```
-- PRACTICAL 2: SQL DDL OBJECTS

-- 1️⃣ Create Database
CREATE DATABASE CollegeDB2;
USE CollegeDB2;

-- 2️⃣ Create Department Table
CREATE TABLE Department (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(30) UNIQUE,
    location VARCHAR(20) NOT NULL
);

-- 3️⃣ Create Student Table with Constraints
CREATE TABLE Student (
    roll_no INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50) NOT NULL,
    dept_id INT,
    marks INT CHECK (marks BETWEEN 0 AND 100),
    city VARCHAR(30) DEFAULT 'Nashik',
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);

-- 4️⃣ Insert Data into Department Table
INSERT INTO Department (dept_name, location) VALUES
('Computer', 'Building A'),
('IT', 'Building B'),
('ENTC', 'Building C');
```

```
-- 5️⃣ Insert Data into Student Table
INSERT INTO Student (name, dept_id, marks, city) VALUES
('Akash', 1, 85, 'Nashik'),
('Neha', 2, 92, 'Pune'),
('Rohan', 1, 78, 'Nashik'),
('Meera', 3, 67, 'Mumbai'),
('Anjali', 2, 88, DEFAULT);

-- 6️⃣ Create a View
CREATE VIEW Student_View AS
SELECT s.roll_no, s.name, s.marks, d.dept_name
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id;

-- 7️⃣ Create an Index for faster search
CREATE INDEX idx_marks ON Student(marks);

-- 8️⃣ Create a Sequence Equivalent (MySQL uses AUTO_INCREMENT)
-- Simulating a sequence table
CREATE TABLE roll_sequence (
    id INT AUTO_INCREMENT PRIMARY KEY
);
INSERT INTO roll_sequence VALUES (), (), ();

-- 9️⃣ Create a Synonym (Only in Oracle)
-- In MySQL, use an alias instead:
-- SELECT * FROM Student s;
```

```
-- ************************************
-- OUTPUT QUERIES
-- ************************************
-- View all Students
SELECT * FROM Student;

-- View Created View
SELECT * FROM Student_View;

-- Check Indexes
SHOW INDEX FROM Student;

-- Verify Foreign Key Relation
SELECT s.name, d.dept_name
FROM Student s
JOIN Department d ON s.dept_id = d.dept_id;
```

🔹 **1. What is DDL in SQL?**

Answer:
DDL stands for **Data Definition Language**.
It is used to define or modify the structure of the database, such as **CREATE, ALTER, DROP, TRUNCATE, RENAME** commands.

🔹 **2. What is the difference between DDL and DML?**

Answer:

- **DDL** changes the structure (tables, views, indexes).
- **DML** changes the data inside the tables (INSERT, UPDATE, DELETE).

🔹 3. What are constraints in SQL?

Answer:
Constraints are rules applied to table columns to ensure data integrity.
Examples: **PRIMARY KEY, FOREIGN KEY, UNIQUE, NOT NULL, CHECK, DEFAULT**.

🔹 4. What is a Primary Key?

Answer:
A **Primary Key** uniquely identifies each record in a table and cannot be null or duplicated.

🔹 5. What is a Foreign Key?

Answer:
A **Foreign Key** links two tables together.
It enforces a relationship between a column in one table and the primary key of another.

🔹 6. What is a View?

Answer:
A **View** is a **virtual table** based on the result of an SQL query.
It does not store data physically, it just displays data from one or more tables.

🔹 7. Why do we use Views?

Answer:

- To **simplify complex queries**.
- To **enhance security** (restrict access to certain columns).
- To **reuse common queries** easily.

🔹 8. What is an Index and why is it used?

Answer:
An **Index** improves the **speed of data retrieval** operations on a table.
It's like an index in a book — used to find information faster.

🔹 9. What is AUTO_INCREMENT in MySQL?

Answer:
It automatically generates a unique number whenever a new record is inserted into a table (similar to **SEQUENCE** in Oracle).

🔹 10. What is a CHECK constraint?

Answer:
It limits the values entered in a column.
Example: CHECK (marks BETWEEN 0 AND 100) ensures marks are within that range.

🔹 11. What is a DEFAULT constraint?

Answer:
It assigns a **default value** to a column if no value is provided during insert.
Example: city VARCHAR(30) DEFAULT 'Nashik'.

🔹 12. What is a UNIQUE constraint?

Answer:
Ensures all values in a column are **different**, but unlike primary key, it can allow **NULL**.

🔹 13. What is a Synonym in SQL?

Answer:
A **Synonym** is an **alias name** for a database object like a table or view.
(Available in **Oracle**, not MySQL).

🔹 14. Can a table have more than one foreign key?

Answer:
Yes, a table can have multiple foreign keys referring to different parent tables.

🔹 15. What's the difference between a View and a Table?

Answer:

- A **Table** physically stores data.

- A **View** is a virtual representation of data from one or more tables.

🔹 **16. What is the use of the Department table in this practical?**

Answer:
It demonstrates a **parent table** for establishing a **foreign key relationship** with the Student table.

🔹 **17. What does SHOW INDEX FROM Student; do?**

Answer:
It displays all indexes created on the **Student** table, including index name, type, and columns involved.

🔹 **18. Why did we create Student_View?**

Answer:
To display student details along with their department names easily using a **JOIN** inside a view.

🔹 **19. What happens if you delete a record from Department that has related students?**

Answer:
It will give a **foreign key constraint error**, unless **ON DELETE CASCADE** is defined.

🔹 **20. Why did we use JOIN in the view?**

Answer:
To **combine related data** from the Student and Department tables based on the **dept_id** relationship.

```
-- PRACTICAL 3: SQL DML STATEMENTS

-- 1️⃣ Create and Use Database
CREATE DATABASE CollegeDB3;
USE CollegeDB3;

-- 2️⃣ Create Tables
CREATE TABLE Department (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(30) UNIQUE,
    location VARCHAR(20)
);

CREATE TABLE Student (
    roll_no INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    dept_id INT,
    marks INT,
    city VARCHAR(30),
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);

-- 3️⃣ Insert Data into Department Table
INSERT INTO Department (dept_name, location) VALUES
('Computer', 'Building A'),
('IT', 'Building B'),
('ENTC', 'Building C');
```

```
-- 4️⃣ Insert Data into Student Table
INSERT INTO Student (name, dept_id, marks, city) VALUES
('Akash', 1, 85, 'Nashik'),
('Neha', 2, 92, 'Pune'),
('Rohan', 1, 78, 'Nashik'),
('Meera', 3, 67, 'Mumbai'),
('Anjali', 2, 88, 'Pune');

-- ************************************
-- 10 SQL QUERIES USING DML STATEMENTS
-- ************************************

-- 1️⃣ Display all students
SELECT * FROM Student;

-- 2️⃣ Display students having marks greater than 80
SELECT name, marks FROM Student WHERE marks > 80;

-- 3️⃣ Display students from Nashik or Pune
SELECT name, city FROM Student WHERE city IN ('Nashik', 'Pune');

-- 4️⃣ Increase marks by 5 for Computer department students
UPDATE Student
SET marks = marks + 5
WHERE dept_id = (SELECT dept_id FROM Department WHERE dept_name = 'Computer');

-- 5️⃣ Delete students having marks less than 70
```

```
SET SQL_SAFE_UPDATES = 0;

DELETE FROM Student WHERE marks < 70;


-- 6️⃣ Display average marks of each department

SELECT d.dept_name, AVG(s.marks) AS avg_marks

FROM Student s

JOIN Department d ON s.dept_id = d.dept_id

GROUP BY d.dept_name;


-- 7️⃣ Display the highest marks among all students

SELECT MAX(marks) AS highest_marks FROM Student;


-- 8️⃣ Display students whose name starts with 'A'

SELECT * FROM Student WHERE name LIKE 'A%';


-- 9️⃣ Use ORDER BY to show students sorted by marks descending

SELECT name, marks FROM Student ORDER BY marks DESC;


-- 🔟 Display city-wise number of students

SELECT city, COUNT(*) AS total_students

FROM Student

GROUP BY city;
```

🔹 1. What is DML in SQL?

Answer:
DML stands for **Data Manipulation Language**.
It is used to manage data inside tables.
Examples: INSERT, UPDATE, DELETE, SELECT.

🔹 2. What is the difference between DDL and DML?

Answer:

- **DDL** changes the **structure** of the database (e.g., CREATE, ALTER).
- **DML** changes the **data** inside tables (e.g., INSERT, UPDATE).

