SIES College of Arts, Science & Commerce

RISE WITH EDUCATION

NAAC REACCREDITED - 'A' GRADE

ISO 9001 : 2008

S.I.E.S College of Arts, Science and Commerce (Empowered Autonomous)
Sion(W), Mumbai – 400 022.

CERTIFICATE

This is to certify that Miss/Mr **Ansari Abu Fahad Ashfaque Ahmed** Roll No. **FMCS2526096** has successfully completed the necessary course of experiments in the subject of **Advanced Database Management System** during the academic year**2025 – 2026** complying with the requirements of **University of Mumbai**, for the course of **MSc Computer Science [Semester- I].**

Asst. Prof. In-Charge
MS. SONI YADAV

Examination date:

Examiner's Signature & Date:

Signature of HOD
DR. MANOJ SINGH

College Seal

INDEX

Sr. No	Date	Topic	Signature
1	03/09/2025	Create a Distributed Database System using Vertical Fragmentation and Execute Queries on the Fragments.	
2	19/09/2025	Create a Distributed Database System using Horizontal Fragmentation and Execute Queries on the Fragments.	
3	06/09/2025	Create an Object-Oriented Database and Execute Queries on it.	
4	19/09/2025	Create an XML database and demonstrate insert, update and delete operations on these tables. Issue queries on it.	
5	23/09/2025	Create a table that stores spatial data and issue queries on it.	
6	04/10/2025	Create a temporal database and issue queries on it.	
7	08/10/2025	Demonstrate the accessing, storing and performing CRUD operations in MongoDB.	

Practical No: 1

Aim: To study and implement Vertical Fragmentation of a database relation using Tables, Views, and Materialized Views.

Example 1

- **Fragment1:** Create table Student Personal & Insert Values in it.

```
CREATE TABLE STUDENT_PERSONAL (
    ROLL_NO INT PRIMARY KEY,
    NAME    VARCHAR(50),
    ADDRESS  VARCHAR(100),
    PHONE    VARCHAR(15)
);

insert into STUDENT_PERSONALVALUES
(1, 'ZAMAM', 'KURLA',913636),
(2, 'AFZAL', 'SION' ,9185236),
(3, 'FAHAD', 'BYCULLA' ,9196325),
SELECT * FROM STUDENT_PERSONAL;
```

```
Table STUDENT_PERSONAL created.        3 rows inserted.

Elapsed: 00:00:00.018                  Elapsed: 00:00:00.363
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.005 seconds

	ROLLNO	NAME	ADDRESS	PHONE
1	1	ZAMAM	KURLA	913636
2	2	AFZAL	SION	9185236
3	3	FAHAD	BYCULLA	9196325

- **Fragment2**: Create table Student Academic to hold academic information & Insert Values in it.

```
CREATE TABLE STUDENT_ACADEMIC (
    ROLL_NO INT PRIMARY KEY,
    COURSE  VARCHAR(50),
    MARKS    INT
);


INSERT INTO STUDENT_ACADEMIC VALUES
    ( 1,'BSC-CS', 90 ),
    ( 2,'BSC-IT',89 ),
    ( 3,'MSC-CS',75)
SELECT * FROM STUDENT_ACADEMIC;
```

```
Table STUDENT_ACADEMIC created.          3 rows inserted.

Elapsed: 00:00:00.014                    Elapsed: 00:00:00.333
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.004 seconds

	ROLLNO	COURSE	MARKS
1	1	BSC CS	79
2	2	BSC IT	89
3	3	MSC CS	69

- Create main table by joining two fragments

```
SELECT
    p.ROLL_NO,
    p.NAME,
    p.ADDRESS,
    p.PHONE,
    a.COURSE,
    a.MARKS
FROM STUDENT_PERSONAL p
JOIN STUDENT_ACADEMIC a
ON p.ROLL_NO = a.ROLL_NO;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.011 seconds

	ROLLNO	NAME	ADDRESS	PHONE	COURSE	MARKS
1	1	ZAMAM	KURLA	913636	BSC CS	79
2	2	AFZAL	SION	9185236	BSC IT	89
3	3	FAHAD	BYCULLA	9196325	MSC CS	69

Example 2

- **Base Table :** Create Employee Table As a Base Table & Insert Values in it.

```
CREATE TABLE EMPLOYEE(
    EMPID INT PRIMARY KEY,
    NAME VARCHAR(50),
    DEPARTMENT VARCHAR (50),
    SALARY DECIMAL(10,2),
    LOCATION VARCHAR(50),
    CONTACTNO VARCHAR(15)
);

INSERT INTO EMPLOYEE VALUES
 (1,'zamam','cs',10000,'mumbai','87452103'),
 (2,'afzal','it',20000,'sion','85464654'),
 (3,'fahad','data sci',30000,'kurla','85456432'),
 (4,'zoheb','cyber sec',40000,'bandra','87985465'),
 (5,'sami','EH',50000,'byculla','84984354');

SELECT * FROM EMPLOYEE;
```

```
5 rows inserted.

Elapsed: 00:00:00.029
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download ▾ Execution time: 0.004 seconds

	EMPID	NAME	LOCATION	CONTACTNO	DEPARTMENT	SALARY
1	1	zamam	mumbai	87452103	cs	10000
2	2	afzal	sion	85464654	it	20000
3	3	fahad	kurla	85456432	data sci	30000
4	4	zoheb	bandra	87985465	cyber sec	40000
5	5	sami	byculla	84984354	EH	50000

- Create two Separate Fragments **Using View.** Then Join those fragments to reconstruct base table data.

1.

```
CREATE VIEW EMP_PERSONAL AS
    SELECT EMPID,NAME,LOCATION,CONTACTNO
    FROM EMPLOYEE


SELECT * FROM EMP_PERSONAL;
```

```
View EMP_PERSONAL created.

Elapsed: 00:00:00.009
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.007 seconds

	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354

2.

```
CREATE VIEW EMP_WORK AS
    SELECT EMPID,DEPARTMENT,SALARY
    FROM EMPLOYEE

SELECT * FROM EMP_WORK;
```

```
View EMP_WORK created.

Elapsed: 00:00:00.011
```

Download | Execution time: 0 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000

3. Reconstruct Base table by joining two fragments(view).

```
SELECT p.EMPID,p.NAME,p.LOCATION,p.CONTACTNO,
       w.DEPARTMENT,w.SALARY
FROM EMP_PERSONAL p
JOIN EMP_WORK w
ON p.EMPID = w.EMPID
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.004 seconds

	EMPID	NAME	LOCATION	CONTACTNO	DEPARTMENT	SALARY
1	1	zamam	mumbai	87452103	cs	10000
2	2	afzal	sion	85464654	it	20000
3	3	fahad	kurla	85456432	data sci	30000
4	4	zoheb	bandra	87985465	cyber sec	40000
5	5	sami	byculla	84984354	EH	50000

4. Insert New Records In Base Table then Verify Whethere is View Updated or Not.

```
INSERT INTO EMPLOYEE VALUES
(6,'hanzla','backend',60000,'thane','87984658'),
(7,'Aqib','AI',70000,'bandraeast','87845614'),
(8,'samikapadia','IATA',80000,'vikroli','8484655');
```

```
3 rows inserted.

Elapsed: 00:00:00.004
```

```
SELECT * FROM EMPLOYEE;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.003 seconds

	EMPID	NAME	DEPARTMENT	SALARY	LOCATION	CONTACTNO
1	1	zamam	cs	10000	mumbai	87452103
2	2	afzal	it	20000	sion	85464654
3	3	fahad	data sci	30000	kurla	85456432
4	4	zoheb	cyber sec	40000	bandra	87985465
5	5	sami	EH	50000	byculla	84984354
6	6	hanzla	backend	60000	thane	87984658
7	7	Aqib	AI	70000	bandraeast	87845614
8	8	samikapadia	IATA	80000	vikroli	8484655

```
SELECT * FROM EMP_PERSONAL;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.009 seconds

	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354
6	6	hanzla	thane	87984658
7	7	Aqib	bandraeast	87845614
8	8	samikapadia	vikroli	8484655

```
SELECT * FROM EMP_WORK;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.011 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000
6	6	backend	60000
7	7	AI	70000
8	8	IATA	80000

- Create two Separate Fragments **Using Table.**

1.
```
CREATE TABLE EMP_PERSONAL1 AS
    SELECT EMPID,NAME,LOCATION,CONTACTNO
    FROM EMPLOYEE


SELECT * FROM EMP_PERSONAL1;
```

```
Table EMP_PERSONAL1 created.

Elapsed: 00:00:00.038
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.009 seconds

	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354
6	6	hanzla	thane	87984658
7	7	Aqib	bandraeast	87845614
8	8	samikapadia	vikroli	8484655

2.
```
CREATE TABLE EMP_WORK1 AS
    SELECT EMPID,DEPARTMENT,SALARY
    FROM EMPLOYEE

SELECT * FROM EMP_WORK1;
```

```
Table EMP_WORK1 created.

Elapsed: 00:00:00.041
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.011 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000
6	6	backend	60000
7	7	AI	70000
8	8	IATA	80000

3. Insert New Records In Base Table then Verify Whethere table is Updated or Not.

```
INSERT INTO EMPLOYEE VALUES
    (9,'amey','finance',90000,'ratnagiri','84518285')
```

```
1 row inserted.

Elapsed: 00:00:00.003
```

```
SELECT * FROM EMPLOYEE;
```

EMPID	NAME	DEPARTMENT	SALARY	LOCATION	CONTACTNO
1	zamam	cs	10000	mumbai	87452103
2	afzal	it	20000	sion	85464654
3	fahad	data sci	30000	kurla	85456432
4	zoheb	cyber sec	40000	bandra	87985465
5	sami	EH	50000	byculla	84984354
6	hanzla	backend	60000	thane	87984658
7	Aqib	AI	70000	bandraeast	87845614
8	samikapadia	IATA	80000	vikroli	8484655
9	amey	finance	90000	ratnagiri	84518285

```
SELECT * FROM EMP_PERSONAL1;
```

	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354
6	6	hanzla	thane	87984658
7	7	Aqib	bandraeast	87845614
8	8	samikapadia	vikroli	8484655

```
SELECT * FROM EMP_WORK1;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.011 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000
6	6	backend	60000
7	7	AI	70000
8	8	IATA	80000

- Create two Separate Fragments **Using Materialized View.**

1.

```
CREATE MATERIALIZED VIEW EMP_PERSONAL2 AS
    SELECT EMPID,NAME,LOCATION,CONTACTNO
    FROM EMPLOYEE

SELECT * FROM EMP_PERSONAL2;
```

```
Materialized view EMP_PERSONAL2 created.

Elapsed: 00:00:00.087
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.012 seconds

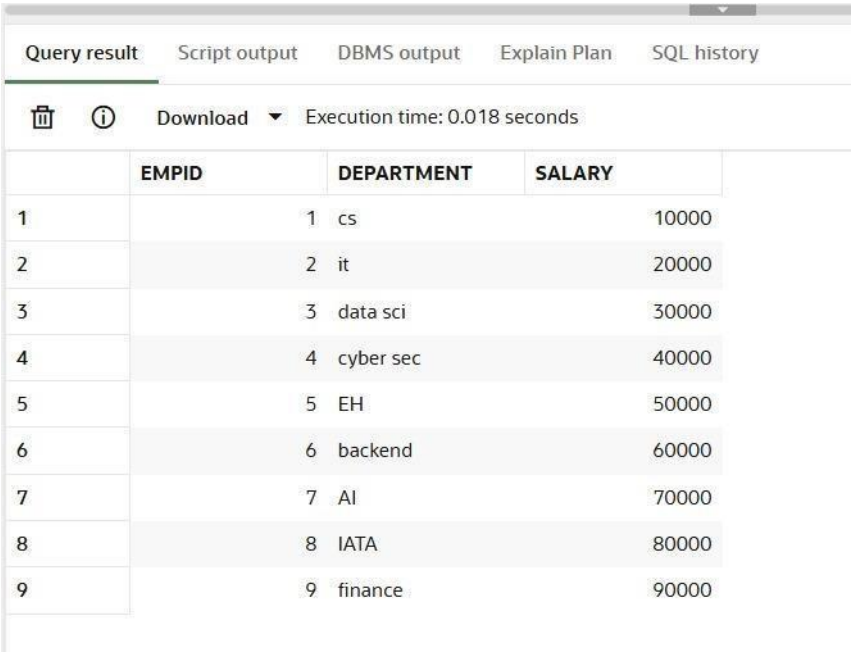
	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354
6	6	hanzla	thane	87984658
7	7	Aqib	bandraeast	87845614
8	8	samikapadia	vikroli	8484655
9	9	amey	ratnagiri	84518285

2.
```
CREATE MATERIALIZED VIEW EMP_WORK2 AS
    SELECT EMPID,DEPARTMENT,SALARY
    FROM EMPLOYEE

SELECT * FROM EMP_WORK2;
```

```
Materialized view EMP_WORK2 created.

