Online Mobile Shopping Database

Project submitted to the

SRM University – AP, Andhra Pradesh

for the partial fulfillment of the requirements to award the degree of

Bachelor of Technology

In

Computer Science and Engineering

School of Engineering and Sciences

Submitted by

Cheedella Karthikeya gannesh

[AP23110010592]

Under the Guidance of

Dr. SUBHANKAR GHATAK

SRM University–AP

Neerukonda, Mangalagiri, Guntur

Andhra Pradesh – 522 240

[April, 2025]

Certificate

Date: 16-apr-25

This is to certify that the work present in this Project entitled "**Online Mobile Shopping Database**" has been carried out by cheedella karthikeya ganesh **(AP23110010592)** under my/our supervision. The work is genuine, original, and suitable for submission to the SRM University – AP for the award of Bachelor of Technology/Master of Technology in **School of Engineering and Sciences**.

Supervisor

(Signature)

Dr. SUBHANKAR GHATAK

Assistant Professor

Department of Computer Science Engineering

Acknowledgements

The excitement of finishing our Online Mobile Shopping System project is so thrilling, but it wouldn't be the same without expressing a huge thank you to all those who assisted us. This was a challenging task, and to see it completed makes us so happy, but we know that a lot of help was needed to complete it. From coding SQL to correcting our ER diagram, we had individuals mentoring us throughout the process. Their advice and support made our efforts something we're beyond proud of, and we would like to express our appreciation to them.

I would like to thank my project guide, Dr. Subhankar Ghatak Sir, Department of Computer Science & Engineering, SRM University, Andhra Pradesh, for his incredible assistance and guidance. Sir, your tutorials and advice simplified things for us, such as when you taught us to properly use foreign keys or construct queries that produce the proper results. Every time we were lost, such as with views or normalization, you provided us with ideas on how to solve it. A great big thank you as well to my project teammates for being such a wonderful team—we coded, debugged, and pushed each other through. Our friends and families were great as well, allowing us to concentrate on the project and even coming up with cool ideas such as putting up reviews for mobiles. Thank you all for this successful project that we couldn't have done without you.

Cheedella Karthikeya Ganesh

(AP23110010592)

Table of Contents

Abstract

Online shopping is growing fast, and mobile phones are one of the most popular things people buy. This project is about building a database for an **Online Mobile Shopping System** using MySQL to make shopping easier for users and simpler for store managers. We created ten tables—Users, Admins, Categories, Mobiles, Cart, CartItems, Orders, OrderDetails, Payments, and Reviews—to store all the data, like customer details, phone models, and order history. An ER diagram was drawn to show how tables connect, like how one user can have many orders. We normalized the database to Third Normal Form (3NF) to keep it clean and avoid repeated data. This helped us make the system fast and reliable for tasks like adding phones to a cart or checking payments.

We also wrote SQL queries with subqueries, joins, and aggregate functions to pull out useful information, like which mobiles have the best ratings or how much users spent. Views were added to make it easy to see things like order summaries. To test the system, we put in sample data, like users buying iPhones and leaving reviews. The project worked well, showing how a database can run an online store smoothly. It taught us a lot about designing databases and using MySQL for real-world problems. This system could grow bigger with features like search filters, but for now, it's a solid start for managing a mobile shop.

Abbreviations

Working on the **Online Mobile Shopping System** meant using a lot of technical terms, and we shortened some to make things easier. These abbreviations helped us talk about the project without repeating long phrases. Below is a list of the ones we used most, so anyone reading our report can understand what they mean. Each one is tied to something important in our database, like tables or concepts we learned.

- **DBMS**: Database Management System – The software we used (MySQL) to create and manage our database.
- **ER**: Entity-Relationship – The diagram we drew to show how tables connect, like Users to Orders.
- **PK**: Primary Key – A unique field in a table, like user_id in Users, to identify each row.
- **FK**: Foreign Key – A field that links tables, like category_id in Mobiles connecting to Categories.
- **SQL**: Structured Query Language – The language we used to write queries, like finding average ratings.
- **3NF**: Third Normal Form – The level we normalized our database to, to avoid repeated data.
- **UI**: User Interface – The front-end part (not built here) where users would shop if this was a real website.

We used these abbreviations a lot while coding and writing the report. They saved time and made it easier to explain our work. For example, saying "ER diagram" is quicker than "Entity-Relationship diagram" every time. We hope this list makes our project clearer for everyone reading it, especially since databases can get confusing with so many terms. If we missed any, it's because we tried to keep things simple and focused on what mattered most for our mobile shopping system.

1. Identification of Project Related to DBMS

The Online Mobile Shopping System is a handy and easy project that allows individuals to purchase mobile phones online. It is designed to demonstrate the functionality of a Database Management System (DBMS) in everyday life. The system utilizes MySQL to store and manage all the valuable information such as customer information, mobile phones, shopping cart, orders, payments, and reviews.

The project contains 10 interrelated tables: Users, Admins, Categories, Mobiles, Cart, CartItems, Orders, OrderDetails, Payments, and Reviews. The tables are related through foreign keys to maintain the data accurate and simple to manage. The data is stored in a neat and tidy manner (up to Third Normal Form) so there is no redundant or duplicate data.

Key features of this system are:

- Displaying mobiles by category
- Adding products to cart
- Placing an order and viewing the status of the orders
- Handling user and admin accounts
- Secure payment
- Writing reviews and ratings

This system works similar to actual shopping websites like Amazon or Flipkart but solely for mobile phones. It shows how DBMS concepts such as tables, queries, and relationships are useful in creating a functional shopping platform.

This project is a great representation of how database systems are implemented in online shopping. It makes the students learn DBMS through a real project and illustrates how online shops deal with their data in the background.

2. Project Background

Online shopping is becoming increasingly popular these days, and a lot of people like to purchase things like mobile phones online rather than visiting shops. In order to learn how such systems function internally, this project—Online Mobile Shopping System—was developed by a student as a means to learn and demonstrate the actual application of Database Management System (DBMS) concepts.

The system assists in handling the data of customers, information about products, shopping carts, orders, payments, and reviews in a systematic manner. Often in actual online stores, issues such as duplicate information, incorrect orders, or loss of products occur because the backend system is not properly managed. Through this project, it is illustrated that by using an appropriate database system, such issues are prevented.

By keeping all the data in one location with MySQL, the system maintains the data clean and linked. This makes it easy to track orders, verify available mobiles, and process payments with ease. It also enables admins to modify stock and users to leave reviews.

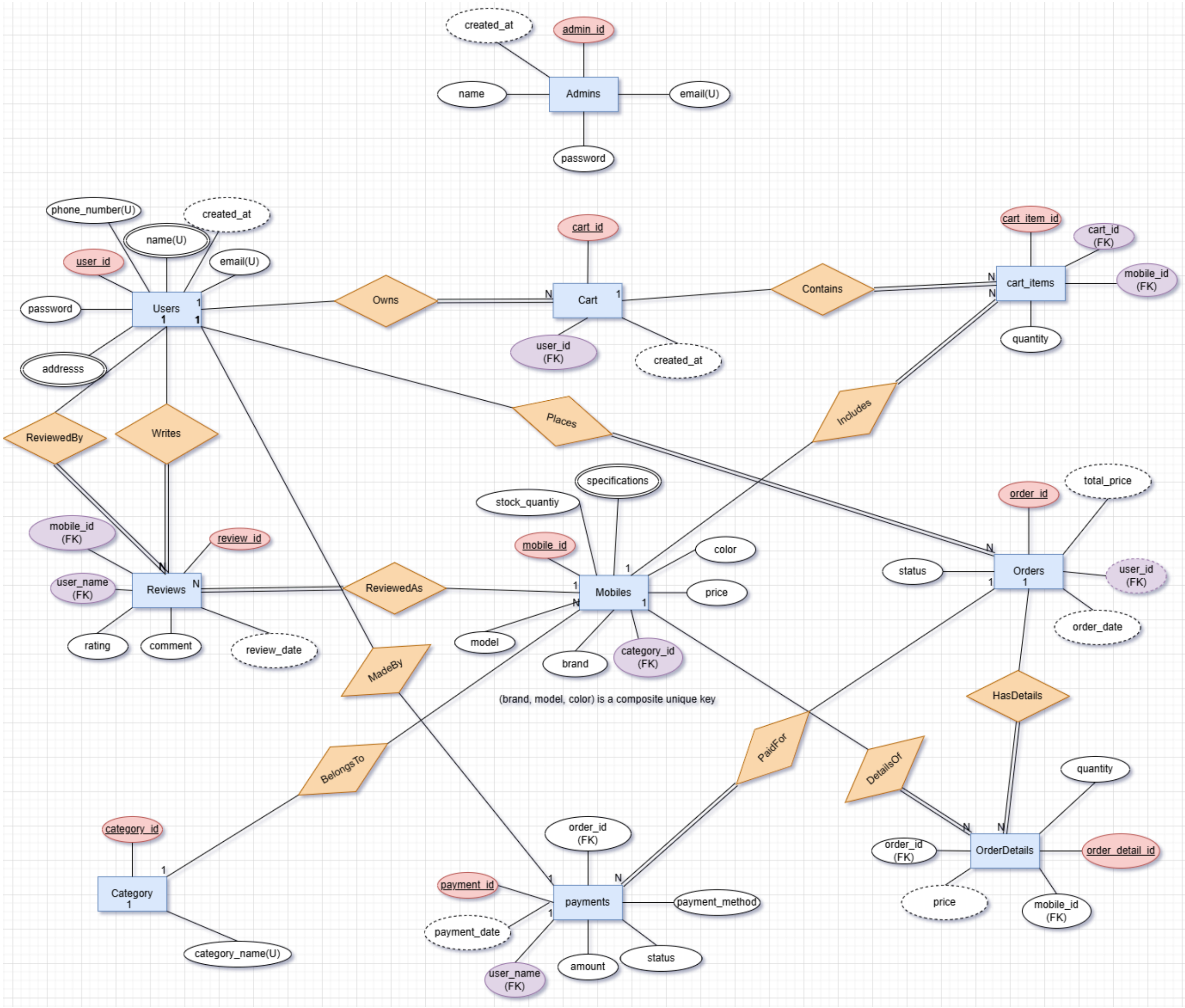
This project was done after witnessing how tough it can be to control data manually in large shopping systems. It provides a simple and clear example of how DBMS can make online shopping smooth and dependable. It is both a learning project as well as a working model of how actual e-commerce websites like Amazon or Flipkart conduct mobile phone sales using databases.

