**Mobile Shop Management System Project Report**

**Introduction:**

The mobile shop management system is a Database Management System that helps mobile shop owners manage their inventory, sales, and customer data. The system typically includes modules for managing the following:

* **Inventory:** The inventory module allows the shop owner to track the stock levels of all mobile phones in the store. This information can be used to ensure that the shop has enough stock to meet customer demand and to identify which phones are selling well and which are not.
* **Sales:** The sales module allows the shop owner to track the sale of mobile phones. This information can be used to generate reports on sales trends, identify top-selling phones, and track the performance of individual salespersons.
* **Customers:** The customer module allows the shop owner to store customer information, such as name, address, and phone number. This information can be used to target marketing campaigns and to provide personalized customer service.

**Project Objectives:**

The objectives of this project were to develop a mobile shop management system that would:

* Be easy to use and navigate
* Provide shop owners with the information they need to manage their inventory, sales, and customer data
* Be flexible enough to accommodate the needs of different types of mobile shops

**Project Methodology:**

The project was developed using the following methodology:

1. **Requirements gathering:** The first step was to gather requirements from potential users of the system. This involved interviewing mobile shop owners and salespersons to identify their needs and pain points.
2. **System design:** Based on the gathered requirements, a system design was created. This design outlined the architecture of the system, the functional requirements of each module, and the data that would be stored in the system.
3. **Development:** The system was developed using PostgreSQL. The development process involved creating the database schema, writing the code for each module, and testing the system for functionality and performance.

***Functional Requirement***

1. Supplier Management:
   * The system should allow the addition, modification, and deletion of supplier records in the Supplier table.
   * A supplier must have a unique supplier\_id as the primary key.
   * The system should store and manage supplier contact details such as email, phone number, and name (first name, middle name, last name).
2. Mobile Management:
   * The system should support the addition, modification, and deletion of mobile device records in the Mobile table.
   * Mobile devices have attributes like price, brand, model, RAM, ROM, battery, processor, screen size, operating system, and display quantity.
3. Stock Management:
   * The system should manage stock records in the Stock table, including purchase date, quantity, and linking to a specific supplier and mobile device.
   * The quantity field in the Stock table should be updated based on purchases.
4. Payment Management:
   * The system should handle payment transactions in the Payment table, recording payment date, amount, payment mode, and linking to a specific supplier and stock.
   * The total amount paid by a supplier for a particular stock should be calculated and updated in the Payment table.
5. Employee Management:
   * The system should allow the addition, modification, and deletion of employee records in the Employee table.
   * Each employee is identified by a unique employee\_id and has attributes like name, salary, joining date, and role.
6. Warranty Management:
   * The system should manage warranty details in the Warranty table, including type, duration, and starting date.
   * Each warranty is identified by a unique warranty\_id.
7. Customer Management:
   * The system should handle customer records in the Customer table, including name, address, email, and phone number.
   * Each customer is identified by a unique customer\_id.
8. State Pincode Information:
   * The system should store state and city information corresponding to pin codes in the StatePincodeInfo table.
   * Each pin code should be unique.
9. Order Management:
   * The system should manage customer orders in the OrderTable, including quantity, ordered date, and linking to a specific mobile device, customer, employee, and warranty.
   * The quantity in the Stock table should be updated based on orders.
10. Bill Management:
    * The system should generate bills in the Bill table, including total amount, date, and GST number.
    * Each bill should be linked to a specific customer order.

These functional requirements provide an overview of the essential features and capabilities the database support. They serve as a foundation for developing and implementing the database in a project.

***DDL Queries***

-- Table 1: Supplier Table

CREATE TABLE Supplier (

supplier\_id NUMERIC PRIMARY KEY,

email VARCHAR(255),

phoneno NUMERIC(15),

fname VARCHAR(50),

mname VARCHAR(50),

lname VARCHAR(50)

);

-- Table 2: Mobile Table

CREATE TABLE Mobile (

mobile\_id NUMERIC PRIMARY KEY,

price NUMERIC(10,2),

brand VARCHAR(50),

model VARCHAR(50),

RAM NUMERIC,

ROM NUMERIC,

battery VARCHAR(50),

processor VARCHAR(50),

screen\_size NUMERIC(4,2),

operating\_system VARCHAR(50),

display\_quantity NUMERIC

);

