CS2102 Web-based database application development

Group 25:

Student Name	Student Number
Wong Shun Min	A0184053J
Wen Shufa	A0185482W
Andy Aw Bo Yang	A0192255A
Liao Meng	A0164769L

Table of Contents:

1. Project responsibilities

Wong Shun Min: Rider login page, creating new user riders, rider summary page, customer profile page, viewing past deliveries, ability to rate and review a foodlist, change password page, report

Wen Shufa: Restaurant Staff login page, creating new user staff, restaurant summary page, add new food and new promotion, view promotion summary, viewing monthly summary, change password page, report and readme file

Andy Aw Bo Yang: Customer login page, creating new customers, ordering food, viewing reviews of the food and checkout page, report

Liao Meng: FDS Manager login page, view information of all orders/customers/riders/delivery locations, check and view orders/customers/riders/delivery location statistics given a time period (start/end date or month-year) and a customer/rider id or a location, report

2. Description of application's data requirements and functionalities

Users are required to fill in all fields, otherwise an alert message will pop up. This requirement is checked in javascript.

For the customer, the credit card number is 8 digits long. The customer is also not allowed to reuse his old password when he/she chooses to edit the password.

For each food, the number of food to be selected is chosen by a slider. The range of the slider is limited to the daily limit of the food. After a customer orders the daily limit number of a particular food (which will cause the number of food remaining to become 0), the food will then be deemed not available and thus it will be removed from the food list in the orders page, preventing the customer from ordering food that is not available.

The orders page displays all available foods that are sold in all the restaurants in a table. To search for food, there is a text box on top of the table that will search for food, restaurant or food category based on the text input. This is achieved using javascript.

The customer can only select food items provided by the same restaurant in one order. The customer can only checkout their order when the total cost of order (before restaurant promotion discount is applied) exceeds the minimum amount set by this restaurant. Before checkout, the customer can choose the number of reward points to use to offset the total cost of order plus delivery. The customer selects the number of reward points to use with a slider. If the customer has no reward points, then there will not be any slider.

There is a drop down list below the delivery location input box which shows the last five delivery locations. When the user clicks on any of the delivery locations in the dropdown box, the delivery location box will be automatically filled up with the choice of the user.

There is a logout button on the top right corner of any profile page.

Registration of new staff requires providing a valid restaurant name, username and password. It requires the same information to log in to the staff account.

Adding new food requires providing food name, daily limit, category, price. The day limit is an integer between 1 to 1000. Price attribute is a decimal number from 1 to 1000. The food name cannot be an existing food in the current restaurant. Adding a new promotion for a restaurant requires entering all information about description, discount, Start date and End date. Discount is a decimal from 1 to 99. The format of start and end date will be checked such that the end date would be after start date by the following javascript. The new promotion will replace the old promotion in this restaurant. Customer’s orders based on the old promotion(s) will not be affected.

When users want to change password, they must reset to a new password and need to re-enter their correct new password to make the reset successful.

The total cost to be paid is calculated by:

$$Total\ cost\ = (\ food\ cost\ * \frac{100\ -\ restaurant\ discount}{100}) + (delivery\ fee\ * \frac{100\ -\ delivery\ promo}{100}\ ) - reward\ points$$

Where delivery fee is the number of foods bought multiplied by 5.

Criteria for determining a rider's monthly salary: (number of deliveries completed by the rider * 5) + monthly base salary of the rider. For part-time riders, monthly base salary = weekly base salary * 4. Salary earned from each delivery is constant ($5), regardless of the amount of the delivery fee.

For every $500 purchase, a customer earns a reward point.

When the manager wants to check order/customer/rider statistics, he/she has to either specify a time period (start date and end data) or a month-year, and the id of a rider and customer should be an integer. The application will reload the page and alert the user if start date is after start date, or id input is not an integer, or the id does not exist in the database. When the manager wants to check location statistics, he/she has to select a location that exists in the database and specify a date.

2.1 Overview of system functionalities

There exists 4 types of users in this FDS: Customers, restaurant staff, delivery riders and FDS managers.

2.1.1 Customers

a. Able to login to view his own data (e.g. past orders, reward points)
b. Able to rate his/her past orders
c. Able to change his/her password
d. Able to view all food items that are available for purchase
e. Able to review any food from any restaurant
f. Able to order food (and use reward points as well)
g. Able to search for food based on food name, restaurant name and category name

2.1.2 Restaurant Staff

a. Able to login to view his/her restaurant's data
b. Able to change his/her password
c. Able to add new food items into the menu
d. Able to add new promotions
e. Able to view a monthly summary of the restaurant for different month
f. Able to view summary of a promotion (the average number of orders received, duration)

2.1.3 Delivery Riders

a. Able to login to view his/her data
b. Able to change his/her password
c. Able to view a summary of their work (number of deliveries, total working hours etc, salary, job status)
d. Able to view past deliveries

2.1.4 FDS Managers

a. Able to login
b. Able to view all orders made
c. Able to view order statistics within a time period/month
d. Able to view all riders and their information
e. Able to view delivery statistics of one rider within a time period/month
f. Able to view all customers and their information
g. Able to view order statistics of one customer within a time period/month
h. Able to view all locations where some orders are placed on
i. Able to view order statistics of one location in a date

2.2 Data constraints

Promotions:
Each promotion is unique. Hence the unique promo id for each promotion
Primary key(promoId);

