

DBMS MINOR PROJECT

COURSE CODE:MC-209

MATHEMATICS AND COMPUTING

B.TECH 3rd SEMESTER



DELHI TECHNOLOGICAL UNIVERSITY

DEPARTMENT APPLIED MATHEMATICS

SUBMITTED TO:- Ms.Himani Pokhriyal

SUBMITTED BY:- Lokesh Godara 23/MC/82

Krrishnav Gupta 23/MC/78

Acknowledgment

We are deeply grateful to Ms. Himani Pokhriyal for her invaluable mentorship and guidance throughout the course of our Hospital Management System project. This endeavor would not have been possible without her expertise, dedication, and encouragement. Ms. Pokhriyal's insightful feedback, patient guidance, and support provided us with the knowledge and confidence to navigate the challenges of this project, allowing us to refine our skills and better understand the intricacies of hospital management systems.

From the initial stages of planning to the final implementation, Ms. Pokhriyal's expertise helped us in making informed decisions, understanding complex requirements, and overcoming technical challenges. Her thorough approach to teaching and her readiness to offer assistance whenever needed created a supportive learning environment that encouraged us to give our best and strive for excellence.

Thank you, ma'am, for your invaluable contributions to our project and for your unwavering support. It has been a privilege to work under your guidance, and we are honored to have had the opportunity to learn from you.

Table of content

- 1. Title Page**
- 2. Acknowledgment**
- 3. Abstract**
- 4. Introduction**
- 5. System Design**
 - 1. 5.1 ER Diagram**
 - 2. 5.2 Relationships and Data Flow**
 - 3. 5.3 Data Integrity and Constraints**
 - 4. 5.4 System Workflow**
- 6. Implementation**
 - 1. 6.1 Database Setup**
 - 2. 6.2 Table Structure and Relationships**
 - 3. 6.3 Data Insertion and Population**
 - 4. 6.4 SQL Commands and Queries**
- 7. Queries and Reports**
- 8. Testing and Validation**
 - 1. 8.1 Unit Testing**
 - 2. 8.2 Validation of Constraints**
 - 3. 8.3 Transaction Testing**
 - 4. 8.4 Validation of Queries and Reports**
 - 5. 8.5 Access Control Testing**
- 9. Conclusion**

Abstract

The Hospital Management System (HMS) is designed to streamline and automate the processes within a hospital, ensuring efficient management of patient records, appointments, medical histories, staff, and resources. The system leverages a Database Management System (DBMS) to securely store, manage, and retrieve hospital-related data, providing an integrated platform for healthcare providers to manage essential functions and improve patient care quality.

The primary objective of the HMS is to enable seamless coordination between departments, reduce paperwork, and minimize human errors by centralizing data access. This system includes modules for patient registration, doctor and staff management, appointments, billing, inventory management, and reports generation. By adopting relational databases and enforcing database integrity and security constraints, the system ensures data consistency, confidentiality, and quick access for authorized users.

The HMS aims to facilitate real-time data processing, allowing medical professionals and administrative staff to perform their duties more effectively. In addition, it provides scalability to accommodate future growth and integration with other healthcare systems. The Hospital Management System thus serves as a comprehensive tool for improving operational efficiency, data accuracy, and patient satisfaction within a hospital setting.

Introduction

The **Hospital Management System (HMS)** project, developed by Lokesh Godara and Krrishnav, is designed to streamline and automate the essential processes within a hospital environment. Efficient management of patient records, scheduling, billing, and other administrative tasks is vital to delivering high-quality healthcare. This system provides a centralized platform that reduces paperwork, minimizes human error, and enhances coordination among departments, ultimately improving the patient experience.

Built with a focus on data integrity, security, and ease of use, the HMS incorporates key modules such as patient registration, doctor and staff management, appointment scheduling, billing, and inventory management. By using Database Management System (DBMS) principles, we have ensured that critical data is securely stored and readily accessible for authorized personnel. This system allows healthcare providers to retrieve and manage information quickly, enabling them to focus more on patient care.

In developing this project, we applied essential DBMS concepts, including normalization and transaction management, to ensure efficient data handling and high performance. The Hospital Management System is thus an effective tool for improving operational efficiency and enhancing the quality of care within a hospital.

