digital wallets (e.g., **Paytm**, **PhonePe**, **Google Pay**) and UPI (Unified Payments Interface) has transformed the payment landscape, making online transactions easier.

- **Social Commerce**: Social media platforms like **Instagram** and **WhatsApp** are increasingly being used for e-commerce through business pages and direct selling.

6. **Consumer Behavior**: Indian consumers are highly price-sensitive, so discounts and festive sales (like Flipkart's **Big Billion Days** and Amazon's **Great Indian Festival**) play a critical role in driving e-commerce traffic. Personalization, customer reviews, and fast delivery are significant factors in consumer decisions.

Future of Indian E-commerce:

With the growing number of internet users and evolving customer preferences, Indian e-commerce is set for significant growth. Technologies like artificial intelligence (AI), machine learning (ML), and augmented reality (AR) are expected to transform customer experiences, while innovative models like **hyperlocal commerce**, **subscription services**, and **quick commerce** will continue to evolve.

In summary, Indian e-commerce is a dynamic and fast-growing sector that has transformed how consumers shop and businesses operate. It is a key driver of the digital economy, with great potential for further innovation and expansion.

Introduction

This project demonstrates various SQL concepts implemented for an Indian e-commerce domain. We will create a user-defined database with tables related to customers, orders, products, and transactions. The project will cover SQL operations such as DDL, DML, TCL, joins, constraints, views, subqueries, and set operators.

ER Diagram

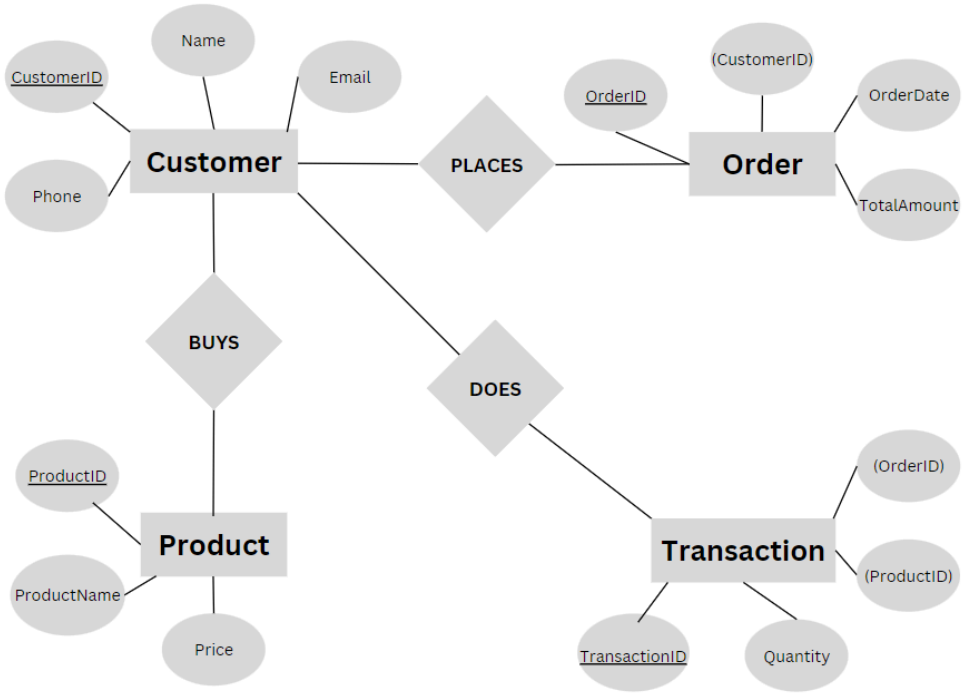
Below is the ER Diagram of the Indian e-commerce system showing the relationships between customers, orders, products, and transactions.

Entities:

1. **Customers** – Includes customer details like customer_id, name, email, contact details, and location.
2. **Orders** – Captures order-related information such as order_id, customer_id (foreign key), order_date, total_amount, and payment_status.
3. **Products** – Contains product-specific information like product_id, name, price, category, and stock_quantity.
4. **Transactions** – Stores payment information, referencing order_id as a foreign key.

The relationships between these entities are as follows:

- **Customers** and **Orders**: One-to-Many (A customer can place multiple orders).
- **Orders** and **Products**: Many-to-Many (Each order can have multiple products, and each product can be part of multiple orders).
- **Orders** and **Transactions**: One-to-One (Each order has one associated transaction).



