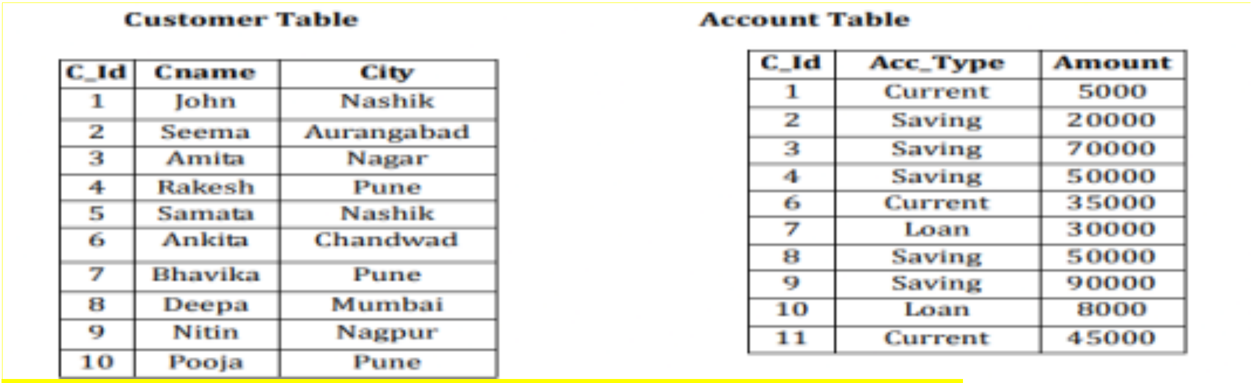
Q1 Use MySQL –(Joins and Subqueries)

Create Customer and Account table and add rows shown below



Customer Table

C_Id	Cname	City
1	John	Nashik
2	Seema	Aurangabad
3	Amita	Nagar
4	Rakesh	Pune
5	Samata	Nashik
6	Ankita	Chandwad
7	Bhavika	Pune
8	Deepa	Mumbai
9	Nitin	Nagpur
10	Pooja	Pune

Account Table

C_Id	Acc_Type	Amount
1	Current	5000
2	Saving	20000
3	Saving	70000
4	Saving	50000
6	Current	35000
7	Loan	30000
8	Saving	50000
9	Saving	90000
10	Loan	8000
11	Current	45000

1. Show the cname, Acc_Type, amount information of customer who is having an saving account. 2. Display the data using Natural, left and right join.

3. Display the information of customers living in the same city as of ‘pooja’.

4. Display the information of account, having less amount than average amount throughout the bank. 5. Display the C_id having maximum amount in account.

6. Display the amount and acc_type of those customers whose amount is the minimum amount of that Acc_type. 7. Display the amount of those accounts whose amount is higher than amount of any saving account amount

```
mysql> select * from Customer
    -> ;
+------+-------+--------+
| C_id | cname | city   |
+------+-------+--------+
|    1 | Pooja | Pune   |
|    2 | Ravi  | Mumbai |
|    3 | Sneha | Pune   |
|    4 | Arjun | Delhi  |
|    5 | Meera | Mumbai |
+------+-------+--------+
5 rows in set (0.001 sec)
```

```
mysql> select * from Account
    -> ;
+------+----------+----------+
| C_id | Acc_type | amount   |
+------+----------+----------+
|    1 | Saving   | 50000.00 |
|    2 | Current  | 30000.00 |
|    3 | Saving   | 45000.00 |
|    4 | Fixed    | 80000.00 |
|    5 | Saving   | 20000.00 |
+------+----------+----------+
5 rows in set (0.001 sec)

mysql> SELECT c.cname, a.Acc_type, a.amount
    -> FROM Customer c
    -> JOIN Account a ON c.C_id = a.C_id
    -> WHERE a.Acc_type = 'Saving';
+-------+----------+----------+
| cname | Acc_type | amount   |
+-------+----------+----------+
| Pooja | Saving   | 50000.00 |
| Sneha | Saving   | 45000.00 |
| Meera | Saving   | 20000.00 |
+-------+----------+----------+
3 rows in set (0.003 sec)

mysql> SELECT *
    -> FROM Customer
    -> NATURAL JOIN Account;
+------+-------+--------+----------+----------+
| C_id | cname | city   | Acc_type | amount   |
+------+-------+--------+----------+----------+
|    1 | Pooja | Pune   | Saving   | 50000.00 |
|    2 | Ravi  | Mumbai | Current  | 30000.00 |
|    3 | Sneha | Pune   | Saving   | 45000.00 |
|    4 | Arjun | Delhi  | Fixed    | 80000.00 |
|    5 | Meera | Mumbai | Saving   | 20000.00 |
+------+-------+--------+----------+----------+
5 rows in set (0.007 sec)
```

```
mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount
    -> FROM Customer c
    -> LEFT JOIN Account a ON c.C_id = a.C_id;
+------+-------+--------+----------+----------+
| C_id | cname | city   | Acc_type | amount   |
+------+-------+--------+----------+----------+
|    1 | Pooja | Pune   | Saving   | 50000.00 |
|    2 | Ravi  | Mumbai | Current  | 30000.00 |
|    3 | Sneha | Pune   | Saving   | 45000.00 |
|    4 | Arjun | Delhi  | Fixed    | 80000.00 |
|    5 | Meera | Mumbai | Saving   | 20000.00 |
+------+-------+--------+----------+----------+
5 rows in set (0.004 sec)

mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount FROM Customer c LEFT JOIN Account using
(C_id);
ERROR 1054 (42S22): Unknown column 'a.Acc_type' in
'field list'
mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount FROM Customer c LEFT JOIN Account using
(c.C_id);
ERROR 1064 (42000): You have an error in your SQL
syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near
'.C_id)' at line 1
mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount FROM Customer c LEFT JOIN Account using
(C_id);
ERROR 1054 (42S22): Unknown column 'a.Acc_type' in
'field list'
mysql> SELECT cname, Acc_type, amount
    -> FROM Customer
    -> JOIN Account USING (C_id)
    -> WHERE Acc_type = 'Saving';
+-------+----------+----------+
| cname | Acc_type | amount   |
+-------+----------+----------+
| Pooja | Saving   | 50000.00 |
| Sneha | Saving   | 45000.00 |
| Meera | Saving   | 20000.00 |
+-------+----------+----------+
3 rows in set (0.003 sec)

mysql> SELECT cname, Acc_type, amount FROM Customer
LEFT JOIN Account USING (C_id) WHERE Acc_type =
'Saving';
+-------+----------+----------+
| cname | Acc_type | amount   |
+-------+----------+----------+
| Pooja | Saving   | 50000.00 |
| Sneha | Saving   | 45000.00 |
| Meera | Saving   | 20000.00 |
+-------+----------+----------+
3 rows in set (0.002 sec)

mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount FROM Customer c LEFT JOIN Account a ON c.C_id
= a.C_id;
+------+-------+--------+----------+----------+
| C_id | cname | city   | Acc_type | amount   |
+------+-------+--------+----------+----------+
|    1 | Pooja | Pune   | Saving   | 50000.00 |
|    2 | Ravi  | Mumbai | Current  | 30000.00 |
|    3 | Sneha | Pune   | Saving   | 45000.00 |
|    4 | Arjun | Delhi  | Fixed    | 80000.00 |
|    5 | Meera | Mumbai | Saving   | 20000.00 |
+------+-------+--------+----------+----------+
5 rows in set (0.003 sec)

mysql> SELECT c.C_id, c.cname, c.city, a.Acc_type,
a.amount
    -> FROM Customer c
    -> RIGHT JOIN Account a ON c.C_id = a.C_id;
+------+-------+--------+----------+----------+
| C_id | cname | city   | Acc_type | amount   |
+------+-------+--------+----------+----------+
|    1 | Pooja | Pune   | Saving   | 50000.00 |
|    2 | Ravi  | Mumbai | Current  | 30000.00 |
|    3 | Sneha | Pune   | Saving   | 45000.00 |
|    4 | Arjun | Delhi  | Fixed    | 80000.00 |
|    5 | Meera | Mumbai | Saving   | 20000.00 |
+------+-------+--------+----------+----------+
5 rows in set (0.003 sec)

mysql>   SELECT city FROM Customer WHERE cname =
'Pooja'
    -> ;
+------+
| city |
+------+
| Pune |
+------+
1 row in set (0.006 sec)

mysql> SELECT *
    -> FROM Customer
    -> WHERE city = (
    ->   SELECT city FROM Customer WHERE cname =
'Pooja'
    -> );
+------+-------+------+
| C_id | cname | city |
+------+-------+------+
|    1 | Pooja | Pune |
|    3 | Sneha | Pune |
+------+-------+------+
2 rows in set (0.013 sec)

mysql>
mysql>   SELECT AVG(amount) FROM Account
    ->
    -> ;
+--------------+
| AVG(amount)  |
+--------------+
| 45000.000000 |
+--------------+
1 row in set (0.005 sec)

mysql> SELECT *
    -> FROM Account
    -> WHERE amount < (
    ->   SELECT AVG(amount) FROM Account
    -> );
+------+----------+----------+
| C_id | Acc_type | amount   |
+------+----------+----------+
|    2 | Current  | 30000.00 |
|    5 | Saving   | 20000.00 |
+------+----------+----------+
2 rows in set (0.006 sec)

mysql> SELECT C_id
    -> FROM Account
    -> WHERE amount = (
    ->   SELECT MAX(amount) FROM Account
    -> );
+------+
| C_id |
+------+
|    4 |
+------+
1 row in set (0.003 sec)

mysql> SELECT Acc_type, amount
    -> FROM Account a1
    -> WHERE amount = (
    ->   SELECT MIN(amount)
    ->   FROM Account a2
    ->   WHERE a1.Acc_type = a2.Acc_type
    -> );
+----------+----------+
| Acc_type | amount   |
+----------+----------+
| Current  | 30000.00 |
| Fixed    | 80000.00 |
| Saving   | 20000.00 |
+----------+----------+
3 rows in set (0.001 sec)

mysql> SELECT *
    -> FROM Account
    -> WHERE amount > ANY (
    ->   SELECT amount
    ->   FROM Account
    ->   WHERE Acc_type = 'Saving'
    -> );
+------+----------+----------+
| C_id | Acc_type | amount   |
+------+----------+----------+
|    1 | Saving   | 50000.00 |
|    2 | Current  | 30000.00 |
|    3 | Saving   | 45000.00 |
|    4 | Fixed    | 80000.00 |
+------+----------+----------+
4 rows in set (0.002 sec)

mysql>
mysql>  SELECT amount
    ->   FROM Account
    ->   WHERE Acc_type = 'Saving'
    -> ;
+----------+
| amount   |
+----------+
| 50000.00 |
| 45000.00 |
| 20000.00 |
+----------+
3 rows in set (0.013 sec)

mysql> SELECT *
```

```
    -> FROM Account
    -> WHERE amount > ANY (
    ->   SELECT amount
    ->   FROM Account
    ->   WHERE Acc_type = 'Saving'
    -> );
+------+----------+----------+
| C_id | Acc_type | amount   |
+------+----------+----------+
|    1 | Saving   | 50000.00 |
|    2 | Current  | 30000.00 |
|    3 | Saving   | 45000.00 |
|    4 | Fixed    | 80000.00 |
+------+----------+----------+
4 rows in set (0.003 sec)

mysql> SELECT * FROM Account WHERE amount > (   SELECT
max(amount)   FROM Account   WHERE Acc_type = 'Saving'
);
+------+----------+----------+
| C_id | Acc_type | amount   |
+------+----------+----------+
|    4 | Fixed    | 80000.00 |
+------+----------+----------+
1 row in set (0.004 sec)
```

2. Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-

Schema:

1. Borrower(Rollin, Name, DateofIssue, NameofBook, Status)

2. Fine(Roll_no,Date,Amt)

☐ Accept roll_no & name of book from user.

☐ Check the number of days (from date of issue), if days are between 15 to 30 then fine amount will be Rs 5per day.

- If no. of days>30, per day fine will be Rs 50 per day & for days less than 30, Rs. 5 per day.
- After submitting the book, status will change from I to R.
- If condition of fine is true, then details will be stored into fine table.

```
-- Set user input variables
SET @roll_no = 101;
SET @book_name = 'Mathematics';

-- Declare variables to hold fetched data
SET @date_issue = NULL;
SET @status = NULL;
SET @days = 0;
SET @fine = 0;

-- Fetch date of issue and status
SELECT DateofIssue, Status
INTO @date_issue, @status
FROM Borrower
WHERE Rollin = @roll_no AND NameofBook = @book_name;

-- Check if book exists
IF @date_issue IS NULL THEN
   SELECT 'No record found for this Roll No and Book.' AS Message;
ELSE
   -- Check if book is already returned
   IF @status = 'R' THEN
      SELECT 'Book is already returned.' AS Message;
   ELSE
      -- Calculate number of days
      SET @days = DATEDIFF(CURDATE(), @date_issue);

      -- Calculate fine
      IF @days > 30 THEN
         SET @fine = (30 * 5) + ((@days - 30) * 50);
```

```
        ELSEIF @days >= 15 THEN
            SET @fine = @days * 5;
        ELSE
            SET @fine = 0;
        END IF;

        -- Insert fine if applicable
        IF @fine > 0 THEN
            INSERT INTO Fine(Roll_no, Date, Amt)
            VALUES (@roll_no, CURDATE(), @fine);
            SELECT CONCAT('Fine of Rs. ', @fine, ' recorded.') AS Message;
        ELSE
            SELECT 'No fine applicable.' AS Message;
        END IF;

        -- Update Borrower status
        UPDATE Borrower
        SET Status = 'R'
        WHERE Rollin = @roll_no AND NameofBook = @book_name;

        SELECT 'Book status updated to Returned.' AS Message;
    END IF;
END IF;
```

```
DELIMITER $$
mysql>
mysql> CREATE PROCEDURE return_book(
    ->     IN p_roll_no INT,
    ->     IN p_book_name VARCHAR(100)
    -> )
    -> BEGIN
    ->     DECLARE v_date_issue DATE;
    ->     DECLARE v_status CHAR(1);
    ->     DECLARE v_days INT;
    ->     DECLARE v_fine DECIMAL(10,2) DEFAULT 0;
    ->     DECLARE done INT DEFAULT 0;
    ->
    ->     -- Handler for no matching book
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;
    ->
    ->     -- Fetch the DateofIssue and Status
    ->     SELECT DateofIssue, Status
    ->     INTO v_date_issue, v_status
    ->     FROM Borrower
    ->     WHERE Rollin = p_roll_no AND NameofBook =
p_book_name;
    ->
    ->     -- Check if record was found
    ->     IF done = 1 THEN
    ->         SELECT 'No record found for this Roll No
and Book.' AS Message;
    ->     ELSE
    ->         -- Check if book is already returned
    ->         IF v_status = 'R' THEN
    ->             SELECT 'Book is already returned.'
AS Message;
    ->         ELSE
    ->             -- Calculate number of days
    ->             SET v_days = DATEDIFF(CURDATE(),
v_date_issue);
    ->
    ->             -- Calculate fine
    ->             IF v_days > 30 THEN
    ->                 SET v_fine = (30 * 5) + ((v_days
- 30) * 50);
    ->             ELSEIF v_days >= 15 THEN
    ->                 SET v_fine = v_days * 5;
    ->             ELSE
    ->                 SET v_fine = 0;
    ->             END IF;
    ->
    ->             -- Insert into Fine table if
applicable
    ->             IF v_fine > 0 THEN
    ->                 INSERT INTO Fine(Roll_no, Date,
Amt)
```

```
    ->                 VALUES (p_roll_no, CURDATE(),
v_fine);
    ->                 SELECT CONCAT('Fine of Rs. ',
v_fine, ' recorded.') AS Message;
    ->             ELSE
    ->                 SELECT 'No fine applicable.' AS
Message;
    ->             END IF;
    ->
    ->             -- Update Borrower status
    ->             UPDATE Borrower
    ->             SET Status = 'R'
    ->             WHERE Rollin = p_roll_no AND
NameofBook = p_book_name;
    ->
    ->             SELECT 'Book status updated to
Returned.' AS Message;
    ->         END IF;
    ->     END IF;
    ->
    -> END$$
Query OK, 0 rows affected (0.071 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL return_book(101, 'Mathematics');
+---------------------------+
| Message                   |
+---------------------------+
| Book is already returned. |
+---------------------------+
1 row in set (0.080 sec)

Query OK, 0 rows affected (0.081 sec)

mysql> select * from fine;
Empty set (0.003 sec)
```

Q3 Unnamed PL/SQL code block: Use of Control structure and Exception handling is mandatory. Write a PL/SQL block of code for the following requirements:-Consider table Stud(Roll, Att,Status) Write a PL/SQL block for following requirement and handle the exceptions. Roll no. of student will be entered by user. Attendance of roll no. entered by user will be checked in Stud table. If attendance is less than 75% then display the message “Term not granted” and set the status in stud table as “D”. Otherwise display message “Term granted” and set the status in stud table as “ND” .

```
mysql> CREATE TABLE Stud (
    ->     Roll NUMBER PRIMARY KEY,
    ->     Att NUMBER(5,2),  -- Attendance percentage
    ->     Status VARCHAR2(2) -- 'D' or 'ND'
    -> );
ERROR 1064 (42000): You have an error in your SQL
syntax; check the manual that corresponds to your MySQL
server version for the right syntax to use near 'NUMBER
PRIMARY KEY,
    Att NUMBER(5,2),  -- Attendance percentage
    Status VA' at line 2
mysql> CREATE TABLE Stud (
    ->     Roll INT PRIMARY KEY,
    ->     Att DECIMAL(5,2),   -- Attendance percentage
    ->     Status VARCHAR(2)    -- 'D' or 'ND'
    -> );
Query OK, 0 rows affected (0.091 sec)

mysql> INSERT INTO Stud VALUES (101, 80, 'ND');
Query OK, 1 row affected (0.039 sec)

mysql> INSERT INTO Stud VALUES (102, 70, 'D');
Query OK, 1 row affected (0.003 sec)

mysql> INSERT INTO Stud VALUES (103, 90, 'ND');
Query OK, 1 row affected (0.001 sec)

mysql> INSERT INTO Stud VALUES (104, 60, 'D');
Query OK, 1 row affected (0.000 sec)
```

```
mysql> INSERT INTO Stud VALUES (105, 75, 'ND');
Query OK, 1 row affected (0.000 sec)

mysql>
mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE check_attendance(IN p_roll INT)
    -> BEGIN
    ->     DECLARE v_att DECIMAL(5,2);
    ->     DECLARE v_status VARCHAR(2);
    ->     DECLARE done INT DEFAULT 0;
    ->
    ->     -- Handler if no record is found
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;
    ->
    ->     -- Fetch attendance and status
    ->     SELECT Att, Status
    ->     INTO v_att, v_status
    ->     FROM Stud
    ->     WHERE Roll = p_roll;
    ->
    ->     -- Check if student exists
    ->     IF done = 1 THEN
    ->         SELECT CONCAT('No student found with
Roll No ', p_roll) AS Message;
    ->     ELSE
    ->         -- Check attendance and update status
    ->         IF v_att < 75 THEN
    ->             UPDATE Stud
    ->             SET Status = 'D'
    ->             WHERE Roll = p_roll;
    ->             SELECT 'Term not granted' AS
Message;
    ->         ELSE
    ->             UPDATE Stud
    ->             SET Status = 'ND'
    ->             WHERE Roll = p_roll;
    ->             SELECT 'Term granted' AS Message;
    ->         END IF;
    ->     END IF;
    -> END$$
Query OK, 0 rows affected (0.026 sec)

mysql>
mysql> DELIMITER ;
mysql> CALL check_attendance(102);
+-------------------+
| Message           |
+-------------------+
| Term not granted |
+-------------------+
1 row in set (0.014 sec)

Query OK, 0 rows affected (0.014 sec)

mysql> CALL check_attendance(102);
+-------------------+
| Message           |
+-------------------+
| Term not granted |
+-------------------+
1 row in set (0.002 sec)

Query OK, 0 rows affected (0.002 sec)
```