3. Description of the Project

The Online Mobile Shopping System is a database designed to support a website for purchasing mobile phones. Though not a live website, it includes all the essential components for an e-commerce platform. Here's how it works:

- **Users:** Customers who register with their name, email, password, address, and phone number. They can browse phones, add them to a cart, place orders, make payments, and leave reviews. For example, Tasneem might order two iPhones and give them a 4-star rating.
- **Admins:** Store managers who oversee operations. They can add new phone models, update prices, manage stock, or update order statuses. For instance, Aisha could add a new iPhone or mark an order as "Shipped."
- **Categories:** Groups that organize mobile phones into types, such as Smartphones, Tablets, Accessories, Laptops, or Wearables. This helps users filter and find products easily.
- **Mobiles:** The products for sale, with details like brand (e.g., Apple), model (e.g., iPhone 5), price, color, specifications (e.g., camera, storage), and stock quantity. Each mobile is linked to a category, like Smartphones.
- **Cart:** A virtual shopping basket where users add phones they want to buy. Each cart is tied to a user, allowing them to adjust quantities or remove items before checkout. For example, Bhanu might add three iPhone 5s to their cart.
- **CartItems:** Tracks the specific phones and their quantities in a user's cart. For instance, Nishanth's cart might include two iPhone 6s, with this table listing the mobile IDs and quantities.
- **Orders:** Created when a user checks out their cart. The system records the order date, status (e.g., Pending, Delivered), and total cost, linked to the user.
- **OrderDetails:** Specifies the phones, quantities, and prices in each order. For example, Prem's order of one iPhone 4 would be detailed here with its price and quantity.
- **Payments:** Manages payment methods like Credit Card, Debit Card, UPI, Net Banking, or Cash on Delivery, tracking whether payments succeed or fail.
- **Reviews:** Allows users to rate phones (1-5 stars) and leave comments to guide others. For example, Prem might praise an iPhone 4's "awesome Retina display" in a review

4. ER Diagram


created_at
admin_id
name
Admins
email(U)
password
phone_number(U)
created_at
name(U)
user_id
email(U)
password
Users
addresss
Owns
cart_id
Cart
user_id (FK)
created_at
Contains
cart_item_id
cart_id (FK)
cart_items
mobile_id (FK)
quantity
Places
Includes
ReviewedBy
Writes
mobile_id (FK)
review_id
user_name (FK)
Reviews
rating
comment
review_date
ReviewedAs
specifications
stock_quantiy
mobile_id
color
Mobiles
price
model
brand
category_id (FK)
(brand, model, color) is a composite unique key
order_id
total_price
status
Orders
user_id (FK)
order_date
MadeBy
HasDetails
BelongsTo
PaidFor
DetailsOf
quantity
category_id
Category
category_name(U)
order_id (FK)
payment_id
payments
payment_method
payment_date
user_name (FK)
amount
status
order_id (FK)
OrderDetails
order_detail_id
price
mobile_id (FK)
1
N

5. Description of ER Diagram

A graphical model of the Online Mobile Shopping System, the Entity-Relationship (ER) diagram is meticulously designed to show the structure of the database. It delineates entities, their attributes, and how they relate to one another, outlining clearly how data is structured to facilitate an e-commerce platform for mobile phones. Constructed using relational database management principles, the diagram acts as a blueprint for understanding the relationships between customers, products, and transactions in the system.

Entities and Attributes:

Users:

- **Description:** Represents customers shopping for mobiles.
- **Attributes:**
  - user_id (primary key): A unique identifier for each user.
  - name (unique): The customer's full name.
  - email (unique): A unique email address for contact and login.
  - password: A secure field for user authentication.
  - address: The delivery address, optional for flexibility.
  - phone_number (unique): A unique contact number for communication.
- **Foreign Key:** None.
- **Purpose:** Stores customer information for account management and order processing.

Admins:

- **Description:** Represents store administrators managing the system.
- **Attributes:**
  - admin_id (primary key): A unique identifier for each admin.
  - name: The admin's full name.
  - email (unique): A unique email for admin login.
  - password: A secure field for admin authentication.
- **Foreign Key:** None.
- **Purpose:** Centralizes admin data for system oversight, such as updating stock or reviewing orders.

Categories:

- **Description:** Groups mobile phones by type, such as Smartphones or Tablets.
- **Attributes:**
  - category_id (primary key): A unique identifier for each category.
  - category_name (unique): The name of the category, like "Wearables."
- **Foreign Key:** None.

- **Purpose:** Organizes products for easier browsing and filtering.

Mobiles:

- **Description:** Details the mobile phones available for purchase.
- **Attributes:**
  - mobile_id (primary key): A unique identifier for each mobile.
  - brand: The manufacturer, e.g., Apple.
  - model: The specific phone model, e.g., iPhone 15.
  - price: The cost of the mobile.
  - color: The color variant, e.g., Blue Titanium.
  - specifications: Technical details, like storage or camera specs.
  - stock_quantity: Available units, with a check to ensure non-negative values.
  - category_id: Links to the category, optional if unclassified.
- **Foreign Key:**
  - category_id: References Categories(category_id) with ON DELETE SET NULL.
- **Purpose:** Stores product data for display and inventory management.

Cart:

- **Description:** Represents a user's shopping cart.
- **Attributes:**
  - cart_id (primary key): A unique identifier for each cart.
  - user_name: The associated user's name.
  - created_at: Timestamp of cart creation.
- **Foreign Key:**
  - user_name: References Users(name) with ON DELETE CASCADE.
- **Purpose:** Tracks items a user intends to buy before checkout.

CartItems:

- **Description:** Lists specific mobiles in a cart.
- **Attributes:**
  - cart_item_id (primary key): A unique identifier for each item.
  - cart_id: Links to the cart.
  - mobile_id (FK): The mobile selected.
  - quantity: Number of units, ensuring positive values.
- **Foreign Keys:**
  - cart_id: References Cart(cart_id) with ON DELETE CASCADE.
  - mobile_id: References Mobiles(mobile_id) with ON DELETE CASCADE.
- **Purpose:** Details cart contents for accurate order preparation.

Orders:

- **Description:** Records customer purchases.
- **Attributes:**
  - order_id (primary key): A unique identifier for each order.
  - user_name: The ordering user's name.
  - order_date: Timestamp of order placement.
  - status: Order state, e.g., Pending, Shipped, Delivered, or Cancelled.
  - total_price: Total cost of the order.
- **Foreign Key:**
  - user_name: References Users(name) with ON DELETE CASCADE.
- **Purpose:** Tracks order history and status for users and admins.

OrderDetails:

- **Description:** Specifies items within an order.
- **Attributes:**
  - order_detail_id (primary key): A unique identifier for each detail.
  - order_id: Links to the order.
  - mobile_id (FK): The purchased mobile.
  - quantity: Number of units ordered.
  - price: Price per unit at purchase time.
- **Foreign Keys:**
  - order_id: References Orders(order_id) with ON DELETE CASCADE.
  - mobile_id: References Mobiles(mobile_id) with ON DELETE CASCADE.
- **Purpose:** Provides granular order information for fulfillment and records.

Payments:

- **Description:** Manages transaction details for orders.
- **Attributes:**
  - payment_id (primary key): A unique identifier for each payment.
  - order_id: Links to the order.
  - user_name: The paying user's name.
  - amount: Payment amount.
  - payment_method: Method used, e.g., Credit Card, UPI, or Cash on Delivery.
  - status: Payment outcome, either Successful or Failed.
  - payment_date: Timestamp of the transaction.
- **Foreign Keys:**
  - order_id: References Orders(order_id) with ON DELETE CASCADE.
  - user_name: References Users(name) with ON DELETE CASCADE.
- **Purpose:** Ensures secure and traceable payment processing.

Reviews:

- **Description:** Captures user feedback on purchased mobiles.
- **Attributes:**
    - review_id (primary key): A unique identifier for each review.
    - user_name: The reviewing user's name.
    - mobile_id (FK): The reviewed mobile.
    - rating: A score between 1 and 5.
    - comment: Optional feedback text.
    - review_date: Timestamp of the review.
- **Foreign Keys:**
    - user_name: References Users(name) with ON DELETE CASCADE.
    - mobile_id: References Mobiles(mobile_id) with ON DELETE CASCADE.
- **Purpose:** Collects customer opinions to enhance trust and product quality.

Relationships:

- **Users to Cart:** A "Owns" relationship, with user_name linking to Users. One user has one cart (1:1).
- **Users to Orders:** A "Places" relationship, with user_name linking to Users. One user can place multiple orders (1:N).
- **Users to Payments:** A "MadeBy" relationship, with user_name linking to Users. One user can make multiple payments (1:N).
- **Users to Reviews:** A "ReviewedBy" relationship, with user_name linking to Users. One user can write multiple reviews (1:N).
- **Categories to Mobiles:** A "BelongsTo" relationship, with category_id linking to Categories. One category can include multiple mobiles (1:N), optional for mobiles.
- **Cart to CartItems:** A "Contains" relationship, with cart_id linking to Cart. One cart can have multiple items (1:N).
- **Mobiles to CartItems:** An "Includes" relationship, with mobile_id linking to Mobiles. One mobile can appear in multiple cart items (1:N).
- **Orders to OrderDetails:** A "HasDetails" relationship, with order_id linking to Orders. One order can have multiple details (1:N).
- **Mobiles to OrderDetails:** A "DetailsOf" relationship, with mobile_id linking to Mobiles. One mobile can appear in multiple order details (1:N).
- **Orders to Payments:** A "PaidFor" relationship, with order_id linking to Orders. One order has one payment (1:1).
- **Mobiles to Reviews:** A "ReviewedAs" relationship, with mobile_id linking to Mobiles. One mobile can have multiple reviews (1:N).

