

# **DELHI TECHNOLOGICAL UNIVERSITY**



***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: <ol style="list-style-type: none"> <li>a. Create</li> <li>b. Create table with constraints (NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY)</li> <li>c. Alter Table:               <ol style="list-style-type: none"> <li>i. Add column</li> <li>ii. Drop column</li> <li>iii. Add/drop constraint</li> <li>iv. Rename column</li> </ol> </li> <li>d. Drop Table</li> </ol>	2/09/24	
3	Implement the following DML statements: <ol style="list-style-type: none"> <li>a. Insert</li> <li>b. Update</li> <li>c. Delete</li> <li>d. Truncate</li> </ol>	9/09/24	
4	Implement the following SELECT statements: <ol style="list-style-type: none"> <li>a. Simple SELECT statement</li> <li>b. Where clause+ IN/NOT IN</li> <li>c. Aggregate functions</li> <li>d. Group by + Having</li> <li>e. Order by</li> <li>f. Views</li> <li>g. Inbuilt Functions (e.g., Date)</li> </ol>	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: <ol style="list-style-type: none"> <li>a. Variables</li> </ol>	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.

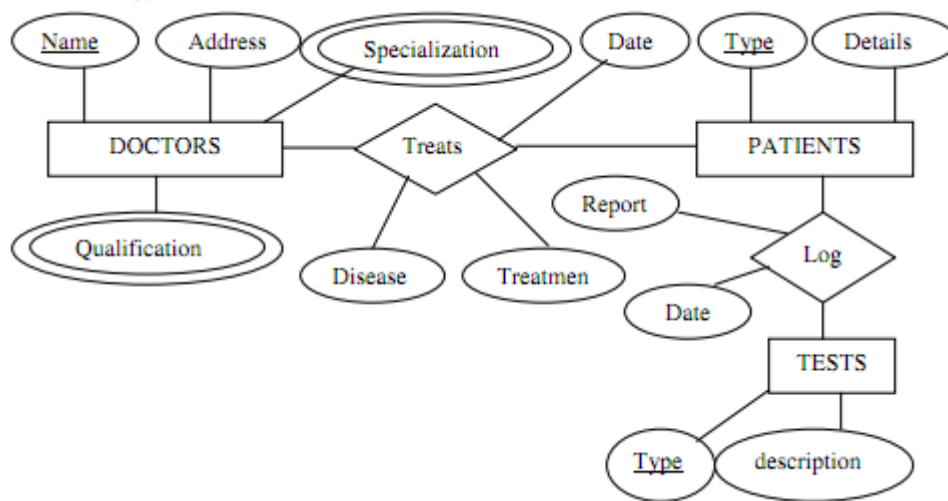
## 6. Reporting:

- 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**

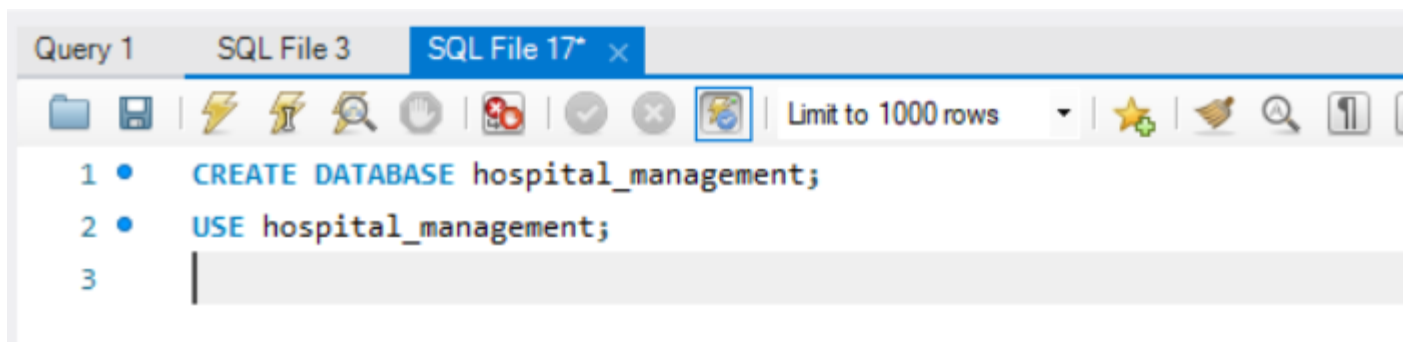
**ii. Drop column**

**iii. Add/drop constraint**

**iv. Rename column**

**d. Drop Table**

**a. Create**



```
1 • CREATE DATABASE hospital_management;
2 • USE hospital_management;
3 |
```

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

```
Query 1    SQL File 3    SQL File 17* x
Limit to 1000 rows

1 CREATE TABLE Patients (
2     patient_id INT PRIMARY KEY,
3     patient_name VARCHAR(100) NOT NULL,
4     age INT CHECK (age > 0),
5     gender VARCHAR(10),
6     phone VARCHAR(15) UNIQUE,
7     disease VARCHAR(100),
8     date_of_admission DATE DEFAULT (CURRENT_DATE())
9 );
10
```

