#10 - Database Management System

Documentation and Relational Algebra Statements

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

#1 - Database Management System

ER Diagram and Plan

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

Hotel Management

Hotel management is a vital aspect of the hospitality industry. A well-designed and implemented hotel management database management system (DBMS) can streamline operations and improve guest satisfaction. The database management system allows for effective data dictionary management, strong security management and a versatile multi-user-access control. The DBMS we will discuss here is designed for hotels to manage their day-to-day operations and keep track of guests, rooms, employees, bookings, payments, and services.

Entities

Hotel: The name, location, and contact details of the hotel are some of the attributes of the primary entity in our DBMS, the Hotel. Since this entity exists on its own, it is regarded as our owner and dominant entity. The main key attribute is the hotel ID because it is distinct.

Room: Each hotel contains a number of rooms, and the Room entity has properties such as room number, kind, floor, and cost. The main key attribute is the room ID. This organization is one of our subordinate organizations because it depends on the hotel.

Guest: The Guest entity is used by guests to make reservations at the hotel. Its features include name, contact information, and past reservations. The main key attribute is visitor ID. A guest entity is one of our subordinate entity types even though it is dependent on our hotel and booking entities and could not exist without the values it contains.

Employee: The Employee entity possesses properties like a name, a job title, and contact details. The main key attribute is the employee ID. The employee entity is one of our strong entity types since it dominates the service entity.

Booking: The Booking entity has attributes such as the guest ID, room ID, check-in and check-out dates, and any special requests. The booking ID is the main key attribute. The booking entity is very dependent on the other entities and will be considered one of our weak entity types

Payment: The Payment entity has attributes such as the booking ID, payment method, and amount paid. The payment ID is the primary key attribute. Most entities depend on the payment entity therefore is it one of our strong entity types

Service: The Service entity has attributes such as the name, service description, cost, and employee ID of the person providing the service. The service ID is the primary key attribute. The service entity is dependent on the employee entity, if the employee is removed, the service is no longer available, therefore, it is considered one of our weaker entity types.

Relationships

- The association between a hotel and its many rooms is formed by using the hotel ID as a foreign key in the room entity. We shall categorize this relationship as a one-to-many relationship.

- A Room may be reserved by more than one Guest; this relationship is established through the Booking entity's usage of the room ID as a foreign key. The relationship can be described as a many-to-many relationship because a Guest has the option of booking several rooms.

- A Guest may make several Bookings: This connection is made possible by the Booking entity's use of the guest ID as a foreign key. A one-to-many relationship could be used to describe the relationship.

- The association between an Employee and a certain Hotel is built by using the hotel ID as a foreign key in the Employee entity. One hotel can employ a large number of workers, making this arrangement a one-to-many relationship.

- A Booking is associated with a specific Room and Guest: This relationship is established through the use of the room ID and guest ID as foreign keys in the Booking entity. One booking is associated with only one room, however it can be associated with multiple guests. There are two relationships defined here, there is a one-to-one relationship between the Booking entity and the Room entity. The other relationship is a one-to-many relationship between the Booking entity and the Guest entity.

- A Payment is made in respect to a particular Booking: This connection is made possible by using the booking ID as a foreign key in the Payment entity. The relationship between the Payment entity and the Booking object is one-to-one since each payment refers to just one booking..

- A Room can have multiple Services: This relationship is established through the use of the room ID as a foreign key in the Service entity. This relationship is defined as a one-to-many relationship.

- A Service is provided by a specific Employee: This relationship is established through the use of the employee ID as a foreign key in the Service entity. As one service can be provided by only one specialized employee, the relationship is one-to-one.

- A Guest can request multiple Services during their stay: This relationship is established through the use of the guest ID as a foreign key in the Service entity. This relationship is defined as a one-to-many relationship.

- An Employee can provide a specific Service to multiple Guests: This relationship is established through the use of the employee ID as a foreign key in the Service entity. This relationship can be defined as a one-to-many relationship.

In conclusion, the DBMS outlined here is designed to manage the day-to-day operations of a hotel, including guest information, room availability, employee management, bookings, payments, and services. The database design of the computerized hotel management system aids to structure and manage their data in a logical way. The detailed data model consists of value parameters, attributes, primary key, foreign key and relationships between entities. These established entities and relationships allow for easy tracking and management of all aspects of the hotel's operations.

#2 - Database Management System

ER Model

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

The hotel management system can be represented as an ER model consisting of several entities and their relationships.

The main entity in the system is the Hotel, which has attributes such as name, location, and contact details. The Hotel entity is the owner entity since it exists on its own, and the hotel ID attribute is the primary key for this entity.

Another entity in the system is the Room, which has properties like room number, kind, floor, and cost. The Room entity is a subordinate entity since it depends on the Hotel entity, and the room ID attribute is the primary key for this entity. The relationship between the Hotel and Room entities is a one-to-many relationship since one hotel can have many rooms.

The Guest entity is used for guests to make reservations at the hotel. It has attributes like name, contact information, and past reservations, and the guest ID attribute is the primary key for this entity. The Guest entity is a subordinate entity since it depends on the Hotel and Booking entities, and it could not exist without the values it contains.

The Employee entity is another entity in the system, and it possesses properties like a name, job title, and contact details. The employee ID attribute is the primary key for this entity, and the Employee entity is a strong entity since it dominates the Service entity.

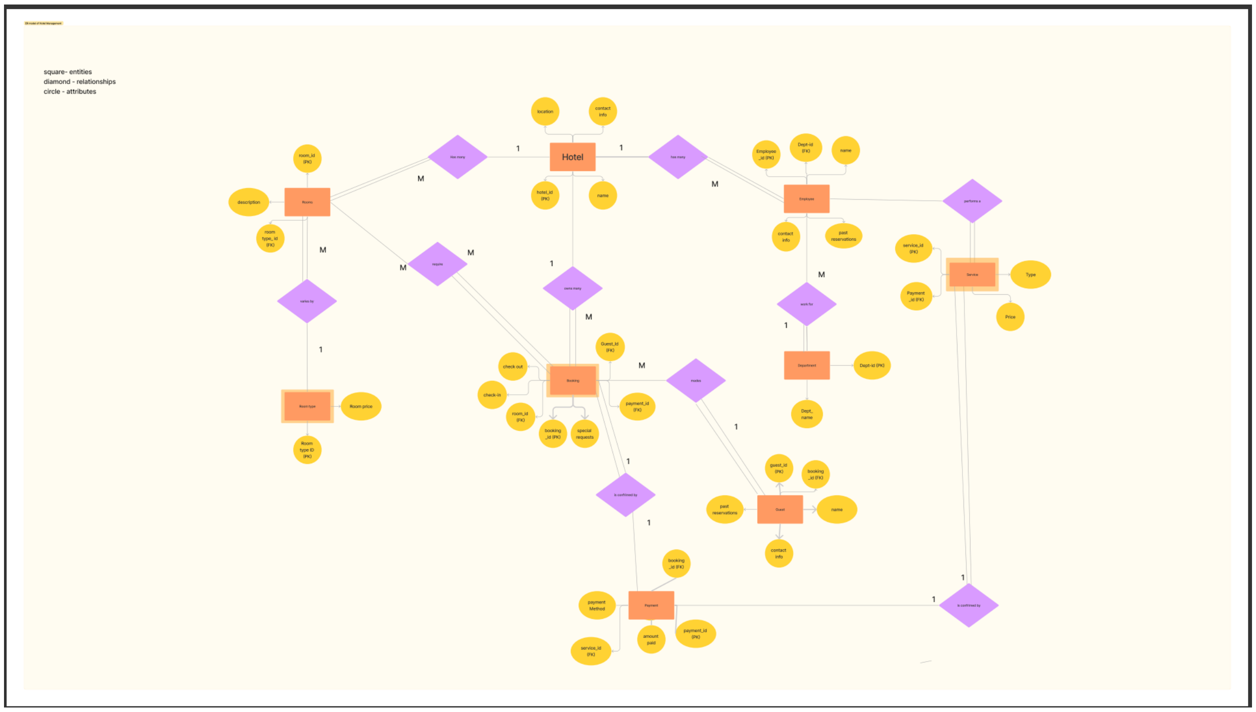
The Booking entity has attributes such as the guest ID, room ID, check-in and check-out dates, and any special requests. The booking ID attribute is the primary key for this entity, and it is a weak entity since it depends on the other entities. The relationship between the Booking entity and the Guest entity is a one-to-many relationship since one guest can make many bookings, and the relationship between the Booking entity and the Room entity is a one-to-one relationship since one booking is associated with only one room.

The Payment entity has attributes such as the booking ID, payment method, and amount paid. The payment ID attribute is the primary key for this entity, and the Payment entity is a strong entity since most entities depend on it.

The Service entity has attributes such as the name, service description, cost, and employee ID of the person providing the service. The service ID attribute is the primary key for this entity, and it is a weak entity since it depends on the Employee entity. The relationship between the Service entity and the Room entity is a one-to-many relationship since one room can have multiple services, and the relationship between the Service entity and the Guest entity is a one-to-many relationship since one guest can request multiple services. The relationship between the Service entity and the Employee entity is a one-to-one relationship since one service can be provided by only one specialized employee.

In summary, the hotel management system's ER model includes several entities such as Hotel, Room, Guest, Employee, Booking, Payment, and Service, and their relationships are defined by primary keys, foreign keys, and one-to-one or one-to-many relationships. This model helps to structure and manage data in a logical way, allowing for easy tracking and management of all aspects of the hotel's operations.

A representation of this can be shown below



#3 - Database Management System

Table Creation and SQL introduction

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

The above code is a SQL script that creates a relational database schema for a hotel management system. The schema consists of several tables with relationships between them, each representing a different aspect of hotel operations.

The "Hotel" table contains information about each hotel, such as the hotel name, location, and contact information. It has a primary key of "hotel_id" to ensure that each hotel is uniquely identified within the table.

The "Room" table contains information about each room in a hotel, including the room number, the type of room, and the hotel to which it belongs. It has a primary key of "room_id" and foreign keys that reference the "Hotel" and "RoomType" tables to establish the relationships between the tables.

The "RoomType" table contains information about each type of room, including the room price. It has a primary key of "roomtype_id" that is referenced by the "Room" table to link each room with its corresponding room type.

The "Guest" table contains information about each guest, including their name and contact information. It also has a foreign key that references the "Booking" table to link each guest with their corresponding booking.

The "Employee" table contains information about each hotel employee, including their name, job title, and contact information. It has a primary key of "employee_id" and a foreign key that references the "Department" table to link each employee with their corresponding department. Additionally, it has a foreign key that references the "Hotel" table to link each employee with their corresponding hotel.

The "Department" table contains information about each department within a hotel, such as housekeeping, front desk, and maintenance. It has a primary key of "dept_id" that is referenced by the "Employee" table to link each employee with their corresponding department.

The "Booking" table contains information about each guest booking, including the check-in and check-out dates, any special requests, and the room and guest associated with the booking. It has a primary key of "booking_id" and foreign keys that reference the "Guest", "Room", and "Payment" tables to establish the relationships between them.

The "Payment" table contains information about each guest payment, including the payment method and amount paid. It has a primary key of "payment_id" and foreign keys that reference the "Booking" and "Service" tables to link each payment with its corresponding booking and any associated services.

The "Service" table contains information about any additional services provided to guests, such as room service or cleaning. It has a primary key of "service_id" and foreign keys that reference the "Employee" and "Room" tables to link each service with the employee and room associated with it. It also includes information about the cost and description of each service.

