DOORDASH-3: PROJECT REPORT

PROJECT TITLE: DOORDASH

COURSE NO-SECTION NO.: CS 6360.001

TEAM NUMBER: TEAM - 18 (DOORDASH-3)

TEAM MEMBERS(NET-IDs):

1. KRISHNA TEJA YELISETTI (KXY200016)

2. SATYA SOMEPALLI (SXS190436)

3. SNEHA SIRIPURAPU (SXS200117)

DATA REQUIREMENTS:

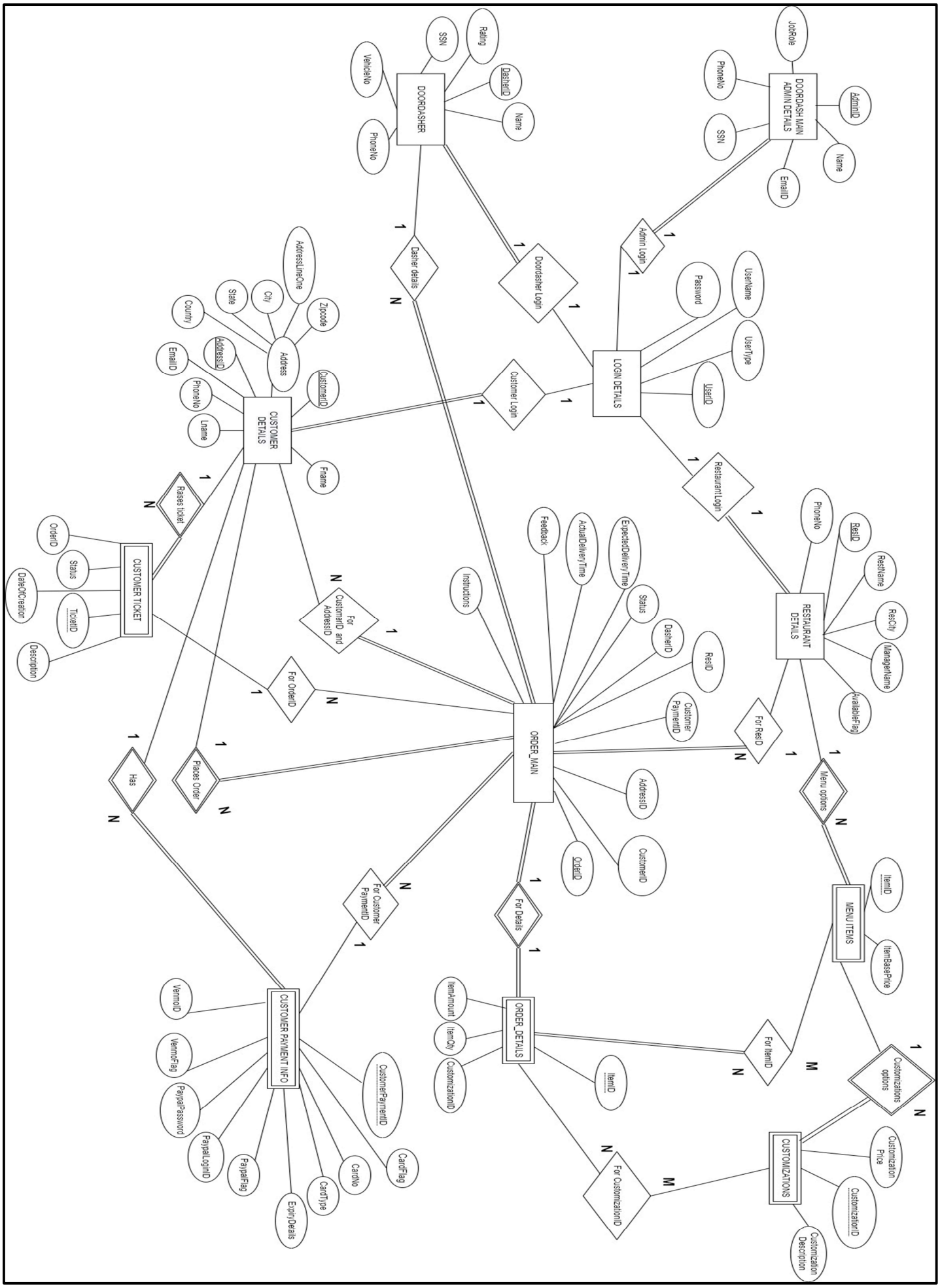
Main Actors involved:

1. Customer
2. Restaurant Admin
3. Door Dasher
4. DoorDash Admin

The database follows the below functionalities:

1. There are multiple users in our Doordash application like Customer, Restaurant_admin and DoordDasher along with Doordash_admin.
2. A customer can register into the Doordash application and login into the application using their credentials.
3. A customer can search for the restaurants.
4. A customer can select items from a particular restaurant only in the cart.
5. A customer can add items to his/her cart.
6. A customer can check out from his/her cart and place the order.
7. A customer can make payment for the order and can have three types of payment methods: Card, PayPal, and Venmo. A customer can have multiple cards.
8. A customer can check the status of the order.
9. A customer can contact the restaurant for any queries.
10. A Restaurant Admin can add/edit/delete his/her restaurant profile in Doordash.
11. A Restaurant Admin can add/edit/delete menu items of his/her restaurant.
12. When an order is placed by the customer, the restaurant gets notifications.
13. A Door Dasher gets a notification from all the nearby restaurants within a specific radius after the restaurant accepts the customers' orders.
14. A Door Dasher can select an available order from the received notifications according to his/her choice.
15. A Door Dasher receives a notification when the order is ready in the selected restaurant.
16. The restaurant will keep changing the status of the order until it is picked by the dasher and after that, the dasher will change the status of the order until it is delivered.
17. Customers can give feedback on an order and also give ratings.
18. Customers can also raise any number of tickets on a specific order in case of any problems or queries. Customers can raise tickets in general also without involving any order in the ticket.
19. Doordash admin will have all privileges like edit or delete Customer details or Restaurant details or Doordasher details.
20. Admin will look over the raised tickets and resolve them timely and change the status of the ticket accordingly.

ER-DIAGRAM:


DOORDASH MAIN ADMIN DETAILS
JobRole
AdminID
PhoneNo
SSN
Name
EmailID
DOORDASHER
Rating
SSN
VehicleNo
DasherID
PhoneNo
Name
Admin Login
Dasher details
Doordasher Login
LOGIN DETAILS
UserName
Password
UserType
UserID
Customer Login
CUSTOMER DETAILS
AddressLineOne
City
State
Country
Zipcode
Address
AddressID
EmailID
PhoneNo
CustomerID
Lname
Fname
RestaurantLogin
RESTAURANT DETAILS
PhoneNo
ResID
RestName
ResCity
ManagerName
AvailableFlag
Raise ticket
CUSTOMER TICKET
OrderID
Status
TicketID
DateOfCreation
Description
For CustomerID and AddressID
For OrderID
Places Order
Has
ORDER_MAIN
Instructions
FeedBack
ActualDeliveryTime
ExpectedDeliveryTime
Status
DasherID
ResID
Customer PaymentID
AddressID
CustomerID
OrderID
For ResID
Menu options
MENU ITEMS
ItemID
ItemBasePrice
For Details
For Customer PaymentID
ORDER_DETAILS
ItemAmount
ItemQty
CustomizationID
ItemID
For ItemID
Customizations options
CUSTOMIZATIONS
Customization Price
CustomizationID
Customization Description
For CustomizationID
CUSTOMER PAYMENT INFO
VenmoID
VenmoFlag
PaypalPassword
PaypalLoginID
PaypalFlag
ExpiryDetails
CardType
CardNo
CardFlag
CustomePaymentID

ASSUMPTIONS:

1. Login credentials of every user type (let it be Customer, Restaurant admin, Doordasher or Doordash Admin) will be stored in a single table and will be recognized using a unique UserID allocated to every user.
2. A customer can have multiple addresses, that's the reason why a Customer along with a specific address will be uniquely recognized while placing an Order.
3. If a customer doesn't have any payment method, he/she will have to add at least one payment method first to place an order.
4. A customer can have only a maximum of one customization for a selected item in the cart. For e.g., if an item 'A' has C1 and C2 customizations available, then while placing an order Customer can only choose a maximum of one customization among C1 and C2 for item 'A' or none at all.
5. Customer Payment Info and Customer Ticket are weak entities in our scenario, i.e., although a ticket has a ticketid or a payment method has a payment id, it will be uniquely recognized if it is combined with its specific owner,i.e., CustomerID in our case. For example, there can be tickets with ticket-ids 1 & 2 for customer-ids 100 and 101 both and these can be uniquely recognized only if customer-id and ticket-id are combined. The same type of example can be applied to payment methods.
6. A ticket can be raised on a specific order or can be raised in general without involving any order. For e.g, A customer can raise a ticket on orderid 'N' or he/she can raise a general ticket for asking a general query. In the latter case, orderid details won't be stored in the database.
7. Similarly, Menu items and Customization ids for a specific item are also weak entities in our scenario where item-id values and customization-id values can repeat for different restaurants and different items in the same restaurant respectively.

