CS2102 PROJECT TEAM 56

Table Of Content


```
LEOW JIT YONG (A0183105N)
LIM WEI DONG (A0183105N)
HOON CHEE PING (A0183105N)
LEE EE JIAN (A0183105N)
```


1. Introduction

The objective of the team project is to create a food delivery service application for the following stakeholders :

- Customers - Browse and place orders on food items from Restaurants
- Restaurants - Receive and prepare orders from Consumers

- Delivery rider - Deliver orders from Restaurants to Consumers
- Food Delivery Service (FDS) Manager - Manage operations of the application

The following sections will attempt to mainly explain the design considerations for the aforementioned application.

2. Roles and Responsibilities

Leow Jit Yong - Fullstack Developer

Lim Wei Dong - Fullstack Developer

Hoon Chee Ping - Fullstack Developer

Lee Ee Jian - Fullstack Developer

3. Application

3.1. Data Requirements and Constraints

The following are the data requirements and constraints for the various entities / relations in our project:

Users :

- Each user entity stores the userId, the user's name, and date of creation of account
- Each user is uniquely identifiable by userId (primary key)
- Each user must be one of : (i) Customer (ii) Restaurant Staff (iii) Rider (iv) FDS Manager

Customers

- The customer entity is a user, and stores the userId, name, date of creation of account, and credit card information for payment of orders
- Each customer is identified by userId (primary key) which is referenced from users (foreign key)

Riders

- The rider entity is a user, and stores the userId, name, date of creation of account, and the area he lives in
- The area attribute is used to facilitate the application's assignment of orders to riders based on locality
- Each rider is identified by userId (primary key) which is referenced from users (foreign key)
- Each rider must be either : (i) Full-Time Rider (ii) Part-Time Rider

Restaurant Staff

- The restaurant staff entity is a user and stores the userId, name, date of creation of account
- Each restaurant staff is identified by userId (primary key) which is referenced from users (foreign key)
- Every restaurant staff must work in exactly one restaurant

FDS Manager

- The FDS manager entity is a user and stores the userId, name, date of creation of account
- Each FDS Manager is identified by userId (primary key) which is referenced from users (foreign key)

Part-Time Rider

- The part-time rider entity is a rider and stores the userId, name, date of creation of account
- Each part-time rider works on a weekly work schedule
- Each part-time rider is identified by userId (primary key) which is referenced from riders (foreign key)

Full-Time Rider //still need?

- The full-time rider entity is a rider and stores the userId, name, date of creation of account
- Each full-time rider works on a monthly work schedule
- Each full-time rider is identified by userId (primary key) which is referenced from riders (foreign key)

Weekly Work Schedule (WWS)

- The WWS entity stores the scheduleId, userId, start date and end date of the schedule.
- Each WWS is uniquely identifiable by it's scheduleId (primary key), and belongs to a specific rider which is referenced by userId (foreign key)

Monthly Work Schedule (MWS)

- The MWS is a weak entity consisting of 4 WWS, and stores the scheduleId of each WWS (scheduleId1, scheduleId2, scheduleId3, scheduleId4)
- Each MWS is uniquely identifiable by the 4 scheduleId of the WWS it consists of (primary key), which is referenced from the WWS(foreign key)

Intervals

- The interval entity stores the intervalId, scheduleId, start time and end time of interval
- Each interval is uniquely identifiable by intervalId
- Each interval must belong to exactly one WWS that is referenced by scheduleId (foreign key)

Restaurant

- The restaurant entity stores the restaurant name, area of locality, and a minimum order amount

of any order to go through

- Each restaurant is uniquely identifiable by restaurant name
- Restaurants with the same name will have their location appended to thier restaurant name (e.g. Mac@WestCoastPark)

Food

- The food entity stores the food name and category of the food
- Each food is uniquely identified by its food name (primary key).

Orders

- The order entity stores the orderId, userId of the customer, the promotional code used, the restaurant name that the promo code is applicable to, the mode of payment by the customer, time of order being placed, delivery location, and reward points being used to offset the price
- Each order is uniquely identified by the orderId (primary key)
- Each order references the userId of the customer who created the order (foreign key)
- Each order references the promo code, together with the restaurant name that the promo code is applicable to (foreign key) to check if the promo code is valid
- Each order must be delivered exactly once by a rider

Promotions

- Each promotion entity stores the promo code of the promotion, the description, the creator of the promotion, the restaurant name it is applicable to, the unit of measurement of the discount, the rate of discount, and the start and end date of the promotion
- Each promotion is uniquely identifiable by the promo code coupled with the name of restaurant it is applicable to (primary key).
- For the same promo code, every restaurant that it is applicable to will be recorded in the promotions table. This facilitates the checking of the validity of use of the promo code

CustomerPromotions ?

