SSN College Of Engineering

Kalavakkam – 603110

Department of Computer Science and Engineering

UCS2404 Database Management Systems

Mini Project

Food Delivery Management System

Abhishek Reddy Lingareddy (3122225001003)

Darrshan Hariharan (3122225001022)

Deepak Chandar S (3122225001023)

SSN College Of Engineering

Kalavakkam – 603110

Department of Computer Science and Engineering

UCS2404 Database Management Systems

Assignment 1

Food Delivery Management System

Abhishek Reddy Lingareddy (3122225001003)

Darrshan Hariharan (3122225001022)

Deepak Chandar S (3122225001023)

ABSTRACT

Our proposed system is an online food ordering system that enables ease for the customers. It overcomes the disadvantages of the traditional queueing system. Our proposed system is a medium to order online food hassle-free from restaurants. This system improves the method of taking the order from customers. The online food ordering system sets up a food menu online, and customers can easily place the order as per their wish. Also, with a food menu, customers can easily track the orders. This system also provides a feedback system in which users can rate the food items. Additionally, the proposed system can recommend hotels and food based on the ratings given by the user, and the hotel staff will be informed for improvements along with the quality. The payment can be made online or via a pay-on-delivery system. For more secure ordering, separate accounts are maintained for each user by providing them an ID and a password.

Keywords: online food ordering system, customer convenience, restaurant menu, order tracking, feedback system, food ratings, recommendations, payment, secure ordering, SQL, NetBeans, JAVA Swing GUI

Assumptions about the food delivery system

- Only delivery partners in the same pin code as the restaurant preparing the order can take the order.
- Every address stored in the database ends with the pin code.
- Restaurant rating is not averaged.
- Dish rating is not averaged.
- Delivery partner’s salary is derived from their join year.
- Restaurants must periodically refresh their live orders table to know if any previous order has been cancelled, or if any new order has been assigned.
- Delivery partner must click the “get order” button when he/she is ready to be assigned an order. Thus, orders are assigned to the delivery partners on a first-come first-serve basis.
- Restaurants and Delivery partners sign in using their assigned ID and a password which is a combination of their ID and phone number.
- Time taken to prepare the food is not factored into the delivering process.

R1: USERS (user_id, name, pwd, gender, dob, phone,address)

The following functional dependencies are possible for USERS relation.

Let So denote the set of all FD's.

So = { user_id → name, pwd, gender, dob, phone, phone → user_id, name, pwd, gender,dob, name→ pwd, name→dob, user_id,name→name, user_id,pwd→pwd,name,pwd→user_id, phone,name→name, phone,pwd→pwd, dob→name, dob→pwd}

From the above set So, the FD's name→ pwd, name→dob, dob→name, dob→pwd are time-dependent and the FD's user_idname→name, user_id,pwd→pwd, name,pwd→user_id, phone,name→name and phonepwd→pwd are trivial.

Removing FD's, we get the set of FD's
So = {user_id→name,pwd,gender,dob,phone, phone → user_id,name,pwd,gender,dob}

Finding canonical cover of S:

Step-1: Through decomposition of user_id→name,pwd,gender,dob,phone, and phone→user_id,name,pwd,gender,dob we get the following set of FD's

S1 = {user_id→name, user_id→pwd, user_id→gender, user_id→dob, user_id→phone, phone→user_id, phone→name, phone→ pwd,phone→gender, phone→dob}

Step-2: LHS of each FD is already irreducible.

S2 = { user_id→name, user_id→pwd, user_id→gender, user_id→dob, user_id→phone, phone→user_id,phone→name, phone→pwd, phone→gender, phone→dob}

Step 3: Eliminating redundant FD's
By eliminating user_id→name, we get user_id+ = {user_id, pwd, gender, dob, phone, name}

Thus, we can remove user_id→name as user_id+ contains all the attributes of USdobRS relation, including name
Similarly, we keep removing redundant phonegenders

user_id→pwd => user_id+ = {user_id, gender, dob, phone, name, pwd} contains pwd
user_id→gender => user_id+ = {user_id, dob, phone, name, pwd, gender} contains gender
user_id→dob => user_id+ = {user_id,phone, name, pwd, gender, dob } contains dob
user_id→phone => user_id+ = {user_id} hence, cannot be removed
phone→user_id => phone+ = {phone, name, pwd, gender, dob} doesn't contain user_id

phone→ name => phone+= {phone,user_id,pwd,gender,dob} doesn't contain name
phone→pwd => phone+ = {phone,user_id, name, gender, dob } doesn't contain pwd
phone→gender => phone+ = {phone, user_id, name, pwd, dob} doesn't contain gender
phone→dob => phone+ = {phone, user_id, name, pwd,gender) doesn't contain dob

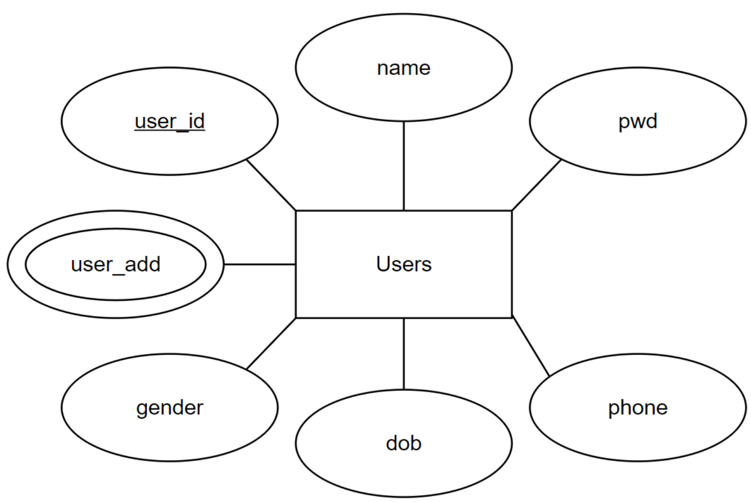
The only FD's which can be removed are user_id→name,user_id→pwd, user_id→gender, user_id→dob, which results in the set of FD's

S3 = {user_id →phone, phone→user_id, phone→ name, phone→ pwd, phone→ gender, phone → dob}

Applying union axiom where necessary, we get the canonical cover
Sc = {user_id →phone; phone→ user_id,name,pwd,gender,dob}

Now, user_id+ = {user_id, phone, name, pwd, gender, dob} contains all attributes of USERS relation. Thus, user_id is a superkey and, since it is minimal, also a candidate key of USERS relation.

Similarly phone+ = {phone, user_id, name, pwd, gender, dob} contains all attributes of USERS relation. Thus, phone is a superkey and, since it is minimal, also a candidate key of USERS relation.



R2: DEL_PARTNERS (emp_id, emp_name, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal)

The following functional dependencies are possible for DEL_PARTNERS relation:

Let So denote the set of possible FDs.
So = { emp_id → emp_name, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal,
emp_phone → emp_id, emp_name, emp_dob, starting_add, license_no, join_year, base_sal,
license_no → emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal,
join_year → base_sal,
emp_id, emp_name → emp_id,
emp_phone, join_year → emp_phone,
license_no, join_year → license_no,
emp_name → emp_id,
join_year → emp_name }

From the above set So the FDs emp_name → emp_id and join_year → emp_name are time-dependent and the FDs emp_id, emp_name → emp_id, emp_phone, join_year → emp_phone and license_no, join_year → license_no are trivial FDs.

Removing such FDs, we get the set of FDs:
S = { emp_id → emp_name, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal,
emp_phone → emp_id, emp_name, emp_dob, starting_add, license_no, join_year, base_sal,
license_no → emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal,
join_year → base_sal }

Finding canonical cover of S:
Step-1: Through decomposition of emp_id → emp_name, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal, emp_phone → emp_id, emp_name, emp_dob, starting_add, license_no, join_year, base_sal and license_no → emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal, we get the following set of FDs:

S1 = { emp_id → emp_name, emp_id → emp_phone, emp_id → emp_dob, emp_id → starting_add, emp_id → license_no, emp_id → join_year, emp_id → base_sal,
emp_phone → emp_id, emp_phone → emp_name, emp_phone → emp_dob, emp_phone → starting_add, emp_phone → license_no, emp_phone → join_year, emp_phone → base_sal,
license_no → emp_id, license_no → emp_name, license_no → emp_phone, license_no → emp_dob, license_no → starting_add, license_no → join_year, license_no → base_sal,

join_year → base_sal }

Step-2: LHS of each FD is already irreducible.

S2 = { emp_id → emp_name, emp_id → emp_phone, emp_id → emp_dob, emp_id → starting_add, emp_id → license_no, emp_id → join_year, emp_id → base_sal,
emp_phone → emp_id, emp_phone → emp_name, emp_phone → emp_dob, emp_phone → starting_add, emp_phone → license_no, emp_phone → join_year, emp_phone → base_sal,
license_no → emp_id, license_no → emp_name, license_no → emp_phone, license_no → emp_dob, license_no → starting_add, license_no → join_year, license_no → base_sal,
join_year → base_sal }

Step-3: Eliminating redundant FDs

By removing emp_id → emp_name, we get emp_id+ = { emp_id, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal, emp_name }
Thus, emp_id → emp_name can be removed as emp_id+ contains emp_name.

Similarly, the following FDs can be removed:
emp_id → emp_phone, emp_id → emp_dob, emp_id → starting_add, emp_id → join_year, emp_id → base_sal

By removing emp_phone → emp_id, we get emp_phone+ = { emp_phone, emp_name, emp_dob, starting_add, license_no, join_year, base_sal, emp_id }
Thus, emp_phone → emp_id can be removed as emp_phone+ contains emp_id.

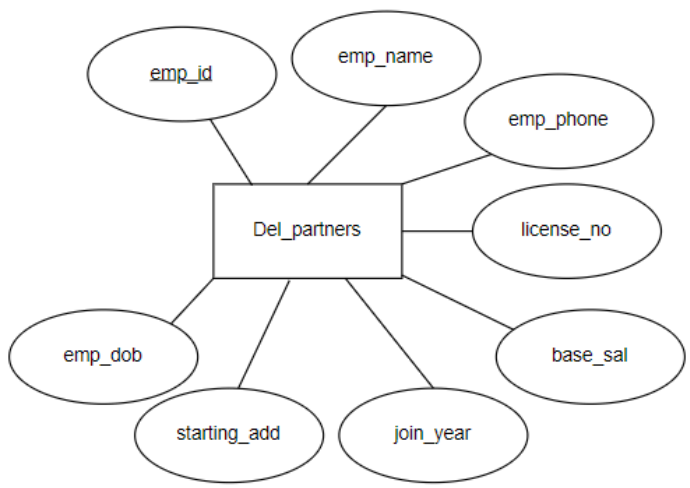
Similarly, the following FDs can be removed:
emp_phone → emp_name, emp_phone → emp_dob, emp_phone → starting_add, emp_phone → join_year, emp_phone → base_sal

The resultant set of FDs is:
S3 = { emp_id → license_no,
emp_phone → license_no,
license_no → emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal,
join_year → base_sal }

Applying union axiom where necessary, we get the canonical cover:
Sc = { emp_id → license_no,
emp_phone → license_no,
license_no → emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal,
join_year → base_sal }

Now,
emp_id+ = { emp_id, license_no, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal }
emp_phone+ = { emp_phone, license_no, emp_id, emp_name, emp_dob, starting_add, join_year, base_sal }
license_no+ = { license_no, emp_id, emp_name, emp_phone, emp_dob, starting_add, join_year, base_sal }
join_year+ = { join_year, base_sal }

Thus, emp_id+, emp_phone+ and license_no+ contain all attributes and hence, are superkeys. Moreover, since emp_id, emp_phone and license_no are minimal superkeys, they are also the candidate keys of DEL_PARTNERS relation.



R3: RESTAURANTS (rest_id, rest_name, owner, rest_add, rest_phone, cuisine, rating)

The functional dependencies set Fn contains {rest_id→name, owner, rest_add, rest_phone, cuisine, rating, rest_add→rest_id, rest_name, owner, rest_phone, cuisine, rating, rest_phone→rest_id, name, owner, rest_add, cuisine, rating, rest_name, owner, rest_id→cuisine, rating, name, owner, rest_add→cuisine, rating, rest_name, owner, rest_phone→cuisine, rating, name→owner}

Each Restaurant in the DB can be uniquely identified by its rest_id or its address or its rest_phone. Hence, each of these attributes can functionally determine every other attribute. Similarly, combinations of restaurant details involving rest_name and owner attributes can also uniquely identify each restaurant.

Step 1: By decomposition,
F1 = {rest_id→name, rest_id→owner, rest_id→rest_add, rest_id→rest_phone, rest_id→cuisine, rest_id→rating, rest_add→rest_id, rest_add→rest_name, rest_add→owner, rest_add→rest_phone, rest_add→cuisine, rest_add→rating, rest_phone→rest_id, rest_phone→name, rest_phone→owner, rest_phone→rest_add, rest_phone→cuisine, rest_phone→rating, rest_name, owner, rest_id→cuisine, name, owner, rest_id→rating, rest_name, owner, rest_add→cuisine, name, owner, rest_add→rating, rest_name, owner, rest_phone→cuisine, name, owner, rest_phone→rating, rest_name→rest_add}

Step 2: name, owner, rest_id→cuisine
Checking rest_name→cuisine
name+= {rest_name, cuisine}
owner, rest_id are not extraneous

Checking owner→cuisine
owned_by+ = {owner, cuisine}
name, rest_id are not extraneous

Checking rest_id→cuisine
rest_id+ = {rest_id, cuisine, rest_name, owner, rest_add, rest_phone, rating}
name, owner are extraneous

rest_name, owner, rest_id→rating
Checking rest_id→rating
rest_id+ = {rest_id, rating, rest_name, owner, rest_add, rest_phone, cuisine}
rest_name, owner are extraneous

rest_name, owner, rest_add→cuisine.
Checking rest_add→cuisine
rest_add+= {rest_add, cuisine, rest_id, rest_name, owner, rest_phone, rating}

rest_name, owner are extraneous

rest_name, owner, rest_add→rating
Checking rest_add→rating
rest_add+ = {rest_add, rating, rest_id, rest_name, owner, rest_phone, cuisine}
rest_name, owner are extraneous

rest_name, owner, rest_phone→cuisine
Checking rest_phone→cuisine
rest_phone+ = {rest_phone, cuisine, rest_id, rest_name, owner, rest_add, rating}
rest_name, owner are extraneous

rest_name, owner, rest_phone→rating
Checking rest_phone→rating
rest_phone+= {rest_phone, rating, rest_id, rest_name, owner, rest_add, cuisine}
rest_name, owner are extraneous.

rest_id→rating
rest_id+ = {rest_id, rating, rest_name, owner, rest_add, rest_phone, cuisine}
rest_id+ contains rating
Therefore rest_id→rating can be removed.

rest_name→owned_by
rest_name+ = {rest_name}
rest_name+ doens't contain owned_by
Therefore rest_name→owned_by can't be removed.

rest_id→cuisine
rest_id+ = {rest_id, cuisine, rest_name, owner, rest_add, rest_phone, rating}
rest_id+ contains cuisine
Therefore rest_id→cuisine can be removed.

