**Project Title**

**Grocery Price Comparison Application**

**Team: 2**

1.Afroz Inamdar L3SID-174

2.Fnu Butul Parveen (Team Lead) L3SID-227

3.Di Chen L3SID-760

4.Rohith Puvvala Subramanyam L3SID-536

5.Rohitkumar Yadav L3SID-961

6.Suhas Anand Balagar L3SID-540

**The choice of database project**

| The choice of database engine | Mysql workbench 8.0.16 |
| --- | --- |
| DB application technologies | Flask |
| Frameworks | Flask |
| Languages | SQL, Python |
| DB access technology | Rest API |

**Introduction**

The “Grocery Price Comparison” application enables the user to compare the price of his/her shopping list across different grocery stores. The customer has to sign up for the first time using the app or sign in to login into the account. The customer can add/remove multiple items to a shopping list.The customer can see the prices of various items in the shopping list across various stores. The application will display the individual prices of the products in various stores and automatically place the order at the store offering the cheapest price of the item where the required quantity is available. In case a particular store has less quantity than the customer’s requirement, the available quantity in the store will be ordered and the remaining quantity will be ordered from the next cheapest store. The order for the shopping list in different grocery stores in the locality will be delivered to the zip code of the user. The customer can place multiple orders at stores, which is offering the best price for a set of items. Each store has an Authorized person, who can update the price of each item in the corresponding store.

**Purpose of Application/database and Intended Users**

The main purpose of this Application is to enable customers to compare the rates of different grocery items at different grocery stores and provide the recommendation of the store offering the cheapest price for customer’s Shopping List. This application is intended for customers who are on the lookout for ways to easily compare grocery rates for the same set of items at different stores. The customer has the ability to place multiple orders at the grocery store offering best prices.

**Objects/ Roles:**

**Grocery\_store**: Consists of various categories of items and delivers them to customers.

**Grocery\_item:** various categories of products available for purchase

**Order**: information about the products purchased by customers

**Roles**:

**Customer**: people who use this comparison app to place orders

**Role 11:** Customer

**Authorized\_person**: people who are in charge of the grocery stores and finish the order

**Role 21:** manage store info.

**Planned functionality and operations Customer**

* + Sign up and Sign in
  + Add/ remove items from Shopping List
  + Compare items price
  + Place orders

**Authorized\_Person**

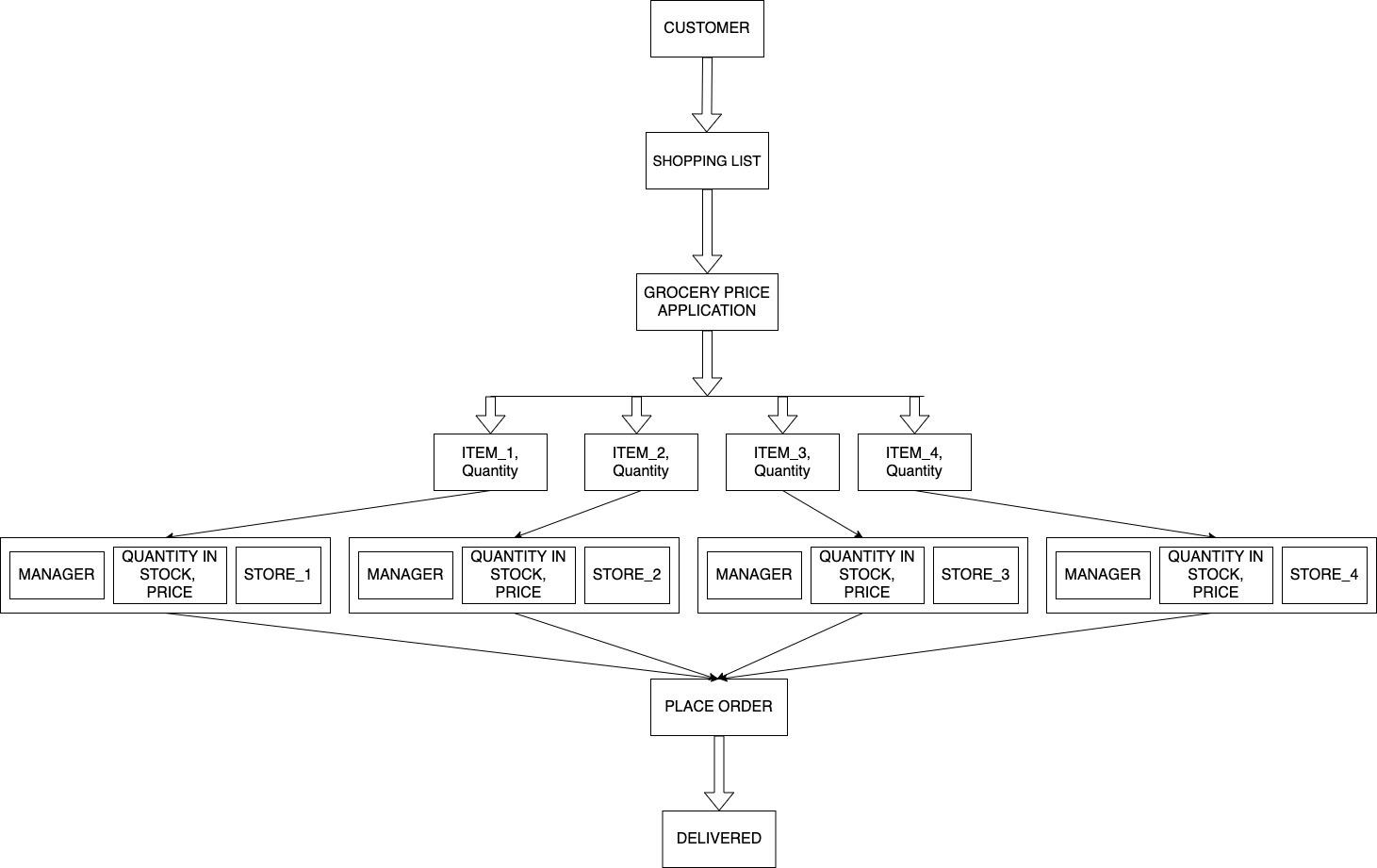
* + Sign in
  + Manage and Update item information

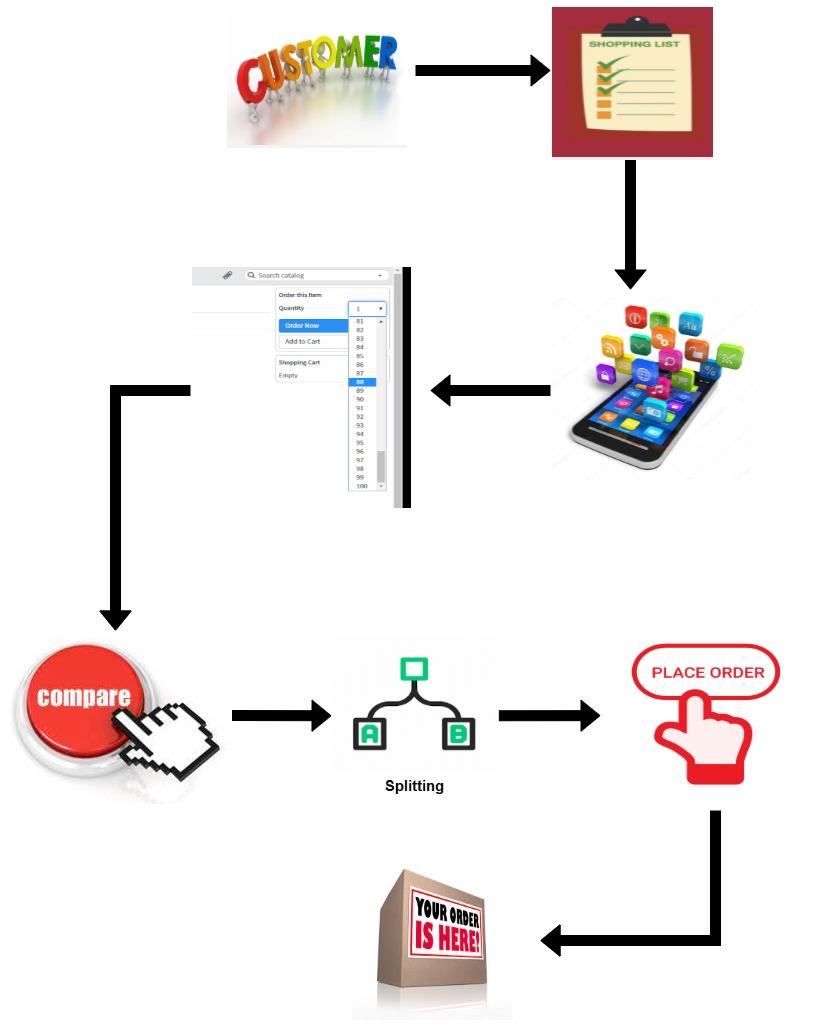
**Scenarios**

Obama found that he was running out of grocery items in his house. He decided to place an order but got confused about which store would give him the best deal to order items for the coming month in bulk quantity. He then signed up to the Grocery comparison app by providing his information such as email, phone number, etc. He then signed in to the app by entering his customer\_ID and password. He added multiple Items (from different categories) he needed for his house in the Shopping list covering details of quantity and the brand he needed on the grocery comparison app. His total price list amounted to 200 dollars at store A. Then Obama chose the stores within 5 miles of his house and found there are 10 stores and began to compare, on the app he realized that store B was giving him the same list of items at 50 dollars less than store A. And by splitting his order into two parts, one from store B and the other from store C he can save up to 75 dollars. His happiness knew no bounds, realizing that he saved 50 dollars by just using the app and found the best deal available in the stores nearby his address. Later, he realized he needed to add Milk and remove Bread from his Shopping List. He Modified the Shopping List and compared the prices again.

The customer can split the order based on an offer from different grocery stores for multiple items, some items can be ordered from store A, decides that store B was offering the best deal for items in his shopping list, and decides to place an order at store B and this might reduce the total cost. Before he pays, he is asked to provide his home address and payment method. Then he opts to pay with a credit card. After two hours of waiting, a delivery guy brings the items he bought to his front door. But when he is about to open the milk, he finds the expiration date is only three days, so he decides to contact the store and requests the Authorized person (Manager) for a return and refund. Obama is later notified of a price drop for one of his shopping lists a week later, he places another at store X and gets the groceries he wants with the best deal.

**Final overall architecture:block/component diagram**

****

****

**Planned functionality and operations**

* Each grocery store has a unique store\_id, a store\_name, a store website and a store location, including street, zip code, city, state. Each store must have 1 authorized person, must offer no less than M items, and may ProcessOrder up to M orders per day.
* Each authorized person has a unique id m\_id, a phone number m\_phone. Each authorized person must manage exactly one grocery store of store\_id. The authorized person must sign in to his/her store portal using his/her password and must manage a single grocery store.
* Offers table has a unique store\_id for each store which offers items with unique id for each item. The offers table also has the price of each item, best\_before\_date for each item and the quantity of the item in the stock.
* Each order has a unique order\_ID, ordered details such as order\_time, order\_date, as well as, Delivery\_date. Each payment is done by payment\_ mode either cash or card. Each order must be CompareAndSplit by exactly a single shopping\_list and must be placeOrder by exactly 1 customer.Each order must be ProcessOrder exactly by one grocery store. Each order must record upto N items and each item may be recorded upto M order. Each order\_id must have at least one shopping list.
* Each Shopping\_List must have a unique shopping\_list\_id, and it has a derived attribute total\_cost must contain upto N Items. Each shopping\_List must be owned by exactly 1 customer. Each shopping\_list may be CompareAndSplit upto M orders by the following algorithm. First we list out the stores that deliver to the customers and then we compare the quantity\_In\_Store with the lowest price, and if it is less than the demand, we make up for it from the store with the second lowest price, and so on.
* Each Customer has to sign up with a unique customerID, encrypted customerPassword, and must provide details such as customer\_Name, customer\_Phone, customer\_Email, and a composite attribute customer\_address which contains attributes like street, zip code, city, state. The customer can sign in by entering his/her customerID and encrypted customerPassword. Each Customer may placeOrder upto N orders. Each Customer may own at least one or more ShoppingLists.
* Each Item has a unique id Item\_ID, along with unit, brand. An Item may be offered by up to N different stores and offer may consist of price and Best\_Before\_Date and derived attributes based on price\_Per\_Unit and quantity\_In\_Stock. Items must be classified into different categories such as canned, Diary, Snacks, Bread & Bakery, Frozen Foods, Meat & Seafood, and Miscellaneous. Each item may contain up to M shopping\_List. Each item may be recorded up to by M orders.
* In contains table each shopping list must contain upto N items. It also has the quantity attribute to show the quantity of items in that particular shopping list.
* In the records table each order must contain upto N items. It also has the quantity attribute to show the quantity of items in that particular order.
* Each grocery store may process upto m orders and each order must be processed by one grocery store.