Schema

The schema of the database is defined as follows:
Customers(customer_id, name, email, phone_number, city, registration_date)
Orders(order_id, customer_id (FK), order_date, total_amount, payment_status)
Products(product_id, name, category, price, stock_quantity)
Transactions(transaction_id, order_id (FK), payment_method, payment_date)

SQL Programs

1. Table Creation and Data Insertion

SQL Commands:

-- Parent Table: Customers

```
CREATE TABLE Customers (
   customer_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   email VARCHAR(100) UNIQUE,
   phone_number VARCHAR(15),
   city VARCHAR(50),
   registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

```
mysql> DESCRIBE Customers;
+-------------------+--------------+------+-----+-------------------+-------------------+
| Field             | Type         | Null | Key | Default           | Extra             |
+-------------------+--------------+------+-----+-------------------+-------------------+
| customer_id       | int          | NO   | PRI | NULL              | auto_increment    |
| name              | varchar(100) | NO   |     | NULL              |                   |
| email             | varchar(100) | YES  | UNI | NULL              |                   |
| phone_number      | varchar(15)  | YES  |     | NULL              |                   |
| city              | varchar(50)  | YES  |     | NULL              |                   |
| registration_date | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+-------------------+--------------+------+-----+-------------------+-------------------+
6 rows in set (0.00 sec)
```

-- Child Table 1: Orders

```
CREATE TABLE Orders (
   order_id INT AUTO_INCREMENT PRIMARY KEY,
   customer_id INT,
   order_date DATE,
   total_amount DECIMAL(10, 2),
```

```
    payment_status VARCHAR(50),

    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

```
mysql> DESCRIBE Orders;
+----------------+---------------+------+-----+---------+----------------+
| Field          | Type          | Null | Key | Default | Extra          |
+----------------+---------------+------+-----+---------+----------------+
| order_id       | int           | NO   | PRI | NULL    | auto_increment |
| customer_id    | int           | YES  | MUL | NULL    |                |
| order_date     | date          | YES  |     | NULL    |                |
| total_amount   | decimal(10,2) | YES  |     | NULL    |                |
| payment_status | varchar(50)   | YES  |     | NULL    |                |
+----------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

-- Child Table 2: Products

```
CREATE TABLE Products (

    product_id INT AUTO_INCREMENT PRIMARY KEY,

    name VARCHAR(100) NOT NULL,

    category VARCHAR(50),

    price DECIMAL(10, 2),

    stock_quantity INT CHECK (stock_quantity >= 0)
);
```

```
mysql> DESCRIBE Products;
+----------------+---------------+------+-----+---------+----------------+
| Field          | Type          | Null | Key | Default | Extra          |
+----------------+---------------+------+-----+---------+----------------+
| product_id     | int           | NO   | PRI | NULL    | auto_increment |
| name           | varchar(100)  | NO   |     | NULL    |                |
| category       | varchar(50)   | YES  |     | NULL    |                |
| price          | decimal(10,2) | YES  |     | NULL    |                |
| stock_quantity | int           | YES  |     | NULL    |                |
+----------------+---------------+------+-----+---------+----------------+
5 rows in set (0.00 sec)
```

Example Records:

INSERT INTO Customers (name, email, phone_number, city) VALUES
('Amit Kumar', 'amit.k@example.com', '9123456789', 'Delhi'),
('Sneha Verma', 'sneha.v@example.com', '9876543210', 'Mumbai');

```
mysql> SELECT * FROM Customers;
+-------------+-------------+--------------------+--------------+--------+---------------------+
| customer_id | name        | email              | phone_number | city   | registration_date   |
+-------------+-------------+--------------------+--------------+--------+---------------------+
|           1 | Amit Kumar  | amit.k@example.com | 9123456789   | Delhi  | 2024-10-16 22:16:11 |
|           2 | Sneha Verma | sneha.v@example.com | 9876543210   | Mumbai | 2024-10-16 22:16:11 |
+-------------+-------------+--------------------+--------------+--------+---------------------+
2 rows in set (0.00 sec)
```

INSERT INTO Orders (customer_id, order_date, total_amount, payment_status) VALUES
(1, '2024-09-01', 1500.50, 'Paid'),
(2, '2024-09-02', 2500.75, 'Pending');