The SQL script is presented below:

```
CREATE TABLE Hotel (
  hotel_id INT PRIMARY KEY,
  hotel_name VARCHAR(50),
  location VARCHAR(100),
  contact_info VARCHAR(20)
);

CREATE TABLE Room (
  room_id INT PRIMARY KEY,
  hotel_id INT,
  room_number INT,
  roomtype_ID INT,
  FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id),
  FOREIGN KEY (roomtype_id) REFERENCES Room(roomtype_id)
);

CREATE TABLE RoomType (
  roomtype_id INT PRIMARY KEY,
  room_price DECIMAL(10,2)
);

CREATE TABLE Guest (
  guest_id INT PRIMARY KEY,
  booking_id INT,
  guest_name VARCHAR(50),
  guest_contact_info VARCHAR(20),
  FOREIGN KEY (booking_id) REFERENCES Booking(booking_id)
);

CREATE TABLE Employee (
  employee_id INT PRIMARY KEY,
  dept_id INT,
  employee_name VARCHAR(50),
  job_title VARCHAR(50),
  contact VARCHAR(20),
  hotel_id INT,
  FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);
```

```sql
CREATE TABLE Department (
  dept_ID INT PRIMARY KEY,
  dept_name VARCHAR(50)
);

CREATE TABLE Booking (
  booking_id INT PRIMARY KEY,
  guest_id INT,
  room_id INT,
  payment_id INT,
  checkin_date DATE,
  checkout_date DATE,
  requests VARCHAR(100),
  FOREIGN KEY (guest_id) REFERENCES Guest(guest_id),
  FOREIGN KEY (room_id) REFERENCES Room(room_id),
  FOREIGN KEY (payment_id) REFERENCES Payment(payment_id)
);

CREATE TABLE Payment (
  payment_id INT PRIMARY KEY,
  booking_id INT,
  service_id INT,
  payment_method VARCHAR(20),
  amount_paid DECIMAL(10,2),
  FOREIGN KEY (booking_id) REFERENCES Booking(booking_id),
  FOREIGN KEY (service_id) REFERENCES Service(service_id)
);

CREATE TABLE Service (
  service_id INT PRIMARY KEY,
  employee_id INT,
  room_id INT,
  service_type VARCHAR(50),
  service_description VARCHAR(100),
  cost DECIMAL(10,2),
  FOREIGN KEY (employee_id) REFERENCES Employee(employee_id),
  FOREIGN KEY (room_id) REFERENCES Room(room_id)
);
```

#4 - Database Management System

Hotel Management Database System SQL Script with Dummy Data and Queries

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

`CREATE DATABASE hotelmanagement;` - This query creates a new database named "hotelmanagement".

`USE hotelmanagement;` - This query selects the "hotelmanagement" database for use. All subsequent queries will be executed in this database.

`CREATE TABLE Hotel (...)` - This query creates a new table named "Hotel" with four columns: hotel_id, hotel_name, location, and contact_info. The hotel_id column is set as the primary key for the table.

`CREATE TABLE RoomType (...)` - This query creates a new table named "RoomType" with two columns: roomtype_id and room_price. The roomtype_id column is set as the primary key for the table.

`CREATE TABLE Room (...)` - This query creates a new table named "Room" with three columns: room_id, hotel_id, and roomtype_id. The room_id column is set as the primary key for the table, and both hotel_id and roomtype_id columns have foreign key constraints that reference the Hotel and RoomType tables, respectively.

`CREATE TABLE Guest (...)` - This query creates a new table named "Guest" with three columns: guest_id, guest_name, and guest_contact_info. The guest_id column is set as the primary key for the table.

`CREATE TABLE Booking (...)` - This query creates a new table named "Booking" with six columns: booking_id, guest_id, room_id, checkin_date, checkout_date, and requests. The booking_id column is set as the primary key for the table, and both guest_id and room_id columns have foreign key constraints that reference the Guest and Room tables, respectively.

`CREATE TABLE Department (...)` - This query creates a new table named "Department" with two columns: dept_id and dept_name. The dept_id column is set as the primary key for the table.

`CREATE TABLE Employee (...)` - This query creates a new table named "Employee" with five columns: employee_id, dept_id, employee_name, job_title, and contact. The employee_id column is set as the primary key for the table, and the dept_id column has a foreign key constraint that references the Department table.

`CREATE TABLE Service_Type (...)` - This query creates a new table named "Service_Type" with two columns: service_type_id and service_description. The service_type_id column is set as the primary key for the table.

`CREATE TABLE Service (...)` - This query creates a new table named "Service" with four columns: booking_id, service_type_id, service_date, and amount. Both booking_id and service_type_id columns have foreign key constraints that reference the Booking and Service_Type tables, respectively.

`INSERT INTO Hotel (...)` - This query inserts rows of data into the Hotel table.

`SELECT * FROM Hotel;` - This query selects all rows and columns from the Hotel table.

`INSERT INTO Guest (...)` - This query inserts rows of data into the Guest table.
`INSERT INTO RoomType (...)` - This query inserts rows of data into the RoomType table.
`INSERT INTO Room (...)` - This query inserts rows of data into the Room table.
`INSERT INTO Department (...)` - This query inserts rows of data into the Department table.
`INSERT INTO Employee (...)` - This query inserts rows of data into the Employee table.
`INSERT INTO Service_Type (...)` - This query inserts rows of data into the Service_Type table.
`INSERT INTO Booking (...)` - This query inserts rows of data into the Booking table.
`INSERT INTO Service (...)` - This query inserts rows of data into the Service

Here is the implementation of the code in SQL

```
create database hotelmanagement; -- How to create

use hotelmanagement; -- Jumps into the database, must use it before making
any changes/queries

CREATE TABLE Hotel (  -- Every table must implement a primary key
  hotel_id INT PRIMARY KEY,
  hotel_name VARCHAR(50) NOT NULL, -- Characters up to 50 can be inputted
into this attribute
  location VARCHAR(100) NOT NULL,  -- Characters up to 50 can be inputted
into this attribute
  contact_info VARCHAR(20) NOT NULL  -- Characters up to 50 can be inputted
into this attribute
);

-- Defining the room type which can be single, double, queen, presidental
suite etc
CREATE TABLE RoomType (
  roomtype_id INT PRIMARY KEY, -- Int can only accept numerical values
  room_price DECIMAL(10,2) --  10 beginning and 2 after
);

CREATE TABLE Room (
  room_id INT PRIMARY KEY, -- The room_ID will also refer to room number
  hotel_id INT NOT NULL,
  roomtype_ID INT,
```

```sql
  FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id),
  FOREIGN KEY (roomtype_id) REFERENCES RoomType(roomtype_id)
);

-- Record the guest information separately, could possibly be used for
future promotional emails to guests
CREATE TABLE Guest (
  guest_id INT PRIMARY KEY,
  guest_name VARCHAR(50) NOT NULL,
  guest_contact_info VARCHAR(50) NOT NULL -- Contact info can be listed as
email and phone number
);

-- Defines booking register, all booking transactions are recorded in this
table
CREATE TABLE Booking (
  booking_id INT PRIMARY KEY,
  guest_id INT NOT NULL,
  room_id INT ,
  checkin_date DATE,
  checkout_date DATE,
  requests VARCHAR(100),
  FOREIGN KEY (guest_id) REFERENCES Guest(guest_id),
  FOREIGN KEY (room_id) REFERENCES Room(room_id)
);

-- Defines department which connect employees; such as Reception,
Room-service, management, kitchen etc
CREATE TABLE Department (
  dept_ID INT PRIMARY KEY,
  dept_name VARCHAR(50)
);

-- Defines employee information and their contact info along with their
superior
CREATE TABLE Employee (
  employee_id INT PRIMARY KEY,
  dept_id INT,
  employee_name VARCHAR(50),
  job_title VARCHAR(50),
  contact VARCHAR(20),
  hotel_id INT,
  FOREIGN KEY (dept_id) REFERENCES Department(dept_id),
  FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id)
);

-- This table provides the service type which can be room service, bar
service, restauraunt service, long distance call etc
CREATE TABLE Service_Type (
      service_type_id INT primary key,
    service_description varchar(100)
);

CREATE TABLE Service (
  booking_id INT,
  service_type_id INT,
  service_date date,
  amount DECIMAL(10,2),
  FOREIGN KEY (booking_id) REFERENCES Booking(booking_id),
  FOREIGN KEY (service_type_id) REFERENCES Service_Type(service_type_id)
);

INSERT INTO Hotel VALUES (123, "Embassy" , "Toronto", 4166786545);
SELECT * FROM Hotel;


INSERT INTO Hotel (hotel_id, hotel_name, location, contact_info)
VALUES
(100, 'Chateau Lake Louise', 'Alberta', '555-555-1212'),
(101, 'The Fairmont Banff Springs', 'Alberta', '555-555-1213'),
(102, 'The Fairmont Chateau Whistler', 'British Columbia', '555-555-1214'),
(103, 'The Fairmont Jasper Park Lodge', 'Alberta', '555-555-1215');

INSERT INTO Guest (guest_id, guest_name, guest_contact_info)
VALUES
(100, 'John Doe', 'johndoe@email.com'),
(101, 'Jane Doe', 'janedoe@email.com'),
(102, 'John Smith', 'johnsmith@email.com'),
(103, 'Jane Smith', 'janesmith@email.com');

INSERT INTO RoomType (roomtype_id, room_price)
VALUES
(1, 100),
(2, 200),
(3, 300),
(4, 400);
```

```sql
INSERT INTO Room (room_id, hotel_id, roomtype_id)
VALUES
(101, 100, 1),
(102, 101, 2),
(103, 102, 3),
(104, 103, 4);

INSERT INTO Department (dept_id, dept_name)
VALUES
(100, 'Reception'),
(101, 'Room-service'),
(102, 'Management'),
(103, 'Kitchen');

INSERT INTO Employee (employee_id, dept_id, employee_name, job_title,
contact, hotel_id)
VALUES
(100, 100, 'Jane Doe', 'Receptionist', '555-555-1216', 100),
(101, 101, 'John Smith', 'Room-service', '555-555-1217', 100),
(102, 102, 'Jane Smith', 'Manager', '555-555-1218', 100),
(103, 103, 'John Doe', 'Chef', '666-666-3434',100);

-- Define Service Type
INSERT INTO Service_Type (service_type_id, service_description)
VALUES (1, 'Room Service'), (2, 'Bar Service'), (3, 'Restaurant Service'),
(4, 'Long Distance Call');

INSERT INTO Booking (booking_id, guest_id, room_id, checkin_date,
checkout_date, requests)
VALUES
(100, 100, 101, '2022-01-01', '2022-01-02', 'Early Check-In'),
(101, 100, 101, '2022-01-03', '2022-01-04', 'Late Check-Out'),
(102, 101, 102, '2022-01-05', '2022-01-06', 'Extra Beds'),
(103, 101, 103, '2022-01-07', '2022-01-08', 'Special Requests');

INSERT INTO Service (booking_id, service_type_id, service_date, amount)
VALUES (100, 1, '2022-12-23', 10.50),
       (101, 2, '2022-12-24', 15.75),
       (102, 3, '2022-12-25', 20.50),
       (103, 4, '2022-12-26', 5.25);

       --
```

```
SELECT * FROM Employee;
SELECT * FROM Department;
SELECT * FROM Hotel;
SELECT * FROM Guest;
SELECT * FROM Booking;
SELECT * FROM Service;
SELECT * FROM Room;
SELECT * FROM roomtype;
SELECT * FROM service_type;

-- 10 simple queries for Assignment 4

SELECT DISTINCT * FROM Hotel WHERE hotel_id = 123;--
SELECT * FROM Hotel WHERE location = "Toronto" ORDER BY hotel_id DESC;
SELECT * FROM RoomType WHERE roomtype_id = 1 GROUP BY roomtype_id;--
SELECT * FROM Room WHERE hotel_id = 100 ORDER BY room_id ASC;
SELECT * FROM Guest WHERE guest_name = "John Doe" ORDER BY guest_name DESC;
SELECT * FROM Booking WHERE booking_id = 100 ORDER BY checkin_date ASC;
SELECT * FROM Department WHERE dept_id = 100 ORDER BY dept_id ASC;
SELECT DISTINCT employee_id, employee_name, dept_id FROM Employee ORDER BY
dept_id, employee_name;
SELECT service_type_id, service_description FROM Service_Type ORDER BY
service_type_id;
SELECT DISTINCT guest_id, guest_name FROM Guest ORDER BY guest_name ASC;--

--  CREATE VIEW regen_employee AS
--  SELECT employee_id, employee_name, dept_id FROM Employee e;
--
--  SELECT * FROM regen_employee;



CREATE VIEW booking_details_by_guest AS
SELECT g.guest_name, g.guest_contact_info, b.booking_id, r.room_id,
b.checkin_date, b.checkout_date, b.requests
FROM Guest g
JOIN Booking b ON g.guest_id = b.guest_id
JOIN Room r ON b.room_id = r.room_id;

SELECT * FROM booking_details_by_guest;

CREATE VIEW employee_list AS
```

```sql
SELECT e.employee_id, e.employee_name, e.job_title, d.dept_name,
h.hotel_name, h.location
FROM Employee e
JOIN Department d ON e.dept_id = d.dept_ID
JOIN Hotel h ON e.hotel_id = h.hotel_id;

SELECT * FROM employee_list;

CREATE VIEW revenue_by_room_type AS
SELECT rt.roomtype_id, rt.room_price, SUM(rt.room_price) AS total_revenue
FROM RoomType rt
JOIN Room r ON rt.roomtype_id = r.roomtype_id
JOIN Booking b ON r.room_id = b.room_id
GROUP BY rt.roomtype_id, rt.room_price;

Select * FROM revenue_by_room_type;

CREATE VIEW service_summary_view AS
SELECT g.guest_name, st.service_description, SUM(s.amount) AS total_amount
FROM Booking b
JOIN Guest g ON b.guest_id = g.guest_id
JOIN Service s ON b.booking_id = s.booking_id
JOIN Service_Type st ON s.service_type_id = st.service_type_id
GROUP BY g.guest_name, st.service_description;

Select * FROM service_summary_view;
```