- The customer promotions entity is a type of promotion, and stores the promo code, the restaurant name it is applicable to, and the (?)
- Each customer promotion is uniquely identified by the promo code coupled with the name of restaurant it is applicable to (primary key), which it references from Promotions (foreign key)

DeliveryPromotions ?

- The customer promotions entity is a type of promotion, and stores the promo code, the restaurant name it is applicable to, and the (?)
- Each customer promotion is uniquely identified by the promo code coupled with the name of restaurant it is applicable to (primary key), which it references from Promotions (foreign key)

MinSpendingPromotions ?

- The customer promotions entity is a type of promotion, and stores the promo code, the restaurant name it is applicable to, and the (?)
- Each customer promotion is uniquely identified by the promo code coupled with the name of restaurant it is applicable to (primary key), which it references from Promotions (foreign key)

Sells

- Sells is a relation between restaurants and food, and stores the restaurant name, food name, the price, as well as the quantity that is available for each food
- Each sells relation is uniquely identified by the restaurant name coupled with the food name (primary key)
- The restaurant name is referenced from restaurants (foreign key), while the food name is referenced from the food (foreign key)

Contains

- Contains is an aggregate relation between the sells relation and Orders entity, and stores the orderId it belongs to, the restaurant name and food name of the food, the quantity of the food ordered, as wells as the review of the ordered food item
- For the same orderId, each food item being ordered will recorded in the contains table. This facilitates reviewing each food item individually, as well as keeping track of the quantity ordered per food item
- Each contains entry is uniquely identified by orderId,the restaurant name and food name (primary key)
- The restaurant and food name is referenced by the sells relation (foreign key), and the orderId is referenced from the orders entity

Delivers

- Delivers is a relation between riders and orders, and stores the orderId for the order being delivered, the userId of the rider, the time he departs for the restaurant, the time he arrives at the restaurant, the time he leaves the restaurant, the delivery time to the customer, and the rating received for the delivery
- Each deliver is uniquely identified by orderId since every order must be delivered exactly once (primary key), and references rider for userId (foreign key)

3.2. Functionalities

The FDS application fulfils the following functionalities:

Customers	- Create / Update / Delete account - View his / her monthly statistics : (i) past orders (ii) past reveiws on orders - Browse / Search for food items by (i) name (ii) food category (iii) restaurant

Restaurant Staff	- Create / Update / Delete account - View his / her monthly statistics : (i) Total number of completed orders (ii) Total cost of all completed orders (excluding delivery fees) (iii) Top 5 favorite food items (in terms of the number of orders for that item). - View details of created promotions: (i) Duration (in terms of the number of days/hours) of the campaign (ii) Average number of orders received during the promotion
Delivery Riders	- Create / Update / Delete account (Full-time OR Part-time) - Declare their monthly schedule (Full-time) or weekly schedule (Part-time) - View his / her monthly statistics ((i) Orders delievered (ii) Hours worked (iii) Ratings received (iv) Salary earned (v) time taken to deliver food)
FDS Manager	- View monthly summary information for each Customers: (i) Total number of new customers (ii) Total number of orders (iii) Total cost of all orders - View monthly summary information for each Rider: (i) Total number of orders delivered by the rider fo (ii) Total number of hours worked by the rider (iii) Total salary earned by the rider (iv) Average delivery time by the rider (v) Ratings received by the rider for all the orders delivered (vi) Average rating received by the rider for all the orders delivered - View monthly summary information for Deliveries: (i) For each hour and for each delivery location area, the total number of orders placed at that hour for that location area.

3.3. Interesting / Non-trivial Functions

?

4. ER Model

4.1. Design considerations

(1) Promotions as an ISA relation to all sub promotions. By abstracting out attributes that are common to all promotions, we are able to achieve extensibility for promotions. This means it is easy to extend promotions and create more sub promotions. Restaurant Staff or the FDS Managers are able to create new types of sub promotions by identifying unique attributes which the sub promotion is based off.

By abstracting out key attributes of the promotion, different restaurants can now also create the same type of promotion but with the ability to customise it to thier needs e.g. start and end date, rate of discount etc.

