Online Fashion Retail Inventory and Order Management System

Vineesh K (211627034)

Rehan Reddy (211627007)

Gokul Babu (211627011)

M Mehul (211627017)

Sparsh Agarwalla (211627038)

**Introduction**

This project aims to create an end-to-end software to enable the functioning of inventory and order processes for an online fashion retail business and provide smooth operation of stocking and delivery.

Large scale warehouses and fulfillment centers have become a norm since the dot-com boom and the emergence of online retail which has transformed the way we shop forever and also gave rise to large-scale warehouses to fulfill the ever-growing consumer demand, managing these huge warehouses and the flow of inventory inside them requires specialized software as it is too big and has lots of moving parts which makes it hard to be managed by a team of employees as it leaves room for error which can potentially delay company operations and cause loss of capital. Therefore, here arises a need for a comprehensive and responsive end to end software solution.

This software’s applications will mainly be based in large warehouses and fulfillment centers. The primary customers for this software suite include online retailers based mainly in the fashion industry who sell a wide range of products.

The software suite is built by using Python (PyQt5 library) and MySQL database.

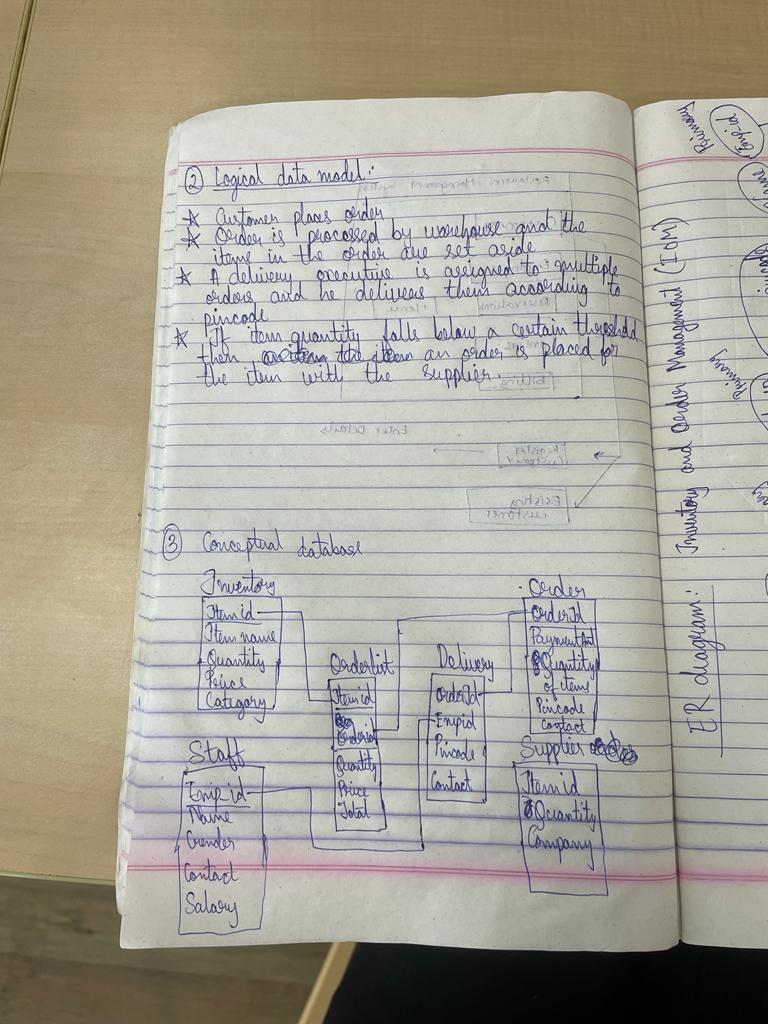
**Objectives**

The goal is to create an Inventory and Order Management System for an Online Fashion Retail company which will:

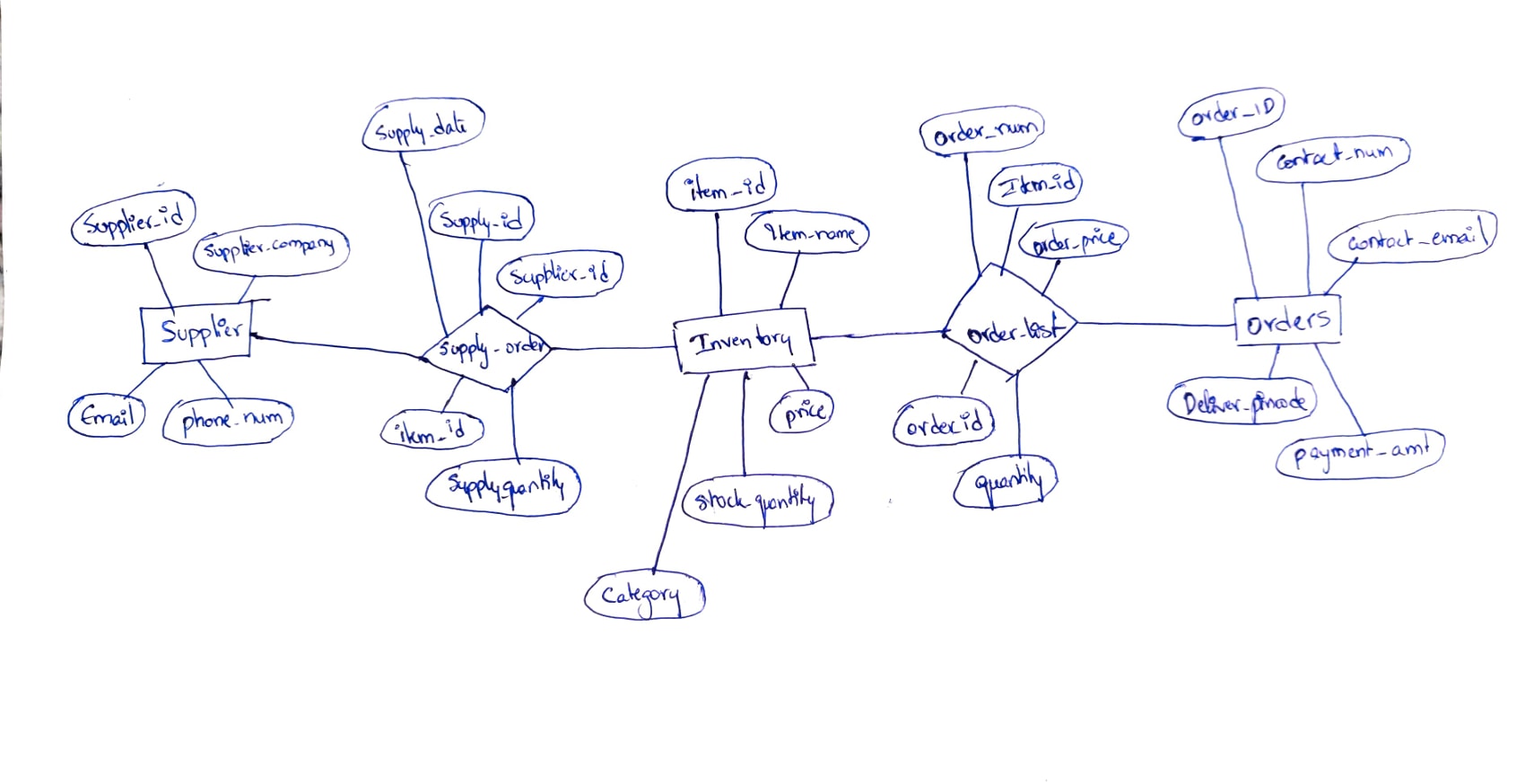
* Enable retaining inventory at an optimum level and never falling below the critical level
* Eliminate overstocking and understocking problems
* Enable auto-stocking which ensures more orders can be processed and fulfilled
* Streamlines the order process, making the entire process more efficient and cost effective.
* Eliminate clunky and non-responsive software which makes work slow
* Eliminate the need for multiple software products by including multiple features in the same software suite

