School of Engineering and Applied Science, Ahmedabad University

BTech ICT Sem-4

Course: Database Management System

ONLINE FOOD DELIVERY SYSTEM

Muskan Matwani ---1741027

Naishi Shah ---1741033

Yesha Shastri ---1741035

Devshree Patel ---1741075

**Description of the Project**

The Online Food Delivery System(OFDS) is a medium for users to order food from a variety of restaurants, cafes and food outlets using one platform and with great ease. This system takes basic input from the users to facilitate its functionalities and to let the users order food with ease.

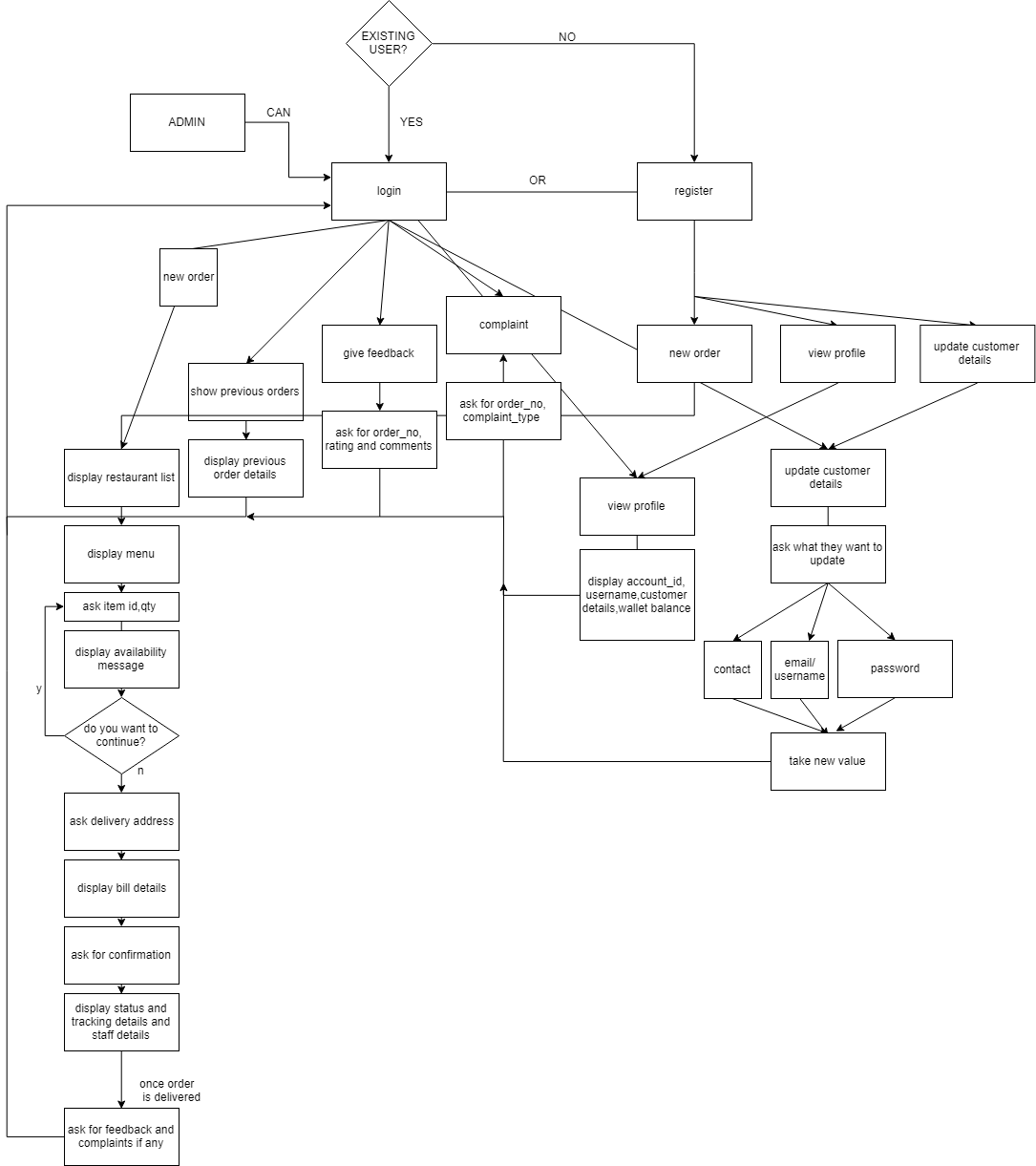
With this system, the users can order different types of food items in varying quantities and track their order till delivery. The payment for all orders is done via wallet. The user must be logged in to his/her account to avail any functionality. All the operations are done using the username or ID of the customer.

The **OFDS** has many features, which are listed below:

1. **Membership:** There are 2 types of memberships, regular and premium.
   1. Regular Member:
      1. Can get delivery from all the restaurants just like a premium member
      2. Delivery fee based on the distance of the delivery address from the restaurant
      3. Can avail discounts after successfully placing n orders.
   2. Premium Member:
      1. Can get delivery from all food joints associated with the system
      2. No delivery fee on any order regardless of the distance between the delivery address and the food outlet
      3. A certain amount is deducted from premium wallets as a fee.
2. **Modes of access:** There are 2 modes of access,namely, administrator and customer. Both hold accounts but have different privileges and rights.
   1. Administrator:
      1. Can access order details and produce reports for a restaurant’s performance, the number of orders it has received till date and the top n most active customers that the system hosts.
      2. Cannot access a customer’s personal profile and details

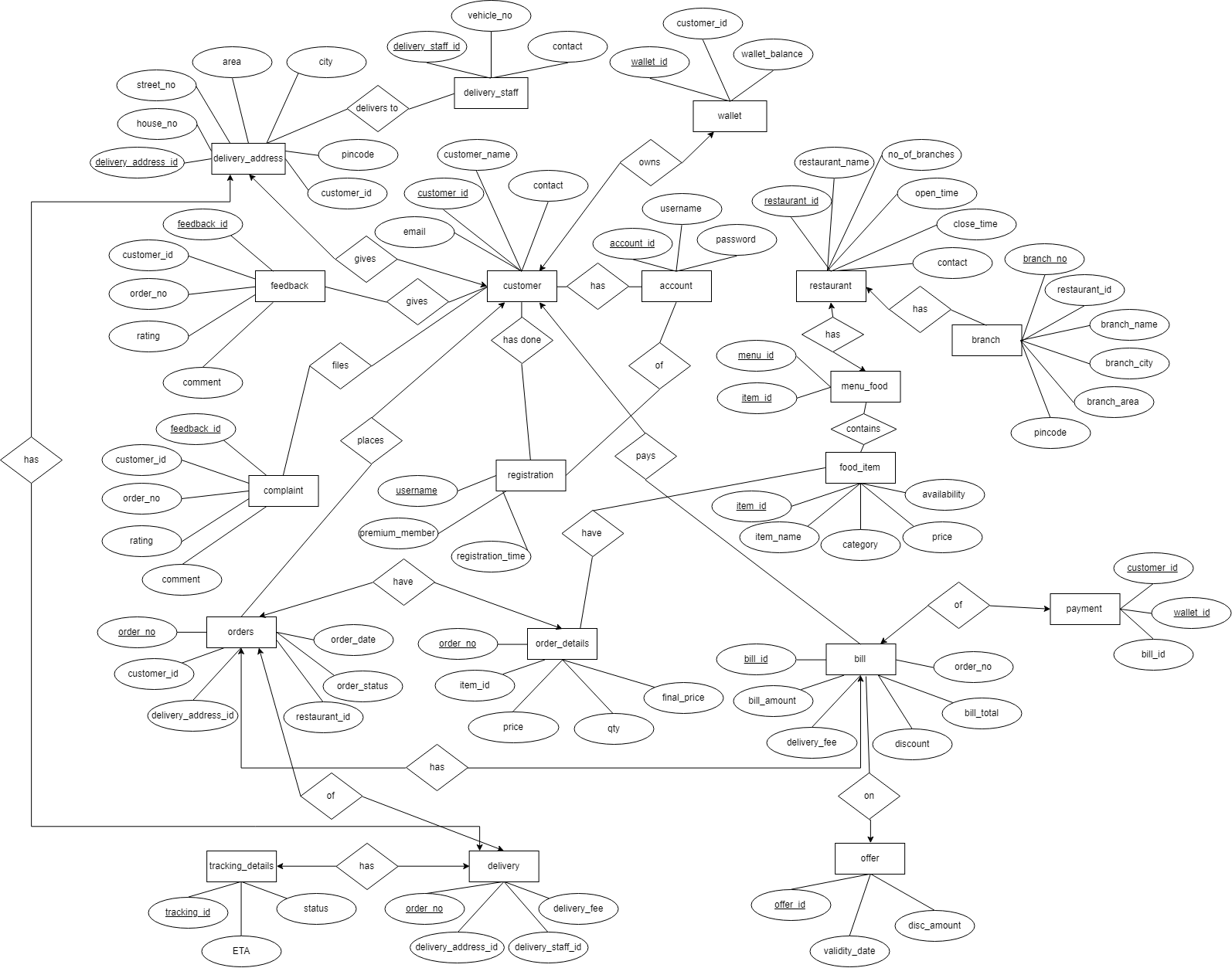
* 1. Customer:
     1. Can access his/her own profiles, order details and can update his/her own profiles
     2. Has no access to meta-level data

1. **Customer privileges:** Customers can do the following tasks:
   1. Place a new order:
      1. All customers can place any number of orders from any restaurants in the system.
      2. They have to select a restaurant, choose items to be ordered with their respective quantities and provide the delivery address, then he/she can view the bill details and confirm the order.After this, the necessary details are displayed to the customer
   2. Feedbacks and complaints
      1. The customer can give feedback for specific orders and also file complaints if any.
   3. Previous orders
      1. Customers can view their order history
2. **New Registrations:** New customers can register by providing appropriate details for their account and they can then place order,if an already an old member then can view previous orders. Registrations are also open for administrators.
3. **Flow of the system:** The OFDS provides its services to its members only. Thus, you must login, if you have an account, or register to the system by providing imperative details. You can login either as a customer or as an administrator. Once logged in, the customer can perform the above mentioned tasks and similarly for the admin.