Similarly, rest_add→rest_id, rest_add→rest_name, rest_add→owner, rest_add→rest_phone, rest_add→cuisine, rest_add→rating can be removed.

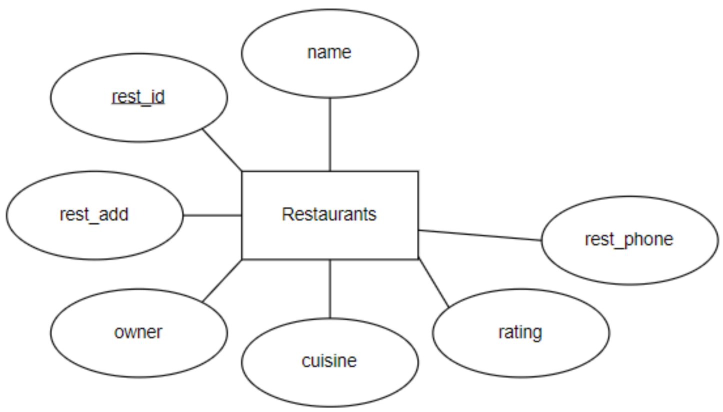
F3 ={rest_id→rest_phone, rest_add→rest_phone, rest_phone→rest_id, rest_phone→rest_name, rest_phone→owner, rest_phone→rest_add, rest_phone→cuisine, rest_phone→rating}
= {rest_id→rest_phone, rest_add→rest_phone, rest_phone→rest_id, rest_phone→rest_name, rest_phone→owner, rest_phone→rest_add, rest_phone→cuisine, rest_phone→rating, rest_phone→rest_id→rest_name, owner, rest_add, rest_phone, cuisine, rating}

Finding SuperKeys:
rest_id+ = {rest_id, rest_phone, rest_name, owner, rest_add, cuisine, rating}
rest_id+ is a superkey

rest_add+ = {rest_add, rest_phone, rest_name, owner, rest_id, cuisine, rating}
rest_add+ is a Superkey.

rest_phone+ = {rest_phone, rest_id, rest_name, owner, rest_add, cuisine, rating}
rest_phone is a Superkey

rest_id, rest_add, and rest_phone are minimal Super Keys. rest_id is chosen as the primary key and rest_add, rest_phone are chosen as the alternate keys.



R4: Menu (item_name, rest_id, veg, rating, price)

Fn = {item_name, rest_id → item_name, item_name, rest_id → rest_id, item_name, rest_id, veg → item_name, rest_id, item_name → rest_id, item_name → price, item_name, rest_id → veg, rating, price, item_name → veg}

Here, item_name, rest_id → item_name, item_name, rest_id → rest_id, item_name, rest_id, veg → item_name, rest_id are trivial FDs and item_name → price, item_name → rest_id are time-dependent FDs.

Fn = {item_name, rest_id → veg, rating, price, item_name → veg}

Finding canonical cover:
Step 1: By decomposition.
F1 = {item_name, rest_id → veg, item_name, rest_id → rating, item_name, rest_id → price, item_name → veg}

Step 2: item_name, rest_id → veg
checking item_name → veg
item_name+ = {item_name, veg}
.: rest_id is not extraneous
checking rest_id → veg
rest_id+ = {rest_id, veg}
.: item_name is not extraneous

Similarly, there are no extraneous attributes in
item_name, rest_id → rating and item_name, rest_id → price.

F2 = { item_name, rest_id → veg, item_name, rest_id → rating, item_name, rest_id → price, item_name → veg}

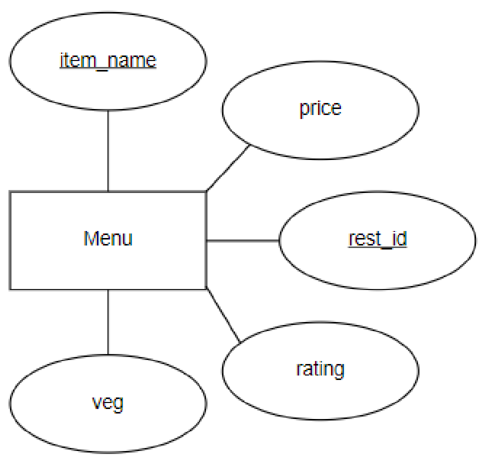
Step 3: Finding redundant FDs

item_name, rest_id → veg
item_name, rest_id+ = {item_name, rest_id, rating, price, veg}
item_name, rest_id+ contains veg
item_name, rest_id → veg can be removed.

item_name, rest_id+ does not contain rating
item_name, rest_id → rating is not redundant
item_name, rest_id → price
item_name, rest_id+ = {item_name, rest_id, rating, veg}
item_name, rest_id+ does not contain price
item_name, rest_id → price is not redundant

F3 = { item_name, rest_id → rating, item_name, rest_id → price, item_name → veg} =
{item_name, rest_id → rating, price, item_name → veg}

Finding superkeys:
item_name, rest_id+ = {item_name, rest_id, rating, price, veg}
item_name, rest_id is a Superkey and it is minimal
item_name, rest_id is a candidate key
item_name+ = {item_name, veg}
item_name is not a superkey.
item_name, rest_id is chosen as a composite key.



R5: ORDERS(items, order_no, user_id, rest_id, emp_id, ord_time, delv_time, delv_add, qty)

Fn = {order_no, user_id→order_no, order_no, rest_id→order_no, user_id, rest_id→user_id, rest_id→order_no, user_id, rest_id→order_no, user_id, order_no, user_id, rest_id→ord_time, delv_time, delv_add, emp_id→order_no, user_id, rest_id, ord_time, delv_time, delv_add}

Here, order_no, user_id→order_no, order_no, rest_id→order_no, user_id, rest_id→user_id, rest_id→order_no, user_id, rest_id→order_no, user_id, order_no, user_id, rest_id→ord_time, delv_time, delv_add are trivial FDs.

Also, user_id→order_no and rest_id→user_id are time-dependent FDs.

order_no→ord_time, delv_time, delv_add {order_no can uniquely identify all the tuples}
ord_time, delv_time→order_no, user_id, rest_id, emp_id, delv_add {Every order can only be delivered once at a time}
emp_id, delv_time→order_no, user_id, rest_id, ord_time, delv_add {An employee can only deliver one order at a given time}

Fn = {order_no→user_id, rest_id, ord_time, delv_time, delv_add}
Fn = {order_no→ord_time, delv_time, delv_add, ord_time, delv_time→order_no, user_id, rest_id, emp_id, delv_add, emp_id, delv_time→order_no, user_id, rest_id, ord_time, delv_add}

Step 1: By Decomposition

F1 = { order_no→user_id, order_no→rest_id, order_no→ord_time, order_no→delv_time, order_no→delv_add, ord_time, delv_time→order_no, ord_time, delv_time→user_id, ord_time, delv_time→rest_id, ord_time, delv_time→delv_add, emp_id, delv_time→order_no, emp_id, delv_time→user_id, emp_id, delv_time→rest_id, emp_id, delv_time→ord_time, emp_id, delv_time→delv_add}

Step 2:
ord_time, delv_time→user_id
Checking ord_time→user_id
ord_time+ = {ord_time, user_id}
delv_time is not extraneous

emp_id, delv_time→order_no
Checking emp_id→order_no
emp_id+ = {emp_id, order_no}
delv_time is not extraneous

Similarly, we can obtain that there are no extraneous attributes in F1.

Step 3: Removing Redundant FDs
order_no→user_id
order_no+ = {order_no, ord_time, delv_time, delv_add, user_id} => order_no+ contains user_id
order_no→user_id can be removed

order_no→rest_id
order_no+ = {order_no, user_id, rest_id, ord_time, delv_time, delv_add} => order_no+ contains rest_id
order_no→rest_id can be removed

order_no→ord_time
order_no+ = { order_no, delv_time, delv_add} => order_no+ does not contain ord_time
order_no→ord_time cannot be removed

order_no→delv_time
order_no+ = {order_no, user_id, rest_id, delv_add, ord_time, delv_time} => order_no+ contains delv_time
order_no→delv_time can be removed

order_no→delv_add
order_no+= {order_no, ord_time, delv_time} => order_no+ does not contain delv_add
order_no→delv_add cannot be removed.

ord_time, delv_time→order_no
ord_time+ = { ord_time, delv_time, delv_add, user_id, rest_id, order_no} => ord_time+ contains order_no
ord_time, delv_time→order_no can be removed

ord_time, delv_time→user_id
ord_time+ = { ord_time, delv_time, delv_add, user_id, rest_id, order_no} => ord_time+ contains user_id
ord_time, delv_time→user_id can be removed

Similarly, ord_time, delv_time→rest_id and ord_time, delv_time→delv_add can be removed
emp_id, delv_time→order_no
emp_id+ = { emp_id, user_id, rest_id, ord_time, delv_add, delv_time} => emp_id+ does not contain order_no
emp_id, delv_time→order_no cannot be removed

Similarly, emp_id, delv_time→user_id, emp_id, delv_time→rest_id, emp_id, delv_time→ord_time, emp_id, delv_time→delv_add cannot be removed

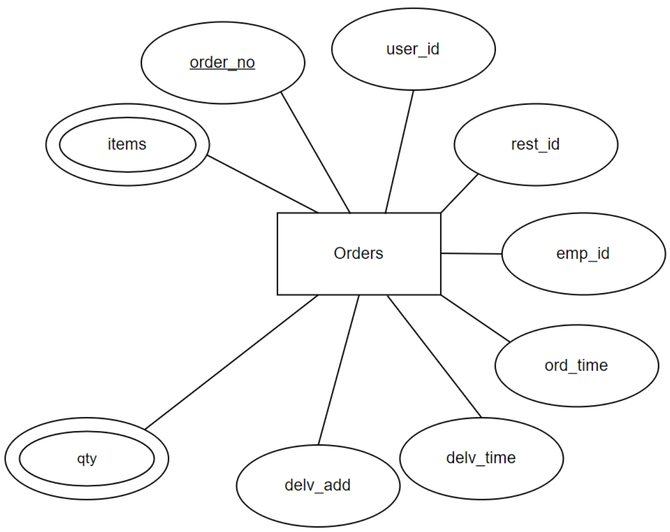
F2 = { order_no→ord_time, emp_id, delv_time→order_no, user_id, rest_id, ord_time, delv_add }
= {order_no→ord_time emp_id, delv_time→order_no, user_id, rest_id, ord_time, delv_add}

Finding superkeys:-
order_no+ = {order_no, ord_time, emp_id, delv_time, user_id, rest_id, delv_add}
order_no is a superkey

emp_id, delv_time+ = {emp_id, delv_time, ord_time, user_id, rest_id, delv_add, order_no}
emp_id, delv_time is a superkey

order_no, emp_id, delv_time are minimal superkeys and hence are candidate keys.

order_no is chosen as a primary key and emp_id, delv_time are alternate keys.



R6: PAYMENTS (payment_id, order_no, bill_total)

Trivial: payment_id, order_no→ payment_id, order_no bill_total→order_no, payment_id, bill_total→ payment_id

Time-dependent: None

The functional dependencies set F contains {payment_id→order_no, bill_total, order_no→ payment_id, bill_total, payment_id, bill_total → order_no, order_no, bill_total → payment_id}

The payment in the DB can be uniquely identified by its payment_id and/or its order_no. Hence, each of these attributes can functionally determine the other attributes.

Step 1: By decomposition,

F1 = { payment_id→order_no, payment_id→bill_total, order_no→payment_id, order_no→bill_total, payment_id, bill_total→order_no, order_no, bill_total→payment_id}

Step 2:

payment_id, bill_total→order_no

Checking payment_id→order_no

payment_id+ = {payment_id, order_no, bill_total}

bill_total is extraneous

order_no, bill_total→payment_id

Checking order_no→payment_id

order_no+ = {order_no, payment_id, bill_total}

bill_total is extraneous

F2 = { payment_id→order_no, payment_id→bill_total, order_no→payment_id, order_no→bill_total}

Step 3: Removing Redundant FDs

payment_id→order_no

payment_id+ = {payment_id, bill_total}

payment_id+ does not contain order_no

Therefore, payment_id→order_no can't be removed payment_id → bill_total

payment_id+ = {payment_id, order_no, bill_total}

payment_id+ contains bill_total

Therefore, payment_id→bill_total can be removed

order_no→payment_id

order_no+ = {order_no, bill_total}

order_no+ does not contain payment_id

Therefore, order_no→payment_id can't be removed

order_no→bill_total

order_no+ = {order_no, payment_id}

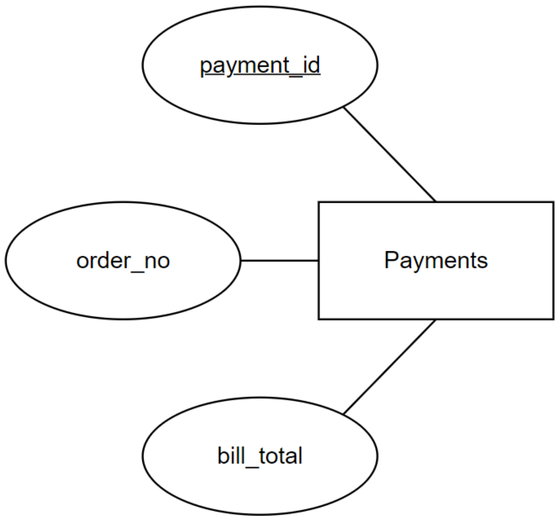
order_no+ does not contain bill_total

Therefore, order_no→bill_total can't be removed

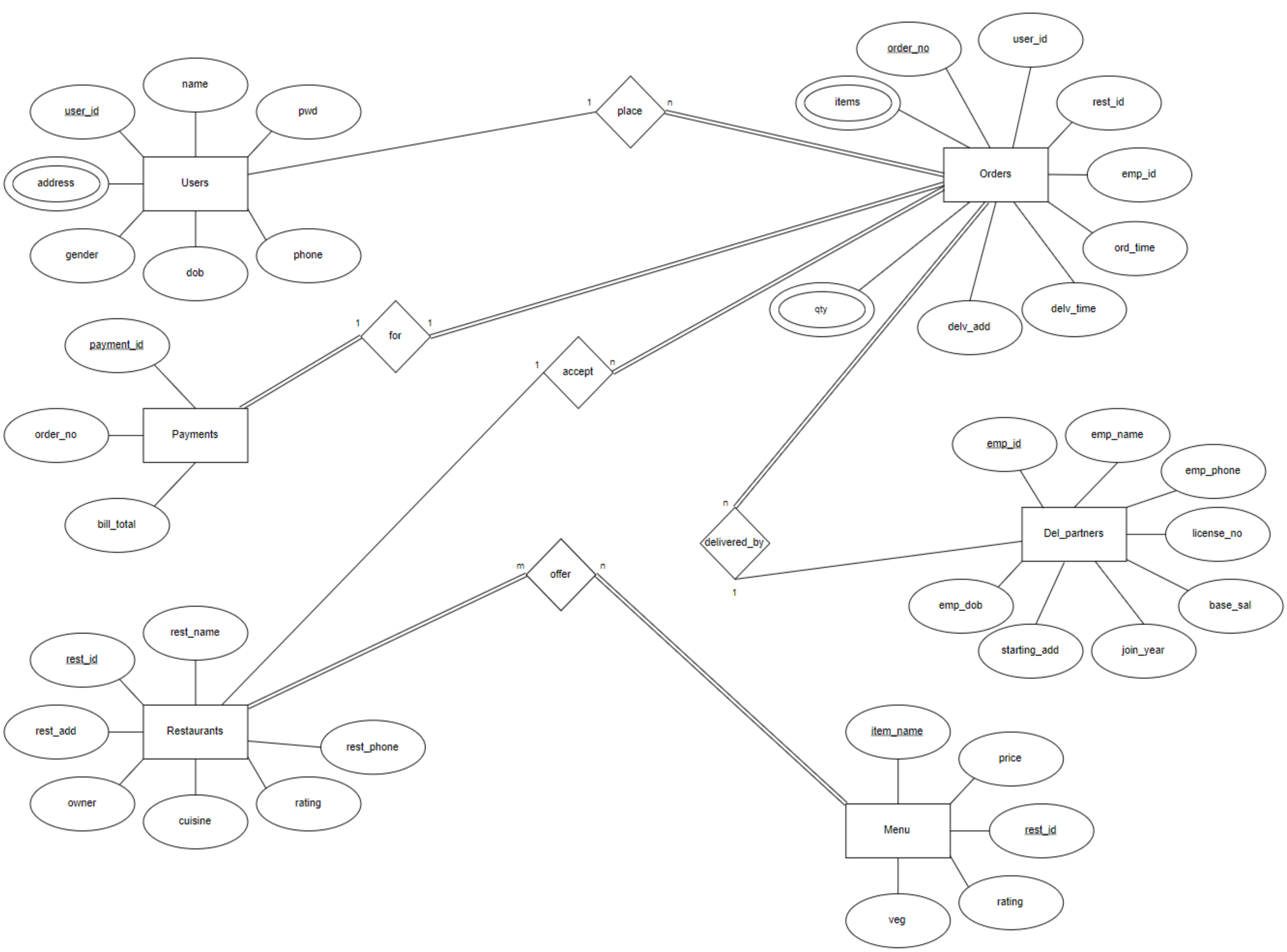
payment_id+ = {payment_id, order_no, bill_total}