ER-DIAGRAM TO RELATIONAL SCHEMA:

LOGIN_DETAILS:

UserID	UserType	Username	Password

Primary Key: UserID

DOORDASH_MAIN_ADMIN DETAILS:

AdminID	Name	EmailID	SSN	PhoneNo	JobRole

Primary Key: AdminID
Foreign key references: DOORDASH_MAIN_ADMIN_DETAILS(AdminID) ----> LOGIN_DETAILS(UserID)

DOORDASHER:

DasherID	Name	PhoneNo	VehicleNo	SSN	Rating

Primary Key: DasherID
Foreign key references: DOORDASHER (DasherID) ----> LOGIN_DETAILS(UserID)

RESTAURANT_DETAILS:

ResID	RestName	ResCity	PhoneNo	ManagerName	ResRating	AvailableFlag

Primary Key: ResID
Foreign key references: RESTAURANT_DETAILS (ResID) ----> LOGIN_DETAILS(UserID)

MENU_ITEMS:

ResID	ItemID	ItemBasePrice

Primary Key: ResID, ItemID
Foreign key references: MENU_ITEMS(ResID) ----> RESTAURANT_DETAILS (ResID)

CUSTOMIZATIONS:

ResID	ItemID	CustomizationID	CustomizationPrice	CustomizationDescription

Primary Key: ResID, ItemID, CustomizationID
Foreign key references: CUSTOMIZATIONS (ResID) ----> RESTAURANT_DETAILS (ResID),
CUSTOMIZATIONS (ItemID) ----> MENU_ITEMS (ItemID)

CUSTOMER_DETAILS:

CustomerID	AddressID	FName	LName	PhoneNo	EmailID	AddressLineOne	City	State	Zipcode	Country

Primary Key: CustomerID, AddressID
Foreign key references: CUSTOMER_DETAILS (CustomerID) ----> LOGIN_DETAILS(UserID)

CUSTOMER_PAYMENT_INFO:

CustomerID	CustomerPaymentID	CardFlag	CardNo	CardType	ExpiryDetails	PaypalFlag	PaypalLoginID	PaypalPassword	VenmoFlag	VenmoID

Primary Key: CustomerID, CustomerPaymentID
Foreign key references: CUSTOMER_PAYMENT_INFO(CustomerID) ----> CUSTOMER_DETAILS (CustomerID)

CUSTOMER_TICKET:

CustomerID	TicketID	OrderID	DateOfCreation	Status	Description

Primary Key: CustomerID, TicketID
Foreign key references: CUSTOMER_TICKET (CustomerID) ----> CUSTOMER_DETAILS (CustomerID),
CUSTOMER_TICKET (OrderID) ----> ORDER_MAIN (OrderID)

ORDER_MAIN:

OrderID	CustomerID	AddressID	ResID	CustomerPaymentID	DasherID	Status	ExpectedDeliveryTime	ActualDeliveryTime	Feedback	Instructions

Primary Key: OrderID
Foreign key references: ORDER_MAIN(CustomerID) ----> CUSTOMER_DETAILS (CustomerID),
ORDER_MAIN(AddressID) ----> CUSTOMER_DETAILS (AddressID), ORDER_MAIN(ResID) ----> RESTAURANT_DETAILS (ResID),
ORDER_MAIN(CustomerPaymentID) ----> CUSTOMER_PAYMENT_INFO (CustomerPaymentID),
ORDER_MAIN(DasherID) ----> DOORDASHER(DasherID)

ORDER_DETAILS:

