DELHI TECHNOLOGICAL UIVERSITY

NAME: **Ishaan Gupta**

ROLL NO: **23/MC/68**

BRANCH: **MCE**

SUBJECT: **DATABASE MANAGEMENT SYSTEM**

SUBJECT CODE:MC209

SUBMITTED TO: **Prof. Brij Kishore Tyagi & Miss Shweta Kalson**

No. Of Experiments	Name of Experiment	Date	Signature
1	Synopsis of the Project, along with its E-R Diagram	12/08/2024	
2	Implement the following DDL statements: a. Create b. Create table with constraints (NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY) c. Alter Table: i. Add column ii. Drop column iii. Add/drop constraint iv. Rename column d. Drop Table	2/09/24	
3	Implement the following DML statements: a. Insert b. Update c. Delete d. Truncate	9/09/24	
4	Implement the following SELECT statements: a. Simple SELECT statement b. Where clause+ IN/NOT IN c. Aggregate functions d. Group by + Having e. Order by f. Views g. Inbuilt Functions (e.g., Date)	16/09/24	
5	Implement and perform Nested Queries along with Joins (Inner join, Outer join, Left join, Right join)	30/09/24	
6	Introduction to PL/SQL. Create a PL/SQL block and implement the following: a. Variables	7/10/24	

	b. Packages c. Procedures d. Functions		
7	Perform Exception handling in a PL/SQL block.	7/10/24	
8	Project Report with Code and Screenshots	14/10/24	
9	Implement Triggers in SQL.	14/10/24	
10	Implement the following transaction statements: a. Commit b. Rollback c. Savepoint	21/10/24	

Practical -1

Synopsis of the Project, along with its E-R Diagram

Synopsis of Hospital Management System

Project Title: Hospital Management System (HMS)

Objective: The primary objective of this Hospital Management System (HMS) is to create a comprehensive database solution that manages the daily operations and administrative tasks of a hospital. This system aims to improve the efficiency and accuracy of various processes, including patient registration, appointment scheduling, medical records management, billing, and staff management.

Introduction: In the modern healthcare environment, hospitals face challenges such as managing patient data, scheduling appointments, and ensuring seamless communication among healthcare providers. The Hospital Management System seeks to address these challenges by providing a centralized platform that streamlines operations, enhances patient care, and optimizes resource management.

Key Features:

1. **Patient Registration:**
    - Collect and store patient information, including personal details, medical history, and contact information.
    - Assign a unique patient ID for easy identification.
2. **Appointment Scheduling:**
    - Allow patients to book appointments with doctors online.
    - Manage appointment availability and send reminders to patients.
3. **Medical Records Management:**
    - Maintain electronic medical records (EMRs) for each patient.
    - Facilitate easy access to patient history, diagnoses, treatments, and prescriptions.
4. **Billing and Invoicing:**
    - Automate the billing process for medical services rendered.
    - Generate invoices and manage payment records.
5. **Staff Management:**
    - Manage doctor, nurse, and administrative staff information.

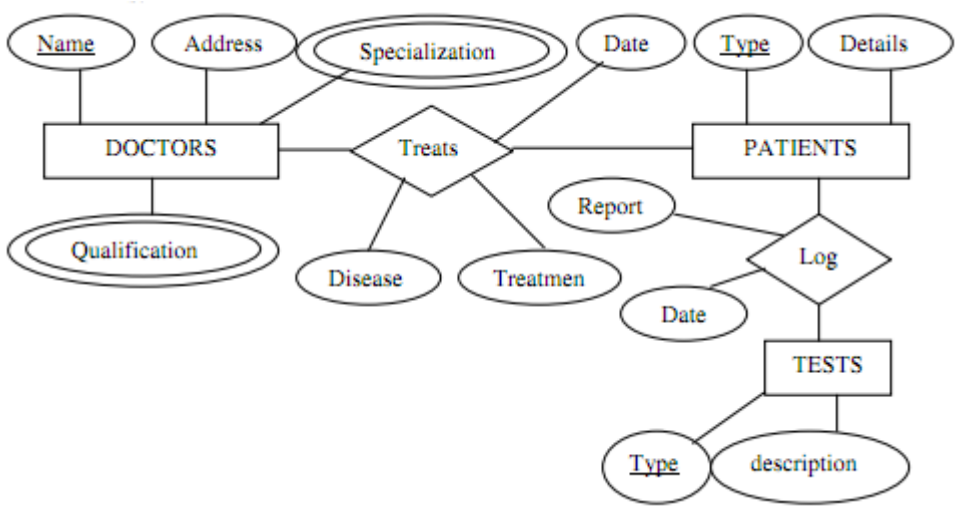
- Track staff schedules, availability, and performance.

6. **Reporting:**
   - Generate reports on patient statistics, financials, and operational efficiency.
   - Provide insights to aid decision-making and improve services.

Technologies Used:

- **Database Management System:** MySQL

Conclusion: The Hospital Management System will significantly enhance the efficiency of hospital operations, improve patient experience, and ensure accurate management of medical records. By implementing a robust database solution, the system will facilitate better healthcare delivery, ultimately leading to improved patient outcomes.



Practical -2

Implement the following DDL statements:

a. Create

b. Create table with constraints (NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY)

c. Alter Table:

i. Add column

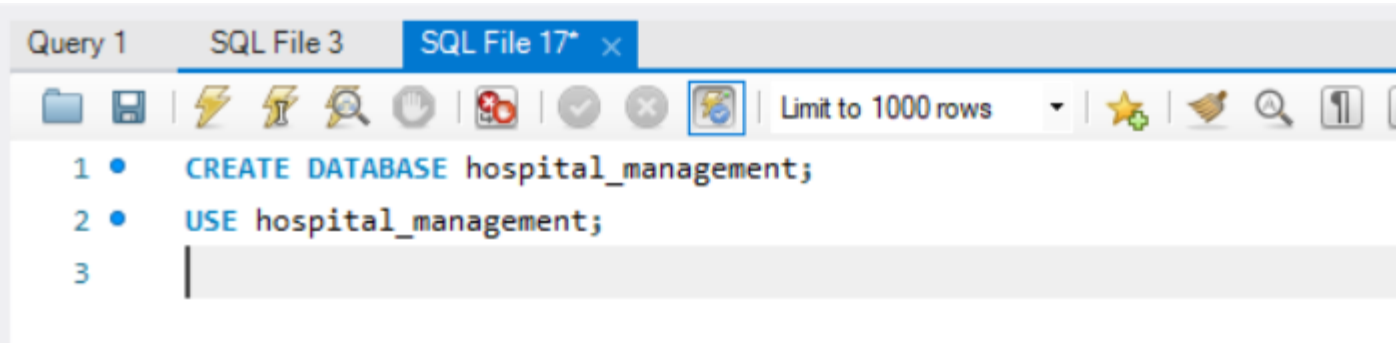
ii. Drop column

iii. Add/drop constraint

iv. Rename column

d. Drop Table

a. Create



b. Create table with constraints (NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY)

Query 1 SQL File 3 SQL File 17*

Limit to 1000 rows

```sql
CREATE TABLE Patients (
    patient_id INT PRIMARY KEY,
    patient_name VARCHAR(100) NOT NULL,
    age INT CHECK (age > 0),
    gender VARCHAR(10),
    phone VARCHAR(15) UNIQUE,
    disease VARCHAR(100),
    date_of_admission DATE DEFAULT (CURRENT_DATE())
);
```

c. Alter Table:

i. Add column

Query 1 SQL File 3 SQL File 17*

Limit to 1000 rows

```sql
ALTER TABLE Patients ADD doctor_assigned VARCHAR(100);
```

ii. Drop column

ients patients patients patients patients doctors doctors doctors

Limit to 1000 rows

```sql
ALTER TABLE Patients DROP COLUMN disease;
```

iii. Add/drop constraint

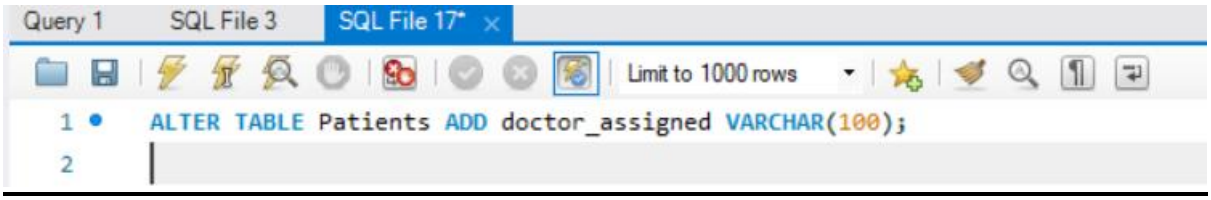
Add a NOT NULL constraint on doctor_assigned

Query 1 SQL File 3 SQL File 17*

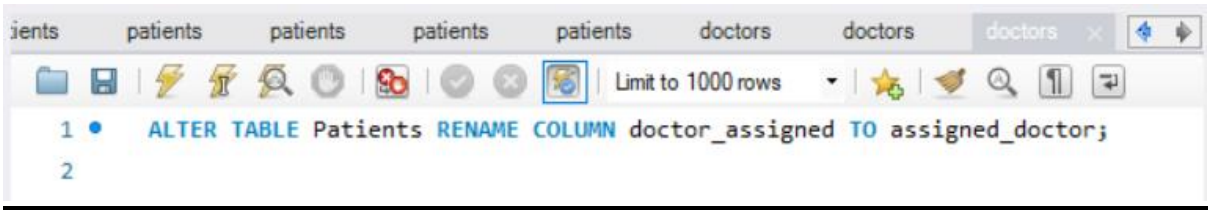
Limit to 1000 rows

```sql
ALTER TABLE Patients MODIFY doctor_assigned VARCHAR(100) NOT NULL;
```

