DBMS PROJECT - GROUP 67

Aryman Srivastava - 2020184
Dheeraj - 2020194

Online Retail Shop

Scope & Objective

The Online Retail Store project is aimed at delivering a comprehensive e-commerce platform to customers. The platform will have a user-friendly interface that enables customers to easily browse and purchase products. The key features of the system include a product catalog, customer accounts, order tracking, and others aimed at enhancing the shopping experience.

Stakeholders

The stakeholders involved in the project are:

1) **Customers**: The customers are the primary focus of the project. They will be using the online store to purchase products and avail of the various services offered.

2) **Suppliers**: These individuals will provide goods and products to the store for sale. They will play a critical role in ensuring that the store has a wide range of products for customers to choose from.

3) **Employees**: Employees are responsible for managing the store operations. Their key tasks include managing products, restocking items, and ensuring that the store is running smoothly.

The primary objective of the Online Retail Store project is to provide a seamless and user-friendly shopping experience to customers. The project aims to deliver all the necessary features and functionality to make it easy for customers to find and purchase the products they want.

Database Schema

Keys :-

PK- Primary Key
NN- Not Null
UQ- unique
FK- Foreign Key
AI - Auto_Increment
C()- Check (> OR < OR =)
DF()- Default (Value)

TABLES	FIELDS	DATATYPE	CONSTRAINTS
Address	Address_id HouseNo City Line_1 Pincode	INT VARCHAR(10) VARCHAR(50 VARCHAR(100) INT	PK,AI NN NN NN
Customer	C_id F_name M_name L_name Phn_n Email Gender Address_id Pass	INT VAHRCHAR(50) VAHRCHAR(50) VAHRCHAR(50) VAHRCHAR(16) VAHRCHAR(100) EMU(M,F,OTHER) INT VARCAHR(30)	PK,AI NN NN NN UQ,NN NN FF NN
PRODUCT	P_id P_name P_type Price S_quantity Descr	INT VARCHAR(100) VARCHAR(50) DECIMAL(10,2) INT TEXT	PK,AI NN NN NN, C(>0) NN,DF(0),C(>0)

Cart	Ca_id C_id Count T_cost	INT INT INT DECIMAL(10,2)	PK,AI NN,FK NN NN

Cart_Products	Cp_id Ca_id P_id Count	INT INT INT INT	PK,AI NN,FK FK NN
Order_	O_id C_id O_date Ship_date T_cost	INT INT DATE DATE DECIMAL(10,2)	PK,AI NN NN NN NN
Employees	Emp_id F_name L_name M_name Phn_no Email Gender Pass	INT VARCHAR(50) VARCHAR(50) VARCHAR(50) VARCHAR(15) VARCHAR(100) ENUM(M,F,OTHER) VARCHAR(30)	PK,AI NN NN DF(‘’) NN NN,UQ NN NN
Suppliers	Sup_id Sup_name P_id Quantity Phn_no Email_id	INT VARCHAR(100) INT INT VARCHAR(15) VARCHAR(100)	PK,AI NN NN,FK NN NN NN,UQ
Payment	Payment_id C_id O_id P_mode Total_cost	INT INT INT VARCHAR(20) DECIMAL(10,2)	PK,AI NN,FK FK NN NN
Complaints	Complain_id P_id Complain_date Comments E_id	INT INT DATE TEXT INT	PK,AI NN,FK NN NN FK

Shipping_Details	Ship_id P_id Pos	INT INT VARCHAR(200)	PK,AI NN NN
Feedback	Feedback_id C_id P_id Comm	INT INT INT TEXT	PK,AI FK NN,FK NN

Relationships

The Online Retail Store project involves creating a comprehensive e-commerce platform that provides customers with an easy-to-use shopping experience. The database model used in this project follows the example of successful e-commerce platforms such as Big Bazaar and Amazon. The relationships between different entities in the database model include:

1. **Has (Between Address, Customer):** This relationship represents the fact that a customer can have multiple addresses. It is marked as n..1, indicating that one customer can have multiple addresses.
2. **Chooses (Between Customer, Product):** This relationship represents the customer's action of choosing a product that they wish to purchase. It is a 1..n relationship, meaning that one customer can choose multiple products.
3. **Add To Cart (Between Product, Cart):** This relationship represents the customer's action of adding a chosen product to their cart. It is marked as n..1, indicating that multiple products can be added to one cart.
4. **Manages (Between Manager, Product, Complaint, Feedback):** This is a quadruple/multi-relationship between Manager, Product, Feedback, and Complaint entities. It represents the employee's responsibility of managing the products, feedback, and complaints provided by the customer.
5. **Supplied by (Between Supplier, Product):** This relationship represents the act of suppliers supplying products to the store. The purchasing of products is managed by employees.
6. **Gets (Between Customer, Order_):** This relationship represents the customer's receipt of the final order details.

7. **Has (Between Order_, Shipping Details):** This relationship represents the fact that each order detail must have a corresponding shipping detail entity.
8. **For (Between Order_, Payment):** This relationship represents the payment details for the order details.
9. **Does (Between Customer, Payment):** This relationship represents the customer's action of paying for the orders that were added to the cart.
10. **Files (Between Customer, Complaints):** This relationship represents the customer's filing of a complaint for a product.
11. **Gives (Between Customer, Feedback):** This relationship represents the customer's provision of feedback about a product.

Weak Entity

The database model we are developing follows the example of Big Bazaar and Amazon. Weak entities are those entities that do not have enough information/attributes to form a primary key of its own and depend on other entities. In our database entities such as **order_, shipping details, payments** are taken as weak entities.

The **order_ entity** is developed only when a user adds products to the cart and goes for final payment. The **shipping details** entity also comes into action when an order detail has been generated and the customer chooses a shipping address. The **payment** entity is present for the customer to pay for the orders and place orders.

