Database Management System
Mini Project Report

Project Title: Cash And Carry Mart
Team Member 1: Bijjavemula Chandrashekar
SRN: PES1UG23CS149
Team Member 2: Ayush Mittal
SRN: PES1UG23CS126
Github: https://github.com/giganiga6969/Dbms_miniproject

Description:

This project is a Cash & Carry Mart Database Management System designed to manage core retail operations on Customers, Products, Orders, and Inventory using a Node.js/Express application connected to a MySQL database. The system provides both a live dashboard for business insights and a management interface for CRUD operations. Key functionalities include automated membership discount calculation during order creation and the implementation of advanced database features like Triggers and a Stored Procedure to ensure data integrity and query efficiency.

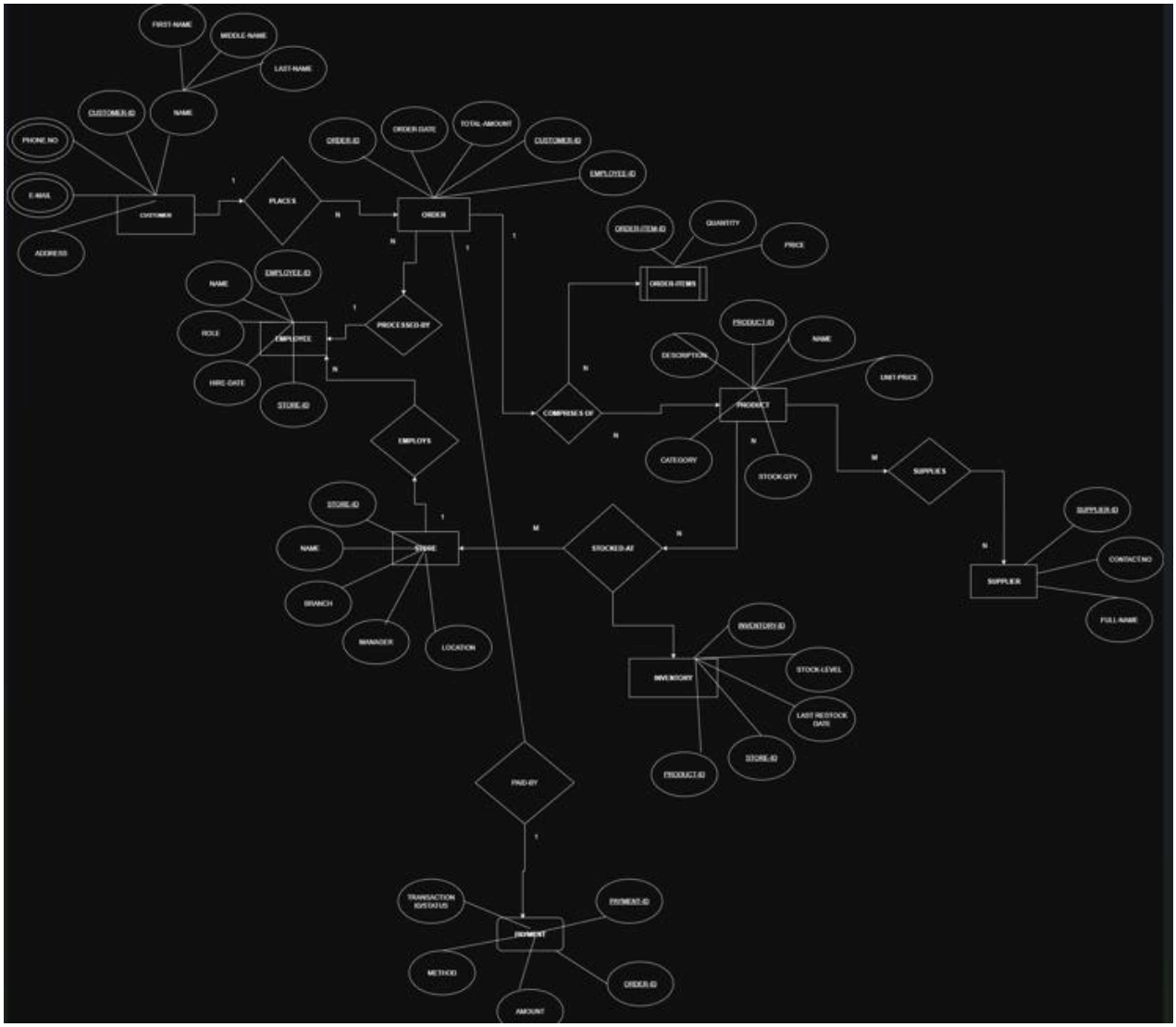
User Requirement Specification:

System Functionality	Description
Data Persistence & Integrity	All business data (Customers, Products, etc.) must be stored and managed in a normalized MySQL database
Full CRUD Support	Enable Create, Read, Update, and Delete operations for major entities via a web interface.
Advanced SQL Objects	Implement at least one Trigger (for automatic data update) and one Stored Procedure (for complex lookups).
Reporting and Visualization	Display real-time aggregated data on a dashboard using charts.
Transactional Logic	New orders must be created as a transaction, including the calculation and then displaying the order reciept