#5 - Database Management System

Hotel Management Database System - Python UI Implementation

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

This is the Python program we developed for a menu-driven Hotel Management Database. The main() function displays a menu with four options (Drop Tables, Create Tables, Populate Tables, and Query Tables) and an Exit option. The user selects an option by entering a number or the letter "E". Depending on the user's choice, the program calls the corresponding function to drop tables, create tables, populate tables, or query tables. The program continues to display the menu until the user chooses to exit by entering the letter "E". The implementation of the UI is listed below.

```python
import mysql.connector

# Function to connect to MySQL database
def connect():
    connection = mysql.connector.connect(
        host="localhost",
        user="sqluser",
        password="password",
        database="hotelmanagement"
    )
    return connection

# Function to drop tables
def drop_all_tables():
    connection = connect()
    cursor = connection.cursor()
    cursor.execute("DROP TABLE IF EXISTS employee, department, hotel, guest, 
booking, service, room, roomtype, service_type")
    connection.commit()
    cursor.close()
    connection.close()
    print("All tables dropped successfully")

# Function to create tables
def create_tables():
    connection = connect()
    cursor = connection.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Hotel (
            hotel_id INT PRIMARY KEY,
            hotel_name VARCHAR(50) NOT NULL,
            location VARCHAR(100) NOT NULL,
            contact_info VARCHAR(20) NOT NULL
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS RoomType (
            roomtype_id INT PRIMARY KEY,
            room_price DECIMAL(10,2)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Room (
            room_id INT PRIMARY KEY,
            hotel_id INT NOT NULL,
            roomtype_id INT,
            FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id),
            FOREIGN KEY (roomtype_id) REFERENCES RoomType(roomtype_id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Guest (
            guest_id INT PRIMARY KEY,
            guest_name VARCHAR(50) NOT NULL,
            guest_contact_info VARCHAR(50) NOT NULL
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Booking (
            booking_id INT PRIMARY KEY,
            guest_id INT NOT NULL,
            room_id INT ,
            checkin_date DATE,
            checkout_date DATE,
            requests VARCHAR(100),
            FOREIGN KEY (guest_id) REFERENCES Guest(guest_id),
            FOREIGN KEY (room_id) REFERENCES Room(room_id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Department (
            dept_ID INT PRIMARY KEY,
            dept_name VARCHAR(50)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Employee (
            employee_id INT PRIMARY KEY,
            dept_id INT,
            employee_name VARCHAR(50),
            job_title VARCHAR(50),
            contact VARCHAR(20),
            hotel_id INT,
            FOREIGN KEY (dept_id) REFERENCES Department(dept_id),
            FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Service_Type (
            service_type_id INT PRIMARY KEY,
            service_description VARCHAR(100)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Service (
            booking_id INT,
            service_type_id INT,
            service_date DATE,
            amount DECIMAL(10,2),
            FOREIGN KEY (booking_id) REFERENCES Booking(booking_id),
            FOREIGN KEY (service_type_id) REFERENCES
Service_Type(service_type_id)
        )
    """)
    connection.commit()
    cursor.close()
    connection.close()
    print("Tables created successfully")

# Function to populate tables
def populate_tables():
    connection = connect()
    cursor = connection.cursor()

    cursor.execute("INSERT INTO Hotel VALUES (123, 'Embassy', 'Toronto',
4166786545)")

    cursor.execute("""
        INSERT INTO Hotel (hotel_id, hotel_name, location, contact_info)
        VALUES
        (100, 'Chateau Lake Louise', 'Alberta', '555-555-1212'),
        (101, 'The Fairmont Banff Springs', 'Alberta', '555-555-1213'),
        (102, 'The Fairmont Chateau Whistler', 'British Columbia',
'555-555-1214'),
        (103, 'The Fairmont Jasper Park Lodge', 'Alberta', '555-555-1215')
    """)

    cursor.execute("""
        INSERT INTO Guest (guest_id, guest_name, guest_contact_info)
        VALUES
        (100, 'John Doe', 'johndoe@email.com'),
        (101, 'Jane Doe', 'janedoe@email.com'),
        (102, 'John Smith', 'johnsmith@email.com'),
        (103, 'Jane Smith', 'janesmith@email.com')
    """)

    cursor.execute("""
        INSERT INTO RoomType (roomtype_id, room_price)
        VALUES
        (1, 100),
        (2, 200),
        (3, 300),
        (4, 400)
    """)

    cursor.execute("""
        INSERT INTO Room (room_id, hotel_id, roomtype_id)
        VALUES
        (101, 100, 1),
        (102, 101, 2),
        (103, 102, 3),
        (104, 103, 4)
    """)

    cursor.execute("""
        INSERT INTO Department (dept_id, dept_name)
        VALUES
        (100, 'Reception'),
        (101, 'Room-service'),
        (102, 'Management'),
        (103, 'Kitchen')
    """)

    cursor.execute("""
        INSERT INTO Employee (employee_id, dept_id, employee_name, job_title,
contact, hotel_id)
        VALUES
        (100, 100, 'Jane Doe', 'Receptionist', '555-555-1216', 100),
        (101, 101, 'John Smith', 'Room-service', '555-555-1217', 100),
        (102, 102, 'Jane Smith', 'Manager', '555-555-1218', 100),
        (103, 103, 'John Doe', 'Chef', '666-666-3434', 100)
    """)

    cursor.execute("""
        -- Define Service Type
        INSERT INTO Service_Type (service_type_id, service_description)
        VALUES (1, 'Room Service'), (2, 'Bar Service'), (3, 'Restaurant
Service'), (4, 'Long Distance Call')
    """)

        # Insert Booking data
    cursor.execute("INSERT INTO Booking (booking_id, guest_id, room_id,
checkin_date, checkout_date, requests) VALUES (100, 100, 101, '2022-01-01',
'2022-01-02', 'Early Check-In'), (101, 100, 101, '2022-01-03', '2022-01-04',
'Late Check-Out'), (102, 101, 102, '2022-01-05', '2022-01-06', 'Extra Beds'),
(103, 101, 103, '2022-01-07', '2022-01-08', 'Special Requests')")

    # Insert Service data
    cursor.execute("INSERT INTO Service (booking_id, service_type_id,
service_date, amount) VALUES (100, 1, '2022-12-23', 10.50), (101, 2,
'2022-12-24', 15.75), (102, 3, '2022-12-25', 20.50), (103, 4, '2022-12-26',
5.25)")

    connection.commit()
    cursor.close()
    connection.close()
    print("Tables populated successfully")

def query_tables():
    connection = connect()
    cursor = connection.cursor()

    # Query 1: Using EXISTS to check for guest with bookings
    cursor.execute("SELECT guest_name FROM Guest g WHERE EXISTS (SELECT * FROM
Booking b WHERE b.guest_id = g.guest_id)")
    print("Query 1: Guests with Bookings")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 2: Using UNION to combine guest names and department names
    cursor.execute("SELECT guest_name FROM Guest UNION SELECT room_id FROM
room")
    print("Query 2: Guest and Rooms")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 3: Using ORDER_BY to find hotels in Toronto
    cursor.execute("SELECT * FROM Hotel WHERE location = 'Toronto' ORDER BY
hotel_id DESC")
    print("Query 3: Available Locations in Toronto")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 4: Using COUNT to find the number of bookings for each guest
    cursor.execute("SELECT guest_name, COUNT(*) FROM Booking b JOIN Guest g ON
b.guest_id = g.guest_id GROUP BY guest_name")
    print("Query 4: Number of bookings for each guest")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 5: Using GROUP BY and HAVING to find guests with multiple bookings
    cursor.execute("SELECT guest_name, COUNT(*) FROM Booking b JOIN Guest g ON
b.guest_id = g.guest_id GROUP BY guest_name HAVING COUNT(*) > 1")
    print("Query 5:  Guests with multiple bookings")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 6: Using GROUP BY and HAVING to find rooms with multiple bookings
    cursor.execute("SELECT room_id, COUNT(*) FROM Booking GROUP BY room_id
HAVING COUNT(*) > 1")
    print("Query 6: Rooms with Multiple Bookings")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 7: Using GROUP BY and HAVING to find hotels with a high occupancy
rate
    cursor.execute("SELECT r.hotel_id, h.hotel_name, COUNT(*)*100.0 / (SELECT
COUNT(*) FROM Room WHERE hotel_id = r.hotel_id) as occupancy_rate FROM Booking
b JOIN Room r ON b.room_id = r.room_id JOIN Hotel h ON r.hotel_id = h.hotel_id
GROUP BY r.hotel_id, h.hotel_name HAVING occupancy_rate > 80")
    print("Query 7: Hotels with High Occupancy Rates")
    for row in cursor.fetchall():
        print(row)
    print("")

    # Query 8: Using UNION to combine guest names and service descriptions
    cursor.execute("SELECT employee_name FROM Employee UNION SELECT
service_type_id FROM Service")
    print("Query 8: Employee and Service Descriptions")
    for row in cursor.fetchall():
        print(row)
    print("")

# Query 9: Using MINUS to find rooms that have not been booked
    cursor.execute("SELECT room_id FROM room WHERE room_id NOT IN (SELECT
room_id FROM booking)")
    print("Query 9: Available Rooms")
    for row in cursor.fetchall():
        print(row)
    print("")

    cursor.close()
    connection.close()



# Main function for menu
def main():
    while True:

print("=================================================================")
        print("| Main Menu - Hotel Management Database:
|")
        print("| Select desired option:             |")

print("-----------------------------------------------------------------")
        print("1) Drop Tables")
        print("2) Create Tables")
        print("3) Populate Tables")
        print("4) Query Tables")
        print("E) End/Exit")
        choice = input("Choose: ")
        if choice == "1":
            drop_all_tables()
        elif choice == "2":
            create_tables()
        elif choice == "3":
            populate_tables()
        elif choice == "4":
            query_tables()
        elif choice == "E":
            break

main()
```

