**Design the following ecommerce database**

**Table 1: Products**

* **product\_id** (Primary Key)
* **product\_name**
* **price**
* **description**

**Table 2: Customers**

* **customer\_id** (Primary Key)
* **first\_name**
* **last\_name**
* **email**
* **phone**
* **address**

**Table 3: Orders**

* **order\_id** (Primary Key)
* **customer\_id** (Foreign Key referencing Customers)
* **order\_date**
* **total\_amount**

**Table 4: Order Items**

* **item\_id** (Primary Key)
* **order\_id** (Foreign Key referencing Orders)
* **product\_id** (Foreign Key referencing Products)
* **quantity**
* **subtotal**

**Table 5: Payment Methods**

* **method\_id** (Primary Key)
* **customer\_id** (Foreign Key referencing Customers)
* **method\_type** (e.g., credit card, PayPal)
* **method\_details** (e.g., card number, PayPal email)

Now, let's add constraints to these tables:

**Products Table Constraints:**

* **product\_id** is a primary key, ensuring uniqueness.
* **price** must be a non-negative value.
* **product\_name** and **description** should not be empty.

**Customers Table Constraints:**

* **customer\_id** is a primary key, ensuring uniqueness.
* **email** should be unique and in a valid email format.
* **phone** could have a format constraint (e.g., a certain number of digits).

**Orders Table Constraints:**

* **order\_id** is a primary key, ensuring uniqueness.
* **customer\_id** references a valid customer in the Customers table.
* **order\_date** should not be in the future.
* **total\_amount** must be a non-negative value.

**Order Items Table Constraints:**

* **item\_id** is a primary key, ensuring uniqueness.
* **order\_id** references a valid order in the Orders table.
* **product\_id** references a valid product in the Products table.
* **quantity** must be a positive integer.
* **subtotal** must be calculated as **quantity \* product price**.

**Payment Methods Table Constraints:**

* **method\_id** is a primary key, ensuring uniqueness.
* **customer\_id** references a valid customer in the Customers table.
* **method\_type** could be restricted to a predefined set of values.
* **method\_details** would depend on the **method\_type** and its associated requirements.

Answer:

-- Table: Products

CREATE TABLE Products (

product\_id INT PRIMARY KEY,

product\_name VARCHAR(255) NOT NULL,

price DECIMAL(10, 2) NOT NULL,

description TEXT

);

-- Table: Customers

CREATE TABLE Customers (

customer\_id INT PRIMARY KEY,

first\_name VARCHAR(50) NOT NULL,

last\_name VARCHAR(50) NOT NULL,

email VARCHAR(100) UNIQUE NOT NULL,

phone VARCHAR(20),

address TEXT

);

-- Table: Orders

CREATE TABLE Orders (

order\_id INT PRIMARY KEY,

customer\_id INT REFERENCES Customers(customer\_id),

order\_date DATE NOT NULL,

total\_amount DECIMAL(10, 2) NOT NULL

);

-- Table: Order Items

CREATE TABLE OrderItems (

item\_id INT PRIMARY KEY,

order\_id INT REFERENCES Orders(order\_id),

product\_id INT REFERENCES Products(product\_id),

quantity INT NOT NULL,

subtotal DECIMAL(10, 2) NOT NULL

);

-- Table: Payment Methods

CREATE TABLE PaymentMethods (

method\_id INT PRIMARY KEY,

customer\_id INT REFERENCES Customers(customer\_id),

method\_type VARCHAR(50) NOT NULL,

method\_details TEXT

);

Insertion of the data

-- Insert data into Products

INSERT INTO Products (product\_id, product\_name, price, description)

