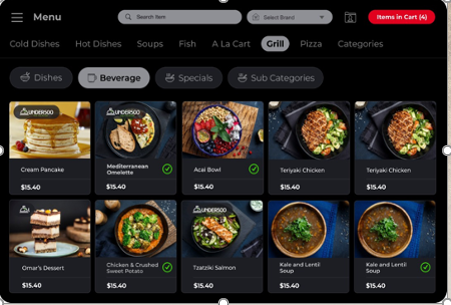
|  |
| --- |
| **HANOI UNIVERSITY OF SCIENCE AND TECHNOLOGY**  **\*\*\*\*\*\*\*\*\*\*\*\*\*\*\*\***    **FINAL PROJECT REPORT**  **DATABASE LAB**  ***(IT3290E)***  ***TOPIC: RESTAURANT MANAGEMENT APP***  **GROUP MEMBERS:**  **Trần Quang Hải – 20194755** |
| **Nguyễn Hoàng Vũ – 20190100**  **Nguyễn Ngọc Linh – 20194790**  **Trần Hải Đăng – 20194738**  **INSTRUCTOR: Mr. Nguyen Hong Phuong** |

1. **Purpose of the application:**

- The app was developed to help restaurant or café owners, or even store owners in managing the business.

- Seeing the need to frequently update the menu, stocks and suppliers in the food industry, we created this app to understand the pragmatic usage of databases.



- As we can see from the figure, we can see all the dishes with their images, prices, … We can search for categories we want and add them into cart manually, easy to use.

**2) User for the application:**

- This app is designed for small business owners and entrepreneurs in the culinary industry.

- Users can create a local database to keep track of their menu, stock, orders, suppliers and use these data for financial analysis.

