Supermarket Management System

Course: Database Systems

Course ID: CS F212

BY

Abhiram H	2021B4A71134P
Ankur Renduchintala	2021B5A71159P
Sai Vara Prasad	2021B5A71541P
Varun Sai Sajja	2021B3A71054P

BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI

17 April, 2024

TABLE OF CONTENTS

PROBLEM DESCRIPTION

Our goal is to create an application that manages the inventory and customer purchases of a supermarket. The idea is motivated by Akshay, the on-campus supermarket. Most of the transactions (payment by ID card, faculty members) in Akshay are recorded in a register. To solve that issue, which may be present in other stores as well, we decided to implement this project.

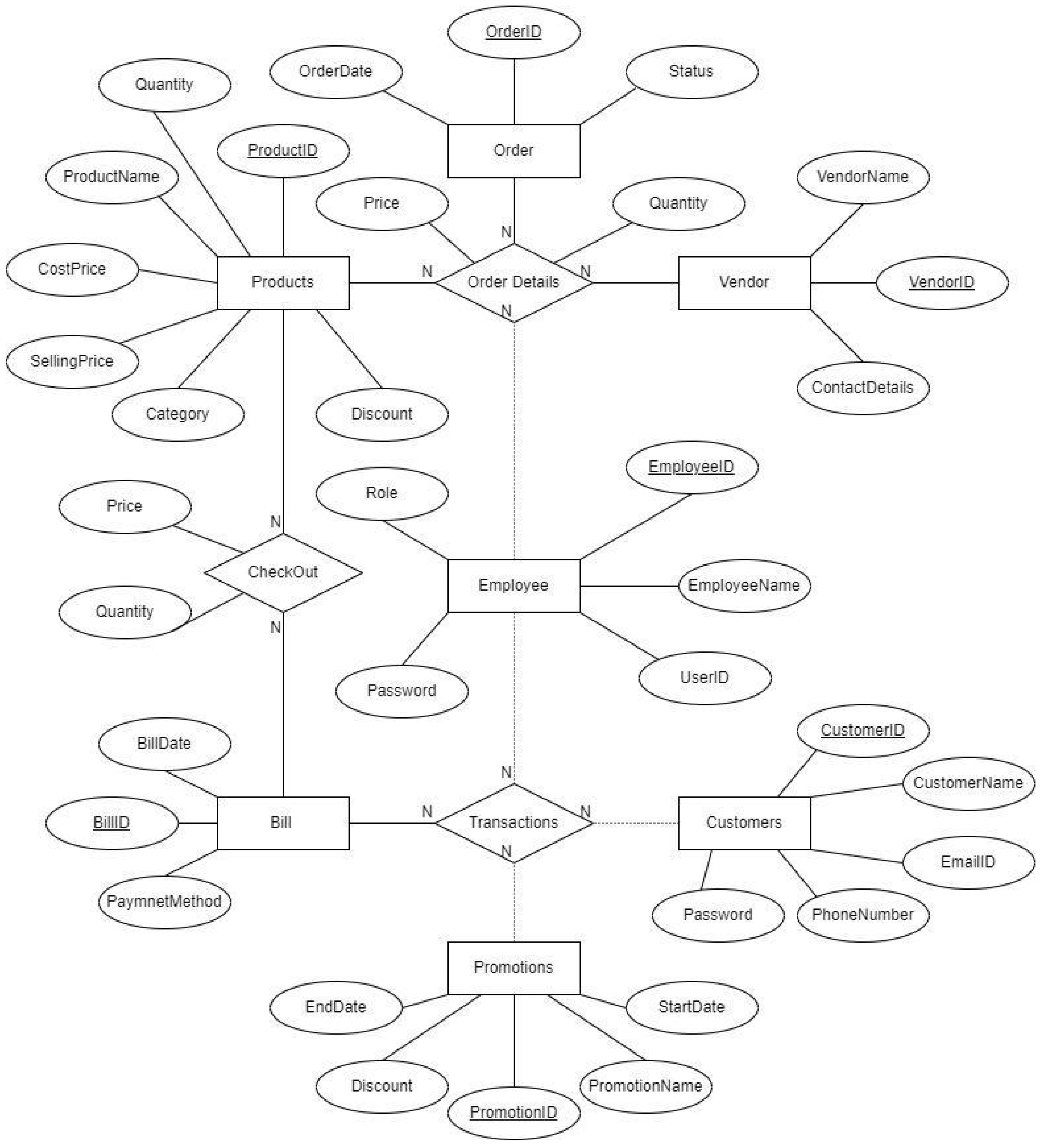
FEATURES

The main features of the project are:

1. **Inventory Management**: Through a user-friendly interface, employees can add, update, and delete inventory items. The robust search and browsing functionality will enable quick access to specific products or categories, streamlining the inventory management process. This seamless inventory management will ensure that the supermarket maintains optimal stock and meets customer demands promptly.
2. **Inventory Management**: Enable employees to monitor stock levels and receive notifications when inventory falls below a predefined threshold, ensuring proactive replenishment of popular items and minimizing stockouts. Automated low stock alerts will trigger timely reordering processes, helping maintain sufficient inventory.
3. **Transaction Recording and Analysis**: The system will meticulously record and maintain comprehensive data on all customer transactions processed and orders placed by employees. This detailed transactional history will serve as a valuable resource for generating insightful reports and conducting in-depth analysis of sales performance. The supermarket can track key metrics such as total revenue, profit margins, and expenses across various time periods.
4. **Point-of-Sale (POS) System**: To facilitate seamless and efficient customer transactions, the system will incorporate a user-friendly point-of-sale (POS) interface tailored for supermarket operations. This interface will help cashiers to enter product information, apply various discounts or promotions. Upon completion of each transaction, the system will generate itemized receipts for customers (available in their dashboard), providing a detailed breakdown of purchased items, quantities, and prices.
5. **Promotions and Discounts:** The system will enable employees to seamlessly apply promotions and discounts during the checkout process. The point-of-sale (POS) interface will prominently display all applicable promotions and discounts, allowing cashiers to easily select and apply them to customer purchases.
6. **User Authentication**: To safeguard the integrity and confidentiality of the system, user authentication measures will be implemented, ensuring that only authorized employees can access and interact with the application. A role-based access control model will be employed, defining distinct user roles such as customers and employees, each with appropriate permissions tailored to their respective roles. Furthermore, to enhance security, passwords will be securely hashed using industry-standard algorithms before storage, preventing plaintext exposure.

7. **Customer Management**: Allow customers to view customer information and transaction history. Capture customer details during checkout for future reference and marketing purposes. Allow customers to check the availability of specified items.
8. **Vendor Management**: Provide access to vendor information and purchasing history to facilitate communication and ordering. Streamline the procurement process by allowing employees to create purchase orders and track deliveries.

ER MODEL



The various tables that we used in the project are:

1. **Products**

- Attributes: ProductID (Primary Key), ProductName, CostPrice, SellingPrice, Quantity, Discount, ProductCategories

- Functional Dependency: ProductName → Category
- Foreign Key: ProductName references ProductName from ProductCategories.

2. **Vendors**
   - Attributes: VendorID (Primary Key), VendorName, ContactDetails
3. **Orders**
   - Attributes: OrderID (Primary Key), Status, OrderDate
4. **Customers**
   - Attributes: CustomerID (Key), CustomerName, EmailID, PhoneNumber, Password
5. **Employees**
   - Attributes: EmployeeID (Primary Key), EmployeeName, UserID, Password, Role
6. **Promotions**
   - Attributes: PromotionID (Primary Key), PromotionName, Discount, StartDate, EndDate
7. **Bills**
   - Attributes: BillID (Primary Key), BillDate, PaymentMethod

Some of the Relationship Tables that have been used are:

1. **OrderDetails**
   - Attributes: OrderID, VendorID, ProductID, EmployeeID, Quantity
   - Candidate Key: OrderID, VendorID, ProductID
   - Foreign Keys: OrderID references OrderID from Orders, VendorID references VendorID from Vendors, ProductID references ProductID from Products, and EmployeeID references EmployeeID from Employees.
2. **Transactions**
   - Attributes: BillID (Primary Key), PromotionID, CustomerID, EmployeeID
   - Foreign Keys: PromotionID references PromotionID from Promotions, BillID references BillID from Bills, CustomerID references CustomerID Customers, and EmployeeID references EmployeeID from Employees.
3. **Checkout**
   - Attributes: BillID, ProductID, Price, Quantity
   - Candidate Key: BillID, ProductID
   - Foreign Keys: ProductID references ProductID from Products and BillID references BillID from Bills.

NORMALIZATION

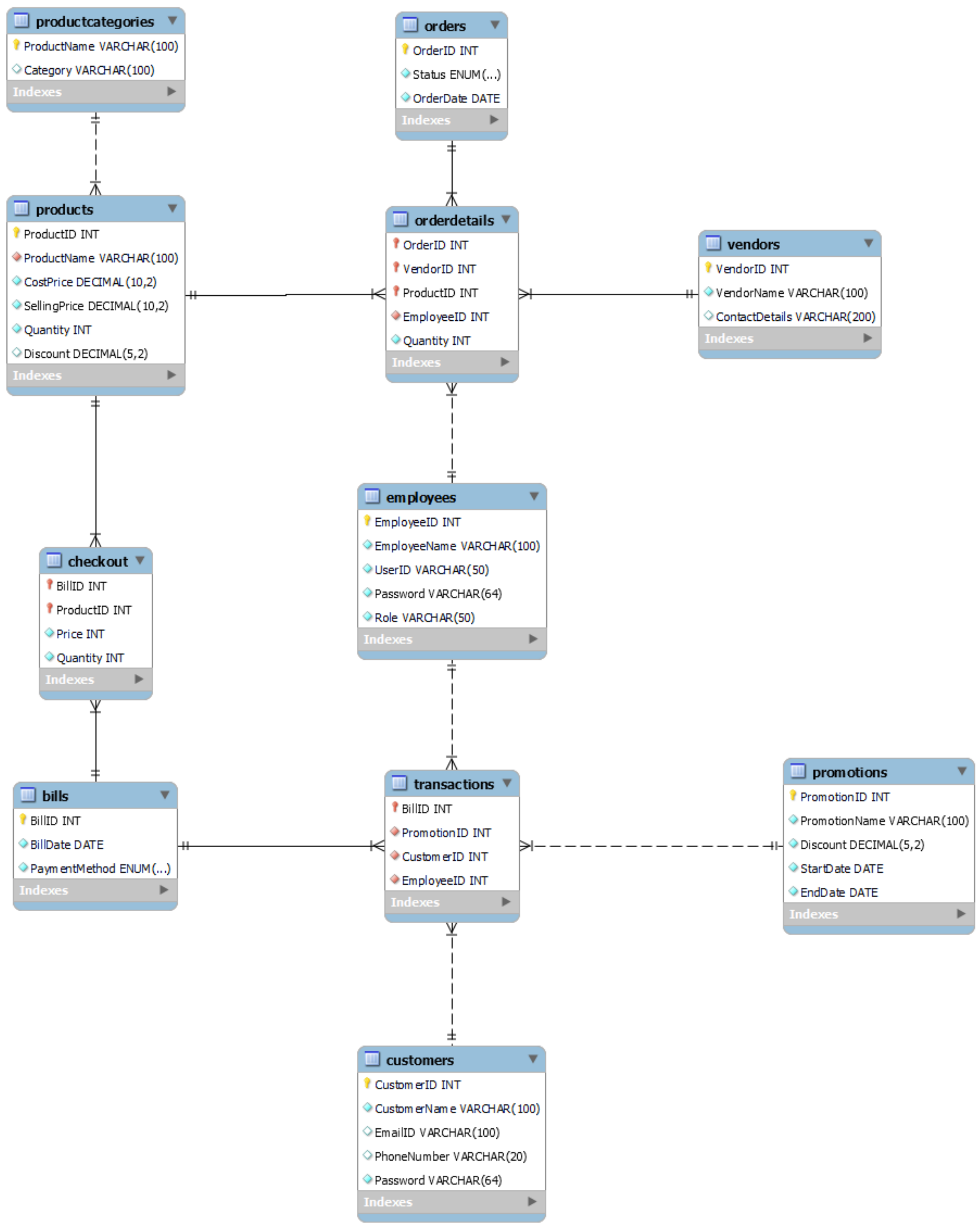
All the tables in the given database schema are at least in the Second Normal Form (2NF) of normalization. This means that all non-key attributes are fully functionally dependent on the primary key of their respective tables, and there are no partial dependencies.

However, the *Products* table originally had a transitive functional dependency, which violates the Third Normal Form (3NF) of normalization. The attribute *Category* was transitively dependent on the *ProductName* attribute, forming the transitive functional dependency *ProductName* → *Category*.

To eliminate this transitive dependency and bring the data into the Third Normal Form (3NF), we split the *Products* table into two separate tables:

Products table and **ProductCategories** table:

After this normalization step, all the tables in the database schema are in the Boyce-Codd Normal Form (BCNF. By normalizing the database to BCNF, we have achieved a well-structured and efficient database design that minimizes data redundancy, maintains data integrity, and facilitates easier data manipulation and retrieval.

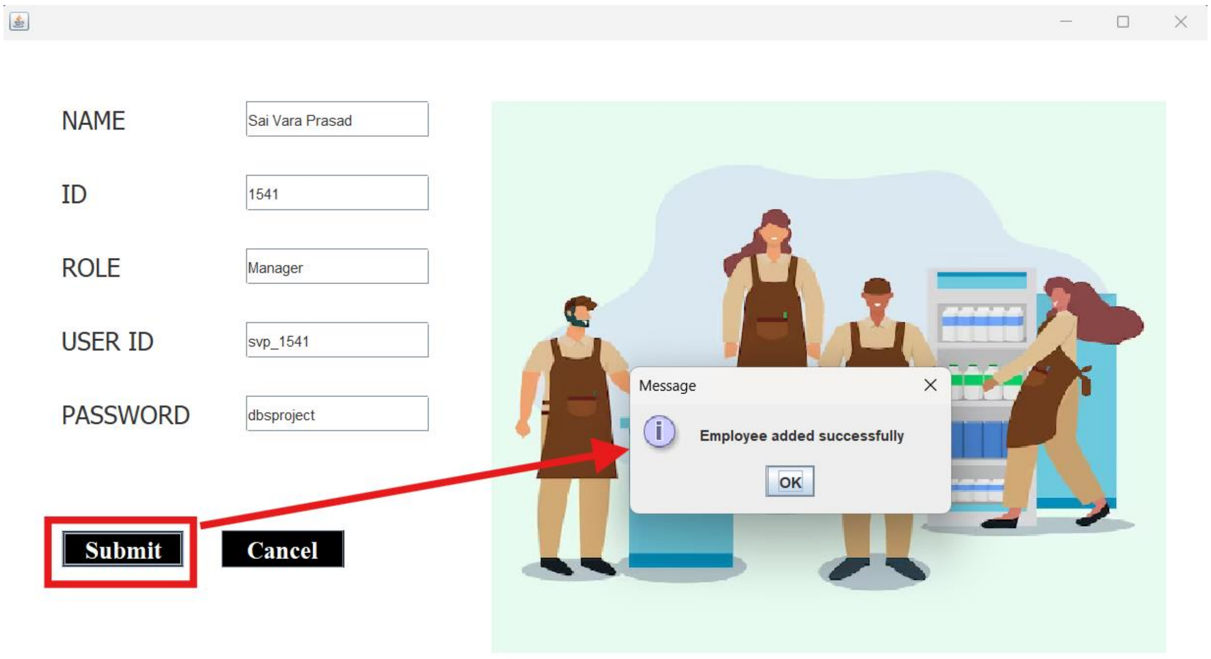


SCREEN SHOTS

1. AddEmployee Class:

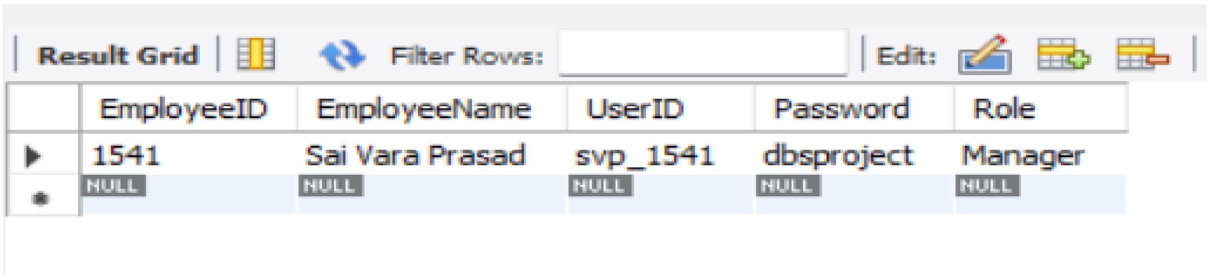
Using this class, we can add new employee details to our database. This class takes the following inputs: employee name, ID, role, UserID and password.
The table involved in this operation is **Employees Table**.
The query used to perform this operation is

insert into Employeesvalues (id,name,userID,password,role).


NAME: Sai Vara Prasad
ID: 1541
ROLE: Manager
USER ID: svp_1541
PASSWORD: dbsproject

Message
Employee added successfully

Figure 1 AddEmployee UI. Upon entering the details and clicking on Submit button, the dialog box is displayed.