(2)

4.2. Constraints not captured by ER Model

5. Database Relational Schema

Users schema

```
CREATE TABLE Users (
    userId       SERIAL,
    name        VARCHAR(100),
    PRIMARY KEY (userId)
);
```

Restaurants schema

```
CREATE TABLE Restaurants (
    rname       VARCHAR(200),
    minOrderAmt    NUMERIC(8, 2),
    area         VARCHAR(20),
    PRIMARY KEY (rname),
    CHECK(area = 'central' OR
        area = 'west' OR
        area = 'east' OR
        area = 'north' OR
        area = 'south')
);
```

Food schema

```
CREATE TABLE Food (
    fname       VARCHAR(20),
    category         VARCHAR(20) NOT NULL,
    PRIMARY KEY (fname),
    CHECK (category = 'western' OR
        category = 'chinese' OR
        category = 'japanese' OR
        category = 'korean' OR
        category = 'fusion')
);
```

Sells schema

```
CREATE TABLE Sells (
    rname       VARCHAR(20) REFERENCES Restaurants
                            on DELETE CASCADE
                            on UPDATE CASCADE,
    fname       VARCHAR(20) REFERENCES Food
                            on DELETE CASCADE
                            on UPDATE CASCADE,
    price       NUMERIC(8, 2) NOT NULL,
    availability    INTEGER DEFAULT 10,
    PRIMARY KEY (rname, fname)
);
```

Restaurant Staff schema

```
CREATE TABLE Restaurant_Staff (
    userId      INTEGER,
    rname         VARCHAR(20) REFERENCES Restaurants
                            on DELETE CASCADE
                            on UPDATE CASCADE,
    PRIMARY KEY (userId),
    FOREIGN KEY (userId) REFERENCES Users
                            on DELETE CASCADE
                            on UPDATE CASCADE
);
```

Customers schema

```
CREATE TABLE Customers (
    userId      INTEGER,
    creditCardInfo   VARCHAR(100),
    PRIMARY KEY (userId),
    FOREIGN KEY (userId) REFERENCES Users
                            on DELETE CASCADE
                            on UPDATE CASCADE
);
```

Riders schema

```
CREATE TABLE Riders (
    userId      INTEGER,
    area          VARCHAR(20) NOT NULL,
    PRIMARY KEY (userId),
    FOREIGN KEY (userId) REFERENCES Users
                            on DELETE CASCADE
                            on UPDATE CASCADE,
    CHECK(area = 'central' OR
        area = 'west' OR
        area = 'east' OR
        area = 'north' OR
        area = 'south')
);
```

Part-time schema

```
CREATE TABLE Part_Time
(
    userId               INTEGER,
    PRIMARY KEY (userId),
    FOREIGN KEY (userId) REFERENCES Riders
                            on DELETE CASCADE
                            on UPDATE CASCADE
    --        DEFERRABLE INITIALLY DEFERRED
);
```

Weekly Work Schedules (WWS) schema

```
CREATE TABLE Weekly_Work_Schedules
(
    scheduleId              SERIAL,
    userId                  INTEGER,
    startDate               TIMESTAMP,
    endDate                 TIMESTAMP,
    PRIMARY KEY (scheduleId),
    FOREIGN KEY (userId) REFERENCES Riders (userId),
        check ((endDate::date - startDate::date) = 6)
);
```

Monthly Work Schedules (MWS) schema

```
CREATE TABLE Monthly_Work_Schedules (
    scheduleId1             INTEGER REFERENCES Weekly_Work_Schedules
                                    ON DELETE CASCADE,
    scheduleId2             INTEGER REFERENCES Weekly_Work_Schedules
                                    ON DELETE CASCADE,
    scheduleId3             INTEGER REFERENCES Weekly_Work_Schedules
                                    ON DELETE CASCADE,
    scheduleId4             INTEGER REFERENCES Weekly_Work_Schedules
                                    ON DELETE CASCADE,
    PRIMARY KEY (scheduleId1, scheduleId2, scheduleId3, scheduleId4)
);
```

Intervals schema

```
CREATE TABLE Intervals
(
    intervalId              SERIAL,
    scheduleId              INTEGER,
    startTime               TIMESTAMP,
    endTime                 TIMESTAMP,
    PRIMARY KEY (intervalId),
    FOREIGN KEY (scheduleId) REFERENCES Weekly_Work_Schedules (scheduleId)
                                ON DELETE CASCADE,
        check (DATE_PART('minutes', startTime) = 0
        AND
           DATE_PART('seconds', startTime) = 0
        AND
           DATE_PART('minutes', endTime) = 0
        AND
           DATE_PART('seconds', startTime) = 0
        AND
           DATE_PART('hours', endTime) - DATE_PART('hours', startTime) <= 4
        AND
           startTime::date = endTime::date
        AND
           DATE_PART('hours', endTime) > DATE_PART('hours', startTime)
        AND
           startTime::time >= '10:00'
        AND
           endTime::time <= '22:00'
        )
);
```

Promotions schema

```
CREATE TABLE Promotions (
    promoCode       VARCHAR(20),
    promoDesc      VARCHAR(200),
    createdBy       VARCHAR(50), --?
    applicableTo     VARCHAR(200) REFERENCES Restaurants(rname)
                                    ON DELETE CASCADE,
    discUnit         VARCHAR(20) NOT NULL,
    discRate         VARCHAR(20) NOT NULL,
    startDate       TIMESTAMP NOT NULL,
    endDate       TIMESTAMP NOT NULL,
    PRIMARY KEY (promoCode, applicableTo)
);
```