**Design**

Initial Design and Discussion: Logical Data Model and Conceptual Database

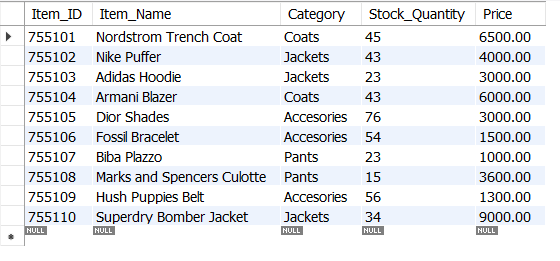


Conceptual ER diagram:



**Normalization**

Table name - Inventory



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Item\_ID | Item\_Name | Category | Stock\_Quantity | Price |
| 755101 | Nordstrom Trench Coat | Coats | 45 | 6500 |
| 755102 | Nike Puffer | Jackets | 43 | 4000 |
| 755103 | Adidas Hoodie | Jackets | 23 | 3000 |

Now considering the schema,

Inventory (Item\_ID, Item\_Name, Category, Stock\_Quantity, Price)

In this instance Item\_ID → Item\_Name, Category, Stock\_Quantity, Price making it a trivial functional dependency. Also, item\_id is a minimal superkey. So, Item\_ID is a primary key. Hence, Inventory schema is in 1NF.

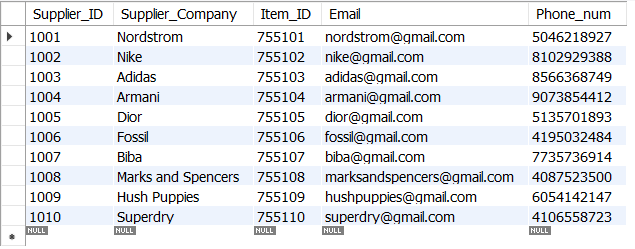
Checking the conditions for 2NF, the conditions are:

* The relations should be in 1NF
* There should be no partial dependency

First condition is satisfied. A partial dependency occurs whenever a non-prime attribute depends functionally on a part of the given candidate key. In this table Inventory, there are no partial dependencies so the second condition is also satisfied.

Functional Dependencies Item\_Name, Category, Stock\_Quantity and Price are fully dependent on Item\_ID.

Table name - Supplier

****

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Supplier\_ID | Supplier\_Company | Item\_ID | Email | Phone\_Num |
| 1001 | Nordstrom | 755101 | nordstrom@gmail.com | 7046218927 |
| 1002 | Nike | 755102 | nike@yahoo.in | 8102929388 |
| 1003 | Adidas | 755103 | adidas@hotmail.com | 8566368749 |

Now considering the schema,

Supplier (supplier\_ID, Supplier\_company, Item\_ID, Email, Phone\_Num)

In this instance Supplier\_ID → Supplier\_company, Item\_ID, Email, Phone\_Num making it a trivial functional dependency. Also, Supplier\_id is a minimal superkey. So, Supplier\_ID is a primary key. Hence, Supplier schema is in 1NF.

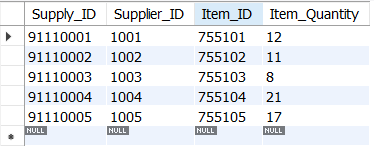
Checking the conditions for 2NF, the conditions are:

* The relations should be in 1NF
* There should be no partial dependency

First condition is satisfied. A partial dependency occurs whenever a non-prime attribute depends functionally on a part of the given candidate key. In this table Supplier, there are no partial dependencies so the second condition is also satisfied.

Functional Dependencies Supplier\_company, Item\_ID, Email, and Phone\_Num are fully dependent on Supplier\_ID.

Table name - Supplier\_Orders



|  |  |  |  |
| --- | --- | --- | --- |
| Supply\_ID | Item\_ID | Supplier\_ID | Supply\_Quantity |
| 91110001 | 755101 | 1001 | 12 |
| 91110002 | 755102 | 1002 | 11 |
| 91110003 | 755103 | 1003 | 8 |

Now considering the schema,

Supplier\_Orders(Supply\_ID, Item\_ID, Supplier\_ID, Supply\_Quantity)

In this instance Supply\_ID → Item\_ID, Supplier\_ID, Supply\_Quantity making it a trivial functional dependency. Also, Supply\_ID is a minimal superkey. So, Supply\_ID is a primary key. Hence, Inventory schema is in 1NF.

Checking the conditions for 2NF, the conditions are:

* The relations should be in 1NF
* There should be no partial dependency

First condition is satisfied. A partial dependency occurs whenever a non-prime attribute depends functionally on a part of the given candidate key. In this table Supply\_Orders, there are no partial dependencies so the second condition is also satisfied.

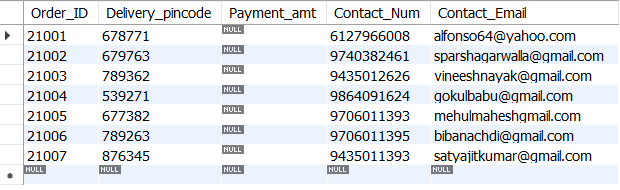
Functional Dependencies Supplierr\_ID, Item\_ID, Supply\_Quantity, are fully dependent on Supply\_ID.

Checking the conditions for 3NF, the conditions are:

* The relations should be in 2NF
* Non-prime should not determine non-prime

First condition is already satisfied. Supply\_ID is a prime attribute and all other attributes are non-prime so Supply\_ID determines all other attributes. So conditions for 3NF are satisfied.

Table name - Orders



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_ID | Delivery\_pincode | Payment\_amt | Contact\_Num | Contact\_Email |
| 21001 | 678771 | null | 6127966008 | alfonso64@yahoo.com |
| 21002 | 679763 | null | 9740382461 | sparshagarwalla@gmail.com |
| 21003 | 789362 | null | 9435012626 | vineeshnayak@gmail.com |

Now considering the schema,

Orders (Order\_ID, Delivery\_pincode, Payment\_amt, Contact\_Num, Contact\_Email)

In this instance Order\_ID → Delivery\_Pincode, Payment\_amt, Contact\_Num, Contact\_Email making it a trivial functional dependency. Also, Order\_ID is a minimal superkey. So, Order\_ID is a primary key. Hence, Supplier schema is in 1NF.