Elapsed: 00:00:00.104
```



Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.018 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000
6	6	backend	60000
7	7	AI	70000
8	8	IATA	80000
9	9	finance	90000

3. Insert New Data In Base Table then Verify Whethere Materialized View is Updated or Not

```
INSERT INTO EMPLOYEE VALUES
   (10,'john','deve',100000,'bangalore','84546485'),
(11,'Alice','frnot end',110000,'chennai','85446565')
```

```
1 row inserted.

Elapsed: 00:00:00.008
```

```
SELECT * FROM EMPLOYEE;
```

	EMPID	NAME	DEPARTMENT	SALARY	LOCATION	CONTACTNO
1	10	john	deve	100000	bangalore	84546485
2	11	Alice	frnot end	110000	chennai	85446565
3	1	zamam	cs	10000	mumbai	87452103
4	2	afzal	it	20000	sion	85464654
5	3	fahad	data sci	30000	kurla	85456432
6	4	zoheb	cyber sec	40000	bandra	87985465
7	5	sami	EH	50000	byculla	84984354
8	6	hanzla	backend	60000	thane	87984658
9	7	Aqib	AI	70000	bandraeast	87845614
10	8	samikapadia	IATA	80000	vikroli	8484655
11	9	amey	finance	90000	ratnagiri	84518285

4. Check Materialized View Data Before Refresh.

```
SELECT * FROM EMP_PERSONAL2;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.009 seconds

	EMPID	NAME	LOCATION	CONTACTNO
1	1	zamam	mumbai	87452103
2	2	afzal	sion	85464654
3	3	fahad	kurla	85456432
4	4	zoheb	bandra	87985465
5	5	sami	byculla	84984354
6	6	hanzla	thane	87984658
7	7	Aqib	bandraeast	87845614
8	8	samikapadia	vikroli	8484655

```
SELECT * FROM EMP_WORK2;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.011 seconds

	EMPID	DEPARTMENT	SALARY
1	1	cs	10000
2	2	it	20000
3	3	data sci	30000
4	4	cyber sec	40000
5	5	EH	50000
6	6	backend	60000
7	7	AI	70000
8	8	IATA	80000

5. Refresh Materialized View .

```
EXEC DBMS_MVIEW.REFRESH('Emp_Personal2');
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.045
```

```
EXEC DBMS_MVIEW.REFRESH('Emp_Work2');
```

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.067
```

6. Check Materialized View Data After Refresh .

```
SELECT * FROM EMP_PERSONAL2;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.003 seconds

	EMPID	NAME	LOCATION	CONTACTNO
1	10	john	bangalore	84546485
2	11	Alice	chennai	85446565
3	1	zamam	mumbai	87452103
4	2	afzal	sion	85464654
5	3	fahad	kurla	85456432
6	4	zoheb	bandra	87985465
7	5	sami	byculla	84984354
8	6	hanzla	thane	87984658
9	7	Aqib	bandraeast	87845614
10	8	samikapadia	vikroli	8484655
11	9	amey	ratnagiri	84518285

```
SELECT * FROM EMP_WORK2;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.128 seconds

	EMPID	DEPARTMENT	SALARY
1	10	deve	100000
2	11	frnot end	110000
3	1	cs	10000
4	2	it	20000
5	3	data sci	30000
6	4	cyber sec	40000
7	5	EH	50000
8	6	backend	60000
9	7	AI	70000
10	8	IATA	80000
11	9	finance	90000

Exercise:

1. Create the Books table and insert at least 6 records with different authors, publishers, and categories.

2. Perform Vertical Fragmentation of the Books relation into two fragments:

I. Fragment 1 (Book Info): (BookID, Title, Author, Publisher)
II. Fragment 2 (Sales Info): (BookID, Year, Price, Category)

- Implement fragmentation using:
  (a) Views
  (b) Materialized Views (ON DEMAND refresh)

3. Insert 2 new records into the Books table.

I. Show whether the new records appear in both fragments.
II. If using materialized views, demonstrate the refresh process.
III. Compare the behavior of View vs. Materialized View.

4. Write an SQL query to reconstruct the original Books relation from the fragmented tables.

I. Show the combined output.
II. Explain why BookID must be replicated in both fragments.

OUTPUTS:

1. Create the Books table and insert at least 6 records with different authors, publishers, and categories.

```
CREATE TABLE Books  (
    BookID INT PRIMARY KEY,
    Title VARCHAR(100),Author VARCHAR(100),
    Publisher VARCHAR(100),Year INT,
    Price DECIMAL(6,2),Category VARCHAR(50)
);

INSERT INTO Books VALUES
(1, 'Midnight's Children', 'Salman Rushdie', 'Jonathan Cape', 1981,
350.00, 'Fiction'),
(2, 'The White Tiger', 'Aravind Adiga', 'HarperCollins India', 2008,
420.00, 'Fiction'),
(3, 'Five Point Someone', 'Chetan Bhagat', 'Rupa Publications', 2004,
300.00, 'Contemporary Fiction'),
(4, 'The Alchemist', 'Paulo Coelho', 'HarperOne', 1988, 275.00,
'Fiction'),
(5, 'Ikigai', 'Héctor García', 'Penguin Random House', 2016, 300.00,
'Self-help'),
(6, 'Sapiens', 'Yuval Noah Harari', 'Harper', 2011, 450.00, 'Non-
fiction');
```

```
Table BOOKS created.                    6 rows inserted.

Elapsed: 00:00:00.016                   Elapsed: 00:00:00.439
```

2. Perform Vertical Fragmentation of the Books relation into two fragments:

A. Fragmentation using View

I. Fragment 1 (Book Info): (BookID, Title, Author, Publisher)

```
CREATE VIEW Book_Info AS
SELECT BookID, Title, Author, Publisher
FROM Books;

SELECT * FROM BOOK_INFO;
```

```
View BOOK_INFO created.

Elapsed: 00:00:00.008
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper

II. Fragment 2 (Sales Info): (BookID, Year, Price, Category)

```
CREATE VIEW Sales_Info AS
SELECT BookID, Year, Price, Category
FROM Books;

SELECT * FROM SALES_INFO;
```

```
View SALES_INFO created.

Elapsed: 00:00:00.008
```

	BOOKID	YEAR	PRICE	CATEGORY
1	1	1981	350	Fiction
2	2	2008	420	Fiction
3	3	2004	300	Contemporary Fiction
4	4	1988	275	Fiction
5	5	2016	300	Self-help
6	6	2011	450	Non-fiction

B. Fragmentation using Materialized View

I. Fragment 1 (Book _nfo1): (BookID, Title, Author, Publisher)

```
CREATE MATERIALIZED VIEW Book_Info1 AS
SELECT BookID, Title, Author, Publisher
FROM Books;

SELECT * FROM BOOK_INFO1;
```

```
Materialized view BOOK_INFO1 created.

Elapsed: 00:00:00.112
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper

II. Fragment 2 (Sales_Info2): (BookID, Year, Price, Category)

```
CREATE MATERIALIZED VIEW Sales_Info1 AS
SELECT BookID, Year, Price, Category
FROM Books;

SELECT * FROM SALES_INFO1;
```

```
Materialized view SALES_INFO1 created.

Elapsed: 00:00:00.313
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper

3. Insert 2 new records into the Books table.

```
INSERT INTO Books VALUES
(7, 'Deep Work', 'Cal Newport', 'Grand Central', 2016, 18.50,
'Productivity'),
(8, 'Dune', 'Frank Herbert', 'Chilton Books', 1965, 25.00, 'Sci-
Fi');
```

```
2 rows inserted.

Elapsed: 00:00:00.005
```

I. Show whether the new records appear in both fragments.

I. For view

```
SELECT * FROM BOOK_INFO;
SELECT * FROM SALES_INFO;
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper
7	7	Deep Work	Cal Newport	Grand Central
8	8	Dune	Frank Herbert	Chilton Books

	BOOKID	YEAR	PRICE	CATEGORY
1	1	1981	350	Fiction
2	2	2008	420	Fiction
3	3	2004	300	Contemporary Fiction
4	4	1988	275	Fiction
5	5	2016	300	Self-help
6	6	2011	450	Non-fiction
7	7	2016	18.5	Productivity
8	8	1965	25	Sci-Fi

II. If using materialized views, demonstrate the refresh process.

II. Before refresh

```
SELECT * FROM BOOK_INFO1;
SELECT * FROM SALES_INFO1;
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper

III. Refresh

```
EXEC DBMS_MVIEW.REFRESH('BOOK_INFO1');
EXEC DBMS_MVIEW.REFRESH('SALES_INFO1');
```

```
PL/SQL procedure successfully completed.

Elapsed: 00:00:00.056

PL/SQL procedure successfully completed.

Elapsed: 00:00:00.043
```

IV. After Refresh

```
SELECT * FROM BOOK_INFO1;
SELECT * FROM SALES_INFO1;
```

	BOOKID	TITLE	AUTHOR	PUBLISHER
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape
2	2	The White Tiger	Aravind Adiga	HarperCollins India
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications
4	4	The Alchemist	Paulo Coelho	HarperOne
5	5	Ikigai	Héctor García	Penguin Random House
6	6	Sapiens	Yuval Noah Harari	Harper
7	7	Deep Work	Cal Newport	Grand Central
8	8	Dune	Frank Herbert	Chilton Books

	BOOKID	YEAR	PRICE	CATEGORY
1	1	1981	350	Fiction
2	2	2008	420	Fiction
3	3	2004	300	Contemporary Fiction
4	4	1988	275	Fiction
5	5	2016	300	Self-help
6	6	2011	450	Non-fiction
7	7	2016	18.5	Productivity
8	8	1965	25	Sci-Fi

III. Compare the behavior of View vs. Materialized View.

View:

A view acts as a virtual table that represents the result of a query executed directly on the Books table at the time of access. It does not store data physically. Each time the view is queried, it dynamically fetches the latest data from the base table, ensuring that any recent additions, updates, or deletions in the Books table are immediately reflected. This characteristic makes views highly suitable for applications requiring real-time accuracy. Additionally, views require no maintenance in terms of refreshing or updating.

Materialized View:

A materialized view is a physical copy of the result set derived from a query on the Books table, stored as an actual table within the database. Unlike a view, it does not automatically reflect changes made to the underlying Books table. Instead, it requires a manual or scheduled refresh to update its data. The key advantage of materialized views lies in their ability to provide significantly improved query performance, especially for complex aggregations or large datasets, by avoiding the need to recompute the query results each time. However, this performance gain comes at the cost of data latency and additional maintenance overhead.

4. Write an SQL query to reconstruct the original Books relation from the fragmented tables.

I. Show the combined output.

