Project 6 - OC Pizzeria Database

Description of the functional domain

A user can place an order using the web app, or the iOS/Android application. The order is then sent to the server which will then contact the chosen OC Pizzeria with the users order. The order is then displayed on a touch screen monitor that is near by the chefs station. The cooks will recognize the order and then mark the order as in progress using the same monitor. To keep the customer informed on their order during each step of the cooking process, the chefs at the pizzeria will update the order status using the kitchen order display. When the order is complete the chef will update the database by deducting the amount used for each ingredient as well as informing the user of the order status. The user will then pick up the order at the cashier when it is ready or wait for the delivery of the order. When the delivery is finished the driver will then mark the order as complete.

Technical Specification

1. To handle the orders for OC Pizzeria, there will be a native app released for iOS and Android, as well as a web application.
2. The web application will use HTML, CSS, and Javascript. While the iOS app will be written on Swift and the Android app on Java.
3. The Server will communicate with the client through REST APIs over HTTP to handle the orders asynchronously. The server will be written in NodeJS.
4. The database is written using MySQL.
5. When the client interacts with the mobile application this will send a request to the server using JSON, which will then be handled by the REST API to query the database for the specified request and serve the results.

Actors

User - Each user can log in to the app, browse through the menu for food to purchase, and check on the status of their order while it is being prepared. While

also being able to check their past orders, save credit card info and check/scan any coupons the user has for OCPizzeria through the app.

Chefs - By the chefs inside the OC Pizza store there will be an iPad to show what the upcoming orders are, and for the chefs to mark each step of the cooking process. When an order is put through the order process, their ingredients will automatically be deducted from the database.

Delivery Executive - Each order will have a receipt printed out when it is assigned as finished. This receipt will have the address to the person who made the order for the delivery person. When the delivery person comes back to the store they will input that each order delivered was complete.

Branch Manager(keeps tab on the stocks) - The branch manager of each location will be able to view the inventory of their branch through the mobile application. This can be accessed by logging into the app, and inside their account information there will be a tab to view the inventory. The inventory will be managed by the database and the chefs will primarily be the ones deducting products from inventory.

SQL Scripts:

Branches

Creating Branches Table:

```
CREATE TABLE Branches(
    branch_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    name VARCHAR(20) NOT NULL,
    location VARCHAR(50) NOT NULL,
    supervisor_ID INT NOT NULL,
    phone INT NOT NULL);
```

Adding Data to Branches Table:

```
INSERT INTO Branches VALUES (
  1, "Cambridge", "84 Harvard Ave.", 1, 6172938472);
```

```
INSERT INTO Branches
  (name, location, supervisor_ID, phone) VALUES
   ("Waltham", "227 Putnum Ave.", 2, 8573828393);
INSERT INTO Branches
  (name, location, supervisor_ID, phone) VALUES
  ("Belmont", "289 Western Ave.", 3, 8578839037);
INSERT INTO Branches
  (name, location, supervisor_ID, phone) VALUES
  ("Andover", "11 Elmwood Rd.", 4, 6178837993);
INSERT INTO Branches
  (name, location, supervisor_ID, phone) VALUES
  ("Newtonville", "566 Lemon St.", 5, 6175260365);
```

Employees

Create Employees Table:

```
  CREATE TABLE Employees(
    emp_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    first_name VARCHAR(20) NOT NULL,
    last_name VARCHAR(20) NOT NULL,
    age INT NOT NULL,
    phone INT NOT NULL,
    hire_date DATE NOT NULL,
    position INT NOT NULL,
    supervisor_ID INT NOT NULL,
    salary INT NOT NULL);
```