-- Table 3: Stock Table

CREATE TABLE Stock (

stock\_id NUMERIC PRIMARY KEY,

mobile\_id NUMERIC,

purchase\_date DATE,

quantity NUMERIC,

supplier\_id NUMERIC,

FOREIGN KEY (supplier\_id) REFERENCES stock(supplier\_id),

FOREIGN KEY (mobile\_id) REFERENCES Mobile(mobile\_id)

);

-- Table 4: Payment Table

CREATE TABLE Payment (

payment\_id NUMERIC PRIMARY KEY,

payment\_date DATE,

payment\_amount DECIMAL(10,2),

payment\_mode VARCHAR(50),

supplier\_id NUMERIC,

stock\_id NUMERIC,

FOREIGN KEY (supplier\_id) REFERENCES Supplier,

FOREIGN KEY (stock\_id) REFERENCES Stock

);

-- Table 5: Employee Table

CREATE TABLE Employee (

employee\_id NUMERIC PRIMARY KEY,

name VARCHAR(100),

salary NUMERIC(10,2),

joining\_date DATE,

role VARCHAR(50)

);

-- Table 6: Customer Table

CREATE TABLE Customer (

customer\_id NUMERIC PRIMARY KEY,

fname VARCHAR(50),

mname VARCHAR(50),

lname VARCHAR(50),

street VARCHAR(255),

city VARCHAR(50),

email VARCHAR(255),

phone\_no NUMERIC

);

-- Table 7: Warranty Table

CREATE TABLE Warranty (

warranty\_id NUMERIC PRIMARY KEY,

type VARCHAR(50),

duration NUMERIC,

starting\_date DATE

);

-- Table 8: State Pincode Info Table

CREATE TABLE StatePincodeInfo (

pincode NUMERIC PRIMARY KEY,

city VARCHAR(50),

state VARCHAR(50)

);

-- Table 9: Order Table

CREATE TABLE OrderTable (

order\_id NUMERIC PRIMARY KEY,

quantity NUMERIC,

mobile\_id NUMERIC,

customer\_id NUMERIC,

ordered\_date DATE,

employee\_id NUMERIC,

warranty\_id NUMERIC,

FOREIGN KEY (mobile\_id) REFERENCES Mobile(mobile\_id),

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id),

FOREIGN KEY (employee\_id) REFERENCES Employee(employee\_id),

FOREIGN KEY (warranty\_id) REFERENCES Warranty(warranty\_id)

);

-- Table 10: Bill Table

CREATE TABLE Bill (

bill\_id NUMERIC PRIMARY KEY,

total\_amount NUMERIC(10,2),

date DATE,

gst\_no VARCHAR(50),

order\_id NUMERIC,

customer\_id NUMERIC,

FOREIGN KEY (order\_id) REFERENCES OrderTable(order\_id),

FOREIGN KEY (customer\_id) REFERENCES Customer(customer\_id)

);

***SQL Queries***

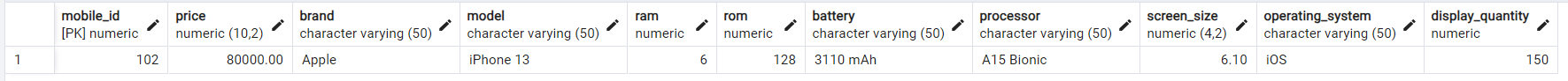
-- 1 : Retrieve the details of the most expensive mobile:

SELECT \*

FROM MOBILE

ORDER BY PRICE DESC

LIMIT 1;

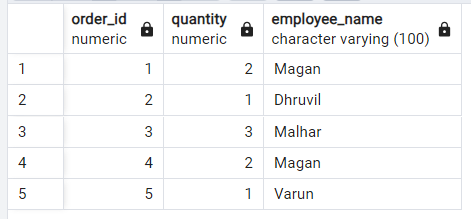


-- 2 : Retrieve the order details with the corresponding employee names.