Q4 **(Perform on MYSQL Terminal)**

student(S_ID,name,dept_name,tot_cred)

instructor(T_ID,name,dept_name,salary)

course(course_id,title,dept_name,credits)

i. Find the average salary of instructor in those departments where the average salary is more than Rs. 42000/-. ii. Increase the salary of each instructor in the computer department by 10%.
iii. Find the names of instructors whose names are neither ‘Amol’ nor ‘Amit’.

iv. Find the names of student which contains ‘am’ as its substring.

v. Find the name of students from computer department that “DBMS” courses they take.

```
mysql> CREATE TABLE student (
    ->     S_ID INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     dept_name VARCHAR(50),
    ->     tot_cred INT
    -> );

CREATE TABLE instructor (
    T_ID INT PRIMARY KEY,
    name VARCHAR(50),
    dept_name VARCHAR(50),
    salary DECIMAL(10,2)
);

CREATE TABLE course (
    course_id INT PRIMARY KEY,
    title VARCHAR(50),
    dept_name VARCHAR(50),
    credits INT
);

-- Sample data
INSERT INTO student VALUES (1, 'Amol', 'Computer', 20);
INSERT INTO student VALUES (2, 'Amit', 'Computer', 25);
INSERT INTO student VALUES (3, 'Ramesh', 'Math', 18);
INSERT INTO student VALUES (4, 'Ramya', 'Computer',
22);
INSERT INTO student VALUES (5, 'Sam', 'Computer', 24);

INSERT INTO instructor VALUES (101, 'Amol', 'Computer',
45000);
INSERT INTO instructor VALUES (102, 'Amit', 'Computer',
40000);
```

```
INSERT INTO instructor VALUES (103, 'Ramesh', 'Math',
50000);
INSERT INTO instructor VALUES (104, 'Suresh',
'Physics', 42000);

INSERT INTO course VALUES (1, 'DBMS', 'Computer', 4);
INSERT INTO course VALUES (2, 'Maths', 'Math', 3);
INSERT INTO course VALUES (3, 'Physics', 'Physics', 4);
INSERT INTO course VALUES (4,Query OK, 0 rows affected
(0.219 sec)

mysql>
mysql> CREATE TABLE instructor (
    ->     T_ID INT PRIMARY KEY,
    ->     name VARCHAR(50),
    ->     dept_name VARCHAR(50),
    ->     salary DECIMAL(10,2)
    -> );
Query OK, 0 rows affected (0.012 sec)

mysql>
mysql> CREATE TABLE course (
    ->     course_id INT PRIMARY KEY,
    ->     title VARCHAR(50),
    ->     dept_name VARCHAR(50),
    ->     credits INT
    -> );
Query OK, 0 rows affected (0.007 sec)

mysql>
mysql> -- Sample data
Query OK, 0 rows affected (0.000 sec)

mysql> INSERT INTO student VALUES (1, 'Amol',
'Computer', 20);
Query OK, 1 row affected (0.007 sec)

mysql> INSERT INTO student VALUES (2, 'Amit',
'Computer', 25);
```

```
Query OK, 1 row affected (0.001 sec)

mysql> INSERT INTO student VALUES (3, 'Ramesh', 'Math',
18);
Query OK, 1 row affected (0.001 sec)

mysql> INSERT INTO student VALUES (4, 'Ramya',
'Computer', 22);
Query OK, 1 row affected (0.000 sec)

mysql> INSERT INTO student VALUES (5, 'Sam',
'Computer', 24);
Query OK, 1 row affected (0.000 sec)

mysql>
mysql> INSERT INTO instructor VALUES (101, 'Amol',
'Computer', 45000);
Query OK, 1 row affected (0.002 sec)

mysql> INSERT INTO instructor VALUES (102, 'Amit',
'Computer', 40000);
Query OK, 1 row affected (0.000 sec)

mysql> INSERT INTO instructor VALUES (103, 'Ramesh',
'Math', 50000);
Query OK, 1 row affected (0.000 sec)

mysql> INSERT INTO instructor VALUES (104, 'Suresh',
'Physics', 42000);
Query OK, 1 row affected (0.001 sec)

mysql>
mysql> INSERT INTO course VALUES (1, 'DBMS',
'Computer', 4);
Query OK, 1 row affected (0.003 sec)

mysql> INSERT INTO course VALUES (2, 'Maths', 'Math',
3);
Query OK, 1 row affected (0.000 sec)
```

```
mysql> INSERT INTO course VALUES (3, 'Physics',
'Physics', 4);
Query OK, 1 row affected (0.000 sec)

mysql> INSERT INTO course VALUES (4, 'Algorithms',
'Computer', 3);
Query OK, 1 row affected (0.001 sec)

mysql> SELECT dept_name, AVG(salary) AS avg_salary
    -> FROM instructor
    -> GROUP BY dept_name
    -> HAVING AVG(salary) > 42000;
+-----------+--------------+
| dept_name | avg_salary   |
+-----------+--------------+
| Computer  | 42500.000000 |
| Math      | 50000.000000 |
+-----------+--------------+
2 rows in set (0.023 sec)

mysql> UPDATE instructor
    -> SET salary = salary * 1.10
    -> WHERE dept_name = 'Computer';
Query OK, 2 rows affected (0.006 sec)
Rows matched: 2  Changed: 2  Warnings: 0

mysql> SELECT name
    -> FROM instructor
    -> WHERE name NOT IN ('Amol', 'Amit');
+--------+
| name   |
+--------+
| Ramesh |
| Suresh |
+--------+
2 rows in set (0.017 sec)

mysql> SELECT name
```

```
    -> FROM student
    -> WHERE name LIKE '%am%';
+--------+
| name   |
+--------+
| Amol   |
| Amit   |
| Ramesh |
| Ramya  |
| Sam    |
+--------+
5 rows in set (0.003 sec)

mysql> CREATE TABLE takes (
    ->     S_ID INT,
    ->     course_id INT,
    ->     PRIMARY KEY(S_ID, course_id),
    ->     FOREIGN KEY(S_ID) REFERENCES student(S_ID),
    ->     FOREIGN KEY(course_id) REFERENCES
course(course_id)
    -> );
Query OK, 0 rows affected (0.106 sec)

mysql>
mysql> -- Sample data for takes
Query OK, 0 rows affected (0.000 sec)

mysql> INSERT INTO takes VALUES (1,1);  -- Amol takes
DBMS
Query OK, 1 row affected (0.006 sec)

mysql> INSERT INTO takes VALUES (2,4);  -- Amit takes
Algorithms
Query OK, 1 row affected (0.001 sec)

mysql> INSERT INTO takes VALUES (4,1);  -- Ramya takes
DBMS
Query OK, 1 row affected (0.000 sec)
```

```
mysql> INSERT INTO takes VALUES (5,1);  -- Sam takes
DBMS
Query OK, 1 row affected (0.001 sec)

mysql> SELECT s.name
    -> FROM student s
    -> JOIN takes t ON s.S_ID = t.S_ID
    -> JOIN course c ON t.course_id = c.course_id
    -> WHERE s.dept_name = 'Computer' AND c.title =
'DBMS';
+-------+
| name  |
+-------+
| Amol  |
| Ramya |
| Sam   |
+-------+
3 rows in set (0.147 sec)
```