#6 - Database Management System

Hotel Management Database System - Functional Dependencies using Set Notation

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/05/2023

Function Dependencies using Set notation

Functional Dependencies for Hotel table:

{hotel_id} -> {hotel_name, location, contact_info}

In the Hotel table, the hotel_id is the primary key and uniquely identifies each row in the table. The hotel_name and location are dependent on the hotel_id, as each hotel has a unique name and location. The contact_info is also dependent on the hotel_id, as each hotel has a unique contact information.

Functional Dependencies for RoomType table:

{roomtype_id} -> {room_type, room_price}

In the RoomType table, the roomtype_id is the primary key and uniquely identifies each row in the table. The room_type and room_price are dependent on the roomtype_id, as each room type has a unique type and price.

Functional Dependencies for Room table:

{room_id} -> {hotel_id, roomtype_id}

In the Room table, the room_id is the primary key and uniquely identifies each row in the table. The hotel_id and roomtype_id together uniquely identify each room. The hotel_id is dependent on the room_id, as each room belongs to a specific hotel. The roomtype_id is also dependent on the room_id, as each room has a specific type.

Functional Dependencies for Guest table:

{guest_id} -> {guest_name, guest_email, guest_phone_number}

In the Guest table, the guest_id is the primary key and uniquely identifies each row in the table. The guest_name, guest_email, and guest_phone_number are dependent on the guest_id, as each guest has a unique name, email, and phone number.

Functional Dependencies for Booking table:

{booking_id} -> {guest_id, room_id, checkin_date, checkout_date, requests}

{guest_id} -> {booking_id} {room_id}

-> {booking_id}

In the Booking table, the booking_id is the primary key and uniquely identifies each row in the table. The guest_id and room_id together uniquely identify each booking. The checkin_date, checkout_date, and requests are dependent on the booking_id, as each booking has a specific check-in and check-out date, as well as any specific requests made by the guest.

Functional Dependencies for Department table:

{dept_ID} -> {dept_name}

In the Department table, the dept_ID is the primary key and uniquely identifies each row in the table. The dept_name is dependent on the dept_ID, as each department has a unique name.

Functional Dependencies for Employee table:

{employee_id} -> {dept_id, employee_name, job_title, contact, hotel_id}

{dept_id} -> {employee_id, hotel_id}

In the Employee table, the employee_id is the primary key and uniquely identifies each row in the table. The dept_id and hotel_id together uniquely identify each employee. The employee_name, job_title, and contact are dependent on the employee_id, as each employee has a unique name, job title, and contact information.

Functional Dependencies for Service_Type table:

{service_type_id} -> {service_description}

In the Service_Type table, the service_type_id is the primary key and uniquely identifies each row in the table. The service_description is dependent on the service_type_id, as each service has a unique description.

#7 - Database Management System

Hotel Management Database System - Decomposition of 2NF and 3NF

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

\

<u>Creating Transitive and Partial Dependencies:</u>

Table: Booking

Booking ID	Guest ID	Room Number	Guest Name	Check-in Date	Check-out Date	service_description	amount _paid
1	101	101	John Doe	2023-04-01	2023-04-05	Extra Blankets	$400
2	102	102	Jane Smith	2023-04-02	2023-04-06	Extra Water	$480
3	103	101	Bob Johnson	2023-04-01	2023-04-04	Extra Shampoo	$300

Here are the functional dependencies of the Booking table:

Booking ID, room_id -> Guest ID, Check-in Date, Check-out Date, amount_paid, guest_name, service_description.

guest ID -> guest Name (This is an example of a transitive dependency, as guest_id can determine guest_name although they are both non-key attributes)

room_id-> service_description (This is an example of a partial dependency, as room_id can determine services, as room_id is a part of the compound key, making this a partial dependency as it doesnt need booking_id to determine services)

Decompose Table to 2NF

Table: Booking

Booking ID	Guest ID	Room_id	Guest Name	Check-in Date	Check-out Date	amount _paid
1	101	101	John Doe	2023-04-01	2023-04-05	$400
2	102	102	Jane Smith	2023-04-02	2023-04-06	$480
3	103	101	Bob Johnson	2023-04-01	2023-04-04	$300

Table: Service_Type

Service_Type_id	Service_description	Room_id
101	Extra Blankets	101
102	Extra Water	102
103	Extra Shampoo	101

This decomposition removes the partial dependency between room_id and service_description by separating them into two tables and linking them through the room_id column. Now, service_description is fully dependent on room_id, and the data is stored in a more efficient and normalized form.

Decompose Table to 3NF:

Table: Booking

Reservation ID	Guest ID	Room Number	Check-in Date	Check-out Date	amount_paid
1	101	101	2023-04-01	2023-04-05	$400
2	102	102	2023-04-02	2023-04-06	$480
3	103	101	2023-04-01	2023-04-04	$300

Table: Service_Type

Service_Type_id	Service_description	Room_id
101	Extra Blankets	101
102	Extra Water	102
103	Extra Shampoo	101

Table: Guest

Guest ID	Guest Name	Guest Email	Phone Number
101	John Doe	johndoe@gmail.com	123-456-7890
102	Jane Smith	janesmith@gmail.com	123-456-7890
103	Bob Johnson	bjohnson@yahoo.com	123-456-7890

This further decomposition removes the transitive dependency between the two non keys; guest_id and guest_name, by separating them into two tables and linking them through the guest_id column. Now, guest_name is fully dependent on guest_id, and the data is stored in a more efficient and normalized form.

#8 - Database Management System

Hotel Management Database System - Bernstein’s Algorithm and BCNF

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/19/2023

\

Creating Transitive and Partial Dependencies:

Table 1: Booking (adding data in the modified tables)

Booking ID	Guest ID	Room Number	Guest Name	Check-in Date	Check-out Date	service_description	amount _paid
1	101	101	John Doe	2023-04-01	2023-04-05	Extra Blankets	$400
2	102	102	Jane Smith	2023-04-02	2023-04-06	Extra Water	$480
3	103	101	Bob Johnson	2023-04-01	2023-04-04	Extra Shampoo	$300

Here are the functional dependencies of the Booking table:

Booking ID, room_id -> Guest ID, Check-in Date, Check-out Date, amount_paid, guest_name, service_description.

guest ID -> guest Name (This is an example of a transitive dependency, as guest_id can determine guest_name although they are both non-key attributes)

room_id-> service_description (This is an example of a partial dependency, as room_id can determine services, as room_id is a part of the compound key, making this a partial dependency as it doesnt need booking_id to determine services)

Displaying All FD's

Hotel: {hotel_id} -> {hotel_name, location}
HotelContact: {hotel_id} -> {hotel_phone_number}
RoomType: {roomtype_id} -> {room_type}
RoomPrice: {roomtype_id, hotel_id} -> {room_price}
Room: {room_id} -> {hotel_id, roomtype_id, requests}
Guest: {guest_id} -> {guest_name, guest_email, guest_phone_number}

Booking:
{Booking ID, room_id} -> **{**Guest ID, Check-in Date, Check-out Date, amount_paid, guest_name, service_description}
[{guest ID} -> {guest Name}
{room_id} -> **{**service_description}

Department: {dept_ID} -> {dept_name}
Employee: {employee_id} -> {dept_id, employee_name, job_title, contact, hotel_id}
Service_Type: {service_type_id} -> {service_description}

Algorithms

To normalize the Booking table, we can use Bernstein's algorithm to decompose it into smaller tables that are in 3NF.

First, we list all the functional dependencies:

booking_id, room_id -> guest_id, checkin_date, checkout_date, amount_paid, guest_name, service_description
guest_id -> guest_name
room_id -> service_description
Next, we use these functional dependencies to create a set of tables in 3NF:

Table 2: Guest

guest_id (primary key)
Guest_name

Table 3: Room

room_id (primary key)
hotel_id
roomtype_id
Requests
Service_description

Table 4: Booking

booking_id (primary key)
guest_id (foreign key references Guest(guest_id))
room_id (foreign key references Room(room_id))
checkin_date
checkout_date
amount_paid
The first two tables were created based on the transitive and partial dependencies respectively, and the third table was created to capture the remaining attributes of the original Booking table.

This set of tables is now in 3NF since there are no partial or transitive dependencies in any of the tables.

Steps:

1. List all the functional dependencies for the original table.
2. Use the functional dependencies to create a set of tables in 3NF that capture all the attributes of the original table.
3. Verify that there are no partial or transitive dependencies in any of the new tables.
4. Add foreign key constraints to maintain referential integrity between the tables.

3NF verification:

Here's an explanation of how each functional dependency in the set is in 3NF or BCNF:

Hotel: {hotel_id} -> {hotel_name, location}
This functional dependency is already in 3NF as each non-key attribute (hotel_name, location) is dependent only on the primary key (hotel_id).

HotelContact: {hotel_id} -> {hotel_phone_number}
This functional dependency is already in 3NF as each non-key attribute (hotel_phone_number) is dependent only on the primary key (hotel_id).

RoomType: {roomtype_id} -> {room_type}
This functional dependency is already in 3NF as each non-key attribute (room_type) is dependent only on the primary key (roomtype_id).

RoomPrice: {roomtype_id, hotel_id} -> {room_price}
This functional dependency is already in 3NF as each non-key attribute (room_price) is dependent only on the combination of primary keys (roomtype_id, hotel_id).

Room: {room_id} -> {hotel_id, roomtype_id, requests}
This functional dependency is already in 3NF as each non-key attribute (hotel_id, roomtype_id, requests) is dependent only on the primary key (room_id).

Guest: {guest_id} -> {guest_name, guest_email, guest_phone_number}
This functional dependency is already in 3NF as each non-key attribute (guest_name, guest_email, guest_phone_number) is dependent only on the primary key (guest_id).

Booking: {booking_id} -> {guest_id, room_id, checkin_date, checkout_date, amount_paid}
This functional dependency is already in 3NF as each non-key attribute (guest_id, room_id, checkin_date, checkout_date, amount_paid) is dependent only on the primary key (booking_id).