SELECT O.ORDER\_ID, O.QUANTITY, E.NAME AS EMPLOYEE\_NAME

FROM ORDERTABLE O

JOIN EMPLOYEE E ON O.EMPLOYEE\_ID = E.EMPLOYEE\_ID;



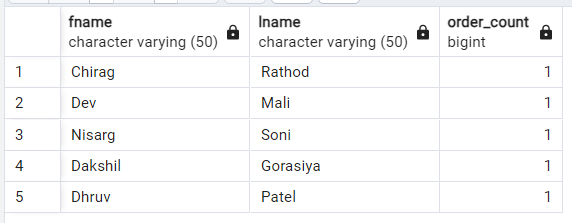
-- 3 : Retrieve the number of orders placed by each customer.

SELECT C.FNAME, C.LNAME, COUNT(O.ORDER\_ID) AS ORDER\_COUNT

FROM CUSTOMER C

JOIN ORDERTABLE O ON C.CUSTOMER\_ID = O.CUSTOMER\_ID

GROUP BY C.CUSTOMER\_ID;



-- 4 : List the mobiles with their brand and model having a price greater than 30000.

SELECT BRAND, MODEL

FROM MOBILE

WHERE PRICE > 30000;



-- 5 : Show the customers who have placed orders with a total amount greater than 50000.

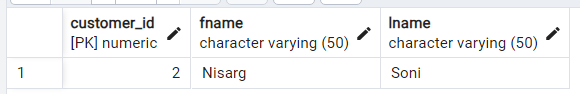
SELECT DISTINCT C.CUSTOMER\_ID, C.FNAME, C.LNAME

FROM CUSTOMER C

JOIN ORDERTABLE O ON C.CUSTOMER\_ID = O.CUSTOMER\_ID

JOIN BILL B ON O.ORDER\_ID = B.ORDER\_ID

WHERE B.TOTAL\_AMOUNT > 50000;



-- 6 : List the customers who have ordered a mobile with a warranty duration of 24 months.

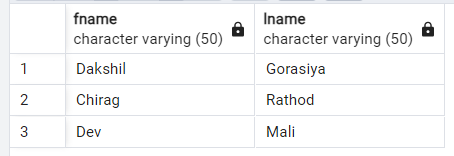
SELECT C.FNAME, C.LNAME

FROM CUSTOMER C

JOIN ORDERTABLE O ON C.CUSTOMER\_ID = O.CUSTOMER\_ID

JOIN WARRANTY W ON O.WARRANTY\_ID = W.WARRANTY\_ID

WHERE W.DURATION = 24;

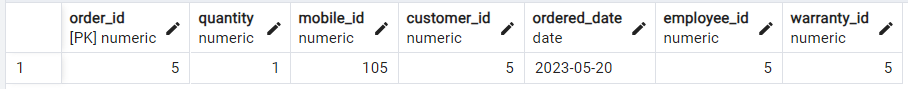


-- 7 : Retrieve the orders placed in May 2023.

SELECT \*

FROM ORDERTABLE

WHERE ORDERED\_DATE BETWEEN '2023-05-01' AND '2023-05-31';



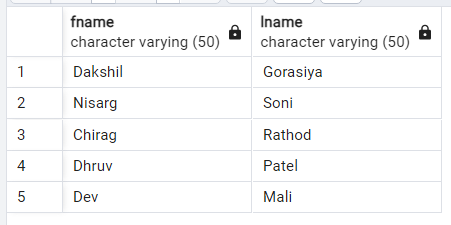
-- 8 : Display the customers who live in Gujarat.

SELECT C.FNAME, C.LNAME

FROM CUSTOMER C

JOIN STATEPINCODEINFO SPI ON C.CITY = SPI.CITY

WHERE SPI.STATE = 'Gujarat';



-- 9 : List the customers who have ordered mobiles with a screen size greater than 6.0 inches.

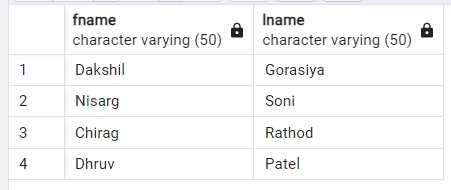
SELECT C.FNAME, C.LNAME

FROM CUSTOMER C

JOIN ORDERTABLE O ON C.CUSTOMER\_ID = O.CUSTOMER\_ID

JOIN MOBILE M ON O.MOBILE\_ID = M.MOBILE\_ID

WHERE M.SCREEN\_SIZE > 6.0;



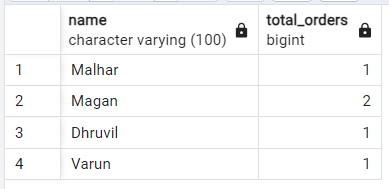
-- 10 : Display the total number of orders placed by each employee.