🔹 3. What is the purpose of the INSERT statement?

Answer:
It is used to **add new records** (rows) into a table.
Example:

```
INSERT INTO Student (name, marks) VALUES ('Akash', 85);
```

🔹 4. What is the use of the UPDATE statement?

Answer:
It is used to **modify existing records** in a table.
Example:

```
UPDATE Student SET marks = marks + 5 WHERE city = 'Nashik';
```

🔹 5. What is the DELETE statement used for?

Answer:
It is used to **remove records** from a table.
Example:

```
DELETE FROM Student WHERE marks < 70;
```

🔹 6. Why do we use SET SQL_SAFE_UPDATES = 0;?

Answer:
In MySQL, it disables safe mode to allow updates or deletes **without a key column** in the WHERE clause.

🔹 **7. What is a subquery?**

Answer:
A **subquery** is a query inside another query.
Example:

```
WHERE dept_id = (SELECT dept_id FROM Department WHERE dept_name='Computer');
```

🔹 **8. What is the purpose of GROUP BY in SQL?**

Answer:
It groups rows that have the same values into summary rows.
Used with aggregate functions like **COUNT**, **AVG**, **MAX**, **SUM**.

🔹 **9. What are aggregate functions in SQL?**

Answer:
Functions that perform a calculation on a set of values and return a single value:

- AVG()
- SUM()
- MAX()
- MIN()
- COUNT()

🔹 **10. What does the ORDER BY clause do?**

Answer:
It sorts the result set in **ascending or descending** order.
Example:

```
SELECT * FROM Student ORDER BY marks DESC;
```

🔹 **11. What does the LIKE operator do?**

Answer:
It is used for **pattern matching** in text.
Example:
LIKE 'A%' finds all names starting with ‘A’.

🔹 **12. What does the IN operator do?**

Answer:
It allows checking if a value matches any value in a list.
Example:

WHERE city IN ('Nashik', 'Pune');

🔹 **13. What is the use of the JOIN clause in this practical?**

Answer:
It combines data from **Student** and **Department** tables based on their **dept_id** relationship.

🔹 **14. What is the output of this query?**

SELECT MAX(marks) AS highest_marks FROM Student;

Answer:
It displays the **highest marks** scored among all students.

🔹 **15. What is the difference between DELETE and TRUNCATE?**

Answer:

- DELETE removes specific rows (DML command).
- TRUNCATE removes all rows quickly (DDL command).

🔹 **16. What happens if we don’t write a WHERE clause in UPDATE or DELETE?**

Answer:
All records in the table will be updated or deleted — which can lead to **data loss**.

🔹 **17. What is the use of AVG() function here?**

Answer:
To calculate **average marks** of each department using GROUP BY.

🔹 **18. How do you find the number of students in each city?**

Answer:
Using COUNT() with GROUP BY:

SELECT city, COUNT(*) FROM Student GROUP BY city;

🔹 **19. What is a correlated subquery?**

Answer:
A subquery that refers to a column from the outer query — it executes once per row.

🔹 **20. Can a table have multiple foreign keys?**

Answer:
Yes, if it needs to reference multiple parent tables.

```
-- ************************************
-- PRACTICAL 4: Joins, Subqueries and Views
-- ************************************

-- 1️⃣ Create and Use Database
CREATE DATABASE CollegeDB4;
USE CollegeDB4;

-- 2️⃣ Create Department Table
CREATE TABLE Department (
    dept_id INT PRIMARY KEY AUTO_INCREMENT,
    dept_name VARCHAR(30),
    location VARCHAR(20)
);

-- 3️⃣ Create Student Table
CREATE TABLE Student (
    roll_no INT PRIMARY KEY AUTO_INCREMENT,
    name VARCHAR(50),
    dept_id INT,
    marks INT,
    city VARCHAR(30),
    FOREIGN KEY (dept_id) REFERENCES Department(dept_id)
);

-- 4️⃣ Insert Data into Department Table
INSERT INTO Department (dept_name, location) VALUES
('Computer', 'Building A'),
```

```
('IT', 'Building B'),
('ENTC', 'Building C');

-- 5️⃣ Insert Data into Student Table
INSERT INTO Student (name, dept_id, marks, city) VALUES
('Akash', 1, 85, 'Nashik'),
('Neha', 2, 92, 'Pune'),
('Rohan', 1, 78, 'Nashik'),
('Meera', 3, 67, 'Mumbai'),
('Anjali', 2, 88, 'Pune'),
('Kiran', NULL, 70, 'Delhi');  -- Student without department

-- ************************************
-- PART A: JOINS (All Types)
-- ************************************

-- 1️⃣ INNER JOIN: Students with their department
SELECT s.name, s.marks, d.dept_name
FROM Student s
INNER JOIN Department d ON s.dept_id = d.dept_id;

-- 2️⃣ LEFT JOIN: All students, including those without departments
SELECT s.name, d.dept_name
FROM Student s
LEFT JOIN Department d ON s.dept_id = d.dept_id;

-- 3️⃣ RIGHT JOIN: All departments, even if no student enrolled
SELECT s.name, d.dept_name
```

```
FROM Student s
RIGHT JOIN Department d ON s.dept_id = d.dept_id;

-- 4️⃣ FULL JOIN (MySQL does not support directly – use UNION)
SELECT s.name, d.dept_name
FROM Student s LEFT JOIN Department d ON s.dept_id = d.dept_id
UNION
SELECT s.name, d.dept_name
FROM Student s RIGHT JOIN Department d ON s.dept_id = d.dept_id;

-- 5️⃣ SELF JOIN Example (Not very useful here, so simulate mentorship)
SELECT s1.name AS Student1, s2.name AS Student2
FROM Student s1, Student s2
WHERE s1.city = s2.city AND s1.roll_no <> s2.roll_no;

-- ************************************
-- PART B: SUBQUERIES
-- ************************************

-- 6️⃣ Display students having marks greater than the average
SELECT name, marks
FROM Student
WHERE marks > (SELECT AVG(marks) FROM Student);

-- 7️⃣ Find students who belong to the same department as 'Akash'
SELECT name FROM Student
WHERE dept_id = (SELECT dept_id FROM Student WHERE name = 'Akash');
```

```
-- 8️⃣ Display department name for student ‘Neha’ using subquery
SELECT dept_name
FROM Department
WHERE dept_id = (SELECT dept_id FROM Student WHERE name = 'Neha');

-- 9️⃣ Find students who have the highest marks
SELECT name, marks
FROM Student
WHERE marks = (SELECT MAX(marks) FROM Student);

-- ************************************
-- PART C: VIEW CREATION
-- ************************************

-- 🔟 Create a view of student name, dept, and marks
CREATE VIEW StudentInfo AS
SELECT s.roll_no, s.name, s.marks, d.dept_name
FROM Student s
LEFT JOIN Department d ON s.dept_id = d.dept_id;

-- Display data from the view
SELECT * FROM StudentInfo;
```

🔹 1. What is a JOIN in SQL?

Answer: A **JOIN** is used to combine rows from two or more tables based on a related column (usually a **foreign key**).

🔹 2. What are the different types of JOINs?

Answer:

1. **INNER JOIN** – returns records that have matching values in both tables.
2. **LEFT JOIN** – returns all records from the left table and matching ones from the right.
3. **RIGHT JOIN** – returns all records from the right table and matching ones from the left.
4. **FULL JOIN** – combines results of LEFT and RIGHT joins (all records).
5. **SELF JOIN** – joins a table with itself.

🔹 3. Explain INNER JOIN with example.

Answer:
INNER JOIN returns only matching records.

```
SELECT s.name, d.dept_name
FROM Student s
INNER JOIN Department d ON s.dept_id = d.dept_id;
```

🔹 4. What is the difference between LEFT JOIN and RIGHT JOIN?

Answer:

- **LEFT JOIN** → All records from left table + matched ones from right.
- **RIGHT JOIN** → All records from right table + matched ones from left.

🔹 5. What is a FULL JOIN?

Answer:
It returns **all records** when there is a match in either left or right table.
MySQL doesn't support it directly — we use:

LEFT JOIN ... UNION ... RIGHT JOIN

🔹 **6. What is a SELF JOIN?**

Answer:
A table joins with itself to compare records within the same table.
Example:

SELECT s1.name, s2.name

FROM Student s1, Student s2

WHERE s1.city = s2.city AND s1.roll_no <> s2.roll_no;

🔹 **7. What is a Subquery?**

Answer:
A **Subquery** is a query inside another query.
Used to fetch data that will be used in the main query's condition.