```
SELECT
    B.BookID, B.Title, B.Author, B.Publisher,
    S.Year, S.Price, S.Category
FROM Book_Info B
JOIN Sales_Info S ON B.BookID = S.BookID;
```

	BOOKID	TITLE	AUTHOR	PUBLISHER	YEAR	PRICE	CATEGORY
1	1	Midnight's Children	Salman Rushdie	Jonathan Cape	1981	350	Fiction
2	2	The White Tiger	Aravind Adiga	HarperCollins India	2008	420	Fiction
3	3	Five Point Someone	Chetan Bhagat	Rupa Publications	2004	300	Contemporary Fiction
4	4	The Alchemist	Paulo Coelho	HarperOne	1988	275	Fiction
5	5	Ikigai	Héctor García	Penguin Random House	2016	300	Self-help
6	6	Sapiens	Yuval Noah Harari	Harper	2011	450	Non-fiction
7	7	Deep Work	Cal Newport	Grand Central	2016	18.5	Productivity
8	8	Dune	Frank Herbert	Chilton Books	1965	25	Sci-Fi

The combine output of view & materialized view are same because they contain attributes.

II. Explain why BookID must be replicated in both fragments.

→ BookID must be replicated in all fragments because it is the primary key that uniquely identifies each record in the Books table. Replicating it ensures Whether the table is horizontally or vertically fragmented, BookID acts as the common reference that allows the system to reassemble the original table, perform joins, maintain referential integrity, and support update or delete operations. Without replicating BookID in every fragment, the records may lose their identity, making it difficult to manage and relate data correctly across fragments.

Practical No: 2

Aim : To study and implement **Horizontal Fragmentation** of a database relation using **Tables, Views, and Materialized Views**.

Horizontal Fragmentation

Definition:

Horizontal fragmentation is a database design technique where a table (relation) is **divided into subsets of rows** based on specific conditions applied to one or more attributes. Each fragment contains **some of the rows** of the original table, but all columns remain the same.

-- Step 1: Create Base Table

```
CREATE TABLE Employee (
    EmpID INT PRIMARY KEY,
    EmpName VARCHAR(50),
    Department VARCHAR(30),
    Salary DECIMAL(10,2),
    Location VARCHAR(30)
);
```

```
SQL> CREATE TABLE Employee (
        EmpID INT PRIMARY KEY,
        EmpName VARCHAR(50),
        Department VARCHAR(30),...
Show more...


Table EMPLOYEE created.

Elapsed: 00:00:00.014
```

```
INSERT INTO Employee VALUES
(1,'AMIT','HR',40000,'DELHI'),
(2,'AREEB','IT',80000,'MUMBAI'),
(3,'KAIF','FINANCE',70000,'AMRITSAR'),
(4,'ZAMAM','HR',60000,'KASHMIR'),
(5,'SAMI','IT',50000,'KOLKATA'),
(6,'AYESHA','FINANCE',90000,'CHENNAI'),
(7,'MARIYA','HR',450000,'PUNJAB'),
(8,'SARA','IT',410000,'DELHI'),
(9,'AMMAR','FINANCE',240000,'MUMBAI'),
(10,'SAUD','HR',340000,'PUNE');
```

```
SQL> INSERT INTO Employee VALUES
     (1,'AMIT','HR',40000,'DELHI'),
     (2,'AREEB','IT',80000,'MUMBAI'),
     (3,'KAIF','FINANCE',70000,'AMRITSAR'),...
Show more...


10 rows inserted.

Elapsed: 00:00:00.021
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	1	AMIT	HR	40000	DELHI
2	2	AREEB	IT	80000	MUMBAI
3	3	KAIF	FINANCE	70000	AMRITSAR
4	4	ZAMAM	HR	60000	KASHMIR
5	5	SAMI	IT	50000	KOLKATA
6	6	AYESHA	FINANCE	90000	CHENNAI
7	7	MARIYA	HR	450000	PUNJAB
8	8	SARA	IT	410000	DELHI
9	9	AMMAR	FINANCE	240000	MUMBAI
10	10	SAUD	HR	340000	PUNE

- **Fragment 1:** Create table **EMP_IT_HIGH** to store IT employees with salary **greater than 60,000.**

```
CREATE TABLE EMP_IT_HIGH AS
SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000;
Select * from EMP_IT_HIGH;
```

```
SQL> CREATE TABLE EMP_IT_HIGH AS
     SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000


Table EMP_IT_HIGH created.

Elapsed: 00:00:00.198
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	2	AREEB	IT	80000	MUMBAI
2	8	SARA	IT	410000	DELHI

- **Fragment 2**: Create view **EMP_FIN_BTW1** to store Finance employees with **salary between 50,000 and 80,000.**

```
CREATE TABLE EMP_FIN_BTW AS
SELECT * FROM EMPLOYEE WHERE Department='FINANCE' AND Salary BETWEEN 50000 AND 80000;
select * from EMP_FIN_BTW;
```

```
SQL> CREATE TABLE EMP_FIN_BTW AS
     SELECT * FROM EMPLOYEE WHERE Department='FINANCE' AND Salary BETWEEN 50000 AND 80000


Table EMP_FIN_BTW created.

Elapsed: 00:00:00.039
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	3	KAIF	FINANCE	70000	AMRITSAR

- **Fragment 3:** Create table **EMP_HR** to store all employees from the HR department.

```
CREATE TABLE EMP_HR AS
SELECT * FROM EMPLOYEE WHERE Department='HR';
Select * from EMP_HR;
```

```
SQL> CREATE TABLE EMP_HR AS
     SELECT * FROM EMPLOYEE WHERE Department='HR'


Table EMP_HR created.

Elapsed: 00:00:00.036
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	1	AMIT	HR	40000	DELHI
2	4	ZAMAM	HR	60000	KASHMIR
3	7	MARIYA	HR	450000	PUNJAB
4	10	SAUD	HR	340000	PUNE

- **Fragment 4**: Create table **EMP_OTHER** to store all employees who do not belong to IT, Finance, or HR, or whose salaries do not meet the conditions of the other fragments.

```
CREATE TABLE EMP_OTHER AS
SELECT * FROM EMPLOYEE
WHERE (Department NOT IN ('IT','FINANCE','HR'))
   OR (Department='IT' AND Salary<=60000)
   OR (Department='FINANCE' AND (Salary<50000 OR Salary>80000));
Select * from EMP_OTHER;
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	5	SAMI	IT	50000	KOLKATA
2	6	AYESHA	FINANCE	90000	CHENNAI
3	9	AMMAR	FINANCE	240000	MUMBAI

- **Step 4: Reconstruct the original EMPLOYEE table by using UNION on all fragments (EMP_IT_HIGH, EMP_FIN_BTW, EMP_HR, and EMP_OTHER).**

```
SELECT * FROM EMP_IT_HIGH
UNION
SELECT * FROM EMP_FIN_BTW
UNION
SELECT * FROM EMP_HR
UNION
SELECT * FROM EMP_OTHER;
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	2	AREEB	IT	80000	MUMBAI
2	8	SARA	IT	410000	DELHI
3	3	KAIF	FINANCE	70000	AMRITSAR
4	1	AMIT	HR	40000	DELHI
5	4	ZAMAM	HR	60000	KASHMIR
6	7	MARIYA	HR	450000	PUNJAB
7	10	SAUD	HR	340000	PUNE
8	5	SAMI	IT	50000	KOLKATA
9	6	AYESHA	FINANCE	90000	CHENNAI
10	9	AMMAR	FINANCE	240000	MUMBAI

- **Step 3: Create views for the fragments (EMP_IT_HIGH1, EMP_FIN_BTW1, EMP_HR1) to dynamically display the latest data from the EMPLOYEE table.**

```
CREATE VIEW EMP_IT_HIGH1 AS
SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000;

CREATE VIEW EMP_FIN_BTW1 AS
SELECT * FROM EMPLOYEE WHERE Department='FINANCE' AND Salary BETWEEN 50000 AND 80000;

CREATE VIEW EMP_HR1 AS
SELECT * FROM EMPLOYEE WHERE Department='HR';
```

```
SQL> CREATE VIEW EMP_IT_HIGH1 AS
     SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000



View EMP_IT_HIGH1 created.

Elapsed: 00:00:00.008
```

```
SQL> CREATE VIEW EMP_FIN_BTW1 AS
     SELECT * FROM EMPLOYEE WHERE Department='FINANCE' AND Salary BETWEEN 50000 AND 80000



View EMP_FIN_BTW1 created.

Elapsed: 00:00:00.006
```

```
SQL> CREATE VIEW EMP_HR1 AS
     SELECT * FROM EMPLOYEE WHERE Department='HR'



View EMP_HR1 created.

Elapsed: 00:00:00.006
```

- **Step 4: Insert new data into the EMPLOYEE table and check whether the changes are automatically reflected in the views.**

```
INSERT INTO EMPLOYEE VALUES
(11,'KAMLESH','HR',60000,'DELHI'),
(12,'HAMZAH','IT',80000,'MUMBAI');

SELECT * FROM EMP_IT_HIGH1
```

```
SQL> INSERT INTO EMPLOYEE VALUES
     (11,'KAMLESH','HR',60000,'DELHI'),
     (12,'HAMZAH','IT',80000,'MUMBAI')


2 rows inserted.

Elapsed: 00:00:00.004
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	2	AREEB	IT	80000	MUMBAI
2	8	SARA	IT	410000	DELHI
3	12	HAMZAH	IT	80000	MUMBAI

- **Step 5: Create a materialized view (EMP_IT_HIGH2) to store a snapshot of IT employees with salary greater than 60,000, and refresh it manually to update new data**

```
CREATE MATERIALIZED VIEW EMP_IT_HIGH2 AS
SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000;
```

```
SQL> CREATE MATERIALIZED VIEW EMP_IT_HIGH2 AS
     SELECT * FROM EMPLOYEE WHERE Department='IT' AND Salary>60000


Materialized view EMP_IT_HIGH2 created.

Elapsed: 00:00:00.235
```

- **Step 6: Insert new employee (not visible until refresh).**

```
INSERT INTO EMPLOYEE VALUES
(13,'TAHIR','IT',70000,'DELHI');
```

```
SQL> INSERT INTO EMPLOYEE VALUES
     (13,'TAHIR','IT',70000,'DELHI')


1 row inserted.

Elapsed: 00:00:00.005
```

■ won't show TAHIR yet

```
SELECT * FROM EMP_IT_HIGH2;
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	2	AREEB	IT	80000	MUMBAI
2	8	SARA	IT	410000	DELHI
3	12	HAMZAH	IT	80000	MUMBAI

Refresh: Update the materialized view (EMP_IT_HIGH2) manually using DBMS_MVIEW.REFRESH to reflect the latest data from the base table.

```
EXEC DBMS_MVIEW.REFRESH('EMP_IT_HIGH2');
```

```
SQL> EXEC DBMS_MVIEW.REFRESH('EMP_IT_HIGH2')


PL/SQL procedure successfully completed.

Elapsed: 00:00:00.102
```

Check the Table After Refresh

```
SELECT * FROM EMP_IT_HIGH2;
```

	EMPID	EMPNAME	DEPARTMENT	SALARY	LOCATION
1	2	AREEB	IT	80000	MUMBAI
2	8	SARA	IT	410000	DELHI
3	12	HAMZAH	IT	80000	MUMBAI
4	13	TAHIR	IT	70000	DELHI

PRACTICAL NO: 3

Aim : Create different types that include attributes and methods. Define tables for these types by adding a sufficient number of tuples . Demonstrate insert, update and delete operations on these tables. Execute queries on them

Question No : 1

I. Create Student type with the following field:
   - Roll_no number
   - Name varchar
   - Marks number

II. Create table Student for Student type Object.

III. Insert data into object table Student.

IV. Display all the records of object table Student.

V. Display name marks of Student whose rollno 2.

VI. Update marks of Student whose rollno is 1.

VII. Delete Student whose rollno is 3.

OUTPUTS:

1. Create Student type with the following field:

   Roll_no number , Name varchar, Marks number

```
CREATE OR REPLACE TYPE STUDENTTYPE AS OBJECT (
         ROLL_NO NUMBER,
         NAME    VARCHAR(20),
         MARKS   NUMBER
);
```

```
Type STUDENTTYPE compiled

Elapsed: 00:00:00.024
```

2. Create table Student for Student type Object.