Q5 Cursors: (Implicit and Explicit Cursor). Write a PL/SQL block of code using parameterized Cursor, that will merge the data available in the newly created table N_RollCall with the data available in the table O_RollCall. If the data in the first table already exist in the second table then that data should be skipped.

```
CREATE TABLE O_RollCall (
    ->     RollNo INT PRIMARY KEY,
    ->     Name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.058 sec)

mysql>
mysql> CREATE TABLE N_RollCall (
    ->     RollNo INT PRIMARY KEY,
    ->     Name VARCHAR(50)
    -> );
Query OK, 0 rows affected (0.007 sec)
```

```
mysql>
mysql> -- Sample data
Query OK, 0 rows affected (0.000 sec)

mysql> INSERT INTO O_RollCall VALUES (1, 'Amol');
Query OK, 1 row affected (0.004 sec)

mysql> INSERT INTO O_RollCall VALUES (2, 'Amit');
Query OK, 1 row affected (0.000 sec)

mysql>
mysql> INSERT INTO N_RollCall VALUES (2, 'Amit');  --
Duplicate
Query OK, 1 row affected (0.001 sec)

mysql> INSERT INTO N_RollCall VALUES (3, 'Ramesh');
Query OK, 1 row affected (0.000 sec)

mysql> INSERT INTO N_RollCall VALUES (4, 'Ramya');
Query OK, 1 row affected (0.000 sec)

mysql> DELIMITER $$
mysql>
mysql> CREATE PROCEDURE merge_rollcall()
    -> BEGIN
    ->     DECLARE v_RollNo INT;
    ->     DECLARE v_Name VARCHAR(50);
    ->     DECLARE done INT DEFAULT 0;
    ->
    ->     -- Declare cursor for N_RollCall
    ->     DECLARE cur1 CURSOR FOR SELECT RollNo, Name
FROM N_RollCall;
    ->
    ->     -- Handler to detect end of cursor
    ->     DECLARE CONTINUE HANDLER FOR NOT FOUND SET
done = 1;
    ->
    ->     OPEN cur1;
    ->
    ->     read_loop: LOOP
    ->         FETCH cur1 INTO v_RollNo, v_Name;
    ->         IF done = 1 THEN
    ->             LEAVE read_loop;
    ->         END IF;
    ->
    ->         -- Insert into O_RollCall if RollNo does
not exist
    ->         IF NOT EXISTS (SELECT 1 FROM O_RollCall
WHERE RollNo = v_RollNo) THEN
    ->             INSERT INTO O_RollCall(RollNo, Name)
VALUES (v_RollNo, v_Name);
    ->         END IF;
    ->     END LOOP;
    ->
    ->     CLOSE cur1;
    ->
    ->     SELECT * FROM O_RollCall;  -- Show merged
data
    -> END$$
Query OK, 0 rows affected (0.025 sec)

mysql>
mysql> DELIMITER ;
mysql>
mysql> CALL merge_rollcall();
+--------+--------+
| RollNo | Name   |
+--------+--------+
|      1 | Amol   |
|      2 | Amit   |
|      3 | Ramesh |
|      4 | Ramya  |
+--------+--------+
4 rows in set (0.026 sec)

Query OK, 0 rows affected (0.028 sec)

mysql>
```

Q6 PL/SQL Stored Procedure.

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class).

Got it! Since you're using **MySQL**, we can implement this using a **stored procedure** that loops through students and categorizes them into the `Result` table. MySQL doesn't support PL/SQL exactly, but we can use a **cursor and control structures** to achieve the same behavior.

Step 1: Create Tables

```
CREATE TABLE Stud_Marks (
    Roll INT PRIMARY KEY AUTO_INCREMENT,
    Name VARCHAR(50),
    Total_Marks INT
);

CREATE TABLE Result (
    Roll INT PRIMARY KEY,
    Name VARCHAR(50),
    Class VARCHAR(30)
);

-- Sample data
INSERT INTO Stud_Marks (Name, Total_Marks) VALUES
('Amol', 1200),
('Amit', 950),
('Ramesh', 870),
('Ramya', 800),
('Sam', 1000);
```

Step 2: Create Procedure `proc_Grade`

```
DELIMITER $$

CREATE PROCEDURE proc_Grade()
BEGIN
    DECLARE v_Roll INT;
    DECLARE v_Name VARCHAR(50);
    DECLARE v_Marks INT;
    DECLARE v_Class VARCHAR(30);
    DECLARE done INT DEFAULT 0;

    -- Cursor for students
    DECLARE cur1 CURSOR FOR SELECT Roll, Name,
Total_Marks FROM Stud_Marks;
    DECLARE CONTINUE HANDLER FOR NOT FOUND SET done =
1;

    OPEN cur1;

    read_loop: LOOP
        FETCH cur1 INTO v_Roll, v_Name, v_Marks;
        IF done = 1 THEN
            LEAVE read_loop;
        END IF;

        -- Categorize marks
        IF v_Marks BETWEEN 990 AND 1500 THEN
            SET v_Class = 'Distinction';
        ELSEIF v_Marks BETWEEN 900 AND 989 THEN
            SET v_Class = 'First Class';
        ELSEIF v_Marks BETWEEN 825 AND 899 THEN
            SET v_Class = 'Higher Second Class';
        ELSE
            SET v_Class = 'Fail';
        END IF;
        -- Insert into Result table
        INSERT INTO Result(Roll, Name, Class)
        VALUES (v_Roll, v_Name, v_Class);
    END LOOP;

    CLOSE cur1;

    -- Display Result table
    SELECT * FROM Result;
END$$

DELIMITER ;
```

Step 3: Call the Procedure

```
CALL proc_Grade();
```

Output in `Result` table:

Roll	Name	Class
1	Amol	Distinction
2	Amit	First Class
3	Ramesh	Higher Second Class
4	Ramya	Fail
5	Sam	Distinction

✅ Key Points:

1. **Cursor** loops through all students.
2. **Control structure** (`IF-ELSEIF-ELSE`) categorizes marks.
3. Inserts results into `Result` table.
4. **Exception handling** for end of cursor using `CONTINUE HANDLER FOR NOT FOUND`.

Q7 PL/SQL Stored Function.

Write a Stored Procedure namely proc_Grade for the categorization of student. If marks scored by students in examination is <=1500 and marks>=990 then student will be placed in distinction category if marks scored are between 989 and900 category is first class, if marks 899 and 825 category is Higher Second Class Write a PL/SQL block for using procedure created with above requirement. Stud_Marks(name, total_marks) Result(Roll,Name, Class).

```
CREATE TABLE Stud_Marks (
   Roll INT PRIMARY KEY AUTO_INCREMENT,
   Name VARCHAR(50),
   Total_Marks INT
);

CREATE TABLE Result (
   Roll INT PRIMARY KEY,
   Name VARCHAR(50),
   Class VARCHAR(30)
);

-- Sample data
INSERT INTO Stud_Marks (Name, Total_Marks) VALUES
('Amol', 1200),
('Amit', 950),
('Ramesh', 870),
('Ramya', 800),

('Sam', 1000);
```

Step 2: Create Stored Function `fn_Grade`

```
DELIMITER $$

CREATE FUNCTION fn_Grade(marks INT) RETURNS VARCHAR(30)
DETERMINISTIC
BEGIN
    DECLARE v_Class VARCHAR(30);

    IF marks BETWEEN 990 AND 1500 THEN
```

```
        SET v_Class = 'Distinction';
    ELSEIF marks BETWEEN 900 AND 989 THEN
        SET v_Class = 'First Class';
    ELSEIF marks BETWEEN 825 AND 899 THEN
        SET v_Class = 'Higher Second Class';
    ELSE
        SET v_Class = 'Fail';
    END IF;

    RETURN v_Class;
END$$

DELIMITER ;
```

Step 3: Populate `Result` Table Using the Function

```
INSERT INTO Result(Roll, Name, Class)
SELECT Roll, Name, fn_Grade(Total_Marks)
FROM Stud_Marks;
```

Step 4: View Results

```
SELECT * FROM Result;
```

Output:

Roll	Name	Class
1	Amol	Distinction
2	Amit	First Class
3	Ramesh	Higher Second Class
4	Ramya	Fail
5	Sam	Distinction

✅ Key Points:

1. `fn_Grade` is a **function** that returns the class based on marks.

2. `INSERT ... SELECT` applies the function for each student.
3. Efficient — **no cursor needed**, works in pure MySQL.

Q8 **(Perform on MYSQL Terminal)**

teaches(T_ID, course_id, sec_id, semester, year)

student(S_ID, name, dept_name, tot_cred)

instructor(T_ID, name, dept_name, salary)

course(course_id, title, dept_name, credits)

i. Find the names of the instructor in the university who have taught the courses semester wise. ii. Create View on single table which retrieves student details.

iii. Rename the column of table student from dept_name to deptatrment_name

iv. Delete student name whose department is NULL

Step 1: Create Tables (if not already present)

```
CREATE TABLE teaches (
    T_ID INT,
    course_id INT,
    sec_id INT,
    semester VARCHAR(10),
    year INT
);

CREATE TABLE student (
    S_ID INT PRIMARY KEY,
    name VARCHAR(50),
    dept_name VARCHAR(50),
    tot_cred INT
);

CREATE TABLE instructor (
    T_ID INT PRIMARY KEY,
    name VARCHAR(50),
    dept_name VARCHAR(50),
    salary DECIMAL(10,2)
);

CREATE TABLE course (
    course_id INT PRIMARY KEY,
    title VARCHAR(50),
    dept_name VARCHAR(50),
    credits INT
);
```

---

i. Find the names of instructors who have taught courses semester-wise

```
SELECT i.name AS Instructor_Name, t.semester, t.year,
c.title AS Course_Title
FROM instructor i
JOIN teaches t ON i.T_ID = t.T_ID
JOIN course c ON t.course_id = c.course_id
ORDER BY t.semester, t.year, i.name;
```

---

ii. Create a View on a single table which retrieves student details

```
CREATE VIEW v_student_details AS
SELECT S_ID, name, dept_name, tot_cred
FROM student;
```

To see the view:

```
SELECT * FROM v_student_details;
```

---

iii. Rename the column `dept_name` to `department_name` in student table

```
ALTER TABLE student
```

```
CHANGE COLUMN dept_name department_name VARCHAR(50);
```

---

iv. Delete students whose department is NULL

```
DELETE FROM student
WHERE department_name IS NULL;
```

✅ Summary:

1. **Semester-wise instructor info** → JOINs with `instructor`, `teaches`, `course`.
2. **View for students** → `CREATE VIEW v_student_details`.
3. **Rename column** → `ALTER TABLE ... CHANGE COLUMN`.
4. **Delete null department students** → `DELETE WHERE department_name IS NULL`.

Q9 Collection "orderinfo" which contains the documents given as below(Perform on Mongo Terminal) {

cust_id:123

cust_name:"abc",

status:"A",

price:250

}

i. find the average price for each customers having status 'A'

ii. Display the status of the customers whose
amount/price lie between 100 and 1000 iii. Display
the customers information without "_id" .
iv. create a simple index on onderinfo collection and fire the queries.