🔹 **8. What are the types of subqueries?**

Answer:

1. **Single-row subquery** – returns one value.
2. **Multi-row subquery** – returns multiple values.
3. **Correlated subquery** – executes once for each row of the outer query.

🔹 **9. Example of a subquery in this practical.**

Answer:

SELECT name, marks

FROM Student

WHERE marks > (SELECT AVG(marks) FROM Student);

👉 Finds students scoring above average.

🔹 **10. What is the difference between a subquery and a join?**

Answer:

- **JOIN** → Combines data from multiple tables.
- **SUBQUERY** → Uses one query's result inside another query.

🔹 11. What is a VIEW?

Answer:
A **VIEW** is a **virtual table** that displays data from one or more tables.
Example:

CREATE VIEW StudentInfo AS

SELECT s.name, d.dept_name, s.marks

FROM Student s LEFT JOIN Department d ON s.dept_id = d.dept_id;

🔹 12. Why do we use Views?

Answer:

- To **simplify complex queries**.
- To **improve data security**.
- To **present specific data** to users.

🔹 13. Can we update a view?

Answer:
Yes, if the view is based on a **single table** and does not contain aggregate functions or GROUP BY.

🔹 14. What does this query return?

SELECT name, marks FROM Student

WHERE marks > (SELECT AVG(marks) FROM Student);

Answer:
It returns names of students whose marks are **greater than the class average**.

🔹 **15. What is the purpose of the FULL JOIN using UNION here?**

Answer:
To simulate a **FULL OUTER JOIN** since MySQL does not support it directly.

🔹 **16. What happens when a student doesn't have a department?**

Answer:
In **LEFT JOIN**, such students are shown with NULL in dept_name.

🔹 **17. What does the query below do?**

```
SELECT dept_name

FROM Department

WHERE dept_id = (SELECT dept_id FROM Student WHERE name='Neha');
```

Answer:
It shows the **department name of Neha** using a subquery.

🔹 **18. Can a view be created using a JOIN?**

Answer:
Yes, a view can be created using one or more **JOINs** between tables.

🔹 **19. What is the difference between a correlated and non-correlated subquery?**

Answer:

- **Correlated subquery** executes once for each row of outer query.
- **Non-correlated subquery** executes once and returns a fixed result.

🔹 **20. Why is 'Kiran' shown with NULL department in LEFT JOIN?**

Answer:
Because Kiran's dept_id is **NULL**, and there's **no matching record** in the Department table.

```
-- PRACTICAL 5: Unnamed PL/SQL Block
-- Control Structure + Exception Handling

-- Drop old tables if exist
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Borrower';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Fine';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

-- 1️⃣ Create Tables
CREATE TABLE Borrower (
   Roll_no NUMBER PRIMARY KEY,
   Name VARCHAR2(50),
   DateofIssue DATE,
   NameofBook VARCHAR2(50),
   Status CHAR(1)
);

CREATE TABLE Fine (
   Roll_no NUMBER,
   Date_ DATE,
```

```
    Amt NUMBER(10,2)
);

-- 2️⃣ Insert Sample Data
INSERT INTO Borrower VALUES (1, 'Akash', TO_DATE('10-OCT-2024', 'DD-MON-YYYY'),
'DBMS', 'I');
INSERT INTO Borrower VALUES (2, 'Neha', TO_DATE('20-OCT-2024', 'DD-MON-YYYY'), 'Java',
'I');
INSERT INTO Borrower VALUES (3, 'Rohan', TO_DATE('25-OCT-2024', 'DD-MON-YYYY'), 'AI',
'I');
COMMIT;


-- 3️⃣ PL/SQL Block (No user input pop-ups)
DECLARE
    v_roll Borrower.Roll_no%TYPE := 1;       -- Set Roll Number here (change manually)
    v_book Borrower.NameofBook%TYPE := 'DBMS'; -- Set Book Name here (change manually)
    v_dateissue Borrower.DateofIssue%TYPE;
    v_days NUMBER;
    v_fine NUMBER := 0;
    v_status Borrower.Status%TYPE;
    e_already_returned EXCEPTION;
BEGIN
    -- Fetch Borrower Details
    SELECT DateofIssue, Status INTO v_dateissue, v_status
    FROM Borrower
    WHERE Roll_no = v_roll AND NameofBook = v_book;


    -- Check if already returned
    IF v_status = 'R' THEN
        RAISE e_already_returned;
```

```
END IF;

-- Calculate days
v_days := SYSDATE - v_dateissue;
DBMS_OUTPUT.PUT_LINE('📚 Days since issue: ' || v_days);

-- Fine calculation
IF v_days BETWEEN 15 AND 30 THEN
   v_fine := (v_days - 14) * 5;
ELSIF v_days > 30 THEN
   v_fine := (16 * 5) + ((v_days - 30) * 50);
ELSE
   v_fine := 0;
END IF;

-- Update book status to Returned
UPDATE Borrower
SET Status = 'R'
WHERE Roll_no = v_roll AND NameofBook = v_book;

-- Insert fine if applicable
IF v_fine > 0 THEN
   INSERT INTO Fine VALUES (v_roll, SYSDATE, v_fine);
   DBMS_OUTPUT.PUT_LINE('💰 Fine imposed: Rs. ' || v_fine);
ELSE
   DBMS_OUTPUT.PUT_LINE('✅ No fine applicable.');
END IF;
```

```
    DBMS_OUTPUT.PUT_LINE('📗 Book Returned Successfully.');


EXCEPTION
    WHEN NO_DATA_FOUND THEN
        DBMS_OUTPUT.PUT_LINE('❌ Error: No record found.');
    WHEN e_already_returned THEN
        DBMS_OUTPUT.PUT_LINE('⚠️ Book already returned.');
    WHEN OTHERS THEN
        DBMS_OUTPUT.PUT_LINE('⚡ Unexpected error: ' || SQLERRM);
END;
/
```

🔹 **1. What is PL/SQL?**

Answer:
PL/SQL stands for **Procedural Language/Structured Query Language**.
It extends SQL by adding **programming features** like loops, conditions, and exception handling.

🔹 **2. What is an unnamed (anonymous) PL/SQL block?**

Answer:
An **unnamed block** is a PL/SQL program that has **no name** and is **not stored** in the database.
It's written between DECLARE, BEGIN, and END keywords and executes once.

🔹 **3. What are the sections of a PL/SQL block?**

Answer:

1. **DECLARE** → declare variables and constants
2. **BEGIN** → main executable statements
3. **EXCEPTION** → handles errors
4. **END;** → ends the block

🔹 **4. What are Control Structures in PL/SQL?**

Answer:
Control structures control the flow of execution.
They are:

- **IF-THEN-ELSE**
- **CASE**
- **LOOPS (FOR, WHILE, LOOP)**

🔹 **5. What are Exceptions in PL/SQL?**

Answer:
Exceptions are **runtime errors** that interrupt normal program execution.
They can be handled using the EXCEPTION section.

🔹 **6. What are the types of Exceptions?**

Answer:

1. **Predefined exceptions** (e.g., NO_DATA_FOUND, ZERO_DIVIDE).
2. **User-defined exceptions** (declared by the programmer using EXCEPTION keyword).

🔹 **7. What is the purpose of this PL/SQL block?**

Answer:
To calculate **fine** based on how long a book is borrowed,
update the **status** of the book as returned,
and insert fine details (if any) into the **Fine** table.

🔹 **8. What does this line mean?**

v_days := SYSDATE - v_dateissue;

Answer:
It calculates the **number of days** between the book's issue date and today's date.

🔹 **9. What are the fine rules used here?**

Answer:

- **0–14 days** → No fine
- **15–30 days** → ₹5 per extra day
- **Above 30 days** → ₹50 per extra day

🔹 **10. What is the use of the IF and ELSIF conditions here?**

Answer:
They implement **decision-making logic** to calculate fine based on the number of days.

🔹 **11. What does this line do?**

UPDATE Borrower SET Status = 'R' WHERE Roll_no = v_roll;

Answer:
It changes the status of the borrowed book from **'I' (Issued)** to **'R' (Returned)**.

🔹 **12. What is the purpose of this exception?**

WHEN NO_DATA_FOUND THEN

DBMS_OUTPUT.PUT_LINE('❌ Error: No record found.');

Answer:
It handles the case when the **Roll_no or book name doesn't exist** in the Borrower table.