**Major Tasks – Team Management**

**1.Afroz Inamdar L3SID-174**

* Project proposal Idea
* EER Diagram(Customer, Shopping List)
* Schema Diagram(Customer, Shopping List)
* Table Creation(Customer, Shopping List)
* Project Report(1-7) / Slides(1-5)
* Password Encryption and Transaction
* Application Design
* Implementation(Customer Sign up)

**2.Fnu Butul Parveen (Team Lead) L3SID-227**

* Project proposal Idea
* EER Diagram(Record, Order)
* Schema Diagram(Record, Order)
* Functional Dependencies and Normalization
* Table Creation (Record, Order) and Insertion values into tables
* Project Report(8-14) / Slides(6-10)
* Application Design
* Implementation(Customer Log in)

**3.Di Chen L3SID-760**

* Project proposal Idea
* EER Diagram(Offer, Grocery Store)
* Schema Diagram(Offer, Grocery Store)
* Table Creation(Offer, Grocery Store
* Functional Dependencies and Normalization
* Stored procedures and views
* Project Report(15-20)/Slides(11-14)
* Application Design
* Implementation(Store Inventory update)

**4. Rohith Puvvala Subramanyam L3SID-536**

* Project proposal Idea
* EER Diagram(Authorized person, Place Order and Grocery)
* Schema Diagram(Authorized person, Place Order and Grocery)
* Table Creation(Authorized person, Place Order and Grocery)
* Stored procedures and views
* Project Report(21-27)/Slides(15-18)
* Implementation(Item Page)

**5. Rohitkumar Yadav L3SID-961**

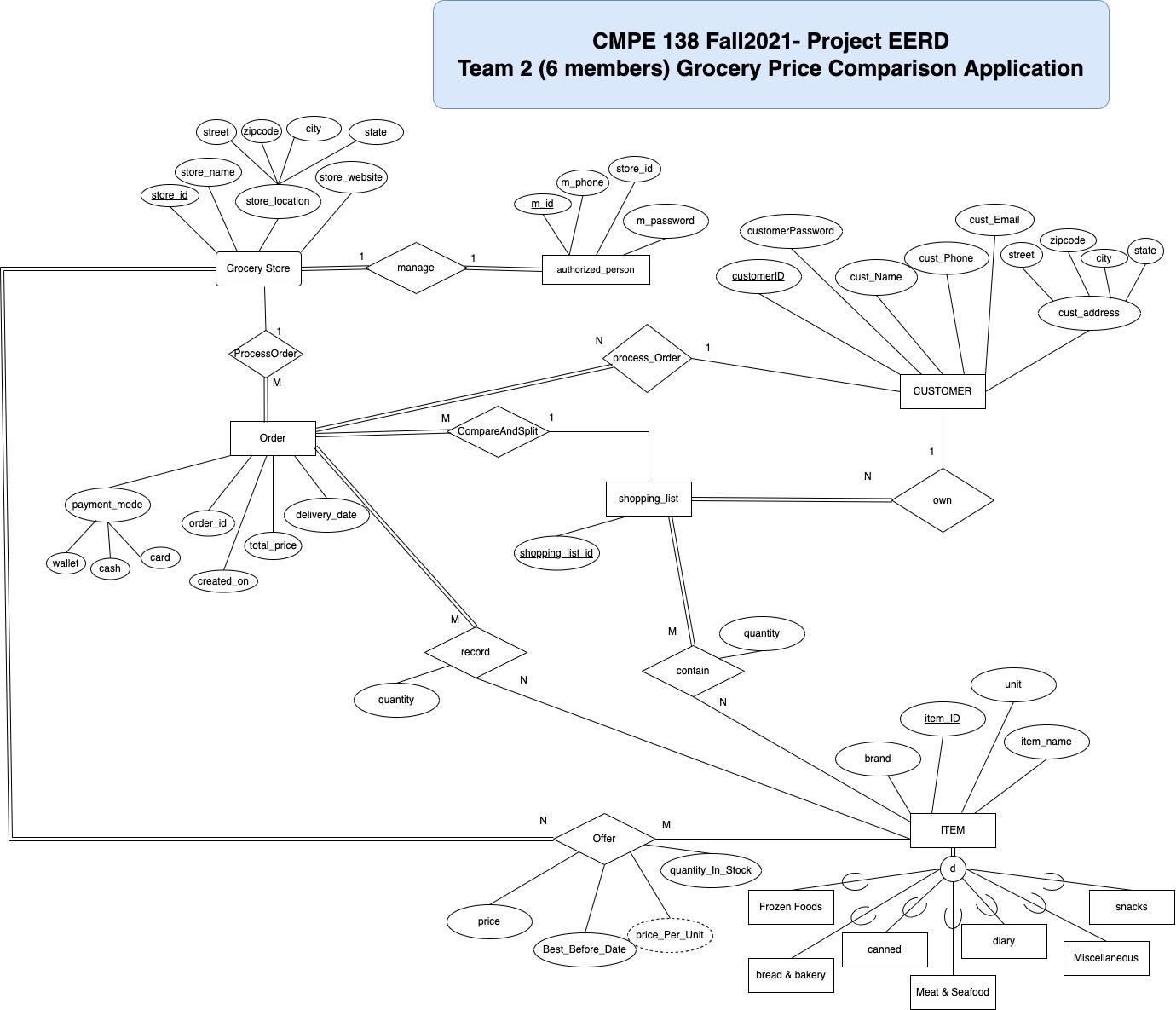
* Project proposal Idea
* EER Diagram(Contain, Item)
* Schema Diagram(Contain, Item)
* Table Creation(Contain, Item)
* Stored procedures and views
* Project Report(28-34)/Slides(19-22)
* Password Encryption and Transaction
* Implementation(Authorised user login)

**6. Suhas Anand Balagar L3SID-540**

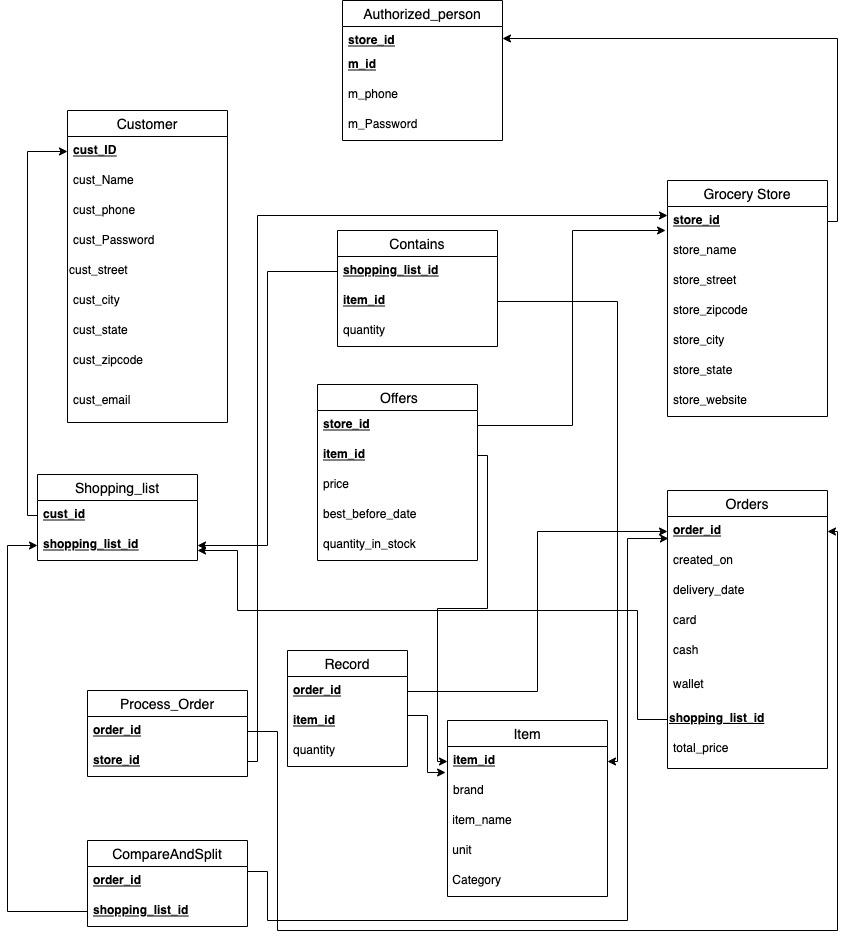
* Project proposal Idea
* EER Diagram(Compare and split, Order)
* Schema Diagram(Compare and split, Order)
* Table Creation(Compare and split, Order)
* Application Design
* Implementation(Compare and split algorithm, order Processing ) and Final design of DB application.
* Log Tracing and Testing
* Project Report(36-41)/Slides(23-28)

**The final design of EER diagram**

After the approval of EERD diagram, Quantity\_in\_stock is changed to a regular attribute as it should be an attribute of store inventory, it will be updated after someone made purchase and it could be derived from nowhere. Therefore, now it’s changed to a regular attribute as below.

****

**Schema diagram**



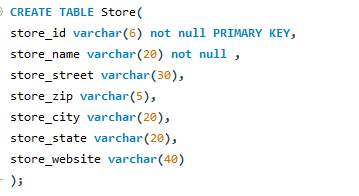
We have implemented the disjoint specializations as a category column in Store Table as the subsets do not have their Individual Attributes.

**Specification of each DB object:**

**Grocery store:**

| **store\_id** | **store\_name** | **store\_street** | **store\_zipcode** | **store\_city** | **store\_state** | **store\_website** |
| --- | --- | --- | --- | --- | --- | --- |

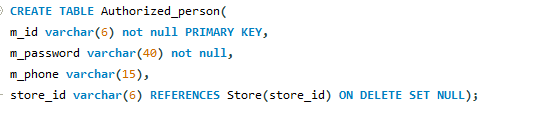
* **store\_id**: Each store must have a unique id for identifying the stores of the same chain.
* **store\_name:** Store name for each store will be useful for the customers while selecting items.
* **store\_address:** It is a composite attribute which has street, zip code, city, state. It is useful for the customer for locating the particular store.
* **store\_website:** Each store should have a store website for posting all the deals.



**Authorized\_person:**

| **m\_id** | ***store\_id*** | **m\_phone** | **m\_password** |
| --- | --- | --- | --- |

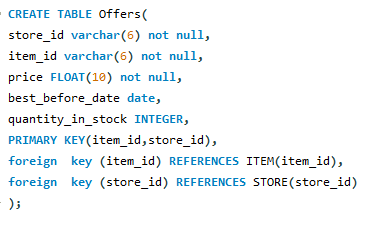
* **m\_id:** Each authorised person must have a unique id for logging in, update the store portal with quantity of items in stock, price of the items.
* **store\_id:** Authorised person must bind to a store to be able to change item and quantity for each store.
* **m\_phone:** Each authorised person has a phone number for interacting with the customers.
* **m\_password:** Each authorised person must have a password for logging in to the store portal.



**Offers:**

| ***store\_id*** | ***item\_id*** | **price** | **best\_before\_date** | **quantity\_in\_stock** |
| --- | --- | --- | --- | --- |

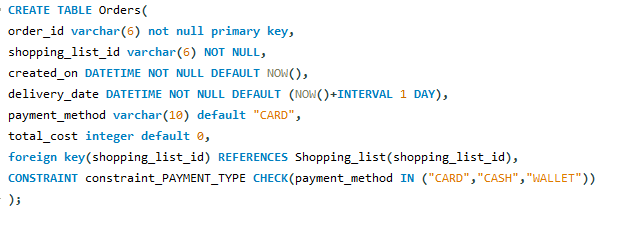
* **store\_id:** Each store must have a unique id for identifying the stores of the same chain as some stores offer different items and different prices based on their location.
* **item\_id:** Each item must have a unique id for identifying it in the store.
* **price:** It indicates the cost of each individual item.
* **best\_before\_date:** It indicates the date before which the item should be used. For some items like furniture there is no need of best\_before\_date.
* **quantity\_in\_stock:** It indicates the quantity of each item in the stock so that the customer can find out the right amount to order based on the items in the stock in the store.



**Orders:**

| **order\_id** | **order\_time** | **order\_date** | **delivery\_date** | **payment\_mode** | ***shopping\_list\_id*** | **total\_price** |
| --- | --- | --- | --- | --- | --- | --- |

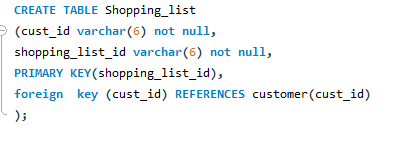
* **order\_id:** Each order must have a unique order\_id to deliver to the right customer without any discrepancies.
* **order\_time:** It indicates the time at which the customer has placed the order.
* **order\_date:** It indicates the date on which the customer has placed the order.
* **delivery\_date:** It indicates the expected delivery date of the order to the customer.
* **payment\_mode:** It indicates the mode of payment the customer pays the amount to be paid, customers have options like card and cash on delivery.
* **shopping\_list\_id:** Each order must be compared and split by one shopping list with a unique shopping\_list\_id.
* **total\_price:** Each order must be compared and split by one shopping list and total price must be computed accordingly



**Shopping\_list:**

| ***cust\_id*** | **shopping\_list\_id** |
| --- | --- |

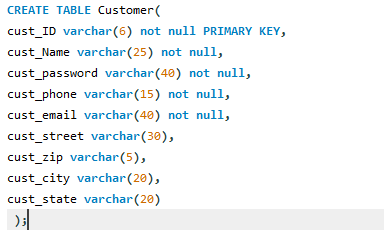
* **cust\_id:** Each customer must have a unique shopping list in order to avoid discrepancies.
* **shopping\_list\_id:** Each order must be compared and split by one shopping list with a unique shopping\_list\_id.



**Customer:**

| **cust\_ID** | **cust\_Name** | **cust\_Password** | **cust\_phone** | **cust\_email** | **cust\_street** | **cust\_zip** | **cust\_city** | **cust\_state** |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |

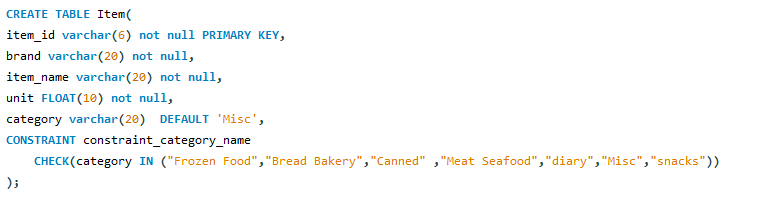
* **cust\_ID:** Each customer must have a unique customer id to login into the store portal for ordering the items he/she needed.
* **cust\_Name:** Each customer must provide his/her name in order to identify while interacting or delivering the items.
* **cust\_Password:** Each customer must have a password in order to log into the store portal with cust\_id.
* **cust\_phone:** Each customer must have a phone number in order to reach out while delivering the order.
* **cust\_email:** It is used to give updates regarding the order of the customer. It is also useful for the verification purpose.
* **cust\_address:** Each customer needs to provide an address which includes street, zip, city, state for the delivery purpose.



**Item**:

| **item\_id** | **brand** | **item\_name** | **unit** | **category** |
| --- | --- | --- | --- | --- |

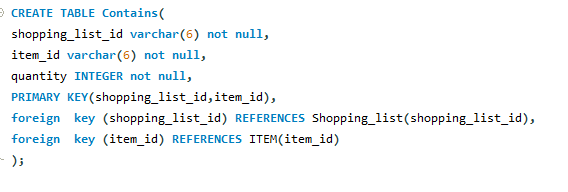
* **item\_id:** Each item is assigned with a unique item\_id which is useful while locating the item in store.
* **brand:** It indicates the brand name of the item which is helpful to the customers to select the item of their preferred brands.
* **item\_name:** It indicates the name of the item which helps the customer while choosing the items.
* **unit:** It indicates the weight or volume of the item which would be helpful for the customer to choose the right amount of the item.
* **category:** It indicates the category of the item which would be helpful for the customer while ordering the items which reduces the burden of searching for the item.



