E-commerce order management system

Database schema

Tables creation

1)Categories

```
create table categories (
    category_id int primary key,
    category_name varchar(100) not null
);
```

2)Products

```
create table Products (
    product_id int primary key,
    name  varchar(100),
    category_id int,
    price decimal(10,2),
    stock_quantity int,
    added_date date,
    FOREIGN KEY (category_id) REFERENCES Categories(category_id)
);
```

3)Customers

```
CREATE TABLE Customers (
    customer_id INT PRIMARY KEY,
    name VARCHAR(100),
    email VARCHAR(100),
    phone_number VARCHAR(15),
    address VARCHAR(200),
    registration_date DATE
);
```

4)Orders

```
CREATE TABLE Orders (
    order_id INT PRIMARY KEY,
    customer_id INT,
    order_date DATE,
    total_amount DECIMAL(10,2),
    status VARCHAR(20),
    FOREIGN KEY (customer_id) REFERENCES Customers(customer_id)
);
```

5)Order_item

```
CREATE TABLE Order_Items (
    order_item_id INT PRIMARY KEY,
    order_id INT,
    product_id INT,
    quantity INT,
    subtotal DECIMAL(10,2),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id),
    FOREIGN KEY (product_id) REFERENCES Products(product_id)
);
```

6)Payments

```
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY,
    order_id INT,
    payment_date DATE,
    payment_method VARCHAR(20),
    payment_status VARCHAR(20),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

7)Shipping

```
CREATE TABLE Shipping (
    shipping_id INT PRIMARY KEY,
    order_id INT,
    shipping_date DATE,
    delivery_date DATE,
    shipping_status VARCHAR(20),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
);
```

Data insertion in table(4 entries in each table)

1)Categories

```
INSERT INTO Categories VALUES
(1,'Electronics'),
(2,'Clothing'),
(3,'Books'),
(4,'Home Appliances')
;
```

2)Products

```
INSERT INTO Products VALUES
(101,'Laptop',1,55000,10,'2024-01-10'),
(102,'T-Shirt',2,999,50,'2024-02-05'),
(103,'Java Programming Book',3,650,100,'2023-12-15'),
(104,'Washing Machine',4,28000,8,'2024-03-01')
;
```

3) Customers

```
INSERT INTO Customers VALUES
(1,'Rahul Sharma','rahul@gmail.com','9876543210','Mumbai','2023-05-10'),
(2,'Anjali Verma',NULL,'9123456789','Delhi','2024-01-15'),
(3,'Amit Patel','amit@yahoo.com','9988776655','Ahmedabad','2022-11-20'),
(4,'Sneha Iyer','sneha@gmail.com','9090909090','Pune','2023-07-25')
;
```

4)Orders

```
INSERT INTO Orders VALUES
```

```
(1001,1,'2024-02-10',55999,'Delivered'),
(1002,2,'2024-03-05',999,'Pending'),
(1003,3,'2024-01-25',28650,'Shipped'),
(1004,1,'2023-12-12',3500,'Cancelled')
;
```

5)Order_item

```
INSERT INTO Order_Items VALUES
(1,1001,101,1,55000),
(2,1002,102,1,999),
(3,1003,104,1,28000),
(4,1003,103,1,650)
;
```

6)Payments

```
INSERT INTO Payments VALUES
(201,1001,'2024-02-10','Credit Card','Paid'),
(202,1002,'2024-03-05','UPI','Pending'),
(203,1003,'2024-01-25','PayPal','Paid'),
(204,1004,'2023-12-12','UPI','Failed')
;
```

7)Shipping

```
INSERT INTO Shipping VALUES
(301,1001,'2024-02-11','2024-02-15','Delivered'),
(302,1002,'2024-03-06',NULL,'Dispatched'),
(303,1003,'2024-01-26',NULL,'In Transit'),
(304,1004,NULL,NULL,'Cancelled')
;
```

Tasks and functionalities

Answers

1)implement crud operations

->insert

```
INSERT INTO Categories VALUES (1,'Electronics'),(2,'Clothing');

INSERT INTO Products VALUES
(101,'Laptop',1,55000,10,'2024-01-10'),
(102,'T-Shirt',2,999,50,'2024-02-05');

INSERT INTO Customerss VALUES
(1,'Rahul Sharma','rahul@gmail.com','9876543210','Mumbai','2023-05-10');
```

->update

```
UPDATE Products
SET stock_quantity = stock_quantity - 1
WHERE product_id = 101;
```

->delete

```
DELETE FROM Orders
```

```
WHERE status = 'Cancelled'
AND order_date < DATE_SUB(CURDATE(), INTERVAL 30 DAY);
```

2)use sql clauses

->

```
SELECT * FROM Orders
WHERE order_date >= DATE_SUB(CURDATE(), INTERVAL 6 MONTH);
```

->

```
SELECT * FROM Products
ORDER BY price DESC
LIMIT 5;
```

->

```
SELECT customer_id, COUNT(*) AS total_orders
FROM Orders
GROUP BY customer_id
HAVING COUNT(*) > 3;
```

3)apply sql operators

->

```
SELECT o.*
FROM Orders o
JOIN Payments p ON o.order_id = p.order_id
WHERE o.status = 'Pending'
AND p.payment_status = 'Paid';
```

->

