**CS 2102: Group Project**

# **Group 57 – Restaurant Reservation**

# **Group Members:**

## Oliver Zheng

## Adrianna Fu

## Lua Jun Kai Edenuis

**Project Responsibilities**

|  |  |
| --- | --- |
| Oliver Zheng | Developed majority of the web app, helped design the ER Diagram, wrote some minor queries. |
| Lua Jun Kai Edenuis | Wrote triggers, helped design the ER diagram, implemented reservation in the web app, and generated initialization data. |
| Adrianna Fu | Wrote a few interesting queries, helped design the ER diagram, implemented incentives in the web app, and helped with data initialization. |

**Project Description**

Our web app allows two types of users to use restaurant reservation system: customers, who can make reservations at and leave reviews for restaurants, and owners, who manage restaurant branch pages and respond to reviews.

All users can:

* Create an account with a unique email address

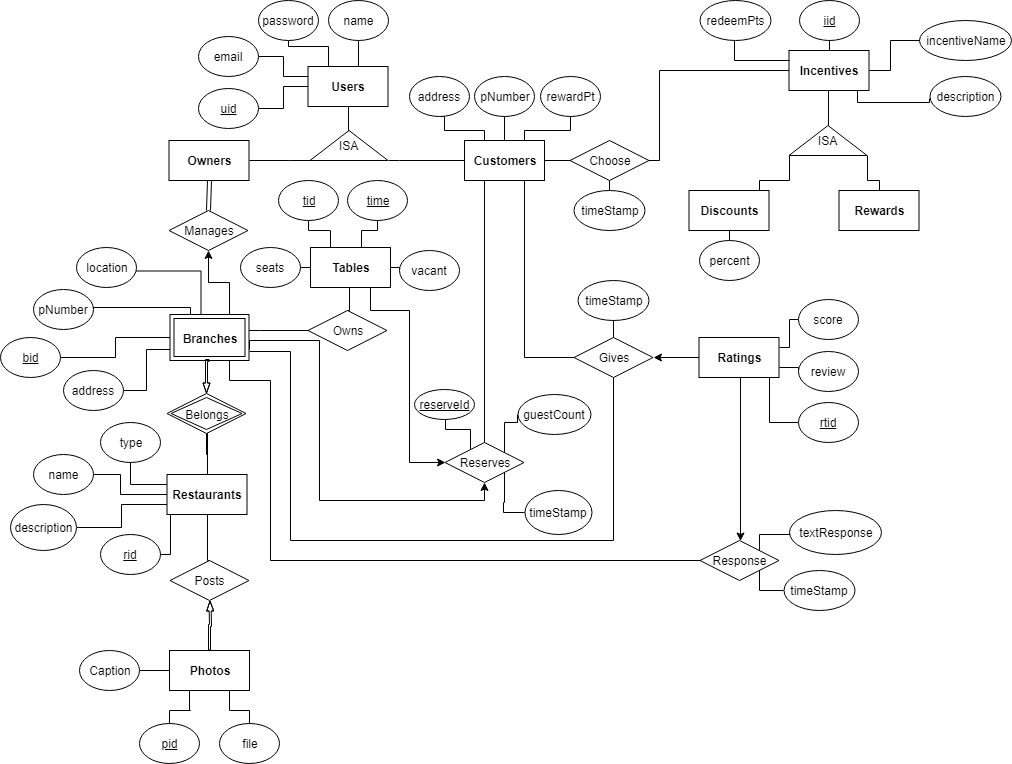
Customers can:

* Search up restaurants by name
* Make reservations at a restaurant branch for a certain time and date in the future for a set number of guests
* Leave reviews for restaurants they have visited
* Earn points for making reservations
* Use points to redeem rewards

Owners can:

* Respond to reviews from customers
* See reservations
* Add restaurant branches that they manage

**ER Model**



**Relational Schema**

create table Users (

uid SERIAL,

name VARCHAR(50) NOT NULL,

email VARCHAR(50) UNIQUE,

password VARCHAR(50) NOT NULL,

primary key (uid)

);

create table Customers (

uid INT NOT NULL,

address VARCHAR(50),

pNumber CHAR(8) NOT NULL UNIQUE,

rewardPt INT DEFAULT 0 CHECK (rewardPt >= 0),

foreign key (uid) references Users(uid) on delete cascade

);

create table Restaurants (

rid SERIAL,

name TEXT NOT NULL UNIQUE,

type TEXT,

description TEXT,

primary key (rid)

);

create table Branches (

rid INT NOT NULL,

bid INT NOT NULL,

pNumber CHAR(8) UNIQUE,

address VARCHAR(100) NOT NULL,

location TEXT,

primary key (bid, rid),

foreign key (rid) references Restaurants (rid) on delete cascade

);

create table Owners (

uid INT,

rid INT,

bid INT,

primary key (rid, bid),

foreign key (uid) references Users (uid) on delete cascade,

foreign key (rid, bid) references Branches (rid, bid) on delete cascade

);

CREATE TABLE Incentives (

iid SERIAL,

incentiveName TEXT NOT NULL UNIQUE,

description TEXT,

redeemPts INT CHECK(redeemPts >= 0),

primary key (iid)

);

CREATE TABLE Discounts (

iid INT,

percent INT CHECK(percent > 0 and percent <= 100),

PRIMARY KEY (iid),

FOREIGN KEY (iid) REFERENCES Incentives (iid) on delete cascade

);

CREATE TABLE Prizes (

iid INT,

PRIMARY KEY (iid),

FOREIGN KEY (iid) REFERENCES Incentives (iid) on delete cascade

);

create table Choose (

timestamp TIMESTAMPTZ NOT NULL,

uid INT NOT NULL,

iid INT NOT NULL,

foreign key (uid) references Users (uid) on delete cascade,

foreign key (iid) references Incentives (iid) on delete cascade

);

create table Ratings (

rtid SERIAL,

uid INT NOT NULL,

score NUMERIC (1) DEFAULT 0 CHECK(score>0 and score<=5),

review TEXT,

PRIMARY KEY (rtid),

FOREIGN KEY (uid) references Users (uid) on delete cascade

);

create table Gives (

timeStamp TIMESTAMPTZ NOT NULL,

uid INT,

rtid INT,

rid INT NOT NULL,

bid INT NOT NULL,

PRIMARY KEY (uid, rtid, rid, bid),

FOREIGN KEY (rtid) references Ratings (rtid) on delete cascade,

FOREIGN KEY (uid) references Users (uid) on delete cascade,

FOREIGN KEY (rid, bid) references Branches (rid, bid) on delete cascade

);

create table Response (

timeStamp TIMESTAMPTZ NOT NULL,

rtid INT,

rid INT NOT NULL,

bid INT NOT NULL,

textResponse TEXT NOT NULL,

PRIMARY KEY (rtid, rid, bid),

FOREIGN KEY (rtid) references Ratings(rtid) on delete cascade,

FOREIGN KEY (rid, bid) references Branches(rid, bid) on delete cascade

);

CREATE TABLE Photos (

pid SERIAL,

rid INT NOT NULL,

caption TEXT,

file TEXT NOT NULL,

PRIMARY KEY (pid),

FOREIGN KEY (rid) references Restaurants on delete cascade

);

create table Reserves (

reserveId SERIAL,

timeStamp TIMESTAMPTZ NOT NULL,

guestCount INT NOT NULL CHECK(guestCount > 0),

uid

PRIMARY KEY (reserveId)

);

create table Tables (

tid SERIAL,

time VARCHAR(4),

rid INT NOT NULL,

bid INT NOT NULL,

reserveId INT UNIQUE,

vacant BOOLEAN NOT NULL,

seats INT NOT NULL CHECK(seats > 0),

PRIMARY KEY (tid, time),

FOREIGN KEY (reserveId) references Reserves (reserveId),

FOREIGN KEY (rid, bid) references Branches (rid, bid)

);

**Triggers**

**Trigger 1:** Applies a unique constraint for a branch’s location on each individual restaurant (i.e. no restaurant should have two branches in the same location).

CREATE OR REPLACE FUNCTION branch\_location\_check()

RETURNS TRIGGER AS

$$

DECLARE count NUMERIC;

BEGIN

SELECT COUNT(\*) into count

FROM Branches B

WHERE NEW.rid = B.rid

AND NEW.address = B.address;

IF count > 0 THEN

RAISE NOTICE 'There is already a branch in that location!';

RETURN NULL;

ELSE

RETURN NEW;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER branch\_location\_check

BEFORE INSERT OR UPDATE ON Branches

FOR EACH ROW

EXECUTE PROCEDURE branch\_location\_check();

**Trigger 2**: Customers cannot redeem a reward when they have insufficient points.

CREATE OR REPLACE FUNCTION redeem\_points\_check()

RETURNS TRIGGER AS

$$

DECLARE points\_available NUMERIC;

DECLARE points\_needed NUMERIC;

BEGIN

SELECT rewardPt INTO points\_available

FROM Customers C

WHERE NEW.uid = C.uid;

IF EXISTS(SELECT 1 FROM Incentives I2 where I2.iid=NEW.iid) THEN

SELECT I3.redeemPts INTO points\_needed

FROM Incentives I3

Where I3.iid = NEW.iid;

END IF;

IF points\_available >= points\_needed THEN

Update Customers C1

SET rewardPt = points\_available - points\_needed

WHERE C1.uid = NEW.uid;

RETURN NEW;

ELSE

RAISE NOTICE 'Not enough points to redeem reward!';

RETURN NULL;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER redeem\_points\_check

BEFORE INSERT OR UPDATE ON Choose

FOR EACH ROW

EXECUTE PROCEDURE redeem\_points\_check();

**Trigger 3:** When a customer leaves a rating, the rating must have either a score or a review.

CREATE OR REPLACE FUNCTION ratings\_review\_check()

RETURNS TRIGGER AS

$$

BEGIN

IF NEW.score IS NULL AND NEW.review IS NULL THEN

RAISE NOTICE 'Invalid!';

RETURN NULL;

ELSE

RETURN NEW;

END IF;

END;

$$ LANGUAGE plpgsql;

CREATE TRIGGER ratings\_review\_check

BEFORE INSERT OR UPDATE ON Ratings

FOR EACH ROW

EXECUTE PROCEDURE ratings\_review\_check();

**Interesting Queries**

**Query 1:** Average rating for a restaurant branch (single location)

select r.name, b.location, avg(rt.score)

from ((branches b inner join gives g on b.bid = g.bid)

  inner join ratings rt on rt.rtid = g.rtid)

  inner join restaurants r on r.rid = b.rid

group by (r.name, b.location);

**Query 2:** Average rating for a restaurant (all the branches/locations of a restaurant)

select r.name, avg(rt.score)

from (restaurants r inner join branches b on r.rid = b.rid

inner join gives g on g.bid = b.bid

inner join ratings rt on rt.rtid = g.rtid)

group by r.rid;

**Query 3**: Average rating given by a customer

select u.name, avg(rt.score)

from (ratings rt natural join customers c)

inner join users u on c.uid = u.uid

group by (u.name);

**Query 4:** Labels each user as either a customer or an owner

create or replace view accountTypes (uid, isCustomer, isOwner) as

select uid,

coalesce((select true from Customers C where C.uid = U.uid), false),

coalesce((select true from (select distinct uid from Owners O) as A where A.uid = U.uid), false)

from Users U;

**Software Tools/Frameworks**

We used NodeJS and PostgreSQL for back-end and Bootstrap for front-end, as suggested by the Web App Development Guide. We used the HTML feature of ejs to handle the view (what the user sees). Some other tools we used to complete the project included Passport, Session, Express JS, and Google Fonts. Additionally, we used GitHub and Dropbox Paper to collaborate when writing the program.

**Application in Action**

~ insert screenshots ~

**Summary**

We are happy with the outcome of our project, though at the end we felt pressed for time as a group member left halfway through the semester and we were a little late to transforming the ER diagram into a functional web app.

We had trouble coming up with an ER diagram that fulfilled the requirements without adding trivial features, and had to consult the professor a few times. Even then, we still had to make changes while we implemented the database into the website. We should have started coding earlier, even with an incomplete ER diagram, so that we could have added tables where it made sense in the program instead of starting later with a complete but flawed ER diagram. The lesson here is the same as most projects: start early!

Additionally, the web app we created to complement our restaurant reservation database was more complex than the start guide led us to believe. It was far more complicated to implement than the example given in the guide, and the lecture on application development did not provide enough information work on the project. We had to learn from other resources (mainly online) to successfully finish our project.

Lastly, the project felt much more centered on web development than on the logic behind querying and using databases. We spent more time making webpages look nice than writing queries, which makes me wonder how effective this project is as a culmination of this class. However, it was a lot of fun to see the end product and we got to practice writing triggers and interesting queries and see them in action.