Kathmandu University

Department of Computer Science and Engineering

Dhulikhel, Kavre

Database Management System (DBMS) Report

[Code No: COMP 232]

Submitted by:

Sophiya Shrestha

Roll no. 46

Computer Science (2nd Year/2nd Semester)

Submitted to:

Mr. Sanjog Sigdel

Department of Computer Science and Engineering

Submission Date: 31-12-2024

Table of Contents

1) Introduction

Database Management Systems (DBMS) play a crucial role in managing, organizing, and retrieving data efficiently.

In this report, we explore the creation and manipulation of a database designed to manage patient records, illustrating the concepts learned in class. By leveraging SQL and relational algebraic operations, the report demonstrates the design, implementation, and querying of relational databases. Key aspects such as table creation, data insertion, complex queries, and transaction management are covered in detail.

This report also highlights the power and flexibility of SQL in handling complex datasets, emphasizing its capability to perform intricate operations like joins, set operations, and data modifications while maintaining data integrity. Additionally, the practical application of transactions showcases how DBMS ensures consistency and reliability, even in multi-step operations.

2. Database Design

Database: Patient_Records

The Patient Records database consists of three interrelated tables:

Patients: Stores general information about patients.

Attributes: PatientID (Primary Key), Name, Age, Gender, Email.

Appointments: Records patient appointments.

Attributes: AppointmentID (Primary Key), PatientID (Foreign Key), Date, Doctor, Reason.

Medical_History: Maintains medical history of patients.

Attributes: HistoryID (Primary Key), PatientID (Foreign Key), Diagnosis, Treatment, Notes.

3) ER Diagram

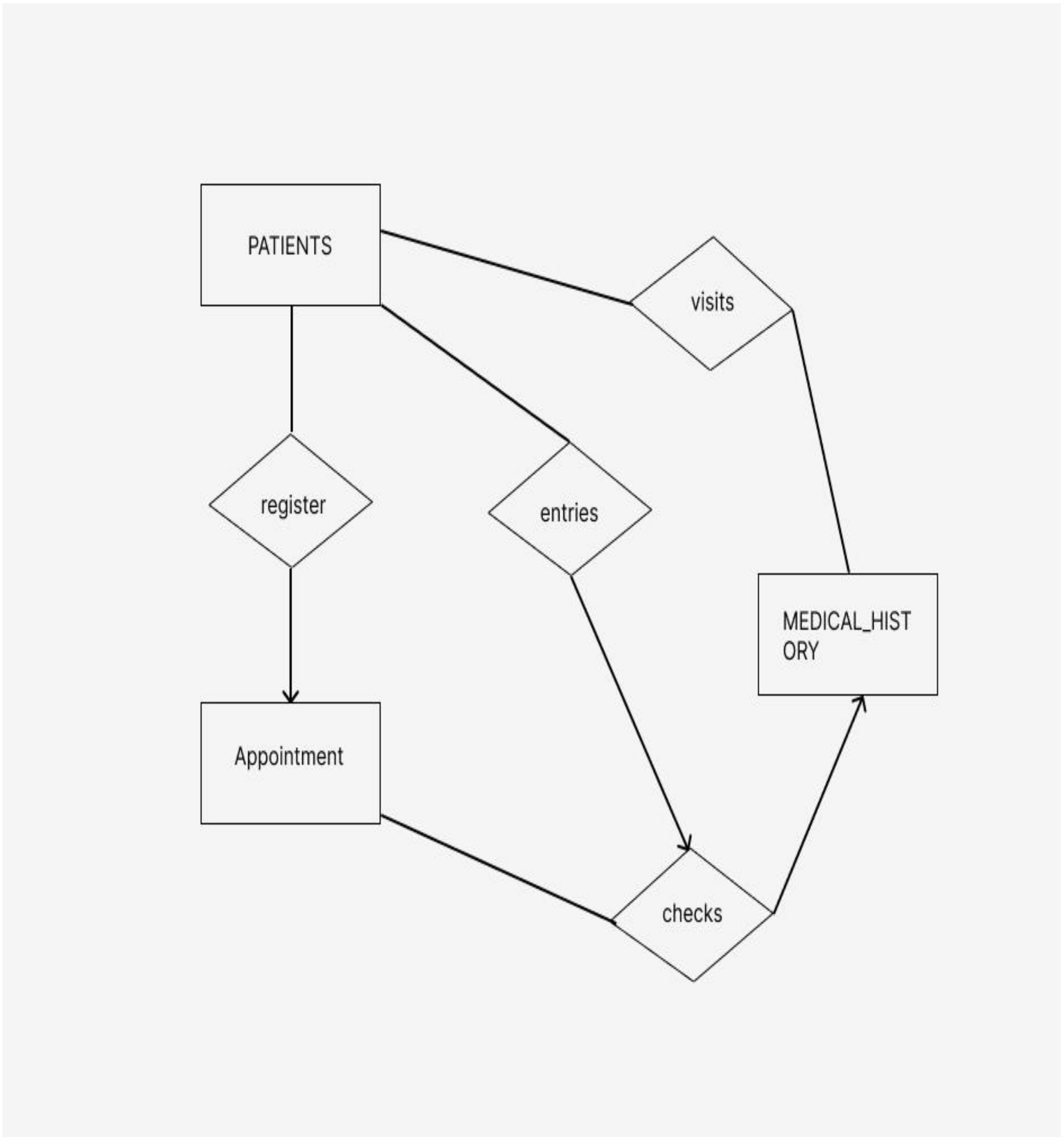


Figure 1 ER Diagram

3.1) Explanation of ER diagram

Relationships:

A Patient can have multiple Appointments (One-to-Many relationship).

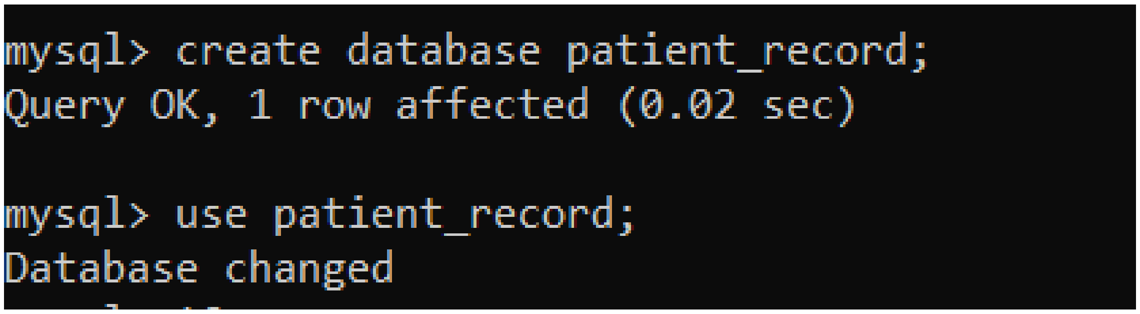
A Patient can have multiple entries in Medical History (One-to-Many relationship)

A doctor checks for multiple diagnoses or treatments documented in the medical history (One-to-Many)

A patient may visit multiple doctors over time, and a doctor may treat multiple patients. (Many-to-Many)

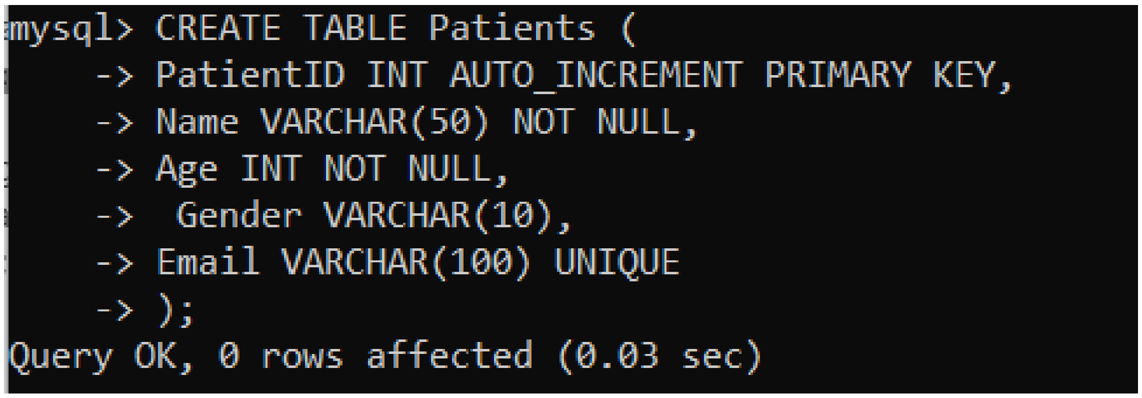
4. Implementation

Creating a database



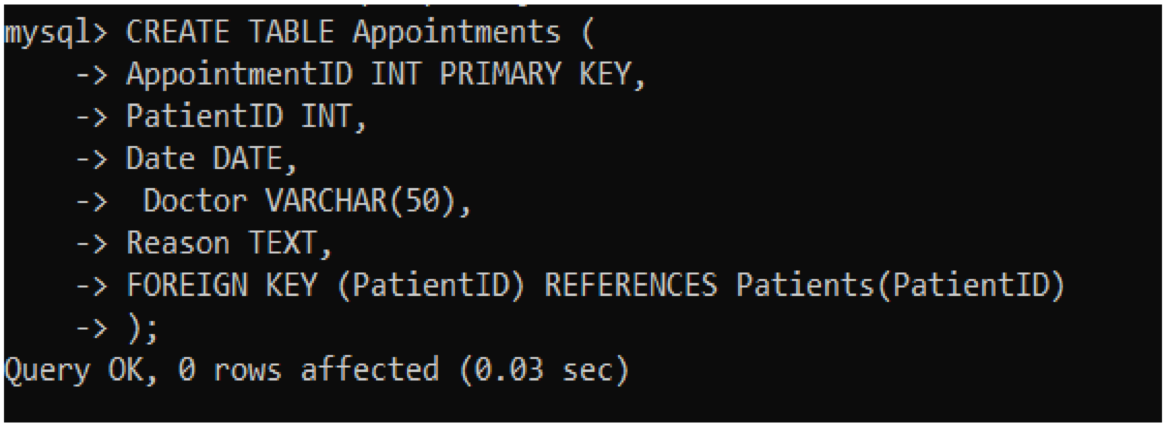
```
mysql> create database patient_record;
Query OK, 1 row affected (0.02 sec)

mysql> use patient_record;
Database changed
```

Create Patients Table



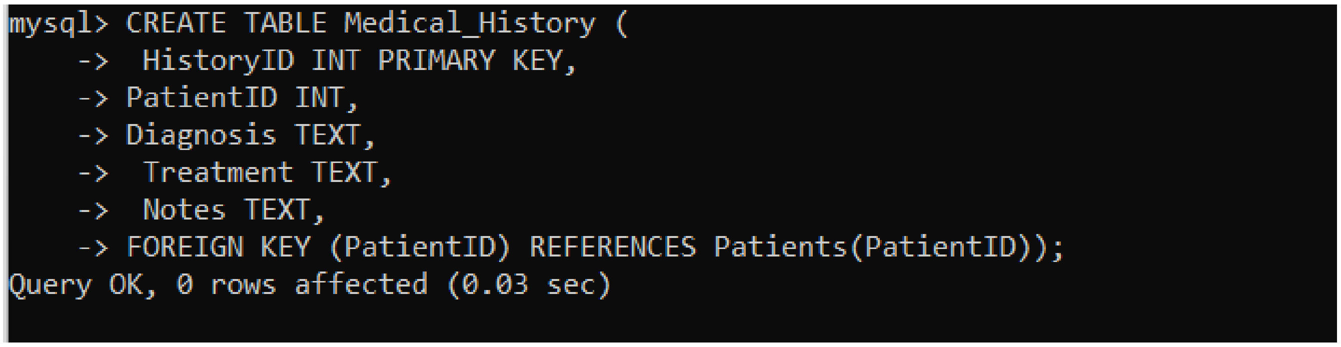
```
mysql> CREATE TABLE Patients (
    -> PatientID INT AUTO_INCREMENT PRIMARY KEY,
    -> Name VARCHAR(50) NOT NULL,
    -> Age INT NOT NULL,
    ->  Gender VARCHAR(10),
    -> Email VARCHAR(100) UNIQUE
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Create Appointments Table



```
mysql> CREATE TABLE Appointments (
    -> AppointmentID INT PRIMARY KEY,
    -> PatientID INT,
    -> Date DATE,
    ->  Doctor VARCHAR(50),
    -> Reason TEXT,
    -> FOREIGN KEY (PatientID) REFERENCES Patients(PatientID)
    -> );
Query OK, 0 rows affected (0.03 sec)
```

Creating Medical_History Table:

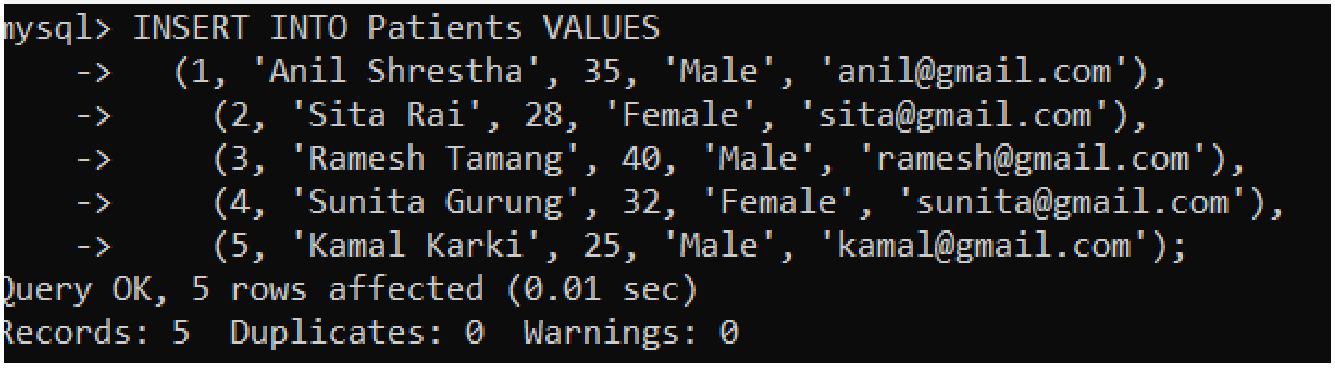