OrderID	ItemID	CustomizationID	ItemQty	ItemAmount

Primary Key: OrderID, ItemID
Foreign key references: ORDER_DETAILS(OrderID) ----> ORDER_MAIN (OrderID),
ORDER_DETAILS(ItemID) ----> MENU_ITEMS (ItemID),
ORDER_DETAILS(CustomizationID) ----> CUSTOMIZATIONS(CustomizationID)

NORMALIZATION OF RELATIONAL SCHEMAS:

Normalization rules:

- ➔ **Normalization -** A technique for organizing data in a database is called Normalization. A database must be normalized in order to reduce the redundancy of the data and ensure that only related data is stored in the database tables. It also prevents any issues caused by database modifications like insertions, deletions, and updations. Normalization is carried out in practice so that the resulting designs are of high quality and meet the desirable properties.
- ➔ **Normal Form - To** certify that a relation schema is in a certain normal form, we use the keys and FDs of the relation.
  - ◆ **First Normal Form**
    - First Normal Form disallows Composite attributes, multivalued attributes, and nested relations where attributes whose values for an individual tuple are non-atomic.
    - Most RDBMS allow relations to be in 1st Normal form. Our above-defined relations are in 1NF.
  - ◆ **Second Normal Form**
    - In the Second Normal Form Non-prime, the attribute is fully functional dependent on the primary key.
    - In the above CUSTOMER_DETAILS table, some non-prime attributes like FName, LName, PhoneNo, EmailID are partially dependent on the primary key that is CustomerID. So, it does not satisfy 2NF.
    - We decompose the table wherein one table CustomerID uniquely identifies FName, LName, PhoneNo, EmailID. And we retain other attributes in the original table.
  - ◆ **Third Normal Form**
    - In the Third Normal Form no non-prime attribute is transitively dependent on the primary key.
    - Our above-defined relations are in 3NF except for the above CUSTOMER_DETAILS which will be in 3NF after decomposition into CUSTOMER_DETAILS_MAIN and CUSTOMER_ADDRESS_DETAILS as shown below.

NORMALIZATION:

LOGIN_DETAILS:

UserID	UserType	Username	Password

Note: The above table doesn't need any normalization

DOORDASH_MAIN_ADMIN_DETAILS:

AdminID	Name	EmailID	SSN	PhoneNo	JobRole

Note: The above table doesn't need any normalization

DOORDASHER:

DasherID	Name	PhoneNo	VehicleNo	SSN	Rating

Note: The above table doesn't need any normalization

RESTAURANT_DETAILS:

ResID	RestName	ResCity	PhoneNo	ManagerName	ResRating	AvailableFlag

Note: The above table doesn't need any normalization

MENU_ITEMS:

ResID	ItemID	ItemBasePrice

Note: The above table doesn't need any normalization

CUSTOMIZATIONS:

ResID	ItemID	CustomizationID	CustomizationPrice	CustomizationDescription

Note: The above table doesn't need any normalization

CUSTOMER_DETAILS:

CustomerID	AddressID	FName	LName	PhoneNo	EmailID	AddressLineOne	City	State	Zipcode	Country

Here Fname, Lname, PhoneNo and EmailID specifically depends on only CustomerID. Therefore, it doesn't follow 2NF.

Normalization of CUSTOMER_DETAILS Table:

CUSTOMER_DETAILS_MAIN:

CustomerID	FName	LName	PhoneNo	EmailID

Primary Key: CustomerID

CUSTOMER_ADDRESS_DETAILS:

CustomerID	AddressID	AddressLineOne	City	State	Zipcode	Country

Primary Key: CustomerID, AddressID

Foreign key references: CUSTOMER_ADDRESS_DETAILS(CustomerID) ----> CUSTOMER_DETAILS_MAIN(CustomerID)

CUSTOMER_PAYMENT_INFO:

CustomerID	CustomerPaymentID	CardFlag	CardNo	CardType	ExpiryDetails	PaypalFlag	PaypalLoginID	PaypalPassword	VenmoFlag	VenmoID

Note: The above table doesn't need any normalization

CUSTOMER_TICKET:

CustomerID	TicketID	OrderID	DateOfCreation	Status	Description

Note: The above table doesn't need any normalization

ORDER_MAIN:

OrderID	CustomerID	AddressID	ResID	CustomerPaymentID	DasherID	Status	ExpectedDeliveryTime	ActualDeliveryTime	Feedback	Instructions

