Adrian Flores Rangel Database Documentation

1. Database Schema

Users

Column	Type	Description
user_id	INT (PK)	Unique ID for each user
username	VARCHAR(50)	User's username (unique)
email	VARCHAR(100)	User's email (unique)
password	VARCHAR(255)	Encrypted password
role	ENUM	'customer' or 'admin'

Products

Column	Type	Description
product_id	INT (PK)	Unique product identifier
product_name	VARCHAR(100)	Name of the product
category	VARCHAR(50)	Product category
price	DECIMAL(10,2)	Price of the product
stock_quantity	INT	Quantity in stock

Orders

Column	Type	Description
order_id	INT (PK)	Unique order ID
user_id	INT (FK)	User who placed the order
order_date	DATETIME	Date and time of the order
total_amount	DECIMAL(10,2)	Total amount of the order
order_status	ENUM	Status: 'pending', 'shipped', etc.

OrderDetails

Column	Type	Description
order_detail_id	INT (PK)	Unique ID for each order item
order_id	INT (FK)	Reference to the order
product_id	INT (FK)	Reference to the product
quantity	INT	Quantity of the product
unit_price	DECIMAL(10,2)	Price per unit at order time

Payments

Column	Type	Description
payment_id	INT (PK)	Unique ID for each payment
order_id	INT (FK)	Associated order
payment_date	DATETIME	When the payment was made
payment_method	ENUM	'credit_card', 'paypal', etc.
amount	DECIMAL(10,2)	Amount paid

Reviews

Column	Type	Description
review_id	INT (PK)	Unique ID for each review
product_id	INT (FK)	Reviewed product
user_id	INT (FK)	Reviewer
review_text	TEXT	Review content
rating	INT	Rating (1 to 5)
review_date	DATETIME	When the review was made

2. Sample Data

Table	Code
Users	INSERT INTO Users (username, email, password, role) VALUES ('alice123', 'alice@example.com', 'hashed_password1', 'customer'), ('bob_admin', 'bob@example.com', 'hashed_password2', 'admin');

Table	Code
Products	INSERT INTO Products (product_name, category, price, stock_quantity) VALUES ('Wireless Mouse', 'Electronics', 25.99, 150), ('Yoga Mat', 'Fitness', 20.00, 120);

Table	Code
Orders	INSERT INTO Orders (user_id, order_date, total_amount, order_status) VALUES (1, '2025-05-01 10:15:00', 101.98, 'delivered');

Table	Code
OrderDetails	INSERT INTO OrderDetails (order_id, product_id, quantity, unit_price) VALUES (1, 1, 2, 25.99);

Table	Code
Payments	INSERT INTO Payments (order_id, payment_date, payment_method, amount) VALUES (1, '2025-05-01 10:16:00', 'credit_card', 101.98);

Table	Code
Reviews	INSERT INTO Reviews (product_id, user_id, review_text, rating, review_date) VALUES (1, 1, 'Very responsive and fits well in hand.', 5, '2025-05-02 11:00:00');

3. Query Explanations

Query	Code	Explanation
Get Products by Category	SELECT * FROM Products WHERE category = 'Electronics';	Returns all products in the specified category.
Get User Details by ID	SELECT * FROM Users WHERE user_id = 1;	Fetches user profile using user_id.
Get Order History for a User	SELECT order_id, order_date, total_amount, order_status FROM Orders WHERE user_id = 1;	Lists orders placed by a specific user.
Get Products in an Order	SELECT P.product_name, OD.quantity, OD.unit_price FROM OrderDetails OD JOIN Products P ON OD.product_id = P.product_id WHERE OD.order_id = 1;	Shows what was purchased in a given order.
Average Rating of a Product	SELECT AVG(rating) AS avg_rating FROM Reviews WHERE product_id = 1;	Gives the average user rating of a specific product.
Monthly Revenue Report	SELECT DATE_FORMAT(payment_date, '%Y-%m') AS month, SUM(amount) AS total_revenue FROM Payments WHERE payment_date BETWEEN '2025-05-01' AND '2025-05-31' GROUP BY month;	Returns total revenue for a given month.

4.Advanced Features

Trigger: Update Order Status After Payment	CREATE TRIGGER after_payment_insert
	AFTER INSERT ON Payments
	FOR EACH ROW
	BEGIN
	UPDATE Orders
	SET order_status = 'shipped'
	WHERE order_id = NEW.order_id;
	END;
		This trigger automatically executes after a new record is inserted into the Payments table. It updates the **order_status** field in the corresponding Orders record to 'shipped'. This ensures that once a payment is recorded, the system reflects that the order is ready for shipping, maintaining consistency between the payments and orders.

Stored Procedure: Most Active Users	CREATE PROCEDURE GetMostActiveUsers(IN min_orders INT)
	BEGIN
	SELECT
	U.user_id,
	U.username,
	
	COUNT(O.order_id) AS total_orders,
	
	SUM(O.total_amount) AS total_spent
	FROM Users U
	JOIN Orders O ON U.user_id = O.user_id
	GROUP BY
	U.user_id
	HAVING
	total_orders >= min_orders;
	END;
		This stored procedure retrieves a list of users who have placed a number of orders greater than or equal to a specified threshold (min_orders). For each user, it returns their ID, username, total number of orders, and the total amount spent. It is useful for identifying highly engaged customers and generating user activity reports.