COMSATS University Islamabad, Lahore Campus

Block–C, Department of Computer Science

COMSATS University Islamabad, Lahore Campus

Defence Road, Off Raiwind Road, Lahore

CSC371: Database Systems

Section: A

Assignment No. 4

Team Members Registration ID:

SP21-BCS-045-A-Abdul Hadi Farooq

SP21-BCS-015-A-Muhammad Ayan

SP21-BCS-065-A-Najam ul Hasan

Submitted To: Dr. Hamid Turab

Table of Contents

Abstract

This proposal report is written for ‘Super Shoe Store’, a shoe store. This report highlights the working of the store and all of its aspects.

For managing stock, suppliers and the customers, based on all the requirements. A relational database is proposed here along with specifications and suggestions.

Domain description

Super Shoe Store is a small shoe store selling shoes to the customers for 5 years and shifted to online selling since last 2 years; the store is quite new in its online selling but they are quite experienced in on store selling. The store has following departments

• Sales Department, in Lahore;

• Stock Department, in Lahore;

• Delivery Department, in Lahore;

Whenever a customer places an order it is managed and listed by sales department which then send it to Stock department which checks the availability of the product, if the product is available in specified quantity then order is processed and order is delivered by the delivery service of store otherwise product order is not processed and unavailability is notified to the customer. The inventory supply and suppliers information is also managed in stock department. After order delivery, the cash is taken on delivery.

Current Situation

The majority of the information is now handled manually, which takes a lot of time, looks complex where it is tough to enter and retrieve data. Currently there is no proper database system , just a billing machine which prints bill of the items entered in real time. The stock and accounts are managed on paper-based register or files.

The sales department are working on one mobile phone and one laptop. On laptop the salesperson checks the orders after the order is placed, then manually the products included in the order are checked in the stock and the customer is notified whether the order is processed or the product is unavailable. If it is available, the customer gets a confirmation call and the item is set for delivery by the store. The supply by the suppliers arrives monthly and it is recorded monthly. This system is inefficient, , inaccurate and unsustainable in the long run.

Requirement Specification

To simplify the sales, stocks and delivery, they are divided into furthermore subprocesses in this study, namely:

Placement:

The process of order placement by the customer from the sales is considered as Placement for this report.

Inventory:

Once the monthly shoe stock is received it is stored accordingly in warehouse, this process is termed as Inventory throughout the report.

Delivery:

The delivery of the product is considered as delivery for this report.

Functional Requirements

Id	Function	Entity	Priority
Placement			
1	Insert/ Update/Delete/ View of the orders placed.	Order details, Order	High
2	A list of all the orders to be processed and delivered in two working days.	Orders, Order Detail	Low
Inventory			
3	Insert/Update/Delete/View of all the Inventory.	Products, Suppliers, Categories, Shippers	High

4	View the total Available Quantity of any Category present with respect to their suppliers.	Categories, Products, Suppliers	High
5	View total stock available in inventory.	Products	High
Delivery			
6	Insert/Update/Delete/View of the Orders confirmed and set to delivery.	Orders, Order details ,Customers	High
7	A list of all the Orders delivered.	Order, Order Details,	Low

Non-Functional Requirements

• The database must be available to Sales and Stock Departments.

• Certain functions, such as insert, update, delete should only be available to the authorized staff members.

• Analytical reports shall only be available to Owner.

• Only authorized staff should be able to use the system.

• The System must be capable of coping with the projected growth of the business.

Assumptions

Placement

• All the orders placed are dealt by sales department only.

• The order and customer details are shared by sales department and delivery department, Stocks department is not concerned by it.

• The order cancellation, exchange and any refunds are dealt by sales department and delivery and stock department is updated by sales department.

Inventory

• Each category of shoes is placed in different container.

• Shoes are placed in specified storage boxes.

• All the exchanged and refunded products are added at the end of the month.

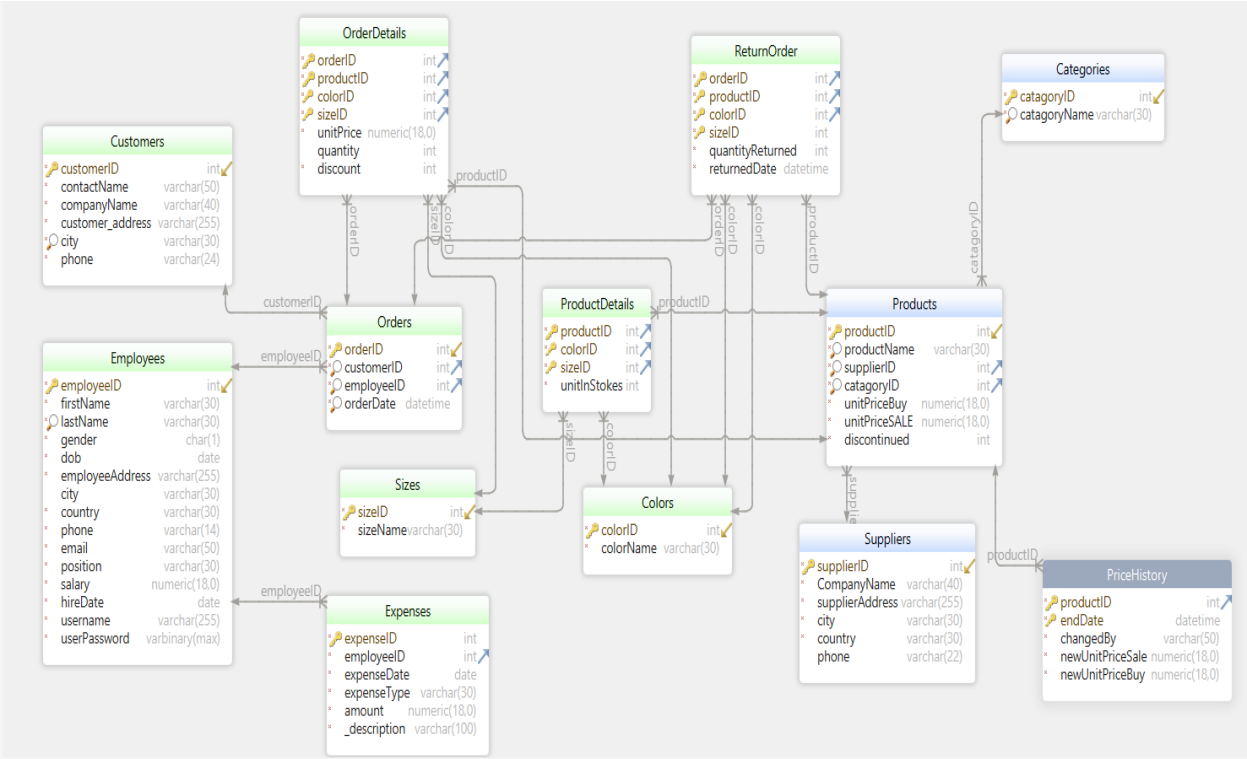
Delivery

• The delivery is performed every 4 working days.

• There is no delivery on Sundays.

• Refund at the time of delivery is not entertained.

Entity Relationship Diagram



Entities & attributes

Customer

<u>Customer-ID</u>	Varchar (5)
Contact Name	Varchar (40)
Company name	Varchar (60)
Customer address	Varchar (60)
City	Varchar (10)
phone	Varchar (15)

Customer-ID is a primary key and will be a unique code allocated to any customer. Country, city, address, postal-code and phone attributes will be used to keep information for delivery department.

Employee

Employee-ID	Integer
Last-Name	Varchar (20)
First-Name	Varchar (10)
gender	char
Birth-Date	Date
address	Varchar (60)
Hire-Date	Date

Employee will keep information relevant to employees, responsible for handling orders and those supervising other employees. Employee-ID is a primary key and will be a unique code allocated to any employee. Hire-Date attribute will allow to keep information about years of service.

Order

Order-ID	Integer
Customer-ID (FK)	Varchar (5)
Employee-ID (FK)	Integer
Order-Date	Date

Order will keep information about all orders received. Order-ID is selected as primary key. By interfacing with the database of sales department, it will allow to track any delivery up to the very detailed level of the employee who handled the order, on which order date, required date, shipped date, via which ship.

Order Detail

Order-ID (FK)	Integer
Product-ID (FK)	Integer
Color ID (FK)	Integer
Size ID (FK)	Integer
Unit-Price	Money
Quantity	Integer
Discount	Integer

Order-Detail will keep detailed information about all the orders received for one product-ID and quantity ordered. **Order-ID**, **Product-ID, color ID and size ID** will act as **composite primary key**.

Product

Product-ID	Integer

Product-Name	Varchar (40)
Supplier-ID (FK)	Integer
Category-ID (FK)	Integer
Unit-Price-buy	Money
Uni-Price-Sale	Money
Discontinued	Bit

Product will keep information about all products, their supplier, to which category they belong, buying and selling unit price. Product-ID is a primary key and will be a unique code allocated to any product.

Product Details

Product-ID	Integer
Color ID	Integer
Size ID	Integer
Units in stock	Integer

Product Details will keep information about all products details, such as their color, size, and how many units are in stock for each and every specific products size and color. Product-ID is a primary key and will be a unique code allocated to any product. Logistics department will keep track of units-on-order, so that when units-in-stock fall below reorder-level, the system will give a notification.

Return Order

Order-ID (FK)	Integer
Product-ID (FK)	Integer
Color ID (FK)	Integer
Size ID (FK)	Integer
Quantity returned	Integer
Return date	Date

Order-Detail will keep detailed information about all the orders received for one product-ID and quantity ordered. **Order-ID**, **Product-ID, color ID and size ID** will act as **composite primary key**.

Expenses

Expense ID	Integer
Employee ID (FK)	Integer
Expense date	Date
Expense type	Varchar (20)

Amount	Money
Description	Varchar (100)

Expenses will keep information about all expenses such as any maintenance costs, bills and employee salaries. This table data is used in various reports covered later on.

Price History

Product ID	Integer
End date	Date
Changed by	Varchar (50)
New unit price buy	Money
New unit price sale	Money

Price history will keep information about the change in prices of all products due to any reason such as seasonal sales. **Product ID and End date** form a composite primary key.

Supplier

Supplier-ID	Integer
Company-Name	Varchar (40)
Address	Varchar (60)
City	Varchar (15)
Country	Varchar (15)
Phone	Varchar (24)

Supplier will keep information about all suppliers, their company name and phone number. Supplier-ID is selected as primary key and will be a unique code allocated to any supplier.

Category

Category-ID	Integer
Category-Name	Varchar (15)

Category will keep information about the category to which a product belongs to. Category-ID is a primary key and will be a unique code allocated to any category. Category-Name will provide information about how the products are categorized.

Relationships & labels

Customer

- A customer can ***place many or no*** orders.
- An order must be ***placed by*** a customer.

Employee

- An employee can ***handle many or no*** orders.
- An order must be ***handled*** by an employee.

Order

- An order can ***have*** more than one order details.
- An order detail must ***belong-to*** an order.
- An order can ***have many*** order details.
- An order detail is a ***part-of*** an order.

Product

- A product can ***have many or no*** order details.
- An order detail must ***belong-to*** a product.
- A supplier must ***supply*** a product.
- A supplier can ***supply many*** products
- A product must be ***supplied by*** a supplier.
- A category can ***have many*** products.
- A product must ***belong-to*** a category.

Validation

In accordance with the user requirements, it can be clearly seen from the above description of entities and their relationships that the model is capturing all the information that is of interest to the business; hence the completeness of the conceptual model can be validated against the set of user requirements.

Since all the M:N relationships are resolved at this stage and the structure of the data is in a natural form; also while designing the conceptual model the very factor of interfacing with other systems of delivery and sales is also taken into consideration; therefore it can be validated that there exist no conflicts in the system.

User requirements relating to storage are captured by the Products, Suppliers and Categories entities. **Requirement ID 1, 2, 6 and 7** are catered by the

provision of **Order-date** of the **Orders entity**. **Requirement ID 3, 4 and 5** are catered by **units-in-stock** attributes of the **Product entity**.