SELECT E.NAME, COUNT(O.ORDER\_ID) AS TOTAL\_ORDERS

FROM EMPLOYEE E

JOIN ORDERTABLE O ON E.EMPLOYEE\_ID = O.EMPLOYEE\_ID

GROUP BY E.EMPLOYEE\_ID;



***Functions***

-- 1 : Function to Calculate Total Stock Value by Mobile Brand

CREATE OR REPLACE FUNCTION calculate\_stock\_value\_by\_brand(brand\_name VARCHAR(50))

RETURNS NUMERIC(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_value NUMERIC(10, 2);

BEGIN

SELECT SUM(price \* quantity) INTO total\_value

FROM Stock

JOIN Mobile ON Stock.mobile\_id = Mobile.mobile\_id

WHERE brand = brand\_name;

RETURN total\_value;

END;

$$

select calculate\_stock\_value\_by\_brand('Apple');

-- 2 : Function to Calculate Total Salary Expense by Role

CREATE OR REPLACE FUNCTION calculate\_salary\_expense\_by\_role(role\_name VARCHAR(50))

RETURNS NUMERIC(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_salary NUMERIC(10, 2);

BEGIN

SELECT SUM(salary) INTO total\_salary

FROM Employee

WHERE role = role\_name;

RETURN total\_salary;

END;

$$

select calculate\_sal

-- 3 : Function to Retrieve Order Details by ID

CREATE OR REPLACE FUNCTION get\_order\_details(orderid NUMERIC)

RETURNS TABLE (

quantity NUMERIC,

mobile\_brand VARCHAR(50),

f\_name VARCHAR(150),

l\_name VARCHAR(150),

ordered\_date DATE

)

LANGUAGE plpgsql

AS $$

BEGIN

RETURN QUERY SELECT o.quantity, m.brand, c.fname, c.lname, o.ordered\_date

FROM OrderTable o

JOIN Mobile m ON o.mobile\_id = m.mobile\_id

JOIN Customer c ON o.customer\_id = c.customer\_id

WHERE order\_id = orderid;

END;

$$

select \* from get\_order\_details(1);

-- 4 : Function to Calculate Total Revenue by Date Range

CREATE OR REPLACE FUNCTION calculate\_total\_revenue\_by\_date\_range(start\_date DATE, end\_date DATE)

RETURNS NUMERIC(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_revenue NUMERIC(10, 2);

BEGIN

SELECT SUM(total\_amount) INTO total\_revenue

FROM Bill

WHERE date BETWEEN start\_date AND end\_date;

RETURN total\_revenue;

END;

$$

select calculate\_total\_revenue\_by\_date\_range('2023-03-01','2023-03-28');

-- 5 : Function to Retrieve Stock Quantity by Mobile ID

CREATE OR REPLACE FUNCTION get\_stock\_quantity\_by\_mobile\_id(mobileid NUMERIC)

RETURNS NUMERIC

LANGUAGE plpgsql

AS $$

DECLARE

stock\_quantity NUMERIC;

BEGIN

SELECT SUM(quantity) INTO stock\_quantity

FROM Stock

WHERE Stock.mobile\_id = mobileid;

RETURN stock\_quantity;

END;

$$

select get\_stock\_quantity\_by\_mobile\_id(101);

-- 6 : Function to Calculate Total Payment Amount by Payment Mode

CREATE OR REPLACE FUNCTION calculate\_total\_payment\_amount\_by\_mode(paymentMode VARCHAR(50))

RETURNS DECIMAL(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_amount DECIMAL(10, 2);

BEGIN

SELECT SUM(payment\_amount) INTO total\_amount

FROM Payment

WHERE Payment.payment\_mode = paymentMode;

RETURN total\_amount;

END;

$$

select calculate\_total\_payment\_amount\_by\_mode('Cash');

-- 7 : Function to Retrieve State for a Given Pincode

CREATE OR REPLACE FUNCTION get\_state\_by\_pincode(pc NUMERIC)