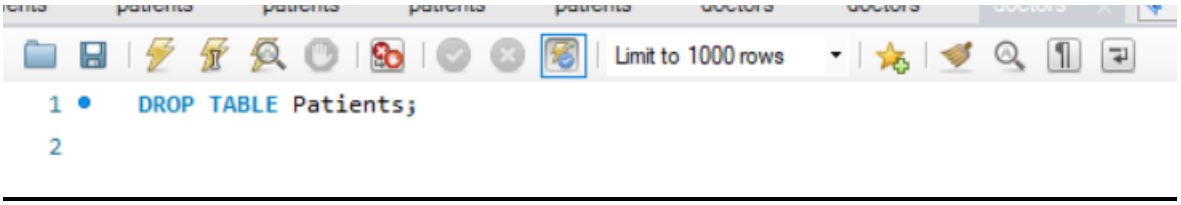
Drop the NOT NULL constraint



iv. Rename column



d. Drop Table



Practical -3

Implement the following DML statements:

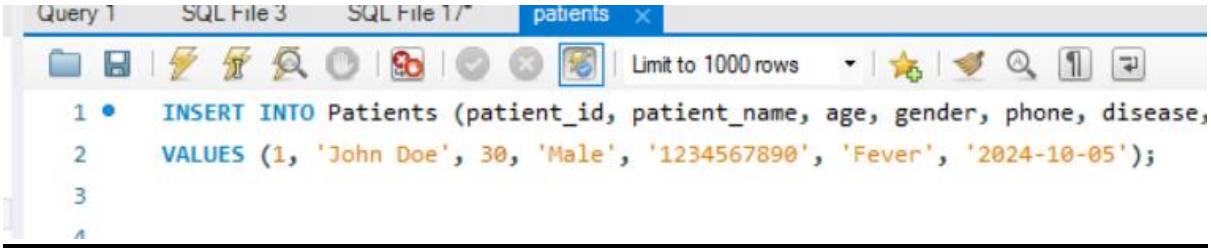
a. Insert

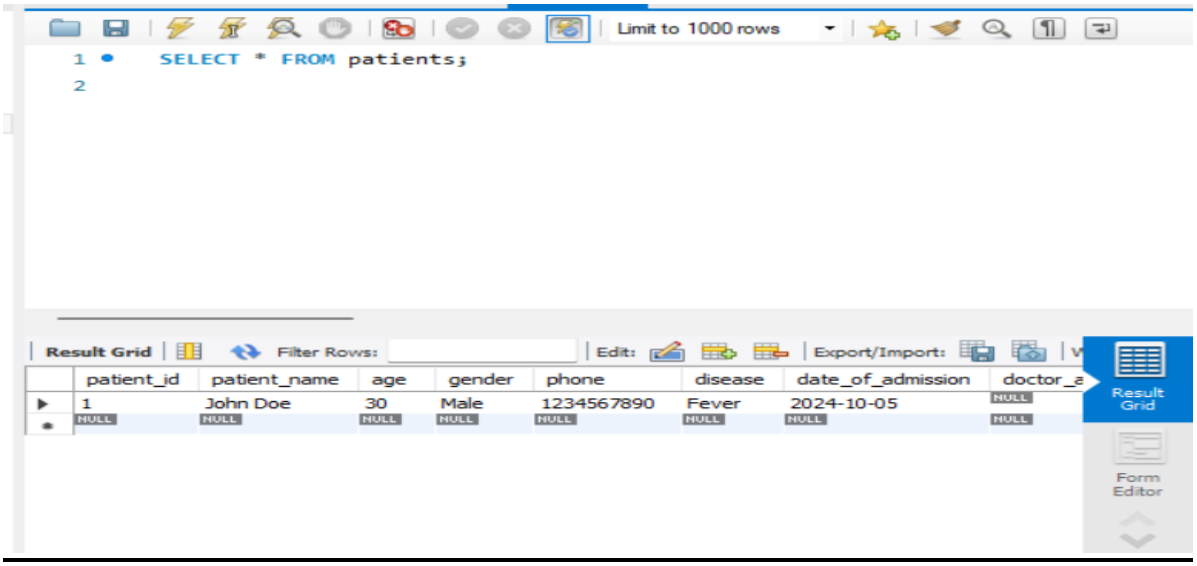
b. Update

c. Delete

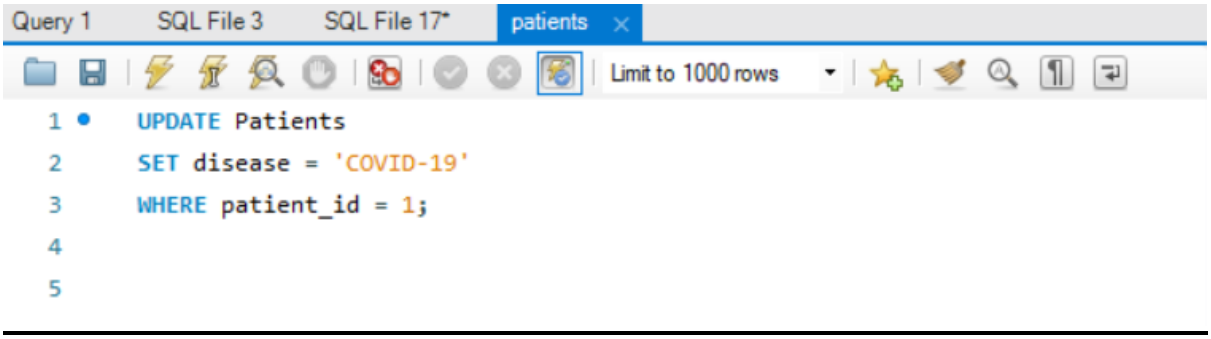
d. Truncate

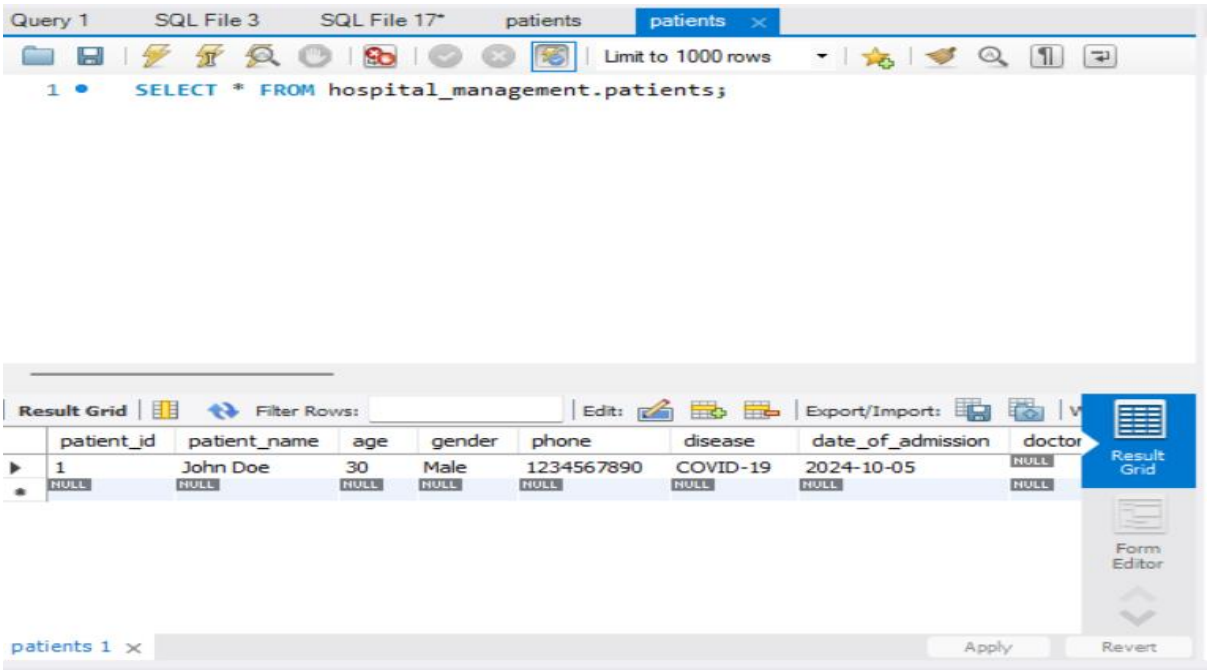
a. Insert



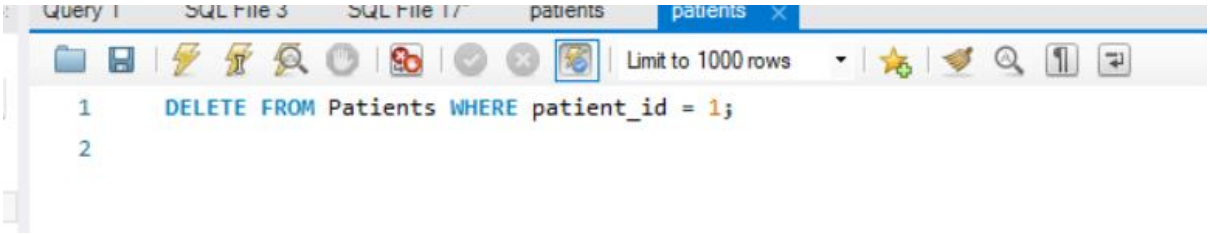


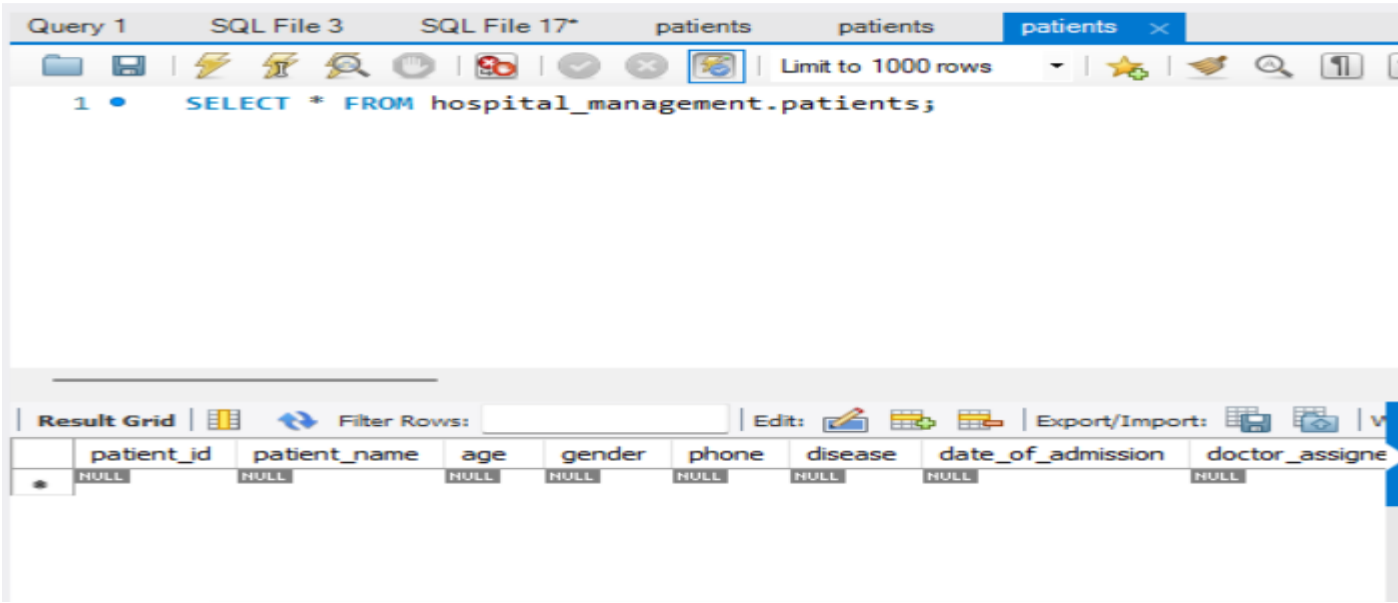
b. Update



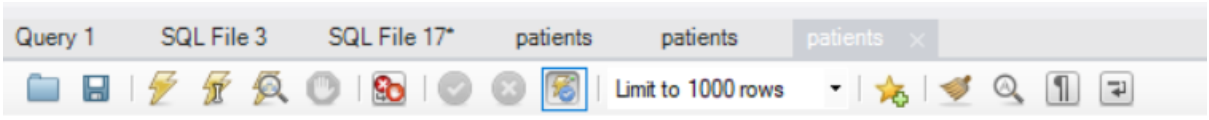


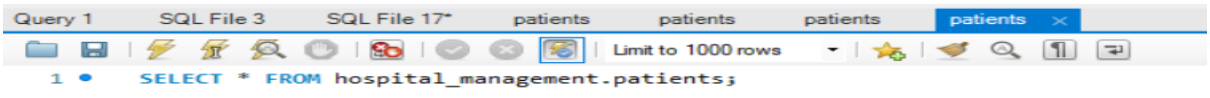
c. Delete

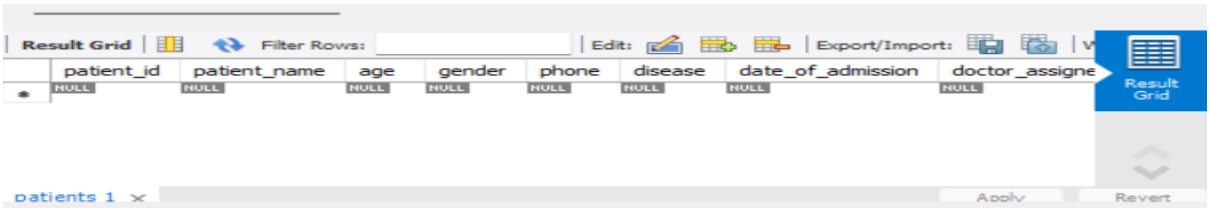




d. Truncate


Query 1
SQL File 3
SQL File 17*
patients
patients
patients
Limit to 1000 rows
1 TRUNCATE TABLE Patients;
2


Query 1
SQL File 3
SQL File 17*
patients
patients
patients
patients
Limit to 1000 rows
1 SELECT * FROM hospital_management.patients;


Result Grid
Filter Rows:
Edit:
Export/Import:
patient_id
patient_name
age
gender
phone
disease
date_of_admission
doctor_assigne
NULL
Result Grid
patients 1
Apply
Revert


Context Help
Snippets
Output
Action Output
Time Action Message Duration / Fetch