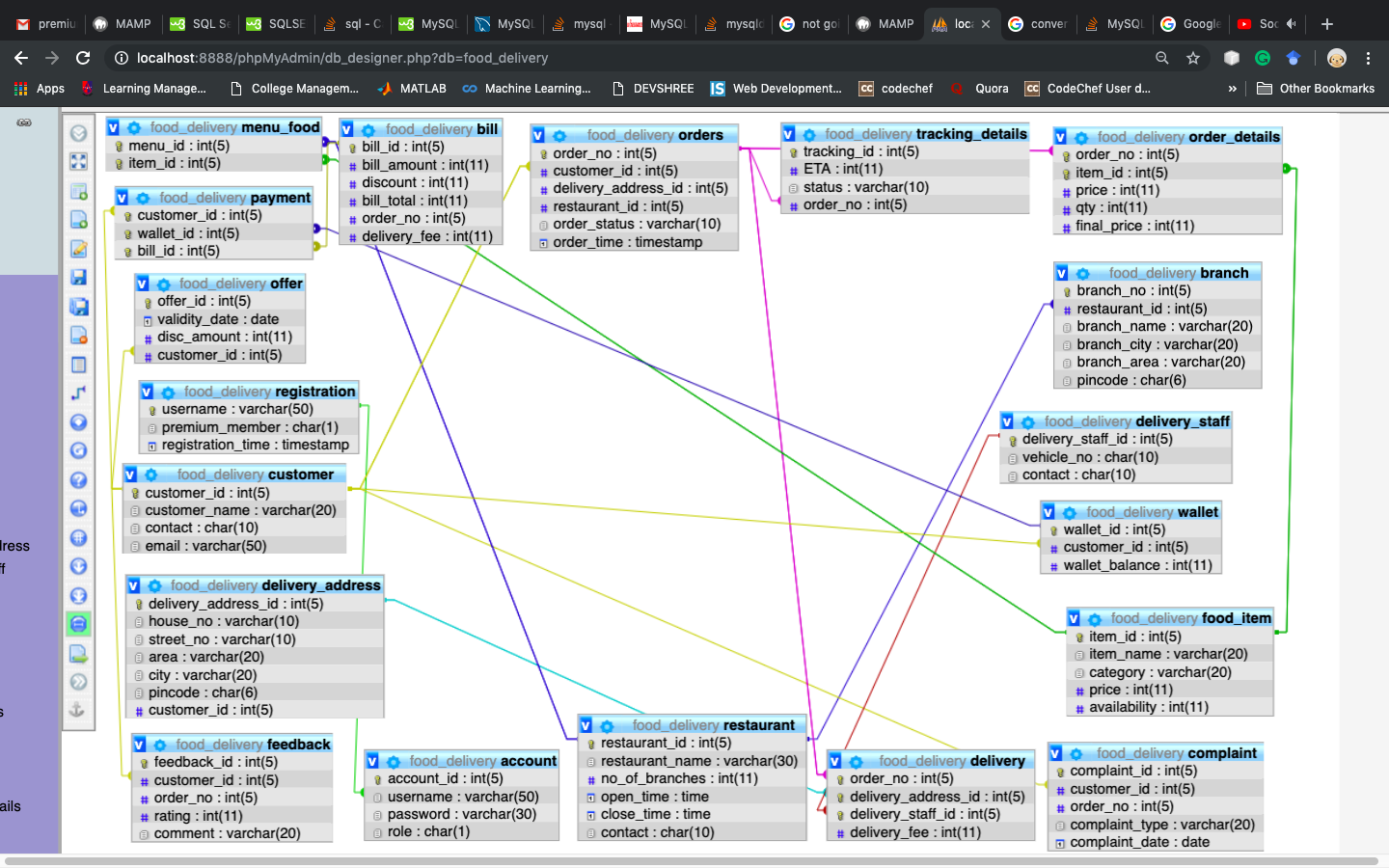


1. **Limitations:** This project has its own limitations;
   1. There is only one mode of payment i.e. wallet
   2. One restaurant can have only one menu
   3. ETA cannot be calculated corresponding to real-time data. Also, status cannot be updated based on ETA
2. **Assumptions**:
   1. The username will only be in the form of email.
   2. Some amount is assumed to be already present in the wallet at the time of registration (user need not add the balance).
   3. The wallet is assumed to be refilled automatically when there is not enough balance.
   4. Pincode is assumed as an area id wherein nearby areas will have small difference in the value of area id and vice versa.
   5. ETA is set to be minimum ten minutes and maximum 30 mins according to the distance from restaurant to the delivery address.
   6. Once placed, the order cannot be cancelled.
   7. Delivery fee will not exceed 30 rupees.
   8. Once a customer uses the given offer based on the n number of orders,the customer cannot be valid for the offer again after another n orders.
   9. Once a customer becomes a premium member there is no choice to revert the decision.
   10. Payment can be done only through the wallet. Payment through other modes like COD, credit card, net banking are not allowed.

**ER diagram**

****

**Relational Schema**

****

**Table Design (Data Dictionary)**

1. Table name: *customer*

Description: This table will store customer details.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| customer\_id | int | 5 | PK | Stores the customer ID |
| customer\_name | varchar | 20 | - | Stores the name of the customer |
| contact | char | 10 | - | Stores the contact number of the customer |
| email | varchar | 50 | - | Stores the email ID |

1. Table name: *orders*

Description: This table will store order details like the restaurant from which food has been ordered, the customer ID, order status and the order date

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| order\_no | int | 5 | PK | Stores the order ID |
| customer\_id | int | 5 | FK referred from customer | Stores the ID of the customer |
| delivery\_address\_id | int | 5 | - | Stores the delivery address ID of the customer |
| restaurant\_id | int | 5 | - | Stores the restaurant ID |
| order\_status | varchar | 10 | - | Shows the status of the order |
| order\_time | timestamp | - | - | Stores date and time of the placed order |

1. Table name: *account*

Description: This table stores the account details of customer like his/her username, password and ID

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| customer\_id | int | 5 | PK | Stores the ID of the customer |
| username | varchar | 50 | FK referred from registration | Stores the username i.e. the email of the customer |
| password | varchar | 30 | - | Stores the password for this account |

1. Table name: *delivery*

Description: This table will store delivery details of orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| order\_no | int | 5 | PK, FK referred from orders | Stores the order ID |
| delivery\_address\_id | int | 5 | PK, FK referred from delivery address | Stores the delivery address ID of the customer |
| delivery\_staff\_id | int | 5 | PK, FK referred from delivery staff | Stores the delivery staff ID of the delivery person |
| delivery\_fee | int | 11 | - | Stores the delivery fee on a delivery |

1. Table name: *registration*

Description: This table stores the registration details of customers like username, premium membership and the registration time

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| username | varchar | 50 | PK | Stores the username i.e. the email of the customer |
| premium\_member | char | 1 | - | Stores y if he/she is a premium member, n otherwise |
| registration\_time | timestamp | - |  | Stores the time of registration |

1. Table name: *restaurant*

Description: This table stores the restaurant details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| restaurant\_id | int | 5 | PK | Stores the restaurant ID |
| restaurant\_name | varchar | 30 | - | Stores the restaurant name |
| no\_of\_branches | int | 11 | - | Stores the number of branches of a particular restaurant |
| open\_time | time | - | - | Stores the opening time of the restaurant |
| close\_time | time | - | - | Stores the closing time of the restaurant |
| contact | char | 10 | - | Stores the contact number of the restaurant |

1. Table name: *bill*

Description: This table stores the bill details of orders

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| bill\_id | int | 5 | PK | Stores the bill ID |
| bill\_amount | int | 11 | - | Stores the bill amount |
| delivery\_fee | int | 11 | - | Stores the delivery fee on an order |
| discount | int | 11 | - | Stores the discount on an order |
| bill\_total | int | 11 | - | Stores the total amount of the bill after discount |
| order\_no | int | 5 | - | Stores the order number of the order corresponding to the bill |

1. Table name: *order\_details*

Description: This table stores the order details specifying the items of that order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| order\_no | int | 5 | PK, FK referred from orders | Stores the order number |
| item\_id | int | 5 | PK, FK referred from food\_item | Stores the item ID |
| price | float | - | - | Stores the price of the item to be ordered |
| qty | int | 11 | - | Stores the quantity of the item to be ordered |
| final\_price | float | - | - | Stores the final price of the item corresponding to the quantity |

1. Table name: *branch*

Description: This table stores the branch details of a restaurant

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| branch\_no | int | 5 | PK | Stores the branch number |
| restaurant\_id | int | 5 | FK referred from restaurant | Stores the restaurant ID |
| branch\_name | varchar | 20 | - | Stores the branch name |
| branch\_city | varchar | 20 | - | Stores the branch city |
| branch\_area | varchar | 20 | - | Stores the area where a particular branch is located |
| pincode | char | 6 |  | Stores the pincode of that branch |

1. Table name: *delivery\_address*

Description: This table stores the delivery address details for an order delivery

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| delivery\_address\_id | int | 5 | PK | Stores the delivery address ID |
| house\_no | varchar | 10 | - | Stores the house number |
| street\_no | varchar | 10 | - | Stores the street number |
| area | varchar | 20 | - | Stores the area of the address |
| city | varchar | 20 | - | Stores the city of the address |
| pincode | char | 6 | - | Stores the pincode of the address |
| customer\_id | int | 5 | - | Stores the customer ID whose address is being stored |