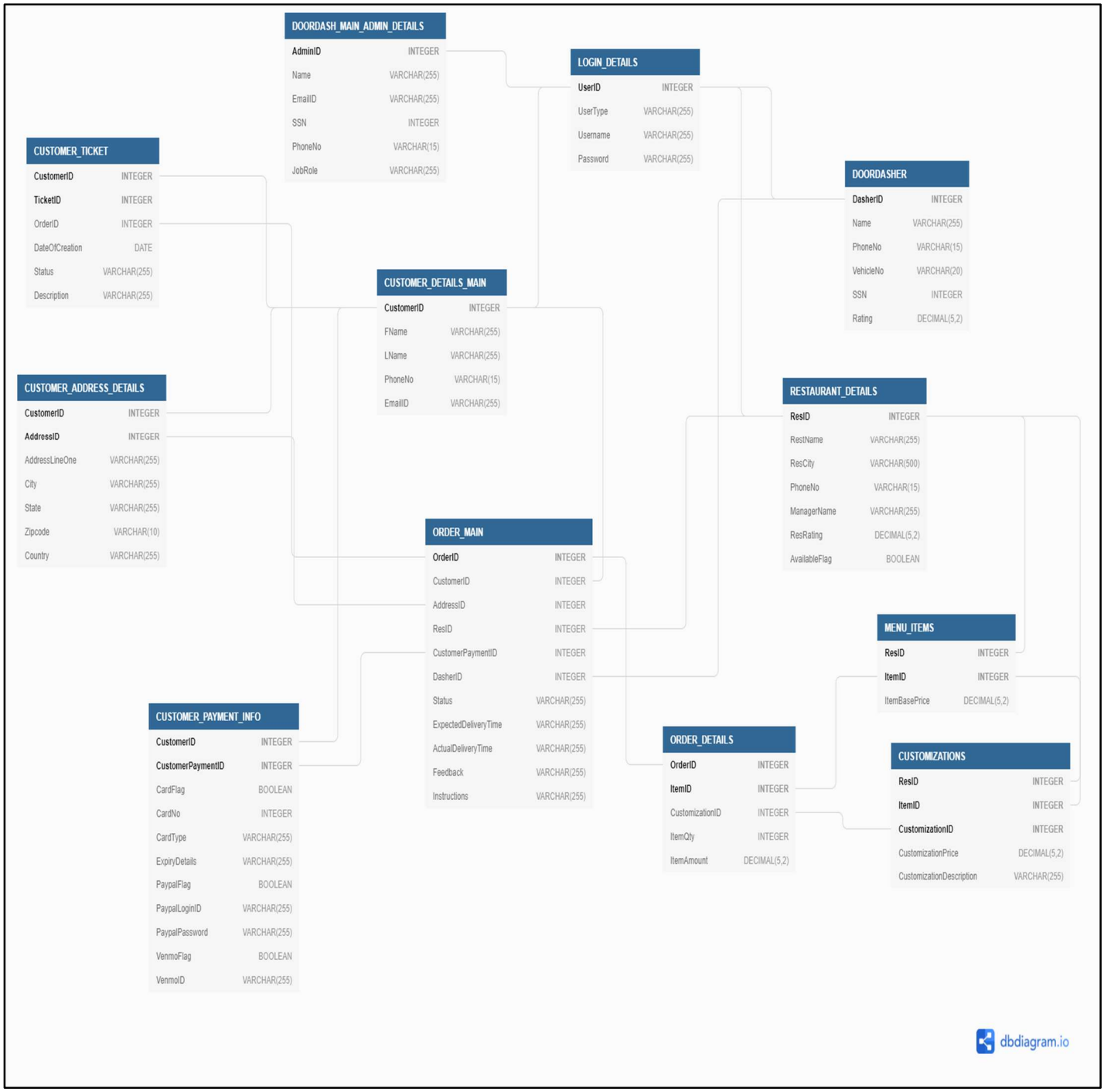
Note: The above table doesn't need any normalization

ORDER_DETAILS:

OrderID	ItemID	CustomizationID	ItemQty	ItemAmount

Note: The above table doesn't need any normalization

FINAL RELATIONAL SCHEMA AFTER NORMALIZATION:



SQL QUERIES:

CREATE TABLE QUERIES:

```
CREATE TABLE  LOGIN_DETAILS
(UserID INTEGER,
UserType VARCHAR(255) NOT NULL,
Username VARCHAR(255) NOT NULL,
Password VARCHAR(255) NOT NULL,
PRIMARY KEY (UserID)
);

CREATE TABLE  DOORDASH_MAIN_ADMIN_DETAILS
(AdminID INTEGER,
Name VARCHAR(255) NOT NULL,
EmailID VARCHAR(255) NOT NULL,
SSN INTEGER NOT NULL,
PhoneNo VARCHAR(15) NOT NULL,
JobRole VARCHAR(255) NOT NULL,
PRIMARY KEY (AdminID)
);

CREATE TABLE  DOORDASHER
( DasherID INTEGER,
Name VARCHAR(255) NOT NULL,
PhoneNo VARCHAR(15) NOT NULL,
VehicleNo VARCHAR(20) NOT NULL,
SSN INTEGER NOT NULL UNIQUE,
Rating DECIMAL(5,2),
PRIMARY KEY (DasherID)
);

CREATE TABLE  RESTAURANT_DETAILS
( ResID INTEGER,
RestName VARCHAR(255) NOT NULL,
ResCity VARCHAR(500) NOT NULL,
PhoneNo VARCHAR(15) NOT NULL,
ManagerName VARCHAR(255) NOT NULL,
ResRating DECIMAL(5,2),
AvailableFlag BOOLEAN NOT NULL,
PRIMARY KEY(ResID)
);
```

```
CREATE TABLE MENU_ITEMS
(ResID INTEGER,
ItemID INTEGER,
ItemBasePrice DECIMAL(5,2),
PRIMARY KEY(ResID,ItemID)
);

CREATE TABLE  CUSTOMIZATIONS
( ResID INTEGER,
ItemID INTEGER,
CustomizationID INTEGER,
CustomizationPrice DECIMAL(5,2) NOT NULL,
CustomizationDescription VARCHAR(255) NOT NULL,
PRIMARY KEY(ResID,ItemID,CustomizationID)
);

CREATE TABLE  CUSTOMER_DETAILS_MAIN
( CustomerID INTEGER,
FName  VARCHAR(255) NOT NULL,
LName  VARCHAR(255) NOT NULL,
PhoneNo  VARCHAR(15) NOT NULL,
EmailID  VARCHAR(255) NOT NULL,
PRIMARY KEY(CustomerID)
);

CREATE TABLE  CUSTOMER_ADDRESS_DETAILS
(CustomerID INTEGER,
AddressID INTEGER,
AddressLineOne  VARCHAR(255) NOT NULL,
City VARCHAR(255) NOT NULL,
State VARCHAR(255) NOT NULL,
Zipcode VARCHAR(10) NOT NULL,
Country  VARCHAR(255) NOT NULL DEFAULT 'USA',
PRIMARY KEY(CustomerID,AddressID)
);

CREATE TABLE  CUSTOMER_PAYMENT_INFO
(CustomerID INTEGER,
CustomerPaymentID INTEGER,
CardFlag BOOLEAN NOT NULL,
CardNo INTEGER,
CardType  VARCHAR(255),
ExpiryDetails  VARCHAR(255),
PaypalFlag BOOLEAN NOT NULL,
PaypalLoginID  VARCHAR(255),
PaypalPassword VARCHAR(255),
```

```
VenmoFlag BOOLEAN NOT NULL,
VenmoID VARCHAR(255),
PRIMARY KEY(CustomerID, CustomerPaymentID)
);

CREATE TABLE  CUSTOMER_TICKET
( CustomerID INTEGER,
TicketID INTEGER,
OrderID INTEGER,
DateOfCreation DATE NOT NULL,
Status VARCHAR(255) NOT NULL,
Description  VARCHAR(255) NOT NULL,
PRIMARY KEY(CustomerID,TicketID));

CREATE TABLE ORDER_MAIN(
OrderID INTEGER,
CustomerID INTEGER NOT NULL,
AddressID INTEGER NOT NULL,
ResID INTEGER NOT NULL,
CustomerPaymentID INTEGER NOT NULL,
DasherID INTEGER NOT NULL,
Status VARCHAR(255) NOT NULL,
ExpectedDeliveryTime VARCHAR(255) NOT NULL,
ActualDeliveryTime VARCHAR(255),
Feedback VARCHAR(255),
Instructions VARCHAR(255),
PRIMARY KEY(OrderID));

CREATE TABLE ORDER_DETAILS(
OrderID INTEGER,
ItemID INTEGER,
CustomizationID INTEGER,
ItemQty INTEGER NOT NULL,
ItemAmount DECIMAL(5,2),
PRIMARY KEY(OrderID,ItemID));
```