1 13:10:40 SELECT * FROM patients LIMIT 0, 1000 0 row(s) returned 0.000 sec / 0.000 sec
2 13:11:47 INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date_... 1 row(s) affected 0.000 sec
3 13:11:58 INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date_... Error Code: 1062. Duplicate entry '1' for key 'patients.PRIMARY' 0.000 sec
4 13:12:52 SELECT * FROM table_name LIMIT 0, 1000 Error Code: 1146. Table 'hospital_management.table_name' doesn't exist 0.000 sec
5 13:13:02 SELECT * FROM patients LIMIT 0, 1000 1 row(s) returned 0.000 sec / 0.000 sec
6 13:13:54 UPDATE Patients SET disease = 'COVID-19' WHERE patient_id = 1 1 row(s) affected Rows matched: 1 Changed: 1 Warnings: 0 0.016 sec
7 13:14:30 SELECT * FROM hospital_management.patients LIMIT 0, 1000 1 row(s) returned 0.000 sec / 0.000 sec
8 13:15:04 DELETE FROM Patients WHERE patient_id = 1 1 row(s) affected 0.000 sec
9 13:15:45 SELECT * FROM hospital_management.patients LIMIT 0, 1000 0 row(s) returned 0.000 sec / 0.000 sec
10 13:16:14 TRUNCATE TABLE Patients 0 row(s) affected 0.047 sec

Practical -4

Implement the following SELECT statements:

a. Simple SELECT statement

b. Where clause+ IN/NOT IN

c. Aggregate functions

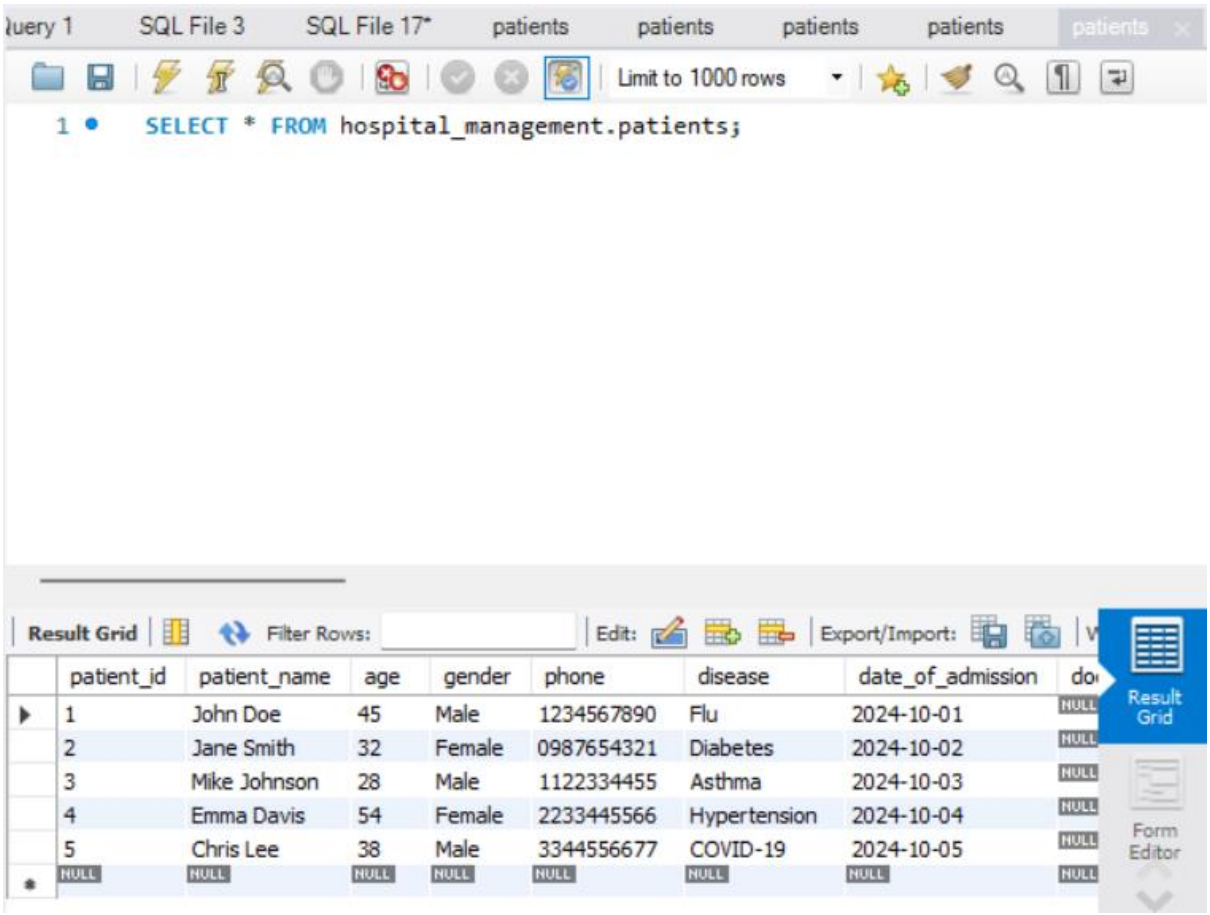
d. Group by + Having

e. Order by

f. Views

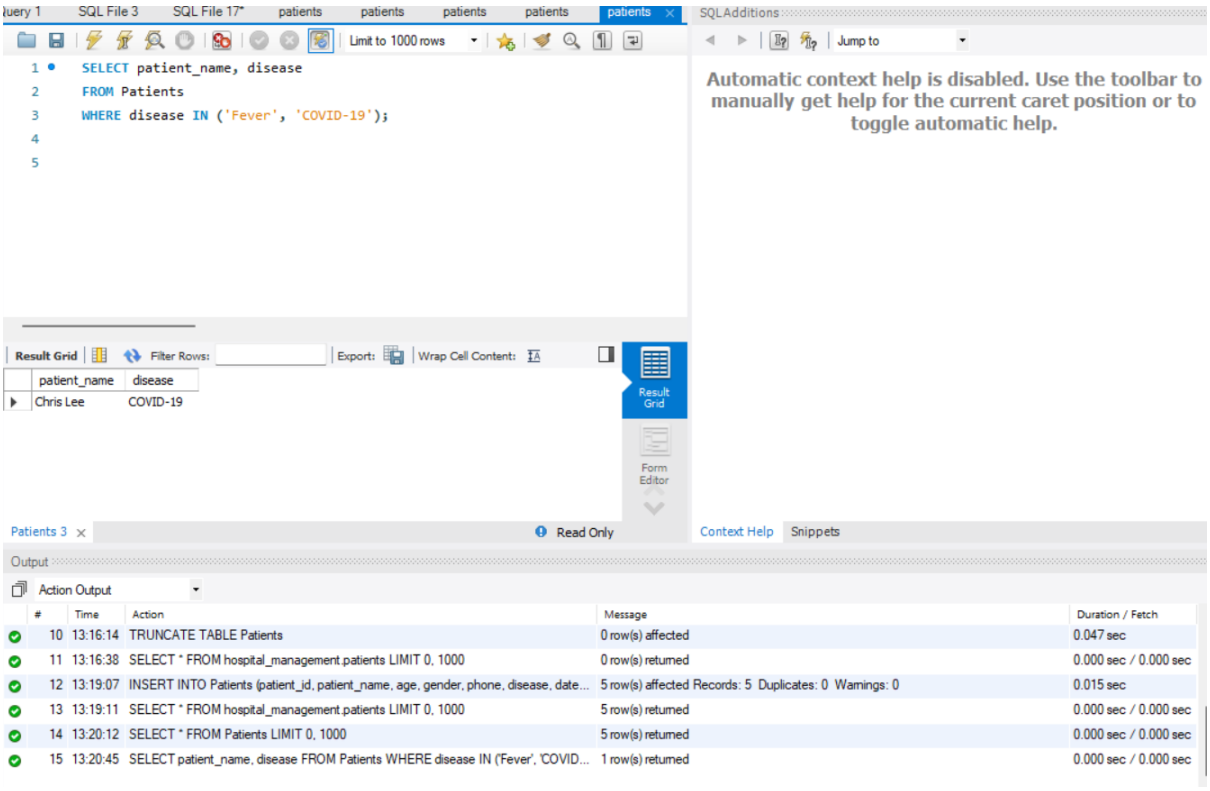
g. Inbuilt Functions (e.g., Date)

a. Simple SELECT statement



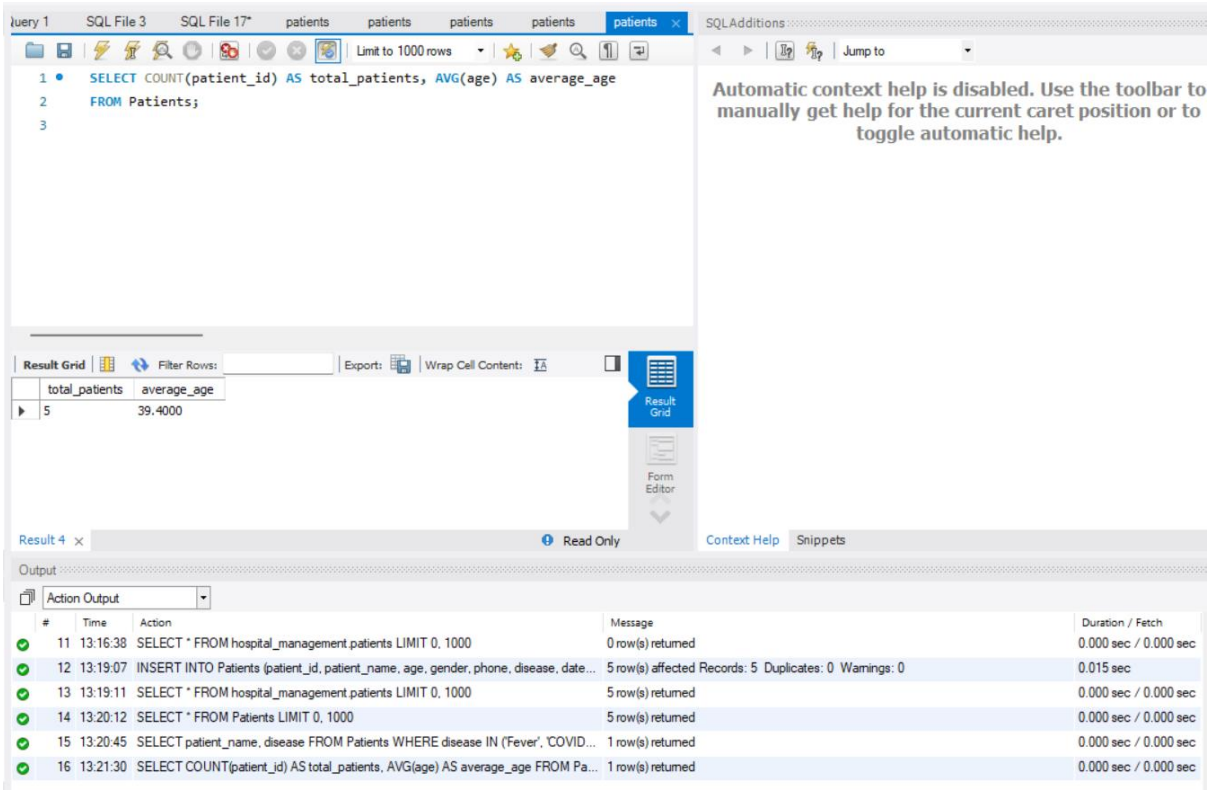
patient_id	patient_name	age	gender	phone	disease	date_of_admission	do
1	John Doe	45	Male	1234567890	Flu	2024-10-01	NULL
2	Jane Smith	32	Female	0987654321	Diabetes	2024-10-02	NULL
3	Mike Johnson	28	Male	1122334455	Asthma	2024-10-03	NULL
4	Emma Davis	54	Female	2233445566	Hypertension	2024-10-04	NULL
5	Chris Lee	38	Male	3344556677	COVID-19	2024-10-05	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

b. Where clause + IN/NOT IN:



c. Aggregate Functions (e.g., COUNT, AVG):

sql



d. Group By + Having

```
1  SELECT disease, COUNT(patient_id) AS total_cases
2  FROM Patients
3  GROUP BY disease
4
5
```