1. Table name:*delivery\_staff*

Description: This table stores the details of the delivery staff

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| delivery\_staff\_id | int | 5 | PK | Stores the ID of the delivery staff |
| vehicle\_no | char | 10 | - | Stores the registration number of the vehicle used |
| contact | char | 10 | - | Stores the contact number of the staff member |

1. Table name: *payment*

Description: This table stores the payment details corresponding to a particular bill

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| customer\_id | int | 5 | PK,FK referred from customer | Stores the customer ID of the customer making the payment |
| wallet\_id | int | 5 | PK, FK referred from wallet | Stores the wallet ID of the customer making the payment |
| bill\_id | int | 5 | PK, FK referred from bill | Stores the bill ID of an order |

1. Table name: *wallet*

Description: This table stores the wallet details corresponding to a particular customer account

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| wallet\_id | int | 5 | PK | Stores the wallet ID of the customer |
| customer\_id | int | 5 | FK referred from customer | Stores the customer ID of the customer |
| wallet\_balance | int | 11 | - | Stores the wallet balance of the respective wallet |

1. Table name: *food\_item*

Description: This table stores the food item details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| item\_id | int | 5 | PK | Stores the item ID |
| item\_name | varchar | 20 | - | Stores the item name |
| category | varchar | 20 | - | Stores the category of the item |
| price | int | 11 | - | Stores the price of the item |
| availability | int | 11 | - | Stores the availability of the item |

1. Table name: *menu\_food*

Description: This table stores the menu details corresponding to a particular restaurant

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| menu\_id | int | 5 | PK, FK referred from restaurant | Stores the menu ID |
| item\_id | int | 5 | PK, FK referred from food\_item | Stores the item ID |

1. Table name: *feedback*

Description: This table stores the feedback details

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| feedback\_id | int | 5 | PK | Stores the feedback ID |
| customer\_id | int | 5 | FK referred from customer | Stores the customer ID of the customer giving the feedback |
| order\_no | int | 5 | - | Stores the order number on which feedback is given |
| rating | int | 11 | - | Stores the rating for the feedback |
| comment | varchar | 20 | - | Stores the comments on the order |

1. Table name: *complaint*

Description: This table stores the complaints pertaining to a particular order

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| complaint\_id | int | 5 | PK | Stores the complaint ID |
| customer\_id | int | 5 | FK referred from customer | Stores the customer ID of the customer |
| order\_no | int | 5 | - | Stores the order number |
| complaint\_type | varchar | 20 | - | Stores the complaint type |
| complaint\_date | date | - | - | Stores the complaint date |

1. Table name: *tracking\_details*

Description: This table stores the wallet details corresponding to a particular customer account

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| tracking\_id | int | 5 | PK | Stores the tracking ID of the order |
| ETA | time | - | - | Stores the estimated time of arrival of the delivery |
| status | varchar | 10 | - | Stores the status of the order delivery |
| order\_no | int | 5 | FK referred from orders | Stores the order number corresponding to the tracking details |

1. Table name: *offer*

Description: This table stores the wallet details corresponding to a particular customer account

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Fieldname** | **Data type** | **Size** | **Constraint** | **Description** |
| offer\_id | int | 5 | PK | Stores the offer ID |
| validity\_date | date | - | - | Stores the validity date of the offer |
| disc\_amount | int | 11 | - | Stores the discount amount of the offer |
| customer\_id | int | 5 | FK referred from customer | Stores the customer ID of the customer who avails the offer |

**Procedures**

1. **Check validity of username:** It checks for the proper format of entered username for both user and admin.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `chk\_user`(IN `uname` VARCHAR(50), OUT `flag` INT)

BEGIN

DECLARE pos int;

DECLARE sub varchar(50);

DECLARE b int;

DECLARE a varchar(50);

DECLARE cur2 CURSOR FOR SELECT username1 FROM utable;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

If (POSITION("@" IN uname) >0 or POSITION("." IN uname) >0 or POSITION("\_" IN uname) >0) THEN

SET pos= POSITION("@" IN uname);

SET sub= SUBSTR(uname,pos+1);

OPEN cur2;

SET b = 0;

WHILE b = 0 DO

FETCH cur2 INTO a;

if (a=sub) THEN

set flag=1;

end if;

END WHILE;

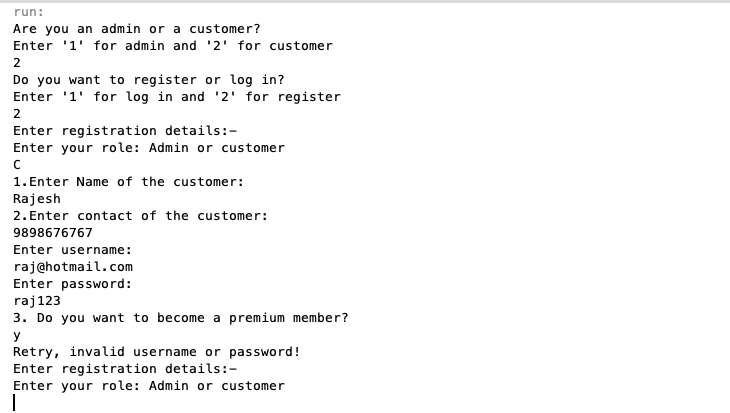
CLOSE cur2;

END if;

END$$

DELIMITER ;

**Screenshot**: displays invalid when done registration with invalid constraints of username



1. **Check validity of password:** It checks for the proper format of entered password for both user and admin.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `password\_check`(IN `pass` VARCHAR(30), OUT `flag` INT)

BEGIN

if(length(pass)>=8)

then

if(~((strcmp(upper(pass),pass)) and (strcmp(lower(pass),pass))))

then

if((pass like '%$%') or (pass like '%@%') or (pass like '%\_%') or (pass like '%&%') or (pass like '%\*%') or (pass like '%#%') or (pass like '%%%') or (pass like '%^%') or (pass like "%'%") or (pass like '%"%')) then

set flag = 1;

else

set flag = 0;

end if;

else

set flag = 0;

end if;

else

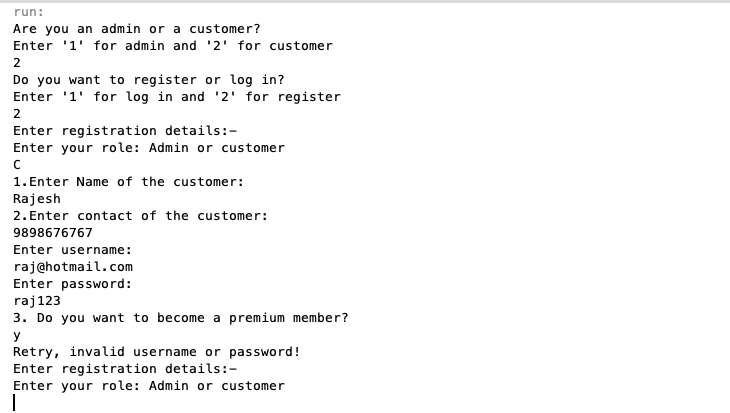
set flag = 0;

end if;

END$$

DELIMITER ;

**Screenshot:**  displays invalid when done registration with invalid constraints of password



1. **Registration:** As soon as a customer registers, the details are inserted into the registration,customer and account tables. Also, the initial wallet balance is set in the wallet table.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `register1`(IN `username` VARCHAR(50), IN `name` VARCHAR(20), IN `password1` VARCHAR(20), IN `phone` CHAR(10), IN `premium\_member1` CHAR(1), IN `role1` CHAR(1))

BEGIN

DECLARE id int(5);

INSERT INTO registration(username, premium\_member,registration\_time) VALUES (username, premium\_member1,CURRENT\_TIMESTAMP);

INSERT INTO customer(customer\_name, contact, email) VALUES (name, phone, username);

SET id = (select customer.customer\_id from customer where customer.email=username);

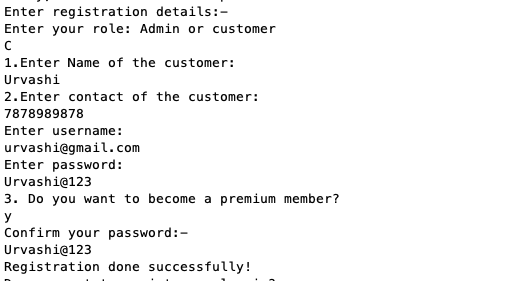
INSERT INTO wallet(customer\_id,wallet\_balance) values(id,100000);

INSERT INTO account(account.username, account.password, account.role) VALUES (username, password1, role1);

END$$

DELIMITER ;

**Screenshot:** Enter details into registration, account, customer and sets a minimum balance for wallet balance of that particular customer.



1. **User Login:** This procedure checks whether the entered username and password of user exist or not.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `user\_login`(IN `username1` VARCHAR(50), IN `pass` VARCHAR(30), OUT `flag` INT)

BEGIN

DECLARE b int;

DECLARE a varchar(50);

DECLARE c varchar(30);

DECLARE cur1 CURSOR FOR SELECT account.username,account.password FROM account where role='C';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

SET b = 0;

SET flag = 0;

OPEN cur1;

FETCH cur1 INTO a,c;

WHILE b = 0 DO

if (a=username1) and (c=pass) then

set flag = 1;

end if;

FETCH cur1 INTO a,c;

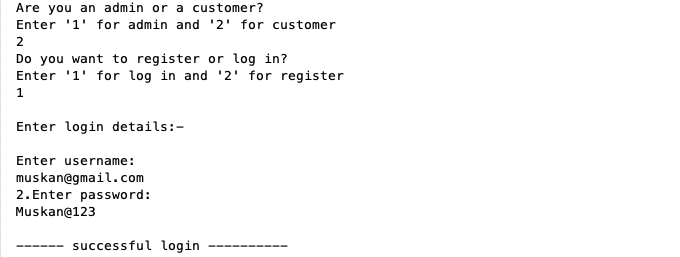
END WHILE;

CLOSE cur1;

END$$

DELIMITER ;

**Screenshot:** It checks whether the username and password matches with the registered details and lets the user log in if it is correct.