```
CREATE TABLE STUDENTS1 OF STUDENTTYPE
```

```
Table STUDENTS1 created.

Elapsed: 00:00:00.020
```

3. Insert data into object table Student.

```
INSERT INTO STUDENT1 VALUES(STUDENTTYPE1(1,'ZAMAM',100));
INSERT INTO STUDENT1 VALUES(STUDENTTYPE1(2,'FAHAD',69));
INSERT INTO STUDENT1 VALUES(STUDENTTYPE1(3,'ZOHEB',79));
INSERT INTO STUDENT1 VALUES(STUDENTTYPE1(4,'AFZAL',90));
INSERT INTO STUDENT1 VALUES(STUDENTTYPE1(5,'SAMI',0));
```

```
1 row inserted.
```

- Elapsed: 00:00:00.002

```
1 row inserted.
```

- Elapsed: 00:00:00.001

```
1 row inserted.
```

- Elapsed: 00:00:00.001

```
1 row inserted.
```

- Elapsed: 00:00:00.002

```
1 row inserted.
```

- Elapsed: 00:00:00.001

4. Display all the records of object table Student.

```
SELECT * FROM STUDENTS1;
```

Query result | Script output | DBMS output | Explain Plan | SQL history

Download | Execution time: 0.008 seconds

	ROLL_NO	NAME	MARKS
1	1	ZAMAM	100
2	2	FAHAD	69
3	3	ZOHEB	79
4	4	AFZAL	90
5	5	SAMI	0

5. Display name marks of Student whose rollno 3.

```
SELECT
    S.NAME,
    S.MARKS,
    S.ROLL_NO
FROM
    STUDENTS1 S
WHERE
    S.ROLL_NO = 3;
```

	NAME	MARKS	ROLL_NO
1	ZOHEB	79	3

6. Update marks of Student whose rollno is 1.

```
UPDATE STUDENTS1 S SET S.MARKS = 90 WHERE S.ROLL_NO = 1;
```

```
1 row updated.

Elapsed: 00:00:00.004
```

```
SELECT * FROM STUDENTS1;
```

	ROLL_NO	NAME	MARKS
1	1	ZAMAM	90
2	2	FAHAD	69
3	3	ZOHEB	79
4	4	AFZAL	90
5	5	SAMI	0

7. Delete Student whose rollno is 3.

```
DELETE FROM STUDENTS1 WHERE ROLL_NO = 3;
```

```
1 row deleted.

Elapsed: 00:00:00.003
```

```
SELECT * FROM STUDENTS1;
```

	ROLL_NO	NAME	MARKS
1	1	ZAMAM	90
2	2	FAHAD	69
3	4	AFZAL	90
4	5	SAMI	0

Question No : 2

I. Create Type Addresstype with the Following Field:
   - Street varchar
   - City varchar
   - Pincode number

II. Create Type Person Containing Addresstype the Following Field:
   - Person_id number
   - Name varchar
   - DOB date
   - Address Addresstype
   - Member Function get_age

III. Implement get_age Function.

IV. Create Table for Object.

V. Insert Record into Table.

VI. Query Object Data.

OUTPUTS:

1. Create Type Addresstype with the Following Field:

   Street varchar , City varchar, Pincode number

```
CREATE OR REPLACE TYPE ADDRESSTYPE AS OBJECT (
    STREET VARCHAR(20),
    CITY VARCHAR(20),
    PINCODE   NUMBER
);
```

```
Type ADDRESSTYPE compiled

Elapsed: 00:00:00.014
```

2. Create Type Person Containing Addresstype the Following Field:

   Person_id number, Name varchar, DOB date, Address Addresstype, Member Function get_age

```
CREATE OR REPLACE TYPE PERSONTYPE1 AS OBJECT (
        PERSON_ID NUMBER,
        NAME VARCHAR(30),
        DOB  DATE,
        ADDRESS ADDRESSTYPE,
        MEMBER FUNCTION GET_AGE RETURN NUMBER
);
```

```
Type PERSONTYPE1 compiled

Elapsed: 00:00:00.049
```

3. Implement get_age Function.

```
CREATE OR REPLACE TYPE BODY PERSONTYPE1 AS
    MEMBER FUNCTION GET_AGE RETURN NUMBER IS
    BEGIN
        RETURN TRUNC(MONTHS_BETWEEN(SYSDATE, DOB) / 12);
    END;
END;
```

```
Type Body PERSONTYPE1 compiled

Elapsed: 00:00:00.020
```

4. Create Table for Object.

```
CREATE TABLE PERSONS OF PERSONTYPE1 (
    CONSTRAINT PK_PERSON PRIMARY KEY ( PERSON_ID )
) OBJECT IDENTIFIER IS PRIMARY KEY;
```

```
Table PERSONS created.

Elapsed: 00:00:00.032
```

5. Insert Record into Table.

```
INSERT INTO PERSONS VALUES ( PERSONTYPE1(1,
                                         'Zamam Ansari',
                                         DATE '2005-05-07',
                                         ADDRESSTYPE('pipe road ', 'Mumbai',
                                         400070)) );

INSERT INTO PERSONS VALUES ( PERSONTYPE1(2,
                                         'Fahad Ansari',
                                         DATE '2006-06-24',
                                         ADDRESSTYPE('LIG Road ', 'MUMBAI ',
                                         610001)) );

INSERT INTO PERSONS VALUES ( PERSONTYPE1(3,
                                         'Shiman Ansari',
                                         DATE '2003-03-07',
                                         ADDRESSTYPE('MIG ROAD    ', 'MUMBAI',
                                         500999)) );
```

```
1 row inserted.

Elapsed: 00:00:00.031

1 row inserted.

Elapsed: 00:00:00.002

1 row inserted.

Elapsed: 00:00:00.002
```

6. Query Object Data.

- Select Query

```
SELECT * FROM PERSONS;
```

	PERSON_ID	NAME	DOB	ADDRESS
1	1	ZAMAM ANSARI	5/7/2005, 12:00:00	{"street":"PIPE ROAD
2	2	FAHAD ANSARI	6/4/2006, 12:00:00	{"street":"LIG R(
3	3	SHIMAN ANSARI	3/7/2003, 12:00:00	{"street":"MIG ROAD'
4	4	AQIB ANSARI	5/7/2007, 12:00:00	{"street":"BANDRA R(

```
SELECT P.NAME,P.DOB, P.ADDRESS.CITY FROM PERSONS P;
```

	NAME	ADDRESS.CITY	DOB
1	ZAMAM ANSARI	MUMBAI	5/7/2005, 12:00:00
2	FAHAD ANSARI	MUMBAI	6/4/2006, 12:00:00
3	SHIMAN ANSARI	MUMBAI	3/7/2003, 12:00:00
4	AQIB ANSARI	MUMBAI	5/7/2007, 12:00:00

```
SELECT P.NAME, P.GET_AGE() AS AGE FROM PERSONS P;
```

	NAME	AGE
1	ZAMAM ANSARI	20
2	FAHAD ANSARI	19
3	SHIMAN ANSARI	22
4	AQIB ANSARI	18

- Update query

```
SELECT
    P.PERSON_ID,
    P.NAME,
    P.ADDRESS.CITY
FROM
    PERSONS P
```

```
UPDATE PERSONS P
SET
     P.ADDRESS.CITY = 'JAMMU'
WHERE
     P.PERSON_ID = 1;
```

```
SELECT
    P.PERSON_ID,
    P.NAME,
    P.ADDRESS.CITY
FROM
    PERSONS P;
```

	PERSON_ID	NAME	ADDRESS.CITY
1	1	ZAMAM ANSARI	MUMBAI
2	2	FAHAD ANSARI	MUMBAI
3	3	SHIMAN ANSARI	MUMBAI
4	4	AQIB ANSARI	MUMBAI

```
1 row updated.

Elapsed: 00:00:00.006
```

	PERSON_ID	NAME	ADDRESS.CITY
1	1	ZAMAM ANSARI	JAMMU
2	2	FAHAD ANSARI	MUMBAI
3	3	SHIMAN ANSARI	MUMBAI
4	4	AQIB ANSARI	MUMBAI

- Delete query

```
SELECT
   P.PERSON_ID,
   P.NAME,
   P.ADDRESS.CITY
FROM
   PERSONS P

DELETE FROM PERSONS
 WHERE
   PERSON_ID = 3;

SELECT
   P.PERSON_ID,
   P.NAME,
   P.ADDRESS.CITY
FROM
   PERSONS P;
```

	PERSON_ID	NAME	ADDRESS.CITY
1	1	ZAMAM ANSARI	JAMMU
2	2	FAHAD ANSARI	MUMBAI
3	3	SHIMAN ANSARI	MUMBAI
4	4	AQIB ANSARI	MUMBAI

```
1 row deleted.

Elapsed: 00:00:00.006
```

	PERSON_ID	NAME	ADDRESS.CITY
1	1	ZAMAM ANSARI	JAMMU
2	2	FAHAD ANSARI	MUMBAI
3	4	AQIB ANSARI	MUMBAI

Practical No: 4

Aim: *To study and implement storage, retrieval, and manipulation of **XML** data in a relational database using the **XMLTYPE** data type, **XPath** functions (**EXTRACTVALUE**), and **XMLTABLE** for converting **XML** into relational form.*

Definition:

***XML (Extensible Markup Language)** is a structured and self-descriptive format used to store and exchange data. In databases, the XMLTYPE data type is used to store XML documents. It allows users to insert, query, update, and delete XML data using SQL/XML functions such as EXTRACTVALUE, XMLQUERY, and XMLTABLE. This provides flexibility to handle both structured (relational) and semi-structured (XML) data within the same database.*

Step 1 — Create table to store XML

```
1   CREATE TABLE employee_xml(
2       id NUMBER PRIMARY KEY,
3       emp_data XMLTYPE
4   );
5
```

```
SQL> CREATE TABLE employee_xml(
         id NUMBER PRIMARY KEY,
         emp_data XMLTYPE
     )


Table EMPLOYEE_XML created.

Elapsed: 00:00:00.067
```

Step 2: Insert employee details in XML format into the employee_xml table using the XMLTYPE data type.

```
INSERT INTO employee_xml VALUES(
  101,
  XMLTYPE('<Employee>
    <name>zamam ansari</name>
    <department>CS</department>
    <salary>60000</salary>
  </Employee>')
);
```

```
SQL> INSERT INTO employee_xml VALUES(
       101,
       XMLTYPE('<Employee>
         <name>zamam ansari</name>...
Show more...


1 row inserted.

Elapsed: 00:00:00.050
```

```
INSERT INTO employee_xml VALUES(
  102,
  XMLTYPE('<Employee>
    <name>fahad ansari</name>
    <department>IT</department>
    <salary>65000</salary>
  </Employee>')
);
```

```
SQL> INSERT INTO employee_xml VALUES(
       102,
       XMLTYPE('<Employee>
         <name>fahad ansari</name>...
Show more...



1 row inserted.

Elapsed: 00:00:00.003
```

```
INSERT INTO employee_xml VALUES(
  103,
  XMLTYPE('<Employee>
    <name>afzal shaikh</name>
    <department>MSC-CS</department>
    <salary>70000</salary>
  </Employee>')
);
```

```
SQL> INSERT INTO employee_xml VALUES(
       103,
       XMLTYPE('<Employee>
         <name>afzal shaikh</name>...
Show more...



1 row inserted.

Elapsed: 00:00:00.004
```

```
INSERT INTO employee_xml VALUES(
  104,
  XMLTYPE('<Employee>
    <name>Akshay</name>
    <department>BMS</department>
    <salary>40000</salary>
  </Employee>')
);
```

```
SQL> INSERT INTO employee_xml VALUES(
       104,
       XMLTYPE('<Employee>
         <name>Akshay</name>...
Show more...