order_no+= {order_no, payment_id, bill_total}

Here payment_id & order_no are Superkeys and also they are minimal. Such that, payment_id is chosen as the primary key and order_no as an alternate key.



ENTITY RELATIONSHIP DIAGRAM:



ER MODEL TO RELATIONAL MODEL CONVERSION:

Users Entity:
The users entity contains a multi-valued attribute, user_add which may contain multiple addresses added by a single user. Hence, while mapping the entity to a relation, we create a separate relation called address with the attribute user_add, and a foreign key user_id taken from the superior entity, users.

Orders Entity:
The orders entity contains two multi-valued attributes, items and qty which contain the names of the items ordered and their quantities respectively. Hence, while mapping the entity to a relation, we create a separate relation called order_details with the attributes item_name and qty, and a foreign key order_no taken from the superior entity, orders.

Users place Orders Relationship:
The users relation corresponds to the users entity (partial participation) with the primary key as user_id. Similarly, the orders relation corresponds to the orders entity (total participation) with the primary key as order_no. The relationship “users place orders” has a cardinality ratio 1:n, hence, the primary key user_id of users relation becomes the foreign key of the orders relation.

Orders delivered_by Del_partners Relationship:
The orders relation corresponds to the orders entity (total participation) with the primary key as order_no. Similarly, the del_partners relation corresponds to the del_partners entity (partial participation) with the primary key as emp_id. The relationship “orders delivered_by del_partners” has a cardinality ratio n:1, hence, the primary key emp_id of del_partners relation becomes the foreign key of the orders relation.

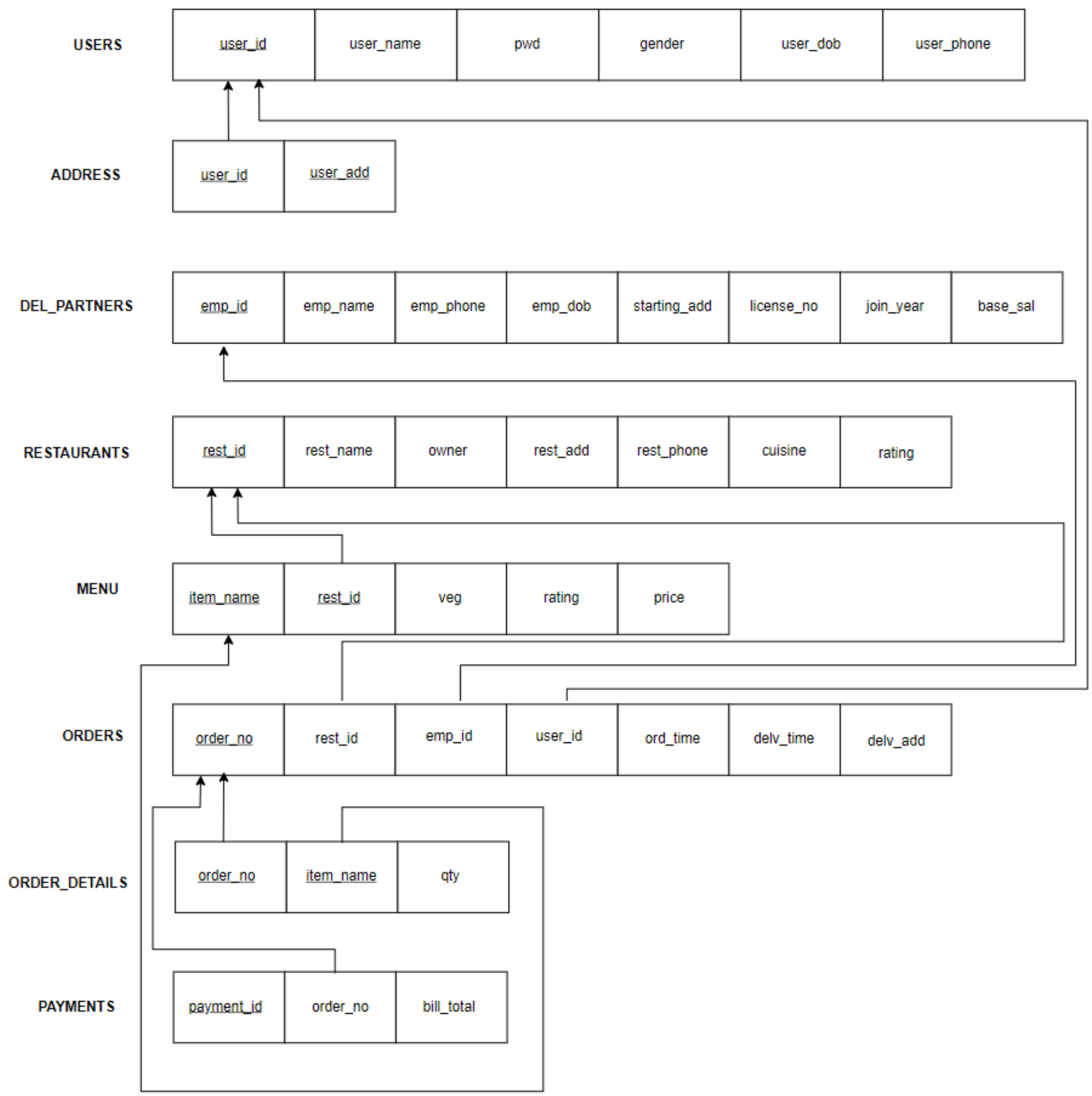
Restaurants accept Orders Relationship:
The restaurants relation corresponds to the restaurants entity (partial participation) with the primary key as rest_id. Similarly, the orders relation corresponds to the orders entity (total participation) with the primary key as order_no. The relationship “restaurants accept orders” has a cardinality ratio 1:n, hence, the primary key rest_id of restaurants relation becomes the foreign key of the orders relation.

Restaurants offer Menu Relationship:
The restaurants relation corresponds to the restaurants entity (total participation) with the primary key as rest_id. Similarly, the menu relation corresponds to the menu entity (total participation) with the primary key as item_name. The relationship “restaurants offer menu” has a cardinality ratio 1:n, hence, the primary key rest_id of restaurants relation becomes the foreign key of the menu relation.

Payments for Orders Relationship:
The payments relation corresponds to the payments entity with the primary_key as payment_id. Similarly, the orders relation corresponds to the orders entity (partial participation) with the

primary key as order_no. The relationship “payments for orders” has a cardinality ratio 1:1, hence, the primary key order_no of the partial side (orders entity) becomes the foreign key of the total side (payments entity).

SCHEMA DIAGRAM:



SSN College Of Engineering

Kalavakkam – 603110

Department of Computer Science and Engineering

UCS2404 Database Management Systems

Assignment 2

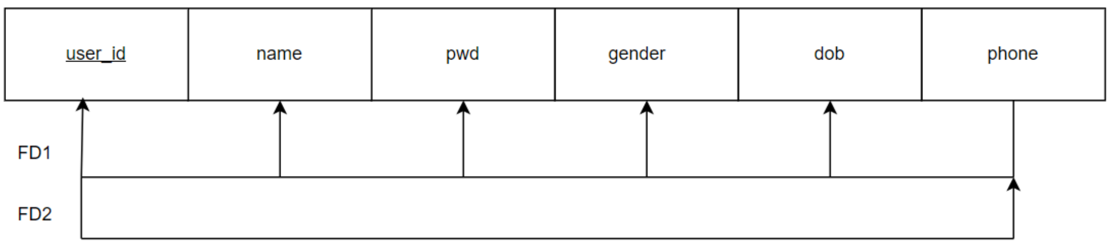
Food Delivery Management System

Abhishek Reddy Lingareddy (3122225001003)

Darrshan Hariharan (3122225001022)

Deepak Chandar S (3122225001023)

R1: Users



Canonical Cover: {user_id->phone,
phone -> user_id, name, pwd, gender, dob}

1NF:

The domains of all the attributes of the relation are neither multi-valued nor composite.
Therefore, the users relation is in 1NF.

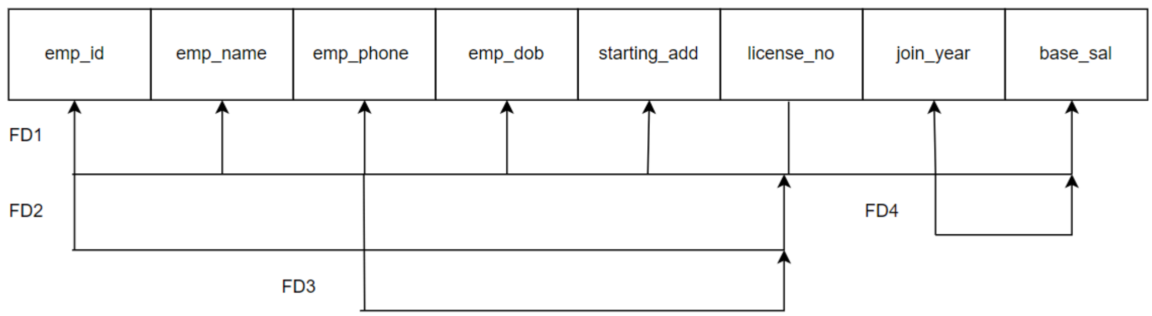
2NF:

User_id and phone are candidate keys of the relation and thus there are no partial functional dependencies. FD1 and FD2 are full functional dependencies. Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the users relation is in 2NF.

3NF:

In the users relation, user_id -> phone and phone -> user_id, name, pwd, gender, dob. The attribute phone is a candidate key.
For all FDs X− > A in R, then either:
(a) X is a superkey of R, or
(b) A is a prime attribute of R
Therefore, the users relation is in 3NF.

R2: DEL_PARTNERS



Canonical Cover:

{license_no -> emp_id, emp_name, emp_phone, emp_dob, starting_add, license_no, join_year, base_sal,
emp_id -> license_no,
emp_phone ->license_no, join_year -> base_sal}

1NF:

The domains of all the attributes of the relation are neither multi-valued nor composite.
Therefore, the del_partners relation is in 1NF.

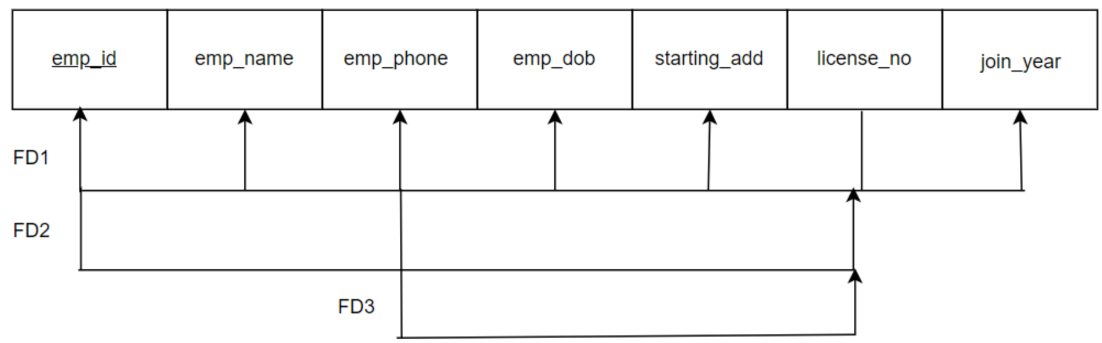
2NF:

Emp_id, emp_phone and license_no are candidate keys of the relation.
Therefore, FD1, FD2, FD3 are all full functional dependencies.
Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the del_partners relation is in 2NF.

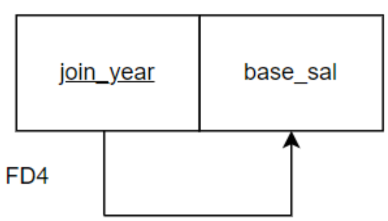
3NF:

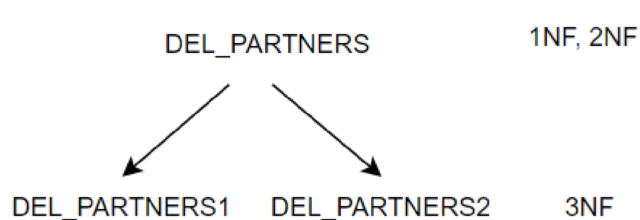
In the del_partners relation, FD1 and FD4 create a transitive functional dependency and the attribute join_year is not a candidate key of the relation. Therefore, the relation is not in 3NF.
Hence, the relation is decomposed into two relations, del_partners1 and del_partners2 with join_year as the primary key of del_partners2 and foreign key of del_partners1.