1. **Admin Login:** Checks whether the username and password of admin exist or not.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `admin\_login`(IN `username1` VARCHAR(50), IN `pass` VARCHAR(30), OUT `flag` INT)

BEGIN

DECLARE b int;

DECLARE a varchar(50);

DECLARE c varchar(30);

DECLARE cur1 CURSOR FOR SELECT account.username,account.password FROM account where role='A';

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

SET b = 0;

SET flag = 0;

OPEN cur1;

FETCH cur1 INTO a,c;

WHILE b = 0 DO

if (a=username1) and (c=pass) then

set flag = 1;

end if;

FETCH cur1 INTO a,c;

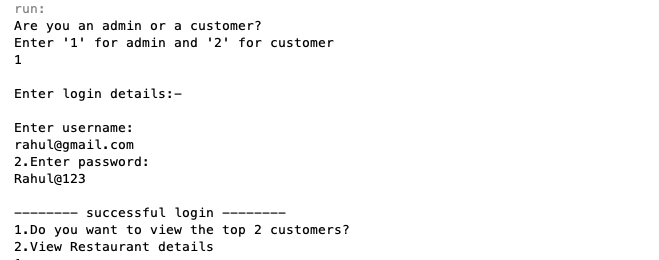
END WHILE;

CLOSE cur1;

END$$

DELIMITER ;

**Screenshot:** It checks whether the username and password matches with the registered details and lets the admin log in if it is correct.



1. **Display details of the top 2 customers:** It displays order details of top 2 customers to admin based on the number of orders.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `top\_2`()

BEGIN

DECLARE b,r,c,qty,i,d int;

DECLARE temp\_cust\_id int(5);

DECLARE a varchar(10);

DECLARE name varchar(20);

DECLARE temp\_order\_no int(5);

DECLARE cust\_name varchar(20);

DECLARE contact char(10);

DECLARE order\_time date;

DECLARE price,bill\_amt,disc,final\_price int(11);

BLOCK1: BEGIN

DECLARE cur3 cursor for SELECT customer\_id from orders group by customer\_id order by count(order\_no) desc limit 5;

DECLARE continue handler for not found set r = 1;

open cur3;

set r = 0;

fetch cur3 into c;

WHILE r = 0 DO

set temp\_cust\_id = c;

set cust\_name = (select customer.customer\_name from customer where customer\_id=temp\_cust\_id limit 1);

set contact = (select customer.contact from customer where customer\_id=temp\_cust\_id limit 1);

BLOCK2: BEGIN

DECLARE cur1 CURSOR FOR SELECT order\_no from orders where orders.customer\_id = temp\_cust\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

SET b = 0;

OPEN cur1;

SET b = 0;

FETCH cur1 INTO a;

WHILE b = 0 DO

set temp\_order\_no = a;

set order\_time = (select orders.order\_date from orders where order\_no = temp\_order\_no);

set bill\_amt = (select bill.bill\_amount from bill where order\_no = temp\_order\_no);

set disc = (select bill.discount from bill where order\_no = temp\_order\_no);

set final\_price = (select bill.bill\_total from bill where order\_no = temp\_order\_no);

BLOCK3: BEGIN

DECLARE cur2 CURSOR FOR SELECT item\_id from order\_details where order\_details.order\_no = temp\_order\_no;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET c = 1;

SET c = 0;

OPEN cur2;

SET c = 0;

FETCH cur2 INTO d;

WHILE c = 0 DO

set name = (select food\_item.item\_name from food\_item where item\_id=d);

set qty = (select order\_details.qty from order\_details where order\_no=temp\_order\_no and item\_id = d);

set price = (select order\_details.price from order\_details where order\_no=temp\_order\_no and item\_id = d);

select temp\_cust\_id,cust\_name,contact,temp\_order\_no,order\_time,d,name,qty,price,bill\_amt,disc,final\_price;

fetch cur2 into d;

END WHILE;

CLOSE cur2;

END BLOCK3;

FETCH cur1 INTO a;

END WHILE;

CLOSE cur1;

END BLOCK2;

fetch cur3 into c;

END WHILE;

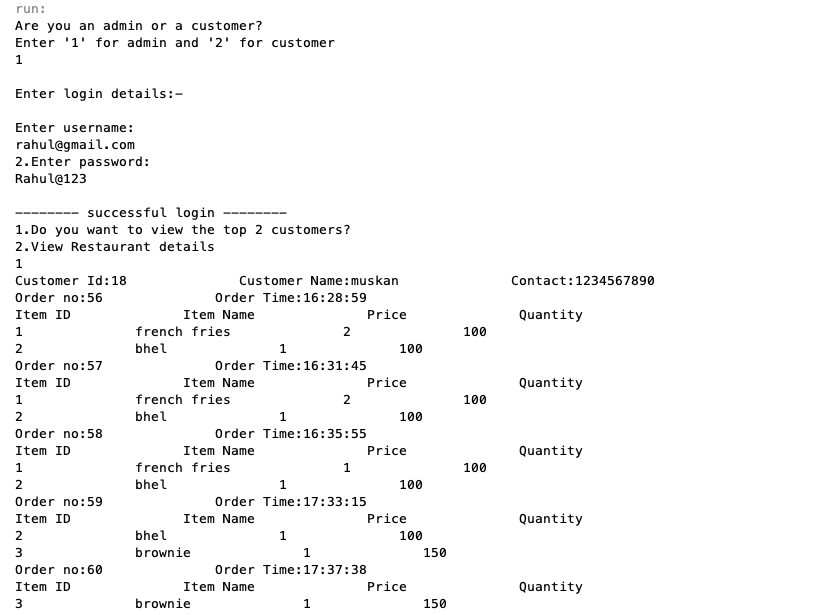
CLOSE cur3;

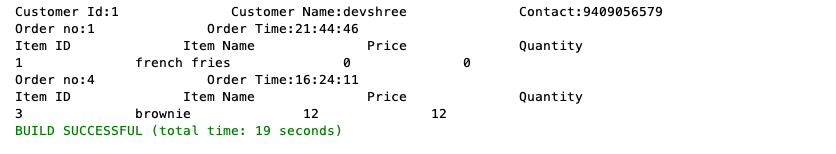
END BLOCK1;

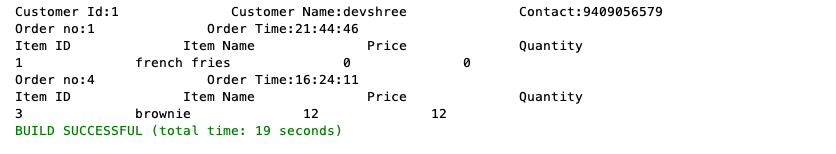
END$$

DELIMITER ;

**Screenshot:** It displays top 2 customer details including name, contact and ID and their order details of each order previously ordered. It also displays item details of each order.







1. **Display restaurant details:** It displays the details of the restaurants to the admin.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `restaurant\_details3`()

BEGIN

DECLARE a int;

DECLARE z int;

DECLARE order\_N int;

DECLARE temp\_order\_no int;

DECLARE temp\_rest\_id int;

DECLARE b,r int;

DECLARE c varchar(10);

DECLARE name varchar(20);

DECLARE city varchar(20);

DECLARE temp\_name varchar(20);

DECLARE temp\_city varchar(20);

DECLARE d int;

DECLARE e,f time;

DECLARE g char(10);

BLOCK1: BEGIN

DECLARE cur1 cursor for select restaurant\_id from restaurant;

DECLARE continue handler for not found set b = 1;

open cur1;

set b=0;

set r=0;

set z=0;

fetch cur1 into a;

WHILE b = 0 DO

set temp\_rest\_id = a;

set c = (select restaurant\_name from restaurant where restaurant\_id=temp\_rest\_id);

set d = (select no\_of\_branches from restaurant where restaurant\_id=temp\_rest\_id);

set e = (select open\_time from restaurant where restaurant\_id = temp\_rest\_id);

set f = (select close\_time from restaurant where restaurant\_id = temp\_rest\_id);

set g = (select contact from restaurant where restaurant\_id = temp\_rest\_id);

BLOCK2: BEGIN

DECLARE cur2 cursor FOR SELECT branch\_name,branch\_city from branch where branch.restaurant\_id = temp\_rest\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET z = 1;

open cur2;

set z=0;

fetch cur2 into name,city;

WHILE z = 0 DO

set temp\_name = name;

set temp\_city = city;

BLOCK3: BEGIN

DECLARE cur3 CURSOR FOR SELECT count(order\_no) from orders where orders.restaurant\_id = temp\_rest\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET r = 1;

SET r = 0;

OPEN cur3;

SET r = 0;

FETCH cur3 INTO order\_N;

WHILE r = 0 DO

set temp\_order\_no = order\_N;

select temp\_rest\_id ,c ,d ,e ,f ,g ,temp\_name ,temp\_city ,temp\_order\_no;

FETCH cur3 INTO order\_N;

end while;

CLOSE cur3;

END BLOCK3;

FETCH cur2 INTO name, city;

end while;

CLOSE cur2;

END BLOCK2;

FETCH cur1 into a;

end while;

CLOSE cur1;

END BLOCK1;

END$$

DELIMITER ;

**Screenshot:** It displays all the restaurant details with descending order of number of orders ordered in that particular restaurant. Like if 1st restaurant has got the highest number of orders till date, it will be displayed first.