Checking the conditions for 2NF, the conditions are:

* The relations should be in 1NF
* There should be no partial dependency

First condition is satisfied. A partial dependency occurs whenever a non-prime attribute depends functionally on a part of the given candidate key. In this table Orders, there are no partial dependencies so the second condition is also satisfied.

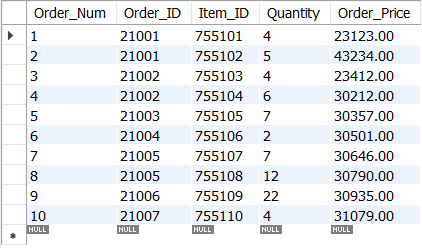
Functional Dependencies Delivery\_Pincode, Payment\_amt, Contact\_Num, Contact\_Email are fully dependent on Order\_ID.

Checking the conditions for 3NF, the conditions are:

* The relations should be in 2NF
* Non-prime should not determine non-prime

First condition is already satisfied. order\_ID is a prime attribute and all other attributes are non-prime so order\_ID determines all other attributes. So conditions for 3NF are satisfied.

Table name - Order\_List



|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| Order\_Num | Order\_ID | Item\_ID | Quantity | Order\_Price |
| 1 | 21001 | 755101 | 4 | 23123 |
| 2 | 21001 | 755102 | 5 | 43234 |
| 3 | 21002 | 755102 | 4 | 23412 |

Now considering the schema,

Order\_List (Order\_Num, Order\_ID, Item\_ID, Quantity, Order\_Price)

In this instance Order\_Num → Order\_ID, Item\_ID, Quantity, Order\_Price making it a trivial functional dependency. Also, Order\_Num is a minimal superkey. So, Order\_Num is a primary key. Hence, Supplier schema is in 1NF.

Checking the conditions for 2NF, the conditions are:

* The relations should be in 1NF
* There should be no partial dependency

First condition is satisfied. A partial dependency occurs whenever a non-prime attribute depends functionally on a part of the given candidate key. In this table Order\_List, there are no partial dependencies so the second condition is also satisfied.

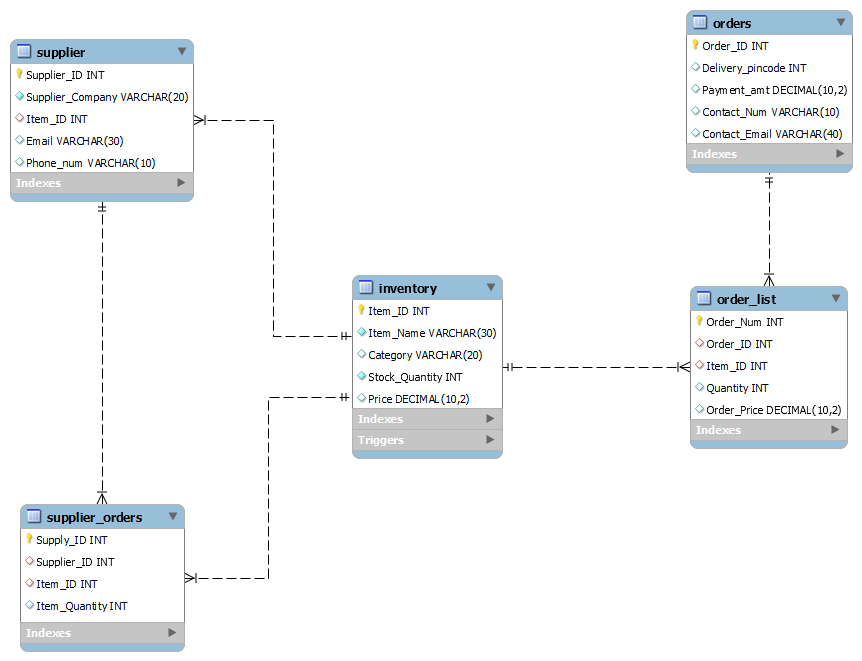
Functional Dependencies Order\_ID, Item\_ID, Quantity, Order\_Price are fully dependent on Order\_Num.

Checking the conditions for 3NF, the conditions are:

* The relations should be in 2NF
* Non-prime should not determine non-prime

First condition is already satisfied. Order\_Num is a prime attribute and all other attributes are non-prime so Order\_num determines all other attributes. So conditions for 3NF are satisfied.

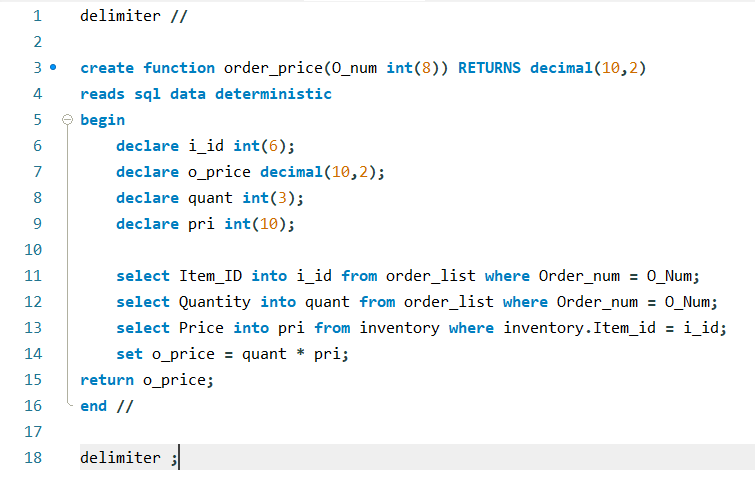
**Final ER Diagram**



**Implementation**

Functions:

**order\_price():**



This function calculates the order price that is the price for each item added multiplied by its quantities, it takes Order\_Num from order\_list table as input to identify that part of the order. These are order prices that are calculated here are later used to calculate the final payment amount, this happens when we take the sum of order prices.

Order\_price = Item price \* quantity is calculated here.

Code:

**delimiter //**

**create function order\_price(O\_num int(8)) RETURNS decimal(10,2)**

**reads sql data deterministic**

**begin**

**declare i\_id int(6);**

**declare o\_price decimal(10,2);**

**declare quant int(3);**

**declare pri int(10);**

**select Item\_ID into i\_id from order\_list where Order\_num = O\_Num;**

**select Quantity into quant from order\_list where Order\_num = O\_Num;**

**select Price into pri from inventory where inventory.Item\_id = i\_id;**

**set o\_price = quant \* pri;**

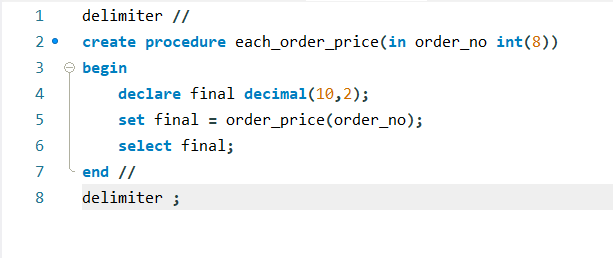
**return o\_price;**

**end //**

**delimiter ;**

Procedures:

**each\_order\_price():**



This procedure calls the order\_price() function and prints its values in the form of a table and takes Order\_Num as input to identify that specific record for which it needs to provide Order\_Price after it is calculated in the order\_price() function.