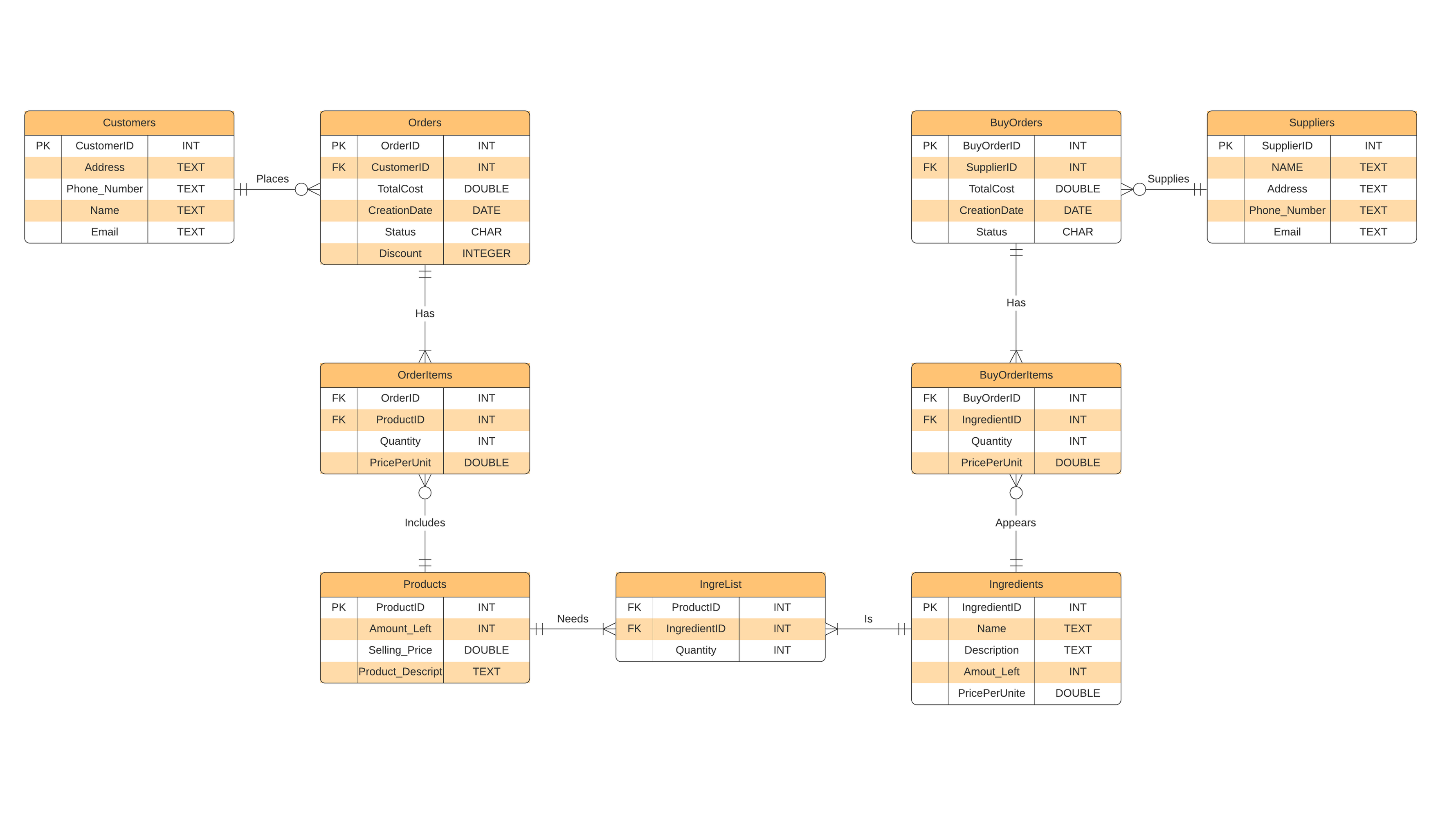
**3) Tools for application design:**

- We use JavaFX, using Java programming language in order to design, create, test, debug and deploy client application.

- In addition, we use SQLite for storing and managing database locally for higher performance and faster query execution suitable for applications.

- Both JavaFX and SQLite support for us to make this application.

**4) Entity-Relationship Diagram:**

**Normalize to 3NF:**

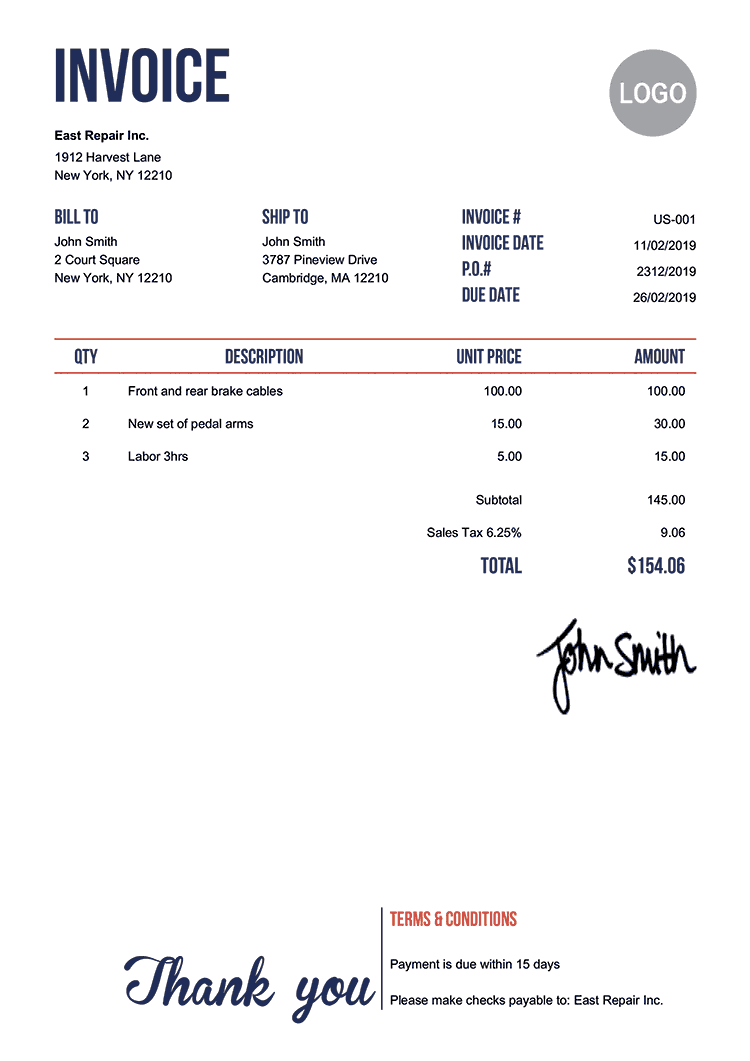
***Step 1:*** Find the minimal cover of FDs, which contains  
CustomerID --> CustomerAddress  
CustomerID --> CustomerPhoneNumber  
CustomerID --> CustomerName  
CustomerID --> CustomerEmail  
OrderID --> OrderTotalCost  
OrderID --> OrderCreationDate  
OrderID --> OrderStatus  
OrderID --> Discount  
OrderID,ProductID --> OrderItemQuantity  
OrderID,ProductID --> OrderItemPricePerUnit  
ProductID --> ProductAmountLeft  
ProductID --> ProductSellingPrice  
ProductID --> ProductDescript  
ProductID,IngredientID --> IngreListQuantity  
IngredientID --> IngredientName  
IngredientID --> IngredientDescription  
IngredientID --> IngredientAmountLeft  
IngredientID --> IngredientPricePerUnit  
IngredientID,BuyOrderID --> BuyOrderItemQuantity  
BuyOrderID --> BuyOrderTotalCost  
BuyOrderID --> BuyOrderCreationDate  
BuyOrderID --> BuyOrderStatus  
SupplierID --> SupplierName  
SupplierID --> SupplierAddress  
SupplierID --> SupplierPhoneNumber  
SupplierID --> SupplierEmail