1. **Display order history:** It displays all the previous order details to the user.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `display`(IN `customer\_id` INT)

BEGIN

DECLARE name varchar(30);

DECLARE b int;

DECLARE qty int;

DECLARE a int;

DECLARE c int;

DECLARE d int;

DECLARE price1 int(11);

DECLARE order\_time TIMESTAMP;

DECLARE bill\_amt,disc,final\_price int;

DECLARE temp\_item\_id int;

DECLARE temp\_order\_no int;

BLOCK1: BEGIN

DECLARE cur1 CURSOR FOR SELECT orders.order\_no from orders where orders.customer\_id = customer\_id;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

OPEN cur1;

SET b = 0;

FETCH cur1 INTO a;

WHILE b = 0 DO

set temp\_order\_no = a;

set order\_time = (select orders.order\_time from orders where orders.order\_no = temp\_order\_no);

BLOCK2: BEGIN

DECLARE cur2 CURSOR FOR SELECT order\_details.item\_id from order\_details where order\_details.order\_no = temp\_order\_no;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET c = 1;

OPEN cur2;

SET c = 0;

FETCH cur2 INTO d;

WHILE c = 0 DO

set temp\_item\_id = d;

set name = (select food\_item.item\_name from food\_item where food\_item.item\_id=d);

set qty = (select order\_details.qty from order\_details where order\_details.order\_no=temp\_order\_no and item\_id = d);

set price1 = (select price from order\_details where order\_details.order\_no=temp\_order\_no and item\_id = d);

FETCH cur2 into d;

END WHILE;

CLOSE cur2;

END BLOCK2;

set bill\_amt = (select bill.bill\_amount from bill where bill.order\_no = temp\_order\_no limit 1);

set disc = (select bill.discount from bill where bill.order\_no = temp\_order\_no limit 1);

set final\_price = (select bill.bill\_total from bill where bill.order\_no = temp\_order\_no limit 1);

select temp\_order\_no as order\_no,order\_time,temp\_item\_id as item\_id, name,qty,price1 as Price,bill\_amt as bill\_amount,disc as discount,final\_price;

FETCH cur1 INTO a;

END WHILE;

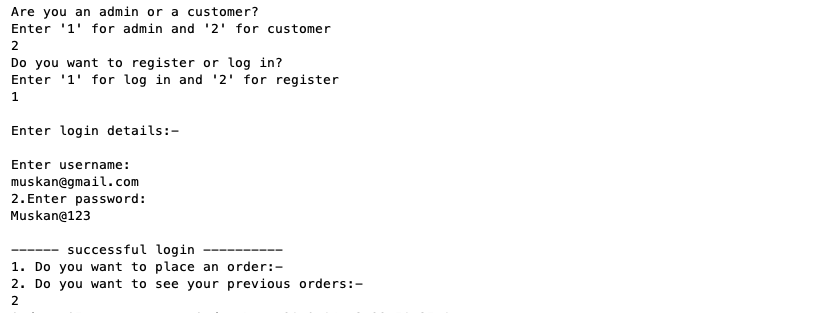
CLOSE cur1;

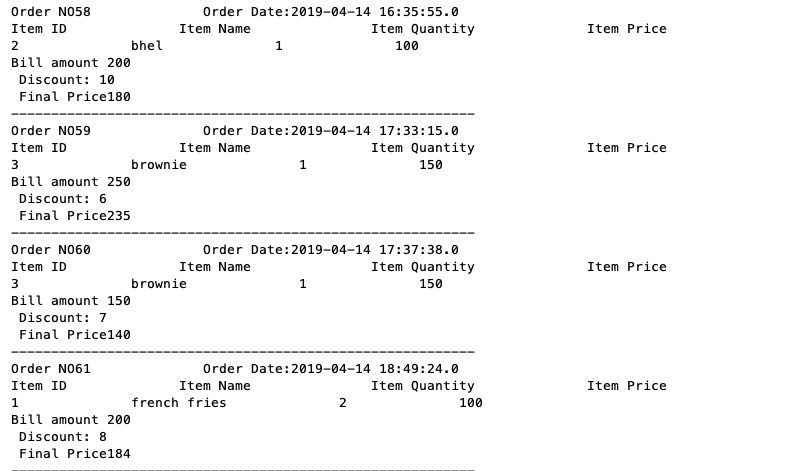
END BLOCK1;

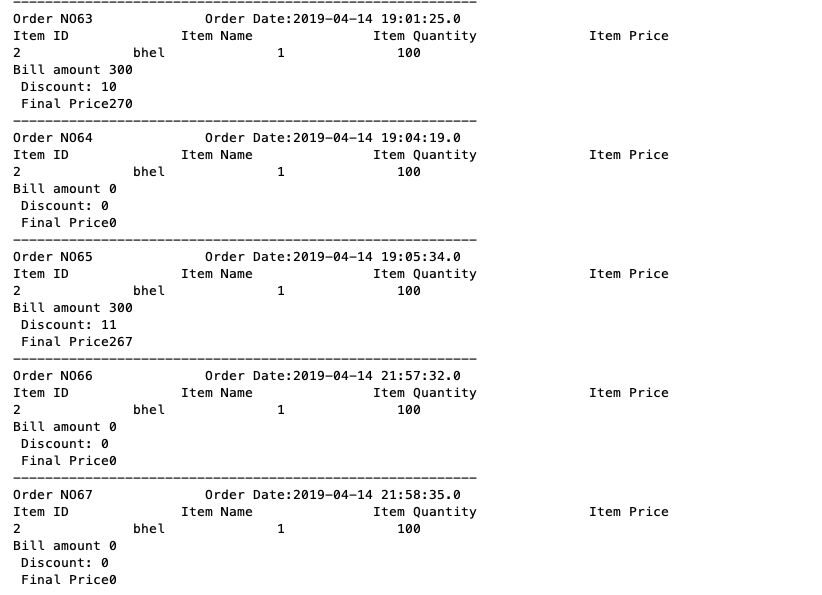
END$$

DELIMITER ;

**Screenshot:** It displays previous order details of that particular customer. For example: here user ‘muskan’ can view its previous orders as shown.







1. **Place an order:** It inserts the details into the order table when a customer places an order.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `orderInsert`(IN `cust\_id` INT(5), IN `rest\_id` INT(5))

BEGIN

DECLARE deli\_add\_id int(5);

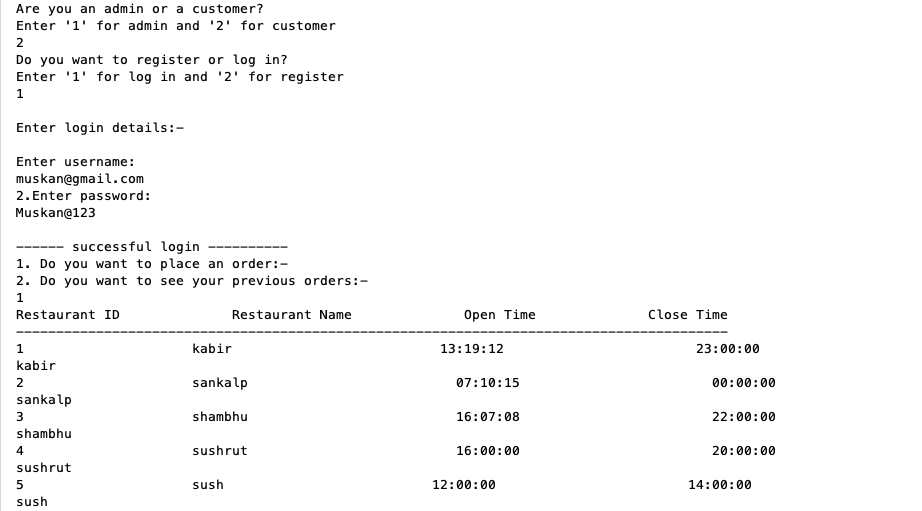
set deli\_add\_id = (select delivery\_address.delivery\_address\_id from delivery\_address where delivery\_address.customer\_id=cust\_id limit 1);

insert into orders(customer\_id, delivery\_address\_id, restaurant\_id, order\_status) values(cust\_id, deli\_add\_id, rest\_id, 'confirmed');

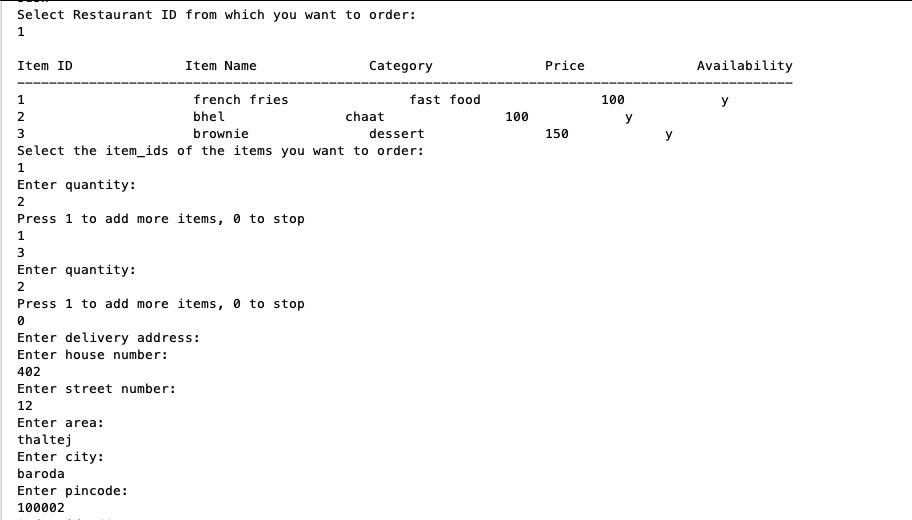
END$$

DELIMITER ;