ShiftDay:
This represents the days for the different MWS shifts. Since the shifts are unique from each other, the primary key is hence the shift day id.
Primary key(ShiftDayId);

ShiftHour:
This represents the hours for the MWS. Since each hour shifts are unique from each other, the primary key is hence the shift day id.
Primary key(ShiftHourId);

Sessions:
Each session represents the shifts for WWS. Each shift can only be 1 or 2 hours as there must be an hour of break between two consecutive hours. The start and end intervals must be between 10 to 22 (24h time) as the FDS operates from 10am to 10pm.
Primary key(Sessionsid);
TotalHours Integer check(TotalHours>=1 and TotalHours<=2),
EndInterval Integer check(EndInterval >= 10 and EndInterval <= 22),
StartInterval Integer check(EndInterval >= 10 and EndInterval <= 22)

MWS:
This represents the monthly work schedule.
Primary key(MWSId);

Customer:
This represents the customer. Each customer must be unique.
Primary key(Cid);

Riders:
This represents each rider. Each rider must be unique.Also, this table/class is the parent table to part time riders and full time riders (part time riders and full time riders inherit from this table)
Primary key(riderid);

DeliverPromotions:
This represents the promotions for the delivery. Each promotion is unique.
Primary key(pid);

Delivers:
This represents an order delivered by a rider. Each delivery is unique (thus primary key), and is being delivered by a rider. It also has a promotion, which is referenced to deliverpromotions. As a result, the riderid and the pid (deliver promotion's promo id) are foreign keys
Primary key(did);
FOREIGN KEY(riderid) REFERENCES Riders(riderid);
FOREIGN KEY(pid) REFERENCES deliverpromotions;
riderid integer not null;

Restaurants:
This represents the restaurants in the system. Each restaurant name is unique. Likewise, given that each restaurant requires that each order has a minimum cost, the minimal cost value is hence required, thus not null.
primary key(rname);
constraint restaurant_fkey FOREIGN KEY (promoId) REFERENCES Promotions (promoId) deferrable initially deferred;

minimalCost Integer NOT NULL;
rname varchar(100) NOT NULL;

Food_categories:
Each category is unique, thus the category is primary key. Category_meaning is what the category stands for, thus it cannot be null. As no 2 category can share the same category_meaning value, category_meaning is thus unique.
PRIMARY KEY (category);
category_meaning text UNIQUE NOT NULL;

Foods:
This represents the food and the restaurant that sells them. The primary key is a food name-restaurant pair because there might exist two restaurants that sell the same food. Thus, instead of only food being unique, the uniqueness of each row is determined by the foodname-restaurant pair. Since each food belongs to a restaurant, and one restaurant has many food, the restaurant name is hence the foreign key here as well as its value not being allowed to be null.
primary key(fname, rname);
FOREIGN KEY (rname) REFERENCES Restaurants (rname) on delete cascade;
fname varchar(100) NOT NULL;
rname varchar(100) NOT NULL;

Foodlists:
This represents the foodlist given an order. Since each foodlist is unique from one another, it is hence the primary key. Likewise, each foodlist is delivered by 1 delivery and no 2 foodlist can be delivered by the same delivery, thus Did (to represent delivery id) is unique. The Did foreign key is needed to get information about the delivery. Since each foodlist must be ordered by a customer, cid (customer id) thus cannot be null. Cid is also foreign key to reference customer table. Since each foodlist only exists because its corresponding customer record exists, and no foodlist should exist if its corresponding customer record do not exist, it hence is a weak entity with the owner being the customer table. Thus it should have a “on delete cascade” for its foreign key to customer.
flId Integer primary key;
Cid int not null references Customer(cid) on delete cascade;
unique(Did);
FOREIGN KEY (Did) REFERENCES Delivers(Did);

Consists:
This represents the content given a foodlist. Since 1 foodlist needs many foods, this table exists to ensure 1 foodlist can have many foods. This table works by allowing 1 foodlist to have many “consists” records, which each “consists” record is joined to one specific food. Thus, it allows 1 foodlist to be joined to many foods. Consists id (coid) is just to make sure each record can be differentiated. Foreign key is fname and rname (food name and restaurant name) pair from foods to join to foods and flid (foodlist id) to joins to foodlist so as to associate a foodlist to many foods. Since each “consists” only exists because its corresponding foodlist record exists, and no foodlist should exist if its corresponding customer record do not exist, it hence is a weak entity with the owner being the foodlist table. Thus it should have a “on delete cascade” for its foreign key to foodlist..
Primary key (coid);
FOREIGN KEY (fname, rname) REFERENCES Foods (fname, rname);
FOREIGN KEY (flId) REFERENCES Foodlists (flId) on delete cascade;

Reviews:
Each review is unique to a food list, thus flid (food list id) is a primary key and foreign key to the food list.
Flid int references Foodlists(flid);
primary key(Flid);

PartTimeRiders:
Part time riders is a child table/class to riders, thus the references to riders table and the “on delete cascade” option on riders too.
Primary key(riderid) primary key REFERENCES Riders on delete cascade;

WWS:
WWS refers to the weekly work schedule of a part timer. Since each WWS information is different from one another, the WWSid is hence the primary key. For DayOfWeek, it uses number to represent Monday to Sunday (basically the day in the week that the rider works), thus the >= 1 and <= 7 constraint. Rider id is the foreign key to the actual part time rider as one rider may have multiple WWS records.