List of Softwares/Tools/Programming languages Used

Component	Software/Tool/Language
Database Management System	MySQL
Backend Framework	Python (Flask)
Database Driver	mysql-connector-python
Frontend Technologies	HTML, CSS, JavaScript
Frontend Libraries	Native JavaScript; Bootstrap for UI Flask (api) for backend api calls

ER Diagram:



Relational Schema:

CUSTOMER(customer_id PK, name, phone UNIQUE, email UNIQUE, address)

STORE(store_id PK, name, branch, manager, location, opening_hours)

PRODUCT(product_id PK, name, brand, category, price, stock_qty, in_stock)

INVENTORY(inventory_id PK,
store_id FK →
STORE.store_id,
product_id FK → PRODUCT.product_id,
stock_level, reorder_level,
UNIQUE(store_id, product_id))

CART(cart_id PK,
customer_id FK → CUSTOMER.customer_id,
product_id FK → PRODUCT.product_id,
quantity, created_at, updated_at, status)

ORDERS(order_id PK,
order_date, status, total_amount,
customer_id FK → CUSTOMER.customer_id)

PAYMENT(payment_id PK,
order_id UNIQUE FK → ORDERS.order_id,
amount, payment_date, method, transaction_status)

DDL Commands:

```
create database dbmsmp;
use dbmsmp;

CREATE TABLE CUSTOMER (
   customer_id    INT    AUTO_INCREMENT    PRIMARY
   KEY, name VARCHAR(100) NOT NULL,
   phone VARCHAR(20) UNIQUE NOT NULL,
   email VARCHAR(100) UNIQUE NOT NULL,
   address VARCHAR(255) NOT NULL
);

CREATE TABLE STORE (
   store_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   branch VARCHAR(100),
   manager VARCHAR(100) NOT NULL,
   location VARCHAR(255) NOT NULL,
   opening_hours VARCHAR(100) NOT NULL
);

CREATE TABLE PRODUCT (
   product_id INT AUTO_INCREMENT PRIMARY KEY,
   name VARCHAR(100) NOT NULL,
   brand VARCHAR(100),
   category VARCHAR(100) NOT NULL,
   price DECIMAL(10,2) NOT NULL CHECK (price >= 0),
   stock_qty INT DEFAULT 0 CHECK (stock_qty >= 0),
   in_stock BOOLEAN NOT NULL DEFAULT TRUE
);
```

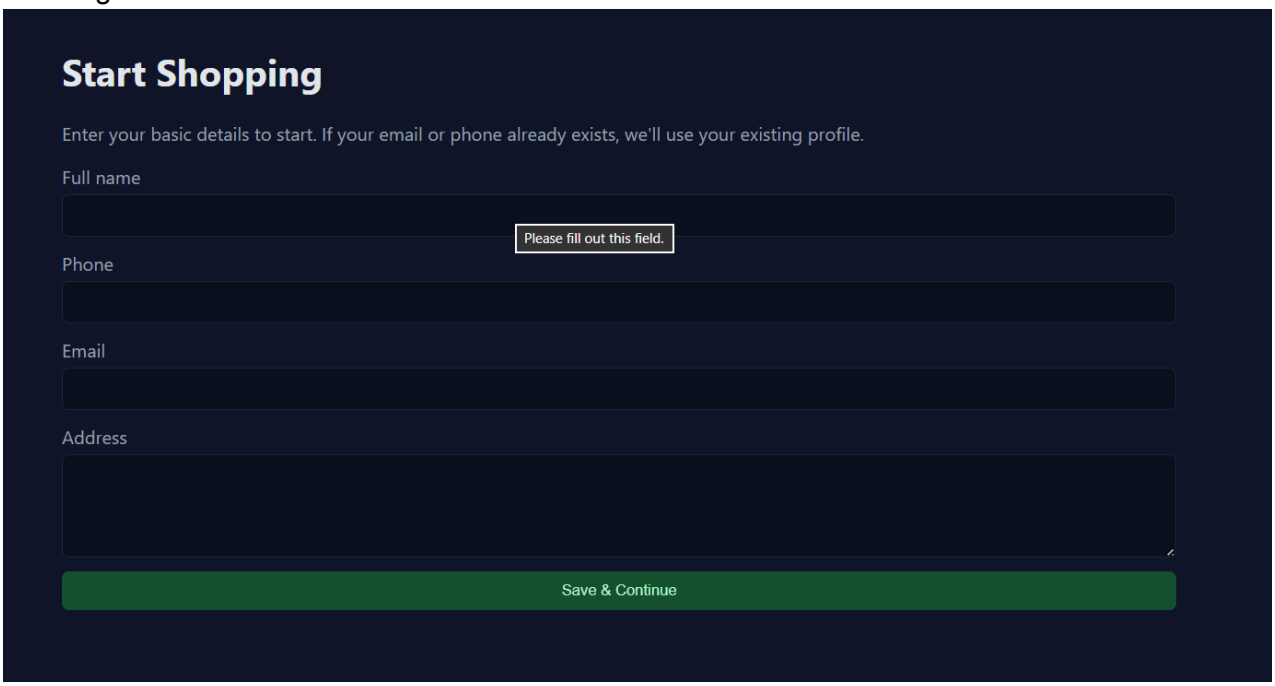
```
CREATE TABLE INVENTORY (
   inventory_id INT AUTO_INCREMENT PRIMARY KEY,
   store_id INT NOT NULL,
   product_id INT NOT NULL,
   stock_level INT DEFAULT 0 CHECK (stock_level >= 0),
   reorder_level INT DEFAULT 0 CHECK (reorder_level >= 0),
   FOREIGN KEY (store_id) REFERENCES STORE(store_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
   UNIQUE (store_id, product_id)
);

CREATE TABLE CART (
   cart_id INT AUTO_INCREMENT PRIMARY KEY,
   customer_id INT NOT NULL,
   product_id INT NOT NULL,
   quantity INT DEFAULT 1 CHECK (quantity > 0),
   created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   updated_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP ON UPDATE
CURRENT_TIMESTAMP,
   status ENUM('active','abandoned','checked_out') DEFAULT 'active',
   FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
   FOREIGN KEY (product_id) REFERENCES PRODUCT(product_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE ORDERS (
   order_id INT AUTO_INCREMENT PRIMARY KEY,
   order_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   status ENUM('pending','completed','cancelled') DEFAULT 'pending',
   total_amount DECIMAL(12,2) NOT NULL CHECK (total_amount >= 0),
   customer_id INT NOT NULL,
   FOREIGN KEY (customer_id) REFERENCES CUSTOMER(customer_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);

CREATE TABLE PAYMENT (
   payment_id INT AUTO_INCREMENT PRIMARY KEY,
   order_id INT UNIQUE NOT NULL,
   amount DECIMAL(12,2) NOT NULL CHECK (amount >= 0),
   payment_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
   method ENUM('cash','card','upi','wallet') NOT NULL,
   transaction_status ENUM('success','failed','pending') DEFAULT 'pending',
   FOREIGN KEY (order_id) REFERENCES ORDERS(order_id)
```

```
        ON DELETE CASCADE ON UPDATE CASCADE
);
```

