HAPPY HOUR DATABASE

Hyeon Hong, Cary Kawamoto & Pengfei Zhu

Table of Contents

Introduction

Our project is to create a database used in the Happy Hours Mobile Application. The app focuses on helping people find discounts during happy hours and place reservations for a desired time and date at listed restaurants and bars. The rating and reviews are linked to registered users in this App.

This database keeps track of each restaurant's name, phone number, address, ID, type of cuisine or establishment (category), description, happy hour time frames, and meals offered. The phone number, address, and ID are unique for each restaurant. Each registered customer is described by his or her name, app-assigned customer number, date of birth, gender, and email. For each customer, the email and customer number are unique. Each customer can rate a restaurant on a scale of 1-5, and view all of the ratings they have assigned in the past. The customer also can check the day, time, and (if applicable) a description of happy hour.

The database provides menu information to customers, listing names, prices, and menu descriptions. This database also keeps track of reservations made by registered customers. For a given reservation, the database will keep track of its date, time, party size (headcount), the customer who made the reservation, and the restaurant at which it was placed. This database can keep track of how far is each customer is away from restaurants by miles to provide location information to App users.

In this database, our group has created seven tables to hold the data as follows:

RESTAURANT (Restaurant_id, Name, Phone, Address, Category, Description)

CUSTOMER (Fname, Lname, Email, DOB, Gender, Customer_id)

HAPPY_HOUR (Day, Start_time, End_time, Description, Re_id)

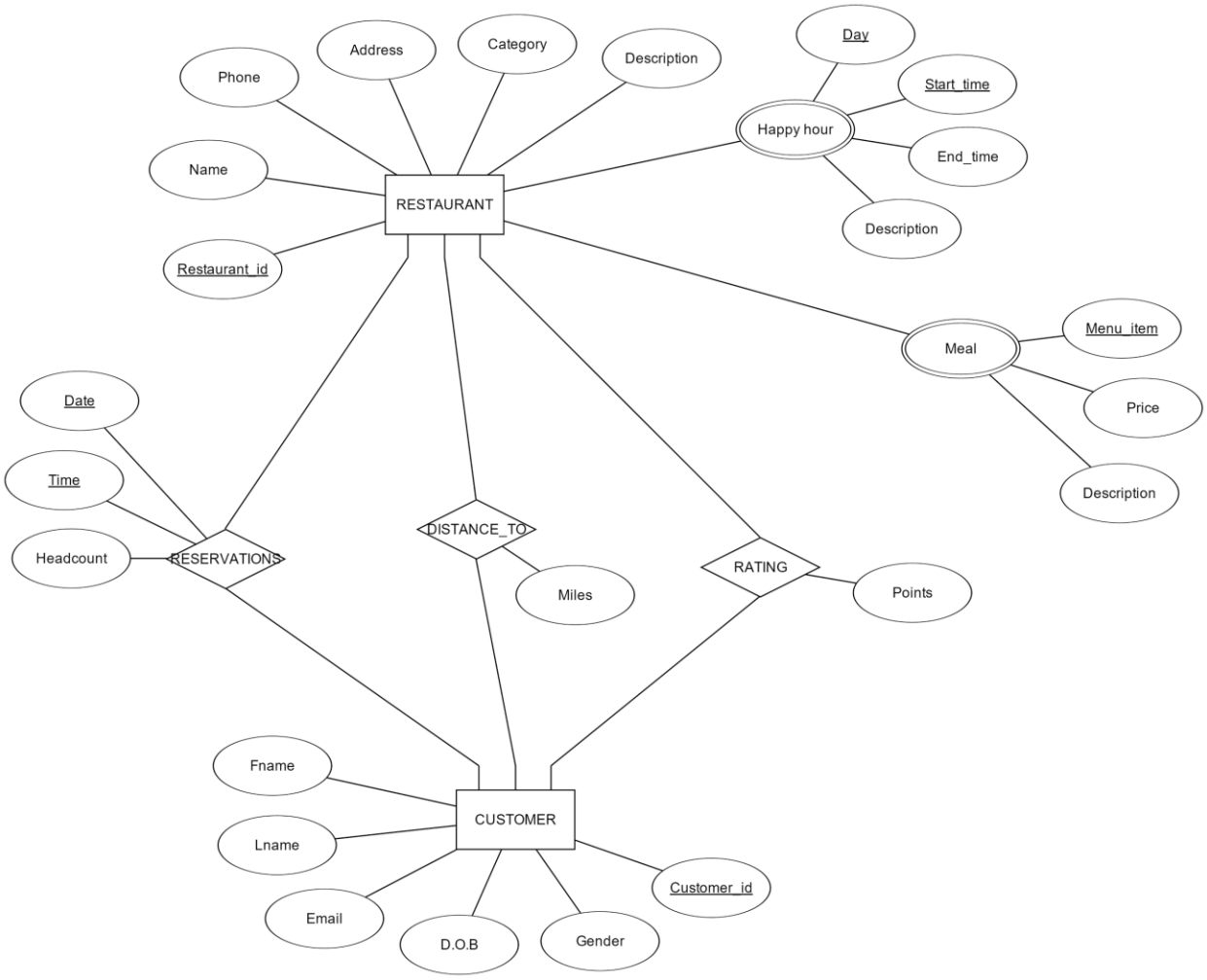
MEAL (Menu_item, Price, Description, Res_id)