WWSid Integer check (WWSid > 0);
DayOfWeek Integer check ((DayOfWeek >= 1) and (DayOfWeek <= 7));
primary key (WWSid);
FOREIGN KEY(riderid) REFERENCES parttimeriders (riderid);

FullTimeRiders:
Full time riders is a child table/class to riders, thus the references to riders table and the “on delete cascade” option on riders too. Since each full time rider needs a MWS schedule, where the information is obtained from the MWS tabe, MWSID is hence not null and a foreign key to MWS table
riderid Integer primary key references Riders on delete cascade;
mwsid int not null;
foreign key (mwsid) references MWS(mwsid);

Holds:
This serves to join a WWS record to the actual information. Since one WWS record represent a particular day a part time worker works, and in a particular day there are many shifts, we use this “Holds” table to link WWS to the shift hour information (shift hours stored in the table Sessions). Thus, the primary key is the WWSid-Sessionsid pair for unique identification while the foreign key is WWSid to WWS table and Sessionsid to the Sessions table.
foreign key (WWSid) references WWS(wwsid);
foreign key (Sessionsid) references Sessions(Sessionsid);
Primary Key (WWSid, Sessionsid);

Comprises:
This serves to link MWS to the MWS shift information. Foreign key is ShiftDayId to ShiftDay to get the days for the MWS shift and ShiftHourId to ShiftHour to get the hours for the MWS shift.
Primary Key (MWSid),
foreign key (ShiftDayId) references ShiftDay (ShiftDayId),
foreign key (ShiftHourId) references ShiftHour (ShiftHourId),
foreign key (MWSid) references MWS (MWSid)

customerLogin:
This is the customer login information. The primary key and the foreign key is CId since each customer has unique information and that the customer information is stored in the customer table. Since each customer needs a username and password, the username and password field cannot be null. Likewise, since no two customers can have the same username, the username is hence unique. Finally, since customer login information can only exist when a customer exist, and that no customer login information should exist if its corresponding customer don’t exist, this is a weak entitiy too. Thus the “on delete cascade” for the foreign key constraint to customer.
Username text NOT NULL,
Password text NOT NULL,
PRIMARY KEY (Cid),
Unique(Username),
foreign Key (cid) references Customer(cid) on delete cascade

riderLogin :
The primary key and foreign key is riderid as each rider should have unique information and their information is stored in the Riders table. By having the unique username, riders can create their own account and login to search their own information.
PRIMARY KEY (riderid),
FOREIGN KEY (riderid) REFERENCES Riders(riderid)
UNIQUE(Username),
Username text NOT NULL,
Password text NOT NULL

staffLogin:
Staff should work in an existing restaurant and this is guaranteed by setting the restaurant name as the forign key and a not null value. Restaurant and username should be unique as under the same restaurant, staff need to use the different username to login.
PRIMARY KEY (staffid)
FOREIGN KEY (Restaurant_name) REFERENCES Restaurants (rname)

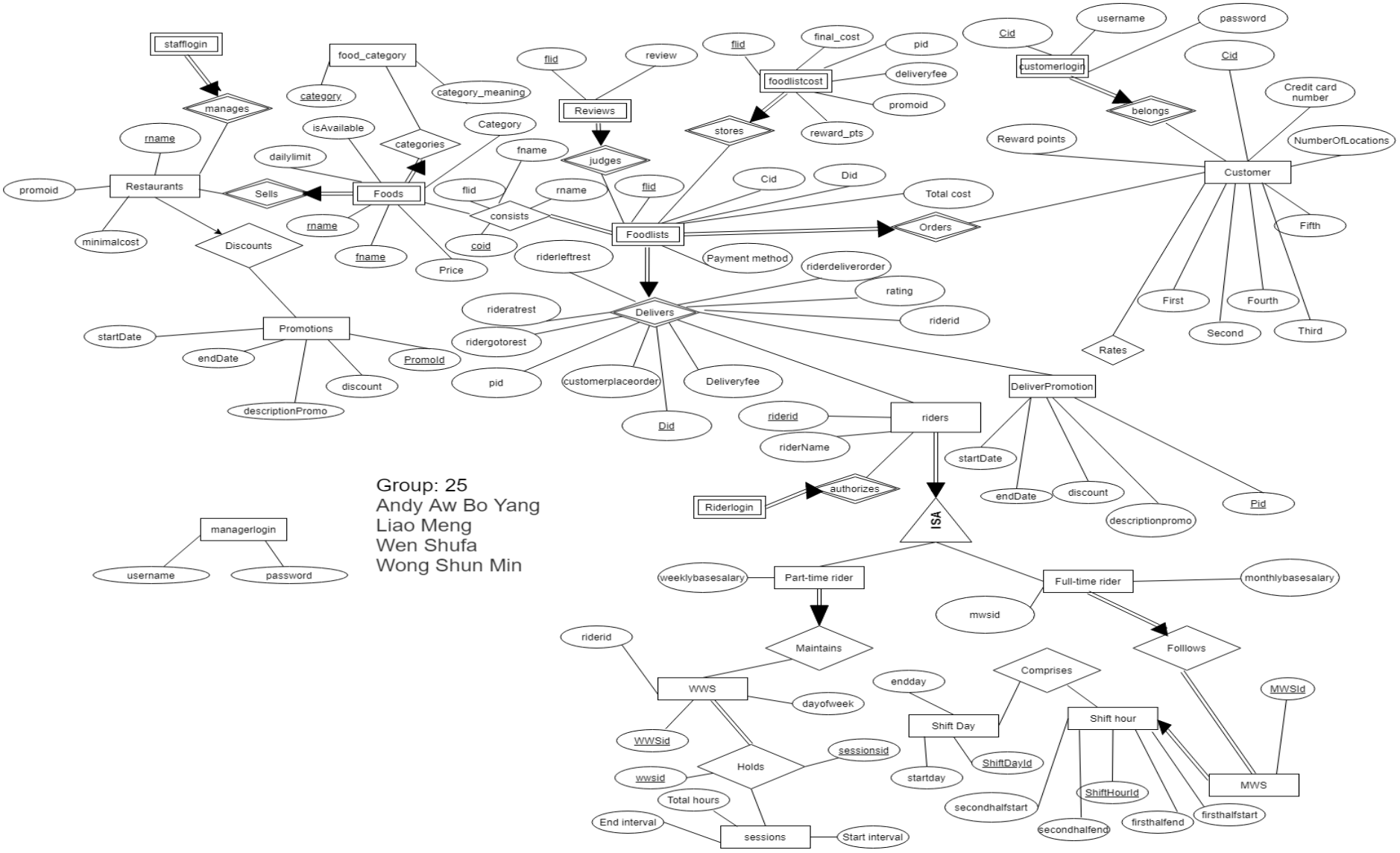
unique(Restaurant_name, Username);
Username text NOT NULL,
Password text NOT NULL,
Restaurant_name text NOT NULL