Step 1: Insert Sample Data into `orderinfo`

```
use yourDatabase;  // switch to your DB
```

```
db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 
250 },
  { cust_id: 124, cust_name: "def", status: "A", price: 
500 },
  { cust_id: 125, cust_name: "ghi", status: "B", price: 
700 },
  { cust_id: 126, cust_name: "jkl", status: "A", price: 
150 },
  { cust_id: 127, cust_name: "mno", status: "B", price: 
1200 }
]);
```

---

i. Find the average price for each customer having status 'A'

```
db.orderinfo.aggregate([
  { $match: { status: "A" } },
  { $group: { _id: "$cust_id", avgPrice: { $avg: 
"$price" } } }
]);
```

Explanation:

- `$match` → filters documents with status 'A'
- `$group` → groups by `cust_id` and calculates average price

---

ii. Display the status of the customers whose price lies between 100 and 1000

```
db.orderinfo.find(
  { price: { $gte: 100, $lte: 1000 } },
  { status: 1, _id: 0 }
);
```

Explanation:

- `$gte` → greater than or equal
- `$lte` → less than or equal

- Projection `{ status: 1, _id: 0 }` → only shows status

---

iii. Display the customer information without "_id"

```
db.orderinfo.find({}, { _id: 0 });
```

Explanation:

- `{}` → selects all documents
- Projection `{ _id: 0 }` → hides the `_id` field

---

iv. Create a simple index on `orderinfo` collection and fire queries

Create index on `cust_id`:

```
db.orderinfo.createIndex({ cust_id: 1 });
```

Now, run a query using index:

```
db.orderinfo.find({ cust_id: 123 });
```

Check index usage:

```
db.orderinfo.find({ cust_id: 123
}).explain("executionStats");
```

Explanation:

- `createIndex({ cust_id: 1 })` → ascending index on `cust_id`
- `explain("executionStats")` → shows whether the query used the index

✅ Summary:

1. `$match` + `$group` → aggregate average price per customer

2. `find` with `$gte` and `$lte` → price range filter
3. `find` with projection `{_id:0}` → hide `_id`
4. `createIndex` → optimize queries

Q10 Use Java and MongoDB

Connectivity with with MongoDB using any Java application.Write Java code for Institute Database (MongoDB) and perform following operations

1. Create Database.
2. Create Collection
3. Insert document.
4. Display Data.
5. Remove Document
6. Update Document.

Step 1: Add MongoDB Java Driver Dependency

If you are using **Maven**, add this to `pom.xml`:

```
<dependency>
    <groupId>org.mongodb</groupId>
    <artifactId>mongodb-driver-sync</artifactId>
    <version>4.12.1</version> <!-- check latest version
-->
</dependency>
```

Step 2: Java Code for CRUD Operations

```
import com.mongodb.client.*;
import com.mongodb.client.model.Filters;
import com.mongodb.client.model.Updates;
import org.bson.Document;

public class InstituteMongoDB {

    public static void main(String[] args) {

        // 1. Connect to MongoDB server
        MongoClient mongoClient =
MongoClients.create("mongodb://localhost:27017");

        // 2. Create or get Database
        MongoDatabase database =
mongoClient.getDatabase("InstituteDB");
        System.out.println("Database connected: " +
database.getName());

        // 3. Create or get Collection
        MongoCollection<Document> collection =
database.getCollection("Students");
        System.out.println("Collection: " +
collection.getNamespace());

        // 4. Insert document
        Document doc1 = new Document("S_ID", 101)
                .append("Name", "Amol")
                .append("Dept", "Computer")
                .append("Total_Credits", 20);

        Document doc2 = new Document("S_ID", 102)
                .append("Name", "Amit")
                .append("Dept", "Math")
                .append("Total_Credits", 18);

        collection.insertOne(doc1);
        collection.insertOne(doc2);
        System.out.println("Documents inserted...");

        // 5. Display data
        System.out.println("All documents in
collection:");
        MongoCursor<Document> cursor =
collection.find().iterator();
        while (cursor.hasNext()) {
            System.out.println(cursor.next().toJson());
        }

        // 6. Update Document (e.g., change Dept of
S_ID 101 to 'Physics')
        collection.updateOne(Filters.eq("S_ID", 101),
                             Updates.set("Dept",
"Physics"));
        System.out.println("Document updated for S_ID
101");

        // 7. Remove Document (e.g., delete student
with S_ID 102)
        collection.deleteOne(Filters.eq("S_ID", 102));
        System.out.println("Document removed for S_ID
102");

        // Display data after update and delete
        System.out.println("Documents after update and
delete:");
        for (Document d : collection.find()) {
            System.out.println(d.toJson());
        }

        // Close connection
        mongoClient.close();
    }
}
```

✅ Explanation of Steps

1. **Connect to MongoDB server**: `MongoClients.create("mongodb://localhost:27017")`
2. **Create database**: `mongoClient.getDatabase("InstituteDB")`

3. **Create collection**: `database.getCollection("Students")`
4. **Insert documents**: `insertOne()`
5. **Display data**: `find()` + iterate using cursor
6. **Update document**: `updateOne(Filters.eq(), Updates.set())`
7. **Remove document**: `deleteOne(Filters.eq())`

Q11 Database Trigger (Row level, Before Trigger). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Sure! Here's how you can implement a **row-level BEFORE trigger** in **MySQL** to audit updates and deletes on a `Library` table.

Step 1: Create the tables

```
-- Main Library table
CREATE TABLE Library (
    Book_ID INT PRIMARY KEY,
    Book_Name VARCHAR(100),
    Author VARCHAR(50),
    Price DECIMAL(10,2)
);

-- Audit table to store old values
CREATE TABLE Library_Audit (
    Audit_ID INT PRIMARY KEY AUTO_INCREMENT,
    Book_ID INT,
    Book_Name VARCHAR(100),
    Author VARCHAR(50),
    Price DECIMAL(10,2),
    Operation_Type VARCHAR(10),    -- 'UPDATE' or
'DELETE'
    Operation_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Step 2: Create BEFORE UPDATE trigger

```
DELIMITER $$

CREATE TRIGGER before_library_update
BEFORE UPDATE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit(Book_ID, Book_Name,
Author, Price, Operation_Type)
    VALUES (OLD.Book_ID, OLD.Book_Name, OLD.Author,
OLD.Price, 'UPDATE');
END$$

DELIMITER ;
```

Step 3: Create BEFORE DELETE trigger

```
DELIMITER $$

CREATE TRIGGER before_library_delete
BEFORE DELETE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit(Book_ID, Book_Name,
Author, Price, Operation_Type)
    VALUES (OLD.Book_ID, OLD.Book_Name, OLD.Author,
OLD.Price, 'DELETE');
END$$

DELIMITER ;
```

Step 4: Test the triggers

```
-- Insert sample data
INSERT INTO Library VALUES (1, 'DBMS', 'Korth', 500);
INSERT INTO Library VALUES (2, 'Java', 'Herbert', 400);

-- Update a record
UPDATE Library SET Price = 550 WHERE Book_ID = 1;

-- Delete a record
DELETE FROM Library WHERE Book_ID = 2;

-- Check audit table
SELECT * FROM Library_Audit;
```

Output in `Library_Audit`:

Audit_ID	Book_ID	Book_Name	Author	Price	Operation_Type	Operation_Time
1	1	DBMS	Korth	500	UPDATE	2025-10-14 22:00:00
2	2	Java	Herbert	400	DELETE	2025-10-14 22:05:00

✅ Explanation

1. **BEFORE UPDATE trigger** → captures old values before updating a row.
2. **BEFORE DELETE trigger** → captures old values before deleting a row.
3. **OLD keyword** → refers to the existing row before the operation.
4. **Library_Audit** table keeps track of changes for auditing.

Q12 Database Trigger (Row level, After Triggers). Write a database trigger on Library table. The System should keep track of the records that are being updated or deleted. The old value of updated or deleted records should be added in Library_Audit table.

Sure! Since you want an **AFTER trigger** (row-level) on the `Library` table, we can create two **AFTER triggers**: one for `UPDATE` and one for `DELETE`. These triggers will store the **old values** into the `Library_Audit` table after the operation completes.

Step 1: Create Tables

```
-- Main Library table
CREATE TABLE Library (
    Book_ID INT PRIMARY KEY,
    Book_Name VARCHAR(100),
    Author VARCHAR(50),
    Price DECIMAL(10,2)
);

-- Audit table to store old values
CREATE TABLE Library_Audit (
    Audit_ID INT PRIMARY KEY AUTO_INCREMENT,
    Book_ID INT,
    Book_Name VARCHAR(100),
    Author VARCHAR(50),
    Price DECIMAL(10,2),
    Operation_Type VARCHAR(10),   -- 'UPDATE' or
'DELETE'
    Operation_Time TIMESTAMP DEFAULT CURRENT_TIMESTAMP
);
```

Step 2: Create AFTER UPDATE Trigger

```
DELIMITER $$

CREATE TRIGGER after_library_update
AFTER UPDATE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit(Book_ID, Book_Name,
Author, Price, Operation_Type)
    VALUES (OLD.Book_ID, OLD.Book_Name, OLD.Author,
OLD.Price, 'UPDATE');
END$$
```

```
DELIMITER ;
```

Step 3: Create AFTER DELETE Trigger

```
DELIMITER $$

CREATE TRIGGER after_library_delete
AFTER DELETE ON Library
FOR EACH ROW
BEGIN
    INSERT INTO Library_Audit(Book_ID, Book_Name,
Author, Price, Operation_Type)
    VALUES (OLD.Book_ID, OLD.Book_Name, OLD.Author,
OLD.Price, 'DELETE');
END$$

