**S.D.M.E Society’s**

# SDM COLLEGE OF ENGINEERING AND TECHNOLOGY DHAVALAGIRI, DHARWAD 580002



**DEPARTMENT OF INFORMATION SCIENCE ENGINEERINGDBMS PROJECT ON**

**GROCERY STORE MANAGEMENT SYSTEM**

**Under the Guidance Prof Anita Dixit Submitted By,**

**ANANYA R UPADHYA USN: 2SD21IS006 CHINMAY RAJESH KULKARNI USN: 2SD21IS012 DARSHAN KULKARNI USN: 2SD21IS014 FAIYAZAHAMAD S MULLANAVAR USN: 2SD21IS015 GAGAN U BIRADAR USN: 2SD21IS016**

5th semester Academic Year 2023-24

**Table of Contents**

|  |  |  |
| --- | --- | --- |
| **Sl.no** | **Title** | **Page no** |
| 1 | Abstract | 1 |
| 2 | Introduction | 2 |
| 3 | Data requirements and functional requirements | 3-4 |
| 4 | ER diagram and schema diagram | 5-6 |
| 5 | Functional dependencies and normalization | 7-10 |
| 6 | Implementation Phase | 11-13 |
| 7 | Conclusion | 14 |
| 8 | Appendix A: code | 15-26 |
| 9 | Appendix B: screenshots | 27-29 |

**Abstract:**

The Grocery Store Management System is a comprehensive solution designed to streamline and automate the operational processes of a grocery store. This Database Management System (DBMS) project aims to enhance the efficiency and accuracy of managing inventory, sales, and customer information. It includes modules for adding, updating, and deleting product information, along with features for categorizing items to improve organizational structure.

Security is a priority, with implemented measures like user authentication and access controls to safeguard sensitive data. The system is designed with scalability in mind, ensuring it can adapt to the changing needs of the grocery store.

Overall, the Grocery Store Management System offers a user-friendly interface coupled with robust functionality. It serves as a valuable tool for store owners and managers, optimizing their daily tasks and contributing significantly to the growth and success of the grocery business

# CHAPTER 1 : INTRODUCTION

The retail landscape, especially in the grocery sector, has witnessed a significant transformation with the advent of technology. In this context, the Grocery Store Management System serves as a pivotal solution aimed at revolutionizing the traditional operational framework of grocery stores. This Database Management System (DBMS) project is meticulously crafted to address the complexities and challenges faced by store owners and managers in efficiently managing their day-to-day activities.

Efficient point-of-sale functionality is a cornerstone of the project, aiming to expedite customer transactions, ensure accuracy, and facilitate seamless payment processes. Electronic receipt generation and centralized sales data storage further contribute to informed decision-making and strategic planning.

The Grocery Store Management System also acknowledges the importance of customer relationships in a competitive market. The integrated Customer Relationship Management (CRM) module captures valuable data on customer preferences and purchase history, enabling personalized marketing strategies and fostering customer loyalty.

# Necessity of Project

This project aims to enhance the administrative activities of grocery stores by providing a structured platform to store and retrieve crucial data. It caters to the storage of employee details, including personal and professional information, as well as comprehensive records of the store's inventory. Additionally, the system facilitates the seamless recording of customer transactions, ensuring accurate and organized data management.

# Advantages

The advantages of this Grocery Store Management System are multifold. It offers a centralized database for employee and inventory details, promoting efficient management and retrieval of information. Furthermore, the system enables grocery stores to maintain a detailed record of customer transactions, fostering better customer service and inventory control. Overall, the Grocery Store Management System acts as an indispensable tool for grocery stores, optimizing their operations and contributing to a more efficient and customer-centric shopping experience.

# CHAPTER 2 :DATA REQUIREMENTS AND FUNCTIONAL REQUIREMENTS

## Functional and Performance requirement

* Data in database should be updated within seconds.
* Load time of UI should not take more time.
* The upload time should not take more than a minute.
* Login Validation should be done within less time.
* Response to customer inquiry must be done within minutes.
* Response time should not be more.
* Products should be displayed according to customer’s search

## Database Requirements

Php My admin is used for storing the credentials of the user and the trip details, employee details and customer details are stored in it.

## Software Quality Attributes

* Correctness:

The System should give within time as per customer comfortability.

* Flexibility:

System should be flexible enough to provide space to add new features and to handle them conveniently.

* Integrity:

System should focus on securing the customer information and avoid data losses as much as possible and maintain its confidentiality.

## Hardware requirements

Laptop or Pc with following specifications:

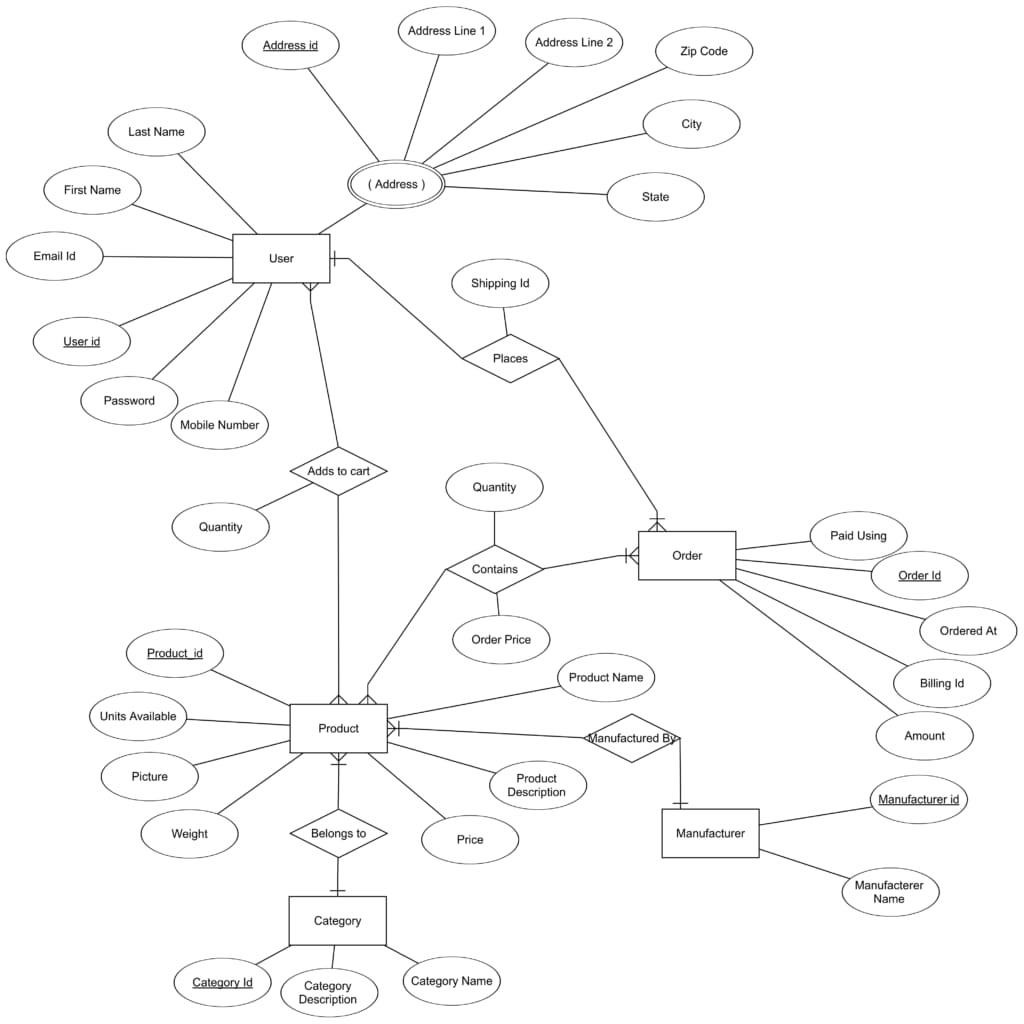
* OS - Windows 64 bit
* Processor: Intel core i5 or above is preferred
* RAM: Minimum 8GB
* Graphic Card: Any graphic card can be used

## Product Functions

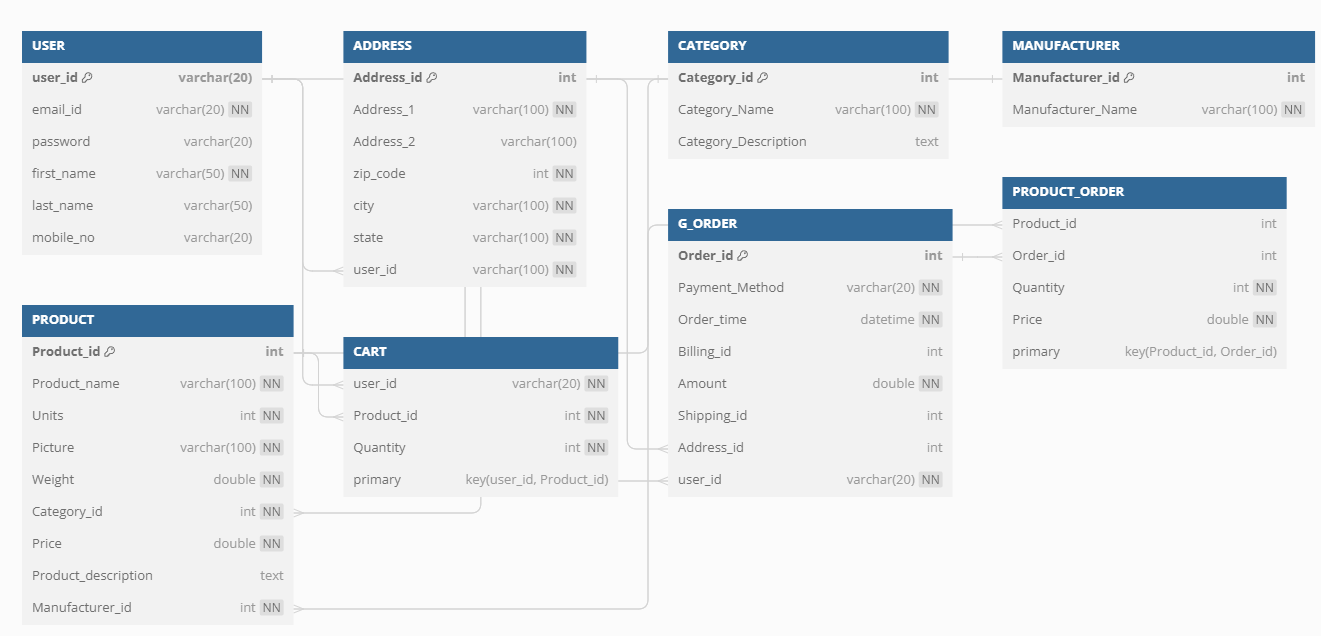
* User login
* Product’s details
* Customer’s details
* Order details
* Confirm order/s

# CHAPTER 3 : ER-DIAGRAM AND SCHEMA DIAGRAM

* 1. **ER-DIAGRAM**



# SCHEMA DIAGRAM



**CHAPTER 4:**

# FUNCTIONAL DEPENDENCIES AND NORMALIZATION

* 1. **Normalization**

Normalization is the process of decomposing unsatisfactory bad relations by breaking up their attributes into smaller relations.

* + 1. First Normal Form (1NF)

The relations are said to be in 1NF if there are no multivalued attributes or nested relations.

* + 1. Second Normal Form (2NF)

The relations are said to be in 2NF if it is in 1NF and no non-key attributes are functionally dependent on part of the primary key.

* + 1. Third Normal Form (3NF)

# Functional dependencies

User Table:

Functional Dependencies:

User\_Id → Email\_Id, Password, First\_Name, Last\_Name, Mobile\_no Email\_Id → User\_Id (if Email\_Id is unique per user)

Mobile\_no → User\_Id (if Mobile\_no is unique per user)

Address Table:

Functional Dependencies:

Address\_id → Address\_1, Address\_2, Zip\_Code, City, State, User\_Id User\_Id → Address\_id (if each user has only one address)

Product Table:

Functional Dependencies:

Product\_id → Product\_Name, Units, Picture, Weight, Category\_id, Price, Product\_Description,

Manufacturer\_id

Category\_id → Product\_id (if a product belongs to only one category)

Manufacturer\_id → Product\_id (if a product is manufactured by only one manufacturer)

Category Table:

Functional Dependencies:

Category\_id → Category\_description, Category\_Name

Manufacturer Table:

Functional Dependencies: Manufacturer\_id → Manufacturer\_Name

G\_Order Table:

Functional Dependencies:

Order\_id → Payment\_Method, Order\_time, Billing\_id, Amount, Shipping\_id, Address\_id, User\_id

Cart (User\_Product) Table:

Functional Dependencies:

(User\_id, Product\_id) → Quantity

User\_id → (Product\_id, Quantity) (if each user has only one entry per product)

Product\_Order Table:

Functional Dependencies:

(Product\_id, Order\_id) → Quantity, Price

Product\_id → (Order\_id, Quantity, Price) (if each product in an order has a unique entry)

In each functional dependency, the left side represents the determinant (the attribute or set of attributes that uniquely determines the right side).

For example, in "User\_Id → Email\_Id, Password, First\_Name, Last\_Name, Mobile\_no," User\_Id is the determinant, and it uniquely determines the other attributes in the user table.

Functional dependencies are used to ensure that the database design adheres to normalization principles, minimizing data redundancy and maintaining data integrity. They are crucial for defining relationships between attributes in a table.

# Relationships:

## User Table:

One-to-many between User and Address:

A user can have multiple addresses, but each address is associated with only one user. One-to-many between User and Cart:

A user can have multiple entries in the cart, but each entry in the cart is associated with only one user.

Many-to-many between User and Product (through Cart):

A user can have multiple products in the cart, and a product can be in the carts of multiple users.

## Category Table:

One-to-many between Category and Product:

A category can have multiple products, but each product belongs to only one category.

## Manufacturer Table:

One-to-many between Manufacturer and Product:

A manufacturer can produce multiple products, but each product is manufactured by only one manufacturer.