EmployeeID	EmployeeName	UserID	Password	Role
1541	Sai Vara Prasad	svp_1541	dbsproject	Manager
NULL	NULL	NULL	NULL	NULL

Figure 2 We can see here that the employee details have been added to the table.

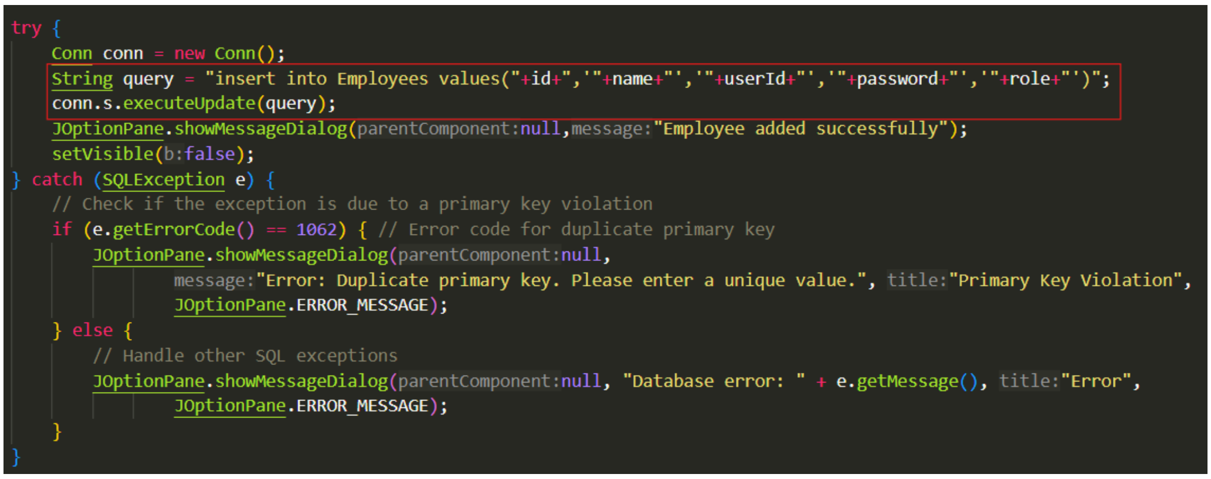
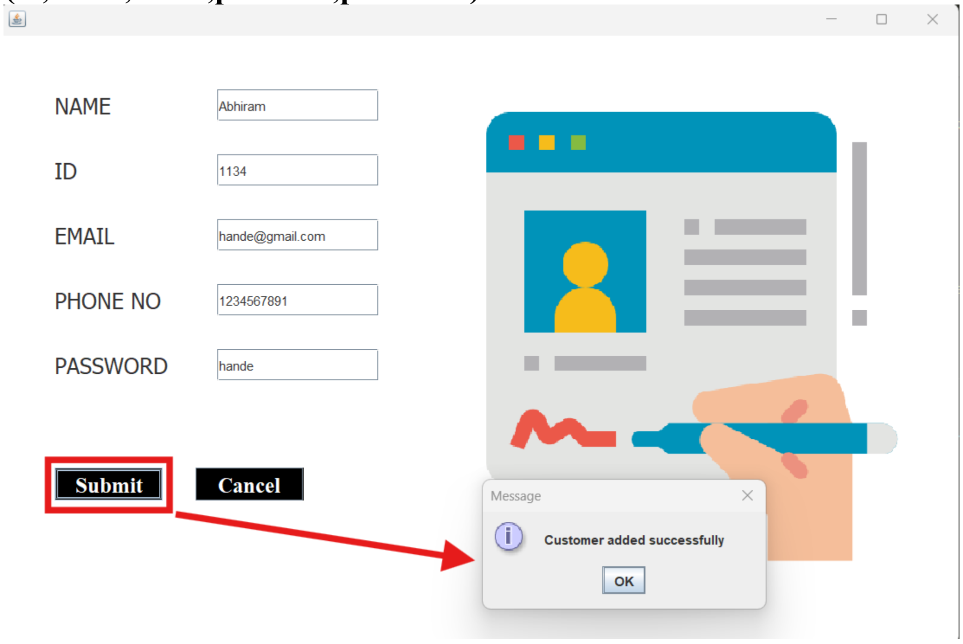

```
try {
    Conn conn = new Conn();
    String query = "insert into Employees values("+id+",'"+name+"','"+userId+"','"+password+"','"+role+"')";
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null,message:"Employee added successfully");
    setVisible(b:false);
} catch (SQLException e) {
    // Check if the exception is due to a primary key violation
    if (e.getErrorCode() == 1062) { // Error code for duplicate primary key
        JOptionPane.showMessageDialog(parentComponent:null,
                message:"Error: Duplicate primary key. Please enter a unique value.", title:"Primary Key Violation",
                JOptionPane.ERROR_MESSAGE);
    } else {
        // Handle other SQL exceptions
        JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 3. AddEmployee SQL Query

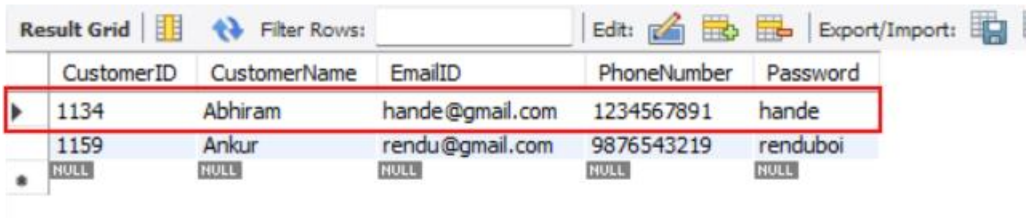
2. **AddCustomer Class:**

Using this class, we can add new customer details to our database. This class takes the following inputs: customer name, ID, email, phone number, and password.
The table involved in this operation is **Customers Table.**
The query used to perform this operation is **insert into Customers values (id,name,email,phoneno,password)**



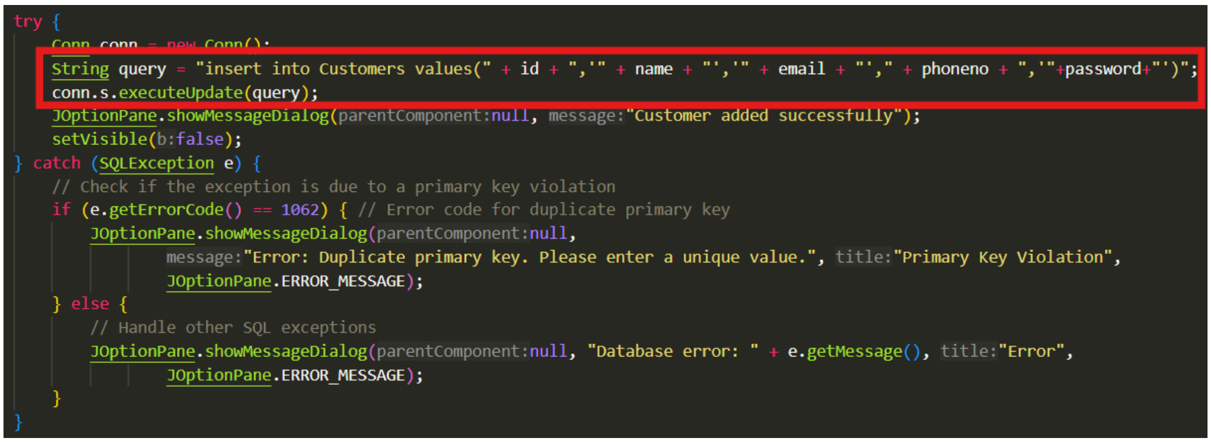
NAME	Abhiram
ID	1134
EMAIL	hande@gmail.com
PHONE NO	1234567891
PASSWORD	hande

Figure 4 AddCustomer UI. Upon entering the details and clicking on submit, a dialog box is shown saying the execution is successful.



CustomerID	CustomerName	EmailID	PhoneNumber	Password
1134	Abhiram	hande@gmail.com	1234567891	hande
1159	Ankur	rendu@gmail.com	9876543219	renduboi
NULL	NULL	NULL	NULL	NULL

Figure 5 We can see that the previously entered details have been updated to the Customers table.



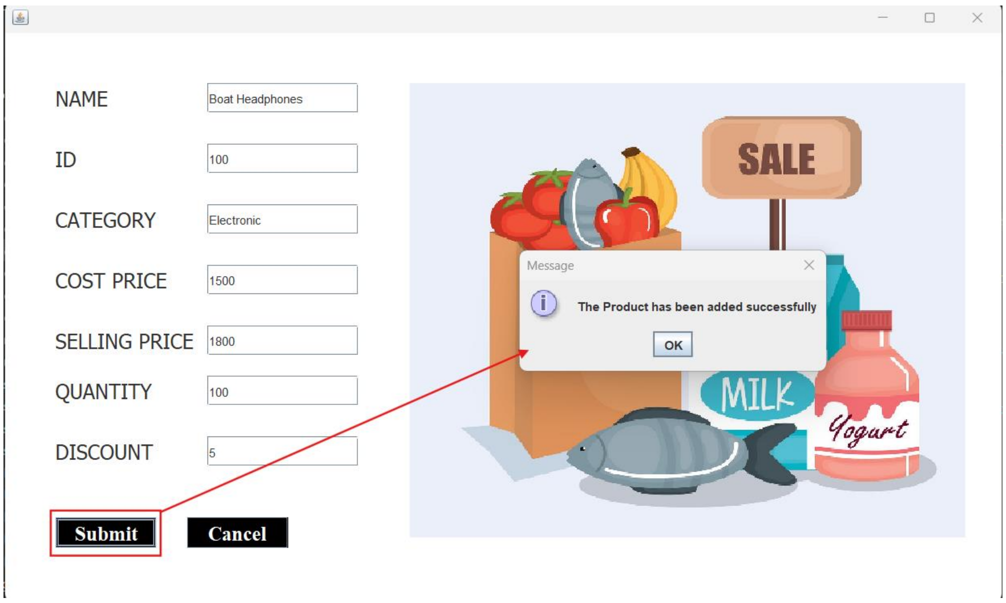
```
try {
    Conn conn = new Conn();
    String query = "insert into Customers values(" + id + ",'" + name + "','" + email + "'," + phoneno + ",'"+password+"')";
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"Customer added successfully");
    setVisible(b:false);
} catch (SQLException e) {
    // Check if the exception is due to a primary key violation
    if (e.getErrorCode() == 1062) { // Error code for duplicate primary key
        JOptionPane.showMessageDialog(parentComponent:null,
                message:"Error: Duplicate primary key. Please enter a unique value.", title:"Primary Key Violation",
                JOptionPane.ERROR_MESSAGE);
    } else {
        // Handle other SQL exceptions
        JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 6 AddCustomer SQL Query

3. AddItem Class:

Using this class, we can add new product details to our database (basically, to the inventory). This class takes the following inputs: product name, ID, product category, cost price of the product, selling price of the product, quantity of product purchased, discount to be set on each product.

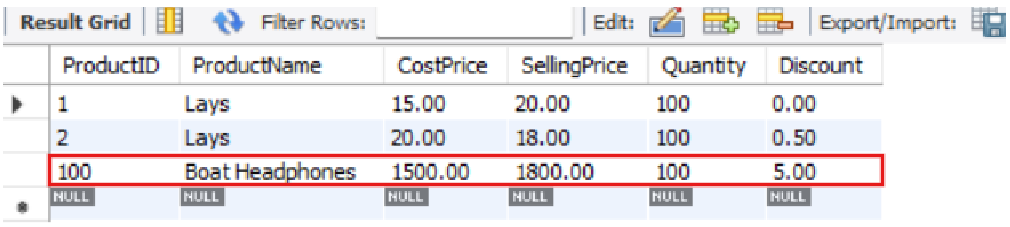
The tables involved in this operation are **Products Table and ProductCategories Table.** The queries used to perform this operation are
insert into ProductCategories values (name,category);
insert into Products values (id, name, costprice, sellingprice, quantity, discount);



NAME	Boat Headphones
ID	100
CATEGORY	Electronic
COST PRICE	1500
SELLING PRICE	1800
QUANTITY	100
DISCOUNT	5

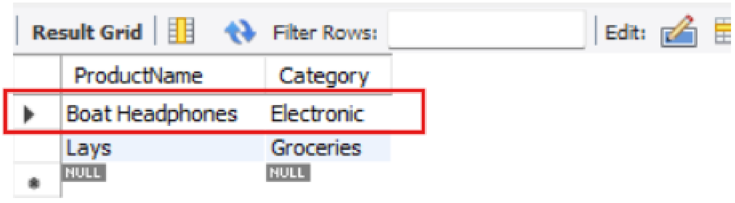
Message: The Product has been added successfully

Figure 7 AddItem UI. After entering the details, and clicking on the submit button, a confirmation dialog box will be shown.



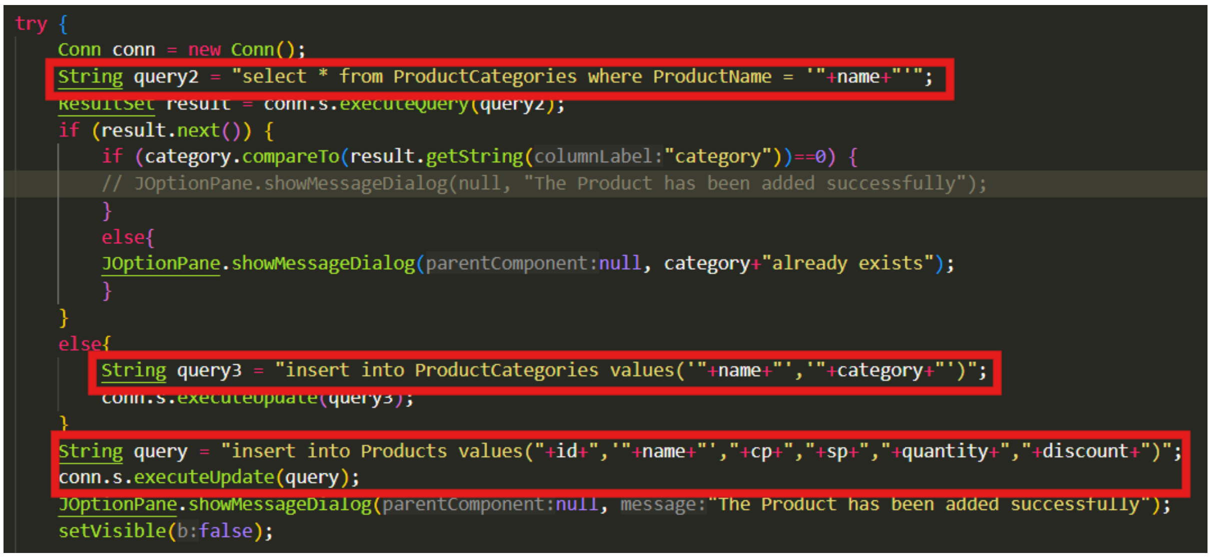
ProductID	ProductName	CostPrice	SellingPrice	Quantity	Discount
1	Lays	15.00	20.00	100	0.00
2	Lays	20.00	18.00	100	0.50
100	Boat Headphones	1500.00	1800.00	100	5.00
NULL	NULL	NULL	NULL	NULL	NULL

Figure 8 We can see that the product has been added to products table successfully



