# DBMS Project - Grocery Store App

# by Ryan Bracewell and Mauricio Flores

**1. Introduction:**

The goal of the project was to make an app for both Shoppers and Staff of the grocery store. This would allow Shoppers to add items to their cart from the stores inventory using the user interface provided. Upon completion, the shopper will be given a final total based on the sum of all the prices of the items the Shopper added to their cart.

Similarly, the Staff members can login using a username and password combination on the app to gain access to the inventory screen of the store. This will allow them to add and remove items from the Grocery Store’s inventory screen thus allowing Staff members to adapt their app’s inventory based on the store’s physical inventory.

**2. System Specifications:**

**2.1: Front-end:**

For the front end we used Python with a library called PySimpleGUI. With this library we were able to quickly create an example window for testing purposes and with even further development we were able to make use of different windows and window layouts to create the app. The different windows that are available on the app are as follows.

* Main menu
* Shopper Cart
* Staff Login
* Staff Inventory Screen

With these windows we were able to create a fully working application in a very short amount of time and few workarounds. This was the basis for the front end of the application.

**2.2: Back-end:**

For the back end database, we decided to go with MySQL as it was easiest to setup and get integrated with the UI. MySQL is a relational database management system that is used in the professional field for many different purposes including data warehousing, e-commerce and even logging applications so we felt it was the perfect fit for our small grocery store inventory database for our app.

**2.3: Software Requirements:**

* OS: Any modern OS that supports Python 3
* Languages: Python
* Libraries/Drivers: PySimpleGUI, MySQL Connector
* Database: MySQL

**3. Database Design:**

**3a.Entity List with Attribute Breakdown**

List of Entities:

* Employee
* Orders
* Product
* Supplier
* Emp\_has\_orders
* Orders\_has\_supplier
* Product\_has\_orders

**Employee:**

1. EmployeeID
2. FirstName
3. LastName
4. jobTitle

**Orders:**

1. orderID
2. supplierID
3. employeeID
4. orderdate
5. SKU
6. Quantity

**Product:**

1. SKU
2. UPC
3. Name
4. Dept/loc
5. unitSize
6. price
7. quantity
8. numsold

**Supplier:**

1. SupplierID
2. Name
3. Address
4. City
5. Country
6. Phone

**Emp\_has\_orders:**

1. EmployeeID
2. OrderID

**Orders\_has\_supplier:**

1. OrderID
2. SupplierID

**Product\_has\_orders:**

1. SKU
2. UPC
3. OrderID

**ER-Model:**

Diagram

Description automatically generated

**3b. Logical Design:**

**1. Employee: Table of employee records**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | EmployeeID | Integer | Employee ID number | Primary key |
| 2 | firstName | Varchar(45) | Employee first name |  |
| 3 | lastName | Varchar(45) | Employee last name |  |
| 4 | jobTitle | Varchar(45) | Employee job title |  |

Schema definition: CREATE  TABLE  Employee (employeeID integer primary key, Firstname varchar(45) ,lastName varchar(45), jobTitle varchar(45));

**2. Orders: Details for product orders**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | OrderID | Integer | Order ID number | Primary key |
| 2 | SupplierID | Integer | Supplier ID number | Foreign key |
| 3 | EmployeeID | Integer | Employee who placed order | Foreign key |
| 4 | orderDate | Date | Date order placed |  |
| 5 | SKU | Integer | SKU of product ordered | Foreign Key |
| 6 | Quantity | Integer | Number of product ordered |  |

Schema definition: CREATE TABLE Orders (OrderID integer primary key, SupplierID integer, foreign key (supplierID) references Supplier(SupplierID), EmployeeID integer, foreign key (employeeID) references Employee(employeeID), orderDate date, SKU integer, foreign key (SKU) references Product (SKU), Quantity integer);

**3. Product: Product information**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | SKU | Integer | Stock-keeping-unit | Primary key |
| 2 | UPC | Integer | Universal Product Code | Primary key |
| 3 | name | Varchar(45) | Product name |  |
| 4 | Dept/loc | Varchar(45) | Product department or location if appropriate |  |
| 5 | unitSize | Varchar(45) | Size per unit of product sold |  |
| 6 | Quantity | Integer | Number of product in-stock |  |
| 7 | numSold | Integer | Number of product sold |  |

Schema definition: CREATE TABLE Product (SKU integer primary key, UPC integer primary key, name varchar(45), dept/loc varchar(45), unitSize varchar(45), quantity integer, numSold integer);