Department: {dept_ID} -> {dept_name}
This functional dependency is already in 3NF as each non-key attribute (dept_name) is dependent only on the primary key (dept_ID).

Employee: {employee_id} -> {dept_id, employee_name, job_title, contact, hotel_id}
This functional dependency is already in 3NF as each non-key attribute (dept_id, employee_name, job_title, contact, hotel_id) is dependent only on the primary key (employee_id).

Service_Type: {service_type_id} -> {service_description}
This functional dependency is already in 3NF as each non-key attribute (service_description) is dependent only on the primary key (service_type_id).

#9 - Database Management System

Hotel Management System - Graphical User Interfaces

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/19/2023

Hotel Management Database GUI

This GUI program allows users to manage a hotel management database. It provides the following functionalities:

- Create tables in the database
- Populate tables with sample data
- Query all tables in the database
- Query a specific table by its name
- Delete a record from a table by its primary ID
- Search for a record in a table by its primary ID

Required Libraries

The following libraries are required to run this program:

- **tkinter**
- **scrolledtext**
- **GUI_Functions (a module created by Group 40)**

Usage

1. Install the required libraries.
2. Run the databaseGUI.py file.
3. Click on the "Create Tables" button to create the necessary tables in the database.
4. Click on the "Populate Tables" button to insert sample data into the tables.
5. Select a table from the listbox to view its contents.
6. To search for a specific record, enter its primary ID in the search bar and click the "Search Record" button.
7. To delete a record, select its table from the listbox, enter its primary ID in the search bar, and click the "Delete Record" button.

Functions

The following functions are available in this program:

- drop_tables(): Drops all tables from the database.
- create_tables(): Creates the necessary tables in the database.
- populate_tables(): Populates the tables with sample data.

- query_all_tables(): Queries all tables in the database.
- query_table(table_name): Queries a specific table by its name.
- display_tables(*tables): Displays the contents of one or more tables in the text box.
- delete_record(table_name, record_id): Deletes a record from a table by its primary ID.
- search_record(table_name, record_id): Searches for a record in a table by its primary ID.

GUI Design

The GUI design is created using the tkinter library. It includes the following elements:

- Main window with a custom title and background color
- Custom font and color scheme
- Buttons to perform database operations
- Label to display status messages
- Search bar to search for and delete records
- Text box to display query results
- Listbox to display table names

Authors

This program was created by Group 40 for the CP363 course in 2023-03-26.

The GUI implementation is shown below:

```
"""
-----------------------------------------------
File: GUI_Functions.py
Authors: Group 40
Course: CP363
Date: 2023-03-26
-----------------------------------------------
"""


import mysql.connector

# Function to connect to MySQL database
def connect():
    connection = mysql.connector.connect(
        host="localhost",
        user="sqluser",
        password="password",
        database="hotelmanagement"
    )
    return connection

# Function to drop tables
def drop_all_tables():
    connection = connect()
    cursor = connection.cursor()
    cursor.execute("""
    DROP TABLE IF EXISTS
    Employee, Department, Hotel, HotelContact,
    Guest, Booking, Room, RoomType,
    RoomPrice, Service_type""")
    connection.commit()
    cursor.close()
    connection.close()
    print("All tables dropped successfully")

# Function to create tables
def create_tables():
    connection = connect()
    cursor = connection.cursor()
```

```
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Hotel (
        hotel_id INT PRIMARY KEY,
        hotel_name VARCHAR(50) NOT NULL,
        location VARCHAR(100) NOT NULL
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS HotelContact (
        hotel_id INT PRIMARY KEY,
        hotel_phone_number VARCHAR(20) NOT NULL,
        hotel_email VARCHAR(20) NOT NULL,
        FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS RoomType (
        roomtype_id INT PRIMARY KEY,
        room_type VARCHAR(50) NOT NULL
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS RoomPrice (
        roomtype_id INT PRIMARY KEY,
        hotel_id INT,
        room_price DECIMAL(10,2) NOT NULL,
        FOREIGN KEY (roomtype_id) REFERENCES RoomType(roomtype_id),
        FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Room (
        room_id INT PRIMARY KEY,
        hotel_id INT NOT NULL,
        roomtype_id INT,
        requests VARCHAR(100),
        FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id),
        FOREIGN KEY (roomtype_id) REFERENCES RoomType(roomtype_id)
    )
""")
cursor.execute("""
    CREATE TABLE IF NOT EXISTS Guest (
        guest_id INT PRIMARY KEY,
            guest_name VARCHAR(50) NOT NULL,
            guest_email VARCHAR(50) NOT NULL,
            guest_phone_number VARCHAR(50) NOT NULL
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Booking (
            booking_id INT NOT NULL,
            guest_id INT NOT NULL,
            room_id INT NOT NULL,
            guest_name VARCHAR(100),
            amount_paid DECIMAL(10,2),
            checkin_date DATE,
            checkout_date DATE,
            service_description VARCHAR(100),
            PRIMARY KEY (booking_id, room_id),
            FOREIGN KEY (guest_id) REFERENCES Guest(guest_id),
            FOREIGN KEY (room_id) REFERENCES Room(room_id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Department (
            department_ID INT PRIMARY KEY,
            dept_name VARCHAR(50)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Employee (
            employee_id INT PRIMARY KEY,
            department_id INT,
            employee_name VARCHAR(50),
            job_title VARCHAR(50),
            contact VARCHAR(20),
            hotel_id INT,
            FOREIGN KEY (department_id) REFERENCES
Department(department_id),
            FOREIGN KEY (hotel_id) REFERENCES Hotel(hotel_id)
        )
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS Service_Type (
            service_type_id INT PRIMARY KEY,
            service_description VARCHAR(100),
            amount DECIMAL(10,2),
            service_date DATE,
            room_id INT,
            FOREIGN KEY (room_id) REFERENCES Room(room_id)
        )
    """)

    connection.commit()
    cursor.close()
    connection.close()
    print("Tables created successfully")

# Function to populate tables
def populate_tables():
    connection = connect()
    cursor = connection.cursor()

    cursor.execute("INSERT INTO Hotel VALUES (123, 'Embassy', 'Toronto')")

    cursor.execute("""
        INSERT INTO Hotel (hotel_id, hotel_name, location)
        VALUES
        (100, 'Chateau Lake Louise', 'Alberta'),
        (101, 'The Fairmont Banff Springs', 'Alberta'),
        (102, 'The Fairmont Chateau Whistler', 'British Columbia'),
        (103, 'The Fairmont Jasper Park Lodge', 'Alberta')
    """)
    cursor.execute("""
        INSERT INTO HotelContact (hotel_id, hotel_phone_number,
hotel_email)
        VALUES
        (100, '123-456-7890', 'hotel1@gmail.com'),
        (101, '987-654-3210', 'hotel2@gmail.com'),
        (102, '555-123-4567', 'hotel3@gmail.com'),
        (103, '444-555-6666', 'hotel4@gmail.com')
    """)

    cursor.execute("""
        INSERT INTO Guest (guest_id, guest_name, guest_email,
guest_phone_number)
        VALUES
        (100, 'John Doe', 'johndoe@email.com', '123-456-7890'),
        (101, 'Jane Doe', 'janedoe@email.com', '123-456-7890'),
        (102, 'John Smith', 'johnsmith@email.com', '123-456-7890'),
        (103, 'Jane Smith', 'janesmith@email.com', '123-456-7890')
    """)

    cursor.execute("""
        INSERT INTO RoomType (roomtype_id, room_type)
        VALUES
        (1, 'Single'),
        (2, 'Double'),
        (3, 'Suite'),
        (4, 'Single')
    """)

    cursor.execute("""
        INSERT INTO RoomPrice (roomtype_id, hotel_id, room_price)
        VALUES
        (1, 100, 80.00),
        (2, 101, 90.00),
        (3, 102, 100.00),
        (4, 103, 80.00)
    """)

    cursor.execute("""
        INSERT INTO Room (room_id, hotel_id, roomtype_id, requests)
        VALUES
        (101, 100, 1,'Room Service' ),
        (102, 101, 2, 'Bar Service'),
        (103, 102, 3, 'Restaurant Service'),
        (104, 103, 4, 'Long Distance Call')
    """)

    cursor.execute("""
        INSERT INTO Department (department_id, dept_name)
        VALUES
        (100, 'Reception'),
        (101, 'Room-service'),
        (102, 'Management'),
        (103, 'Kitchen')
    """)

    cursor.execute("""
        INSERT INTO Employee (employee_id, department_id, employee_name,
job_title, contact, hotel_id)
```