RETURNS VARCHAR(50)

LANGUAGE plpgsql

AS $$

DECLARE

state\_name VARCHAR(50);

BEGIN

SELECT state INTO state\_name

FROM StatePincodeInfo

WHERE pincode = pc;

RETURN state\_name;

END;

$$

select get\_state\_by\_pincode(380001);

-- 8 : Function to Caculate Total Revenue by Employee

CREATE OR REPLACE FUNCTION calculate\_total\_revenue\_by\_employee(employeeid NUMERIC)

RETURNS NUMERIC(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_revenue NUMERIC(10, 2);

BEGIN

SELECT SUM(total\_amount) INTO total\_revenue

FROM Bill

JOIN OrderTable ON Bill.order\_id = OrderTable.order\_id

WHERE OrderTable.employee\_id = employeeid;

RETURN total\_revenue;

END;

$$

select calculate\_total\_revenue\_by\_employee(1);

-- 9 : Function to Retrieve Mobiles with Low Stock

CREATE OR REPLACE FUNCTION get\_low\_stock\_mobiles(threshold\_quantity NUMERIC)

RETURNS TABLE (

mobile\_id NUMERIC,

brand VARCHAR(50),

model VARCHAR(50),

current\_stock NUMERIC

)

LANGUAGE plpgsql

AS $$

BEGIN

RETURN QUERY SELECT M.mobile\_id, M.brand, M.model, S.quantity

FROM Mobile M

JOIN Stock S ON M.mobile\_id = S.mobile\_id

WHERE S.quantity < threshold\_quantity;

END;

$$

select \* from get\_low\_stock\_mobiles(30);

-- 10 : Function to Calculate Total Revenue by City

CREATE OR REPLACE FUNCTION calculate\_total\_revenue\_by\_city(ct VARCHAR(50))

RETURNS NUMERIC(10, 2)

LANGUAGE plpgsql

AS $$

DECLARE

total\_revenue NUMERIC(10, 2);

BEGIN

SELECT SUM(total\_amount) INTO total\_revenue

FROM Bill

JOIN OrderTable ON Bill.order\_id = OrderTable.order\_id

JOIN Customer ON OrderTable.customer\_id = Customer.customer\_id

WHERE Customer.city = ct;

RETURN total\_revenue;

END;

$$

select calculate\_total\_revenue\_by\_city('Anand');

*Procedure*

-- 1 : Procedure to Update Employee Salary

CREATE OR REPLACE PROCEDURE update\_employee\_salary(

IN employeeid NUMERIC,

IN new\_salary NUMERIC(10, 2)

)

AS $$

BEGIN

UPDATE Employee

SET salary = new\_salary

WHERE Employee.employee\_id = employeeid;

END;

$$ LANGUAGE plpgsql;

CALL update\_employee\_salary(1,50000);

select salary from employee where employee\_id = 1;

-- 2 : Procedure to Update Stock Quantity for a Mobile

CREATE OR REPLACE PROCEDURE update\_stock\_quantity(

IN mobileid NUMERIC,

IN new\_quantity NUMERIC

)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE Stock

SET quantity = quantity + new\_quantity

WHERE Stock.mobile\_id = mobileid;

END;

$$

CALL update\_stock\_quantity(102, 20);

select quantity from stock where mobile\_id = 102;

-- 3 : Procedure to Insert Employee

CREATE OR REPLACE PROCEDURE insert\_employee(

IN in\_employee\_id NUMERIC,

IN in\_name VARCHAR(100),

IN in\_salary NUMERIC(10,2),

IN in\_joining\_date DATE,

IN in\_role VARCHAR(20)

)

LANGUAGE plpgsql

AS $$

BEGIN

INSERT INTO employee (employee\_id, name, salary, joining\_date, role)

VALUES (in\_employee\_id, in\_name, in\_salary, in\_joining\_date, in\_role);

END;

$$

CALL insert\_employee(6, 'Tirth', 15000, '2023-11-23', 'Salesperson');

select \* from employee;

-- 4 : Procedure to Delete Expired Warranties

CREATE OR REPLACE PROCEDURE delete\_expired\_warranties()