1 row inserted.

Elapsed: 00:00:00.004
```

```
INSERT INTO employee_xml VALUES(
  105,
  XMLTYPE('<Employee>
    <name>shariq</name>
    <department>MSC IT</department>
    <salary>80000</salary>
  </Employee>')
);
```

```
SQL> INSERT INTO employee_xml VALUES(
       105,
       XMLTYPE('<Employee>
         <name>shariq</name>...
Show more...



1 row inserted.

Elapsed: 00:00:00.011
```

Step 3: Display all rows from the employee_xml table to view the stored XML data.

```
SELECT * FROM employee_xml;
```

	ID	EMP_DATA
1	101	<Employee> <name>
2	102	<Employee> <name>
3	103	<Employee> <name>
4	104	<Employee> <name>
5	105	<Employee> <name>

Step 4: Extract the employee names from the XML documents using EXTRACTVALUE and display them.

```
SELECT EXTRACTVALUE(emp_data, '/Employee/name') AS Employee_name
FROM employee_xml;
```

	EMPLOYEE_NAME
1	zamam ansari
2	fahad ansari
3	afzal shaikh
4	Akshay
5	shariq

Step 5: Extract the employee salaries from the XML documents using EXTRACTVALUE and display them.

```
SELECT EXTRACTVALUE(emp_data, '/Employee/salary') AS Employee_salary
FROM employee_xml;
```

	EMPLOYEE_SALARY
1	60000
2	65000
3	70000
4	40000
5	80000

Step 6: Extract both employee names and salaries from the XML documents and filter for employees with salary greater than 50,000.

```
SELECT EXTRACTVALUE(emp_data, '/Employee/name') AS name,
       EXTRACTVALUE(emp_data, '/Employee/salary') AS salary
FROM employee_xml
WHERE TO_NUMBER(EXTRACTVALUE(emp_data, '/Employee/salary')) > 50000;
```

	NAME	SALARY
1	zamam ansari	60000
2	fahad ansari	65000
3	afzal shaikh	70000
4	shariq	80000

Step 7: Update the XML data of a specific employee (e.g., Fahad Ansari) to modify their salary.

```
UPDATE employee_xml
SET emp_data = XMLTYPE('<Employee>
    <name>Fahad ansari</name>
    <department>CS</department>
    <salary>90000</salary>
</Employee>')
WHERE id = 102;
```

```
SQL> UPDATE employee_xml
     SET emp_data = XMLTYPE('<Employee>
         <name>Fahad ansari</name>
         <department>CS</department>...
Show more...


1 row updated.

Elapsed: 00:00:00.006
```

Step 8: Verify the update by selecting the updated row from the employee_xml table.

```
SELECT * FROM employee_xml WHERE id = 102;
```

	ID	EMP_DATA
1	102	<Employee> <name>

Step 9: Delete a specific employee row (e.g., id = 102) from the employee_xml table.

```
DELETE FROM employee_xml WHERE id = 101;
```

```
SQL> DELETE FROM employee_xml WHERE id = 101


1 row deleted.

Elapsed: 00:00:00.007
```

	ID	EMP_DATA
1	102	<Employee> <name>
2	103	<Employee> <name>
3	104	<Employee> <name>
4	105	<Employee> <name>

Step 10: Convert the XML data into relational columns using XMLTABLE and display it as a normal table with id, name, department, and salary.

```
SELECT e.id,
       x.name,
       x.department,
       x.salary
FROM employee_xml e,
     XMLTABLE(
       '/Employee'
       PASSING e.emp_data
       COLUMNS
         name       VARCHAR2(40) PATH 'name',
         department VARCHAR2(50) PATH 'department',
         salary     NUMBER       PATH 'salary'
     ) x;
```

	ID	NAME	DEPARTMENT	SALARY
1	102	Fahad ansari	CS	90000
2	103	afzal shaikh	MSC-CS	70000
3	104	Akshay	BMS	40000
4	105	shariq	MSC IT	80000

Exercise

Q1: Tasks

1. **Create a table library_xml with:**
   - **id (Primary Key)**
   - **book_data (XMLTYPE)**
2. **Insert at least 4 books into the table. Example data:**
   - ***The Alchemist*, Paulo Coelho, Fiction, 300, 1988**
   - ***Clean Code*, Robert C. Martin, Programming, 550, 2008**
   - ***Rich Dad Poor Dad*, Robert Kiyosaki, Finance, 400, 1997**
   - ***Atomic Habits*, James Clear, Self-help, 500, 2018**
3. **Write a query to display all books with Title, Author, Genre, Price, Year using XMLTABLE.**
4. **Retrieve only those books where Price > 400.**
5. **Update the Price of *The Alchemist* to 350.**
6. **Delete the record of the book *Rich Dad Poor Dad*.**
7. **Use XMLTABLE with WHERE condition to fetch all *Programming* books.**

Solution:

1. **Create a table library_xml with two columns: id (Primary Key) and book_data (XMLTYPE).**

```
CREATE TABLE library_xml (
  id       NUMBER PRIMARY KEY,
  book_data XMLTYPE
);
```

```
SQL> CREATE TABLE library_xml (
       id       NUMBER PRIMARY KEY,
       book_data XMLTYPE
     )


Table LIBRARY_XML created.

Elapsed: 00:00:00.022
```

2. **Insert at least four book records in XML format into the book_data column.**

```
INSERT INTO library_xml (id, book_data) VALUES (
  1,
  XMLTYPE('<?xml version="1.0"?>
           <Book>
             <Title>The Alchemist</Title>
             <Author>Paulo Coelho</Author>
             <Genre>Fiction</Genre>
             <Price>300</Price>
             <Year>1988</Year>
           </Book>')
);
```

```
SQL> INSERT INTO library_xml (id, book_data) VALUES (
       1,
       XMLTYPE('<?xml version="1.0"?>
                <Book>...
Show more...
```

```
1 row inserted.

Elapsed: 00:00:00.005
```

```
INSERT INTO library_xml (id, book_data) VALUES (
  2,
  XMLTYPE('<?xml version="1.0"?>
           <Book>
             <Title>Clean Code</Title>
             <Author>Robert C. Martin</Author>
             <Genre>Programming</Genre>
             <Price>550</Price>
             <Year>2008</Year>
           </Book>')
);
```

```
SQL> INSERT INTO library_xml (id, book_data) VALUES (
       2,
       XMLTYPE('<?xml version="1.0"?>
                <Book>...
Show more...
```

```
1 row inserted.

Elapsed: 00:00:00.162
```

```
INSERT INTO library_xml (id, book_data) VALUES (
  3,
  XMLTYPE('<?xml version="1.0"?>
           <Book>
             <Title>Rich Dad Poor Dad</Title>
             <Author>Robert Kiyosaki</Author>
             <Genre>Finance</Genre>
             <Price>400</Price>
             <Year>1997</Year>
           </Book>')
);
```

```
SQL> INSERT INTO library_xml (id, book_data) VALUES (
       3,
       XMLTYPE('<?xml version="1.0"?>
                <Book>...
Show more...
```

```
1 row inserted.

Elapsed: 00:00:00.004
```

```
INSERT INTO library_xml (id, book_data) VALUES (
  4,
  XMLTYPE('<?xml version="1.0"?>
          <Book>
            <Title>Atomic Habits</Title>
            <Author>James Clear</Author>
            <Genre>Self-help</Genre>
            <Price>500</Price>
            <Year>2018</Year>
          </Book>')
);
```

```
SQL> INSERT INTO library_xml (id, book_data) VALUES (
       4,
       XMLTYPE('<?xml version="1.0"?>
                <Book>...
Show more...


1 row inserted.

Elapsed: 00:00:00.004
```

3. Write a query using XMLTABLE to display all book details (Title, Author, Genre, Price, Year).

```
SELECT x.title,
       x.author,
       x.genre,
       x.price,
       x.year
FROM library_xml l,
     XMLTABLE('/Book'
       PASSING l.book_data
       COLUMNS
         Title  VARCHAR2(100) PATH 'Title',
         Author VARCHAR2(100) PATH 'Author',
         Genre  VARCHAR2(50)  PATH 'Genre',
         Price  NUMBER        PATH 'Price',
         Year   NUMBER        PATH 'Year'
     ) x;
```

	TITLE	AUTHOR	GENRE	PRICE	YEAR
1	The Alchemist	Paulo Coelho	Fiction	300	1988
2	Clean Code	Robert C. Martin	Programming	550	2008
3	Rich Dad Poor Dad	Robert Kiyosaki	Finance	400	1997
4	Atomic Habits	James Clear	Self-help	500	2018

4. **Retrieve only those books where the Price is greater than 400.**

```
SELECT x.title, x.author, x.genre, x.price, x.year
FROM library_xml l,
     XMLTABLE('/Book'
       PASSING l.book_data
       COLUMNS
         Title  VARCHAR2(100) PATH 'Title',
         Author VARCHAR2(100) PATH 'Author',
         Genre  VARCHAR2(50)  PATH 'Genre',
         Price  NUMBER        PATH 'Price',
         Year   NUMBER        PATH 'Year'
     ) x
WHERE x.price > 400;
```

	TITLE	AUTHOR	GENRE	PRICE	YEAR
1	Clean Code	Robert C. Martin	Programming	550	2008
2	Atomic Habits	James Clear	Self-help	500	2018

5. **Update the price of the book “The Alchemist” to 350.**

```
UPDATE library_xml
SET book_data = updateXML(book_data, '/Book/Price/text()', '350')
WHERE XMLExists('/Book[Title="The Alchemist"]' PASSING book_data);
```

```
SQL> UPDATE library_xml
     SET book_data = updateXML(book_data, '/Book/Price/text()', '350')
     WHERE XMLExists('/Book[Title="The Alchemist"]' PASSING book_data)


1 row updated.

Elapsed: 00:00:00.021
```

6. **Delete the record of the book “Rich Dad Poor Dad” from the table.**

```
DELETE FROM library_xml
WHERE XMLExists('/Book[Title="Rich Dad Poor Dad"]' PASSING book_data);

COMMIT;
```

```
SQL> DELETE FROM library_xml
     WHERE XMLExists('/Book[Title="Rich Dad Poor Dad"]' PASSING book_data)


1 row deleted.

Elapsed: 00:00:00.128
```

7. Use XMLTABLE with a WHERE condition to display only those books that belong to the Programming genre.

```
SELECT x.title, x.author, x.genre, x.price, x.year
FROM library_xml l,
     XMLTABLE('/Book'
       PASSING l.book_data
       COLUMNS
         Title  VARCHAR2(100) PATH 'Title',
         Author VARCHAR2(100) PATH 'Author',
         Genre  VARCHAR2(50)  PATH 'Genre',
         Price  NUMBER        PATH 'Price',
         Year   NUMBER        PATH 'Year'
     ) x
WHERE x.genre = 'Programming';
```

	TITLE	AUTHOR	GENRE	PRICE	YEAR
1	Clean Code	Robert C. Martin	Programming	550	2008

Practical No: 5

Aim : To install and configure PostgreSQL with PostGIS extension and perform spatial database operations such as creating spatial tables, inserting records with geographic data, and running spatial queries.

Question:
"Write the steps to install PostgreSQL with PostGIS extension and demonstrate the use of spatial queries by creating a table of cities. Insert at least four cities with their latitude and longitude values, and perform the following operations:

1. Display all the cities.
2. Find the distance between Mumbai and Delhi.
3. List all cities within 500 km of Mumbai.
4. Find the nearest city to Delhi."

Step1 : Install Postgresql
https://www.postgresql.org/download/windows/
Step2: During installation , select StackBuilder -> It allows to install postGIS extension.
Step3: after installat open pgAdmin4 application
Start menu -> pgAdmin4
Step4 : Connect to server
In the left Expand servers -> PostgreSQL -> Enter the password which you set during installation
Step5 : Use StackBuilder to install PostGIS

1) Open StackBuilder
2) Select your PostgreSQL version
3) In the categories list expand “Spatial Extensions”
4) Select PostGIS
5) Click Next-> Download->Install

Step6: Create database

1) Open pGAdmin4 -> Connect to server
2) In the left expand Servers -> PostgreSQL ->Database
3) Right click on database ->create ->database

Fill database name(spatial_db), no change for other option then click on save.
Step7: Enable PostGIS in your database

1) Open pgAdmin4
2) Connect to server

3) Right click on your database(spatial_db)
4) Select Query Tool

A new tab will open with white editor panel where we will type sql command.

Code :

1)

Enable PostGIS Extension

→ This activates PostGIS inside your PostgreSQL database, so it can handle spatial (geographic) data types and functions

```
-- Enable PostGIS Extension
CREATE EXTENSION postgis;
```

```
CREATE EXTENSION

Query returned successfully in 1 secs 65 msec.
```

2)

Verify Installation

→ Running the version check ensures PostGIS is installed correctly and shows the version being used

```
-- Verify installation
SELECT PostGIS_full_version();
```

	postgis_full_version text
1	POSTGIS="3.5.3 3.5.3" [EXTENSION] PGSQL="170" GEOS="3.13.1-CAPI-1.19.2" PROJ=

3)

Create a Table for Storing Cities

→ A new table (`cities`) is created with columns for ID, name, population, and a spatial column (`location`) to store latitude/longitude points.

To Check the table Go to Database/Spartial_db/schema/table

```
-- Create a table for storing cities
CREATE TABLE cities (
    id SERIAL PRIMARY KEY,
    name VARCHAR(100),
    population INTEGER,
    location GEOGRAPHY(Point, 4326) --4326 is Deafult Port Number
);
```

```
CREATE TABLE

Query returned successfully in 75 msec.
```

4)

Insert

Records with Spatial Data

→ Cities are inserted with their names, populations, and geographic coordinates (longitude, latitude)

```
-- Insert records with spatial (latitude/longitude) data
INSERT INTO cities (name, population, location)
VALUES
    ('Mumbai', 20000000, ST_GeogFromText('SRID=4326;POINT(72.8777 19.0760)')),
    ('Delhi', 19000000, ST_GeogFromText('SRID=4326;POINT(77.1025 28.7041)')),
    ('Bangalore', 10000000, ST_GeogFromText('SRID=4326;POINT(77.5946 12.9716)')),
    ('Chennai', 9000000,  ST_GeogFromText('SRID=4326;POINT(80.2707 13.0827)')),
    ('Bhopal', 8000000,  ST_GeogFromText('SRID=4326;POINT(23.1996395 77.241076)')),
    ('Pune', 7000000,  ST_GeogFromText('SRID=4326;POINT(18.5248706 73.6981514)'));
```

```
INSERT 0 6

Query returned successfully in 55 msec.
```

5)

View All Cities

→ Displays all the data stored in the `cities` table for verification.

```
select * from cities;
```

	id [PK] integer	name character varying (100)	population integer	location geography
1	1	Mumbai	20000000	0101000020E6100000C0EC9E3C2C385240FA7E6ABC74133340
2	2	Delhi	19000000	0101000020E6100000C3F5285C8F465340151DC9E53FB43C40
3	3	Bangalore	10000000	0101000020E6100000E78C28ED0D6653405396218E75F12940
4	4	Chennai	9000000	0101000020E6100000BEC117265311544027C286A7572A2A40
5	5	Bhopal	8000000	0101000020E6100000637D03931B3337406FF607CA6D4F5340
6	6	Pune	7000000	0101000020E6100000C4A16DEB5D86324006AA3583AE6C5240

6)

Distance between Mumbai and Delhi

→ Calculates the straight-line (geodesic) distance between Mumbai and Delhi in kilometers.

```sql
-- Distance between Mumbai and Delhi (in kilometers)
SELECT ST_Distance(
    (SELECT location FROM cities WHERE name='Mumbai'),
    (SELECT location FROM cities WHERE name='Delhi')
) / 1000 AS distance_km; --Defacult in Meter Convert into KM
```

	distance_km double precision
1	1149.60838771384

7)

Cities within 500 km of Mumbai

→ Finds all cities stored in the table that are located within 500 km of Mumbai.

```sql
-- Cities within 500 km of Mumbai
SELECT name, population
FROM cities
WHERE ST_DWithin(
    location,
    ST_GeogFromText('SRID=4326;POINT(72.8777 19.0760)'), --From Mumbai
    500000   -- 500 km in meters
);
```

	name character varying (100)	population integer
1	Mumbai	20000000

8)

Nearest City to Delhi

→ Searches the table and identifies which city is geographically closest to Delhi.

```sql
-- Nearest city to Delhi
SELECT name, population
FROM cities
WHERE name <> 'Delhi'
ORDER BY location <-> ST_GeogFromText('SRID=4326;POINT(77.1025 28.7041)')
LIMIT 1;
```

	name character varying (100)	population integer
1	Mumbai	20000000

PRACTICAL NO: 6

Aim: To create and manage a **Temporal Table** in SQL that stores **employee salary history** and allows tracking of data changes **over time** using date fields (valid_from and valid_to).

Temporal data means **data that changes over time** — in other words, it's data that keeps a **history of values** for the same entity.

It not only stores the **current value**, but also **when** that value was valid (the time period).

Types of Temporal Data

1. **Valid-time data** – When the data was true in the real world.
2. **Transaction-time data** – When the data was stored or changed in the database.
3. **Bi-temporal data** – Combines both valid-time and transaction-time.

Question:

1. Create a database named temporal_db and use it.
2. Create a table named Employee_salary_history with the following structure: emp_id (INT)
    - emp_name (VARCHAR(30))
    - salary (FLOAT)
    - valid_from (DATE)
    - valid_to (DATE)
    - Make (emp_id, valid_from) as the **primary key**.
3. Insert the following records into the table:
4. Display all records from the table Employee_salary_history.
5. Display all employees with their **current salary**
6. Find the salary of all employees **on 15-08-2023**.
7. Display all employees whose salary record started in **the year 2024**.
8. Find all employees who have **more than 2 salary revisions**.

Solution:

1. Create a database named temporal_db and use it.

Code:

```
create database temporal_db;
use temporal_db;
```

```
MariaDB [(none)]> create database temporal_db;
Query OK, 1 row affected (0.003 sec)
```

2. Create a table named Employee_salary_history with the following structure:

emp_id (INT)

- emp_name (VARCHAR(30))
- salary (FLOAT)
- valid_from (DATE)
- valid_to (DATE)
- Make (emp_id, valid_from) as the **primary key**.

Code:

```
create table Employee_salary_history (
  emp_id int,
  emp_name varchar(30),
  salary float,
  valid_from date,
  valid_to date,
  primary key (emp_id, valid_from)
);
```

```
MariaDB [(none)]> use temporal_db
Database changed
MariaDB [temporal_db]> create table Employee_salary_history (
    ->   emp_id int,
    ->   emp_name varchar(30),
    ->   salary float,
    ->   valid_from date,
    ->   valid_to date,
    ->   primary key (emp_id, valid_from)
    -> );
Query OK, 0 rows affected (0.005 sec)
```

3. Insert the following records into the table:

Code:

```
insert into Employee_salary_history values
(1, 'Amit', 30000, '2023-01-01', '2023-06-30'),
(1, 'Amit', 35000, '2023-07-01', '2023-12-31'),
(1, 'Amit', 40000, '2024-01-01', NULL),
(2, 'Afzal', 35000, '2023-01-01', '2023-06-30'),
(2, 'Afzal', 45000, '2023-07-01', '2023-12-31'),
(3, 'Zamaan', 25000, '2023-01-01', '2023-12-31'),
(3, 'Zamaan', 30000, '2024-01-01', NULL);
```

```
MariaDB [temporal_db]> insert into Employee_salary_history values
    -> (1, 'Amit', 30000, '2023-01-01', '2023-06-30'),
    -> (1, 'Amit', 35000, '2023-07-01', '2023-12-31'),
    -> (1, 'Amit', 40000, '2024-01-01', NULL),
    -> (2, 'Afzal', 35000, '2023-01-01', '2023-06-30'),
    -> (2, 'Afzal', 45000, '2023-07-01', '2023-12-31'),
    -> (3, 'Zamaan', 25000, '2023-01-01', '2023-12-31'),
    -> (3, 'Zamaan', 30000, '2024-01-01', NULL);
Query OK, 7 rows affected (0.029 sec)
Records: 7  Duplicates: 0  Warnings: 0
```

4. Display all records from the table Employee_salary_history

Code:

```
Select * From Employee_Salary_history
```

```
MariaDB [temporal_db]> select * from Employee_salary_history;
+--------+----------+--------+------------+------------+
| emp_id | emp_name | salary | valid_from | valid_to   |
+--------+----------+--------+------------+------------+
|      1 | Amit     |  30000 | 2023-01-01 | 2023-06-30 |
|      1 | Amit     |  35000 | 2023-07-01 | 2023-12-31 |
|      1 | Amit     |  40000 | 2024-01-01 | NULL       |
|      2 | Afzal    |  35000 | 2023-01-01 | 2023-06-30 |
|      2 | Afzal    |  45000 | 2023-07-01 | 2023-12-31 |
|      3 | Zamaan   |  25000 | 2023-01-01 | 2023-12-31 |
|      3 | Zamaan   |  30000 | 2024-01-01 | NULL       |
+--------+----------+--------+------------+------------+
7 rows in set (0.001 sec)
```

5. Display all employees with their **current salary**

Code:

```
select emp_id, emp_name, salary
from Employee_salary_history
 where valid_to is null or valid_to >= curdate();
```

```
MariaDB [temporal_db]> select emp_id, emp_name, salary
    -> from Employee_salary_history
    -> where valid_to is null or valid_to >= curdate();
+--------+----------+--------+
| emp_id | emp_name | salary |
+--------+----------+--------+
|      1 | Amit     |  40000 |
|      3 | Zamaan   |  30000 |
+--------+----------+--------+
2 rows in set (0.001 sec)
```

6. Find the salary of all employees **on 15-08-2023**

Code:

```
select emp_id, emp_name, salary
from Employee_salary_history
where '2023-08-15' between valid_from and ifnull(valid_to, '9999-12-31');
```

```
MariaDB [temporal_db]> select emp_id, emp_name, salary
    -> from Employee_salary_history
    -> where '2023-08-15' between valid_from and ifnull(valid_to, '9999-12-31');
+--------+----------+--------+
| emp_id | emp_name | salary |
+--------+----------+--------+
|      1 | Amit     |  35000 |
|      2 | Afzal    |  45000 |
|      3 | Zamaan   |  25000 |
+--------+----------+--------+
3 rows in set (0.001 sec)
```

7. Display all employees whose salary record started in **the year 2024**

Code:

```
select emp_id, emp_name, salary
from Employee_salary_history
where year(valid_from) = 2024;
```

```
MariaDB [temporal_db]> select emp_id, emp_name, salary
    -> from Employee_salary_history
    -> where year(valid_from) = 2024;
+--------+----------+--------+
| emp_id | emp_name | salary |
+--------+----------+--------+
|      1 | Amit     |  40000 |
|      3 | Zamaan   |  30000 |
+--------+----------+--------+
2 rows in set (0.003 sec)
```

8. Find all employees who have **more than 2 salary revisions**.

Code:

```
select emp_id, emp_name, count(*) as revisions
from Employee_salary_history
group by emp_id, emp_name
having count(*) > 2;
```