🔹 **13. What is a user-defined exception in this code?**

Answer:
e_already_returned — raised when the book's status is already 'R' (returned).

🔹 **14. What does DBMS_OUTPUT.PUT_LINE do?**

Answer:
It displays messages or output in the SQL Developer console (used for debugging or user messages).

🔹 **15. What happens if no fine is applicable?**

Answer:
The block prints:

✅ No fine applicable.

📗 Book Returned Successfully.

and no record is inserted into the Fine table.

🔹 **16. Why did we use COMMIT in this practical?**

Answer:
To **save changes** permanently to the database after inserting data into the Borrower table.

🔹 **17. What is the difference between a predefined and user-defined exception?**

Answer:

- **Predefined** → built-in exceptions like NO_DATA_FOUND.
- **User-defined** → declared manually, e.g., e_already_returned.

🔹 **18. Why did we use EXCEPTION WHEN OTHERS THEN NULL; at the start?**

Answer:
To **safely drop tables** if they exist, avoiding errors when they don't.

🔹 **19. What is the output when a record doesn't exist for given roll number?**

Answer:

❌ Error: No record found.

🔹 **20. What is the use of SYSDATE in PL/SQL?**

Answer:
SYSDATE returns the **current system date and time**.

```
-- PRACTICAL 6: Parameterized Cursor Example

-- 1️⃣ Drop existing tables if they exist
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE N_RollCall';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE O_RollCall';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

-- 2️⃣ Create Old Table (Existing Records)
CREATE TABLE O_RollCall (
   roll_no NUMBER PRIMARY KEY,
   name VARCHAR2(50),
   class VARCHAR2(20)
);

-- 3️⃣ Create New Table (New Records)
CREATE TABLE N_RollCall (
   roll_no NUMBER PRIMARY KEY,
   name VARCHAR2(50),
   class VARCHAR2(20)
);
```

```
-- 4️⃣ Insert Data into Old Table
INSERT INTO O_RollCall VALUES (1, 'Akash', 'FY');
INSERT INTO O_RollCall VALUES (2, 'Neha', 'SY');
INSERT INTO O_RollCall VALUES (3, 'Rohan', 'TY');

-- 5️⃣ Insert Data into New Table
INSERT INTO N_RollCall VALUES (2, 'Neha', 'SY');     -- duplicate
INSERT INTO N_RollCall VALUES (3, 'Rohan', 'TY');    -- duplicate
INSERT INTO N_RollCall VALUES (4, 'Meera', 'FY');    -- new
INSERT INTO N_RollCall VALUES (5, 'Saurabh', 'TY');  -- new
COMMIT;

-- 6️⃣ PL/SQL Block: Merge Using Parameterized Cursor
DECLARE
   -- Parameterized cursor: takes Roll Number as input
   CURSOR c_new(p_roll N_RollCall.roll_no%TYPE) IS
      SELECT roll_no, name, class
      FROM N_RollCall
      WHERE roll_no = p_roll;

   v_roll N_RollCall.roll_no%TYPE;
   v_name N_RollCall.name%TYPE;
   v_class N_RollCall.class%TYPE;
   v_exists NUMBER;
BEGIN
   DBMS_OUTPUT.PUT_LINE('--- Starting Merge Operation ---');
```

```
    -- Loop through all records in N_RollCall
    FOR rec IN (SELECT roll_no FROM N_RollCall) LOOP
        -- Check if Roll Number already exists in Old Table
        SELECT COUNT(*) INTO v_exists FROM O_RollCall WHERE roll_no = rec.roll_no;

        IF v_exists = 0 THEN
            -- Fetch data from N_RollCall using parameterized cursor
            OPEN c_new(rec.roll_no);
            FETCH c_new INTO v_roll, v_name, v_class;
            CLOSE c_new;

            -- Insert new record into Old Table
            INSERT INTO O_RollCall VALUES (v_roll, v_name, v_class);
            DBMS_OUTPUT.PUT_LINE('✅ Inserted Roll No: ' || v_roll || ' | Name: ' || v_name);
        ELSE
            DBMS_OUTPUT.PUT_LINE('⚠️ Skipped Duplicate Roll No: ' || rec.roll_no);
        END IF;
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('--- Merge Completed Successfully! ---');
END;
/
```

🔹 1. What is a cursor in PL/SQL?

Answer:
A **cursor** is a **temporary memory area** used to store and process the result set of a query row-by-row.

🔹 2. Why do we use cursors?

Answer:
Cursors are used when we need to **process multiple rows** returned by a query **one at a time**, instead of all at once.

🔹 3. What are the types of cursors?

Answer:

1. **Implicit Cursor** → Automatically created by Oracle for single-row SQL statements (like INSERT, UPDATE, DELETE).
2. **Explicit Cursor** → Created by the programmer for handling multi-row queries.
3. **Parameterized Cursor** → Accepts parameters to make it more dynamic.

🔹 4. What is a parameterized cursor?

Answer:
A **parameterized cursor** accepts **parameters (arguments)** when it's opened, so it can fetch specific records dynamically.
Example from your code:

```
CURSOR c_new(p_roll N_RollCall.roll_no%TYPE) IS

SELECT * FROM N_RollCall WHERE roll_no = p_roll;
```

🔹 5. What is the purpose of this program?

Answer:
To **merge data** from the new table (**N_RollCall**) into the old table (**O_RollCall**), while **avoiding duplicates** using a parameterized cursor.

🔹 6. What is the logic behind the merge operation?

Answer:

1. Loop through all records of N_RollCall.
2. Check if each roll number already exists in O_RollCall.
3. If it doesn't exist → insert it.
4. If it already exists → skip it.

🔹 **7. What is the purpose of this SQL statement?**

SELECT COUNT(*) INTO v_exists FROM O_RollCall WHERE roll_no = rec.roll_no;

Answer:
It checks whether the **roll number** from the new table already exists in the old table.

🔹 **8. What happens when a duplicate roll number is found?**

Answer:
The block prints:

⚠️ Skipped Duplicate Roll No: <number>

and skips insertion for that record.

🔹 **9. What are the cursor steps in PL/SQL?**

Answer:

1. **Declare** the cursor
2. **Open** the cursor
3. **Fetch** data row-by-row
4. **Close** the cursor

Example:

OPEN c_new(rec.roll_no);

FETCH c_new INTO v_roll, v_name, v_class;

CLOSE c_new;

🔹 **10. What is the difference between an explicit and parameterized cursor?**

Answer:

- **Explicit cursor** → predefined query, fixed.
- **Parameterized cursor** → accepts parameters to make query dynamic (flexible).

🔹 **11. What happens if you forget to close a cursor?**

Answer:
It remains open in memory, which can cause **memory leaks or errors** (too many open cursors).

🔹 **12. Why is DBMS_OUTPUT.PUT_LINE used here?**

Answer:
To display the progress and results of the cursor operation, such as which records were inserted or skipped.

🔹 **13. What are v_roll, v_name, and v_class used for?**

Answer:
They temporarily store the **data fetched** from the cursor before inserting it into the old table.

🔹 **14. What is the use of FOR rec IN (SELECT ...) LOOP?**

Answer:
It automatically **fetches each record** from the result set and iterates over it — eliminating the need to open and close a cursor manually.

🔹 **15. What is the advantage of using parameterized cursors?**

Answer:
They make the program **more reusable and efficient**, as the same cursor can be used to fetch different records by passing parameters.

🔹 **16. What will be the final output in your example?**

Answer:

--- Starting Merge Operation ---

⚠️ Skipped Duplicate Roll No: 2

⚠️ Skipped Duplicate Roll No: 3

✅ Inserted Roll No: 4 | Name: Meera

✅ Inserted Roll No: 5 | Name: Saurabh

--- Merge Completed Successfully! ---

🔹 **17. What are cursor attributes?**

Answer:
Cursor attributes provide information about the state of the cursor:

- %FOUND – Returns TRUE if a row is fetched.
- %NOTFOUND – Returns TRUE if no row is fetched.
- %ROWCOUNT – Number of rows fetched.
- %ISOPEN – Returns TRUE if cursor is open.

🔹 **18. Why did we use COMMIT before the cursor block?**

Answer:
To save the inserted data permanently in both tables before the merge operation begins.