RATING (Points, Cust_id, Restr_id)

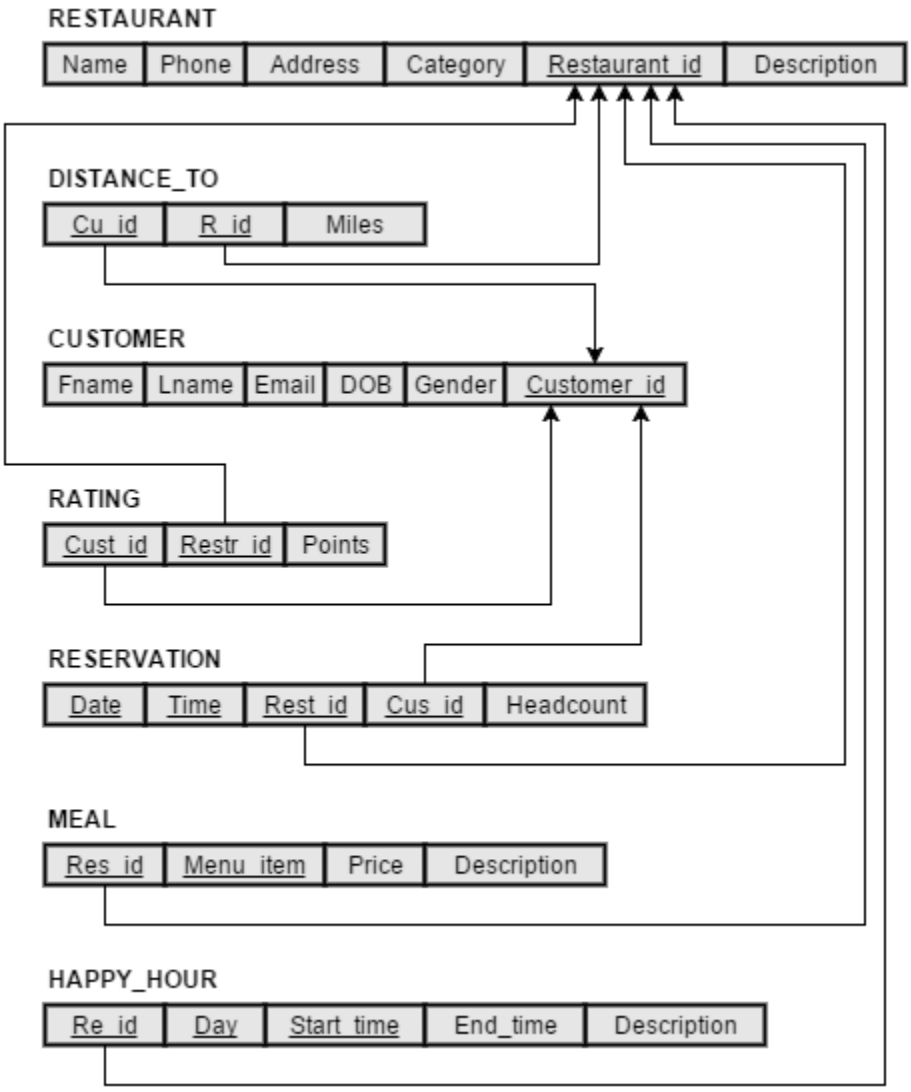
DISTANCE_TO (Cu_id, R_id, Miles)