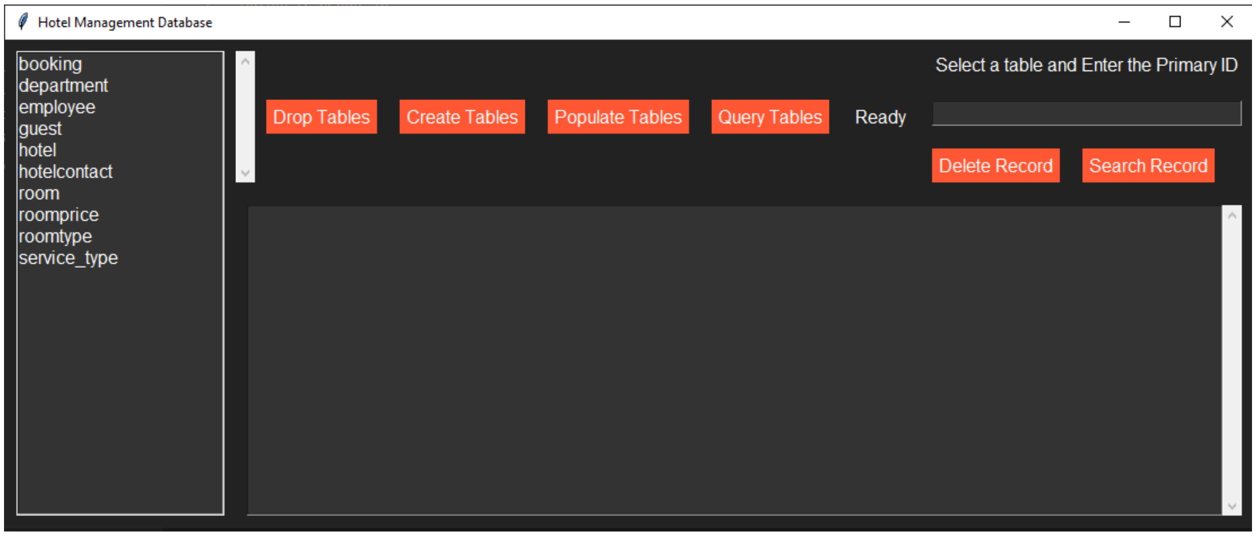
```
        VALUES
        (100, 100, 'Jane Doe', 'Receptionist', '555-555-1216', 100),
        (101, 101, 'John Smith', 'Room-service', '555-555-1217', 100),
        (102, 102, 'Jane Smith', 'Manager', '555-555-1218', 100),
        (103, 103, 'John Doe', 'Chef', '666-666-3434', 100)
    """)

    cursor.execute("""
        INSERT INTO Service_Type (service_type_id, service_description,
amount, service_date) VALUES
        (1, 'Room Service', 50.00,'2023-03-12'),
        (2, 'Bar Service', 80.00,'2023-03-25'),
        (3, 'Restaurant Service', 60.00,'2023-06-15'),
        (4, 'Long Distance Call', 10.00,'2023-02-12')
    """)

        # Insert Booking data
    cursor.execute("""
    INSERT INTO Booking (booking_id, guest_id, room_id, guest_name,
amount_paid, checkin_date, checkout_date, service_description) VALUES
    (100, 100, 101, 'John Doe', 100.00, '2022-01-01', '2022-01-02', 'Room
Service'),
    (101, 100, 101, 'Jane Doe', 90.00, '2022-01-03', '2022-01-04', 'Bar
Service'),
    (102, 101, 102, 'John Smith', 80.00,'2022-01-05', '2022-01-06',
'Restaurant Service'),
    (103, 101, 103, 'Jane Smith', 80.00, '2022-01-07', '2022-01-08', 'Long
Distance Call')
    """)



    connection.commit()
    cursor.close()
    connection.close()
    print("Tables populated successfully")

def query_all_tables():
    connection = connect()
    cursor = connection.cursor()
    results = []
    # Query 1: Using EXISTS to check for guest with bookings
    cursor.execute("SELECT * FROM hotel")
    print("Query 1: Guests with Bookings")
    results.append("Hotel")
    results.append(cursor.fetchall())



    # Query 2: Using UNION to combine guest names and department names
    cursor.execute("SELECT * FROM hotelcontact")
    print("Query 2: Guest and Rooms")
    results.append("Hotel_Contact")
    results.append(cursor.fetchall())

    # Query 3: Using ORDER_BY to find hotels in Toronto
    cursor.execute("SELECT * FROM room")
    print("Query 3: Available Locations in Toronto")
    results.append("Room")
    results.append(cursor.fetchall())

    # Query 4: Using COUNT to find the number of bookings for each guest
    cursor.execute("SELECT * FROM roomtype")
    print("Query 4: Number of bookings for each guest")
    results.append("RoomType")
    results.append(cursor.fetchall())

    # Query 5: Using GROUP BY and HAVING to find guests with multiple
bookings
    cursor.execute("SELECT * FROM roomprice")
    print("Query 5:  Guests with multiple bookings")
    results.append("RoomPrice")
    results.append(cursor.fetchall())

    # Query 6: Using GROUP BY and HAVING to find rooms with multiple
bookings
    cursor.execute("SELECT * FROM guest")
    print("Query 6: Rooms with Multiple Bookings")
    results.append("Guest")
    results.append(cursor.fetchall())

    # Query 7: Using GROUP BY and HAVING to find hotels with a high
occupancy rate
    cursor.execute("SELECT * FROM department")
    print("Query 7: Hotels with High Occupancy Rates")
    results.append("Department")
    results.append(cursor.fetchall())

    # Query 8: Using UNION to combine guest names and service descriptions
    cursor.execute("SELECT * FROM employee")
    print("Query 8: Employee and Service Descriptions")
    results.append("Employee")
    results.append(cursor.fetchall())

# Query 9: Using MINUS to find rooms that have not been booked
    cursor.execute("SELECT * FROM booking")
    print("Query 9: Available Rooms")
    results.append("Booking")
    results.append(cursor.fetchall())

# Query 10: Using MINUS to find rooms that have not been booked
    cursor.execute("SELECT * FROM service_type")
    print("Query 9: Available Rooms")
    results.append("Service_Type")
    results.append(cursor.fetchall())

    cursor.close()
    connection.close()
    return results

def get_table_names():
    connection = connect()
    cursor = connection.cursor()

    cursor.execute("SHOW Tables")
    result = cursor.fetchall()

    cursor.close()
    connection.close()
    return result

def query_table(table_name):
    connection = connect()
    cursor = connection.cursor()
    cursor.execute(f"SELECT * FROM `{table_name[0]}`")
    table_data = cursor.fetchall()
    cursor.close()
    connection.close()
    return table_data



def delete_record(table_name, record_id):
    try:
        connection = connect()
        cursor = connection.cursor()
        # Delete an existing record from a table in the database
        cursor.execute("DELETE FROM {} WHERE 
{}_id=%s".format(table_name[0], table_name[0]), (record_id,))
        connection.commit()
    except Exception as e:
        print(e)
    finally:
        cursor.close()
        connection.close()

def search_record(table_name, record_id):
    try:
        connection = connect()
        cursor = connection.cursor()
        if table_name[0] == "roomprice":
            table_name[0] = "roomtype"
        # Search an existing record from a table in the database
        query = f"SELECT * FROM {table_name[0]} WHERE {table_name[0]}_id = 
{record_id}"
        cursor.execute(query)
        result = cursor.fetchall()
    except Exception as e:
        print(e)
    finally:
        cursor.close()
        connection.close()
        return result
```

```
"""
---------------------------------------------
File: databaseGUI.py
Authors: Group 40
Course: CP363
```

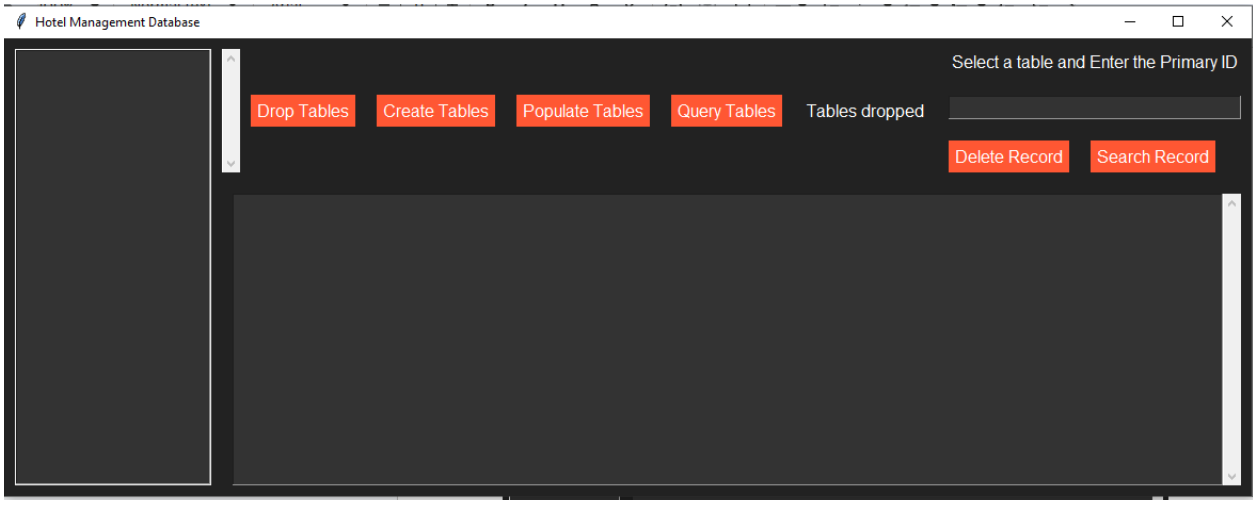
```
Date: 2023-03-26
-----------------------------------------------
"""
import tkinter as tk
from tkinter import scrolledtext
import GUI_Functions

# Create the main window
root = tk.Tk()
root.title("Hotel Management Database")
root.configure(bg="#2C3E50")

# Create custom font
font_style = ("Helvetica", 12)

# Create custom color scheme
bg_color = "#222222"
fg_color = "#FFFFFF"
button_bg = "#FF5733"
button_fg = "#FFFFFF"
listbox_bg = "#333333"
listbox_fg = "#FFFFFF"
textbox_bg = "#333333"
textbox_fg = "#FFFFFF"
label_fg = "#FFFFFF"
# Configure the window
root.configure(bg=bg_color)



def drop_tables():
    GUI_Functions.drop_all_tables()
    status_label.config(text="Tables dropped")
    table_listbox.delete(0, tk.END) # Clear the listbox
    get_table_names() # Update the listbox to show only existing tables

def create_tables():
    GUI_Functions.create_tables()
    status_label.config(text="Tables created")
    display_tables()
    get_table_names()

def populate_tables():
    GUI_Functions.populate_tables()
    status_label.config(text="Tables populated")
    display_tables()

def query_all_tables():
    results = GUI_Functions.query_all_tables()
    display_tables(*results)
    status_label.config(text="Tables queried")

def display_tables(*tables):
    text_box.delete("1.0", "end")
    for i, table in enumerate(tables):
        text_box.insert("end", str(table))
        if i < len(tables) - 1:
            text_box.insert("end", "\n\n")

def query_table(table_name):
    results = GUI_Functions.query_table(table_name)
    display_tables(*results)
    status_label.config(text="Table {} queried".format(table_name[0]))

def get_table_names():
    table_names = GUI_Functions.get_table_names()
    for name in table_names:
        table_listbox.insert("end", name)

# def add_record(table_name, values):
#     GUI_Functions.add_record(table_name, values)
#     status_label.config(text="Record Added")
#     display_tables()

# def update_record(table_name, record_id, values):
#     GUI_Functions.update_record(table_name, record_id, values)
#     status_label.config(text="Record Updated")
#     display_tables()

def delete_record(table_name, record_id):
    GUI_Functions.delete_record(table_name, record_id)
    status_label.config(text="Record Deleted")
    display_tables()

def search_record(table_name, record_id):
    results = GUI_Functions.search_record(table_name, record_id)
    status_label.config(text="Record Found")
    print(results)
    if len(results) == 0:
            status_label.config(text="No Record Found")
            text_box.delete("1.0", "end")
            text_box.insert("end", "NO RECORD FOUND")
    else:
        status_label.config(text="Record Found")
        display_tables(*results)

# Create the buttons
drop_button = tk.Button(root, text="Drop Tables", command=drop_tables,
font=font_style, bg=button_bg, fg=button_fg, borderwidth=2, relief="flat",
highlightthickness=0)
create_button = tk.Button(root, text="Create Tables", command=create_tables,
font=font_style, bg=button_bg, fg=button_fg, borderwidth=2, relief="flat",
highlightthickness=0)
populate_button = tk.Button(root, text="Populate Tables",
command=populate_tables, font=font_style, bg=button_bg, fg=button_fg,
borderwidth=2, relief="flat", highlightthickness=0)
query_button = tk.Button(root, text="Query Tables", command=query_all_tables,
font=font_style, bg=button_bg, fg=button_fg, borderwidth=2, relief="flat",
highlightthickness=0)
delete_button = tk.Button(root, text="Delete Record", command=lambda:
delete_record(table_listbox.get(table_listbox.curselection()),
search_entry.get()), font=font_style, bg=button_bg, fg=button_fg,
borderwidth=2, relief="flat", highlightthickness=0)
search_button = tk.Button(root, text="Search Record", command=lambda:
search_record(table_listbox.get(table_listbox.curselection()),
search_entry.get()), font=font_style, bg=button_bg, fg=button_fg,
borderwidth=2, relief="flat", highlightthickness=0)
# Create a label to display status messages
status_label = tk.Label(root, text="Ready", font=font_style, fg=label_fg,
bg=bg_color)

# Create the labels and entry box for the search bar
search_label = tk.Label(root, text="Select a table and Enter the Primary ID",
font=font_style, fg=label_fg, bg=bg_color)
search_entry = tk.Entry(root, width=30, font=font_style, bg=textbox_bg,
fg=textbox_fg)

# Create a text box to display the query results
text_box = scrolledtext.ScrolledText(root, height=15, width=50, wrap="word",
font=font_style, bg=textbox_bg, fg=textbox_fg)

# Create a listbox to display table names
table_listbox = tk.Listbox(root, font=font_style, bg=listbox_bg, fg=listbox_fg)
get_table_names()
table_listbox.bind("<<ListboxSelect>>", lambda event:
```

```
query_table(table_listbox.get(table_listbox.curselection())))
table_listbox_scrollbar = tk.Scrollbar(root, orient="vertical",
command=table_listbox.yview)
table_listbox.config(yscrollcommand=table_listbox_scrollbar.set)


# Pack the buttons, labels, and entry box into the window
table_listbox.pack(side="left", padx=10, pady=10, fill="both", expand=False)
text_box.pack(side="bottom", padx=10, pady=10, fill="both", expand=False)
table_listbox_scrollbar.pack(side="left", padx=0, pady=10, fill="y")
drop_button.pack(side="left", padx=10, pady=10)
create_button.pack(side="left", padx=10, pady=10)
populate_button.pack(side="left", padx=10, pady=10)
query_button.pack(side="left", padx=10, pady=10)
status_label.pack(side="left", padx=10, pady=10)
search_label.pack(side="top", padx=10, pady=10)
search_entry.pack(side="top", padx=10, pady=10)
delete_button.pack(side="left", padx=10, pady=10)
search_button.pack(side="left", padx=10, pady=10)


# Start the main loop
root.mainloop()
```