VALUES

(1, 'Laptop', 999.99, 'High-performance laptop'),

(2, 'Smartphone', 599.99, 'Latest model smartphone'),

(3, 'Tablet', 299.99, 'Portable tablet device'),

(4, 'Desktop PC', 799.99, 'Powerful desktop computer'),

(5, 'Wireless Headphones', 149.99, 'Noise-cancelling wireless headphones'),

(6, 'Smart Watch', 199.99, 'Fitness and smart features'),

(7, 'Camera', 499.99, 'High-resolution digital camera'),

(8, 'Gaming Console', 399.99, 'Next-gen gaming console'),

(9, 'Printer', 129.99, 'Wireless color printer'),

(10, 'External Hard Drive', 89.99, '1TB portable external hard drive'),

(11, 'Smart TV', 899.99, 'Ultra HD smart television'),

(12, 'Bluetooth Speaker', 69.99, 'Portable Bluetooth speaker'),

(13, 'Wireless Mouse', 29.99, 'Ergonomic wireless mouse'),

(14, 'Keyboard', 49.99, 'Mechanical gaming keyboard'),

(15, 'Fitness Tracker', 79.99, 'Activity and health tracker'),

(16, 'VR Headset', 299.99, 'Virtual reality headset'),

(17, 'Monitor', 249.99, '24-inch LED monitor'),

(18, 'Projector', 599.99, 'HD home theater projector'),

(19, 'Coffee Maker', 79.99, 'Programmable coffee maker'),

(20, 'Microwave Oven', 129.99, 'Compact microwave oven'),

(21, 'Blender', 39.99, 'High-speed blender'),

(22, 'Toaster', 24.99, '2-slice toaster'),

(23, 'Air Fryer', 89.99, 'Healthy cooking air fryer'),

(24, 'Robot Vacuum', 249.99, 'Smart robotic vacuum cleaner'),

(25, 'Hair Dryer', 49.99, 'Ionic hair dryer');

-- Insert data into Customers

INSERT INTO Customers (customer\_id, first\_name, last\_name, email, phone, address)

VALUES

(1, 'John', 'Doe', 'john@example.com', '123-456-7890', '123 Main St'),

(2, 'Jane', 'Smith', 'jane@example.com', '987-654-3210', '456 Elm St'),

(3, 'Michael', 'Johnson', 'michael@example.com', '555-123-4567', '789 Oak Ave'),

(4, 'Emily', 'Williams', 'emily@example.com', '777-888-9999', '567 Pine Rd'),

(5, 'David', 'Brown', 'david@example.com', '222-333-4444', '234 Maple Ln'),

(6, 'Sarah', 'Jones', 'sarah@example.com', '111-222-3333', '890 Cedar St'),

(7, 'Christopher', 'Wilson', 'chris@example.com', '444-555-6666', '456 Birch Ave'),

(8, 'Jessica', 'Martinez', 'jessica@example.com', '666-777-8888', '678 Elm St'),

(9, 'Matthew', 'Anderson', 'matthew@example.com', '888-999-0000', '345 Oak Rd'),

(10, 'Amanda', 'Taylor', 'amanda@example.com', '333-444-5555', '123 Pine Ave'),

(11, 'Daniel', 'Harris', 'daniel@example.com', '555-666-7777', '789 Maple St'),

(12, 'Laura', 'Lee', 'laura@example.com', '777-888-9999', '456 Cedar Rd'),

(13, 'Ryan', 'Martin', 'ryan@example.com', '222-333-4444', '234 Birch Ave'),

(14, 'Melissa', 'Walker', 'melissa@example.com', '111-222-3333', '890 Elm St'),

(15, 'Kevin', 'Thompson', 'kevin@example.com', '444-555-6666', '456 Oak Ave'),

(16, 'Ashley', 'Garcia', 'ashley@example.com', '666-777-8888', '678 Pine St'),

(17, 'Jacob', 'Miller', 'jacob@example.com', '888-999-0000', '345 Cedar Rd'),

(18, 'Stephanie', 'Davis', 'stephanie@example.com', '333-444-5555', '123 Birch Ave'),

(19, 'Joseph', 'Rodriguez', 'joseph@example.com', '555-666-7777', '789 Elm St'),

(20, 'Megan', 'Lopez', 'megan@example.com', '777-888-9999', '456 Oak Rd'),

(21, 'William', 'Perez', 'william@example.com', '222-333-4444', '234 Pine Ave'),

(22, 'Nicole', 'Hernandez', 'nicole@example.com', '111-222-3333', '890 Cedar St'),

(23, 'Charles', 'Gonzalez', 'charles@example.com', '444-555-6666', '456 Birch Ave'),

(24, 'Elizabeth', 'Moore', 'elizabeth@example.com', '666-777-8888', '678 Elm St'),

(25, 'Andrew', 'Jackson', 'andrew@example.com', '888-999-0000', '345 Oak Rd');

-- Insert data into Orders

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

VALUES

(1, 3, '2023-08-01', 999.99),

(2, 5, '2023-08-02', 1599.98),

(3, 7, '2023-08-03', 299.99),

(4, 2, '2023-08-04', 799.99),

(5, 9, '2023-08-05', 149.99),

(6, 4, '2023-08-06', 199.99),

(7, 6, '2023-08-07', 499.99),

(8, 8, '2023-08-08', 399.99),

(9, 10, '2023-08-09', 129.99),

(10, 1, '2023-08-10', 89.99),

(11, 12, '2023-08-11', 899.99),

(12, 14, '2023-08-12', 69.99),

(13, 16, '2023-08-13', 29.99),

(14, 18, '2023-08-14', 49.99),

(15, 20, '2023-08-15', 79.99),

(16, 22, '2023-08-16', 299.99),

(17, 24, '2023-08-17', 249.99),

(18, 25, '2023-08-18', 599.99),

(19, 23, '2023-08-19', 79.99),

(20, 21, '2023-08-20', 129.99),

(21, 19, '2023-08-21', 39.99),

(22, 17, '2023-08-22', 24.99),

(23, 15, '2023-08-23', 89.99),

(24, 13, '2023-08-24', 249.99),

(25, 11, '2023-08-25', 49.99);