**Contains:**

| ***shopping\_list\_id*** | ***item\_id*** | **quantity** |
| --- | --- | --- |

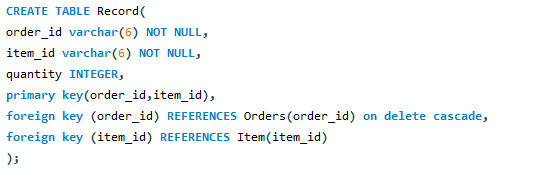
* **shopping\_list\_id:** Each order must be compared and split by one shopping list with a unique shopping\_list\_id.
* **item\_id:** Each item is assigned with a unique item\_id which is useful while locating the item in store.
* **quantity:** It indicates the quantity of the items in the shopping list.



**Record:**

| ***order\_id*** | ***item\_id*** | **quantity** |
| --- | --- | --- |

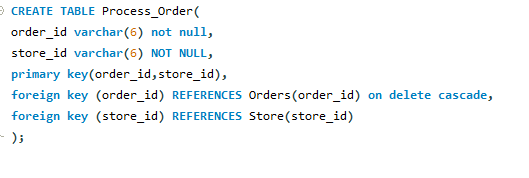
* **order\_id:** Each order must have a unique order\_id to deliver to the right customer without any discrepancies.
* **item\_id:** Each item is assigned with a unique item\_id which is useful while locating the item in store.
* **quantity:** It indicates the quantity of the items in the shopping list.



**Process\_Order:**

| ***order\_id*** | ***store\_id*** |
| --- | --- |

* **order\_id:** Each order must have a unique order\_id to deliver to the right customer without any discrepancies.
* **store\_id:** Each store must have a unique id for identifying the stores of the same chain as some stores offer different items and different prices based on their location.



**Functional dependencies of each table and normalization**

**Functional dependencies:**

* Customer

Cust\_id -> { cust\_name, cust\_phone, cust\_password, cust\_street, cust\_city, cust\_state, cust\_zipcode, cust\_email}

* Process\_order

{ order\_id, store\_id }

* Authorized\_person

{ store\_id, m\_id} -> { m\_password, m\_phone}

* Contains

{ shopping\_list\_id, Item\_id } -> quantity

* Offers

{ store\_id, item\_id} -> { price, best\_before\_date, quantity\_in\_stock}

* Record

{ order\_id, item\_id } -> quantity

* Item

Item\_id -> { brand, item\_name, unit, category }

* grocery store

store\_id -> { store\_name, store\_street, store\_zipcode, store\_city, store\_state, store\_website }

* Orders

{ Order\_id, shopping\_list\_id } -> { created\_on, delivery\_date, card, cash, valid, total\_price }

* Shopping\_list

{ Shopping\_list\_id, cust\_id }

**Normalization:**

Normalization is used to decompose a larger, complex table into simple and smaller ones. This helps us in removing all the redundant data. In most tables, there will be a lot of redundant information that isn't needed, thus it's wiser to break up this large database into smaller tables that only include unique data.

**First normal form:**

A relation schema is in 1NF, if and only if:

All attributes in the relation are atomic(indivisible value)

And there are no repeating elements or groups of elements.

**Second normal form:**

A relation is said to be in 2NF, if and only if:

It is in 1st Normal Form.

No partial dependency exists between non-key attributes and key attributes.

**Third Normal form:**

A relation R is said to be in 3NF if and only if: It is in 2NF.No transitive dependency exists between non-key attributes and key attributes through another non-key attribute

**Normalization for Grocery store comparison:**

* Customer (cust\_ID, cust\_Name, cust\_phone, cust\_Password, cust\_street, cust\_city, cust\_state, cust\_zip, cust\_email)

Normalization form: 3NF

PK: cust\_ID

* Shopping\_list ( Shopping\_list\_id, cust\_id)

Normalization form: 3NF

Composite PK: ( Shopping\_list\_id, cust\_id)

FK: (cust\_id) -> customer (cust\_id)

* Process\_order (order\_id, store\_id)

Normalization form: 3NF

Composite PK: (order\_id, store\_id)

FK: (order\_id) -> orders(order\_id)

(store\_id) -> grocery\_store(store\_id)

* CompareAndSplit (order\_id, shopping\_list\_id)

Normalization form: 3NF

Composite PK: (order\_id, shopping\_list\_id)

FK: (order\_id) -> orders(order\_id)

(shopping\_list\_id) -> shopping\_list (shopping\_list\_id)

* Authorized\_person (store\_id, m\_id, m\_password, m\_phone)

Normalization form: 3NF

Composite PK: (store\_id, m\_id)

FK: (store\_id) -> grocery\_store(store\_id)

* Contains (shopping\_list\_id, Item\_id, quantity)

Normalization form: 3NF

Composite PK: (shopping\_list\_id, Item\_id)

FK: (shopping\_list\_id) -> shopping\_list (shopping\_list\_id)

(Item\_id) -> item (Item\_id)

* Offers (store\_id, item\_id, price, best\_before\_date, quantity\_in\_stock)

Normalization form: 3NF

Composite PK: (store\_id, Item\_id)

FK: (store\_id) -> grocery\_store(store\_id)

(Item\_id) -> item (Item\_id)

* Record (order\_id, item\_id, quantity)

Normalization form: 3NF

Composite PK: (order\_id, item\_id)

FK: (order\_id) -> orders(order\_id)

(item\_id) -> item (item\_id)

* Item (item\_id, brand, item\_name, unit, category)

Normalization form: 3NF

PK: item\_id

* grocery store (store\_id, store\_name, store\_street, store\_zip, store\_city, store\_state, store\_website)

Normalization form: 3NF

PK: store\_id

* Orders (order\_id, created\_on, delivery\_date, card, cash, shopping\_list\_id, total\_price)

Normalization form: 3NF

Composite PK: (order\_id, shopping\_list\_id)

FK: (shopping\_list\_id) -> shopping\_list (shopping\_list\_id)

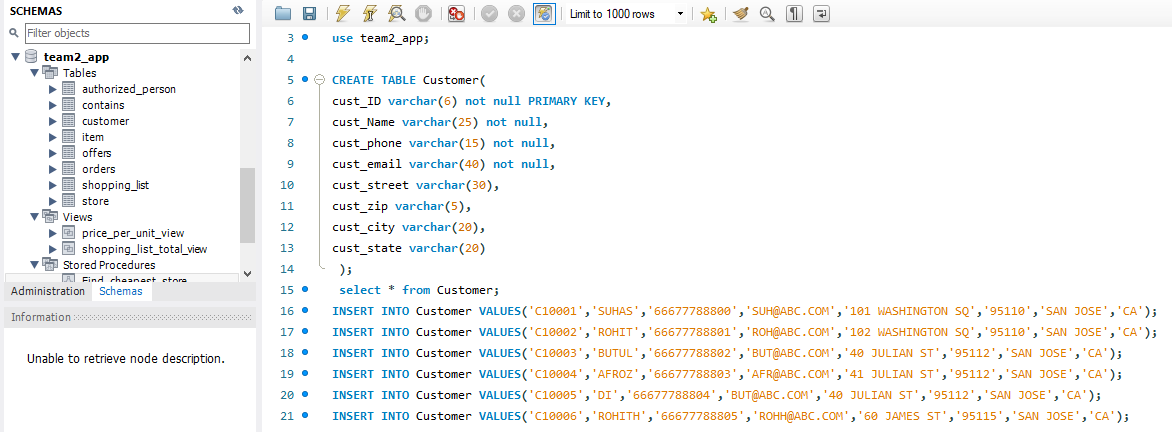
**User login password**

We have used the Customer table for the user login password, based on registration the customer choses a password for registering the grocery price comparison application, the password is **encrypted** and stored in the database of the customer table.

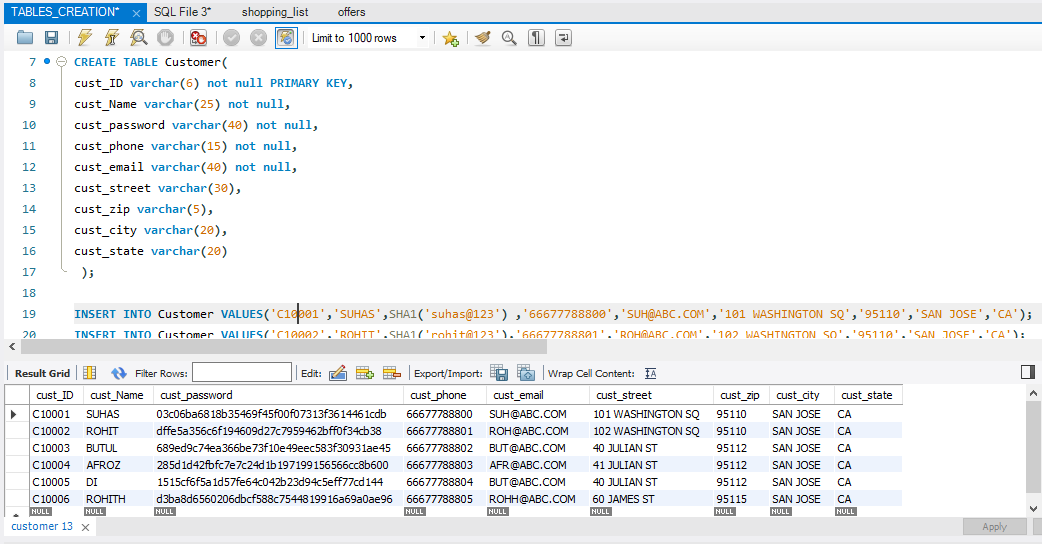
We have used a password in SHA1() function format, the SHA1() function in MySQL is used to encrypt a string using the SHA-1 algorithm. The SHA1 algorithm, which stands for the secure hash algorithm, generates a 160-bit checksum for a user-supplied string. If the string supplied as an argument is NULL, the MySQL SHA1() method returns NULL. The SHA1() function takes only one parameter, which is the **encrypted** string.

After adding the password from the application the customer table is updated in the below screenshot.

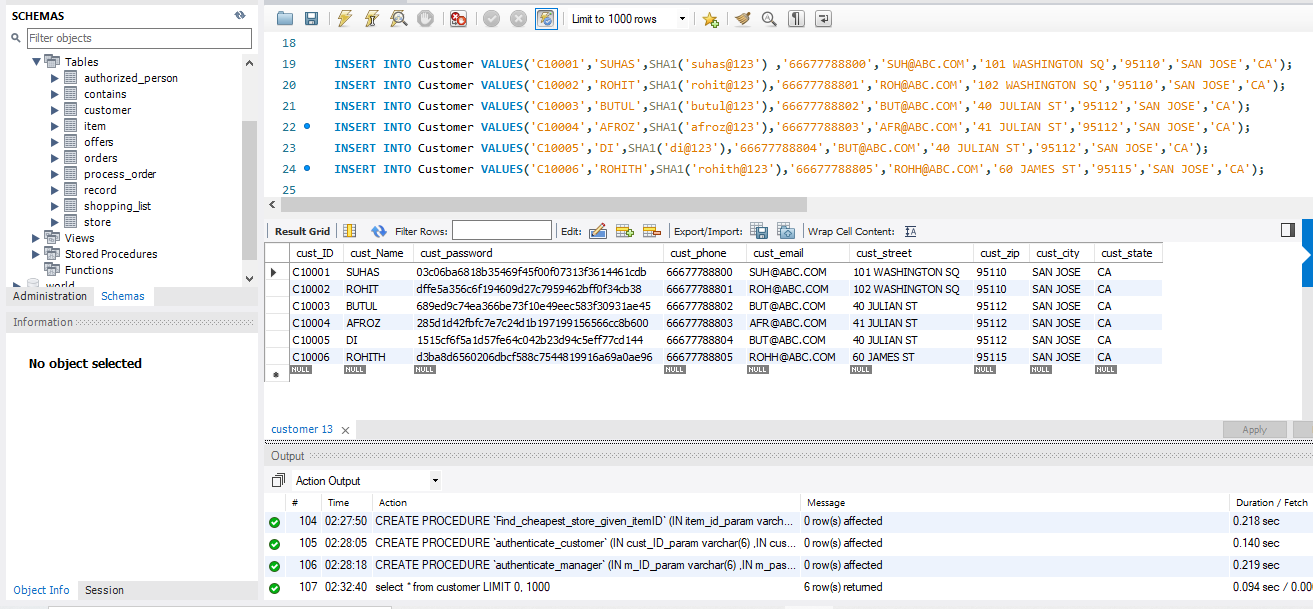
In the customer table without password.

****

In the customer table after the password in DB-side, it is encrypted.

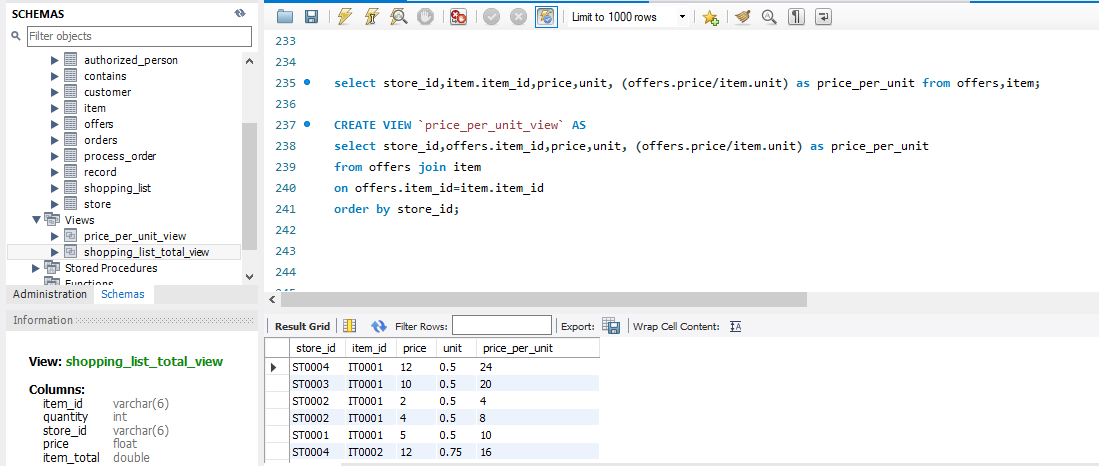
****

Screenshot after adding the column as cust\_password in the customer table and that encrypted password is stored in the customer table in the below screenshot.