### **c. Alter Table:**

#### **i. Add column**

```
Query 1    SQL File 3    SQL File 17* x
Limit to 1000 rows

1 ALTER TABLE Patients ADD doctor_assigned VARCHAR(100);
2
```

#### **ii. Drop column**

```
patients  patients  patients  patients  patients  doctors  doctors  doctors x
Limit to 1000 rows

1 ALTER TABLE Patients DROP COLUMN disease;
2
```

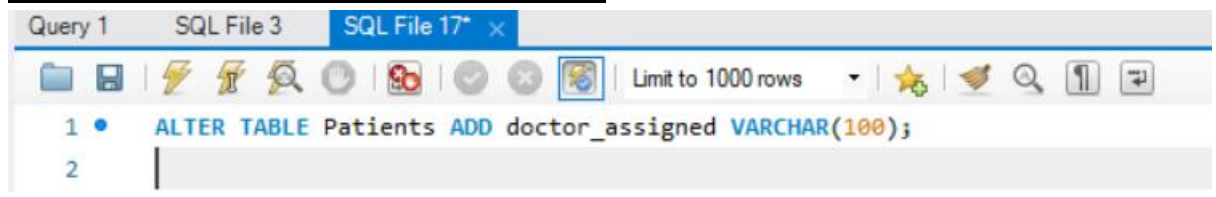
#### **iii. Add/drop constraint**

##### **Add a NOT NULL constraint on doctor assigned**

```
Query 1    SQL File 3    SQL File 17* x
Limit to 1000 rows

1 ALTER TABLE Patients MODIFY doctor_assigned VARCHAR(100) NOT NULL;
2
```

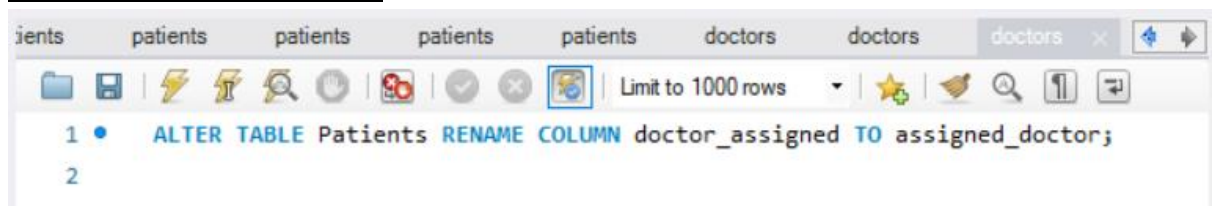
## **Drop the NOT NULL constraint**



The screenshot shows a SQL IDE window with a tab labeled 'SQL File 17\*'. The toolbar includes icons for file operations, execution, and search. The query editor contains the following SQL statement:

```
1 • ALTER TABLE Patients ADD doctor_assigned VARCHAR(100);
2 |
```

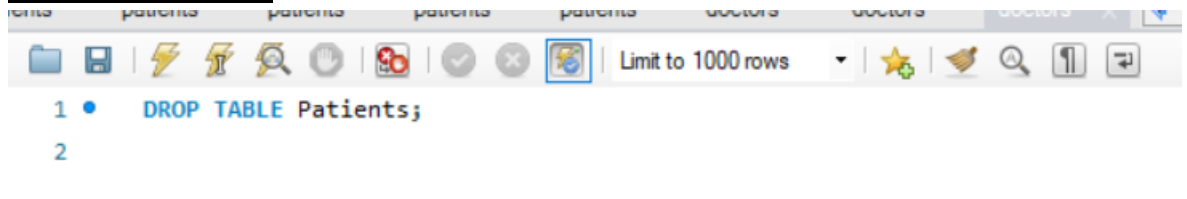
## **iv. Rename column**



The screenshot shows a SQL IDE window with a tab labeled 'doctors'. The toolbar includes icons for file operations, execution, and search. The query editor contains the following SQL statement:

```
1 • ALTER TABLE Patients RENAME COLUMN doctor_assigned TO assigned_doctor;
2 |
```

## **d. Drop Table**



The screenshot shows a SQL IDE window with a tab labeled 'doctors'. The toolbar includes icons for file operations, execution, and search. The query editor contains the following SQL statement:

```
1 • DROP TABLE Patients;
2 |
```



## Practical -3

Implement the following DML statements:

a. Insert

b. Update

c. Delete

d. Truncate

a. Insert

The screenshot shows a database client interface with a tab labeled 'patients'. The SQL editor contains the following code:

```
1 • INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease,  
2 VALUES (1, 'John Doe', 30, 'Male', '1234567890', 'Fever', '2024-10-05');  
3  
4
```

Below the editor, the 'Result Grid' is displayed, showing the result of the INSERT statement:

	patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_s
▶	1	John Doe	30	Male	1234567890	Fever	2024-10-05	NULL
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

The interface also includes a toolbar with various icons and a 'Limit to 1000 rows' dropdown.

b. Update

The screenshot shows the same database client interface with the 'patients' tab. The SQL editor contains the following code:

```
1 • UPDATE Patients  
2 SET disease = 'COVID-19'  
3 WHERE patient_id = 1;  
4  
5
```

The interface includes the same toolbar and 'Limit to 1000 rows' dropdown as the previous screenshot.

Query 1   SQL File 3   SQL File 17\*   patients   patients x

Limit to 1000 rows

```
1 • SELECT * FROM hospital_management.patients;
```

Result Grid

	patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor
▶	1	John Doe	30	Male	1234567890	COVID-19	2024-10-05	NULL
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

patients 1 x   Apply   Revert

### c. Delete

Query 1   SQL File 3   SQL File 17\*   patients   patients x

Limit to 1000 rows

```
1 DELETE FROM Patients WHERE patient_id = 1;
2
```

Query 1   SQL File 3   SQL File 17\*   patients   patients   patients x

Limit to 1000 rows

```
1 • SELECT * FROM hospital_management.patients;
```

Result Grid

	patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_assigne
•	NULL	NULL	NULL	NULL	NULL	NULL	NULL	NULL

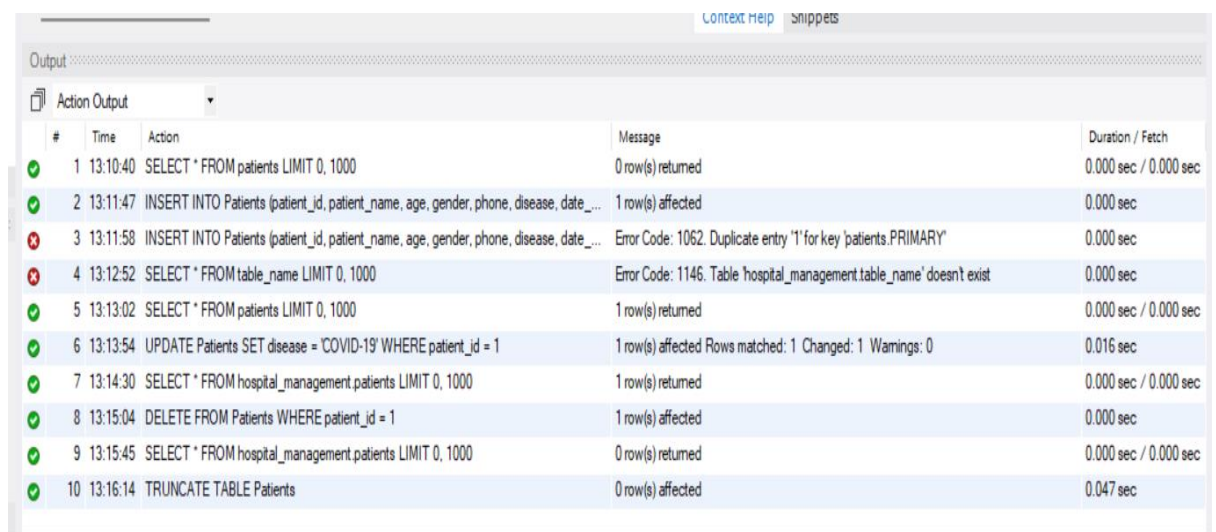
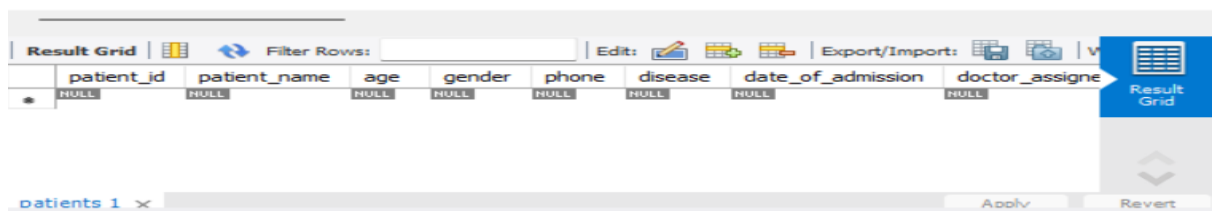
### d. Truncate



```
1 TRUNCATE TABLE Patients;  
2
```



```
1 • SELECT * FROM hospital_management.patients;
```



## **Practical -4**

**Implement the following SELECT statements:**

**a. Simple SELECT statement**

**b. Where clause+ IN/NOT IN**

**c. Aggregate functions**

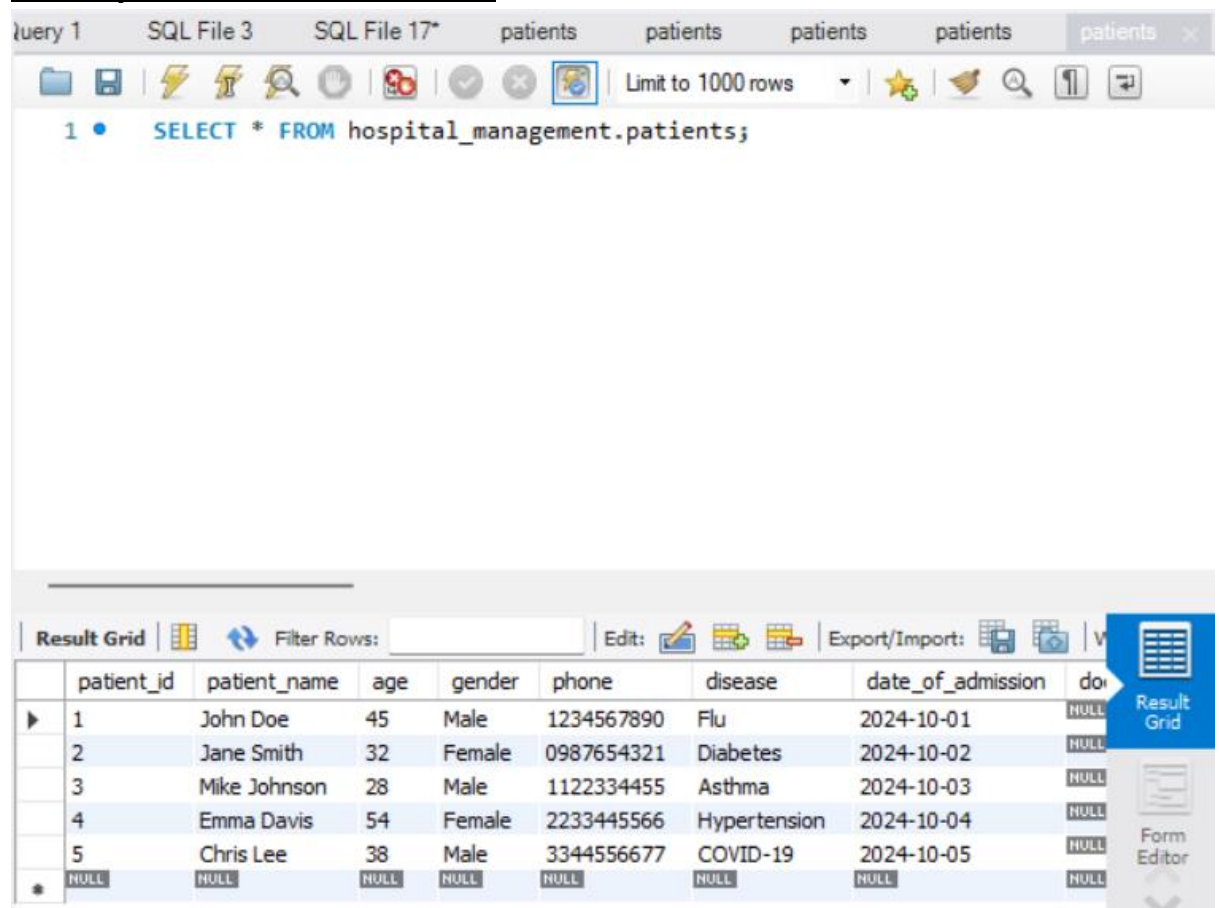
**d. Group by + Having**

**e. Order by**

**f. Views**

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

**a. Simple SELECT statement**



The screenshot shows a database management tool interface. At the top, there are tabs for 'Query 1', 'SQL File 3', 'SQL File 17\*', and several 'patients' tabs. Below the tabs is a toolbar with various icons, including a 'Limit to 1000 rows' dropdown. The main area displays a SQL query: `1 • SELECT * FROM hospital_management.patients;`. Below the query, there is a 'Result Grid' section. The 'Result Grid' has a toolbar with 'Filter Rows:', 'Edit:', and 'Export/Import:' options. The grid itself contains the following data:

	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

On the right side of the 'Result Grid', there are two buttons: 'Result Grid' and 'Form Editor'.