Total Participation

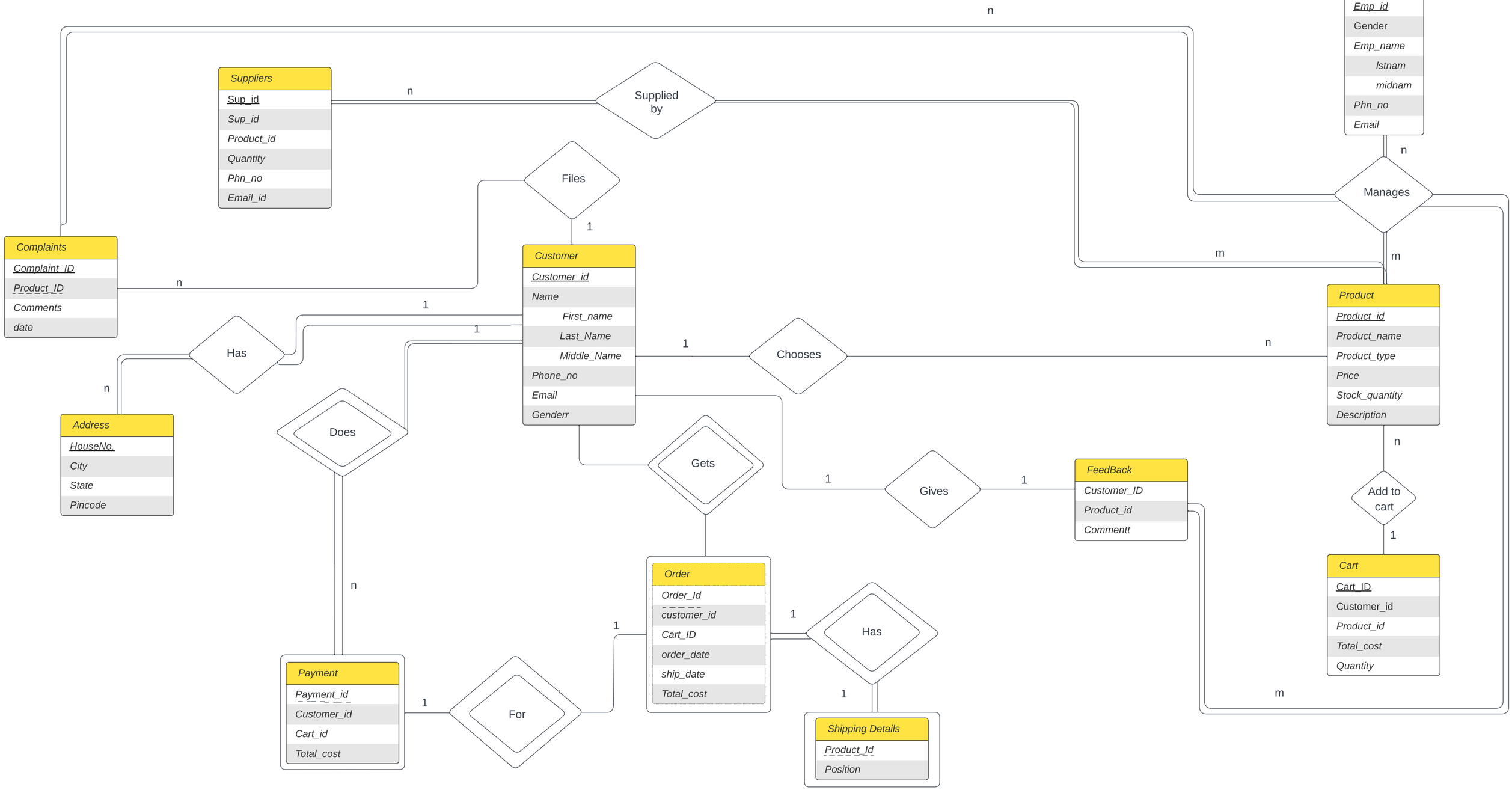
Some relationships in the database model include total participation. These include **Does, Has, Supplied by, and Manages**. The **Does** relationship indicates that each customer must pay for their orders in order for the order to be placed. The **Has** relationship indicates that each order detail must have shipping details and each customer must have addresses. The **Supplied by** relationship indicates that each product must be supplied by a supplier. The **Manages** relationship has total participation since all products, complaints, and feedback are managed by employees only.

Additional Functionalities

Two additional functionalities were developed in the Customer table. The **Foreign Key Attribute Address_Id** was added to ensure that when a customer removes an address, the corresponding rows in the table using that address will have its value set to null. Another functionality is that when a customer leaves the store, all of their stored data will be removed from the database, which is achieved using the **Foreign Key C_Id**.

DBMS ER diagram (UML notation)

Aryman Srivastava Dheeraj | February 17, 2023



Relational Algebraic Operations

Query 1 ->

update_cart_product(id, prod_id, count_set) =
IF
σ Ca_id = id ∧ P_id = prod_id (cart_products) ≠ ∅
THEN
(cart_products ⋈ Ca_id = id ∧ P_id = prod_id (ρ P_id ← P_id, COUNT ← count_set (product))) ⋈ Ca_id = id ∧ P_id = prod_id (cart_products) → π COUNT (cart_products)
ELSE
π Ca_id, P_id, count_set (σ P_id = prod_id (product)) → cart_products

Query 2->

UPDATE_cart_products := π Ca_id, P_id, MAX(0, count - 1) (σ Ca_id = 1 ∧ P_id = 5 ∧ count > 0 (cart_products))

DELETE_cart_products := ρ temp(cart_products) (σ Ca_id = 1 ∧ P_id = 5 ∧ count = 0 (cart_products)) − cart_products

Query 3->

IF_cart_exists :=
σ C_id = c_id (cart)

IF IF_cart_exists ≠ ∅ THEN
UPDATE_cart :=
(cart ⋈ C_id = c_id (SELECT_cart_products)) →
π C_id, count + SUM, T_cost + (SUM * 5.5) (cart)
ELSE
INSERT_cart :=
π c_id, SUM, SUM * 5.5 (σ Ca_id = c_id (SELECT_cart_products)) → cart