```
mysql> CREATE TABLE Medical_History (
    ->  HistoryID INT PRIMARY KEY,
    -> PatientID INT,
    -> Diagnosis TEXT,
    ->  Treatment TEXT,
    ->  Notes TEXT,
    -> FOREIGN KEY (PatientID) REFERENCES Patients(PatientID));
Query OK, 0 rows affected (0.03 sec)
```

The PatientID in the MedicalHistory and Appointments tables establishes a relationship with the Patients table, ensuring that every medical record and appointment is linked to a patient.

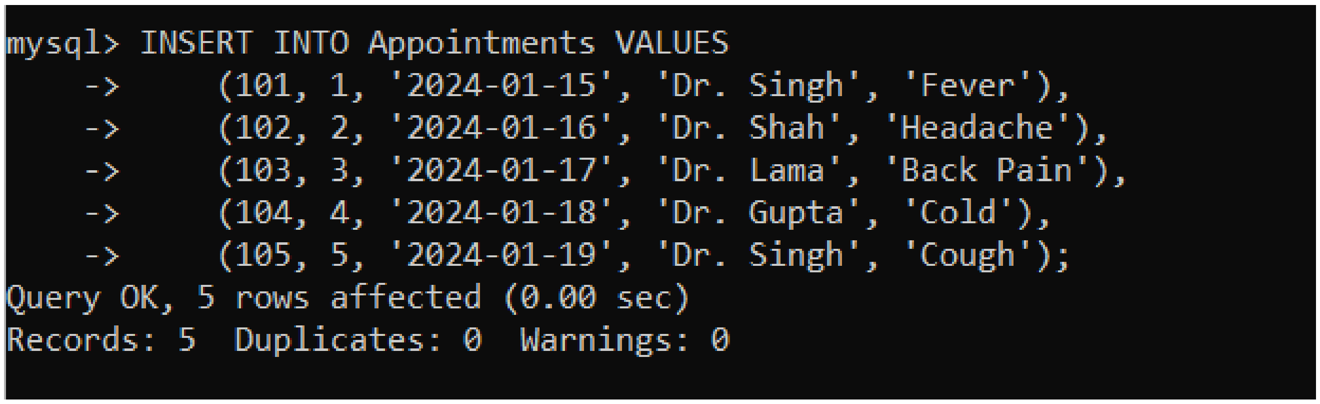
4.2) Inserting data into table:

Insert into Patients Table:



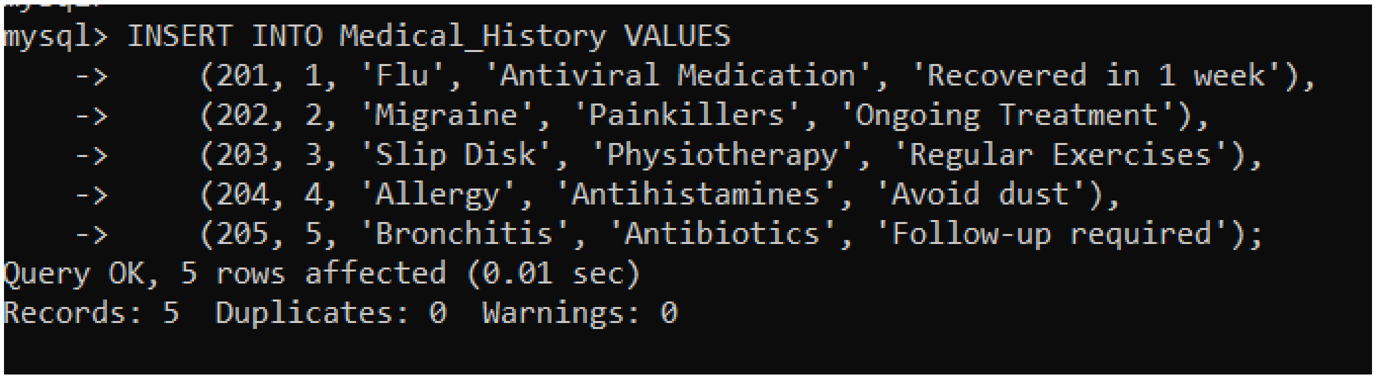
```
mysql> INSERT INTO Patients VALUES
    ->   (1, 'Anil Shrestha', 35, 'Male', 'anil@gmail.com'),
    ->     (2, 'Sita Rai', 28, 'Female', 'sita@gmail.com'),
    ->     (3, 'Ramesh Tamang', 40, 'Male', 'ramesh@gmail.com'),
    ->     (4, 'Sunita Gurung', 32, 'Female', 'sunita@gmail.com'),
    ->     (5, 'Kamal Karki', 25, 'Male', 'kamal@gmail.com');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

Insert into Appointments Table:



```
mysql> INSERT INTO Appointments VALUES
    ->     (101, 1, '2024-01-15', 'Dr. Singh', 'Fever'),
    ->     (102, 2, '2024-01-16', 'Dr. Shah', 'Headache'),
    ->     (103, 3, '2024-01-17', 'Dr. Lama', 'Back Pain'),
    ->     (104, 4, '2024-01-18', 'Dr. Gupta', 'Cold'),
    ->     (105, 5, '2024-01-19', 'Dr. Singh', 'Cough');
Query OK, 5 rows affected (0.00 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

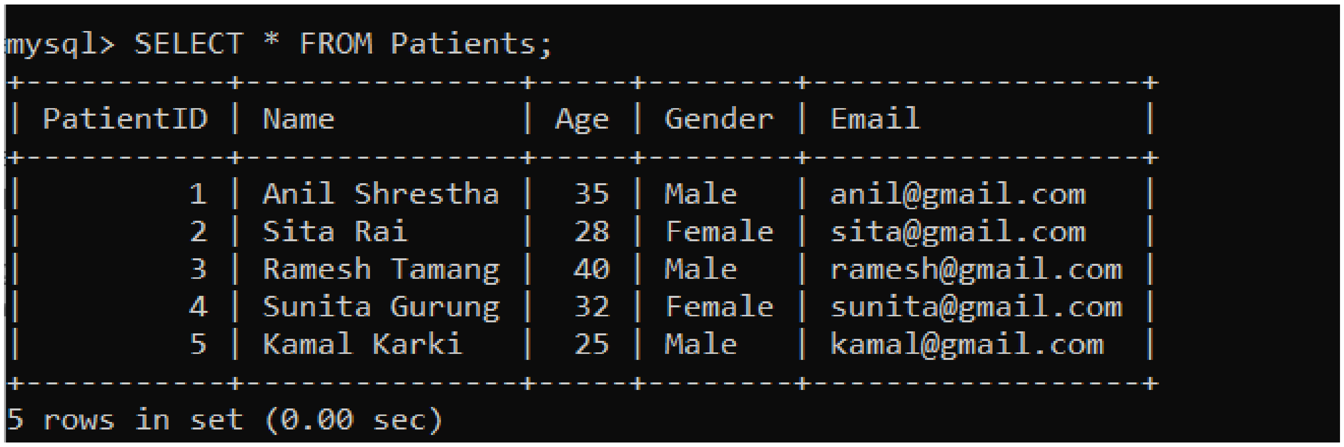
Insert into Medical_History Table:



```
mysql> INSERT INTO Medical_History VALUES
    ->     (201, 1, 'Flu', 'Antiviral Medication', 'Recovered in 1 week'),
    ->     (202, 2, 'Migraine', 'Painkillers', 'Ongoing Treatment'),
    ->     (203, 3, 'Slip Disk', 'Physiotherapy', 'Regular Exercises'),
    ->     (204, 4, 'Allergy', 'Antihistamines', 'Avoid dust'),
    ->     (205, 5, 'Bronchitis', 'Antibiotics', 'Follow-up required');
Query OK, 5 rows affected (0.01 sec)
Records: 5  Duplicates: 0  Warnings: 0
```

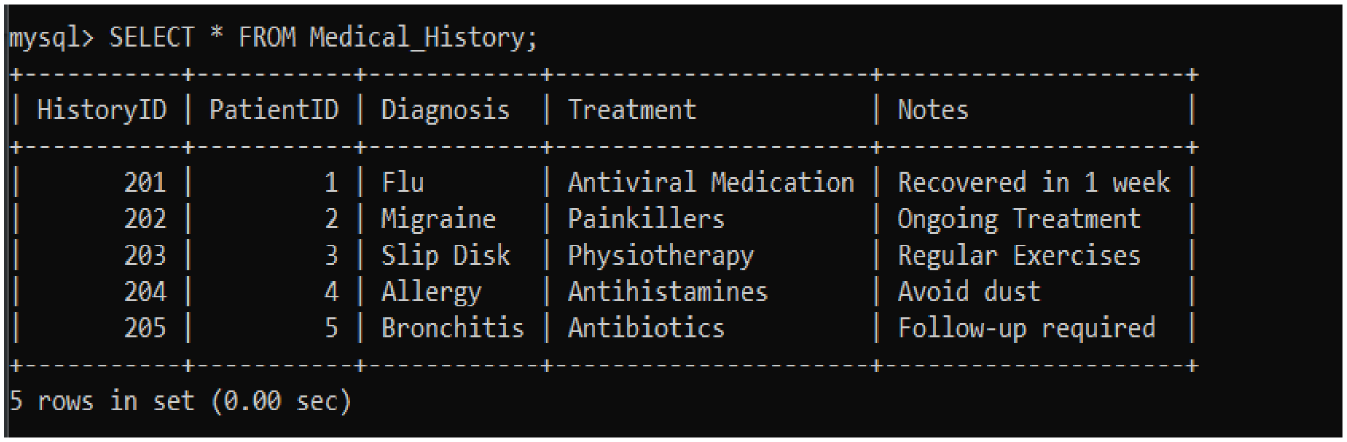
4.3) View Tables:

Patients Table:



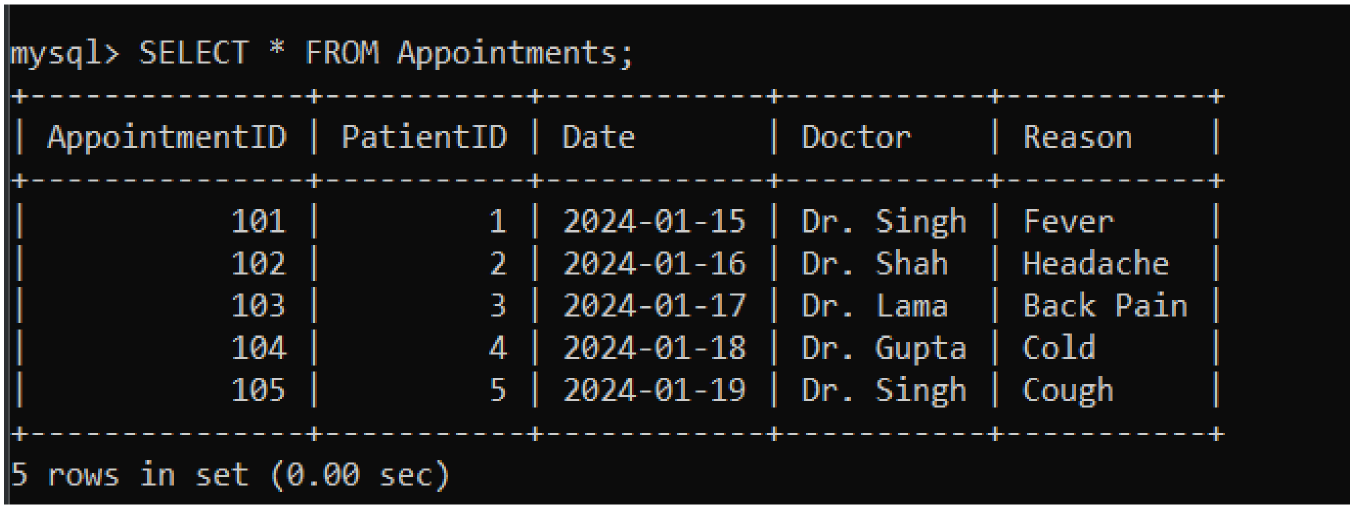
```
mysql> SELECT * FROM Patients;
+-----------+----------------+-----+--------+-------------------+
| PatientID | Name           | Age | Gender | Email             |
+-----------+----------------+-----+--------+-------------------+
|         1 | Anil Shrestha  |  35 | Male   | anil@gmail.com    |
|         2 | Sita Rai       |  28 | Female | sita@gmail.com    |
|         3 | Ramesh Tamang  |  40 | Male   | ramesh@gmail.com  |
|         4 | Sunita Gurung  |  32 | Female | sunita@gmail.com  |
|         5 | Kamal Karki    |  25 | Male   | kamal@gmail.com   |
+-----------+----------------+-----+--------+-------------------+
5 rows in set (0.00 sec)
```

Medical_History Table:



```
mysql> SELECT * FROM Medical_History;
+-----------+-----------+------------+----------------------+---------------------+
| HistoryID | PatientID | Diagnosis  | Treatment            | Notes               |
+-----------+-----------+------------+----------------------+---------------------+
|       201 |         1 | Flu        | Antiviral Medication | Recovered in 1 week |
|       202 |         2 | Migraine   | Painkillers          | Ongoing Treatment   |
|       203 |         3 | Slip Disk  | Physiotherapy        | Regular Exercises   |
|       204 |         4 | Allergy    | Antihistamines       | Avoid dust          |
|       205 |         5 | Bronchitis | Antibiotics          | Follow-up required  |
+-----------+-----------+------------+----------------------+---------------------+
5 rows in set (0.00 sec)
```

Appointments Table:



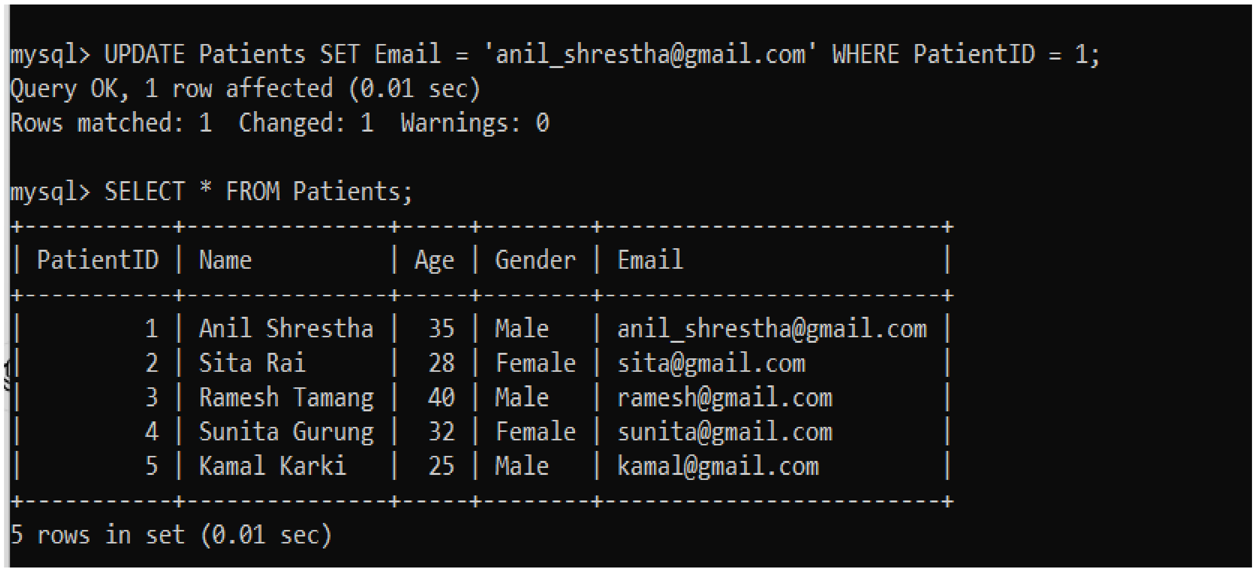
```
mysql> SELECT * FROM Appointments;
+---------------+-----------+------------+-----------+-----------+
| AppointmentID | PatientID | Date       | Doctor    | Reason    |
+---------------+-----------+------------+-----------+-----------+
|           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|           105 |         5 | 2024-01-19 | Dr. Singh | Cough     |
+---------------+-----------+------------+-----------+-----------+
5 rows in set (0.00 sec)
```

5. Query Execution

Basic SQL Queries

5.1) Modify Data:

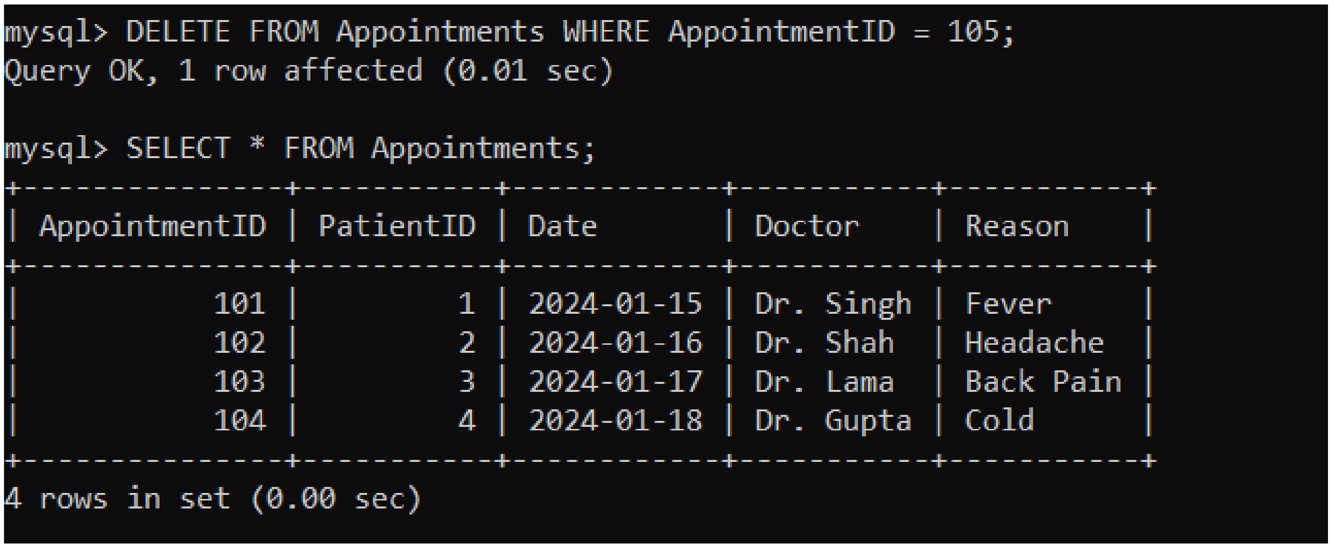
Updates the email of the patient with PatientID 1.



```
mysql> UPDATE Patients SET Email = 'anil_shrestha@gmail.com' WHERE PatientID = 1;
Query OK, 1 row affected (0.01 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql> SELECT * FROM Patients;
+-----------+---------------+-----+--------+-------------------------+
| PatientID | Name          | Age | Gender | Email                   |
+-----------+---------------+-----+--------+-------------------------+
|         1 | Anil Shrestha |  35 | Male   | anil_shrestha@gmail.com |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com          |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com        |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com        |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com         |
+-----------+---------------+-----+--------+-------------------------+
5 rows in set (0.01 sec)
```

5.2) Delete Data:

Removes the appointment with AppointmentID 105.

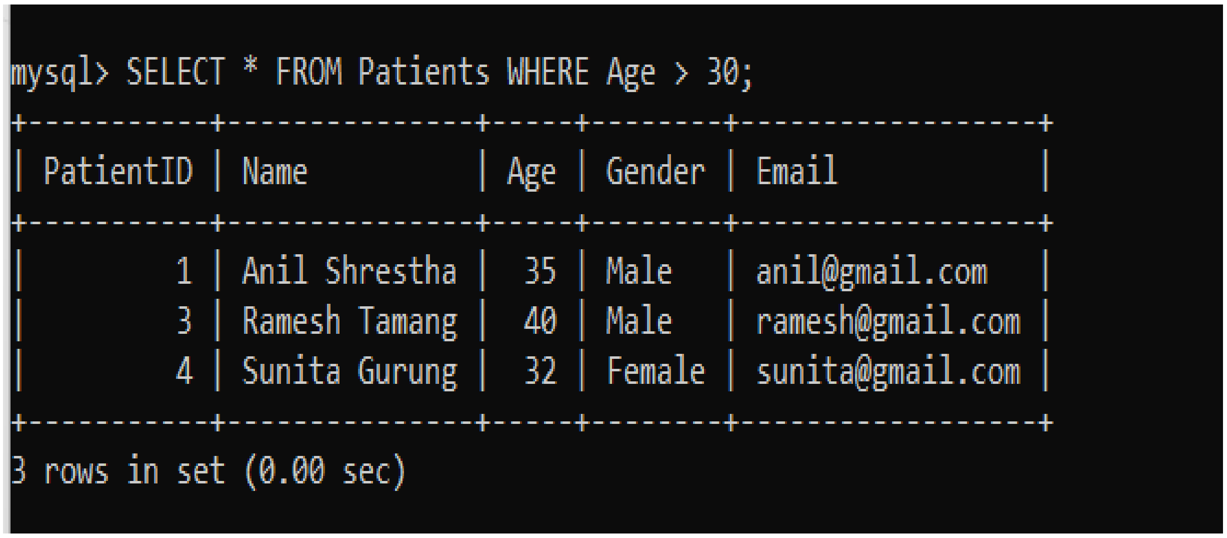


```
mysql> DELETE FROM Appointments WHERE AppointmentID = 105;
Query OK, 1 row affected (0.01 sec)

mysql> SELECT * FROM Appointments;
+---------------+-----------+------------+-----------+-----------+
| AppointmentID | PatientID | Date       | Doctor    | Reason    |
+---------------+-----------+------------+-----------+-----------+
|           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
+---------------+-----------+------------+-----------+-----------+
4 rows in set (0.00 sec)
```

5.3) Relational Algebra Operations

a. Selection (σ):

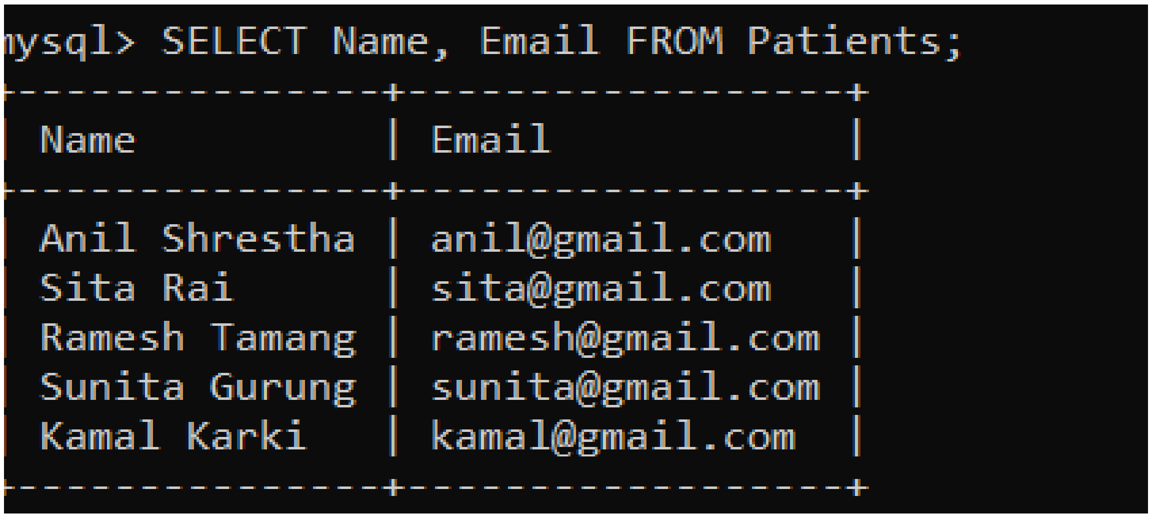
Retrieves all patients aged over 30.



```
mysql> SELECT * FROM Patients WHERE Age > 30;
+-----------+---------------+-----+--------+------------------+
| PatientID | Name          | Age | Gender | Email            |
+-----------+---------------+-----+--------+------------------+
|         1 | Anil Shrestha |  35 | Male   | anil@gmail.com   |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |
+-----------+---------------+-----+--------+------------------+
3 rows in set (0.00 sec)
```

b. Projection (π):

Displays only the names and emails of patients.

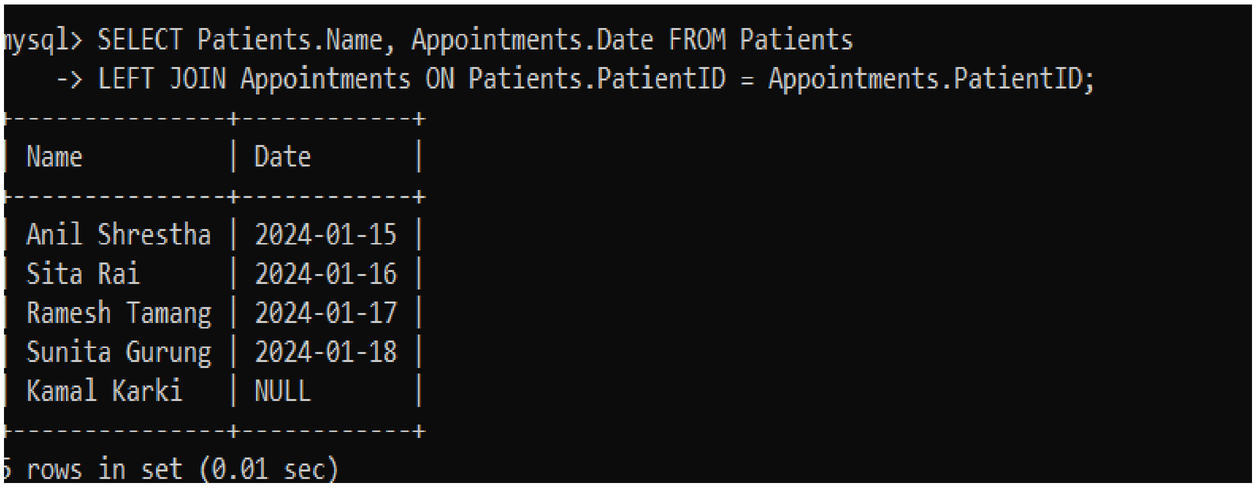


```
ysql> SELECT Name, Email FROM Patients;
---------------+------------------+
 Name          | Email            |
---------------+------------------+
 Anil Shrestha | anil@gmail.com   |
 Sita Rai      | sita@gmail.com   |
 Ramesh Tamang | ramesh@gmail.com |
 Sunita Gurung | sunita@gmail.com |
 Kamal Karki   | kamal@gmail.com  |
---------------+------------------+
```

5.4) Joins:

Left join:

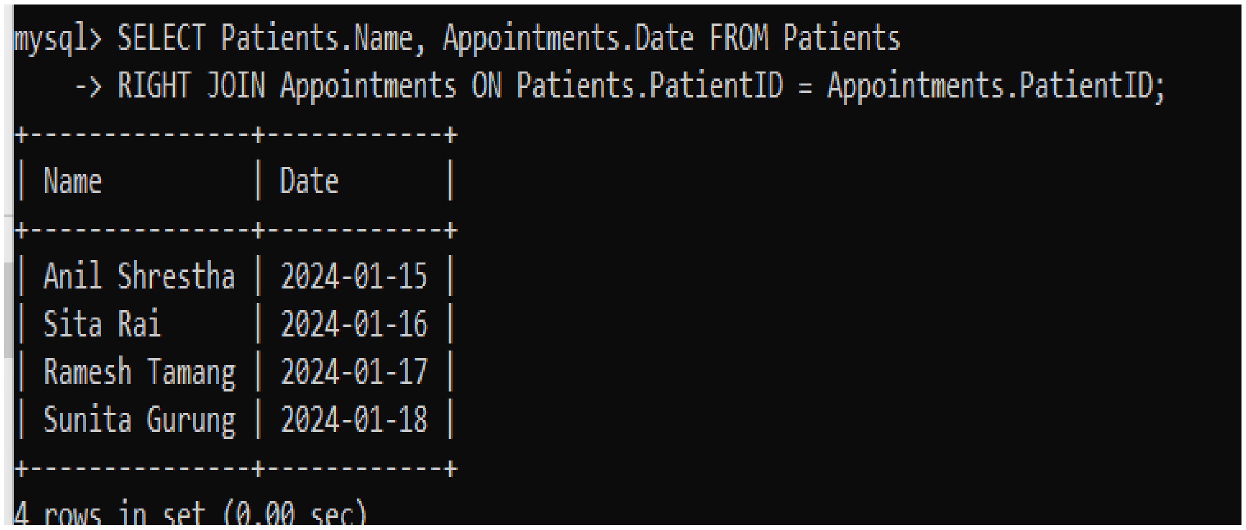
The LEFT JOIN retrieves all records from the Patients table and the matching rows from the Appointments table. If there is no match, NULL values are shown for the Appointments table columns.