Add Data to Employees Table:

```
INSERT INTO Employees VALUES
```

```
  (100, "Christina", "Marshall", 32, 6178273823, "2019-03-11", 1, 100, 80000);
INSERT INTO Employees
  (first_name, last_name, age, phone, hire_date, position, supervisor_ID, salary) VALUES
  ("Rita", "Yu", 27, 8572938472, "2018-07-29", 1, 100, 65000);
INSERT INTO Employees
  (first_name, last_name, age, phone, hire_date, position, supervisor_ID, salary) VALUES
  ( "Alexio", "David", 22, 8574438432, "2018-02-19", 3, 101, 40000);
INSERT INTO Employees
  (first_name, last_name, age, phone, hire_date, position, supervisor_ID, salary) VALUES
  ("Damon", "Cropper", 34, 6173628394, "2018-08-04", 4, 101, 35000);
INSERT INTO Employees
  (first_name, last_name, age, phone, hire_date, position, supervisor_ID, salary) VALUES
  ("Ashley", "Torres", 19, 6176359573, "2019-08-27", 2, 101, 50000);
```

Positions

Create Positions Table:

```
  CREATE TABLE Positions(
    pos_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
    title VARCHAR(20) NOT NULL,
    starting_sal INT NOT NULL);
```

Add data to Positions

```
INSERT INTO Positions VALUES (
```

```
  1, "Supervisor", 65000);
INSERT INTO Positions
  (title, starting_sal) VALUES
  ("Cook", 50000);
INSERT INTO Positions
  (title, starting_sal) VALUES
  ("Driver", 40000);
INSERT INTO Positions
  (title, starting_sal) VALUES
  ("Cashier", 35000);
```

Inventory

Create Inventory Table:

```
    CREATE TABLE Inventory(
      food_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
      name VARCHAR(20) NOT NULL,
      quantity INT NOT NULL);
```

```
INSERT INTO Inventory VALUES
  (1, "Tomato Sauce", 302);
INSERT INTO Inventory (name, quantity)
  VALUES ("Pizza Dough", 102);
INSERT INTO Inventory (name, quantity)
  VALUES ("Pepperoni", 234);
INSERT INTO Inventory (food_ID, name, quantity)
  VALUES (4, "Mozzarella Cheese", 400);
INSERT INTO Inventory (food_ID, name, quantity)
  VALUES (5, "Coke", 382);
```

Customers

Create Customers Table:

```
    CREATE TABLE Customers(
      customer_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
      name VARCHAR(20) NOT NULL,
      address VARCHAR(50) NOT NULL,
      card_no INT);
```

Add data to Customers

```
INSERT INTO Customers VALUES
  (0, "NO NAME", "in-store", 0000000000000000);
INSERT INTO Customers
  (name, address, card_no) VALUES
  ("James Phillips", "293 Stonewall St.", 5172637872638372);
INSERT INTO Customers
  (name, address, card_no) VALUES
  ("Stephine Dorset", "12 Lincoln Way", 6273938173627281);
INSERT INTO Customers
  (name, address, card_no) VALUES
  ("Damien White", "89 Parker St.", 8374894774892839);
INSERT INTO Customers
  (name, address, card_no) VALUES
  ("Jacob Smith", "20 Third St.", 2998772617289927);
```

Orders

Create Orders Table:

```
    CREATE TABLE Orders(
      order_no INT PRIMARY KEY AUTO_INCREMENT,
      customer_ID INT NOT NULL,
      items VARCHAR(20) NOT NULL,
      order_type INT NOT NULL,
      time_placed TIME NOT NULL,
```

```
      pay_method INT NOT NULL);
```

Add data to Orders table

```
INSERT INTO Orders VALUES
  (100001, 0, "ORDER 100001", 3, "19:22:24", 1);
INSERT INTO Orders
  (customer_ID, items, order_type, time_placed, pay_method) VA
LUES
  (10001, "ORDER 100002", 2, "16:02:43", 3);
INSERT INTO Orders
  (customer_ID, items, order_type, time_placed, pay_method) VA
LUES
  (10004, "ORDER 100003", 1, "17:43:52", 2);
INSERT INTO Orders
  (customer_ID, items, order_type, time_placed, pay_method) VA
LUES
  (10004, "ORDER 100004", 1, "17:43:52", 4);
INSERT INTO Orders
  (customer_ID, items, order_type, time_placed, pay_method) VA
LUES
  (10002, "ORDER 100005", 1, "20:17:58", 3);
```