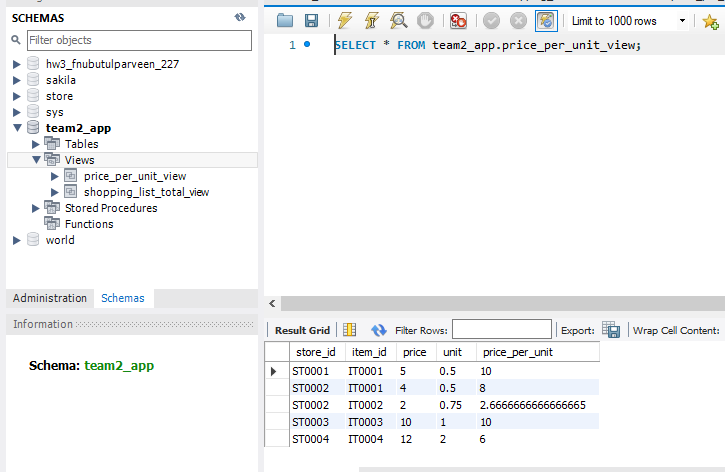
****

**Stored procedures and views:**

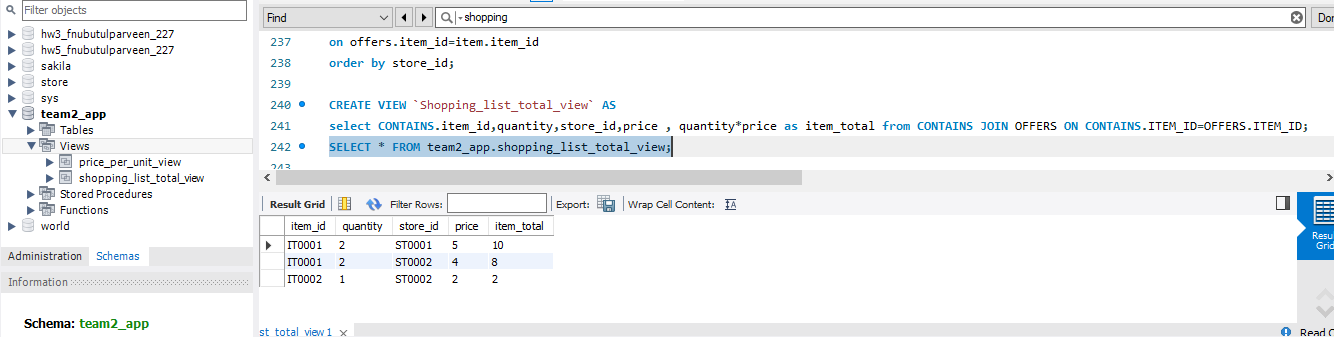
**price\_per\_unit\_view:** Based on store\_ID and Item\_ID, price per unit is calculated based on offers, the price\_ per\_unit is displayed.



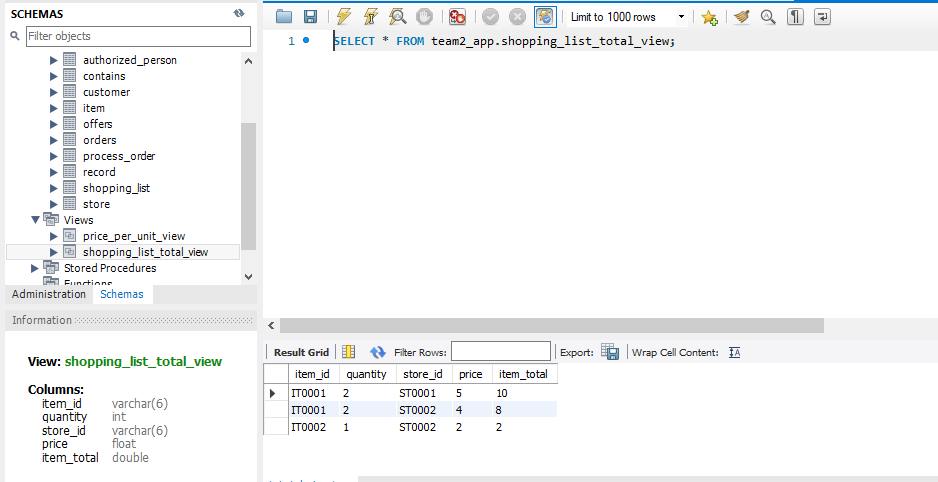
After view, the price\_per\_unit\_view can be displayed based on price and unit after calculation.

****

**shopping\_list\_total\_view:** In Shopping\_list\_view, it selects the item\_id, quantity,store\_id, price,item\_total after selecting the items based on offers for items it is shown below.

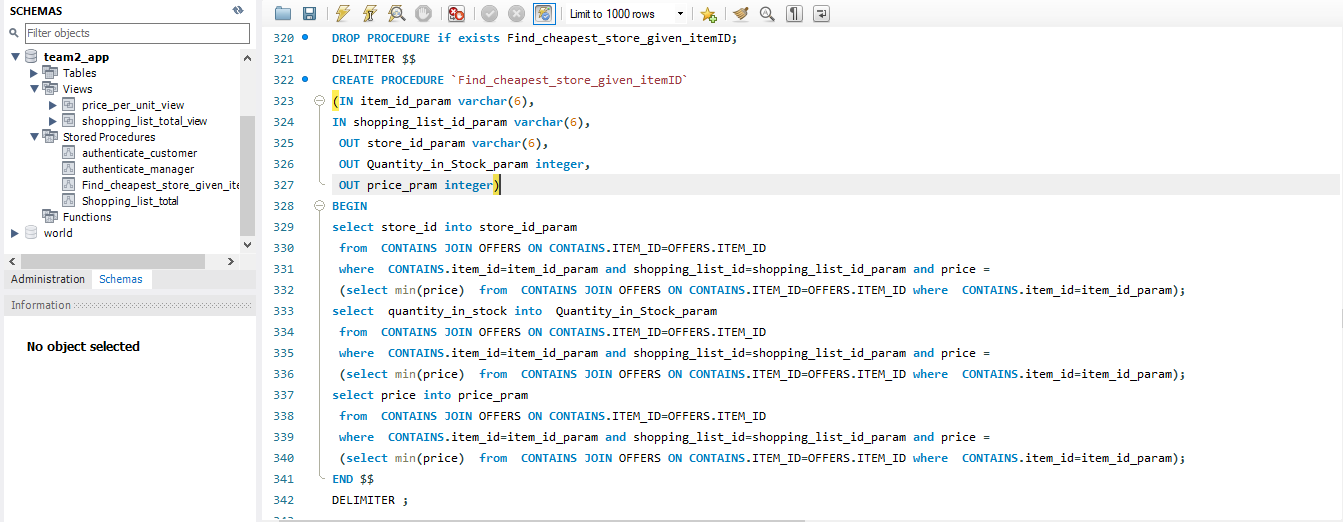
****

After executing the shopping\_list\_view is displayed.

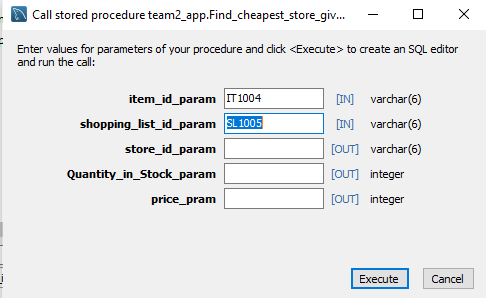
****

**Stored Procedure:**

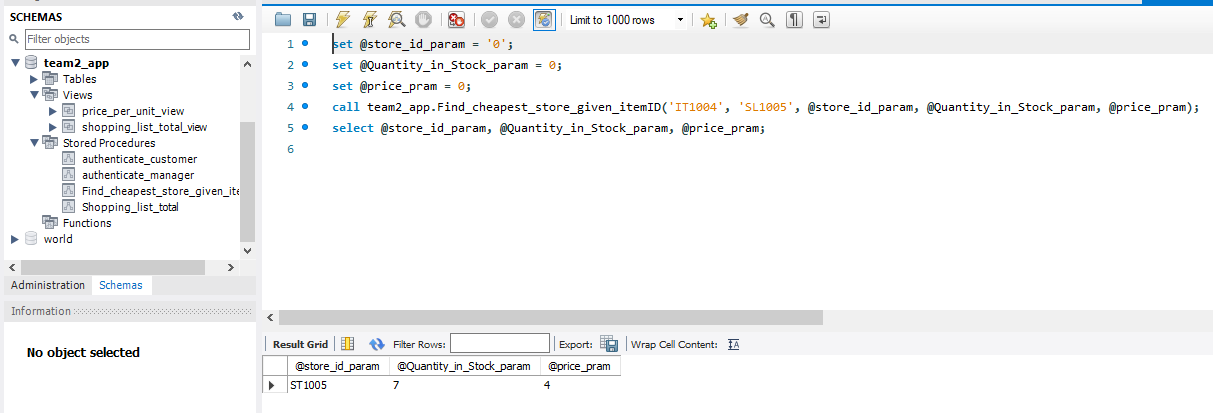
**Find\_cheapest\_store\_given\_itemID:** It will find the cheapest store based on price and item among all stores. Based on the availability of the item in-store and contains, offers for that particular item related to price in the Grocery store.

****

Enter Item\_id to check the cheapest price in-store, It basically checks the based on Item\_ID\_param and shopping\_list\_id\_param which store is offering the cheapest price with store\_ID,Quantity\_in\_Stock\_param and price\_param.

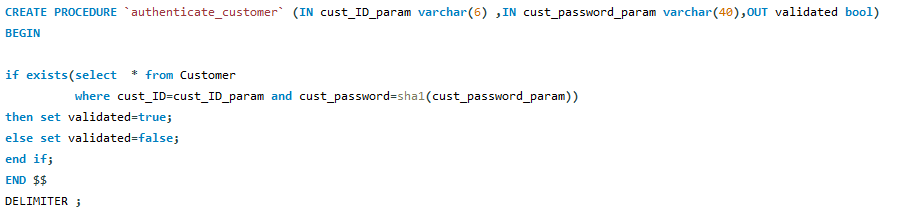
****

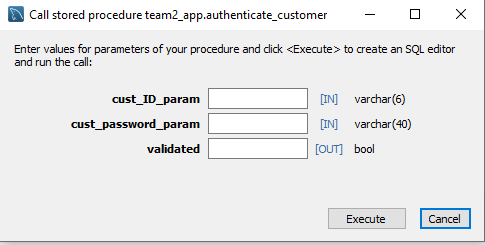
The inputs are item\_id\_param,store\_id & outputs are store\_id\_param, Quantity\_in\_Stock\_param, and price\_param. This procedure will find the cheapest store for the given item.

****

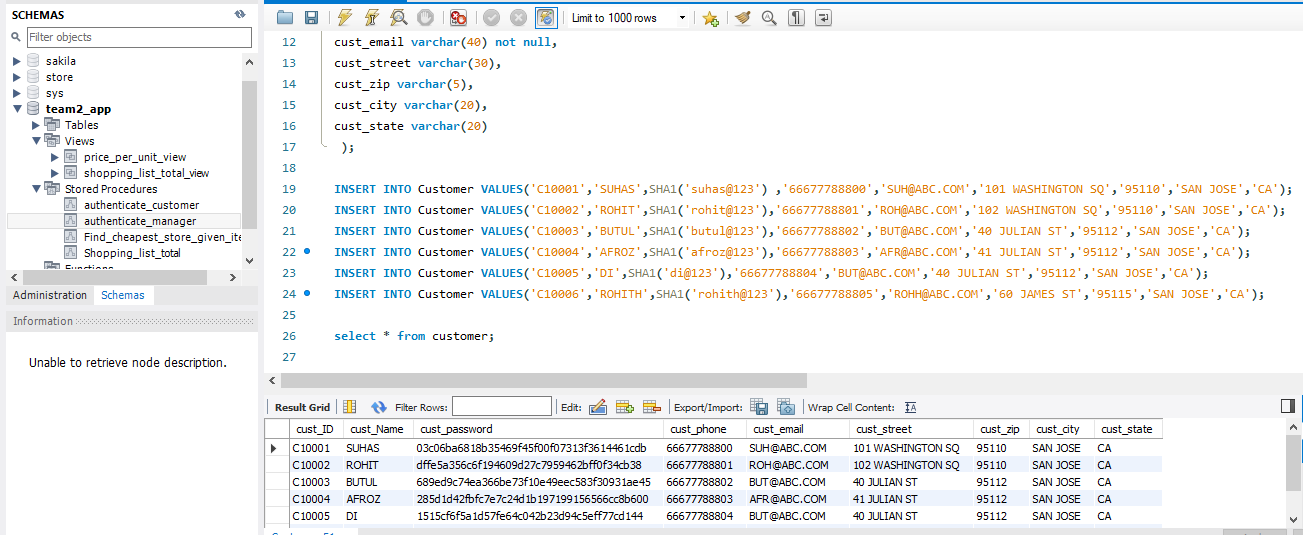
**Authenticate\_customer:**

Customer is a user for Grocery price application to place an order or select items based on offers or can see a list of items in store, Customer needs to register the Grocery price application after registering the user Cust\_ID and password are used for login the application and that password is stored in Database side.