**4. Supplier: Table of suppliers**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | SupplierID | Integer | Supplier ID number | Primary key |
| 2 | name | Varchar(45) | Supplier Business name |  |
| 3 | address | Varchar(45) | Business address of supplier |  |
| 4 | city | Varchar(45) | City supplier resides in |  |
| 5 | country | Varchar(45) | Country of supplier |  |
| 6 | phone | Varchar(45) | Phone number of supplier |  |

Schema definition: CREATE TABLE Supplier(SupplierID integer primary key, name varchar(45), address varchar(45), city varchar(45), country varchar(45), phone varchar(45));

**4. Emp\_has\_orders: Orders placed by Employee**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | employeeID | Integer | Employee ID number | Foreign key |
| 2 | orderID | Integer | Order ID number | Foreign key |

Schema definition: CREATE TABLE emp\_has\_orders (employeeID integer foreign key references Employee(employeeID), orderID integer foreignkey references Orders(orderID));

**5. Orders\_has\_supplier: Orders corresponding to their Supplier**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | orderID | Integer | Order ID number | Foreign key |
| 2 | supplierID | Integer | supplier ID number | Foreign key |

Schema definition: CREATE TABLE orders\_has\_supplier (orderID integer foreign key references Orders(orderID), supplierID integer foreign key references Supplier(supplierID));

**6. Product\_has\_orders: Products which have orders pending**

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| S.No | Field Name | Data Type | Description | Constraints |
| 1 | SKU | Integer | Product SKU | Foreign key |
| 2 | UPC | Integer | Product UPC | Foreign key |
| 3 | orderID | Integer | Order ID number | Foreign key |

Schema definition: CREATE TABLE Product\_has\_orders (SKU integer foreign key references Product(SKU), UPC integer foreign key references Product(UPC), orderID integer foreign key references Orders(orderID));

**4. Implementation:**

**a. Main Menu:**

The main menu of the program has three buttons.

* Shoppers – Pressing this will go to the Shopping Cart window, which is the main window customers will use to interface with the database, and purchase products.
* Staff – Pressing this will go to a Log-In window that will prompt the user for a valid username and password, which is checked against the users registered in the MySQL database.
* Exit – Pressing this will exit the program.

**b. Shopper Menu:**

* Graphical user interface, text

  Description automatically generatedYour cart – Here the name and price of the product SKU entered by the customer will appear.
* The text field below ‘Your cart’ will accept a valid numerical SKU, pressing the ‘Enter’ button and pressing the ‘Enter’ key on the keyboard will enter the value.

Hitting ‘Enter’ will clear the value in the input field, to allow for fast entry.

* Total – Here the customer running total is shown
* Finish – Pressing this button will activate a pop-up which will display the total, a thank you message, then return to the main menu.

Text

Description automatically generated

* Here is a screenshot displaying a few items added to the cart with the appropriate total.
* Graphical user interface, text, application

  Description automatically generatedHere is the pop-up window that displays when the ‘Finish’ button is pressed.

**c. Log-In Window:**

**Graphical user interface, application

Description automatically generated**

* Name – Username input field to login to the DB
* Password – Corresponding password to the entered username
* Error Output – If an invalid username/password is entered, this textbox will display the message, if a login is successful then no message will be displayed, and the user will proceed to the Staff Menu window.
* Log In – Pressing this will send the information in the Name and Password fields to the database for verification. Hitting this button or the ‘Enter’ key on the keyboard will send the credentials for verification.
* Back – Pressing this will send the user back to the main menu.

**Graphical user interface, text, application

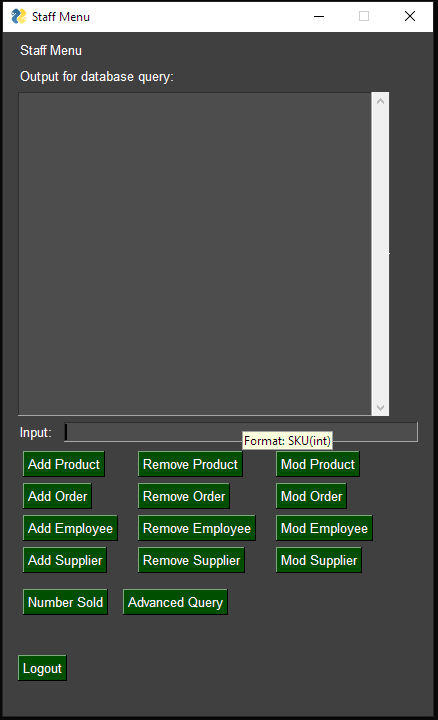
Description automatically generated**

* This screenshot displays a sample login attempt with an invalid username and password
* The error output field displaying a message showing access denied for the supplier credentials.
* Additional attempts to login to the menu are still possible.