Orders_Type

Create Order_Type Table:

```
    CREATE TABLE Order_Type(
      type_ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
      title VARCHAR(20) NOT NULL);
```

Add data to Order_Type:

```
INSERT INTO Order_Type VALUES
  (1, "Delivery");
INSERT INTO Order_Type
```

```
  (title) VALUES
  ("Pick Up");
INSERT INTO Order_Type
  (title) VALUES
  ("In Store");
```

Deliveries

Create Deliveries Table:

```
    CREATE TABLE Deliveries(
      delivery_ID INT PRIMARY KEY NOT NULL,
      order_no INT NOT NULL,
      time_placed TIME NOT NULL,
      time_delivered TIME NOT NULL,
      driver_ID INT NOT NULL);
```

Add data into Deliveries:

```
INSERT INTO Deliveries VALUES
  (1, 100003, "17:43:52", "18:17:20", 102);
INSERT INTO Deliveries
  (order_no, time_placed, time_delivered, driver_ID) VALUES
  (100004, "17:43:52", "18:17:20", 102);
INSERT INTO Deliveries VALUES
  (order_no, time_placed, time_delivered, driver_ID) VALUES
  (100005, "20:17:58", "20:50:12", 102);
```

Transactions

Create Transactions Table:

```
    CREATE TABLE Transactions(
      transaction_no INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
```

```
      price VARCHAR(20) NOT NULL,
      pay_method INT NOT NULL,
      customer_ID INT NOT NULL);
```

Add data to Transactions table:

```
INSERT INTO Transactions VALUES
  (100001, "23.99", 1, 0);
INSERT INTO Transactions
  (price, pay_method, customer_ID) VALUES
  ("15.99", 3, 10001);
INSERT INTO Transactions
  (price, pay_method, customer_ID) VALUES
  ("45.99", 2, 10004);
INSERT INTO Transactions
  (price, pay_method, customer_ID) VALUES
  ("0.00", 4, 10004);
INSERT INTO Transactions
  (price, pay_method, customer_ID) VALUES
  ("18.99", 3, 10002);
```

Pay Method

Create Pay_Method Table:

```
    CREATE TABLE Pay_Method(
      ID INT PRIMARY KEY AUTO_INCREMENT NOT NULL,
      title VARCHAR(20) NOT NULL);
```

Adding Data to Pay_Method Table:

```
INSERT INTO Pay_Method VALUES (
  1, "Cash");
INSERT INTO Pay_Method (name) VALUES (
```

```
  "Credit");
INSERT INTO Pay_Method (name) VALUES (
  "Debit");
INSERT INTO Pay_Method (name) VALUES (
  "Cupon");
```

Example Queries:

IN:

List all the cooks, drivers and cashiers who were hired before 2019.

```
SELECT first_name, last_name, hire_date
FROM Employees
WHERE position NOT IN (1)
AND hire_date > "2018%";
```

LIKE:

List all the names and addresses of customers that live on a street.

```
SELECT name, address
FROM Customers
WHERE address
LIKE "%St.";
```

JOIN:

Get the total amount paid for each customer.

```
SELECT SUM(Transactions.price) AS AmountPaid, Customers.name A
S Names
FROM Transactions JOIN Customers
ON Transactions.customer_ID = Customers.customers_ID
GROUP BY name
ORDER BY AmountPaid DESC;
```

GROUP BY:

List the total age of all employees in each position.

```
SELECT SUM(age) AS total_age, position
FROM Employees
```

```
GROUP BY position
ORDER BY total_age DESC;
```

HAVING:

Find the total amount paid to each position and the names of each position not including the supervisors.

```
SELECT SUM(Employees.salary) AS amountPaid, Employees.positio
n, Positions.title AS title
FROM Employees JOIN Positions
ON Employees.position = Positions.pos_ID
GROUP BY Employees.position
HAVING Employees.position > 1;
```