ProductName	Category
Boat Headphones	Electronic
Lays	Groceries
NULL	NULL

Figure 9 We can also see that the ProductCategories table has been updated

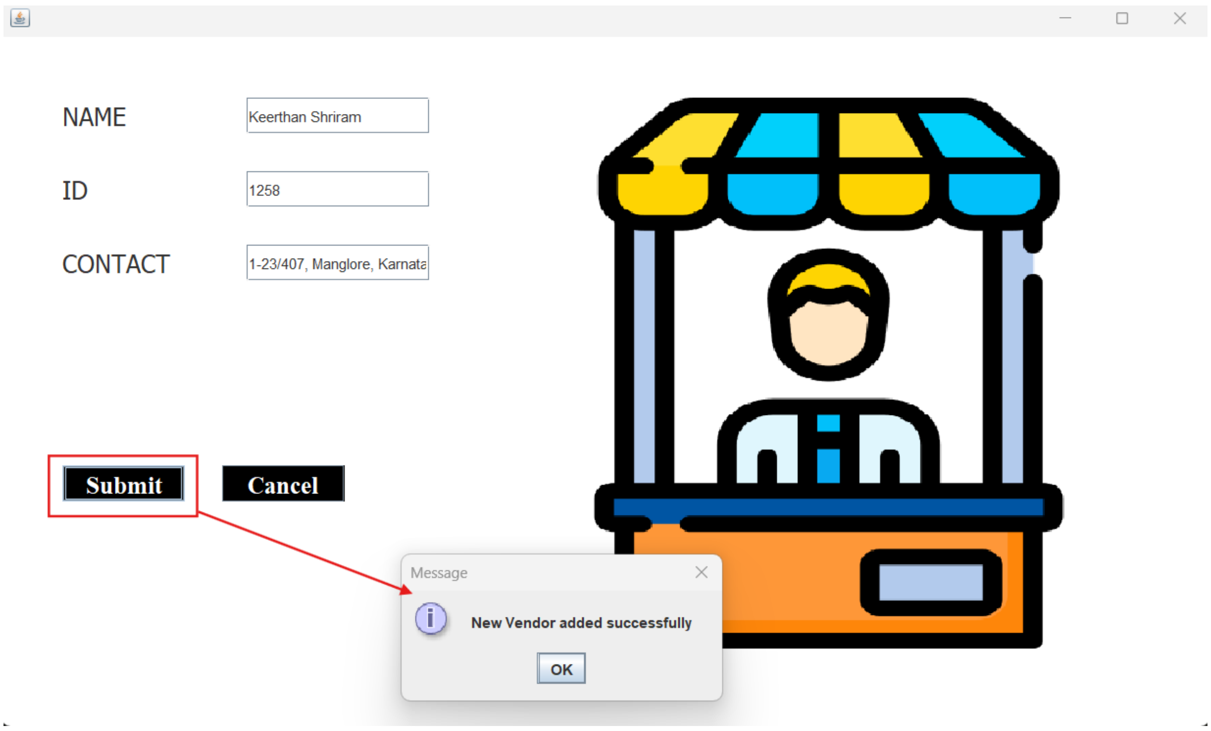


```
try {
    Conn conn = new Conn();
    String query2 = "select * from ProductCategories where ProductName = '"+name+"'";
    ResultSet result = conn.s.executeQuery(query2);
    if (result.next()) {
        if (category.compareTo(result.getString(columnLabel:"category"))==0) {
        // JOptionPane.showMessageDialog(null, "The Product has been added successfully");
        }
        else{
        JOptionPane.showMessageDialog(parentComponent:null, category+"already exists");
        }
    }
    else{
        String query3 = "insert into ProductCategories values('"+name+"','"+category+"')";
        conn.s.executeUpdate(query3);
    }
    String query = "insert into Products values("+id+",'"+name+"',"+cp+","+sp+","+quantity+","+discount+")";
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Product has been added successfully");
    setVisible(b:false);
```

Figure 10 AddItem SQL Query

4. AddVendor Class:

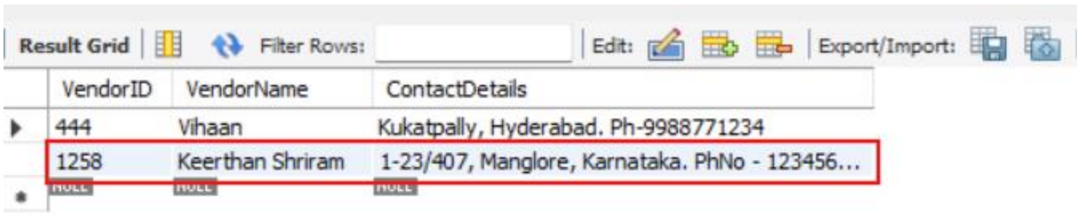
Using this class, we can add new vendor details to our database. This class takes the following inputs: vendor name, ID and other contact details. Here the contact details can include phone numbers, addresses etc.
The table involved in this operation is **Vendors Table.**
The query used to perform this operation is **insert into Vendors values (id,name,contact)**



NAME	Keerthan Shriram
ID	1258
CONTACT	1-23/407, Manglore, Karnata

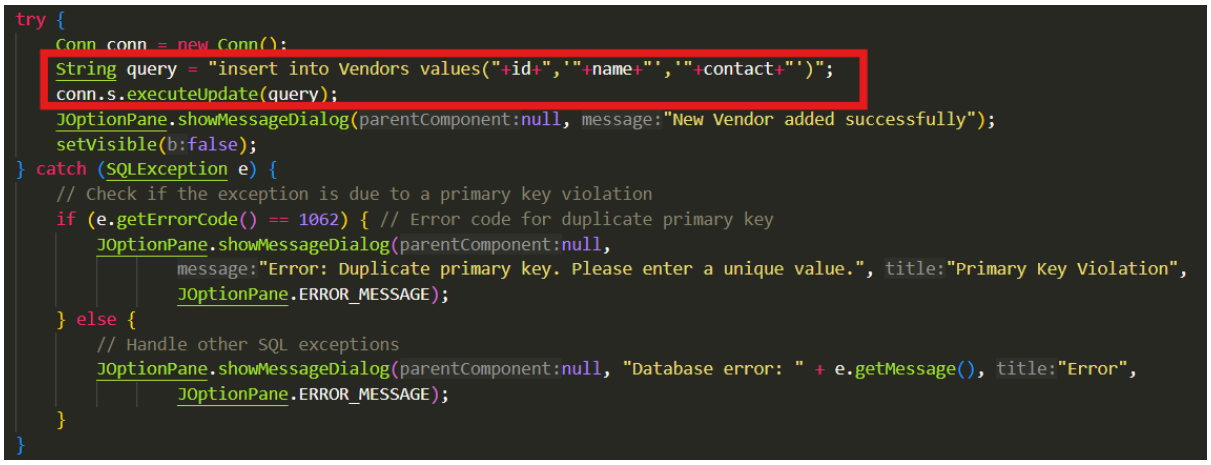
New Vendor added successfully

Figure 11 AddVendor UI. We have to enter the details and click on submit button. The message will be shown upon successful addition to database.



	VendorID	VendorName	ContactDetails
▶	444	Vihaan	Kukatpally, Hyderabad. Ph-9988771234
	1258	Keerthan Shriram	1-23/407, Manglore, Karnataka. PhNo - 123456...
*	NULL	NULL	NULL

Figure 12 Here we can see that the above entered details have been added to the vendors table successfully



```
try {
    Conn conn = new Conn();
    String query = "insert into Vendors values("+id+",'"+name+"','"+contact+"')";
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"New Vendor added successfully");
    setVisible(b:false);
} catch (SQLException e) {
    // Check if the exception is due to a primary key violation
    if (e.getErrorCode() == 1062) { // Error code for duplicate primary key
        JOptionPane.showMessageDialog(parentComponent:null,
                message:"Error: Duplicate primary key. Please enter a unique value.", title:"Primary Key Violation",
                JOptionPane.ERROR_MESSAGE);
    } else {
        // Handle other SQL exceptions
        JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

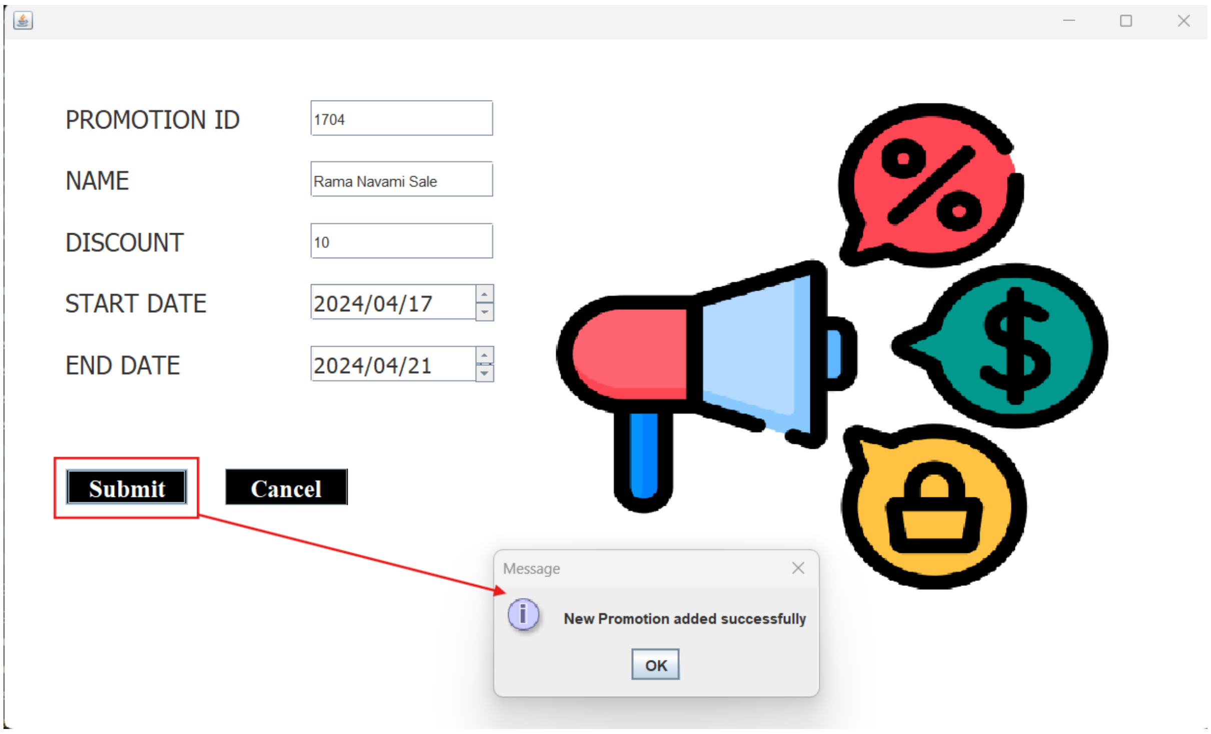
Figure 13 AddVendor SQL Query

5. **AddPromotion Class**

As it is a supermarket, it will have seasonal sales/offers or some other promotions. To keep a track of those, this class is created. Using this class, we can add new promotion details to our database. This class takes the following inputs: promotion name, ID, discount, start date, end date.

The table involved in this operation is **Promotions Table.**

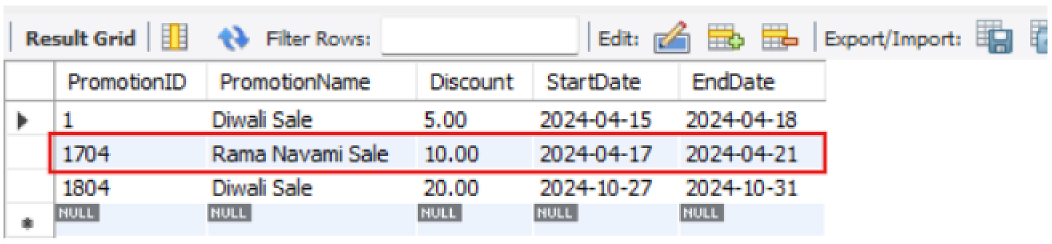
The query used to perform this operation is **insert into Promotions values (id, name, discount, start date, end date).**



PROMOTION ID	1704
NAME	Rama Navami Sale
DISCOUNT	10
START DATE	2024/04/17
END DATE	2024/04/21

New Promotion added successfully

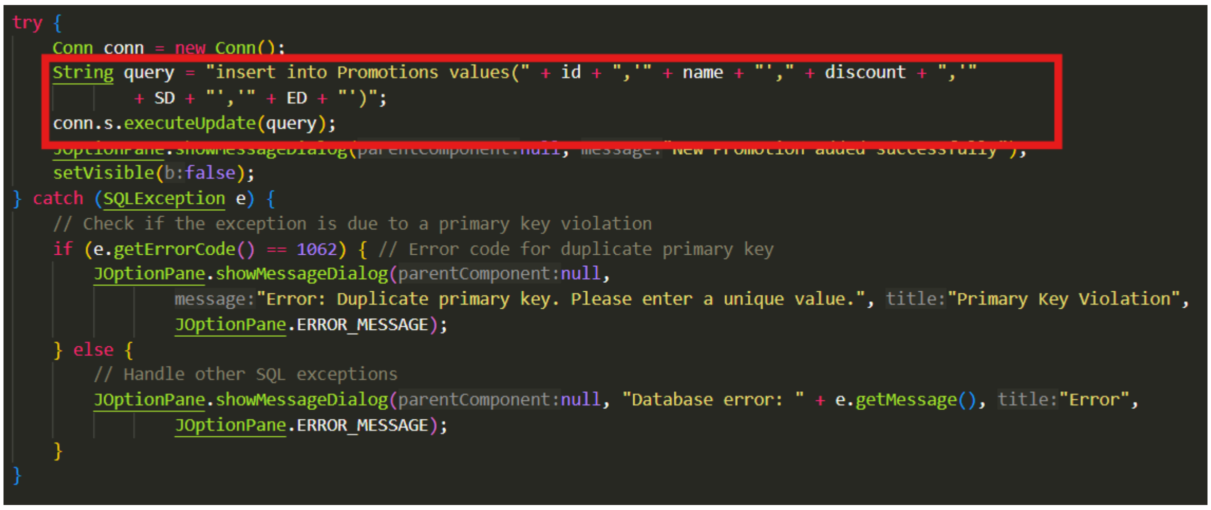
Figure 14 AddPromotions UI. We have to enter the details and click on submit button



	PromotionID	PromotionName	Discount	StartDate	EndDate
▶	1	Diwali Sale	5.00	2024-04-15	2024-04-18
	1704	Rama Navami Sale	10.00	2024-04-17	2024-04-21
	1804	Diwali Sale	20.00	2024-10-27	2024-10-31
*	NULL	NULL	NULL	NULL	NULL

Result Grid
Filter Rows:
Edit:
Export/Import:

Figure 15 We can see that the details which we entered, got updated successfully to the Promotions table

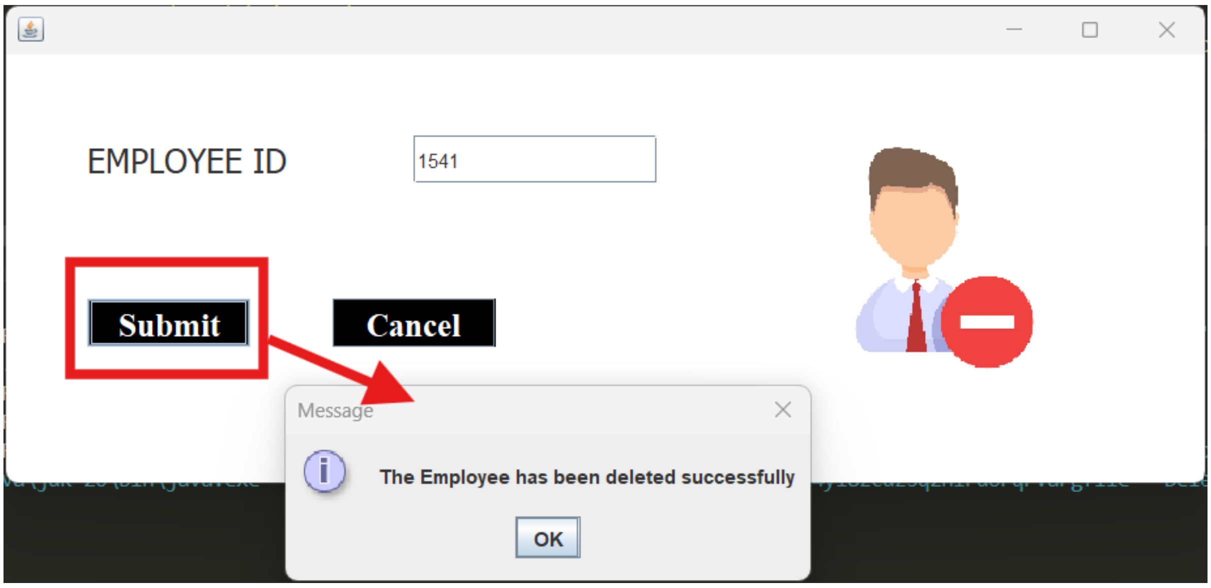