**b. Where clause + IN/NOT IN:**

Query 1 SQL File 3 SQL File 17\* patients patients patients patients patients x SQLAdditions

Limit to 1000 rows

```

1 • SELECT patient_name, disease
2 FROM Patients
3 WHERE disease IN ('Fever', 'COVID-19');
4
5

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid

patient_name	disease
Chris Lee	COVID-19

Patients 3 x Read Only Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
10	13:16:14	TRUNCATE TABLE Patients	0 row(s) affected	0.047 sec
11	13:16:38	SELECT * FROM hospital_management.patients LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	13:19:07	INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.015 sec
13	13:19:11	SELECT * FROM hospital_management.patients LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
14	13:20:12	SELECT * FROM Patients LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
15	13:20:45	SELECT patient_name, disease FROM Patients WHERE disease IN ('Fever', 'COVID...	1 row(s) returned	0.000 sec / 0.000 sec

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

sql

Query 1 SQL File 3 SQL File 17\* patients patients patients patients patients x SQLAdditions

Limit to 1000 rows

```

1 • SELECT COUNT(patient_id) AS total_patients, AVG(age) AS average_age
2 FROM Patients;
3

```

Automatic context help is disabled. Use the toolbar to manually get help for the current caret position or to toggle automatic help.

Result Grid

total_patients	average_age
5	39,4000

Result 4 x Read Only Context Help Snippets

Output

Action Output

#	Time	Action	Message	Duration / Fetch
11	13:16:38	SELECT * FROM hospital_management.patients LIMIT 0, 1000	0 row(s) returned	0.000 sec / 0.000 sec
12	13:19:07	INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date...	5 row(s) affected Records: 5 Duplicates: 0 Warnings: 0	0.015 sec
13	13:19:11	SELECT * FROM hospital_management.patients LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
14	13:20:12	SELECT * FROM Patients LIMIT 0, 1000	5 row(s) returned	0.000 sec / 0.000 sec
15	13:20:45	SELECT patient_name, disease FROM Patients WHERE disease IN ('Fever', 'COVID...	1 row(s) returned	0.000 sec / 0.000 sec
16	13:21:30	SELECT COUNT(patient_id) AS total_patients, AVG(age) AS average_age FROM Pa...	1 row(s) returned	0.000 sec / 0.000 sec

### d. Group By + Having

The screenshot shows a SQL IDE with a query editor and a result grid. The query editor contains the following SQL code:

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

The result grid displays the following data:

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

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' input field. The result grid has a 'Result Grid' button and a 'Form Editor' button.

### e. Order By

The screenshot shows a SQL IDE with a query editor and a result grid. The query editor contains the following SQL code:

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

The result grid displays the following data:

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

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' input field. The result grid has a 'Result Grid' button and a 'Form Editor' button. The status bar at the bottom indicates 'Patients 7' and 'Read Only'.

### f. Views

The screenshot shows a SQL IDE with a query editor. The query editor contains the following SQL code:

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

The IDE interface includes a toolbar with icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' input field.

Query 1   SQL File 3   SQL File 17\*   patients   patients   patients   patients   patients

Limit to 1000 rows

```
1 • SELECT * FROM PatientInfo;
2
```

Result Grid   Filter Rows:   Export:   Wrap Cell Content:   Read Only

	patient_name	age	disease
▶	John Doe	45	Flu
	Jane Smith	32	Diabetes
	Mike Johnson	28	Asthma
	Emma Davis	54	Hypertension
	Chris Lee	38	COVID-19

PatientInfo 8 ×

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

Query 1   SQL File 3   SQL File 17\*   patients   patients   patients   patients   patients

Limit to 1000 rows