**Screenshot**:-It asks user to place an order in the preferred restaurant by displaying all the restaurant details



**Screenshot**:It asks user to choose the item to order along with the quantity



1. **Adding items to the placed order:** It inserts the details of order like the number of items, their quantity and so on into the order\_details table.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `orderDetailsInsert`(IN `cust\_id` INT, IN `qty1` INT, IN `it\_id` INT)

BEGIN

DECLARE order\_id int(5);

DECLARE pr,final\_pr int(11);

set order\_id = (select max(order\_no) from orders where orders.customer\_id=cust\_id);

set pr = (select food\_item.price from food\_item where food\_item.item\_id=it\_id);

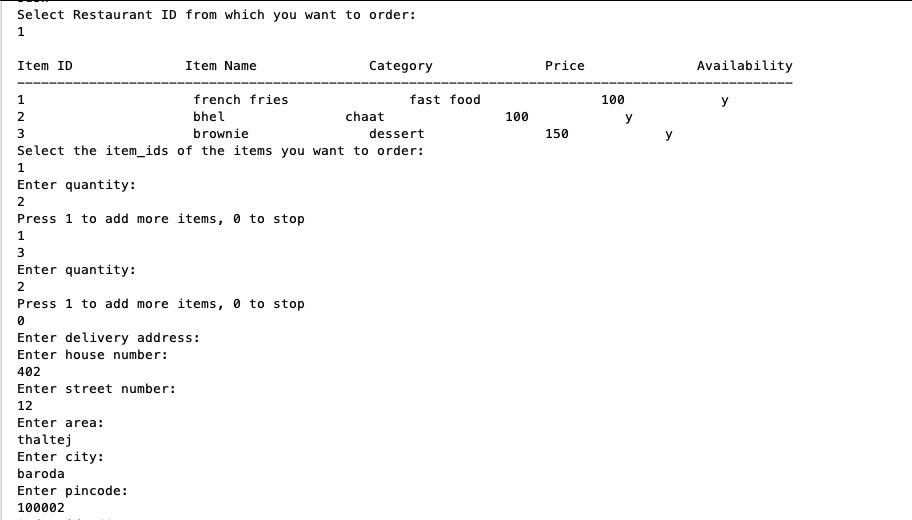
set final\_pr = (pr\*qty1);

insert into order\_details(order\_no, item\_id, price, qty, final\_price) values(order\_id, it\_id, pr, qty1, final\_pr);

END$$

DELIMITER ;

Screenshot: inserts into order details of every item by multiplying price and quantity.



1. **Delivery Address:** The details of the delivery address are inserted via this procedure.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `deliveryAddress\_insert`(IN `house\_no1` VARCHAR(10), IN `street\_no1` VARCHAR(10), IN `area1` VARCHAR(20), IN `city1` VARCHAR(20), IN `pincode1` CHAR(6), IN `customer\_id` INT(5))

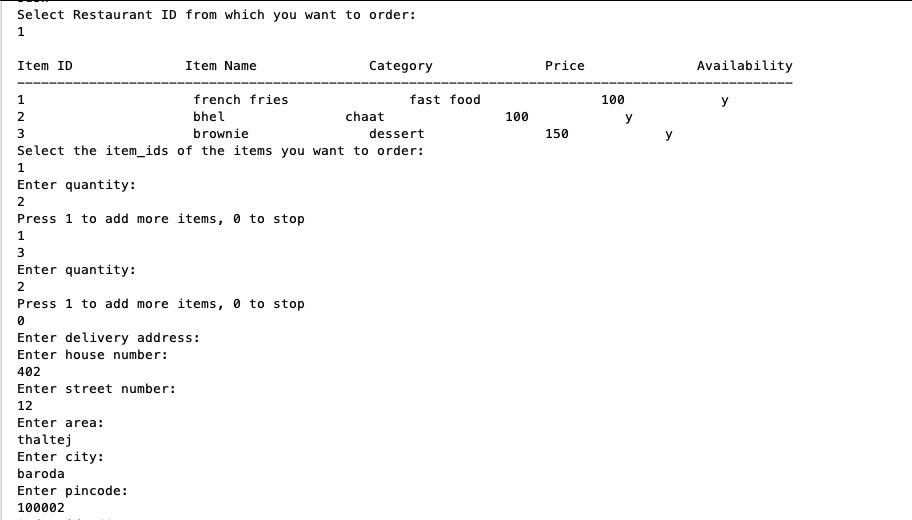
BEGIN

INSERT INTO delivery\_address(house\_no,street\_no,area,city,pincode,customer\_id) values(house\_no1, street\_no1, area1, city1, pincode1,customer\_id);

END$$

DELIMITER ;

Screenshot: Enters delivery address into its table.



1. **Estimated Time of Arrival:** This procedure decides which branch of the requested restaurant will be delivering to the delivery address by calculating the minimum distance. Then the estimated time of arrival will be calculated based on the distance of the restaurant to the delivery address.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `eta`(IN `cust\_id` INT(5), IN `restaurant\_id1` INT(5))

begin

Declare b,min int;

Declare a,c char(6);

Declare eta int;

Declare pin char(6);

DECLARE order\_id,did int(5);

Declare cur1 cursor for select b.pincode from branch b,restaurant r where r.restaurant\_id = restaurant\_id1 and b.restaurant\_id = r.restaurant\_id;

Declare continue handler for not found set b = 1;

set did = (select max(delivery\_address\_id) from delivery\_address where d.customer\_id=cust\_id);

Set pin = (select delivery\_address.pincode from delivery\_address where delivery\_address.delivery\_address\_id = did);

set min = 5;

set b = 0;

set order\_id = (select max(order\_no) from orders where orders.customer\_id=cust\_id limit 1);

open cur1;

fetch cur1 into a;

while b=0 Do

if((a-c)<=min) then

set min = (a-c);

end if;

fetch cur1 into a;

end while;

close cur1;

if(min<=5) then

set eta = 10;

end if;

if(min>5 and min<=10) then

set eta = 20;

end if;

if(min>10 and min<=15) then

set eta = 30;

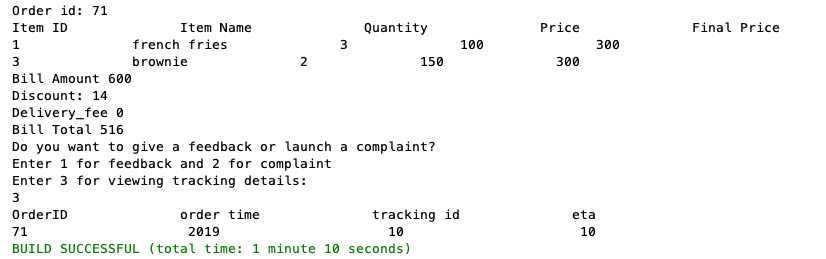
end if;

insert into tracking\_details(order\_no,eta,status) values(order\_id,eta,'OD');

end$$

DELIMITER ;

**Screenshot:**It displays the tracking details of the particular order with its order\_id



1. **Delivery Fee:** Delivery fee of the order is calculated based on the Estimated Time of Arrival (ETA). Also, if the customer is a premium member then the delivery fee is released. The fee is then inserted into the delivery table.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `delivery\_fee`(IN `cust\_id` INT(5))

BEGIN

DECLARE a,x int(5);

DECLARE d,d\_id int(5);

DECLARE c char;

DECLARE deli\_fee int;

SET a = (SELECT max(order\_no) from orders,customer where customer.customer\_id = orders.customer\_id and customer.customer\_id = customer\_id1);

SET c = (SELECT registration.premium\_member from registration,customer where registration.username = customer.email and customer.customer\_id = customer\_id1);

SET x = (SELECT tracking\_details.tracking\_id from tracking\_details where tracking\_details.order\_no=a);

SET d = (SELECT ETA from tracking\_details where tracking\_details.tracking\_id = x);

SET d\_id = (SELECT max(delivery\_address.delivery\_address\_id) from delivery\_address where delivery\_address.customer\_id=customer\_id1);

if (d = 10) then

set deli\_fee = 10;

end if;

if(d = 20) then

set deli\_fee = 20;

end if;

if(d = 30) then

set deli\_fee = 30;

end if;

if(c='y') THEN

set deli\_fee = 0;

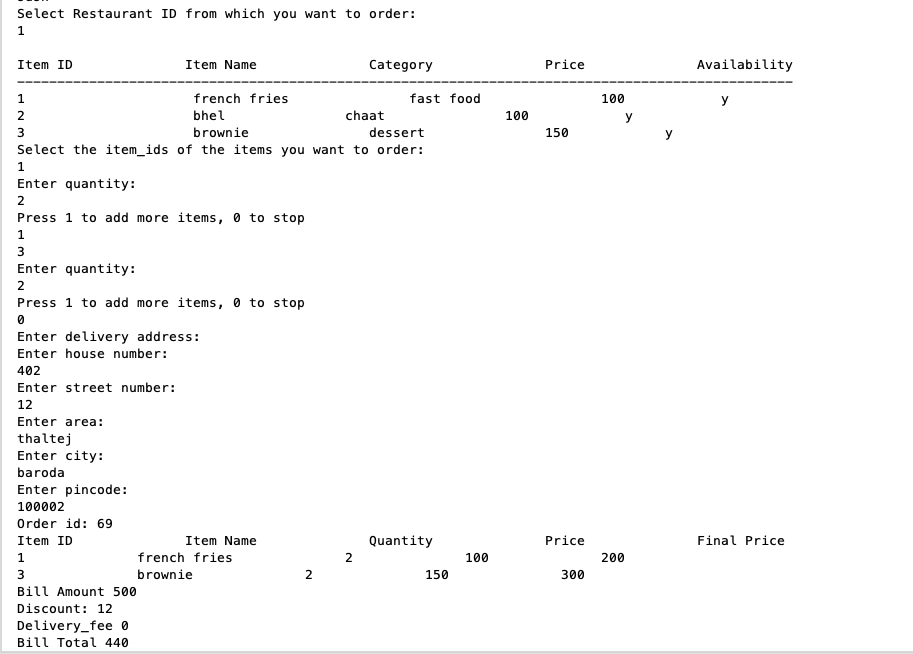
end if;

insert into delivery (delivery\_address\_id, delivery\_staff\_id, order\_no,delivery\_fee) values(d\_id, 1, a,deli\_fee);

END$$

DELIMITER ;

**Screenshot**: It displays the delivery fee as 0, because the logged in user is a premium member.Otherwise the fee is calculated according to the distance between customer’s address and branch address of the restaurant.