Throughout the conceptual design phase Sales process has been viewed as three sub processes namely supply, categorizing products and handling orders; hence this business process break down has virtually eliminated the chances of redundancy and the repetition of any entities or concepts.

Every care has been taken to keep the data in natural form; as it is quite obvious from the use of composite primary keys instead of an artificial one. Yet again the break-down of the process into further sub processes has made the representation of the model very simple; hence it will be easy for the developer to implement resulting into a user-friendly system as a final product.

Description

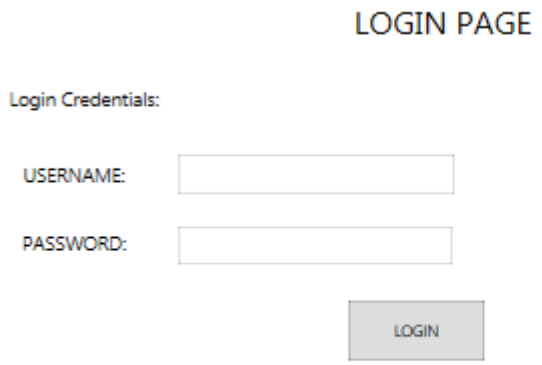
Entity Relationship Diagram illustrates the structure of the Sales section of the proposed system. It can be observed from the diagram that the Super Shoe Store's Sales department is performing three main functions. Those are of receiving the products from suppliers, categorizing the products appropriately, which is handled by stocks department, and then handling orders upon the placement of an order.

On Order it is a must requirement to record the Order ID. Since any product can have many orders and there could be many products on any one order, therefore to resolve this M:N relationship an Order Detail link entity is used, once an order is placed then comes the stage of delivering it.

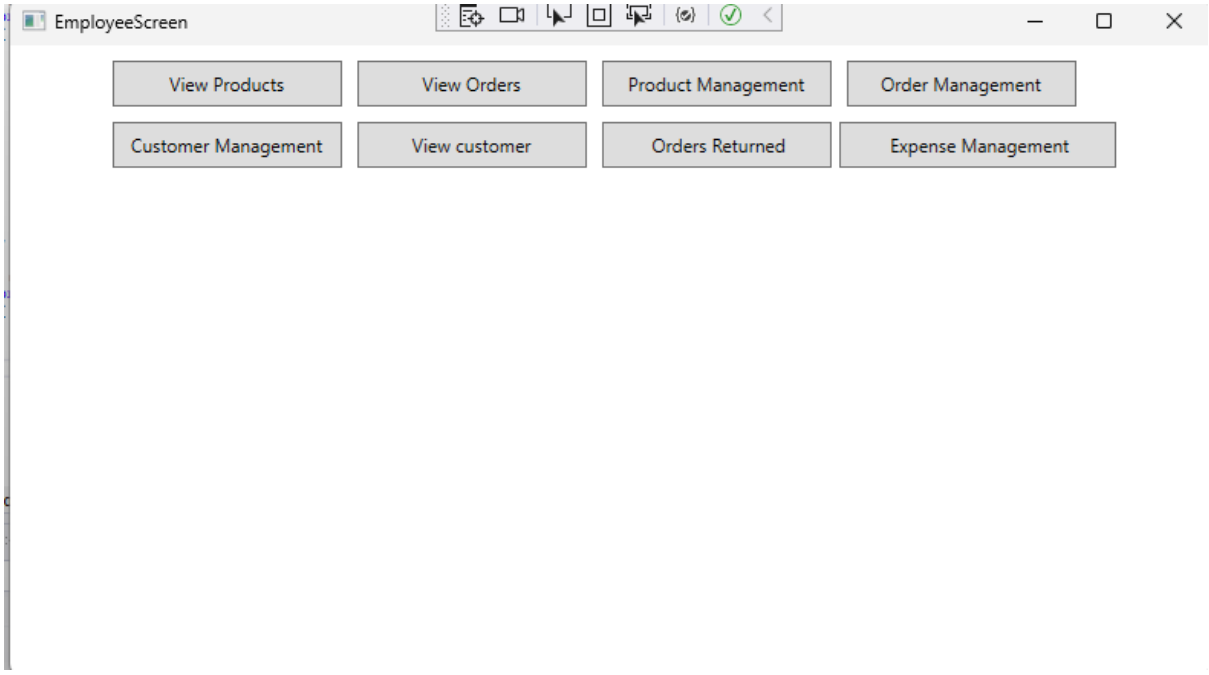
Since it is assumed that all products are unique, this makes the lookup for the exact available quantity and units-on-order very straightforward. This also allows the system to keep track of units in stock so that once it falls below reorder level, the system will give a notification. Therefore, the design allows for the update of the quantities stored, and ensures in-demand products always in stock.

Screenshots Of Developed Frontend\GUI

Login-form

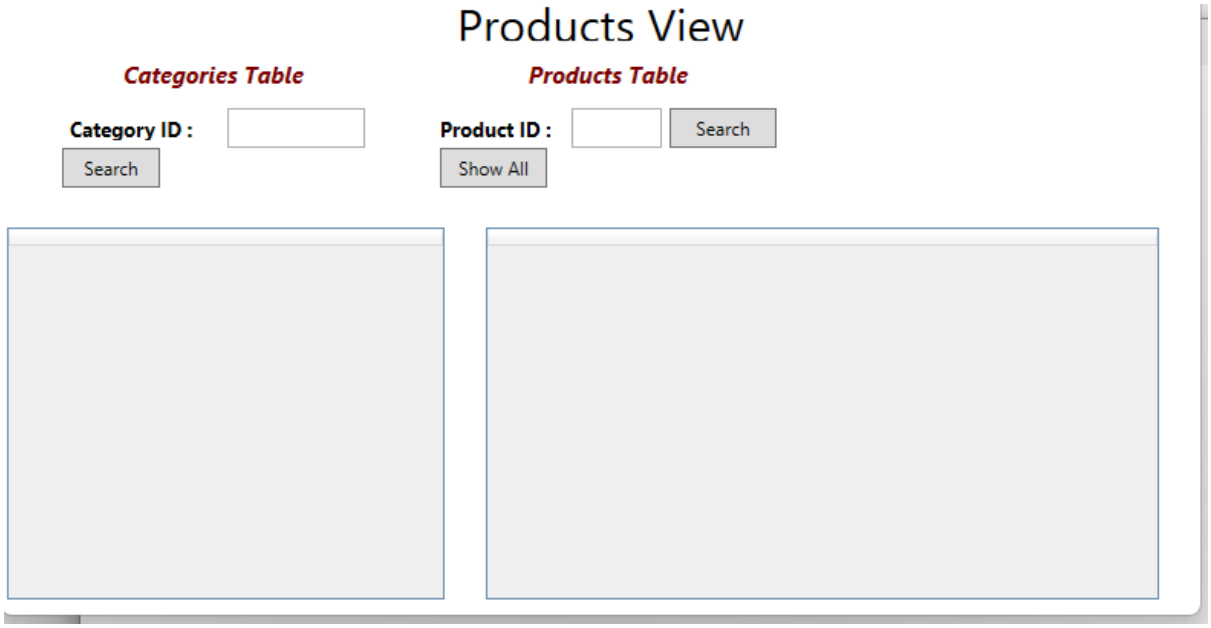


Employee Screen



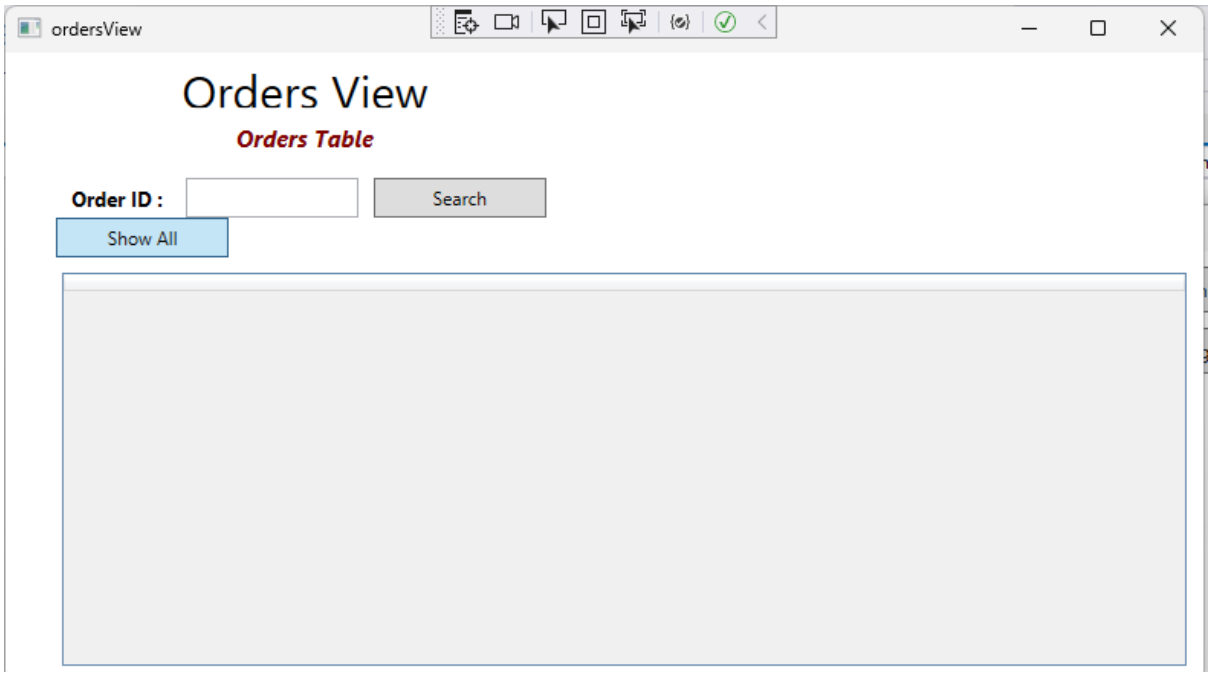
Product view

This shows only necessary information in form of views implemented at backend when view products button is pressed.



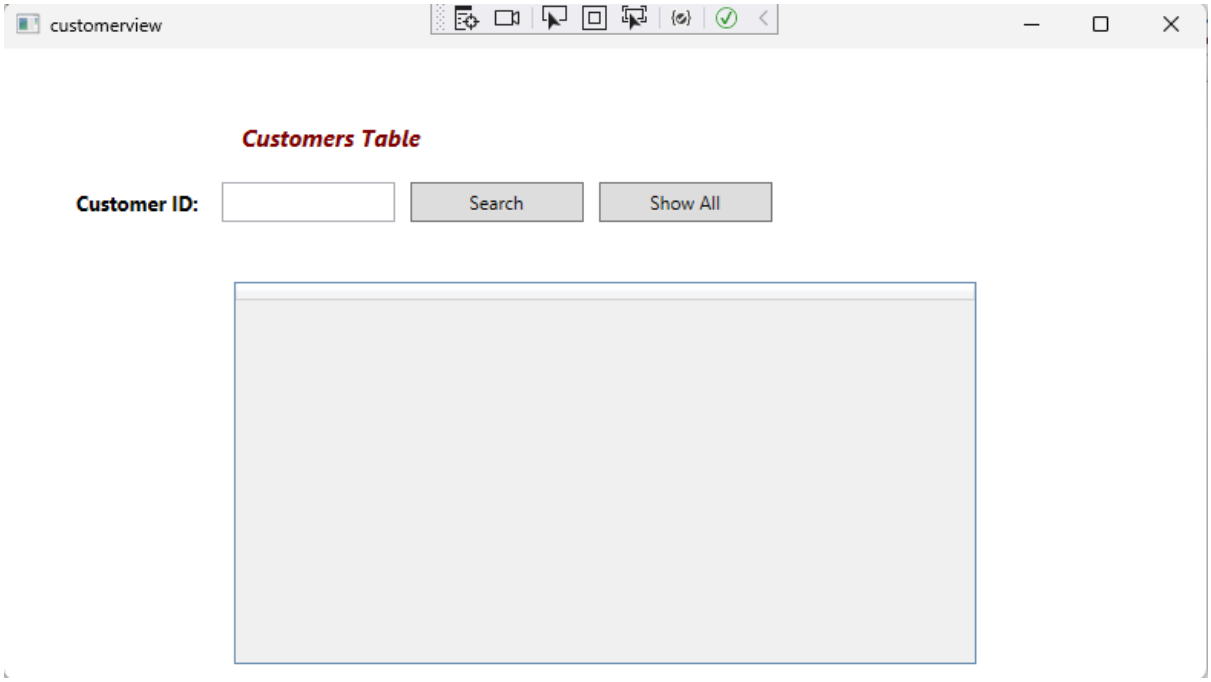
Order view

This shows only necessary information in form of views implemented at backend when view orders button is pressed.