RESERVATION (Date, Time, Rest_id, Cus_id, Headcount)

ER Diagram


Address
Category
Description
Phone
Day
Start_time
Happy hour
End_time
Name
RESTAURANT
Description
Restaurant_id
Meal
Menu_item
Price
Description
Date
Time
Headcount
RESERVATIONS
DISTANCE_TO
Miles
RATING
Points
Fname
Lname
CUSTOMER
Customer_id
Email
D.O.B
Gender

Relational Data Model



Populated Database

Create Tables

```
.open HAPPY_HOURS.db


CREATE TABLE CUSTOMER
(
fname VARCHAR(15) NOT NULL,
lname VARCHAR(15) NOT NULL,
email VARCHAR(30),
dob DATE,
gender CHAR(1) CHECK (gender in ('M', 'F')),
customer_id INT NOT NULL CHECK (customer_id > 0),


PRIMARY KEY(customer_id)
);


CREATE TABLE RESTAURANT
(
name VARCHAR(30) NOT NULL,
phone CHAR(10),
address VARCHAR(50),
category VARCHAR(15),
restaurant_id INT NOT NULL CHECK (restaurant_id > 0),
description VARCHAR(100),

PRIMARY KEY(restaurant_id)
);


CREATE TABLE RESERVATION
(
date DATE NOT NULL,
time INT NOT NULL CHECK (time >= 0 and time <= 2359 and time %100 < 60),
rest_id INT NOT NULL CHECK (rest_id > 0),
cus_id INT NOT NULL CHECK (cus_id > 0),
headcount INT CHECK (headcount > 0),
```

```
PRIMARY KEY (date, time, rest_id, cus_id),

constraint RESERFK1
FOREIGN KEY(cus_id) REFERENCES CUSTOMER(customer_id) on update cascade on
delete cascade,
constraint RESERFK2
FOREIGN KEY(rest_id) REFERENCES RESTAURANT(restaurant_id) on update cascade on
delete cascade
);


CREATE TABLE RATING
(
cust_id INT NOT NULL CHECK (cust_id > 0),
restr_id INT NOT NULL CHECK (restr_id > 0),
points INT CHECK (points between 1 and 5),

PRIMARY KEY(cust_id, restr_id),

constraint RATINGFK1
FOREIGN KEY(cust_id) REFERENCES CUSTOMER(customer_id) on update cascade on
delete cascade,
constraint RATINGFK2
FOREIGN KEY(restr_id) REFERENCES RESTAURANT(restaurant_id) on update cascade on
delete cascade
);


CREATE TABLE MEAL
(
res_id INT NOT NULL CHECK (res_id > 0),
menu_item VARCHAR(30) NOT NULL,
price DECIMAL(4, 2) NOT NULL CHECK (price > 0),
description VARCHAR(100),

PRIMARY KEY (res_id, menu_item),

constraint MEALFK
FOREIGN KEY(res_id) REFERENCES RESTAURANT(restaurant_id) on update cascade on
delete cascade
);
```

```
CREATE TABLE HAPPY_HOUR
(
re_id INT NOT NULL CHECK (re_id > 0),
day VARCHAR(10) NOT NULL CHECK (day in ('Monday', 'Tuesday', 'Wednesday', 'Thursday',
'Friday', 'Saturday', 'Sunday')),
start_time INT NOT NULL CHECK (start_time  >= 0 and start_time <= 2359 and
start_time %100 < 60),
end_time INT NOT NULL CHECK (end_time  >= 0 and end_time <= 2359 and end_time %100
< 60),
description VARCHAR(100),

PRIMARY KEY (re_id, day, start_time),

constraint HAPPYFK
FOREIGN KEY(re_id) REFERENCES RESTAURANT(restaurant_id) on update cascade on
delete cascade
);


CREATE TABLE DISTANCE_TO
(
cu_id VARCHAR(15) NOT NULL CHECK (cu_id > 0),
r_id int NOT NULL CHECK (r_id > 0),
miles decimal(5, 2) CHECK (miles >= 0),

Primary key (cu_id, r_id),

constraint DISFK1
FOREIGN KEY(cu_id) references CUSTOMER(customer_id) on update cascade on delete
cascade,
constraint DISFK2
FOREIGN KEY(r_id) references RESTAURANT(restaurant_id) on update cascade on delete
cascade
);
```

