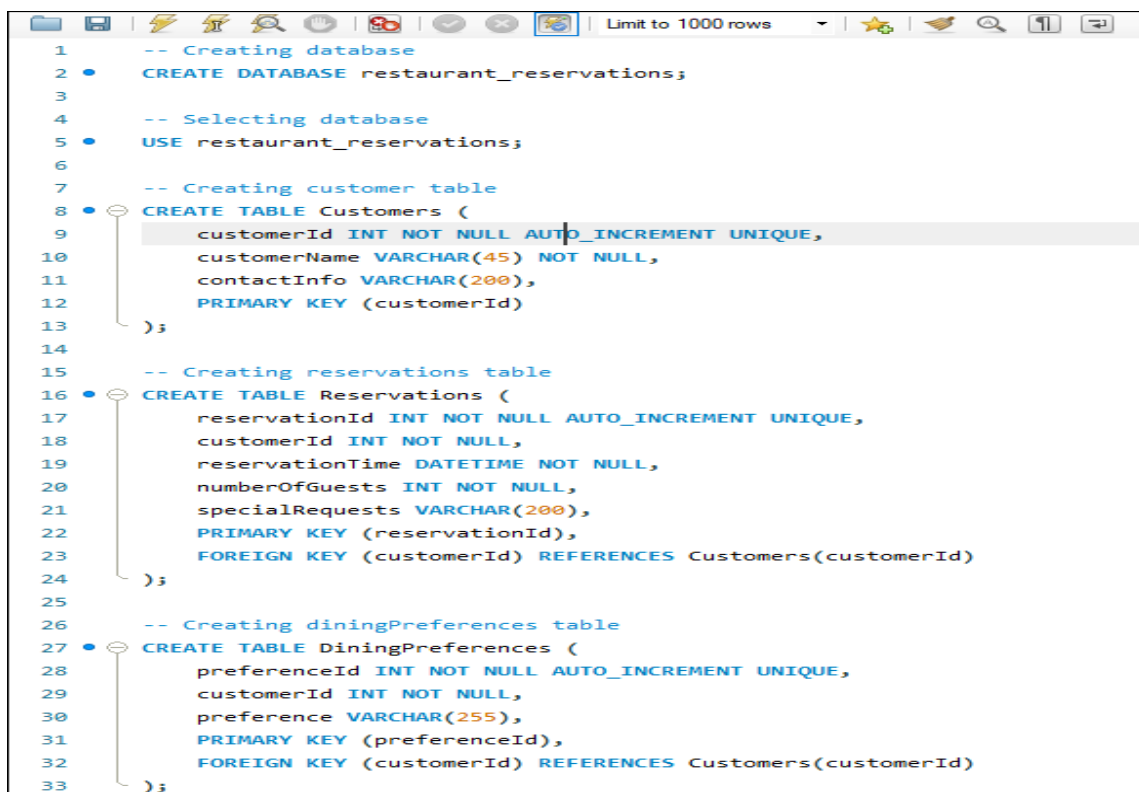


Starting off the project by first understanding the needs of the business, which in this case is the restaurant. Being that there is some freedom to this project I picked a restaurant from the internet called Schipper's. A database called restaurant\_reservations is first created. All the data required for the restaurant portal will be kept in this database. Next, we choose this database to make sure that everything that comes after is done in it. Three tables are then created: Reservations, DiningPreferences, and Customers. CustomerId is an auto-incremented primary key in the Customers database, which also has columns for customerName, contactInfo, and customerId. ReservationId is the main key and customerId is a foreign key that links to the Customers table. The Reservations table contains reservationId, customerId, reservationTime, numberOfGuests, and specialRequests. PreferenceId is the main key and customerId is a foreign key that links to the Customers table. The DiningPreferences table contains preference, customerId, and preference.

A screenshot of a MySQL command window. The window has a toolbar at the top with icons for file operations, execution, and navigation. Below the toolbar, the text "Limit to 1000 rows" is visible. The main area contains SQL code for creating a database and three tables. The code is as follows:

```
1  -- Creating database
2  CREATE DATABASE restaurant_reservations;
3
4  -- Selecting database
5  USE restaurant_reservations;
6
7  -- Creating customer table
8  CREATE TABLE Customers (
9      customerId INT NOT NULL AUTO_INCREMENT UNIQUE,
10     customerName VARCHAR(45) NOT NULL,
11     contactInfo VARCHAR(200),
12     PRIMARY KEY (customerId)
13 );
14
15 -- Creating reservations table
16 CREATE TABLE Reservations (
17     reservationId INT NOT NULL AUTO_INCREMENT UNIQUE,
18     customerId INT NOT NULL,
19     reservationTime DATETIME NOT NULL,
20     numberOfGuests INT NOT NULL,
21     specialRequests VARCHAR(200),
22     PRIMARY KEY (reservationId),
23     FOREIGN KEY (customerId) REFERENCES Customers(customerId)
24 );
25
26 -- Creating diningPreferences table
27 CREATE TABLE DiningPreferences (
28     preferenceId INT NOT NULL AUTO_INCREMENT UNIQUE,
29     customerId INT NOT NULL,
30     preference VARCHAR(255),
31     PRIMARY KEY (preferenceId),
32     FOREIGN KEY (customerId) REFERENCES Customers(customerId)
33 );
```