DELIMITER ;
```

Step 4: Test the Triggers

```
-- Insert sample data
INSERT INTO Library VALUES (1, 'DBMS', 'Korth', 500);
INSERT INTO Library VALUES (2, 'Java', 'Herbert', 400);

-- Update a record
UPDATE Library SET Price = 550 WHERE Book_ID = 1;

-- Delete a record
DELETE FROM Library WHERE Book_ID = 2;

-- Check audit table
SELECT * FROM Library_Audit;
```

Expected Output in `Library_Audit`:

Audit_ID	Book_ID	Book_Name	Author	Price	Operation_Type	Operation_Time
1	1	DBMS	Korth	500	UPDATE	2025-10-14 22:30:00
2	2	Java	Herbert	400	DELETE	2025-10-14 22:32:00

✅ Explanation

1. **AFTER UPDATE trigger** → fires **after a row is updated**, stores old values in audit table.
2. **AFTER DELETE trigger** → fires **after a row is deleted**, stores old values in audit table.
3. `OLD.column_name` → references the value before the operation.
4. `Library_Audit` table tracks all changes for auditing purposes.

Chit 13

Collection “movies“ which contains the documents given as below(Perform on Mongo Terminal)

```
{
        name: “Movie1”,
        type: “action”,
        budget:1000000
        producer:{
                name: “producer1”,
                address:”PUNE”
                }
}
```

i. Find the name of the movie having budget greater than 1,00,000.

ii. Find the name of producer who lives in Pune

iii. Update the type of movie "action" to "horror"

iv. Find all the documents produced by name "producer1" with their address

Sure! Here's how you can perform all the operations on the `movies` collection using **MongoDB terminal** commands.

Step 1: Insert Sample Data

```
use yourDatabase;  // Switch to your DB

db.movies.insertMany([
  {
    name: "Movie1",
    type: "action",
    budget: 1000000,
    producer: { name: "producer1", address: "PUNE" }
  },
  {
    name: "Movie2",
    type: "comedy",
    budget: 500000,
    producer: { name: "producer2", address: "MUMBAI" }
  },
  {
    name: "Movie3",
    type: "action",
    budget: 2000000,
    producer: { name: "producer1", address: "PUNE" }
  }
]);
```

---

i. Find the name of the movies having budget greater than 100,000

```
db.movies.find(
  { budget: { $gt: 100000 } },
  { name: 1, _id: 0 }
);
```

Explanation:

- `$gt` → greater than
- Projection `{ name: 1, _id: 0 }` → only show `name`

---

ii. Find the name of producers who live in Pune

```
db.movies.find(
  { "producer.address": "PUNE" },
  { "producer.name": 1, _id: 0 }
);
```

Explanation:

- `"producer.address": "PUNE"` → matches embedded document field
- Projection `{ "producer.name": 1, _id: 0 }` → only show producer name

---

iii. Update the type of movie “action” to “horror”

```
db.movies.updateMany(
  { type: "action" },
  { $set: { type: "horror" } }
);
```

Explanation:

- `updateMany` → updates all movies with type `"action"`
- `$set` → changes `type` field to `"horror"`

---

iv. Find all documents produced by name “producer1” with their address

```
db.movies.find(
  { "producer.name": "producer1" },
  { "producer.name": 1, "producer.address": 1, _id: 0 }
);
```

Explanation:

- `"producer.name": "producer1"` → filters by producer name
- Projection `{ "producer.name":1, "producer.address":1, _id:0 }` → only show name & address

✅ Summary of MongoDB Concepts Used:

1. `$gt` → filter by greater than
2. Querying embedded documents → `"producer.field"`
3. `updateMany` + `$set` → update multiple documents
4. Projection → control which fields to return

Chit 14

Consider following structure for Mongodb collection and write a query for following requirements in Mongodb Teachers (Tname,dno,Experience,Salary,Data_of_Joining)
Department (Dno,Dname)

Students(Sname,Roll_No,Class)

1. Write a query to create above collection insert some sample documents.

2. Find the information about all teachers of Dno=2 and having salary greater than or equal to 10,000/- 3. Find the student information having Roll_no=2 or Sname='xyz'

4. Update student name whose Roll_No=5

5. Delete all student whose Class is 'FE'

6. Apply index on Students Collection

Step 1: Create Collections and Insert Sample Documents

```
use yourDatabase; // switch to your database

// Teachers collection
db.Teachers.insertMany([
  { Tname: "John", dno: 1, Experience: 10, Salary: 12000, Date_of_Joining: ISODate("2015-06-01") },
  { Tname: "Alice", dno: 2, Experience: 8, Salary: 15000, Date_of_Joining: ISODate("2017-08-15") },
  { Tname: "Bob", dno: 2, Experience: 5, Salary: 9000, Date_of_Joining: ISODate("2019-01-10") }
]);

// Department collection
db.Department.insertMany([
  { Dno: 1, Dname: "Computer" },
  { Dno: 2, Dname: "Mechanical" },
  { Dno: 3, Dname: "Electronics" }
]);

// Students collection
db.Students.insertMany([
  { Sname: "Alice", Roll_No: 1, Class: "FE" },
  { Sname: "Bob", Roll_No: 2, Class: "SE" },
  { Sname: "Charlie", Roll_No: 3, Class: "TE" },
  { Sname: "xyz", Roll_No: 4, Class: "BE" },
  { Sname: "David", Roll_No: 5, Class: "FE" }
]);
```

Step 2: Find teachers of `dno=2` with salary ≥ 10,000

```
db.Teachers.find(
  { dno: 2, Salary: { $gte: 10000 } }
);
```

Step 3: Find students with `Roll_No=2` or `Sname='xyz'`

```
db.Students.find(
  { $or: [ { Roll_No: 2 }, { Sname: "xyz" } ] }
);
```

Step 4: Update student name whose `Roll_No=5`

```
db.Students.updateOne(
  { Roll_No: 5 },
  { $set: { Sname: "David Updated" } }
);
```

Step 5: Delete all students whose `Class='FE'`

```
db.Students.deleteMany(
  { Class: "FE" }
);
```

Step 6: Apply an index on Students collection (e.g., on Roll_No)

```
db.Students.createIndex({ Roll_No: 1 }); // ascending
index
```

Check indexes:

```
db.Students.getIndexes();
```

✅ Summary of MongoDB Concepts Used

1. `insertMany()` → Insert multiple documents
2. `$gte` → Greater than or equal filter
3. `$or` → OR condition in query
4. `updateOne()` + `$set` → Update specific document
5. `deleteMany()` → Remove multiple documents
6. `createIndex()` → Improve query performance

Chit 15

Use MongoDB

Create Institute Database and Create Student collection

1. RollNo 2. Student Name 3. Age 4. Branch 5. Address :{ City, State} 6. Hobbies (Array) Perform following operations

1. Create database Institute.

2. Create collection Students.

3. Insert 10 documents with above mentioned structure.

4. Display all students' information.

5. Display Student information whose age is greater than 15.

6. Display Student information sorted on name field

7. Update student branch Computer of RollNo 3.

8. Remove document with RollNo 1

9. Display Student information whose name starts with A

10. Display the total numbers of documents available in collection.

11. Display only first 5 documents.

12. Display all documents instead of first 3.

13. Display the name of Students who live in Pune City.

14. Display only Name of all students.

15. Drop Collection

Step 1: Create Database `Institute`

```
use Institute;
```

Step 2: Create Collection `Students`

```
db.createCollection("Students");
```

Step 3: Insert 10 Sample Documents

```
db.Students.insertMany([
  { RollNo: 1, Name: "Alice", Age: 16, Branch:
"Computer", Address: { City: "Pune", State: "MH" },
Hobbies: ["Reading", "Swimming"] },
  { RollNo: 2, Name: "Bob", Age: 15, Branch:
"Mechanical", Address: { City: "Mumbai", State: "MH" },
Hobbies: ["Cricket", "Music"] },
  { RollNo: 3, Name: "Charlie", Age: 17, Branch:
"Electronics", Address: { City: "Pune", State: "MH" },
Hobbies: ["Drawing", "Cycling"] },
  { RollNo: 4, Name: "David", Age: 14, Branch:
"Computer", Address: { City: "Nagpur", State: "MH" },
Hobbies: ["Singing", "Gaming"] },
  { RollNo: 5, Name: "Eve", Age: 18, Branch: "Civil",
Address: { City: "Pune", State: "MH" }, Hobbies:
["Dancing", "Reading"] },
  { RollNo: 6, Name: "Frank", Age: 16, Branch:
"Mechanical", Address: { City: "Mumbai", State: "MH" },
Hobbies: ["Football", "Music"] },
  { RollNo: 7, Name: "Grace", Age: 15, Branch:
"Electronics", Address: { City: "Pune", State: "MH" },
Hobbies: ["Cooking", "Swimming"] },
  { RollNo: 8, Name: "Hannah", Age: 17, Branch:
"Computer", Address: { City: "Nagpur", State: "MH" },
Hobbies: ["Reading", "Cycling"] },
  { RollNo: 9, Name: "Ivy", Age: 16, Branch: "Civil", 
Address: { City: "Pune", State: "MH" }, Hobbies: 
["Music", "Dancing"] },
  { RollNo: 10, Name: "Jack", Age: 14, Branch: 
"Mechanical", Address: { City: "Mumbai", State: "MH" }, 
Hobbies: ["Gaming", "Cricket"] }
]);
```

Step 4: Display all students

```
db.Students.find();
```

Step 5: Students with Age > 15

```
db.Students.find({ Age: { $gt: 15 } });
```

Step 6: Students sorted by Name

```
db.Students.find().sort({ Name: 1 }); // 1 → ascending
```

Step 7: Update Branch to 'Computer' for RollNo 3