```
1 • SELECT patient_name, date_of_admission
2 FROM Patients
3 WHERE date_of_admission = CURRENT_DATE;
4
```

Result Grid   Filter Rows:   Export:   Wrap Cell Content:   Result Grid   Form Editor

	patient_name	date_of_admission
▶	Chris Lee	2024-10-05



## 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

Query 1 SQL File 3 SQL File 17\* patients patients patients patients patients

1 • `SELECT * FROM hospital_management.patients;`

Limit to 1000 rows

Result Grid

	patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_id
▶	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

Result Grid Form Editor

patients patients patients patients patients doctors doctors doctors

1 • `SELECT * FROM hospital_management.doctors;`

Limit to 1000 rows

Result Grid

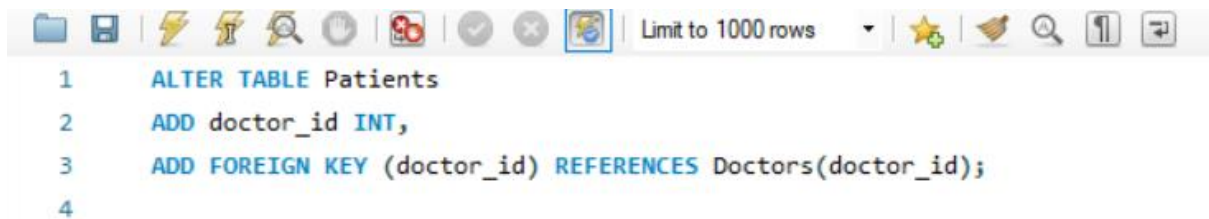
	doctor_id	doctor_name	specialization	phone
▶	1	Dr. Smith	Cardiologist	1234000000
	2	Dr. Brown	Endocrinologist	1234000001
	3	Dr. Green	Pulmonologist	1234000002
	4	Dr. Taylor	General Physician	1234000003
*	NULL	NULL	NULL	NULL

Result Grid Form Editor



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.



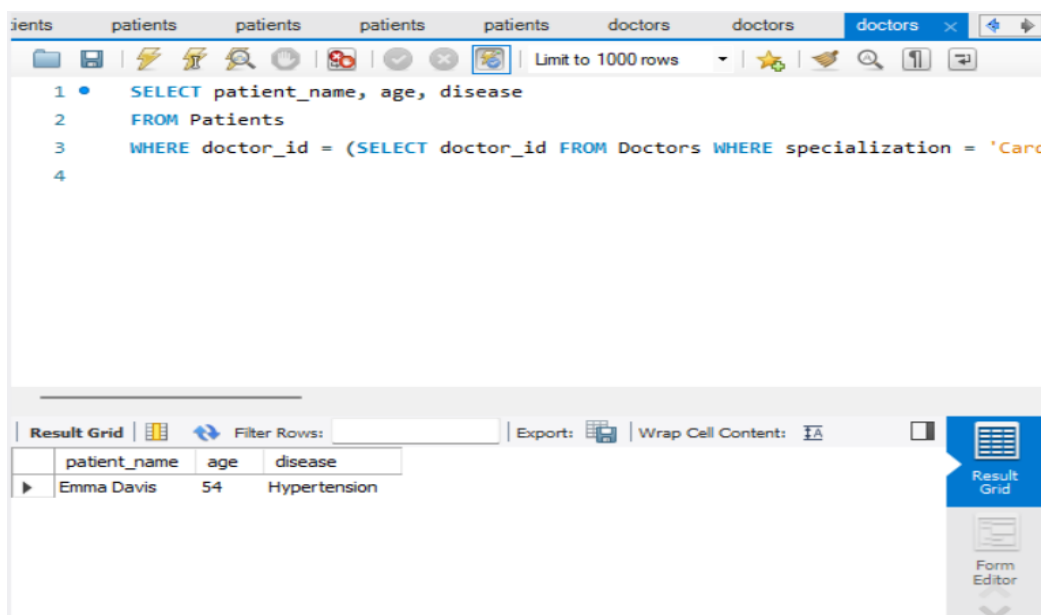
```
1 ALTER TABLE Patients
2 ADD doctor_id INT,
3 ADD FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id);
4
```

## 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

The screenshot shows a database query editor with a toolbar at the top. The query is as follows:

```
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
```

Below the query, the results are displayed in a table with the following columns: patient\_name, age, disease, doctor\_name, and specialization. The results are:

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

On the right side of the results table, there are two buttons: "Result Grid" and "Form Editor".

## Left join

The screenshot shows a database query editor with a toolbar at the top. The query is as follows:

```
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
```

Below the query, the results are displayed in a table with the following columns: patient\_name, age, disease, doctor\_name, and specialization. The results are:

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

On the right side of the results table, there are two buttons: "Result Grid" and "Form Editor".

## Right join

The screenshot shows a database query editor with a toolbar at the top. The query is as follows:

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

Below the query editor is the "Result Grid" section. It includes a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. The result grid displays the following data:

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

On the right side of the result grid, there are buttons for "Result Grid" and "Form Editor".

## Outer join

The screenshot shows a database query editor with a toolbar at the top. The query is as follows:

```
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 UNION
5 SELECT P.patient_name, P.age, P.disease, D.doctor_name, D.specialization
6 FROM Patients P
7 RIGHT JOIN Doctors D ON P.doctor_id = D.doctor_id;
8
```

Below the query editor is the "Result Grid" section. It includes a "Filter Rows:" input field, an "Export:" button, and a "Wrap Cell Content:" checkbox. The result grid displays the following data:

	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

On the right side of the result grid, there are buttons for "Result Grid" and "Form Editor".

# 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

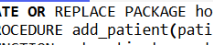


   EDITOR CH


queries.sql 42wqj695p 

```
1 DECLARE
2   patient_id NUMBER := 1001;
3   patient_name VARCHAR2(50) := 'John Doe';
4 BEGIN
5   DBMS_OUTPUT.PUT_LINE('Patient ID: ' || patient_id);
6   DBMS_OUTPUT.PUT_LINE('Patient Name: ' || patient_name);
7 END;
8 /
9
```

STDIN  
Input for the program (Optional)  
Output:  
Patient ID: 1001  
Patient Name: John Doe

## b. Packages

   EDITOR CH

queries.sql 42wqj695p 