1. **Provision of offer/discount:** If a customer has placed orders a given number of times then that customer will be rewarded with a discount of 100 rupees. The validity of this offer will be 7 days from the present date.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `offer\_1`(IN `customer\_id1` INT)

BEGIN

DECLARE count1,date int;

DECLARE dt date;

DECLARE st date;

DECLARE discount int;

SET count1 = (SELECT count(order\_no) from orders where orders.customer\_id = customer\_id1 group by orders.customer\_id);

if(count1>=2) then

set discount = 100;

set st = (select CURDATE());

set dt = DATE\_ADD(CURRENT\_TIMESTAMP, INTERVAL 7 DAY);

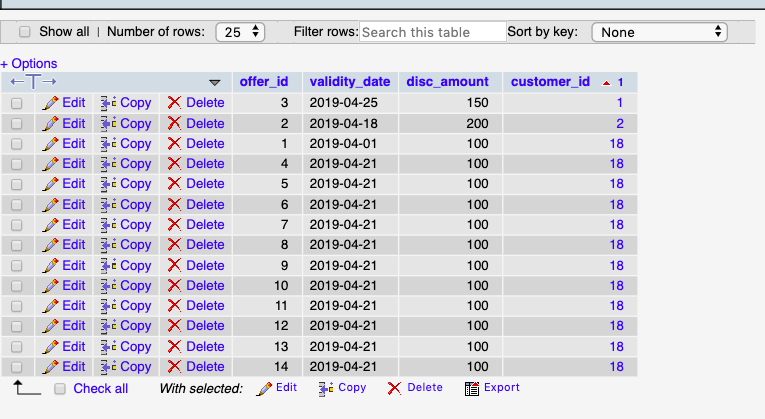
insert into offer(customer\_id,validity\_date,disc\_amount) values(customer\_id1,dt, discount);

end if;

END$$

DELIMITER ;

**Screenshot:** It enters details into offer according to the previous number of orders that user has placed.



1. **Insertion of bill details:** The details of bill like bill amount, discount, delivery fee are inserted into the bill table.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `bill\_details4`(IN `cust\_id` INT(5))

BEGIN

DECLARE fee,bill\_amt,bill\_total,discount int(11);

DECLARE count1,o\_no,order\_id int(5);

DECLARE q int(11);

DECLARE p int(11);

DECLARE b int;

DECLARE cur CURSOR FOR SELECT price,qty FROM order\_details where order\_details.order\_no = (select max(orders.order\_no) from orders where orders.customer\_id=cust\_id limit 1);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

SET bill\_amt=0;

SET fee=(SELECT delivery.delivery\_fee from orders,delivery where delivery.order\_no = (select max(order\_no) from orders where customer\_id=cust\_id) limit 1);

OPEN cur;

SET b = 0;

set fee=0;

FETCH cur INTO p,q;

WHILE b = 0 DO

SET bill\_amt=bill\_amt+(p\*q);

FETCH cur INTO p,q;

END WHILE;

select bill\_amt;

set order\_id = (select max(orders.order\_no) from orders where orders.customer\_id=cust\_id);

set discount = 0;

set discount = (select max(offer.offer\_id) from offer where offer.customer\_id=cust\_id and offer.validity\_date>=CURRENT\_DATE);

set bill\_total = bill\_amt - (bill\_amt\*discount)/100;

insert into bill(order\_no,bill\_amount,discount,bill\_total,delivery\_fee) values(order\_id,bill\_amt,discount,bill\_total,fee);

CLOSE cur;

END$$

DELIMITER ;

1. **Display Bill details:** Details of the bill are displayed via this procedure.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `bill\_procedure`(IN `cust\_id` INT(5))

BEGIN

DECLARE order\_id,d int(5);

DECLARE c int;

DECLARE name varchar(20);

DECLARE bill\_total,qty,price,final\_pr,bill\_amount,discount,delivery\_fee int(11);

DECLARE cur2 CURSOR FOR SELECT order\_details.item\_id from order\_details where order\_details.order\_no = (select max(order\_no) from orders where orders.customer\_id=cust\_id LIMIT 1);

DECLARE CONTINUE HANDLER FOR NOT FOUND SET c = 1;

set order\_id = (select max(order\_no) from orders where orders.customer\_id=cust\_id limit 1);

set bill\_total = (select bill.bill\_total from bill where bill.order\_no=order\_id limit 1);

set bill\_amount = (SELECT bill.bill\_amount from bill where bill.order\_no=order\_id limit 1);

set discount = (SELECT bill.discount from bill where bill.order\_no=order\_id limit 1);

set delivery\_fee = (SELECT bill.delivery\_fee from bill where bill.order\_no=order\_id limit 1);

OPEN cur2;

SET c = 0;

FETCH cur2 INTO d;

WHILE c = 0 DO

set name = (select food\_item.item\_name from food\_item where item\_id=d);

set qty = (select order\_details.qty from order\_details where order\_no=order\_id and item\_id = d);

set price = (select order\_details.price from order\_details where order\_no=order\_id and item\_id = d limit 1);

set final\_pr = (select final\_price from order\_details where order\_no=order\_id and item\_id = d);

select order\_id as ord\_id,d,name,qty,price,final\_pr,bill\_amount,discount,delivery\_fee,bill\_total;

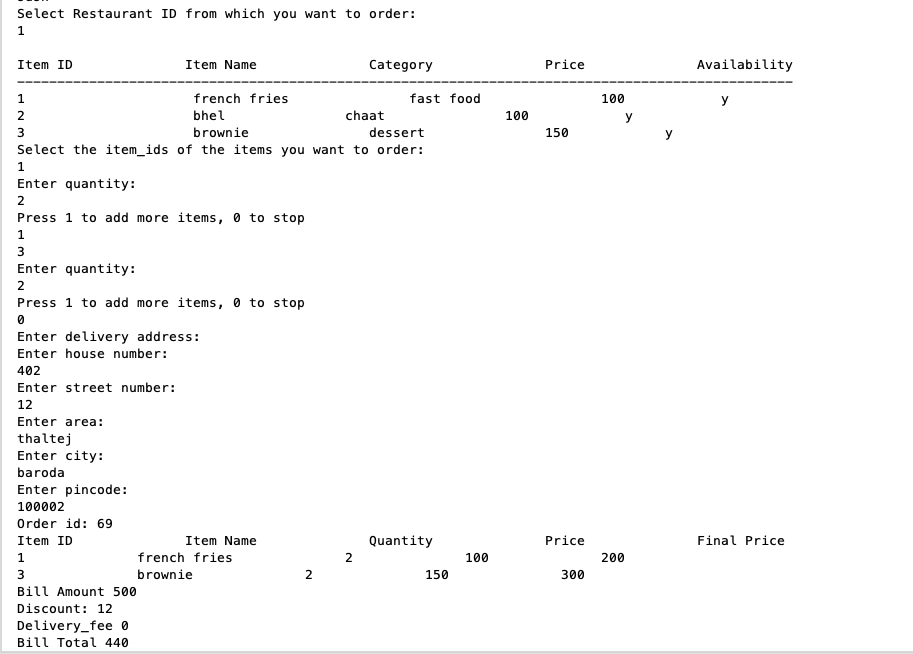
FETCH cur2 INTO d;

END WHILE;

CLOSE cur2;

END$$

DELIMITER ;

**Screenshot**: It displays the bill details of the order with all items he/she has ordered, discount and final bill amount.

1. **Feedback:** Takes in the feedback given by the customer.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `feedback1`(IN `order\_no` INT(5), `cust\_id` INT(5), `rating` INT(11), `comment1` VARCHAR(20))

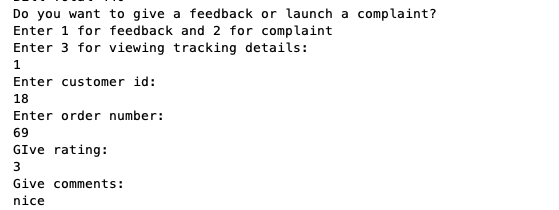
BEGIN

INSERT INTO feedback(customer\_id,feedback.order\_no,feedback.rating,feedback.comment) values(cust\_id,order\_no,rating, comment1);

END$$

DELIMITER ;

**Screensho**t: Allows user to file a feedback.



1. **Complaint:** Inserts the complaint type and other details of complaint filed by the customer.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `complaint`(IN `order\_no` INT, `cust\_id` INT, `complaint\_type` VARCHAR(50), `complaint\_date` DATE)

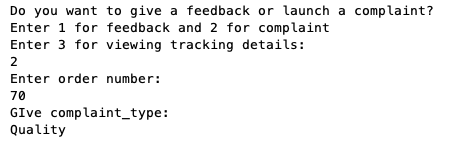
BEGIN

INSERT INTO complaint(customer\_id,order\_no,complaint\_type,complaint\_date) values(cust\_id,order\_no,complaint\_type,complaint\_date);

END$$

DELIMITER ;

**Screenshot**: Allows user to file a complaint.



**19. Premium Wallet**: This procedure checks if the registered customer is a premium member or not. If the customer is a premium member then an amount of 500 rupees will be deducted from the existing wallet balance.