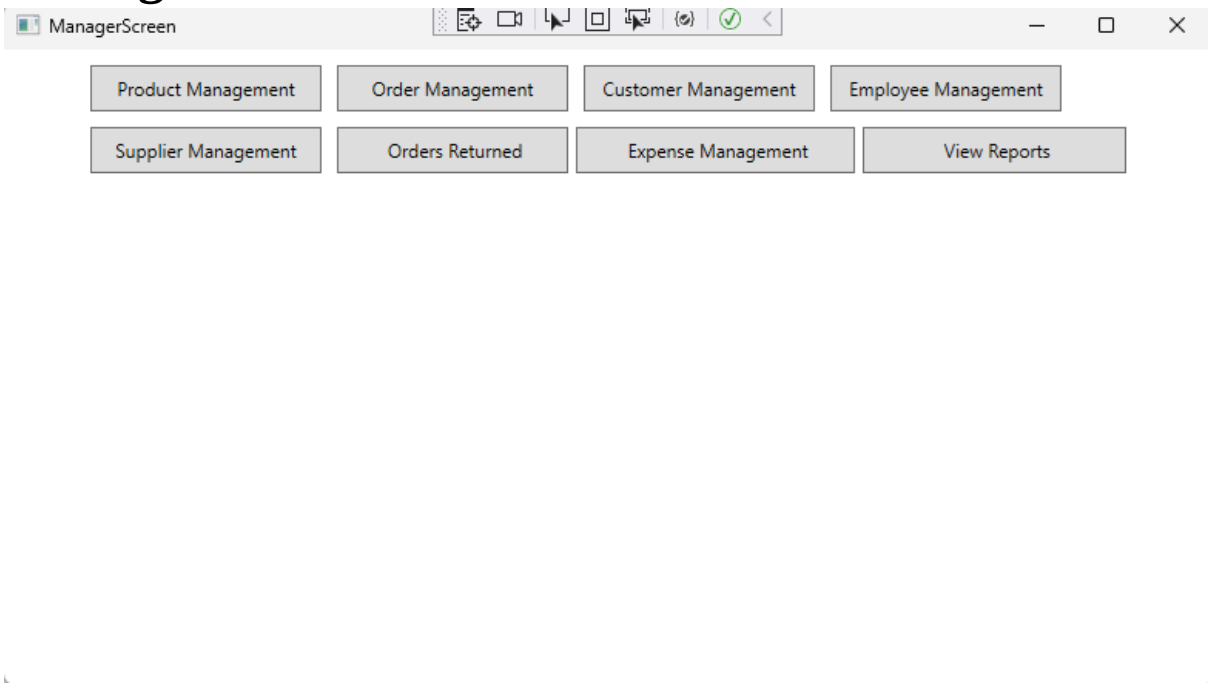
Customer view

This shows only necessary information in form of views implemented at backend when view customer button is pressed.

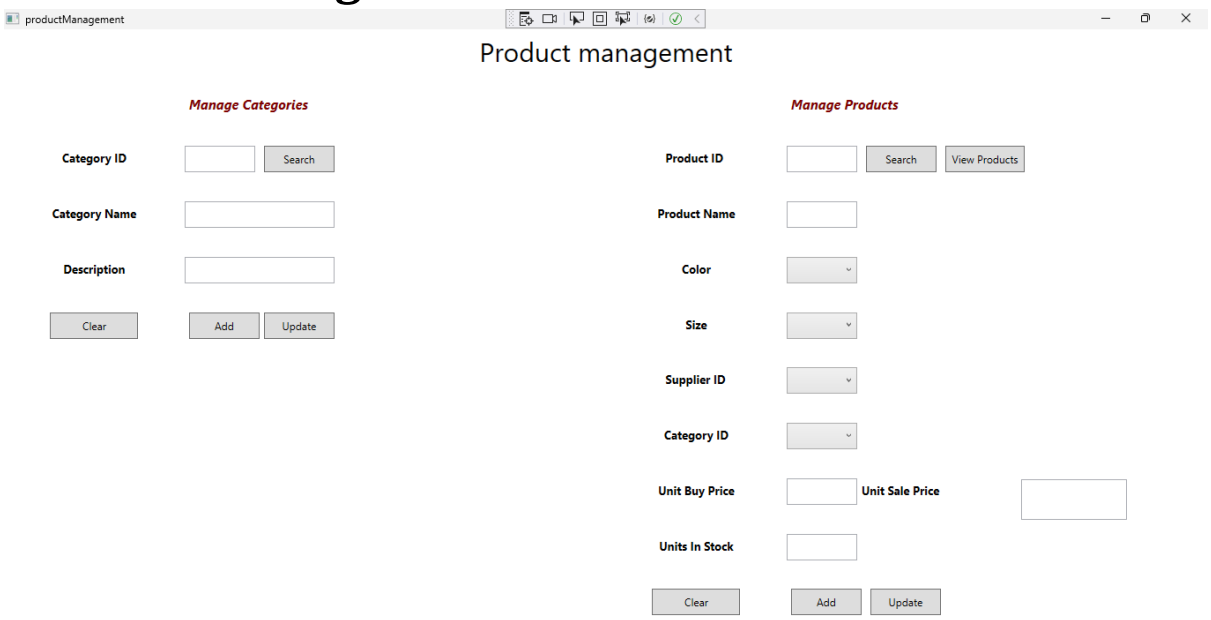


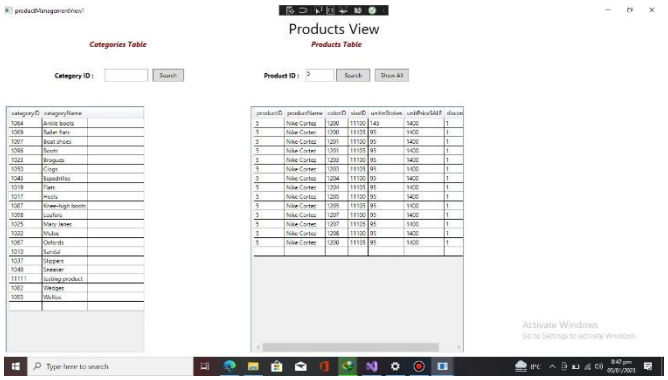
The rest screens are also present in manager screen.

Manager Screen

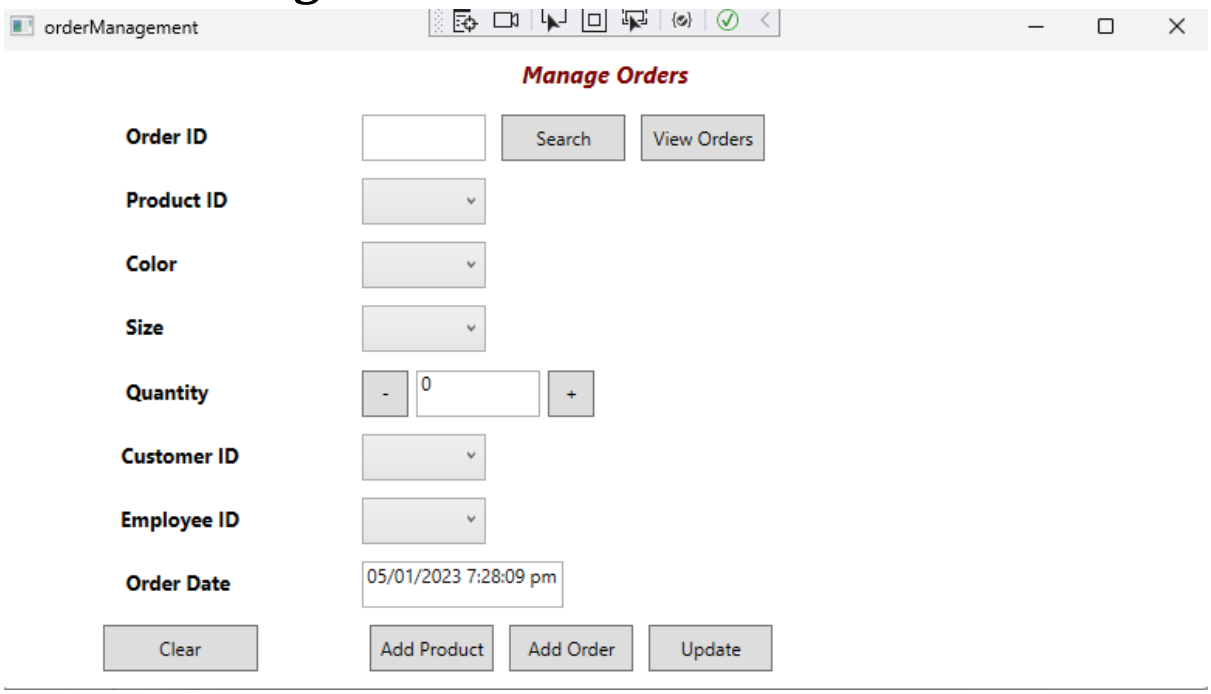


Product Management

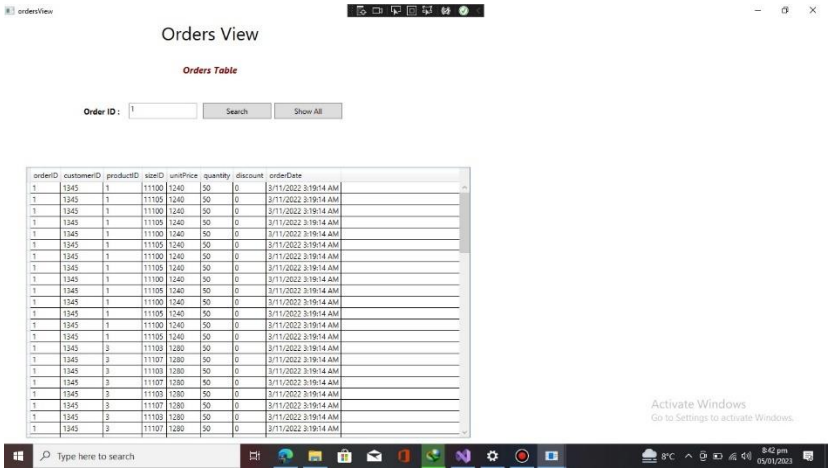




Order Management



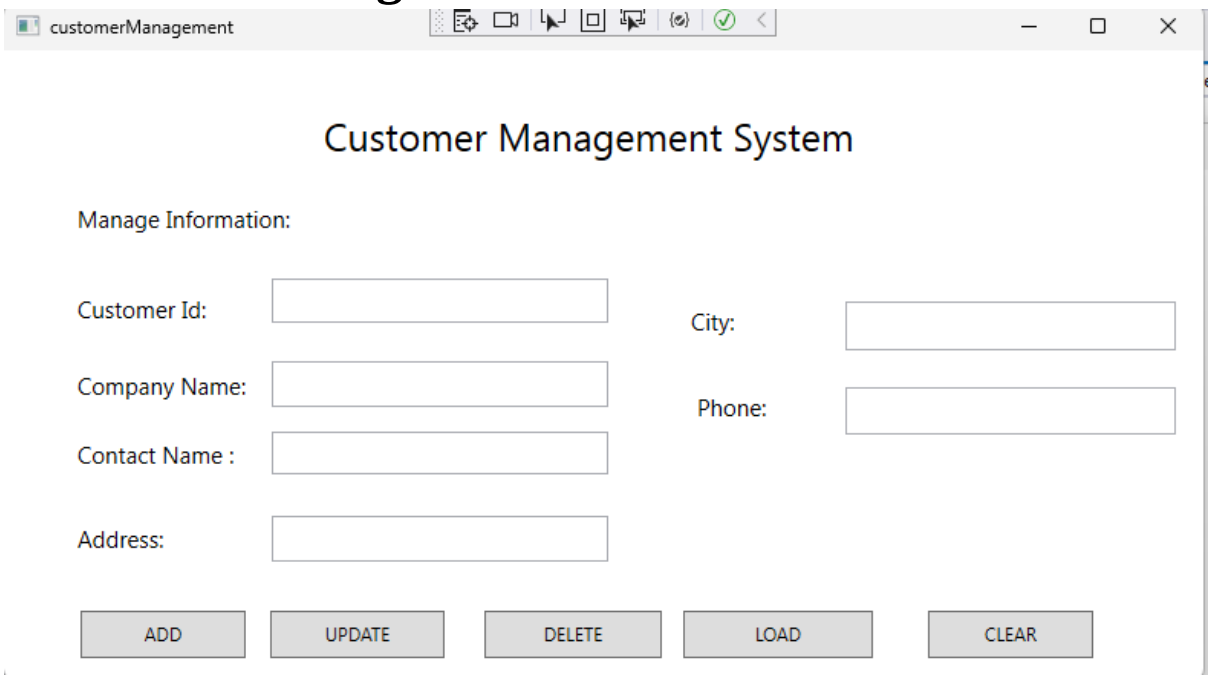
Order ID	
Product ID	
Color	
Size	
Quantity	0
Customer ID	
Employee ID	
Order Date	05/01/2023 7:28:09 pm