```
try {
    Conn conn = new Conn();
    String query = "insert into Promotions values(" + id + ",'" + name + "'," + discount + ",'"
            + SD + "','" + ED + "')";
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"New Promotion added successfully");
    setVisible(b:false);
} catch (SQLException e) {
    // Check if the exception is due to a primary key violation
    if (e.getErrorCode() == 1062) { // Error code for duplicate primary key
        JOptionPane.showMessageDialog(parentComponent:null,
                message:"Error: Duplicate primary key. Please enter a unique value.", title:"Primary Key Violation",
                JOptionPane.ERROR_MESSAGE);
    } else {
        // Handle other SQL exceptions
        JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error",
                JOptionPane.ERROR_MESSAGE);
    }
}
```

Figure 16 AddPromotions SQL Query

6. DeleteEmployee Class:

Using this class, we can erase the information about an employee from our database permanently. The **primary key for the Employees table is EmployeeID**. With the help of that we performed the deletion operation.
The table involved in this operation is **Employees Table**.
The SQL query is :
delete from Employees where EmployeeID = {id entered}

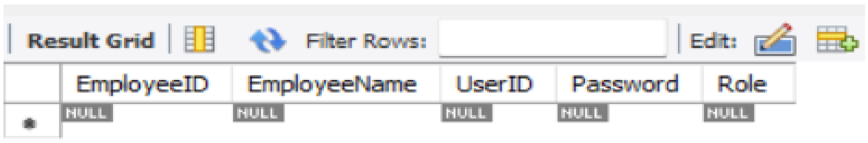


EMPLOYEE ID 1541

Message

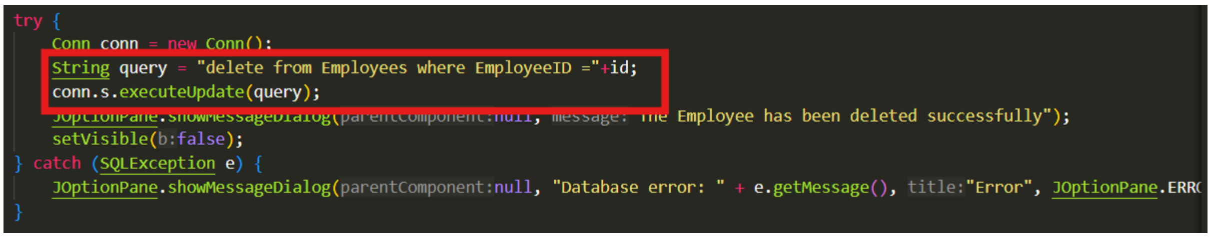
The Employee has been deleted successfully

Figure 17 DeleteEmployee UI. Upon entering the ID details and clicking submit, this dialog box will be shown



	EmployeeID	EmployeeName	UserID	Password	Role
●	NULL	NULL	NULL	NULL	NULL

Figure 18 Here we can see that previously entered employee details are deleted from the table.

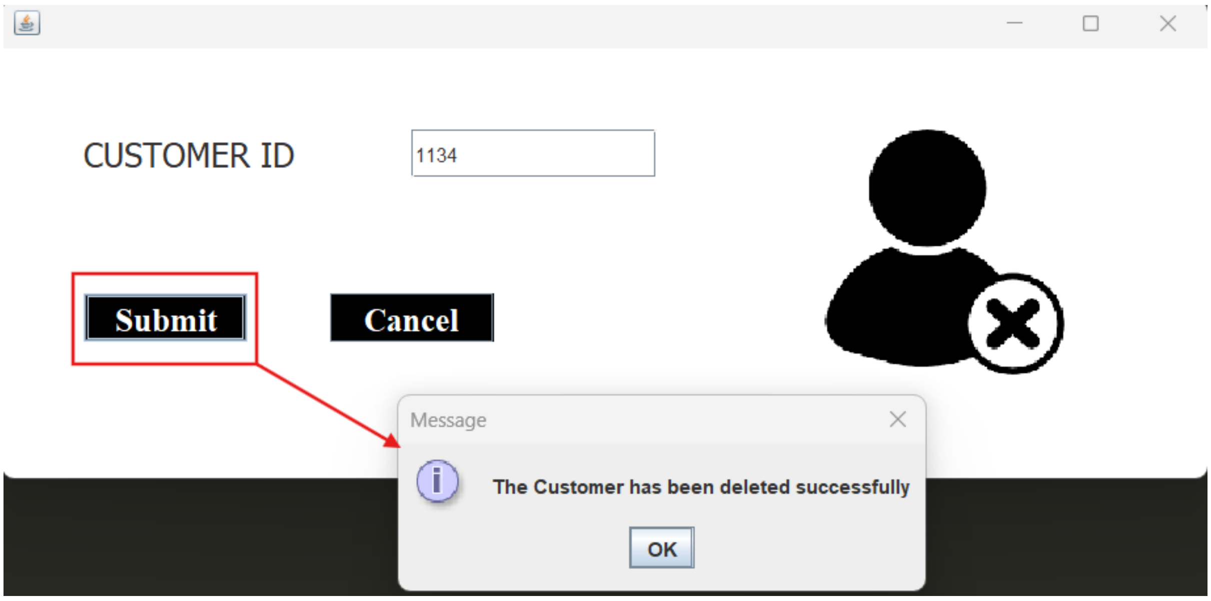


```
try {
    Conn conn = new Conn();
    String query = "delete from Employees where EmployeeID ="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message: "The Employee has been deleted successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERR
}
```

Figure 19 DeleteEmployee SQL Query

7. **DeleteCustomer Class**

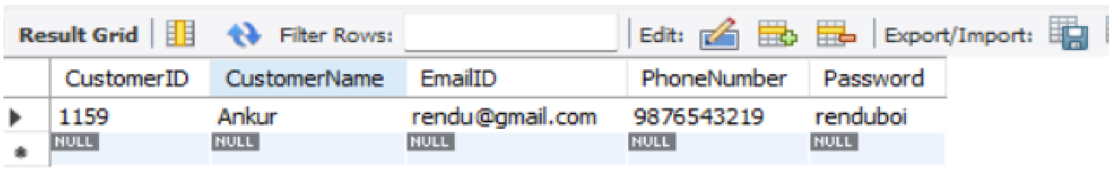
Using this class, we can erase the information about a customer from our database permanently. The **primary key for the Customers table is CustomerID**. With the help of that we performed the deletion operation.
The table involved in this operation is **Customers Table**.
The SQL query is **delete from Customers where CustomerID = {id entered}.**



CUSTOMER ID: 1134

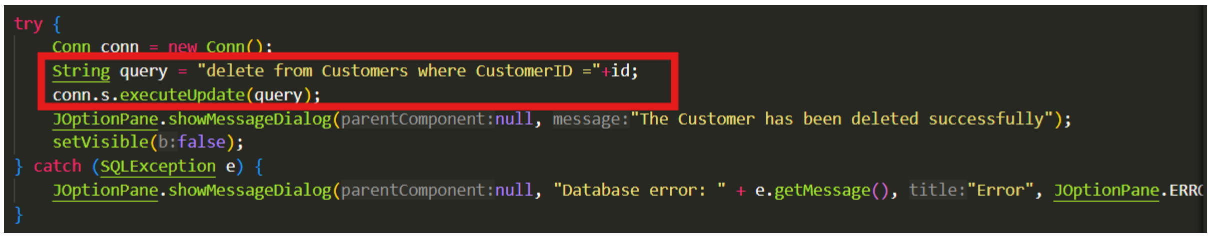
The Customer has been deleted successfully

Figure 20 DeleteCustomer UI. We have to enter the customer ID and their details will be deleted from database permanently



	CustomerID	CustomerName	EmailID	PhoneNumber	Password
▶	1159	Ankur	rendu@gmail.com	9876543219	renduboi
●	NULL	NULL	NULL	NULL	NULL

Figure 21 As you can see, the customer with 1134 ID has been deleted from table after executing the deletion operation

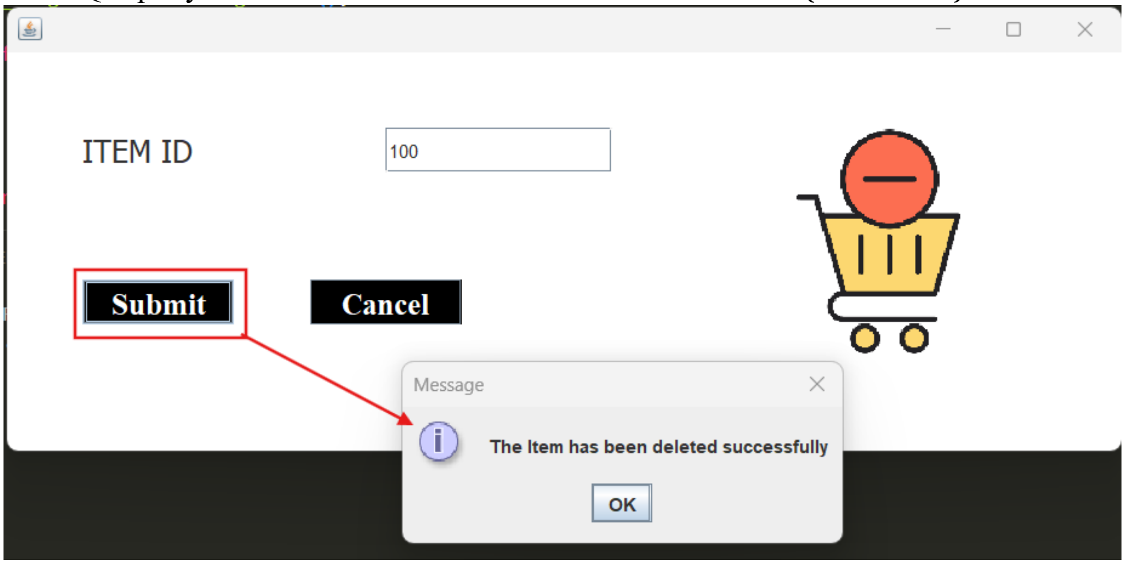


```
try {
    Conn conn = new Conn();
    String query = "delete from Customers where CustomerID ="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Customer has been deleted successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERR
}
```

Figure 22 DeleteCustomer SQL Query

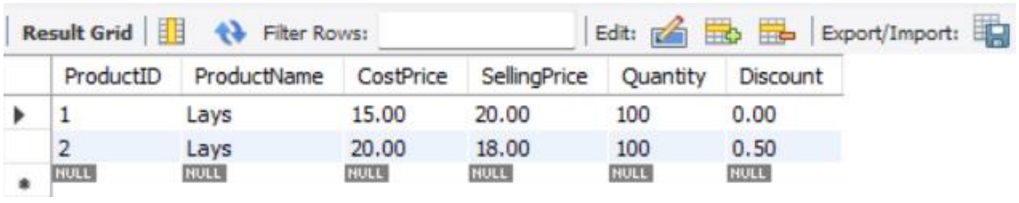
8. DeleteItem Class

Using this class, we can erase the information about a product from our database permanently. The **primary key for the Products table is ProductID**. With the help of that we performed the deletion operation.
The table involved in this operation is **Products Table**.
The SQL query is **delete from Products where ProductID = {id entered}.**



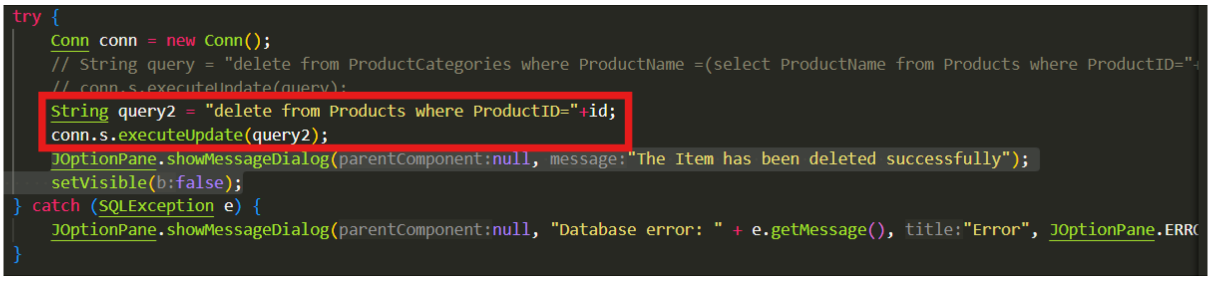
The Item has been deleted successfully

Figure 23 DeleteItem UI. We have to enter the Product ID and click on submit button



ProductID	ProductName	CostPrice	SellingPrice	Quantity	Discount
1	Lays	15.00	20.00	100	0.00
2	Lays	20.00	18.00	100	0.50
NULL	NULL	NULL	NULL	NULL	NULL

Figure 24 Here we can see that the product with ID=100 has been deleted



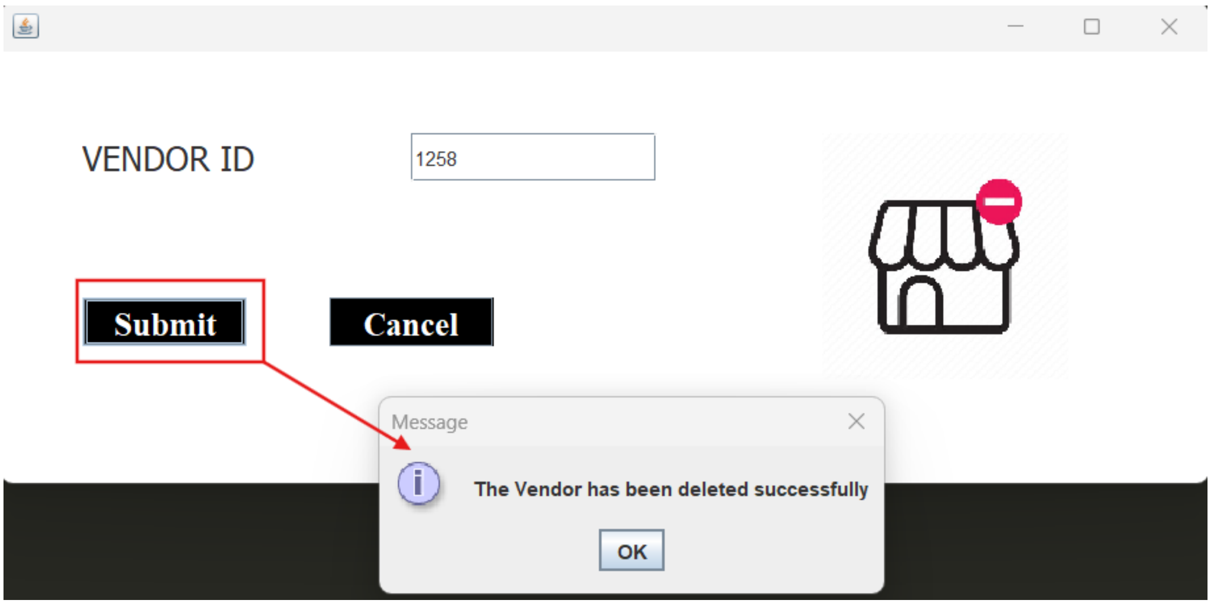
```
try {
    Conn conn = new Conn();
    // String query = "delete from ProductCategories where ProductName =(select ProductName from Products where ProductID="+
    // conn.s.executeUpdate(query);
    String query2 = "delete from Products where ProductID="+id;
    conn.s.executeUpdate(query2);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Item has been deleted successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERR
}
```

Figure 25 DeleteItem SQL Query

9. DeleteVendor Class

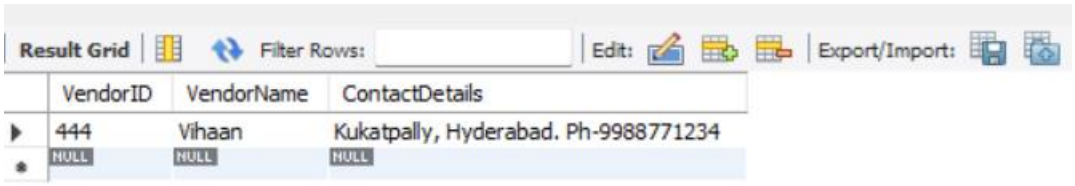
Using this class, we can erase the information about a vendor from our database permanently. The **primary key for the Vendors table is VendorID**. With the help of that we performed the deletion operation.

The table involved in this operation is **Vendors Table**.
The SQL query is **delete from Vendors where VendorID = {id entered}.**



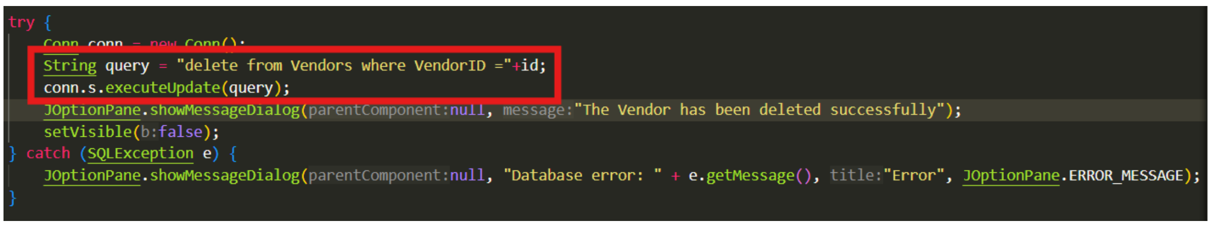
The Vendor has been deleted successfully

Figure 26 DeleteVendor UI. We have to enter the vendor id and click on submit. A success message is shown upon smooth execution of query.



VendorID	VendorName	ContactDetails
444	Vihaan	Kukatpally, Hyderabad. Ph-9988771234
NULL	NULL	NULL

Figure 27 Here we can see that the vendor with ID=1258 has been deleted from the table

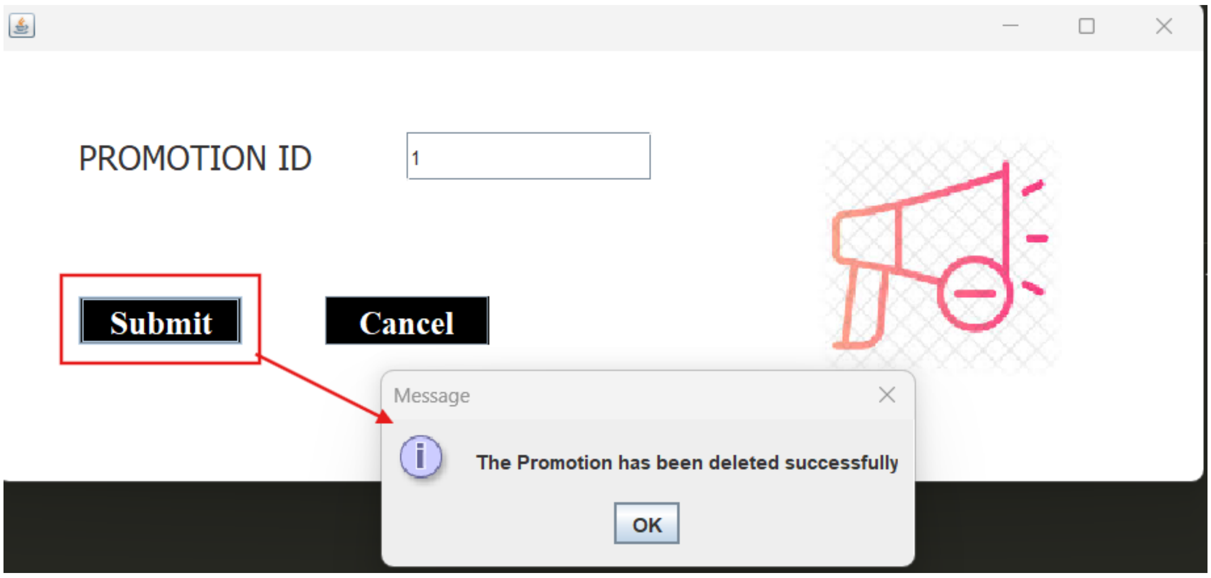