-- Insert data into OrderItems

INSERT INTO OrderItems (item\_id, order\_id, product\_id, quantity, subtotal)

VALUES

(1, 1, 3, 2, 599.98),

(2, 2, 5, 1, 149.99),

(3, 3, 8, 3, 1199.97),

(4, 4, 12, 1, 69.99),

(5, 5, 18, 2, 499.98),

(6, 6, 23, 1, 89.99),

(7, 7, 2, 1, 599.99),

(8, 8, 15, 2, 159.98),

(9, 9, 11, 1, 899.99),

(10, 10, 1, 1, 999.99),

(11, 11, 9, 1, 129.99),

(12, 12, 14, 2, 99.98),

(13, 13, 6, 1, 199.99),

(14, 14, 4, 1, 799.99),

(15, 15, 10, 3, 269.97),

(16, 16, 7, 1, 499.99),

(17, 17, 17, 2, 499.98),

(18, 18, 20, 1, 129.99),

(19, 19, 13, 1, 29.99),

(20, 20, 21, 1, 39.99),

(21, 21, 24, 2, 499.98),

(22, 22, 25, 1, 49.99),

(23, 23, 19, 3, 239.97),

(24, 24, 3, 1, 299.99),

(25, 25, 22, 1, 234.99);

-- Insert data into PaymentMethods

INSERT INTO PaymentMethods (method\_id, customer\_id, method\_type, method\_details)

VALUES

(1, 3, 'Credit Card', 'john.doe@example.com'),

(2, 5, 'PayPal', 'jane.smith@example.com'),

(3, 7, 'Credit Card', 'michael.johnson@example.com'),

(4, 2, 'PayPal', 'emily.williams@example.com'),

(5, 9, 'Credit Card', 'david.brown@example.com'),

(6, 4, 'Credit Card', 'sarah.jones@example.com'),

(7, 6, 'PayPal', 'christopher.wilson@example.com'),

(8, 8, 'Credit Card', 'jessica.martinez@example.com'),

(9, 10, 'PayPal', 'matthew.anderson@example.com'),

(10, 1, 'Credit Card', 'amanda.taylor@example.com'),

(11, 12, 'Credit Card', 'daniel.harris@example.com'),

(12, 14, 'PayPal', 'laura.lee@example.com'),

(13, 16, 'Credit Card', 'ryan.martin@example.com'),

(14, 18, 'Credit Card', 'melissa.walker@example.com'),

(15, 20, 'PayPal', 'kevin.thompson@example.com'),

(16, 22, 'Credit Card', 'ashley.garcia@example.com'),

(17, 24, 'Credit Card', 'jacob.miller@example.com'),

(18, 25, 'PayPal', 'stephanie.davis@example.com'),

(19, 23, 'Credit Card', 'joseph.rodriguez@example.com'),

(20, 21, 'Credit Card', 'megan.lopez@example.com'),

(21, 19, 'PayPal', 'william.perez@example.com'),

(22, 17, 'Credit Card', 'nicole.hernandez@example.com'),

(23, 15, 'Credit Card', 'charles.gonzalez@example.com'),

(24, 13, 'PayPal', 'elizabeth.moore@example.com'),

(25, 11, 'Credit Card', 'andrew.jackson@example.com');

**Exercise 1: Retrieve all columns from the Customers table.**

SELECT \* FROM Customers;

**Exercise 2: Retrieve the first name and last name of customers.**

SELECT

first\_name,

last\_name

FROM Customers;

**Exercise 3: Retrieve unique cities from the Customers table.**

SELECT DISTINCT city FROM Customers;

**Exercise 4: Retrieve customers who have the email domain 'example.com'.**

SELECT \* FROM Customers

WHERE email LIKE '%@example.com';

**Exercise 5: Retrieve customers who have a phone number starting with '555'.**

SELECT \* FROM Customers

WHERE phone LIKE '555%';

**Exercise 6: Retrieve customers who live in 'Oak St' or 'Elm St'.**

SELECT \* FROM Customers

WHERE address LIKE '%Oak St%' OR address LIKE '%Elm St%';