CRUD operation Screenshots:

Create operation:

Creating a new order which inserts into the database.

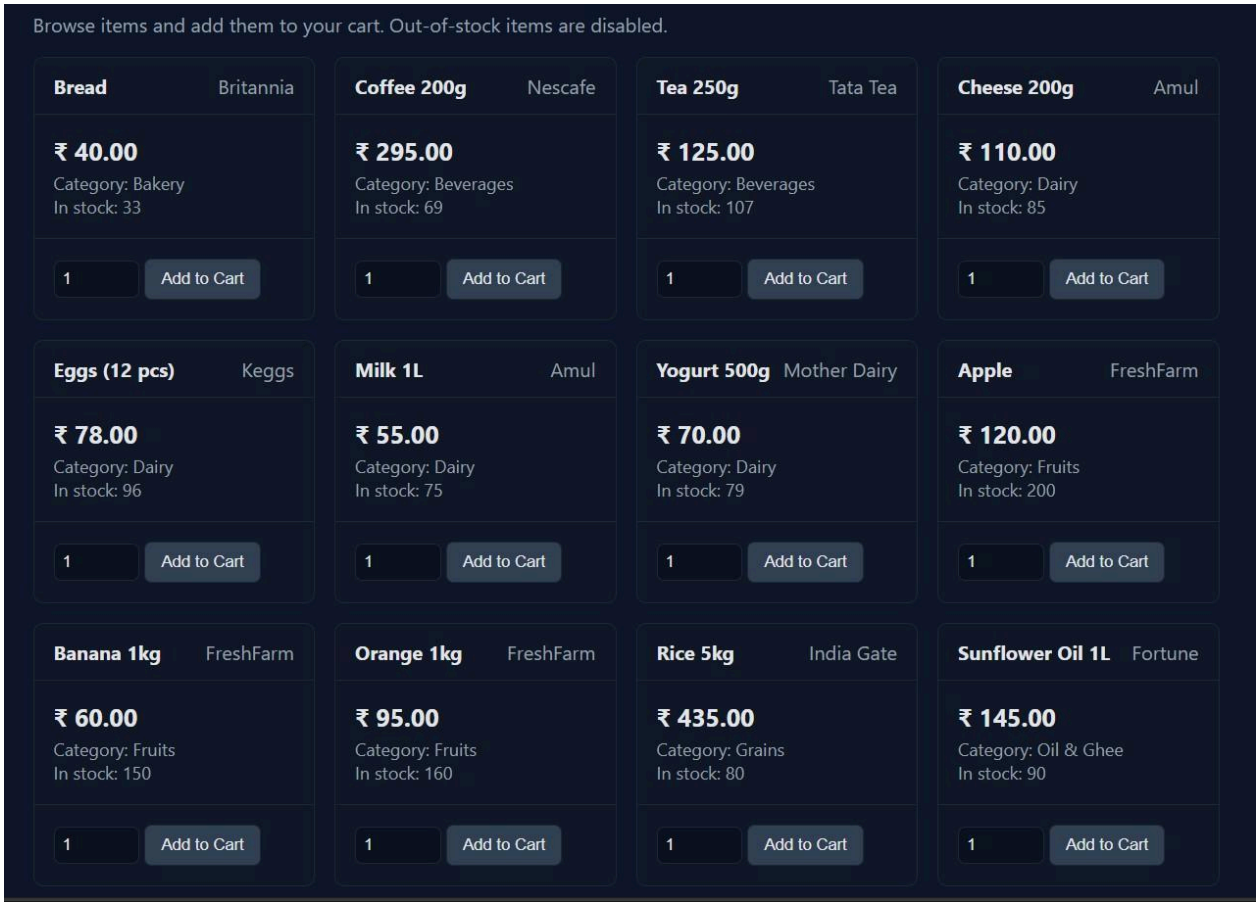


Query for create:

```
cur = conn.cursor()
cur.execute(
    """
    INSERT INTO CUSTOMER(name, phone, email, address)
    VALUES (%s, %s, %s, %s)
    """,
    (name, phone, email, address),
)
customer_id = cur.lastrowid
cur.close()
cur2 = conn.cursor()
cur2.execute(
    """
    INSERT INTO CART(customer_id, product_id, quantity, status)
    VALUES (%s, %s, %s, 'active')
    """,
    (cid, product_id, qty),
)
cur2.close()
cur3 = conn.cursor()
```

```
cur3.execute(
    """
    INSERT INTO ORDERS(status, total_amount, customer_id)
    VALUES('completed', %s, %s)
    """,
    (str(total), cid),
)
order_id = cur3.lastrowid
cur3.close()
cur4 = conn.cursor()
cur4.execute(
    """
    INSERT INTO PAYMENT(order_id, amount, method, transaction_status)
    VALUES(%s, %s, %s, 'success')
    """,
    (order_id, str(total), payment_method),
)
cur4.close()
```

Read operation:
As soon as u login , we are about to read and display all the products present in the database.



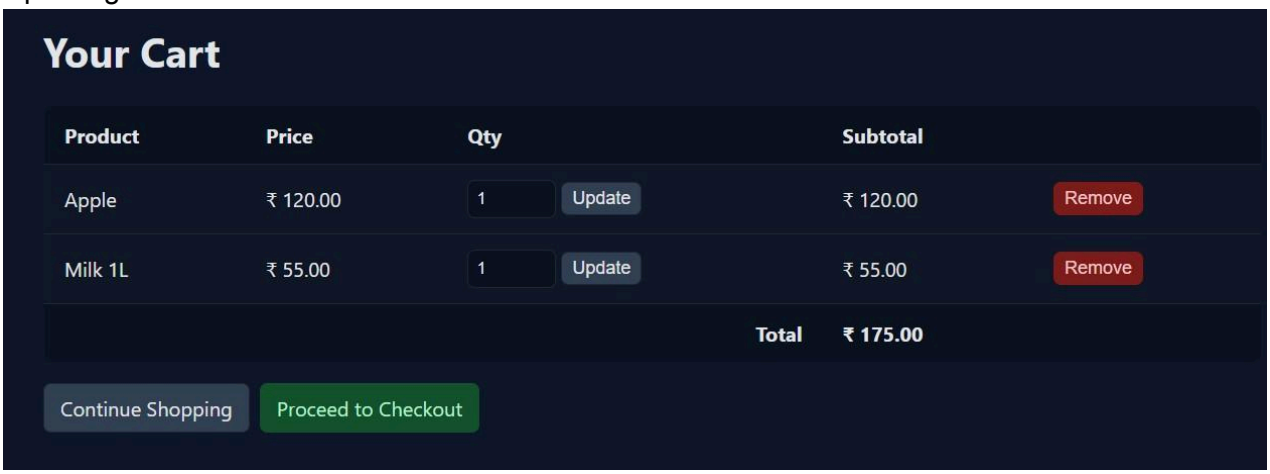
Query for read operation:

```
def fetch_customer_by_email_or_phone(conn, email, phone):
    cur = conn.cursor(dictionary=True)
    cur.execute(
        "SELECT * FROM CUSTOMER WHERE email = %s OR phone = %s LIMIT 1",
        (email, phone),
    )
    row = cur.fetchone()
    cur.close()
    return row
cur = conn.cursor(dictionary=True)
cur.execute(
    "SELECT product_id, name, brand, category, price, stock_qty, in_stock FROM PRODUCT
ORDER BY category, name"
)
products = cur.fetchall()
cur.close()
cur.execute(
    """
    SELECT ct.cart_id, ct.product_id, ct.quantity, p.name, p.brand, p.price, p.stock_qty, p.in_stock
```