***Step 2:*** Find all candidate keys. The set of candidates keys is { (CustomerID,OrderID,ProductID,IngredientID,BuyOrderItemPricePerUnit,BuyOrderID,SupplierID), }. The set of key attributes is: {CustomerID,OrderID, ProductID, IngredientID,BuyOrderItemPricePerUnit,BuyOrderID,SupplierID }.

***Step 3:*** Merge FDs with same LHS and whose RHS are non-key attributes, we get the set F1 which contains:  
CustomerID --> CustomerEmail,CustomerName,CustomerPhoneNumber,CustomerAddress  
OrderID --> Discount,OrderStatus,OrderCreationDate,OrderTotalCost  
OrderID,ProductID --> OrderItemPricePerUnit,OrderItemQuantity  
ProductID --> ProductDescript,ProductSellingPrice,ProductAmountLeft  
ProductID,IngredientID --> IngreListQuantity  
IngredientID --> IngredientPricePerUnit,IngredientAmountLeft,IngredientDescription,IngredientName  
IngredientID,BuyOrderID --> BuyOrderItemQuantity  
BuyOrderID --> BuyOrderStatus,BuyOrderCreationDate,BuyOrderTotalCost  
SupplierID --> SupplierEmail,SupplierPhoneNumber,SupplierAddress,SupplierName

***Step 4:*** Check each FD in the set F1 for violation of 3NF, and split table accordingly.  
Checking FD CustomerID --> CustomerEmail,CustomerName,CustomerPhoneNumber,CustomerAddress  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
CustomerID,CustomerEmail,CustomerName,CustomerPhoneNumber,CustomerAddress  
with FDs  
CustomerID --> CustomerEmail,CustomerName,CustomerPhoneNumber,CustomerAddress

Checking FD OrderID --> Discount,OrderStatus,OrderCreationDate,OrderTotalCost  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
OrderID,Discount,OrderStatus,OrderCreationDate,OrderTotalCost  
with FDs  
OrderID --> Discount,OrderStatus,OrderCreationDate,OrderTotalCost  
Checking FD OrderID,ProductID --> OrderItemPricePerUnit,OrderItemQuantity  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
OrderID,ProductID,OrderItemPricePerUnit,OrderItemQuantity  
with FDs  
OrderID,ProductID --> OrderItemPricePerUnit,OrderItemQuantity  
Checking FD ProductID --> ProductDescript,ProductSellingPrice,ProductAmountLeft  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
ProductID,ProductDescript,ProductSellingPrice,ProductAmountLeft  
with FDs  
ProductID --> ProductDescript,ProductSellingPrice,ProductAmountLeft  
Checking FD ProductID,IngredientID --> IngreListQuantity  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
ProductID,IngredientID,IngreListQuantity  
with FDs  
IngredientID,ProductID --> IngreListQuantity  
Checking FD IngredientID --> IngredientPricePerUnit,IngredientAmountLeft,IngredientDescription,IngredientName  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
IngredientID,IngredientPricePerUnit,IngredientAmountLeft,IngredientDescription,IngredientName with FDs  
IngredientID --> IngredientPricePerUnit,IngredientAmountLeft,IngredientDescription,IngredientName  
Checking FD IngredientID,BuyOrderID --> BuyOrderItemQuantity  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
IngredientID,BuyOrderID,BuyOrderItemQuantity  
with FDs  
BuyOrderID,IngredientID --> BuyOrderItemQuantity  
Checking FD BuyOrderID --> BuyOrderStatus,BuyOrderCreationDate,BuyOrderTotalCost  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
BuyOrderID,BuyOrderStatus,BuyOrderCreationDate,BuyOrderTotalCost  
with FDs  
BuyOrderID --> BuyOrderStatus,BuyOrderCreationDate,BuyOrderTotalCost  
Checking FD SupplierID --> SupplierEmail,SupplierPhoneNumber,SupplierAddress,SupplierName  
The FD violates 3NF as its LHS is not a superkey (and RHS is a set of non-key attributes).  
The following 3NF table is obtained:  
SupplierID,SupplierEmail,SupplierPhoneNumber,SupplierAddress,SupplierName  
with FDs  
SupplierID --> SupplierEmail,SupplierPhoneNumber,SupplierAddress,SupplierName  
  
***Step 5:*** Finally, add the following table into normalized 3NF table set (obtained by removing RHS attributes of FDs using which we produced a table):  
CustomerID,OrderID,ProductID,IngredientID,BuyOrderItemPricePerUnit,BuyOrderID,SupplierID with FDs



Take a look as this example:

- We need to keep the info of the customer (Name, Address…) -> Customer table to keep it.