ALTER TABLE (TRIGGERED ACTIONS) QUERIES:

```
ALTER TABLE DOORDASH_MAIN_ADMIN_DETAILS ADD
CONSTRAINT adminidfk FOREIGN KEY(AdminID)
REFERENCES LOGIN_DETAILS(UserID) ON DELETE
CASCADE;

ALTER TABLE DOORDASHER ADD CONSTRAINT dasheridfk
FOREIGN KEY(DasherID)  REFERENCES
LOGIN_DETAILS(UserID) ON DELETE CASCADE;


ALTER TABLE RESTAURANT_DETAILS ADD CONSTRAINT
residfk FOREIGN KEY(ResID)  REFERENCES
LOGIN_DETAILS(UserID) ON DELETE CASCADE;


ALTER TABLE MENU_ITEMS ADD CONSTRAINT residfk2
FOREIGN KEY(ResID)  REFERENCES
RESTAURANT_DETAILS(ResID) ON DELETE CASCADE;


ALTER TABLE CUSTOMIZATIONS ADD CONSTRAINT residfk3
FOREIGN KEY(ResID)  REFERENCES
RESTAURANT_DETAILS(ResID) ON DELETE CASCADE;

ALTER TABLE CUSTOMIZATIONS ADD CONSTRAINT itemidfk
FOREIGN KEY(ItemID) REFERENCES  MENU_ITEMS(ItemID) ON
DELETE CASCADE;

ALTER TABLE CUSTOMER_DETAILS_MAIN ADD CONSTRAINT
customeridfk FOREIGN KEY(CustomerID) REFERENCES
LOGIN_DETAILS(UserID) ON DELETE CASCADE;

ALTER TABLE CUSTOMER_ADDRESS_DETAILS ADD CONSTRAINT
customeridfk2 FOREIGN KEY(CustomerID) REFERENCES
CUSTOMER_DETAILS_MAIN(CustomerID) ON DELETE CASCADE;


ALTER TABLE CUSTOMER_PAYMENT_INFO ADD CONSTRAINT
customeridfk3 FOREIGN KEY(CustomerID) REFERENCES
CUSTOMER_DETAILS_MAIN(CustomerID) ON DELETE CASCADE;

ALTER TABLE CUSTOMER_TICKET ADD CONSTRAINT customeridfk4
FOREIGN KEY(CustomerID) REFERENCES
CUSTOMER_DETAILS_MAIN(CustomerID)  ON DELETE CASCADE;


ALTER TABLE CUSTOMER_TICKET ADD CONSTRAINT orderidfk
FOREIGN KEY(OrderID) REFERENCES  ORDER_MAIN(OrderID) ON
DELETE CASCADE;
```

```
ALTER TABLE ORDER_MAIN ADD CONSTRAINT customeridfk5
FOREIGN KEY(CustomerID) REFERENCES
CUSTOMER_DETAILS_MAIN(CustomerID) ON DELETE CASCADE;

ALTER TABLE ORDER_MAIN ADD CONSTRAINT addressidfk FOREIGN
KEY(AddressID) REFERENCES  CUSTOMER_ADDRESS_DETAILS (AddressID)
ON DELETE CASCADE;

ALTER TABLE ORDER_MAIN ADD CONSTRAINT residfk4 FOREIGN
KEY(ResID) REFERENCES  RESTAURANT_DETAILS(ResID) ON
DELETE CASCADE;

ALTER TABLE ORDER_MAIN ADD CONSTRAINT paymentidfk
FOREIGN KEY(CustomerPaymentID) REFERENCES
CUSTOMER_PAYMENT_INFO(CustomerPaymentID) ON DELETE
CASCADE;

ALTER TABLE ORDER_MAIN ADD CONSTRAINT dasheridfk2
FOREIGN KEY(DasherID) REFERENCES
DOORDASHER(DasherID) ON DELETE CASCADE;

ALTER TABLE ORDER_DETAILS ADD CONSTRAINT orderidfk2
FOREIGN KEY(OrderID) REFERENCES  ORDER_MAIN(OrderID) ON
DELETE CASCADE;

ALTER TABLE ORDER_DETAILS ADD CONSTRAINT itemidfk2
FOREIGN KEY(ItemID) REFERENCES  MENU_ITEMS(ItemID) ON
DELETE CASCADE;

ALTER TABLE ORDER_DETAILS ADD CONSTRAINT
customizationid FOREIGN KEY(CustomizationID) REFERENCES
CUSTOMIZATIONS(CustomizationID) ON DELETE CASCADE;
```