```
SELECT * FROM Products
WHERE NOT stock_quantity = 0;
```

->

```
SELECT * FROM Customers
WHERE registration_date > '2022-12-31'
OR customer_id IN (
   SELECT customer_id
   FROM Orders
   GROUP BY customer_id
   HAVING SUM(total_amount) > 10000
);
```

4)sorting and grouping data

->

```
SELECT * FROM Products ORDER BY price DESC;
```

->

```
SELECT customer_id, COUNT(order_id) AS total_orders
FROM Orders
GROUP BY customer_id;
```

->

```
SELECT c.category_name, SUM(oi.subtotal) AS revenue
FROM Order_Items oi
JOIN Products p ON oi.product_id = p.product_id
JOIN Categories c ON p.category_id = c.category_id
GROUP BY c.category_name;
```

5)use aggregate functions

->

```
SELECT SUM(total_amount) AS total_revenue FROM Orders;
```

->

```
SELECT product_id, SUM(quantity) AS total_sold
FROM Order_Items
GROUP BY product_id
ORDER BY total_sold DESC
LIMIT 1;
```

->

```
SELECT AVG(total_amount) AS avg_order_value FROM Orders;
```

6)establish primary and foreign key relationships

-> **Linking Orders to Customers**

The **Orders** table uses order_id as the **Primary Key**.
The field customer_id in the **Orders** table is a **Foreign Key** that references customer_id in the **Customers** table.
This relationship ensures that **every order is placed by a valid customer**.

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

- **Linking Orders to Products**
  The **Order_Items** table acts as a bridge between **Orders** and **Products**.
  order_id is a **Foreign Key** referencing Orders(order_id).
  product_id is a **Foreign Key** referencing Products(product_id).
  This ensures that **each order contains valid products with defined quantities**.

```
FOREIGN KEY (order_id) REFERENCES Orders(order_id),
        FOREIGN KEY (product_id) REFERENCES Products(product_id)
```

->

Establishing Relationship Between Payments and Orders

- The **Payments** table uses payment_id as the **Primary Key**.
- The order_id field in the **Payments** table is a **Foreign Key** referencing Orders(order_id).
- This ensures that **every payment is associated with a valid order**.

```
    FOREIGN KEY (order_id) REFERENCES Orders(order_id)
```

7)implement joints

->inner join

```
SELECT p.name, c.category_name
FROM Products p
INNER JOIN Categories c ON p.category_id = c.category_id;
```

->left join

```
SELECT o.order_id, c.name
FROM Orders o
LEFT JOIN Customers c ON o.customer_id = c.customer_id;
```

->right join

```
SELECT o.order_id, s.shipping_status
FROM Orders o
RIGHT JOIN Shipping s ON o.order_id = s.order_id
WHERE s.shipping_status IS NULL;
```

->full outer join

```
SELECT c.customer_id
FROM Customers c
LEFT JOIN Orders o ON c.customer_id = o.customer_id
WHERE o.order_id IS NULL;
```

8)use subqueries

->

```
SELECT * FROM Orders
WHERE customer_id IN (
   SELECT customer_id FROM Customers
   WHERE registration_date > '2022-12-31'
);
```

->

```
SELECT customer_id
FROM Orders
GROUP BY customer_id
ORDER BY SUM(total_amount) DESC
LIMIT 1;
```

->

```
SELECT * FROM Products
WHERE product_id NOT IN (
   SELECT product_id FROM Order_Items
);
```

9)implement date and time functions

->

```
SELECT MONTH(order_date) AS month, COUNT(*) AS orders
FROM Orders
GROUP BY MONTH(order_date);
```

->

```
SELECT DATEDIFF(delivery_date, shipping_date) AS delivery_days
FROM Shipping;
```

->

```
SELECT DATE_FORMAT(order_date,'%d-%m-%Y') FROM Orders;
```

10)use string manipulation functions

->

```
SELECT UPPER(name) FROM Products;
```

->

```
SELECT TRIM(name) FROM Customers;
```

->

```
SELECT IFNULL(email,'Not Provided') FROM Customers;
```

11)implement windows functions

->

```
SELECT customer_id,
SUM(total_amount) AS spending,
RANK() OVER (ORDER BY SUM(total_amount) DESC) AS rank_no
FROM Orders
GROUP BY customer_id;
```

->

```
SELECT MONTH(order_date) AS month,
SUM(total_amount) AS monthly_revenue,
SUM(SUM(total_amount)) OVER (ORDER BY MONTH(order_date)) AS cumulative_revenue
FROM Orders
GROUP BY MONTH(order_date);
```

->

```
SELECT order_id,
SUM(total_amount) OVER (ORDER BY order_date) AS running_total
FROM Orders;
```

12)apply case expressions

->

```
SELECT customer_id,
CASE
   WHEN SUM(total_amount) > 50000 THEN 'Gold'
```

```
    WHEN SUM(total_amount) BETWEEN 20000 AND 50000 THEN 'Silver'
    ELSE 'Bronze'
END AS Loyalty_Status
FROM Orders
GROUP BY customer_id;
```

->

```
SELECT product_id,
CASE
    WHEN SUM(quantity) > 500 THEN 'Best Seller'
    WHEN SUM(quantity) BETWEEN 200 AND 500 THEN 'Popular'
    ELSE 'Regular'
END AS Product_Status
FROM Order_Items
GROUP BY product_id;
```