managerLogin:
PRIMARY KEY (mid),
UNIQUE (Username)
Username text NOT NULL,
Password text NOT NULL

foodlistCost:
Foodlistcost stored auxiliary information regarding the food lists, such as the reward points used, the promo id used etc. Since each foodlist is unique, and one foodlist joins to exactly one foodlistcost, the primary key and foreign key is flid (foodlist id) on foodlist. Likewise, since foodlistcost information can only exist when a foodlist record exist, and that no foodlistcost record should exist if its corresponding foodlist don't exist, this makes it a weak entitiy. Thus there is a "on delete cascade" for the foreign key to foodlist..
Primary key (flid)
Foreign key (flid) references foodlists (flid) on delete cascade

3. Entity Relationship Model

3.1 ER Model



3.2 Constraints not captured by ER Model

-There must be at least five riders (part-time or full-time) working at each hourly interval.
-There must be at least one hour of break between two consecutive hour intervals.
-Each employee is either a part time or full time employee
-The total number of hours in each WWS must be at least 10 and at most 48.
-The food bought in an order cannot be bought from different restaurants
-The customer cannot order a non-available food
-The customer cannot order a food if the total price is smaller than the minimum cost stipulated by the restaurant
-Each work day must consist of eight work hours comprising two four-hour periods with an hour break in between.
-The delivery timing/hours has to be between 10am to 10pm
-The number representing the days of the week must be between 1 to 7
-The four WWS in MWS is equivalent, where each WWS in MWS must consist of five consecutive work day
-There is only 4 work hours shift for each work day
-The total cost of each order is the sum of the cost of the ordered food items and a delivery fee
-Customers could opt to make their payment using a pre-registered credit card or paying cash on delivery.
-Each hour interval must start and end on the hour, and its duration must not exceed four hours.

4. Relational Schema

4.1 Constraints that are not enforced by relational schema

-There must be at least five riders (part-time or full-time) working at each hourly interval.
-There must be at least one hour of break between two consecutive hour intervals.
-Each employee is either a part time or full time employee
-The total number of hours in each WWS must be at least 10 and at most 48.
-The food bought in an order cannot be bought from different restaurants
-The customer cannot order a non-available food
-The customer cannot order a food if the total price is smaller than the minimum cost stipulated by the restaurant
-Each work day must consist of eight work hours comprising two four-hour periods with an hour break in between.

4.2 Normalization form

```
create table Promotions(
    promoId Integer primary key,
    descriptionPromo varchar(200),
    discount decimal,
    startDate date,
    endDate Date
);
```

Since promoId is a superkey of the table and there are no other dependencies, Promotions table is in 3NF.

```
CREATE TABLE ShiftDay (
    ShiftDayId Integer primary key,
    StartDay VARCHAR(100),
    EndDay VARCHAR(100)
);
```