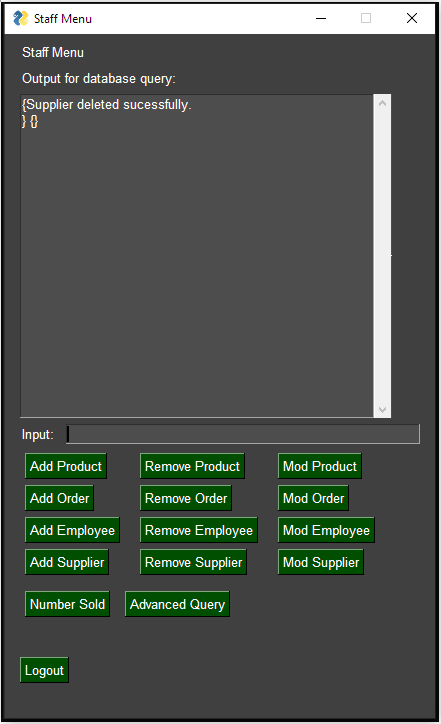
**d. Staff Menu:**

* Graphical user interface

  Description automatically generatedAdd X – This column of buttons allows for a user to add new entries to the corresponding table.
* Remove X – This column of buttons allows for a user to remove entries from the corresponding table.
* Mod X – This column of buttons allows for a user to modify entries from the corresponding table.
* Hovering over any the buttons on this menu shows a tooltip describing the format needed for the input field to perform the corresponding operation.
* The Number Sold button will display all products ordered by their numSold from greatest to least.
* The Advanced Query button will execute any SQL statement in the Input field when pressed.
* The logout button will return the user to the main menu.

****

* This screenshot displays the tooltip shown on hovering over a button.
* Successfully entering/removing or modifying the database will display a success message.



* A sample message on a successful executed database command.

**5. Source Code:**

import PySimpleGUI as sg

import mysql.connector

from datetime import datetime

from getpass import getpass

from mysql.connector import connect, Error

def mainMenu():

sg.theme('dark')

layout = [[sg.Text('Main Menu')],

[sg.Button('Shoppers', size = (20,5))],

[sg.Button('Staff',size = (20,5))],

[sg.Button('Exit',size = (20,5))]]

return sg.Window('Main Menu', layout, finalize=True)

def shopperMenu():

sg.theme('dark')

layout = [[sg.Text('Shopper Menu')],

[sg.Text("Your cart:")],

[sg.Text(size = (40,20), key = ('Cart'))],

[sg.Text('Enter the SKU of your product to add to the cart:'), sg.Input(key =('SKU'),size=(25,1),do\_not\_clear=False)],

[sg.Button('Enter',bind\_return\_key = True)],[sg.Text('Total:')],[sg.Text('0.00', size = (9,2), key = ('Total'))],

[sg.Button('Finish')]]

return sg.Window('Shopper Menu', layout, finalize=True)

def staffLogin():

sg.theme('dark')

layout = [[sg.Text('Staff Login')],

[sg.Text('Name:'),sg.Input(key = ('USERNAME'),enable\_events=True)],

[sg.Text('Password:'),sg.Input(key = ('PASSWORD'),password\_char='\*',enable\_events=True)],

[sg.Text('Error Output:')],

[sg.Output(key = ('ERROR CODE'), size = (50,0))],

[sg.Button('Log In', bind\_return\_key = True)],[sg.Button('Back')]]

return sg.Window('Staff Login', layout, finalize=True)

def staffMenu():

#contains buttons have tooltips to show to user what variables to enter, and their order

add\_block = [[sg.Button('Add Product', tooltip = 'Format: SKU (int), UPC(int), name(string),dept/loc(string),unitSize(string), price(float), quantity(int),0')],[sg.Button('Add Order', tooltip = 'Format: orderID (int), supplierID(int), employeeID(int), orderDate(date, YY/MM/DD), SKU (int), quantity(int)')] ,[sg.Button('Add Employee', tooltip = 'Format: employeeID (int), firstname (string), lastname(string), jobTitle(string)')],[sg.Button('Add Supplier',tooltip = 'Format: supplierID (int), name(sting), address(string),city(string),country(string),phone(string)')]]

remove\_block = [[sg.Button('Remove Product', tooltip ='Format: SKU(int)')], [sg.Button('Remove Order', tooltip = 'Format: orderID(int)')],[sg.Button('Remove Employee', tooltip = 'Format: employeeID(int)')],[sg.Button('Remove Supplier',tooltip = 'Format: supplierID(int)')]]