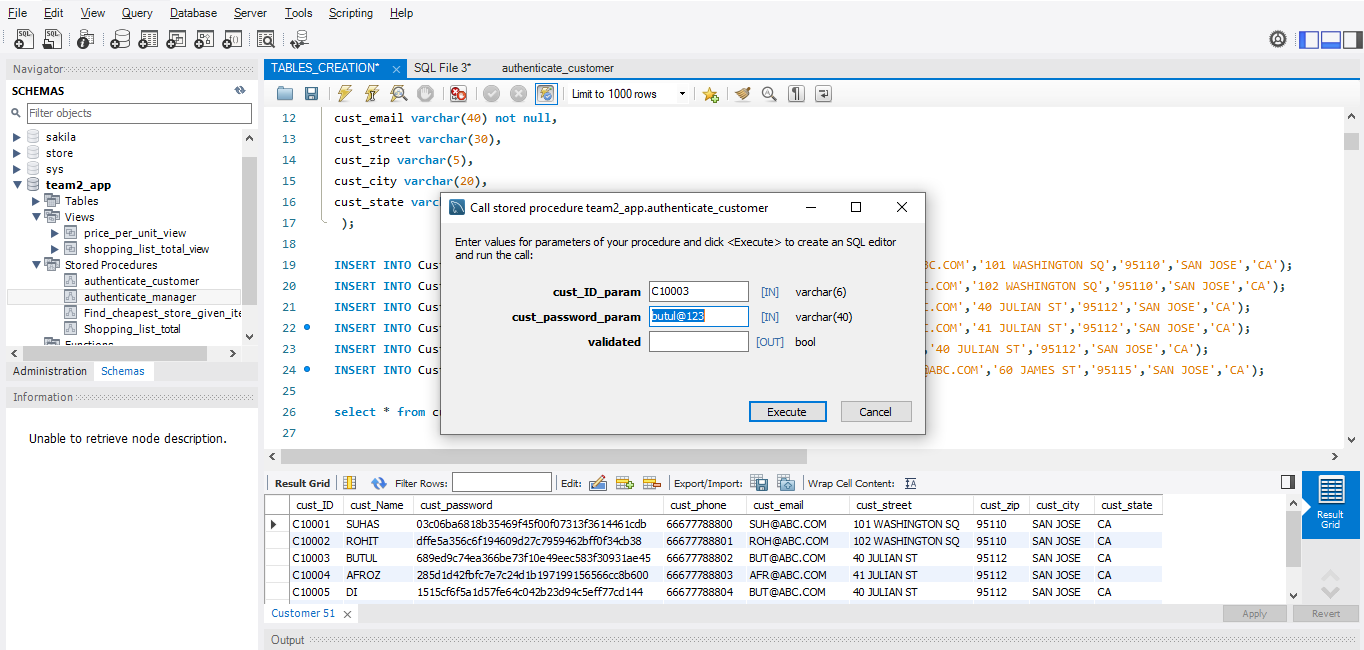
****

****

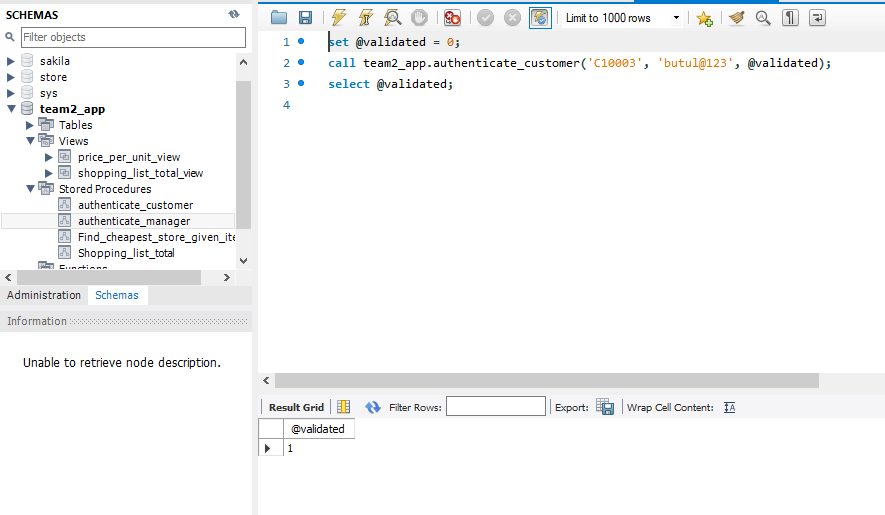
The inputs are customer ID and customer password and output is validated when both the inputs are correct, it will validate and log in the customer. The password of the customer stored in the database is encrypted in the table. The customer is the only one who gets to know the password because the customer created the password and that will be encrypted in the customer table afterward.

****

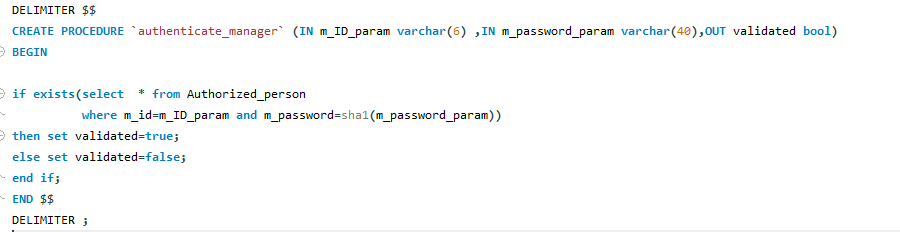
The encrypted password is stored in the table as above**.**

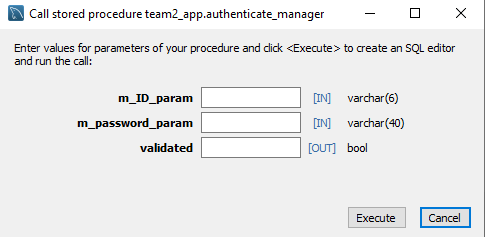
****

After executing this, it will check and validate the customer id and password it shows the details.

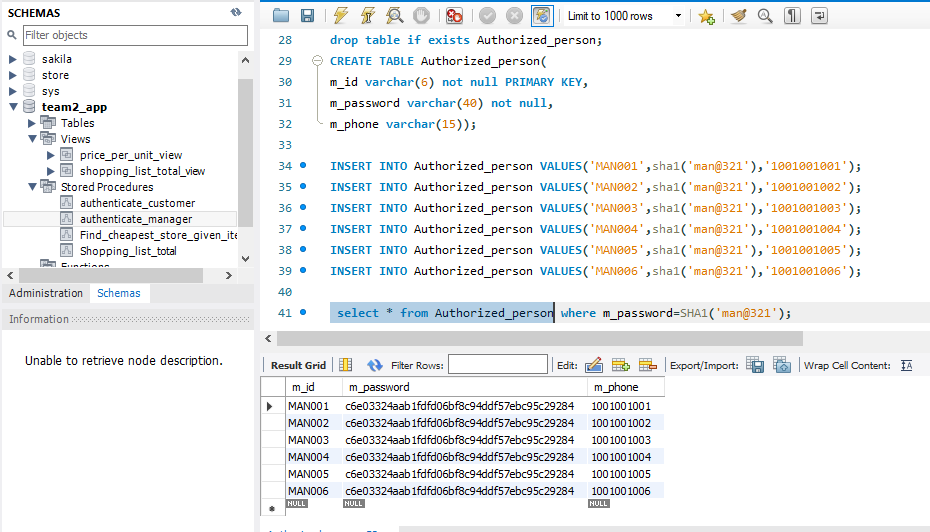
**Authenticate\_manager:**

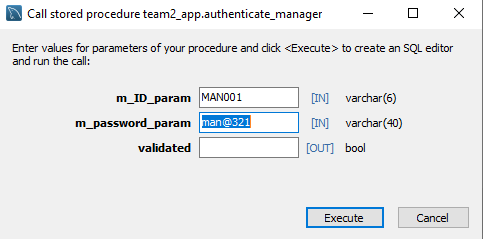
The manager will have an ID and password for logging into the application. This stored procedure will verify the password and authenticate the login activity for the manager. It will take the username and encrypt the password before storing it in the table while registering. While logging in, the password entered will be encrypted and compared to the password entered while registration. If both the values match the activity will be authenticated.

****

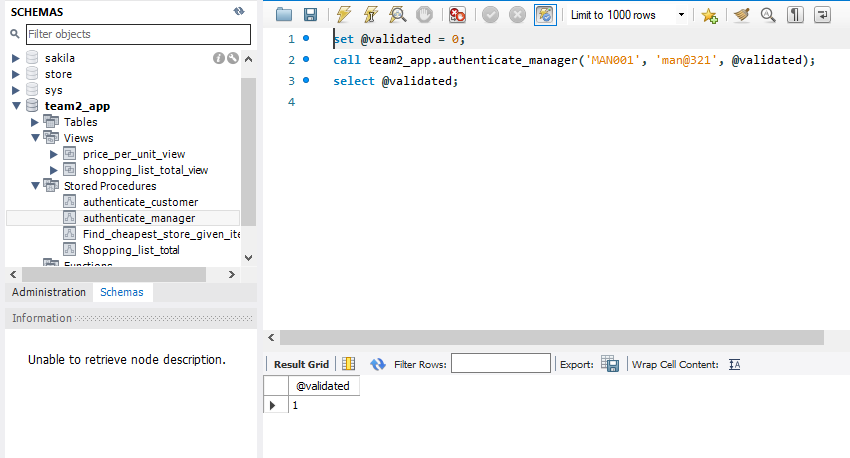
****

The inputs are the manager ID and manager password after entering it will validate the details. The manager only knows the password, the encrypted password is stored in the table below.

****

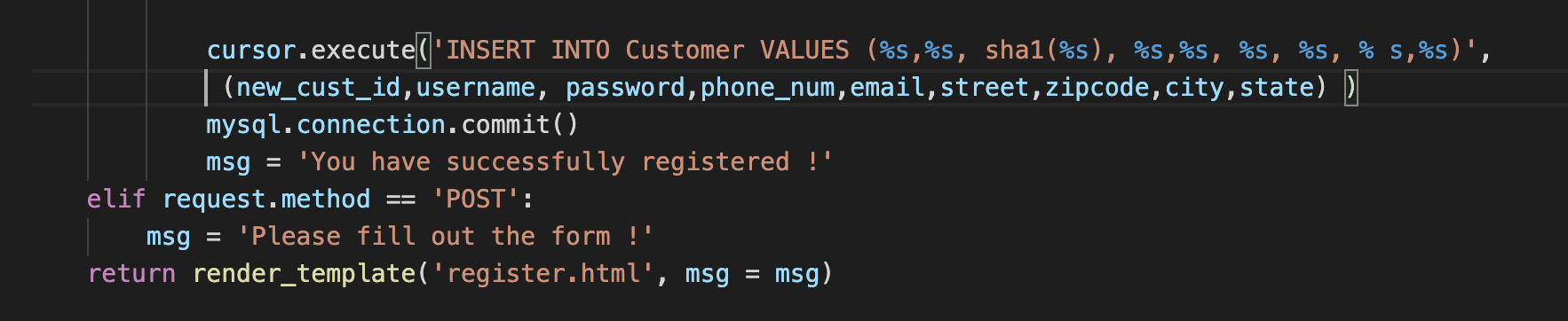
****

Click on execute to verify the details of manager ID and manager password for the grocery store**.**

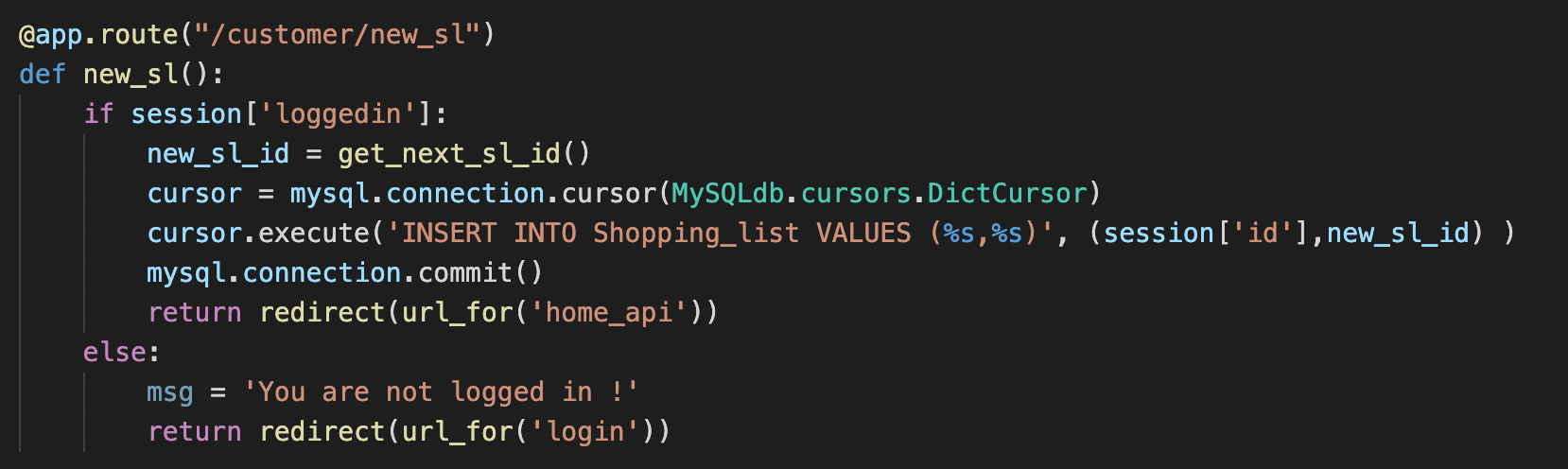
****

**Explicit multi-SQL-statement DB transactions from Flask backend:**

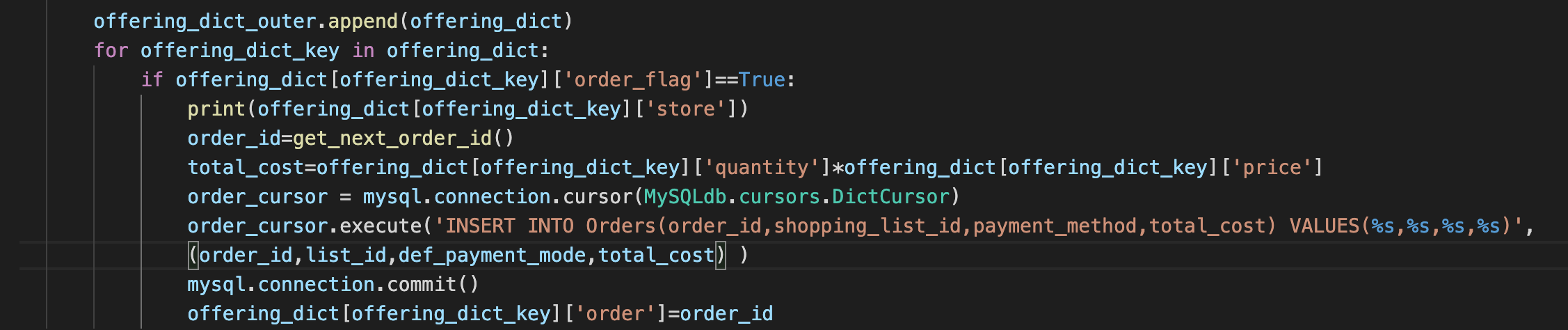
1. **sign-up new customer:**If customer sign-up the grocery price comparison application the customer table is updated.