- The order ID (#), date, totalCost… information about the invoice will be stored in Orders table.

- The quantities, priceperunit … of the product also need to be stored in OrderItems table.

- Moreover, we need to keep the name of the product selling price, amount left (in order to trace what we have left for customer to order). Therefore, the Products table exists.

**5) Query execution in the app:**

**- CreatingQuery:**

**+ createTableCustomers:**

CREATE TABLE "Customers" (

"CustomerID" INTEGER,

"Address" VARCHAR(40),

"Phone\_Number" VARCHAR(14),

"Name" VARCHAR(40),

"Email" VARCHAR(40),

PRIMARY KEY("CustomerID" AUTOINCREMENT)

);

**+ createTableSuppliers:**

CREATE TABLE "Suppliers" (

"SupplierID" INTEGER,

"Name" VARCHAR(40),

"Address" VARCHAR(40),

"Phone\_Number" VARCHAR(14),

"Email" VARCHAR(40),

PRIMARY KEY("SupplierID" AUTOINCREMENT)

);

**+ createTableProducts:**

CREATE TABLE "Products" (

"ProductID" INTEGER,

"Amount\_Left" INTEGER,

"Selling\_Price" DOUBLE,

"Product\_Descript" VARCHAR(40),

PRIMARY KEY("ProductID" AUTOINCREMENT)

);

**+ createTableBuyOrders:**

CREATE TABLE "BuyOrders" (

"BuyOrderID" INTEGER,

"SupplierID" INTEGER,

"Totalcost" DOUBLE,

"CreationDate" DATETIME,

"Status" VARCHAR(1),

CONSTRAINT "check\_status\_buyorders" CHECK("Status" = 'C' OR "Status" = 'P' OR "Status" = 'S' OR "Status" = 'F'),

PRIMARY KEY("BuyOrderID" AUTOINCREMENT)

);

**+ createTableBuyOrderItems:**

CREATE TABLE "BuyOrderItems" (

"BuyOrderID" INTEGER,

"IngredientID" INTEGER,

"Quantity" INTEGER,

"PricePerUnit" DOUBLE,

CONSTRAINT "buyorderitems\_fk\_ingredients" FOREIGN KEY("IngredientID") REFERENCES "Ingredients"("IngredientID") ON DELETE SET NULL,

CONSTRAINT "buyorderitems\_fk\_buyorders" FOREIGN KEY("BuyOrderID") REFERENCES "BuyOrders"("BuyOrderID") ON DELETE SET NULL

);

**+ createTableOrders:**

CREATE TABLE "Orders" (

"OrderID" INTEGER,

"CustomerID" INTEGER,

"Totalcost" DOUBLE,

"CreationDate" DATETIME,

"Status" VARCHAR(1),

"Discount" INTEGER,

CONSTRAINT "orders\_fk\_customers" FOREIGN KEY("CustomerID") REFERENCES "Customers"("CustomerID") ON DELETE SET NULL,

CONSTRAINT "check\_status" CHECK("Status" = 'C' OR "Status" = 'P' OR "Status" = 'S' OR "Status" = 'F'),

PRIMARY KEY("OrderID" AUTOINCREMENT)

);

**+ createTableOrderItems:**

CREATE TABLE "OrderItems" (

"OrderID" INTEGER,

"ProductID" INTEGER,

"Quantity" INTEGER,

"PricePerUnit" DOUBLE,

CONSTRAINT "orderitems\_fk\_orders" FOREIGN KEY("OrderID") REFERENCES "Orders"("OrderID") ON DELETE SET NULL,

CONSTRAINT "orderitems\_fk\_products" FOREIGN KEY("ProductID") REFERENCES "Products"("ProductID") ON DELETE SET NULL

);

**+ createTableIngreList:**

CREATE TABLE "IngreList" (

"ProductID" INTEGER,

"IngredientID" INTEGER,

"Quantity" INTEGER,

CONSTRAINT "ingrelist\_fk\_products" FOREIGN KEY("ProductID") REFERENCES "Products"("ProductID") ON DELETE SET NULL,