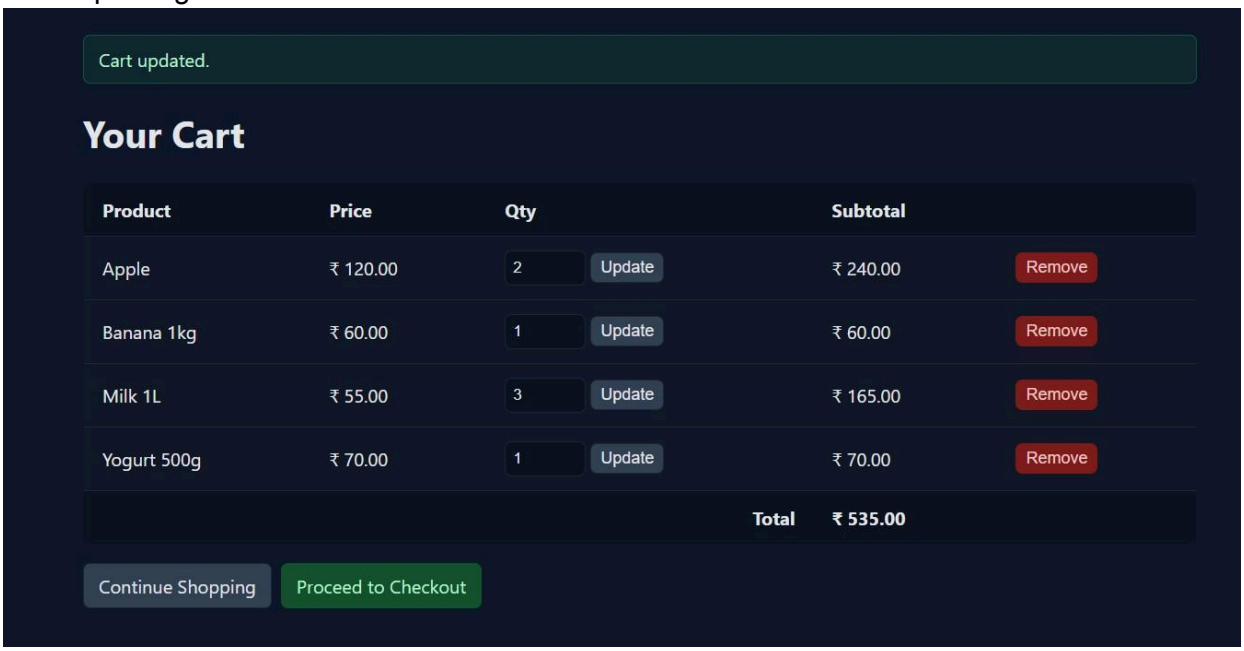
```
    FROM CART ct
    JOIN PRODUCT p ON p.product_id = ct.product_id
    WHERE ct.customer_id = %s AND ct.status = 'active'
    ORDER BY p.name
    """,
    (cid,),
)
rows = cur.fetchall()
```

Update Operation:
Updating:

Updating:



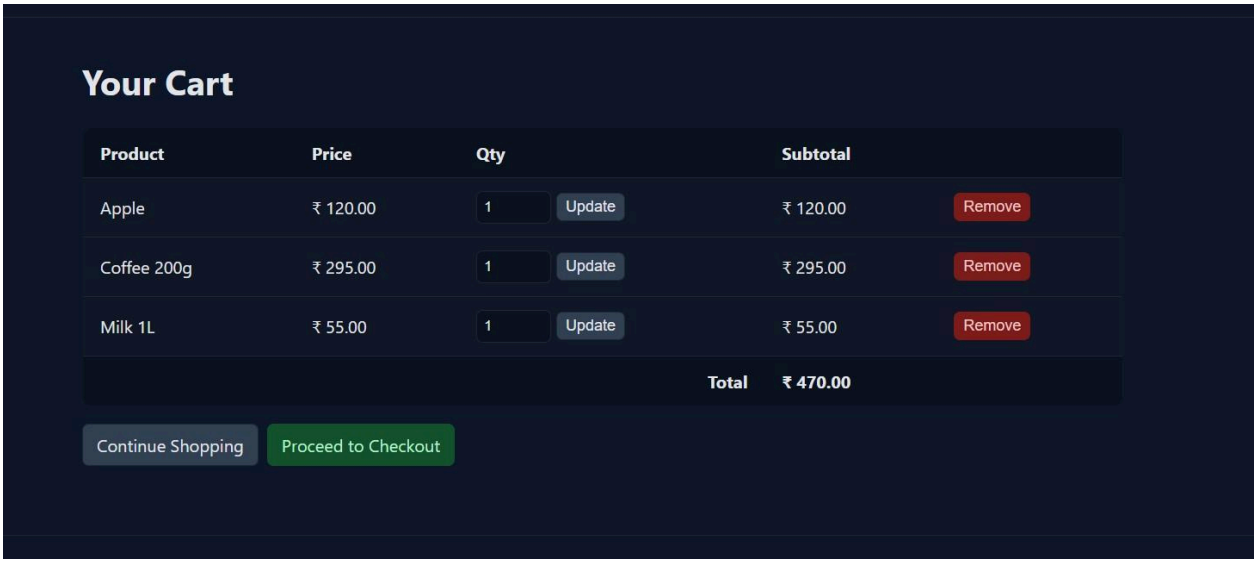
After Updating:



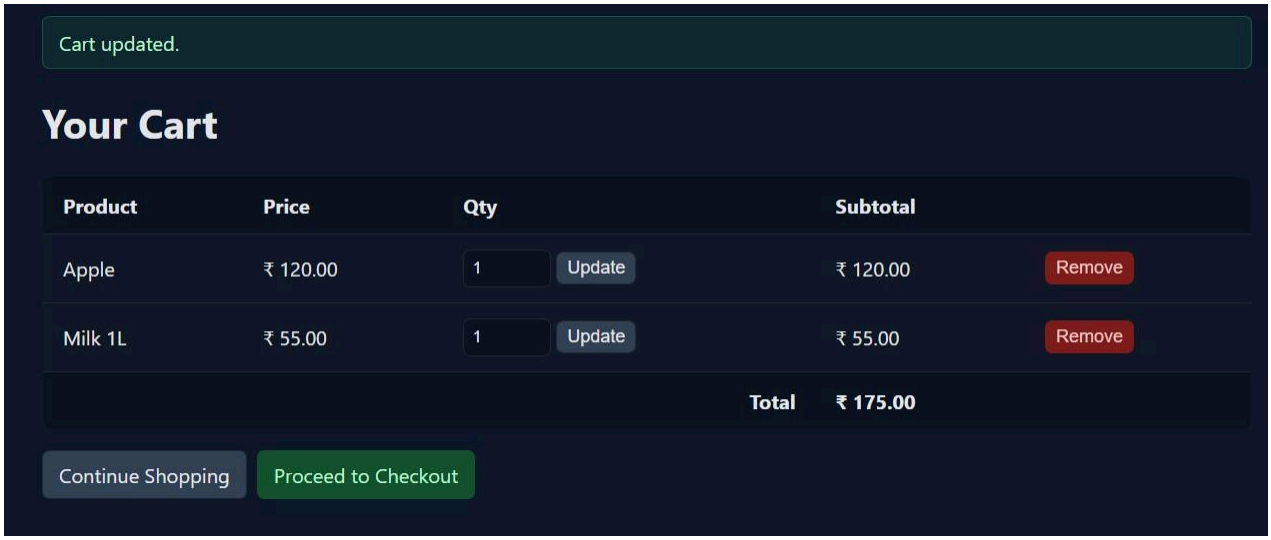
Query for updating the product:

```
cur2 = conn.cursor()
cur2.execute(
    "UPDATE CART SET quantity = %s WHERE cart_id = %s",
    (new_qty, existing['cart_id']),
)
cur2.close()
cur = conn.cursor()
cur.execute(
    "UPDATE CART SET quantity = %s WHERE cart_id = %s AND customer_id = %s AND status
= 'active'",
    (qty, cart_id, cid),
)
cur.close()
cur2 = conn.cursor()
cur2.execute(
    "UPDATE CART SET status = 'checked_out' WHERE cart_id = %s",
    (it['cart_id'],),
)
cur2.close()
```

Delete Operation:



After:



Query for deleting the product:

```
cur = conn.cursor()
cur.execute(
   "DELETE FROM CART WHERE cart_id =
%s AND customer_id = %s AND status =
'active'",
   (cart_id, cid),
)
cur.close()
 cur = conn.cursor()
cur.execute(
   "DELETE FROM CART WHERE cart_id = %s AND customer_id = %s AND status = 'active'",
   (cart_id, cid),
)
cur.close()
```

List of Functionalities/Features and Associated Screenshots using Front End:

1. **index.html** displays the **list of all available products** in the supermarket application. It fetches product details such as **name, brand, category, price, stock quantity, and availability** from the database and shows them to the user.

The page acts as the **shopping dashboard** where the customer can:

- View all products grouped by category
- See product availability (in_stock flag)
- Add items to the cart
- Navigate to the cart or checkout pages

2. **cart.html** allows customers to manage items in their active cart:

- View all products added to the cart
- See price, quantity, stock availability
- Update quantity
- Remove items from the cart
- View the total payable amount

This page internally performs **UPDATE** and **DELETE** operations on the CART table.