Orders schema

```
CREATE TABLE Orders (
    orderId         INTEGER,
    userId              INTEGER NOT NULL REFERENCES Customers ON DELETE CASCADE ON
UPDATE CASCADE,
    promoCode      VARCHAR(20),
    applicableTo        VARCHAR(200),
    modeOfPayment       VARCHAR(10) NOT NULL,
    timeOfOrder  TIMESTAMP NOT NULL,
    deliveryLocation    VARCHAR(100) NOT NULL,
    usedRewardPoints    INTEGER DEFAULT 0,
    givenRewardPoints   INTEGER NOT NULL,
    PRIMARY KEY(orderId),
    FOREIGN KEY(promoCode, applicableTo)  REFERENCES Promotions,
    CHECK(modeOfPayment = 'cash' OR
          modeOfPayment ='credit')
);
```

Contains schema

```
CREATE TABLE Contains (
    orderId         INTEGER REFERENCES Orders
                                ON DELETE CASCADE
                                ON UPDATE CASCADE,
    rname           VARCHAR(100),
    fname          VARCHAR(100),
    foodQty       INTEGER NOT NULL,
    reviewContent            VARCHAR(300),
    PRIMARY KEY(orderId, rname, fname),
    FOREIGN KEY(rname, fname) REFERENCES Sells(rname, fname),
    CHECK(foodQty >= 1)
);
```

Delivers schema

```
CREATE TABLE Delivers (
    orderId                 INTEGER REFERENCES Orders
                                    ON DELETE CASCADE
                                    ON UPDATE CASCADE,
    userId                  INTEGER NOT NULL,
    departTimeForRestaurant  TIMESTAMP,
    departTimeFromRestaurant TIMESTAMP,
    arrivalTimeAtRestaurant  TIMESTAMP,
    deliveryTimetoCustomer   TIMESTAMP,
    rating             INTEGER,
    PRIMARY KEY (orderId),
    FOREIGN KEY (userId) REFERENCES Riders
                         ON DELETE CASCADE,
    CHECK(rating <= 5)
);
```

MinSpendingPromotions schema

```
CREATE TABLE MinSpendingPromotions (
    promoCode      VARCHAR(20),
    applicableTo    VARCHAR(200),
    minAmt        NUMERIC(8, 2) DEFAULT 0,
    PRIMARY KEY (promoCode, applicableTo),
    FOREIGN KEY (promoCode, applicableTo) REFERENCES Promotions
                                          ON DELETE CASCADE
                                          ON UPDATE CASCADE
);
```

CustomerPromotions schema

```
CREATE TABLE CustomerPromotions (
    promoCode         VARCHAR(20),
    applicableTo       VARCHAR(200),
    minTimeFromLastOrder   INTEGER, -- # of days
    PRIMARY KEY (promoCode, applicableTo),
    FOREIGN KEY (promoCode, applicableTo) REFERENCES Promotions
                                          ON DELETE CASCADE
                                          ON UPDATE CASCADE
);
```

5.1. Constraints not captured by Relational Schema

Intervals - For the same rider, no intervals should overlap with one another. There must be at least 1 hour of break between any 2 consecutive intervals. Intervals must fall within the start and end date of the WWS they belong to.

Weekly Work Schedule - For each worker, there should be no overlapping WWS. Each WWS must be at least 10 hours and at most 48 hours in total. Each WWS must be declared for exactly 7 consecutive days.

Monthly Work Schedule - For each week in of the MWS, the 4 comprising WWS must be equivalent. Each WWS should have 5 consecutive work days, that comprise of intervals using the pre-defined shifts for full-time riders. Each MWS should last for 28 days exactly, and there should not be any overlapping MWS for the same rider.

Promotions - Every promotion applied to an order has to be checked that it fulfils the promotions constraints such as the minAmount as well as timelastOrdered

Riders - During the operation hours of the FDS, there should be at least five riders (part-time or full-time) working at each hourly interval.

Orders - Quantity of food ordered for a particular food item cannot exceed it's availability. Total cost order must hit a certain minimum order amount set by the restaurant.

5.2. 3NF/BCNF

6. Triggers

6.1. 1

6.2. 2

6.3. 3

7. Complex Queries

7.1. 1

7.2. 2

7.3. 3

8. Software tools / Frameworks

Frontend :

Platform : Node.js

Framework : Express.js

Database : PostgreSQL [v?]

Languages used

- Javascript
- SQL for database

9. Application Screenshot

10. Conclusion