Since ShiftDayId is a superkey of the table and there are no other dependencies, ShiftDay table is in 3NF.

```
CREATE TABLE ShiftHour (
    ShiftHourId Integer primary key,
    FirstHalfStart Integer,
    FirstHalfEnd Integer,
    SecondHalfStart Integer,
    SecondHalfEnd Integer
```

```
);
```
Since ShiftHourId is a superkey of the table and there are no other dependencies, ShiftHour table is in 3NF.

```
CREATE TABLE Sessions (
    Sessionsid Integer primary key,
    TotalHours Integer check(TotalHours>=1 and TotalHours<=2),
    EndInterval Integer check(EndInterval >= 10 and EndInterval <= 22),
    StartInterval Integer check(EndInterval >= 10 and EndInterval <= 22)
);
```
Since Sessions is a superkey of the table, Sessions table is in 3NF.

```
CREATE TABLE MWS (
    MWSId Integer primary key
);
```
Since MWSid is a superkey of the table, MWS table is in 3NF.

```
CREATE TABLE Customer (
    Cid int PRIMARY KEY,
    Reward_pts int default 0,
    CC_no text,
    FirstLoc text,
    SecondLoc text,
    ThirdLoc text,
    FourthLoc text,
    FifthLoc text,
    No_of_loc int default 0
);
```
Since Cid is a superkey of the table, Customer table is in 3NF.

```
create table Riders (
    riderid Integer primary key,
    ridername VARCHAR(60)
);
```
Since there are only 2 columns, Riders table is in BCNF, which implies that it is also in 3NF.

```
create table DeliverPromotions (
    pid INTEGER primary key,
    descriptionPromo varchar(200),
    discount decimal,
    startDate DATE,
    endDate Date
);
```
Since pid is a superkey of the table and there are no other dependencies, DeliverPromotions is in 3NF.

```
create table Delivers (
    did INTEGER primary key,
    deliveryfee decimal,
    customerplaceorder timestamp,
    ridergotorest timestamp,
    rideratrest timestamp,
    riderleftrest timestamp,
    riderdeliverorder timestamp,
    riderid integer not null,
    FOREIGN KEY(riderid) REFERENCES Riders(riderid),
    rating INTEGER,
    pid integer,
    FOREIGN KEY(pid) REFERENCES deliverpromotions
);
```
Since did is a superkey of the table and there are no other dependencies, Delivers is in 3NF.

```
create table Restaurants(
    rname varchar(100) NOT NULL,
    promoId Integer,
    minimalCost Integer NOT NULL,
    primary key(rname),
    constraint restaurant_fkey FOREIGN KEY (promoId) REFERENCES Promotions (promoId) deferrable initially deferred
);
```
Since rname is a superkey of the table and there are no other dependencies, Restaurants is in 3NF.

```
CREATE TABLE Food_categories (
    category VARCHAR(60),
    category_meaning text UNIQUE NOT NULL,
    PRIMARY KEY (category)

);
```
Since there are only 2 attributes, Food_categories table is in BCNF, which implies that it is also in 3NF.

```
create table Foods(
    fname varchar(100) NOT NULL,
    rname varchar(100) NOT NULL,
    dailyLimit Integer,
    isavailable boolean,
    category VARCHAR(60) references Food_categories (category),
    price decimal,
    primary key(fname, rname),
    FOREIGN KEY (rname) REFERENCES Restaurants (rname) on delete cascade
);
```
Since (fname, rname) is a superkey and there are no other dependencies, it is in 3NF.

```
create table Foodlists(
    flId Integer primary key,
    Cid int not null references Customer(cid) on delete cascade,
    Riderid int,
    Promoid int,
    Order_time date,
    Restaurant_name text,
    Payment_method text,
    Total_cost numeric default 0,
    Delivery_location text,
    Did integer,
    unique(Did),
    FOREIGN KEY (Did) REFERENCES Delivers(Did)
);
```
If this table is normalized to 3NF/BCNF, it would create an extra table with that two column.Currently, if the table is being updated, only this table is needed to be updated. However, if we normalize this table, we would have to update more tables.

```
create table Consists(
    Coid integer,
    flId Integer,
    fname varchar(100),
    rname varchar(100),

    Primary key (coid),
    FOREIGN KEY (fname, rname) REFERENCES Foods (fname, rname),
    FOREIGN KEY (flId) REFERENCES Foodlists (flId) on delete cascade
);
```
Since coid is a superkey and there are no other dependencies, it is in 3NF.

```
CREATE TABLE Reviews (
```

```
    Review text,
    Flid int references Foodlists(flid),
    primary key(Flid)
);
```
Since there are only 2 columns, Reviews table is in BCNF, which implies that it is also in 3NF.

```
create table PartTimeRiders (
    riderid Integer primary key REFERENCES Riders on delete cascade,
    weeklybasesalary Integer
);
```
Since there is only 2 columns, this is already in BCNF, thus also means it is in 3NF

```
CREATE TABLE WWS (
    WWSid Integer check (WWSid > 0),
    DayOfWeek Integer check ((DayOfWeek >= 1) and (DayOfWeek <= 7)),
    riderid Integer,
    primary key (WWSid),
            FOREIGN KEY(riderid) REFERENCES parttimeriders (riderid)
);
```
Since WWSid is a superkey and there are no other dependencies, it is in 3NF.

```
create table FullTimeRiders (
    riderid Integer primary key references Riders on delete cascade,
    mwsid int not null,
    foreign key (mwsid) references MWS(mwsid),
    monthlybasesalary Integer
);
```
Since riderid is a superkey and there are no other dependencies, it is in 3NF.

```
CREATE TABLE Holds (
    WWSid Integer,
    foreign key (WWSid) references WWS(wwsid),
    Sessionsid Integer,
    foreign key (Sessionsid) references Sessions(Sessionsid),
    Primary Key (WWSid, Sessionsid)
);
```
Since (WWSid, Sessionsid) is a superkey and there are no other dependencies, it is in 3NF.

```
CREATE TABLE Comprises (
    MWSid Integer,
    Primary Key (MWSid),
    ShiftDayId Integer,
    ShiftHourId Integer,
    foreign key (ShiftDayId) references ShiftDay (ShiftDayId),
    foreign key (ShiftHourId) references ShiftHour (ShiftHourId),
    foreign key (MWSid) references MWS (MWSid)
);
```
Since MWSid is a superkey and there are no other dependencies, it is in 3NF.

```
CREATE TABLE customerLogin (
    Username text NOT NULL,
    Password text NOT NULL,
    Cid int,
    PRIMARY KEY (Cid),
    Unique(Username),
    foreign Key (cid) references Customer(cid) on delete cascade
);
```
Cid is a superkey. Even though there is a username -> CId dependency, since Cid is a prime attribute and since there are no other dependencies, it is in 3NF.

```
CREATE TABLE riderLogin (
   Username text NOT NULL,
   Password text NOT NULL,
   riderid Integer,
   PRIMARY KEY (riderid),
   Unique(Username),
   foreign Key (riderid) references Riders(riderid)
);
```
Riderid is a superkey. Even though there is a username -> riderid dependency, since riderid is a prime attribute and since there are no other dependencies, it is in 3NF.

```
CREATE TABLE staffLogin (
   Username text NOT NULL,
   Password text NOT NULL,
   Restaurant_name text not null,
   staffid Integer,
   PRIMARY KEY (staffid),
   unique(Restaurant_name, Username),
   foreign Key (Restaurant_name) references Restaurants(rname)
);
```
Staffid is a superkey. Even though there is a (username,rname) -> staffid dependency, since staffid is a prime attribute and since there are no other dependencies, it is in 3NF.

```
CREATE TABLE managerLogin (
   Username text NOT NULL,
   Password text NOT NULL,
   mid Integer,
   PRIMARY KEY (mid),
   UNIQUE(Username)
);
```
mid is a superkey. Even though there is a username -> mid dependency, since mid is a prime attribute and since there are no other dependencies, it is in 3NF.

```
CREATE TABLE foodlistCost (
   flid integer,
   Reward_pts integer default 0,
   promoId integer,
   pid integer,
   deliveryfee integer,
   final_cost numeric default 0,
   Primary Key (flid),
   foreign key (flid) references foodlists on delete cascade

);
```
Since flid is the key of foodlistCost, and there are no other dependencies, foodlistcost is hence in 3NF.