```
try {
    Conn conn = new Conn();
    String query = "delete from Vendors where VendorID ="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Vendor has been deleted successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERROR_MESSAGE);
}
```

Figure 28 DeleteVendor SQL Query

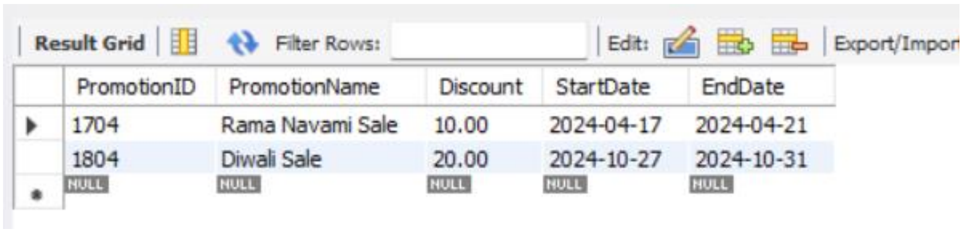
10. **DeletePromotion Class**

Using this class, we can erase the information about a promotion from our database permanently. The **primary key for the Promotions table is PromotionID**. With the help of that we performed the deletion operation.
The table involved in this operation is **Promotions Table**.
The SQL query is **delete from Promotions where PromotionID = {id entered}.**



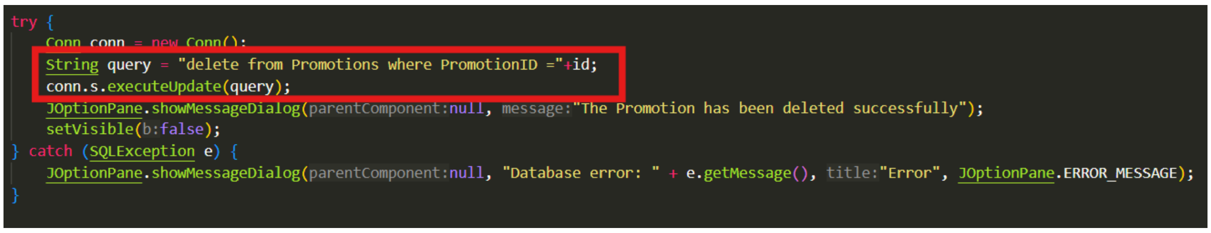
The Promotion has been deleted successfully

Figure 29 DeletePromotion UI. We have to enter the PromotionID of the promotion which we want to delete and click on submit.



	PromotionID	PromotionName	Discount	StartDate	EndDate
▶	1704	Rama Navami Sale	10.00	2024-04-17	2024-04-21
	1804	Diwali Sale	20.00	2024-10-27	2024-10-31
*	NULL	NULL	NULL	NULL	NULL

Figure 30 We can see that the promotion with ID=1 has been deleted

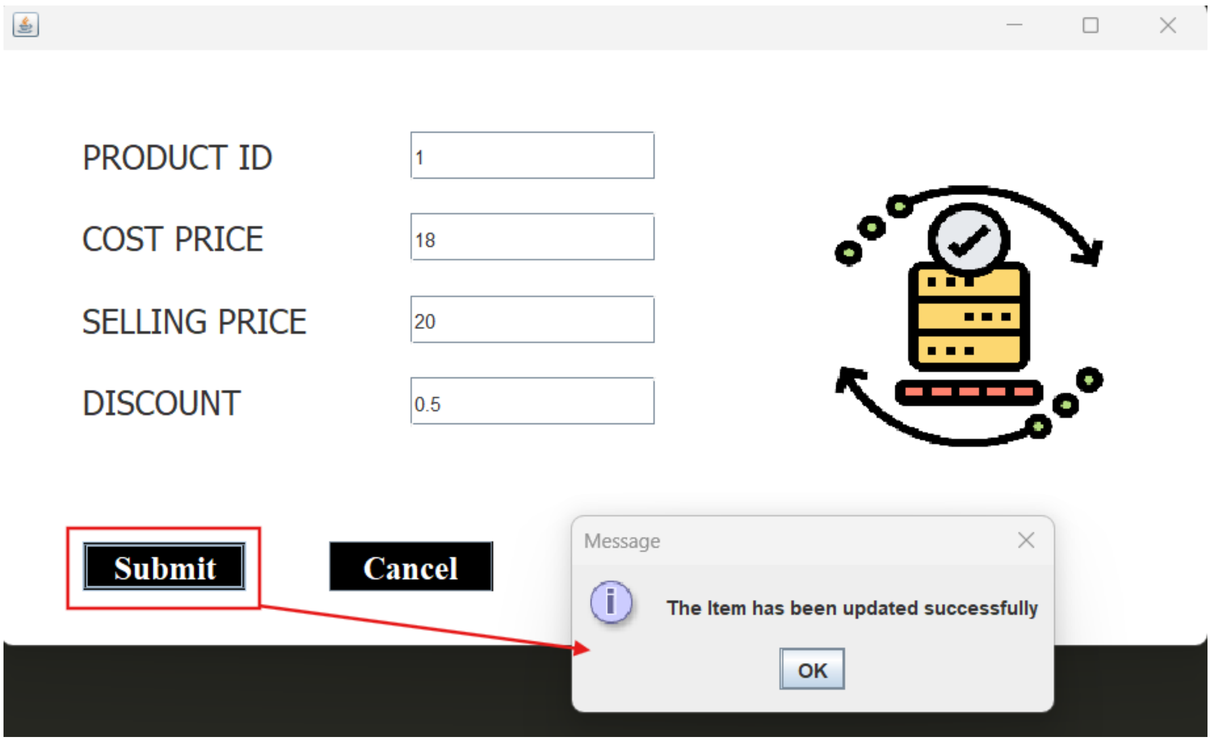


```
try {
    Conn conn = new Conn();
    String query = "delete from Promotions where PromotionID ="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Promotion has been deleted successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERROR_MESSAGE);
}
```

Figure 31 DeletePromotion SQL Query

11. UpdateItem Class

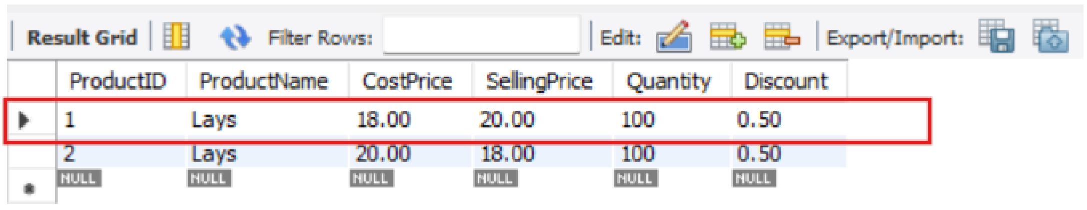
Sometimes, we like to update the details about a particular product or promotions etc. So perform that operation, this class has been created. This class will the Products Table. With the help of **ProductID as primary key**, we can perform the update operation. Through this class we can update the cost price, selling price, discount of a particular product.
The table involved in this operation is **Products Table**.
The SQL query is **update Products set CostPrice = {entered CP}, SellingPrice = {entered SP}, Discount = {entered discount} where ProductID = {entered ID}.**



PRODUCT ID	1
COST PRICE	18
SELLING PRICE	20
DISCOUNT	0.5

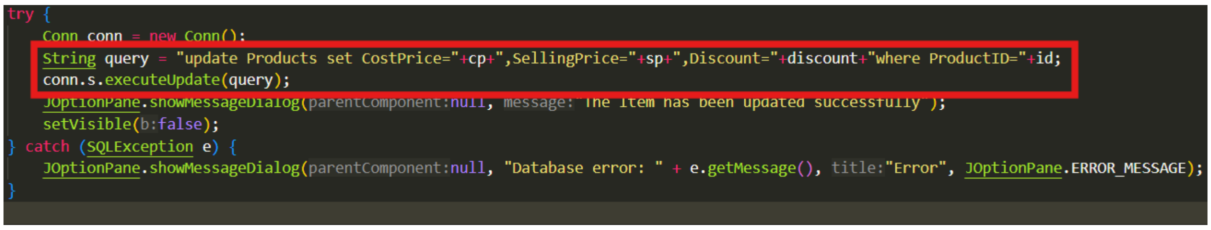
The Item has been updated successfully

Figure 32 UpdateItem UI. Here we tried to update the cost price and discount of product with ID=1



ProductID	ProductName	CostPrice	SellingPrice	Quantity	Discount
1	Lays	18.00	20.00	100	0.50
2	Lays	20.00	18.00	100	0.50
NULL	NULL	NULL	NULL	NULL	NULL

Figure 33 Here we can see that product with ID=1's, the cost price has been changed from 15 to 18 and discount has been changed from 0 to 0.5%

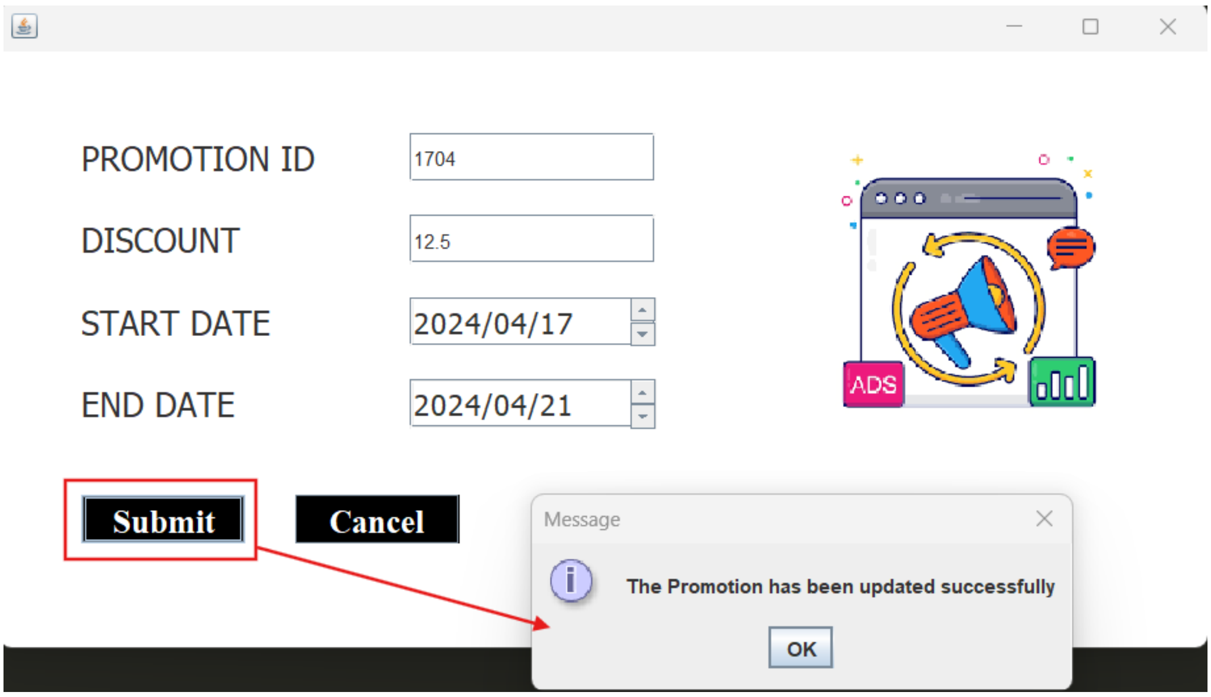