disease	total_cases
Flu	1
Diabetes	1
Asthma	1
Hypertension	1
COVID-19	1

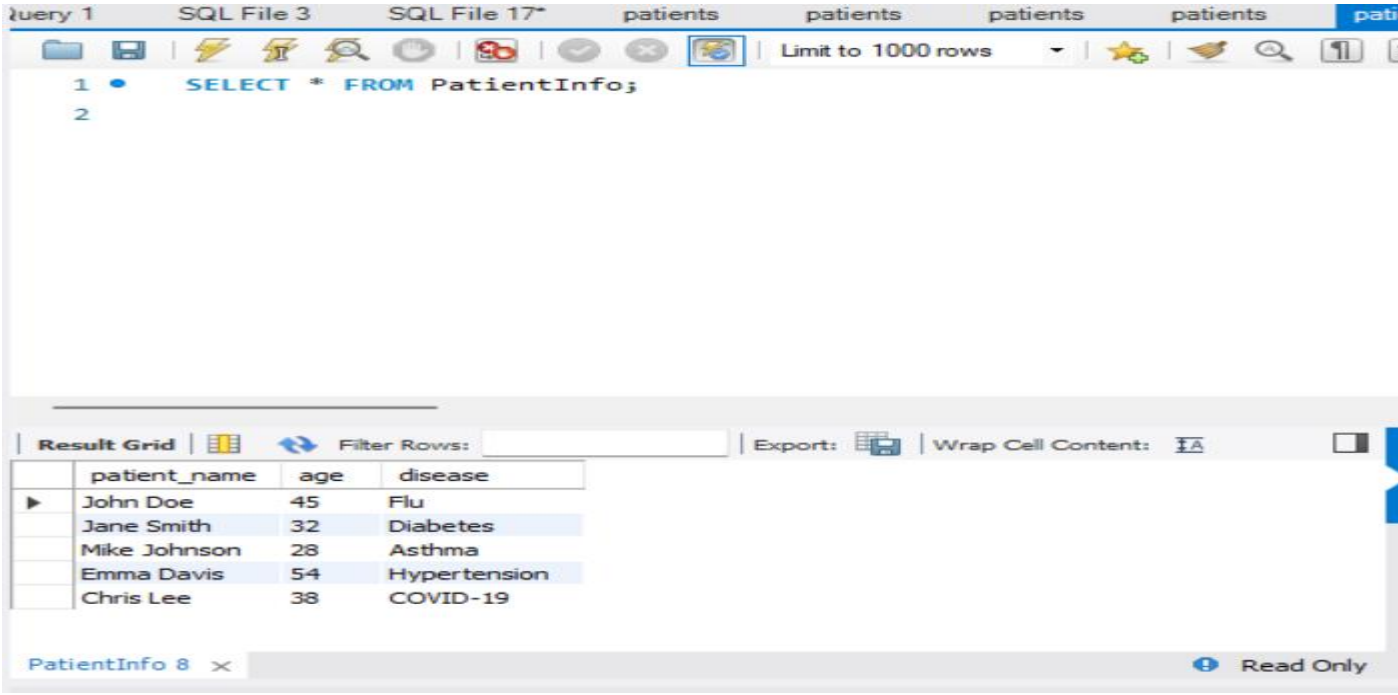
e. Order By

```
1  SELECT patient_name, age FROM Patients ORDER BY age DESC;
2
```

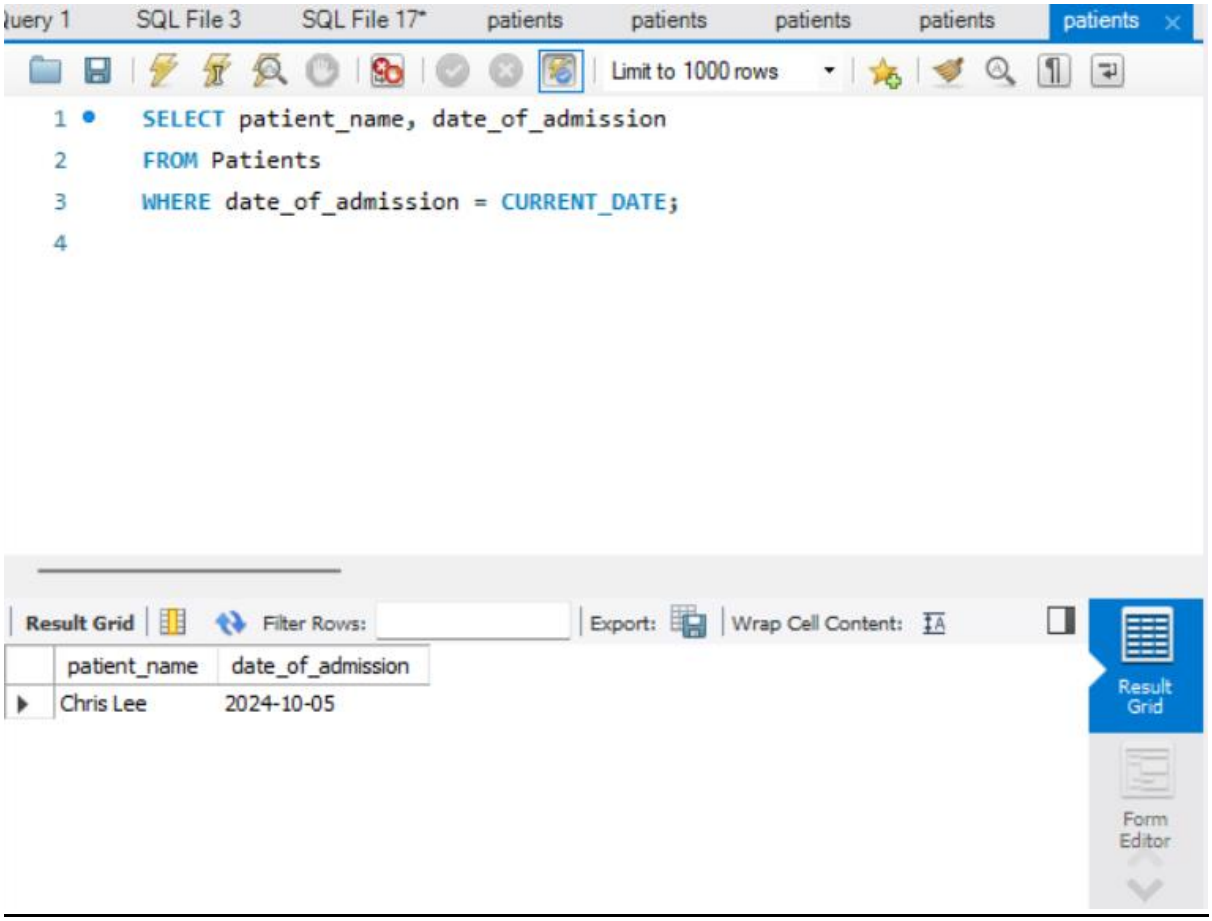
patient_name	age
Emma Davis	54
John Doe	45
Chris Lee	38
Jane Smith	32
Mike Johnson	28

f. Views

```
1  CREATE VIEW PatientInfo AS
2  SELECT patient_name, age, disease FROM Patients;
3
```



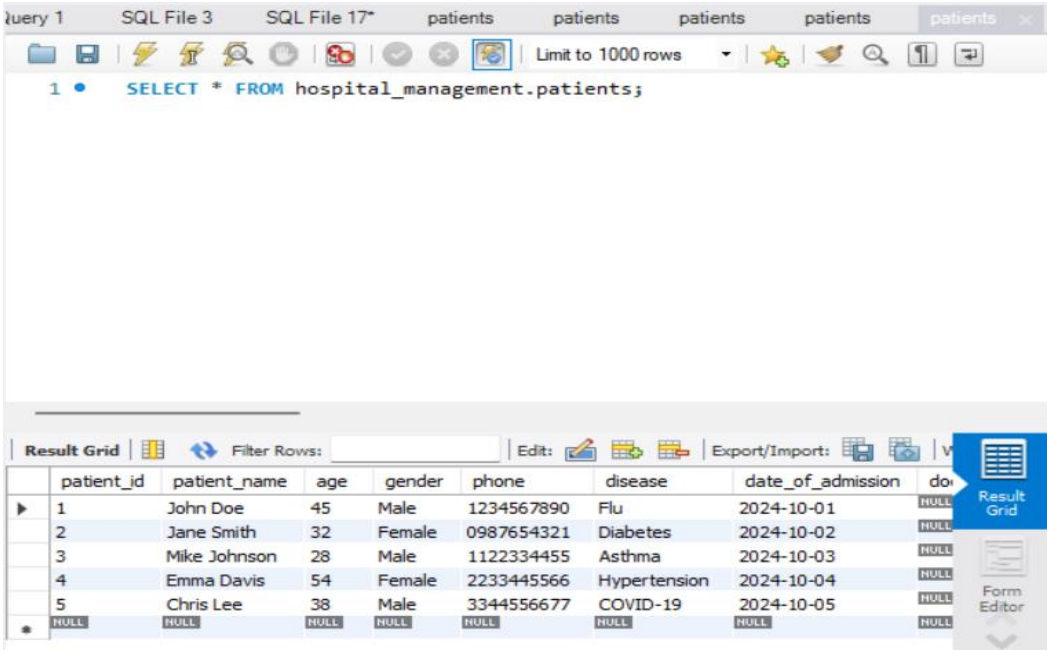
g. Inbuilt Functions (e.g., Date)