```
1 CREATE OR REPLACE PACKAGE hospital_pkg IS
2   PROCEDURE add_patient(patient_name VARCHAR2, patient_age NUMBER);
3   FUNCTION get_patient_count RETURN NUMBER;
4 END hospital_pkg;
5 /
6 CREATE OR REPLACE PACKAGE BODY hospital_pkg IS
7   v_patient_count NUMBER := 0;
8
9   PROCEDURE add_patient(patient_name VARCHAR2, patient_age NUMBER) IS
10    BEGIN
11      v_patient_count := v_patient_count + 1;
12      DBMS_OUTPUT.PUT_LINE('Added patient: ' || patient_name);
13    END add_patient;
14
15   FUNCTION get_patient_count RETURN NUMBER IS
16    BEGIN
17      RETURN v_patient_count;
18    END get_patient_count;
19
20 END hospital_pkg;
21 /
22 DECLARE
23   patient_count NUMBER;
24 BEGIN
25   hospital_pkg.add_patient('Alice Green', 29);
26   hospital_pkg.add_patient('Bob Brown', 45);
27   patient_count := hospital_pkg.get_patient_count;
28   DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
29 END;
30 /
```

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

## **c. Procedures**

```
1  DECLARE
2    patient_count NUMBER;
3  BEGIN
4    patient_count := hospital_pkg.get_patient_count;
5    DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
6  END;
7  /
8
```

Statement processed.  
Total Patients: 0

## **d. Functions**

```
1  DECLARE
2    patient_count NUMBER;
3  BEGIN
4    patient_count := hospital_pkg.get_patient_count;
5    DBMS_OUTPUT.PUT_LINE('Total Patients: ' || patient_count);
6  END;
7  /
8
```

Statement processed.  
Total Patients: 0

# Practical -7

## Perform Exception handling in a PL/SQL block.

```
1  DECLARE
2      patient_id NUMBER := 1;          -- Example patient ID
3      patient_name VARCHAR2(50);      -- Variable to hold patient name
4      age NUMBER := 0;                -- Example variable for division by zero
5      result NUMBER;
6  BEGIN
7      -- Attempt to retrieve patient name
8      SELECT name INTO patient_name
9      FROM patients
10     WHERE id = patient_id;
11
12     -- Display patient name
13     DBMS_OUTPUT.PUT_LINE('Patient Name: ' || patient_name);
14
15     -- Example of division by zero to trigger an exception
16     result := 100 / age;
17     DBMS_OUTPUT.PUT_LINE('Result: ' || result);
18
19 EXCEPTION
20     -- Handle the NO_DATA_FOUND exception
21     WHEN NO_DATA_FOUND THEN
22         DBMS_OUTPUT.PUT_LINE('Error: Patient with ID ' || patient_id || ' was not found.');
```

```
14
15     -- Example of division by zero to trigger an exception
16     result := 100 / age;
17     DBMS_OUTPUT.PUT_LINE('Result: ' || result);
18
19 EXCEPTION
20     -- Handle the NO_DATA_FOUND exception
21     WHEN NO_DATA_FOUND THEN
22         DBMS_OUTPUT.PUT_LINE('Error: Patient with ID ' || patient_id || ' was not found.');
```

```
23
24     -- Handle division by zero
25     WHEN ZERO_DIVIDE THEN
26         DBMS_OUTPUT.PUT_LINE('Error: Division by zero occurred.');
```

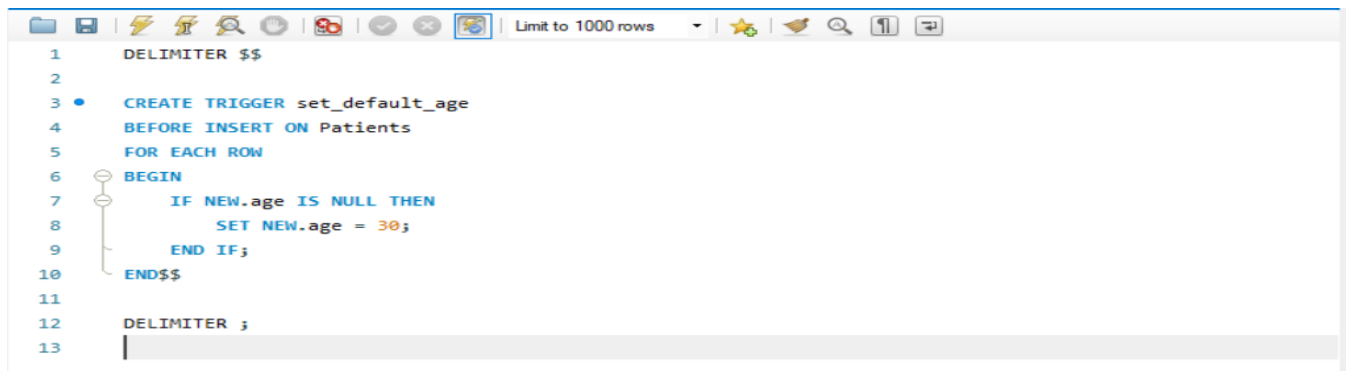
```
27
28     -- Handle any other exception
29     WHEN OTHERS THEN
30         DBMS_OUTPUT.PUT_LINE('An unexpected error occurred: ' || SQLERRM);
31 END;
32 /
33
```

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

## **Practical -9**

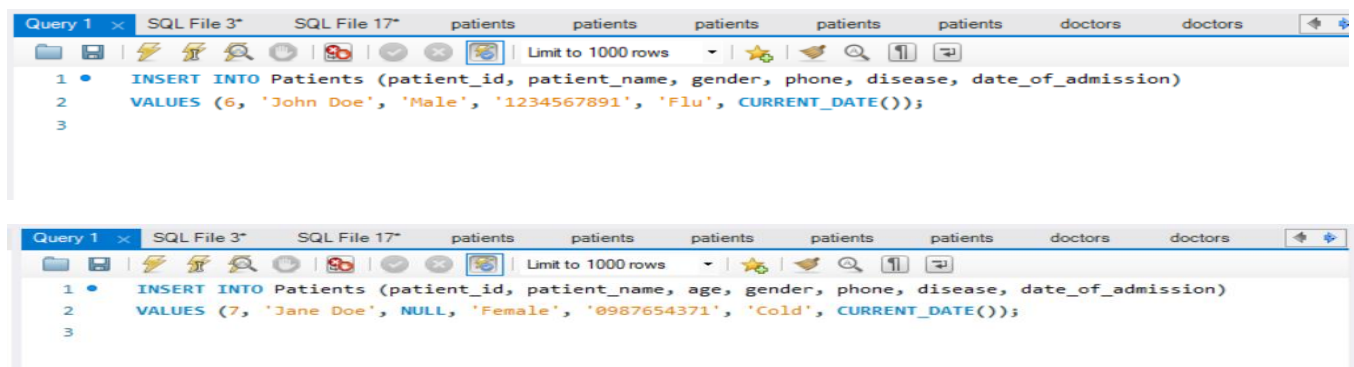
### **Implement Triggers in SQL.**

#### **Step 1: Create the Trigger**