No. Of Experiments	Name of Experiment	Date	Signature
1	System Design of the Project, along with its E-R Diagram	6/08/2024	
2	Implement the following DDL statements: <ol style="list-style-type: none"> Create Create table with constraints (NOT NULL, UNIQUE, DEFAULT, CHECK, PRIMARY KEY, FOREIGN KEY) Alter Table: <ol style="list-style-type: none"> Add column Drop column Add/drop constraint Rename column Drop Table 	13/08/24	
3	Implement the following DML statements: <ol style="list-style-type: none"> Insert Update Delete Truncate 	20/08/24	
4	Implement the following SELECT statements: <ol style="list-style-type: none"> Simple SELECT statement Where clause+ IN/NOT IN Aggregate functions Group by + Having Order by Views Inbuilt Functions (e.g., Date) 	27/08/24	

5	Implement and perform Nested Queries along with Joins (Inner join, Outer join, Left join, Right join)	3/09/24	
6	Introduction to PL/SQL. Create a PL/SQL block and implement the following: a. Variables	10/09/24	
	b. Packages c. Procedures d. Functions		
7	Perform Exception handling in a PL/SQL block.	8/10/24	
8	Project Report with Code and Screenshots	13/10/24	
9	Implement Triggers in SQL.	22/10/24	
10	Implement the following transaction statements: a. Commit b. Rollback c. Savepoint	22/10/24	

Practical -1

Sytem design 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.

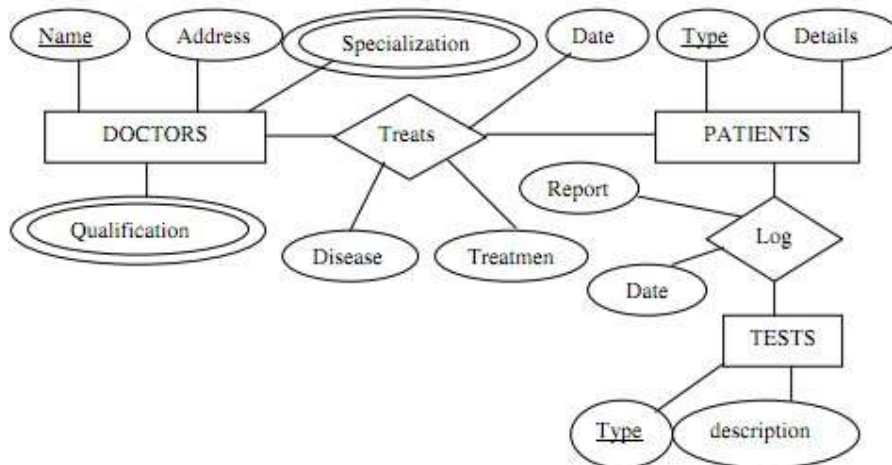
6. Reporting:

- 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:

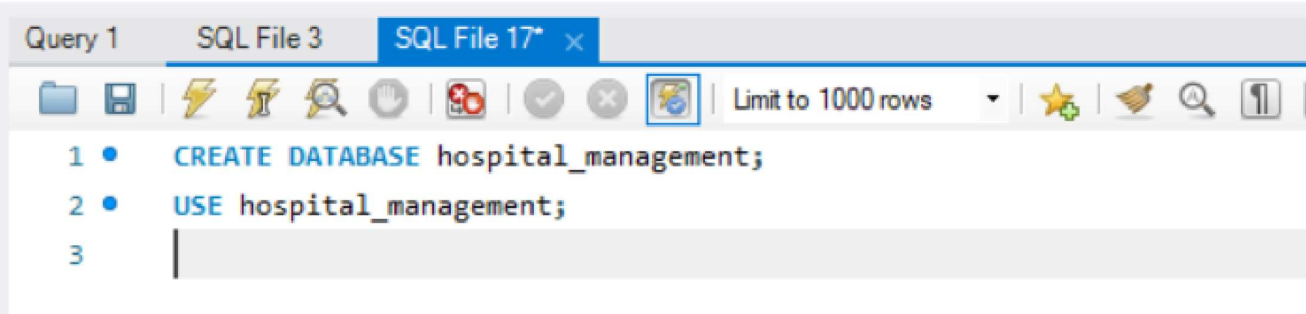
i. Add column ii. Drop

column iii. Add/drop

constraint iv. Rename

column d. Drop Table

a. Create



The screenshot shows a SQL IDE interface with three tabs: 'Query 1', 'SQL File 3', and 'SQL File 17*'. The 'SQL File 17*' tab is active, displaying a query editor with the following SQL statements:

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

The editor includes a toolbar with various icons for file operations, execution, and search. A dropdown menu shows 'Limit to 1000 rows'.