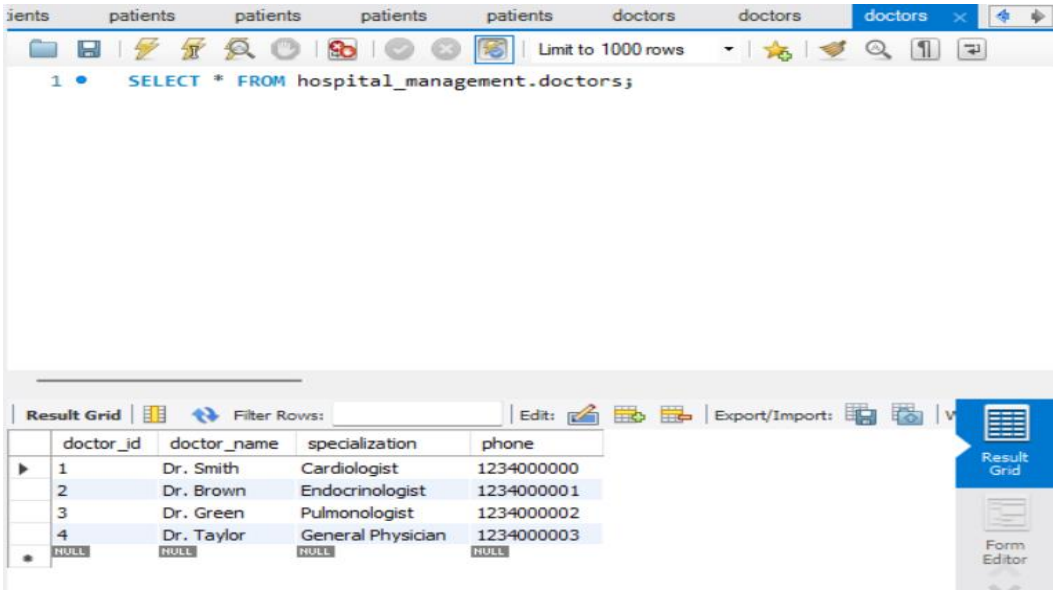


Practical -5

Implement and perform Nested Queries along with Joins (Inner join, Outer join, Left join, Right join)

We have 2 tables named Patients and doctors





Add a doctor_id to the Patients Table (for relational queries)

We need to update the Patients table to include a reference to the Doctors table, linking patients to doctors using a foreign key.

```
1   ALTER TABLE Patients
2   ADD doctor_id INT,
3   ADD FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id);
4
```

Update Patient Records to Assign Doctors

```
1   UPDATE Patients
2   SET doctor_id = 1 WHERE disease = 'Hypertension';
3   UPDATE Patients
4   SET doctor_id = 2 WHERE disease = 'Diabetes';
5   UPDATE Patients
6   SET doctor_id = 3 WHERE disease = 'Asthma';
7   UPDATE Patients
8   SET doctor_id = 4 WHERE disease = 'Flu';
9   UPDATE Patients
10  SET doctor_id = 4 WHERE disease = 'COVID-19';
11
12
```

Nested Queries

```
1   SELECT patient_name, age, disease
2   FROM Patients
3   WHERE doctor_id = (SELECT doctor_id FROM Doctors WHERE specialization = 'Card
4
```

patient_name	age	disease
Emma Davis	54	Hypertension

Inner join

```
1  SELECT P.patient_name, P.age, P.disease, D.doctor_name, D.specialization
2  FROM Patients P
3  INNER JOIN Doctors D ON P.doctor_id = D.doctor_id;
4
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

patient_name	age	disease	doctor_name	specialization
John Doe	45	Flu	Dr. Taylor	General Physician
Jane Smith	32	Diabetes	Dr. Brown	Endocrinologist
Mike Johnson	28	Asthma	Dr. Green	Pulmonologist
Emma Davis	54	Hypertension	Dr. Smith	Cardiologist
Chris Lee	38	COVID-19	Dr. Taylor	General Physician

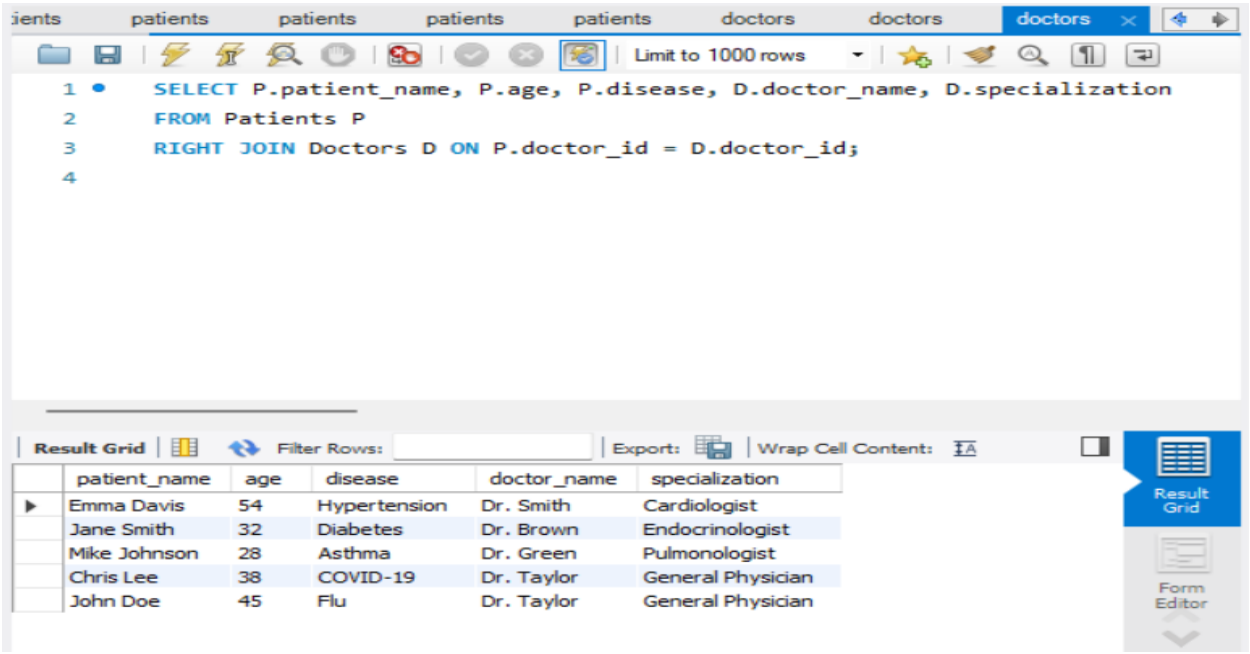
Left join