```
mysql> SELECT * FROM Orders;
+----------+-------------+------------+--------------+----------------+
| order_id | customer_id | order_date | total_amount | payment_status |
+----------+-------------+------------+--------------+----------------+
|        1 |           1 | 2024-09-01 |      1500.50 | Paid           |
|        2 |           2 | 2024-09-02 |      2500.75 | Pending        |
+----------+-------------+------------+--------------+----------------+
2 rows in set (0.00 sec)
```

INSERT INTO Products (name, category, price, stock_quantity) VALUES
('Mobile Phone', 'Electronics', 12000.00, 50),
('Laptop', 'Electronics', 55000.00, 30);

```
mysql> SELECT * FROM Products;
+------------+--------------+-------------+----------+----------------+
| product_id | name         | category    | price    | stock_quantity |
+------------+--------------+-------------+----------+----------------+
|          1 | Mobile Phone | Electronics | 12000.00 |             50 |
|          2 | Laptop       | Electronics | 55000.00 |             30 |
+------------+--------------+-------------+----------+----------------+
2 rows in set (0.00 sec)
```

2. Renaming and Updating Tables

Renaming Orders table to CustomerOrders:

ALTER TABLE Orders RENAME TO CustomerOrders;

```
mysql> SHOW TABLES;
+------------------+
| Tables_in_shopdb |
+------------------+
| customerorders   |
| customers        |
| products         |
+------------------+
3 rows in set (0.00 sec)
```

Updating total_amount for a specific order:

UPDATE CustomerOrders SET total_amount = total_amount + 500 WHERE order_id = 1;

```
mysql> SELECT * FROM CustomerOrders;
+----------+-------------+------------+--------------+----------------+
| order_id | customer_id | order_date | total_amount | payment_status |
+----------+-------------+------------+--------------+----------------+
|        1 |           1 | 2024-09-01 |      2000.50 | Paid           |
|        2 |           2 | 2024-09-02 |      2500.75 | Pending        |
+----------+-------------+------------+--------------+----------------+
2 rows in set (0.00 sec)
```

3. Constraints Implementation

Adding constraints like NOT NULL, AUTO_INCREMENT, PRIMARY KEY, and FOREIGN KEY:

```
CREATE TABLE Transactions (
    transaction_id INT AUTO_INCREMENT PRIMARY KEY,
    order_id INT NOT NULL,
    payment_method VARCHAR(50) NOT NULL,
    payment_date DATE,
    FOREIGN KEY (order_id) REFERENCES CustomerOrders(order_id)
);
```

```
mysql> DESCRIBE Transactions;
+----------------+-------------+------+-----+---------+----------------+
| Field          | Type        | Null | Key | Default | Extra          |
+----------------+-------------+------+-----+---------+----------------+
| transaction_id | int         | NO   | PRI | NULL    | auto_increment |
| order_id       | int         | NO   | MUL | NULL    |                |
| payment_method | varchar(50) | NO   |     | NULL    |                |
| payment_date   | date        | YES  |     | NULL    |                |
+----------------+-------------+------+-----+---------+----------------+
4 rows in set (0.00 sec)
```

4. Joins Implementation

INNER JOIN between CustomerOrders and Customers:

SELECT Customers.name, CustomerOrders.order_date, CustomerOrders.total_amount
FROM Customers
INNER JOIN CustomerOrders ON Customers.customer_id = CustomerOrders.customer_id;

```
mysql> SELECT Customers.name, CustomerOrders.order_date, CustomerOrders.total_amount
    -> FROM Customers
    -> INNER JOIN CustomerOrders ON Customers.customer_id = CustomerOrders.customer_id;
+-------------+------------+--------------+
| name        | order_date | total_amount |
+-------------+------------+--------------+
| Amit Kumar  | 2024-09-01 |      2000.50 |
| Sneha Verma | 2024-09-02 |      2500.75 |
+-------------+------------+--------------+
2 rows in set (0.00 sec)
```

5. Subqueries

A subquery that fetches orders placed by a specific customer:

SELECT * FROM CustomerOrders WHERE customer_id = (SELECT customer_id FROM Customers WHERE name = 'Amit Kumar');