```
1 DELIMITER $$
2
3 • CREATE TRIGGER set_default_age
4 BEFORE INSERT ON Patients
5 FOR EACH ROW
6 BEGIN
7     IF NEW.age IS NULL THEN
8         SET NEW.age = 30;
9     END IF;
10 END$$
11
12 DELIMITER ;
13
```

#### **Step 2: Insert Data**



```
Query 1 x SQL File 3* SQL File 17* patients patients patients patients patients doctors doctors
1 • INSERT INTO Patients (patient_id, patient_name, gender, phone, disease, date_of_admission)
2 VALUES (6, 'John Doe', 'Male', '1234567891', 'Flu', CURRENT_DATE());
3

Query 1 x SQL File 3* SQL File 17* patients patients patients patients patients doctors doctors
1 • INSERT INTO Patients (patient_id, patient_name, age, gender, phone, disease, date_of_admission)
2 VALUES (7, 'Jane Doe', NULL, 'Female', '0987654321', 'Cold', CURRENT_DATE());
3
```

#### **Step 3: Verify the Output**

Query 1 x SQL File 3\* SQL File 17\* patients patients patients patients patients doctors doctors

Limit to 1000 rows

```

1 • SELECT * FROM Patients WHERE patient_id IN (6, 7);
2

```

Result Grid

	patient_id	patient_name	age	gender	phone	disease	date_of_admission	doctor_assigned	doctor_id
▶	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

Form Editor

## **Practical -10**

**Implement the following transaction statements:**

**a. Commit**

**b. Rollback**

**c. Savepoint**

**a. Commit**

**c. Savepoint**

Query 1 x SQL File 3 SQL File 17\* patients patients patients patients patients doctors doctors

Limit to 1000 rows

```

1  -- Step 1: Start the transaction
2 • START TRANSACTION;
3
4  -- Step 2: Update patient to assign a doctor
5 • UPDATE Patients SET doctor_id = 2 WHERE patient_id = 1;
6
7  -- Step 3: Create a savepoint after the first update
8 • SAVEPOINT Update1;
9
10 -- Step 4: Update another patient
11 • UPDATE Patients SET doctor_id = 1 WHERE patient_id = 2;
12
13 -- Step 5: Check whether to commit or rollback
14 -- If something goes wrong, rollback to savepoint
15 -- ROLLBACK TO Update1;
16
17 -- Step 6: Commit the transaction if everything is fine
18 • COMMIT;
19

```



Query 1 x SQL File 3 SQL File 17\* patients patients patients patients patients doctors doctors

Limit to 1000 rows

```
1 • SELECT * FROM Patients;
2
```

Result Grid

	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

Result Grid  
Form Editor  
Field Types

## **b. Rollback**

Query 1 x SQL File 3 SQL File 17\* patients patients patients patients patients doctors doctors

Limit to 1000 rows

```
1 • ROLLBACK;
2
3
```

Query 1 x SQL File 3 SQL File 17\* patients patients patients patients patients doctors doctors

Limit to 1000 rows

```
1 • SELECT * FROM Patients;
2
```

Result Grid

	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

Result Grid  
Form Editor  
Field Types