Query 4->
SELECT_T_cost :=
π T_cost (σ C_id = id (cart))

INSERT_order :=
(order_ ⋈ O_id=max(O_id)+1) →
(π C_id, max(O_id)+1, CURDATE(), DATE_ADD(CURDATE(), INTERVAL 15 DAY), SELECT_T_cost)

Query 5->
INSERT_payment :=
π C_id, O_id, T_cost, "UPI" (σ C_id = id (order_)) →
(σ O_id = max(O_id) (order_) ⋈ O_id, C_id, T_cost, P_mode) →
(π C_id, max(O_id), T_cost, P_mode)

Query 6->
INSERT_shipping_details :=
π P_id, "SHIPPING STARTED" (σ Ca_id = id (cart_products))

Query 7->
SELECT_product :=
π P_id, p_type, price (product) ⋈
(π p_type, max_price (ρ p_type/p_type2, max_price (γ p_type; max(price))))
(product)

Query 8->
SELECT_product :=
π P_id, p_type, price (product) ⋈
(π p_type, min_price (ρ p_type/p_type2, min(price) (γ p_type; min(price))))
(product)

Query 9->

T1 ← product ⋈ cart_products σ Ca_id = id (cart_products)
T2 ← π P_id, COUNT (T1)product ← product - T2 ⋈ product.P_id = T2.P_id
product.S_quantity ← product.S_quantity - T2.COUNT σ T2.P_id = product.P_id
(product)

Query 10->

id_e = (π Emp_id (employees)) ×_{RAND()} (employees)
complaints_temp = σ P_id = id (complaints)
complaints_temp ← complaints_temp ⋈ id_e (complaints_temp)
complaints_temp ← π P_id, Complain_date, Comments, Emp_id → E_id
(complaints_temp)
complaints_final = complaints_temp ∪ σ ¬(P_id = id) (complaints)

*Note -> The update operation is performed using **'->'** this sign and Null value is represented by **'∅'.**

QUERIES

QUERY 1
ENTER OR UPDATE ALL THE CHOOSEN USER ITEMS INTO CART PRODUCT ITEMS

```
DROP FUNCTION IF EXISTS update_cart_product;
DELIMITER //
CREATE FUNCTION update_cart_product(id INT, prod_id INT, count_set INT) RETURNS INT
DETERMINISTIC
BEGIN
        IF EXISTS (SELECT * FROM cart_products WHERE Ca_id = id AND P_id = prod_id)
THEN
                UPDATE cart_products
                SET COUNT = count_set
                WHERE Ca_id =id AND P_id = prod_id;
        ELSE
                INSERT INTO cart_products (Ca_id, P_id, COUNT)
                SELECT id, P_id, count_set
                FROM product
                WHERE product.P_id = prod_id;
        END IF;
   return row_count();
END //
DELIMITER ;

SELECT update_cart_product(1, 7, 9);
```

QUERY 2
REMOVE A PRODUCT FROM CART PRODUCT

```
UPDATE cart_products
SET count = count - 1
WHERE ca_id = 1 AND p_id = 5 AND count > 0;

DELETE FROM cart_products
WHERE ca_id = 1 AND p_id = 5 AND count = 0;
```

QUERY 3
ENTER ALL THE CART PRODUCT ITEMS INTO CART

```
DROP PROCEDURE IF EXISTS update_cart;
DELIMITER //
```

```
CREATE PROCEDURE update_cart(c_id Int)
BEGIN
        DECLARE count1 INT;
        SELECT SUM(COUNT) INTO count1 FROM cart_products WHERE ca_id = c_id;

        IF EXISTS (SELECT * FROM cart WHERE cart.C_id = c_id) THEN
      UPDATE cart
      SET count = count + count1, T_cost = T_cost + (count1 * 5.5)
      WHERE cart.c_id=c_id;
        ELSE
                INSERT INTO cart (c_id, COUNT, T_cost)
                VALUES (c_id, count1, count1 * 5.5);
        end if;
end //
DELIMITER ;

CALL update_cart(1);
SELECT * FROM cart;
```