```
mysql> SELECT * FROM CustomerOrders WHERE customer_id = (SELECT customer_id FROM Customers WHERE name = 'Amit Kumar');
+----------+-------------+------------+--------------+----------------+
| order_id | customer_id | order_date | total_amount | payment_status |
+----------+-------------+------------+--------------+----------------+
|        1 |           1 | 2024-09-01 |      2000.50 | Paid           |
+----------+-------------+------------+--------------+----------------+
1 row in set (0.00 sec)
```

6. Set Operations and Transactions

Start a transaction, insert a new order, and commit the transaction:

START TRANSACTION;

INSERT INTO CustomerOrders (customer_id, order_date, total_amount, payment_status)

VALUES (1, '2024-10-10', 3000.00, 'Paid');

COMMIT;

```
mysql> SELECT * FROM CustomerOrders;
+----------+-------------+------------+--------------+----------------+
| order_id | customer_id | order_date | total_amount | payment_status |
+----------+-------------+------------+--------------+----------------+
|        1 |           1 | 2024-09-01 |      2000.50 | Paid           |
|        2 |           2 | 2024-09-02 |      2500.75 | Pending        |
|        3 |           1 | 2024-10-10 |      3000.00 | Paid           |
+----------+-------------+------------+--------------+----------------+
3 rows in set (0.00 sec)
```

7. Views

View for all completed transactions:

CREATE VIEW CompletedTransactions AS
SELECT order_id, total_amount, payment_status FROM CustomerOrders WHERE payment_status = 'Paid';

```
mysql> SELECT * FROM CompletedTransactions;
+----------+--------------+----------------+
| order_id | total_amount | payment_status |
+----------+--------------+----------------+
|        1 |      2000.50 | Paid           |
|        3 |      3000.00 | Paid           |
+----------+--------------+----------------+
2 rows in set (0.00 sec)
```

8. Set Operators

Using UNION to combine query results:

SELECT product_id, name FROM Products

UNION

SELECT product_id, name FROM CustomerOrders;

```
mysql> SELECT product_id, name FROM Products
    -> UNION
    -> SELECT order_id AS product_id, 'Order' AS name FROM CustomerOrders;
+------------+--------------+
| product_id | name         |
+------------+--------------+
|          1 | Mobile Phone |
|          2 | Laptop       |
|          1 | Order        |
|          3 | Order        |
|          2 | Order        |
+------------+--------------+
5 rows in set (0.00 sec)
```

Key Concepts and SQL Commands Explained:

Key Concepts and SQL Commands Explained

1. Data Definition Language (DDL)

DDL commands are used to define and modify database structures such as tables.

CREATE: Used to create a new table.
Example:

```
CREATE TABLE Customer (

    CustomerID INT PRIMARY KEY,

    Name VARCHAR(100),

    Email VARCHAR(100)

);
```

ALTER: Used to modify an existing table.
Example:

```
ALTER TABLE Customer ADD PhoneNumber VARCHAR(15);
```

DROP: Deletes a table and all its data.
Example:

```
DROP TABLE Transaction;
```

2. Data Manipulation Language (DML)

DML commands handle data manipulation within the database, like inserting, updating, and deleting records.

INSERT: Adds new data into the table.
Example:

```
INSERT INTO Customer (CustomerID, Name, Email)

VALUES (1, 'John Doe', 'john@example.com');
```

UPDATE: Modifies existing data.
Example:

```
UPDATE Product

SET Price = 500

WHERE ProductID = 101;
```

DELETE: Removes data from the table.
Example:

```
DELETE FROM Order WHERE OrderID = 202;
```

3. Data Control Language (DCL)

DCL commands manage permissions and access controls in the database.

GRANT: Gives privileges to users.
Example:

```
GRANT SELECT ON Product TO user123;
```

REVOKE: Takes back privileges from users.
Example:

```
REVOKE INSERT ON Customer FROM user123;
```

4. Transaction Control Language (TCL)

TCL commands manage database transactions to ensure data integrity.

COMMIT: Saves the changes made during the transaction.
Example:

```
COMMIT;
```

ROLLBACK: Reverts changes made in a transaction if something goes wrong.
Example:

```
ROLLBACK;
```

SAVEPOINT: Sets a savepoint within a transaction.
Example:

```
SAVEPOINT sp1;
```

5. Types of Joins

INNER JOIN: Retrieves records that have matching values in both tables.

Example:

```
mysql> SELECT Customers.Name AS CustomerName, Orders.OrderID

FROM Customers

INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

```
mysql> SELECT Customers.Name AS CustomerName, Orders.OrderID
    -> FROM Customers
    -> INNER JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
+--------------+---------+
| CustomerName | OrderID |
+--------------+---------+
| John Doe     |     201 |
| John Doe     |     203 |
| Jane Smith   |     202 |
+--------------+---------+
3 rows in set (0.00 sec)
```

LEFT JOIN: Retrieves all records from the left table and matched records from the right table.