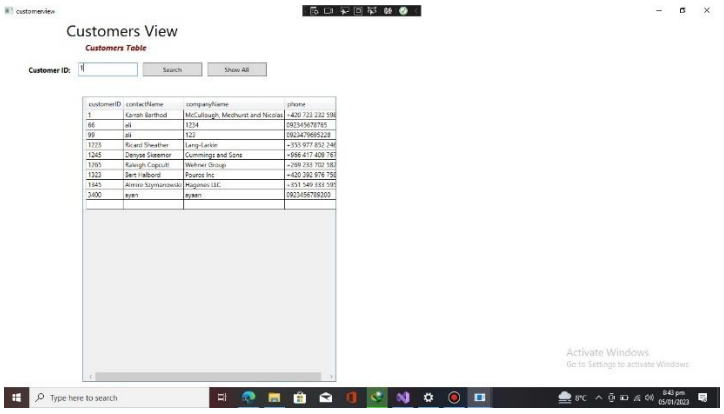


Order ID : 1

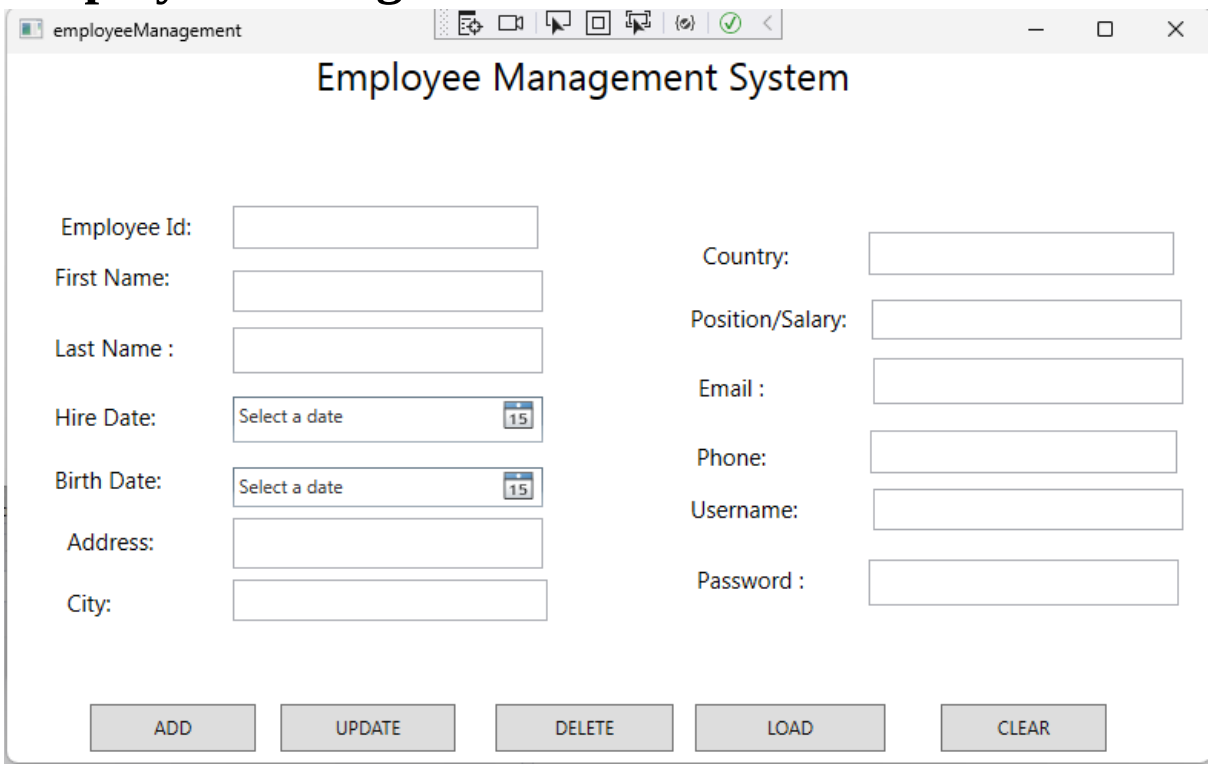
orderID	customerID	productID	sizeID	unitPrice	quantity	discount	orderDate
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11100	1240	50	0	3/11/2022 3:19:14 AM
1	1345	1	11105	1240	50	0	3/11/2022 3:19:14 AM
1	1345	3	11103	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11107	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11103	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11107	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11103	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11107	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11103	1280	50	0	3/11/2022 3:19:14 AM
1	1345	3	11107	1280	50	0	3/11/2022 3:19:14 AM