Code:

**delimiter //**

**create procedure each\_order\_price(in order\_no int(8))**

**begin**

**declare final decimal(10,2);**

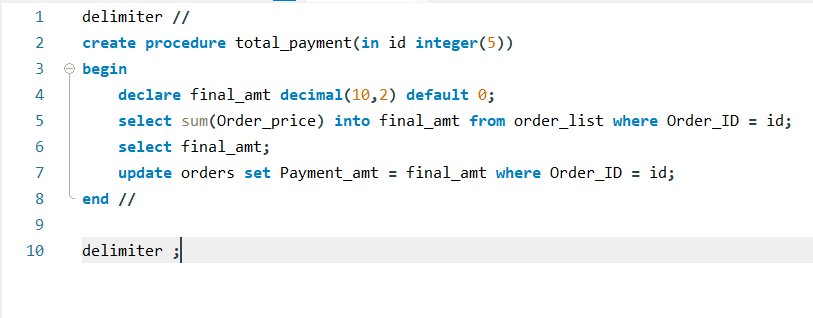
**set final = order\_price(order\_no);**

**select final;**

**end //**

**delimiter ;**

**total\_payment():**



This procedure calculates the final or total payment amount for an order. It takes Order\_ID as input and returns the sum of Order\_Price which it then outputs. It also updates the final payment amount in the orders table.

Total Payment = Order\_price + Order\_price + …

Code:

**delimiter //**

**create procedure total\_payment(in id integer(5))**

**begin**

**declare final\_amt decimal(10,2) default 0;**

**select sum(Order\_price) into final\_amt from order\_list where Order\_ID = id;**

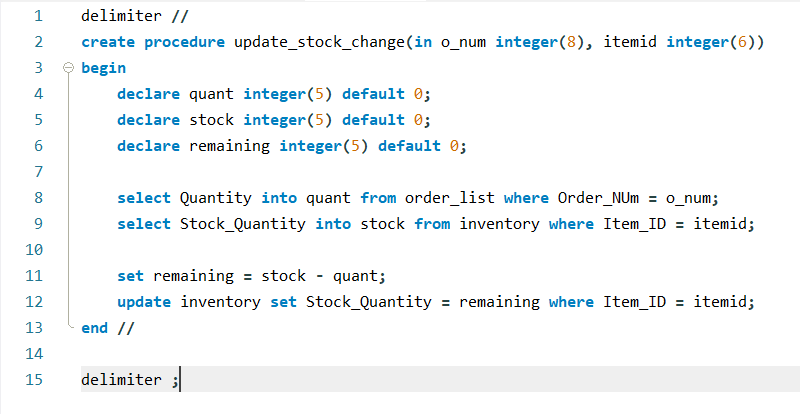
**select final\_amt;**

**update orders set Payment\_amt = final\_amt where Order\_ID = id;**

**end //**

**delimiter ;**

**update\_stock\_change():**



This procedure calculates the remaining stock quantity after a new order is placed. It subtracts the stock quantity from the quantity placed in the order. It also updates the value in Stock\_Quantity in inventory table.

Remaining stock quantity = Stock quantity – Quantity in order

Code:

**delimiter //**

**create procedure update\_stock\_change(in o\_num integer(8), itemid integer(6))**

**begin**

**declare quant integer(5) default 0;**

**declare stock integer(5) default 0;**

**declare remaining integer(5) default 0;**

**select Quantity into quant from order\_list where Order\_NUm = o\_num;**

**select Stock\_Quantity into stock from inventory where Item\_ID = itemid;**

**set remaining = stock - quant;**

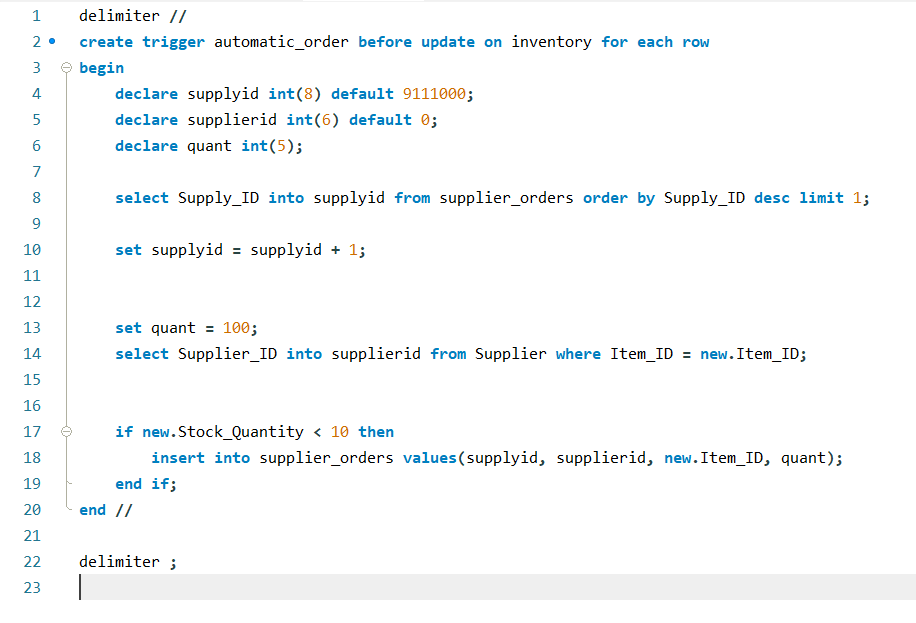
**update inventory set Stock\_Quantity = remaining where Item\_ID = itemid;**

**end //**

**delimiter ;**

Triggers:

**automatic\_order:**

****

This trigger executes whenever the stock quantity goes below 10 and orders 100 additional items. It places an order with the supplier by inserting the item data into the supplier\_orders table.

Code:

**delimiter //**

**create trigger automatic\_order before update on inventory for each row**

**begin**

**declare supplyid int(8) default 9111000;**

**declare supplierid int(6) default 0;**

**declare quant int(5);**

**select Supply\_ID into supplyid from supplier\_orders order by Supply\_ID desc limit 1;**

**set supplyid = supplyid + 1;**

**set quant = 100;**

**select Supplier\_ID into supplierid from Supplier where Item\_ID = new.Item\_ID;**

**if new.Stock\_Quantity < 10 then**

**insert into supplier\_orders values(supplyid, supplierid, new.Item\_ID, quant);**

**end if;**

**end //**

**delimiter ;**

Python Functions:

**insertInventory():**

Data from the text fields on the inventory page is taken as input and is then inserted into the inventory table by executing an sql statement after clicking on the confirm details button.