🔹 **19. What is the difference between a cursor and a loop?**

Answer:
A **cursor** is used to fetch query results row-by-row,
while a **loop** is used to repeat a set of statements a number of times.

🔹 **20. What will happen if both tables have exactly same records?**

Answer:
No new records will be inserted — the program will skip all as duplicates.

```
-- PRACTICAL 7: Stored Procedure - proc_Grade

-- 1️⃣ Drop existing tables if any
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Stud_Marks';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Result';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

-- 2️⃣ Create Tables
CREATE TABLE Stud_Marks (
   roll NUMBER PRIMARY KEY,
   name VARCHAR2(50),
   total_marks NUMBER(5)
);

CREATE TABLE Result (
   roll NUMBER,
   name VARCHAR2(50),
   class VARCHAR2(30)
);
```

```
-- 3️⃣ Insert Sample Data
INSERT INTO Stud_Marks VALUES (1, 'Akash', 1450);
INSERT INTO Stud_Marks VALUES (2, 'Neha', 940);
INSERT INTO Stud_Marks VALUES (3, 'Rohan', 870);
INSERT INTO Stud_Marks VALUES (4, 'Meera', 780);
COMMIT;

-- 4️⃣ Create Stored Procedure
CREATE OR REPLACE PROCEDURE proc_Grade (
    p_roll Stud_Marks.roll%TYPE,
    p_name Stud_Marks.name%TYPE,
    p_marks Stud_Marks.total_marks%TYPE
)
IS
    v_class VARCHAR2(30);
BEGIN
    -- Categorize Student
    IF p_marks >= 990 AND p_marks <= 1500 THEN
        v_class := 'Distinction';
    ELSIF p_marks BETWEEN 900 AND 989 THEN
        v_class := 'First Class';
    ELSIF p_marks BETWEEN 825 AND 899 THEN
        v_class := 'Higher Second Class';
    ELSE
        v_class := 'Fail';
    END IF;

    -- Insert result
```

```
    INSERT INTO Result (roll, name, class)
    VALUES (p_roll, p_name, v_class);

    DBMS_OUTPUT.PUT_LINE('Student: ' || p_name || ' | Class: ' || v_class);
END;
/

-- ************************************
-- Block to Use Procedure proc_Grade
-- ************************************

DECLARE
    v_roll Stud_Marks.roll%TYPE;
    v_name Stud_Marks.name%TYPE;
    v_marks Stud_Marks.total_marks%TYPE;
BEGIN
    -- Loop through all students and call procedure
    FOR rec IN (SELECT * FROM Stud_Marks) LOOP
        proc_Grade(rec.roll, rec.name, rec.total_marks);
    END LOOP;

    DBMS_OUTPUT.PUT_LINE('✅ Grades inserted successfully in Result table.');
END;
/

SELECT * FROM Result;
```

🔹 1. What is a Stored Procedure?

Answer:
A **stored procedure** is a **named block of PL/SQL code** stored permanently in the database. It can be called anytime to perform a specific task.

🔹 2. Why do we use Stored Procedures?

Answer:
They:

- Improve **reusability** and **security**.
- **Reduce redundancy** (code written once can be reused).
- **Increase performance** since they are precompiled.

🔹 3. What is the difference between a Procedure and a Function?

Procedure	Function
May or may not return a value	Must return a value
Called using EXEC or inside PL/SQL block	Called inside SQL statements
Can use OUT parameters	Returns a single value via RETURN

🔹 4. What is the purpose of this program?

Answer:
To **categorize students** based on their **total marks** and insert the results (roll, name, class) into the Result table using a stored procedure named proc_Grade.

🔹 5. What are the parameters used in proc_Grade?

Answer:
It uses **three parameters**:

- p_roll → Roll number
- p_name → Student name
- p_marks → Total marks

All are **IN parameters**, meaning values are passed *into* the procedure.

🔹 **6. How does the procedure categorize the students?**

Answer:
Using IF-ELSIF control structure:

- **990–1500** → Distinction
- **900–989** → First Class
- **825–899** → Higher Second Class
- **Below 825** → Fail

🔹 **7. What does this part of the code do?**

INSERT INTO Result (roll, name, class)

VALUES (p_roll, p_name, v_class);

Answer:
It inserts the calculated grade for each student into the Result table.

🔹 **8. What is the purpose of the looping block at the end?**

Answer:
It loops through all records in Stud_Marks and **calls the procedure** for each student automatically.

🔹 **9. What is the output of the program?**

Answer:

Student: Akash | Class: Distinction

Student: Neha | Class: First Class

Student: Rohan | Class: Higher Second Class

Student: Meera | Class: Fail

✅ Grades inserted successfully in Result table.

🔹 **10. What will be the final data in the Result table?**

Roll	Name	Class
1	Akash	Distinction
2	Neha	First Class
3	Rohan	Higher Second Class
4	Meera	Fail

🔹 **11. What is the syntax of creating a stored procedure?**

Answer:

```
CREATE OR REPLACE PROCEDURE procedure_name
(parameter_list)
IS
BEGIN
   -- logic
END;
/
```

🔹 **12. How do you execute or call a stored procedure?**

Answer:

Two ways:

```
EXEC proc_Grade(1, 'Akash', 1450);
```

or inside a block:

```
BEGIN
  proc_Grade(1, 'Akash', 1450);
END;
/
```

🔹 **13. What is the use of CREATE OR REPLACE in procedure creation?**

Answer:
It allows you to **modify and recreate** the same procedure without dropping it manually.

🔹 **14. What are IN, OUT, and IN OUT parameters?**

Type	Description
IN	Passes value *into* procedure (default).
OUT	Returns value *from* procedure.
IN OUT	Both passes and returns value.

🔹 **15. Why is DBMS_OUTPUT.PUT_LINE used here?**

Answer:
It displays the result and class of each student in the console — useful for verifying logic execution.

🔹 **16. What happens if you don't COMMIT before running the procedure?**

Answer:
The initial student data in Stud_Marks might not be saved permanently — so the procedure could fail to fetch it.

🔹 **17. What is the data type of v_class?**

Answer:
v_class is a **VARCHAR2(30)** variable — it stores the category of student's result (Distinction, First Class, etc.).

🔹 **18. What are the benefits of stored procedures in databases?**

Answer:

- Better **performance** (precompiled).
- **Reduced network traffic** (logic executes on server).

- **Code reusability** and **security** (controlled access).

🔹 19. Can a stored procedure call another procedure?

Answer:
Yes, one procedure can call another — this is known as **procedure nesting**.

🔹 20. What is the difference between a procedure and an anonymous block?

Procedure	Anonymous Block
Stored permanently in database	Not stored, executes once
Can be reused	Cannot be reused
Has a name	No name

```
-- PRACTICAL 8: Database Trigger - Library Audit

-- 1️⃣ Drop existing tables if they exist
BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Library_Audit';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

BEGIN
  EXECUTE IMMEDIATE 'DROP TABLE Library';
EXCEPTION WHEN OTHERS THEN NULL;
END;
/

-- 2️⃣ Create Main Table: Library
CREATE TABLE Library (
   book_id NUMBER PRIMARY KEY,
   book_name VARCHAR2(100),
   author VARCHAR2(50),
   price NUMBER(10,2)
);

-- 3️⃣ Create Audit Table: Library_Audit
CREATE TABLE Library_Audit (
   audit_id NUMBER GENERATED ALWAYS AS IDENTITY,
   book_id NUMBER,
   book_name VARCHAR2(100),
```

```
    author VARCHAR2(50),
    price NUMBER(10,2),
    operation_type VARCHAR2(10),
    operation_date DATE
);

-- 4️⃣ Insert Sample Records
INSERT INTO Library VALUES (1, 'DBMS Concepts', 'Korth', 500);
INSERT INTO Library VALUES (2, 'Operating System', 'Galvin', 450);
INSERT INTO Library VALUES (3, 'Computer Networks', 'Tanenbaum', 600);
COMMIT;

-- 5️⃣ Create Trigger for UPDATE and DELETE
CREATE OR REPLACE TRIGGER trg_library_audit
AFTER UPDATE OR DELETE ON Library
FOR EACH ROW
BEGIN
    IF UPDATING THEN
        INSERT INTO Library_Audit (book_id, book_name, author, price, operation_type, operation_date)
        VALUES (:OLD.book_id, :OLD.book_name, :OLD.author, :OLD.price, 'UPDATE', SYSDATE);

        DBMS_OUTPUT.PUT_LINE('🔄 Book Updated: ' || :OLD.book_name);
    ELSIF DELETING THEN
        INSERT INTO Library_Audit (book_id, book_name, author, price, operation_type, operation_date)
        VALUES (:OLD.book_id, :OLD.book_name, :OLD.author, :OLD.price, 'DELETE', SYSDATE);
```

```
      DBMS_OUTPUT.PUT_LINE('❌ Book Deleted: ' || :OLD.book_name);
   END IF;
END;
/

-- Run above code first then run following command one by one
-- 1
UPDATE Library
SET price = 550
WHERE book_id = 1;

--2
DELETE FROM Library
WHERE book_id = 2;

--3
SELECT * FROM Library_Audit;
```

🔹 **1. What is a trigger in a database?**

Answer:
A **trigger** is a **stored PL/SQL block** that automatically executes (fires) when a specified event (INSERT, UPDATE, DELETE) occurs on a table.