```
ysql> SELECT Patients.Name, Appointments.Date FROM Patients
    -> LEFT JOIN Appointments ON Patients.PatientID = Appointments.PatientID;
+---------------+------------+
| Name          | Date       |
+---------------+------------+
| Anil Shrestha | 2024-01-15 |
| Sita Rai      | 2024-01-16 |
| Ramesh Tamang | 2024-01-17 |
| Sunita Gurung | 2024-01-18 |
| Kamal Karki   | NULL       |
+---------------+------------+
 rows in set (0.01 sec)
```

Right join:

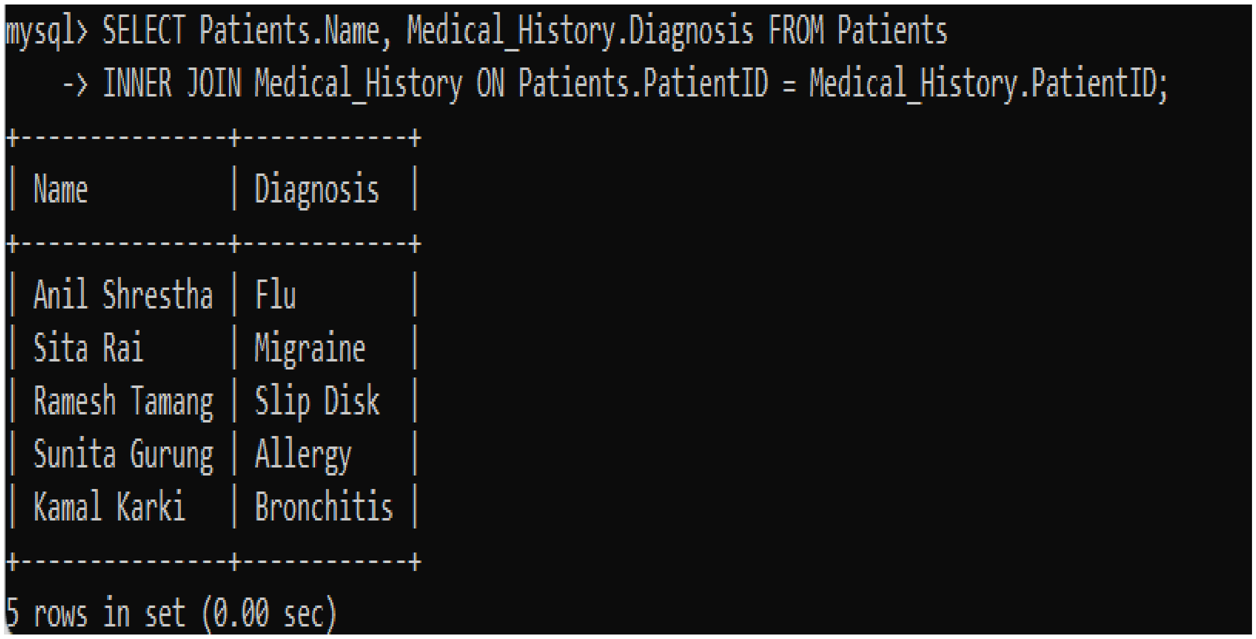
The RIGHT JOIN retrieves all records from the Appointments table and the matching rows from the Patients table. If there is no match, NULL values are shown for the Patients table columns.



```
mysql> SELECT Patients.Name, Appointments.Date FROM Patients
    -> RIGHT JOIN Appointments ON Patients.PatientID = Appointments.PatientID;
+---------------+------------+
| Name          | Date       |
+---------------+------------+
| Anil Shrestha | 2024-01-15 |
| Sita Rai      | 2024-01-16 |
| Ramesh Tamang | 2024-01-17 |
| Sunita Gurung | 2024-01-18 |
+---------------+------------+
4 rows in set (0.00 sec)
```

Inner join:

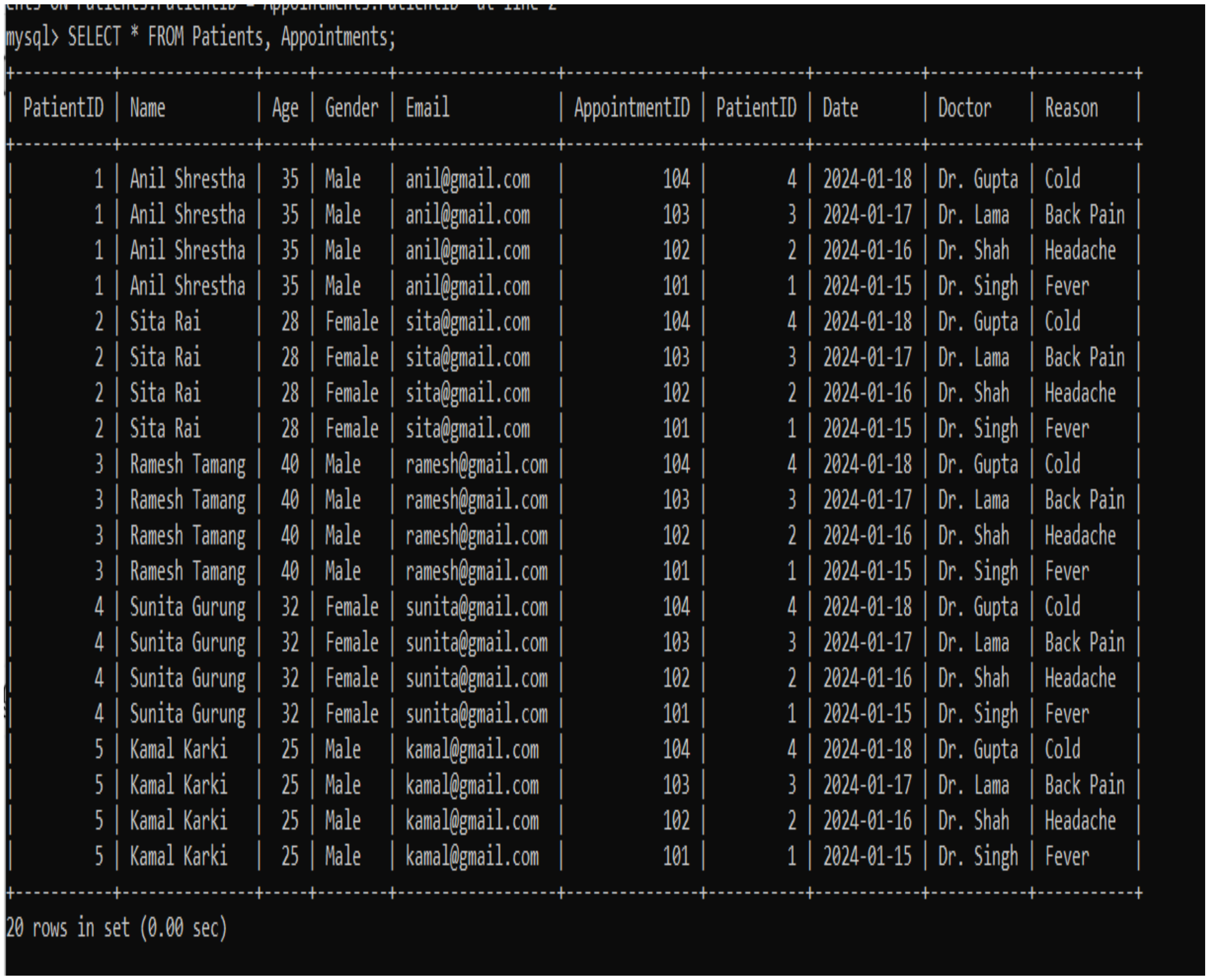
The INNER JOIN returns only the rows where there is a match between the Patients and Medical_History tables based on the PatientID.



```
mysql> SELECT Patients.Name, Medical_History.Diagnosis FROM Patients
    -> INNER JOIN Medical_History ON Patients.PatientID = Medical_History.PatientID;
+---------------+------------+
| Name          | Diagnosis  |
+---------------+------------+
| Anil Shrestha | Flu        |
| Sita Rai      | Migraine   |
| Ramesh Tamang | Slip Disk  |
| Sunita Gurung | Allergy    |
| Kamal Karki   | Bronchitis |
+---------------+------------+
5 rows in set (0.00 sec)
```

5.5) Cartesian Product (×):

Combines every row from the Patients table with every row from the Appointments table.

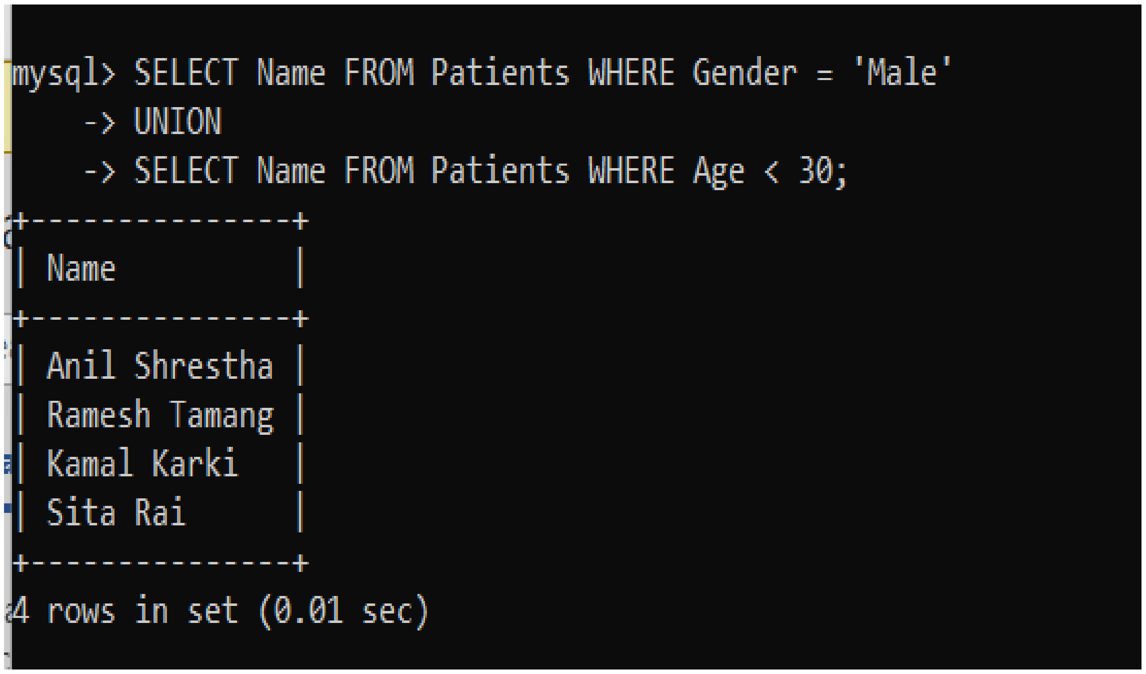