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

c. Alter Table:

i. Add column

The screenshot shows a SQL IDE with three tabs: 'Query 1', 'SQL File 3', and 'SQL File 17*'. The 'Query 1' tab is active, displaying the following SQL code:

```
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
```

Below this, the 'SQL File 17*' tab is active, showing an ALTER statement:

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

ii. Drop column

The screenshot shows a SQL IDE with multiple tabs. The active tab displays the following SQL code:

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

iii. Add/drop constraint

Add a NOT NULL constraint on doctor_assigned

The screenshot shows a SQL IDE with three tabs: 'Query 1', 'SQL File 3', and 'SQL File 17*'. The 'Query 1' tab is active, displaying the following SQL code:

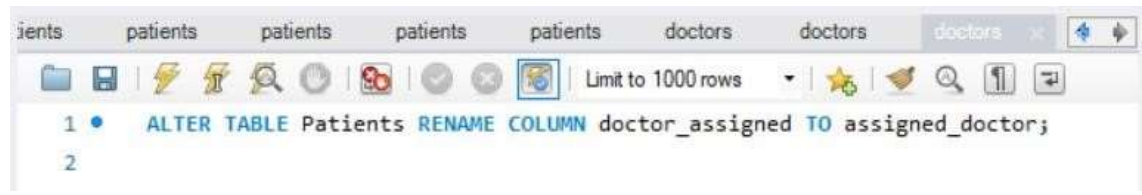
```
1 • ALTER TABLE Patients MODIFY doctor_assigned VARCHAR(100) NOT NULL;  
2
```

Drop the NOT NULL constraint

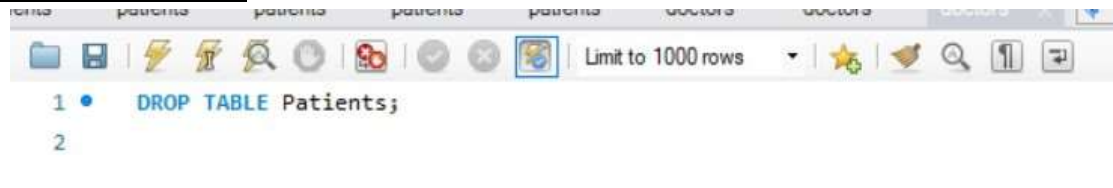
The screenshot shows a SQL IDE with three tabs: 'Query 1', 'SQL File 3', and 'SQL File 17*'. The 'Query 1' tab is active, displaying the following SQL code:

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

iv. Rename column



d. Drop Table



Practical -3

Implement the following DML statements: a.

Insert

b. Update

c. Delete

d. Truncate

a. Insert

The screenshot displays a database management interface with two SQL queries and their results.

Query 1:

```
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
```

Query 2:

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

Result Grid:

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

The interface includes a toolbar with various icons for file operations, a 'Limit to 1000 rows' dropdown, and a 'Filter Rows' input field. The 'Result Grid' tab is active, showing the data returned by the queries. A 'Form Editor' button is visible on the right side.

b. Update

Query 1 SQL File 3 SQL File 17* patients x

Limit to 1000 rows

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

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

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

Limit to 1000 rows

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

The screenshot shows the SQL query editor with the following query entered:

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

The query is highlighted in blue. The editor interface includes a toolbar with various icons for file operations, editing, and execution. The top of the editor shows tabs for 'Query 1', 'SQL File 3', 'SQL File 17*', and 'patients'.