Entity Participation & Cardinality Table

Relationship	Entity 1	Participation 1	Cardinality 1	Entity 2	Participation 2	Cardinality 2
Owns	Users	Total	1	Cart	Total	1
Places	Users	Partial	1	Orders	Total	N
MadeBy	Users	Partial	1	Payments	Total	N
Reviewed By	Users	Total	1	Reviews	Partial	N
Writes	Users	Partial	1	Reviews	Total	N
BelongsTo	Categories	Partial	1	Mobiles	Total	N
Contains	Cart	Partial	1	CartItems	Total	N
Includes	Mobiles	Partial	1	CartItems	Total	N
HasDetails	Orders	Partial	1	OrderDetails	Total	N
DetailsOf	Mobiles	Total	1	OrderDetails	Partial	N
PaidFor	Orders	Total	1	Payments	Partial	1
Reviewed As	Mobiles	Partial	1	Reviews	Total	N

Significance:

The ER diagram establishes the logic of the database, illustrating how entities such as Users and Mobiles are connected through actions like buying or commenting. Primary keys (e.g., user_id, mobile_id) and foreign keys (e.g., user_name, category_id) maintain data integrity and prevent issues like orphaned carts or orders. This diagram serves as an efficient blueprint for the system's architecture, demonstrating how DBMS concepts—such as relational modeling and constraints—support a robust e-commerce platform. It facilitates scalability and readability, making the Online Mobile Shopping System easier to comprehend and maintain.

6. Conversion of ER Diagram into Tables:

Our **Online Mobile Shopping System** ER diagram got turned into a proper database setup, what we call a relational schema. It's like taking a drawing and making it real with tables and connections. Each piece, like users or mobiles, becomes a table, and the links between them use foreign keys to keep everything tied up nice. This makes sure our database works smooth and makes sense for a mobile shop, like one you'd see online.

1. **Users Entity to Users Table**:
    - **Fields**:
        - user_id (INT, Primary Key, AUTO_INCREMENT): Gives each customer a unique number.
        - name (VARCHAR(255), NOT NULL, UNIQUE): Stores their full name, no repeats allowed.
        - email (VARCHAR(255), NOT NULL, UNIQUE): Their email, gotta be one-of-a-kind.
        - password (VARCHAR(255), NOT NULL): Keeps their account safe.
        - address (TEXT): Where to ship stuff, can be empty.
        - phone_number (VARCHAR(15), NOT NULL, UNIQUE): Their phone, no duplicates.
        - created_at (DATETIME, DEFAULT CURRENT_TIMESTAMP): When the user joined.

Users Table

Attribute	Data Type	Constraints
user_id	INT	Primary Key, Auto Increment
name	VARCHAR(255)	Not Null, Unique
email	VARCHAR(255)	Not Null, Unique, Indexed
password	VARCHAR(255)	Not Null
address	TEXT	
phone_number	VARCHAR(15)	Not Null, Unique
created_at	DATETIME	Default = CURRENT_TIMESTAMP

- **Constraints**:
    - None for foreign keys here.
- **Purpose**: Holds all customer info so they can shop, order, and get stuff delivered.

2. **Admins Entity to Admins Table**:
    - **Fields**:
        - admin_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each admin.
        - name (VARCHAR(255), NOT NULL): Admin's full name.
        - email (VARCHAR(255), NOT NULL, UNIQUE): Their special email for logging in.
        - password (VARCHAR(255), NOT NULL): Keeps their account locked tight.
        - created_at (DATETIME, DEFAULT CURRENT_TIMESTAMP): When the admin started.

Admins Table

Attribute	Data Type	Constraints
admin_id	INT	Primary Key, Auto Increment
name	VARCHAR(255)	Not Null
email	VARCHAR(255)	Not Null, Unique
password	VARCHAR(255)	Not Null
created_at	DATETIME	Default = CURRENT_TIMESTAMP

- **Constraints**:
    - No foreign keys needed.
- **Purpose**: Stores admin details so they can run the shop, like adding phones or checking orders.

3. **Categories Entity to Categories Table**:
    - **Fields**:
        - category_id (INT, Primary Key, AUTO_INCREMENT): Unique number for each category.
        - category_name (VARCHAR(100), NOT NULL): Name like "Smartphones" or "Tablets."

Categories Table

Attribute	Data Type	Constraints
category_id	INT	Primary Key, Auto Increment
category_name	VARCHAR(100)	Not Null, Unique

- **Constraints**:
    - No foreign keys here.

   - **Purpose**: Groups phones into types so customers can browse easier.

4. **Mobiles Entity to Mobiles Table**:
   - **Fields**:
     - mobile_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each phone.
     - brand (VARCHAR(100), NOT NULL): Who made it, like Apple.
     - model (VARCHAR(100), NOT NULL): Phone name, like iPhone 15.
     - price (DECIMAL(10,2), NOT NULL): How much it costs.
     - color (VARCHAR(100), NOT NULL): Color, like Blue.
     - specifications (TEXT): Details, like camera or storage.
     - stock_quantity (INT, NOT NULL): How many are left, can't go below zero.
     - category_id (INT): Ties to a category if there is one.

Mobiles Table Schema

Attribute	Data Type	Constraints
mobile_id	INT	Primary Key, Auto Increment
brand	VARCHAR(100)	Not Null
model	VARCHAR(100)	Not Null
price	DECIMAL(10,2)	Not Null, CHECK (price ≥ 0)
color	VARCHAR(100)	Not Null
specifications	TEXT	
stock_quantity	INT	Not Null, CHECK (stock_quantity ≥ 0)
category_id	INT	Foreign Key → Categories(category_id), ON DELETE SET NULL
UNIQUE	(brand, model, color)	Composite Unique Constraint

   - **Constraints**:
     - **Foreign Key**: category_id links to Categories(category_id), sets to NULL if category's gone.
     - **Check**: stock_quantity >= 0 to stop negative stock.
   - **Purpose**: Keeps all phone info for showing products and managing stock.

5. **Cart Entity to Cart Table**:
   - **Fields**:

        - cart_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each cart.
        - user_id (INT, NOT NULL): Who owns the cart.
        - created_at (DATETIME, DEFAULT CURRENT_TIMESTAMP): When the cart started.

Cart Table Schema

Attribute	Data Type	Constraints
cart_id	INT	Primary Key, Auto Increment
user_id	INT	Foreign Key → Users(user_id), ON DELETE CASCADE
created_at	DATETIME	Default = CURRENT_TIMESTAMP

- **Constraints**:
    - **Foreign Key**: user_id links to Users(ID), deletes cart if user's gone.

- **Purpose**: Tracks what customers wanna buy before they check out.

6. **CartItems Entity to CartItems Table**:
    - **Fields**:
        - cart_item_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each item.
        - cart_id (INT, NOT NULL): Which cart it's in.
        - mobile_id (INT, NOT NULL): The phone picked.
        - quantity (INT, NOT NULL): How many they want.

CartItems Table Schema

Attribute	Data Type	Constraints
cart_item_id	INT	Primary Key, Auto Increment
cart_id	INT	Foreign Key → Cart(cart_id), ON DELETE CASCADE
mobile_id	INT	Foreign Key → Mobiles(mobile_id), ON DELETE CASCADE
quantity	INT	Not Null, CHECK (quantity > 0)

- **Constraints**:

        - **Foreign Key**: cart_id links to Cart(cart_id), removes item if cart’s deleted.
        - **Foreign Key**: mobile_id links to Mobiles(ID), removes item if phone’s gone.
        - **Check**: quantity > 0 to make sure they pick at least one.
    - **Purpose**: Lists exactly what’s in a cart so orders are right.

7. **Orders Entity to Orders Table**:
    - **Fields**:
        - order_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each order.
        - user_id (INT, NOT NULL): Who made the order.
        - order_date (DATETIME, DEFAULT CURRENT_TIMESTAMP): When they bought it.
        - status (ENUM, DEFAULT ‘Pending’): Says if it’s Pending, Shipped, Delivered, or Cancelled.
        - total_price (DECIMAL(10,2), NOT NULL): Total cost.

Orders Table Schema

Attribute	Data Type	Constraints
order_id	INT	Primary Key, Auto Increment
user_id	INT	Foreign Key → Users(user_id), ON DELETE CASCADE
order_date	DATETIME	Default = CURRENT_TIMESTAMP
status	ENUM	('Pending', 'Shipped', 'Delivered', 'Cancelled'), Default: 'Pending'
total_price	DECIMAL(10,2)	Not Null, CHECK (total_price ≥ 0)

    - **Constraints**:
        - **Foreign Key**: user_name links to Users(name), deletes order if user’s gone.
        - **Check**: status only allows ‘Pending’, ‘Shipped’, ‘Delivered’, ‘Cancelled’.
    - **Purpose**: Keeps track of what customers bought and where their order’s at.

8. **OrderDetails Entity to OrderDetails Table**:
    - **Fields**:

        - order_detail_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each item in an order.
        - order_id (INT, NOT NULL): Which order it's part of.
        - mobile_id (INT, NOT NULL): The phone bought.
        - quantity (INT, NOT NULL): How many they got.
        - price (DECIMAL(10,2), NOT NULL): Price for each phone.