5. Details of three non-trivial/interesting triggers used in the application

This constraint checks if the food selected/ordered comes from the same restaurant. If there exists a new food that does not belong to the same restaurant, it will fire the trigger.

```
CREATE OR replace FUNCTION check_food_restuarant_constraint() RETURNS TRIGGER
  AS $$
```

```
  declare
      rnamething varchar(100);

  Begin

  SELECT c.rname into rnamething
  FROM consists c
  WHERE c.flid = new.flid
  AND c.rname <> new.rname;

  if rnamething is not null then
      raise exception 'You added a food from this restuarant %. This restuarant is not the same as
the other food restuarant in your foodlist!', new.rname;
  end if;
  return null;


  END

  $$ language plpgsql;

  Drop trigger if exists check_food_restuarant_trigger on consists;
  CREATE trigger check_food_restuarant_trigger
  AFTER INSERT OR UPDATE on consists
  For each row
  Execute FUNCTION check_food_restuarant_constraint();
```

This triggers check if the food that is ordered/chosen by the customer is an actual available food. The trigger will be fired if the customer attempts to order a non-available food.

```
  CREATE OR replace FUNCTION check_food_availability_constraint() RETURNS TRIGGER
  AS $$

  declare
      availabilityCheck boolean;

  Begin

  SELECT f.isavailable into availabilityCheck
  FROM consists c, foods f
  WHERE new.fname = f.fname
  AND new.rname = f.rname
  AND c.coid = new.coid
  AND c.fname = f.fname;

  if availabilityCheck = FALSE then
      raise exception 'You added a non-available food %!', new.fname;
  end if;
  return null;


  END

  $$ language plpgsql;


  Drop trigger if exists check_food_availability_trigger on consists;
  CREATE trigger check_food_availability_trigger
  AFTER INSERT OR UPDATE on consists
```

```
For each row
Execute FUNCTION check_food_availability_constraint();
```

This triggers checks to see if the total cost of the food is more than the minimum cost required by the restaurant. This trigger will be fired if the customer attempts to check out a food that is below the stipulated amount by the restaurant.

```
CREATE OR replace FUNCTION check_food_cost_constraint() RETURNS TRIGGER
AS $$

declare
    total_cost numeric;
    minimum_cost numeric;

Begin

SELECT r.minimalCost into minimum_cost
FROM restaurants r
WHERE new.restaurant_name = r.rname;

SELECT fl.Total_cost into total_cost
FROM foodlists fl
WHERE new.flid = fl.flid;

if total_cost < minimum_cost then
    raise exception 'Total cost is too small!';
end if;
return null;


END

$$ language plpgsql;


Drop trigger if exists check_food_cost_trigger on foodlists;
CREATE trigger check_food_cost_trigger
AFTER UPDATE OF restaurant_name ON foodlists
For each row
Execute FUNCTION check_food_cost_constraint();
```

