SSN COLLEGE OF ENGINEERING

UCS2404

DATABASE MANAGEMENT SYSTEMS

FOOD DELIVERY MANAGEMENT SYSTEM

TEAM MEMBERS

1) DILSHA SINGH D - 3122225001028
2) G KUSHAL VARMA - 3122225001031
3) HANNAH S -3122225001032

TABLE OF CONTENTS

PROBLEM STATEMENT:

Managing food deliveries is a big challenge for restaurants. They need to make sure orders are delivered on time and track delivery workers efficiently. Our project, a Food Delivery Management System, aims to simplify this process. It connects restaurant databases to track orders and available delivery staff in real-time. The system automatically assigns delivery tasks and updates the status of delivery workers. Customers can track their orders, and managers can monitor deliveries to quickly solve any problems. This system helps restaurants deliver food faster, reduce customer wait times, and improve overall service.

TABLES AND THEIR ATTRIBUTES:

USERS

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
user_id	User’s ID	char	Primary key(Check ‘U%’)
username	User’s username	Varchar2	Unique
email	User’s email	Varchar2	Unique
password	User’s Password	char	Check
User_type	Type of user	Varchar2	Check

CUSTOMERS

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
cust_id	Customer ID	char	Primary Key(Check ‘C%’)
name	Customer’s name	Varchar	-
phone	Phone Number of Customer	Number	Unique
location	Address of Customer	Varchar2	-
dob	Date of birth of Customer	Date	-
pincode	Pincode of the address	number	-

RESTAURANTS

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
res_id	Restaurant ID	char	Primary Key(Check ‘R%’)
owner_id	Owner’s id	char	Foreign Key(Check ‘O%’)
rest_name	Restaurant’s name	Varchar2	-
Rest_location	Address of Restaurant	Varchar2	-
Rest_Cuisine_type	Types of Cuisines in Restaurant	Varchar2	-

Rest type	Type(Veg or Non-veg)	Varchar2	Check(Veg or Non-Veg)
pincode	Pincode of the location	number	-
Delv_partner_id	Id of the partner affiliated to the restaurant	char	Foreign key(Check ‘D%’)

MENU_ITEMS

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (IfAny)
item_id	Food’s Id	Char	Primary Key (Composite key) (Check ‘I%’)
rest_id	Restaurant’s Id	Char	Foreign Key (Composite Key) (Check ‘R%’)
name	Food Item’s name	Varchar2	-
description	Food’s description	Varchar2	-
price	Price of the Food	Decimal	-

ORDERS

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (IfAny)
order_id	Ordered Food’s Id	Char	Primary Key (Check ‘O%’)
cust_id	Customer’s Id	Char	Foreign Key (Check ‘C%’)

delivery_person_id	Delivery person's ID	Char	Foreign Key (Check 'D%')
rest_id	Restaurant's Id	Char	Foreign Key (Check 'R%')
order_date	Ordered Date	Date	-
Rec_date	Delivery Date	Date	-

ORDER_ITEM

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
order_id	Ordered Food's Id	Char	Foreign key(Composite key) (Check 'O%')
item_id	Food's Id	Char	Foreign Key (Check 'I%')
quantity	Ordered Quantity	Number	-

DELIVERY_PERSON

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
delivery_person_id	Delivery person's ID	Char	Primary Key (Check 'Dp%')
Delv_partner_id	Delivery_partner_id	Char	Foreign Key (Check 'D%')
name	Delivery Person's name	Varchar2	-

location	Delivery person’s loc	Varchar2	-
pincode	Pincode of the location	number	-
phone	PhoneNumber of Delivery Person	Number	Unique
Veh_no	Vehicle Number	Varchar2	Unique
availability	Vehicle available or not	Varchar2	Check(available or not)

PAYMENT

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (If Any)
pay_id	Payment ID	Char	Primary key (Check ‘P%’)
order_id	Ordered Food’s Id	Char	Foreign Key (Check ‘O%’)
pay_date	Date of payment	Date	-
pay_method	Method of payment (cash or Online)	Varchar2	Check(cash or online)

RATING

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (IfAny)
rate_id	Rating Id	Char	Primary Key (Check ‘r%’)
cust_id	Customer’s Id	Char	Foreign Key (Check ‘C%’)
restau_id	Restaurant’s Id	Char	Foreign Key (Check ‘R%’)

Review	Customer's Review	Varchar2	-
rate_no	Rating Number(1-5)	Number	Check(1-5)

DELIVERY_PARTNER

ATTRIBUTE	DESCRIPTION	DATA TYPE	CONSTRAINT (IfAny)
Dp_id	Delivery partner Id	Char	Primary Key (Check 'D%')
Partner_name	Name of the partner	Varchar2	-
Phone	Admin phone number	number	-
Location	Address of the office	Varchar2	-
Pincode	Pincode of the location	Number	-

FUNCTIONAL DEPENDENCIES:

1)USER:

Possible functional dependencies:

User_id -> user_name, password

user_id -> user_type

User_id, password -> user_type

User_name -> password, user_type

User_name -> user_id

User_id, username, email -> user_id, email

From above : User_id, username, email -> user_id, email is **Trivial** so eliminating

From above: No time dependent attributes

Deriving Minimal sets:

Let F = {User_id -> user_name, password,, user_id -> user_type,, User_id, password -> user_type,, User_name -> password, user_type,, User_name -> user_id}

1) Applying Decomposition:

   1)User_id -> user_name, password

      User_id -> user_name

      User-id -> password

   2)User_name -> password, user_type

      User_name -> password

      User_name -> user_type

F = {user_id -> user_name,, user_id -> password,, user_id -> user_type,, user_id,password->user_type,, user_name -> password,, user_name -> user_type,, user_name -> user_id}

2) Finding Extraneous:

In User_id, password -> user_type

Remove user_id, It is not extraneous

Remove password , It is extraneous

So it becomes user_id -> user_type

So, F = {user_id -> user_name,, user_id -> password,, user_id -> user_type,, user_name -> password,, user_name -> user_type,, user_name -> user_id}

3) Finding Redundant

There is no redundant found

F = {user_id -> user_name, user_id -> password, user_id -> user_type, user_name -> password,, user_name -> user_type,, user_name -> user_id}

So The minimal set for User table is:

{user_id-> user_name,password,user_type,, user_name->
user_id,password_user_type }

CLOSURE

User_id = {user_id, user_name , email, password}

User_name = {user_name, user_id, email, password}

Password - not possible

User_id, user_name are minimal super keys

Which means they are candidate keys

Thus, **user_id → primary key**

---

2)CUSTOMER:

Possible functional dependencies:

Cust_id -> name, phone

Name, phone -> dob

Cust_id -> cust_id, cust_name

Cust_phone -> address ,dob

Phone -> address, cust_id

Phone -> name

Pincode -> address

Cust_id -> pincode

Phone -> pincode

Cust_id, name, dob -> dob

From above :Cust_id, name, dob -> dob is **Trivial,** so eliminating

From above: No time dependent attributes

Deriving Minimal sets:

1) Applying Decomposition:

We get,

F = {Cust_id -> name,, cust_id -> phone,, name, phone -> dob,, Cust_id -> dob,, cust_id -> address,, cust_id -> pincode,, phone -> address,, Phone -> name,, phone -> cust_id,, phone -> pincode,, pincode -> address}

2)Finding Extraneous:

In name, phone -> dob :

Remove name, It is not extraneous

F = {Cust_id -> name,, cust_id -> phone,, phone -> dob,, Cust_id -> dob,, cust_id -> address,, cust_id -> pincode,, phone -> address,, Phone -> name,, phone -> cust_id,, phone -> pincode,, pincode -> address}

3) Finding Redundant

Checking whether cust_id -> name is redundant
cust_id = {cust_id, name, address, cust_phone, cust_dob}
As closure satisfies it is redundant

Checking whether cust_id -> phone is redundant
cust_id = {cust_id, User_id, cust_name, cust_address, cust_phone, cust_dob}
As closure satisfies it is redundant

F = {phone -> dob,, Cust_id -> dob,, cust_id -> address,, cust_id -> pincode,, phone -> address,, Phone -> name,, phone -> cust_id,, phone -> pincode,, pincode -> address}

Other Fds are not redundant

So The minimal set for Customer table is:

{cust_id-> address,dob,pincode,, phone-> cust_id,name,phone,address,dob,pincode,, pincode-> address}

Closure:

Cust_id = {cust_id, user_id, cust_name, cust_address, cust_phone, cust_dob}

phone = {cust_phone, user_id, cust_id, cust_name, cust_address, cust_dob}

For pincode not possible

Cust_id, phone are minimal super keys

Which means they are candidate keys

cust_id → primary key

---

3)RESTAURANT:

Possible Functional Dependencies:

Rest_id -> rest_name, owner_id, pincode, delivery_partner_id

Rest_id -> rest_location
Rest_id -> rest_cuisine, rest_type
Rest_name, owner_id -> rest_id
Owner_id -> rest_location, rest_cuisine
Owner_id -> rest_id, rest_name
Pincode -> rest_location
Rest_name, owner_id, rest_id -> rest_id

From the above:

Rest_name, owner_id, rest_id -> rest_id is Trivial, so eliminating
No time-dependent attributes

Deriving Minimal Sets:

Let F = {Rest_id -> rest_name, owner_id, pincode, delivery_partner_id, Rest_id -> rest_location, Rest_id -> rest_cuisine, rest_type, Rest_name, owner_id -> rest_id, Owner_id -> rest_location, rest_cuisine, Owner_id -> rest_type, Owner_id -> rest_id, rest_name, Pincode -> rest_location, delivery_partner_id}

1. **APPLYING DECOMPOSITION:**

F = {
 Rest_id -> rest_name,
 Rest_id -> owner_id,
 Rest_id -> pincode,
 Rest_id -> delivery_partner_id,
 Rest_id -> rest_location,
 Rest_id -> rest_cuisine,
 Rest_id -> rest_type,
 Rest_name, owner_id -> rest_id,
 Owner_id -> rest_location,
 Owner_id -> rest_cuisine,
 Owner_id -> rest_type,
 Owner_id -> rest_id,
 Owner_id -> rest_name,
 Pincode -> rest_location,
 Pincode -> delivery_partner_id
}

2. **FINDING EXTRANEOUS:** In `Rest_name, owner_id -> rest_id`:

- Remove Rest_name, it is extraneous
- Remove owner_id, it is extraneous
- Removing `Rest_name, owner_id -> rest_id`

F = {
 Rest_id -> rest_name,

Rest_id -> owner_id,
Rest_id -> pincode,
Rest_id -> delivery_partner_id,
Rest_id -> rest_location,
Rest_id -> rest_cuisine,
Rest_id -> rest_type,
Owner_id -> rest_location,
Owner_id -> rest_cuisine,
Owner_id -> rest_type,
Owner_id -> rest_id,
Owner_id -> rest_name,
Pincode -> rest_location,
Pincode -> delivery_partner_id
}

3. **FINDING REDUNDANT:** Checking whether `Rest_id -> rest_name` is redundant:

- Rest_id = {Rest_id, rest_name, owner_id, pincode, delivery_partner_id, rest_location, rest_cuisine, rest_type}
- As closure satisfies it is redundant

Checking whether `Rest_id -> rest_location` is redundant:

- Rest_id = {Rest_id, rest_name, owner_id, pincode, delivery_partner_id, rest_location, rest_cuisine, rest_type}
- As closure satisfies it is redundant

Checking whether `Rest_id -> rest_cuisine` is redundant:

- Rest_id = {Rest_id, rest_name, owner_id, pincode, delivery_partner_id, rest_location, rest_cuisine, rest_type}
- As closure satisfies it is redundant

Checking whether `Rest_id -> rest_type` is redundant:

- Rest_id = {Rest_id, rest_name, owner_id, pincode, delivery_partner_id, rest_location, rest_cuisine, rest_type}
- As closure satisfies it is redundant

F = {
Rest_id -> owner_id,
Owner_id -> rest_location,
Owner_id -> rest_cuisine,
Owner_id -> rest_type,
Owner_id -> rest_id,
Owner_id -> rest_name,
Pincode -> rest_location,
Pincode -> delivery_partner_id
}

Other FDs are not redundant.

The Minimal Set for the Restaurant Table:

{Rest_id -> owner_id, Owner_id -> rest_location,rest_cuisine,rest_type,rest_id,rest_name, Pincode -> rest_location,delivery_partner_id}

Closure:

Rest_id = {Rest_id, owner_id, rest_name, rest_location, rest_cuisine, rest_type, pincode, delivery_partner_id}
Owner_id = {Owner_id, rest_id, rest_name, rest_location, rest_cuisine, rest_type, pincode, delivery_partner_id}
Pincode = {Pincode, rest_location, delivery_partner_id}
Rest_id, owner_id are minimal super keys
Which means they are candidate keys

Rest_id-> primary key

---

4)PAYMENT:

POSSIBLE FUNCTIONAL DEPENDENCIES:

Payment_id -> payment_date

Payment_id -> payment_method

Payment_id -> order_id

Payment_id, order_id -> Payment_id

Payment_date, order_id -> Payment_id

Order_id -> Payment_id, payement_method

FROM ABOVE: Payment_id, order_id -> Payment_id is **Trivial** so eliminating

FROM ABOVE: No time dependent attributes

DERIVING MINIMAL SETS:

Let F = {*Payment_id -> payment_date, Payment_id -> payment_method, Payment_id -> order_id*

Payment_id, order_id -> Payment_id, Order_id -> Payment_id, payement_method}

1)APPLYING DECOMPOSITION:

We get,

F = { Payment_id -> payment_date, Payment_id -> payment_method, Payment_id -> order_id, Order_id -> Payment_id, Order_id -> payment_method ,Payment_date, order_id -> Payment_id}

2)FINDING EXTRANEOUS:

In Payment_date, order_id -> Payment_id:

Remove order_id, It is extraneous

F = { Payment_id -> payment_date, Payment_id -> payment_method, Payment_id -> order_id, Order_id -> Payment_id, Order_id -> payment_method ,Payment_date -> Payment_id }

3)FINDING REDUNDANT:

Checking whether Payment_id -> payment_date is redundant

Payment_id = {payment_id,payment_method,order_id }

closure not satisfied

Checking whether Payment_id -> payment_method is redundant

payment_id = { payment_id,payment_method,order_id,payment_method }

closure is satisfied

It is redundant

Checking whether Payment_id -> order_id is redundant

payment_id = {payment_id,payment_date }

closure not satisfied

Checking whether Order_id -> payment_method is redundant

Order_id = { Order_id,payment_id,payment_id,payment_date }

Closure is not satisfied

Checking whether Order_id -> Payment_id is redundant

Order_id = { Order_id,payment_method }

Closure is not satisfied

F = { Payment_id -> payment_date, Payment_id -> order_id, Order_id -> payment_method, Order_id -> Payment_id }

Other Fds are not redundant

The minimal set for Restaurant table is:

{ Payment_id -> payment_date, Payment_id -> order_id, Order_id -> payment_method, Order_id -> Payment_id }

CLOSURE:

Payment_id = { Payment_id, payment_date, order_id, payment_method }

Order_id = { Payment_id, payment_date, order_id, payment_method }

payment_date ={ payment_date }

payment_method= { payment_method }

So, Payment_ID and Order_ID are minimal superkeys/candidate keys.

Payment_ID->primary key

---

5)DELIVERY PERSON:

Possible Functional Dependencies:

Dp_id -> user_id
Dp_id -> name
Dp_id -> Dp_phone, Dp_vehicleno
Dp_id -> Dp_available, Dp_location
Dp_id -> delivery_partner_id, pincode
User_id -> Dp_id, Dp_location
Dp_phone -> Dp_id
Dp_vehicleno -> Dp_id
Dp_phone, user_id -> Dp_vehicleno

Pincode -> Dp_location

From the above:

Dp_id -> Dp_phone, Dp_id is Trivial, so eliminating
No time-dependent attributes

Deriving Minimal Sets:

Let F = {Dp_id -> user_id, Dp_id -> name, Dp_id -> Dp_phone, Dp_vehicleno, Dp_id -> Dp_available, Dp_location, delivery_partner_id, pincode, User_id -> Dp_id, Dp_location, Dp_phone -> Dp_id, Dp_vehicleno -> Dp_id, Dp_phone, user_id -> Dp_vehicleno, Pincode -> Dp_location}

1. **APPLYING DECOMPOSITION:**

F = {
  Dp_id -> user_id,
  Dp_id -> name,
  Dp_id -> Dp_phone,
  Dp_id -> Dp_vehicleno,
  Dp_id -> Dp_available,
  Dp_id -> Dp_location,
  Dp_id -> delivery_partner_id,
  Dp_id -> pincode,
  User_id -> Dp_id,
  User_id -> Dp_location,
  Dp_phone -> Dp_id,
  Dp_vehicleno -> Dp_id,
  Dp_phone, user_id -> Dp_vehicleno,
  Pincode -> Dp_location
}

2. **FINDING EXTRANEOUS:** In Dp_phone, user_id -> Dp_vehicleno:

- Remove user_id, it is extraneous
- Removing Dp_phone, user_id -> Dp_vehicleno

F = {
  Dp_id -> user_id,
  Dp_id -> name,
  Dp_id -> Dp_phone,
  Dp_id -> Dp_vehicleno,
  Dp_id -> Dp_available,

Dp_id -> Dp_location,
Dp_id -> delivery_partner_id,
Dp_id -> pincode,
User_id -> Dp_id,
User_id -> Dp_location,
Dp_phone -> Dp_id,
Dp_vehicleno -> Dp_id,
Pincode -> Dp_location
}

3. **FINDING REDUNDANT:** Checking whether Dp_id -> user_id is redundant:

- Dp_id = {Dp_id, name, Dp_phone, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode}
- Closure not satisfied

Checking whether Dp_id -> name is redundant:

- Dp_id = {Dp_id, user_id, Dp_phone, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode}
- Closure not satisfied

Checking whether Dp_id -> Dp_phone is redundant:

- Dp_id = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode}
- Closure not satisfied

Checking whether Dp_id -> Dp_vehicleno is redundant:

- Dp_id = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode, Dp_phone}
- Closure is satisfied

Checking whether Dp_id -> Dp_available is redundant:

- Dp_id = {Dp_id, user_id, name, Dp_vehicleno, Dp_location, delivery_partner_id, pincode, Dp_phone}
- Closure not satisfied

Checking whether Dp_id -> Dp_location is redundant:

- Dp_id = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, delivery_partner_id, pincode, Dp_phone}
- Closure is satisfied

Checking whether User_id -> Dp_id is redundant:

- User_id = {User_id, Dp_location, Dp_vehicleno}
- Closure not satisfied

Checking whether User_id -> Dp_location is redundant:

- Dp_id = {User_id, Dp_vehicleno}
- Closure not satisfied

Checking whether Pincode -> Dp_location is redundant:

- Pincode = {Pincode, Dp_location}
- Closure is satisfied

F = {
Dp_id -> user_id,
Dp_id -> name,
Dp_id -> Dp_phone,
Dp_id -> Dp_available,
Dp_id -> delivery_partner_id,
Dp_id -> pincode,
User_id -> Dp_id,
User_id -> Dp_location,
Dp_phone -> Dp_id,
Dp_vehicleno -> Dp_id,
Pincode -> Dp_location
}

Other FDs are not redundant.

The Minimal Set for Delivery Person Table:

{Dp_id -> user_id, Dp_id -> name, Dp_id -> Dp_phone, Dp_id -> Dp_available, Dp_id -> delivery_partner_id, Dp_id -> pincode,User_id -> Dp_id, User_id -> Dp_location, Dp_phone -> Dp_id, Dp_vehicleno -> Dp_id, Pincode -> Dp_location}

Closure:

Dp_id = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, Dp_location, Dp_phone, delivery_partner_id, pincode}
User_id = {User_id, Dp_id, Dp_location, Dp_vehicleno, name, Dp_available, delivery_partner_id, pincode, Dp_phone}

Dp_phone = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode, Dp_phone}
Dp_vehicleno = {Dp_id, user_id, name, Dp_vehicleno, Dp_available, Dp_location, delivery_partner_id, pincode, Dp_phone}
Dp_name = {Dp_name}
Dp_available = {Dp_available}

So, DP_ID, USER_ID, DP_PHONE, DP_VEHICLENO are all minimal superkeys.

DP_ID -> primary key

---

6)ORDERS:

Possible Functional Dependencies:

Order_id -> rest_id, Dp_id
Order_id -> cust_id
Order_id -> order_date, delv_date
Order_id -> item_id, quantity
Rest_id -> order_id
Dp_id -> order_id
Cust_id -> order_id
Order_date, Dp_id -> delv_date
Delv_date, Rest_id -> order_date
Pincode -> location

From the above:

Rest_id -> order_id, Rest_id is Trivial, so eliminating
No time-dependent attributes

Deriving Minimal Sets:

Let F = { Order_id -> rest_id, Dp_id, Order_id -> cust_id, Order_id -> order_date, delv_date, Order_id -> item_id, quantity, Rest_id -> order_id, Dp_id -> order_id, Cust_id -> order_id, Order_date, Dp_id -> delv_date, Delv_date, Rest_id -> order_date }

1. **APPLYING DECOMPOSITION:**

F = {
 Order_id -> rest_id,
 Order_id -> Dp_id,
 Order_id -> cust_id,
 Order_id -> order_date,

Order_id -> delv_date,
Order_id -> item_id,
Order_id -> quantity,
Rest_id -> order_id,
Dp_id -> order_id,
Cust_id -> order_id,
Order_date, Dp_id -> delv_date,
Delv_date, Rest_id -> order_date
}

2. **FINDING EXTRANEOUS:** In Order_date, Dp_id -> delv_date:

- Remove Order_date, it is extraneous
- Removing Order_date, Dp_id -> delv_date
- Adding Dp_id -> delv_date

In Delv_date, Rest_id -> order_date:

- Remove Delv_date, it is extraneous
- Removing Delv_date, Rest_id -> order_date
- Adding Rest_id -> order_date

F = {
Order_id -> rest_id,
Order_id -> Dp_id,
Order_id -> cust_id,
Order_id -> order_date,
Order_id -> delv_date,
Order_id -> item_id,
Order_id -> quantity,
Rest_id -> order_id,
Rest_id -> order_date,
Dp_id -> order_id,
Dp_id -> delv_date,
Cust_id -> order_id
}

3. **FINDING REDUNDANT:** Checking whether Order_id -> rest_id is redundant:

- Order_id = {Order_id, Dp_id, cust_id, order_date, delv_date, item_id, quantity}
- Closure not satisfied

Checking whether Order_id -> Dp_id is redundant:

- Order_id = {Order_id, rest_id, cust_id, order_date, delv_date, item_id, quantity}
- Closure not satisfied

Checking whether Order_id -> cust_id is redundant:

- Order_id = {Order_id, rest_id, Dp_id, order_date, delv_date, item_id, quantity}
- Closure not satisfied

Checking whether Order_id -> order_date is redundant:

- Order_id = {Order_id, rest_id, Dp_id, cust_id, delv_date, item_id, quantity}
- Closure not satisfied

Checking whether Order_id -> delv_date is redundant:

- Order_id = {Order_id, rest_id, Dp_id, cust_id, order_date, item_id, quantity}
- Closure satisfied

Checking whether Order_id -> item_id is redundant:

- Order_id = {Order_id, rest_id, Dp_id, cust_id, order_date, delv_date, quantity}
- Closure not satisfied

Checking whether Order_id -> quantity is redundant:

- Order_id = {Order_id, rest_id, Dp_id, cust_id, order_date, delv_date, item_id}
- Closure satisfied

Checking whether Rest_id -> order_id is redundant:

- Rest_id = {Rest_id, order_date}
- Closure not satisfied

Checking whether Rest_id -> order_date is redundant:

- Rest_id = {Rest_id, order_id, delv_date}
- Closure satisfied

Checking whether Dp_id -> order_id is redundant:

- Dp_id = {Dp_id, delv_date}
- Closure not satisfied

Checking whether Dp_id -> delv_date is redundant:

- Dp_id = {Dp_id, order_id}
- Closure not satisfied

Checking whether Cust_id -> order_id is redundant:

- Cust_id = {Cust_id}

- Closure not satisfied

F = {
Order_id -> rest_id,
Order_id -> Dp_id,
Order_id -> cust_id,
Order_id -> order_date,
Order_id -> item_id,
Rest_id -> order_id,
Dp_id -> order_id,
Dp_id -> delv_date,
Cust_id -> order_id
}

Other FDs are not redundant.

The Minimal Set for Order Table:

{Order_id -> rest_id, Order_id -> Dp_id, Order_id -> cust_id Order_id -> order_date, Order_id -> item_id-> Rest_id, Rest_ID -> order_id, Dp_id -> order_id, Dp_id -> delv_date, Cust_id -> order_id}

Closure:

Order_id,item_id = {Order_id, rest_id, Dp_id, cust_id, order_date, delv_date, item_id}
Rest_id = {Rest_id, order_id, delv_date}
Dp_id = {Dp_id, order_id, delv_date}
Cust_id = {Cust_id, order_id, delv_date}
Order_date = {order_date}
Delv_date = {delv_date}

Order_id and item_id are minimal superkeys.

ORDER_ID, ITEM_ID-> primary key

---

7)MENU_ITEM:

POSSIBLE FUNCTIONAL DEPENDENCIES:

Item_ID,Rest_ID->Item_name

Item_ID,Rest_ID->Item_desc

Item_ID,Rest_ID->Item_price

Item_ID,Rest_ID->Item_ID,Rest_ID

Item_ID,Rest_ID,Item_name->Item_ID,Rest_ID

Item_ID,Rest_ID,Item_desc->Item_ID,Rest_ID

Item_ID,Rest_ID,Item_price->Item_ID,Rest_ID

Item_ID,Rest_ID,Item_name,Item_desc,Item_price->Item_ID,Rest_ID,Item_name,Item_desc,Item_price

FROM ABOVE: Except first three FDs, all the others are **Trivial** so eliminating them.

FROM ABOVE: No time dependent attributes

DERIVING MINIMAL SETS:

Let F = {*Item_ID,Rest_ID->Item_name,* *Item_ID,Rest_ID->Item_desc*

Item_ID,Rest_ID->Item_price}

1)APPLYING DECOMPOSITION:

No decomposition to apply for this set of FDs as there are double attribute FDs on RHS

F = {Item_ID,Rest_ID->Item_name , Item_ID,Rest_ID->Item_desc , Item_ID,Rest_ID->Item_price }

2)FINDING EXTRANEOUS:

Neither Item_ID nor Rest_ID is extraneous in F since there are no singleton sets to check with and that attribute cant be arrived at in closure without it being in the LHS for this table.

F = {Item_ID,Rest_ID->Item_name , Item_ID,Rest_ID->Item_desc , Item_ID,Rest_ID->Item_price }

3)FINDING REDUNDANT:

Checking whether Item_ID,Rest_ID->Item_name is redundant
Item_ID,Rest_ID = {Item_ID,Rest_ID,Item_desc,Item_price}
As closure doesnt satisfy, it is not redundant.

Checking whether Item_ID,Rest_ID->Item_desc is redundant
Item_ID,Rest_ID = {Item_ID,Rest_ID,Item_name,Item_price}
As closure doesnt satisfy, it is not redundant.

Checking whether Item_ID,Rest_ID->Item_price is redundant
Item_ID,Rest_ID = {Item_ID,Rest_ID,Item_name,Item_desc}
As closure doesnt satisfy, it is not redundant.

F = {Item_ID,Rest_ID->Item_name , Item_ID,Rest_ID->Item_desc , Item_ID,Rest_ID->Item_price }

The minimal set for User table is:

F = {Item_ID,Rest_ID->Item_name , Item_ID,Rest_ID->Item_desc , Item_ID,Rest_ID->Item_price }

CLOSURE:

Item_id, Rest_id={Item_id, Rest_id,Item_name,Item_desc,Item_price}

Closure is satisfied for this combination of attributes alone.

Therefore,Item_ID with Rest_ID together is a minimal super key, so candidate key.

Item_ID,Rest_ID → primary key(composite key)

---

8)RATING:

POSSIBLE FUNCTIONAL DEPENDENCIES:

Rating_ID->Rating_ID

Rating_ID->Cust_ID

Rating_ID->review

Rating_ID->Rest_ID

Rating_ID->Rating_num

Review->Rating_ID

Review->Cust_ID

Review->review

Review->Rest_ID

Review->Rating_num

Rating_ID,review->Rating_ID

Rating_ID,review->Cust_ID

Rating_ID,review->review

Rating_ID,review->Rest_ID

Rating_ID,review->Rating_num

FROM ABOVE: No **Trivial FDs**

FROM ABOVE: No time dependent attributes

DERIVING MINIMAL SETS:

Let F= {*Rating_ID->Rating_ID,Rating_ID->Cust_ID,Rating_ID->review,Rating_ID->Rest_ID,Rating_ID->Rating_num,Review->Rating_ID,Review->Cust_ID,Review->review,Review->Rest_ID,Review->Rating_num*

Rating_ID,review->Rating_ID,

Rating_ID,review->Cust_ID,

Rating_ID,review->review,

Rating_ID,review->Rest_ID,

Rating_ID,review->Rating_num,}

1)APPLYING DECOMPOSITION:

No decomposition to be applied here.

F = {Rating_ID->Rating_ID,Rating_ID->Cust_ID,Rating_ID->review,Rating_ID->Rest_ID,Rating_ID->Rating_num,Review->Rating_ID,Review->Cust_ID,Review->review,Review->Rest_ID,Review->Rating_num
Rating_ID,review->Rating_ID,
Rating_ID,review->Cust_ID,
Rating_ID,review->review,
Rating_ID,review->Rest_ID,
Rating_ID,review->Rating_num}

2)FINDING EXTRANEOUS:

The FDs with combination of attributes in LHS is not required at all, since both Rating_ID and review can work independently without requiring the other. So both are extraneous. Hence we can remove the Combination of attributes FDs in the list.

F ={Rating_ID->Rating_ID,Rating_ID->Cust_ID,Rating_ID->review,Rating_ID->Rest_ID,Rating_ID->Rating_num,Review->Rating_ID,Review->Cust_ID,Review->review,Review->Rest_ID,Review->Rating_num}

3)FINDING REDUNDANT:

Through transitive property, we can check that Review->Cust_ID,Review->review,Review->Rest_ID,Review->Rating_num are all redundant as the closure property of D is satisfied without them itself. So the FDs are:

F ={Rating_ID->Rating_ID,Rating_ID->Cust_ID,Rating_ID->review,Rating_ID->Rest_ID,Rating_ID->Rating_num,Review->Rating_ID}

Other Fds are not redundant

The minimal set for User table is:

{Rating_ID->Rating_ID,Rating_ID->Cust_ID,Rating_ID->review,Rating_ID->Rest_ID,Rating_ID->Rating_num,Review->Rating_ID}

CLOSURE:

Rating_ID={Rating_ID,Review,Rest_ID,Cust_ID,Rating_num}

Review={Rating_ID,Review,Rest_ID,Cust_ID,Rating_num}

So both rating_ID and Review are minimal superkeys.

Hence, they are candidate keys.

Rating_ID-> primary key

---

9)DELIVERY PARTNER:

POSSIBLE FUNCTIONAL DEPENDENCIES:

Delivery_partner_ID->Delivery_partner_ID

Delivery_partner_ID->Name

Delivery_partner_ID->Phone

Delivery_partner_ID->Location

Delivery_partner_ID->Pincode

Pincode->Location

Name->Phone

Name->Location

Name->Pincode

Name->Name

Name->Delivery_Person_ID

Delivery_partner_ID,name->Delivery_partner_ID

Delivery_partner_ID,name->phone

Delivery_partner_ID,name->name

Delivery_partner_ID,name->location

Delivery_partner_ID,name->pincode

FROM ABOVE: No **Trivial FDs**

FROM ABOVE: No time dependent attributes

DERIVING MINIMAL SETS:

Let F= {*Delivery_partner_ID->Delivery_partner_ID,Delivery_partner_ID->Cust_ID,Delivery_partner_ID->name,Delivery_partner_ID->Phone,Delivery_partner_ID->Pincode,Name->Delivery_partner_ID,Name->Cust_ID,Name->name,Name->Phone,Name->Pincode*

Delivery_partner_ID,name->Delivery_partner_ID,

Delivery_partner_ID,name->Cust_ID,

Delivery_partner_ID,name->name,

Delivery_partner_ID,name->Phone,

Delivery_partner_ID,name->Pincode,}

1)APPLYING DECOMPOSITION:

No decomposition to be applied here.

F = {Delivery_partner_ID->Delivery_partner_ID,Delivery_partner_ID->Location,Delivery_partner_ID->name,Delivery_partner_ID->Phone,Delivery_partner_ID->Pincode,Name->Delivery_partner_ID,Name->Location,Name->name,Name->Phone,Name->Pincode
Delivery_partner_ID,name->Delivery_partner_ID,
Delivery_partner_ID,name->Location,

Delivery_partner_ID,name->name,
Delivery_partner_ID,name->Phone,
Delivery_partner_ID,name->Pincode}

2)FINDING EXTRANEOUS:

The FDs with combination of attributes in LHS is not required at all, since both Delivery_Partner_ID and name can work independently without requiring the other. So both are extraneous. Hence we can remove the Combination of attributes FDs in the list.

F = {Delivery_partner_ID->Delivery_partner_ID,Delivery_partner_ID->Location,Delivery_partner_ID->name,Delivery_partner_ID->Phone,Delivery_partner_ID->Pincode,Name->Delivery_partner_ID,Name->Location,Name->name,Name->Phone,Name->Pincode}

3)FINDING REDUNDANT:

Through transitive property, we can check that Name->Phone,Name->name,Name->Location,Name->Pincode are all redundant as the closure property of D is satisfied without them itself. So the FDs are:

F ={Delivery_partner_ID->Delivery_partner_ID,Delivery_partner_ID->Phone,Delivery_partner_ID->name,Delivery_partner_ID->Location,Delivery_partner_ID->Pincode,Name->Delivery_partner_ID}

Other Fds are not redundant

The minimal set for User table is:

{Delivery_partner_ID->Delivery_partner_ID,Delivery_partner_ID->Phone,Delivery_partner_ID->name,Delivery_partner_ID->Location,Delivery_partner_ID->Pincode,Name->Delivery_partner_ID}

CLOSURE:

Delivery_partner_ID={Delivery_partner_ID,Name,Location,Phone,Pincode}

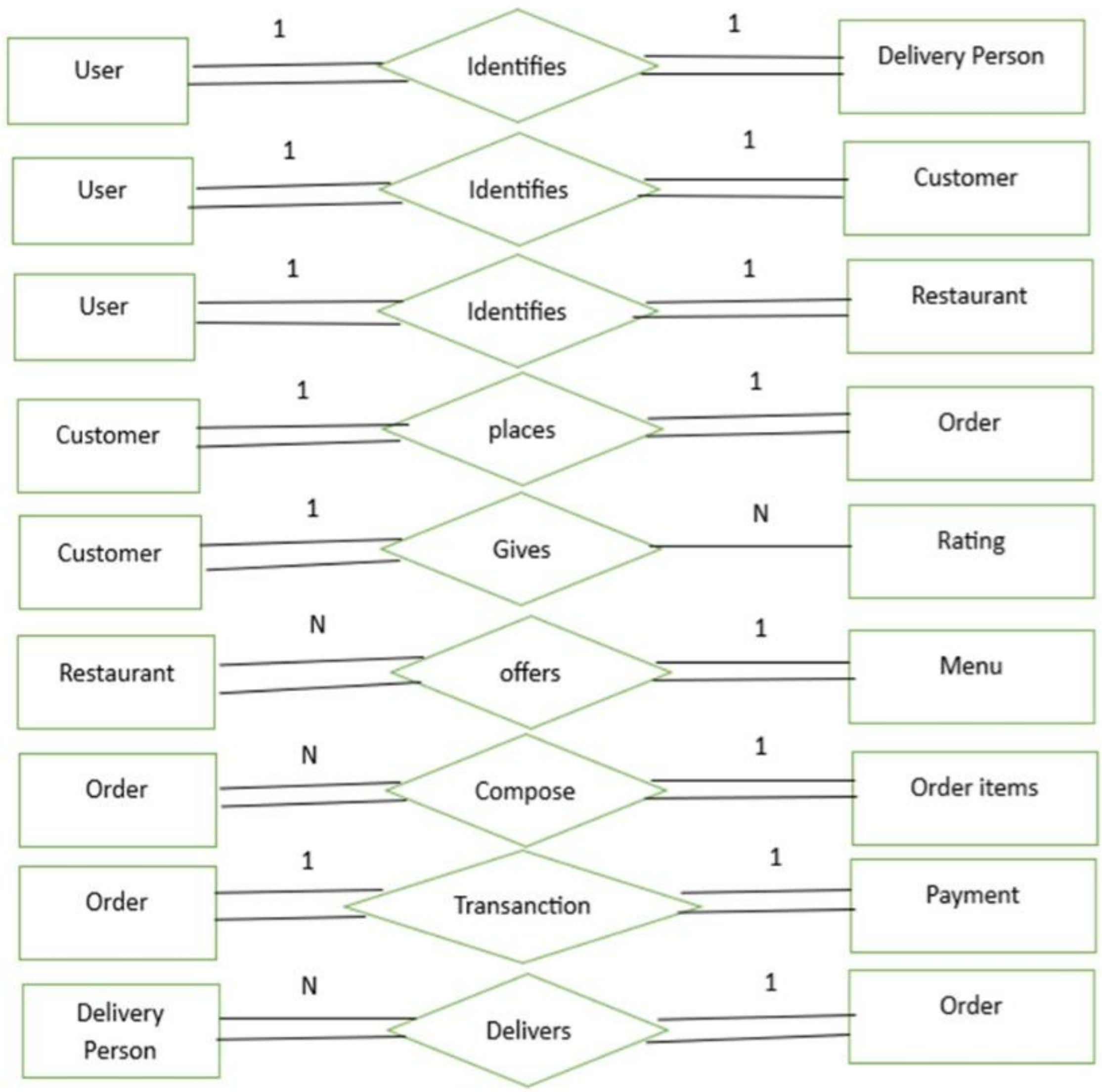
Name={Delivery_partner_ID,Name,Location,Phone,Pincode}

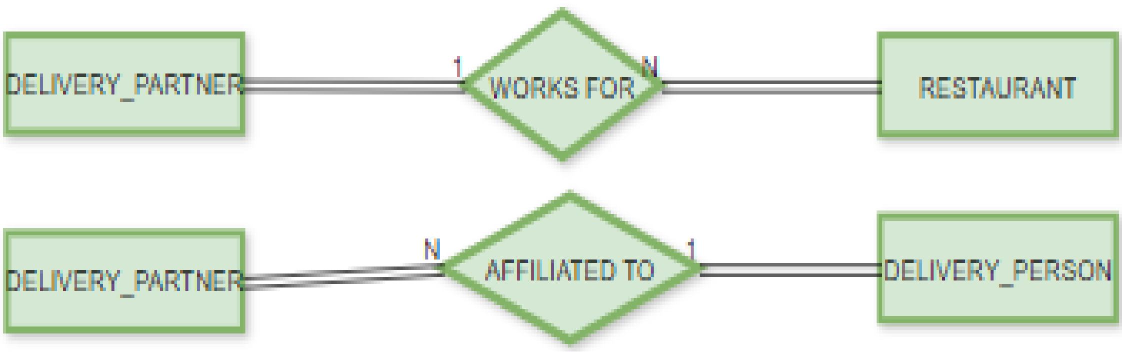
So both Delivery_partner_ID and Name are minimal superkeys.

Hence, they are candidate keys.

DELIVERY_PARTNER_ID-> primary key

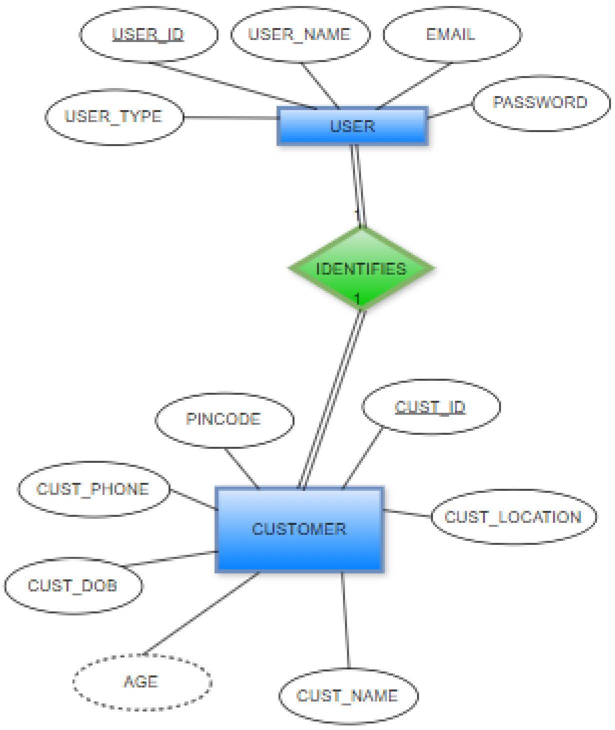
ER RELATIONS:



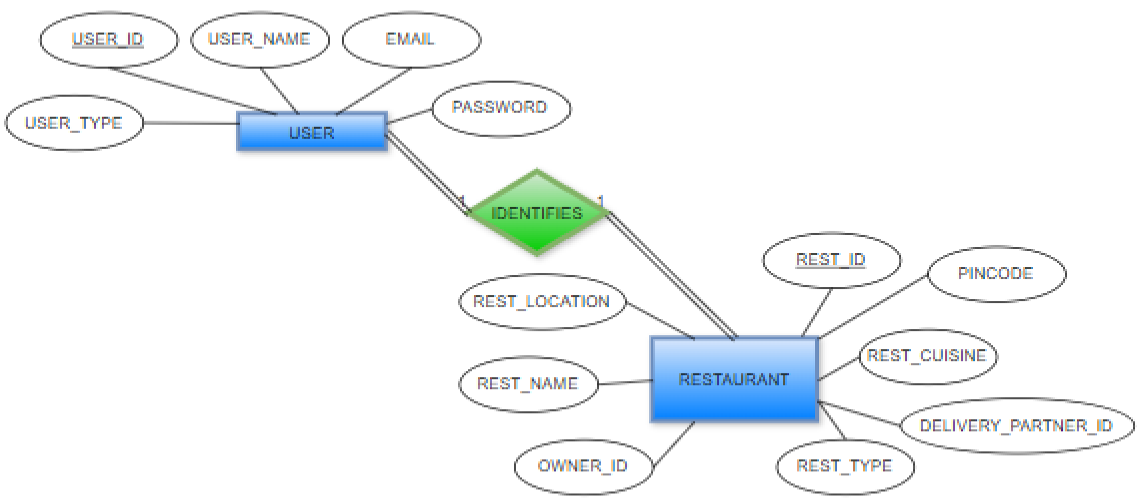


ER-TO-RELATION DIAGRAM:

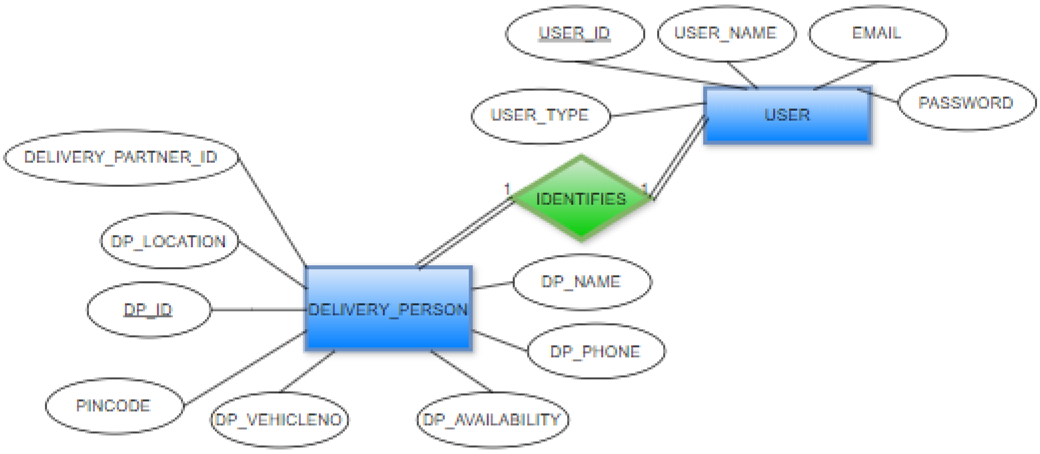
1)USER IDENTIFIES AS COSTUMER



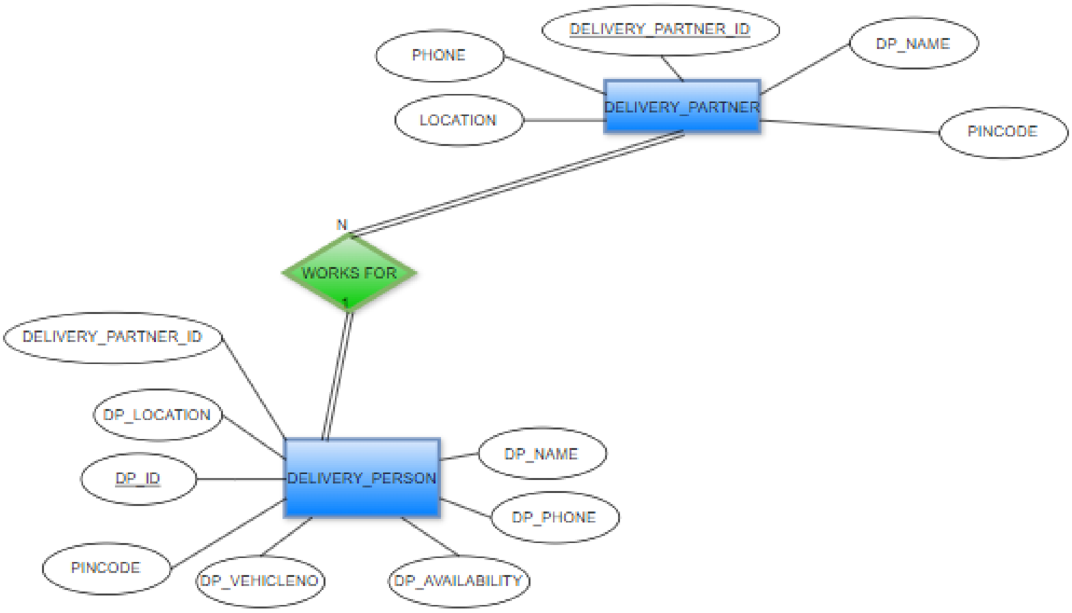
2)USER IDENTIFIES AS RESTAURANT_OWNER



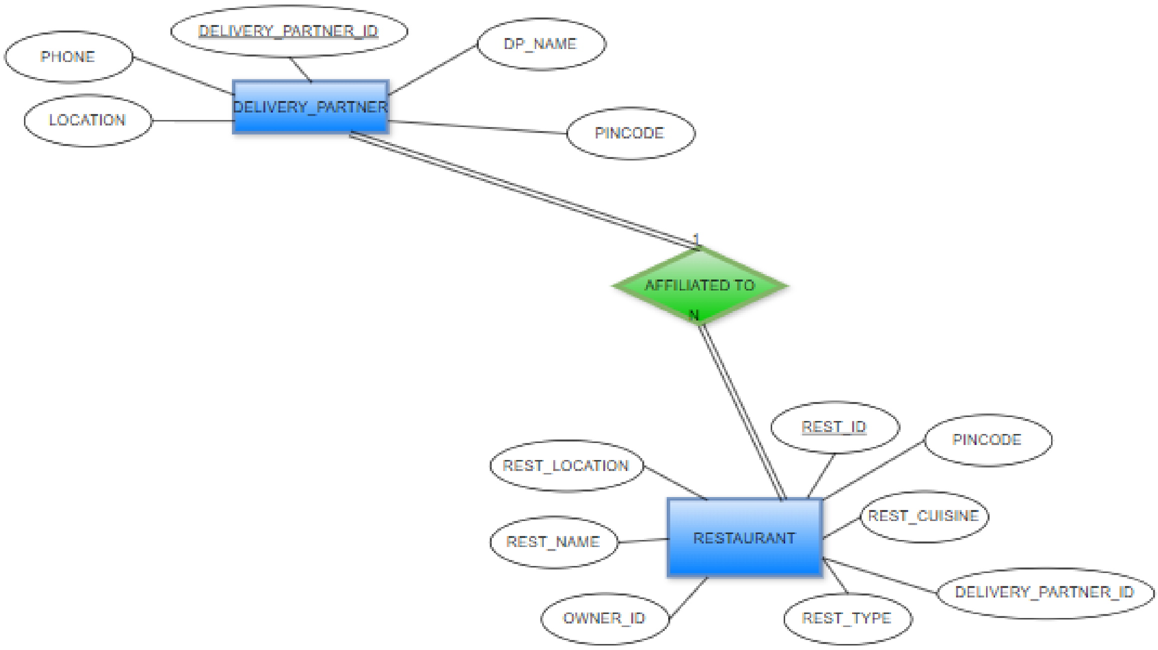
3)USER IDENTIFIES AS DELIVERY_PERSON



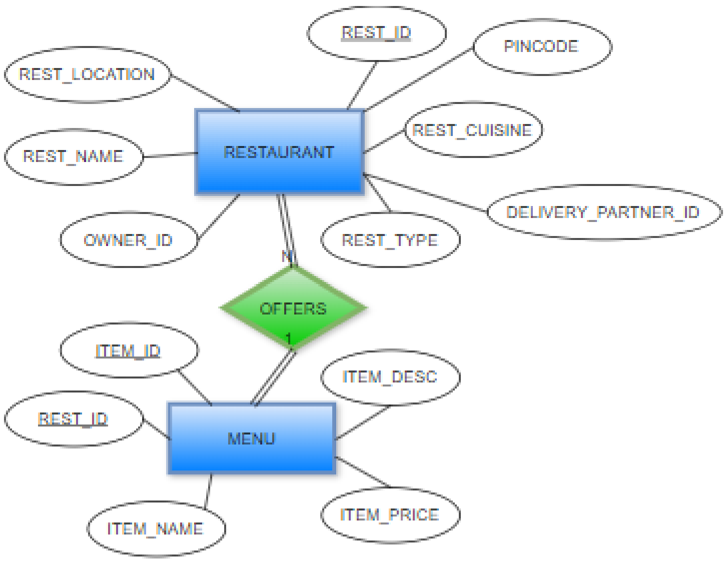
4)DELIVERY_PERSON WORKS FOR DELIVERY_PARTNER



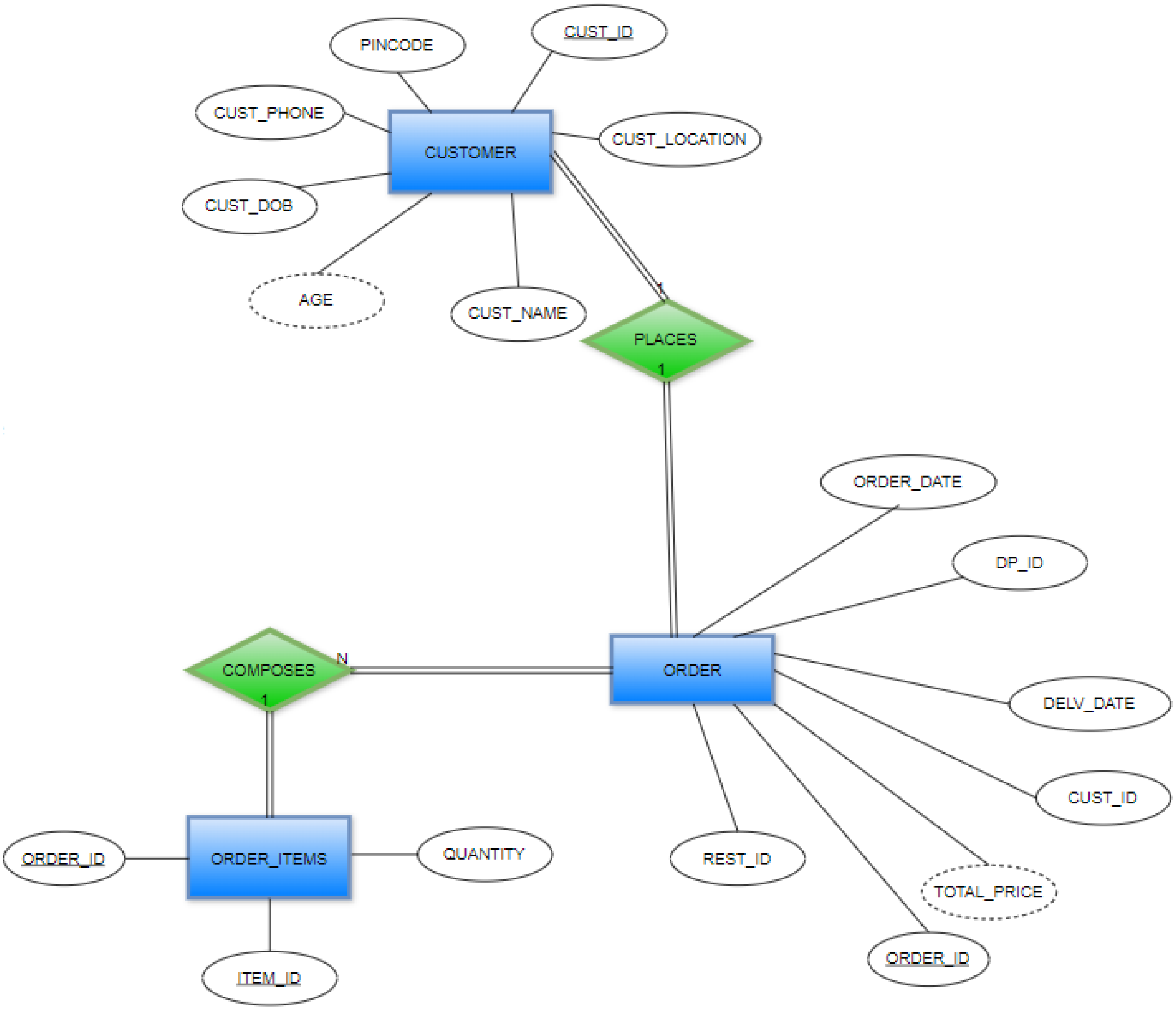
5)DELIVERY_PARTNER IS AFFILIATED TO A RESTAURANT



6)RESTAURANT OFFERS MENU



7)CUSTOMER PLACES ORDER



ORDER TABLE:

MULTIVALUED ATTRIBUTE : item_id

RELATIONAL SCHEMA:

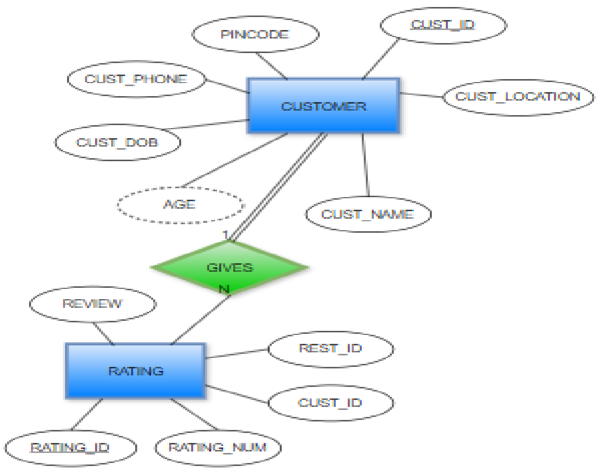
ORDER:

Order_id	Cust_id	Delv_person_id	Order_date	Delv_date	Cust_id	Rest_id	Total_price

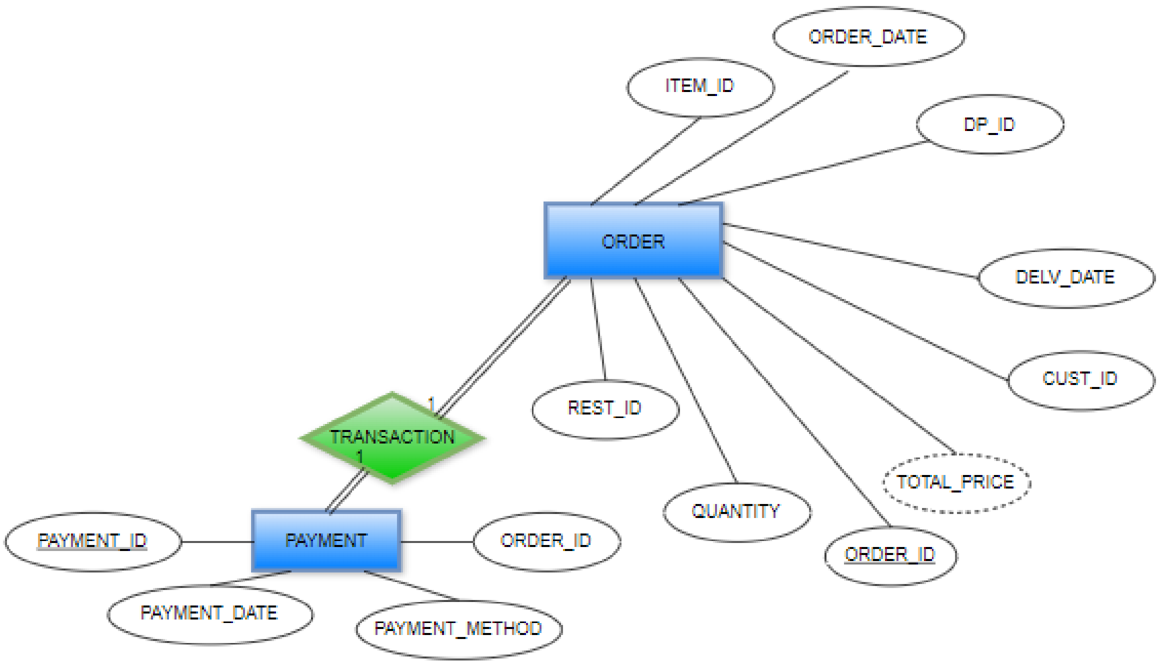
ORDER_ITEM:

Order_id	Item_id	quantity

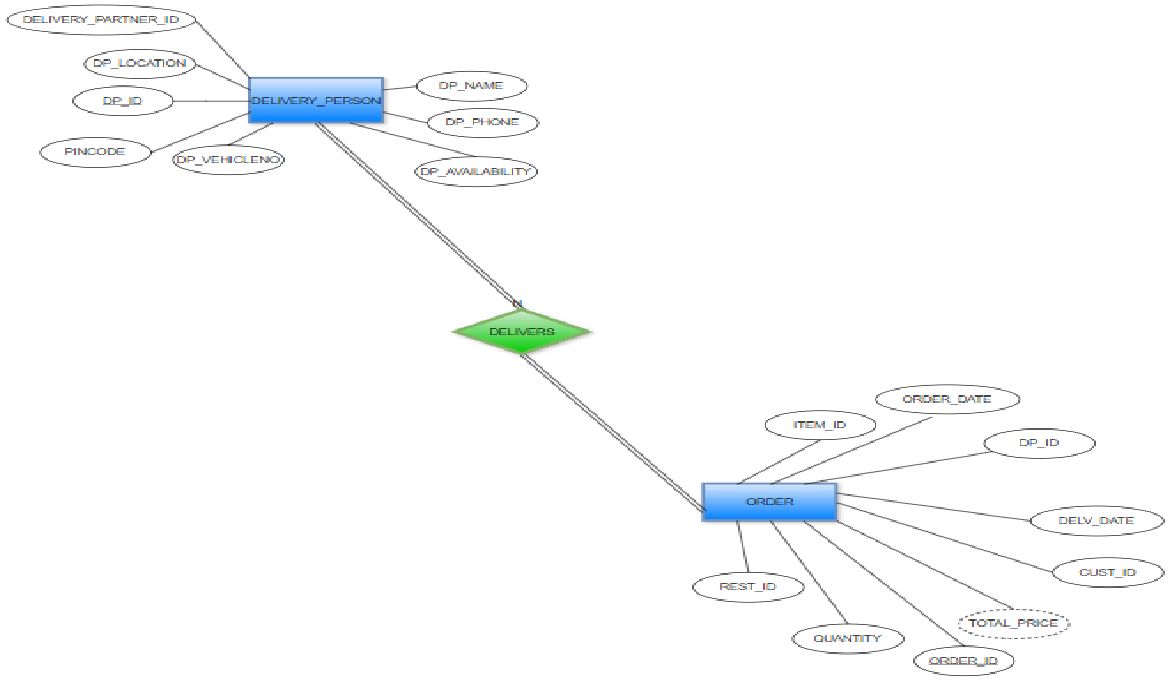
8)CUSTOMER GIVES RATING



9)ORDER TRANSACTIONS PAYMENT



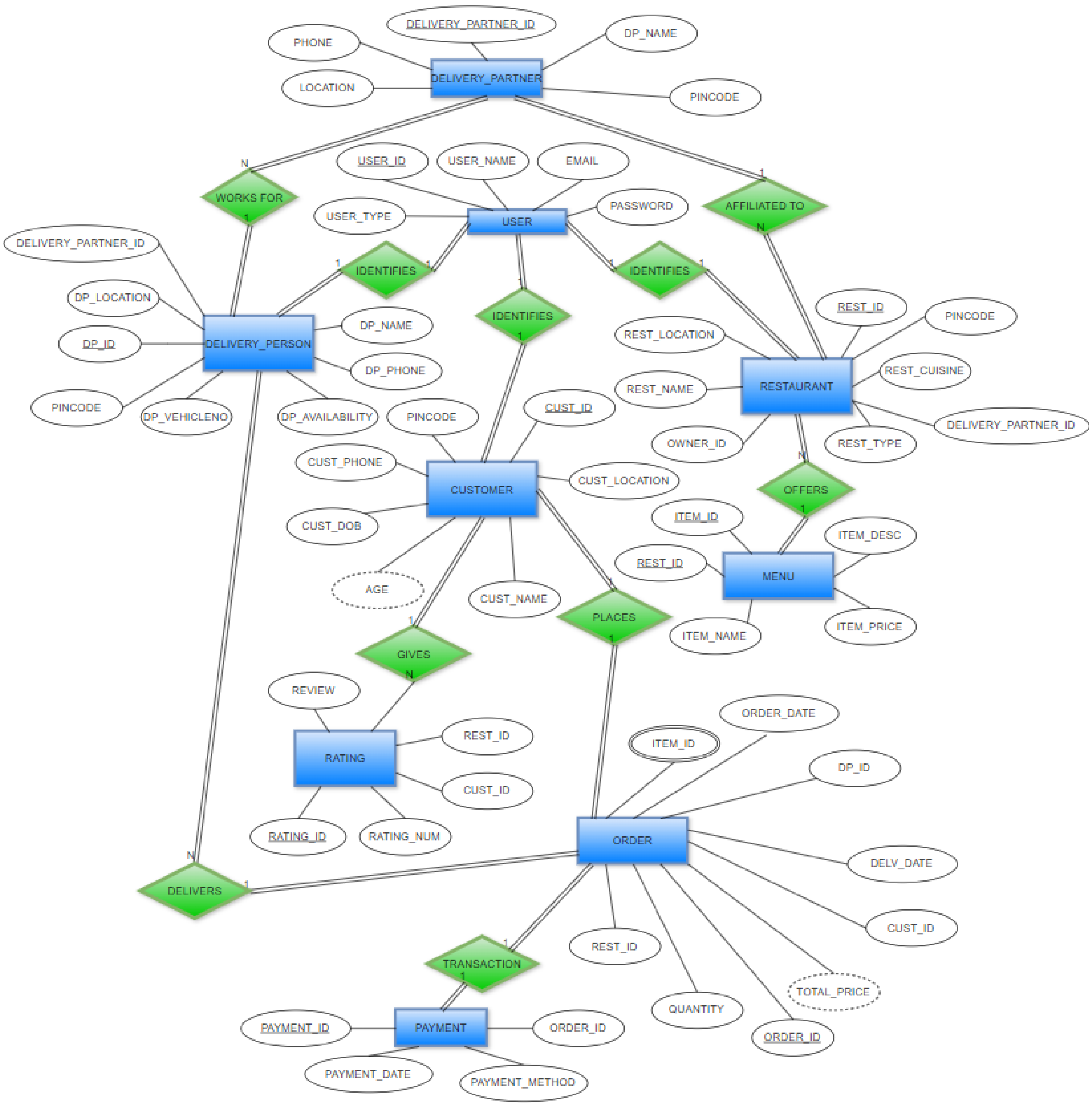
10)DELIVERY_PERSON DELIVERS ORDER



ER DIAGRAM:

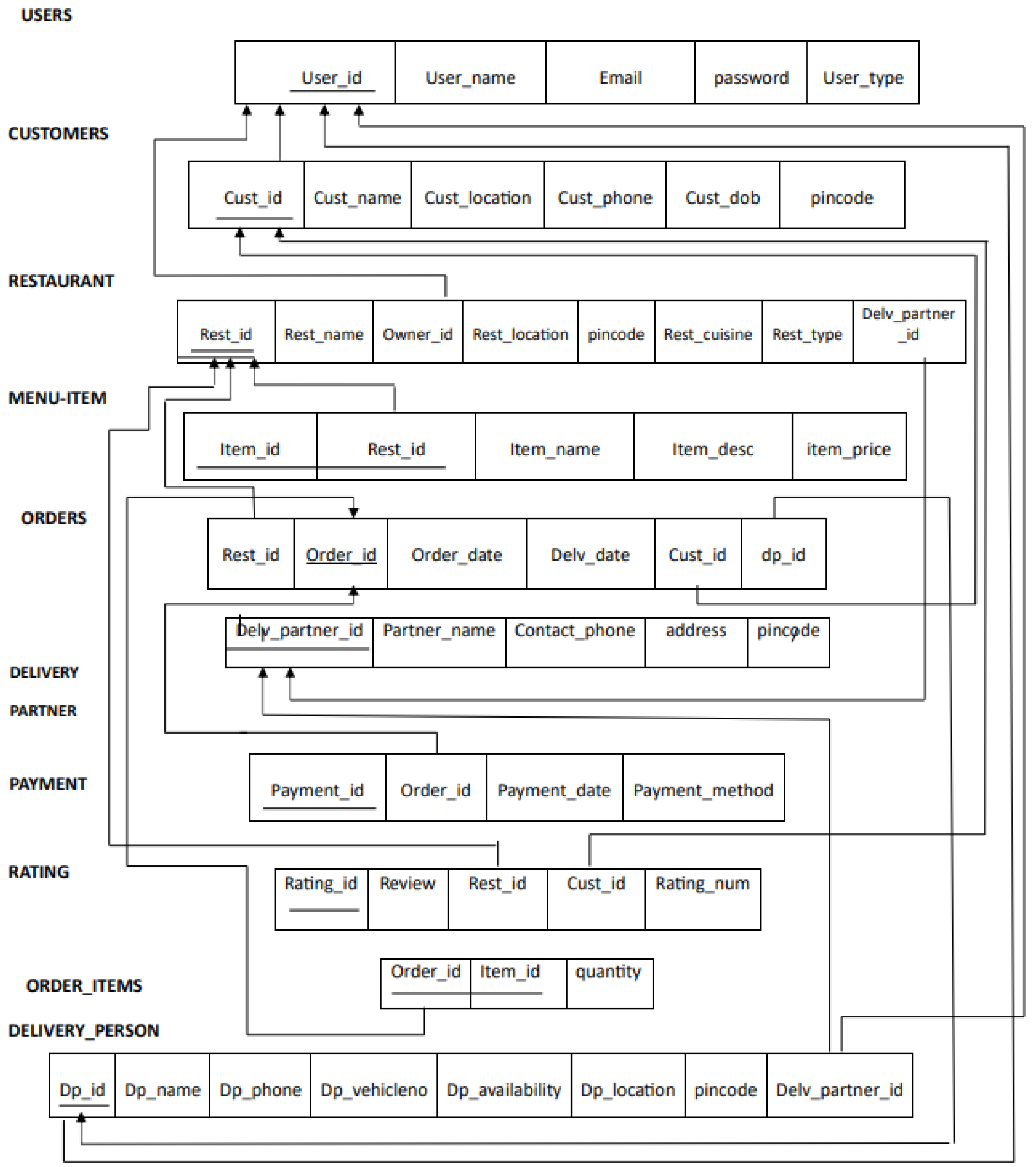
ASSUMPTIONS:

1) No restaurant can have more than one owner.
2) Each person has a unique username, email and phone number.
3) Each delivery Partner maintains only one phone number(admin/helpline number)
4) Each restaurant is affiliated to only one brand(optimality), but several delivery persons can work under the same brand.
5) The delivery person delivering the order depends on the delivery brand under which he is working, when affiliated to a restaurant the order is placed from.
6) The customer can place several items in an order from only one restaurant. Ordering from different restaurants in the same order is a limitation here.
7) Each restaurant has only five Items in the menu.



DELIVERY_PARTNER_ID
PHONE
DP_NAME
DELIVERY_PARTNER
LOCATION
PINCODE
USER_ID
USER_NAME
EMAIL
N
1
WORKS FOR
PASSWORD
AFFILIATED TO
USER_TYPE
USER
N
DELIVERY_PARTNER_ID
IDENTIFIES
IDENTIFIES
IDENTIFIES
DP_LOCATION
REST_ID
PINCODE
DP_NAME
DP_ID
DELIVERY_PERSON
REST_LOCATION
DP_PHONE
REST_CUISINE
REST_NAME
RESTAURANT
PINCODE
DP_VEHICLENO
DP_AVAILABILITY
PINCODE
CUST_ID
DELIVERY_PARTNER_ID
OWNER_ID
REST_TYPE
CUST_PHONE
CUSTOMER
CUST_LOCATION
OFFERS
ITEM_ID
ITEM_DESC
CUST_DOB
REST_ID
MENU
AGE
CUST_NAME
PLACES
ITEM_NAME
ITEM_PRICE
GIVES
REVIEW
ORDER_DATE
REST_ID
ITEM_ID
RATING
DP_ID
CUST_ID
RATING_ID
RATING_NUM
ORDER
DELIVERS
DELV_DATE
CUST_ID
REST_ID
TRANSACTION
TOTAL_PRICE
QUANTITY
PAYMENT_ID
PAYMENT
ORDER_ID
ORDER_ID
PAYMENT_DATE
PAYMENT_METHOD

SCHEMA DIAGRAM(BEFORE NORMALIZATION):



NORMALIZATION OF TABLES

1)USER

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functional dependent on the primary key.

Since user_id is the primary key, and username, email, password and user_type depend on it fully, the table is already in 2NF.

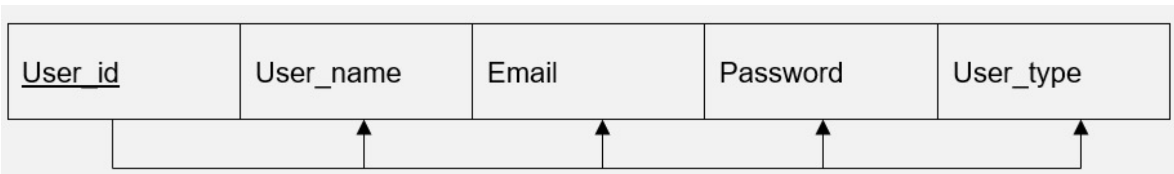
3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

There are no transitive dependencies in the table as user_id directly determines username, email, password and user_type.

Thus, the table is already in 3NF.

Initial schema



Final table:

User_id	User_name	Email	Password	User_type

....

2)CUSTOMERS

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functional dependent on the primary key.

Since cust_id is the primary key, and cust_name, cust_location, cust_phone, cust_dob, pincode depend on it fully, the table is already in 2NF.

3NF (Third Normal Form)

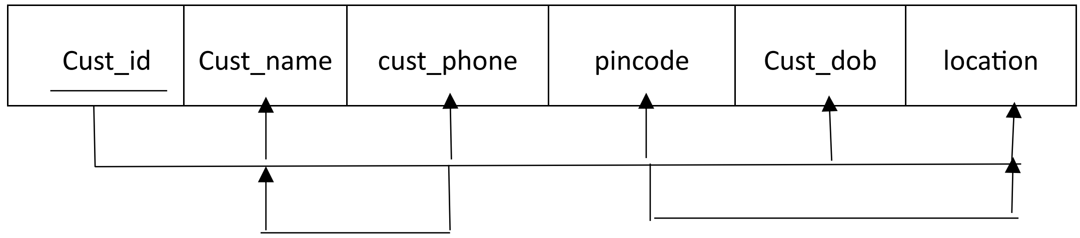
3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

In this table:

Check for transitive dependencies:

In this case, address and pincode have a transitive dependency as each address is associated with a specific pincode. To eliminate this transitive dependency, we split the table into two tables: one for customers and one for addresses.

Initial schema



After normalization using 3nf:

Customer:

Cust_id	Cust_name	Cust_phone	pincode	Cust_dob

Pincode table:

pincode	location

Final tables

Customer table

Cust_id	Cust_phone	pincode	Cust_dob

Pincode table

pincode	location

Contact table

Cust_phone	name

By normalizing the original table structure into Customers and Addresses tables, we ensure the database is in 3NF. This normalization process eliminates redundancy, maintains data integrity, and prevents update anomalies, leading to a more efficient and reliable database design.

3)RESTAURANTS

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functional dependent on the primary key.

Since rest_id is the primary key, and rest_name, owner_id, location, pincode, rest_cuisine, rest_type depend on it fully, the table is already in 2NF.

3NF (Third Normal Form)

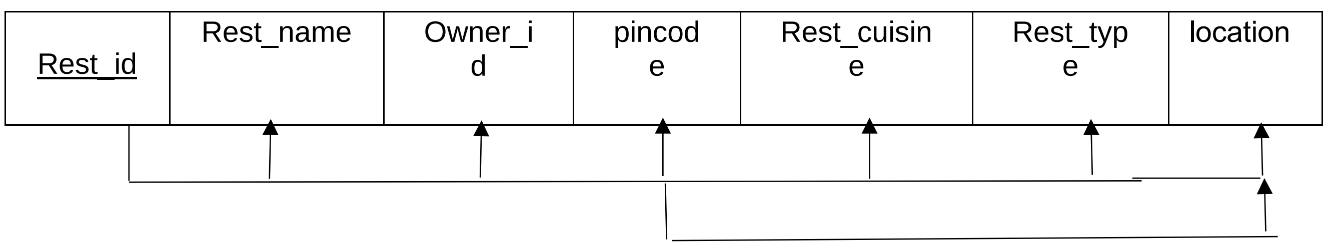
3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

In this table:

Check for transitive dependencies:

In this case, address and pincode have a transitive dependency as each address is associated with a specific pincode. To eliminate this transitive dependency, we split the table into two tables: one for customers and one for addresses.

Initial schema



After normalization using 3nf:

Final tables

Restaurant

Rest_id	Rest_name	Owner_id	pincode	Rest_cuisine	Rest_type

Pincode table:

Pin code	location

4)MENU_ITEMS

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The MENU_ITEMS table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the primary key.

The primary key is itemid.

The non-key attributes (restid, name, description, price) all depend fully on itemid.

Therefore, the MENU_ITEMS table is in 2NF.

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

There are no transitive dependencies in the table because all non-key attributes (restid, name, description, price) directly depend on the primary key (itemid).

Therefore, the MENU_ITEMS table is in 3NF.

Initial schema

item_id	rest_id	name	description	price

Final table

item_id	rest_id	name	description	price

---

5)Orders

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table already seems to comply with 1NF, as all columns contain atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the entire primary key.

It is already in 2nf as there are no partial dependencies

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all attributes are dependent only on the primary key, with no transitive dependencies.

There are no transitive dependencies in the table. Thus, it is already in 3NF.

Initial Schema

Order_id	Cust_id	Delv_person_id	Order_date	Delv_date	Cust_id	Rest_id

Final tables

Order_id	Cust_id	Delv_person_id	Order_date	Delv_date	Cust_id	Rest_id

6)ORDER_ITEMS

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table already seems to comply with 1NF, as all columns contain atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the entire primary key.

It is already in 2nf as there are no partial dependencies

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all attributes are dependent only on the primary key, with no transitive dependencies.

There are no transitive dependencies in the table. Thus, it is already in 3NF.

Initial Schema

Order_id	Item_id	quantity

Final tables

Order_id	Item_id	quantity

7)Delivery Person

1NF (First Normal Form)

1NF requires that all columns contain atomic values and that each column contains only one value per row.

The DELIVERY_PERSON table already complies with 1NF as all columns contain atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the primary key.

The primary key is Dp_id.

All attributes (Dp_name, Dp_phone, Dp_vehicleno, Dp_availability, Dp_location, pincode, Delv_partner_id) depend on Dp_id fully.

Therefore, the DELIVERY_PERSON table is in 2NF as there are no partial dependencies.

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all attributes are dependent only on the primary key, with no transitive dependencies.

In the DELIVERY_PERSON table:

Dp_location is functionally dependent on pincode, which is a non-key attribute, creating a transitive dependency.

Dp_name is functionally dependent on Dp_phone, which is a non-key attribute, creating a transitive dependency.

To remove this transitive dependency, we decompose the table:

Decomposition to Achieve 3NF

Initial Schema

Dp_id	Dp_name	Dp_phone	Dp_vehno	Dp_availability	pincode	Delv_partner_id	location

After normalization using 3nf

DELIVERY_PERSON Table

Dp_id	Dp_name	Dp_phone	Dp_vehno	Dp_availability	pincode	Delv_partner_id

Location table

Pin code	location

Final tables

Delivery_person

Dp_id	Dp_phone	Dp_vehno	Dp_availability	pincode	Delv_partner_id

location

Pin code	location

contact

Dp_phone	Dp_name

8)DELIVERY PARTNER

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The DELIVERY_PARTNER table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the primary key.

The primary key is delivery_partner_id.

The non-key attributes (partner_name, phone, address, pincode) all depend fully on delivery_partner_id.

So, table is in 2NF.

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

There is a transitive dependency because pincode determines address. To remove this, we need to create a separate table for pincode and address.

There is also a transitive dependency because phone determines partner name. To remove this, we need to create a separate table for partner name and phone number.

Initial Schema

delivery_partner_id	partner_name	phone	address	pincode

After normalization using 3nf

DELIVERY_PARTNER Table

delivery_partner_id	partner_name	phone	pincode

Location Table

Pin code	address

Final tables

Delivery_partner

delivery_partner_id	phone	pincode

location

Pin code	address

contact

Phone	name

---

9)Payment

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The given table is already in 1NF because each column contains atomic values.

2NF (Second Normal Form)

2NF requires that the table be in 1NF and that all non-key attributes are fully functional dependent on the primary key.

Since payment_id is the primary key, and Order_id, Payment_date, and Payment_method depend on it fully, the table is already in 2NF.

3NF (Third Normal Form)

3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

There are no transitive dependencies in the table as Payment_id directly determines Order_id, Payment_date, and Payment_method.

Thus, the table is already in 3NF.

Initial Schema

Payment_id	Order_id	Payment_date	Payment_method

Final table:

Payment_id	Order_id	Payment_date	Payment_method

---

10)REVIEW

1NF (First Normal Form)

1NF requires that all columns contain atomic and indivisible values, and each column contains only one value per row.

The RATING table is already in 1NF because each column contains atomic values.

<u>2NF (Second Normal Form)</u>

2NF requires that the table be in 1NF and that all non-key attributes are fully functionally dependent on the primary key.

The primary key is rating_id.

The non-key attributes (cust_id, rest_id, review, rating_no) all depend fully on rating_id.

Therefore, the RATING table is in 2NF.

<u>3NF (Third Normal Form)</u>

3NF requires that the table be in 2NF and that all the attributes are dependent only on the primary key, and not on any other non-key attribute (i.e., there should be no transitive dependency).

There are no transitive dependencies in the table because all non-key attributes (cust_id, rest_id, review, rating_no) directly depend on the primary key (rating_id).

Therefore, the RATING table is in 3NF.

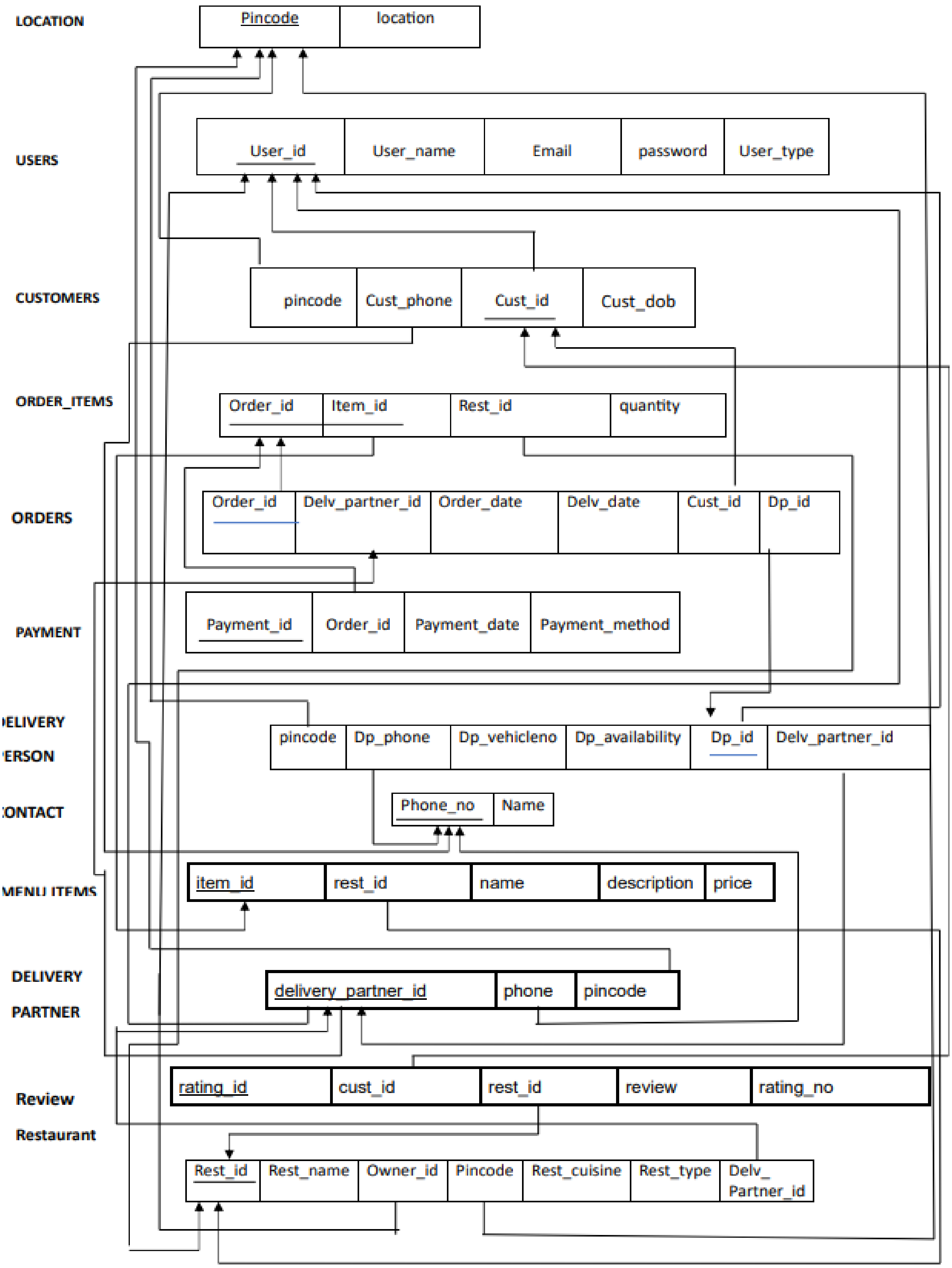
<u>Initial Schema</u>

rating_id	cust_id	rest_id	review	rating_no

<u>Final Table:</u>

<u>rating_id</u>	cust_id	rest_id	review	rating_no

3)SCHEMA DIAGRAM (AFTER NORMALIZATION)



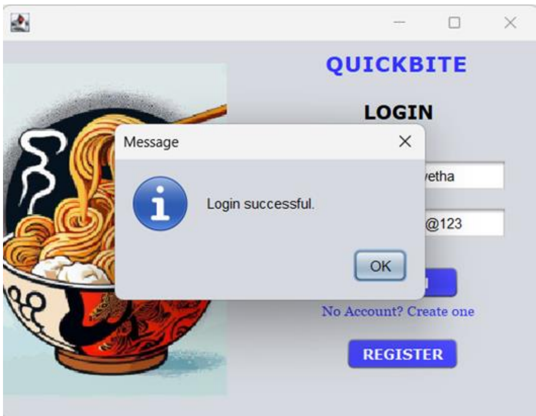
NOVELTY:

- Every entity(customer, restaurant owner, delivery_partner and delivery_person) using the database is allowed to register to the database through a common platform.
- Every order is delivered after 30 seconds to the user for a more optimal database inserting and updating approach and the customer can also confirm the order once its delivered.
- Integration of bill generation as well to intimate the customer about the order details.
- Order Preview is also integrated to intimate the customer about what they are going to order before they proceed to pay for a more user friendly interface.

IMPLEMENTATION

LOGIN PAGE

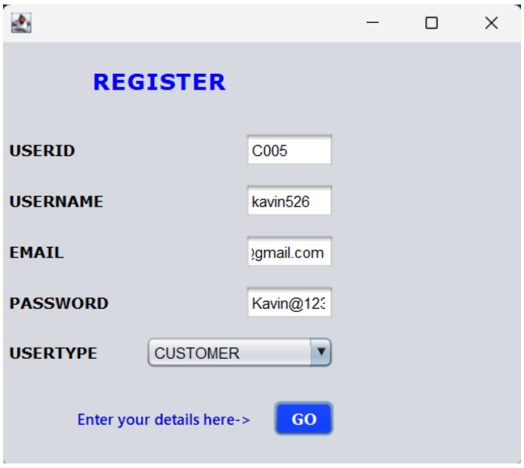
Here the Username and password is taken as input. If an username is not available in the users table, it is redirected to the registration page. If the username exists, but the password is wrong, it shows a message called password wrong. If both the username exists and the password matches the password in the users table, then it is taken to the page where all the restaurants are shown to order from.

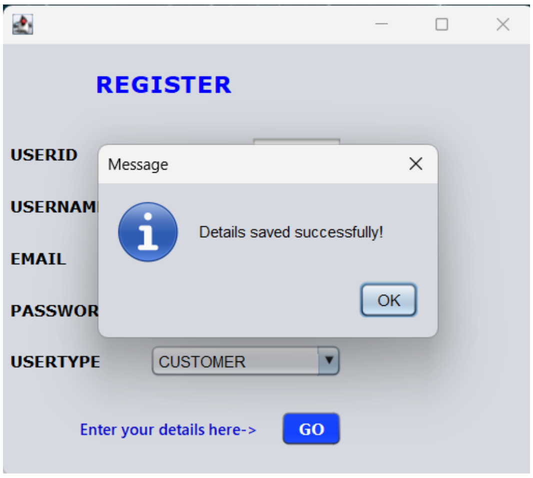


REGISTRATION PAGES

ACCOUNT REGISTRATION:

if account doesn't exist, then it is taken to the account registration page. Here, the userid,username,password,email and usertype is taken and inserted into the users table. When the go button is clicked, it reads the component selected in the combo box and leades to the required registration page.



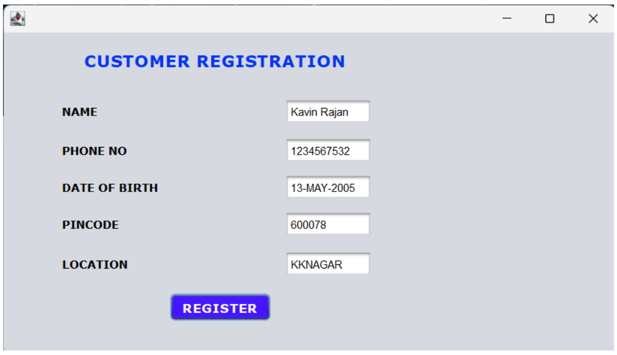


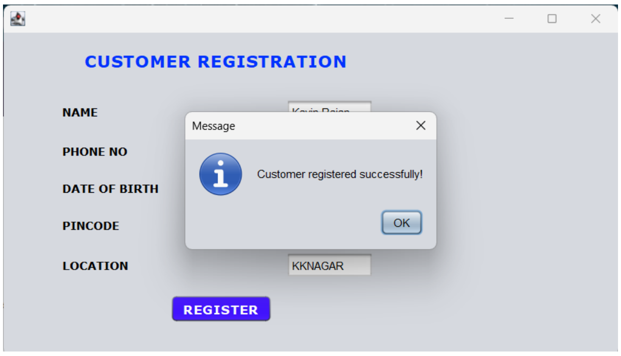
CUSTOMER REGISTRATION:

Here the customer is being registered.

If customer is selected in the combo box, then on clicking go, it leads to this page.

Here the customer name, phone no, pincode, location are taken and all these values are inserted into the customer table, name, phone into contact table, pincode and location into location table. Once we click on register, it navigates back to the login page.



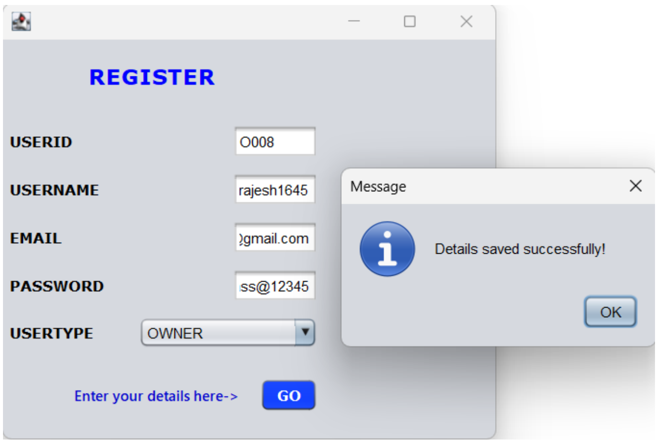


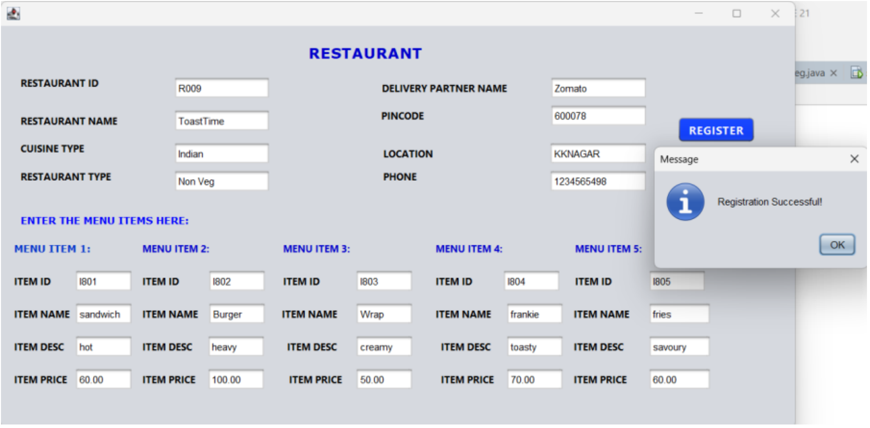
RESTAURANT REGISTRATION:

Here the restaurant is being registered.

If owner is selected in the combo box, then on clicking go, it leads to this page.

Once owner selects his details and clicks on go in the register page, it goes to this page, where the restaurant details like name, cuisine type, restaurant type, location, pincode,rid,phone no and dp_id are takenand inserted into the restaurant table along with name, phone into contact table, pincode and location into location table. Once we click on register, it navigates back to the login page. Also, all the menu items must also be given in the same registration frame. When clicked on register, all the inserting and navigating happens.



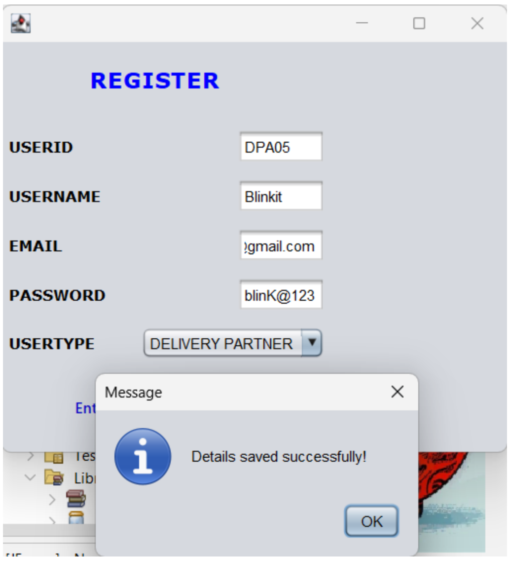


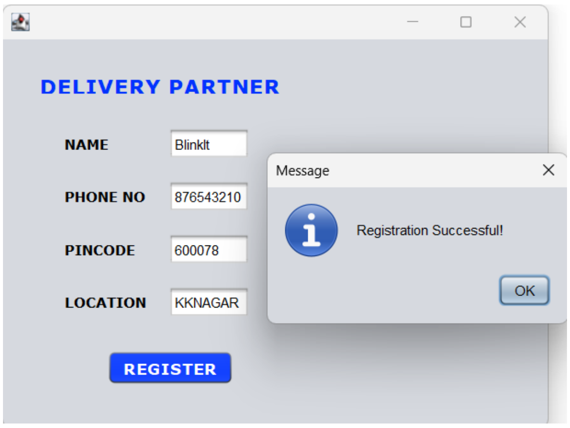
DELIVERY PARTNER REGISTRATION:

Here the delivery partner is being registered.

If delivery partner is selected in the combo box, then on clicking go, it leads to this page.

Inside this page name, the pincode and location and phone no of the delivery partner and insert into delivery_partner table, along with name, phone into contact table, pincode and location into location table. Once we click on register, it navigates back to the login page.



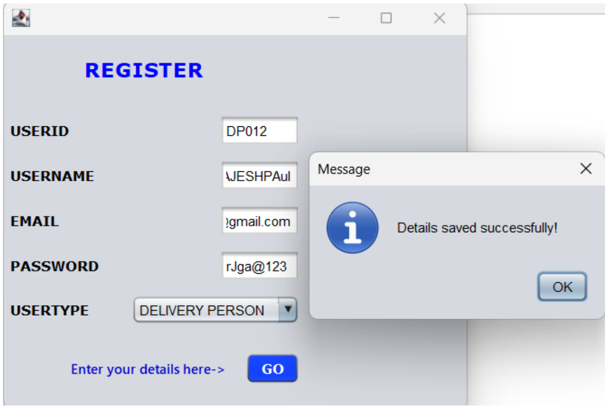


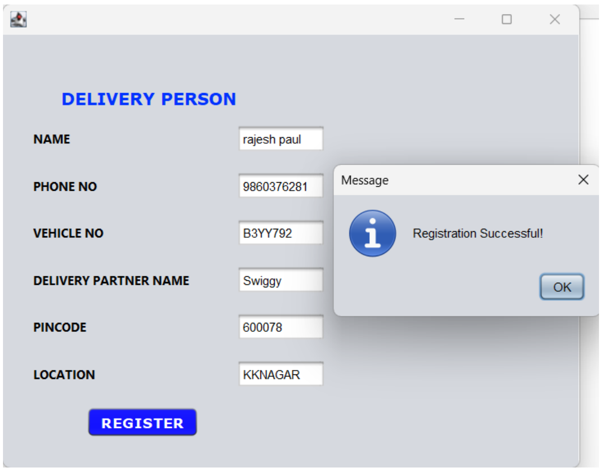
DELIVERY PERSON REGISTRATION:

Here the delivery person is being registered.

If delivery_person is selected in the combo box, then on clicking go, it leads to this page.

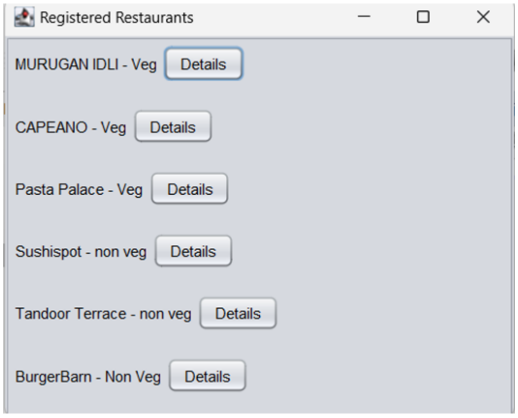
In this page, the delivery person name, the delivery partner name, his location, pincode and phone no and vehicle no is taken and inserted into the delivery person table and when clicked on register, it results in inserting all the values to the specific pages and navigating back to the login page.





SHOW RESTAURANTS PAGE:

This page is shown when the login is successful. The user is required to select from one of the restaurants present and when he selects on one of the restaurants details, it navigates to the orders preview page.

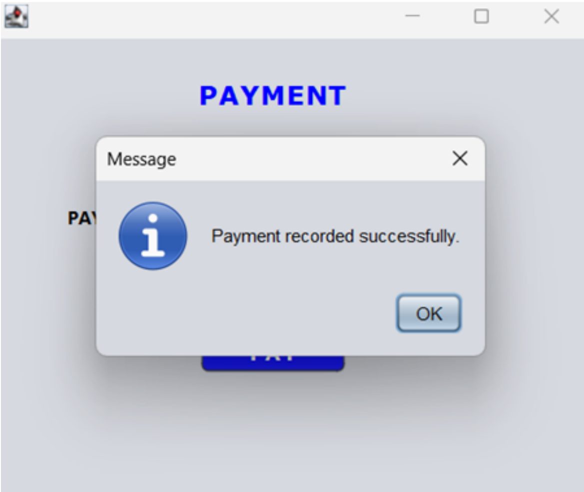


Orders:

When restaurant is selected, it leads to the frame containing all the items in the menu provided by the restaurant each menu item having a text field to select quantity and to select the item. Once select is clicked, it toggles to deselect and a preview panel will open to show the items and their quantity selected.

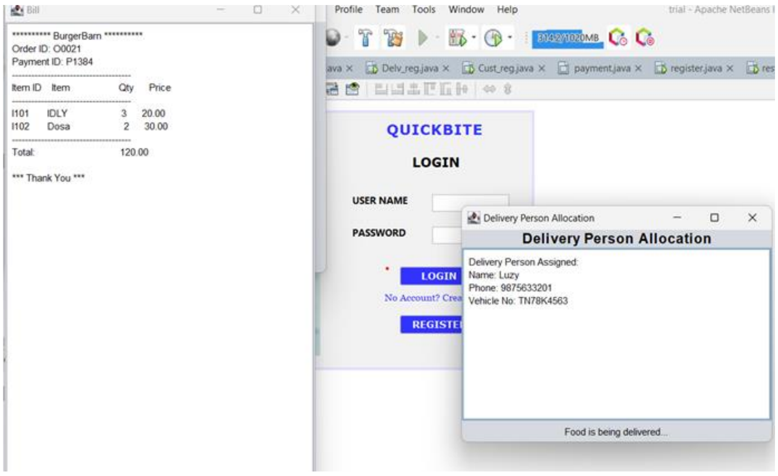
PAYMENT:

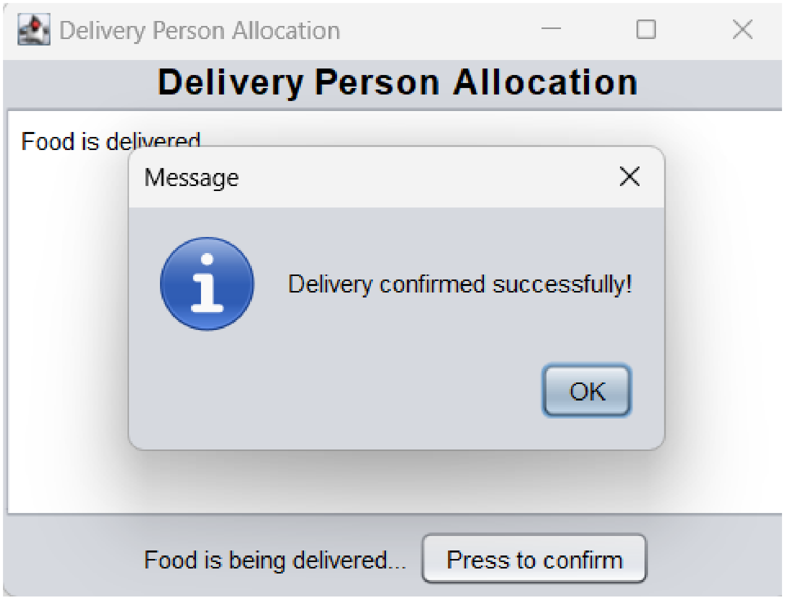
Once the proceed button is clicked on the previous frame, it navigates here, where bill is generated with total amount and the payment method is asked and once given, the delivery person is allotted to the order.



DELIVERY PERSON:

Once the payment method is inputted, it results in the delivery person being allotted according to the algorithm. After that, we wait for 30 seconds for the person to deliver the order. After that, a frame asking the user to confirm the order is used. Once confirmed, it will go back to the login page.





CONCLUSION

Our Food Delivery Management System makes the delivery process easier and faster, ensuring timely deliveries and efficient tracking of delivery staff. This system improves customer satisfaction by reducing wait times and helps restaurants manage deliveries better, boosting their service quality and efficiency.