🔹 **2. What are the types of triggers?**

Answer:

1. **Row-level trigger** – Executes once for each affected row.
2. **Statement-level trigger** – Executes once per triggering statement.
3. **BEFORE trigger** – Executes before the event.
4. **AFTER trigger** – Executes after the event.

🔹 **3. What is the purpose of this trigger (trg_library_audit)?**

Answer:
To **track changes** in the Library table.
Whenever a record is updated or deleted, the **old data** is inserted into the Library_Audit table for record-keeping.

🔹 **4. When does this trigger fire?**

Answer:
The trigger fires **AFTER an UPDATE or DELETE** operation on the Library table.

🔹 **5. What does :OLD refer to in a trigger?**

Answer:
:OLD holds the **previous values** of the row before an UPDATE or DELETE operation.

🔹 **6. What does :NEW refer to in a trigger?**

Answer:
:NEW holds the **new values** of the row being inserted or updated.
(Not used here because we only need old data for audit.)

🔹 **7. What type of trigger is this (row-level or statement-level)?**

Answer:
It's a **row-level trigger** because it includes the clause FOR EACH ROW.

🔹 **8. What happens when we update a book price in the Library table?**

Answer:

1. The trigger fires automatically.
2. The old details (before update) are copied to the Library_Audit table.
3. A message is displayed:
4. 🔄 Book Updated: DBMS Concepts

🔹 **9. What happens when a book is deleted?**

Answer:

1. The trigger fires automatically.
2. The deleted book details are inserted into Library_Audit.
3. Message displayed:
4. ❌ Book Deleted: Operating System

🔹 **10. What is the structure of the Library_Audit table?**

Column Name	Description
audit_id	Auto-generated ID
book_id	Book ID from old record
book_name	Name of the book
author	Author of the book
price	Old price before change

Column Name	Description
operation_type	‘UPDATE’ or ‘DELETE’
operation_date	System date of operation

🔹 **11. Why did we use AFTER UPDATE OR DELETE instead of BEFORE?**

Answer:
Because we want to record **only successful changes** that occurred, not the ones attempted.

🔹 **12. What is the use of DBMS_OUTPUT.PUT_LINE inside the trigger?**

Answer:
It displays a message showing which book was updated or deleted — useful for confirmation.

🔹 **13. What is meant by auditing in databases?**

Answer:
Auditing means **tracking and recording changes** made to data for **accountability and security**.

🔹 **14. Can a trigger modify the same table on which it is defined?**

Answer:
No, it can cause **mutating table error**.
Here, we avoid it by inserting data into a **different table (Library_Audit)**.

🔹 **15. What does this part of the code do?**

VALUES (:OLD.book_id, :OLD.book_name, :OLD.author, :OLD.price, 'UPDATE', SYSDATE);

Answer:
It inserts the **old values** of the record along with the **operation type (‘UPDATE’)** and **current date** into the Library_Audit table.

🔹 **16. What does AFTER keyword mean in a trigger?**

Answer:
It specifies that the trigger will execute **after** the SQL operation has been completed successfully.

🔹 **17. What is the output after running the commands in order?**

Answer:

🔁 Book Updated: DBMS Concepts

❌ Book Deleted: Operating System

🔹 **18. What will be stored in the Library_Audit table?**

book_id	book_name	author	price	operation_type	operation_date
1	DBMS Concepts	Korth	500	UPDATE	[current date]
2	Operating System	Galvin	450	DELETE	[current date]

🔹 **19. What are BEFORE triggers used for?**

Answer:
To **validate or modify data** before it's inserted or updated in the table.

🔹 **20. What are some practical uses of triggers?**

Answer:

- Maintaining audit trails
- Enforcing business rules
- Automatic logging
- Synchronizing tables
- Validating data changes

🧩 Practical 9: Study of Open Source NoSQL Database — MongoDB

(Installation, Basic CRUD Operations, and Execution)

🎯 Aim

To study and perform installation and basic CRUD (Create, Read, Update, Delete) operations using the MongoDB NoSQL database.

⚙️ Step 1: Installation of MongoDB

🪟 For Windows:

Download MongoDB from the official website:

👉 https://www.mongodb.com/try/download/community

Install it using the setup wizard.

During setup, select ✅ Install MongoDB as a Service.

Once installed, MongoDB is available at:

C:\Program Files\MongoDB\Server\<version>\bin

Add this path to your Environment Variables → Path.

💻 Step 2: Verify Installation

Open Command Prompt (cmd) or PowerShell and type:

mongosh

If MongoDB is installed correctly, you will see:

Using MongoDB: 8.x.x

Using Mongosh: 2.x.x

test>

✅ You are now inside the Mongo Shell.

🧱 Step 3: Create Database and Collection

Create a new database:

use collegeDB

Create a collection:

db.createCollection("students")

✅ Output:

{ "ok" : 1 }

🗒️ Step 4: Perform Basic CRUD Operations

➕ CREATE (Insert Data)

db.students.insertMany([

{ roll_no: 1, name: "Akash", dept: "Computer", city: "Nashik", marks: 85 },

{ roll_no: 2, name: "Neha", dept: "IT", city: "Pune", marks: 92 },

{ roll_no: 3, name: "Rohan", dept: "ENTC", city: "Mumbai", marks: 78 },

{ roll_no: 4, name: "Meera", dept: "Computer", city: "Nashik", marks: 88 },

{ roll_no: 5, name: "Saurabh", dept: "IT", city: "Nagpur", marks: 70 }

])

✅ Output:

Acknowledged: true

Inserted 5 documents

🔍 READ (Find Data)

Show all students:

db.students.find()

Show students with marks greater than 80:

db.students.find({ marks: { $gt: 80 } })

Show students from "Nashik":

db.students.find({ city: "Nashik" })

🖍️ UPDATE (Modify Data)

Update a single record:

```
db.students.updateOne(
 { roll_no: 3 },
 { $set: { marks: 82, city: "Aurangabad" } }
)
```

Update multiple records:

```
db.students.updateMany(
 { dept: "IT" },
 { $inc: { marks: 5 } }
)
```

✅ The $set operator modifies specific fields,

✅ The $inc operator increments numerical values.

🗑️ DELETE (Remove Data)

Delete one record:

```
db.students.deleteOne({ roll_no: 5 })
```

Delete multiple records:

```
db.students.deleteMany({ city: "Nashik" })
```

🧠 Step 5: Understanding MongoDB Structure

Concept	Equivalent In SQL	Description
Database	Database	A logical container for collections
Collection	Table	Stores documents

Document	Row	A JSON-like record
Field	Column	Key-value pair inside a document

⚡ Step 6: Execute CRUD Using MongoDB Compass

If using MongoDB Compass (GUI tool):

Open MongoDB Compass

Connect using:

mongodb://localhost:27017

Create a new Database:

Database name: collegeDB

Collection name: students

Perform CRUD using GUI buttons:

Insert Document → Add new student

Filter Box → Search / Read

UPDATE / DELETE buttons → Modify or remove data

✅ Result

MongoDB was successfully installed, configured, and basic CRUD operations were executed using:

Mongo Shell (CLI)

MongoDB Compass (GUI)

📄 Output Example (Final Data in Collection)


```
[
 { "roll_no": 1, "name": "Akash", "dept": "Computer", "city": "Nashik", "marks": 85 },
 { "roll_no": 2, "name": "Neha", "dept": "IT", "city": "Pune", "marks": 97 },
 { "roll_no": 3, "name": "Rohan", "dept": "ENTC", "city": "Aurangabad", "marks": 82 },
```

```
{ "roll_no": 4, "name": "Meera", "dept": "Computer", "city": "Nashik", "marks": 88 }
]
```

🎓 Conclusion

MongoDB, a NoSQL open-source database, was successfully installed and demonstrated using CRUD operations for managing student records. The experiment shows how MongoDB stores data in flexible JSON-like documents instead of rigid tables.