6. Three of the most complex queries(sql)

Given a riderid x, find x's name, job status, number of deliveries made, total working hours and weekly salary. Output is as follows: (Name, rider id, status, number of deliveries, total working hours, salary (weekly)).

```
with totalworkinghours as (
    with fulltimers as (
        with fulltime as (
            select riderid, 40 hours
            from fulltimeriders
        )
        select
            riderid, case when hours is null then 0 else hours end
        from riders left join fulltime using (riderid)
        ),
parttimers as (
        select riderid, case when sum(totalhours) is null then 0 else sum(totalhours) end totalhours
```

```
        from riders left join ((holds natural join sessions) natural join wws) using (riderid)
        group by riderid
        order by riderid
    )
    select riderid, hours + totalhours totalhours
    from fulltimers natural join parttimers
),
totalsalary as (
    with riderbasesalary as (
        SELECT distinct riderid, CASE WHEN wwsid IS NOT NULL then (weeklybasesalary * 4) ELSE
monthlybasesalary END AS salary
        FROM(riders LEFT JOIN fulltimeriders using (riderid) LEFT JOIN parttimeriders USING (riderid) left join wws
using (riderid))
    ), ridernumdeliveries as (
        select riderid, count(*) numofdeliveries
        from delivers
        group by riderid
    )
    select riderid, case when numofdeliveries is null then salary else (salary + (numofdeliveries * 5)) end totalsalary,
salary,
        case when numofdeliveries is null then 0 else numofdeliveries end
    from riderbasesalary left join ridernumdeliveries using(riderid)
    order by riderid
)
select *, case when riderid in (
    select riderid
    from parttimeriders)
    then 'part timer' when riderid in (select riderid from fulltimeriders) then 'full timer' else 'error' end as status, ridername
from(totalworkinghours natural join totalsalary) natural join riders where riderid = x;
```

Given the specific day and time (eg Tuesday at 5 pm) find riders (be it part time or full time) that are available to do the delivery based on their working schedule. Output their rider id.

```
with ftr as(
        select riderid, firsthalfstart, firsthalfend, secondhalfstart, secondhalfend, case when startday = 'Monday'
then 1 when startday = 'Tuesday' then 2 when startday = 'Wednesday' then 3 when startday = 'Thursday' then 4 when
startday = 'Friday' then 5 when startday = 'Saturday' then 6 when startday = 'Sunday' then 7 else 0 end startday, case
when endday = 'Monday' then 8 when endday = 'Tuesday' then 9 when endday = 'Wednesday' then 10 when endday =
'Thursday' then 4 when endday = 'Friday' then 5 when endday = 'Saturday' then 6 when endday = 'Sunday' then 7 else 0
end endday
        from ((fulltimeriders join comprises using (mwsid)) join shifthour using (shifthourid)) join shiftday using
(shiftdayid))

        select riderid from ftr
        where ( 17 between firsthalfstart and firsthalfend or 17 between secondhalfstart and secondhalfend)
        and (2 between startday and endday)

        UNION

        select riderid
        from ((parttimeriders join wws using (riderid)) join holds using (wwsid)) join sessions using (sessionsid)
        where dayofweek = 2
         and 17 between startinterval and endinterval;
```

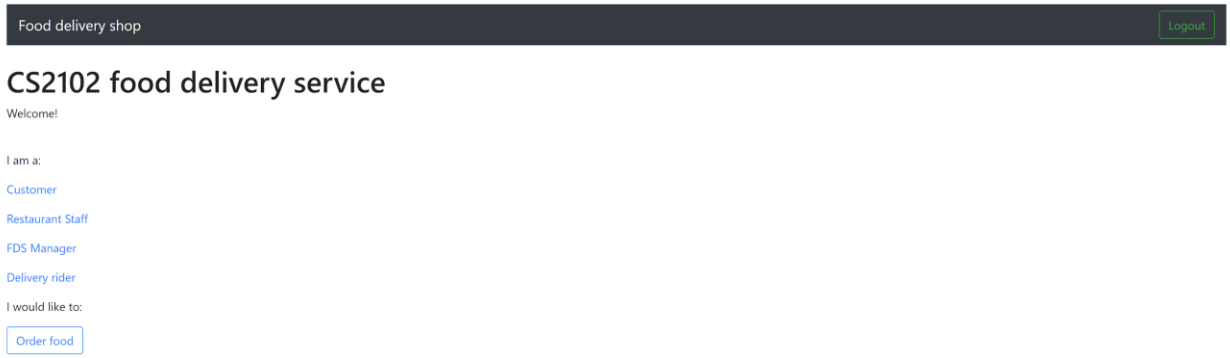
Given a rider id (‘30’), a start date (‘2020-05-01’) and an end date (‘2020-06-01’), calculate the number of orders the given rider has delivered, the number of ratings he/she received (customer may not give a rating for a delivery), the average rating he/she has received (if he/she received no ratings, return ‘no rating’) in 2 decimal places, the average time he/she spends on completing a delivery (defined as the time between when the rider goes to the restaurant and the time when the rider delivers the food to the customer) shown in “hh:mm:ss” format, within the time period (bound by start date inclusive and end date exclusive).