Customer Management

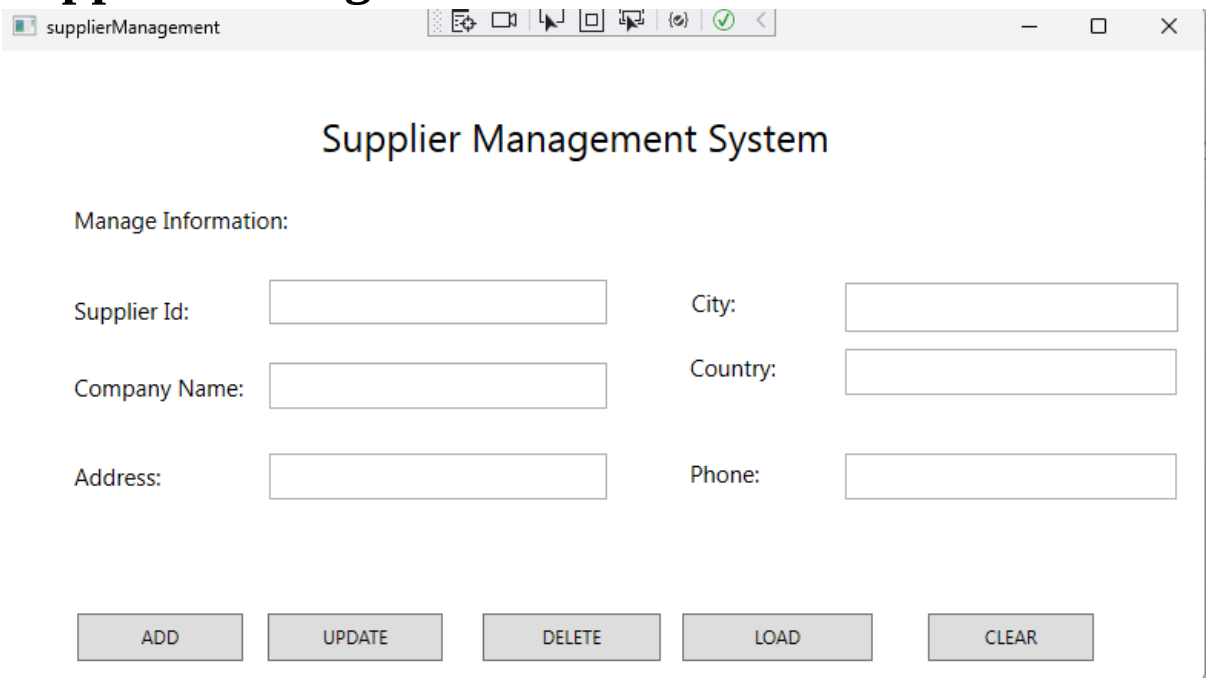




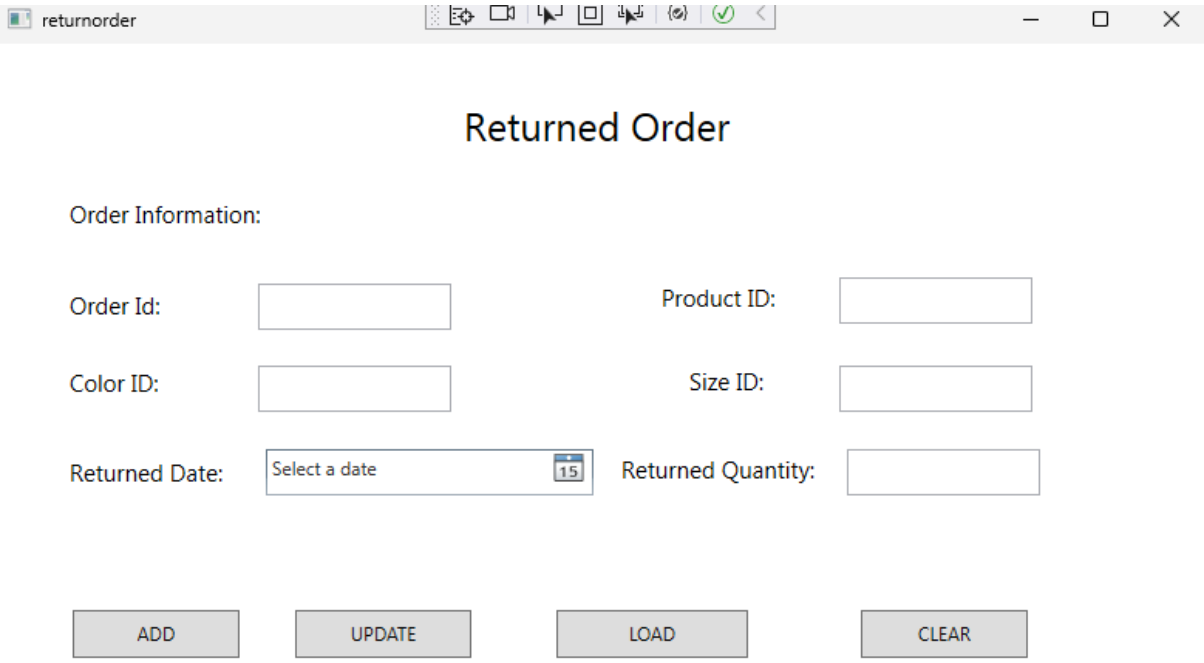
Employee Management



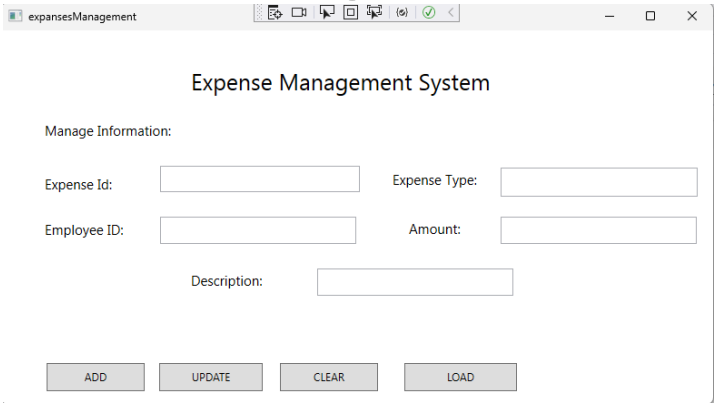
Supplier Management



Returned Order

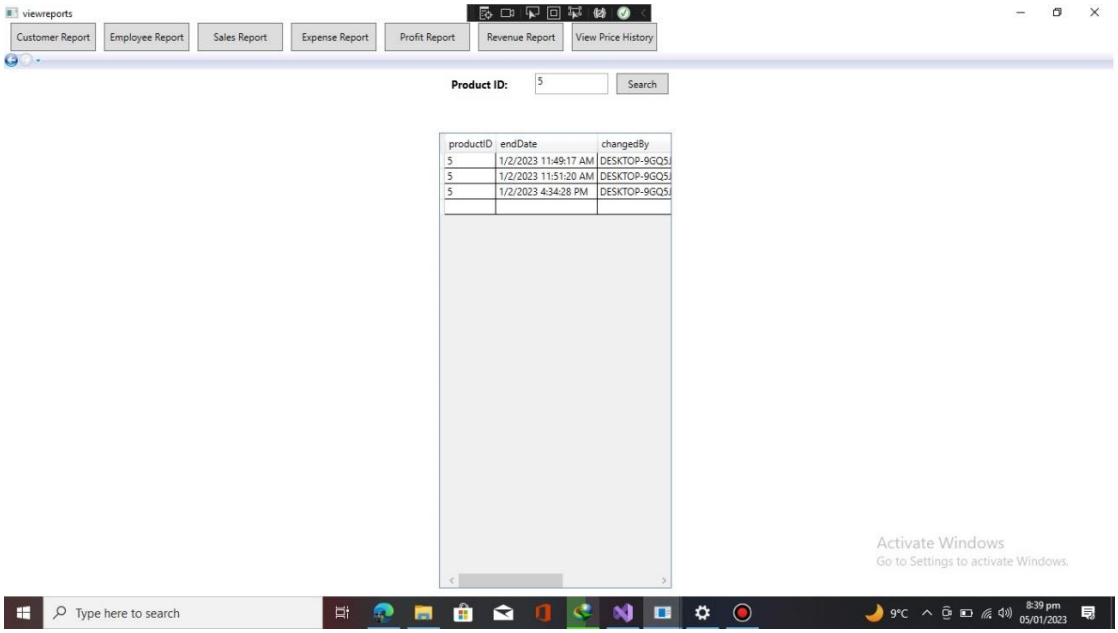


Expense Management

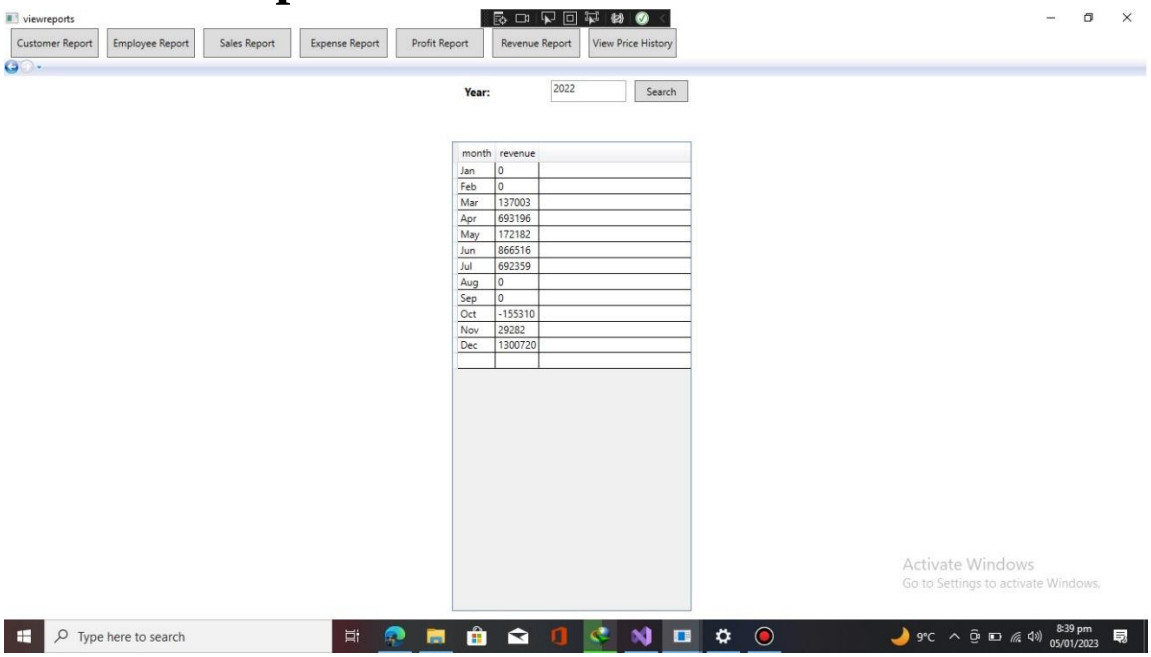


Reports

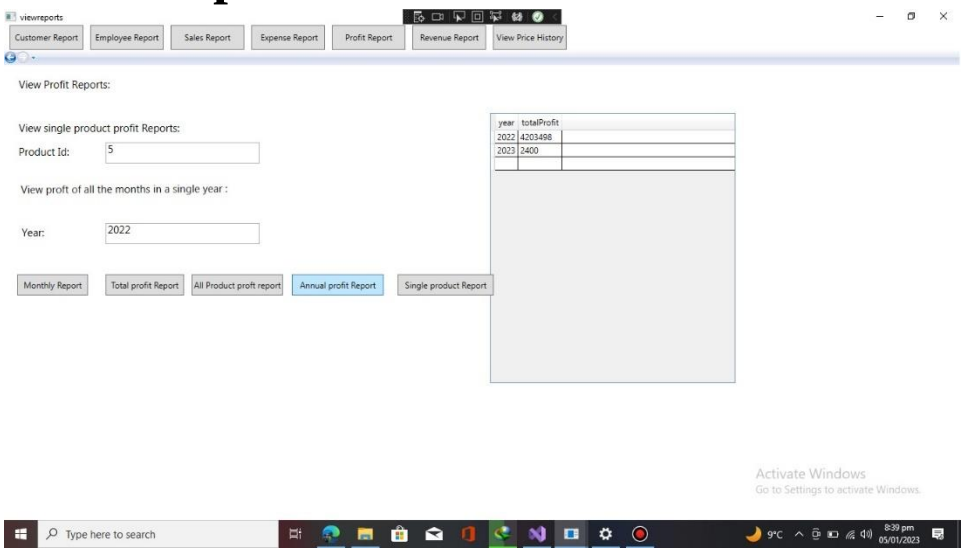
Price History

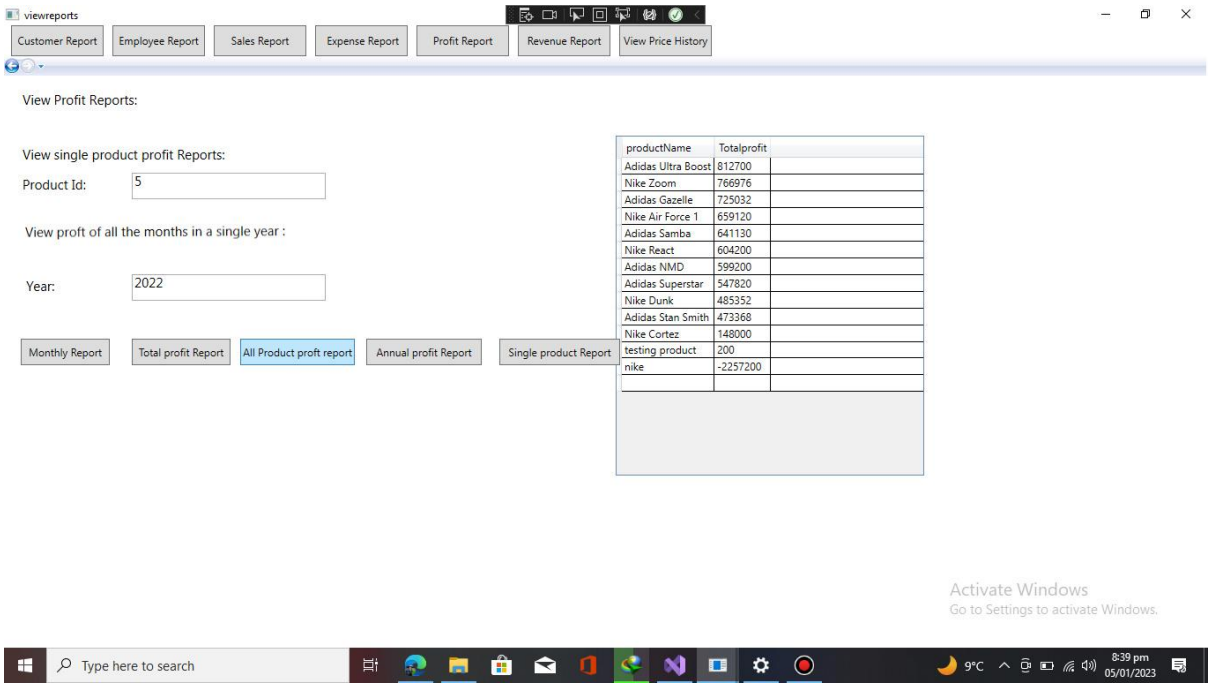


Revenue Report



Profit Report





viewreports

Customer Report | Employee Report | Sales Report | Expense Report | Profit Report | Revenue Report | View Price History

View Profit Reports:

View single product profit Reports:

Product Id: 5

View proft of all the months in a single year :

Year: 2022

Monthly Report | Total profit Report | All Product proft report | Annual profit Report | Single product Report

TotalProfit
4205898

Type here to search 9°C 8:39 pm 05/01/2023

viewreports

Customer Report | Employee Report | Sales Report | Expense Report | Profit Report | Revenue Report | View Price History

View Profit Reports:

View single product profit Reports:

Product Id: 5

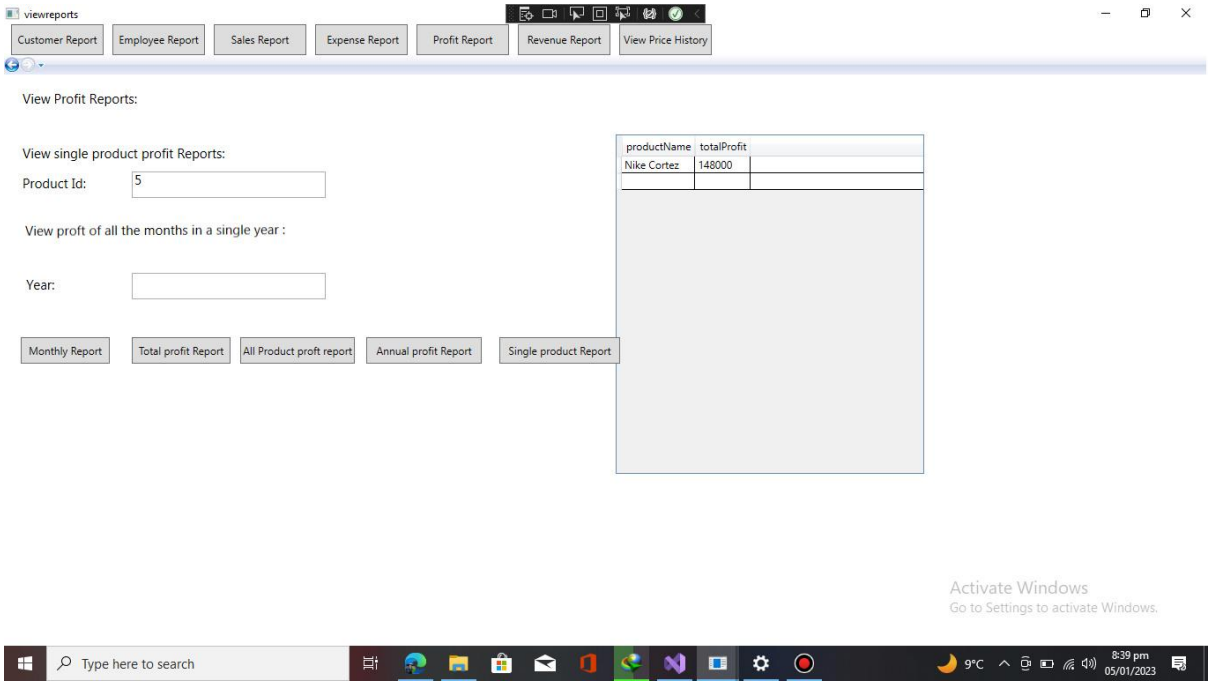
View proft of all the months in a single year :