CONSTRAINT "ingrelist\_fk\_ingredients" FOREIGN KEY("IngredientID") REFERENCES "Ingredients"("IngredientID") ON DELETE SET NULL

);

**+ createTableIngredients:**

CREATE TABLE "Ingredients" (

"IngredientID" INTEGER,

"Name" VARCHAR(40),

"description" VARCHAR(40),

"Amount\_Left" INTEGER,

"PricePerUnit" DOUBLE,

PRIMARY KEY("IngredientID" AUTOINCREMENT)

);

**- BusinessQuery:**

**+ calculateTotalRevenueFromTo:**

SELECT sum(TotalCost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

AND Status = ‘F’;

**+ calculateTotalCostFromToQuery:**

SELECT sum(TotalCost) FROM BuyOrders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

AND Status = ‘F’;

**+ calculateAverageSpendPerOrderFromTo:**

SELECT avg(TotalCost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

**+ calculateTotalRevenueLastXYears:**

SELECT strftime('%Y', CreationDate), sum(Totalcost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%Y', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalRevenueLastXMonths:**

SELECT strftime('%Y-%m', CreationDate), sum(Totalcost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%Y-%m', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalRevenueLastXWeeks:**

SELECT strftime('%Y-%W', CreationDate), sum(Totalcost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%W', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalRevenueLastXDays:**

SELECT strftime(‘%d-%m-%Y', CreationDate), sum(Totalcost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%Y-%m-%d', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalCostLastXMonths:**

SELECT strftime('%Y-%m', CreationDate), sum(Totalcost) FROM BuyOrders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%Y-%m', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalCostLastXWeeks:**

SELECT strftime('%Y-%W', CreationDate), sum(Totalcost) FROM BuyOrders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%W', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC

**+ calculateTotalCostLastXDays:**

SELECT strftime(‘%d-%m-%Y', CreationDate), sum(Totalcost) FROM Orders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY strftime('%Y-%m-%d', CreationDate) ORDER BY strftime('%Y-%m-%d %H:%M:%S', CreationDate) ASC;

**-BuyOrderQuery:**

**+ deleteBuyOrderWithBuyOrderID:**

DELETE FROM BuyOrders

WHERE BuyOrderID = ‘BuyOrderID’;

**+ deleteBuyOrderItemsWithBuyOrderID:**

DELETE FROM BuyOrderItems

WHERE BuyOrderID = ‘BuyOrderID’;

**+ updateBuyOrdersSupplierID:**

UPDATE BuyOrders

SET SupplierID = 'newSupplierID ‘

WHERE BuyOrdersID = ‘BuyOrderID’

**+ updateBuyOrdersStatus:**

UPDATE BuyOrders

SET Status = 'newStatus ‘

WHERE BuyOrdersID = ‘BuyOrderID’

**+ updateBuyOrderItemsQuantity:**

UPDATE BuyOrders

SET Status = 'newQuantity ‘

WHERE BuyOrdersID = ‘BuyOrderID’

**+ updateBuyOrderItemsPricePerUnit:**

UPDATE BuyOrders

SET Status = 'newPricePerUnit ‘

WHERE BuyOrdersID = ‘BuyOrderID’

**+ deleteBuyOrder:**

DELETE FROM BuyOrders

WHERE BuyOrderID = ‘BuyOrderID’;

**+ showBuyOrderItems:**

SELECT \* FROM BuyOrders, BuyOrderItems

WHERE (BuyOrders.BuyOrderID = ‘BuyOrderID’) AND (BuyOrders.BuyOrderID = BuyOrderItems.BuyOrderID);

**+ calculateTotalCost:**

SELECT sum(Quantity \* PricePerUnit) FROM BuyOrderItems

WHERE BuyOrderItems.BuyOrderID = ‘BuyOrderID’;

**+ selectAllBuyOrders:**

SELECT BuyOrderID, SupplierID, Totalcost, CreationDate, Status FROM BuyOrders

**+ insertBuyOrder:**

INSERT INTO BuyOrders (SupplierID, totalCost, status, CreationDate) VALUES

(‘SupplierID’, ‘totalCost’, ‘F’, DATE('NOW'));

**+ recordBuyOrderItems:**

INSERT INTO BuyOrderItems (BuyOrderID, IngredientID, Quantity, PricePerUnit) VALUES (‘BuyOrderID’, ‘IngredientID’ , ‘quantity’, (SELECT PricePerUnit FROM Ingredients WHERE IngredientID = ‘ IngredientID ‘)

**+ lastestBuyOrderID:**

SELECT MAX(BuyOrderID) FROM BuyOrders;

**-CustomersQuery:**

**+ checkExistQuery:**

SELECT count(\*) FROM Customers:

WHERE Phone\_Number = ‘phoneNumber’;

**+ sortAllCustomersByTotalSpendDesc:**

SELECT C.CustomerID, SUM(O.TotalCost) AS Total

FROM Customers C JOIN Orders O ON C.CustomerID = O.CustomerID

GROUP BY C.CustomerID

ORDER BY Total DESC

LIMIT X;

**+ getOneCustomerTotalSpend:**

SELECT SUM(O.TotalCost) AS Total

FROM Orders O

WHERE CustomerID = ‘CustomerID’;

**+ getCustomerName:**

SELECT Name FROM Customer

WHERE Phone\_Number = ‘phoneNumber’;

**+ getCustomersWhoHaveSimilarNameTo:**

SELECT \* FROM customers

WHERE Name LIKE ‘%name%’;

**+ getCustomerName:**

SELECT Name FROM Customers

WHERE CustomerID = ‘ID’

**+ getCustomerID:**

SELECT CustomerID FROM Customers

WHERE Phone\_Number = ‘phoneNumber’;

**+ getOneCustomerTotalSpendFromTo:**

SELECT sum(Totalcost) FROM Orders

WHERE (CustomerID = ‘ID’)

AND CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’;

**+ updateCustomerAddress:**

UPDATE Customers

SET Address = ‘newAddress ‘

WHERE CustomerID = ‘CustomerID’;

**+ updateCustomerEmail:**

UPDATE Customers

SET Email = ‘newEmail’

WHERE CustomerID = ‘CustomerID’;

**+ updateCustomerName**

UPDATE Customers

SET Name = ‘newName’

WHERE CustomerID = ‘CustomerID’;

**+ updateCustomerPhoneNumber:**

UPDATE Customers

SET Phone\_Number = ‘newPhoneNumber ‘

WHERE CustomerID = ‘CustomerID’;

**+ getTopXSpendCustomersFromTo:**

SELECT Customers.Name, Customers.Phone\_Number AS INFO, sum(Totalcost - Discount) as Pay FROM Orders JOIN Customers on Orders.CustomerID = Customers.CustomerID

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Customers.CustomerID

ORDER BY Pay DESC LIMIT X;

**+ getAllCustomer:**

SELECT Name, Phone\_Number, Address, Email, CustomerID FROM Customers;

**+ insertCustomer:**

INSERT INTO Customers (Address, Phone\_Number, Name, Email) VALUES

(‘address, ‘phoneNumber’ ,’name’, ‘email’);

**-IngredientQuery:**

**+ insertIngredient:**

INSERT INTO Ingredients (Name, description, Amount\_Left, PricePerUnit) VALUES

(‘name’, ‘description’, ‘amountleft’, ‘price’);

**+ getAllIngredient:**

SELECT Name, description, Amount\_Left, PricePerUnit FROM Ingredients;

**+ calculateOneIngredientCostFromTo:**

SELECT SUM(OI.PricePerUnit\*OI.Quantity)

FROM BuyOrderItems as OI JOIN BuyOrders AS O ON (OI.BuyOrderID = O.BuyOrderID)

WHERE (O.CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

AND (OI.IngredientID = ‘ID’);

**+ calculateAllIngredientCostFromTo:**

SELECT sum(Totalcost) FROM BuyOrders

WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’;

**+ getIngredientName:**

SELECT Name FROM Ingredients

WHERE IngredientID = ‘ID’;

**+ updateIngredientName:**

UPDATE Ingredients

SET Name = 'newName'

WHERE IngredientID = ‘ID’

**+ getIngredientsWithSimilarName:**

SELECT \* FROM Ingredients

WHERE Name LIKE '% pattern%';

**+ updateIngredientAmountLeft:**

UPDATE Ingredients

SET AmountLeft = 'newAmountLeft'

WHERE IngredientID = ‘ID’;

**+ updateIngreListQuantity**:

UPDATE Ingredients

SET Quantity = 'newQuantity'

WHERE IngredientID = ‘ID’;

**+ updateIngredientdescription:**

UPDATE Ingredients

SET description = 'newdescription'

WHERE IngredientID = ‘ID’;

**+ topXmostcostingredientsYdays:**

SELECT Ingredients.IngredientID, Name, sum(BuyOrderItems.PricePerUnit\*Quantity) as totalcost FROM BuyOrderItems, Ingredients

WHERE BuyOrderItems.IngredientID=Ingredients.IngredientID

AND CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Ingredients.IngredientID

ORDER BY totalcost DESC LIMIT X;

**+ topXmostcostingredientsYmonths:**

SELECT Ingredients.IngredientID, Name, sum(BuyOrderItems.PricePerUnit\*Quantity) as totalcost FROM BuyOrderItems, Ingredients