```
try {
    Conn conn = new Conn();
    String query = "update Products set CostPrice="+cp+",SellingPrice="+sp+",Discount="+discount+"where ProductID="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Item has been updated successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERROR_MESSAGE);
}
```

Figure 34 UpdateItem SQL query

12. UpdatePromotions Class

This class has been created for the similar reasons mentioned above. The store might want to extend the sale period or change the discount amount etc. To perform those operations, this class has been created. Here **PromotionID (primary key)** is used to execute the query.
The table involved in this query is **Promotions Table**.
The SQL query is **update Promotions set Discount = {entered discount}, StartDate = {entered date}, EndDate = {entered date} where PromotionID = {entered ID}.**


PROMOTION ID 1704

DISCOUNT 12.5

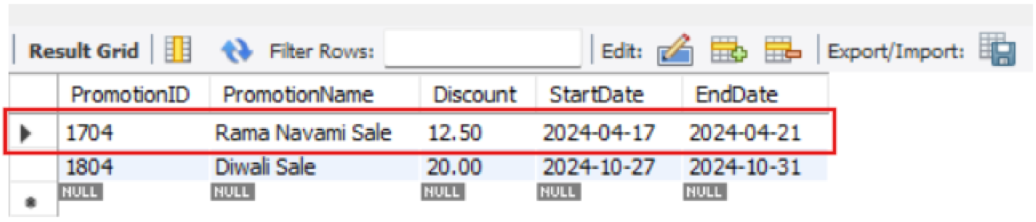
START DATE 2024/04/17

END DATE 2024/04/21

Message

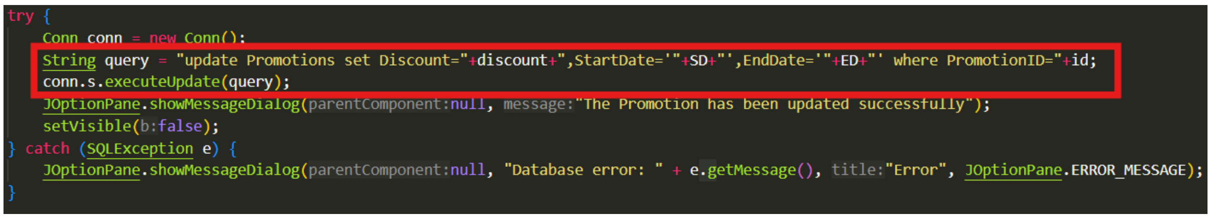
The Promotion has been updated successfully

Figure 35 UpdatePromotion UI. Here we are trying to update the discount amount of the promotion with ID=1704



PromotionID	PromotionName	Discount	StartDate	EndDate
1704	Rama Navami Sale	12.50	2024-04-17	2024-04-21
1804	Diwali Sale	20.00	2024-10-27	2024-10-31
NULL	NULL	NULL	NULL	NULL

Figure 36 Here we can see that the discount price of sale with ID=1704 has been changed from 10% to 12.5%



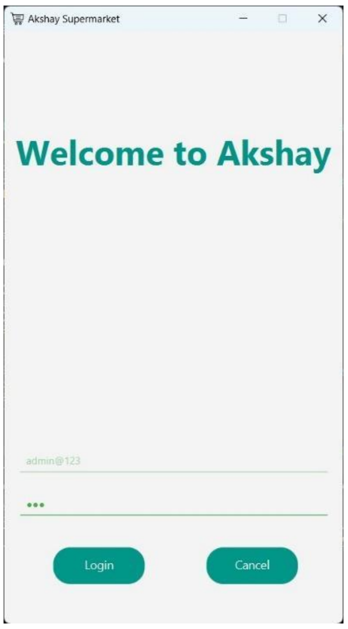
```
try {
    Conn conn = new Conn();
    String query = "update Promotions set Discount="+discount+",StartDate='"+SD+"',EndDate='"+ED+"' where PromotionID="+id;
    conn.s.executeUpdate(query);
    JOptionPane.showMessageDialog(parentComponent:null, message:"The Promotion has been updated successfully");
    setVisible(b:false);
} catch (SQLException e) {
    JOptionPane.showMessageDialog(parentComponent:null, "Database error: " + e.getMessage(), title:"Error", JOptionPane.ERROR_MESSAGE);
}
```

Figure 37 UpdatePromotion SQL Query

13. Login

Here **EmployeeID and CustomerID** are used to execute the query. The tables involved in this query are **Employees** and **Customers**. The SQL query is:

SELECT * FROM employees WHERE UserID = {username} AND Password = {password};
SELECT * FROM customers WHERE EmailID = {email} AND Password = {password};



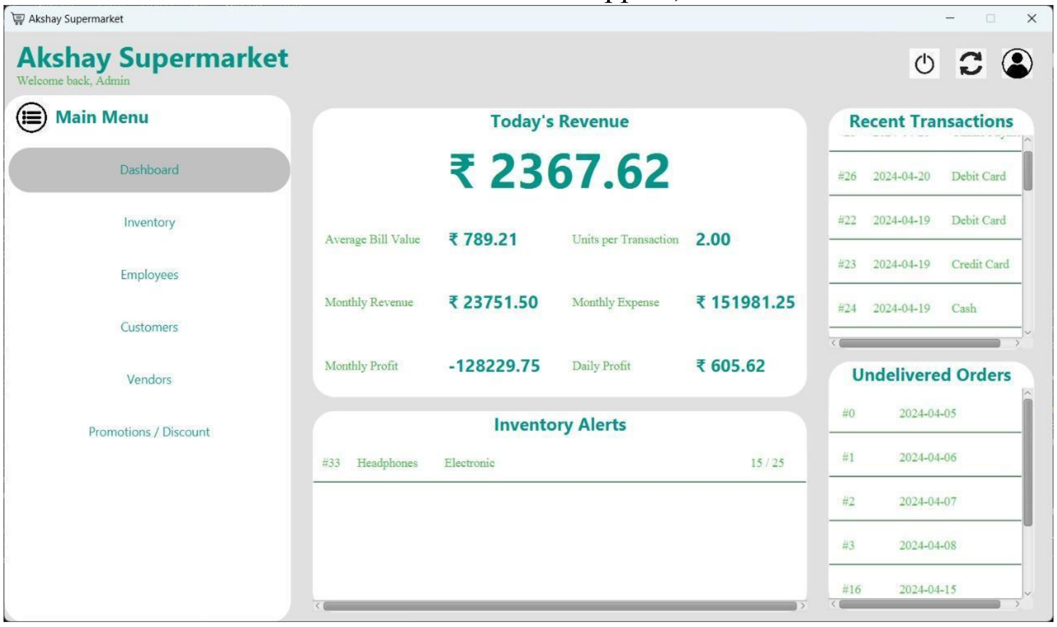
14. Employee Dashboard

Here all tables are used to execute the various queries. The SQL Queries are

- To calculate the total revenue of the day
  SELECT SUM(Checkout.Price * Checkout.Quantity * (1 - COALESCE(Promotions.Discount, 0) / 100)) AS TotalRevenue FROM Checkout JOIN Bills ON Checkout.BillID = Bills.BillID LEFT JOIN Transactions ON Bills.BillID = Transactions.BillID LEFT JOIN Promotions ON Transactions.PromotionID = Promotions.PromotionID WHERE Bills.BillDate = CURDATE();
- To calculate the average bill value
  SELECT AVG(total_bill) AS ATV FROM (SELECT Bills.BillID, SUM(Checkout.Price * Checkout.Quantity * (1 - COALESCE(Promotions.Discount, 0) / 100)) AS total_bill FROM Checkout JOIN Bills ON Checkout.BillID = Bills.BillID LEFT JOIN Transactions ON Bills.BillID = Transactions.BillID LEFT JOIN Promotions ON Transactions.PromotionID = Promotions.PromotionID WHERE Bills.BillDate = CURDATE() GROUP BY BillID) AS subquery;
- To Print the average of items purchased in all transactions done today
  SELECT AVG(item_count) AS UPT FROM (SELECT Bills.BillID, COUNT(*) AS item_count FROM Checkout JOIN Bills ON Checkout.BillID = Bills.BillID WHERE Bills.BillDate = CURDATE() GROUP BY BillID) AS subquery;
- To calculate the profit generated that day
  SELECT (SUM(Checkout.Quantity * Checkout.Price * (1 - COALESCE(Promotions.Discount, 0) / 100)) - SUM(Checkout.Quantity * Products.CostPrice)) AS GrossProfit FROM Checkout JOIN Bills ON Checkout.BillID = Bills.BillID JOIN Products ON Checkout.ProductID = Products.ProductID LEFT JOIN Transactions ON Bills.BillID = Transactions.BillID LEFT JOIN Promotions ON Transactions.PromotionID = Promotions.PromotionID WHERE Bills.BillDate = CURDATE();
- To calculate the revenue generated that month
  SELECT SUM(Checkout.Price * Checkout.Quantity * (1 - COALESCE(Promotions.Discount, 0) / 100)) AS Revenue FROM Checkout JOIN Bills ON Checkout.BillID = Bills.BillID LEFT JOIN Transactions ON Bills.BillID =

Transactions.BillIDLEFT JOIN Promotions ON Transactions.PromotionID = Promotions.PromotionID WHERE YEAR(BillDate) = YEAR(CURDATE()) AND MONTH(BillDate) = MONTH(CURDATE());

- To calculate the monthly expense
  SELECT SUM(od.Quantity * products.CostPrice) AS TotalExpenditure FROM Orders JOIN OrderDetails od ON Orders.OrderID = od.OrderID JOIN Products ON od.ProductID = products.ProductID WHERE MONTH(orders.OrderDate) = MONTH(CURDATE()) AND YEAR(orders.OrderDate) = YEAR(CURDATE());
- To get all items whose quantity are below the threshold
  SELECT * FROM Products NATURAL JOIN ProductCategories WHERE Quantity < 25
- To get recent transactions
  SELECT * FROM Transactions NATURAL JOIN Bills ORDER BY bills.BillDate DESC LIMIT 20;
- To get undelivered orders
  SELECT * FROM orders WHERE Status='Shipped';


Akshay Supermarket

Akshay Supermarket
Welcome back, Admin

Main Menu
Dashboard
Inventory
Employees
Customers
Vendors
Promotions / Discount

Today's Revenue
₹ 2367.62

Average Bill Value	₹ 789.21	Units per Transaction	2.00
Monthly Revenue	₹ 23751.50	Monthly Expense	₹ 151981.25
Monthly Profit	-128229.75	Daily Profit	₹ 605.62

Inventory Alerts

#33	Headphones	Electronic	15 / 25

Recent Transactions

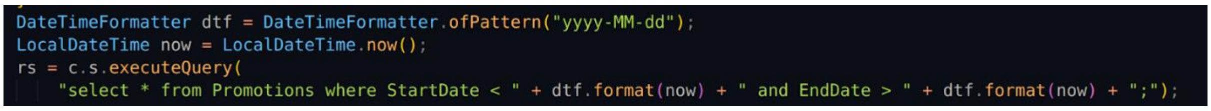
#26	2024-04-20	Debit Card
#22	2024-04-19	Debit Card
#23	2024-04-19	Credit Card
#24	2024-04-19	Cash

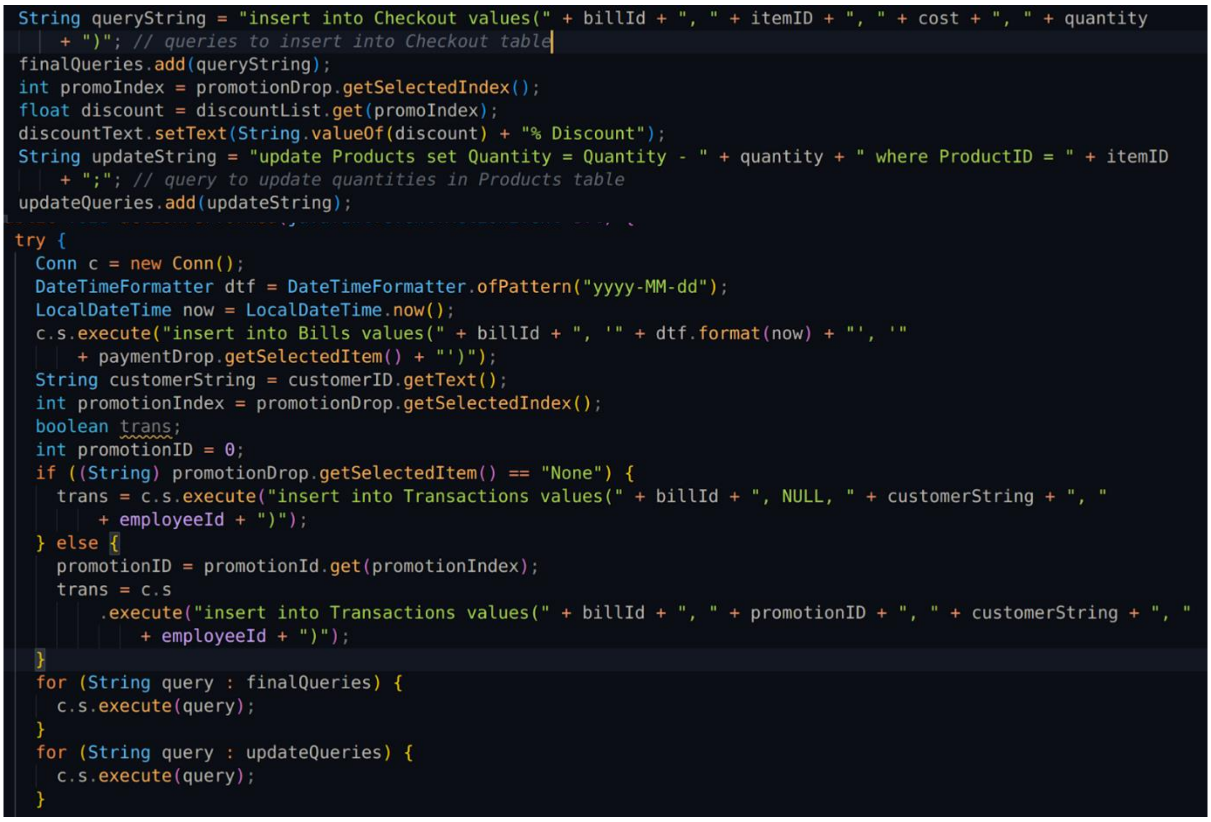
Undelivered Orders

#0	2024-04-05
#1	2024-04-06
#2	2024-04-07
#3	2024-04-08
#16	2024-04-15

15. MakeOrder Class:

This class is used to make an order for the customer and produce the total for the order. It adds entries to the Checkout, Bills and Transactions tables and also updates the quantities in the Products table after the order is successful. Here we provide the UI to add each item to the cart using the Add Item button and display the Grand Total for the Bill. The user has to then select the payment method and enter in the customer ID and promotion and then confirm the order. We store the query strings once each item is added to cart by using an ArrayList data structure as items can be inserted into them dynamically. We fetch data from the Products table to get the Product names, quantities and ID's and also from the Promotions Table to get the list of promotions available at the current time. We do this using the queries as shown in the image below.

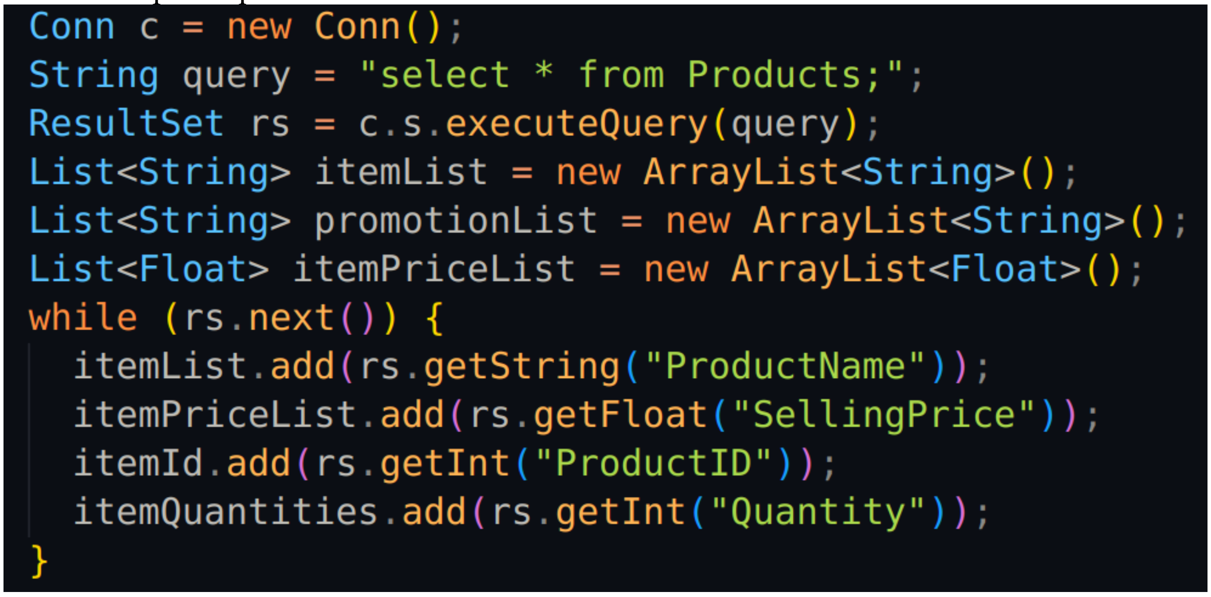

```
DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");
LocalDateTime now = LocalDateTime.now();
rs = c.s.executeQuery(
    "select * from Promotions where StartDate < " + dtf.format(now) + " and EndDate > " + dtf.format(now) + ";");
```



```java
String queryString = "insert into Checkout values(" + billId + ", " + itemID + ", " + cost + ", " + quantity
        + ")"; // queries to insert into Checkout table
finalQueries.add(queryString);
int promoIndex = promotionDrop.getSelectedIndex();
float discount = discountList.get(promoIndex);
discountText.setText(String.valueOf(discount) + "% Discount");
String updateString = "update Products set Quantity = Quantity - " + quantity + " where ProductID = " + itemID
        + ";"; // query to update quantities in Products table
updateQueries.add(updateString);
try {
    Conn c = new Conn();
    DateTimeFormatter dtf = DateTimeFormatter.ofPattern("yyyy-MM-dd");
    LocalDateTime now = LocalDateTime.now();
    c.s.execute("insert into Bills values(" + billId + ", '" + dtf.format(now) + "', '"
            + paymentDrop.getSelectedItem() + "')");
    String customerString = customerID.getText();
    int promotionIndex = promotionDrop.getSelectedIndex();
    boolean trans;
    int promotionID = 0;
    if ((String) promotionDrop.getSelectedItem() == "None") {
        trans = c.s.execute("insert into Transactions values(" + billId + ", NULL, " + customerString + ", "
                + employeeId + ")");
    } else {
        promotionID = promotionId.get(promotionIndex);
        trans = c.s
                .execute("insert into Transactions values(" + billId + ", " + promotionID + ", " + customerString + ", "
                        + employeeId + ")");
    }
    for (String query : finalQueries) {
        c.s.execute(query);
    }
    for (String query : updateQueries) {
        c.s.execute(query);
    }
```

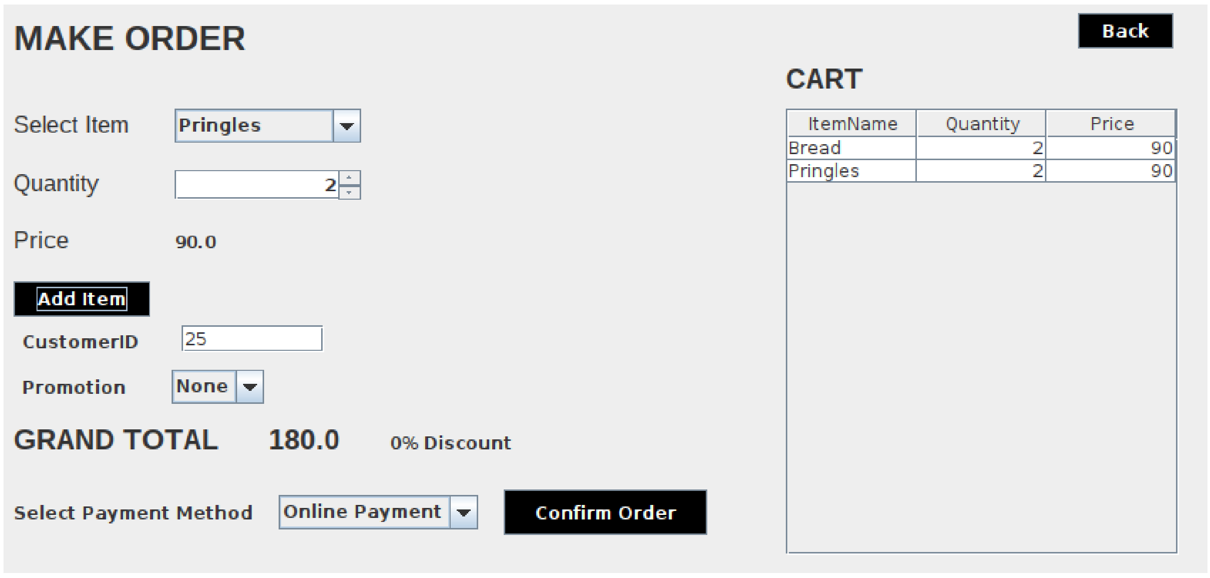
Fig MakeOrder Code

At the end when the user clicks on the Confirm Order button we execute each of the insert and update queries of each item and also create the bill in the Bills table.