**insertSupplier\_list():**

Data from the text fields on the supplier list page is taken as input and is then inserted into the supplier table by executing an sql statement after clicking on the add button.

**insertNew\_order\_details():**

Data from the text fields on the new order page is taken as input and is then inserted into the orders table by executing an sql statement after clicking on the confirm details button.

**insertNew\_order\_details():**

Data from text fields and drop down menus (combo box) on the new order page is taken as input and is then inserted into the order\_list table. each\_order\_price procedure is here and updates the Order\_Price. total\_payment procedure is also called here and shows the final amount in a text label. Here update\_stock\_change procedure is also called and the stock quantity is updated in the inventory table. All these procedures and sql statements are executed each time the add button is clicked.

**loadnew\_order\_data():**

Data from the orders and inventory table is loaded into a table and displayed in new order page.

**loadneworder\_combobox():**

Item\_Names from inventory table are displayed in the drop down menu in new order page.

**display\_item\_details():**

Price from inventory table is displayed in a text label based on the item selected in the drop down menu.

**loadinventorytabledata():**

Data from the inventory table is loaded and displayed in a table widget in the inventory page.

**loadsupplier\_listtabledata():**

Data from the supplier table is loaded and displayed in a table widget in the supplier page.

**loadsupplier\_ordersdata():**

Data from the supplier\_orders table is loaded and displayed in a table widget in the automatic supplier orders page.

**loadmanage\_ordersdata():**

Data from the orders table is loaded and displayed in a table widget in the manage orders page.

**loadmanage\_orders\_items\_data():**

Data from the order\_list is loaded and displayed in a table widget in the manage orders page.

**deleteInventory():**

Data from the inventory table is deleted after selecting it on the table widget and clicking the delete button

**deleteSupplier\_list():**

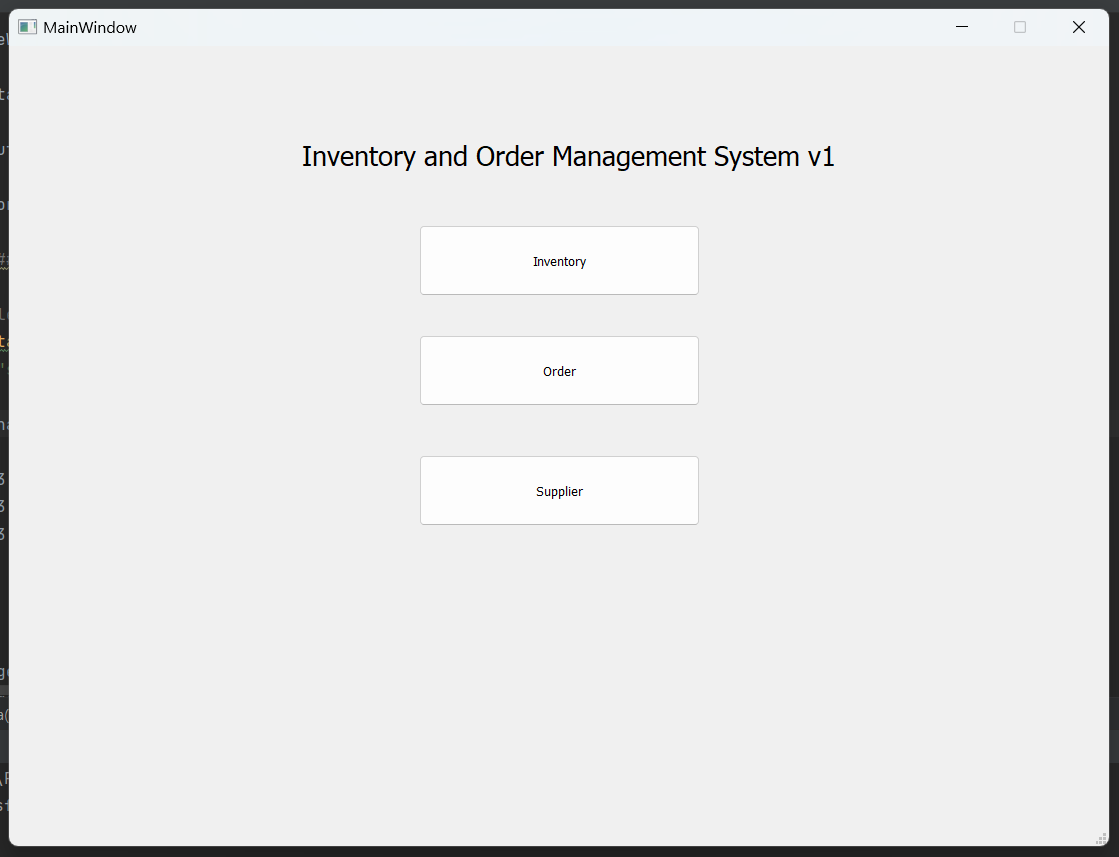
Data from the supplier table is deleted after selecting it on the table widget and clicking the delete button

**deleteOrders():**

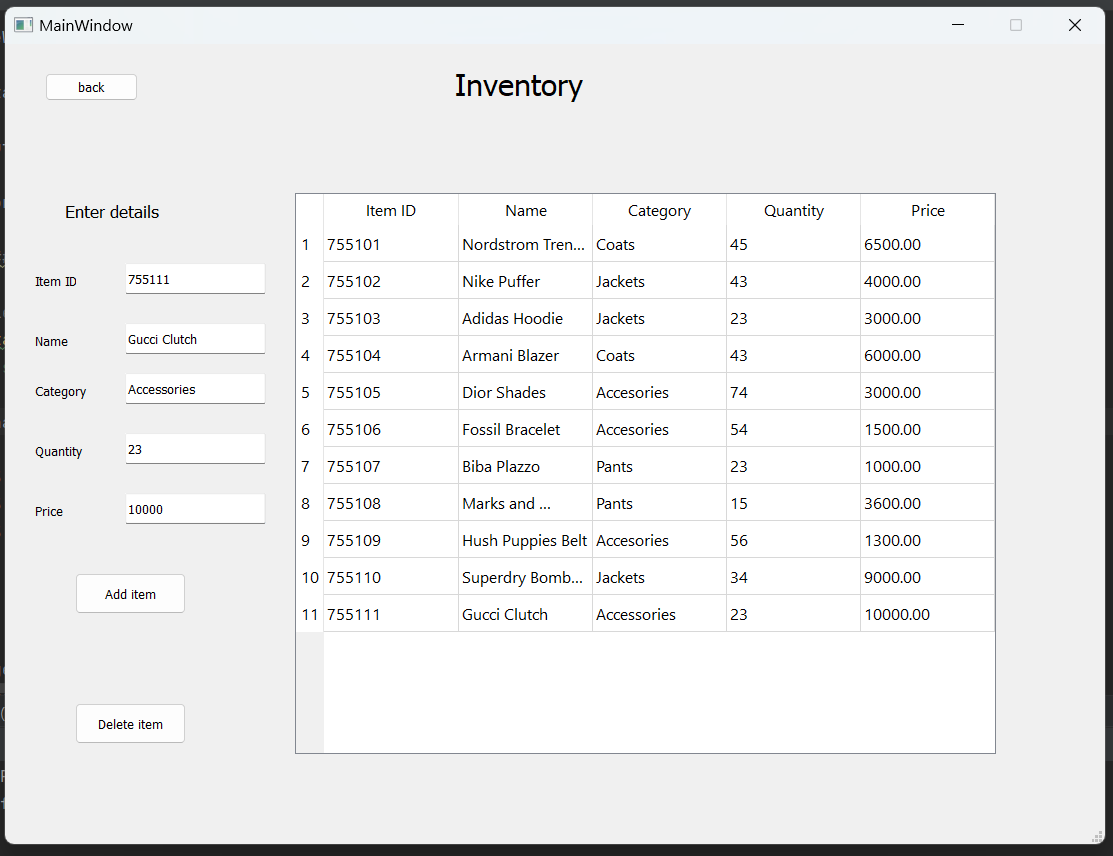
Data from the orders and order\_list table is deleted after selecting it on the table widget and clicking the delete button