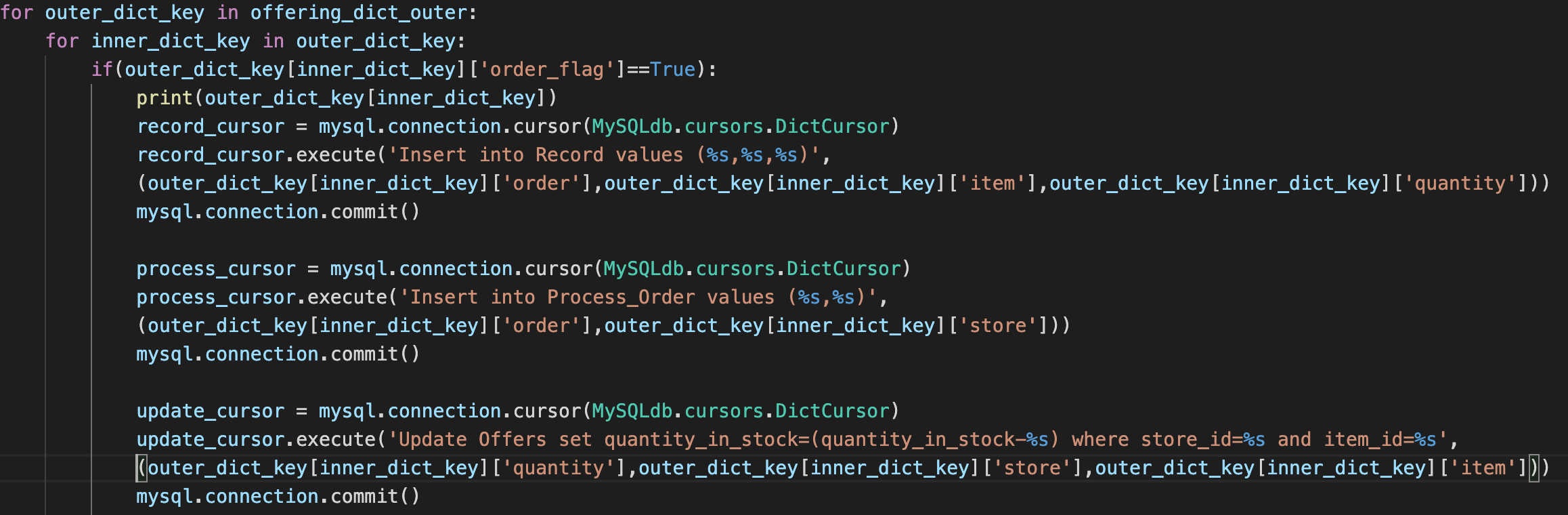
1. **Create a new shopping list:** Added the logic in the backend program to create new shopping lists for customers.



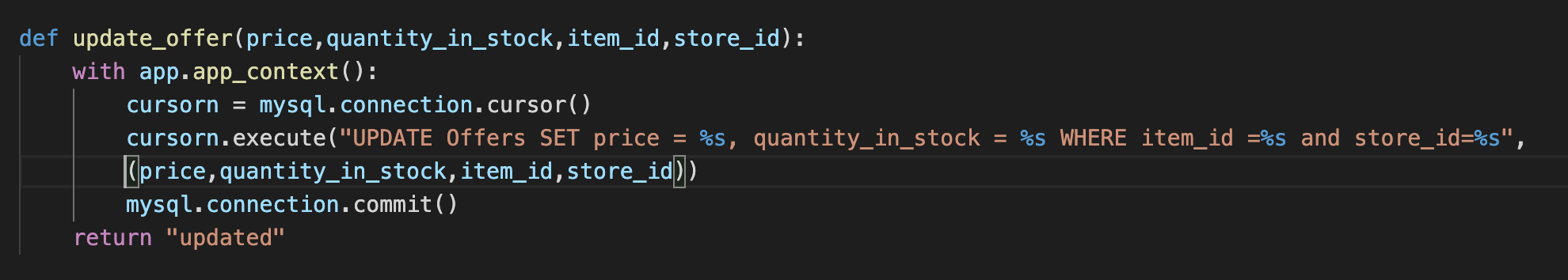
1. **Compare and split orders:** If items added in the shopping list are not in stock in one store, customers can place the multiple orders from different stores based on prices in store in increasing order. Added this logic in the backend to create multiple orders.

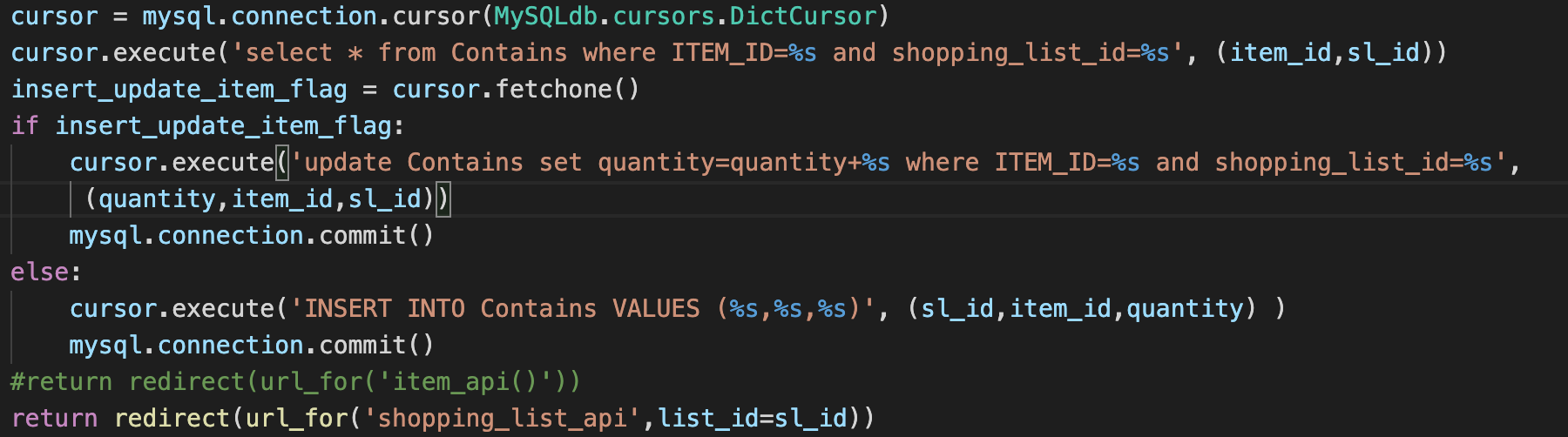


1. **Reduce the inventory stocks:** Once an order is placed the inventory stocks must be reduced. Implemented this logic in the backend using SQL queries and updating record and process tables.



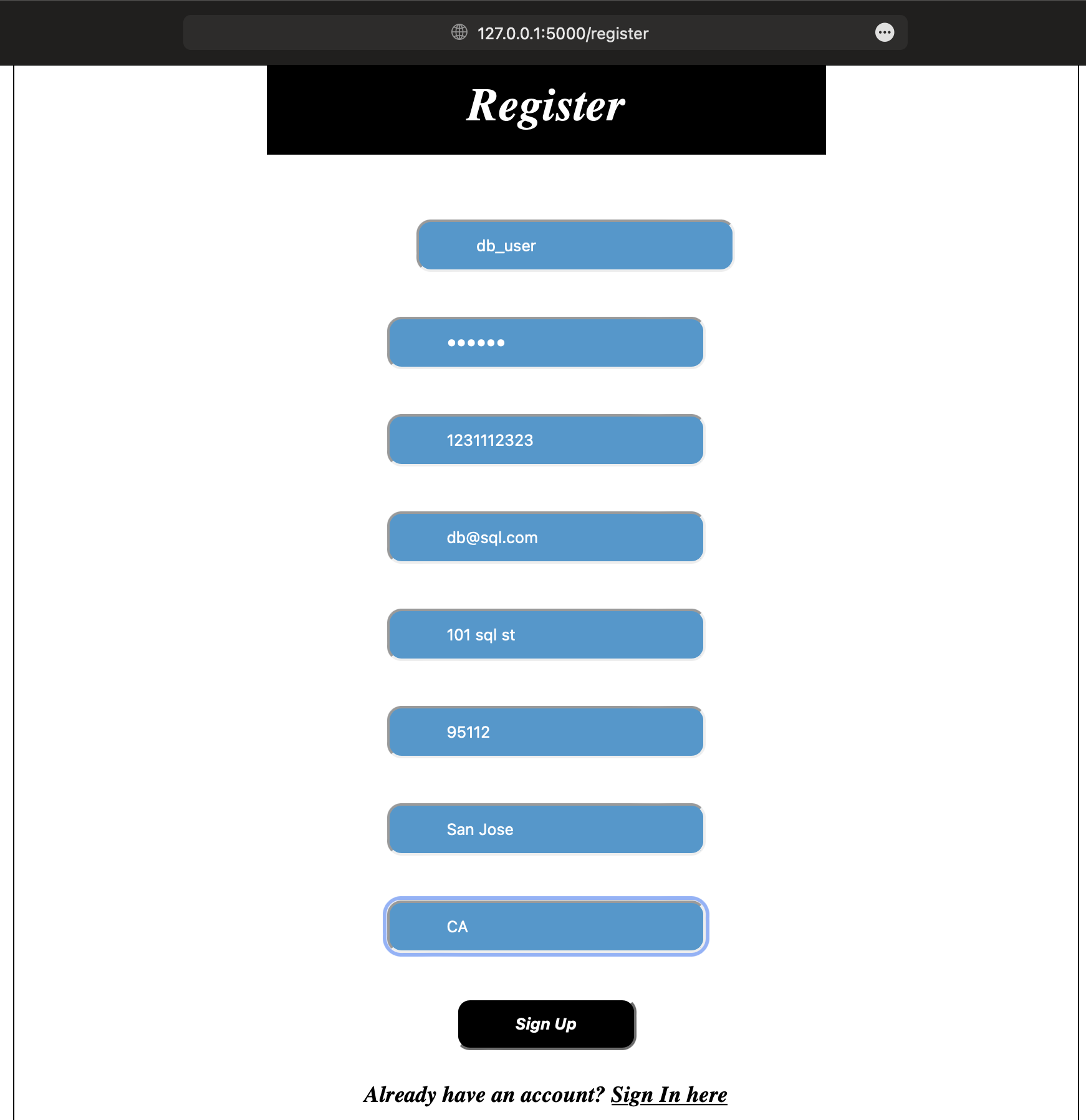
1. **Update inventory table:** A manager can update the inventory table and increase or decrease the item quantity, price and update the offers table accordingly.



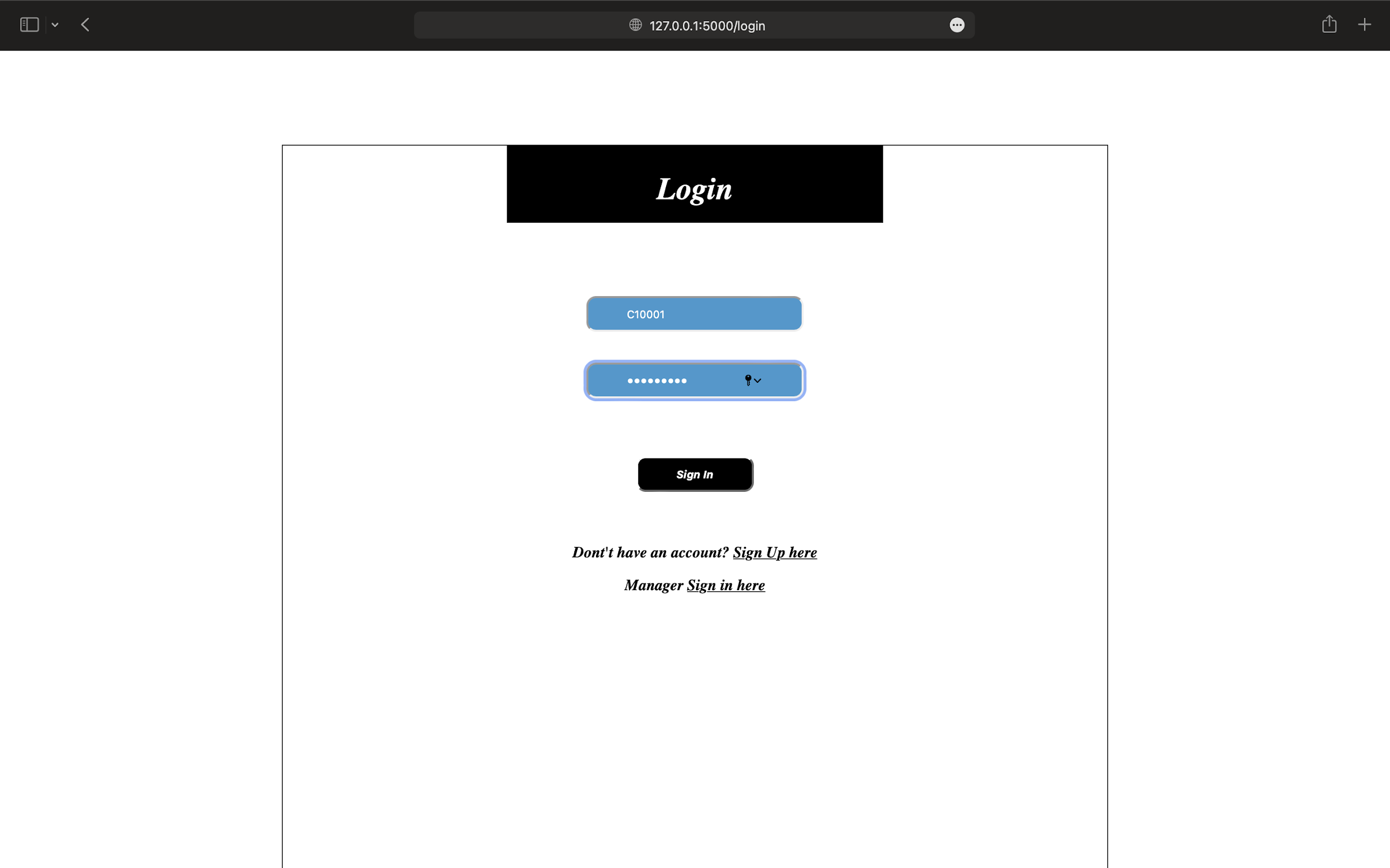
1. **Update shopping list:**  When a customer modifies the shopping list the shopping list table must be updated.  
   