```
1  SELECT P.patient_name, P.age, P.disease, D.doctor_name, D.specialization
2  FROM Patients P
3  LEFT JOIN Doctors D ON P.doctor_id = D.doctor_id;
4
```

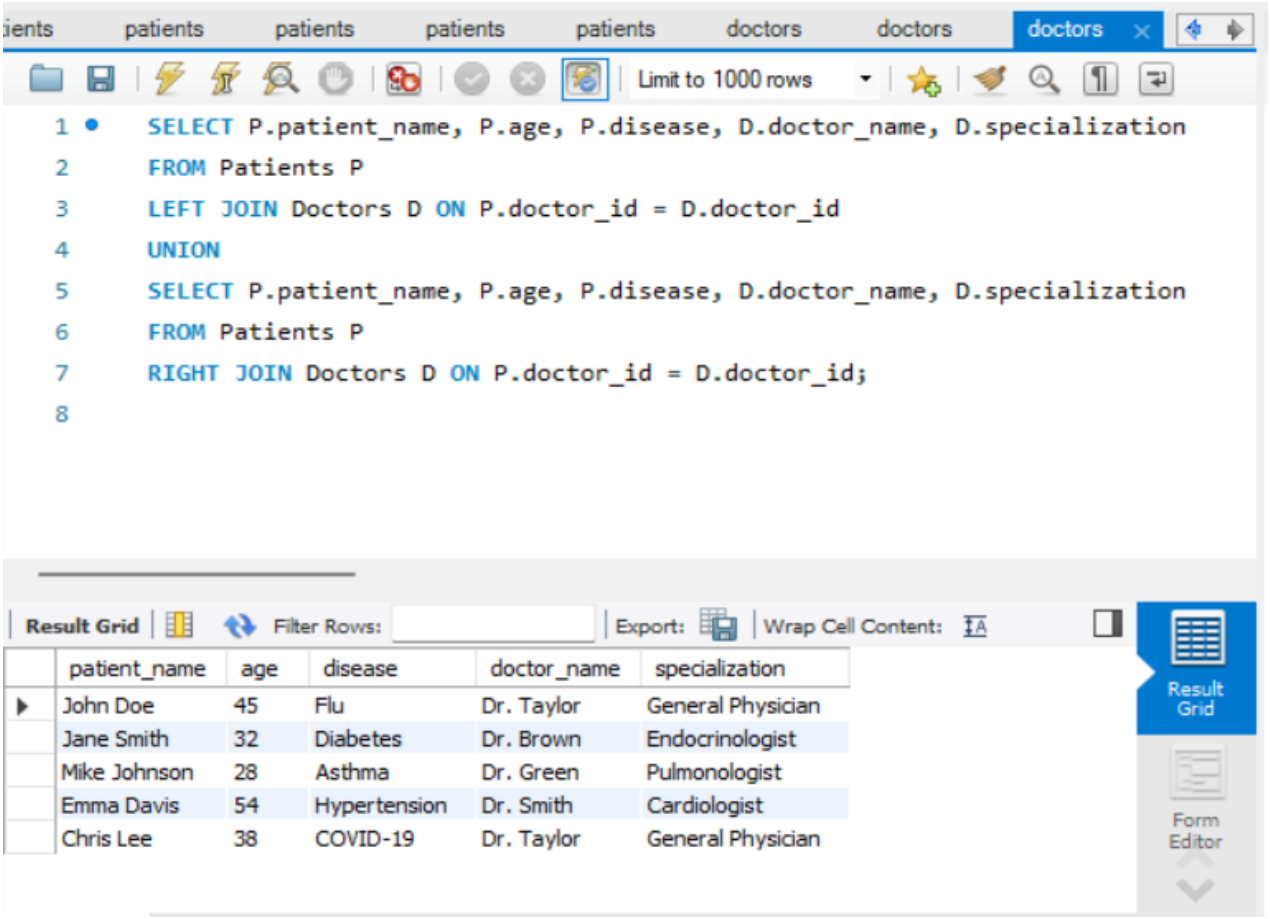
Result Grid | Filter Rows: | Export: | Wrap Cell Content:

patient_name	age	disease	doctor_name	specialization
John Doe	45	Flu	Dr. Taylor	General Physician
Jane Smith	32	Diabetes	Dr. Brown	Endocrinologist
Mike Johnson	28	Asthma	Dr. Green	Pulmonologist
Emma Davis	54	Hypertension	Dr. Smith	Cardiologist
Chris Lee	38	COVID-19	Dr. Taylor	General Physician

Right join



Outer join



Practical -6

Introduction to PL/SQL. Create a PL/SQL block and implement the following:

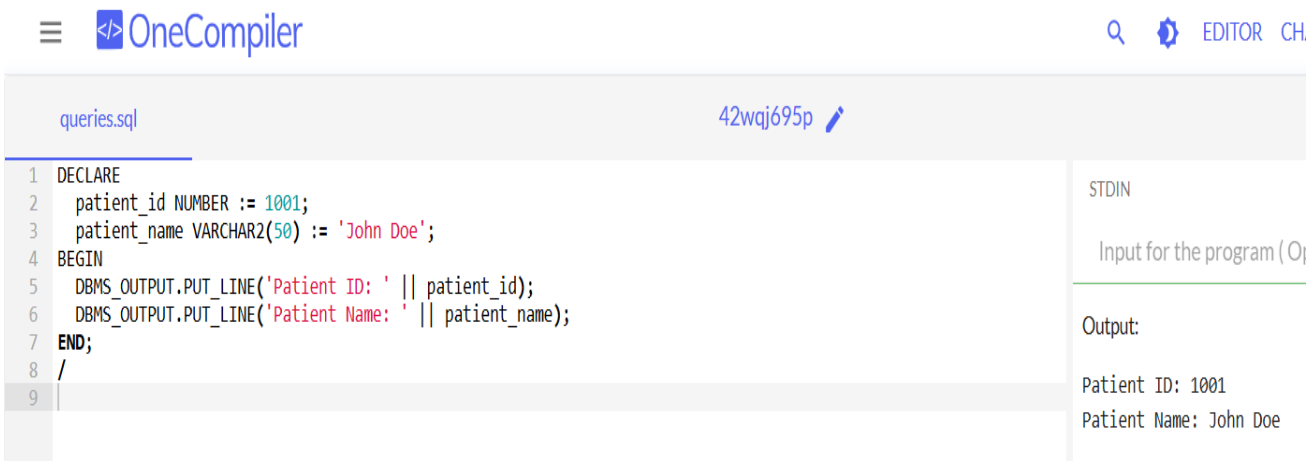
a. Variables

b. Packages

c. Procedures

d. Functions

a. Variables



b. Packages

```
CREATE OR REPLACE PACKAGE hospital_pkg IS
  PROCEDURE add_patient(patient_name VARCHAR2, patient_age NUMBER);
  FUNCTION get_patient_count RETURN NUMBER;
END hospital_pkg;
/
CREATE OR REPLACE PACKAGE BODY hospital_pkg IS
  v_patient_count NUMBER := 0;

  PROCEDURE add_patient(patient_name VARCHAR2, patient_age NUMBER) IS
  BEGIN
    v_patient_count := v_patient_count + 1;
    DBMS_OUTPUT.PUT_LINE('Added patient: ' || patient_name);
  END add_patient;

  FUNCTION get_patient_count RETURN NUMBER IS
  BEGIN
    RETURN v_patient_count;
  END get_patient_count;

END hospital_pkg;
/
DECLARE
  patient_count NUMBER;
BEGIN
  hospital_pkg.add_patient('Alice Green', 29);
  hospital_pkg.add_patient('Bob Brown', 45);
  patient_count := hospital_pkg.get_patient_count;
  DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
END;
/
```

```
STDIN
Input for the program ( Opti
Output:
Added patient: Alice Green
Added patient: Bob Brown
Total Patients: 2
```

c. Procedures

```
DECLARE
  patient_count NUMBER;
BEGIN
  patient_count := hospital_pkg.get_patient_count;
  DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
END;
/
```

```
Statement processed.
Total Patients: 0
```

d. Functions

```
DECLARE
  patient_count NUMBER;
BEGIN
  patient_count := hospital_pkg.get_patient_count;
  DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
END;
/
```

```
Statement processed.
Total Patients: 0
```

Practical -7

Perform Exception handling in a PL/SQL block.

```
DECLARE
   patient_id NUMBER := 1;            -- Example patient ID
   patient_name VARCHAR2(50);          -- Variable to hold patient name
   age NUMBER := 0;                    -- Example variable for division by zero
   result NUMBER;
BEGIN
   -- Attempt to retrieve patient name
   SELECT name INTO patient_name
   FROM patients
   WHERE id = patient_id;

   -- Display patient name
   DBMS_OUTPUT.PUT_LINE('Patient Name: ' || patient_name);

   -- Example of division by zero to trigger an exception
   result := 100 / age;
   DBMS_OUTPUT.PUT_LINE('Result: ' || result);

EXCEPTION
   -- Handle the NO_DATA_FOUND exception
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('Error: Patient with ID ' || patient_id || ' was not found.');
```

```

   -- Example of division by zero to trigger an exception
   result := 100 / age;
   DBMS_OUTPUT.PUT_LINE('Result: ' || result);

EXCEPTION
   -- Handle the NO_DATA_FOUND exception
   WHEN NO_DATA_FOUND THEN
     DBMS_OUTPUT.PUT_LINE('Error: Patient with ID ' || patient_id || ' was not found.');

   -- Handle division by zero
   WHEN ZERO_DIVIDE THEN
     DBMS_OUTPUT.PUT_LINE('Error: Division by zero occurred.');

   -- Handle any other exception
   WHEN OTHERS THEN
     DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
END;
/

```

```
Statement processed.
Patient Name: Alice Green
Error: Division by zero occurred.
```

Practical -9

Implement Triggers in SQL.

Step 1: Create the Trigger

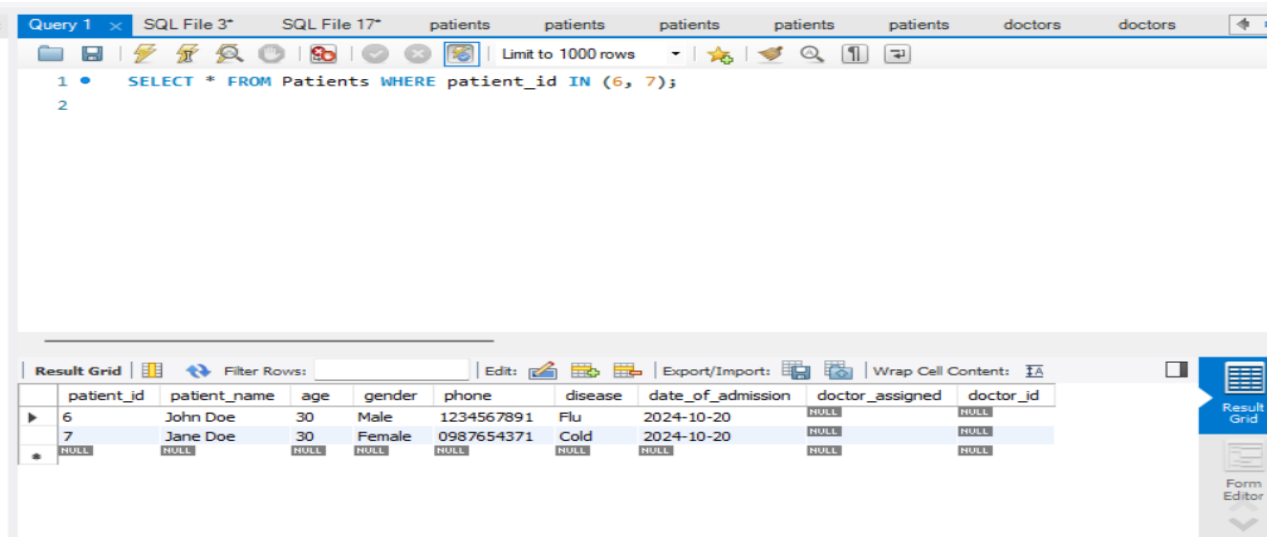
```
DELIMITER $$

CREATE TRIGGER set_default_age
BEFORE INSERT ON Patients
FOR EACH ROW
BEGIN
    IF NEW.age IS NULL THEN
        SET NEW.age = 30;
    END IF;
END$$

DELIMITER ;
```

Step 2: Insert Data

```
INSERT INTO Patients (patient_id, patient_name, gender, phone, disease, date_of_admission)
VALUES (6, 'John Doe', 'Male', '1234567891', 'Flu', CURRENT_DATE());
```