3. checkout.html (Checkout Page) checkout.html shows:

- All items to be purchased
- Total bill amount
- Payment method selection (cash, card, UPI, wallet) Upon confirming:

- Cart items are marked as **checked_out**
- An entry is created in **ORDERS**
- A payment record is created in **PAYMENT**
- Stock is automatically decreased using the SQL trigger

Triggers, Procedures/Functions, Nested Query, Join, Aggregate Queries

Trigger	trg_after_cart_checkedout	CREATE TRIGGER trg_after_cart_checkedout AFTER UPDATE ON CART FOR EACH ROW BEGIN IF OLD.status <> 'checked_out' AND NEW.status = 'checked_out' THEN UPDATE PRODUCT SET stock_qty = stock_qty - NEW.quantity, in_stock = (stock_qty - NEW.quantity > 0)

		WHERE product_id = NEW.product_id;
		
		END IF;
		END;

Stored Procedure	CREATE PROCEDURE sp_create_customer CREATE PROCEDURE sp_update_cart_quantity	Registering a new customer when they enter their details in the application Avoiding direct INSERT queries from the application for better security Ensuring consistent customer creation through a controlled stored procedure Used when a user increases or decreases quantity of an item in their cart Ensures business rules (e.g., no negative quantity, no exceeding stock) Prevents bad data being saved directly into the CART table
Nested Query	Find customers who have more cart items than the average cart quantity	SELECT customer_id, SUM(quantity) AS total_qty FROM CART GROUP BY customer_id HAVING total_qty > (SELECT AVG(total_quantity) FROM (SELECT SUM(quantity) AS total_quantity FROM CART GROUP BY customer_id) AS sub;);

Join Query	Show all customers with the products in their cart	SELECT c.name AS customer_name, p.name AS product_name, ct.quantity AS quantity_in_cart, ct.status FROM CUSTOMER c JOIN CART ct ON c.customer_id = ct.customer_id JOIN PRODUCT p ON p.product_id = ct.product_id;
Aggregate Query	Total quantity purchased by each customer	SELECT c.name AS customer_name, SUM(ct.quantity) AS total_items_bought FROM CUSTOMER c JOIN CART ct ON c.customer_id = ct.customer_id GROUP BY c.customer_id;

Code snippets for invoking the Procedures/Functions/Trigger:

```
-- CUSTOMER PROCEDURES
CREATE PROCEDURE sp_create_customer (
  IN p_name VARCHAR(200),
  IN p_phone VARCHAR(30),
  IN p_email VARCHAR(255),
  IN p_address TEXT,
  OUT p_customer_id INT
)
BEGIN
  INSERT INTO CUSTOMER(name, phone, email, address)
  VALUES (p_name, p_phone, p_email, p_address);
  SET p_customer_id = LAST_INSERT_ID();
END;

CREATE PROCEDURE sp_get_customer_by_email_or_phone (
  IN p_email VARCHAR(255),
  IN p_phone VARCHAR(30)
)
BEGIN
  SELECT * FROM CUSTOMER
  WHERE email = p_email OR phone = p_phone
  LIMIT 1;
```

```
END;

-- PRODUCT PROCEDURES
CREATE PROCEDURE sp_create_product (
  IN p_name VARCHAR(255),
  IN p_brand VARCHAR(128), IN
  p_category VARCHAR(128), IN
  p_price DECIMAL(12,2),
  IN p_stock_qty INT,
  OUT p_product_id INT
)
BEGIN
  INSERT INTO PRODUCT(name, brand, category, price, stock_qty, in_stock)
  VALUES (p_name, p_brand, p_category, p_price, p_stock_qty, p_stock_qty > 0);
  SET p_product_id = LAST_INSERT_ID();
END;

-- CART PROCEDURES
CREATE PROCEDURE sp_add_to_cart (
  IN p_customer_id INT,
  IN p_product_id INT,
  IN p_quantity INT,
  OUT p_success BOOLEAN,
  OUT p_message TEXT
)
BEGIN
  DECLARE v_stock INT;
  DECLARE v_in_stock BOOLEAN;

  SELECT stock_qty, in_stock INTO v_stock, v_in_stock
  FROM PRODUCT WHERE product_id = p_product_id;

  IF v_stock IS NULL THEN
    SET p_success = FALSE;
    SET p_message = 'Product not found';
    LEAVE sp_add_to_cart;
  END IF;

  IF NOT v_in_stock OR p_quantity > v_stock THEN
    SET p_success = FALSE;
    SET p_message = 'Insufficient stock';
    LEAVE sp_add_to_cart;
  END IF;
```

```
  IF EXISTS (SELECT 1 FROM CART
            WHERE customer_id = p_customer_id
            AND product_id = p_product_id
            AND status = 'active') THEN
    UPDATE CART
    SET quantity = quantity + p_quantity
    WHERE customer_id = p_customer_id
    AND product_id = p_product_id
    AND status = 'active';
  ELSE
    INSERT INTO CART(customer_id, product_id, quantity, status)
  VALUES (p_customer_id, p_product_id, p_quantity, 'active');
  END IF;

  SET p_success = TRUE;
  SET p_message = 'Added to cart';
END;

CREATE PROCEDURE sp_update_cart_quantity (
  IN p_cart_id INT,
  IN p_customer_id INT,
  IN p_quantity INT,
  OUT p_success BOOLEAN,
  OUT p_message TEXT
)
BEGIN
  IF p_quantity <= 0 THEN
    DELETE FROM CART
    WHERE cart_id = p_cart_id AND customer_id = p_customer_id AND status = 'active';

    SET p_success = TRUE;
    SET p_message = 'Item removed';
  ELSE
    UPDATE CART SET quantity = p_quantity
    WHERE cart_id = p_cart_id AND customer_id = p_customer_id
    AND status = 'active';

    SET p_success = TRUE;
    SET p_message = 'Quantity updated';
  END IF;
END;

CREATE PROCEDURE sp_remove_from_cart (
  IN p_cart_id INT,
```

```
  IN p_customer_id INT
)
BEGIN
  DELETE FROM CART
  WHERE cart_id = p_cart_id
  AND customer_id = p_customer_id
  AND status = 'active';
END;

-- CHECKOUT PROCEDURE
CREATE PROCEDURE sp_checkout (
  IN p_customer_id INT,
  IN p_payment_method VARCHAR(50),
  OUT p_order_id INT,
  OUT p_success BOOLEAN,
  OUT p_message TEXT
)
BEGIN
  DECLARE v_total DECIMAL(12,2) DEFAULT 0;

  SELECT SUM(ct.quantity * p.price)
  INTO v_total
  FROM CART ct
  JOIN PRODUCT p ON p.product_id = ct.product_id
  WHERE ct.customer_id = p_customer_id AND ct.status = 'active';

  IF v_total IS NULL THEN
    SET p_success = FALSE;
    SET p_message = 'Cart empty';
    LEAVE sp_checkout;
  END IF;

  INSERT INTO ORDERS(status, total_amount, customer_id)
  VALUES ('completed', v_total, p_customer_id);
  SET p_order_id = LAST_INSERT_ID();

  UPDATE CART SET status = 'checked_out'

  WHERE customer_id = p_customer_id AND status = 'active';

  INSERT INTO PAYMENT(order_id, amount, method, transaction_status)
  VALUES (p_order_id, v_total, p_payment_method, 'success');

  SET p_success = TRUE;
```

```
  SET p_message = 'Checkout successful';
END;

DELIMITER $$

-- Keep PRODUCT.in_stock synced with PRODUCT.stock_qty
DROP TRIGGER IF EXISTS trg_product_bi_stock$$
CREATE TRIGGER trg_product_bi_stock
BEFORE INSERT ON PRODUCT
FOR EACH ROW
BEGIN
  SET NEW.in_stock = (NEW.stock_qty > 0);
END$$

DROP TRIGGER IF EXISTS trg_product_bu_stock$$
CREATE TRIGGER trg_product_bu_stock
BEFORE UPDATE ON PRODUCT
FOR EACH ROW
BEGIN
  SET NEW.in_stock = (NEW.stock_qty > 0);
END$$

-- Validate CART insert against available stock
DROP TRIGGER IF EXISTS trg_cart_bi_validate$$
CREATE TRIGGER trg_cart_bi_validate
BEFORE INSERT ON CART
FOR EACH ROW
BEGIN
  DECLARE v_stock INT DEFAULT 0;
  DECLARE v_in_stock TINYINT DEFAULT 0;
  SELECT stock_qty, in_stock INTO v_stock, v_in_stock
  FROM PRODUCT
  WHERE product_id = NEW.product_id
  FOR UPDATE;

  IF v_stock IS NULL THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Product not found';
  END IF;
  IF v_in_stock = 0 OR v_stock <= 0 OR NEW.quantity > v_stock THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock for product';
  END IF;
END$$

-- Validate quantity changes and decrement stock when checking out
DROP TRIGGER IF EXISTS trg_cart_bu_validate_and_checkout$$
CREATE TRIGGER trg_cart_bu_validate_and_checkout
BEFORE UPDATE ON CART
```

```
FOR EACH ROW
BEGIN
  DECLARE v_stock INT DEFAULT 0;
  DECLARE v_in_stock TINYINT DEFAULT 0;
  DECLARE v_new_stock INT DEFAULT 0;

  -- Ensure product exists and lock it
  SELECT stock_qty, in_stock INTO v_stock, v_in_stock
  FROM PRODUCT
  WHERE product_id = NEW.product_id
  FOR UPDATE;

  IF v_stock IS NULL THEN
    SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Product not found';
  END IF;

  -- When cart remains active and qty changes, validate available stock
  IF NEW.status = 'active' THEN
    IF NEW.quantity <= 0 THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Quantity must be > 0';
    END IF;
    IF NEW.quantity > v_stock THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock for requested
quantity';
    END IF;
  END IF;

  -- On transition to checked_out, atomically decrease product stock
  IF OLD.status <> 'checked_out' AND NEW.status = 'checked_out' THEN
    IF NEW.quantity > v_stock THEN
      SIGNAL SQLSTATE '45000' SET MESSAGE_TEXT = 'Insufficient stock to checkout';
    END IF;
    SET v_new_stock = v_stock - NEW.quantity;
    UPDATE PRODUCT
      SET stock_qty = v_new_stock,
          in_stock = (v_new_stock > 0)
      WHERE product_id = NEW.product_id;
  END IF;
END$$

DELIMITER ;

-- One-time sync to correct any in_stock flags for existing data
UPDATE PRODUCT SET in_stock = (stock_qty > 0);
```