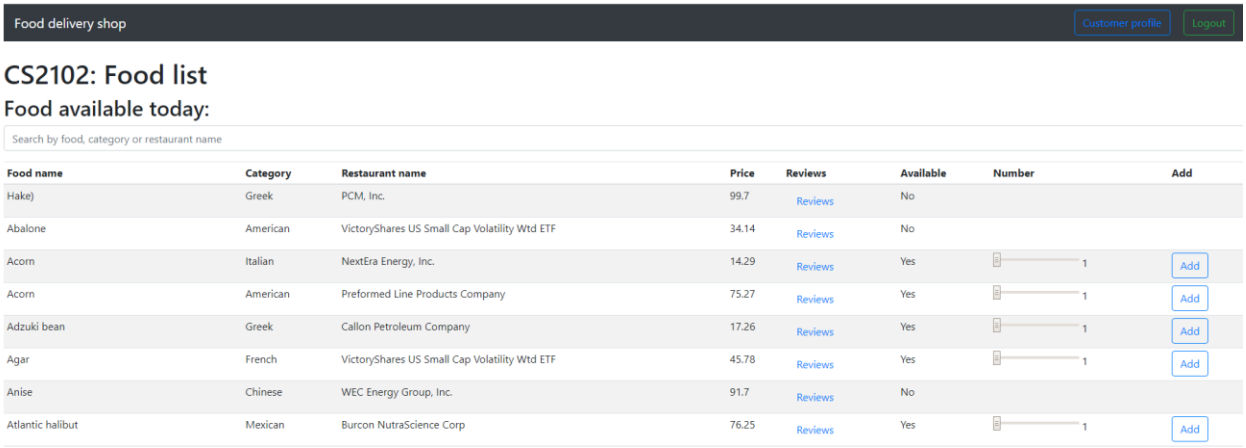
```
with deliverylist as (
    select *
    from (foodlists natural join delivers)
    where order_time >= '2020-05-01' and order_time < '2020-06-01' and riderid = '300'
)
select count(flid) as num, coalesce(count(rating), 0) as ratings,
    coalesce(avg(rating)::decimal(10, 2)::text, 'no rating') as ratingtext,
    count(distinct cid) as customer,
    coalesce(to_char(avg(riderdeliverorder - ridergotorest), 'HH24:MI:SS'), 'NA') as avgtime
    from deliverylist
```

7. Specification of the software tools/frameworks used in your project.

Client tier: HTML and Javascript
Application tier: NodeJS
Database tier: PostgreSQL

8. Screenshots of the application





Food delivery shop

Welcome, our valued customer

Thank you for your support

Click here to order

Customer Id	Reward Points	Credit card number
41	20	45416596

Your past orders

Order number	List of food purchased	Total cost	From	Date	Review your order	Your review	Rate your order	Your rating
10	Click to view food items	351.71	Lombard Medical, Inc.	Tue May 05 2020 00:00:00 GMT+0800 (+08)	Click to review your order	Can be improved	1 2 3 4 5 6 7 8 9 10 Rate!	7
11	Click to view food items	274.59	Lombard Medical, Inc.	Fri May 01 2020 00:00:00 GMT+0800 (+08)	Click to review your order		1 2 3 4 5 6 7 8 9 10 Rate!	5

Food delivery shop

List of food items purchased

Food Item	Quantity	Price(each)	From	Delivery location	Time delivered	Delivery Id	Delivered by
Common octopus	3	6.17	Lombard Medical, Inc.	location6	Tue Dec 31 2019 22:29:28 GMT+0800 (+08)	10	Nadiya
Fruit preserve	2	33.94	Lombard Medical, Inc.	location6	Tue Dec 31 2019 22:29:28 GMT+0800 (+08)	10	Nadiya
Sauce	2	39.02	Lombard Medical, Inc.	location6	Tue Dec 31 2019 22:29:28 GMT+0800 (+08)	10	Nadiya
Scallop	4	46.82	Lombard Medical, Inc.	location6	Tue Dec 31 2019 22:29:28 GMT+0800 (+08)	10	Nadiya

Click here to go back

CS2102 Food delivery service

Rider Statistics from 2020-05-01 to 2020-06-01 (exclusive)

Click here to move back

Rider id	Name	Status	Number of deliveries	Number of customers served	Average delivery time	Number of raitngs received	Average rating	Total weekly working hours	Base monthly salary	Salary earned from delivery
300	Fabien	full timer	7	1	00:12:37	5	6.60 / 10	40	238	35

Order id	Customer id	Order time	Time spent on delivery (in hours:minutes:seconds)	Total time spent (in hours:minutes:seconds)	Delivery location	Delivery fee	Rating on delivery
24	41	2020-05-03 18:05:55	00:09:31	00:17:47	location 9	5	no rating
25	41	2020-05-03 17:05:32	00:03:00	00:08:55	location 9	5	9
26	41	2020-05-03 16:05:26	00:15:06	00:21:01	location 9	45	9
27	41	2020-05-03 15:05:21	00:12:00	00:18:56	location 9	5	6
28	41	2020-05-03 14:05:58	00:13:00	00:23:00	location 9	5	4
29	41	2020-05-03 13:05:03	00:08:05	00:12:27	location 9	5	5
30	41	2020-05-03 12:05:29	00:27:39	00:32:01	location 9	15	no rating

9. Summary of difficulties encountered and lessons learned

For this project, we have faced countless difficulties. Firstly, only one of us had prior knowledge in building a web application, and the rest of us were struggling to understand and use HTML and NodeJS to build our web application. Nevertheless, through this assignment, we learnt how the three tier approach can be implemented using the Model-View-Controller model of nodejs, namely the View (consisting of the html and javascript) being the client, the Controller (consisting of node js code) being the application server and the Model (consisting of the PostgreSQL server) being the database server

Secondly, due to the covid-19 pandemic, communication with each other became harder, and we had to use Zoom as our primary source of communication for the project. Although we faced some difficulties at the start using Zoom, we gradually managed to get used to it.

From this project, we have gained experience in building a web application. However, we not only learnt the technical skills in web application, but soft skills as well. We learnt the importance of clear communication, teamwork and collaboration. We also learnt the importance of planning and starting early as there were some parts of the project we felt could have done better if we better managed and planned our time. Finally, we learnt that it is never a good idea to underestimate what seems like an easy project because it may hide some intricacies that might end up being very troublesome if we are not careful about it.