Example:

```
SELECT Customers.Name AS CustomerName, Orders.OrderID

FROM Customers

LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
```

```
mysql> SELECT Customers.Name AS CustomerName, Orders.OrderID
    -> FROM Customers
    -> LEFT JOIN Orders ON Customers.CustomerID = Orders.CustomerID;
+---------------+---------+
| CustomerName  | OrderID |
+---------------+---------+
| John Doe      |     201 |
| John Doe      |     203 |
| Jane Smith    |     202 |
| Emily Johnson |    NULL |
+---------------+---------+
4 rows in set (0.00 sec)
```

RIGHT JOIN: Retrieves all records from the right table and matched records from the left table.

Example:

```
SELECT Products.Name AS ProductName, Orders.OrderID

FROM Products

RIGHT JOIN Orders ON Products.ProductID = Orders.OrderID;
```

```
mysql> SELECT Products.Name AS ProductName, Orders.OrderID
    -> FROM Products
    -> RIGHT JOIN Orders ON Products.ProductID = Orders.OrderID;
+-------------+---------+
| ProductName | OrderID |
+-------------+---------+
| NULL        |     201 |
| NULL        |     203 |
| NULL        |     202 |
+-------------+---------+
3 rows in set (0.00 sec)
```

6. Aggregate Functions

COUNT: Returns the number of rows that match the criteria.

Example:

```
SELECT COUNT(*) AS TotalOrders FROM Orders;
```

```
mysql> SELECT COUNT(*) AS TotalOrders FROM Orders;
+-------------+
| TotalOrders |
+-------------+
|           3 |
+-------------+
1 row in set (0.00 sec)
```

SUM: Returns the total sum of a numeric column.
Example:

```
SELECT SUM(Price) AS TotalSales FROM Products;
```

```
mysql> SELECT SUM(Price) AS TotalSales FROM Products;
+------------+
| TotalSales |
+------------+
|    1600.00 |
+------------+
1 row in set (0.00 sec)
```

AVG: Returns the average value of a numeric column.
Example:

```
SELECT AVG(Price) AS AverageProductPrice FROM Products;
```

```
mysql> SELECT AVG(Price) AS AverageProductPrice FROM Products;
+--------------------+
| AverageProductPrice |
+--------------------+
|          533.333333 |
+--------------------+
1 row in set (0.00 sec)
```

7. Constraints

Primary Key: Ensures each record in a table is unique.
Example:

```
CREATE TABLE Product (

    ProductID INT PRIMARY KEY,

    Name VARCHAR(100),

    Price DECIMAL(10, 2)

);
```

Foreign Key: Ensures that a column's values match those of another table's primary key.
Example:

```
CREATE TABLE Order (

    OrderID INT PRIMARY KEY,

    CustomerID INT,
```

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

NOT NULL: Ensures that a column cannot have NULL values.
Example:

```
CREATE TABLE Transaction (
    TransactionID INT PRIMARY KEY,
    PaymentMode VARCHAR(50) NOT NULL,
    Status VARCHAR(50)
);
```

CHECK: Ensures that all values in a column meet a specific condition.
Example:

```
ALTER TABLE Product ADD CHECK (Price > 0);
```

UNIQUE: Ensures that all values in a column are unique.
Example:

```
CREATE TABLE Customer (
    CustomerID INT PRIMARY KEY,
    Email VARCHAR(100) UNIQUE
);
```

8. WHERE Clause

Used to filter records that meet a specific condition.