WHERE BuyOrderItems.IngredientID=Ingredients.IngredientID

AND CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Ingredients.IngredientID

ORDER BY totalcost DESC LIMIT X;

**+ topXmostcostingredientsYyears:**

SELECT Ingredients.IngredientID, Name, sum(BuyOrderItems.PricePerUnit\*Quantity) as totalcost FROM BuyOrderItems, Ingredients

WHERE BuyOrderItems.IngredientID=Ingredients.IngredientID

AND CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Ingredients.IngredientID

ORDER BY totalcost DESC LIMIT X;

**-OrderQuery:**

**+ deleteOrderWithID:**

DELETE FROM Orders

WHERE OrderID = ‘OrderID’;

**+ deleteOrderItemsWithID:**

DELETE FROM OrdersItems

WHERE OrderID = ‘OrderID’;

**+ updateOrderItemsQuantity:**

UPDATE Orders

SET Quantity = 'newQuantity'

WHERE OrderID = ‘ID’;

**+ updateOrdersStatus:**

UPDATE Orders

SET Status= 'newStatus'

WHERE OrderID = ‘ID’;

**+ updateOrdersDiscount:**

UPDATE Orders

SET Discount = 'newDiscount'

WHERE OrderID= ‘ID’;

**+ deleteOrder:**

DELETE FROM Orders WHERE OrderID= ‘ID’;

**+ showOrderItems:**

SELECT OI.ProductID, OI.Quantity, P.Product\_Descript, P.Selling\_Price FROM OrderItems AS OI, Products as P

WHERE (OI.OrderID = "OrderID ") AND (OI.ProductID = P.ProductID)

**+ removeItemfromOrder:**

DELETE FROM OrderItems

WHERE OrderID = ‘OrderID’ AND ProductID =’ProductID’;

**+ calculateTotalCost:**

SELECT sum(Quantity \* PricePerUnit) FROM OrderItems

WHERE OrderID = ‘OrderID’;

**+ addOrder:**

INSERT INTO Orders (CustomerID, Totalcost, CreationDate, Status, Discount) VALUES

(‘CustomerID’, ‘TotalCost’, DATE('now'), ‘status’, ‘discount’);

**+ recordItemIntoOrder:**

INSERT INTO OrderItems (OrderID, ProductID, Quantity, PricePerUnit) VALUES

(‘OrderID’, ‘ProductID’, ‘quantity’, (SELECT Selling\_Price FROM Products WHERE ProductID =’ProductID ‘));

**+ displayAllItemsInOrder:**

SELECT OrderItems.ProductID as PID, Products.Product\_Descript AS Name, OrderItems.Quantity AS [Amount], OrderItems.PricePerUnit AS [Price Per Unit] FROM OrderItems, Products

WHERE Products.ProductID = OrderItems.ProductID AND OrderID = ‘OrderID’;

**+ addCustomer:**

INSERT INTO Customers (Address, Phone\_Number, Name, Email) VALUES

(‘address’, ‘phoneNumber’, ‘Name’, ‘email’);

**+ selectAllOrders:**

SELECT OrderID, CustomerID, Totalcost, CreationDate, Status, Discount FROM Orders;

**+ lastOrderID:**

SELECT MAX(OrderID) FROM Orders;

**-ProductQuery:**

**+ selectAll:**

SELECT Product\_Descript, Amount\_Left, Selling\_Price, ProductID FROM Products;

**+ updatePrice:**

UPDATE Products

SET Selling\_Price = ‘newPrice’

WHERE ProductID = ‘ProductID’;

**+ updateName:**

UPDATE Products

SET Name = ‘newName’

WHERE ProductID = ‘ProductID’;

**+ updateAmountLeft:**

UPDATE Products

SET AmountLeft = ‘newAmountLeft’

WHERE ProductID = ‘ProductID’;

**+ calculateOneProductRevenueFromTo:**

SELECT SUM(OI.PricePerUnit\*OI.Quantity) FROM OrderItems as OI

JOIN Orders AS O ON (OI.OrderID = O.OrderID)

WHERE (O.CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

AND (OI.ProductID = ‘ID’);

**+ getProductName:**

SELECT Product\_Descript FROM Products

WHERE ProductID = ‘ID’;

**+ calculateAllProductRevenueFromTo:**

SELECT OI.ProductID, SUM(OI.PricePerUnit\*OI.Quantity) FROM OrderItems as OI

JOIN Orders AS O ON (OI.OrderID = O.OrderID)

WHERE (O.CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY OI.ProductID

ORDER BY Total DESC;