**Output**

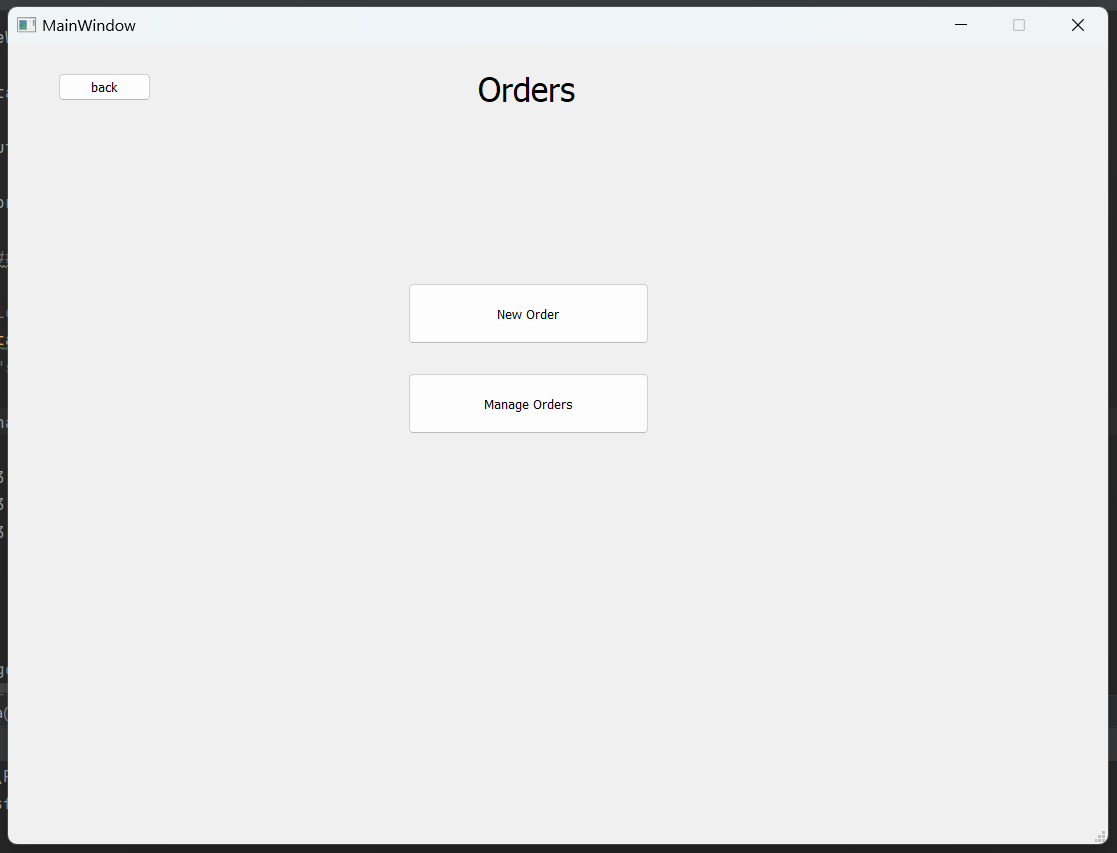
Home page:



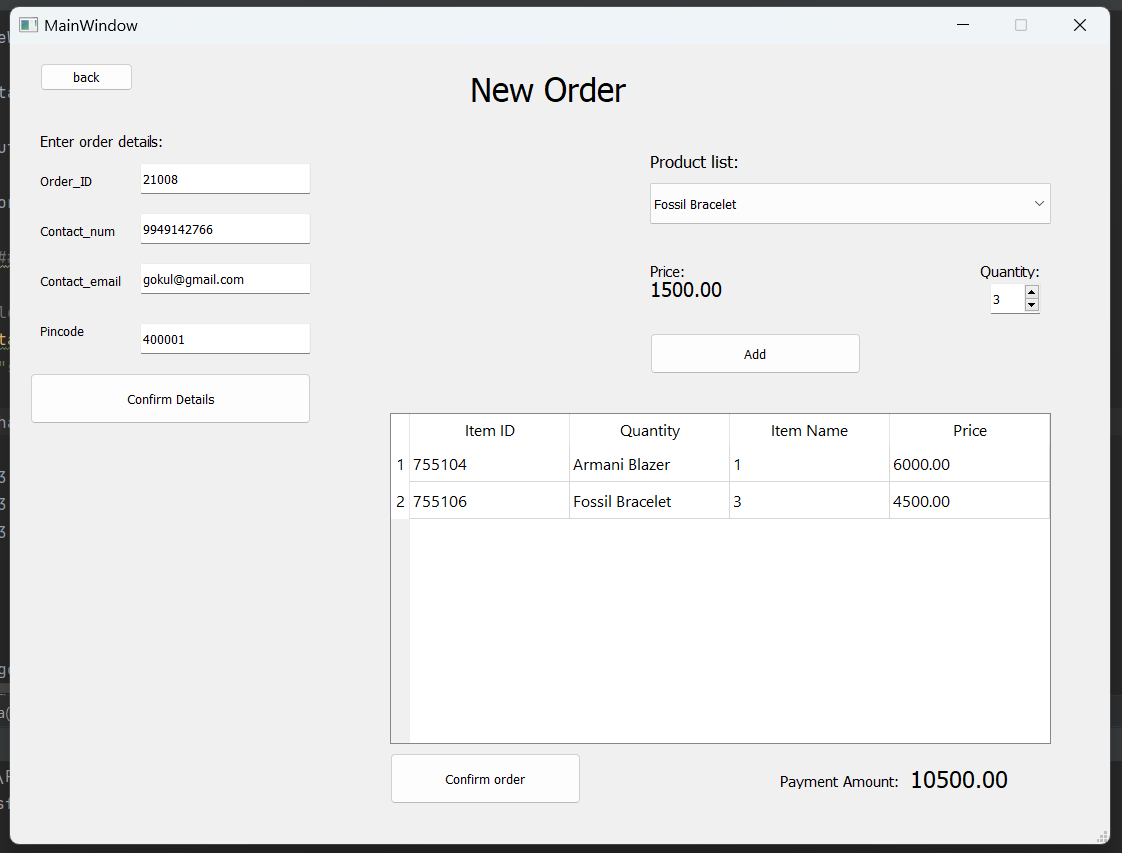
Inventory page:

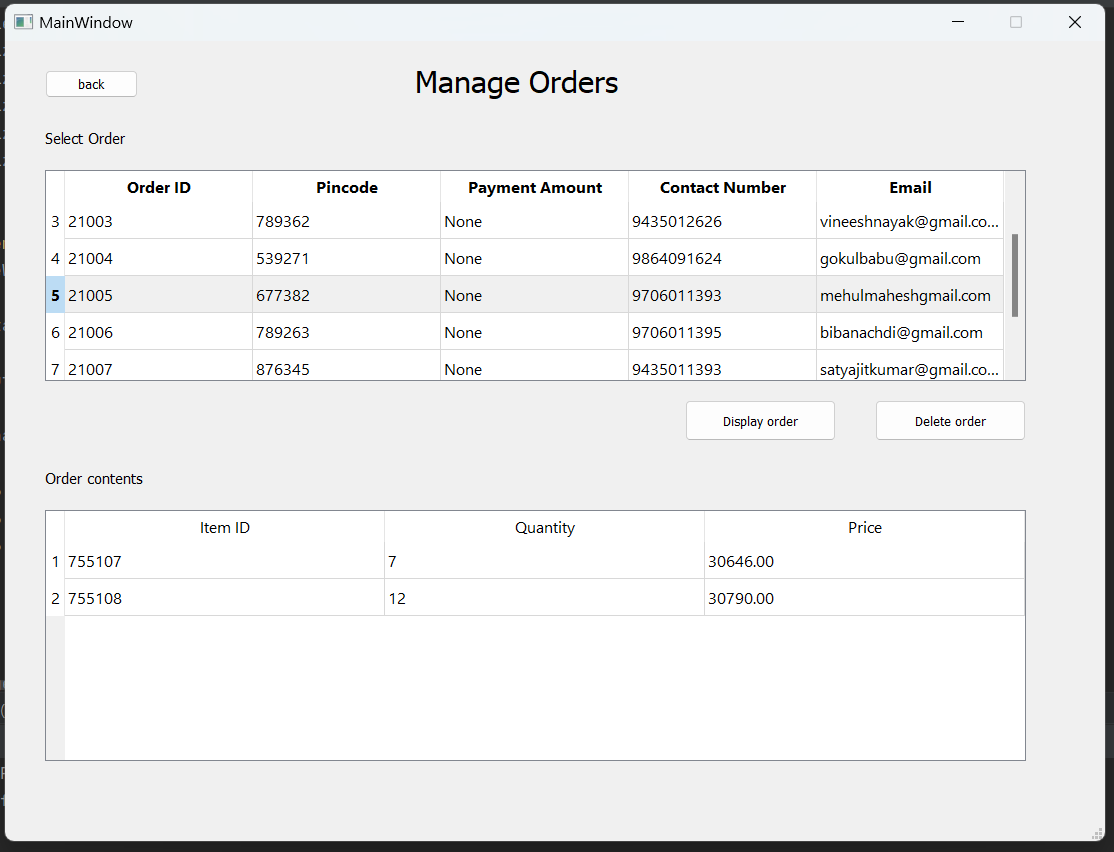


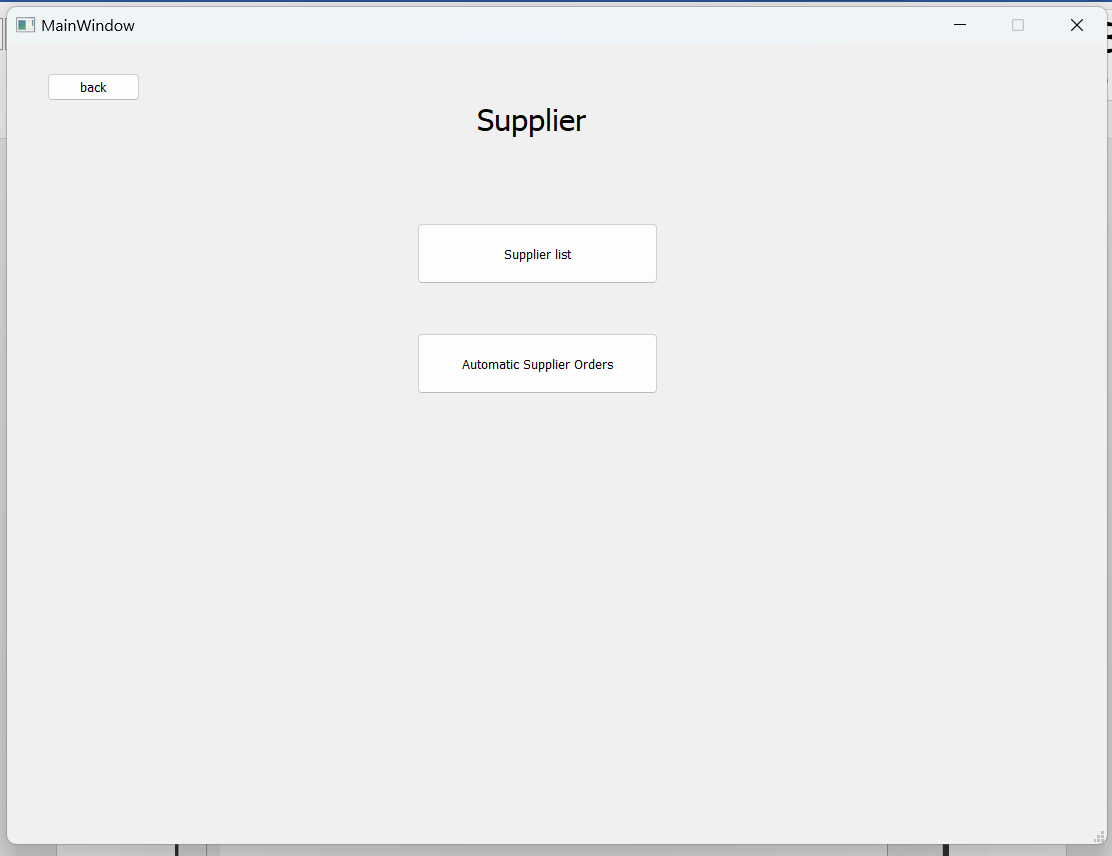
Orders page:



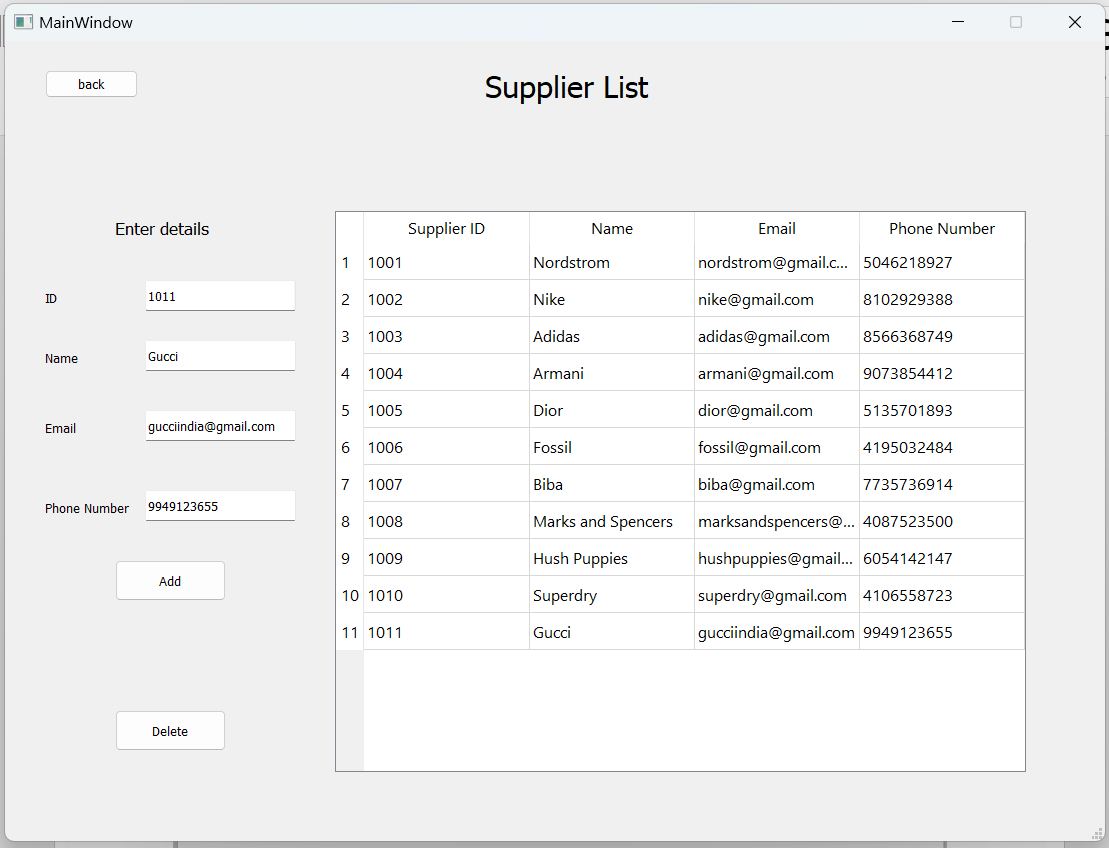
New Order page:



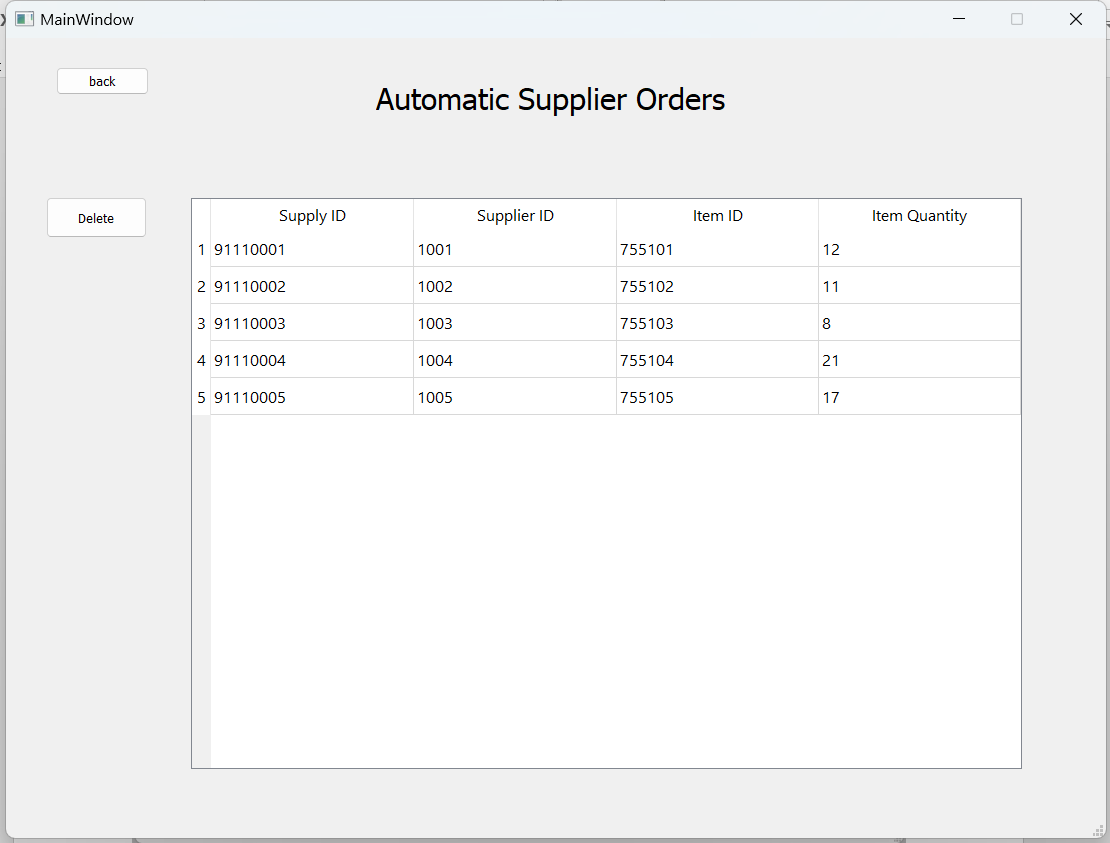
Manage Orders page: 

Supplier page: 

Supplier List page:



Automatic Supplier Orders page:



**Conclusion**

* Developed a software that can add, delete, and update the stocks that automatically performs the backend operation pertaining to a database of many relation tables onto which the changes are being made with every operation performed on the front end.
* This application enables us to update and modify the inventory, supplier list, supplier order, orders, order\_list.

* From this application we can get an update that if a particular inventory or stock is less than some pre-fixed quantity then it will auto-order the product and make it easy for the manager/staff to maintain stock and reorder on time to overcome the “Out of Stock” issue.
* Deals with the calculation of available resources for an accurate inventory control and process management
* Can have a record of customer details which can help us to retrieve the order details and order tracking habits of regular customers.
* From this program we can also keep a track of orders performed by different suppliers.
* Provides an end-to-end software suite for company warehouses to maintain inventory stock and orders efficiently.