```
MariaDB [temporal_db]> select emp_id, emp_name, count(*) as revisions
    -> from Employee_salary_history
    -> group by emp_id, emp_name
    -> having count(*) > 2;
+--------+----------+-----------+
| emp_id | emp_name | revisions |
+--------+----------+-----------+
|      1 | Amit     |         3 |
+--------+----------+-----------+
1 row in set (0.001 sec)
```

Exercise 1

Aim: A university wants to maintain a temporal record of course fee structure for each program. The fee may change from time to time, and the database should keep track of all historical fee changes.

Question:

Task 1: Create Database and Table

Task 2: Insert Sample Data

Insert records as follows:

- BSc Computer Science fee was ₹30,000 from 2022-01-01 to 2022-12-31.
- Later, it was revised to ₹35,000 from 2023-01-01 to 2023-12-31.
- Current fee is ₹40,000 from 2024-01-01 onward.
- MSc Computer Science fee was ₹50,000 from 2023-06-01 and is still valid.

Task 3: Queries:

Write SQL queries for the following:

1. **Find the current fee of all courses.**
2. **List the complete fee history of the course BSc Computer Science.**
3. **Find the fee of all courses on 2023-07-15.**
4. **Show all courses whose fee increased in 2024.**
5. **Find courses that have undergone more than one fee revision.**
6. **Show the highest historical fee among all courses.**
7. **Display the average course fee valid in 2023.**

Solution:

Task 1: Create Database and Table:

Code:

```
CREATE DATABASE university_temporal;
USE university_temporal;
CREATE TABLE Course_Fee_History (
   Course_ID INT,
   Course_Name VARCHAR(100),
   Fee DECIMAL(10,2),
   Valid_From DATE,
   Valid_To DATE,
   PRIMARY KEY (Course_ID, Valid_From)
);
```

```
C:\xampp\mysql\bin>mysql -u root -p
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 10
Server version: 10.4.32-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2018, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> CREATE DATABASE university_temporal;
Query OK, 1 row affected (0.001 sec)

MariaDB [(none)]> USE university_temporal;
Database changed
MariaDB [university_temporal]> CREATE TABLE Course_Fee_History (
    ->     Course_ID INT,
    ->     Course_Name VARCHAR(100),
    ->     Fee DECIMAL(10,2),
    ->     Valid_From DATE,
    ->     Valid_To DATE,
    ->     PRIMARY KEY (Course_ID, Valid_From)
    -> );
Query OK, 0 rows affected (0.016 sec)
```

Task 2: Insert Sample Data

Insert records as follows:

- BSc Computer Science fee was ₹30,000 from 2022-01-01 to 2022-12-31.
- Later, it was revised to ₹35,000 from 2023-01-01 to 2023-12-31.
- Current fee is ₹40,000 from 2024-01-01 onward.

MSc Computer Science fee was ₹50,000 from 2023-06-01 and is still valid

Code:

```
INSERT INTO Course_Fee_History VALUES
(101, 'BSc Computer Science', 30000, '2022-01-01', '2022-12-31'),
(101, 'BSc Computer Science', 35000, '2023-01-01', '2023-12-31'),
(101, 'BSc Computer Science', 40000, '2024-01-01', NULL),
(102, 'MSc Computer Science', 50000, '2023-06-01', NULL);
```

```
MariaDB [university_temporal]> INSERT INTO Course_Fee_History VALUES
    -> (101, 'BSc Computer Science', 30000, '2022-01-01', '2022-12-31'),
    -> (101, 'BSc Computer Science', 35000, '2023-01-01', '2023-12-31'),
    -> (101, 'BSc Computer Science', 40000, '2024-01-01', NULL),
    -> (102, 'MSc Computer Science', 50000, '2023-06-01', NULL);
Query OK, 4 rows affected (0.010 sec)
Records: 4  Duplicates: 0  Warnings: 0
```

Task 3: Queries

1) Find the current fee of all courses

Code:

```
SELECT Course_Name, Fee AS Current_Fee
FROM Course_Fee_History
WHERE Valid_To IS NULL;
```

```
MariaDB [university_temporal]> SELECT Course_Name, Fee AS Current_Fee
    -> FROM Course_Fee_History
    -> WHERE Valid_To IS NULL;
+----------------------+-------------+
| Course_Name          | Current_Fee |
+----------------------+-------------+
| BSc Computer Science |    40000.00 |
| MSc Computer Science |    50000.00 |
+----------------------+-------------+
2 rows in set (0.001 sec)
```

2) List the complete fee history of the course BSc Computer Science

Code:

```
SELECT Course_Name, Fee, Valid_From, Valid_To

FROM Course_Fee_History

WHERE Course_Name = 'BSc Computer Science'

ORDER BY Valid_From;
```

```
MariaDB [university_temporal]> SELECT Course_Name, Fee, Valid_From, Valid_To
    -> FROM Course_Fee_History
    -> WHERE Course_Name = 'BSc Computer Science'
    -> ORDER BY Valid_From;
+---------------------+----------+------------+------------+
| Course_Name         | Fee      | Valid_From | Valid_To   |
+---------------------+----------+------------+------------+
| BSc Computer Science | 30000.00 | 2022-01-01 | 2022-12-31 |
| BSc Computer Science | 35000.00 | 2023-01-01 | 2023-12-31 |
| BSc Computer Science | 40000.00 | 2024-01-01 | NULL       |
+---------------------+----------+------------+------------+
3 rows in set (0.008 sec)
```

3) Find the fee of all courses on 2023-07-15.

SELECT Course_Name, Fee FROM Course_Fee_History
WHERE '2023-07-15' BETWEEN Valid_From AND IFNULL(Valid_To, '9999-12-31');

```
MariaDB [university_temporal]> SELECT Course_Name, Fee
    -> FROM Course_Fee_History
    -> WHERE '2023-07-15' BETWEEN Valid_From AND IFNULL(Valid_To, '9999-12-31');
+---------------------+----------+
| Course_Name         | Fee      |
+---------------------+----------+
| BSc Computer Science | 35000.00 |
| MSc Computer Science | 50000.00 |
+---------------------+----------+
2 rows in set (0.002 sec)
```

4) Show all courses whose fee increased in 2024.

SELECT Course_ID, Course_Name, Fee, Valid_From
FROM Course_Fee_History
WHERE YEAR(Valid_From) = 2024;

```
MariaDB [university_temporal]> SELECT Course_ID, Course_Name, Fee, Valid_From
    -> FROM Course_Fee_History
    -> WHERE YEAR(Valid_From) = 2024;
+-----------+---------------------+----------+------------+
| Course_ID | Course_Name         | Fee      | Valid_From |
+-----------+---------------------+----------+------------+
|       101 | BSc Computer Science | 40000.00 | 2024-01-01 |
+-----------+---------------------+----------+------------+
1 row in set (0.000 sec)
```

5) Find courses that have undergone more than one fee revision.

Code:

SELECT Course_ID ,Course_Name, COUNT(*) AS Revision_Count
FROM Course_Fee_History
GROUP BY Course_ID ,Course_Name
HAVING COUNT(*) > 1;

```
MariaDB [university_temporal]> SELECT Course_ID ,Course_Name, COUNT(*) AS Revision_Count
    -> FROM Course_Fee_History
    -> GROUP BY Course_ID ,Course_Name
    -> HAVING COUNT(*) > 1;
+-----------+---------------------+----------------+
| Course_ID | Course_Name         | Revision_Count |
+-----------+---------------------+----------------+
|       101 | BSc Computer Science |              3 |
+-----------+---------------------+----------------+
1 row in set (0.001 sec)
```

6) Show the highest historical fee among all courses.

Code:

```
SELECT Course_ID, Course_Name, MAX(Fee) AS Highest_Fee
FROM Course_Fee_History
GROUP BY Course_Name;
```

```
MariaDB [university_temporal]> SELECT Course_ID, Course_Name, MAX(Fee) AS Highest_Fee
    -> FROM Course_Fee_History
    -> GROUP BY Course_Name;
+-----------+---------------------+-------------+
| Course_ID | Course_Name         | Highest_Fee |
+-----------+---------------------+-------------+
|       101 | BSc Computer Science |    40000.00 |
|       102 | MSc Computer Science |    50000.00 |
+-----------+---------------------+-------------+
2 rows in set (0.002 sec)
```

7) Display the average course fee valid in 2023.

Code:

```
SELECT Course_ID, Course_Name, AVG(Fee) AS Avg_Fee_2023
FROM Course_Fee_History
WHERE '2023-06-30' BETWEEN Valid_From AND IFNULL(Valid_To, '9999-12-31');
```

```
MariaDB [university_temporal]> SELECT Course_ID, Course_Name, AVG(Fee) AS Avg_Fee_2023
    -> FROM Course_Fee_History
    -> WHERE '2023-06-30' BETWEEN Valid_From AND IFNULL(Valid_To, '9999-12-31');
+-----------+---------------------+--------------+
| Course_ID | Course_Name         | Avg_Fee_2023 |
+-----------+---------------------+--------------+
|       101 | BSc Computer Science | 42500.000000 |
+-----------+---------------------+--------------+
1 row in set (0.002 sec)
```

Practical No: 7

Aim: Demonstrate the accessing, storing and performing CRUD operations in MongoDB.

---

1. What is MongoDB?

MongoDB is a NoSQL database that stores data in BSON (Binary JSON) format.
It's schema-less, document-oriented, and supports high scalability and fast querying.
Each record is called a document, and documents are grouped into collections.

⚙️ 2. Download & Installation Steps

🔹 Step 1: Download MongoDB

Visit the official MongoDB website:

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

Choose:

- Edition: Community Server
- Version: Latest stable release
- Platform: Windows
- Package: .msi (Installer)

🔹 Step 2: Install MongoDB

1. Run the downloaded .msi installer.
2. Select Complete installation.
3. Enable "Install MongoDB as a Service" option.
4. Finish setup and close the wizard.

🔹 Step 3: Add MongoDB to System PATH (if not automatic)

1. Go to
   C:\Program Files\MongoDB\Server\<version>\bin
2. Copy the path.
3. Add it to Environment Variables → PATH.

Open Command Prompt and type:

```
mongo --version
```

4. If you see version details, installation is successful ✅

🔹 Step 4: Start MongoDB Service

Open Command Prompt and type:

net start MongoDB

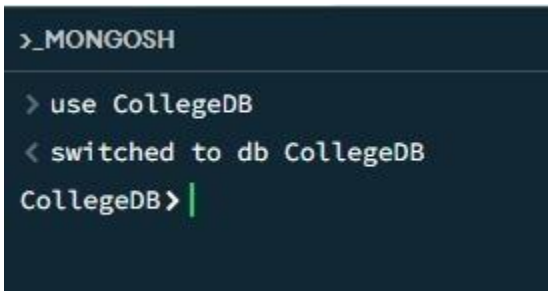
To stop MongoDB:

net stop MongoDB

1) Create / Access Database

Use or create a database named collegeDB:

```
use collegeDB
```



2) Create and Insert Documents (CREATE Operation)

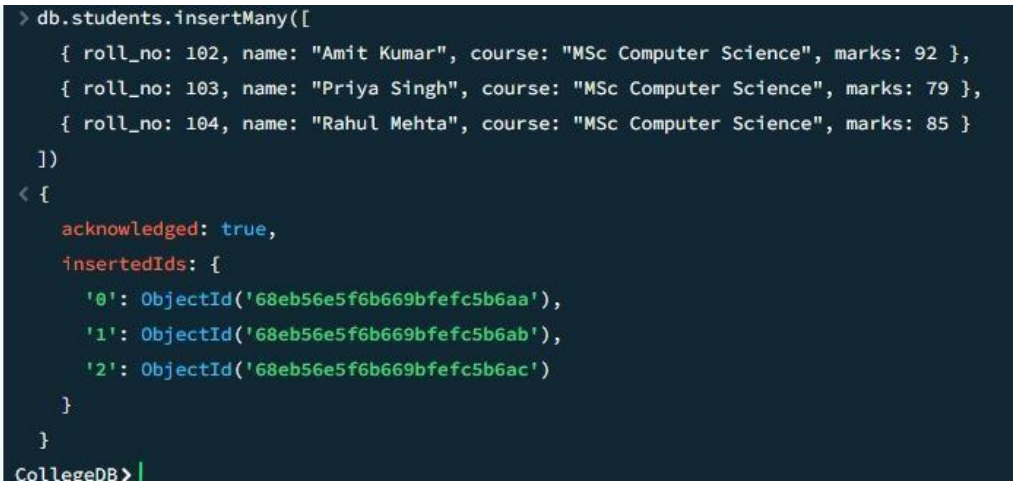
Create a collection named students and insert documents.

➤ Insert One Record