**Final design of DB apps portion:**

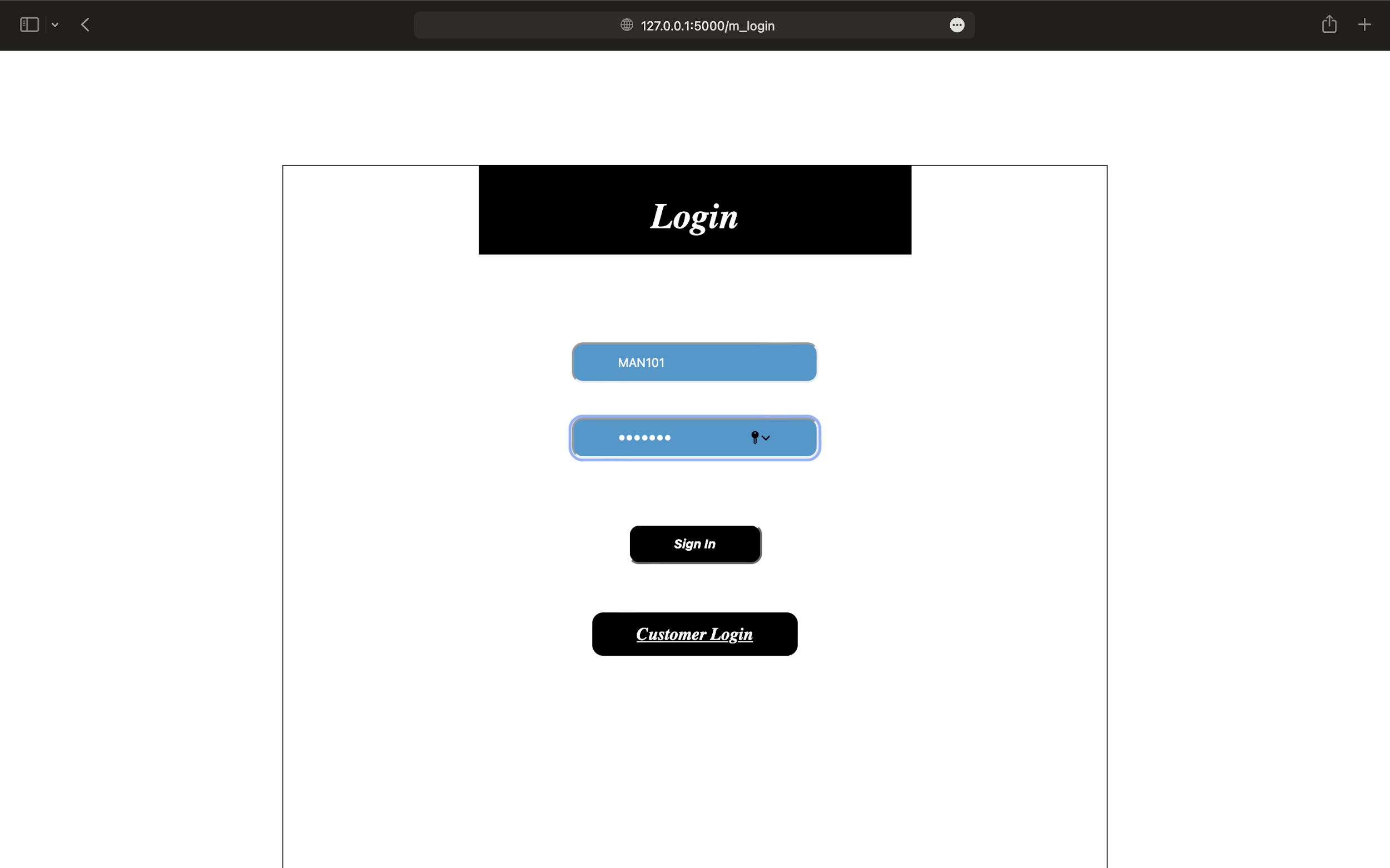
**Customer Sign-up page:** Customers can register their account by signing up in the Grocery Price Comparison Application.For shopping in this application customers need to register and create their account.If they have registered and created an account, they can sign up in the Grocery price comparison application.

****

**Customer Login page:** Once a customer is registered, the customer can log in by providing the registered credentials. The password is SHA1 format, which is encrypted in DB.



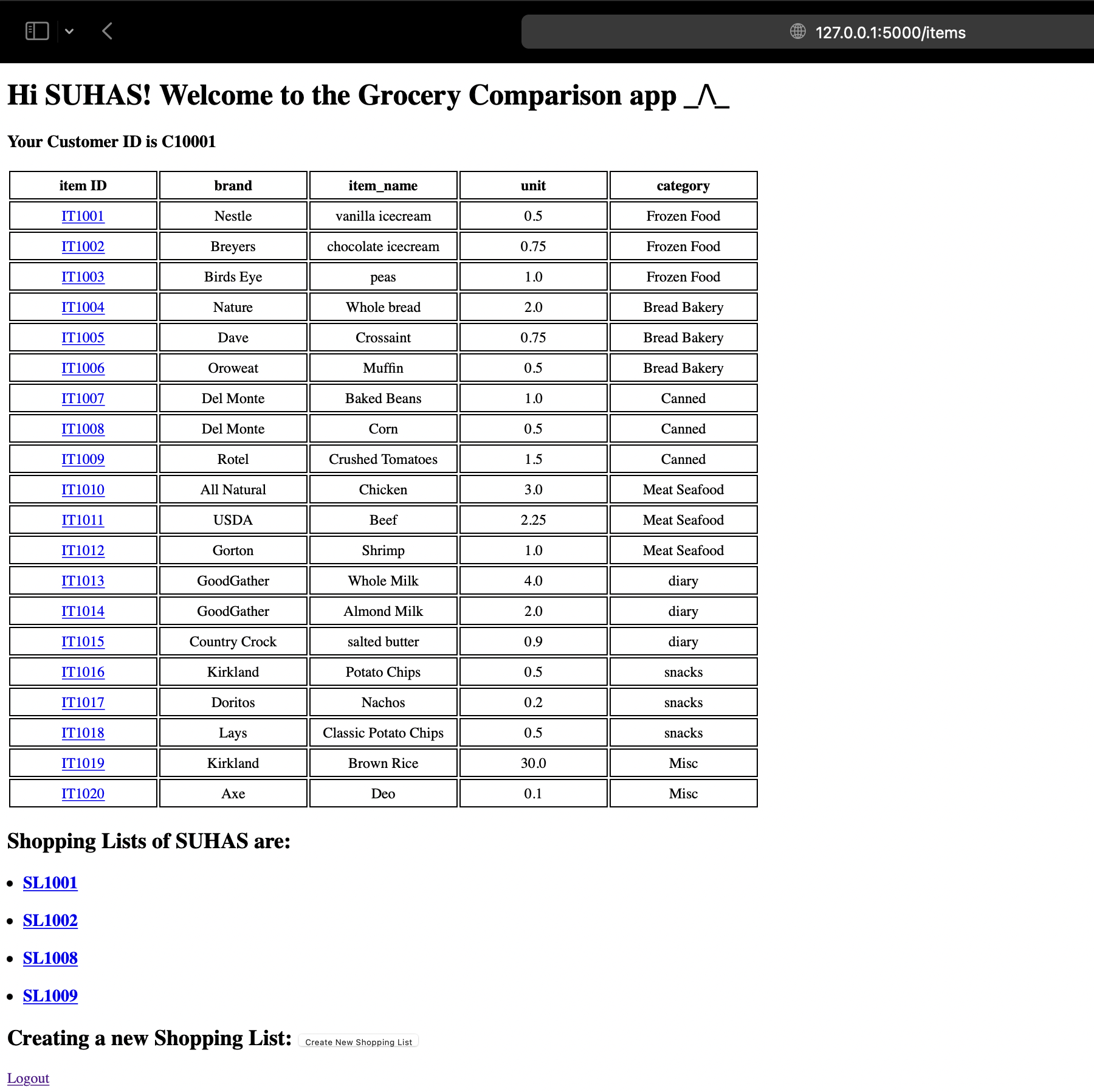
**Manager Login page:** A manager can login by putting the registered credentials and can change the store inventory data.



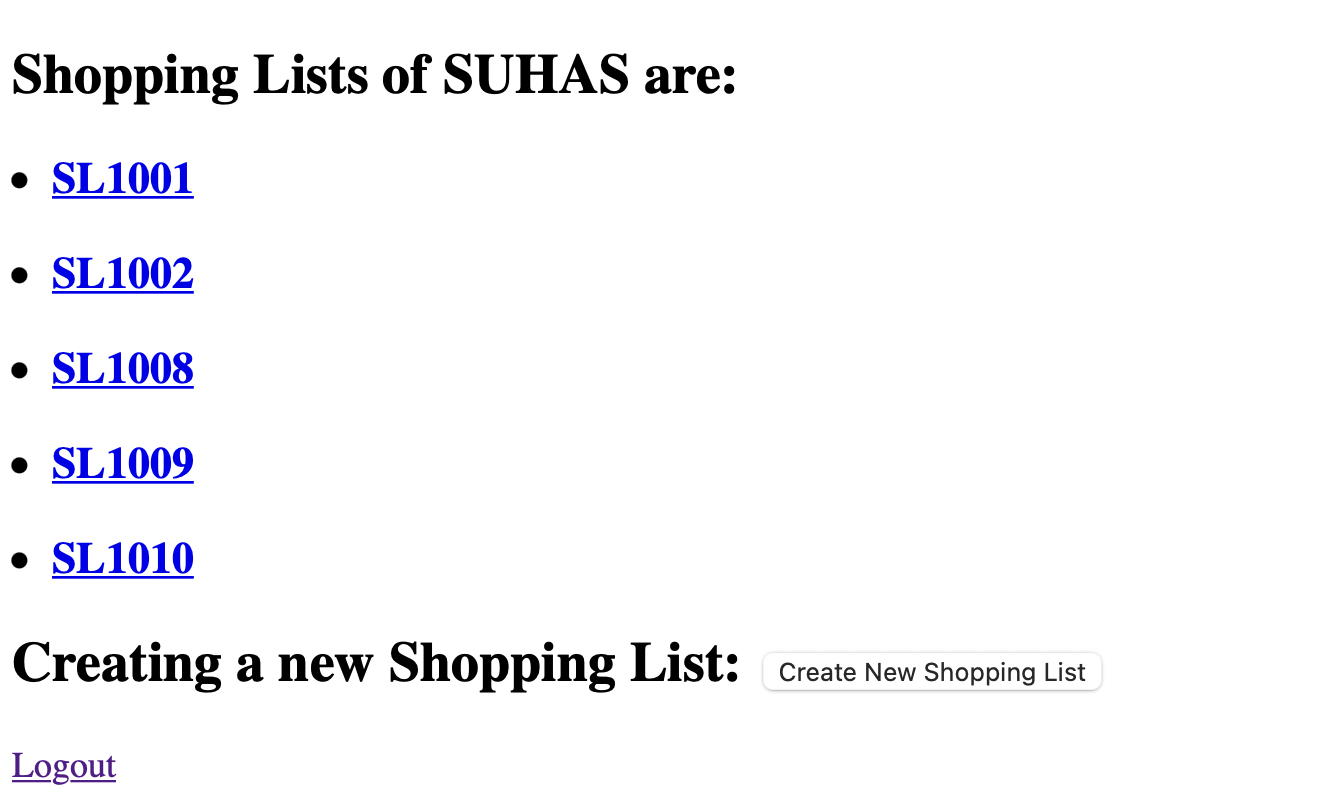
Based on store\_id and item\_id the price and quantity\_in\_stock is displayed, if there is any update in stock, so manager can update the quantity\_in\_stock and price and after updating manager can logout from the application.