Example:

```
SELECT * FROM Product WHERE Price > 400;
```

```
mysql> SELECT * FROM Products WHERE Price > 400;
+-----------+------------+--------+---------------+
| ProductID | Name       | Price  | StockQuantity |
+-----------+------------+--------+---------------+
|       101 | Laptop     | 800.00 |            50 |
|       102 | Smartphone | 500.00 |           150 |
+-----------+------------+--------+---------------+
2 rows in set (0.00 sec)
```

9. GROUP BY and HAVING Clauses

GROUP BY: Groups rows that have the same values into summary rows.
Example:

```
SELECT CustomerID, COUNT(OrderID) AS OrderCount

FROM Orders

GROUP BY CustomerID;
```

```
mysql> SELECT CustomerID, COUNT(OrderID) AS OrderCount
    -> FROM Orders
    -> GROUP BY CustomerID;
+------------+------------+
| CustomerID | OrderCount |
+------------+------------+
|          1 |          2 |
|          2 |          1 |
+------------+------------+
2 rows in set (0.00 sec)
```

HAVING: Filters groups according to a condition.
Example:

```
SELECT CustomerID, COUNT(OrderID) AS OrderCount

FROM Orders

GROUP BY CustomerID

HAVING COUNT(OrderID) > 1;
```

```
mysql> SELECT CustomerID, COUNT(OrderID) AS OrderCount
    -> FROM Orders
    -> GROUP BY CustomerID
    -> HAVING COUNT(OrderID) > 1;
+------------+------------+
| CustomerID | OrderCount |
+------------+------------+
|          1 |          2 |
+------------+------------+
1 row in set (0.00 sec)
```

10. Pattern Matching Using LIKE

The `LIKE` operator is used to search for a specified pattern in a column.

Example:

```
SELECT * FROM Customer WHERE Name LIKE 'J%';
```

```
mysql> SELECT * FROM Customers WHERE Name LIKE 'J%';
+------------+------------+------------------+-------------+-------------+---------------------+
| CustomerID | Name       | Email            | PhoneNumber | City        | RegistrationDate    |
+------------+------------+------------------+-------------+-------------+---------------------+
|          1 | John Doe   | john@example.com | 9876543210  | New York    | 2024-10-16 22:24:50 |
|          2 | Jane Smith | jane@example.com | 9123456789  | Los Angeles | 2024-10-16 22:24:50 |
+------------+------------+------------------+-------------+-------------+---------------------+
2 rows in set (0.00 sec)
```

DESCRIBE TABLES TO SHOW STRUCTURES:

```
mysql> DESCRIBE Customers;
+------------------+--------------+------+-----+-------------------+-------------------+
| Field            | Type         | Null | Key | Default           | Extra             |
+------------------+--------------+------+-----+-------------------+-------------------+
| CustomerID       | int          | NO   | PRI | NULL              |                   |
| Name             | varchar(100) | YES  |     | NULL              |                   |
| Email            | varchar(100) | YES  | UNI | NULL              |                   |
| PhoneNumber      | varchar(15)  | NO   |     | NULL              |                   |
| City             | varchar(50)  | YES  |     | NULL              |                   |
| RegistrationDate | timestamp    | YES  |     | CURRENT_TIMESTAMP | DEFAULT_GENERATED |
+------------------+--------------+------+-----+-------------------+-------------------+
6 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Products;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| ProductID     | int           | NO   | PRI | NULL    |       |
| Name          | varchar(100)  | YES  |     | NULL    |       |
| Price         | decimal(10,2) | YES  |     | NULL    |       |
| StockQuantity | int           | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Orders;
+---------------+---------------+------+-----+---------+-------+
| Field         | Type          | Null | Key | Default | Extra |
+---------------+---------------+------+-----+---------+-------+
| OrderID       | int           | NO   | PRI | NULL    |       |
| CustomerID    | int           | YES  | MUL | NULL    |       |
| OrderDate     | date          | YES  |     | NULL    |       |
| TotalAmount   | decimal(10,2) | YES  |     | NULL    |       |
| PaymentStatus | varchar(50)   | YES  |     | NULL    |       |
+---------------+---------------+------+-----+---------+-------+
5 rows in set (0.00 sec)
```

```
mysql> DESCRIBE Transactions;
+---------------+-------------+------+-----+---------+-------+
| Field         | Type        | Null | Key | Default | Extra |
+---------------+-------------+------+-----+---------+-------+
| TransactionID | int         | NO   | PRI | NULL    |       |
| OrderID       | int         | YES  | MUL | NULL    |       |
| PaymentMode   | varchar(50) | NO   |     | NULL    |       |
| Status        | varchar(50) | YES  |     | NULL    |       |
+---------------+-------------+------+-----+---------+-------+
4 rows in set (0.00 sec)
```