🔹 **1. What is MongoDB?**

Answer:
MongoDB is an **open-source, NoSQL database** that stores data in **flexible, JSON-like documents** instead of traditional tables.

🔹 **2. What does NoSQL mean?**

Answer:
NoSQL means "Not Only SQL."
It represents databases that don't use traditional rows and columns — they store data in **key-value, document, or graph** formats.

🔹 **3. What is the difference between SQL and MongoDB?**

SQL	MongoDB
Uses tables and rows	Uses collections and documents
Fixed schema	Schema-less (flexible)
Joins supported	Embedding or references used
Data stored in relational format	Data stored as JSON-like documents

---

🔹 **4. What is a Collection in MongoDB?**

Answer:
A **Collection** is similar to a **table** in SQL — it stores multiple **documents** (records).

🔹 **5. What is a Document in MongoDB?**

Answer:
A **Document** is a single record in MongoDB, represented in **JSON-like format**:

{ "roll_no": 1, "name": "Akash", "dept": "Computer" }

🔹 **6. What are the CRUD operations in MongoDB?**

Answer:

- **C → Create** → insertOne(), insertMany()
- **R → Read** → find(), findOne()
- **U → Update** → updateOne(), updateMany()
- **D → Delete** → deleteOne(), deleteMany()

🔹 **7. What is the command to create a database in MongoDB?**

Answer:

use collegeDB

This command switches to (or creates) a database named collegeDB.

🔹 **8. What is the command to create a collection?**

Answer:

db.createCollection("students")

🔹 **9. How do you insert multiple documents in MongoDB?**

Answer:

```
db.students.insertMany([
  { roll_no: 1, name: "Akash" },
  { roll_no: 2, name: "Neha" }
])
```

🔹 **10. What is the use of $set and $inc operators?**

Answer:

- $set → modifies or replaces field values.
- $inc → increases (increments) numerical fields.

Example:

db.students.updateOne({ roll_no: 1 }, { $set: { marks: 90 } })

db.students.updateMany({ dept: "IT" }, { $inc: { marks: 5 } })

🔹 **11. How do you display all documents in a collection?**

Answer:

db.students.find()

🔹 **12. How to find students with marks greater than 80?**

Answer:

db.students.find({ marks: { $gt: 80 } })

🔹 **13. How to delete a single document?**

Answer:

db.students.deleteOne({ roll_no: 5 })

🔹 **14. How to delete multiple documents?**

Answer:

db.students.deleteMany({ city: "Nashik" })

🔹 **15. What is the default port number for MongoDB?**

Answer:
27017

🔹 16. What is MongoDB Compass?

Answer:
MongoDB Compass is a **GUI tool** for MongoDB — allows performing CRUD, aggregation, and visualization operations without using code.

🔹 17. How to connect MongoDB Compass to local database?

Answer:
Use the connection string:

mongodb://localhost:27017

🔹 18. What is the structure difference between MongoDB and MySQL?

Concept	MongoDB	MySQL
Database	Database	Database
Collection	Table	Table
Document	Row	Record
Field	Column	Column

🔹 19. What is the use of insertMany() instead of insertOne()?

Answer:
insertMany() allows inserting **multiple records at once**, making the operation faster and efficient.

🔹 20. Can MongoDB handle unstructured data?

Answer:
Yes — MongoDB can store **unstructured, semi-structured, and hierarchical data** in flexible JSON documents.

🧩 PRACTICAL 10 — Step-by-Step Execution Guide

🎯 Aim

To perform basic CRUD operations (Create, Read, Update, Delete)

and use the SAVE method and logical operators in MongoDB.

🧱 Step 1: Create Database and Collection

Open MongoDB Compass.

Click on ➕ Create Database

Enter:

Database Name: collegeDB

Collection Name: students

Click Create Database ✅

🗒️ Step 2: Insert Documents (CREATE Operation)

Now click on collegeDB → students → ADD DATA → Insert Document

Paste this code and click Insert 👇

```
[
 { "roll_no": 1, "name": "Akash", "dept": "Computer", "city": "Nashik", "marks": 85 },
 { "roll_no": 2, "name": "Neha", "dept": "IT", "city": "Pune", "marks": 92 },
 { "roll_no": 3, "name": "Rohan", "dept": "ENTC", "city": "Mumbai", "marks": 78 },
 { "roll_no": 4, "name": "Meera", "dept": "Computer", "city": "Nashik", "marks": 88 },
 { "roll_no": 5, "name": "Saurabh", "dept": "IT", "city": "Nagpur", "marks": 70 }
]
```

✅ You have now inserted multiple documents into the collection.

🔍 Step 3: Retrieve Data (READ Operation)

a) Show all students

In the Filter box, type:

{}

Click Find

👉 Displays all students.

b) Show students with marks greater than 80

{ "marks": { "$gt": 80 } }

c) Show students from Nashik city

{ "city": "Nashik" }

d) Show students from Computer OR IT department

(Logical Operator $or)

{ "$or": [{ "dept": "Computer" }, { "dept": "IT" }] }

e) Show students with marks between 75 and 90

(Logical Operator $and)

{ "$and": [{ "marks": { "$gte": 75 } }, { "marks": { "$lte": 90 } }] }

🖍️ Step 4: Update Data (UPDATE Operation)

Click on UPDATE or run this query in mongosh:

```
db.students.updateOne(
  { "roll_no": 3 },
  { "$set": { "marks": 82, "city": "Aurangabad" } }
)
```

👉 This updates Rohan’s marks and city.

You can also update many records at once:

```
db.students.updateMany(
 { "dept": "IT" },
 { "$inc": { "marks": 5 } }
)
```

👉 Adds 5 marks to all IT students.

🗑️ Step 5: Delete Data (DELETE Operation)

To delete a specific record:

```
db.students.deleteOne({ "roll_no": 5 })
```

To delete all students from a city:

```
db.students.deleteMany({ "city": "Nashik" })
```

💾 Step 6: SAVE Method (Insert or Update Automatically)

MongoDB’s save() method will insert a new document
if it doesn’t exist, or update it if it does exist.

```
db.students.save({ "roll_no": 6, "name": "Priya", "dept": "AI", "city": "Pune", "marks": 95 })
```

👉 Adds a new student if roll_no = 6 doesn’t exist.

👉 If it exists, updates that record.

🧠 Step 7: Use of Logical Operators

Operator Description Example

$and Returns documents that match both conditions { "$and": [{ "dept": "IT" }, { "marks": { "$gt": 80 } }] }

$or Returns documents that match either condition { "$or": [{ "city": "Pune" }, { "city": "Nashik" }] }

$not Returns documents that do not match a condition { "marks": { "$not": { "$gte": 80 } } }

$nor Returns documents that fail all conditions { "$nor": [{ "dept": "IT" }, { "dept": "Computer" }] }

📋 Step 8: Verify All Documents

To confirm final data, run:

{}

and click Find again — this will show the updated collection.

🏁 Step 9: Result

✅ You have successfully demonstrated all CRUD operations:

Create → insertMany()

Read → find() with conditions and logical operators

Update → updateOne() / updateMany()

Delete → deleteOne() / deleteMany()

Save → save() (insert or update)

🔹 1. What does CRUD stand for in MongoDB?

Answer: CRUD stands for:

- **C → Create** – Insert data
- **R → Read** – Retrieve data
- **U → Update** – Modify existing data
- **D → Delete** – Remove data

🔹 2. What command is used to insert multiple documents?

Answer:

```
db.students.insertMany([
 { "roll_no": 1, "name": "Akash" },
 { "roll_no": 2, "name": "Neha" }
])
```

🔹 3. How do you display all records in a MongoDB collection?

Answer:

```
db.students.find({})
```

This command displays all documents in the students collection.

🔹 4. What is the use of the $gt operator?

Answer: $gt means **greater than** — used for numeric comparisons.
Example: db.students.find({ marks: { $gt: 80 } })

🔹 5. What is the difference between $and and $or operators?