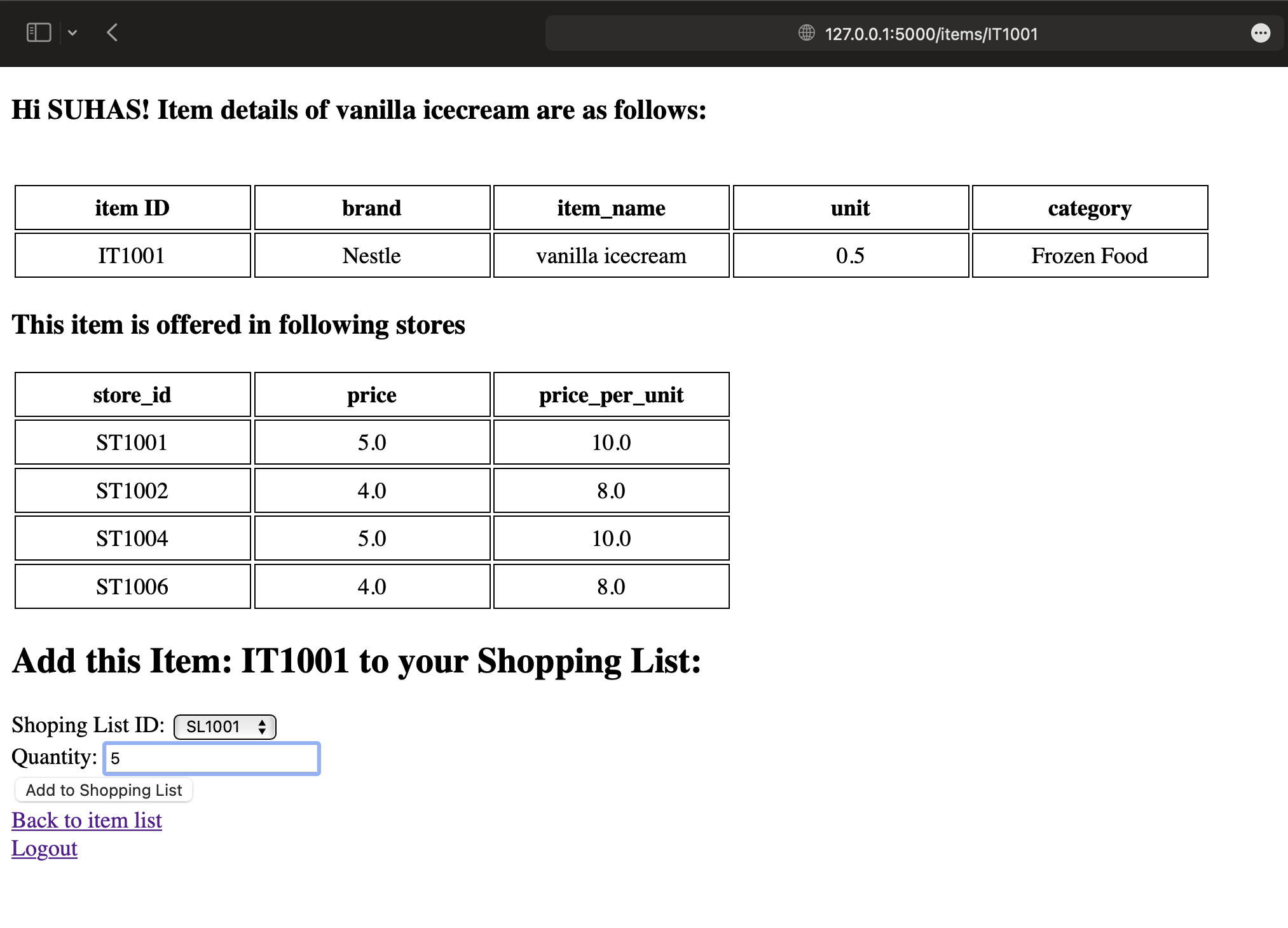


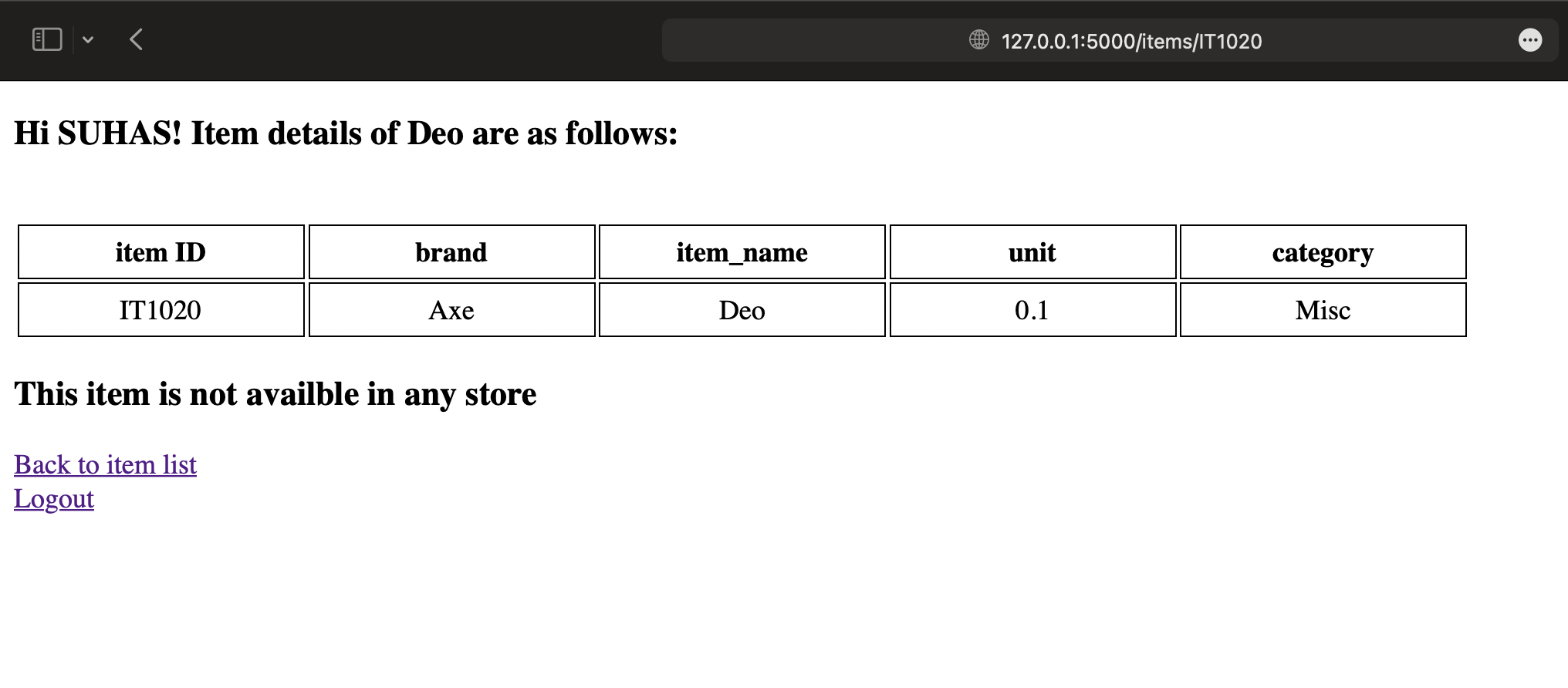
**Item** **Home** **Page:** After logging in, customers can see all the shopping list created and list of all the items available. Customers can add and modify the shopping list.



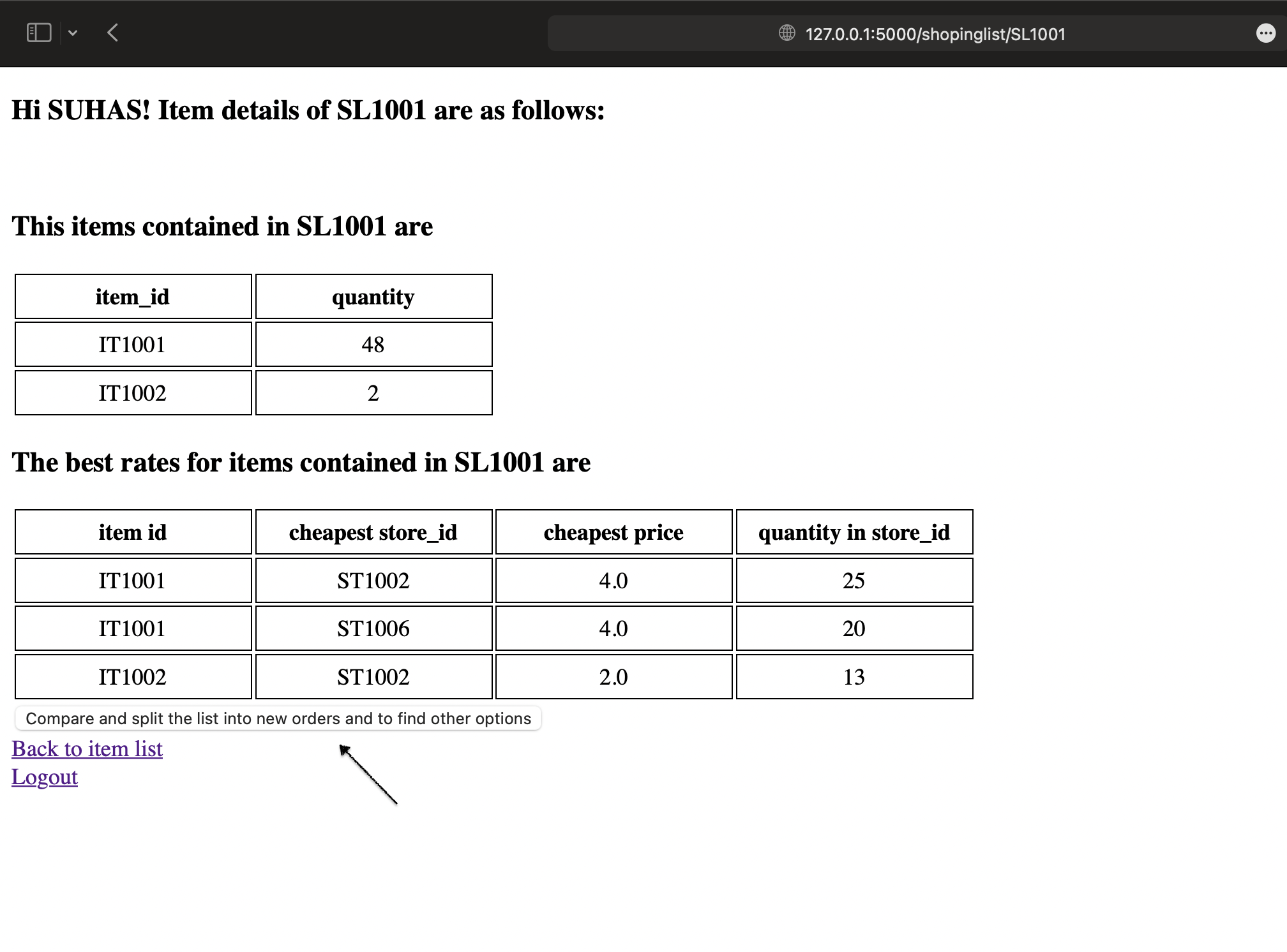
Shopping lists of customer Suhas are created with five shopping lists and he can create new shopping list also.

****

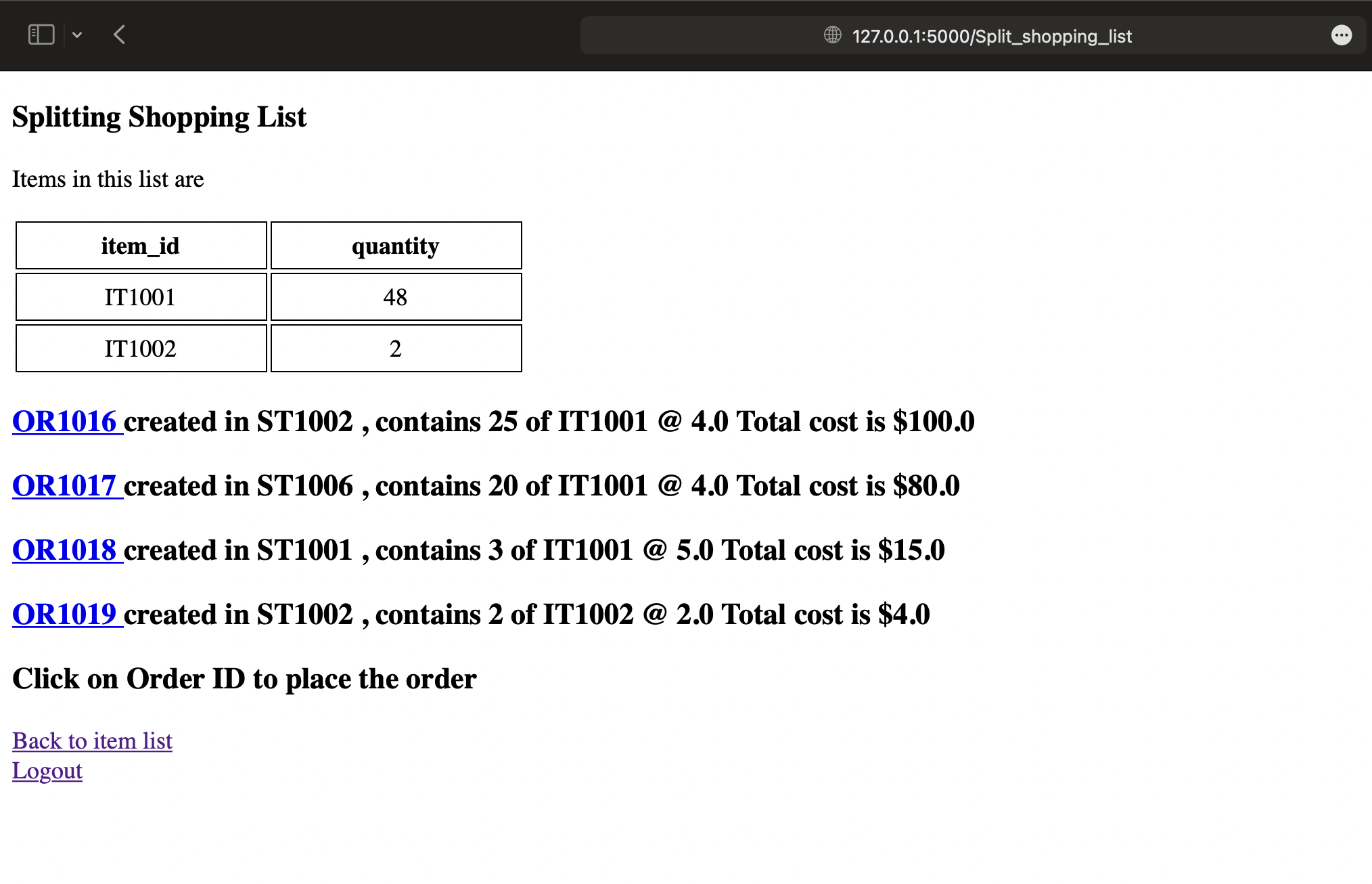
Customer can see the Item ID and brand,item\_name,unit and category based on his preference he can select item and that item is compared with different stores in terms of price and price\_per\_unit.It compares and display the item based upon price.Based on cheapest price the customer can add the item from which store is offering the cheapest price for particular item.Select the shopping\_list ID and quantity to add the item in shopping list and can check the item and quantity 5 it is added to shopping list or not by cross verifying to back\_to\_item\_list.****If an item is not offered in any store the customer will not be able to add the item in the shopping list.

****

**Compare and split orders:** If items added in the shopping list are not in stock in one store, customers can place the multiple orders in different stores based on prices in store in increasing order. This will create multiple orders across the stores having enough stocks available for the item.



Customers can select the items from different stores and quantity based on availability in stores the shopping list is compared and splitted into multiple orders.



**References:**

<https://docs.oracle.com/database/121/COMSC/diagrams.htm#COMSC00016>

<https://www.guru99.com/database-normalization.html>

<https://www.diagrams.net/blog/entity-relationship-tables>

<https://www.w3schools.com/sql/sql_stored_procedures.asp>

<https://www.geeksforgeeks.org/passwords-and-cryptographic-hash-function/>

<https://stackoverflow.com/questions/5693986/mysql-transaction-insert-and-update>