QUERY 4
UPDATE THE ORDER DETAILS FOR THE USER

```
DROP PROCEDURE IF EXISTS UPDATE_ORDER;
DELIMITER //
CREATE PROCEDURE UPDATE_ORDER(id INT)
BEGIN
        DECLARE tcost INT;
   SELECT T_COST INTO tcost FROM cart WHERE cart.C_id = id;
        INSERT INTO order_ (C_id, O_date, ship_date, T_Cost)
   VALUES (id, CURDATE(), DATE_ADD(CURDATE(), INTERVAL 15 DAY), tcost);
END //
DELIMITER ;
CALL UPDATE_ORDER(1);
SELECT * FROM order_;
```

QUERY 6
PAYMENT PROCEDURE FOR THE ORDER DETAILS

```
DROP PROCEDURE IF EXISTS payment_update;
DELIMITER //
CREATE PROCEDURE payment_update(id INT)
BEGIN
        INSERT INTO payment (C_id, O_id, total_cost, P_mode)
```

```
    SELECT order_.C_id, order_.O_id, order_.T_cost, "UPI"
    FROM order_
    WHERE order_.C_id = id
          ORDER BY order_.O_id DESC
    LIMIT 1;
END //
DELIMITER ;

CALL payment_update(1);
SELECT * FROM payment;


QUERY 6
UPDATE SHIPPING DETAILS
DROP PROCEDURE IF EXISTS update_shipping_details;
DELIMITER //
CREATE PROCEDURE update_shipping_details(id INT)
BEGIN
          INSERT INTO shipping_details (P_id, pos)
          SELECT cart_products.P_id, "SHIPPING STARTED"
          FROM cart_products
          WHERE cart_products.Ca_id = id;
END //
DELIMITER ;

CALL update_shipping_details(1);
SELECT * FROM shipping_details;



QUERY 7
GET MAXIMUM PRICED ITEM FROM EACH TYPE IN CATALOGUE
SELECT *
FROM product p1
JOIN (
  SELECT p_type, MAX(price) AS max_price
  FROM product
  GROUP BY p_type
) p2 ON p1.p_type = p2.p_type AND p1.price = p2.max_price;

QUERY 8
GET MINIMUM PRICED ITEM FROM EACH TYPE IN CATALOGUE
SELECT *
FROM product p1
```

```
JOIN (
  SELECT p_type, MIN(price) AS min_price
  FROM product
  GROUP BY p_type
) p2 ON p1.p_type = p2.p_type AND p1.price = p2.min_price;
```

QUERY 9
AFTER CONFIRMING ORDER THE QUANTITY OF PRODUCTS DECREASES IN CATALOGUE

```
DROP PROCEDURE IF EXISTS quantity_change;
DELIMITER //
CREATE PROCEDURE quantity_change(id INT)
BEGIN
        IF EXISTS (SELECT * FROM product, cart_products WHERE cart_products.Ca_id=id
AND cart_products.P_id=product.P_id)
        THEN
                UPDATE product, cart_products
      SET product.S_quantity = product.S_quantity - cart_products.COUNT
      WHERE cart_products.ca_id=id AND product.P_id=cart_products.P_id;
        END IF;
END //
DELIMITER ;

CALL quantity_change(1);
SELECT * FROM product;
```

QUERY 10
FILE A COMPLAINT AND ASSIGN AN EMPLOYEE FOR THE SAME COMPLAINT

```
DROP PROCEDURE IF EXISTS file_complaint;

DELIMITER //
CREATE PROCEDURE file_complaint(id INT)

BEGIN
        DECLARE id_e INT;
        SELECT Emp_id INTO id_e FROM employees ORDER BY RAND() LIMIT 1;
   INSERT INTO complaints (P_id, Complain_date, Comments, E_id)
   VALUES (id, CURDATE(), 'PRODUCT WAS NOT GOOD ENOUGH.', id_e);
END //
DELIMITER ;
```

```
CALL file_complaint(1);
SELECT * FROM complaints;
```