Year: 2022

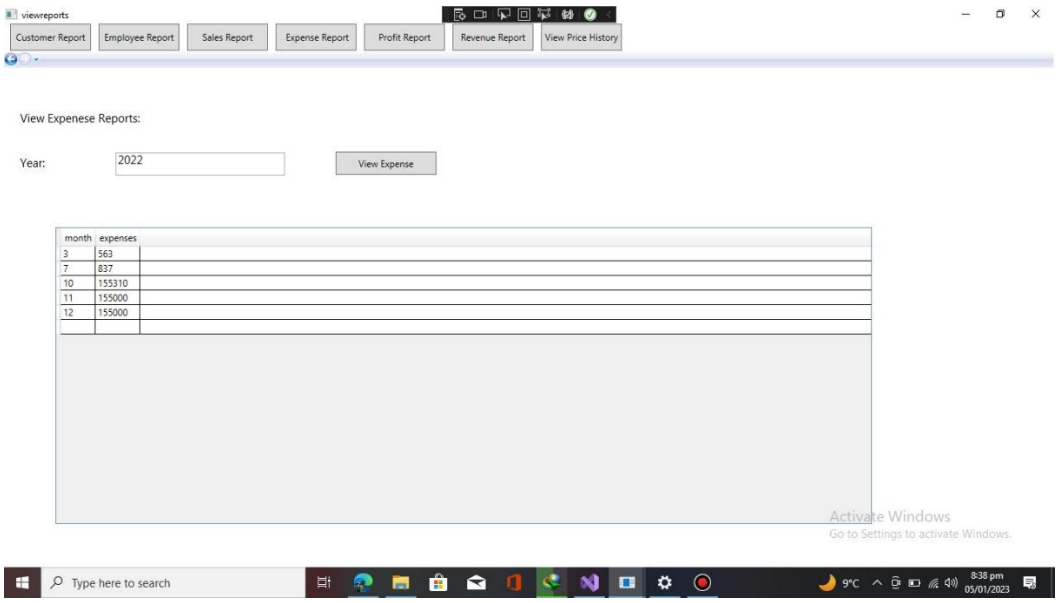
Monthly Report | Total profit Report | All Product proft report | Annual profit Report | Single product Report

month	totalProfit
3	137566
4	693196
5	172182
6	866516
7	693196
11	184282
12	1455720

Type here to search 9°C 8:39 pm 05/01/2023



Expense Report



Sales Report

viewreports

Customer Report | Employee Report | Sales Report | Expense Report | Profit Report | Revenue Report | View Price History

View sales Reports:

View single product sales Reports:

Product Id: 5

View sale report with respect to dates:

Starting date: 01/10/2022

Ending Date: 31/12/2022

View sale report with respect month and year:

Month: 4

year: 2022

Single product Report | Dated Report | Monthly/Yearly Report | Total Sales Report

productName	totalSales
Adidas Samba	3913000
Adidas NMD	3870832
Nike React	3866880
Nike Zoom	3864840
Adidas Ultra Boost	3852800
Nike Cortez	3837750
Adidas Superstar	3780560
Adidas Stan Smith	3759980
nike	3732400
Adidas Gazelle	3721032
Nike Air Force 1	3706052
Nike Dunk	3628156
testing product	1400

Type here to search 9°C 8:38 pm 05/01/2023

viewreports

Customer Report | Employee Report | Sales Report | Expense Report | Profit Report | Revenue Report | View Price History

View sales Reports:

View single product sales Reports:

Product Id: 5

View sale report with respect to dates:

Starting date: 01/10/2022

Ending Date: 31/12/2022

View sale report with respect month and year:

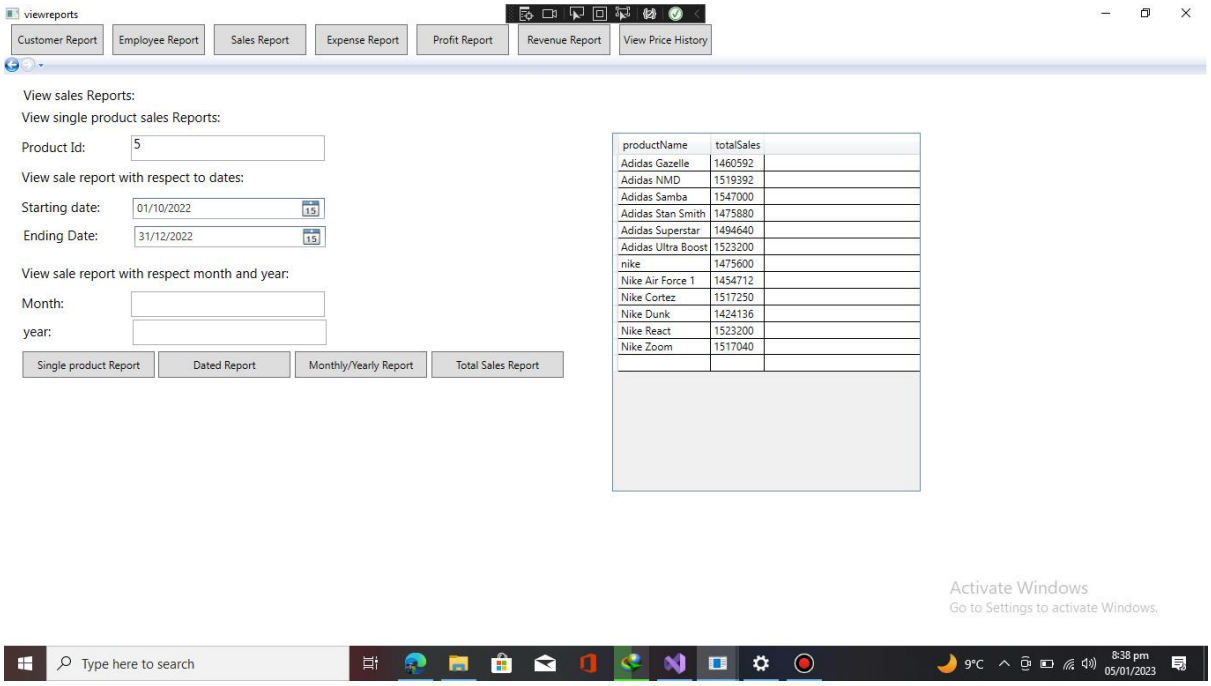
Month: 4

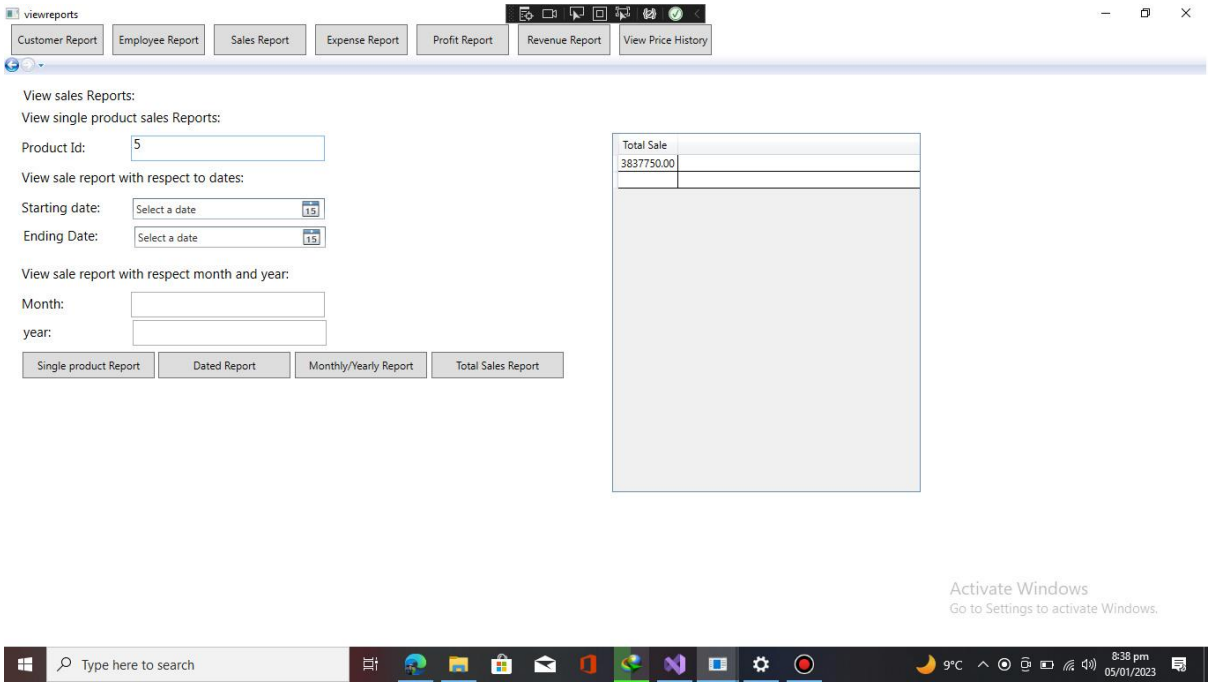
year: 2022

Single product Report | Dated Report | Monthly/Yearly Report | Total Sales Report

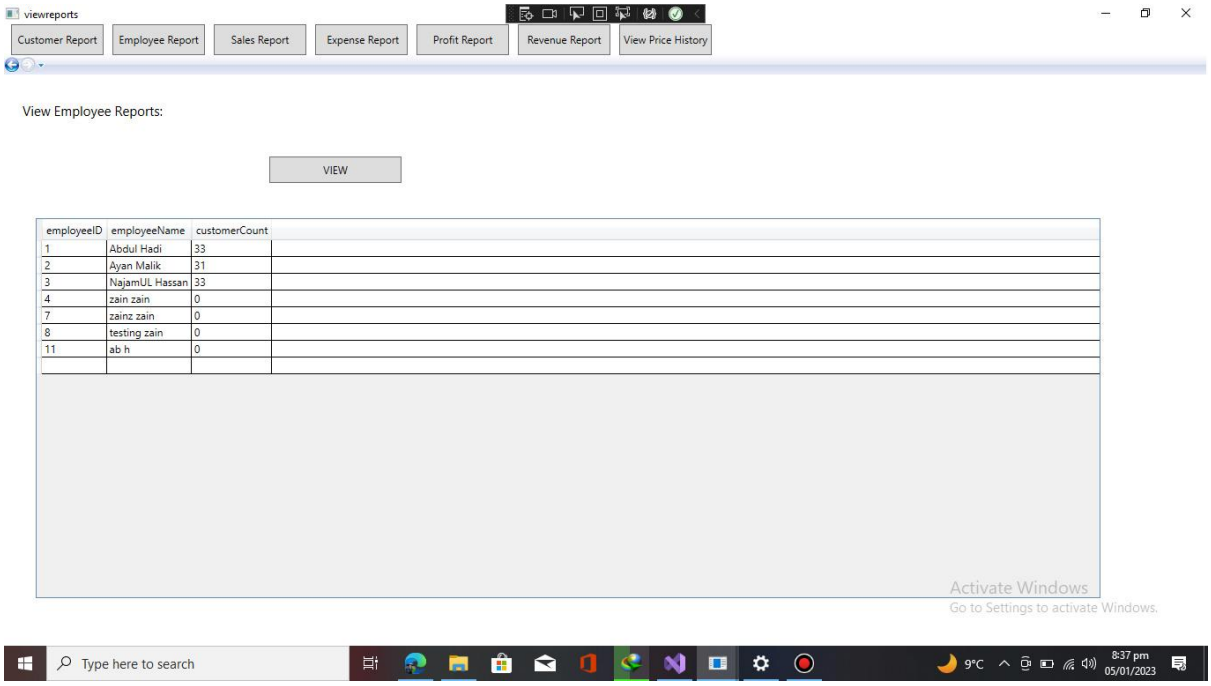
TOTALSALES
4215120

Type here to search 9°C 8:38 pm 05/01/2023

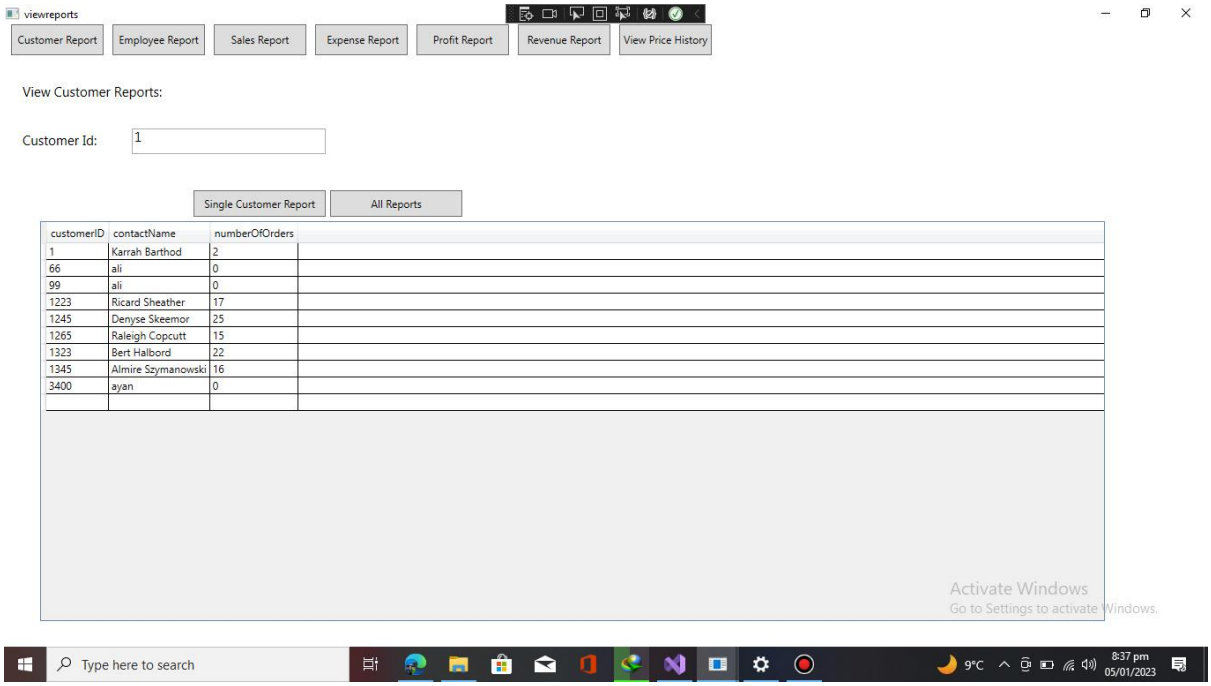


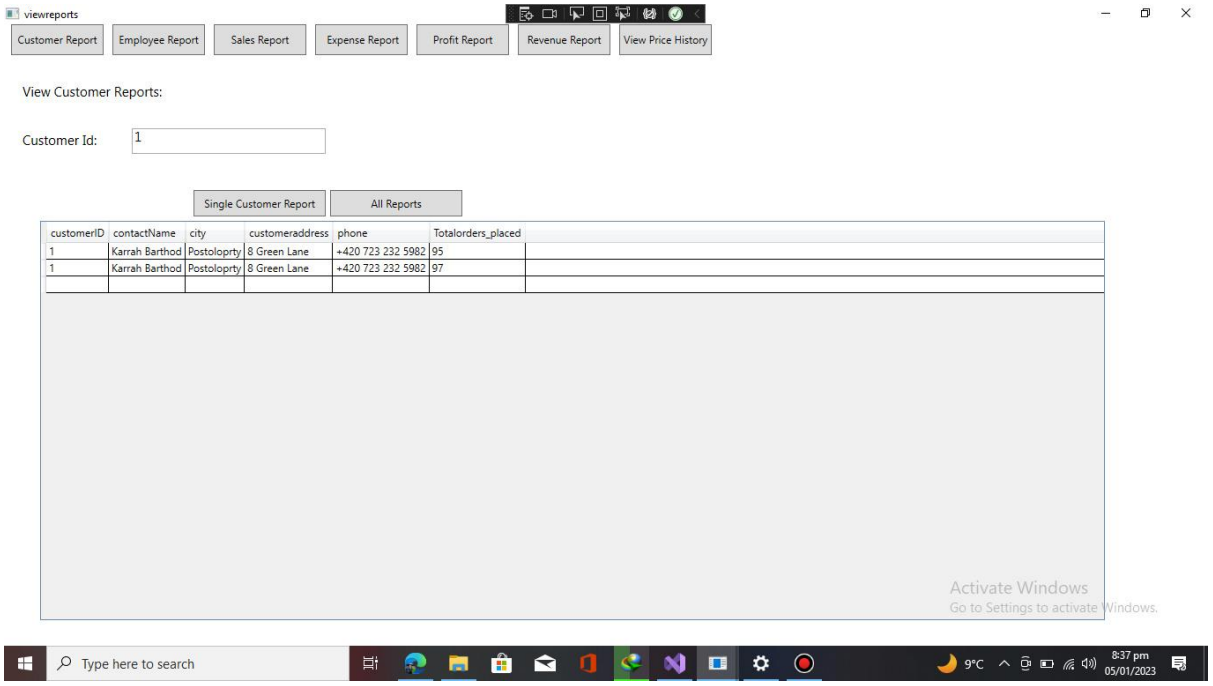


Employee Report



Customer Report





Stored Procedures, Views & Functions

```
----view products-------
CREATE VIEW vwproducts
AS
SELECT p.ProductID, p.ProductName, pd.ColorID, pd.SizeID, pd.UnitsInStock, p.UnitPrice, p.Discontinued
FROM [Product Details] AS pd
INNER JOIN Products AS p
ON pd.ProductID = p.ProductID
-
select * from vwproducts
----view orders------
CREATE VIEW vworders
AS
SELECT o.OrderID, o.CustomerID, od.ProductID, od.SizeID, od.UnitPrice, od.Quantity, od.Discount,o.OrderDate
FROM [Order Details] AS od
INNER JOIN Orders AS o
ON od.OrderID = o.OrderID
select * from vworders
----view Customer-------
CREATE VIEW vwcustomer
AS
select CustomerID,ContactName,CompanyName,Phone
from Customers
select * from vwcustomer
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53)) - Microsoft SQL Server Management Studio

```sql
CREATE PROCEDURE loginProcedure
(
    @username varchar(255),
    @password varchar(255),
    @result bit OUTPUT
)
AS
BEGIN
    DECLARE @salt varchar(255), @hashedPassword VARBINARY(MAX), @storedPassword VARBINARY(MAX);

    -- Get the salt and hashed password for the entered username
    SELECT @salt = phone, @storedPassword = userPassword
    FROM Employees
    WHERE Username = @username

    -- Hash the entered password using the salt
    SET @hashedPassword = HASHBYTES('SHA2_256',  @password+@salt)

    -- Compare the hashed password to the stored password
    IF @hashedPassword = @storedPassword
    BEGIN

        SET @result = 1
    END
    ELSE
    BEGIN

        SET @result = 0
    END

    RETURN @result
END
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53)) - Microsoft SQL Server Management Studio

```sql
        SET @result = 0
    END

    RETURN @result
END



--trigger for changing the price of any product;

CREATE TRIGGER Price_changed ON Products
AFTER UPDATE
AS
BEGIN
    SET NOCOUNT ON;   --it is removing the rows affected message

    -- Check if the unitPriceSale or unitPriceBuy columns have been updated
    IF (EXISTS (SELECT unitPriceSale FROM INSERTED) AND EXISTS (SELECT unitPriceSale FROM DELETED)) OR
       (EXISTS (SELECT unitPriceBuy FROM INSERTED) AND EXISTS (SELECT unitPriceBuy FROM DELETED))
    BEGIN
        -- Check if the unitPriceSale or unitPriceBuy columns have been changed
        IF (SELECT unitPriceSale FROM INSERTED) <> (SELECT unitPriceSale FROM DELETED) OR
           (SELECT unitPriceBuy FROM INSERTED) <> (SELECT unitPriceBuy FROM DELETED)
        BEGIN
            INSERT INTO PriceHistory(productID, endDate, changedBy, newUnitPriceSale, newUnitPriceBuy)
            SELECT productID, GETDATE(), SUSER_SNAME(), unitPriceSale, unitPriceBuy
            FROM inserted;
        END
    END
END;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53)) - Microsoft SQL Server Management Studio

```sql
--trigger for updating unit in stokes when new record is added in OrderDetails


CREATE TRIGGER update_unitInStokes
ON OrderDetails
AFTER INSERT
AS
BEGIN
    UPDATE ProductDetails
    SET unitInStokes = unitInStokes - i.quantity
    FROM ProductDetails pd
    INNER JOIN inserted i
        ON pd.productID = i.productID
        AND pd.colorID = i.colorID
        AND pd.sizeID = i.sizeID;
END;