## G\_Order Table:

One-to-many between User and G\_Order:

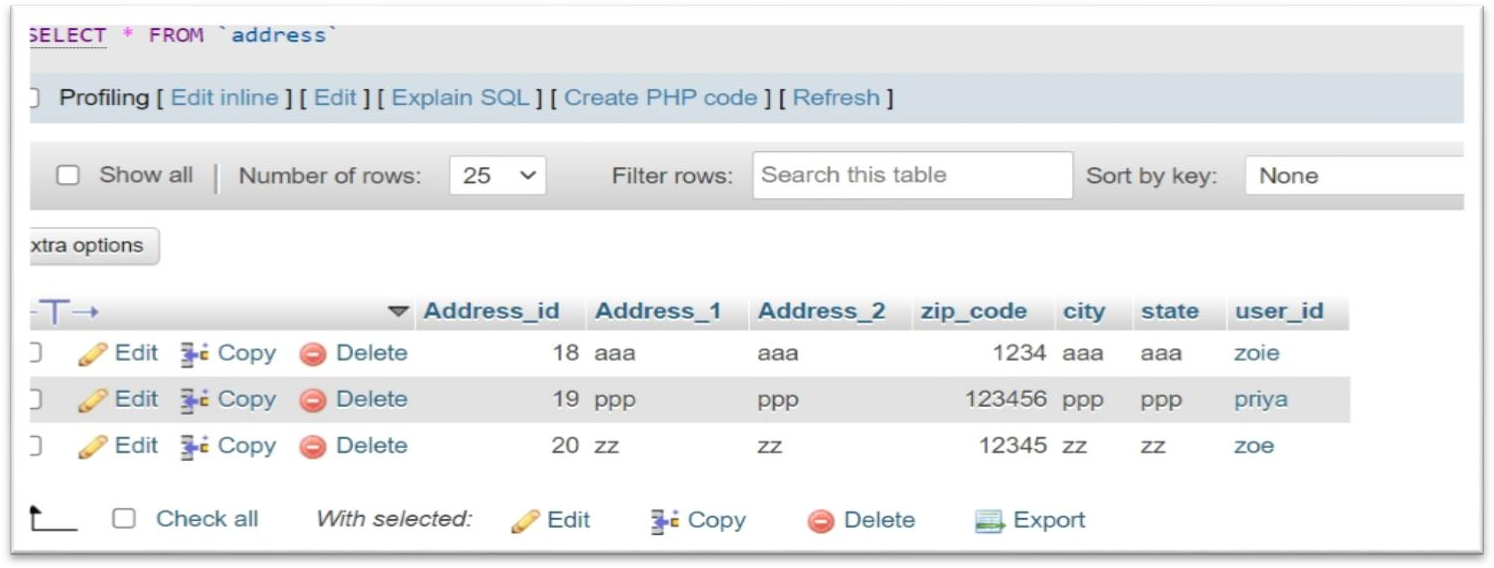
A user can place multiple orders, but each order is associated with only one user. One-to-many between Address and G\_Order (for billing and shipping):

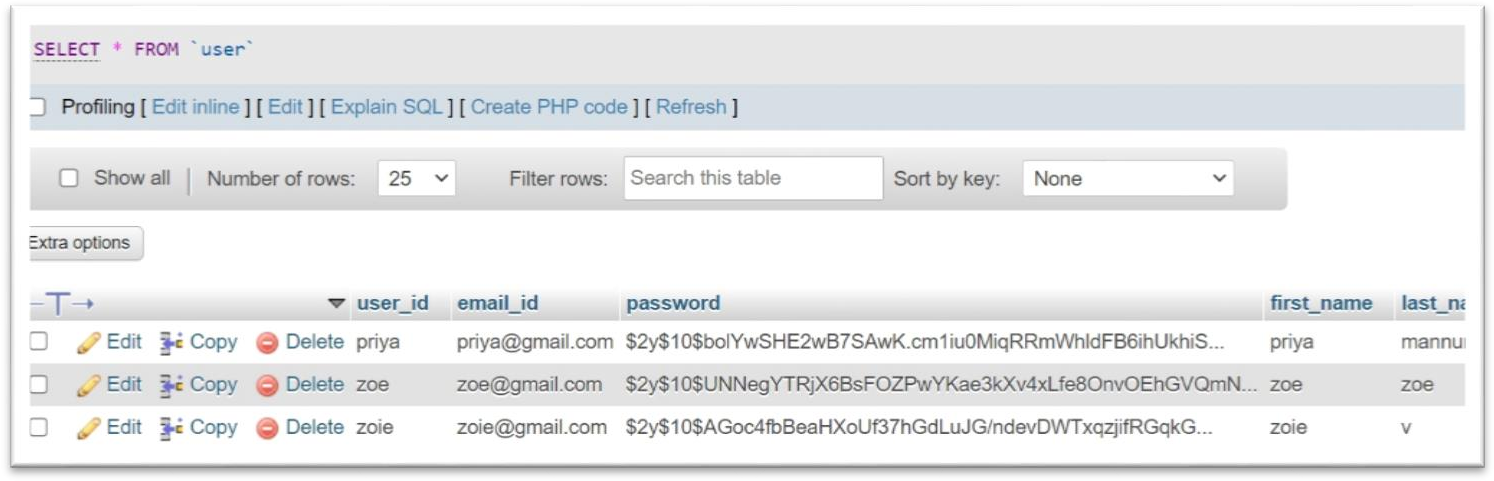
An address can be associated with multiple orders for billing and shipping, but each order is associated with only one address for billing and one for shipping.

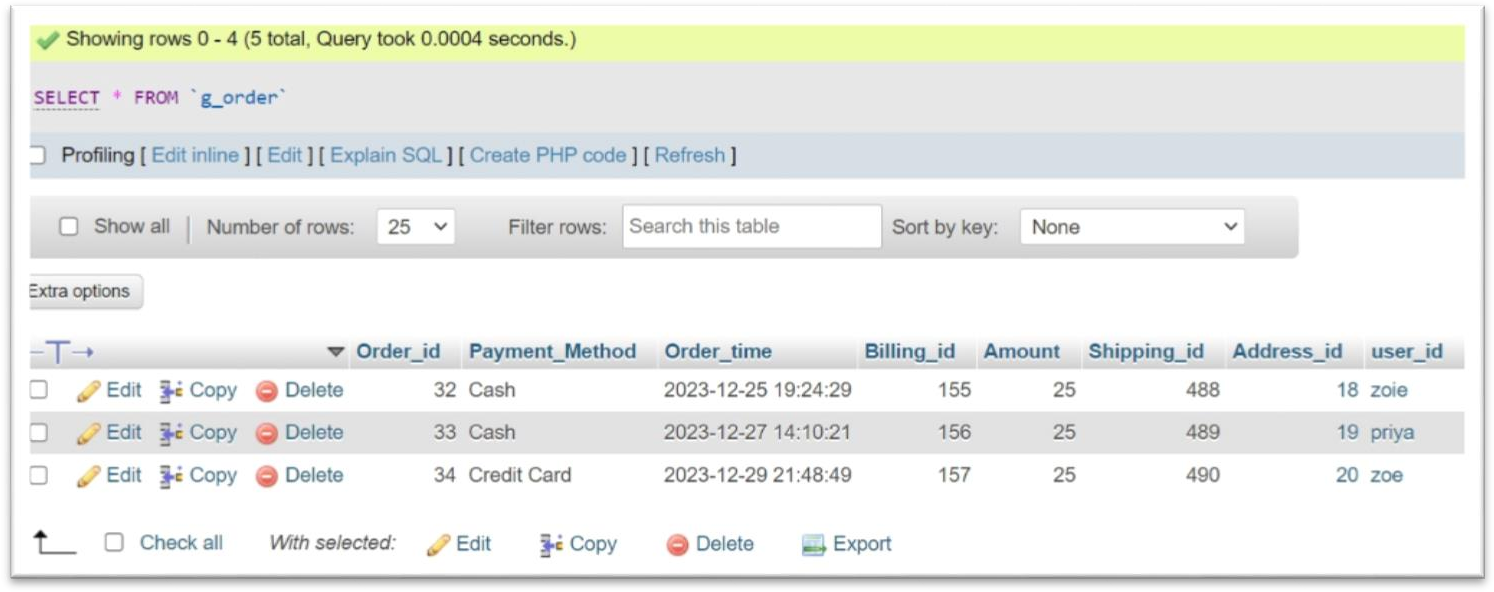
Many-to-many between Product and G\_Order (through Product\_Order):

A product can be in multiple orders, and an order can have multiple products.

# CHAPTER 5: IMPLEMENTATION

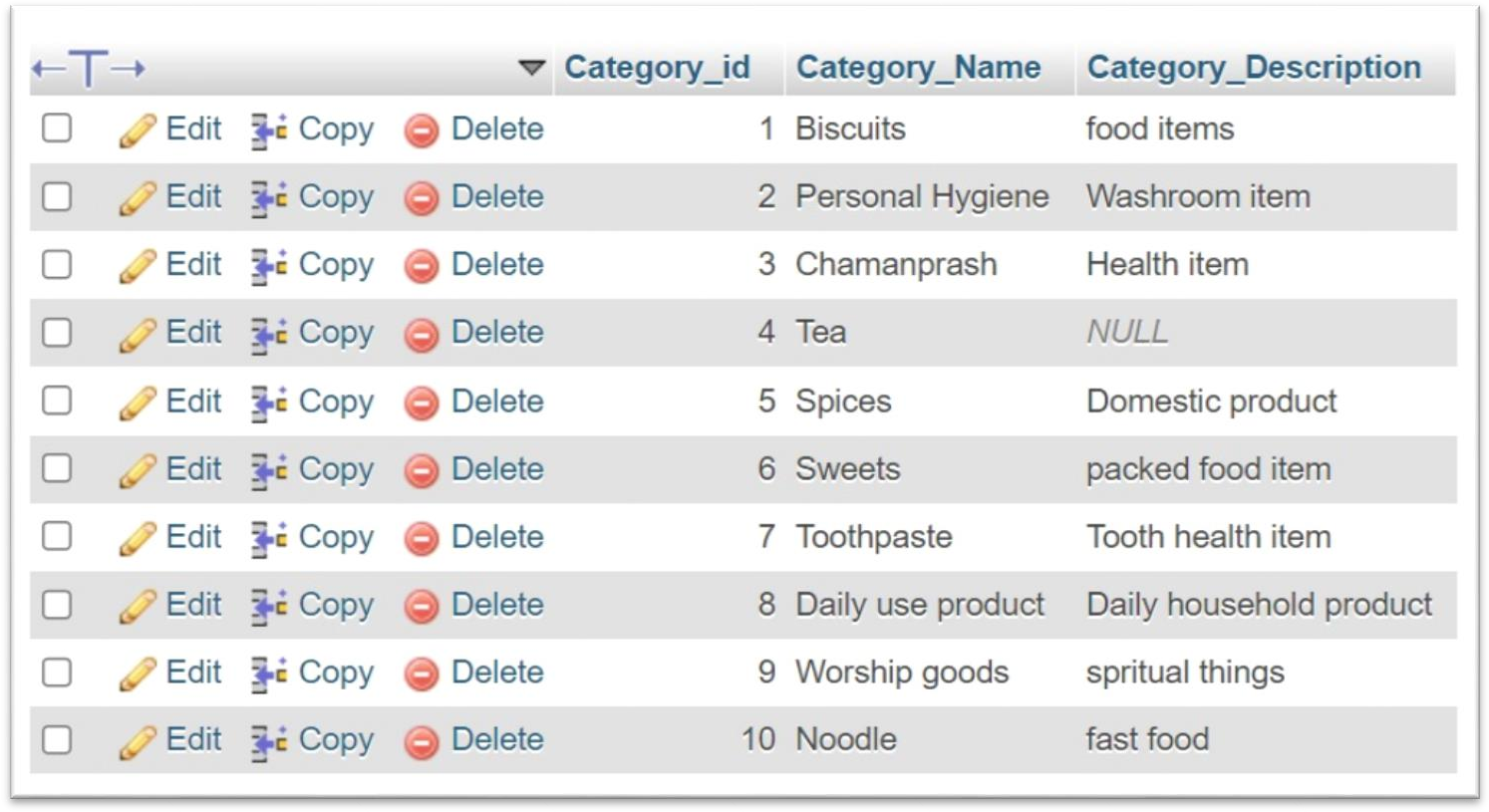


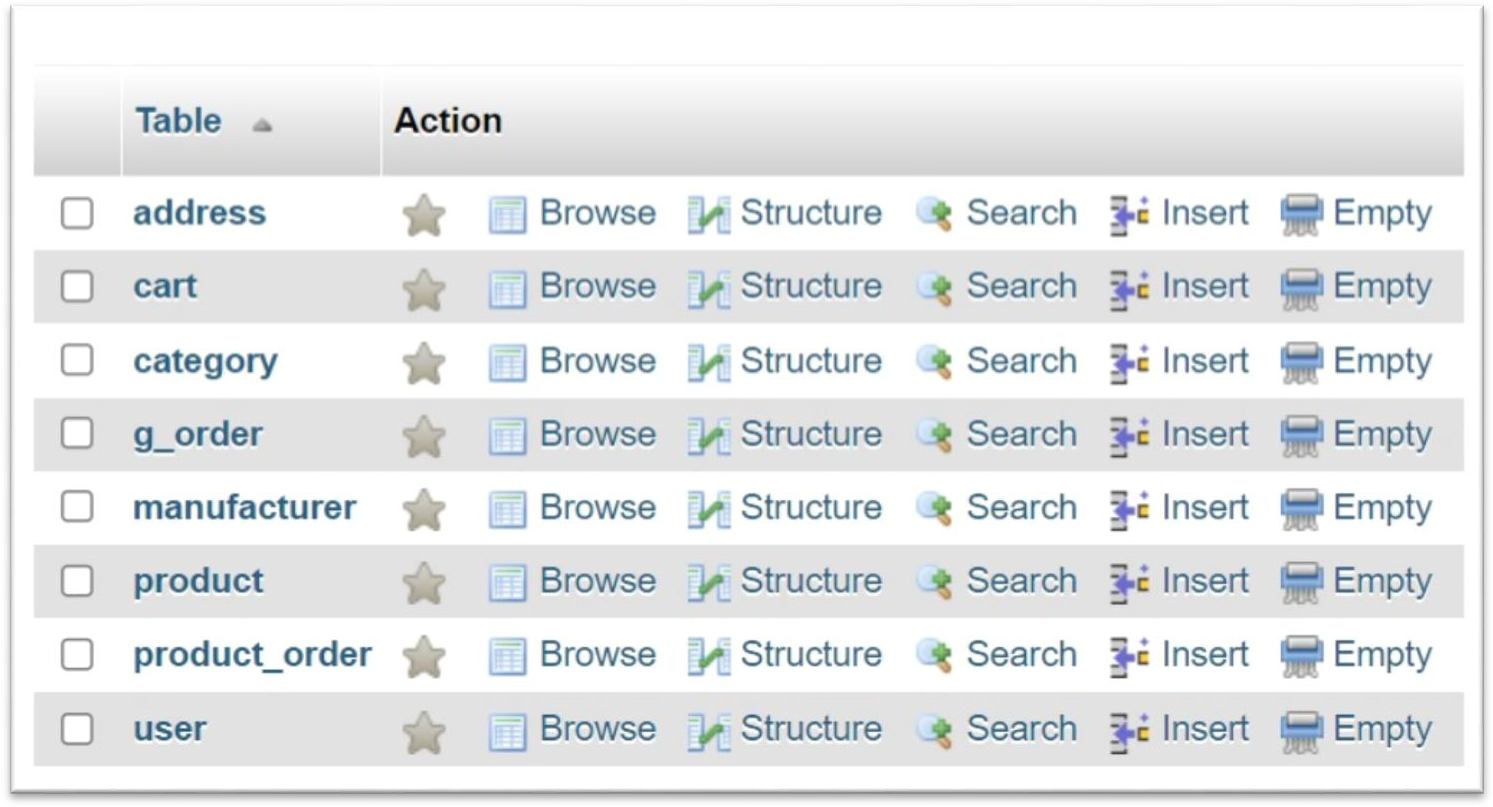












**CHAPTER 6: CONCLUSION**

In conclusion, the Online Grocery Store Database Management System (DBMS) project successfully integrates MySQL, PHP, HTML, CSS, and JavaScript to create a user-friendly platform for customers to seamlessly book their groceries. The project efficiently manages customer details, employee information, vehicle data, and payment transactions, allowing for easy retrieval, insertion, deletion, and modification of data.

This project not only showcases the practical application of database management and web development skills but also underscores the importance of a well-integrated system in providing a smooth user experience. By navigating through the complexities of MySQL and web technologies, the project lays a foundation for continued exploration and growth in the dynamic fields of database management and online retail solutions.

**Appendix A: Code**

#create database

CREATE DATABASE gstore;

#user creation and permission setting

DROP USER IF EXISTS 'gsuser'@'localhost';

CREATE USER 'gsuser'@'localhost' IDENTIFIED BY 'gspass'; GRANT ALL ON gstore.\* TO 'gsuser'@'localhost';

FLUSH PRIVILEGES;

#create the tables CREATE TABLE USER (

user\_id varchar(20) PRIMARY KEY, email\_id varchar(20) UNIQUE NOT NULL, password varchar(20),

first\_name varchar(50) NOT NULL, last\_name varchar(50),

mobile\_no varchar(20)

);

CREATE TABLE ADDRESS(

Address\_id INT PRIMARY KEY AUTO\_INCREMENT,

Address\_1 varchar(100) NOT NULL, Address\_2 varchar(100),

zip\_code INT NOT NULL, city varchar(100) NOT NULL, state varchar(100) NOT NULL,

user\_id varchar(100) NOT NULL,

FOREIGN KEY (User\_id) REFERENCES USER(User\_id),

UNIQUE( Address\_1, Address\_2, zip\_code, city, state, user\_id)

);

CREATE TABLE CATEGORY(

Category\_id INT PRIMARY KEY AUTO\_INCREMENT,

Category\_Name varchar(100) NOT NULL UNIQUE, Category\_Description text default NULL

);

CREATE TABLE MANUFACTURER(

Manufacturer\_id INT PRIMARY KEY AUTO\_INCREMENT, Manufacturer\_Name varchar(100) NOT NULL UNIQUE

);

CREATE TABLE PRODUCT(

Product\_id INT PRIMARY KEY AUTO\_INCREMENT,

Product\_name varchar(100) NOT NULL, Units INT DEFAULT 0 NOT NULL,

Picture varchar(100) DEFAULT "No\_image\_available.svg" NOT NULL, Weight DOUBLE NOT NULL,

Category\_id INT NOT NULL, Price DOUBLE NOT NULL,

Product\_description text, Manufacturer\_id INT NOT NULL,

FOREIGN KEY (Category\_id) REFERENCES CATEGORY(Category\_id),

FOREIGN KEY (Manufacturer\_id) REFERENCES MANUFACTURER(Manufacturer\_id)

);

CREATE TABLE CART(

user\_id varchar(20) NOT NULL, Product\_id INT NOT NULL,

QUANTITY INT DEFAULT 1 NOT NULL,

FOREIGN KEY (user\_id) REFERENCES USER(user\_id),

FOREIGN KEY (Product\_id) REFERENCES PRODUCT(Product\_id), UNIQUE( user\_id, Product\_id)

);

CREATE TABLE G\_ORDER(

Order\_id INT PRIMARY KEY AUTO\_INCREMENT,

Payment\_Method ENUM('Cash','Net Banking','Credit Card','Debit Card') DEFAULT

'Cash' NOT NULL,

Order\_time DATETIME NOT NULL DEFAULT CURRENT\_TIMESTAMP,

Billing\_id INT UNIQUE,

Amount DOUBLE NOT NULL DEFAULT 0,

Shipping\_id INT UNIQUE, Address\_id INT,

user\_id varchar(20) NOT NULL,

FOREIGN KEY (Address\_id) REFERENCES ADDRESS(Address\_id), FOREIGN KEY (user\_id) REFERENCES USER(user\_id)

);

CREATE TABLE PRODUCT\_ORDER(

Product\_id INT, Order\_id INT,

Quantity INT DEFAULT 0, price DOUBLE DEFAULT 0,

FOREIGN KEY (Product\_id) REFERENCES PRODUCT(Product\_id), FOREIGN KEY (Order\_id) REFERENCES G\_ORDER(Order\_id), UNIQUE (Product\_id,Order\_id)

);

#triggers

#to check that product quantity never goes below zero

DELIMITER //

CREATE TRIGGER pqty\_ins BEFORE INSERT ON PRODUCT FOR EACH ROW

BEGIN

IF NEW.Units < 0 THEN SIGNAL SQLSTATE '12345'

SET MESSAGE\_TEXT = 'No of available products cannot be less than 0'; END IF;

END;//

CREATE TRIGGER pqty\_upd BEFORE UPDATE ON PRODUCT FOR EACH ROW

BEGIN

IF NEW.Units < 0 THEN SIGNAL SQLSTATE '12345'

SET MESSAGE\_TEXT = 'No of available products cannot be less than 0'; END IF;

END;// DELIMITER ;

#verify email id is in correct format DELIMITER //

CREATE TRIGGER email\_update\_check BEFORE UPDATE ON USER FOR EACH ROW

BEGIN

IF NOT(SELECT NEW.Email\_id REGEXP '^[^@]+@[^@]+\.[^@]{2,}$') THEN SIGNAL SQLSTATE '40001'

SET MESSAGE\_TEXT = "Invalid Email Id!"; END IF;

END;

//

CREATE TRIGGER email\_insert\_check BEFORE INSERT ON USER FOR EACH ROW

BEGIN

IF NOT(SELECT NEW.Email\_id REGEXP '^[^@]+@[^@]+\.[^@]{2,}$') THEN SIGNAL SQLSTATE '40001'

SET MESSAGE\_TEXT = "Invalid Email Id!"; END IF;

END;

// DELIMITER ;

#verify pin code uniquely determines city and state DELIMITER //

CREATE TRIGGER verify\_zip\_code\_insert BEFORE INSERT ON ADDRESS FOR EACH ROW

BEGIN

DECLARE vcity varchar(100) DEFAULT NULL; DECLARE vstate varchar(100) DEFAULT NULL;

SELECT city, state INTO vcity, vstate FROM ADDRESS WHERE zip\_code = NEW.zip\_code LIMIT 1;

IF vcity IS NOT NULL AND vstate IS NOT NULL THEN

IF NOT (vcity = NEW.city AND vstate = NEW.state) THEN SIGNAL SQLSTATE '13232'

SET MESSAGE\_TEXT = "Pin codes do not match to city/state"; END IF;

END IF;

END;//

CREATE TRIGGER verify\_zip\_code\_update BEFORE UPDATE ON ADDRESS FOR EACH ROW

BEGIN

DECLARE vcity varchar(100) DEFAULT NULL; DECLARE vstate varchar(100) DEFAULT NULL;

SELECT city, state INTO vcity, vstate FROM ADDRESS WHERE zip\_code = NEW.zip\_code AND (NOT address\_id = NEW.address\_id) LIMIT 1;

IF vcity IS NOT NULL AND vstate IS NOT NULL THEN

IF NOT (vcity = NEW.city AND vstate = NEW.state) THEN SIGNAL SQLSTATE '13232'

SET MESSAGE\_TEXT = "Pin codes do not match to city/state"; END IF;

END IF;

END;// DELIMITER ;

#generate billing and shipping id for order DELIMITER //

CREATE TRIGGER genbillshipid BEFORE INSERT ON G\_ORDER FOR EACH ROW

BEGIN

DECLARE OID INT DEFAULT 0;

SELECT auto\_increment INTO OID FROM information\_schema.TABLES WHERE table\_name = 'G\_ORDER' AND table\_schema = 'gstore';

IF(NEW.Billing\_Id IS NULL) THEN

SET NEW.Billing\_id = OID + 123; END IF;

IF(NEW.Shipping\_Id IS NULL) THEN SET NEW.Shipping\_id = OID + 456;

END IF;

END;// DELIMITER ;

#Check that Quantity available in cart is valid DELIMITER //

CREATE TRIGGER rem\_cart\_update BEFORE UPDATE ON CART FOR EACH ROW

BEGIN

IF NEW.QUANTITY <= 0 THEN SIGNAL SQLSTATE '14000'

SET MESSAGE\_TEXT = 'Incorrect quantity entered'; END IF;

END;//

CREATE TRIGGER rem\_cart\_insert BEFORE INSERT ON CART FOR EACH ROW

BEGIN

IF NEW.QUANTITY < 0 THEN SIGNAL SQLSTATE '14000'

SET MESSAGE\_TEXT = 'Incorrect quantity entered'; END IF;

END;// DELIMITER ;

#insert a product into order DELIMITER //

CREATE TRIGGER ins\_po BEFORE INSERT ON PRODUCT\_ORDER FOR EACH ROW

BEGIN

DECLARE cost DOUBLE DEFAULT NULL;

SELECT price INTO cost FROM PRODUCT WHERE product\_id = NEW.Product\_id; SET NEW.Price = cost;

UPDATE PRODUCT SET Units = Units - NEW.Quantity Where PRODUCT.Product\_id = NEW.Product\_id;

UPDATE G\_ORDER SET amount = amount + (NEW.Price)\*(NEW.Quantity) WHERE Order\_id = NEW.Order\_id;

END;//

CREATE TRIGGER upd\_po BEFORE UPDATE ON PRODUCT\_ORDER FOR EACH ROW

BEGIN

UPDATE PRODUCT SET Units = Units - NEW.Quantity + OLD.Quantity Where PRODUCT.Product\_id = NEW.Product\_id;

UPDATE G\_ORDER SET amount = amount + (NEW.Price)\*(NEW.Quantity- OLD.Quantity) WHERE Order\_id = NEW.Order\_id;

END;// DELIMITER ;

#procedures DELIMITER //

CREATE PROCEDURE PLACEORDER(IN username varchar(100),IN pay\_method varchar(100),IN address\_id INT)

BEGIN

DECLARE ord INT DEFAULT 0; DECLARE done INT DEFAULT FALSE; DECLARE pid INT DEFAULT 0; DECLARE Qty INT DEFAULT 0;

DECLARE avlQty INT DEFAULT 0;

DECLARE cur CURSOR FOR SELECT Product\_id,Quantity FROM CART WHERE user\_id = username;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET done = TRUE;

DECLARE exit handler for sqlexception BEGIN

-- ERROR

SIGNAL SQLSTATE '10000' SET MESSAGE\_TEXT = "ERROR IN PROCEDURE";

ROLLBACK;

END;

DECLARE exit handler for sqlwarning BEGIN

-- WARNING

SIGNAL SQLSTATE '10000' SET MESSAGE\_TEXT = "WARNING IN PROCEDURE";

ROLLBACK;

END;

OPEN cur;

START TRANSACTION;

SELECT auto\_increment INTO ord FROM information\_schema.TABLES WHERE table\_name = 'G\_ORDER' AND table\_schema = 'gstore';

INSERT INTO G\_ORDER (Payment\_Method, Address\_id, user\_id) VALUES (pay\_method, address\_id,username);

read\_loop : LOOP

FETCH cur INTO pid,Qty; IF done THEN

LEAVE read\_loop;

END IF;

INSERT INTO PRODUCT\_ORDER(Product\_id, Order\_id, Quantity) VALUES (pid,ord,qty);

END LOOP;

DELETE FROM CART WHERE user\_id = username; COMMIT;

END;// DELIMITER ;

#insertions

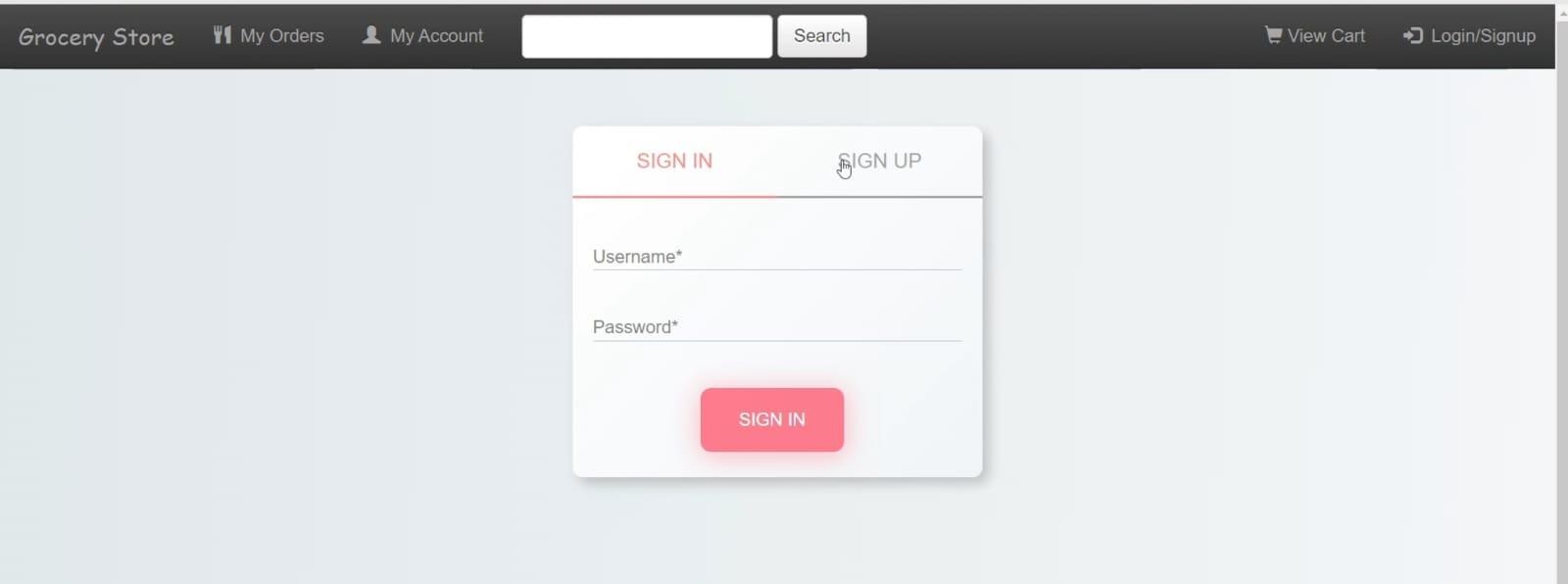
INSERT INTO MANUFACTURER(Manufacturer\_Name) VALUES ('Patanjali'),('Ayur'),('Head & Shoulder'),('Himalaya'),('Park Avenue'),('Amway'),('Haldiram'),('Dettol'),('Savlon'),('Lux'),('Pears'),('Dove'),('Khadi'),('Lif

ebuoy'),('Pantene'),('Aakash'),('Cadbury'),('Ferrero'),('Gits'),('Roopji'),('MTR'),('Chings'),('K norr'),('Sunfeast');

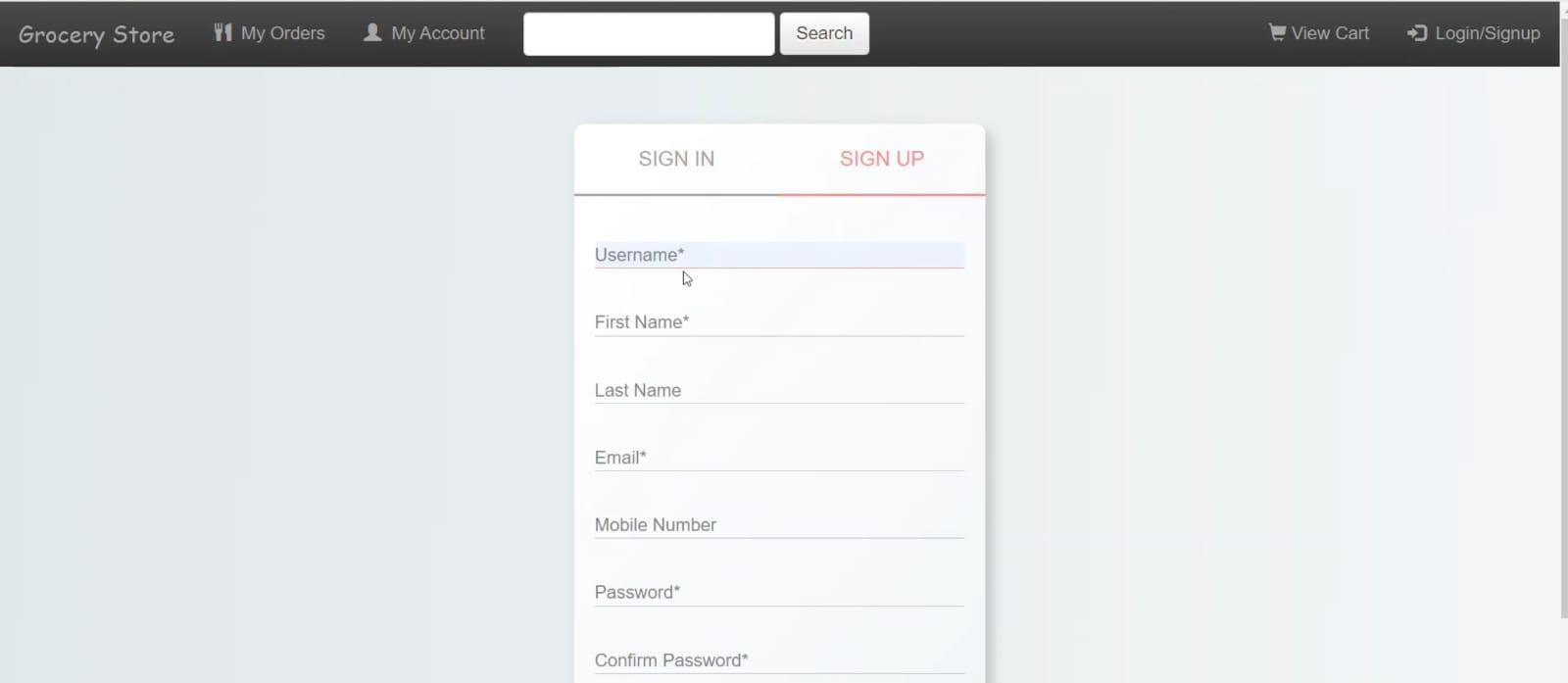
INSERT INTO PRODUCT(Product\_name, Units, Picture, Weight, Category\_id, Price, Product\_description, Manufacturer\_id) VALUES ('Dettol Skincare Soap',75,'Dettol\_skincare.jpg',75,2,25,'The same effectiveness of the Original Dettol Soap in a different package',8),('Dove Beauty Bar Soap',75,'Dove\_Beauty\_Bar\_Soap.jpg',100,2,60,'Feel the feather touch when you bath with dove beauty soap',12),VALUES ('Dove Daily Shine Shampoo', '92', 'Dove\_Daily\_Shine\_Shampoo.jpg', '100', '2', '110', 'Your hair : smoother.', '12');

**Appendix B: Screen Shots**

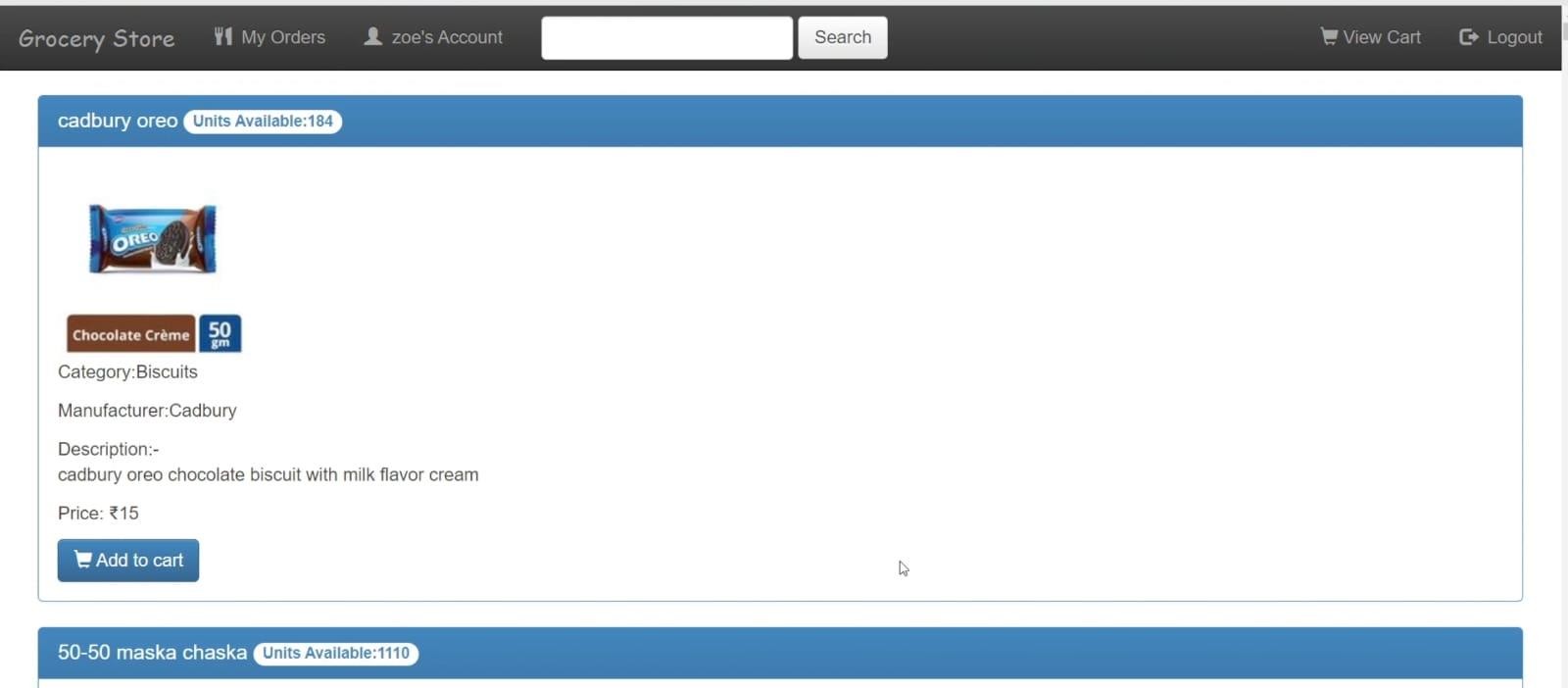
1. **Login page:**

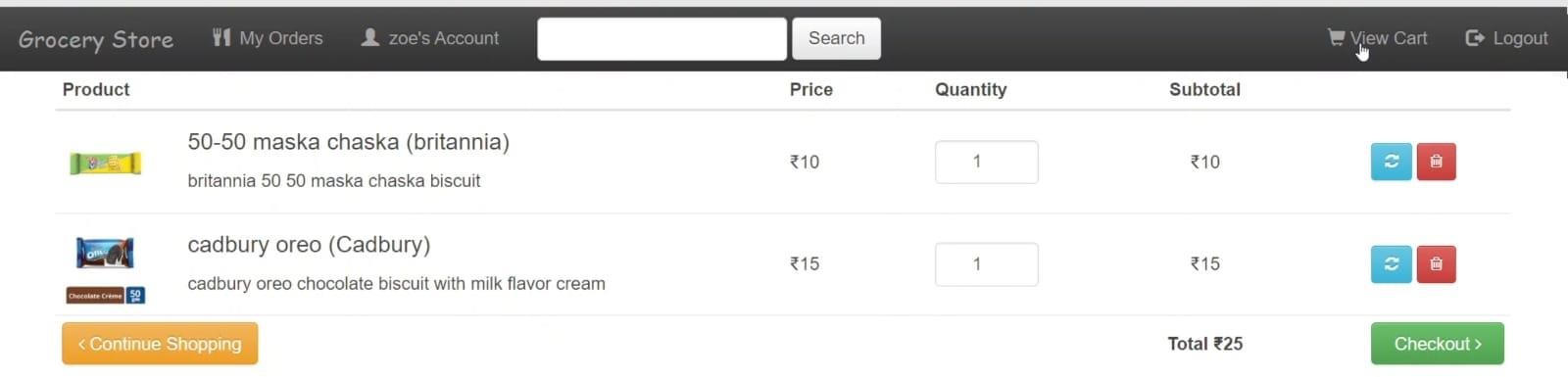


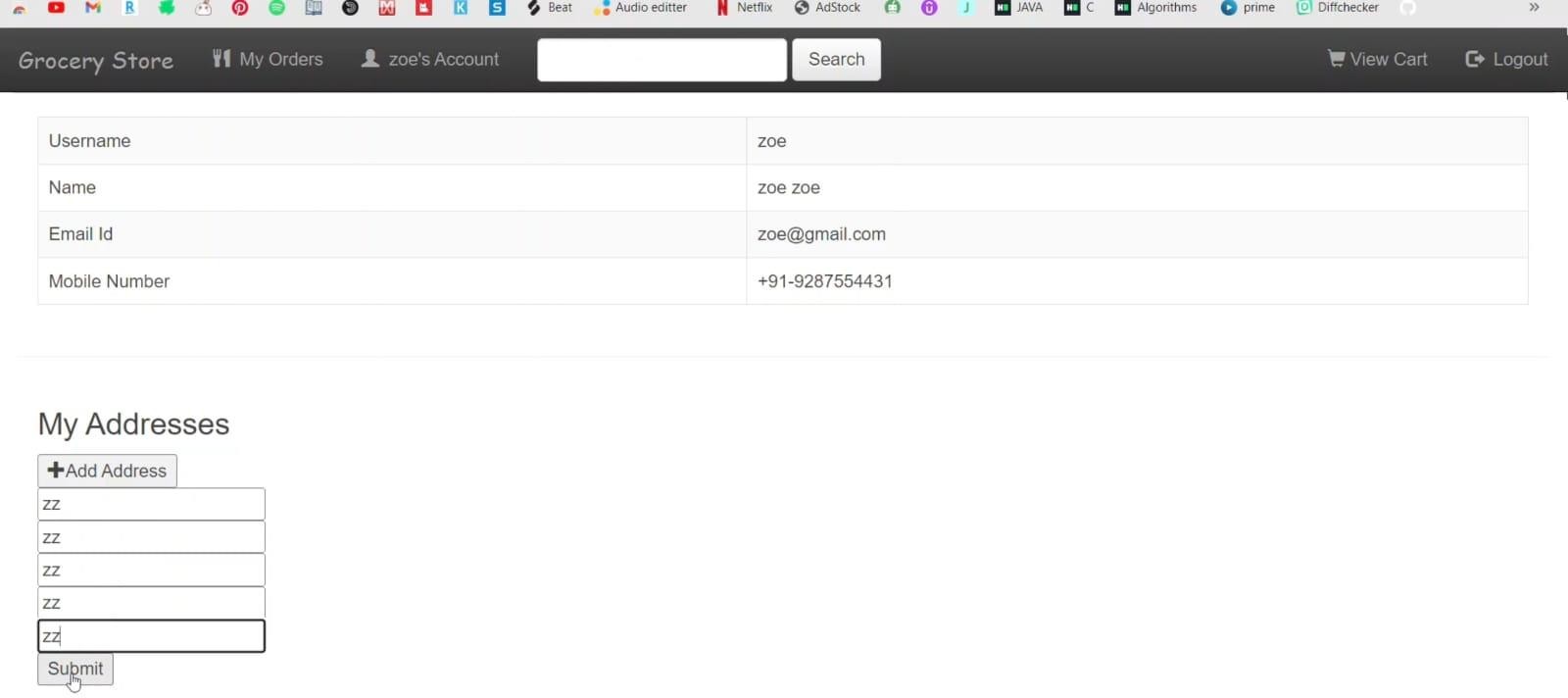
1. **Sign up page:**



1. **Products display page:**



1. **Cart view:**
2. **My orders:**



1. **Order confirmation page:**
2. **Payment Page:**