**+ getProductPrice:**

SELECT Selling\_Price FROM Products

WHERE ProductID = ‘ID’;

**+ getProductsWithSimilarName:**

SELECT \* FROM Products

WHERE Product\_Descript LIKE ‘%pattern%’;

**+ getTopXSellingProductsFromTo:**

SELECT Products.Product\_Descript, sum(Quantity) as Sold FROM OrderItems JOIN Products on OrderItems.ProductID = Products.ProductID

WHERE OrderID IN (SELECT OrderID FROM Orders WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Products.ProductID

ORDER BY Sold DESC LIMIT X;

**+ getLeastXSellingProductsFromTo:**

SELECT Products.Product\_Descript, sum(Quantity) as Sold FROM OrderItems JOIN Products on OrderItems.ProductID = Products.ProductID

WHERE OrderID in (SELECT OrderID FROM Orders WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Products.ProductID

ORDER BY Sold ASC LIMIT X;

+ insertProduct:

SELECT Products.Product\_Descript, sum(Quantity) as Sold FROM OrderItems JOIN Products ON OrderItems.ProductID = Products.ProductID

WHERE OrderID IN (SELECT OrderID FROM Orders WHERE CreationDate BETWEEN ‘yyyy-mm-dd’ AND ‘yyyy-mm-dd’

GROUP BY Products.ProductID

ORDER BY Sold DESC LIMIT X;

**-SupplierQuery:**

**+ getAllSuppliers:**

SELECT SupplierID, Name, Address, Phone\_Number, Email FROM Suppliers;

**+ getSuppliersWithSimilarNameTo:**

SELECT \* FROM Suppliers

WHERE name LIKE '%s%’;

**+ checkExist:**

SELECT count(\*) FROM Suppliers

WHERE Phone\_Number = ‘phoneNumber’;

**+ getSupplierName (ID):**

SELECT Name FROM Suppliers

WHERE SupplierID = ‘ID’;

**+ getSupplierName (phonenumber):**

SELECT Name FROM Suppliers

WHERE Phone\_Number = ‘PhoneNumber’;

**+ updateSupplierName:**

UPDATE Suppliers

SET Name = 'newName’

WHERE SupplierID = ‘ID’;

**+ updateSupplierAddress:**

UPDATE Suppliers

SET Address = 'newAddress’

WHERE SupplierID = ‘ID’;

+ updateSupplierPhoneNumber:

UPDATE Suppliers

SET Phone\_Number = 'newPhoneNumber’

WHERE SupplierID = ‘ID’;

**+ topXbestsuppliersYdays:**

SELECT Name, sum(Totalcost) from BuyOrders, Suppliers

WHERE Suppliers.SupplierID=BuyOrders.SupplierID

AND CreationDate BETWEEN (‘yyyy-mm-dd’) AND (‘yyyy-mm-dd’)

GROUP BY Suppliers.SupplierID

ORDER BY sum(Totalcost) DESC LIMIT "+x;

**+ topXbestsuppliersYmonths:**

SELECT Name, sum(Totalcost) from BuyOrders, Suppliers

WHERE Suppliers.SupplierID=BuyOrders.SupplierID

AND CreationDate BETWEEN (‘yyyy-mm-dd’) AND (‘yyyy-mm-dd’)

GROUP BY Suppliers.SupplierID

ORDER BY sum(Totalcost) DESC LIMIT "+x;

**+ topXbestsuppliersYyears:**

SELECT Name, sum(Totalcost) from BuyOrders, Suppliers

WHERE Suppliers.SupplierID=BuyOrders.SupplierID

AND CreationDate BETWEEN (‘yyyy-mm-dd’) AND (‘yyyy-mm-dd’)

GROUP BY Suppliers.SupplierID

ORDER BY sum(Totalcost) DESC LIMIT "+x;

**-DroppingQuery:**

**+ dropCustomersTable:**

DROP TABLE IF EXISTS Customers;

**+ dropSuppliersTable:**

DROP TABLE IF EXISTS Suppliers;

**+ dropOrdersTable:**

DROP TABLE IF EXISTS Orders;

**+ dropBuyOrdersTable:**

DROP TABLE IF EXISTS BuyOrders;

**+ dropProductsTable:**

DROP TABLE IF EXISTS Products;

**+ dropIngredientsTable:**

DROP TABLE IF EXISTS Ingredients;

**+ dropOrderItemsTable:**

DROP TABLE IF EXISTS OrderItems;

**+ dropBuyOrderItemsTable:**

DROP TABLE IF EXISTS BuyOrderItems;

**+ dropIngreListTable:**

DROP TABLE IF EXISTS IngreList;