OrderDetails Table Schema

Attribute	Data Type	Constraints
order_detail_id	INT	Primary Key, Auto Increment
order_id	INT	Foreign Key → Orders(order_id), ON DELETE CASCADE
mobile_id	INT	Foreign Key → Mobiles(mobile_id), ON DELETE CASCADE
quantity	INT	Not Null, CHECK (quantity > 0)
price	DECIMAL(10,2)	Not Null, CHECK (price ≥ 0)

    - **Constraints**:
        - **Foreign Key**: order_id links to Orders(order_id), deletes details if order's gone.
        - **Foreign Key**: mobile_id links to Mobiles(id), deletes details if phone's gone.
        - **Check**: quantity > 0 to ensure they bought something.
    - **Purpose**: Shows exactly what phones are in each order for shipping and records.

9. **Payments Entity to Payments Table**:
    - **Fields**:
        - payment_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each payment.
        - order_id (INT, NOT NULL): Which order it's for.
        - user_name (VARCHAR(255), NOT NULL): Who paid.
        - amount (DECIMAL(10,2), NOT NULL): How much they paid.
        - payment_method (ENUM, NOT NULL): How they paid, like Credit Card or UPI.
        - status (ENUM, DEFAULT 'Successful'): Says if it worked or failed.
        - payment_date (DATETIME, DEFAULT CURRENT_TIMESTAMP): When they paid.

Payments Table Schema

Attribute	Data Type	Constraints
payment_id	INT	Primary Key, Auto Increment
order_id	INT	Foreign Key → Orders(order_id), ON DELETE CASCADE
user_name	VARCHAR(255)	Not Null, Foreign Key → Users(name), ON DELETE CASCADE
amount	DECIMAL(10,2)	Not Null, CHECK (amount ≥ 0)
payment_method	ENUM	('Credit Card', 'Debit Card', 'UPI', 'Net Banking', 'Cash on Delivery')
status	ENUM	('Successful', 'Failed'), Default = 'Successful'
payment_date	DATETIME	Default = CURRENT_TIMESTAMP

- **Constraints**:
  - **Foreign Key**: order_id links to Orders(order_id), deletes payment if order's gone.
  - **Foreign Key**: user_name links to Users(name), deletes payment if user's gone.
  - **Check**: payment_method only allows 'Credit Card', 'Debit Card', 'UPI', 'Net Banking', 'Cash on Delivery'.
  - **Check**: status only allows 'Successful', 'Failed'.
- **Purpose**: Tracks payments so we know orders are paid for.

10. **Reviews Entity to Reviews Table**:
    - **Fields**:
      - review_id (INT, Primary Key, AUTO_INCREMENT): Unique ID for each review.
      - user_name (VARCHAR(255), NOT NULL): Who wrote it.
      - mobile_id (INT, NOT NULL): The phone they're talking about.
      - rating (INT): Score from 1 to 5.
      - comment (TEXT): What they said, if anything.
      - review_date (DATETIME, DEFAULT CURRENT_TIMESTAMP): When they wrote it.

Reviews Table Schema

Attribute	Data Type	Constraints
review_id	INT	Primary Key, Auto Increment
user_name	VARCHAR(255)	Not Null, Foreign Key → Users(name), ON DELETE CASCADE
mobile_id	INT	Foreign Key → Mobiles(mobile_id), ON DELETE CASCADE
rating	INT	CHECK (rating BETWEEN 1 AND 5)
comment	TEXT	
review_date	DATETIME	Default = CURRENT_TIMESTAMP

- **Constraints**:
  - **Foreign Key**: user_name links to Users(name), deletes review if user's gone.
  - **Foreign Key**: mobile_id links to Mobiles(id), deletes review if phone's gone.
  - **Check**: rating must be between 1 and 5.
- **Purpose**: Saves customer feedback to help others pick phones.

Relationship Implementation:

Foreign keys make sure everything's connected right:

- Mobiles.category_id ties phones to categories.
- Cart.user_name links carts to users.
- CartItems.cart_id and CartItems.mobile_id connect items to carts and phones.
- Orders.user_name ties orders to users.
- OrderDetails.order_id and OrderDetails.mobile_id link order items to orders and phones.
- Payments.order_id and Payments.user_name connect payments to orders and users.
- Reviews.user_name and Reviews.mobile_id link reviews to users and phones.

Integrity Measures:

- If a user gets deleted, their cart, orders, payments, and reviews go too (ON DELETE CASCADE) so there's no loose ends.
- If a mobile's removed, cart items, order details, and reviews for it get deleted (ON DELETE CASCADE) to keep things clean.

- If a category’s deleted, Mobiles.category_id just becomes NULL (ON DELETE SET NULL) so phones don’t vanish.
- Checks make sure stuff like stock_quantity stays above zero, quantity is more than zero, and rating stays between 1 and 5.

7. Description of tables:

1. **Users**:
   - **Purpose**: Stores info about customers who shop.
   - **Fields**: user_id (unique ID), name (like Tasneem), email (unique), password (for login), address (for delivery), phone_number (unique).
   - **Example**: Tasneem's record has user_id = 1, email = tasneem@example.com, and phone = 9876543210.
   - **Why Unique?**: email and phone_number ensure no two users have the same contact info.

2. **Admins**:
   - **Purpose**: Stores info about shop managers.
   - **Fields**: admin_id (unique ID), name (like Aisha Khan), email (unique), password.
   - **Example**: Aisha has admin_id = 1, email = aisha.khan@admin.com.
   - **Why Separate?**: Admins don't shop—they manage—so they're in a different table.

3. **Categories**:
   - **Purpose**: Groups mobiles into types.
   - **Fields**: category_id (unique ID), category_name (like Smartphones).
   - **Example**: category_id = 1 is Smartphones, category_id = 2 is Tablets.
   - **Why?**: Helps users filter phones by type.

4. **Mobiles**:
   - **Purpose**: Stores all phone details.
   - **Fields**: mobile_id (unique ID), brand, model, price, color, specifications, stock_quantity, category_id (FK).
   - **Example**: mobile_id = 1, brand = Apple, model = iPhone, price = 399.99, stock_quantity = 20.
   - **Why?**: This is the core product table users browse.

5. **Cart**:
   - **Purpose**: Tracks each user's shopping cart.
   - **Fields**: cart_id (unique ID), user_id (FK), created_at (when cart was made).
   - **Example**: cart_id = 1, user_id = 1, created_at = 2025-04-01.
   - **Why?**: Lets users save phones before buying.

6. **CartItems**:
   - **Purpose**: Lists phones in a cart.

- **Fields**: cart_item_id (unique ID), cart_id (FK), mobile_id (FK), quantity.
- **Example**: cart_item_id = 1, cart_id = 1, mobile_id=1, quantity = 2.
- **Why?**: Shows what's in a cart and how many

7. **Orders**:
   - **Purpose**: Tracks purchases.
   - **Fields**: order_id (unique ID), user_id (FK), order_date, status (Pending, Shipped, etc.), total_price.
   - **Example**: order_id = 1, user_id = 1, total_price = 799.98.
   - **Why?**: Records what users buy and when.

8. **OrderDetails**:
   - **Purpose**: Lists phones in an order.
   - **Fields**: order_detail_id (unique ID), order_id (FK), mobile_id (FK), quantity, price.
   - **Example**: order_detail_id = 1, order_id = 1, mobile_id = 1, quantity = 2.
   - **Why?**: Breaks down orders into specific items.

9. **Payments**:
   - **Purpose**: Tracks how users pay.
   - **Fields**: payment_id (unique ID), order_id (FK), user_name (FK), amount, payment_method, status, payment_date.
   - **Example**: payment_id = 1, order_id = 1, user_name = Tasneem, amount = 799.98.
   - **Why?**: Ensures payments are recorded correctly.

10. **Reviews**:
    - **Purpose**: Stores user feedback.
    - **Fields**: review_id (unique ID), user_name (FK), mobile_id (FK), rating, comment, review_date.
    - **Example**: review_id = 1, user_name= Tasneem, mobile_id = 1, rating = 4.
    - **Why?**: Helps other users choose phones based on ratings.

8. Normalization of Tables up to 3-NF

Our **Online Mobile Shopping System** database got cleaned up through normalization to make it super efficient, avoid repeated stuff, and stop problems when adding or deleting things. We went through three steps—First Normal Form (1NF), Second Normal Form (2NF), and Third Normal Form (3NF)—to get a tidy setup. It's like organizing a messy shop so everything's easy to find and works right.

Initial Unnormalized Form

At first, we imagined all the data stuffed into one big table called **MobileShop**, holding everything about customers, phones, and orders:

- **Attributes**:
  - Customer details: customer_name, customer_email, customer_phone, customer_address, customer_password.
  - Phone details: phone_brand, phone_model, phone_price, phone_color, phone_specs, phone_stock, category_name.
  - Cart details: cart_items (e.g., "iPhone 15:2"), cart_date.
  - Order details: order_date, order_status, order_total, ordered_items (e.g., "iPhone 15:2,$1000;Galaxy S23:1,$800").
  - Payment details: payment_amounts (e.g., "$1000,$800"), payment_methods (e.g., "UPI,Credit Card"), payment_dates.
  - Review details: review_ratings (e.g., "4,5"), review_comments (e.g., "Great phone, Awesome"), review_dates.
- **Issues**:
  - Lists like cart_items or ordered_items broke the rule of single values per field.
  - Stuff like category_name got repeated for every phone, wasting space.
  - Changing a customer's email meant fixing it in tons of places, which could mess things up.

First Normal Form (1NF)

To fix the mess, we split the data so every field holds just one value and gave tables proper IDs:

- **MobileShop**:
  - **Attributes**: customer_name, customer_email, customer_phone, customer_address, customer_password, phone_model, phone_brand, phone_price, phone_color, phone_specs, phone_stock, category_name, order_date, order_status, order_total, cart_date.
  - **Primary Key**: (customer_name, phone_model).
- **CartItems**:

    - **Attributes**: customer_name, phone_model, cart_quantity, cart_date.
    - **Primary Key**: (customer_name, phone_model, cart_date).
- **OrderedItems**:
    - **Attributes**: customer_name, phone_model, order_quantity, item_price, order_date, order_status, order_total.
    - **Primary Key**: (customer_name, phone_model, order_date).

- **Payments**:
    - **Attributes**: customer_name, order_date, payment_amount, payment_method, payment_date.
    - **Primary Key**: (customer_name, order_date, payment_date).

- **Reviews**:
    - **Attributes**: customer_name, phone_model, review_rating, review_comment, review_date.
    - **Primary Key**: (customer_name, phone_model, review_date).

- **Compliance**:
    - Got rid of lists, so fields like cart_quantity hold one number.
    - Added primary keys to identify rows.
    - Still had issues, like phone_price depending only on phone_model, not the whole key.

Second Normal Form (2NF)

Next, we fixed it so every field depends on the full primary key, not just part of it:

- **MobileShop**:
    - Split because customer and phone stuff didn't all depend on both customer_name and phone_model.

- **Transformation**:
    - **Customers**:
        - **Attributes**: customer_name (Primary Key), customer_email, customer_phone, customer_address, customer_password.
    - **Phones**:
        - **Attributes**: phone_model (Primary Key), phone_brand, phone_price, phone_color, phone_specs, phone_stock, category_name.
    - **Carts**:
        - **Attributes**: cart_id (Primary Key), customer_name, cart_date.
    - **CartItems**:
        - **Attributes**: cart_item_id (Primary Key), cart_id, phone_model, cart_quantity.

  - **Orders**:
    - **Attributes**: order_id (Primary Key), customer_name, order_date, order_status, order_total.
  - **OrderedItems**:
    - **Attributes**: order_detail_id (Primary Key), order_id, phone_model, order_quantity, item_price.
  - **Payments**:
    - **Attributes**: payment_id (Primary Key), order_id, customer_name, payment_amount, payment_method, payment_date.
  - **Reviews**:
    - **Attributes**: review_id (Primary Key), customer_name, phone_model, review_rating, review_comment, review_date.
- **Compliance**:
  - Fields like phone_price now depend on phone_model alone in Phones.
  - Used IDs like cart_id to link tables properly.
  - Still had repeats, like category_name in Phones depending on something else.

Third Normal Form (3NF)

Finally, we got rid of fields depending on other non-key fields to make it super clean:

- **Customers**:
  - Had customer_email depending on customer_name, which is okay since it's the key.

- **Phones**:
  - category_name depended on some category ID, not phone_model.

- **Transformation**:
  - **Users**:
    - **Attributes**: user_id (Primary Key), name, email, password, address, phone_number.
  - **Admins**:
    - **Attributes**: admin_id (Primary Key), name, email, password.
  - **Categories**:
    - **Attributes**: category_id (Primary Key), category_name.
  - **Mobiles**:
    - **Attributes**: mobile_id (Primary Key), brand, model, price, color, specifications, stock_quantity, category_id.
  - **Cart**:
    - **Attributes**: cart_id (Primary Key), user_name, created_at.

- **CartItems**:
    - **Attributes**: cart_item_id (Primary Key), cart_id, mobile_model, quantity.
- **Orders**:
    - **Attributes**: order_id (Primary Key), user_name, order_date, status, total_price.
- **OrderDetails**:
    - **Attributes**: order_detail_id (Primary Key), order_id, mobile_model, quantity, price.
- **Payments**:
    - **Attributes**: payment_id (Primary Key), order_id, user_name, amount, payment_method, status, payment_date.
- **Reviews**:
    - **Attributes**: review_id (Primary Key), user_name, mobile_model, rating, comment, review_date.

- **Compliance**:
    - Moved category_name to Categories, so Mobiles uses category_id.
    - Made sure fields like email in Users depend only on user_id.
    - Linked tables with foreign keys, like user_name in Orders to Users.

Final Schema:

We ended up with 10 tables—Users, Admins, Categories, Mobiles, Cart, CartItems, Orders, OrderDetails, Payments, Reviews—all connected right.

Why 3NF Matters:

- Saves space by not repeating things, like category_name only in Categories.
- Makes changes easy—update one field, like a user's email, without touching every table.
- Stops mistakes, like deleting a phone without losing its reviews, thanks to foreign keys.

Normalization Compliance (Up to 3NF)

Table Name	1NF (Atomicity)	2NF (Partial Dependency Removed)	3NF (Transitive Dependency Removed)	Explanation
Users	Yes	Yes	Yes	All attributes (e.g., name, email) depend only on user_id. No

				lists or derived data.
Admins	Yes	Yes	Yes	Each admin is uniquely identified by admin_id. All fields atomic and non-transitive.
Categories	Yes	Yes	Yes	Each category has unique category_id. No repeated or derived data.
Mobiles	Yes	Yes	Yes	Each mobile identified by mobile_id. category_name moved to Categories via category_id.
Cart	Yes	Yes	Yes	Each cart has cart_id. Linked to Users. Attributes are atomic and dependent on PK.
CartItems	Yes	Yes	Yes	Each item tied to cart_item_id. Quantity depends on full key, not partial.
Orders	Yes	Yes	Yes	Each order identified by order_id. No partial or transitive dependencies.
OrderDetails	Yes	Yes	Yes	Linked to Orders. Attributes like price and quantity

				depend fully on PK.
Payments	Yes	Yes	Yes	Each payment has a unique payment_id. Linked through order_id. Atomic attributes.
Reviews	Yes	Yes	Yes	Each review has review_id. All fields (rating, comment) directly tied to PK.

9. Creation of Data in the Tables:

1. **Users Table**:
   - **Records**:
     - ('Tasneem', 'tasneem@example.com', 'hashedpass123', '123 Rose St, City A', '9876543210')
     - ('Prem', 'prem@example.com', 'hashedpass456', '456 Oak Ave, City B', '9876543211')
     - ('Bhanu', 'bhanu@example.com', 'hashedpass789', '789 Pine Rd, City C', '9876543212')
     - ('Karthikeya', 'karthikeya@example.com', 'hashedpass101', '101 Maple Ln, City D', '9876543213')
     - ('Nishanth', 'nishanth@example.com', 'hashedpass202', '202 Birch St, City E', '9876543214')
     - ('Lokesh', 'lokesh@example.com', 'hashedpass303', '303 Cedar Dr, City F', '9876543215')
     - ('Vamsi', 'vamsi@example.com', 'hashedpass404', '404 Elm St, City G', '9876543216')
     - ('Tharun', 'tharun@example.com', 'hashedpass505', '505 Willow Ct, City H', '9876543217')

Users Table - Records

user_id	name	email	password	address	phone
1	Tasneem	tasneem@example.com	hashedpass123	123 Rose St, City A	9876543210
2	Prem	prem@example.com	hashedpass456	456 Oak Ave, City B	9876543211
3	Bhanu	bhanu@example.com	hashedpass789	789 Pine Rd, City C	9876543212
4	Karthikeya	karthikeya@example.com	hashedpass101	101 Maple Ln, City D	9876543213
5	Nishanth	nishanth@example.com	hashedpass202	202 Birch St, City E	9876543214
6	Lokesh	lokesh@example.com	hashedpass303	303 Cedar Dr, City F	9876543215
7	Vamsi	vamsi@example.com	hashedpass404	404 Elm St, City G	9876543216
8	Tharun	tharun@example.com	hashedpass505	505 Willow Ct, City H	9876543217

   - **Purpose**: Shows a group of customers who shop, with unique emails and phones.

2. **Admins Table**:
   - **Records**:
     - ('Aisha Khan', 'aisha.khan@admin.com', 'adminpass901')
     - ('Rahul Sharma', 'rahul.sharma@admin.com', 'adminpass902')
     - ('Sophie Lee', 'sophie.lee@admin.com', 'adminpass903')
     - ('Carlos Rivera', 'carlos.rivera@admin.com', 'adminpass904')
     - ('Priya Patel', 'priya.patel@admin.com', 'adminpass905')

Admins Table Records

admin_id	name	email	password
1	Aisha Khan	aisha.khan@admin.com	adminpass901
2	Rahul Sharma	rahul.sharma@admin.com	adminpass902
3	Sophie Lee	sophie.lee@admin.com	adminpass903
4	Carlos Rivera	carlos.rivera@admin.com	adminpass904
5	Priya Patel	priya.patel@admin.com	adminpass905

   - **Purpose**: Represents admins who run the shop, like updating stock or checking orders.

3. **Categories Table**:
   - **Records**:
     - ('Smartphones')
     - ('Tablets')
     - ('Accessories')
     - ('Laptops')
     - ('Wearables')

Categories Table Records

category_id	category_name
1	Smartphones
2	Tablets
3	Accessories
4	Laptops
5	Wearables

   - **Purpose**: Sets up different product groups for organizing phones and gadgets.

4. **Mobiles Table**:
   - **Records** (shortened for brevity):
     - ('Apple', 'iPhone', 399.99, 'White', '3.5-inch display, 2MP Camera, 16GB Storage', 20, 1)

- ('Apple', 'iPhone', 399.99, 'Black', '3.5-inch display, 2MP Camera, 16GB Storage', 20, 1)
- ('Apple', 'iPhone 3G', 499.99, 'Black', '3.5-inch display, 3G Connectivity, 16GB Storage', 25, 1)
- ('Apple', 'iPhone 4', 699.99, 'Black', '3.5-inch Retina display, 5MP Camera, 32GB Storage', 35, 1)
- ('Apple', 'iPhone 5', 799.99, 'Black', '4-inch Retina display, 8MP Camera, 64GB Storage', 50, 1)
- ('Apple', 'iPhone 5s', 699.99, 'Silver', '4-inch Retina display, Touch ID, 8MP Camera, 64GB Storage', 50, 1)
- ('Apple', 'iPhone 6', 899.99, 'Silver', '4.7-inch Retina display, 8MP Camera, 128GB Storage', 55, 1)
- ('Apple', 'iPhone 6s', 749.99, 'Silver', '4.7-inch Retina display, 12MP Camera, 128GB Storage', 65, 1)
- ('Apple', 'iPhone 7', 899.99, 'Silver', '4.7-inch Retina display, No Headphone Jack, 12MP Camera, 256GB Storage', 75, 1)
- ('Apple', 'iPhone 8', 799.99, 'Silver', '4.7-inch Retina display, Wireless Charging, 12MP Camera, 256GB Storage', 90, 1)

Mobiles Table Records

mobile_id	brand	model	price	color	specs	stock	category_id
1	Apple	iPhone	399.99	White	3.5-inch display, 2MP Camera, 16GB Storage	20	1
2	Apple	iPhone	399.99	Black	3.5-inch display, 2MP Camera, 16GB Storage	20	1
3	Apple	iPhone 3G	499.99	Black	3.5-inch display, 3G Connectivity, 16GB Storage	25	1
7	Apple	iPhone 4	699.99	Black	3.5-inch Retina display, 5MP Camera, 32GB Storage	35	1
11	Apple	iPhone 5	799.99	Black	4-inch Retina display, 8MP	50	1

					Camera, 64GB Storage		
18	Apple	iPhone 5s	699.99	Silver	4-inch Retina display, Touch ID, 8MP Camera, 64GB Storage	50	1
21	Apple	iPhone 6	899.99	Silver	4.7-inch Retina display, 8MP Camera, 128GB Storage	55	1
27	Apple	iPhone 6s	749.99	Silver	4.7-inch Retina display, 12MP Camera, 128GB Storage	65	1
37	Apple	iPhone 7	899.99	Silver	4.7-inch Retina display, No Headphone Jack, 12MP Camera, 256GB Storage	75	1
45	Apple	iPhone 8	799.99	Silver	4.7-inch Retina display, Wireless Charging, 12MP Camera, 256GB Storage	90	1

- **Purpose**: Lists phones for sale, with different models and colors to show variety.

5. **Cart Table**:
   - **Records**:

        - ('2025-04-01 10:15:23')
        - ('2025-04-02 14:30:45')
        - ('2025-04-03 09:10:12')
        - ('2025-04-04 16:25:37')
        - ('2025-04-05 11:45:00')
        - ('2025-04-05 20:15:19')
        - ('2025-04-06 08:30:55')
        - ('2025-04-06 13:20:40')

Cart Table

cart_id	user_name	creation_date
1	1	2025-04-01 10:15:23
2	2	2025-04-02 14:30:45
3	3	2025-04-03 09:10:12
4	4	2025-04-04 16:25:37
5	5	2025-04-05 11:45:00
6	6	2025-04-05 20:15:19
7	7	2025-04-06 08:30:55
8	8	2025-04-06 13:20:40

- **Purpose**: Shows each customer's shopping cart, ready to hold phones.

6. **CartItems Table**:
    - **Records**:
        - (1, 2) -- Tasneem: 2 iPhones (White)
        - (7, 1) -- Prem: 1 iPhone 4 (Black)
        - (11, 3) -- Bhanu: 3 iPhone 5 (Black)
        - (18, 1) -- Karthikeya: 1 iPhone 5s (Silver)
        - (21, 2) -- Nishanth: 2 iPhone 6 (Silver)
        - (27, 1) -- Lokesh: 1 iPhone 6s (Silver)
        - (37, 1) -- Vamsi: 1 iPhone 7 (Silver)
        - (45, 2) -- Tharun: 2 iPhone 8 (Silver)

CartItems Table

cart_item_id	card_id	mobile_id	quantity
1	1	1	2
2	2	7	1
3	3	11	3
4	4	18	1
5	5	21	2
6	6	27	1
7	7	37	1
8	8	45	2

    - **Purpose**: Lists phones in each cart, showing what customers want to buy.

7. **Orders Table**:
    - **Records**:
        - (1, '2025-04-03 12:00:00', 'Shipped', 799.98)
        - (2, '2025-04-04 15:30:00', 'Delivered', 699.99)
        - (3, '2025-04-05 09:15:00', 'Pending', 2399.97)
        - (4, '2025-04-06 10:00:00', 'Cancelled', 699.99)
        - (5, '2025-04-06 14:45:00', 'Shipped', 1799.98)

Orders Table

order_id	user_id	order_date	status	total_price
1	1	2025-04-03 12:00:00	Shipped	799.98
2	2	2025-04-04 15:30:00	Delivered	699.99
3	3	2025-04-05 09:15:00	Pending	2399.97
4	4	2025-04-06 10:00:00	Cancelled	699.99
5	5	2025-04-06 14:45:00	Shipped	1799.98

    - **Purpose**: Tracks customer orders, with different statuses like Shipped or Cancelled.

8. **OrderDetails Table**:
    - **Records**:
        - (1, 1, 2, 399.99) -- Tasneem: 2 iPhones (White)
        - (2, 7, 1, 699.99) -- Prem: 1 iPhone 4 (Black)
        - (3, 11, 3, 799.99) -- Bhanu: 3 iPhone 5 (Black)
        - (4, 18, 1, 699.99) -- Karthikeya: 1 iPhone 5s (Silver)
        - (5, 21, 2, 899.99) -- Nishanth: 2 iPhone 6 (Silver)

OrderDetails Table

order_detail_id	order_id	mobile_id	quantity	price
1	1	1	2	399.99
2	2	7	1	699.99
3	3	11	3	799.99
4	4	18	1	699.99
5	5	21	2	899.99

    - **Purpose**: Shows exactly which phones are in each order, with quantities and prices.

9. **Payments Table**:
    - **Records**:
        - (1, 'Tasneem', 799.98, 'Credit Card', 'Successful', '2025-04-03 12:05:00')
        - (2, 'Prem', 699.99, 'UPI', 'Successful', '2025-04-04 15:35:00')
        - (3, 'Bhanu', 2399.97, 'Net Banking', 'Successful', '2025-04-05 09:20:00')
        - (4, 'Karthikeya', 699.99, 'Cash on Delivery', 'Failed', '2025-04-06 10:05:00')
        - (5, 'Nishanth', 1799.98, 'Debit Card', 'Successful', '2025-04-06 14:50:00')

Payments Table

payment_id	order_id	user_name	amount	payment_method	payment_status	payment_date
1	1	Tasneem	799.98	Credit Card	Successful	2025-04-03 12:05:00
2	2	Prem	699.99	UPI	Successful	2025-04-04 15:35:00
3	3	Bhanu	2399.97	Net Banking	Successful	2025-04-05 09:20:00
4	4	Karthikeya	699.99	Cash on Delivery	Failed	2025-04-06 10:05:00
5	5	Nishanth	1799.98	Debit Card	Successful	2025-04-06 14:50:00

    - **Purpose**: Records how customers paid, including one failed payment to test issues.

10. **Reviews Table**:
    - **Records**:
        - ('Tasneem', 'iPhone', 4, 'Great phone for its time!', '2025-04-04 10:00:00')
        - ('Prem', 'iPhone 4', 5, 'Love the Retina display!', '2025-04-05 12:30:00')
        - ('Bhanu', 'iPhone 5', 3, 'Good, but battery life could be better.', '2025-04-06 08:15:00')
        - ('Nishanth', 'iPhone 6', 4, 'Solid upgrade, sleek design.', '2025-04-06 15:00:00')
        - ('Vamsi', 'iPhone 7', 5, 'No headphone jack, but amazing performance!', '2025-04-06 16:00:00')

Reviews Table

review_id	user_name	mobile_name	rating	comment	review_date
1	Tasneem	iPhone	4	Great phone for its time!	2025-04-04 10:00:00
2	Prem	iPhone 4	5	Love the Retina display!	2025-04-05 12:30:00
3	Bhanu	iPhone 5	3	Good, but battery life could be better.	2025-04-06 08:15:00
4	Nishanth	iPhone 6	4	Solid upgrade, sleek design.	2025-04-06 15:00:00
5	Vamsi	iPhone 7	5	No headphone jack, but amazing performance!	2025-04-06 16:00:00

- **Purpose**: Captures customer feedback on phones, showing different ratings and comments.

10. SQL Queries on the Created Tables:

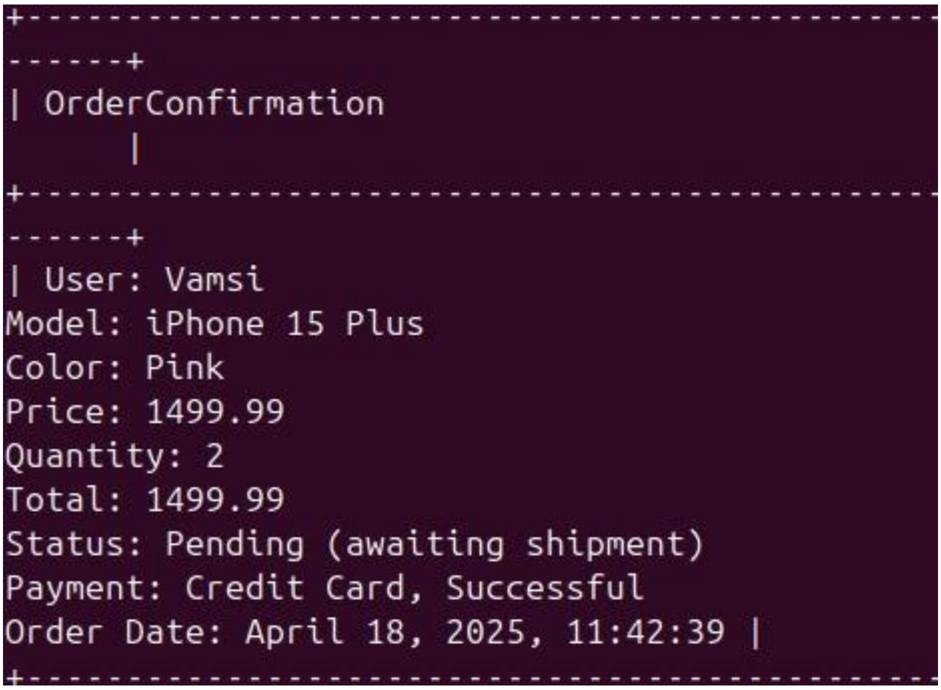
–QUERY TO ORDER A MOBILE

```
DELIMITER $$

CREATE PROCEDURE PlaceOrder()

BEGIN

  DECLARE available_stock INT DEFAULT 0;

  DECLARE userId INT;

  DECLARE mobileId INT;

  DECLARE orderId INT;

  -- Get user_id from user name

  SELECT user_id INTO userId FROM Users WHERE name = @user_name;

  -- Get mobile_id from model and color

  SELECT mobile_id, stock_quantity INTO mobileId, available_stock

  FROM Mobiles

  WHERE model = @model AND color = @color;

  IF available_stock >= @quantity THEN

    START TRANSACTION;

    -- Insert into Orders

    INSERT INTO Orders (user_id, order_date, status, total_price)

    VALUES (userId, NOW(), 'Pending', 1499.99);

    SET orderId = LAST_INSERT_ID();

    -- Insert into OrderDetails

    INSERT INTO OrderDetails (order_id, mobile_id, quantity, price)

    VALUES (orderId, mobileId, @quantity, 1499.99);

    -- Insert into Payments
```

```
INSERT INTO Payments (order_id, user_name, amount, payment_method, status, payment_date)
VALUES (orderId, @user_name, 1499.99, 'Credit Card', 'Successful', NOW());
-- Update stock
UPDATE Mobiles
SET stock_quantity = stock_quantity - @quantity
WHERE mobile_id = mobileId;
COMMIT;
-- Final Output
SELECT CONCAT(
    'User: ', u.name, '\n',
    'Model: ', m.model, '\n',
    'Color: ', m.color, '\n',
    'Price: ', od.price, '\n',
    'Quantity: ', od.quantity, '\n',
    'Total: ', o.total_price, '\n',
    'Status: ', o.status, ' (awaiting shipment)', '\n',
    'Payment: ', p.payment_method, ', ', p.status, '\n',
    'Order Date: ', DATE_FORMAT(o.order_date, '%M %d, %Y, %H:%i:%s')
) AS OrderConfirmation
FROM Orders o
JOIN OrderDetails od ON o.order_id = od.order_id
JOIN Payments p ON o.order_id = p.order_id
JOIN Mobiles m ON od.mobile_id = m.mobile_id
JOIN Users u ON o.user_id = u.user_id
```

```
        WHERE o.order_id = orderId;

    ELSE

        ROLLBACK;

        SELECT 'Insufficient stock for this model and color!' AS Message;

    END IF;

END$$

DELIMITER ;

–INITIALIZING VALUES  IN SIDE THE FUNCTION
SET @user_name = 'Vamsi';

SET @model = 'iPhone 15 plus';

SET @color = 'Pink';

SET @quantity = 2;

CALL PlaceOrder();
```

Output:



```
+---------------------------------------------
------+
| OrderConfirmation
      |
+---------------------------------------------
------+
| User: Vamsi
Model: iPhone 15 Plus
Color: Pink
Price: 1499.99
Quantity: 2
Total: 1499.99
Status: Pending (awaiting shipment)
Payment: Credit Card, Successful
Order Date: April 18, 2025, 11:42:39 |
+---------------------------------------------
```

–User Spending and Purchase Insights Query

```
SELECT
    u.user_id, u.name AS user_name,u.email,u.phone_number,
    COUNT(DISTINCT o.order_id) AS total_orders,
    SUM(od.quantity * od.price) AS total_spent,
    ROUND(AVG(od.quantity * od.price), 2) AS avg_order_value,
    DATE_FORMAT(MAX(o.order_date), '%Y-%m-%d') AS last_order_date,
    -- Subquery 1: Most purchased mobile by the user
    (SELECT CONCAT(m.brand, ' ', m.model)
        FROM OrderDetails od2
        JOIN Orders o2 ON od2.order_id = o2.order_id
        JOIN Mobiles m ON od2.mobile_id = m.mobile_id
        WHERE o2.user_id = u.user_id
        GROUP BY od2.mobile_id
        ORDER BY SUM(od2.quantity) DESC
        LIMIT 1
    ) AS most_purchased_mobile,
    -- Subquery 2: Latest review rating by the user
    (SELECT r.rating
        FROM Reviews r
        WHERE r.user_name = u.name
        ORDER BY r.review_date DESC
        LIMIT 1
    ) AS latest_review_rating
FROM Users u
```

```
JOIN Orders o ON u.user_id = o.user_id

JOIN OrderDetails od ON o.order_id = od.order_id


GROUP BY u.user_id, u.name, u.email, u.phone_number

ORDER BY total_spent DESC

LIMIT 5;
```

Output:

user_id	user_name	email	phone_number	total_orders	total_spent	avg_order_value	last_order_date	most_purchased_mobile	latest_review_rating
7	Vamsi	vamsi@example.com	9876543216	2	5999.96	2999.98	2025-04-18	Apple iPhone 15 Plus	5
3	Bhanu	bhanu@example.com	9876543212	1	2399.97	2399.97	2025-04-05	Apple iPhone 5	3
5	Nishanth	nishanth@example.com	9876543214	1	1799.98	1799.98	2025-04-06	Apple iPhone 6	4
1	Tasneem	tasneem@example.com	9876543210	1	799.98	799.98	2025-04-03	Apple iPhone	4
2	Prem	prem@example.com	9876543211	1	699.99	699.99	2025-04-04	Apple iPhone 4	5

➢ Subqueries

Subquery 1 :most_purchased_mobile

```
SELECT CONCAT(m.brand, ' ', m.model)

FROM OrderDetails od

JOIN Orders o ON od.order_id = o.order_id
JOIN Mobiles m ON od.mobile_id = m.mobile_id

WHERE o.user_id = <user_id_here>

GROUP BY od.mobile_id

ORDER BY SUM(od.quantity) DESC

LIMIT 1;
```

Subquery 2: latest_review_rating

```
SELECT r.rating

FROM Reviews r

WHERE r.user_name = '<user_name_here>'

ORDER BY r.review_date DESC

LIMIT 1;
```

➢ Aggregate Functions

Expression	Description
COUNT(DISTINCT o.order_id) AS total_orders,	-- Total number of orders
SUM(od.quantity * od.price) AS total_spent,	-- Total amount spent
AVG(od.quantity * od.price) AS avg_order_value,	-- Average order value
MAX(o.order_date) AS last_order_date	-- Most recent order

➢ Joins

➢Join with Orders

```
JOIN Orders o ON u.user_id = o.user_id
```

➢Join with OrderDetails

```
JOIN OrderDetails od ON o.order_id = od.order_id
```

➢Join with Mobiles

```
JOIN Mobiles m ON od.mobile_id = m.mobile_id
```

➢GROUP BY Clause

```
GROUP BY

   u.user_id,

   u.name,

   u.email,

   u.phone_number
```

➢ORDER BY Clause

```
ORDER BY total_spent DESC

LIMIT 5;
```

➢Customer Order and Review Summary Query

```
SELECT

   u.user_id,

   u.name AS customer_name,

   COUNT(DISTINCT o.order_id) AS total_orders,

   COUNT(DISTINCT od.order_detail_id) AS total_items,

   MAX(o.order_date) AS last_order_date,

   (

      SELECT SUM(od2.quantity * od2.price)

      FROM Orders o2

      JOIN OrderDetails od2 ON o2.order_id = od2.order_id

      WHERE o2.user_id = u.user_id

   ) AS total_spent,

   (

      SELECT COUNT(*)

      FROM Reviews r

      WHERE r.user_name = u.name

   ) AS total_reviews,

   (

      SELECT AVG(r.rating)

      FROM Reviews r

      WHERE r.user_name = u.name
```

```
    ) AS avg_rating

FROM Users u

LEFT JOIN Orders o ON u.user_id = o.user_id

LEFT JOIN OrderDetails od ON o.order_id = od.order_id

GROUP BY u.user_id, u.name

ORDER BY total_spent DESC;
```

Output:

user_id	customer_name	total_orders	total_items	last_order_date	total_spent	total_reviews	avg_rating
7	Vamsi	2	2	2025-04-18 11:42:39	5999.96	1	5.0000
3	Bhanu	1	1	2025-04-05 09:15:00	2399.97	1	3.0000
5	Nishanth	1	1	2025-04-06 14:45:00	1799.98	1	4.0000
1	Tasneem	1	1	2025-04-03 12:00:00	799.98	1	4.0000
2	Prem	1	1	2025-04-04 15:30:00	699.99	1	5.0000
4	Karthikeya	1	1	2025-04-06 10:00:00	699.99	0	NULL
6	Lokesh	0	0	NULL	NULL	0	NULL
8	Tharun	0	0	NULL	NULL	0	NULL

Subquery Query

Purpose: Find users who have placed orders for mobiles that have received at least one review with a rating of 5.

```
SELECT
   u.user_id,u.name,u.email
FROM
   Users u
WHERE
   u.user_id IN (
      SELECT o.user_id
      FROM Orders o
      INNER JOIN OrderDetails od ON o.order_id = od.order_id
      WHERE od.mobile_id IN (
         SELECT r.mobile_id
         FROM Reviews r
         WHERE r.rating = 5
      )
   )
ORDER BY
   u.name;
```