LANGUAGE plpgsql

AS $$

BEGIN

DELETE FROM Warranty

WHERE starting\_date + interval '1 month' \* duration < CURRENT\_DATE;

END $$;

CALL delete\_expired\_warranties();

-- 5 : Procedure to Update Mobile Prices with Discount

CREATE OR REPLACE PROCEDURE update\_mobile\_prices\_with\_discount(

IN discount\_percentage NUMERIC

)

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE mobile

SET price = price - (price \* discount\_percentage / 100);

END $$;

CALL update\_mobile\_prices\_with\_discount(10);

***Cursor***

-- 1 : Cursor to Retrieve Customer Information

CREATE OR REPLACE FUNCTION Cursor\_to\_get\_customer\_info()

RETURNS SETOF Customer

LANGUAGE plpgsql

AS $$

DECLARE

customer\_cursor CURSOR FOR SELECT \* FROM Customer;

customer\_record Customer;

BEGIN

OPEN customer\_cursor;

LOOP

FETCH customer\_cursor INTO customer\_record;

EXIT WHEN NOT FOUND;

RETURN NEXT customer\_record;

END LOOP;

CLOSE customer\_cursor;

RETURN;

END;

$$

SELECT \* FROM Cursor\_to\_get\_customer\_info();

-- 2 : Cursor to Fetch Employees with Role 'Salesperson'

CREATE OR REPLACE FUNCTION Cursor\_to\_process\_salespersons()

RETURNS SETOF Employee

LANGUAGE plpgsql

AS $$

DECLARE

salespersons\_cursor CURSOR FOR SELECT \* FROM Employee WHERE role = 'Salesperson';

employee\_record Employee%ROWTYPE;

BEGIN

OPEN salespersons\_cursor;

LOOP

FETCH salespersons\_cursor INTO employee\_record;

EXIT WHEN NOT FOUND;

RETURN NEXT employee\_record;

END LOOP;

CLOSE salespersons\_cursor;

RETURN;

END;

$$;

SELECT \* FROM Cursor\_to\_process\_salespersons();

-- 3 : Cursor to Fetch Customers in Rajkot

CREATE OR REPLACE FUNCTION Cursor\_to\_process\_rajkot\_customers()

RETURNS SETOF Customer

LANGUAGE plpgsql

AS $$

DECLARE

rajkot\_customers\_cursor CURSOR FOR SELECT \* FROM Customer WHERE city = 'Rajkot';

customer\_record Customer%ROWTYPE;

BEGIN

OPEN rajkot\_customers\_cursor;

LOOP

FETCH rajkot\_customers\_cursor INTO customer\_record;

EXIT WHEN NOT FOUND;

RETURN NEXT customer\_record;

END LOOP;

CLOSE rajkot\_customers\_cursor;

RETURN;

END;

$$;

SELECT \* FROM Cursor\_to\_process\_rajkot\_customers();

-- 4 : Cursor to Fetch Stock Purchased in 2023

CREATE OR REPLACE FUNCTION Cursor\_to\_process\_stock\_2023()

RETURNS SETOF Stock

LANGUAGE plpgsql

AS $$

DECLARE

stock\_2023\_cursor CURSOR FOR SELECT \* FROM Stock WHERE EXTRACT(YEAR FROM purchase\_date) = 2023;

stock\_record Stock%ROWTYPE;

BEGIN

OPEN stock\_2023\_cursor;

LOOP

FETCH stock\_2023\_cursor INTO stock\_record;

EXIT WHEN NOT FOUND;

RETURN NEXT stock\_record;

END LOOP;

CLOSE stock\_2023\_cursor;

RETURN;

END;

$$;

SELECT \* FROM Cursor\_to\_process\_stock\_2023();

-- 5 : Cursor to Fetch Orders Placed in May 2023:

CREATE OR REPLACE FUNCTION Cursor\_to\_process\_may\_2023\_orders()

RETURNS SETOF OrderTable

LANGUAGE plpgsql

AS $$

DECLARE

may\_orders\_cursor CURSOR FOR SELECT \* FROM OrderTable WHERE ordered\_date BETWEEN '2023-05-01' AND '2023-05-31';

order\_record OrderTable%ROWTYPE;

BEGIN

OPEN may\_orders\_cursor;