```
ents ON Patients.PatientID = Appointments.PatientID' at line 2
mysql> SELECT * FROM Patients, Appointments;
+-----------+---------------+-----+--------+------------------+---------------+-----------+------------+-----------+-----------+
| PatientID | Name          | Age | Gender | Email            | AppointmentID | PatientID | Date       | Doctor    | Reason    |
+-----------+---------------+-----+--------+------------------+---------------+-----------+------------+-----------+-----------+
|         1 | Anil Shrestha |  35 | Male   | anil@gmail.com   |           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|         1 | Anil Shrestha |  35 | Male   | anil@gmail.com   |           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|         1 | Anil Shrestha |  35 | Male   | anil@gmail.com   |           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|         1 | Anil Shrestha |  35 | Male   | anil@gmail.com   |           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com   |           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com   |           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com   |           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com   |           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com  |           104 |         4 | 2024-01-18 | Dr. Gupta | Cold      |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com  |           103 |         3 | 2024-01-17 | Dr. Lama  | Back Pain |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com  |           102 |         2 | 2024-01-16 | Dr. Shah  | Headache  |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com  |           101 |         1 | 2024-01-15 | Dr. Singh | Fever     |
+-----------+---------------+-----+--------+------------------+---------------+-----------+------------+-----------+-----------+
20 rows in set (0.00 sec)
```

5.6) Set Operations:

UNION:

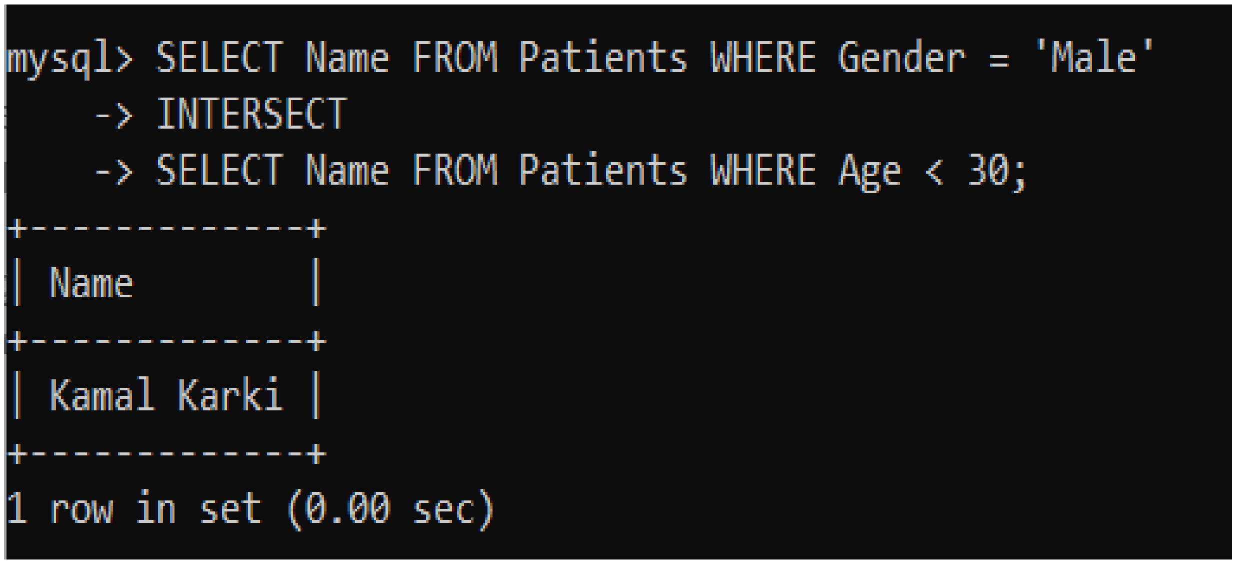
The query returns a distinct list of patient names who are either male or under the age of 3



```
mysql> SELECT Name FROM Patients WHERE Gender = 'Male'
    -> UNION
    -> SELECT Name FROM Patients WHERE Age < 30;
+---------------+
| Name          |
+---------------+
| Anil Shrestha |
| Ramesh Tamang |
| Kamal Karki   |
| Sita Rai      |
+---------------+
4 rows in set (0.01 sec)
```

INTERSECTION:

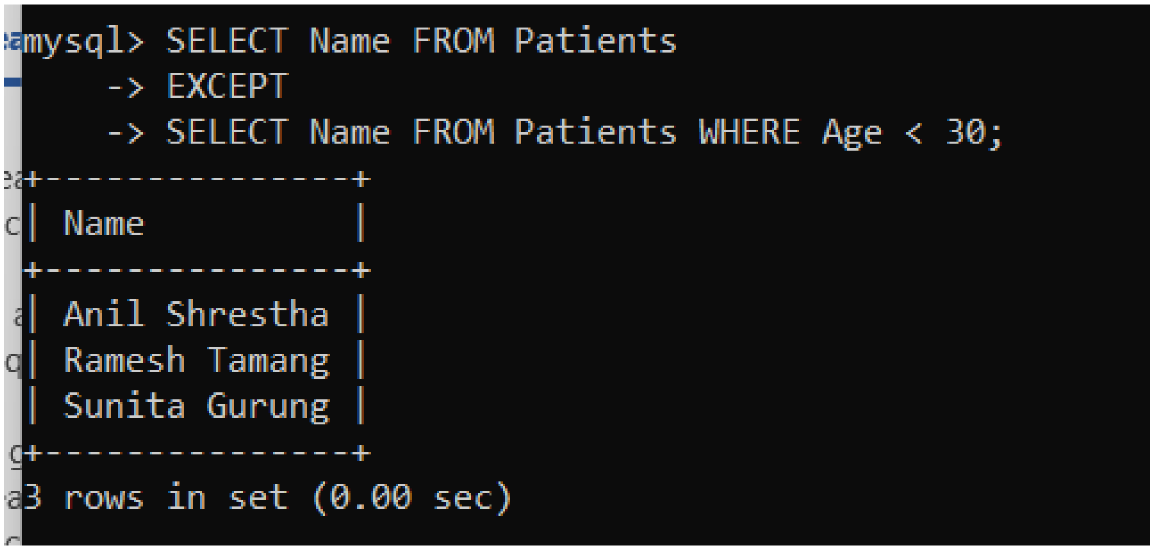
The query returns a list of patient names who are both male and under 30, as the INTERSECT operator keeps only the names present in both queries.



```
mysql> SELECT Name FROM Patients WHERE Gender = 'Male'
    -> INTERSECT
    -> SELECT Name FROM Patients WHERE Age < 30;
+-------------+
| Name        |
+-------------+
| Kamal Karki |
+-------------+
1 row in set (0.00 sec)
```

SET DIFFERENCE (EXCEPT):

The query returns the names of patients who are 30 or older, as the EXCEPT operator excludes the names of patients under 30 from the full list of patient names.



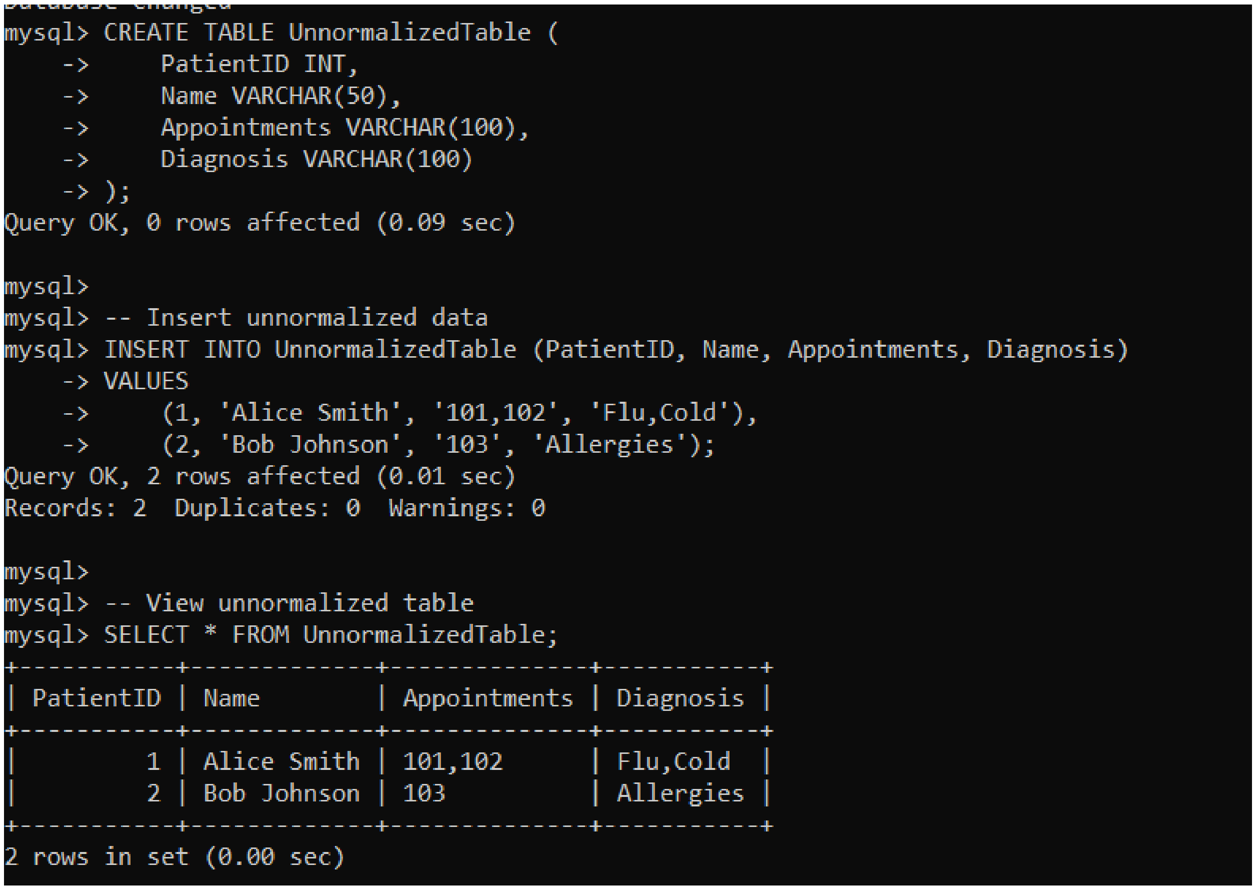
```
mysql> SELECT Name FROM Patients
    -> EXCEPT
    -> SELECT Name FROM Patients WHERE Age < 30;
+----------------+
| Name           |
+----------------+
| Anil Shrestha  |
| Ramesh Tamang  |
| Sunita Gurung  |
+----------------+
3 rows in set (0.00 sec)
```

6. Normalization Technique:

Normalization is a systematic approach to organizing data in a database to reduce redundancy and improve data integrity. Below, the techniques for normalizing a database are discussed, leading to different normal forms:

Unnormalized Table (UNF)

An unnormalized table contains data that may have repeating groups or arrays. Such a table does not satisfy the principles of relational database design.



```
mysql> CREATE TABLE UnnormalizedTable (
    ->     PatientID INT,
    ->     Name VARCHAR(50),
    ->     Appointments VARCHAR(100),
    ->     Diagnosis VARCHAR(100)
    -> );
Query OK, 0 rows affected (0.09 sec)

mysql>
mysql> -- Insert unnormalized data
mysql> INSERT INTO UnnormalizedTable (PatientID, Name, Appointments, Diagnosis)
    -> VALUES
    ->     (1, 'Alice Smith', '101,102', 'Flu,Cold'),
    ->     (2, 'Bob Johnson', '103', 'Allergies');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- View unnormalized table
mysql> SELECT * FROM UnnormalizedTable;
+-----------+-------------+--------------+-----------+
| PatientID | Name        | Appointments | Diagnosis |
+-----------+-------------+--------------+-----------+
|         1 | Alice Smith | 101,102      | Flu,Cold  |
|         2 | Bob Johnson | 103          | Allergies |
+-----------+-------------+--------------+-----------+
2 rows in set (0.00 sec)
```

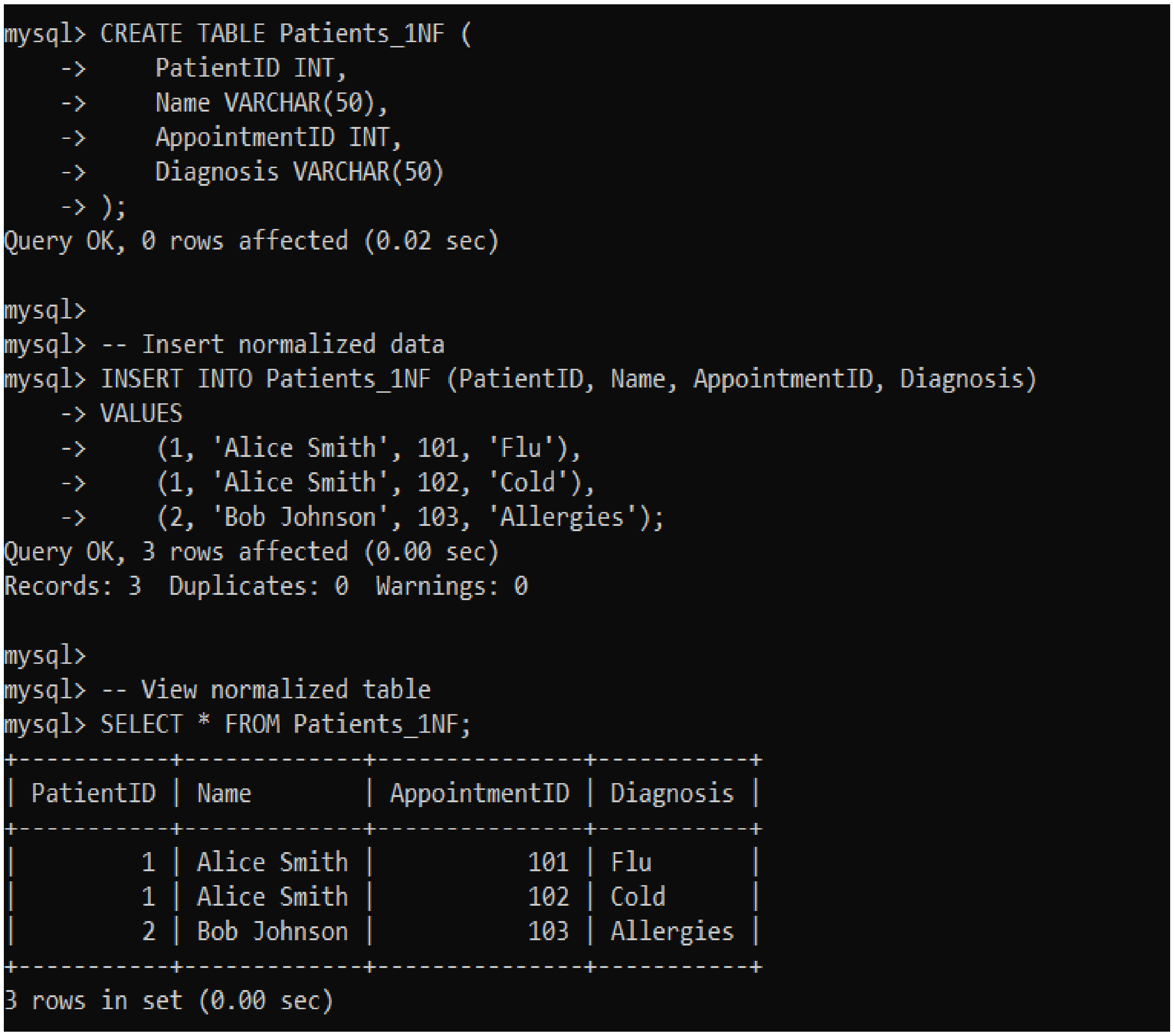
The columns Appointments and Diagnosis store multi-valued attributes (e.g., '101,102' for appointments and 'Flu, Cold' for diagnoses).

There are repeating groups in these columns, violating the rule of atomicity.

Data is not structured to follow relational database principles.

First Normal Form (1NF)

Each column must contain atomic values (no multi-valued attributes or arrays). Each row must be unique, identified by a primary key. There should be no repeating groups.