```java
Conn c = new Conn();
String query = "select * from Products;";
ResultSet rs = c.s.executeQuery(query);
List<String> itemList = new ArrayList<String>();
List<String> promotionList = new ArrayList<String>();
List<Float> itemPriceList = new ArrayList<Float>();
while (rs.next()) {
    itemList.add(rs.getString("ProductName"));
    itemPriceList.add(rs.getFloat("SellingPrice"));
    itemId.add(rs.getInt("ProductID"));
    itemQuantities.add(rs.getInt("Quantity"));
}
```

This image shows how we use ArrayLists to store the values of the Products table for easy access.


MAKE ORDER

Select Item: Pringles

Quantity: 2

Price: 90.0

CustomerID: 25

Promotion: None

GRAND TOTAL 180.0 0% Discount

Select Payment Method: Online Payment

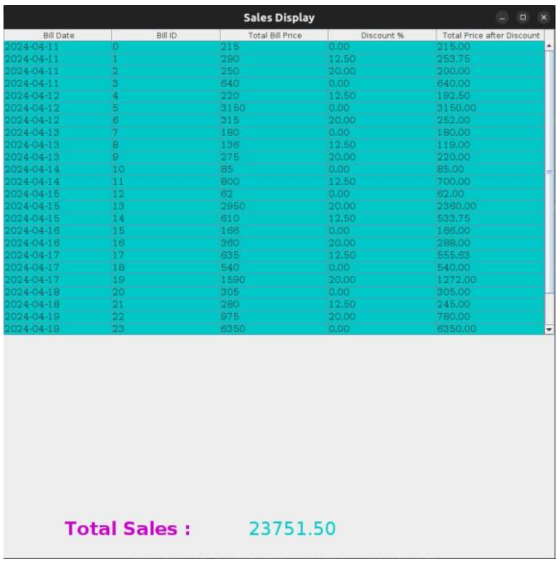
CART

ItemName	Quantity	Price
Bread	2	90
Pringles	2	90

This is an image of the UI of the class with a few items added into the cart and the Customer ID and Promotion added. There is also a back button the user can use to go back to the previous page.

16. DisplaySales Class

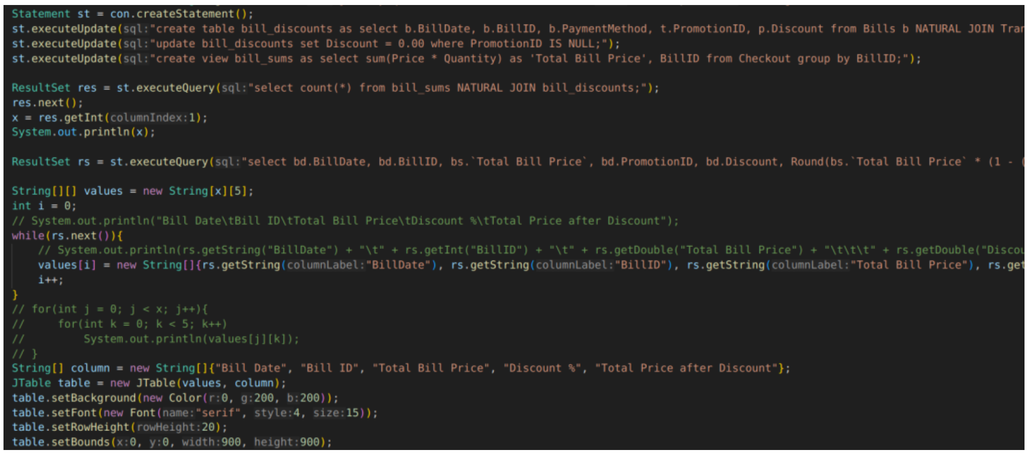
Using this class, we can display all the sales since the beginning. The sales are displayed on a ScrollPane. The Total Sales are also shown:


Sales Display

Bill Date	Bill ID	Total Bill Price	Discount %	Total Price after Discount
2024-04-11	0	215	0.00	215.00
2024-04-11	1	290	12.50	253.75
2024-04-11	2	250	20.00	200.00
2024-04-11	3	640	0.00	640.00
2024-04-12	4	220	12.50	192.50
2024-04-12	5	3150	0.00	3150.00
2024-04-12	6	315	20.00	252.00
2024-04-13	7	180	0.00	180.00
2024-04-13	8	136	12.50	119.00
2024-04-13	9	275	20.00	220.00
2024-04-14	10	85	0.00	85.00
2024-04-14	11	800	12.50	700.00
2024-04-15	12	62	0.00	62.00
2024-04-15	13	2950	20.00	2360.00
2024-04-15	14	610	12.50	533.75
2024-04-16	15	166	0.00	166.00
2024-04-16	16	360	20.00	288.00
2024-04-17	17	635	12.50	555.63
2024-04-17	18	540	0.00	540.00
2024-04-17	19	1590	20.00	1272.00
2024-04-18	20	305	0.00	305.00
2024-04-18	21	280	12.50	245.00
2024-04-19	22	975	20.00	780.00
2024-04-19	23	6350	0.00	6350.00

Total Sales : 23751.50

The code of the table is as follows:



```
Statement st = con.createStatement();
st.executeUpdate(sql:"create table bill_discounts as select b.BillDate, b.BillID, b.PaymentMethod, t.PromotionID, p.Discount from Bills b NATURAL JOIN Trar
st.executeUpdate(sql:"update bill_discounts set Discount = 0.00 where PromotionID IS NULL;");
st.executeUpdate(sql:"create view bill_sums as select sum(Price * Quantity) as 'Total Bill Price', BillID from Checkout group by BillID;");

ResultSet res = st.executeQuery(sql:"select count(*) from bill_sums NATURAL JOIN bill_discounts;");
res.next();
x = res.getInt(columnIndex:1);
System.out.println(x);

ResultSet rs = st.executeQuery(sql:"select bd.BillDate, bd.BillID, bs.`Total Bill Price`, bd.PromotionID, bd.Discount, Round(bs.`Total Bill Price` * (1 - (

String[][] values = new String[x][5];
int i = 0;
// System.out.println("Bill Date\tBill ID\tTotal Bill Price\tDiscount %\tTotal Price after Discount");
while(rs.next()){
    // System.out.println(rs.getString("BillDate") + "\t" + rs.getInt("BillID") + "\t" + rs.getDouble("Total Bill Price") + "\t\t\t" + rs.getDouble("Discou
    values[i] = new String[]{rs.getString(columnLabel:"BillDate"), rs.getString(columnLabel:"BillID"), rs.getString(columnLabel:"Total Bill Price"), rs.get
    i++;
}
// for(int j = 0; j < x; j++){
//     for(int k = 0; k < 5; k++)
//         System.out.println(values[j][k]);
// }
String[] column = new String[]{"Bill Date", "Bill ID", "Total Bill Price", "Discount %", "Total Price after Discount"};
JTable table = new JTable(values, column);
table.setBackground(new Color(r:0, g:200, b:200));
table.setFont(new Font(name:"serif", style:4, size:15));
table.setRowHeight(rowHeight:20);
table.setBounds(x:0, y:0, width:900, height:900);
```

The code employs the following complex query to create a table (that is later dropped) that contains data about the Bills and the Promotions added to the bills:

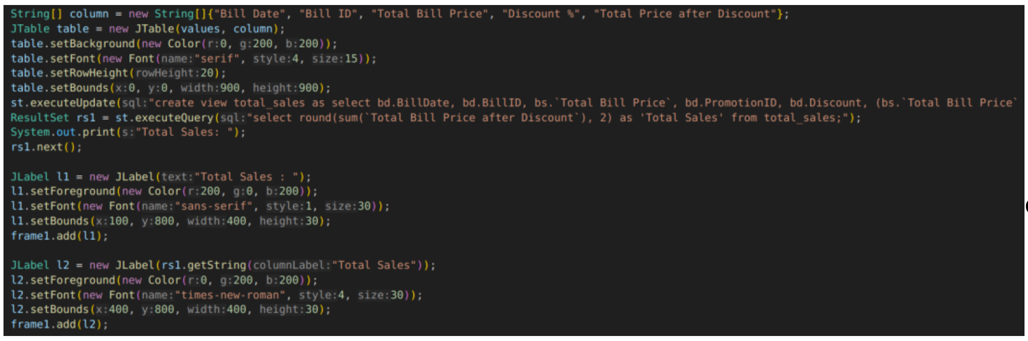
```
"create table bill_discounts as select b.BillDate, b.BillID, b.PaymentMethod, t.PromotionID, p.Discount from Bills b NATURAL JOIN Transactions t LEFT JOIN Promotions p ON t.PromotionID = p.PromotionID;"
```

```
"update bill_discounts set Discount = 0.00 where PromotionID IS NULL;"
```

This next complex query is used to create a view (that is also later dropped) that contains data about the total price per bill:

```
"create view bill_sums as select sum(Price * Quantity) as 'Total Bill Price', BillID from Checkout group by BillID;"
```

The code of the total sales is as follows:



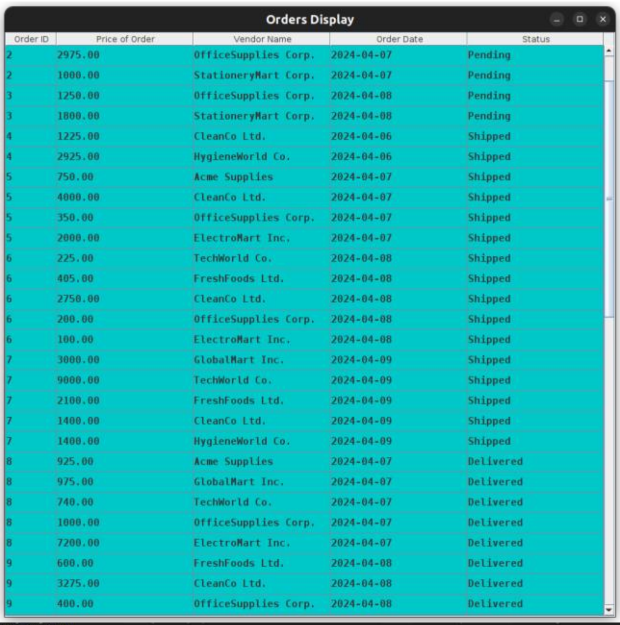
```
String[] column = new String[]{"Bill Date", "Bill ID", "Total Bill Price", "Discount %", "Total Price after Discount"};
JTable table = new JTable(values, column);
table.setBackground(new Color(r:0, g:200, b:200));
table.setFont(new Font(name:"serif", style:4, size:15));
table.setRowHeight(rowHeight:20);
table.setBounds(x:0, y:0, width:900, height:900);
st.executeUpdate(sql:"create view total_sales as select bd.BillDate, bd.BillID, bs.`Total Bill Price`, bd.PromotionID, bd.Discount, (bs.`Total Bill Price`
ResultSet rs1 = st.executeQuery(sql:"select round(sum(`Total Bill Price after Discount`), 2) as 'Total Sales' from total_sales;");
System.out.print(s:"Total Sales: ");
rs1.next();

JLabel l1 = new JLabel(text:"Total Sales : ");
l1.setForeground(new Color(r:200, g:0, b:200));
l1.setFont(new Font(name:"sans-serif", style:1, size:30));
l1.setBounds(x:100, y:800, width:400, height:30);
frame1.add(l1);

JLabel l2 = new JLabel(rs1.getString(columnLabel:"Total Sales"));
l2.setForeground(new Color(r:0, g:200, b:200));
l2.setFont(new Font(name:"times-new-roman", style:4, size:30));
l2.setBounds(x:400, y:800, width:400, height:30);
frame1.add(l2);
```

The code employs the following complex query to create a view (that is also later dropped) that contains data about the total price per bill after discount which is then added up and displayed:

```
"create view total_sales as select bd.BillDate, bd.BillID, bs.`Total Bill Price`, bd.PromotionID, bd.Discount, (bs.`Total Bill Price` * (1 - (bd.Discount / 100))) as `Total Bill Price after Discount` from bill_sums bs NATURAL JOIN bill_discounts bd;"
```

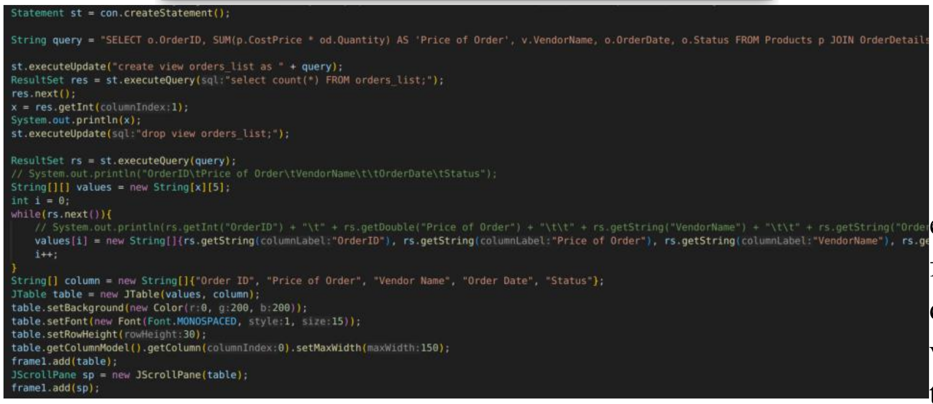
17. DisplayVendors Class

The name of this class can be deceiving because the code displays the orders placed with different vendors:


Orders Display

Order ID	Price of Order	Vendor Name	Order Date	Status
2	2975.00	OfficeSupplies Corp.	2024-04-07	Pending
2	1000.00	StationeryMart Corp.	2024-04-07	Pending
3	1250.00	OfficeSupplies Corp.	2024-04-08	Pending
3	1800.00	StationeryMart Corp.	2024-04-08	Pending
4	1225.00	CleanCo Ltd.	2024-04-06	Shipped
4	2925.00	HygieneWorld Co.	2024-04-06	Shipped
5	750.00	Acme Supplies	2024-04-07	Shipped
5	4000.00	CleanCo Ltd.	2024-04-07	Shipped
5	350.00	OfficeSupplies Corp.	2024-04-07	Shipped
5	2000.00	ElectroMart Inc.	2024-04-07	Shipped
6	225.00	TechWorld Co.	2024-04-08	Shipped
6	405.00	FreshFoods Ltd.	2024-04-08	Shipped
6	2750.00	CleanCo Ltd.	2024-04-08	Shipped
6	200.00	OfficeSupplies Corp.	2024-04-08	Shipped
6	100.00	ElectroMart Inc.	2024-04-08	Shipped
7	3000.00	GlobalMart Inc.	2024-04-09	Shipped
7	9000.00	TechWorld Co.	2024-04-09	Shipped
7	2100.00	FreshFoods Ltd.	2024-04-09	Shipped
7	1400.00	CleanCo Ltd.	2024-04-09	Shipped
7	1400.00	HygieneWorld Co.	2024-04-09	Shipped
8	925.00	Acme Supplies	2024-04-07	Delivered
8	975.00	GlobalMart Inc.	2024-04-07	Delivered
8	740.00	TechWorld Co.	2024-04-07	Delivered
8	1000.00	OfficeSupplies Corp.	2024-04-07	Delivered
8	7200.00	ElectroMart Inc.	2024-04-07	Delivered
9	600.00	FreshFoods Ltd.	2024-04-08	Delivered
9	3275.00	CleanCo Ltd.	2024-04-08	Delivered
9	400.00	OfficeSupplies Corp.	2024-04-08	Delivered

The code of the table is as follows:



```java
Statement st = con.createStatement();

String query = "SELECT o.OrderID, SUM(p.CostPrice * od.Quantity) AS 'Price of Order', v.VendorName, o.OrderDate, o.Status FROM Products p JOIN OrderDetails

st.executeUpdate("create view orders_list as " + query);
ResultSet res = st.executeQuery(sql:"select count(*) FROM orders_list;");
res.next();
x = res.getInt(columnIndex:1);
System.out.println(x);
st.executeUpdate(sql:"drop view orders_list;");

ResultSet rs = st.executeQuery(query);
// System.out.println("OrderID\tPrice of Order\tVendorName\t\tOrderDate\tStatus");
String[][] values = new String[x][5];
int i = 0;
while(rs.next()){
    // System.out.println(rs.getInt("OrderID") + "\t" + rs.getDouble("Price of Order") + "\t\t" + rs.getString("VendorName") + "\t\t" + rs.getString("Orde
    values[i] = new String[]{rs.getString(columnLabel:"OrderID"), rs.getString(columnLabel:"Price of Order"), rs.getString(columnLabel:"VendorName"), rs.ge
    i++;
}
String[] column = new String[]{"Order ID", "Price of Order", "Vendor Name", "Order Date", "Status"};
JTable table = new JTable(values, column);
table.setBackground(new Color(r:0, g:200, b:200));
table.setFont(new Font(Font.MONOSPACED, style:1, size:15));
table.setRowHeight(rowHeight:30);
table.getColumnModel().getColumn(columnIndex:0).setMaxWidth(maxWidth:150);
frame1.add(table);
JScrollPane sp = new JScrollPane(table);
frame1.add(sp);
```

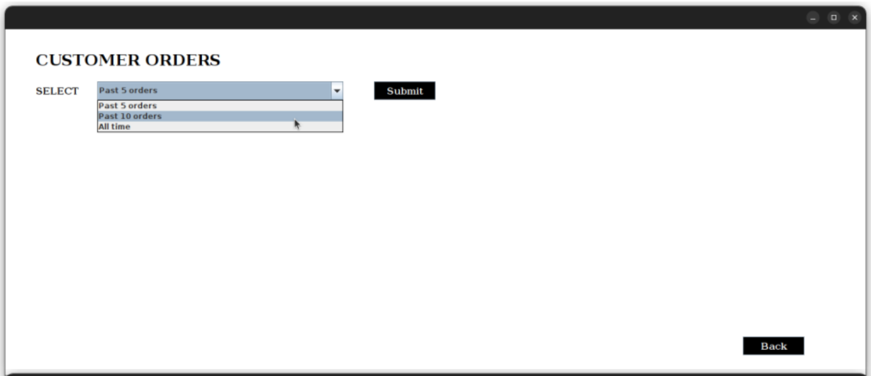
The code employs the following complex query to create a view that contains the data of expense per order which is then displayed:

```
"create view orders_list as SELECT o.OrderID, SUM(p.CostPrice *
od.Quantity) AS 'Price of Order', v.VendorName, o.OrderDate, o.Status FROM
Products p JOIN OrderDetails od ON p.ProductID = od.ProductID JOIN Orders o
ON od.OrderID = o.OrderID JOIN Vendors v ON v.VendorID = od.VendorID GROUP
BY od.OrderID, od.VendorID;"
```

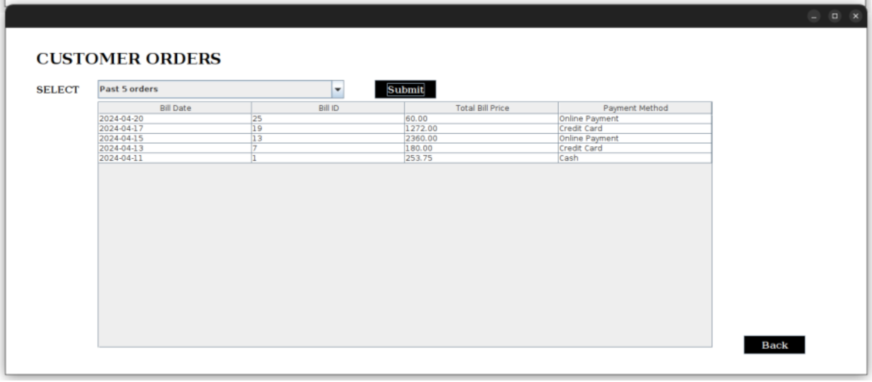
18. PreviousOrders Class

This class allows customers to view their Past 5, Past 10 or All Orders made at the Supermarket. Hence, it uses conditional statements to display different numbers of orders in

the display table.
The ComboBox to choose the different modes is as follows:



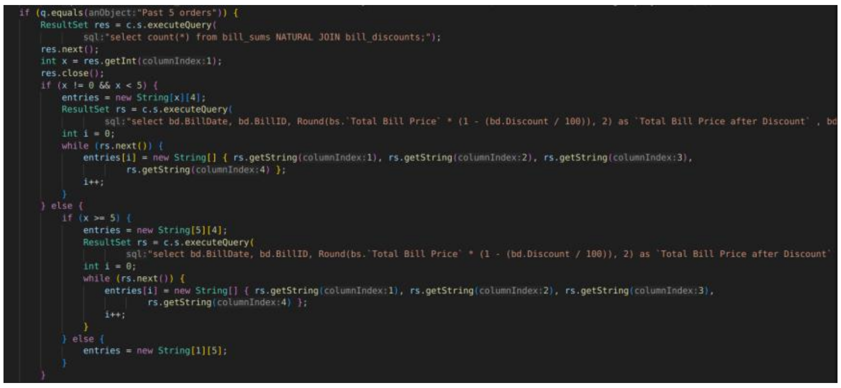
The table shown is as follows:



Bill Date	Bill ID	Total Bill Price	Payment Method
2024-04-20	25	60.00	Online Payment
2024-04-17	19	1272.00	Credit Card
2024-04-15	13	2360.00	Online Payment
2024-04-13	7	180.00	Credit Card
2024-04-11	1	253.75	Cash

We can change the value of the ComboBox and click "Submit" to view the desired table any number of times.

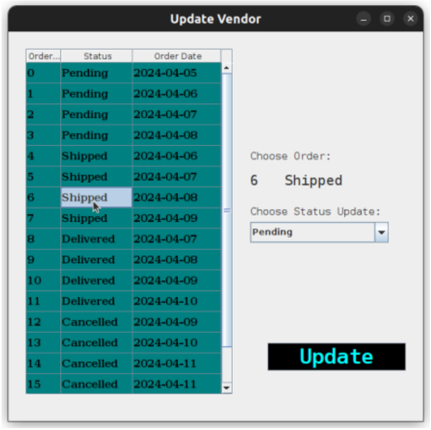
The code remains the same in all three ComboBox options except for setting a "`LIMIT 5`", "`LIMIT 10`", or no limit in the MySQL query. The temporary table and view created are the same as the DisplaySales Class table with the added condition of matching the CustomerID:



```java
if (q.equals(anObject:"Past 5 orders")) {
    ResultSet res = c.s.executeQuery(
            sql:"select count(*) from bill_sums NATURAL JOIN bill_discounts;");
    res.next();
    int x = res.getInt(columnIndex:1);
    res.close();
    if (x != 0 && x < 5) {
        entries = new String[x][4];
        ResultSet rs = c.s.executeQuery(
                sql:"select bd.BillDate, bd.BillID, Round(bs.`Total Bill Price` * (1 - (bd.Discount / 100)), 2) as `Total Bill Price after Discount` , bd
        int i = 0;
        while (rs.next()) {
            entries[i] = new String[] { rs.getString(columnIndex:1), rs.getString(columnIndex:2), rs.getString(columnIndex:3),
                    rs.getString(columnIndex:4) };
            i++;
        }
    } else {
        if (x >= 5) {
            entries = new String[5][4];
            ResultSet rs = c.s.executeQuery(
                    sql:"select bd.BillDate, bd.BillID, Round(bs.`Total Bill Price` * (1 - (bd.Discount / 100)), 2) as `Total Bill Price after Discount`
            int i = 0;
            while (rs.next()) {
                entries[i] = new String[] { rs.getString(columnIndex:1), rs.getString(columnIndex:2), rs.getString(columnIndex:3),
                        rs.getString(columnIndex:4) };
                i++;
            }
        } else {
            entries = new String[1][5];
        }
    }
```

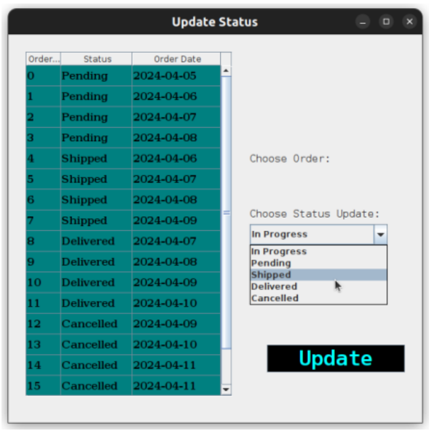
19. UpdateVendor Class

Again, this a misleading class name, because this class allows Employees to change the status of the orders, between “In Progress”, “Pending”, “Shipped”, “Delivered”, and “Cancelled”.


Update Vendor

Order...	Status	Order Date
0	Pending	2024-04-05
1	Pending	2024-04-06
2	Pending	2024-04-07
3	Pending	2024-04-08
4	Shipped	2024-04-06
5	Shipped	2024-04-07
6	Shipped	2024-04-08
7	Shipped	2024-04-09
8	Delivered	2024-04-07
9	Delivered	2024-04-08
10	Delivered	2024-04-09
11	Delivered	2024-04-10
12	Cancelled	2024-04-09
13	Cancelled	2024-04-10
14	Cancelled	2024-04-11
15	Cancelled	2024-04-11

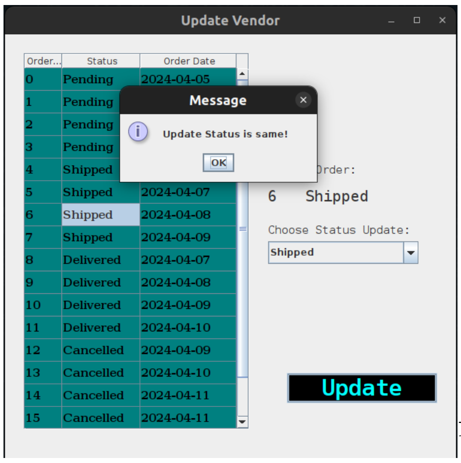
Choose Order:
6 Shipped
Choose Status Update: Pending


Update Status

Order...	Status	Order Date
0	Pending	2024-04-05
1	Pending	2024-04-06
2	Pending	2024-04-07
3	Pending	2024-04-08
4	Shipped	2024-04-06
5	Shipped	2024-04-07
6	Shipped	2024-04-08
7	Shipped	2024-04-09
8	Delivered	2024-04-07
9	Delivered	2024-04-08
10	Delivered	2024-04-09
11	Delivered	2024-04-10
12	Cancelled	2024-04-09
13	Cancelled	2024-04-10
14	Cancelled	2024-04-11
15	Cancelled	2024-04-11

Choose Order:
Choose Status Update: In Progress

It also prevents unnecessary updations by displaying a popup message and another popup message indicating successful Status Change:

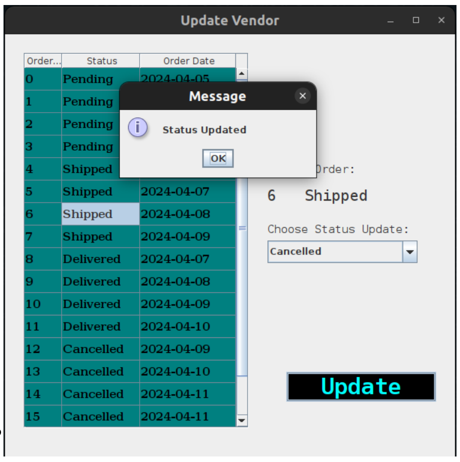

Update Vendor

Order...	Status	Order Date
0	Pending	2024-04-05
1	Pending	
2	Pending	
3	Pending	
4	Shipped	
5	Shipped	2024-04-07
6	Shipped	2024-04-08
7	Shipped	2024-04-09
8	Delivered	2024-04-07
9	Delivered	2024-04-08
10	Delivered	2024-04-09
11	Delivered	2024-04-10
12	Cancelled	2024-04-09
13	Cancelled	2024-04-10
14	Cancelled	2024-04-11
15	Cancelled	2024-04-11

Message
Update Status is same!

rder:
6 Shipped
Choose Status Update: Shipped

The code for this

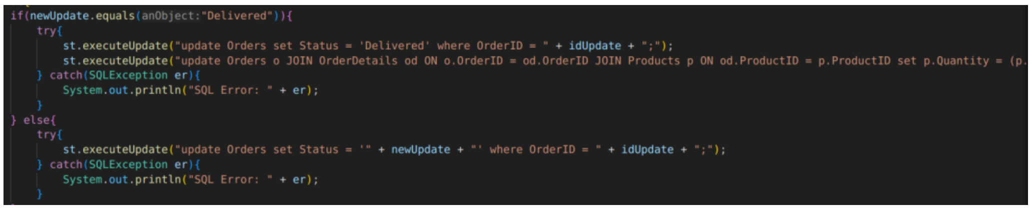

Update Vendor

Order...	Status	Order Date
0	Pending	2024-04-05
1	Pending	
2	Pending	
3	Pending	
4	Shipped	
5	Shipped	2024-04-07
6	Shipped	2024-04-08
7	Shipped	2024-04-09
8	Delivered	2024-04-07
9	Delivered	2024-04-08
10	Delivered	2024-04-09
11	Delivered	2024-04-10
12	Cancelled	2024-04-09
13	Cancelled	2024-04-10
14	Cancelled	2024-04-11
15	Cancelled	2024-04-11

Message
Status Updated

rder:
6 Shipped
Choose Status Update: Cancelled

class


```java
if(newUpdate.equals(anObject:"Delivered")){
    try{
        st.executeUpdate("update Orders set Status = 'Delivered' where OrderID = " + idUpdate + ";");
        st.executeUpdate("update Orders o JOIN OrderDetails od ON o.OrderID = od.OrderID JOIN Products p ON od.ProductID = p.ProductID set p.Quantity = (p.
    } catch(SQLException er){
        System.out.println("SQL Error: " + er);
    }
} else{
    try{
        st.executeUpdate("update Orders set Status = '" + newUpdate + "' where OrderID = " + idUpdate + ";");
    } catch(SQLException er){
        System.out.println("SQL Error: " + er);
    }
```

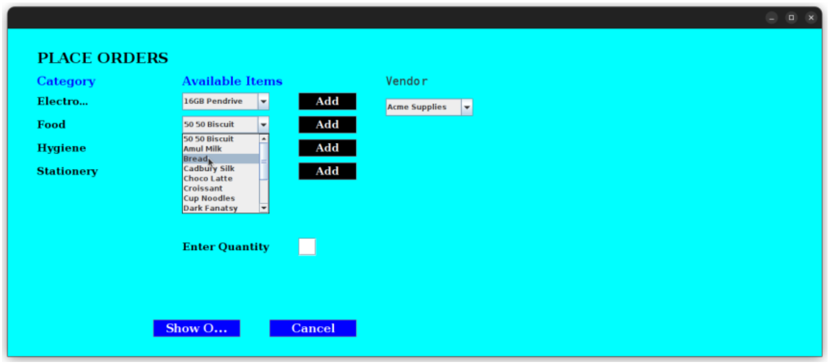
employs a complex update query when the status is changed that works on the join of 3 tables: Orders, OrderDetails and Products.

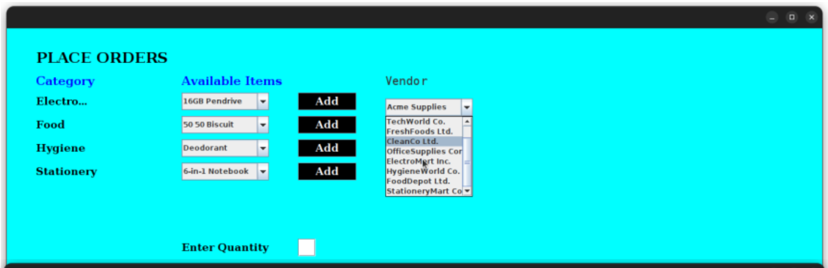
The query adds the Quantity of the Products table with the Quantity of the OrderDetails table when the status is updated to `"Delivered"`:

```
"update Orders o JOIN OrderDetails od ON o.OrderID = od.OrderID JOIN
Products p ON od.ProductID = p.ProductID set p.Quantity = (p.Quantity +
od.Quantity) where o.OrderID = " + idUpdate + ";"
```

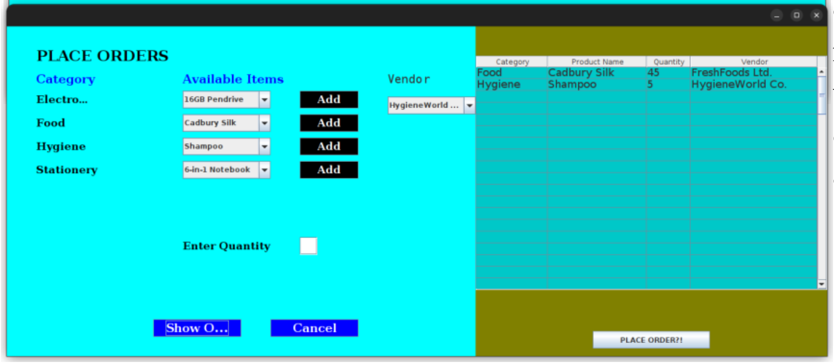
20. MakeOrder Class

This class enables an Employee to make an Order to add to the inventory from the list of Vendors. It has a drop-down menu to choose items in each category and a text field to enter the quantity of goods purchased. It also allows the employee to select the vendor.





After adding the goods to purchase from different vendors, it displays a final summary to review the order, following which the Employee may choose to Cancel the order or PLACE ORDER.



Category	Product Name	Quantity	Vendor
Food	Cadbury Silk	45	FreshFoods Ltd.
Hygiene	Shampoo	5	HygieneWorld Co.

The code for this class is mostly UI-based, so no complex queries are involved. The components added to the window are:

```java
JPanel p1;
JLabel text, category, availableItems, electronic, food, stationery, hygiene, vendor, quantityLabel;
JButton b1, electronicButton, foodButton,  hygieneButton, stationeryButton,  submit, cancel;
JTextField qTextField;
JComboBox<String> electronicsBox,foodBox, hygieneBox, stationeryBox, vendorBox;
JScrollPane jBar;
JTable orderTable;
int count = 0;
int EmployeeID = 2; //Can be changed depending on the Employee that logs in
```

CONCLUSION

In conclusion, the Supermarket Management System project serves as a comprehensive and efficient solution for managing various aspects of a retail business. Through a user-friendly graphical user interface (GUI), the system allows administrators and staff to handle tasks such as inventory management, customer orders, promotions, and more. The project effectively demonstrates the use of Java Swing and JavaFX components to create an interactive and intuitive experience for users.