SQL Course Report: Clinic Management System

Name	**ID**
Manar Khalifa	231001467
Reem Ramadan	231000934
Youstina Ramzy	221001933
Joumana Ehab	231002356

1 Introduction

This report presents a Clinic Management System developed using SQLite and Python with a Tkinterbased GUI, alongside SQL code for a MySQL database. The system manages hospital departments, clinics, doctors, patients, appointments, and doctor-appointment relationships. The report covers the database schema, SQL operations, and the Python application's functionality, demonstrating key SQL concepts such as table creation, foreign key constraints, and joins.

2 Database Schema

The database is designed to manage a hospital's operations, with tables for departments, clinics, doctors, patients, appointments, and doctor-appointment associations. Below is the schema implemented in MySQL, with slight differences from the SQLite version used in the Python application (e.g., explicit primary key values vs. AUTOINCREMENT).

2.1 Tables and Relationships

- **Department**: Stores department information with a primary key `D_ID` and a `name` field.
- **Clinic**: Stores clinic details, including `C_ID`, `Name`, `address`, and `FK_D_ID` (foreign key referencing `Department.D_ID`).
- **Doctor**: Contains doctor information, including `Do_ID`, `name`, `phone_number`, `address`, and `FK_D_ID` (foreign key to `Department`).
- **Patient**: Stores patient data with `P_ID`, `name`, `Job`, `birth_date`, `phone_number`, and `address`.
- **Appointment**: Manages appointment details, including `A_ID`, `Date`, `start_time`, `End_time`, `status`, `cost`, and `FK_P_ID` (foreign key to `Patient`).
- **Doctor_Appointment**: A junction table linking doctors to appointments via `FK_Do_ID` and `FK_A_ID`, with a composite primary key.

The schema enforces referential integrity through foreign key constraints, ensuring that clinics and doctors are associated with valid departments, appointments are linked to valid patients, and doctor-appointment relationships are valid.

3 Python Application Overview

The Python application uses SQLite and Tkinter to provide a GUI for managing clinics and departments. Key features include:

- **Add Clinic**: Users can input a clinic's name, address, and select a department from a dropdown menu populated from the `Department` table. The data is inserted into the `Clinic` table.
- **View Clinics**: A Treeview widget displays all clinics with their IDs, names, addresses, and associated department names, using a LEFT JOIN query.
- **Add Department**: A simple interface allows adding new departments for testing purposes, updating the department dropdown in real-time.

The SQLite database uses `AUTOINCREMENT` for primary keys, unlike the MySQL code, which requires explicit `D_ID` and `C_ID` values.

4 SQL Operations

The provided MySQL code demonstrates core SQL concepts:

```sql
CREATE DATABASE clinic;
USE clinic;

CREATE TABLE Department (
    D_ID INT PRIMARY KEY,
    name VARCHAR(100) NOT
    NULL
);

CREATE TABLE Clinic ( C_ID INT
    PRIMARY KEY, Name
    VARCHAR(100) NOT NULL,
    address VARCHAR(255),
    FK_D_ID INT,
    FOREIGN KEY (FK_D_ID) REFERENCES Department(D_ID)
);
```

4.1 Database and Table Creation

The `CREATE DATABASE` and `CREATE TABLE` statements establish the database and define tables with appropriate data types and constraints. The `NOT NULL` constraint ensures department and clinic names are mandatory, while foreign keys maintain referential integrity.

```
INSERT INTO Department (D_ID, name)
VALUES (1, 'Cardiology'),
       (2, 'Neurology'),
       (3, 'sketal'),
       (4, 'molucular'),
       (5, 'digstive');

INSERT INTO Clinic (C_ID, Name, address, FK_D_ID)
VALUES (1, 'Central␣Clinic', '123␣Main␣St', 1),
       (2, 'Westside␣Clinic', '456␣West␣St', 2),
       (3, 'Eastside␣Clinic', '789␣East␣St', 1);
```

4.2 Data Insertion

The `INSERT` statements populate the `Department` and `Clinic` tables. Note that the department names 'sketal', 'molucular', and 'digstive' appear to be typos (likely intended as 'Skeletal', 'Molecular', and 'Digestive'), which could be corrected for clarity.

```
SELECT
    Clinic.C_ID,
    Clinic.Name,
    Clinic.address,
    Department.name
FROM
    Clinic
JOIN
    Department ON Clinic.FK_D_ID = Department.D_ID;

SELECT
    Clinic.C_ID,
```

4.3 Queries

```
    Clinic.Name,
    Clinic.address
FROM
    Clinic
JOIN
    Department ON Clinic.FK_D_ID = Department.D_ID
WHERE
    Department.name = 'Cardiology';
```

- The first query uses an `INNER JOIN` to retrieve clinic details along with their department names, demonstrating the use of joins to combine data from related tables.
- The second query filters clinics by department (`Cardiology`), showcasing the `WHERE` clause for conditional retrieval.

5 Analysis and Observations

The Python application effectively uses SQLite to manage clinic and department data, with a userfriendly GUI. The MySQL code complements this by defining a more comprehensive schema, including tables for doctors, patients, and appointments, which are not yet implemented in the Python application. Key SQL concepts demonstrated include:

- **Table Creation**: Proper use of primary and foreign keys ensures data integrity.
- **Data Insertion**: Populating tables with meaningful data.
- **Joins**: Combining data from multiple tables to produce meaningful results.
- **Filtering**: Using `WHERE` to retrieve specific records.

The SQLite implementation in the Python code uses `AUTOINCREMENT`, which simplifies primary key management compared to the MySQL code's explicit `D_ID` values. The junction table (`Doctor_Appointment`) effectively handles the many-to-many relationship between doctors and appointments.

6 Recommendations

1. **Correct Typos**: Update department names (e.g., 'sketal' to 'Skeletal', 'molucular' to 'Molecular', 'digstive' to 'Digestive') in the MySQL code for professionalism.
2. **Expand GUI**: Extend the Python application to include management of doctors, patients, and appointments, aligning with the MySQL schema.
3. **Input Validation**: Add checks for duplicate clinic or department names in the Python application to prevent errors.
4. **Error Handling**: Enhance the MySQL code with checks for foreign key violations during insertion.

7 Conclusion

The Clinic Management System demonstrates effective use of SQL and Python for database management. The MySQL code establishes a robust schema with proper constraints, while the Python application provides a functional interface for managing clinics and departments. This project showcases essential SQL skills, including schema design, data insertion, and querying, with potential for further enhancement to cover the full schema.