```
db.students.insertOne({
  roll_no: 101,
  name: "Riya Sharma",
  course: "MSc Computer
  Science", marks: 88
})
```

3) Insert Multiple Records

```
db.students.insertMany([
  { roll_no: 102, name: "Amit Kumar", course: "MSc Computer Science", marks: 92 },
  { roll_no: 103, name: "Priya Singh", course: "MSc Computer Science", marks: 79 },
  { roll_no: 104, name: "Rahul Mehta", course: "MSc Computer Science", marks: 85 }
])
```



---

4) Retrieve Documents (READ Operation)

Display all records:

```
db.students.find()
```

5) Display neatly formatted records:

db.students.find().pretty()

```
> db.students.find().pretty()
< {
    _id: ObjectId('68eb56d6f6b669bfefc5b6a9'),
    roll_no: 101,
    name: 'Riya Sharma',
    course: 'MSc Computer Science',
    marks: 88
  }
  {
    _id: ObjectId('68eb56e5f6b669bfefc5b6aa'),
    roll_no: 102,
    name: 'Amit Kumar',
    course: 'MSc Computer Science',
    marks: 92
  }
  {
    _id: ObjectId('68eb56e5f6b669bfefc5b6ab'),
    roll_no: 103,
    name: 'Priya Singh',
    course: 'MSc Computer Science',
    marks: 79
  }
  {
    _id: ObjectId('68eb56e5f6b669bfefc5b6ac'),
    roll_no: 104,
    name: 'Rahul Mehta',
    course: 'MSc Computer Science',
    marks: 85
  }
```

6) Find students with marks greater than 85:

db.students.find({ marks: { $gt: 85 } })

```
> db.students.find({ marks: { $gt: 85 } })
< {
    _id: ObjectId('68eb56d6f6b669bfefc5b6a9'),
    roll_no: 101,
    name: 'Riya Sharma',
    course: 'MSc Computer Science',
    marks: 88
  }
  {
    _id: ObjectId('68eb56e5f6b669bfefc5b6aa'),
    roll_no: 102,
    name: 'Amit Kumar',
    course: 'MSc Computer Science',
    marks: 92
  }
```

7) Find one record:

```
db.students.findOne({ roll_no: 102 })
```

```
> db.students.findOne({ roll_no: 102 })
< {
    _id: ObjectId('68eb56e5f6b669bfefc5b6aa'),
    roll_no: 102,
    name: 'Amit Kumar',
    course: 'MSc Computer Science',
    marks: 92
  }
```

---

8) Update Documents (UPDATE Operation)

Update a specific student's marks:

```
db.students.updateOne(
  { roll_no: 103 },
  { $set: { marks: 84 } }
)
```

```
> db.students.updateOne(
    { roll_no: 103 },
    { $set: { marks: 84 } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

9) Update course name for multiple records:

```
db.students.updateMany(
  { course: "MSc Computer Science" },
  { $set: { course: "MSc CS" } }
)
```

```
> db.students.updateMany(
    { course: "MSc Computer Science" },
    { $set: { course: "MSc CS" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 4,
    modifiedCount: 4,
    upsertedCount: 0
  }
```

10) Delete Documents (DELETE Operation)

Delete a single record:

db.students.deleteOne({ roll_no: 104 })

```
> db.students.deleteOne({ roll_no: 104 })
< {
    acknowledged: true,
    deletedCount: 1
  }
```

Delete all records where marks < 85:

db.students.deleteMany({ marks: { $lt: 85 } })

```
> db.students.deleteMany({ marks: { $lt: 85 } })
< {
    acknowledged: true,
    deletedCount: 1
  }
```

Exercise Question

1. Create a database studentDB and collection students.
   Insert at least **6 documents** with the following fields:
   - roll_no, name, department, semester, marks.
2. Write a query to **find all students** from the Computer Science department.
3. Display all students who scored **marks between 60 and 90**.
4. Update the semester of student roll_no = 102 from 3 to 4.
5. Delete all students whose marks < 50.
6. Add a new field grade to all documents where:
   - marks ≥ 85 → grade = "A"
   - marks between 70–84 → grade = "B"
   - marks < 70 → grade = "C"
7. Sort all students in **descending order of marks**.
8. Count how many students are in each department

1) Create a database studentDB and collection students

Create Database and Collection

use studentDB

Switches to (or creates) a database named **studentDB**.

db.createCollection("students")

```
> use Students
< switched to db Students
> db.createCollection("students")
< { ok: 1 }
```

2) Insert at least 6 documents with the following fields: roll_no, name, department, semester, marks.

Insert Documents

```
db.students.insertMany([
  { roll_no: 101, name: "Riya Sharma", department: "Computer Science", semester: 3, marks: 88 },
  { roll_no: 102, name: "Amit Kumar", department: "Mathematics", semester: 3, marks: 74 },
  { roll_no: 103, name: "Priya Singh", department: "Computer Science", semester: 4, marks: 92 },
  { roll_no: 104, name: "Rahul Mehta", department: "Physics", semester: 2, marks: 63 },
  { roll_no: 105, name: "Sneha Patel", department: "Computer Science", semester: 1, marks: 48 },
  { roll_no: 106, name: "Vikas Yadav", department: "Mathematics", semester: 2, marks: 81 }
])
```

```
> db.students.insertMany([
    { roll_no: 101, name: "Riya Sharma", department: "Computer Science", semester: 3, marks: 88 },
    { roll_no: 102, name: "Amit Kumar", department: "Mathematics", semester: 3, marks: 74 },
    { roll_no: 103, name: "Priya Singh", department: "Computer Science", semester: 4, marks: 92 },
    { roll_no: 104, name: "Rahul Mehta", department: "Physics", semester: 2, marks: 63 },
    { roll_no: 105, name: "Sneha Patel", department: "Computer Science", semester: 1, marks: 48 },
    { roll_no: 106, name: "Vikas Yadav", department: "Mathematics", semester: 2, marks: 81 }
  ])
< {
    acknowledged: true,
    insertedIds: {
      '0': ObjectId('68eb5ae1bc3c13dcd4f74343'),
      '1': ObjectId('68eb5ae1bc3c13dcd4f74344'),
      '2': ObjectId('68eb5ae1bc3c13dcd4f74345'),
      '3': ObjectId('68eb5ae1bc3c13dcd4f74346'),
      '4': ObjectId('68eb5ae1bc3c13dcd4f74347'),
      '5': ObjectId('68eb5ae1bc3c13dcd4f74348')
    }
  }
```

Q1) Write a query to find all students from the Computer Science department
Code:

db.students.find({ department: "Computer Science" })

```
> db.students.find({ department: "Computer Science" })
< {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74343'),
    roll_no: 101,
    name: 'Riya Sharma',
    department: 'Computer Science',
    semester: 3,
    marks: 88
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74345'),
    roll_no: 103,
    name: 'Priya Singh',
    department: 'Computer Science',
    semester: 4,
    marks: 92
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74347'),
    roll_no: 105,
    name: 'Sneha Patel',
    department: 'Computer Science',
    semester: 1,
    marks: 48
  }
```

Q2) Display all students who scored marks between 60 and 90.
Code:

db.students.find({ marks: { $gte: 60, $lte: 90 } })

```
> db.students.find({ marks: { $gte: 60, $lte: 90 } })
< {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74343'),
    roll_no: 101,
    name: 'Riya Sharma',
    department: 'Computer Science',
    semester: 3,
    marks: 88
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74344'),
    roll_no: 102,
    name: 'Amit Kumar',
    department: 'Mathematics',
    semester: 3,
    marks: 74
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74346'),
    roll_no: 104,
    name: 'Rahul Mehta',
    department: 'Physics',
    semester: 2,
    marks: 63
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74348'),
    roll_no: 106,
    name: 'Vikas Yadav',
    department: 'Mathematics',
    semester: 2,
    marks: 81
  }
```

$gte = greater than or equal to**, $lte =** less than or equal to**.**

Q3) Update the semester of student roll_no = 102 from 3 to 4.

Code:

```
db.students.updateOne(
  { roll_no: 102 },
  { $set: { semester: 4 } }
)
```

```
> db.students.updateOne(
    { roll_no: 102 },
    { $set: { semester: 4 } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Q4) Delete all students whose marks < 50.

Code:

```
db.students.deleteMany({ marks: { $lt: 50 } })
```

Removes all students whose marks are below 50.

```
> db.students.deleteMany({ marks: { $lt: 50 } })
< {
    acknowledged: true,
    deletedCount: 1
  }
```

Q5) Add a new field grade to all documents where:

- marks ≥ 85 → grade = "A"
- marks between 70–84 → grade = "B"
- marks < 70 → grade = "C"

```
> // Grade A
  db.students.updateMany(
    { marks: { $gte: 85 } },
    { $set: { grade: "A" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
> // Grade B
  db.students.updateMany(
    { marks: { $gte: 70, $lt: 85 } },
    { $set: { grade: "B" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 2,
    modifiedCount: 2,
    upsertedCount: 0
  }
> // Grade C
  db.students.updateMany(
    { marks: { $lt: 70 } },
    { $set: { grade: "C" } }
  )
< {
    acknowledged: true,
    insertedId: null,
    matchedCount: 1,
    modifiedCount: 1,
    upsertedCount: 0
  }
```

Q6) Sort all students in descending order of marks.

Code:

```
db.students.find().sort({ marks: -1 })
```

```
>_MONGOSH
> db.students.find().sort({ marks: -1 })
< {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74345'),
    roll_no: 103,
    name: 'Priya Singh',
    department: 'Computer Science',
    semester: 4,
    marks: 92,
    grade: 'A'
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74343'),
    roll_no: 101,
    name: 'Riya Sharma',
    department: 'Computer Science',
    semester: 3,
    marks: 88,
    grade: 'A'
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74348'),
    roll_no: 106,
    name: 'Vikas Yadav',
    department: 'Mathematics',
    semester: 2,
    marks: 81,
    grade: 'B'
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74344'),
    roll_no: 102,
    name: 'Amit Kumar',
    department: 'Mathematics',
    semester: 4,
    marks: 74,
    grade: 'B'
  }
```

```
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74346'),
    roll_no: 104,
    name: 'Rahul Mehta',
    department: 'Physics',
    semester: 2,
    marks: 63,
    grade: 'C'
  }
```

Q7) Count how many students are in each department

```
db.students.aggregate([
  { $group: { _id: "$department", total_students: { $sum: 1 } } }
])
```

```
> db.students.aggregate([
    { $group: { _id: "$department", total_students: { $sum: 1 } } }
  ])
< {
    _id: 'Mathematics',
    total_students: 2
  }
  {
    _id: 'Computer Science',
    total_students: 2
  }
  {
    _id: 'Physics',
    total_students: 1
  }
>
```

✅ Final Check

Display all records with grades:

db.students.find().pretty()

```
> db.students.find().pretty()
< {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74343'),
    roll_no: 101,
    name: 'Riya Sharma',
    department: 'Computer Science',
    semester: 3,
    marks: 88,
    grade: 'A'
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74344'),
    roll_no: 102,
    name: 'Amit Kumar',
    department: 'Mathematics',
    semester: 4,
    marks: 74,
    grade: 'B'
  }
  {
    _id: ObjectId('68eb5ae1bc3c13dcd4f74345'),
    roll_no: 103,
    name: 'Priya Singh',
    department: 'Computer Science',
    semester: 4,
    marks: 92,
    grade: 'A'
  }
```