Operator	Description	Example
$and	Returns documents that satisfy **both** conditions	{ "$and": [{ "marks": { "$gte": 75 } }, { "marks": { "$lte": 90 } }] }

Operator	Description	Example
$or	Returns documents that satisfy **either** condition	{ "$or": [{ "dept": "IT" }, { "dept": "Computer" }] }

🔹 **6. What is the use of $set and $inc in MongoDB?**

Answer:

- $set → Changes the value of a specific field.
- $inc → Increases the value of a field by a given number.

Example:

db.students.updateOne({ roll_no: 3 }, { $set: { city: "Aurangabad" } })

db.students.updateMany({ dept: "IT" }, { $inc: { marks: 5 } })

🔹 **7. How do you delete a single record?**

Answer: db.students.deleteOne({ roll_no: 5 })

🔹 **8. How do you delete multiple records?**

Answer: db.students.deleteMany({ city: "Nashik" })

🔹 **9. What does the save() method do in MongoDB?**

Answer: save() either **inserts a new document** if it doesn't exist
or **updates** an existing document if it already exists (based on _id or key).

Example:

db.students.save({ "roll_no": 6, "name": "Priya", "dept": "AI", "marks": 95 })

🔹 **10. What is the difference between insert() and save()?**

Operation	Behavior
insert()	Always inserts a new document

Operation	Behavior
save()	Updates if record exists, else inserts new

🔹 11. What is the structure of a MongoDB document?

Answer:
A document is a **key-value pair JSON-like object**:

```
{ "roll_no": 1, "name": "Akash", "dept": "Computer" }
```

🔹 12. What are logical operators in MongoDB?

Answer:
Used to combine or negate conditions:

Operator	Description	Example
$and	Matches all conditions	{ "$and": [{ "dept": "IT" }, { "marks": { "$gt": 80 } }] }
$or	Matches any condition	{ "$or": [{ "city": "Pune" }, { "city": "Nashik" }] }
$not	Negates condition	{ "marks": { "$not": { "$gte": 80 } } }
$nor	Fails all conditions	{ "$nor": [{ "dept": "IT" }, { "dept": "Computer" }] }

🔹 13. What is the default primary key in MongoDB?

Answer: _id is automatically generated as the **unique identifier** for each document.

🔹 14. What is the command to update one document?

Answer:

```
db.students.updateOne(
  { roll_no: 3 },
  { $set: { city: "Aurangabad" } }
)
```

🔹 **15. How can you view updated data in Compass?**

Answer: Simply type {} in the **Filter box** and click **Find** — it displays all documents in the current collection.

🔹 **16. What is the output after incrementing marks for IT students?**

Answer: Marks of all IT students will increase by **5**.

🔹 **17. Can MongoDB store data without defining schema?**

Answer: Yes ✅ — MongoDB is **schema-less**, allowing different documents in the same collection to have different fields.

🔹 **18. What is the data model used by MongoDB?**

Answer: Document Data Model, where data is stored as **BSON** (Binary JSON).

🔹 **19. What are the main advantages of MongoDB?**

Answer:

- Schema-less (flexible structure)
- High performance and scalability
- Easy to integrate with modern applications

🔹 **20. What is the final output collection after all CRUD operations?**


```
[
 { "roll_no": 1, "name": "Akash", "dept": "Computer", "city": "Nashik", "marks": 85 },
 { "roll_no": 2, "name": "Neha", "dept": "IT", "city": "Pune", "marks": 97 },
 { "roll_no": 3, "name": "Rohan", "dept": "ENTC", "city": "Aurangabad", "marks": 82 },
 { "roll_no": 4, "name": "Meera", "dept": "Computer", "city": "Nashik", "marks": 88 },
 { "roll_no": 6, "name": "Priya", "dept": "AI", "city": "Pune", "marks": 95 }
]
```

Practical 13

Run the below code here: https://jsonlint.com

1. Simple JSON Object

```
{
  "student": {
    "roll_no": 1,
    "name": "Akash",
    "department": "Computer",
    "marks": 89,
    "city": "Nashik"
  }
}
```

2. JSON Array Object

```
{
  "students": [
    {
      "roll_no": 1,
      "name": "Akash",
      "marks": 89,
      "city": "Nashik"
    },
    {
      "roll_no": 2,
      "name": "Neha",
      "marks": 92,
      "city": "Pune"
```

```
  },
  {
    "roll_no": 3,
    "name": "Rohan",
    "marks": 76,
    "city": "Mumbai"
  }
 ]
}
```

External Viva sathi answer:

When the examiner asks:

“How will you run or show this practical?”

You should say:

“Sir/Ma’am, JSON is a text-based data format. We usually demonstrate it by writing valid JSON structure in an editor like VS Code, Online JSON Viewer, or MongoDB Compass. It doesn’t execute like SQL — it represents structured data.”

Then, you can:

Open VS Code / Notepad / Online JSON Viewer

Paste your JSON code

Show that the structure is valid and formatted properly.

✅ That’s enough to “run” it.

Fakt check kar ki valid ahe ki nahi

🔹 1. What is JSON?

Answer:
JSON stands for **JavaScript Object Notation**.
It is a **lightweight, text-based data format** used to store and exchange data between a server and a client.

🔹 2. What is the use of JSON?

Answer:
JSON is mainly used for **data transfer between applications**, especially between **frontend (like JavaScript)** and **backend (like databases or APIs)**.

🔹 3. What are the main data types supported by JSON?

Answer:

- String
- Number
- Boolean
- Null
- Object
- Array

🔹 4. What is the syntax of a JSON Object?

Answer:
A JSON Object is written inside {} (curly braces) and contains key-value pairs.
Example:

```
{
  "name": "Akash",
  "city": "Nashik"
}
```

🔹 5. What is a JSON Array?

Answer:
A JSON Array is a collection of multiple JSON objects enclosed in square brackets [].
Example:

```
{
  "students": [
    { "name": "Akash", "marks": 89 },
    { "name": "Neha", "marks": 92 }
  ]
}
```

🔹 **6. How is JSON different from XML?**

JSON	XML
Lightweight & easy to read	Heavier & more complex
Uses {} and []	Uses tags < >
Faster to parse	Slower parsing
Human-readable	Less readable

🔹 **7. What is the structure of a valid JSON file?**

Answer:

- Data is written in **key-value pairs**
- Keys must be in **double quotes ("")**
- Each key-value pair is separated by a **comma**
- The file must start and end with {} or []

🔹 **8. What are common uses of JSON?**

Answer:

- Transferring data between web server and client
- Storing configuration files

- APIs and RESTful web services
- Databases like **MongoDB** store data in JSON-like format (BSON)

🔹 9. How do you validate JSON data?

Answer:
You can validate JSON by checking it with tools like
👉 https://jsonlint.com
or editors like **VS Code** — it will highlight syntax errors automatically.

🔹 10. How do you show JSON output in a practical?

Answer:

“Sir/Ma’am, JSON doesn’t execute like SQL — it’s a data format.
I show it by pasting my JSON code in **VS Code** or **JSONLint**,
then format and validate it to confirm it’s a valid structure.”

🔹 11. Can JSON contain nested objects?

Answer:
Yes ✅.
A JSON object can contain another object inside it.
Example:

```
{
  "student": {
    "name": "Akash",
    "address": { "city": "Nashik", "pincode": 422010 }
  }
}
```

🔹 12. What is the MIME type of JSON?

Answer:
application/json

🔹 **13. Is JSON case-sensitive?**

Answer:
Yes ✅
Keys in JSON are **case-sensitive** (e.g., "Name" and "name" are different).

🔹 **14. How is JSON related to MongoDB?**

Answer:
MongoDB stores data in a **BSON format (Binary JSON)** —
which is a binary-encoded version of JSON that supports additional data types.

🔹 **15. What is the output of your JSON practical?**

Answer:
Output 1 — Simple Object:

```
{
  "student": {
    "roll_no": 1,
    "name": "Akash",
    "department": "Computer",
    "marks": 89,
    "city": "Nashik"
  }
}
```

Output 2 — Array Object:

```
{
  "students": [
    { "roll_no": 1, "name": "Akash", "marks": 89, "city": "Nashik" },
    { "roll_no": 2, "name": "Neha", "marks": 92, "city": "Pune" },
    { "roll_no": 3, "name": "Rohan", "marks": 76, "city": "Mumbai" }
```

]

}

🔹 16. Short Viva Summary to Speak:

“In this practical, I created a simple JSON object and an array of objects to represent student data.
JSON is a data format used for data transfer.
I validated my JSON code using JSONLint to ensure it was properly structured and valid.”