Code:

DELIMITER $$

CREATE DEFINER=`root`@`localhost` PROCEDURE `premium\_wallet`(IN `premium` CHAR(1), IN `username` VARCHAR(50))

BEGIN

DECLARE account\_id int(5);

DECLARE wallet\_id1 int(5);

DECLARE premium\_stat char(1);

DECLARE c\_id int(5);

SET c\_id = (SELECT customer\_id from customer where customer.email=username);

SET wallet\_id1 =(SELECT wallet.wallet\_id FROM wallet WHERE wallet.customer\_id=c\_id limit 1);

IF premium\_stat='y' THEN

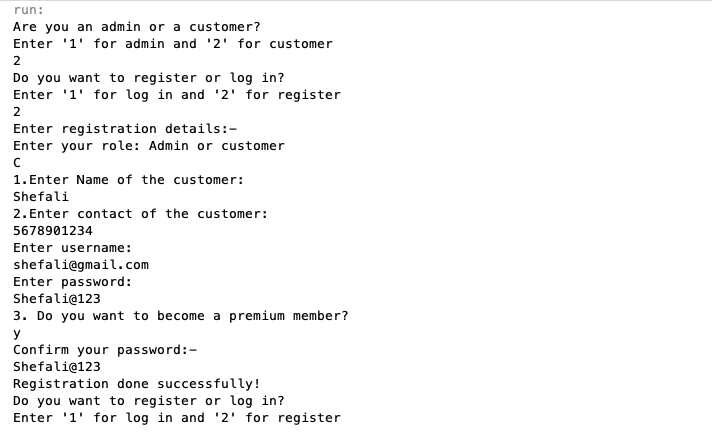
update wallet set wallet.wallet\_balance=wallet.wallet\_balance-500 where wallet.wallet\_id=wallet\_id1;

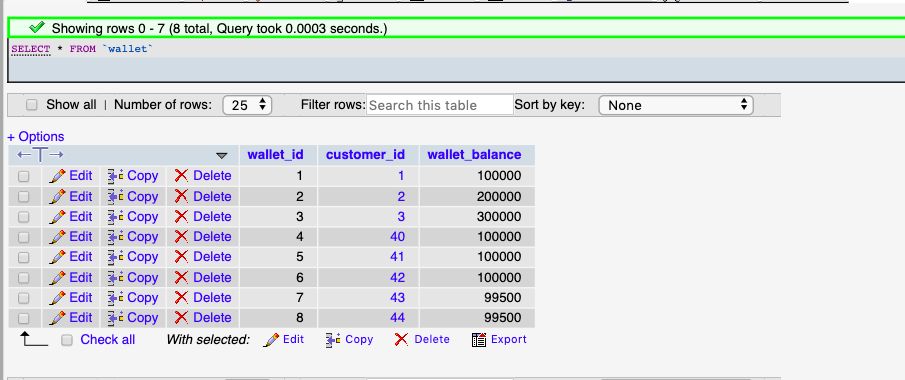
END IF;

END$$

DELIMITER ;

Screenshot: Updates the wallet of premium member by charging 500 as the amount for premium membership after registration



****

**Triggers**

1. **Check availability of food items:** If the ordered food item is available then the new availability status (no of items) will be updated in the food\_item table. If the ordered item is not available then an error message will be displayed.

Code:

CREATE TRIGGER `avail` BEFORE INSERT ON `order\_details`

FOR EACH ROW BEGIN

DECLARE av int(11);

DECLARE msg varchar(50);

SET av= (SELECT food\_item.availability from food\_item WHERE food\_item.item\_id=new.item\_id);

IF av>0 and av>new.qty THEN

UPDATE food\_item set food\_item.availability =(av-new.qty) where food\_item.item\_id=new.item\_id;

ELSE

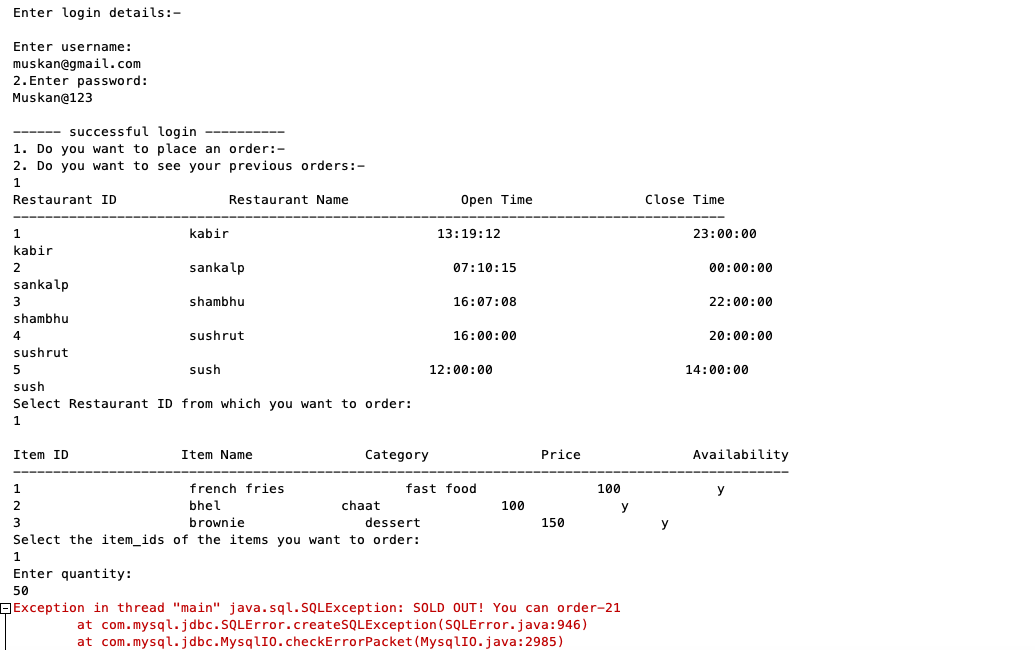
SET msg= concat('SOLD OUT! You can order',av-new.qty);

signal sqlstate '45001' set message\_text = msg;

END IF;

END

Screenshot: Checks the availability of the items the customer wants to order and displays error as sold out when availability is 0.



1. **Check availability of restaurant:** The time of placing the order is checked with the open time and close time of the restaurant. If the corresponding time falls in that range then food items can be ordered and if not then an error message will be displayed.

Code:

CREATE TRIGGER `check\_restro` BEFORE INSERT ON `orders`

FOR EACH ROW BEGIN

DECLARE a,b,c time;

DECLARE msg varchar(128);

SET c=CURRENT\_TIMESTAMP;

SET a=(SELECT open\_time from restaurant where restaurant\_id=new.restaurant\_id);

SET b=(SELECT close\_time from restaurant where restaurant\_id=new.restaurant\_id);

IF c NOT BETWEEN CONVERT(a,TIME) AND CONVERT(b,TIME) THEN

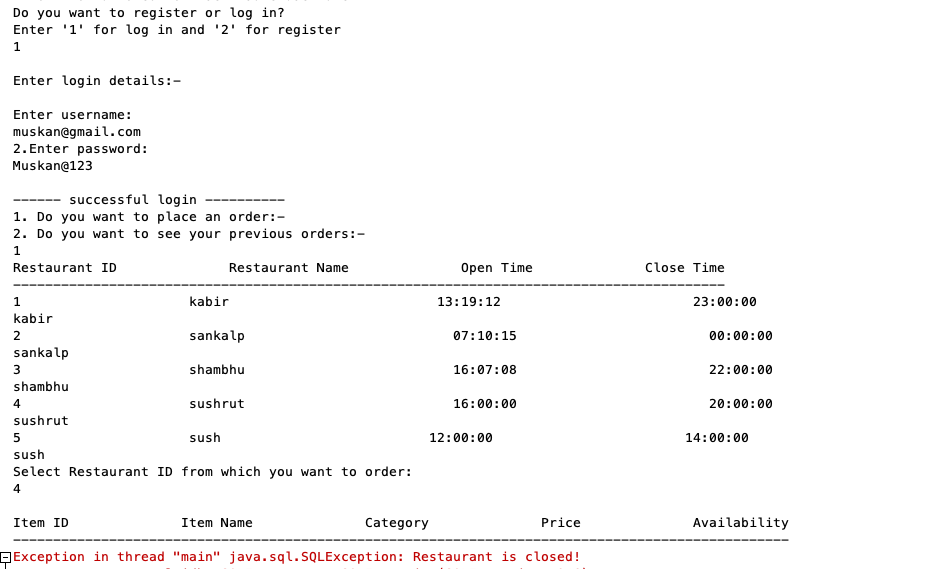
set msg = 'Restaurant is closed!';

signal sqlstate '45001' set message\_text = msg;

END IF;

END

Screenshot: Checks if the user can place an order at a specific restaurant based on its working hours and displays appropriate message when restaurant is closed



1. **On registration:** This trigger checks for the username at the time of registration. If the newly entered username already exists then an error message will be displayed.

Code:

CREATE TRIGGER `trigger1` BEFORE INSERT ON `registration`

FOR EACH ROW BEGIN

DECLARE b int;

DECLARE a varchar(50);

DECLARE msg varchar(100);

DECLARE cur CURSOR FOR SELECT username from registration;

DECLARE CONTINUE HANDLER FOR NOT FOUND SET b = 1;

OPEN cur;

SET b = 0;

FETCH cur INTO a;

WHILE b = 0 DO

if a=new.username then

set msg = 'Error: Not allowed to insert same username....';

signal sqlstate '45006' set message\_text = msg;

end if;

FETCH cur INTO a;

END WHILE;

CLOSE cur;

END

Screenshot: Checks for the uniqueness of the username otherwise throws error