**Exercise 7: Retrieve customers who don't have a phone number listed.**

SELECT \* FROM Customers

WHERE phone IS NULL;

**Exercise 8: Retrieve orders with a total amount greater than $500.**

SELECT \* FROM Orders

WHERE total\_amount > 500;

**Exercise 9: Retrieve orders placed on or after '2023-08-10'.**

SELECT \* FROM Orders

WHERE order\_date >= '2023-08-10';

**Exercise 10: Retrieve orders with a total amount between $100 and $300.**

SELECT \* FROM Orders

WHERE total\_amount BETWEEN 100 AND 300;

**Exercise 11: Retrieve products with a price less than $50.**

SELECT \* FROM Products

WHERE price < 50;

**Exercise 12: Retrieve products with the word 'Smart' in their name.**

SELECT \* FROM Products

WHERE product\_name LIKE '%Smart%';

**Exercise 13: Retrieve products ordered by price in ascending order.**

SELECT \* FROM Products ORDER BY price ASC;

**Exercise 14: Retrieve the highest price from the Products table.**

SELECT

MAX(price) AS highest\_price

FROM Products;

**Exercise 15: Retrieve the total number of customers.**

SELECT COUNT(\*) AS total\_customers FROM Customers;

**Exercise 16: Retrieve the average total amount of orders.**

SELECT

AVG(total\_amount) AS average\_total\_amount

FROM Orders;

**Exercise 17: Retrieve the sum of quantities in OrderItems.**

SELECT

SUM(quantity) AS total\_quantity

FROM OrderItems;

**Exercise 18: Retrieve customers who have placed more than 2 orders.**

SELECT

c.first\_name,

c.last\_name

FROM Customers c

WHERE (

SELECT COUNT(\*) FROM Orders o

WHERE c.customer\_id = o.customer\_id) > 2;

**Exercise 19: Retrieve the most expensive product.**

SELECT \* FROM Products

WHERE price = (SELECT MAX(price) FROM Products);

**Exercise 20: Retrieve the customer names and their total order amounts.**

SELECT

c.first\_name,

c.last\_name,

(

SELECT

SUM(o.total\_amount)

FROM Orders o

WHERE c.customer\_id = o.customer\_id

) AS total\_order\_amount

FROM Customers c;

**Exercise 21: Retrieve products that have been ordered at least once.**

SELECT \* FROM Products

WHERE product\_id IN (SELECT DISTINCT product\_id FROM OrderItems);

**Exercise 22: Retrieve the number of orders placed by each customer.**

SELECT

c.first\_name,

c.last\_name,

(

SELECT COUNT(\*) FROM Orders o

WHERE c.customer\_id = o.customer\_id

) AS order\_count

FROM Customers c;

**Exercise 23: Retrieve the total revenue for each product.**

SELECT

p.product\_name,

(SELECT

SUM(oi.subtotal)

FROM OrderItems oi

WHERE p.product\_id = oi.product\_id) AS total\_revenue

FROM Products p;

**Exercise 24: Retrieve customers who have placed orders with a total amount over $1000.**

SELECT \* FROM Customers

WHERE customer\_id IN (SELECT customer\_id FROM Orders WHERE total\_amount > 1000);

**Exercise 25: Retrieve the product names and their quantities sold.**

SELECT

p.product\_name,

(SELECT

SUM(oi.quantity)

FROM OrderItems oi

WHERE p.product\_id = oi.product\_id) AS total\_quantity\_sold

FROM Products p;

**Exercise 26: Retrieve the total quantity of products in stock.**

SELECT

SUM(quantity\_in\_stock) AS total\_quantity\_in\_stock

FROM Products;

**Exercise 27: Retrieve the customer names who have placed orders in the month of May.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders WHERE EXTRACT(MONTH FROM order\_date) = 5);

**Exercise 28: Retrieve products with a price that is a whole number.**

SELECT \* FROM Products

WHERE price = ROUND(price);

**Exercise 29: Retrieve the customer names and the total amounts of their orders, excluding those who have not placed any orders.**

SELECT

first\_name,

last\_name,

(SELECT

SUM(total\_amount)

FROM Orders

WHERE customer\_id = Customers.customer\_id) AS total\_order\_amount

FROM Customers

WHERE customer\_id IN (SELECT customer\_id FROM Orders);

**Exercise 30: Retrieve the product names that have been ordered at least five times.**

SELECT

product\_name

FROM Products

WHERE product\_id IN (SELECT

product\_id

FROM OrderItems

GROUP BY product\_id HAVING COUNT(\*) >= 5);