DEL_PARTNERS1:

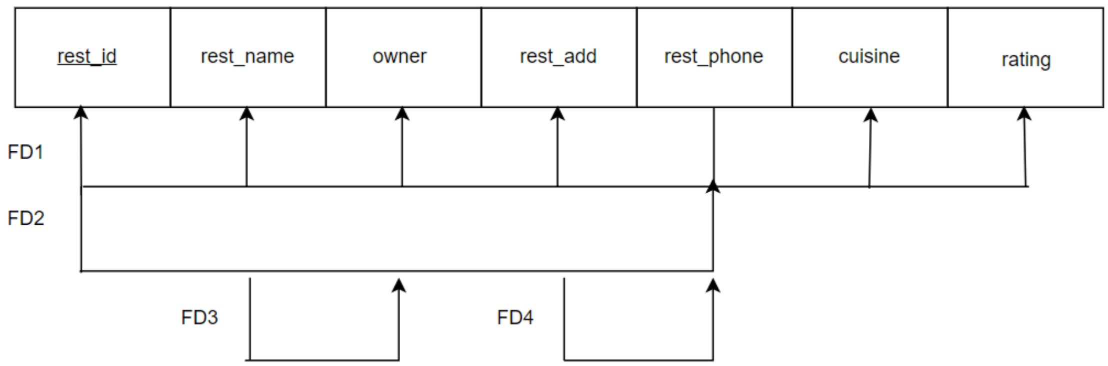


DEL_PARTNERS2:





R3: RESTAURANTS



Candidate Keys:
{rest_phone -> rest_id, rest_name, owner, rest_add, cuisine, rating,
rest_id -> rest_phone,
rest_add -> rest_phone, rest_name -> owner}

1NF:

The domains of all the attributes of the relation are neither multi-valued nor composite.
Therefore, the restaurants relation is in 1NF.

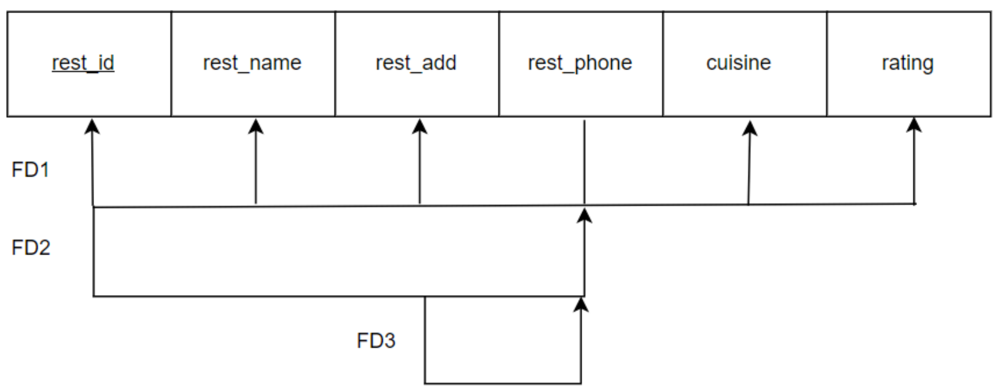
2NF:

Rest_id, rest_add and rest_phone are the candidate keys of the relation.
Therefore, FD1, FD2 AND FD3 are full functional dependencies.
Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the del_partners relation is in 2NF.

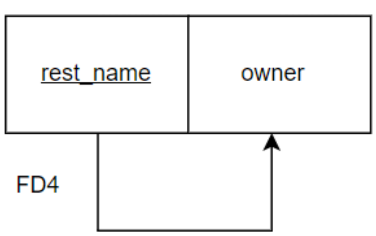
3NF:

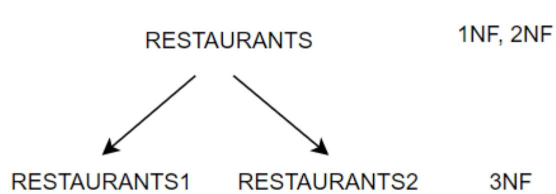
In restaurants relation, FD1 and FD4 create a transitive functional dependency and the attribute rest_name is not a candidate key of the relation. Therefore, the relation is not in 3NF.
Hence, the relation is decomposed into two relations, restraunts1 and restraunts2 with rest_name as the primary key of restraunts2 and foreign key of restraunts1.

RESTRAUNTS1:

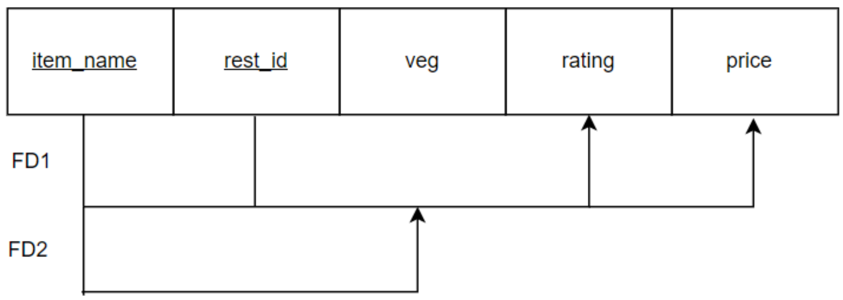


RESTRAUNTS2:





R4: MENU



Candidate Keys:
{item_name, rest_id -> rating, price,
item_name ->veg}

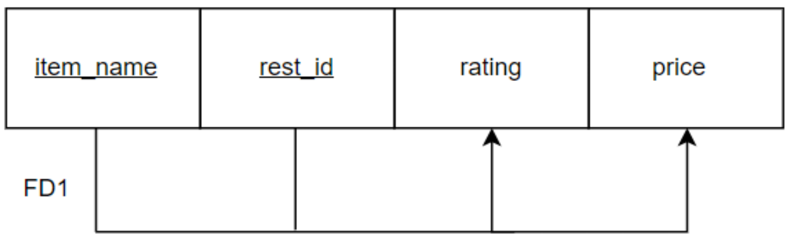
1NF:

The domains of all the attributes of the relation are neither multi-valued nor composite.
Therefore, the menu relation is in 1NF.

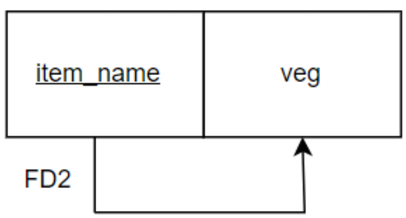
2NF:

Item_id and rest_id is the composite key of the relation. FD1 is a full functional dependency whereas FD2 is a partial functional dependency.
Hence, the relation is not in 2NF.
This is resolved by decomposing the relation into two relations, menu1 and menu2 with item_name as the primary key of the relation menu2.

MENU1:

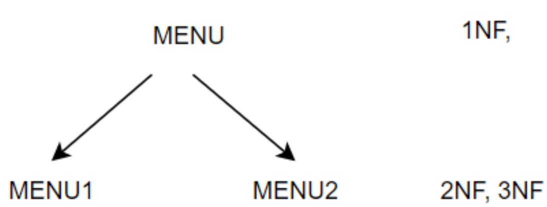


MENU2:

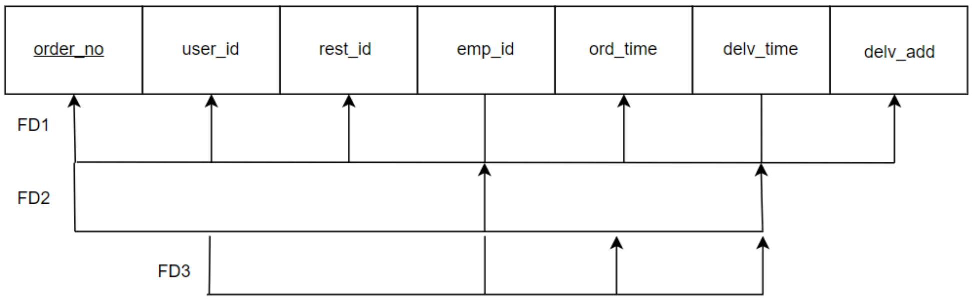


3NF:

Both the relations menu1 and menu2 do not have any transitive functional dependencies, that is, for all FDs X− > A in R, then either:
(a) X is a superkey of R, or
(b) A is a prime attribute of R
Hence they are in 3NF.



R5: ORDERS



Canonical cover: {order_no -> emp_id, delv_time, (user_id, ord_time) -> emp_id, delv_time, (emp_id, delv_time) -> order_no, user_id, rest_id, ord_time, delv_add}

1NF:

The domains of all the attributes are neither multi-valued nor composite. Therefore, the orders relation is in 1NF.

2NF:

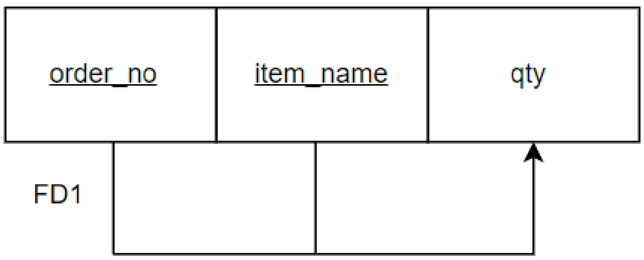
order_no, (user_id, ord_time), and (emp_id, delv_time) are candidate keys of the relation and thus, there are no partial functional dependencies. FD1, FD2 and FD3 are full functional dependencies. Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the orders relation is in 2NF.

3NF:

In the orders relation, order_no, (user_id, ord_time), and (emp_id, delv_time) are the superkeys. For all FDs X->A belonging to the relation R:
(a) X is a superkey of R, or
(b) A is a prime attribute of R

Therefore, the orders relation is in 3NF.

R6: ORDER_DETAILS



Canonical cover: {(order_no, item_name)->qty}

1NF:

The domains of all the attributes are neither multi-valued nor composite.
Therefore, the order_details relation is in 1NF.

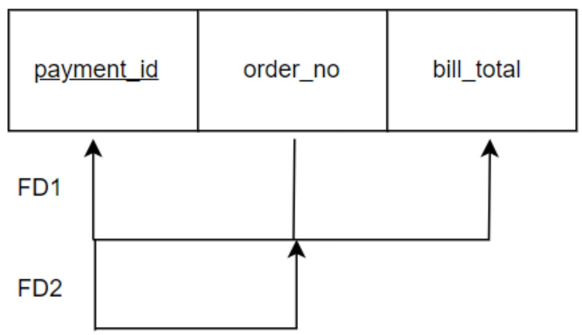
2NF:

(order_no, item_name) is the composite key of the relation and thus, there are no partial functional dependencies. FD1 is a full functional dependency. Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the order_details relation is in 2NF.

3NF:

In the order_details relation, (order_no, item_name) is the superkey. For all FDs X->A belonging to the relation R:
(a) X is a superkey of R, or
(b) A is a prime attribute of R
Therefore, the order_details relation is in 3NF.

R7: PAYMENTS



Canonical cover: {payment_id->order_no, order_no->payment_id, bill_total}

1NF:

The domains of all the attributes are neither multi-valued nor composite.
Therefore, the payments relation is in 1NF.

2NF:

payment_id and order_no are the candidate keys of the relation and thus, there are no partial functional dependencies. FD1 and FD2 are full functional dependencies. Every non-prime attribute is fully functionally dependent on every key of the relation.
Therefore, the payments relation is in 2NF.

3NF:

In the payments relation, payment_id, order_no are the superkeys. For all FDs X->A belonging to the relation R:
(a) X is a superkey of R, or
(b) A is a prime attribute of R

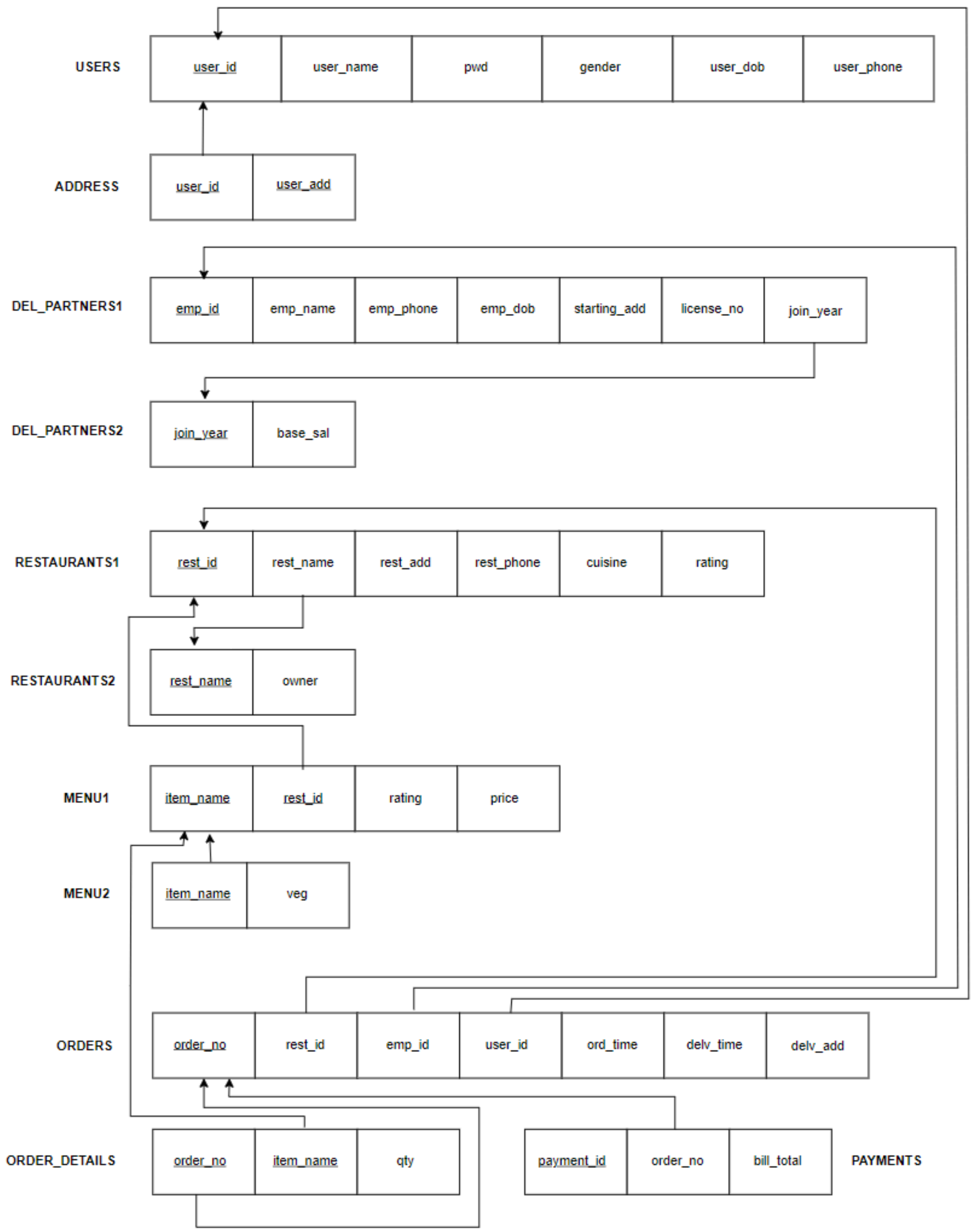
Therefore, the payments relation is in 3NF.

R8: ADDRESS:

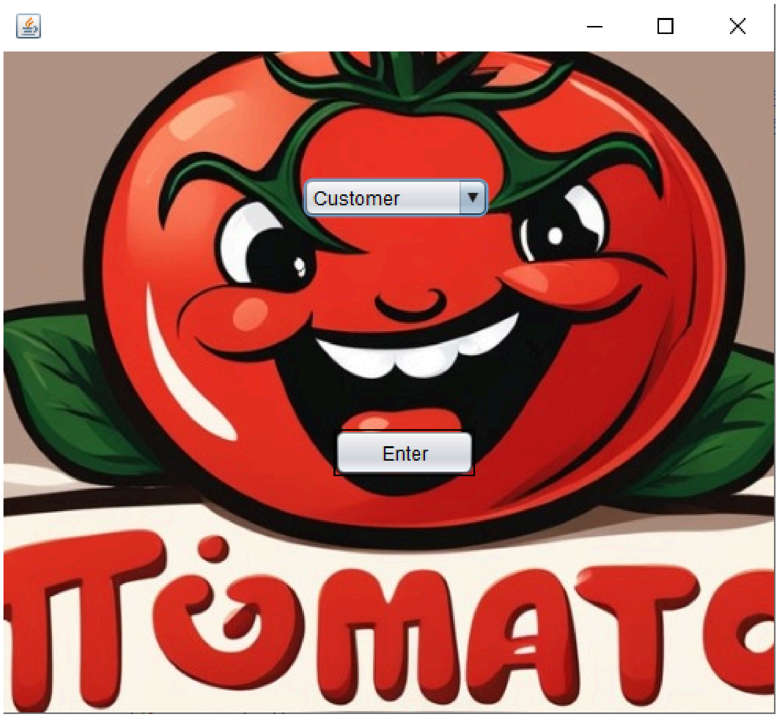
user_id	user_add

This relation satisfies all the three normal forms.

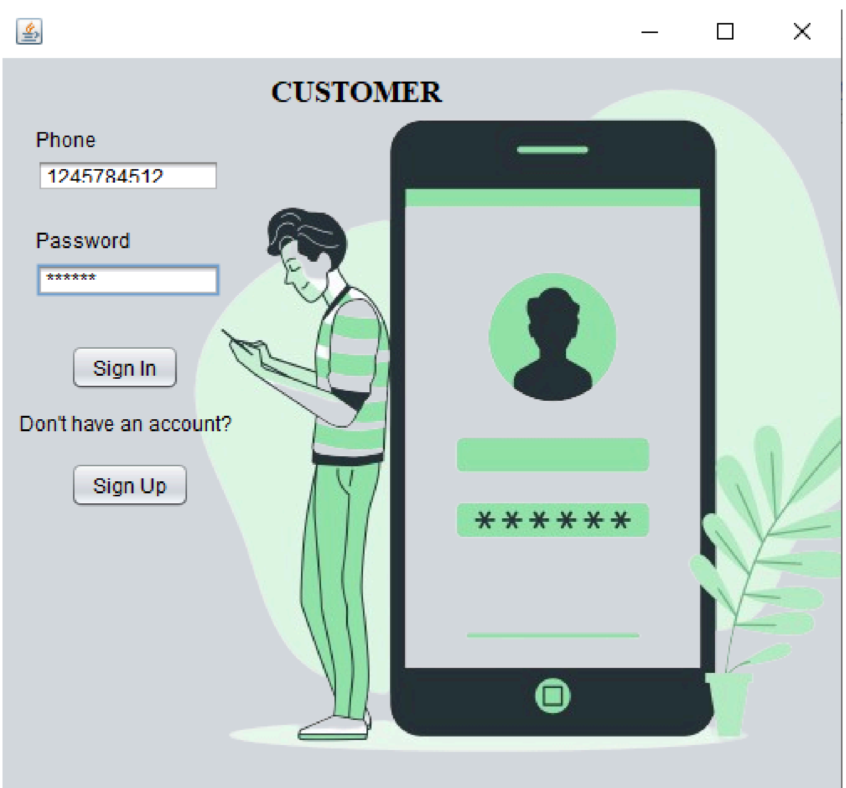
SCHEMA DIAGRAM AFTER NORMALISATION:



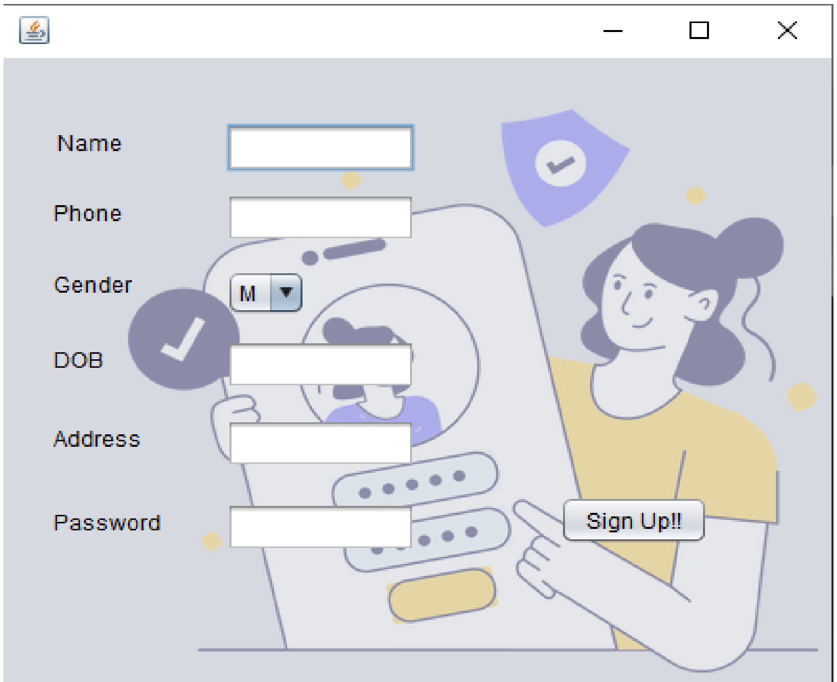
CUSTOMERS:

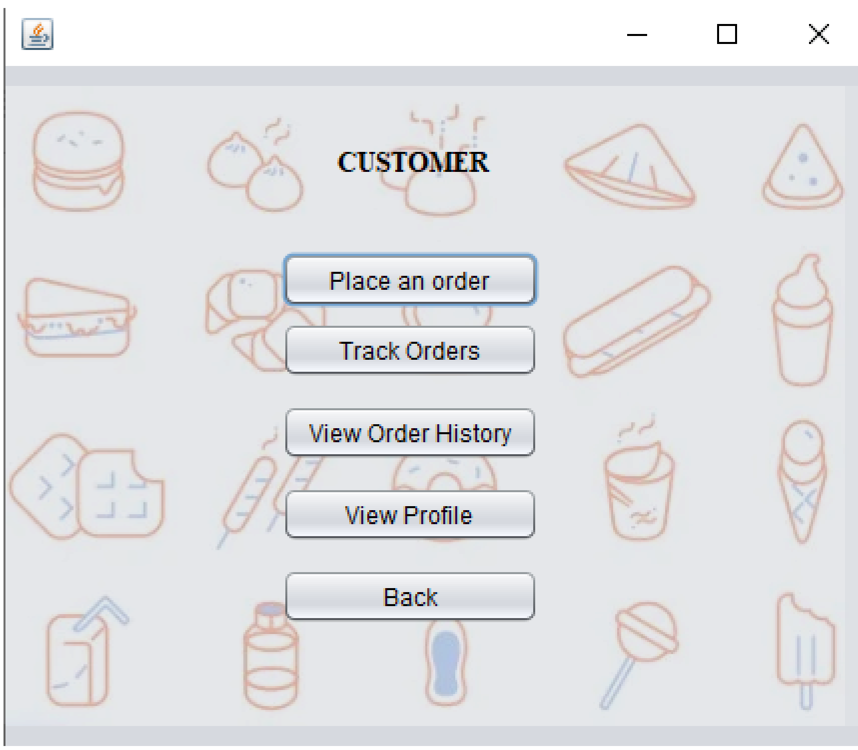


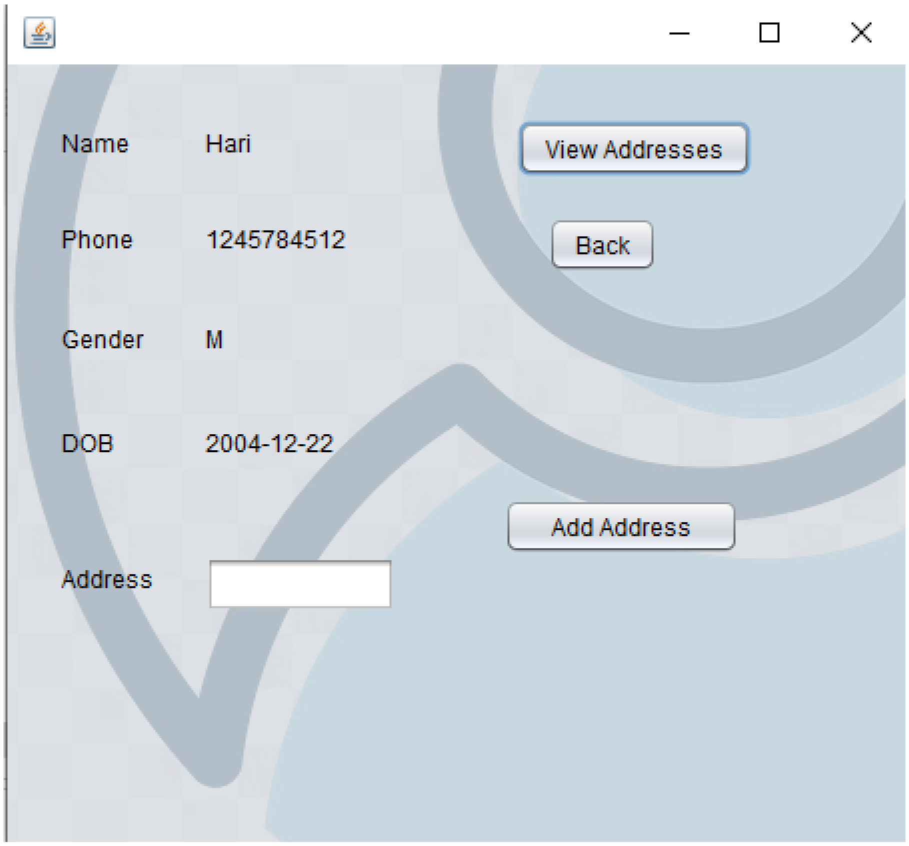
Customer
Enter

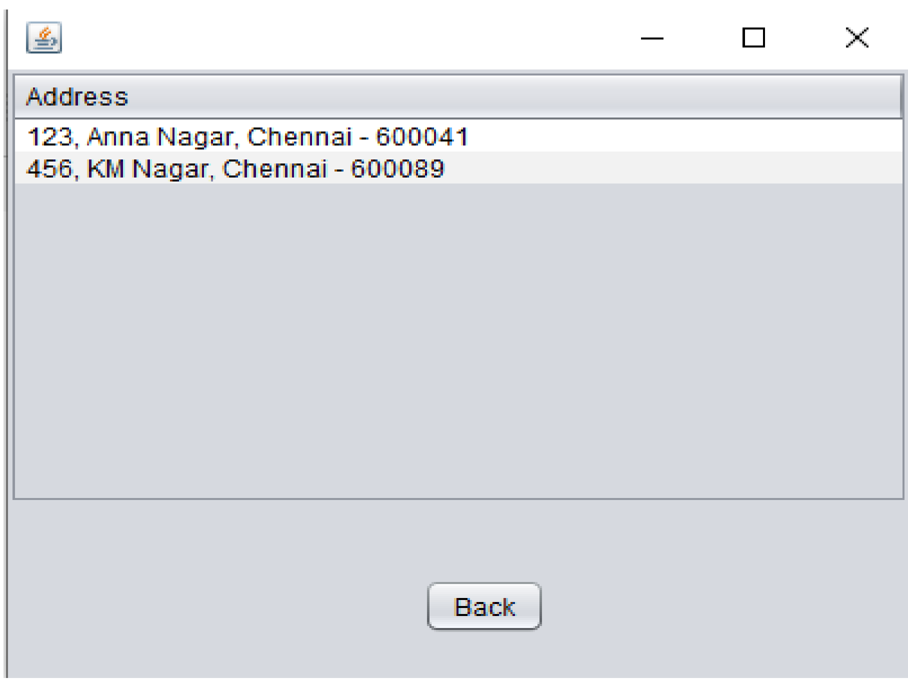


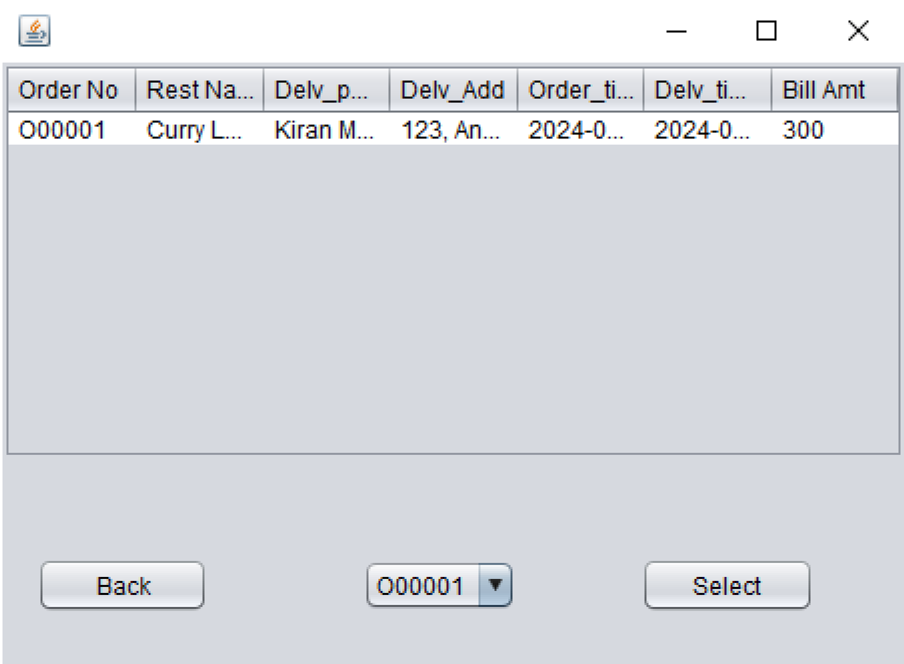
CUSTOMER
Phone
1245784512
Password
******
Sign In
Don't have an account?
Sign Up

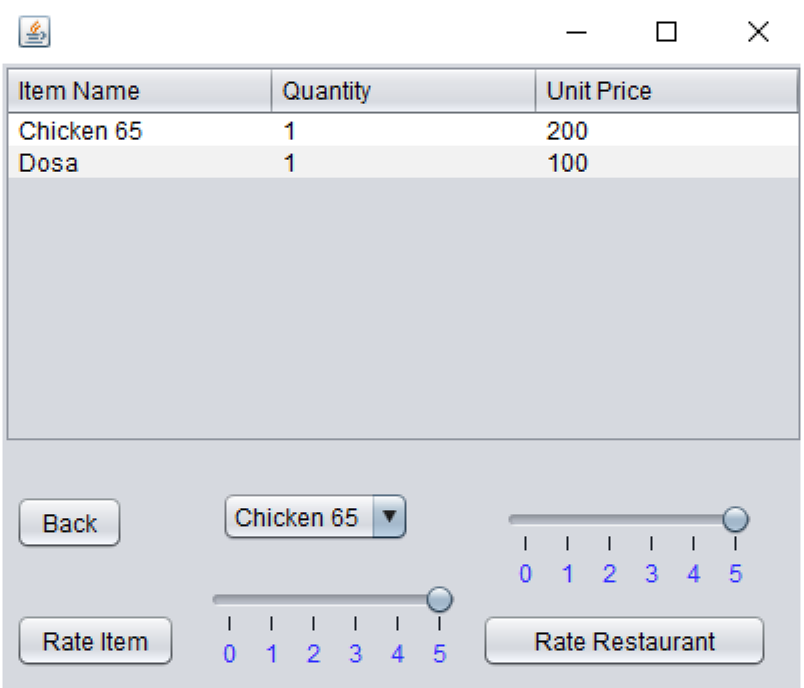

Name
Phone
Gender
M
DOB
Address
Password
Sign Up!!

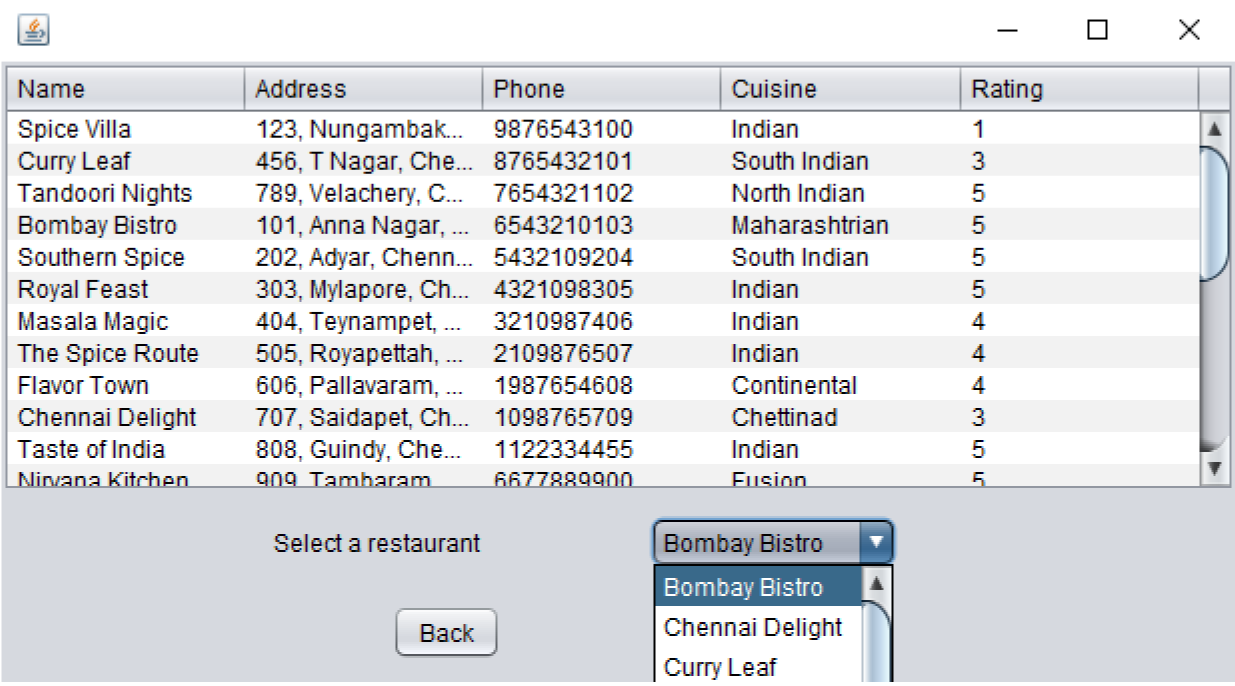

CUSTOMER
Place an order
Track Orders
View Order History
View Profile
Back


Name
Hari
View Addresses
Phone
1245784512
Back
Gender
M
DOB
2004-12-22
Add Address
Address

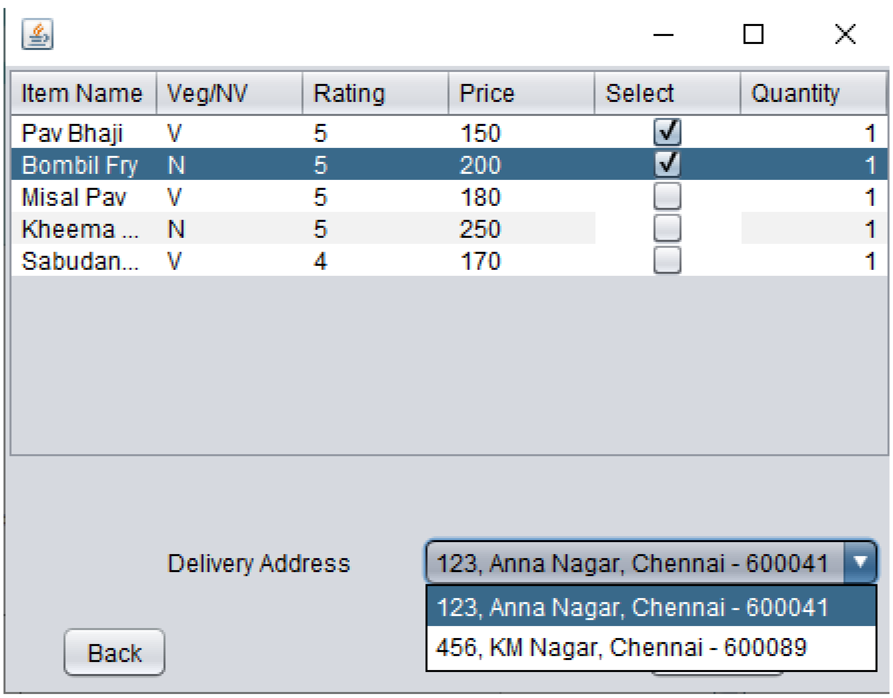

Address
123, Anna Nagar, Chennai - 600041
456, KM Nagar, Chennai - 600089
Back


Order No
Rest Na...
Delv_p...
Delv_Add
Order_ti...
Delv_ti...
Bill Amt
O00001
Curry L...
Kiran M...
123, An...
2024-0...
2024-0...
300
Back
O00001
Select

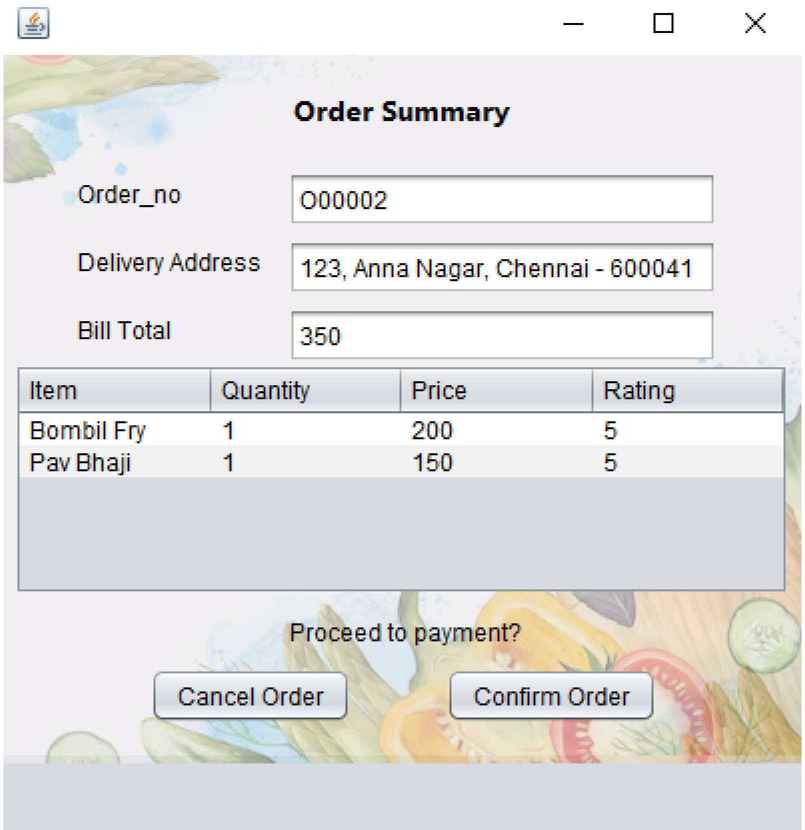

Item Name
Quantity
Unit Price
Chicken 65
1
200
Dosa
1
100
Back
Chicken 65
0 1 2 3 4 5
Rate Item
0 1 2 3 4 5
Rate Restaurant

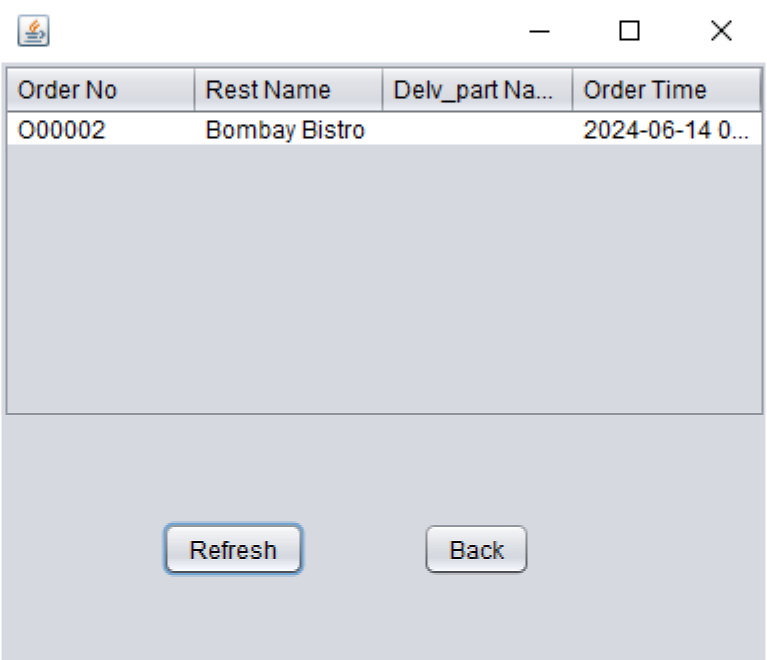


Name	Address	Phone	Cuisine	Rating
Spice Villa	123, Nungambak...	9876543100	Indian	1
Curry Leaf	456, T Nagar, Che...	8765432101	South Indian	3
Tandoori Nights	789, Velachery, C...	7654321102	North Indian	5
Bombay Bistro	101, Anna Nagar, ...	6543210103	Maharashtrian	5
Southern Spice	202, Adyar, Chenn...	5432109204	South Indian	5
Royal Feast	303, Mylapore, Ch...	4321098305	Indian	5
Masala Magic	404, Teynampet, ...	3210987406	Indian	4
The Spice Route	505, Royapettah, ...	2109876507	Indian	4
Flavor Town	606, Pallavaram, ...	1987654608	Continental	4
Chennai Delight	707, Saidapet, Ch...	1098765709	Chettinad	3
Taste of India	808, Guindy, Che...	1122334455	Indian	5
Nirvana Kitchen	909, Tambaram, ...	6677889900	Fusion	5

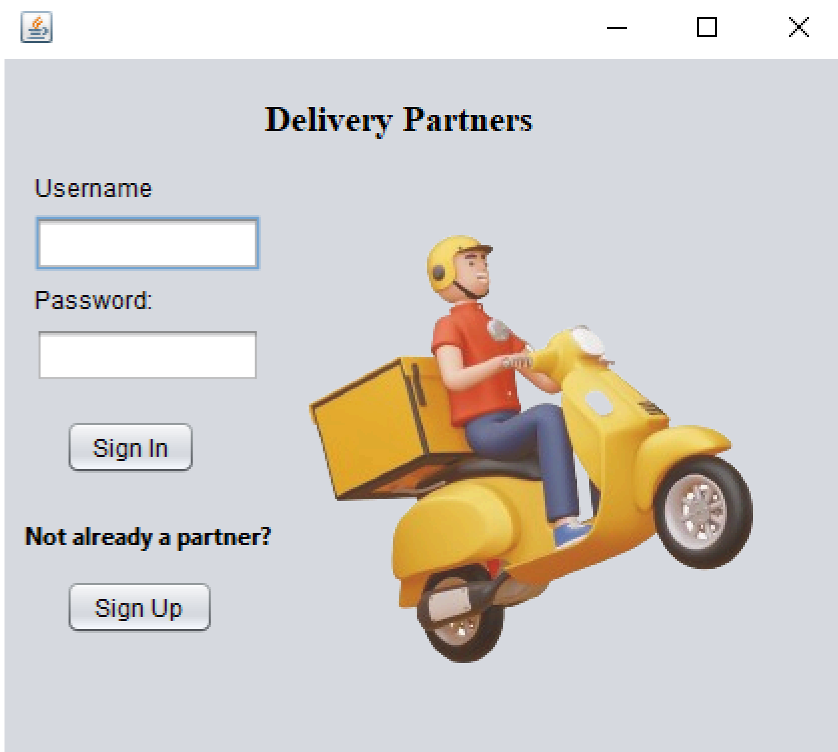


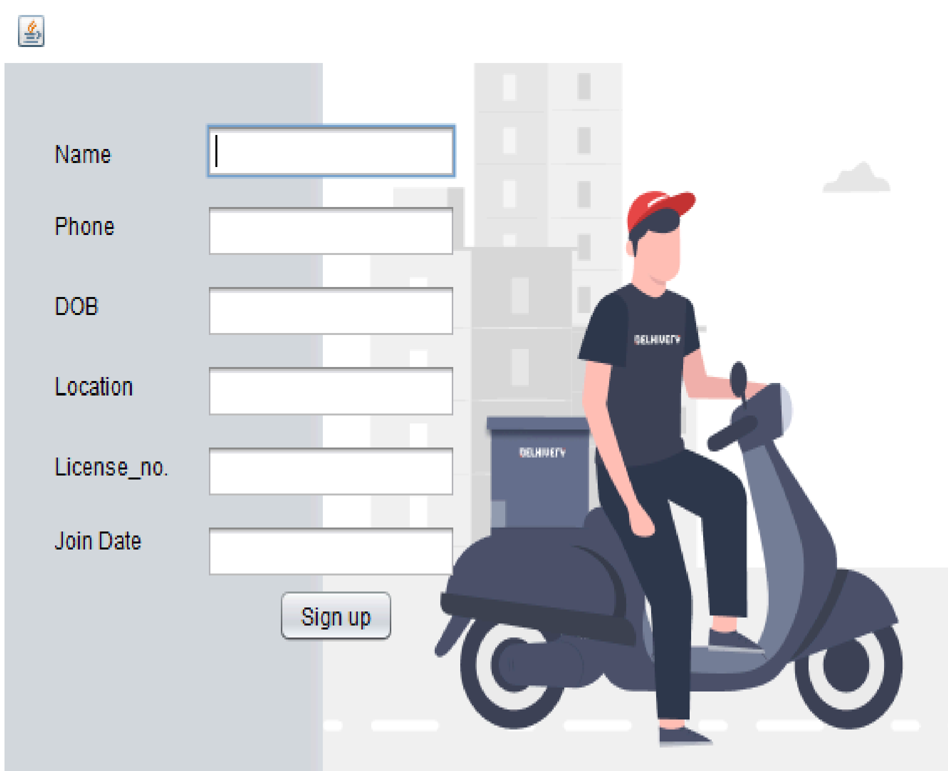
Item Name	Veg/NV	Rating	Price	Select	Quantity
Pav Bhaji	V	5	150	☑	1
Bombil Fry	N	5	200	☑	1
Misal Pav	V	5	180	☐	1
Kheema ...	N	5	250	☐	1
Sabudan...	V	4	170	☐	1

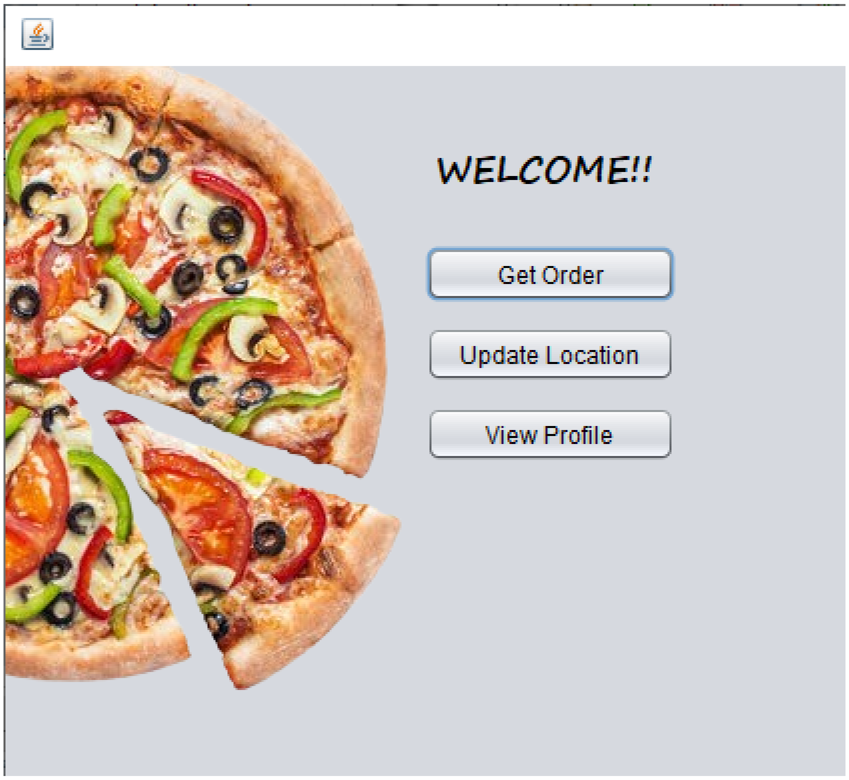

Order Summary
Order_no
O00002
Delivery Address
123, Anna Nagar, Chennai - 600041
Bill Total
350
Item | Quantity | Price | Rating
Bombil Fry | 1 | 200 | 5
Pav Bhaji | 1 | 150 | 5
Proceed to payment?
Cancel Order
Confirm Order

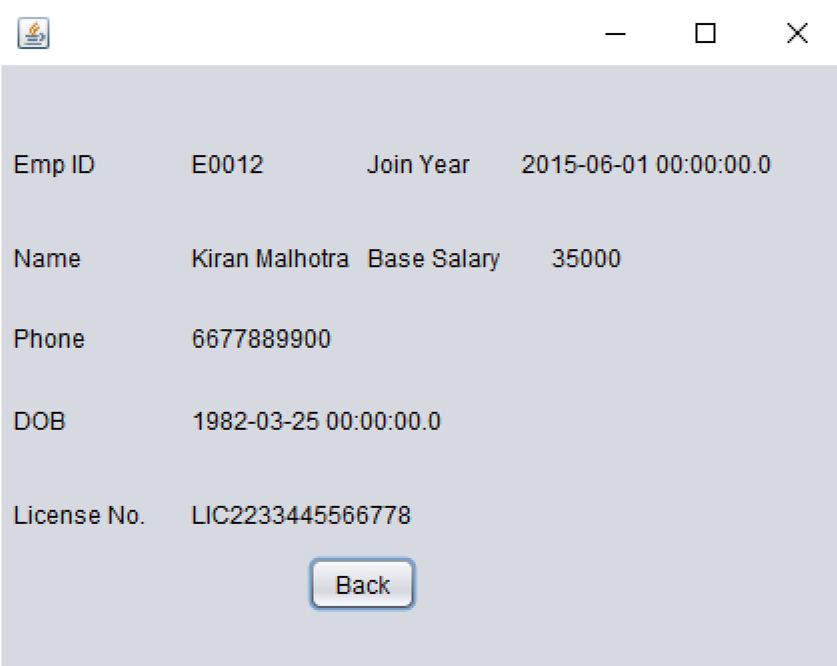

Order No | Rest Name | Delv_part Na... | Order Time
O00002 | Bombay Bistro | | 2024-06-14 0...
Refresh
Back

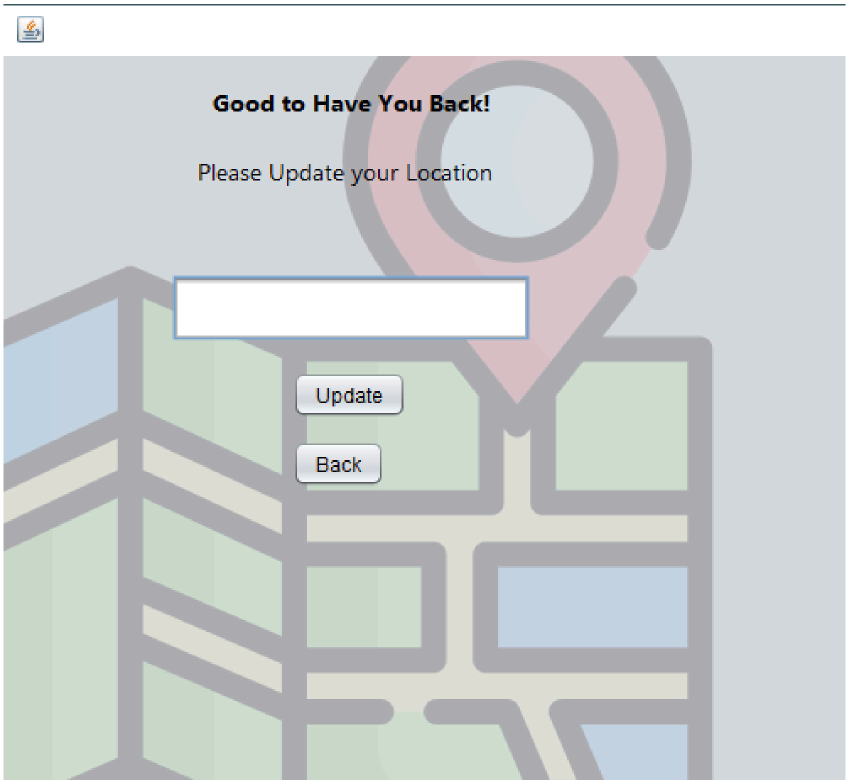
DELIVERY PARTNER:

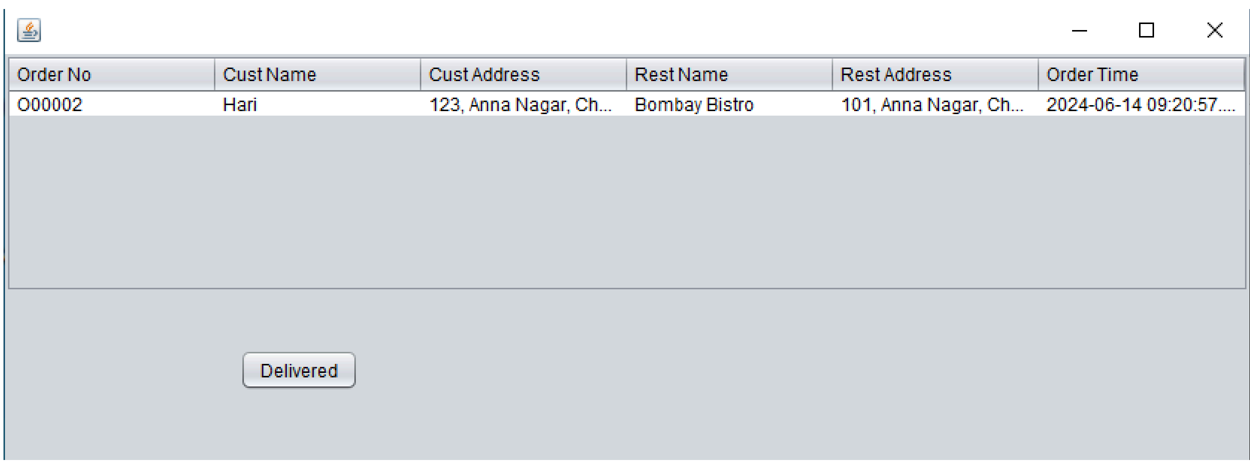




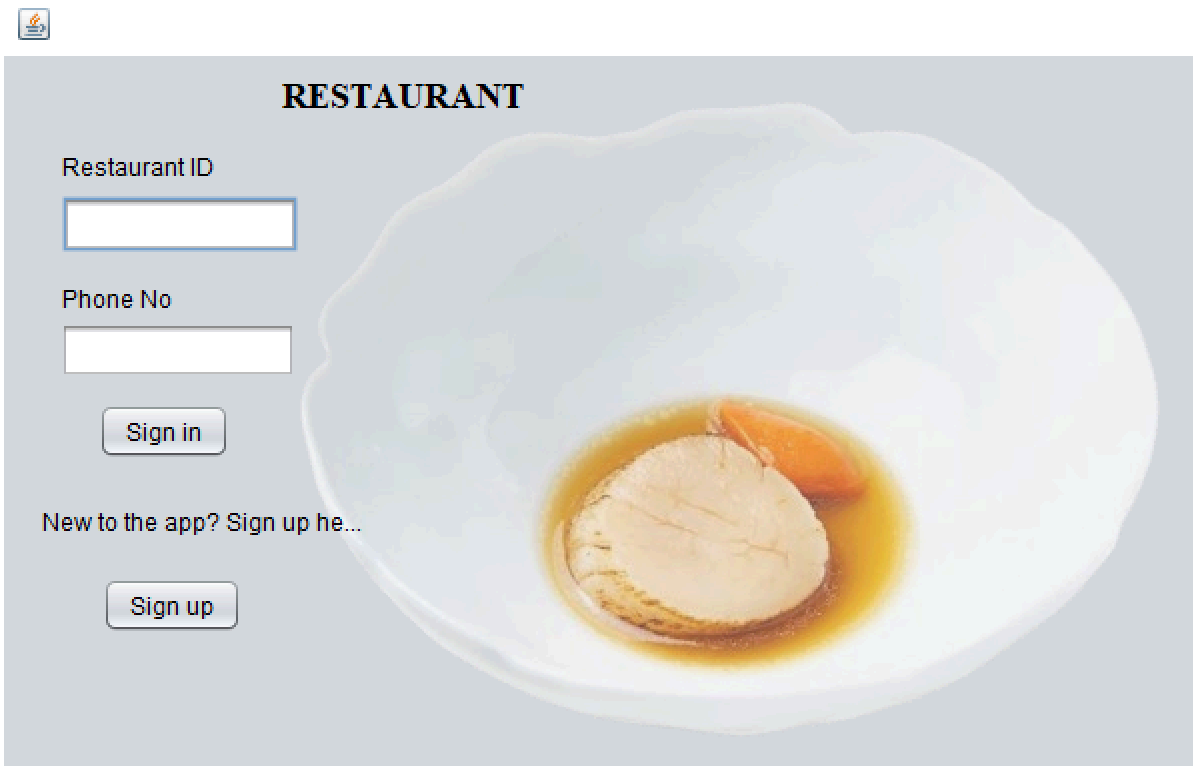

WELCOME!!
Get Order
Update Location
View Profile

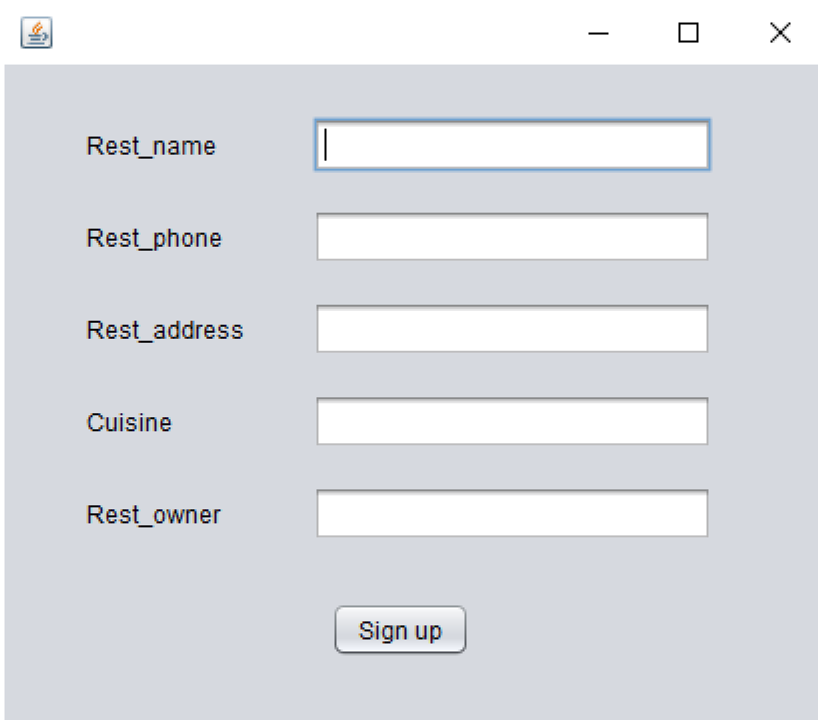

Emp ID
E0012
Join Year
2015-06-01 00:00:00.0
Name
Kiran Malhotra
Base Salary
35000
Phone
6677889900
DOB
1982-03-25 00:00:00.0
License No.
LIC2233445566778
Back

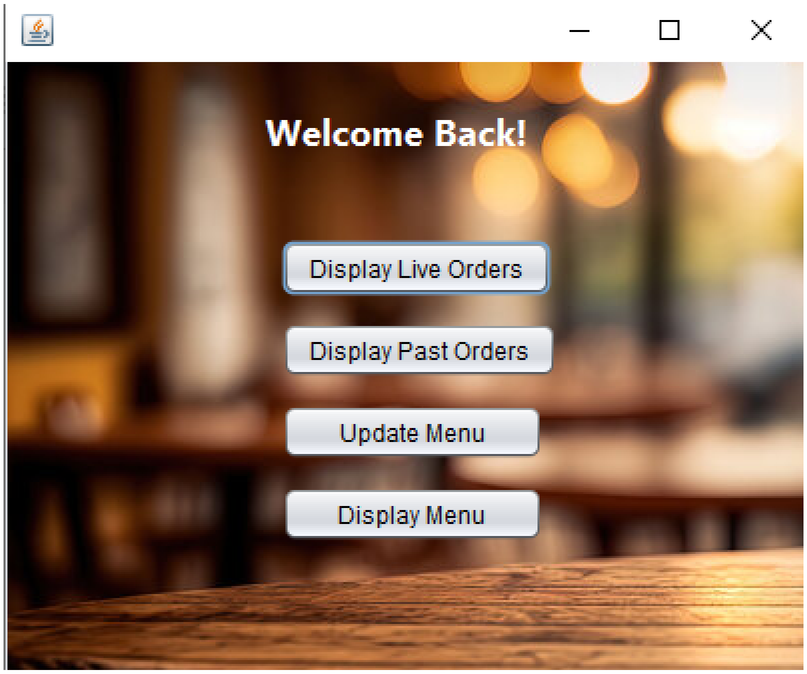

Good to Have You Back!
Please Update your Location
Update
Back

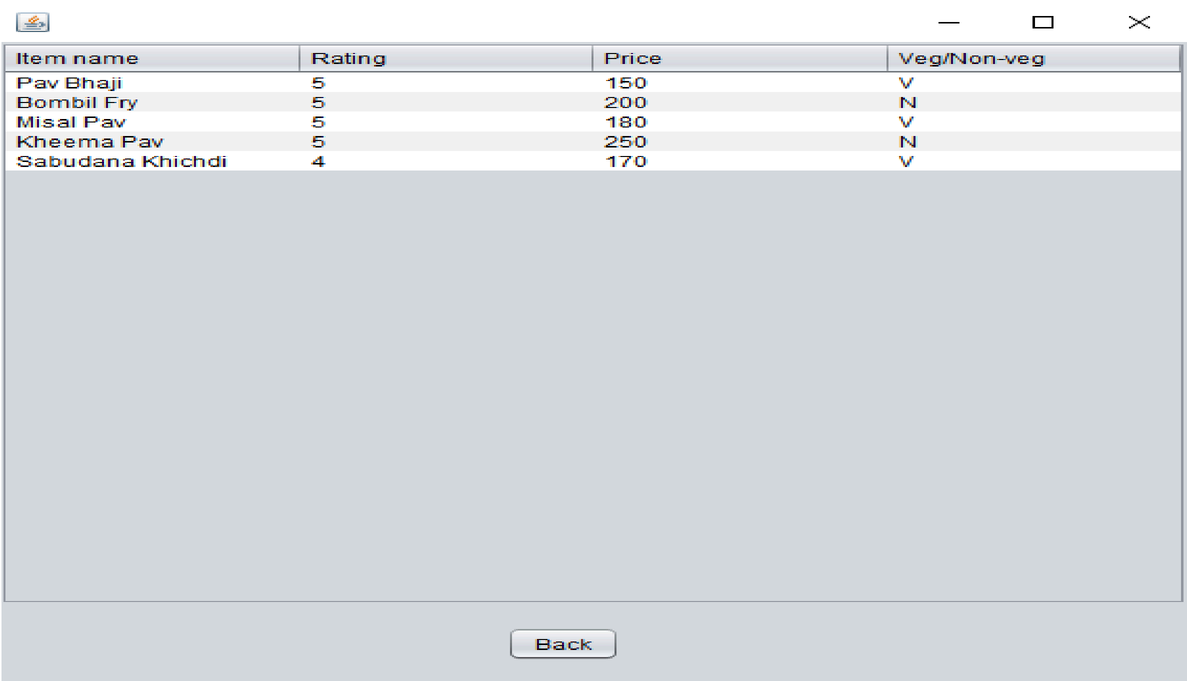

Order No
Cust Name
Cust Address
Rest Name
Rest Address
Order Time
O00002
Hari
123, Anna Nagar, Ch...
Bombay Bistro
101, Anna Nagar, Ch...
2024-06-14 09:20:57....
Delivered

RESTAURANT:

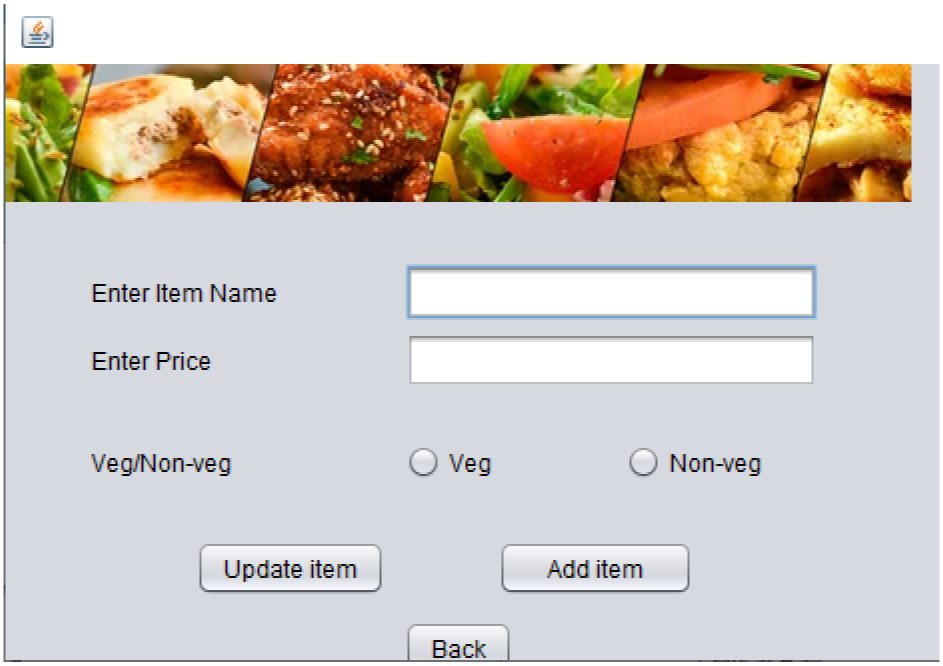

RESTAURANT
Restaurant ID
Phone No
Sign in
New to the app? Sign up he...
Sign up

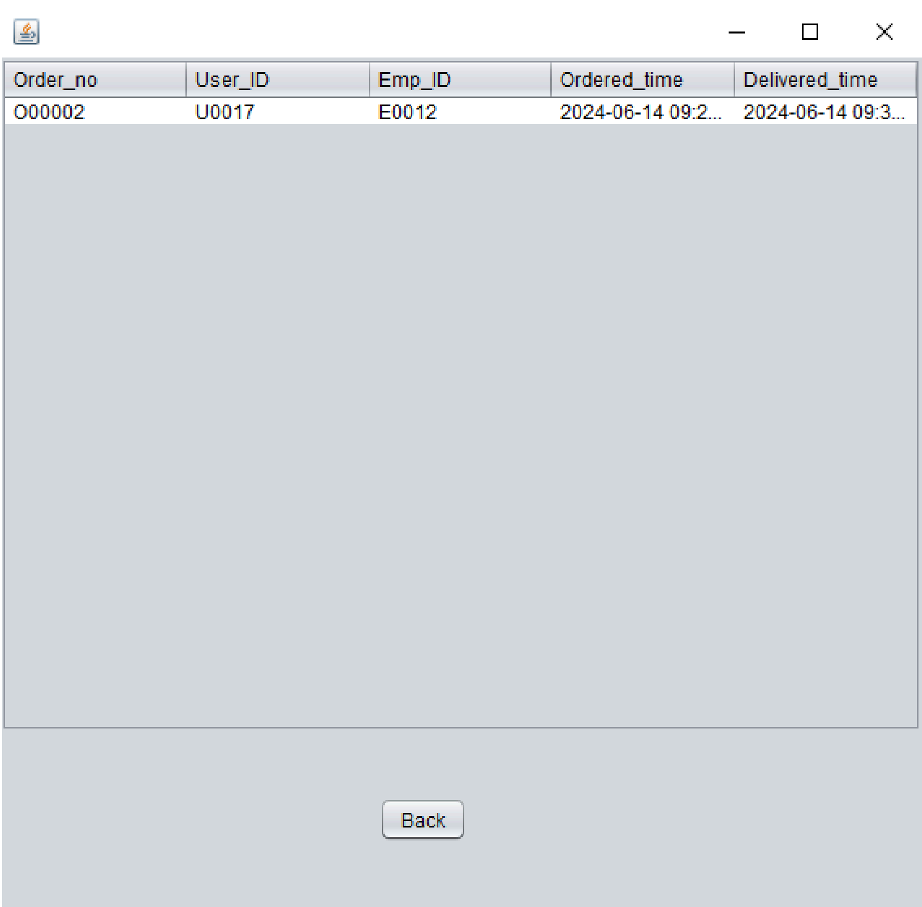

Rest_name
Rest_phone
Rest_address
Cuisine
Rest_owner
Sign up

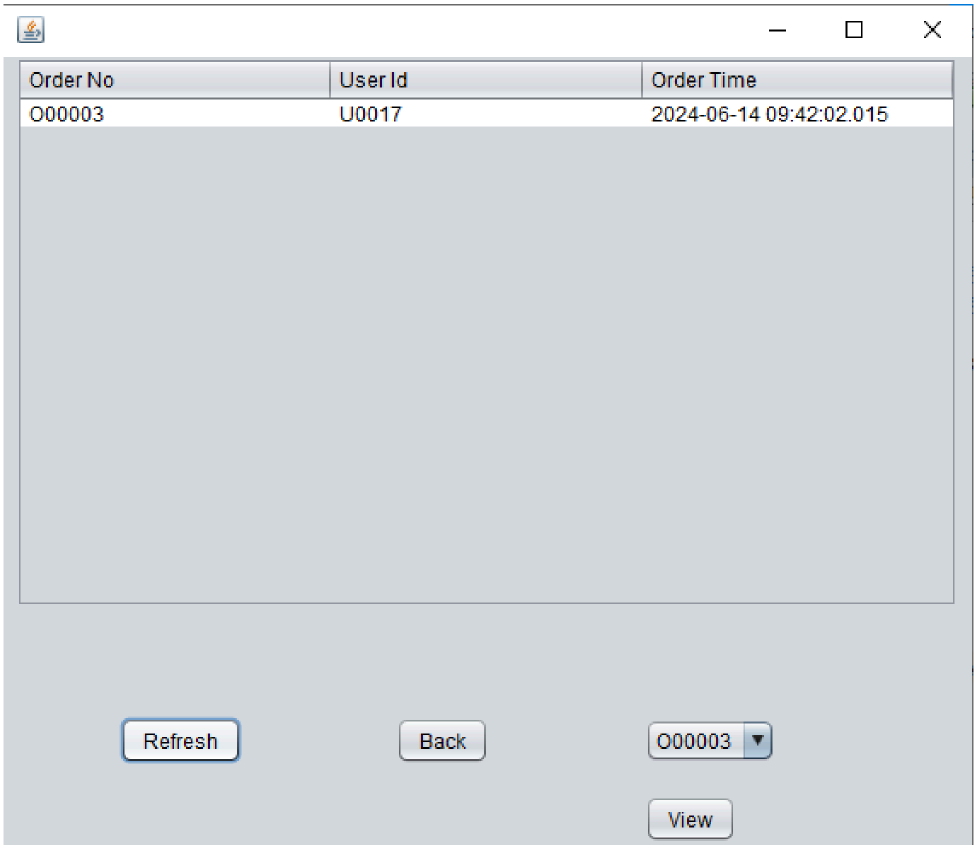


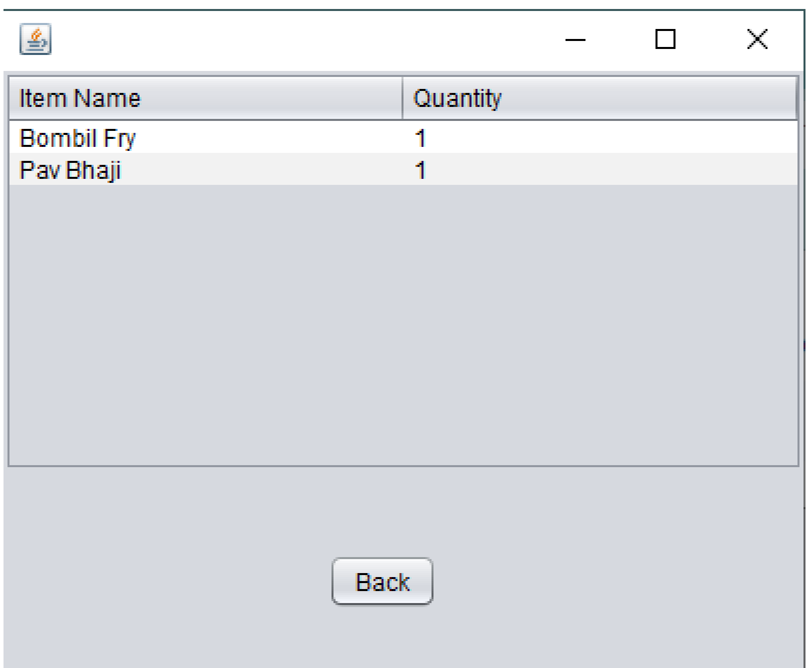


Item name	Rating	Price	Veg/Non-veg
Pav Bhaji	5	150	V
Bombil Fry	5	200	N
Misal Pav	5	180	V
Kheema Pav	5	250	N
Sabudana Khichdi	4	170	V


Enter Item Name
Enter Price
Veg/Non-veg
Veg
Non-veg
Update item
Add item
Back


Order_no
User_ID
Emp_ID
Ordered_time
Delivered_time
O00002
U0017
E0012
2024-06-14 09:2...
2024-06-14 09:3...
Back


Order No
User Id
Order Time
O00003
U0017
2024-06-14 09:42:02.015
Refresh
Back
O00003
View


Item Name
Quantity
Bombil Fry
1
Pav Bhaji
1
Back

Novelties of the Project

1. User-Friendly Interface:

- **Intuitive Design:** Developed using NetBeans JAVA Swing GUI, the interface is designed to be simple and intuitive, enabling users with minimal technical knowledge to easily place orders and navigate the system.

- **Customizable Menu:** The online menu can be easily updated by restaurant staff, ensuring it reflects the latest offerings and promotions.

2. Secure User Accounts:

- **Enhanced Security:** Each user is provided with a unique ID and password, ensuring secure access to their accounts and protecting personal and payment data.

3. Multiple Delivery Addresses:

- **Convenience for Users:** Customers can save and manage multiple delivery addresses within their accounts, making it easy to select different locations for different orders.

4. Comprehensive Rating System:

- **Detailed Feedback:** Users can rate both restaurants and individual food items, providing valuable insights for other customers and actionable feedback for restaurant management.

5. Order Cancellation Feature:

- **Flexible Order Management:** Customers have the option to cancel their orders before a delivery partner is assigned, providing greater flexibility and control over their purchasing decisions.

6. Scalability and Flexibility:

- **Modular Design:** The system's modular architecture allows for easy integration of new features and functionalities, ensuring adaptability to evolving market trends and technological advancements.

- **Scalable Infrastructure:** Built on SQL, the system can efficiently handle a growing number of users and transactions, accommodating the expanding needs of restaurants and customers.