--trigger for returned orders that will update the stokes of the returned product
CREATE TRIGGER update_unitInStokes_returned
ON ReturnOrder
AFTER INSERT
AS
BEGIN
    UPDATE ProductDetails
    SET unitInStokes = unitInStokes + i.quantityReturned
    FROM ProductDetails pd
    INNER JOIN inserted i
        ON pd.productID = i.productID
        AND pd.colorID = i.colorID
        AND pd.sizeID = i.sizeID;
END;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53)) - Microsoft SQL Server Management Studio

```sql
--for inserting in return orders
CREATE PROCEDURE AddReturnOrderDetail
    @orderID INT,
    @productID INT,
    @sizeID INT,
    @colorID INT,
    @quantity INT,
    @result INT OUTPUT
AS
BEGIN
    DECLARE @currentQuantity INT

    -- Check if the record already exists in OrderDetails
    SELECT @currentQuantity = quantity
    FROM OrderDetails
    WHERE orderID = @orderID
    AND productID = @productID
    AND sizeID = @sizeID
    AND colorID = @colorID

    -- If the record does not exist or the input quantity is greater than the current quantity, set the result to 0
    IF @currentQuantity IS NULL OR @quantity > @currentQuantity
    BEGIN
        SET @result = 0
    END
    -- If the record exists and the input quantity is less than or equal to the current quantity, insert a new record in the ReturnOrder table and s
    ELSE
    BEGIN
        INSERT INTO ReturnOrder
        VALUES (@orderID, @productID, @colorID, @sizeID, @quantity, GETDATE())

        SET @result = 1
    END
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
--for updating the return order details
CREATE PROCEDURE UpdateReturnOrderDetail
    @orderID INT,
    @productID INT,
    @sizeID INT,
    @colorID INT,
    @quantity INT,
    @result INT OUTPUT
AS
BEGIN
    DECLARE @currentQuantity INT

    -- Check if the record already exists in OrderDetails
    SELECT @currentQuantity = quantity
    FROM OrderDetails
    WHERE orderID = @orderID
    AND productID = @productID
    AND sizeID = @sizeID
    AND colorID = @colorID

    -- If the record does not exist or the input quantity is greater than the current quantity, set the result to 0
    IF @currentQuantity IS NULL OR @quantity > @currentQuantity
    BEGIN
        SET @result = 0
    END
    -- If the record exists and the input quantity is less than or equal to the current quantity, update the record in the ReturnOrder table and set the result to 1
    ELSE
    BEGIN
        UPDATE ReturnOrder
        SET quantityReturned = @quantity, returnedDate = GETDATE()
        WHERE orderID = @orderID
        AND productID = @productID
        AND colorID = @colorID
        AND sizeID = @sizeID

        SET @result = 1
    END END;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
--for reports


--for profit
alter FUNCTION getProfitByProduct()
RETURNS TABLE
AS
    RETURN
    (
        SELECT p.productName,
               SUM(od.quantity * (od.unitPrice - p.unitPriceBuy)* ((100-od.discount)/100))-
                SUM(CASE WHEN ro.quantityReturned>0 THEN ((od.unitPrice - p.unitPriceBuy) * ro.quantityReturned *((100-od.discount)/100)) ELSE 0 END) AS Total_profit

        FROM Products p
        INNER JOIN OrderDetails od ON p.productID = od.productID
        LEFT JOIN ReturnOrder ro ON od.orderID = ro.orderID AND od.productID = ro.productID and ro.colorID = od.colorID and ro.sizeID = od.sizeID
        GROUP BY p.productName
    );



    select * from getProfitByProduct()
    order by Total_profit desc;


    --testing for this function
    insert into ReturnOrder
    values(1,5,1200,11100,25, GETDATE());
    SELECT * FROM ReturnOrder;



    SELECT * FROM Products WHERE productID = 5;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
--profit from single product
ALTER FUNCTION getProfitBySingleProduct(@id INT)
RETURNS TABLE
AS
    RETURN
    (
        SELECT p.productName, SUM(od.quantity * (od.unitPrice - p.unitPriceBuy) *((100-od.discount)/100))-
        SUM(CASE WHEN ro.quantityReturned>0 THEN ((od.unitPrice - p.unitPriceBuy) * ro.quantityReturned*((100-od.discount)/100)) ELSE 0 END)
        AS totalProfit
        FROM Products p
        INNER JOIN OrderDetails od ON p.productID = od.productID
         LEFT JOIN ReturnOrder ro ON od.orderID = ro.orderID AND od.productID = ro.productID and ro.colorID = od.colorID and ro.sizeID = od.sizeID
        where p.productID = @id
        group by p.productName
    );

    SELECT * FROM getProfitBySingleProduct(5);

    alter FUNCTION getMonthlyProfit(@year INT)
RETURNS TABLE
AS
    RETURN
    (
        SELECT MONTH(o.orderDate) AS month, SUM(od.quantity * (od.unitPrice - p.unitPriceBuy) - ((100-od.discount)/100))-
        SUM(CASE WHEN ro.quantityReturned>0 THEN ((od.unitPrice - p.unitPriceBuy) * ro.quantityReturned *((100-od.discount)/100) ) ELSE 0 END)
        AS totalProfit
        FROM Orders o
        INNER JOIN OrderDetails od ON o.orderID = od.orderID
        inner join Products p on p.productID = od.productID
        LEFT JOIN ReturnOrder ro ON od.orderID = ro.orderID AND od.productID = ro.productID and ro.colorID = od.colorID and ro.sizeID = od.sizeID
        WHERE YEAR(o.orderDate) = @year
        GROUP BY MONTH(o.orderDate)
    );


    select * from getMonthlyProfit(2022)
    order by month asc;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
---yearly profits
    alter FUNCTION getProfitByYear()
RETURNS TABLE
AS
    RETURN
    (
        SELECT YEAR(o.orderDate) AS year,
               SUM(od.quantity * (od.unitPrice - p.unitPriceBuy) * ((100-od.discount)/100))
               - SUM(CASE WHEN ro.quantityReturned>0 THEN ((od.unitPrice - p.unitPriceBuy) * ro.quantityReturned  *((100-od.discount)/100) ) ELSE 0 END) AS totalProfit
        FROM Orders o
        INNER JOIN OrderDetails od ON o.orderID = od.orderID
        INNER JOIN Products p ON od.productID = p.productID
        LEFT JOIN ReturnOrder ro ON od.orderID = ro.orderID AND od.productID = ro.productID AND od.colorID = ro.colorID AND od.sizeID = ro.sizeID
        GROUP BY YEAR(o.orderDate)
    );

    select * from getProfitByYear();

alter function getTotalProfit()
returns table
as
return (
        select sum(Total_profit)as Total_Profit
        from getProfitByProduct()


);

select * from getTotalProfit();
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
--sales reports
CREATE FUNCTION getTotalSales(@month INT, @year INT)
RETURNS TABLE
AS
RETURN

    SELECT SUM(od.quantity * od.unitPrice * ((100 - od.discount)/100)) AS TOTAL_SALES
    FROM OrderDetails od
    JOIN Orders o ON od.orderID = o.orderID
    WHERE MONTH(o.orderDate) = @month AND YEAR(o.orderDate) = @year;

    SELECT * FROM getTotalSales(2,2022);

--sales of all products based on quantity
create function getTotalSalesAllProducts()
returns table
as
RETURN
    (
        SELECT p.productName, SUM(od.quantity * od.unitPrice) AS totalSales
        FROM Products p
        INNER JOIN OrderDetails od ON p.productID = od.productID
        GROUP BY p.productName

    );

    SELECT * FROM getTotalSalesAllProducts()
    ORDER BY totalSales desc;
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
    --sales for specific time range
    CREATE FUNCTION getTotalSalesBetweenDates(@startDate DATE, @endDate DATE)
RETURNS TABLE
AS
RETURN
    (
        SELECT p.productName, SUM(od.quantity * od.unitPrice) AS totalSales
        FROM Products p
        INNER JOIN OrderDetails od ON p.productID = od.productID
        INNER JOIN Orders o ON od.orderID = o.orderID
        WHERE o.orderDate BETWEEN @startDate AND @endDate
        GROUP BY p.productName
    );

    SELECT * FROM getTotalSalesBetweenDates('2022-09-01', '2022-12-31');

    CREATE FUNCTION getTotalSalesForProduct(@productID INT)
RETURNS NUMERIC(18, 2)
AS
BEGIN
    DECLARE @totalSales NUMERIC(18, 2) = 0

    SELECT @totalSales = SUM(od.quantity * od.unitPrice)
    FROM OrderDetails od
    WHERE od.productID = @productID

    RETURN @totalSales
END;

select dbo.getTotalSalesForProduct(5) as [Total Sale];
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

```sql
--reports for customers

CREATE FUNCTION getCustomersByNumberOfOrders()
RETURNS TABLE
AS
RETURN
    (
        SELECT c.customerID, c.contactName, COUNT(o.orderID) AS numberOfOrders
        FROM Customers c
        LEFT JOIN Orders o ON c.customerID = o.customerID
        GROUP BY c.customerID, c.contactName

    );


select * from getCustomersByNumberOfOrders();


--report for single customer
alter function getSingleCustomer(@id INT)
RETURNS TABLE
AS
    RETURN(
        select
        c.customerID, c.contactName,c.city, c.customer_address,c.phone, sum(o.orderID) as Total_orders_placed

        from Customers c
        LEFT JOIN Orders o on c.customerID = o.customerID
        where @id = c.customerID
        GROUP BY c.customerID, c.contactName,c.contactName, c.customer_address,c.city, c.customer_address,c.phone, o.orderID



    );


select * from getSingleCustomer(1);


--for employees
```

operations (4).sql - LAPTOP-5OSRB32A\SQLEXPRESS.master (LAPTOP-5OSRB32A\Ayan (53))* - Microsoft SQL Server Management Studio

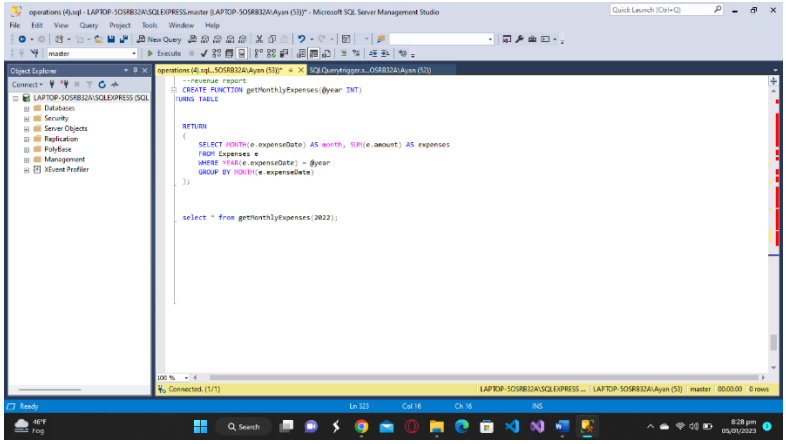
```sql
--for employees

-- ordered by total numbers of orders they have handled
CREATE FUNCTION getEmployeeCustomerCount()
    RETURNS TABLE
    AS
    RETURN
        SELECT e.employeeID, e.firstName + ' ' + e.lastName AS employeeName, COUNT(o.customerID) AS customerCount
        FROM Employees e
        LEFT JOIN Orders o ON e.employeeID = o.employeeID
        GROUP BY e.employeeID, e.firstName, e.lastName;


    --testing
    select * from getEmployeeCustomerCount();


-- returns the information of any employee
CREATE FUNCTION getEmployeeInfo(@id INT)
    RETURNS TABLE
    AS
    RETURN
        SELECT * FROM Employees WHERE employeeID = @id;


        --testing
        SELECT * FROM getEmployeeInfo(1);
```



Normalization Of Tables

Orders Table – 1NF

Partial Dependency

OID	CID	EID	Order Date	PID	Color ID	Size ID	Unit Price	Qty	Discount

Composite Primary Key: Order ID (OID), Product ID (PID), Color ID, Size ID

In orders table, customer ID (CID), employee ID (EID) and Order date are **dependent** on only **Order ID**, which is **part of the whole primary key**.

While unit price, quantity, and discount are **dependent** on only **Product ID, Color ID, Size ID**.

This table contains partial dependency because all attributes are not dependent on the whole primary key.

To fix this we decompose the orders table into two tables.

Order Details Table – 3NF

OID	PID	Color ID	Size ID	Unit Price	Qty	Discount

Composite Primary Key: Order ID (OID), Product ID (PID), Color ID, Size ID

Orders Table – 3NF

OID	CID	EID	Order Date

Primary Key: Order ID (OID)

The new tables do not contain any partial or transitive dependencies, thus they are in 3NF.

Products Table – 2NF

Transitive Dependency

Product ID	Product Name	Supplier ID	Category ID	Category Name	Unit Price	Discontinued

Primary Key: Product ID

In products table **Category Name** is dependent on **Category ID**, both of which are **non-key attributes**. Since, neither of them is dependent on the whole **primary key**, the products table contains **transitive dependency**, because a non-key attribute is dependent on another non-key attribute.

To put the table in **3NF** form, we remove **category Name** and use **category ID** to form another table named **Categories**.

Categories Table – 3NF

Category ID	Category Name

Primary Key: Category ID

We use category ID in the products table as a foreign key.

Products Table – 3NF

Product ID	Product Name	Supplier ID	Category ID	Unit Price	Discontinued

Primary Key: Product ID

Denormalization

Date

Sizes
SIZEID | SIZE Number |

Colors
Color ID | Color Name |

Sizes and colors are used in order details and product details table as composite key.

Product Details
| Pro ID | Col ID | Size ID | Units Stocks | Reorder level |

Product ID, Color ID and Size ID are composite keys.

Order Details
| Order ID | Pro ID | Color ID | Size ID | Unit Price | Quantity | Discount |

Order ID, Product ID, Color ID, Size ID are composite key.

They can be denormalized by removing size and color ~~name~~ table and writing size name and color name in the above 2 tables.

Product Details
| Pro ID | Col Name | Size Name | Units in Stock | Reorder Lvl |

Order Details
| Order ID | Pro ID | Color Name | Size Name | Unit Price | QTY | Discount |

But then data redundancy ~~are~~ is increased and data consistency is compromised. Different anomalies can also occur so it is less beneficial to denormalize..

Conclusion

The entity model presented here seeks to model the data required in the system from the point of view of the Sales Department. When all the three views will be integrated, some of the entities and relationships might need some modification with respect to the requirements of other systems.

Recommendations

Based on the user requirements of the company it is recommended that a centralized database server be placed at any of the offices. Database should have web application interface, connecting all the remote departments through a secure connection. Taking into consideration the ease of information access with the availability of the automated system it is recommended that a security policy should be devised, limiting the system access to authorized staff only. It is anticipated that these steps will greatly improve the speed of response, accuracy of information, cost-effective transactions; hence will enhance the overall business process.