**Exercise 31: Retrieve customers who have placed orders on the first day of the month.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE EXTRACT(DAY FROM order\_date) = 1);

**Exercise 32: Retrieve the product names and their corresponding prices, ordered by product ID in ascending order.**

SELECT product\_name, price FROM Products ORDER BY product\_id;

**Exercise 33: Retrieve the customer names who have placed orders with a total amount greater than the average total amount of all orders.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (

SELECT

customer\_id

FROM Orders

WHERE total\_amount > (SELECT AVG(total\_amount) FROM Orders));

**Exercise 34: Retrieve the product names and the total amounts of their sales, ordered by total amounts in descending order.**

SELECT

p.product\_name,

COALESCE(SUM(oi.subtotal), 0) AS total\_sales\_amount

FROM Products p LEFT JOIN OrderItems oi

ON p.product\_id = oi.product\_id

GROUP BY p.product\_id

ORDER BY total\_sales\_amount DESC;

**Exercise 35: Retrieve the customer names who have placed orders with a total amount greater than the highest total amount of all orders.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE total\_amount > (SELECT MAX(total\_amount) FROM Orders));

**Exercise 36: Retrieve the product names and their corresponding prices, rounded to two decimal places.**

SELECT

product\_name,

ROUND(price, 2) AS rounded\_price

FROM Products;

**Exercise 37: Retrieve customers who have placed orders on the last day of the month.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE EXTRACT(DAY FROM order\_date) = EXTRACT(LAST\_DAY FROM order\_date));

**Exercise 38: Retrieve the product names and the total quantities sold, ordered by total quantities in descending order.**

SELECT

p.product\_name,

COALESCE(SUM(oi.quantity), 0) AS total\_quantities\_sold

FROM Products p LEFT JOIN OrderItems oi

ON p.product\_id = oi.product\_id

GROUP BY p.product\_id

ORDER BY total\_quantities\_sold DESC;

**Exercise 39: Retrieve the customer names and the average total amount of their orders, ordered by average amount in descending order.**

SELECT

first\_name,

last\_name,

(SELECT

AVG(total\_amount)

FROM Orders

WHERE customer\_id = Customers.customer\_id) AS average\_order\_amount

FROM Customers ORDER BY average\_order\_amount DESC;

**Exercise 40: Retrieve customers who have placed orders in February and September.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE EXTRACT(MONTH FROM order\_date) IN (2, 9));

**Exercise 41: Retrieve the customer names who have placed more than 3 orders in August.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE EXTRACT(MONTH FROM order\_date) = 8 GROUP BY customer\_id HAVING COUNT(\*) > 3);

**Exercise 42: Retrieve products with a price in the range of $10 to $30, ordered by product name in ascending order.**

SELECT \* FROM Products

WHERE price >= 10 AND price <= 30

ORDER BY product\_name;

**Exercise 43: Retrieve the customer names and their corresponding addresses, ordered by city in descending order.**

SELECT

first\_name,

last\_name,

address

FROM Customers ORDER BY city DESC;

**Exercise 44: Retrieve customers who have placed orders with a total amount greater than $100 and sort them by their last names in ascending order.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE total\_amount > 100)

ORDER BY last\_name ASC;

**Exercise 45: Retrieve the product names and their corresponding prices, ordered by product name in descending order.**

SELECT

product\_name,

price

FROM Products

ORDER BY product\_name DESC;

**Exercise 46: Retrieve the customer names and the average total amount of their orders, excluding those with no orders, ordered by average amount in ascending order.**

SELECT

first\_name,

last\_name,

(SELECT

AVG(total\_amount)

FROM Orders

WHERE customer\_id = Customers.customer\_id) AS average\_order\_amount

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders)

ORDER BY average\_order\_amount;

**Exercise 47: Retrieve the product names that have a name shorter than 15 characters.**

SELECT product\_name FROM Products

WHERE LENGTH(product\_name) < 15;

**Exercise 48: Retrieve customers who have placed orders with a total amount greater than the average total amount of their orders.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT

customer\_id

FROM Orders

WHERE total\_amount > (SELECT

AVG(total\_amount)

FROM Orders WHERE customer\_id = Customers.customer\_id

)

);

**Exercise 49: Retrieve the product names and their corresponding prices, ordered by price in descending order.**

SELECT

product\_name,

price

FROM Products ORDER BY price DESC;

**Exercise 50: Retrieve customers who have placed orders on an even-numbered day of the month.**

SELECT

first\_name,

last\_name

FROM Customers

WHERE customer\_id IN (SELECT customer\_id FROM Orders

WHERE EXTRACT(DAY FROM order\_date) % 2 = 0);