LOOP

FETCH may\_orders\_cursor INTO order\_record;

EXIT WHEN NOT FOUND;

RETURN NEXT order\_record;

END LOOP;

CLOSE may\_orders\_cursor;

RETURN;

END;

$$;

SELECT \* FROM Cursor\_to\_process\_may\_2023\_orders();

***Trigger***

-- 1 : Trigger to Update Stock Quantity After an Order

CREATE OR REPLACE FUNCTION update\_stock\_quantity\_after\_order()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

UPDATE Stock

SET quantity = quantity - NEW.quantity

WHERE mobile\_id = NEW.mobile\_id;

RETURN NEW;

END;

$$

CREATE TRIGGER update\_stock\_quantity

AFTER INSERT ON ordertable

FOR EACH ROW

EXECUTE FUNCTION update\_stock\_quantity\_after\_order();

insert into ordertable values(100,40,102,1,'2023-01-01',1,1);

select \* from ordertable;

select \* from stock;

-- 2 : Trigger to Check Email Format in Customer Table

CREATE OR REPLACE FUNCTION check\_customer\_email\_format()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

IF NEW.email !~ '^[a-zA-Z0-9.\_%-]+@[a-zA-Z0-9.\_%-]+\.[a-zA-Z]{2,4}$' THEN

RAISE EXCEPTION 'Invalid email format.';

END IF;

RETURN NEW;

END;

$$

CREATE TRIGGER check\_customer\_email\_format\_trigger

BEFORE INSERT OR UPDATE ON Customer

FOR EACH ROW

EXECUTE FUNCTION check\_customer\_email\_format();

insert into customer(customer\_id, email) values(100,'error');

-- 3 : Trigger to Prevent Negative Stock Quantity

CREATE OR REPLACE FUNCTION prevent\_negative\_stock\_quantity()

RETURNS TRIGGER AS $$

BEGIN

IF NEW.quantity < 0 THEN

RAISE EXCEPTION 'Stock quantity cannot be negative.';

END IF;

RETURN NEW;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER check\_negative\_stock\_quantity

BEFORE INSERT OR UPDATE ON Stock

FOR EACH ROW

EXECUTE FUNCTION prevent\_negative\_stock\_quantity();

insert into stock values(999,101,'2020-11-12',-8);

-- 4 : Trigger to Prevent Deleting Suppliers with Active Stock

CREATE OR REPLACE FUNCTION prevent\_delete\_supplier()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

IF TG\_OP = 'DELETE' THEN

-- Check if there are active stocks for the supplier

IF EXISTS (

SELECT 1

FROM Stock

WHERE OLD.supplier\_id = Stock.supplier\_id

) THEN

RAISE EXCEPTION 'Cannot delete supplier with active stock';

END IF;

END IF;

RETURN OLD;

END $$;

CREATE TRIGGER prevent\_delete\_supplier\_trigger

BEFORE DELETE ON Supplier

FOR EACH ROW

EXECUTE FUNCTION prevent\_delete\_supplier();

DELETE FROM Supplier WHERE supplier\_id = 1;

-- 5 : Trigger to Enforce Unique Email

CREATE OR REPLACE FUNCTION enforce\_unique\_email()

RETURNS TRIGGER

LANGUAGE plpgsql

AS $$

BEGIN

IF EXISTS (SELECT 1 FROM customer WHERE email = NEW.email) THEN

RAISE EXCEPTION 'Email must be unique.';

END IF;

RETURN NEW;

END;

$$

CREATE TRIGGER enforce\_unique\_email\_trigger

BEFORE INSERT ON customer

FOR EACH ROW

EXECUTE FUNCTION enforce\_unique\_email();

INSERT INTO customer(customer\_id, email)

VALUES (100, 'dakshilgorasiya@gmail.com');

**Project Results:**

* The mobile shop management system was successfully developed and meets all of the project objectives.
* The mobile shop management system has the potential to significantly impact the operations and profitability of mobile shops.
* By streamlining inventory management, enhancing sales tracking, and fostering customer relationships, the system can contribute to increased sales, reduced costs, and improved customer satisfaction.

**Conclusion:**

The mobile shop management system is a valuable tool for mobile shop owners. The system can help shop owners to improve their efficiency, productivity, and profitability.