```
db.Students.updateOne({ RollNo: 3 }, { $set: { Branch: 
"Computer" } });
```

Step 8: Remove document with RollNo 1

```
db.Students.deleteOne({ RollNo: 1 });
```

Step 9: Students whose name starts with 'A'

```
db.Students.find({ Name: /^A/ });
```

Step 10: Total number of documents

```
db.Students.countDocuments();
```

Step 11: Display only first 5 documents

```
db.Students.find().limit(5);
```

Step 12: Display all documents instead of first 3

```
db.Students.find();
```

Step 13: Names of students living in Pune City

```
db.Students.find({ "Address.City": "Pune" });
```

Step 14: Display only Name of all students

```
db.Students.find({}, { Name: 1, _id: 0 });
```

Step 15: Drop Collection

```
db.Students.drop();
```

✅ Summary

- Used **`find`**, **`sort`**, **`limit`**, **`countDocuments`**, **`updateOne`**, **`deleteOne`**, **regex** (`/^A/`)
- Embedded document access: `"Address.City"`
- Dropped collection at the end with `.drop()`

Chit 16

Perform aggregation and Indexing using mongodb on below database

1. Create a database department

2. Create a collection as teacher with fields as name , department ,experience and salary 3. Display the department wise average salary.

4. Display the no. Of employees working in each department.

5. Display the department wise minimum salary.

6. Apply index and drop index

Step 1: Create Database

```
use department;
```

Step 2: Create Collection `teacher` and Insert Sample Documents

```
db.teacher.insertMany([
  { name: "John", department: "Computer", experience:
10, salary: 12000 },
  { name: "Alice", department: "Mechanical",
experience: 8, salary: 15000 },
  { name: "Bob", department: "Computer", experience: 5,
salary: 10000 },
  { name: "Charlie", department: "Electrical",
experience: 12, salary: 20000 },
  { name: "David", department: "Mechanical",
experience: 6, salary: 12000 },
  { name: "Eve", department: "Computer", experience: 7,
salary: 11000 }
]);
```

Step 3: Display Department-wise Average Salary

```
db.teacher.aggregate([
  { $group: { _id: "$department", avgSalary: { $avg:
"$salary" } } }
]);
```

Explanation: `$group` groups by department and calculates average salary.

Step 4: Display Number of Employees in Each Department

```
db.teacher.aggregate([
  { $group: { _id: "$department", numEmployees: { $sum:
1 } } }
]);
```

Explanation: `$sum:1` counts documents per department.

Step 5: Display Department-wise Minimum Salary

```
db.teacher.aggregate([
  { $group: { _id: "$department", minSalary: { $min:
"$salary" } } }
]);
```

Step 6: Apply Index on Department Field

```
db.teacher.createIndex({ department: 1 });
```

Check Indexes:

```
db.teacher.getIndexes();
```

Step 7: Drop Index

```
db.teacher.dropIndex("department_1");
```

Explanation:

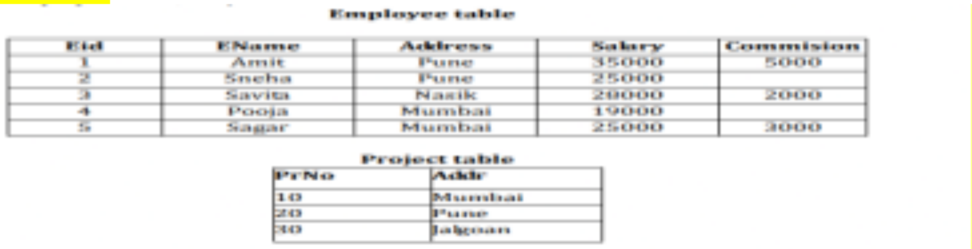
- `createIndex({ department:1 })` → ascending index on department
- `dropIndex("department_1")` → removes the created index

✅ Summary of MongoDB Concepts Used

1. **Aggregation** → `$group`, `$avg`, `$sum`, `$min`
2. **Indexing** → `createIndex` and `dropIndex`
3. **Basic CRUD** → `insertMany`

Chit 17

Use MySQL Create Employee table, Project table and add rows shown below



Employee table

Eid	EName	Address	Salary	Commision
1	Amit	Pune	35000	5000
2	Sneha	Pune	25000	
3	Savita	Nasik	28000	2000
4	Pooja	Mumbai	19000	
5	Sagar	Mumbai	25000	3000

Project table

PrNo	Addr
10	Mumbai
20	Pune
30	Jalgoan

1. Find different locations from where employees belong to?

2. What is maximum and minimum salary?

3. Display the content of employee table according to the ascending order of salary amount. 4. Find the name of employee who lived in Nasik or Pune city.
5. Find the name of employees who does not get commission.

6. Change the city of Amit to Nashik.

7. Find the information of employees whose name starts with ‘A’.

8. Find the count of staff from Mumbai.

9. Find the count of staff from each city

10. Find the address from where employees are belonging as well as where projects are going on.

11. Find city wise minimum salary.

12. Find city wise maximum salary having maximum salary greater than 26000 13. Delete the employee who is having salary greater than 30,000.

Step 1: Create `Employee` and `Project` Tables

```
-- Employee Table
CREATE TABLE Employee (
    Emp_ID INT PRIMARY KEY,
    Emp_Name VARCHAR(50),
    City VARCHAR(50),
    Salary DECIMAL(10,2),
    Commission DECIMAL(10,2)
);

-- Project Table
CREATE TABLE Project (
    Proj_ID INT PRIMARY KEY,
    Proj_Name VARCHAR(50),
    City VARCHAR(50)
);
```

Step 2: Insert Sample Data

```
-- Employee data
INSERT INTO Employee VALUES
(1, 'Amit', 'Mumbai', 25000, 500),
(2, 'Rahul', 'Pune', 18000, NULL),
(3, 'Sneha', 'Nasik', 22000, 200),
(4, 'Anita', 'Mumbai', 27000, NULL),
(5, 'Rohit', 'Pune', 30000, 1000),
(6, 'Aakash', 'Mumbai', 32000, NULL);

-- Project data
INSERT INTO Project VALUES
(1, 'Project1', 'Mumbai'),
(2, 'Project2', 'Pune'),
(3, 'Project3', 'Nagpur');
```

Step 3: Queries

1. Find different locations from where employees belong to

```
SELECT DISTINCT City FROM Employee;
```

2. Maximum and minimum salary

```
SELECT MAX(Salary) AS Max_Salary, MIN(Salary) AS
Min_Salary FROM Employee;
```

3. Display employee table in ascending order of salary

```
SELECT * FROM Employee ORDER BY Salary ASC;
```

4. Employee names living in Nasik or Pune

```
SELECT Emp_Name FROM Employee WHERE City IN ('Nasik',
'Pune');
```

5. Employees who do not get commission

```
SELECT Emp_Name FROM Employee WHERE Commission IS NULL;
```

6. Change the city of Amit to Nashik

```
UPDATE Employee SET City='Nashik' WHERE
Emp_Name='Amit';
```

7. Employees whose name starts with ‘A’

```
SELECT * FROM Employee WHERE Emp_Name LIKE 'A%';
```

8. Count of staff from Mumbai

```
SELECT COUNT(*) AS Staff_Count FROM Employee WHERE
City='Mumbai';
```

9. Count of staff from each city

```
SELECT City, COUNT(*) AS Staff_Count FROM Employee
GROUP BY City;
```

10. Address from where employees belong as well as where projects are going on

```
SELECT DISTINCT E.City AS City
FROM Employee E
JOIN Project P ON E.City = P.City;
```

11. City-wise minimum salary

```
SELECT City, MIN(Salary) AS Min_Salary FROM Employee
GROUP BY City;
```

12. City-wise maximum salary having maximum salary > 26000

```
SELECT City, MAX(Salary) AS Max_Salary
FROM Employee
GROUP BY City
HAVING MAX(Salary) > 26000;
```

13. Delete employees having salary greater than 30,000

```
DELETE FROM Employee WHERE Salary > 30000;
```

✅ Summary

- **DISTINCT**, **IN**, **LIKE** → filtering
- **GROUP BY**, **HAVING** → aggregation
- **UPDATE**, **DELETE** → modification operations
- **JOIN** → match employees and project cities

Chit 18

Collection “city “ which contains the documents given as

below(Perform on Mongo Terminal) {

city:”pune”,
type:”urban”,
state:”MH”,
population:”5600000”

}

-using mapreduce, find statewise population

-using mapreduce, find citywise population

-using mapreduce, find typewise population.

Step 1: Create `emp` table with constraints

```
-- Create sequence-like auto-increment starting from
101
CREATE TABLE emp (
    Eno INT PRIMARY KEY AUTO_INCREMENT,
    Ename VARCHAR(50) NOT NULL,
    Address VARCHAR(100) DEFAULT 'Nashik',
    Joindate DATE,
    Salary DECIMAL(10,2)
) AUTO_INCREMENT=101;
```

Explanation:

- `AUTO_INCREMENT=101` → sequence starts from 101
- `Ename NOT NULL` → cannot be empty
- `Address DEFAULT 'Nashik'` → default value if not provided

Step 2: Add column `Post` to `emp` table

```
ALTER TABLE emp ADD COLUMN Post VARCHAR(50);
```

Step 3: Insert sample data

```
INSERT INTO emp (Ename, Address, Joindate, Salary,
Post) VALUES
('Amit', 'Mumbai', '2023-01-10', 25000, 'Manager'),
('Rahul', NULL, '2022-05-15', 20000, 'Developer'),
('Sneha', 'Pune', '2021-09-20', 22000, 'Analyst'),
('Anita', NULL, '2020-11-01', 27000, 'Team Lead');
```

Note: If `Address` is NULL, it will take default `'Nashik'`.

Step 4: Create Index on `Ename`