```
mysql> CREATE TABLE Patients_1NF (
    ->     PatientID INT,
    ->     Name VARCHAR(50),
    ->     AppointmentID INT,
    ->     Diagnosis VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Insert normalized data
mysql> INSERT INTO Patients_1NF (PatientID, Name, AppointmentID, Diagnosis)
    -> VALUES
    ->     (1, 'Alice Smith', 101, 'Flu'),
    ->     (1, 'Alice Smith', 102, 'Cold'),
    ->     (2, 'Bob Johnson', 103, 'Allergies');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0

mysql>
mysql> -- View normalized table
mysql> SELECT * FROM Patients_1NF;
+-----------+-------------+---------------+-----------+
| PatientID | Name        | AppointmentID | Diagnosis |
+-----------+-------------+---------------+-----------+
|         1 | Alice Smith |           101 | Flu       |
|         1 | Alice Smith |           102 | Cold      |
|         2 | Bob Johnson |           103 | Allergies |
+-----------+-------------+---------------+-----------+
3 rows in set (0.00 sec)
```

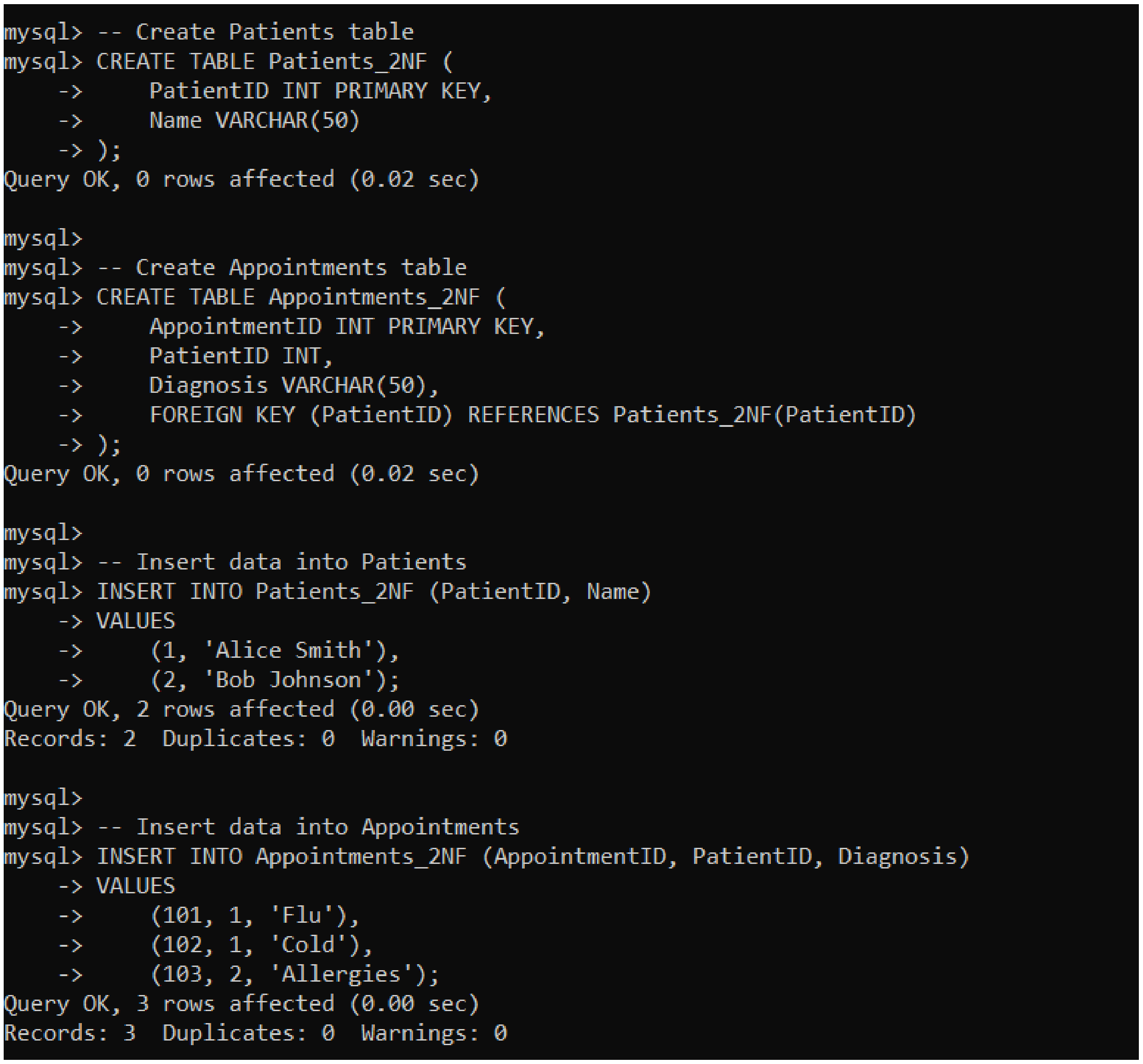
Repeating groups and multi-valued attributes from UNF are eliminated.

Each column contains atomic values (e.g., one appointment per row, one diagnosis per row).

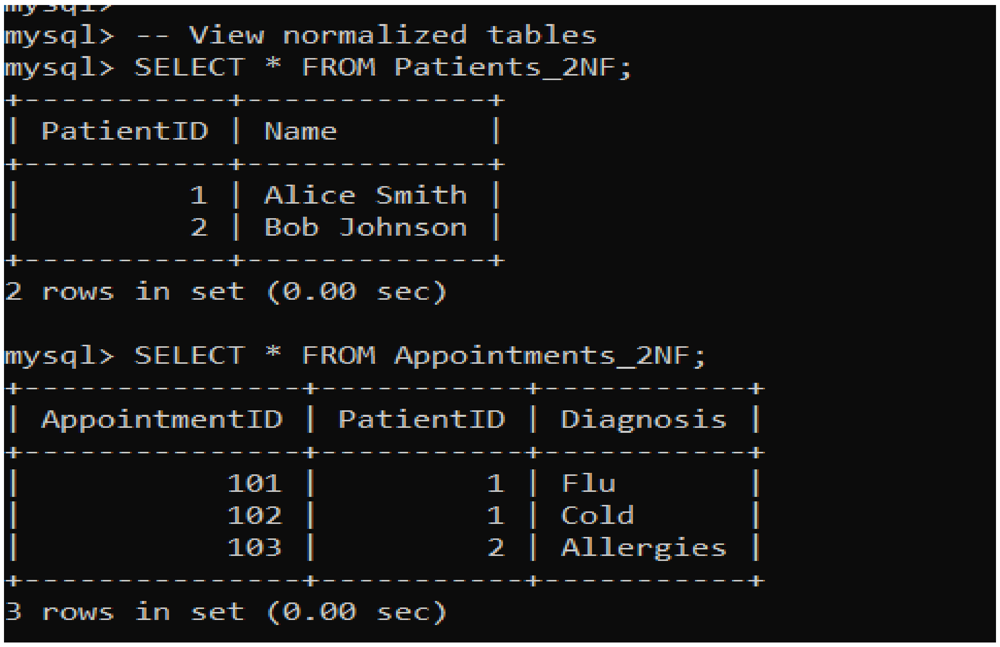
Rows are unique and identifiable, though the table does not yet have a defined primary key.

Second Normal Form (2NF)

Must satisfy all requirements of 1NF. Eliminate partial dependency, ensuring that all non-prime attributes depend on the entire primary key, not just part of it.



```
mysql> -- Create Patients table
mysql> CREATE TABLE Patients_2NF (
    ->     PatientID INT PRIMARY KEY,
    ->     Name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Create Appointments table
mysql> CREATE TABLE Appointments_2NF (
    ->     AppointmentID INT PRIMARY KEY,
    ->     PatientID INT,
    ->     Diagnosis VARCHAR(50),
    ->     FOREIGN KEY (PatientID) REFERENCES Patients_2NF(PatientID)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Insert data into Patients
mysql> INSERT INTO Patients_2NF (PatientID, Name)
    -> VALUES
    ->     (1, 'Alice Smith'),
    ->     (2, 'Bob Johnson');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Appointments
mysql> INSERT INTO Appointments_2NF (AppointmentID, PatientID, Diagnosis)
    -> VALUES
    ->     (101, 1, 'Flu'),
    ->     (102, 1, 'Cold'),
    ->     (103, 2, 'Allergies');
Query OK, 3 rows affected (0.00 sec)
Records: 3  Duplicates: 0  Warnings: 0
```



```
mysql> -- View normalized tables
mysql> SELECT * FROM Patients_2NF;
+-----------+-------------+
| PatientID | Name        |
+-----------+-------------+
|         1 | Alice Smith |
|         2 | Bob Johnson |
+-----------+-------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Appointments_2NF;
+---------------+-----------+-----------+
| AppointmentID | PatientID | Diagnosis |
+---------------+-----------+-----------+
|           101 |         1 | Flu       |
|           102 |         1 | Cold      |
|           103 |         2 | Allergies |
+---------------+-----------+-----------+
3 rows in set (0.00 sec)
```

The table satisfies all 1NF rules.

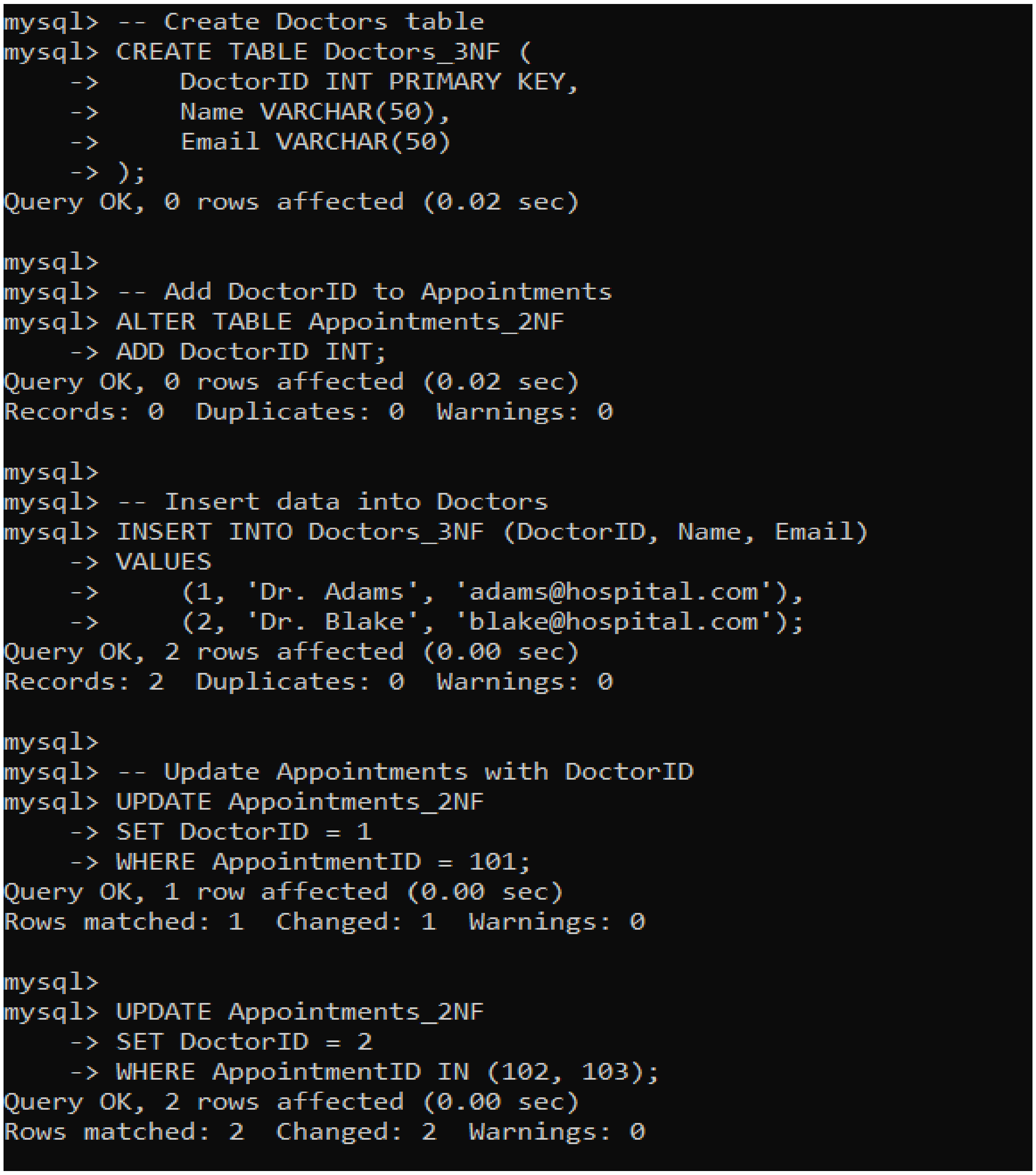
Partial dependencies are removed by splitting the table:

- Name depends only on PatientID, so it is moved to a separate Patients table.
- Diagnosis depends on both AppointmentID and PatientID, so it remains in the Appointments table.

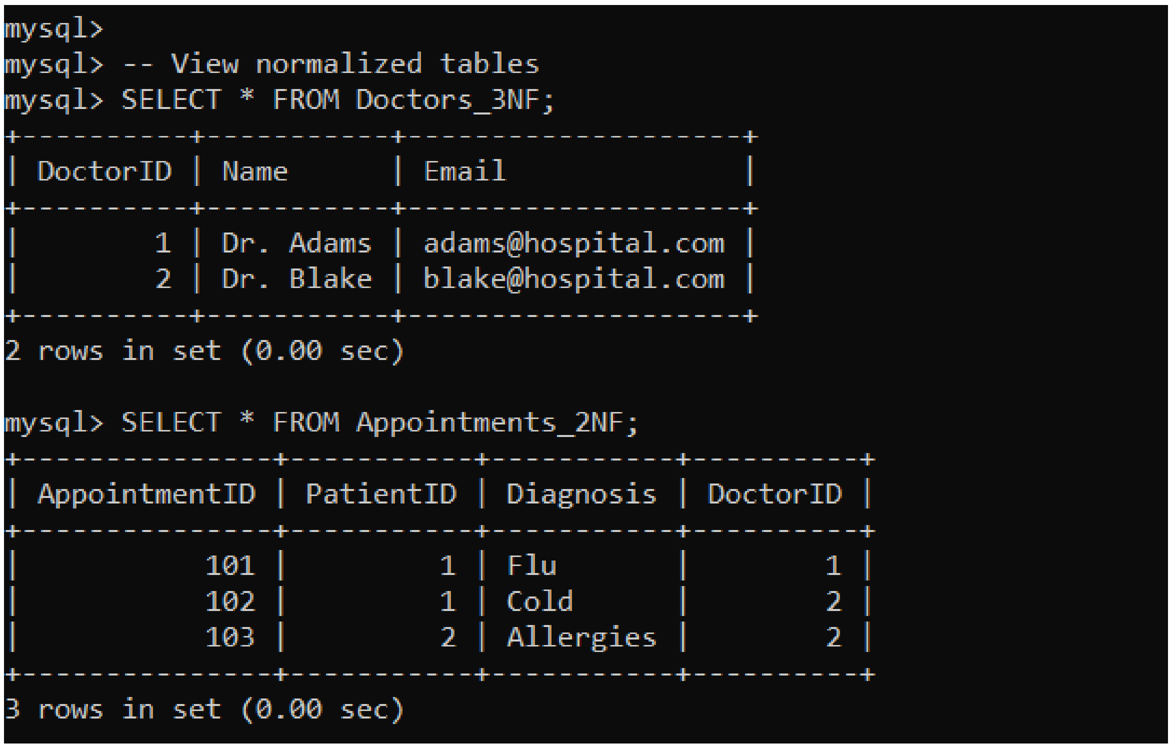
Each non-prime attribute depends on the entire primary key (in the composite case).

Third Normal Form (3NF)

Must satisfy all requirements of 2NF. Eliminate transitive dependency; non-prime attributes should depend only on the primary key and not on other non-prime attributes.