Insert values

INSERT INTO RESTAURANT VALUES
('Chiang''s Gourmet', 2065278888, '7845 Lake City Way NE, Seattle, WA 98115', 'Chinese', 1111, 'Traditional offerings in a casual setting.'),
('Ballard Mandarin Chinese Restaurant', 2067825531, '5500 8th Ave NW, Seattle, WA 98107', 'Chinese', 1112, null),
('Twins Garden', 2067292088, '10007 Aurora Ave N, Seattle, WA 98133', 'Chinese', 1113, 'A family owned establishment in North Seattle.'),
('Bamboo Garden Vegetarian Cuisine', 2062826616, '364 Roy St, Seattle, WA 98109', 'Chinese', 1114, 'Vegan options. Gluten-free, no MSG.'),
('Din Tai Fung', 2065250958, '2621 NE 46th St, Seattle, WA 98105', 'Chinese', 1115, 'Modern style decor, dim sum menu.'),
('Cactus South Lake Union', 2069132250, '350 Terry Ave N, Seattle, WA 98109', 'Mexican', 2101, 'Casual dining with popular lunch specials.'),
('El Farol Mexican Restaurant', 2063259005, '1467 E Republican St, Seattle, WA 98112', 'Mexican', 2102, 'Late night hours.'),
('Mexcaleria Oaxaca', 2062164446, '2123 Queen Anne Ave N, Seattle, WA 98119', 'Mexican', 2103, 'Tequila bar and tapas menu.'),
('Stumbling Monk', 2068600916, '1635 E Olive Way, Seattle, WA 98122', 'Pub', 3001, 'Hard-to-find local favorite.'),
('Quinn''s Pub', 2063257711, '1001 E Pike St, Seattle, WA 98122', 'Pub', 3002, 'Quaint aesthetic with local craft brews on tap.'),
('The George & Dragon Pub', 2065456864, '206 N 36th St, Seattle, WA 98103', 'Pub', 3003, 'English-style pub with lunch specials.'),
('Roxy''s Diner', 2066323963, '462 N 36th St, Seattle, WA 98103', 'Diner', 4321, 'Lunch deli with bar seating.'),
('Lost Lake Cafe & Lounge', 2063235678, '1505 10th Ave, Seattle, WA 98122', 'Diner', 4322, 'Open 24 hours, full bar and lounge seating.'),
('Steelhead Diner', 2066250129, '95 Pine St, Seattle, WA 98101', 'Diner', 4323, 'Comfort food with local ingredients.'),
('Q Nightclub', 2064329306, '1426 Broadway, Seattle, WA 98122', 'Nightclub', 9750, 'VIP seating. Make reservations early.'),
('Neighbours', 2063245358, '1509 Broadway, Seattle, WA 98122', 'Nightclub', 9751, 'Multi-level venue with live DJs.'),
('Foundation Nightclub', 2065357285, '2218 Western Ave #100, Seattle, WA 98121', 'Nightclub', 9752, 'Exclusive guest list, trendy decor.');