Below is a visual display of the database GUI:

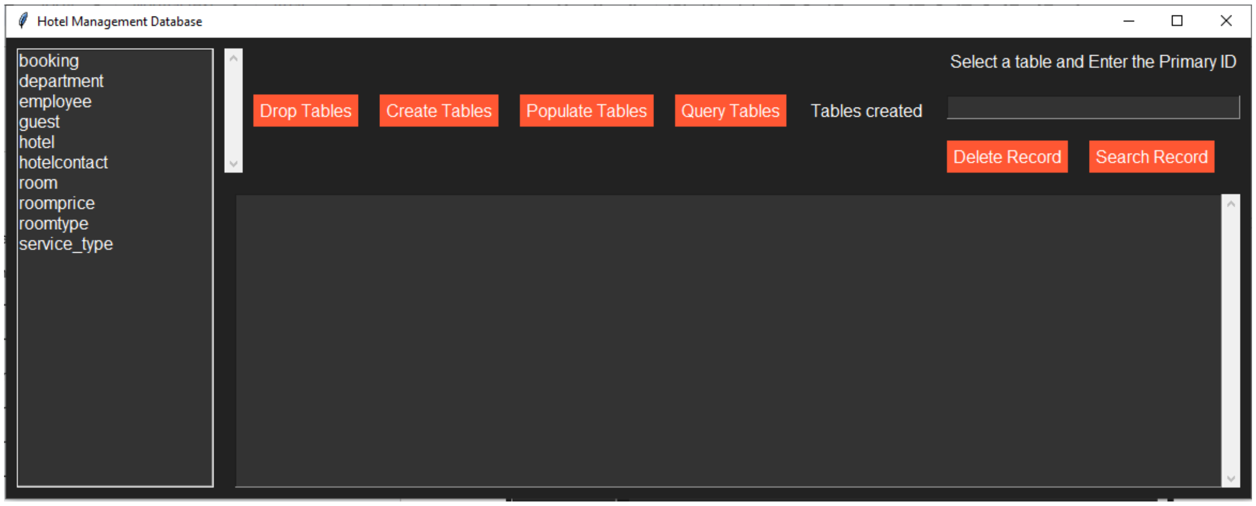


This is the main screen of the GUI, which displays all the options and tables as seen above



Tables dropped

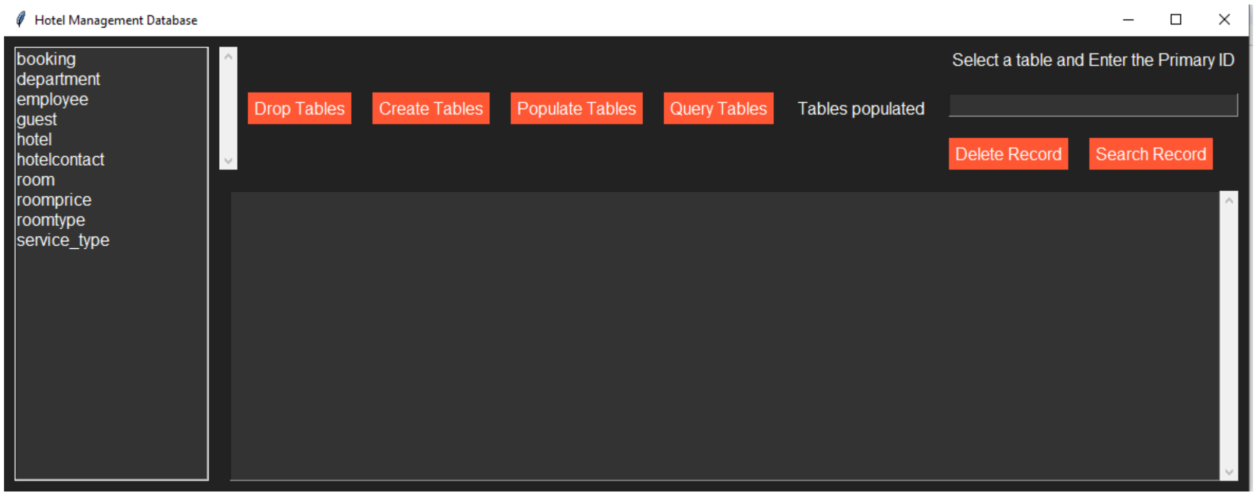
Once Drop Tables is clicked, all the tables are dropped in the actual database and this is reflected in the GUI as well.



booking
department
employee
guest
hotel
hotelcontact
room
roomprice
roomtype
service_type

Tables created

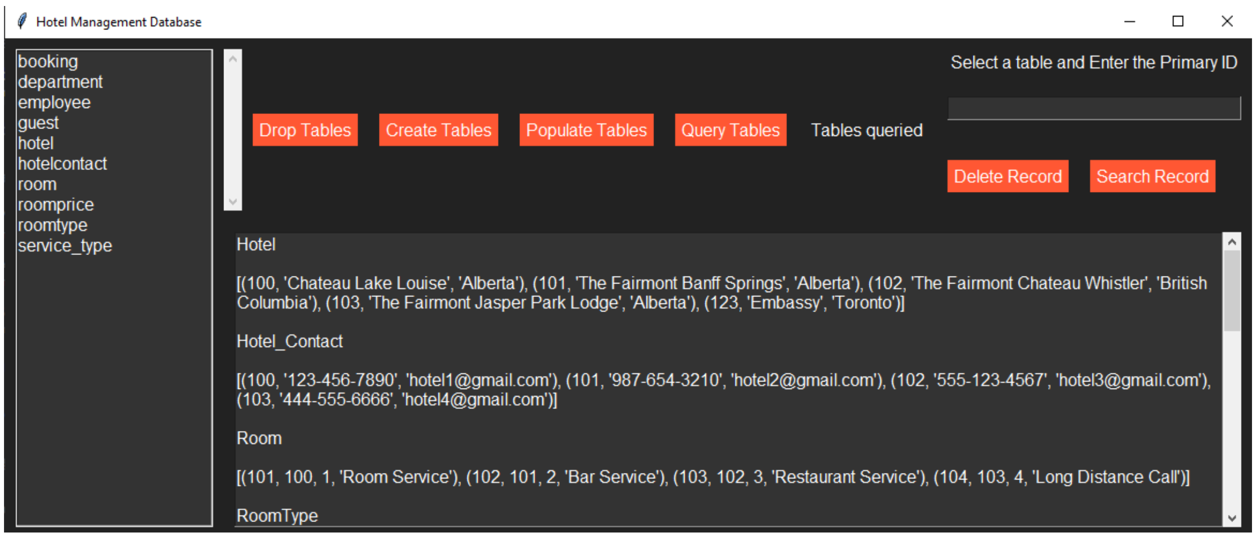
Clicking the Create Tables button will create all the tables and display them in the GUI as well



booking
department
employee
guest
hotel
hotelcontact
room
roomprice
roomtype
service_type

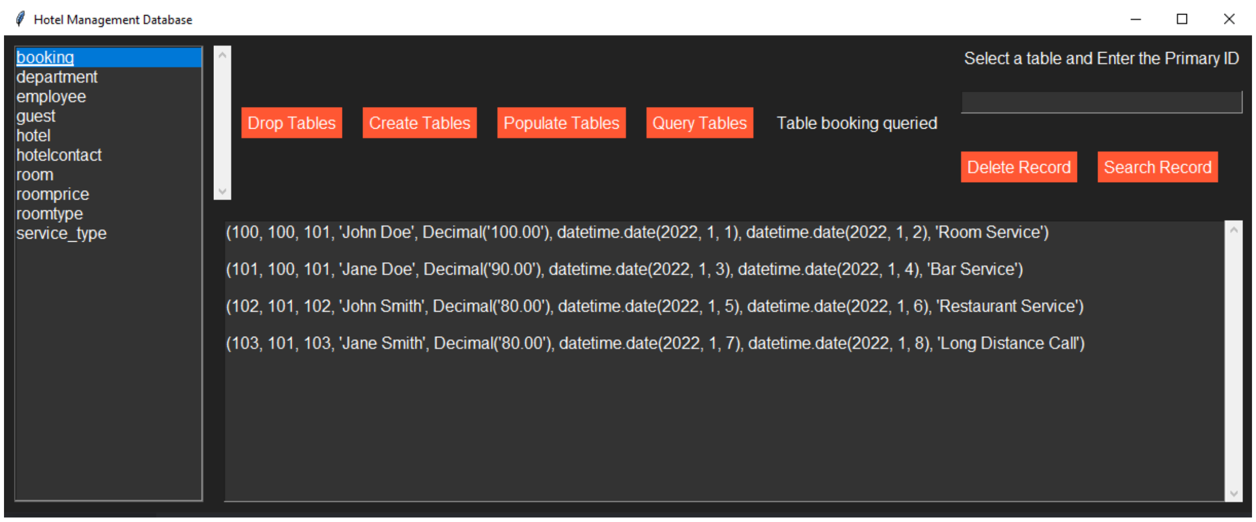
Tables populated

Once the Populate Tables button is pressed, all the tables are populated with dummy data and the status label on the GUI is also updated.



```
Hotel

[(100, 'Chateau Lake Louise', 'Alberta'), (101, 'The Fairmont Banff Springs', 'Alberta'), (102, 'The Fairmont Chateau Whistler', 'British
Columbia'), (103, 'The Fairmont Jasper Park Lodge', 'Alberta'), (123, 'Embassy', 'Toronto')]

Hotel_Contact

[(100, '123-456-7890', 'hotel1@gmail.com'), (101, '987-654-3210', 'hotel2@gmail.com'), (102, '555-123-4567', 'hotel3@gmail.com'),
(103, '444-555-6666', 'hotel4@gmail.com')]

Room

[(101, 100, 1, 'Room Service'), (102, 101, 2, 'Bar Service'), (103, 102, 3, 'Restaurant Service'), (104, 103, 4, 'Long Distance Call')]

RoomType
```