```
mysql> -- Create Doctors table
mysql> CREATE TABLE Doctors_3NF (
    ->     DoctorID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     Email VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Add DoctorID to Appointments
mysql> ALTER TABLE Appointments_2NF
    -> ADD DoctorID INT;
Query OK, 0 rows affected (0.02 sec)
Records: 0  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Insert data into Doctors
mysql> INSERT INTO Doctors_3NF (DoctorID, Name, Email)
    -> VALUES
    ->     (1, 'Dr. Adams', 'adams@hospital.com'),
    ->     (2, 'Dr. Blake', 'blake@hospital.com');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> -- Update Appointments with DoctorID
mysql> UPDATE Appointments_2NF
    -> SET DoctorID = 1
    -> WHERE AppointmentID = 101;
Query OK, 1 row affected (0.00 sec)
Rows matched: 1  Changed: 1  Warnings: 0

mysql>
mysql> UPDATE Appointments_2NF
    -> SET DoctorID = 2
    -> WHERE AppointmentID IN (102, 103);
Query OK, 2 rows affected (0.00 sec)
Rows matched: 2  Changed: 2  Warnings: 0
```



```
mysql>
mysql> -- View normalized tables
mysql> SELECT * FROM Doctors_3NF;
+----------+-----------+--------------------+
| DoctorID | Name      | Email              |
+----------+-----------+--------------------+
|        1 | Dr. Adams | adams@hospital.com |
|        2 | Dr. Blake | blake@hospital.com |
+----------+-----------+--------------------+
2 rows in set (0.00 sec)

mysql> SELECT * FROM Appointments_2NF;
+---------------+-----------+-----------+----------+
| AppointmentID | PatientID | Diagnosis | DoctorID |
+---------------+-----------+-----------+----------+
|           101 |         1 | Flu       |        1 |
|           102 |         1 | Cold      |        2 |
|           103 |         2 | Allergies |        2 |
+---------------+-----------+-----------+----------+
3 rows in set (0.00 sec)
```

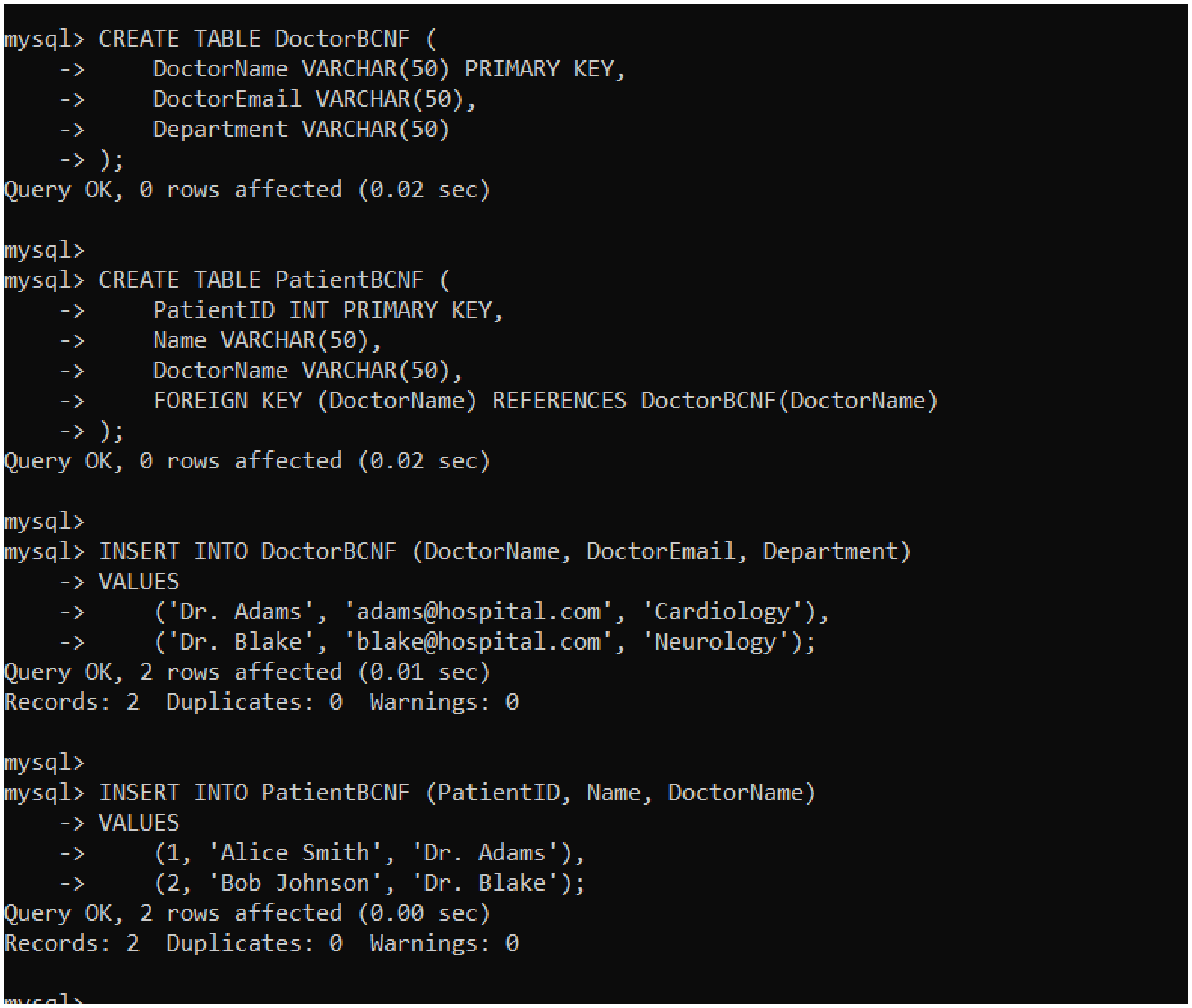
Satisfies all 2NF rules.

Removes transitive dependencies:

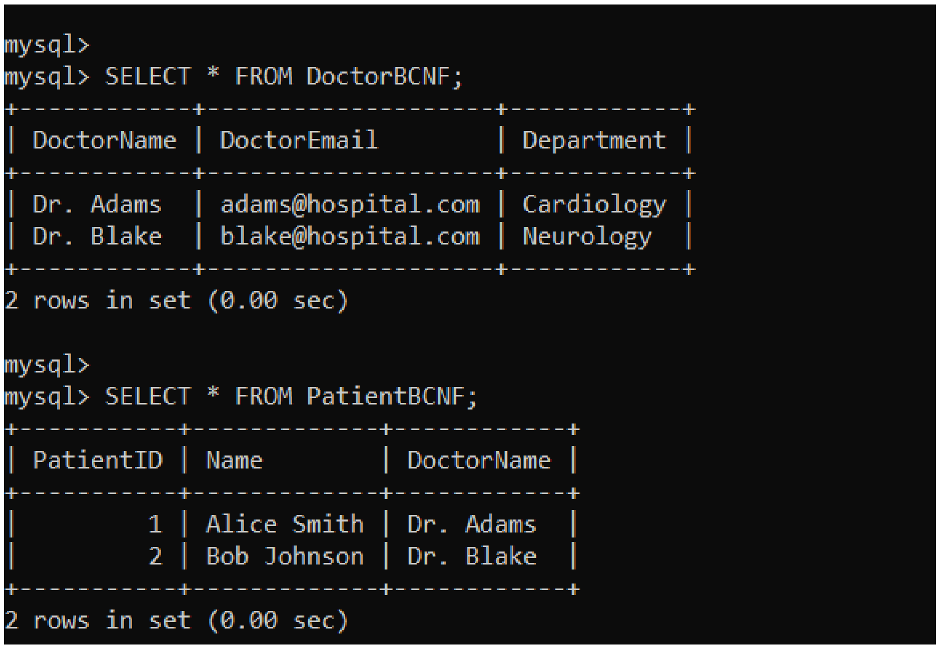
- Doctor_Email depends on Doctor rather than directly on the primary key of Appointments.
- The Doctors table separates this dependency, and DoctorID is used as a foreign key in the Appointments table.

Boyce-Codd Normal Form (BCNF)

Must satisfy all requirements of 3NF. Every determinant (an attribute on which other attributes depend) must be a candidate key.



```
mysql> CREATE TABLE DoctorBCNF (
    ->     DoctorName VARCHAR(50) PRIMARY KEY,
    ->     DoctorEmail VARCHAR(50),
    ->     Department VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> CREATE TABLE PatientBCNF (
    ->     PatientID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     DoctorName VARCHAR(50),
    ->     FOREIGN KEY (DoctorName) REFERENCES DoctorBCNF(DoctorName)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> INSERT INTO DoctorBCNF (DoctorName, DoctorEmail, Department)
    -> VALUES
    ->     ('Dr. Adams', 'adams@hospital.com', 'Cardiology'),
    ->     ('Dr. Blake', 'blake@hospital.com', 'Neurology');
Query OK, 2 rows affected (0.01 sec)
Records: 2  Duplicates: 0  Warnings: 0

mysql>
mysql> INSERT INTO PatientBCNF (PatientID, Name, DoctorName)
    -> VALUES
    ->     (1, 'Alice Smith', 'Dr. Adams'),
    ->     (2, 'Bob Johnson', 'Dr. Blake');
Query OK, 2 rows affected (0.00 sec)
Records: 2  Duplicates: 0  Warnings: 0
```