INSERT INTO CUSTOMER VALUES
('John', 'Smith', 'john_s253@yahoo.com', '1983-12-04', 'M', 40001),
('Erin', 'Bailey', 'eringreene@comcast.net', '1990-02-11', 'F', 72839),
('Catherine', 'Wan', 'kitty9lives@ymail.com', '1989-07-07', 'F', 12398),

('Shayam', 'Patel', 'sp1234@gmail.com', '1995-11-29', 'M', 85932),
('Ken', 'Lee', 'ken.lee@gmail.com', '1992-05-20', 'M', 49823);

INSERT INTO RATING VALUES
(40001, 2101, 4),
(40001, 2103, 5),
(40001, 9750, 3),
(40001, 3002, 5),
(40001, 4322, 3),
(40001, 1111, 1),
(40001, 1115, 5),
(40001, 1112, 3),
(72839, 2101, 5),
(72839, 2102, 2),
(72839, 3001, 2),
(72839, 1115, 5),
(12398, 2101, 4),
(12398, 2102, 1),
(12398, 2103, 5),
(12398, 4323, 2),
(12398, 3002, 3),
(12398, 4321, 2),
(85932, 9752, 5),
(85932, 2102, 3),
(85932, 2103, 3),
(85932, 9751, 4),
(85932, 1113, 2),
(49823, 1114, 3),
(49823, 3003, 3),
(49823, 4321, 2),
(49823, 2103, 2),
(49823, 2101, 5),
(49823, 9750, 3),
(49823, 1113, 2);

INSERT INTO DISTANCE_TO VALUES
(40001, 1111, 0.5),
(40001, 1112, 3.6),
(40001, 1113, 1.7),
(40001, 1114, 14.2),
(40001, 1115, 2.2),
(40001, 2101, 9.6),

(40001, 2102, 3.5),
(40001, 2103, 7.7),
(40001, 3001, 1.2),
(40001, 3002, 10.3),
(40001, 3003, 11.8),
(40001, 4321, 0.2),
(40001, 4322, 6.5),
(40001, 4323, 4.9),
(40001, 9750, 10.5),
(40001, 9751, 13.1),
(40001, 9752, 0.9),
(72839, 1111, 1.5),
(72839, 1112, 3.2),
(72839, 1113, 8.4),
(72839, 1114, 4.3),
(72839, 1115, 4.0),
(72839, 2101, 1.6),
(72839, 2102, 0.5),
(72839, 2103, 2.7),
(72839, 3001, 1.9),
(72839, 3002, 1.0),
(72839, 3003, 11.3),
(72839, 4321, 10.2),
(72839, 4322, 12.5),
(72839, 4323, 4.3),
(72839, 9750, 0.5),
(72839, 9751, 0.1),
(72839, 9752, 0.9),
(12398, 1111, 11.5),
(12398, 1112, 8.2),
(12398, 1113, 4.4),
(12398, 1114, 12.3),
(12398, 1115, 8.0),
(12398, 2101, 8.6),
(12398, 2102, 0.5),
(12398, 2103, 0.7),
(12398, 3001, 1.9),
(12398, 3002, 1.3),
(12398, 3003, 1.2),
(12398, 4321, 3.2),
(12398, 4322, 2.7),
(12398, 4323, 4.8),
(12398, 9750, 9.5),
(12398, 9751, 7.2),

(12398, 9752, 1.9),
(85932, 1111, 1.2),
(85932, 1112, 3.2),
(85932, 1113, 3.4),
(85932, 1114, 10.1),
(85932, 1115, 18.0),
(85932, 2101, 2.6),
(85932, 2102, 13.5),
(85932, 2103, 1.7),
(85932, 3001, 3.9),
(85932, 3002, 4.3),
(85932, 3003, 4.2),
(85932, 4321, 4.2),
(85932, 4322, 1.1),
(85932, 4323, 4.0),
(85932, 9750, 9.2),
(85932, 9751, 7.2),
(85932, 9752, 0.3),
(49823, 1111, 11.2),
(49823, 1112, 5.5),
(49823, 1113, 6.2),
(49823, 1114, 8.1),
(49823, 1115, 2.0),
(49823, 2101, 2.3),
(49823, 2102, 3.5),
(49823, 2103, 10.3),
(49823, 3001, 3.9),
(49823, 3002, 7.3),
(49823, 3003, 0.2),
(49823, 4321, 9.2),
(49823, 4322, 10.7),
(49823, 4323, 7.0),
(49823, 9750, 1.1),
(49823, 9751, 0.2),
(49823, 9752, 4.3);

INSERT INTO RESERVATION VALUES
('2016-02-29', 1830, 1115, 40001, 6),
('2016-02-25', 2000, 4323, 40001, 8),
('2016-03-05', 1200, 3003, 40001, 4),
('2016-03-12', 1900, 2102, 72839, 4),
('2016-03-11', 2200, 9750, 12398, 14),
('2016-02-14', 2015, 2103, 85932, 2),

('2016-03-08', 1230, 3002, 49823, 5),
('2016-03-10', 1230, 3001, 49823, 7),
('2016-03-04', 1930, 1113, 49823, 6),
('2016-03-05', 1830, 2103, 49823, 10),
('2016-02-14', 2230, 9750, 49823, 4);

INSERT INTO HAPPY_HOUR VALUES
(1111, 'Monday', 1800, 2100, 'Free appetizer with an entree.'),
(1112, 'Tuesday', 2100, 2300, 'Half price all beers.'),
(1113, 'Monday', 2100, 2300, null),
(1114, 'Friday', 1100, 1400, 'Free appetizer with an entree.'),
(1114, 'Monday', 1600, 1800, 'Half price beer and wine.'),
(1115, 'Sunday', 1700, 1900, null),
(2101, 'Monday', 1600, 1800, null),
(2101, 'Tuesday', 1100, 1400, 'Taco tuesday! Half price.'),
(2102, 'Thursday', 1700, 1900, 'Half price appetizers with an entree.'),
(2103, 'Tuesday', 1500, 1700, null),
(3001, 'Saturday', 1030, 1330, null),
(3001, 'Monday', 1030, 1330, 'Lunch specials.'),
(3001, 'Thursday', 1700, 1900, 'Half price appetizers with an entree.'),
(3002, 'Friday', 1700, 1900, '75% off local craft beer.'),
(3002, 'Saturday', 0000, 0200, 'Last call: Half price wells.'),
(3002, 'Monday', 1130, 1330, 'Lunch specials.'),
(3003, 'Monday', 1700, 1900, 'Half price appetizers with an entree.'),
(3003, 'Wednesday', 1800, 1930, '75% off local craft beer.'),
(4321, 'Wednesday', 1100, 1300, 'Half price on local craft beer.'),
(4322, 'Thursday', 2300, 0400, 'Special late night appetizer menu.'),
(4323, 'Monday', 1500, 1700, null),
(4323, 'Thursday', 1900, 2200, null),
(9750, 'Tuesday', 0100, 0200, 'Last call: Half price wells.'),
(9750, 'Thursday', 2200, 0100, 'Ladies'' night, half price on select cocktails.'),
(9750, 'Friday', 2000, 2100, 'Early bird: No cover charge.'),
(9751, 'Wednesday', 2300, 0100, 'Ladies'' night, half price on select cocktails.'),
(9751, 'Tuesday', 2200, 0200, null),
(9752, 'Thursday', 0000, 0200, 'Last call: Half price wells.'),
(9752, 'Monday', 2000, 2100, null),
(9752, 'Sunday', 1900, 2100, 'Early bird: No cover charge.');

INSERT INTO MEAL VALUES
(1111, 'Fried Rice', 7.99, 'Your choice of beef, pork or chicken.'),
(1111, 'Roasted Duck', 13.99, 'Roasted Duck in Beijing Style.'),
(1112, 'Mapo Tofu', 5.99, 'Traditional Chinese food in Sichuan.'),

(1112, 'Fried Green Bean', 8.99, 'A spicy item.'),
(1113, 'Orange Beef', 8.99, 'Beef with orange flavor sauce.'),
(1113, 'Sesame Beef', 9.99, 'Sauteed beef with sesame in sweet and spicy sauce.'),
(1114, 'Mongolian Beef', 10.95, 'Beef sauteed with onion and hot pepper, with rice sticks.'),
(1114, 'Hunan Chicken', 8.99, 'Diced chicken served with broccoli, onion, and mushroom in black bean sauce'),
(1115, 'Curry Chicken', 8.99, 'Diced chicken sauteed with curry.'),
(1115, 'Kung Pao Chicken', 7.99, 'Diced chicken served with peanuts and chili pepper.'),
(2101, 'Burrito', 6.99, 'Choice of meat.'),
(2101, 'Quesadilla combo', 8.99, 'Chicken or steak, with two tacos, beans, and rice.'),
(2102, 'Carne asada dinner', 15.99, 'Char-grilled steak and shrimp.'),
(2102, 'Tres enchiladas', 7.99, 'Three enchiladas with red or green sauce, beans, and rice.'),
(2103, 'Shot sampler', 14.99, 'Four shots of our featured tequilas.'),
(3001, 'Appetizer platter', 8.99, 'Chicken tenders, onion rings, fries, and mozzarella sticks.'),
(3002, 'Weekly tap', 5.00, '16 oz. of a local craft brew, on rotation.'),
(3003, 'Fry-up', 12.99, 'Traditional English breakfast, served all day.'),
(4322, 'Cheeseburger', 8.95, 'Includes fries, onion rings, or side salad.'),
(4322, 'Fish & chips', 9.95, 'Alaskan cod with hand-cut fries.'),
(4322, 'Short stack pancakes', 5.95, 'Two pancakes with two eggs, any style.'),
(4322, 'Sirloin steak', 12.95, 'Served with seasonal vegetables. Includes two sides.'),
(4322, 'Grilled cheese sandwich', 7.95, 'With fries, and a cup of tomato soup.'),
(4323, 'Fish & chips', 10.99, 'Alaskan cod with hand-cut fries.'),
(4323, 'Chicken fried steak', 10.99, 'Includes hashbrowns and two eggs.'),
(4323, 'Deli sandwich', 8.99, 'Your choice of turkey, ham, or roast beef, with fries.'),
(4323, 'Caesar salad', 7.99, 'Add grilled chicken for just $2.00 more.'),
(4323, 'Milkshake', 5.49, 'Vanilla, chocolate, or strawberry.'),
(4324, 'Fish & chips', 9.99, 'Alaskan cod with hand-cut fries.'),
(4324, 'Sirloin steak', 12.99, 'Served with seasonal vegetables. Includes two sides.'),
(4324, 'Lumberjack breakfast', 8.99, 'Two pancakes with sausage, bacon, eggs, and hashbrowns.'),
(4324, 'Cobb salad', 7.99, 'Fresh produce from local growers.'),
(4324, 'Club sandwich', 9.49, 'Turkey sandwich on sourdough, with all the trimmings.'),
(9750, 'Martini', 10.00, 'Includes all variations on the classic cocktail.'),
(9751, 'Blended cocktails', 9.00, 'Margaritas and pina coladas.'),
(9752, 'Well drinks', 5.00, 'All basic spirits, including vodka, rum, and gin.');

PRAGMA foreign_keys=1;

Tables

There are seven tables in this database: RESTAURANT, CUSTOMER, DISTANCE_TO, RATING, RESERVATION, HAPPY_HOUR, and MEAL.

SQL queries statement

SQL statement	Purpose
select distinct name from restaurant, happy_hour where re_id = restaurant_id and category = 'Chinese' and start_time < 1800 and end_time > 1600 collate nocase;	List all the Chinese restaurants that offer Happy Hour during 4-6pm. Function: list the restaurants that offer happy hour at the given time
select AVG (points) from rating, customer where cust_id = customer_id and fname = 'John' and lname = 'Smith' collate nocase;	Retrieve the average rating that customer John Smith has rated for all restaurants. Function: retrieve the average rating for a given customer
select name, sum(headcount) from restaurant, happy_hour, reservation where restaurant_id = rest_id and re_id = rest_id and day = 'Thursday' group by name;	List the total headcount for all reservations ever placed at every restaurant that offers Happy Hour on Thursdays. Function: list the total headcount of all reservation placed for the happy hour on given day
select name from restaurant, rating where restr_id = restaurant_id and category = 'Mexican' group by name having avg(points) > 3.5;	Retrieve all the Mexican restaurants that have an average rating higher than 3.5. Function: retrieve restaurants with the given category and rating
select avg(distinct (current_date - dob)) as Average_Age from customer, reservation, happy_hour where cus_id = customer_id and rest_id = re_id and day = 'Monday';	Retrieve the average age of each customer who placed a reservation at restaurants that have Happy Hour on Mondays. Function: retrieve the average age of the customers who place reservations for happy hour on a given day.
select name, menu_item, price from restaurant, meal where restaurant_id = res_id and price < 10 and res_id in (select res_id from happy_hour where day = 'Thursday');	Retrieve a list of all restaurants, menu items, and prices that cost less than $10 for restaurants that offer Happy Hour on Thursdays. Function: retrieve a list of restaurants, menu items, and prices for happy hour with a given price and a given day
select distinct name, miles from restaurant, distance_to, customer, happy_hour where restaurant_id = r_id and customer_id = cu_id and restaurant_id = re_id and fname = 'John' and lname = 'Smith' and miles < 10 and end_time > 1800;	For user John Smith, display the names and distances of all restaurants within 10 miles that have happy hour after 6pm. Function: retrieve a list of restaurant names and distances within a specific range and with a specific happy hour time frame

Challenges & Conclusions

We encountered some interesting situations when designing our database. Since we originally envisioned it as an idea for a mobile app, where people could locate restaurants, find happy hour discounts, place reservations, and plan pub-crawls, we knew that we needed to include distances as an attribute. However, as we had discussed for the exam question regarding ZIP codes, we realized this had to be information either stored in or supplied to the database. We had initially tried to come up with ways to relate distances to ZIP codes or latitude-longitude coordinates, but both of these proved to be impractical due to geographical constraints.

If this were a typical mobile app with distance calculations, it would have a separate function that takes advantage of GPS and geolocation to determine how far a user would be from one of the listed restaurants. For the purposes of our project, we decided that the information in our database, specifically the DISTANCE_TO table, would need to represent a “snapshot” in time. Under normal circumstances, a separate mapping function would need to update this table several times a minute, or upon user request.

Our group also had an interesting discussion in trying to decide how to store time information in our database. We had opted against using the Time data structure built into SQLite, since it was more specific that was practical for our purposes. Ultimately we decided to store time as an int, within the domain of 0 to 2359, as well as checking that time % 100 < 60, since happy hour time frames and reservations are rarely, if ever, more specific than a minute.

Using an int requires less storage space, using only 4 bytes; in the worst case scenario, the string “12:59 am” would require eight chars, taking up at least 16 bytes and as many as 32 depending on how it is represented. With the ASCII value for “:” being higher than “0-9,” it would have also resulted in odd behavior with string comparisons where “9:59 pm” > “10:00 pm” and “1:00 am” > “10:00 am”.

We had also briefly entertained the idea of having a unique username field for each CUSTOMER entity for the ID. Similar to the time issue, an ID assigned by the app that is only ever seen or used by the system is a significant savings on space versus a VARCHAR, especially when the email could also be assumed to be unique for each user.

With data population taking roughly 2-3 hours for only 17 restaurants, 5 customers, and a minimal amount of data for menus, scaling this database to accommodate thousands of users and locations would be a daunting task requiring several hundreds of hours. Over the course of this project, we gained a greater appreciation for the complexity of designing and implementing a mobile application, and a better understanding of SQL and databases.