After Clicking Query Tables, all the tables are queried using simple queries and displayed in the GUI.



```
(100, 100, 101, 'John Doe', Decimal('100.00'), datetime.date(2022, 1, 1), datetime.date(2022, 1, 2), 'Room Service')

(101, 100, 101, 'Jane Doe', Decimal('90.00'), datetime.date(2022, 1, 3), datetime.date(2022, 1, 4), 'Bar Service')

(102, 101, 102, 'John Smith', Decimal('80.00'), datetime.date(2022, 1, 5), datetime.date(2022, 1, 6), 'Restaurant Service')

(103, 101, 103, 'Jane Smith', Decimal('80.00'), datetime.date(2022, 1, 7), datetime.date(2022, 1, 8), 'Long Distance Call')
```

Clicking on any of the tables on the right will display the corresponding table in the GUI

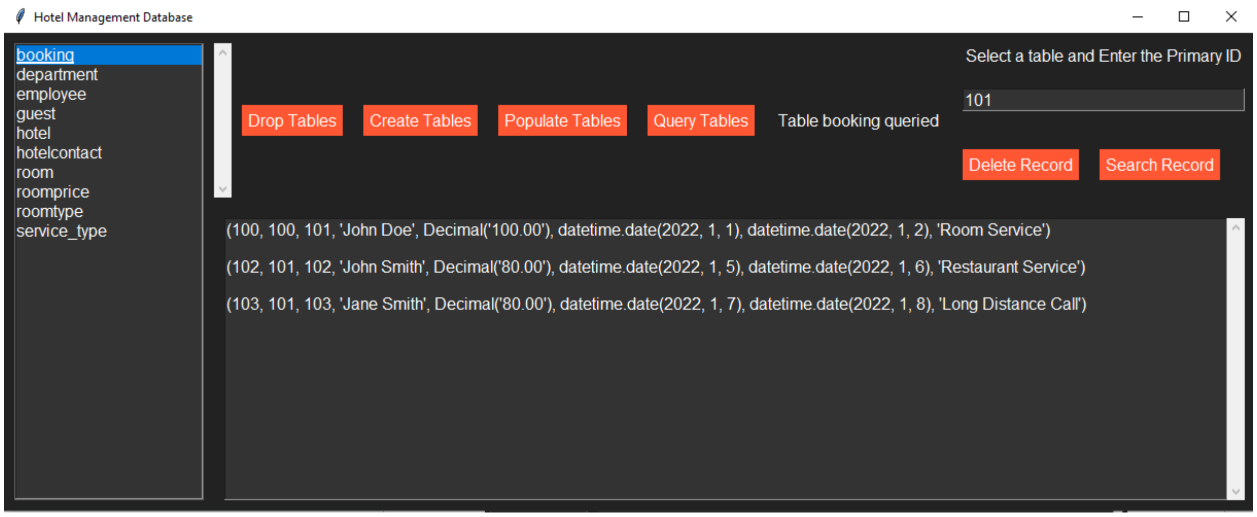


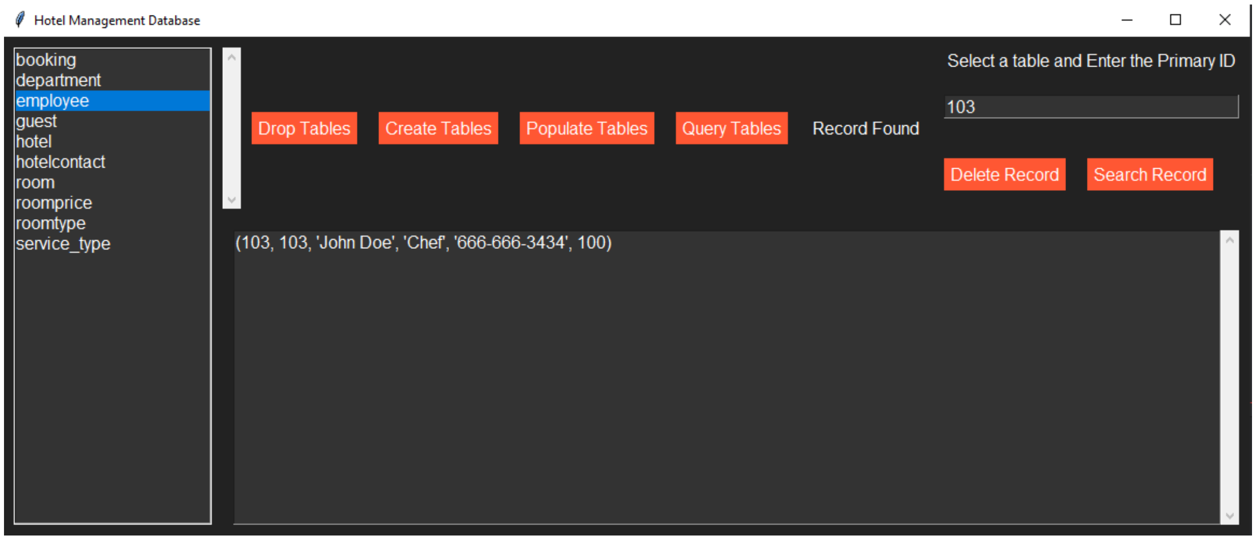
Table booking queried

Select a table and Enter the Primary ID

101

```
(100, 100, 101, 'John Doe', Decimal('100.00'), datetime.date(2022, 1, 1), datetime.date(2022, 1, 2), 'Room Service')
(102, 101, 102, 'John Smith', Decimal('80.00'), datetime.date(2022, 1, 5), datetime.date(2022, 1, 6), 'Restaurant Service')
(103, 101, 103, 'Jane Smith', Decimal('80.00'), datetime.date(2022, 1, 7), datetime.date(2022, 1, 8), 'Long Distance Call')
```

After selecting a table you can enter the primary ID in the search bar and delete it, the row 101 from hotel has been deleted.



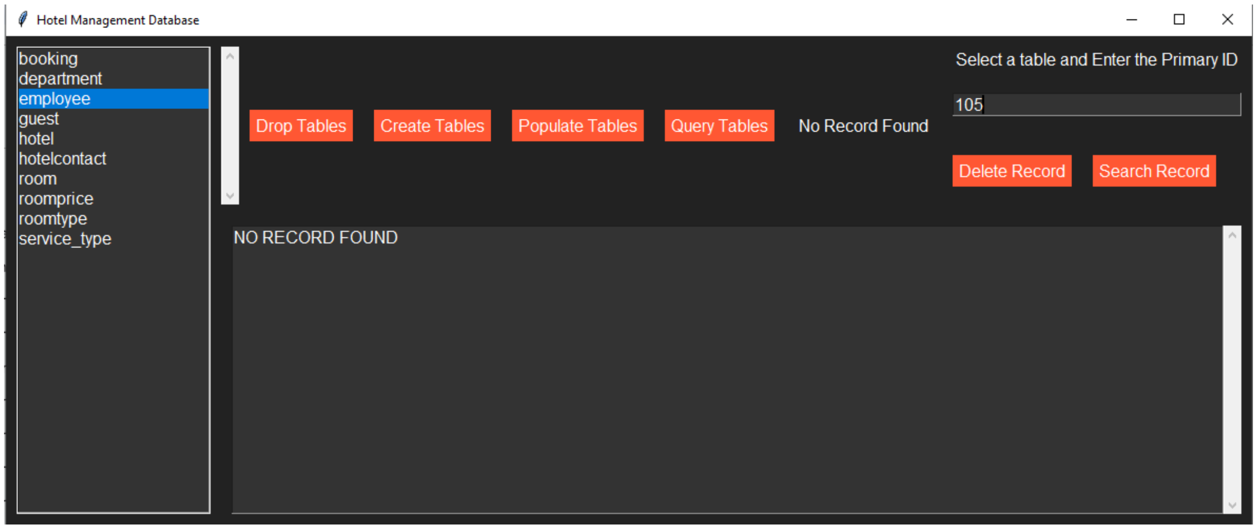
Record Found

Select a table and Enter the Primary ID

103

```
(103, 103, 'John Doe', 'Chef', '666-666-3434', 100)
```

We can also search for a record using the primary ID and selecting the corresponding table from the listbox on the right, this will bring up the record if it exists. If not, the screen below will be displayed:



Select a table and Enter the Primary ID

105

No Record Found

```
NO RECORD FOUND
```

#10 - Database Management System

Documentation and Relational Algebra Statements

Waleed G , Asil K, Kamil K , Arsalan K

Date: 03/12/2023

Turning our queries into relational algebra notation

Our current queries in SQL:

1. SELECT * FROM Employee;
2. SELECT * FROM Department;
3. SELECT * FROM Hotel;
4. SELECT * FROM Guest;
5. SELECT * FROM Booking;
6. SELECT * FROM Service;
7. SELECT * FROM Room;
8. SELECT * FROM roomtype;
9. SELECT * FROM service_type;

10 simple queries

10.
11. SELECT guest_id, guest_name FROM Guest;

Advanced queries

12. SELECT g.guest_name, g.guest_contact_info, b.booking_id, r.room_id, b.checkin_date, b.checkout_date, b.requests
    FROM Guest g
    JOIN Booking b ON g.guest_id = b.guest_id
    JOIN Room r ON b.room_id = r.room_id;

13. SELECT e.employee_id, e.employee_name, e.job_title, d.dept_name, h.hotel_name, h.location
    FROM Employee e
    JOIN Department d ON e.dept_id = d.dept_ID
    JOIN Hotel h ON e.hotel_id = h.hotel_id;

14. SELECT rt.roomtype_id, rt.room_price, SUM(rt.room_price) AS total_revenue
    FROM RoomType rt
    JOIN Room r ON rt.roomtype_id = r.roomtype_id
    JOIN Booking b ON r.room_id = b.room_id
    GROUP BY rt.roomtype_id, rt.room_price;

Now we will convert the above sql queries into relational algebra notation

Relational Algebra Notation:

1. π (Employee)
2. π (Department)
3. π (Hotel)
4. π (HotelContact)
5. π (Guest)
6. π (Booking)
7. π (Room)
8. π (RoomType)
9. π (Service_Type)

10 simple queries in relational algebra notation

10. σ (hotel_id=123) (Hotel)
11. σ (location="Toronto") (Hotel)
12. σ (roomtype_id=1) (RoomType)
13. σ (hotel_id=100) (Room)
14. σ (guest_name="John Doe") (Guest)
15. σ (booking_id=100) (Booking)
16. σ (dept_id=100) (Department)
17. π (employee_id, employee_name, dept_id) (Employee)
18. π (service_type_id, service_description) (Service_Type)
19. π (guest_id, guest_name) (Guest)

Advanced queries in relational algebra notation

20. ρ (G, Guest)
    ρ (B, Booking)
    ρ (R, Room)
    σ (G.guest_id = B.guest_id AND B.room_id = R.room_id) (G >< B >< R)
    π (guest_name, guest_contact_info, booking_id, room_id, checkin_date, checkout_date, requests) (σ (G.guest_id = B.guest_id AND B.room_id = R.room_id) (G >< B >< R))

21. ρ (E, Employee)
    ρ (D, Department)
    ρ (H, Hotel)
    σ (E.dept_id = D.dept_ID AND E.hotel_id = H.hotel_id) (E >< D >< H)
    π (employee_id, employee_name, job_title, dept_name, hotel_name, location) (σ (E.dept_id = D.dept_ID AND E.hotel_id = H.hotel_id) (E >< D >< H))

22. ρ (RT, RoomType)
ρ (R, Room)
ρ (B, Booking)
π (roomtype_id, room_price, total_revenue) (γ (rt.roomtype_id, rt.room_price, SUM(rt.room_price) as total_revenue) (RT >< R >< B))