```
mysql>
mysql> SELECT * FROM DoctorBCNF;
+------------+--------------------+------------+
| DoctorName | DoctorEmail        | Department |
+------------+--------------------+------------+
| Dr. Adams  | adams@hospital.com | Cardiology |
| Dr. Blake  | blake@hospital.com | Neurology  |
+------------+--------------------+------------+
2 rows in set (0.00 sec)

mysql>
mysql> SELECT * FROM PatientBCNF;
+-----------+-------------+------------+
| PatientID | Name        | DoctorName |
+-----------+-------------+------------+
|         1 | Alice Smith | Dr. Adams  |
|         2 | Bob Johnson | Dr. Blake  |
+-----------+-------------+------------+
2 rows in set (0.00 sec)
```

Satisfies all 3NF rules.

In the DoctorBCNF table, DoctorName is the primary key, and all other attributes (DoctorEmail and Department) depend on it. Since DoctorName is the candidate key, the table satisfies BCNF.

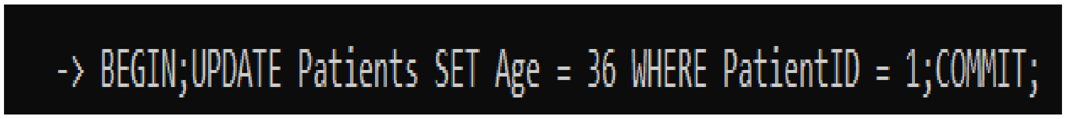
In the PatientBCNF table, PatientID is the primary key, and both Name and DoctorName depend on it. The foreign key DoctorName references the DoctorBCNF table, but still depends on the primary key PatientID, ensuring the table meets BCNF.

Thus, both tables meet the BCNF condition where every determinant is a candidate key.

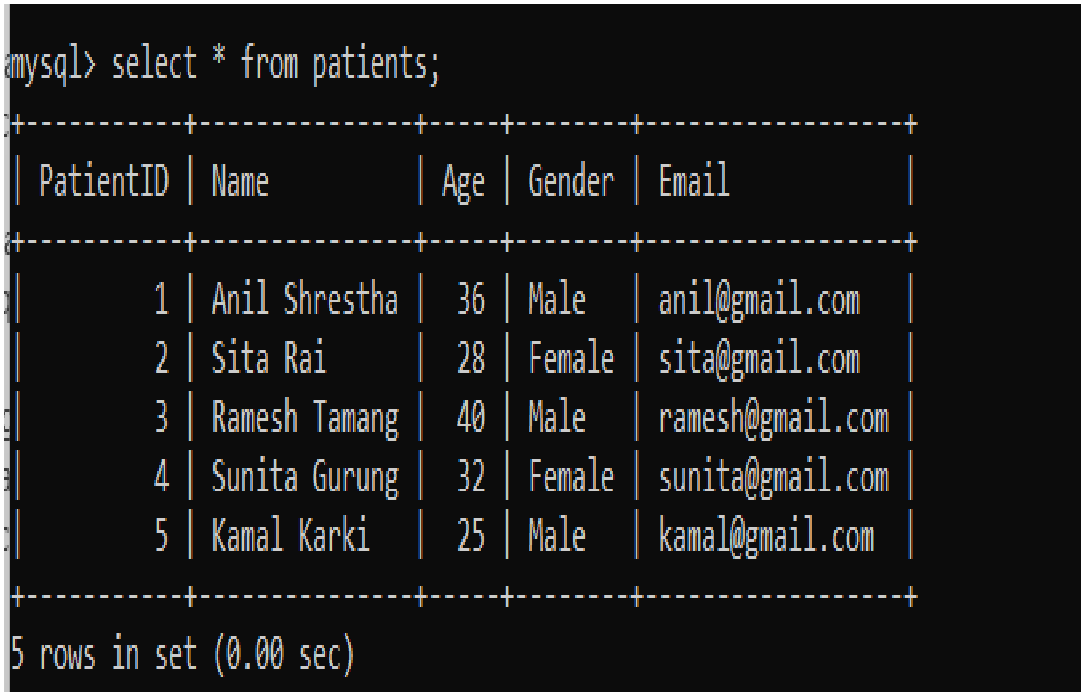
7. Transaction Management

a. Commit:

Commits the transaction to save the updated age.



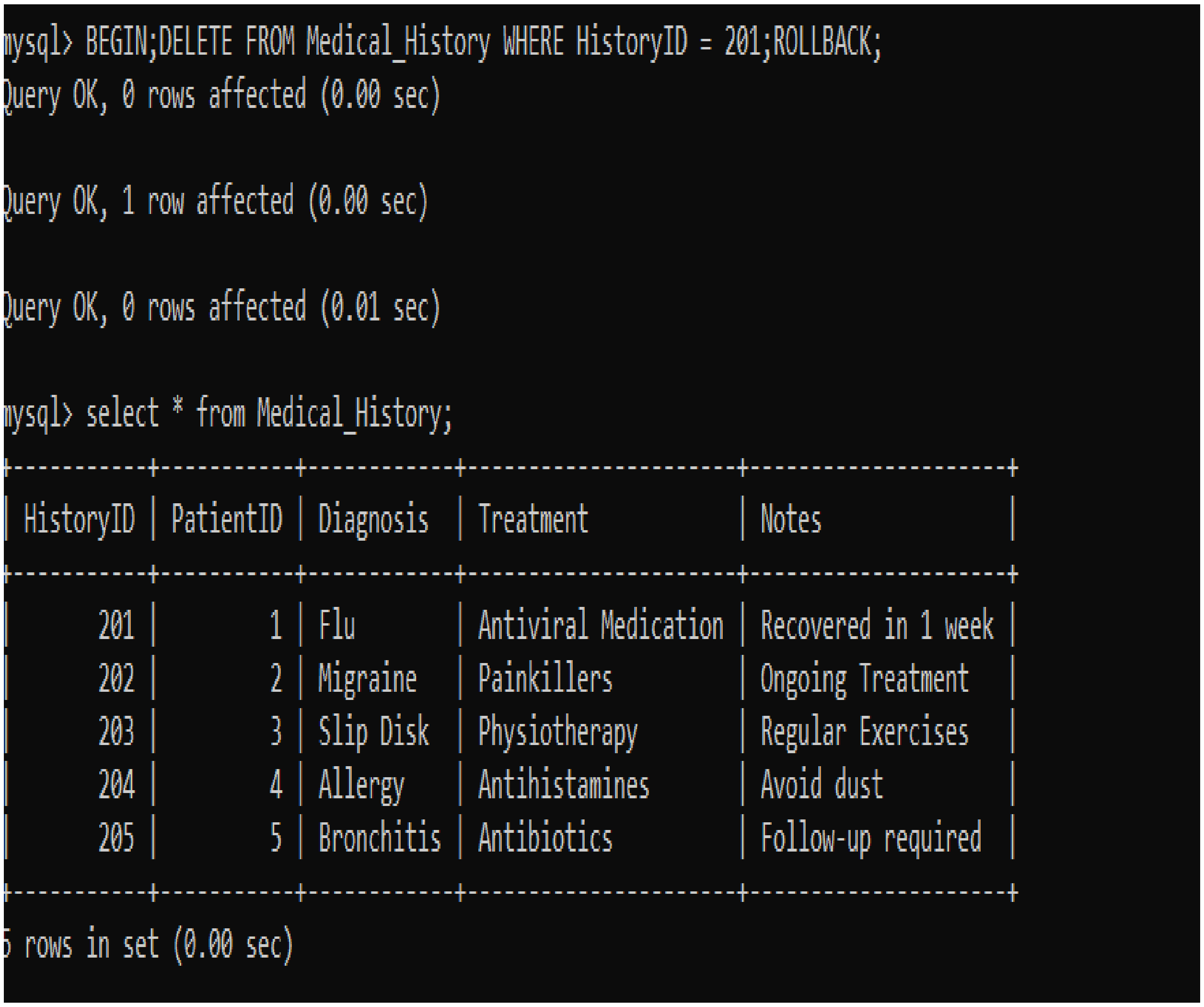
```
    -> BEGIN;UPDATE Patients SET Age = 36 WHERE PatientID = 1;COMMIT;
```



```
mysql> select * from patients;
+-----------+---------------+-----+--------+------------------+
| PatientID | Name          | Age | Gender | Email            |
+-----------+---------------+-----+--------+------------------+
|         1 | Anil Shrestha |  36 | Male   | anil@gmail.com   |
|         2 | Sita Rai      |  28 | Female | sita@gmail.com   |
|         3 | Ramesh Tamang |  40 | Male   | ramesh@gmail.com |
|         4 | Sunita Gurung |  32 | Female | sunita@gmail.com |
|         5 | Kamal Karki   |  25 | Male   | kamal@gmail.com  |
+-----------+---------------+-----+--------+------------------+
5 rows in set (0.00 sec)
```

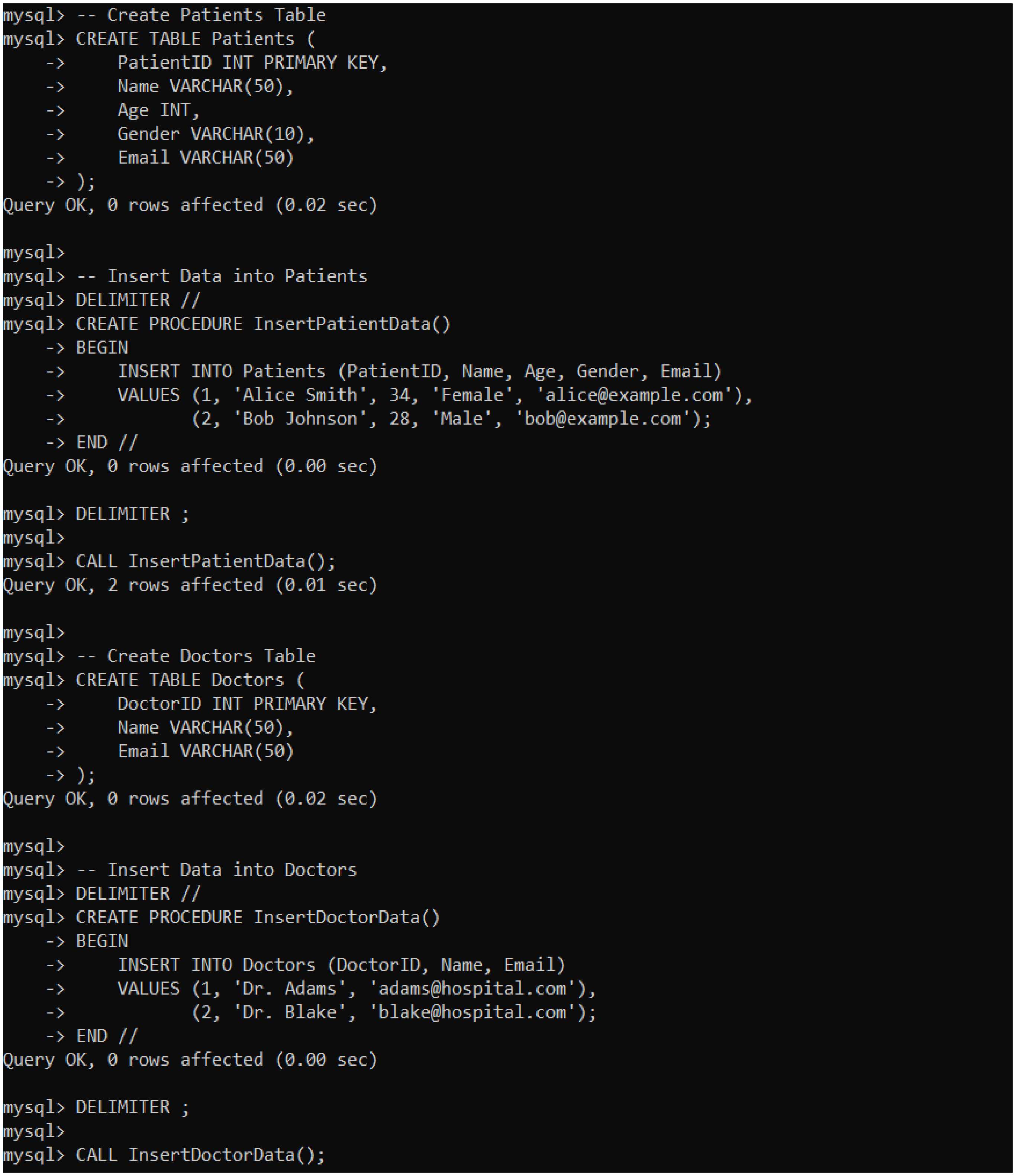
b. Rollback:

This demonstrates a rollback operation. The DELETE statement removes a record from the Medical_History table, but the ROLLBACK command undoes this deletion. Rollback is typically used in scenarios where an error or inconsistency is detected during a transaction, ensuring that the database remains in a consistent state.



```
nysql> BEGIN;DELETE FROM Medical_History WHERE HistoryID = 201;ROLLBACK;
Query OK, 0 rows affected (0.00 sec)

Query OK, 1 row affected (0.00 sec)

Query OK, 0 rows affected (0.01 sec)

nysql> select * from Medical_History;
+-----------+-----------+------------+---------------------+--------------------+
| HistoryID | PatientID | Diagnosis  | Treatment           | Notes              |
+-----------+-----------+------------+---------------------+--------------------+
|       201 |         1 | Flu        | Antiviral Medication | Recovered in 1 week |
|       202 |         2 | Migraine   | Painkillers         | Ongoing Treatment  |
|       203 |         3 | Slip Disk  | Physiotherapy       | Regular Exercises  |
|       204 |         4 | Allergy    | Antihistamines      | Avoid dust         |
|       205 |         5 | Bronchitis | Antibiotics         | Follow-up required |
+-----------+-----------+------------+---------------------+--------------------+
5 rows in set (0.00 sec)
```

8. Stored procedure:



```
mysql> -- Create Patients Table
mysql> CREATE TABLE Patients (
    ->     PatientID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     Age INT,
    ->     Gender VARCHAR(10),
    ->     Email VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Insert Data into Patients
mysql> DELIMITER //
mysql> CREATE PROCEDURE InsertPatientData()
    -> BEGIN
    ->     INSERT INTO Patients (PatientID, Name, Age, Gender, Email)
    ->     VALUES (1, 'Alice Smith', 34, 'Female', 'alice@example.com'),
    ->            (2, 'Bob Johnson', 28, 'Male', 'bob@example.com');
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
mysql> CALL InsertPatientData();
Query OK, 2 rows affected (0.01 sec)

mysql>
mysql> -- Create Doctors Table
mysql> CREATE TABLE Doctors (
    ->     DoctorID INT PRIMARY KEY,
    ->     Name VARCHAR(50),
    ->     Email VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.02 sec)

mysql>
mysql> -- Insert Data into Doctors
mysql> DELIMITER //
mysql> CREATE PROCEDURE InsertDoctorData()
    -> BEGIN
    ->     INSERT INTO Doctors (DoctorID, Name, Email)
    ->     VALUES (1, 'Dr. Adams', 'adams@hospital.com'),
    ->            (2, 'Dr. Blake', 'blake@hospital.com');
    -> END //
Query OK, 0 rows affected (0.00 sec)

mysql> DELIMITER ;
mysql>
mysql> CALL InsertDoctorData();
```

The code first creates two tables (Patients and Doctors) to store data for patients and doctors.

Two stored procedures (InsertPatientData and InsertDoctorData) are defined to insert multiple rows of data into the respective tables.

The stored procedures are executed using CALL to insert sample data into both tables.

9. Conclusion

This project demonstrated the creation and manipulation of a relational database using SQL. By performing various operations, we explored the power of DBMS in managing complex datasets. The queries executed highlight SQL's capabilities in data retrieval, modification, relational algebra, transaction, normalization techniques, etc.