Task 1: SQL Queries [2*7=14] **SET: A**

1. **Write a query to display the total number of orders placed by each user along with the user's first name, last name, and email. Sort the results by the number of orders in descending order.**

```
SELECT Users.first_name, Users.last_name, Users.email, COUNT(Orders.order_id) AS total_orders
FROM Users
JOIN Orders ON Users.user_id = Orders.user_id
GROUP BY Users.user_id
ORDER BY total_orders DESC;
```

2. **Write a query to find the top 5 products with the highest average rating. Display the product name, description, and average rating.**

```
SELECT Products.name, Products.description, AVG(Reviews.rating) AS average_rating
FROM Products
JOIN Reviews ON Products.product_id = Reviews.product_id
GROUP BY Products.product_id
ORDER BY average_rating DESC
LIMIT 5;
```

3. **Write a query to find the shipping carrier that has delivered the most orders.**

```
SELECT Shipping.carrier, COUNT(Shipping.order_id) AS delivered_orders
FROM Shipping
WHERE Shipping.actual_delivery_date IS NOT NULL
GROUP BY Shipping.carrier
ORDER BY delivered_orders DESC
LIMIT 1;
```

4. **Write a query that displays the address that has been used for most orders that have been shipped.**

```
SELECT Addresses.address_id, COUNT(Orders.order_id) AS shipped_orders
FROM Addresses
JOIN Orders ON Addresses.address_id = Orders.address_id
WHERE Orders.status = 'shipped'
GROUP BY Addresses.address_id
ORDER BY shipped_orders DESC
LIMIT 1;
```

5. **Using a nested query, find all users who have never placed an order. Display their first name, last name, and email.**

```
SELECT Users.first_name, Users.last_name, Users.email
FROM Users
WHERE Users.user_id NOT IN (SELECT Orders.user_id FROM Orders)
```

6. **Write a query to display the total revenue generated from each category. Show the category name and total revenue and order the results by total revenue in descending order.**

```
SELECT Categories.name, SUM(Order_Items.price * Order_Items.quantity) AS
total_revenue
FROM Categories
JOIN Products ON Categories.category_id = Products.category_id
JOIN Order_Items ON Products.product_id = Order_Items.product_id
GROUP BY Categories.category_id
ORDER BY total_revenue DESC;
```

7. **Using a JOIN statement, retrieve the order details (order_id, product name, quantity, and price) for all orders placed by a specific user with (user_id = 1).**

```
SELECT Orders.order_id, Products.name, Order_Items.quantity, Order_Items.price
FROM Orders
JOIN Order_Items ON Orders.order_id = Order_Items.order_id
JOIN Products ON Order_Items.product_id = Products.product_id
WHERE Orders.user_id = 1;
```

Task 2: Triggers [4*2 = 8]

1. **Create a trigger that automatically updates the total_amount field in the Orders table when a new record is inserted into the Order_Items table. The trigger should add the price of the new order item multiplied by its quantity to the existing total_amount value for that order.**

```
DELIMITER //
CREATE TRIGGER update_order_total_amount
AFTER INSERT ON Order_Items
FOR EACH ROW
BEGIN
  UPDATE Orders
  SET total_amount = total_amount + (NEW.price * NEW.quantity)
  WHERE order_id = NEW.order_id;
END;
//
```

DELIMITER ;

2. **Create a trigger that automatically updates the stock of a product whenever a new order is placed.**

```
DELIMITER //
CREATE TRIGGER update_product_stock
AFTER INSERT ON Order_Items
FOR EACH ROW
BEGIN
  UPDATE Products
  SET stock = stock - NEW.quantity
  WHERE product_id = NEW.product_id;
END;
//
DELIMITER ;
```

Task 3: Procedures [4]

1. **Create a stored procedure that accepts a user_id and a product_id as input parameters and adds the product to the user's cart. If the product is already in the user's cart, update the quantity. If the product is not in the user's cart, insert a new row with the specified user_id, product_id, and a quantity of 1. Also write the test query to call the procedure.**

```
DELIMITER //
CREATE PROCEDURE add_product_to_cart(IN p_user_id INT, IN p_product_id INT)
BEGIN
  DECLARE cart_exists INT;

  SELECT COUNT(*) INTO cart_exists
  FROM Cart_Items
  WHERE user_id = p_user_id AND product_id = p_product_id;

  IF cart_exists > 0 THEN
    UPDATE Cart_Items
    SET quantity = quantity + 1
    WHERE user_id = p_user_id AND product_id = p_product_id;
  ELSE
    INSERT INTO Cart_Items (user_id, product_id, quantity)
    VALUES (p_user_id, p_product_id, 1);
  END IF;
END;
//
DELIMITER ;

CALL add_product_to_cart(1, 101);
```

Task 4: Functions [4]

2. **Create a user-defined function that accepts a user_id as input and returns the total amount spent by the user on all their orders. The function should return the sum of the total_amount field for all orders placed by the specified user. Write a test query that call that function and print user details along with the total amount.**

```
DELIMITER //
CREATE FUNCTION total_amount_spent(p_user_id INT) RETURNS DECIMAL(10, 2)
BEGIN
  DECLARE total_amount DECIMAL(10, 2);

  SELECT SUM(total_amount) INTO total_amount
  FROM Orders
  WHERE user_id = p_user_id;

  RETURN total_amount;
END;
//
DELIMITER ;
```

```
SELECT Users.user_id, Users.first_name, Users.last_name, Users.email, total_amount_spent(Users.user_id) AS total_spent
FROM Users
WHERE Users.user_id = 1;
```