mod\_block = [[sg.Button('Mod Product', tooltip = 'Format: name(string),unitSize(string), price(float), quantity(int), numSold(int), SKU-to-modify(int)')],[sg.Button('Mod Order', tooltip = 'Format: orderDate(date, YY/MM/DD), quantity(int),orderid-to-modify(int)')] ,[sg.Button('Mod Employee', tooltip = 'Format: firstname (string), lastname(string), jobTitle(string), employeeid-to-modify(int)')],[sg.Button('Mod Supplier',tooltip = 'Format:name(sting), address(string),city(string),country(string),phone(string),supplierID-to-modify(int)')]]

sg.theme('dark')

layout = [[sg.Text('Staff Menu')],

[sg.Text('Output for database query:')],

[sg.Output(key = ('OUTPUT'), size = (50,20))],

[sg.Text('Input:'),sg.Input(key = ('QUERY\_VALUES'),size=(50,1),do\_not\_clear=False)],

[sg.Column(add\_block) ,sg.Column(remove\_block),sg.Column(mod\_block)],

[sg.Button('Number Sold',tooltip = 'No input needed.',pad=(10,10)),sg.Button('Advanced Query')],

[sg.Button('Logout',pad=(5,30))]]

return sg.Window('Staff Menu', layout, finalize=True)

main, shop, login, menu = mainMenu(), None, None, None

#initalize values

shopping\_cart = []

query\_history = []

total = 0.0

while True:

window, event, values = sg.read\_all\_windows()

if window == main and event in (sg.WIN\_CLOSED, 'Exit'):

break

if window == main:

if event =='Shoppers':

main.hide()

shop = shopperMenu()

if event =='Staff':

main.hide()

login = staffLogin()

if event == sg.WIN\_CLOSED:

break;

if window == shop:

if event =='Enter':

shop['Cart'].update(values['SKU'])

user = values['SKU']

with connect(

host ="10.0.0.54",

user = "root",

password = "project"

)as connection:

get\_name = "SELECT name FROM grocerydb.product WHERE SKU = %s"

get\_price = "SELECT price FROM grocerydb.product WHERE SKU = %s"

update\_num\_sold = "UPDATE grocerydb.product SET numsold = numsold + 1 WHERE SKU = %s"

with connection.cursor() as cursor:

cursor.execute(get\_name, (user,))

name\_record =cursor.fetchall()

cursor.execute(get\_price, (user,))

price\_record =cursor.fetchall()

cursor.execute(update\_num\_sold, (user,))

connection.commit()

shopping\_cart.append(name\_record)

shopping\_cart.append(price\_record)

string\_price = str(price\_record[0])

string\_price = string\_price.strip("(),")

price = float(string\_price)

total = total + price

shop['Total'].update(total)

shop['Cart'].update(shopping\_cart)

cursor.close()

connection.close()

if event =='Finish':

sg.popup("Thank you for shopping with us!"+"\n\nYour total is " +str(total)+".");

shop.hide()

main.un\_hide()

if event == sg.WIN\_CLOSED:

break;

if window == login:

entered\_username = values['USERNAME']

entered\_password = values['PASSWORD']

generic\_error = ["Invalid username/password."]

if event == 'Log In':

try:

with connect(

host ="10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

login['ERROR CODE'].update(connection)

connection\_string = str(connection)

if "1045" in connection\_string:

login['ERROR CODE'].update(generic\_error)

else:

login.hide()

menu = staffMenu()

except Error as e:

login['ERROR CODE'].update(e)

if event == 'Back':

login.hide()

main.un\_hide()

if event == sg.WIN\_CLOSED:

break;

if window == menu:

if event =='Add Product':

add\_to\_table = "INSERT INTO grocerydb.product values(%s, %s, %s, %s, %s, %s, %s, %s)"

tell\_user = "Product information added successfully! \n"

query = values['QUERY\_VALUES']

insert\_values = query.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(add\_to\_table, (int(insert\_values[0]),int(insert\_values[1]),str(insert\_values[2]),str(insert\_values[3]),str(insert\_values[4]),float(insert\_values[5]),int(insert\_values[6]),int(insert\_values[7])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_histroy.append(tell\_user)

query\_history.append(insert\_values)

menu['OUTPUT'].update(query\_history)

if event == 'Add Order':