STORED PROCEDURES:

Creating a procedure which will display restaurants based on a given range of ratings(min to max rating):

```
Create or replace procedure sort_rest_ratings
 (
minRating IN DECIMAL(5,2)
maxRating IN DECIMAL(5,2)
 )
AS
myRestName RESTAURANT_DETAILS.RestName%TYPE;
myResRating RESTAURANT_DETAILS.ResRating%TYPE;
CURSOR my_rest_rating IS
SELECT RestName,ResRating FROM Restaurant_Details WHERE
ResRating >= minRating AND ResRating <=maxRating
ORDER BY ResRating DESC;

BEGIN
OPEN my_rest_rating;
dbms_output.put_line('Restaurants having ratings sorted in the given range:\n');
LOOP
FETCH my_rest_rating INTO myRestName,myResRating;
EXIT WHEN(my_rest_rating%NOTFOUND);
dbms_output.put_line(myRestName || ',' || myResRating);
END LOOP;
CLOSE my_rest_rating;
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No Restaurants in the given range');
END sort_rest_ratings;
```

Creating a procedure that calculates the total no. of orders which were delivered before the expected time by dashers.

```
Create or replace procedure fast_delivered_orders AS
myDasher ORDER_MAIN.DasherID%TYPE;
myCount NUMBER;

CURSOR my_fast_orders IS
SELECT DasherID,COUNT(OrderID) FROM ORDER_MAIN WHERE
ActualDeliveryTime < ExpectedDeliveryTime GROUP BY DasherID ORDER BY
COUNT(OrderID) DESC ;

BEGIN
OPEN my_fast_orders;
dbms_output.put_line('Orders Delivered before expected delivery time:\n');
LOOP
FETCH my_fast_orders INTO myDasher,myCount;
EXIT WHEN(my_fast_orders%NOTFOUND);
dbms_output.put_line(myDasher || ',' || myCount);
END LOOP;
CLOSE my_fast_orders;
EXCEPTION
WHEN NO_DATA_FOUND THEN
dbms_output.put_line('No Orders delivered before expected Delivery time');
END fast_delivered_orders;
```

TRIGGERS:

TRIGGER 1:

A trigger to check if a restaurant is available before a customer places an order from that restaurant

```
CREATE or REPLACE TRIGGER is_rest_available_before_order

BEFORE INSERT ON ORDER_MAIN

FOR EACH ROW

DECLARE

Available RESTAURANT_DETAILS.AvailableFlag%TYPE;

BEGIN

SELECT AvailableFlag INTO Available FROM RESTAURANT_DETAILS

 WHERE ResID = :NEW.ResID;

IF (Available == ‘N’) THEN

Raise_Application_Error(-20000, 'You can’t place the order when the restaurant is not available!!');

END IF;

END;
```

TRIGGER 2:

A trigger to check if the customer ‘s location(city) is same as restaurant’s location(city)

```
CREATE or REPLACE TRIGGER is_order_city_sameas_rest_city
BEFORE INSERT ON ORDER_MAIN
FOR EACH ROW
DECLARE
X NUMBER;
BEGIN
SELECT COUNT(*) INTO X from CUSTOMER_DETAILS C INNER JOIN
RESTAURANT_DETAILS R ON C.City= R.ResCity WHERE C.CustomerID= :NEW.CustomerID
AND C.AddressID= :NEW.AddressID AND R.ResID= :NEW.Res.ID

IF (X == 0) THEN
Raise_Application_Error(-20000, 'You can’t place an order as Order city is different from
Restaurant City');
END IF;
END;
```