Output					Context Help	Snippets
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

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

Limit to 1000 rows

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

Result Grid

	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

Result Grid

Form Editor

b. Where clause + IN/NOT IN:

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

Limit to 1000 rows

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

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

Form Editor

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

The screenshot shows a SQL IDE with a query editor and a results pane. The query is:

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

The results pane displays the following data:

total_patients	average_age
5	39.4000

The output pane shows the execution log:

#	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 results pane. The query is:

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

The results pane displays the following data:

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

e. Order By

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: |

	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 x Read Only

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

Limit to 1000 rows

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

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

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

Patients 7 x Read Only

Result Grid
Form Editor

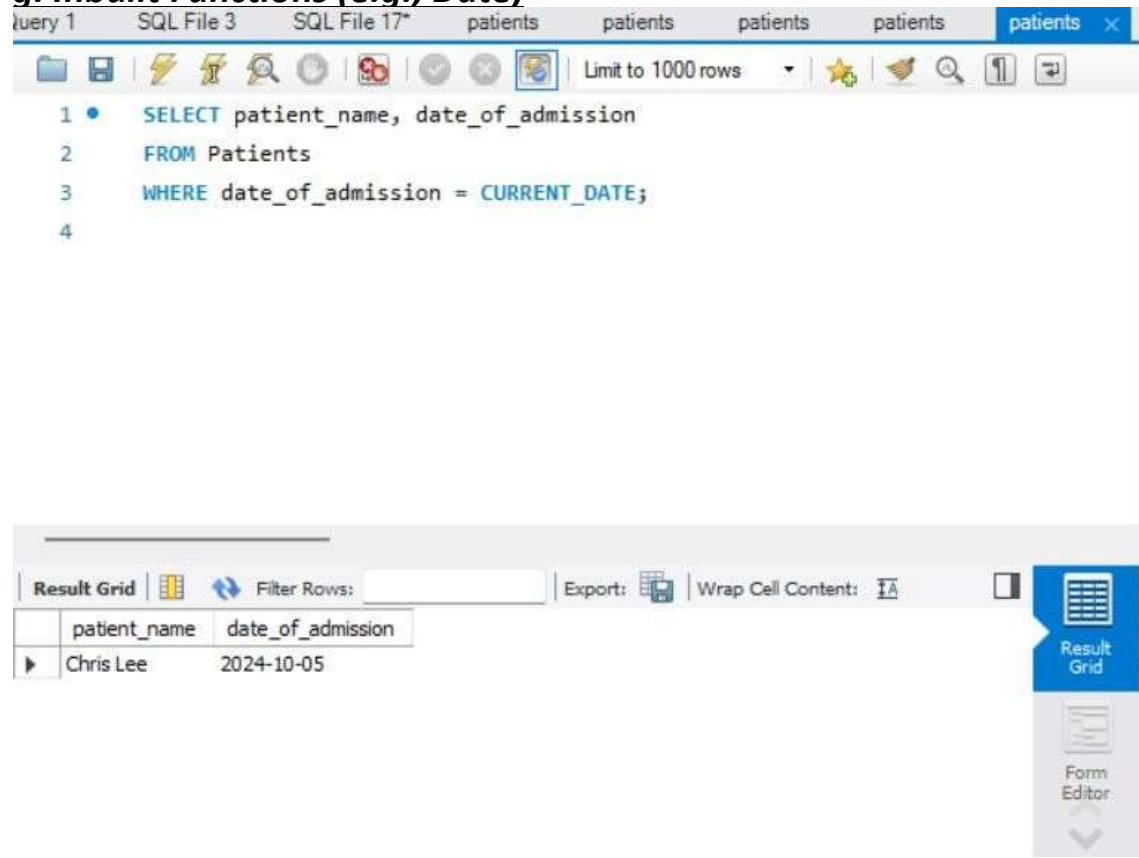
f. Views

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

Limit to 1000 rows

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

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



The screenshot shows a SQL IDE interface. The top toolbar includes icons for file operations, execution, and a 'Limit to 1000 rows' dropdown. The query editor contains the following SQL code:

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

Below the query editor, the 'Result Grid' is displayed with the following data:

	patient_name	date_of_admission
▶	Chris Lee	2024-10-05

On the right side, there is a vertical toolbar with buttons for 'Result Grid' and 'Form Editor'.