```
INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date_of_admission)
VALUES (7, 'Jane Doe', NULL, 'Female', '0987654371', 'Cold', CURRENT_DATE());
```

Step 3: Verify the Output



Practical -10

Implement the following transaction statements:

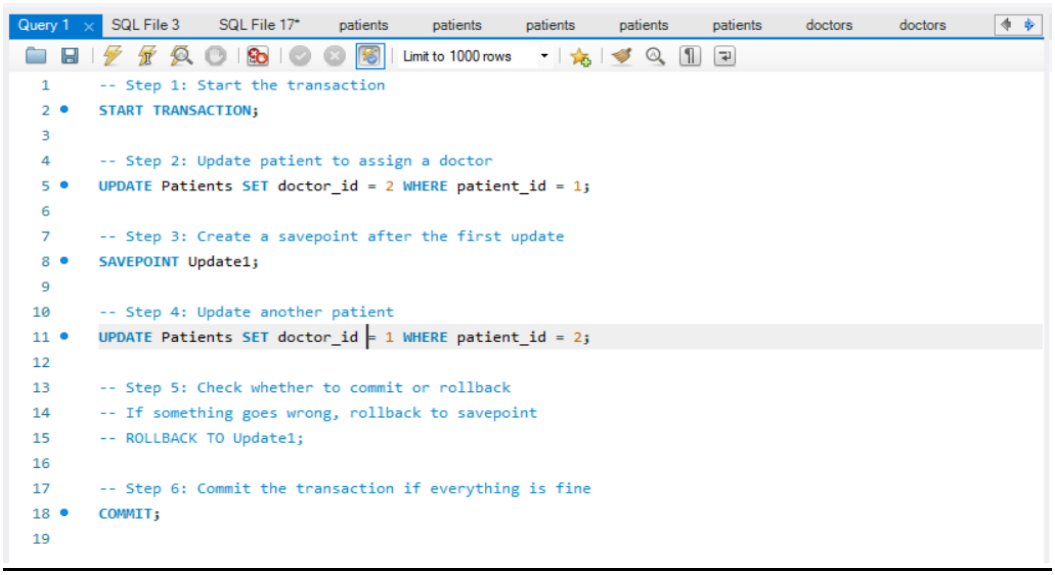
a. Commit

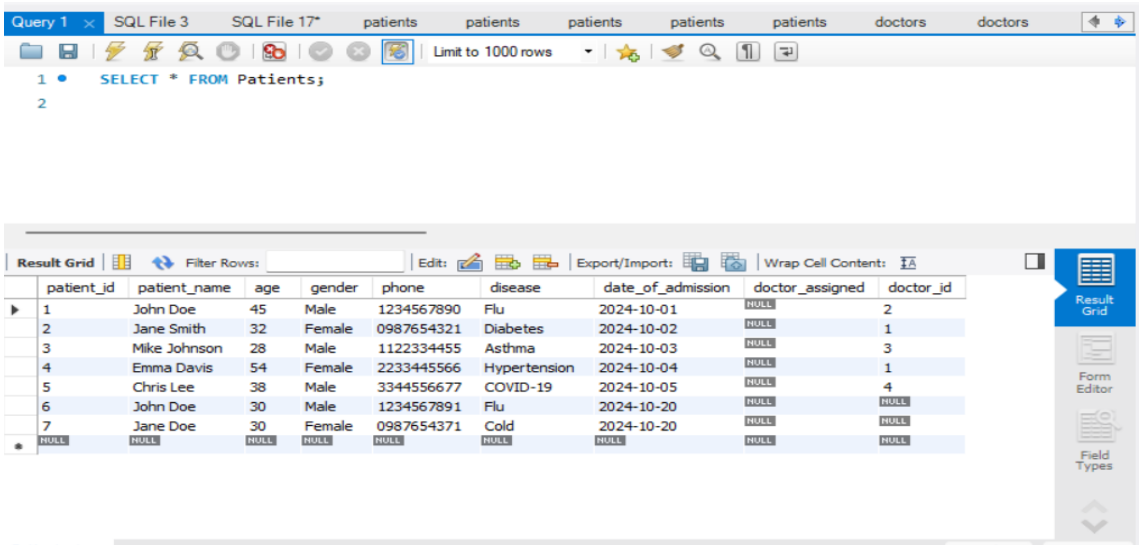
b. Rollback

c. Savepoint

a. Commit

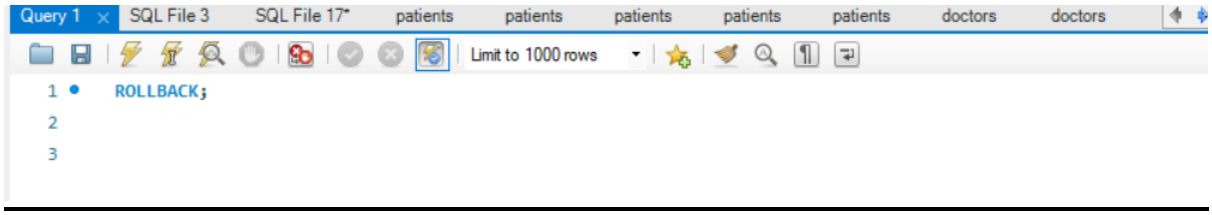
c. Savepoint

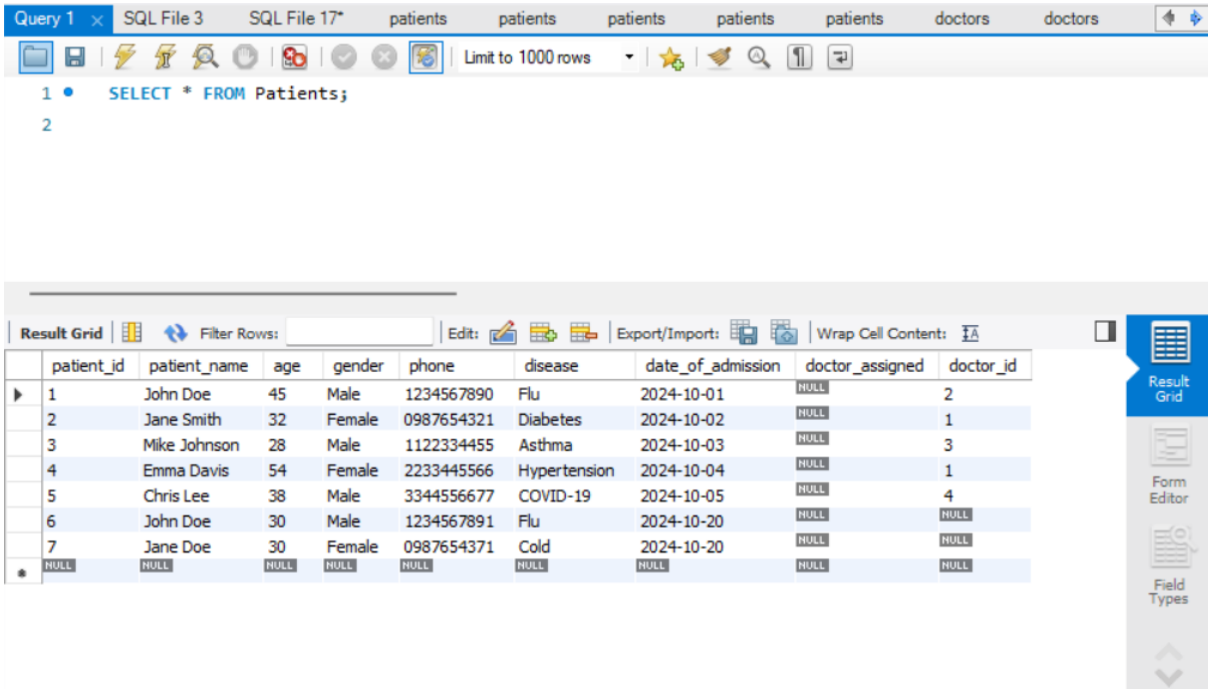




patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_assigned	doctor_id
1	John Doe	45	Male	1234567890	Flu	2024-10-01	NULL	2
2	Jane Smith	32	Female	0987654321	Diabetes	2024-10-02	NULL	1
3	Mike Johnson	28	Male	1122334455	Asthma	2024-10-03	NULL	3
4	Emma Davis	54	Female	2233445566	Hypertension	2024-10-04	NULL	1
5	Chris Lee	38	Male	3344556677	COVID-19	2024-10-05	NULL	4
6	John Doe	30	Male	1234567891	Flu	2024-10-20	NULL	NULL
7	Jane Doe	30	Female	0987654371	Cold	2024-10-20	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

b. Rollback





patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_assigned	doctor_id
1	John Doe	45	Male	1234567890	Flu	2024-10-01	NULL	2
2	Jane Smith	32	Female	0987654321	Diabetes	2024-10-02	NULL	1
3	Mike Johnson	28	Male	1122334455	Asthma	2024-10-03	NULL	3
4	Emma Davis	54	Female	2233445566	Hypertension	2024-10-04	NULL	1
5	Chris Lee	38	Male	3344556677	COVID-19	2024-10-05	NULL	4
6	John Doe	30	Male	1234567891	Flu	2024-10-20	NULL	NULL
7	Jane Doe	30	Female	0987654371	Cold	2024-10-20	NULL	NULL
NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL