Advanced SQL Script: Complete E-Commerce Management System

```sql
-- Creating the E-Commerce Database

CREATE DATABASE EcommerceDB;

USE EcommerceDB;


-- ======================

-- □ TABLE CREATION

-- ======================


-- Users Table (Customers & Admins)

CREATE TABLE Users (

    user_id INT PRIMARY KEY AUTO_INCREMENT,

    username VARCHAR(50) UNIQUE NOT NULL,

    email VARCHAR(100) UNIQUE NOT NULL,

    password_hash VARCHAR(255) NOT NULL,

    role ENUM('Customer', 'Admin') DEFAULT 'Customer',

    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP

);


-- Categories Table

CREATE TABLE Categories (

    category_id INT PRIMARY KEY AUTO_INCREMENT,

    category_name VARCHAR(100) UNIQUE NOT NULL,

    description TEXT,
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);

-- Products Table
CREATE TABLE Products (
    product_id INT PRIMARY KEY AUTO_INCREMENT,
    category_id INT,
    product_name VARCHAR(100) NOT NULL,
    description TEXT,
    price DECIMAL(10,2) NOT NULL CHECK (price > 0),
    stock INT NOT NULL CHECK (stock >= 0),
    created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    FOREIGN KEY (category_id) REFERENCES Categories(category_id) ON DELETE SET NULL
);

-- Orders Table
CREATE TABLE Orders (
    order_id INT PRIMARY KEY AUTO_INCREMENT,
    user_id INT NOT NULL,
    order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    total_amount DECIMAL(10,2) NOT NULL CHECK (total_amount >= 0),
    status ENUM('Pending', 'Shipped', 'Delivered', 'Cancelled') DEFAULT 'Pending',
    FOREIGN KEY (user_id) REFERENCES Users(user_id) ON DELETE CASCADE
);
```

```
-- Order Items Table
CREATE TABLE Order_Items (
    order_item_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT NOT NULL,
    product_id INT NOT NULL,
    quantity INT NOT NULL CHECK (quantity > 0),
    price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE,
    FOREIGN KEY (product_id) REFERENCES Products(product_id) ON DELETE
CASCADE
);

-- Payments Table
CREATE TABLE Payments (
    payment_id INT PRIMARY KEY AUTO_INCREMENT,
    order_id INT NOT NULL,
    payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    payment_method ENUM('Credit Card', 'Debit Card', 'PayPal', 'Bank Transfer') NOT
NULL,
    amount DECIMAL(10,2) NOT NULL CHECK (amount > 0),
    status ENUM('Pending', 'Completed', 'Failed') DEFAULT 'Pending',
    FOREIGN KEY (order_id) REFERENCES Orders(order_id) ON DELETE CASCADE
);

-- ============================
-- □ INDEXING FOR PERFORMANCE
-- ============================

CREATE INDEX idx_users_email ON Users(email);
CREATE INDEX idx_orders_user ON Orders(user_id);
CREATE INDEX idx_products_name ON Products(product_name);
CREATE INDEX idx_payments_order ON Payments(order_id);

-- =================================
-- □ INSERTING SAMPLE DATA
-- =================================

INSERT INTO Users (username, email, password_hash, role) VALUES
('john_doe', 'john@example.com', 'hashed_password_1', 'Customer'),
('alice_smith', 'alice@example.com', 'hashed_password_2', 'Customer'),
('admin_01', 'admin@example.com', 'hashed_password_3', 'Admin');

INSERT INTO Categories (category_name, description) VALUES
('Electronics', 'Electronic devices and accessories'),
('Fashion', 'Clothing, shoes, and accessories'),
('Books', 'Fiction, non-fiction, and educational books');

INSERT INTO Products (category_id, product_name, description, price, stock) VALUES
(1, 'Laptop', 'High-performance laptop', 1200.00, 10),
(1, 'Smartphone', 'Latest model smartphone', 800.00, 20),
(2, 'Jeans', 'Blue denim jeans', 40.00, 50),

(3, 'Python Programming Book', 'Learn Python with real-world examples', 30.00, 100);


INSERT INTO Orders (user_id, total_amount, status) VALUES

(1, 2000.00, 'Pending'),

(2, 800.00, 'Shipped');


INSERT INTO Order_Items (order_id, product_id, quantity, price) VALUES

(1, 1, 1, 1200.00),

(1, 4, 2, 60.00),

(2, 2, 1, 800.00);


INSERT INTO Payments (order_id, payment_method, amount, status) VALUES

(1, 'Credit Card', 2000.00, 'Completed'),

(2, 'PayPal', 800.00, 'Completed');


-- ========================================

-- □ STORED PROCEDURE: FETCH USER ORDERS

-- ========================================


DELIMITER //

CREATE PROCEDURE GetUserOrders(IN userID INT)

BEGIN

    SELECT

        o.order_id, o.order_date, o.total_amount, o.status,
        p.product_name, oi.quantity, oi.price
    FROM Orders o
    JOIN Order_Items oi ON o.order_id = oi.order_id
    JOIN Products p ON oi.product_id = p.product_id
    WHERE o.user_id = userID;
END //
DELIMITER ;

-- =====================================
-- □ TRIGGER: UPDATE STOCK AFTER ORDER
-- =====================================

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

-- ===================================
-- □ VIEW: ORDER SUMMARY FOR ADMINS
-- ======================================

CREATE VIEW OrderSummary AS
SELECT
    o.order_id, u.username, u.email,
    o.total_amount, o.status, o.order_date
FROM Orders o
JOIN Users u ON o.user_id = u.user_id;

-- ==============================================
-- □ COMPLEX QUERY: ORDER & PAYMENT DETAILS
-- ==============================================

SELECT
    o.order_id, u.username, u.email,
    p.product_name, oi.quantity, oi.price,
    o.total_amount, o.status AS order_status,
    pay.payment_method, pay.amount AS payment_amount, pay.status AS payment_status
FROM Orders o
JOIN Users u ON o.user_id = u.user_id
JOIN Order_Items oi ON o.order_id = oi.order_id
JOIN Products p ON oi.product_id = p.product_id
JOIN Payments pay ON o.order_id = pay.order_id
ORDER BY o.order_date DESC;
```

```sql
-- ============================
-- □ TRANSACTIONS EXAMPLES
-- ============================

-- □ Order Placement Transaction
START TRANSACTION;
INSERT INTO Orders (user_id, total_amount, status) VALUES (3, 1350.00, 'Pending');
SET @last_order_id = LAST_INSERT_ID();
INSERT INTO Order_Items (order_id, product_id, quantity, price) VALUES
(@last_order_id, 1, 1, 1200.00),
(@last_order_id, 3, 1, 150.00);
COMMIT;

-- □ Order Cancellation Transaction with ROLLBACK
START TRANSACTION;
DELETE FROM Orders WHERE order_id = 1;
ROLLBACK;

-- ========================================
-- □ DROPPING TABLES (IF CLEANUP NEEDED)
-- ========================================

-- DROP TABLE Order_Items, Orders, Payments, Products, Categories, Users;
```