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

Limit to 1000 rows

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

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

Form Editor

patients patients patients patients patients doctors doctors doctors

Limit to 1000 rows

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

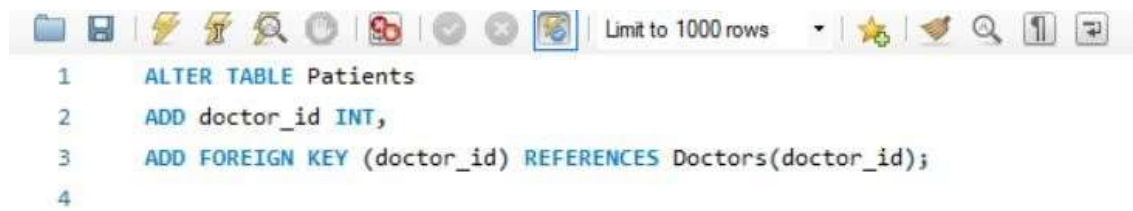
Result Grid

	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

Form Editor

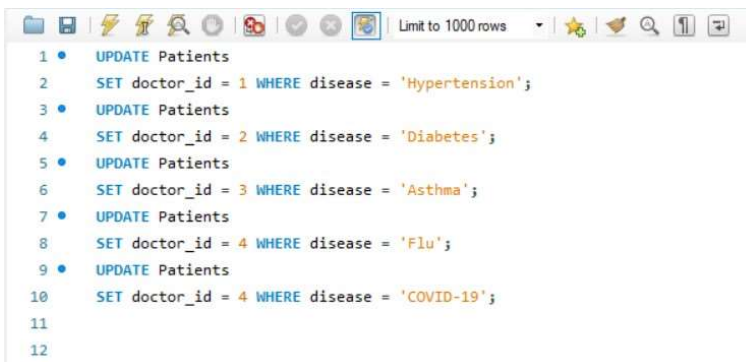
Add a doctor_id to the Patients Table (for relational queries)

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



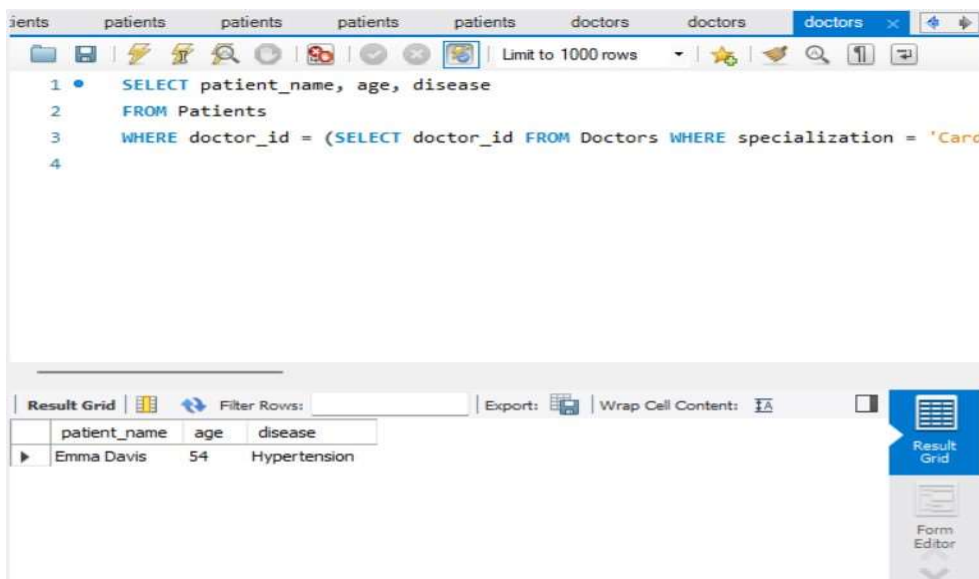
```
1 ALTER TABLE Patients
2 ADD doctor_id INT,
3 ADD FOREIGN KEY (doctor_id) REFERENCES Doctors(doctor_id);
4
```

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 = 'Cardiologist');
4
```

Result Grid

patient_name	age	disease
▶ Emma Davis	54	Hypertension

Export: | Wrap Cell Content: | Result Grid | Form Editor

Inner join

Limit to 1000 rows

```
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

	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

Result Grid
Form Editor

Left join

Limit to 1000 rows

```
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

	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

Result Grid
Form Editor

Right join

patients patients patients patients patients doctors doctors doctors

Limit to 1000 rows

```

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

```

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

	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

Result Grid
Form Editor

Outer join

patients patients patients patients patients doctors doctors doctors

Limit to 1000 rows

```

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

```

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

Result Grid
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

OneCompiler

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

```
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 v DECLARE
2   patient_count NUMBER;
3 v 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 v DECLARE
2   patient_count NUMBER;
3 v 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', '0987654371', '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

```

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

```

	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

Result Grid

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
```

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

b. Rollback