The next thing needed was to create a stored procedure that had to find the reservations and add special requests. In order to organize and encapsulate typical processes, we develop stored procedures. For instance, we may have a stored procedure to locate all reservations for a particular customer or to add a new reservation. To distinguish the SQL commands clearly when defining these stored procedures, we employ delimiters. At the start of the procedure definition, the default semicolon is replaced with a custom delimiter (such as //), and at the conclusion, it is returned to the default. This guarantees that the MySQL server will interpret the entire process as a single command. In order to ensure consistency and lower the possibility of errors, these stored procedures aid in streamlining and standardizing interactions with the database.

```
-- Creating Stored Procedure to find reservations
DELIMITER //
CREATE PROCEDURE findReservations(IN custId INT)
BEGIN
    SELECT * FROM Reservations WHERE customerId = custId;
END //
DELIMITER ;

-- Adding special requests
DELIMITER //
CREATE PROCEDURE addSpecialRequest(IN resId INT, IN requests VARCHAR(200))
BEGIN
    UPDATE Reservations
    SET specialRequests = requests
    WHERE reservationId = resId;
END //
DELIMITER ;

-- Adding reservations with customer check or creation
DELIMITER //
CREATE PROCEDURE addReservation(IN custId INT, IN resTime DATETIME, IN numGuests INT, IN specialReq VARCHAR(200))
BEGIN
    INSERT INTO Reservations (customerId, reservationTime, numberOfGuests, specialRequests)
    VALUES (custId, resTime, numGuests, specialReq);
END //
DELIMITER ;
```

The next thing that had to be done was populate the database. I created 3 tuples for customers, reservations, and dining preferences. I added their name and email address for customers, since this is standard when making a reservation at most high-end restaurants. For the reservations table I added a time, and a special request such as a certain place to sit or an allergy so that it won't be forgotten to the one managing or the host. Lastly dining preferences were created, this was made for custom orders or catering which is something that is done regularly. Here is the code.

```
-- Inserting data into Customers table
INSERT INTO Customers (customerName, contactInfo) VALUES
('Rick James', 'RickJames@gmail.com'),
('Will Ferrel', 'WillFerrel@gmail.com'),
('Jack Black', 'JackBlack@gmail.com');

-- Inserting data into Reservations table
INSERT INTO Reservations (customerId, reservationTime, numberOfGuests, specialRequests) VALUES
(1, '2024-05-16 18:30:00', 4, 'Peanut Allergy'),
(2, '2024-05-16 19:00:00', 2, 'Balcony Seats'),
(3, '2024-05-16 20:00:00', 2, 'No Special Request');

-- Inserting data into DiningPreferences table
INSERT INTO DiningPreferences (customerId, preference) VALUES
(1, 'A5 Wagyu'),
(2, 'Vegan'),
(3, 'Lamb Shank');
```

Now the Python is what we will be focusing on. There are two python files which is the files restaurantServer.py and restaurantDatabase.py. The RestaurantDatabase class, which manages all interactions with the MySQL database, is defined in the restaurantDatabase.py file. This class has methods for managing special requests, adding customers, creating and retrieving reservations, and

CIS 344

Final Project

Mark Munoz

connecting to the database. The RestaurantDatabase class uses the supplied credentials to establish a connection to the MySQL server upon activation. By requesting the customer ID, reservation time, number of guests, and any special requirements, the addReservation function adds a new reservation to the Reservations table. To display all reservations, use the get AllReservations function to extract all records from the Reservations table. Similar capability for managing customers and their reservations is offered by methods like addCustomer, addSpecialRequest, findReservations, and deleteReservation. Lastly The restaurantServer.py file implements the web server using Python's http.server module. It defines the RestaurantPortalHandler class, which extends BaseHTTPRequestHandler to handle HTTP requests and handles the logical and database of the site.