SELECTING DATA FROM TABLES TO SHOW CONTENTS:

```
mysql> SELECT * FROM Customers;
+------------+--------------+-------------------+-------------+-------------+---------------------+
| CustomerID | Name         | Email             | PhoneNumber | City        | RegistrationDate    |
+------------+--------------+-------------------+-------------+-------------+---------------------+
|          1 | John Doe     | john@example.com  | 9876543210  | New York    | 2024-10-16 22:24:50 |
|          2 | Jane Smith   | jane@example.com  | 9123456789  | Los Angeles | 2024-10-16 22:24:50 |
|          3 | Emily Johnson | emily@example.com | 9345678901  | Chicago     | 2024-10-16 22:24:50 |
+------------+--------------+-------------------+-------------+-------------+---------------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Products;
+-----------+------------+--------+---------------+
| ProductID | Name       | Price  | StockQuantity |
+-----------+------------+--------+---------------+
|       101 | Laptop     | 800.00 |            50 |
|       102 | Smartphone | 500.00 |           150 |
|       103 | Tablet     | 300.00 |           100 |
+-----------+------------+--------+---------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Orders;
+---------+------------+------------+-------------+---------------+
| OrderID | CustomerID | OrderDate  | TotalAmount | PaymentStatus |
+---------+------------+------------+-------------+---------------+
|     201 |          1 | 2024-09-15 |     1300.00 | Paid          |
|     202 |          2 | 2024-09-16 |      800.00 | Pending       |
|     203 |          1 | 2024-09-17 |     1500.00 | Paid          |
+---------+------------+------------+-------------+---------------+
3 rows in set (0.00 sec)
```

```
mysql> SELECT * FROM Transactions;
+---------------+---------+-------------+-----------+
| TransactionID | OrderID | PaymentMode | Status    |
+---------------+---------+-------------+-----------+
|           301 |     201 | Credit Card | Completed |
|           302 |     202 | PayPal      | Pending   |
+---------------+---------+-------------+-----------+
2 rows in set (0.00 sec)
```

Conclusion

This project provides a comprehensive overview of the Indian e-commerce sector and demonstrates how various SQL concepts can be effectively applied within this domain. The growth of e-commerce in India, fueled by advancements in technology and government initiatives, presents numerous opportunities and challenges for businesses operating in this space.

Key Highlights:

1. **Database Design**: The schema created for the e-commerce database incorporates essential entities such as Customers, Orders, Products, and Transactions. This structure reflects the relationships among these entities, facilitating efficient data management and retrieval.
2. **SQL Operations**: The project covers a wide range of SQL operations:
    - **DDL (Data Definition Language)** commands are used to create and modify database structures.
    - **DML (Data Manipulation Language)** commands are employed for inserting, updating, and deleting records, showcasing the dynamic nature of the database.
    - **DCL (Data Control Language)** commands manage user permissions, ensuring data security.
    - **TCL (Transaction Control Language)** commands maintain data integrity during transactions, illustrating the importance of commit and rollback functions.
3. **Data Relationships**: The established relationships among entities (e.g., one-to-many between Customers and Orders, many-to-many between Orders and Products) are crucial for reflecting real-world interactions in the database.
4. **Advanced SQL Concepts**: The project delves into advanced SQL concepts such as joins, subqueries, and set operations, enhancing the database's querying capabilities. These concepts are vital for generating meaningful insights and analytics from the data.
5. **Implementation of Constraints**: The project highlights the importance of constraints like primary keys, foreign keys, and checks to ensure data integrity and enforce business rules.
6. **Consumer Insights**: The analysis of consumer behavior and trends in the Indian e-commerce market emphasizes the importance of understanding customer needs and preferences in driving business strategies.

In summary, this project serves as an effective demonstration of how SQL can be leveraged to build a robust database system tailored to the Indian e-commerce landscape. The insights gained from this project not only reinforce the significance of a well-structured database but also highlight the continuous evolution and potential of the e-commerce sector in India. As technology advances, the integration of emerging technologies such as AI and machine learning will further revolutionize the way e-commerce businesses operate, creating a promising future for this industry.

https://github.com/chopra-siddhant/DBMS-PROJECT.git