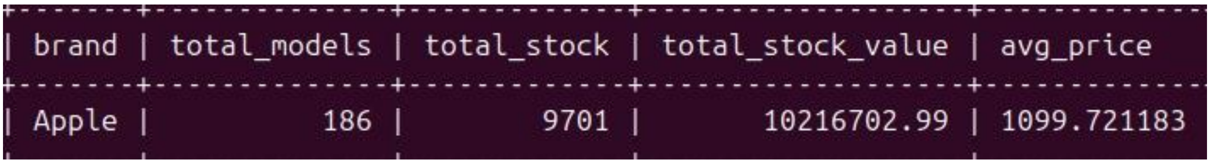
Aggregate Functions Query

Purpose: Calculate the total stock value and average price for each mobile brand.

```
SELECT
   m.brand,COUNT(m.mobile_id) AS total_models,SUM(m.stock_quantity) AS total_stock,SUM(m.stock_quantity * m.price) AS total_stock_value,AVG(m.price) AS avg_price
```

```
FROM
   Mobiles m
GROUP BY
   m.brand
HAVING
   total_stock > 0
ORDER BY
   total_stock_value DESC;
```

Output:



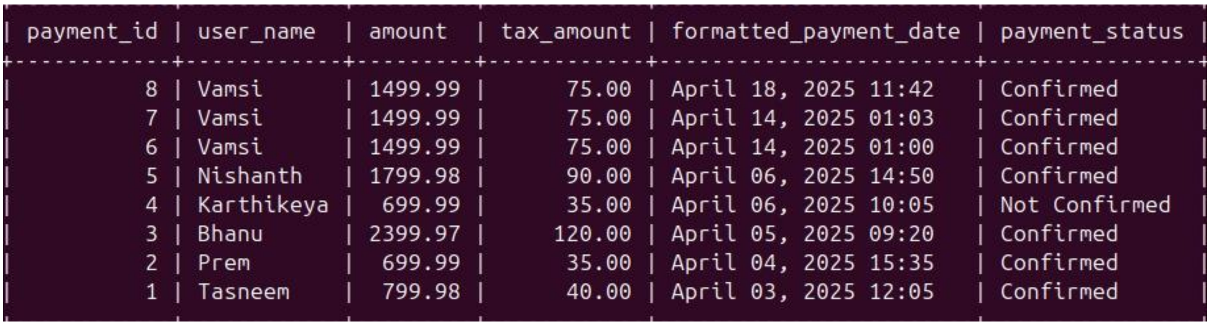
```
+-------+--------------+-------------+-------------------+------------
| brand | total_models | total_stock | total_stock_value | avg_price
+-------+--------------+-------------+-------------------+------------
| Apple |          186 |        9701 |       10216702.99 | 1099.721183
```

3. Functions Query

Purpose: Display payment details with formatted dates, calculated taxes, and status labels.

```
SELECT
   p.payment_id, p.user_name,p.amount,
   ROUND(p.amount * 0.05, 2) AS tax_amount,
   DATE_FORMAT(p.payment_date, '%M %d, %Y %H:%i') AS
formatted_payment_date,
   IF(p.status = 'Successful', 'Confirmed', 'Not Confirmed') AS payment_status
FROM
   Payments p
ORDER BY
   p.payment_date DESC;
```

Output:



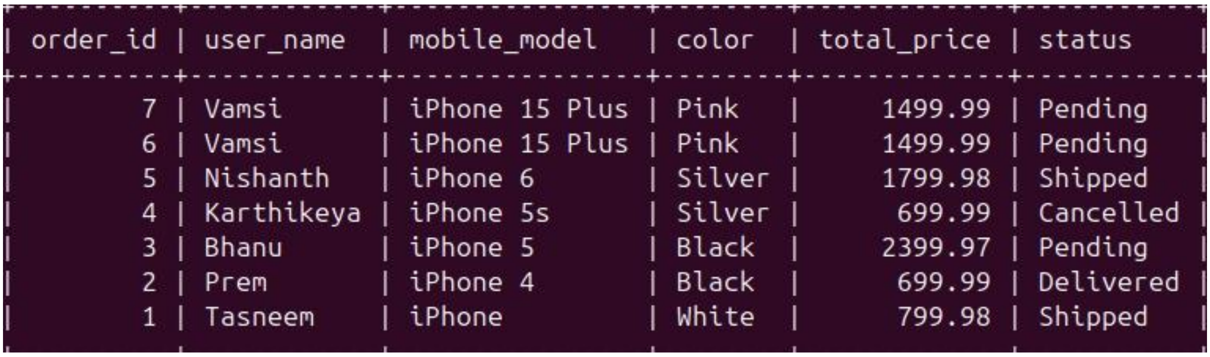
```
| payment_id | user_name  | amount  | tax_amount | formatted_payment_date | payment_status |
+------------+------------+---------+------------+------------------------+----------------+
|          8 | Vamsi      | 1499.99 |      75.00 | April 18, 2025 11:42   | Confirmed      |
|          7 | Vamsi      | 1499.99 |      75.00 | April 14, 2025 01:03   | Confirmed      |
|          6 | Vamsi      | 1499.99 |      75.00 | April 14, 2025 01:00   | Confirmed      |
|          5 | Nishanth   | 1799.98 |      90.00 | April 06, 2025 14:50   | Confirmed      |
|          4 | Karthikeya |  699.99 |      35.00 | April 06, 2025 10:05   | Not Confirmed  |
|          3 | Bhanu      | 2399.97 |     120.00 | April 05, 2025 09:20   | Confirmed      |
|          2 | Prem       |  699.99 |      35.00 | April 04, 2025 15:35   | Confirmed      |
|          1 | Tasneem    |  799.98 |      40.00 | April 03, 2025 12:05   | Confirmed      |
```

4. Joins Query

Purpose: List all orders with user names and mobile models, including order status.

```
SELECT
    o.order_id,u.name AS user_name,m.model AS mobile_model,m.color,o.total_price,o.status
FROM
    Orders o
INNER JOIN
    Users u ON o.user_id = u.user_id
INNER JOIN
    OrderDetails od ON o.order_id = od.order_id
INNER JOIN
    Mobiles m ON od.mobile_id = m.mobile_id
ORDER BY
    o.order_date DESC;
```

Output:



```
+----------+------------+----------------+--------+-------------+-----------+
| order_id | user_name  | mobile_model   | color  | total_price | status    |
+----------+------------+----------------+--------+-------------+-----------+
|        7 | Vamsi      | iPhone 15 Plus | Pink   |     1499.99 | Pending   |
|        6 | Vamsi      | iPhone 15 Plus | Pink   |     1499.99 | Pending   |
|        5 | Nishanth   | iPhone 6       | Silver |     1799.98 | Shipped   |
|        4 | Karthikeya | iPhone 5s      | Silver |      699.99 | Cancelled |
|        3 | Bhanu      | iPhone 5       | Black  |     2399.97 | Pending   |
|        2 | Prem       | iPhone 4       | Black  |      699.99 | Delivered |
|        1 | Tasneem    | iPhone         | White  |      799.98 | Shipped   |
```

11. Creation of Views Using the Tables:

```
CREATE VIEW RecentStockPurchaseInsights AS
SELECT
    u.user_id,u.name AS user_name,u.email,
    COUNT(DISTINCT o.order_id) AS purchase_count,SUM(od.quantity * od.price) AS total_purchase_value,
    COUNT(DISTINCT CASE
        WHEN r.review_date >= DATE_SUB(CURDATE(), INTERVAL 30 DAY)
        THEN r.review_id
        ELSE NULL
    END) AS recent_review_count,
    CONCAT('User: ', u.name, ' purchased $', FORMAT(SUM(od.quantity * od.price), 2), ' of high-stock mobiles') AS purchase_summary
FROM
    Users u
INNER JOIN
    Orders o ON u.user_id = o.user_id
INNER JOIN
    OrderDetails od ON o.order_id = od.order_id
INNER JOIN
    Mobiles m ON od.mobile_id = m.mobile_id
LEFT JOIN
    Reviews r ON m.mobile_id = r.mobile_id
WHERE
    m.mobile_id IN (
    SELECT m2.mobile_id
    FROM Mobiles m2
    WHERE m2.stock_quantity > (
      SELECT AVG(m3.stock_quantity)
      FROM Mobiles m3
    )
  )
GROUP BY
  u.user_id, u.name, u.email
HAVING
  purchase_count > 0
ORDER BY
  total_purchase_value DESC;
```

Output:

user_id	user_name	email	purchase_count	total_purchase_value	recent_review_count	purchase_summary
5	**Nishanth**	**nishanth@example.com**	**1**	**1799.98**	**1**	**User: Nishanth purchased $1,799.98 of high-stock mobiles**

Functionality:

- Displays users who purchased mobiles with above-average stock levels, along with their total purchase value and order count.

- Calculates the number of recent reviews (last 30 days) given by each user for the purchased mobiles.

- Provides a summary message showing the user's name and total purchase value for quick insights.

12.Conclusion:

The *Online Mobile Shopping Database Management System* has been meticulously designed to demonstrate the practical applications and benefits of a DBMS in organizing, retrieving, and managing e-commerce-related data efficiently. Through this project, we have highlighted the importance of structured data storage and normalization to minimize redundancy and ensure data consistency.

The database schema effectively models real-world entities such as users, orders, mobiles, reviews, and stock levels—offering a complete view of an online mobile retail platform. Sample data was inserted to test the system's performance, and a variety of SQL queries were executed to showcase functionality like tracking user purchases, managing inventory, and analyzing customer engagement.

Additionally, a specialized SQL view was created to generate insightful summaries regarding recent purchases of high-stock mobiles, offering analytical value for business decision-making. Plans to visualize the system using an ER diagram further support the structural understanding of entity relationships and data flow.

Overall, this project embodies a solid foundation in database design and implementation, merging academic knowledge with real-world e-commerce scenarios. It not only reinforces key DBMS concepts but also contributes to building scalable and reliable systems for modern online retail platforms.