#works correctly

add\_order = "INSERT INTO grocerydb.orders values (%s, %s, %s, %s, %s, %s)"

tell\_user = "Order information added sucessfully! \n"

query = values['QUERY\_VALUES']

split\_values = query.split(',')

split\_values[3] = datetime.strptime(split\_values[3],'%m/%d/%y')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(add\_order, (int(split\_values[0]),int(split\_values[1]),int(split\_values[2]),split\_values[3],int(split\_values[4]),int(split\_values[5])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(split\_values)

menu['OUTPUT'].update(query\_history)

pass

if event == 'Add Employee':

add\_employee = ("INSERT INTO grocerydb.employee values (%s, %s, %s, %s)")

tell\_user = "Employee information added successfully! \n"

query = values['QUERY\_VALUES']

split\_values = query.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(add\_employee, (int(split\_values[0]),str(split\_values[1]),str(split\_values[2]),str(split\_values[3])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(split\_values)

menu['OUTPUT'].update(query\_history)

if event == 'Add Supplier':

add\_supplier = "INSERT INTO grocerydb.supplier values (%s, %s, %s, %s, %s, %s)"

tell\_user = "Supplier information added successfully! \n"

query = values['QUERY\_VALUES']

split\_values = query.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(add\_supplier, (int(split\_values[0]),str(split\_values[1]),str(split\_values[2]),str(split\_values[3]),str(split\_values[4]),str(split\_values[5])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(split\_values)

menu['OUTPUT'].update(query\_history)

if event == 'Advanced Query':

query = values['QUERY\_VALUES']

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query)

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Remove Product':

query = "DELETE from grocerydb.product where SKU = %s"

tell\_user = "Product deleted sucessfully. \n"

del\_SKU = values['QUERY\_VALUES']

split\_values = del\_SKU.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(del\_SKU,))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Remove Order':

query = "DELETE from grocerydb.orders where orderid = %s"

tell\_user = "Order deleted sucessfully. \n"

del\_order = values['QUERY\_VALUES']

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(del\_order,))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Remove Employee':

query = "DELETE from grocerydb.employee where employeeid = %s"

tell\_user = "Employee deleted sucessfully. \n"

del\_emp = values['QUERY\_VALUES']

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(del\_emp,))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Remove Supplier':

query = "DELETE from grocerydb.supplier where supplierid = %s"

tell\_user = "Supplier deleted sucessfully. \n"

del\_id = values['QUERY\_VALUES']

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(del\_id,))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Mod Product':

query = "UPDATE grocerydb.product SET name = %s, unitsize = %s, price = %s, quantity = %s, numsold = %s WHERE SKU = %s"

tell\_user = "Product modified successfully. \n"

mod\_prod = values ['QUERY\_VALUES']

split\_values = mod\_prod.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(str(split\_values[0]),str(split\_values[1]),float(split\_values[2]),int(split\_values[3]),int(split\_values[4]),int(split\_values[5])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Mod Order':

query = "UPDATE grocerydb.orders SET orderdate = %s, quantity = %s WHERE orderid = %s"

tell\_user = "Order modified successfully. \n"

mod\_order = values ['QUERY\_VALUES']

split\_values = mod\_order.split(',')

split\_values[0] = datetime.strptime(split\_values[0],'%m/%d/%y')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(split\_values[0],int(split\_values[1]),int(split\_values[2])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Mod Employee':

query = "UPDATE grocerydb.employee SET firstname = %s, lastname = %s, jobTitle = %s WHERE employeeid = %s"

tell\_user = "Employee modified successfully. \n"

mod\_emp = values ['QUERY\_VALUES']

modify\_param = mod\_emp.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,modify\_param)

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Mod Supplier':

query = "UPDATE grocerydb.supplier SET name = %s, address = %s, city = %s, country = %s, phone = %s WHERE supplierid = %s"

tell\_user = "Supplier modified successfully. \n"

mod\_sup = values ['QUERY\_VALUES']

split\_values = mod\_sup.split(',')

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query,(str(split\_values[0]),str(split\_values[1]),str(split\_values[2]),str(split\_values[3]),str(split\_values[4]),int(split\_values[5])))

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(tell\_user)

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event == 'Number Sold':

query = "SELECT name,numsold from grocerydb.product order by numsold"

with connect(

host = "10.0.0.54",

user = entered\_username,

password = entered\_password

)as connection:

with connection.cursor() as cursor:

cursor.execute(query)

query\_result = cursor.fetchall()

connection.commit()

cursor.close()

connection.close()

query\_history.append(query\_result)

menu['OUTPUT'].update(query\_history)

if event =='Logout':

menu.hide()

main.un\_hide()

if event == sg.WIN\_CLOSED:

break;

window.close()