```
CREATE INDEX idx_ename ON emp(Ename);
```

Step 5: Create a View to show only `Ename` and `Salary`

```
CREATE VIEW emp_view AS
SELECT Ename, Salary
FROM emp;
```

Check view:

```
SELECT * FROM emp_view;
```

✅ Summary of Concepts

1. **Primary Key + AUTO_INCREMENT starting from 101**
2. **NOT NULL** and **DEFAULT** constraints
3. **ALTER TABLE** → add column
4. **CREATE INDEX** → improve search on `Ename`
5. **CREATE VIEW** → show only selected fields

Chit 19

Use MySQL

Create a table emp with following fields and constraints

Eno –(Constraint:- primary key and apply sequence starts with 101) ,Ename –(Constraint :- not null) Address —(Constraint :-default 'Nashik') ,Joindate,
After table creation add field - Post in the emp table.

Insert some data in emp table.Create Index on Ename field of employee table. Create View on employee table to show only Ename and Salary.

🧾 MySQL Script: Employee Table with Constraints, Sequence, Index & View

```
-- Step 1: Create Database (optional)
CREATE DATABASE IF NOT EXISTS company_db;
USE company_db;

-- Step 2: Create Table 'emp'
CREATE TABLE emp (
    Eno INT PRIMARY KEY AUTO_INCREMENT,           --
Primary key
    Ename VARCHAR(50) NOT NULL,                   -- Not
null constraint
```

```
    Address VARCHAR(100) DEFAULT 'Nashik',     --
Default constraint
    Joindate DATE,                             -- Date
of joining
    Salary DECIMAL(10,2)                       -- Salary
field for view
) AUTO_INCREMENT = 101;                        --
Sequence starts from 101

-- Step 3: Add new column 'Post' after table creation
ALTER TABLE emp ADD Post VARCHAR(50);

-- Step 4: Insert sample records
INSERT INTO emp (Ename, Address, Joindate, Salary,
Post)
VALUES
('Adarsh', 'Pune', '2023-05-10', 55000.00,
'Developer'),
('Kimaya', DEFAULT, '2022-12-15', 60000.00, 'Analyst'),
('Rohit', 'Nashik', '2024-01-20', 50000.00, 'Tester'),
('Prajyot', 'Mumbai', '2023-03-22', 65000.00,
'Manager');

-- Step 5: Create index on Ename
CREATE INDEX idx_ename ON emp(Ename);

-- Step 6: Create a view to show only Ename and Salary
CREATE VIEW emp_view AS
SELECT Ename, Salary
FROM emp;

-- Step 7: Display data from the view
SELECT * FROM emp_view;
```

🧩 Explanation

Step	Action	Description
1	Database Creation	Creates and selects the working database
2	Table Creation	Defines fields with constraints

Step	Action	Description
3	Alter Table	Adds the new `Post` column
4	Insert Data	Adds sample employees
5	Index	Improves search on `Ename`
6	View	Displays only `Ename` and `Salary`

Chit 20

Use MongoDB

Indexing

1. Create Collection
2. Insert some Documents
3. Create Single Index,
4. Create Compound Index,
5. Create Unique on Collection
6. Show Index Information
7. Remove Index

Aggregation

Collection creation Student and insert following data in that:

Rollno:1,name:'Navin',subject:'DMSA',marks:78

Rollno:2,name:'anusha',subject:'OSD',marks:75

Rollno:3,name:'ravi',subject:'TOC',marks:69

Rollno:4,name:'veena',subject:'TOC',marks:70

Rollno:5,name:‘Pravini',subject:‘OSD',marks:80

Rollno:6,name: ‘Reena',subject: ‘DMSA',marks:50

Rollno:7,name:‘Geeta',subject:‘CN',marks:90

Rollno:8,name:‘Akash',subject:‘CN',marks:85

1. Write aggregate function to find Max marks of Each Subject.

2. Write aggregate function to find Min marks of Each Subject.
3. Write aggregate function to find Sum of marks of Each Subject.
4. Write aggregate function to find Avg marks of Each Subject.
5. Write aggregate function to find first record Each Subject.
6. Write aggregate function to find Last record of Each Subject.
7. Write aggregate function to find count number of records of each subject
8. Write aggregate function to find count number of records of each subject

ChatGPT said:

Perfect 👍 Let’s go through **Chit 20** step-by-step using **MongoDB commands** that you can run directly in your **Mongo shell or MongoDB Compass**.

🗂️ Part 1: Indexing

1. Create a Collection

```
use Institute;  // or any database you want
db.createCollection("faculty");
```

---

2. Insert Some Documents

```
db.faculty.insertMany([
  { fid: 1, name: "Amit", dept: "CSE", salary: 55000 },
  { fid: 2, name: "Sneha", dept: "IT", salary: 60000 },
  { fid: 3, name: "Rahul", dept: "CSE", salary: 58000
},
  { fid: 4, name: "Anita", dept: "EXTC", salary: 65000
}
]);
```

---

3. Create Single Index

```
db.faculty.createIndex({ name: 1 });
```

👉 1 = ascending order index

---

4. Create Compound Index

```
db.faculty.createIndex({ dept: 1, salary: -1 });
```

👉 Index on multiple fields (department ascending, salary descending)

---

5. Create Unique Index

```
db.faculty.createIndex({ fid: 1 }, { unique: true });
```

---

6. Show Index Information

```
db.faculty.getIndexes();
```

---

7. Remove Index

```
db.faculty.dropIndex("dept_1_salary_-1");  // use the name returned by getIndexes()
```

🧮 Part 2: Aggregation

1. Create `Student` Collection and Insert Data

```
db.student.insertMany([
  { Rollno: 1, name: "Navin", subject: "DMSA", marks: 78 },
  { Rollno: 2, name: "Anusha", subject: "OSD", marks: 75 },
  { Rollno: 3, name: "Ravi", subject: "TOC", marks: 69 },
  { Rollno: 4, name: "Veena", subject: "TOC", marks: 70 },
  { Rollno: 5, name: "Pravini", subject: "OSD", marks:
80 },
  { Rollno: 6, name: "Reena", subject: "DMSA", marks:
50 },
  { Rollno: 7, name: "Geeta", subject: "CN", marks: 90
},
  { Rollno: 8, name: "Akash", subject: "CN", marks: 85
}
]);
```

---

2. Max Marks of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", MaxMarks: { $max:
"$marks" } } }
]);
```

---

3. Min Marks of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", MinMarks: { $min:
"$marks" } } }
]);
```

---

4. Sum of Marks of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", TotalMarks: { $sum:
"$marks" } } }
]);
```

---

5. Average Marks of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", AvgMarks: { $avg:
"$marks" } } }
]);
```

6. First Record of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", FirstRecord: { $first:
"$$ROOT" } } }
]);
```

7. Last Record of Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", LastRecord: { $last:
"$$ROOT" } } }
]);
```

8. Count Number of Records for Each Subject

```
db.student.aggregate([
  { $group: { _id: "$subject", Count: { $sum: 1 } } }
]);
```

✅ Summary

Concept	MongoDB Command
Create Collection	`db.createCollection()`
Insert Docs	`db.insertMany()`
Single Index	`db.createIndex({ field: 1 })`
Compound Index	`db.createIndex({ field1: 1, field2: -1 })`
Unique Index	`db.createIndex({ field: 1 }, { unique: true })`
Aggregation Functions	`$max`, `$min`, `$sum`, `$avg`, `$first`, `$last`, `$sum:1`

Chit 21

Collection “orderinfo“ which contains the documents given as below(Perform on Mongo Terminal) {

cust_id:123

cust_name:”abc”,

status:”A”,

price:250

}

i. Add “Age” field to the orderinfo collection

ii. Create the complex index on orderinfo collection and fire the queries and drop the duplicates. iii. Display the average price for each customer group by status

iv. Change the customer’s name whose status is “B”

Here’s the **MongoDB solution for Chit 21**, using commands you can run directly in the **Mongo shell (mongosh)** 👇

1☐ Create collection and insert documents

```
use ordersDB

db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 400 },
  { cust_id: 125, cust_name: "pqr", status: "A", price: 350 },
  { cust_id: 126, cust_name: "mno", status: "B", price: 200 }
])
```

2☐ Add “Age” field to the collection

```
db.orderinfo.updateMany({}, { $set: { Age: 30 } })
```

✅ Adds `Age` field with value `30` to all documents.

To verify:

```
db.orderinfo.find()
```

3☐ Create a complex (compound) index

We'll create a compound index on multiple fields — for example: `status`, `cust_name`, and `price`.

```
db.orderinfo.createIndex({ status: 1, cust_name: 1,
price: -1 })
```

✅ This index helps optimize queries filtering by `status` and sorting by price.

To check all indexes:

```
db.orderinfo.getIndexes()
```

4☐ Drop duplicate entries (if any)

If you accidentally have duplicate documents, you can remove them like this:

```
db.orderinfo.aggregate([
  { $group: { _id: { cust_id: "$cust_id" }, uniqueIds:
{ $addToSet: "$_id" }, count: { $sum: 1 } } },
  { $match: { count: { $gt: 1 } } }
]).forEach(doc => {
  doc.uniqueIds.shift(); // keep one
  db.orderinfo.deleteMany({ _id: { $in: doc.uniqueIds }
});
});
```

✅ This removes duplicate records based on `cust_id`.

5☐ Display the average price for each customer grouped by status

```
db.orderinfo.aggregate([
  { $group: { _id: "$status", avgPrice: { $avg:
"$price" } } }
])
```

✅ Output Example:

```
[
  { _id: "A", avgPrice: 300 },
  { _id: "B", avgPrice: 300 }
]
```

6☐ Change the customer’s name whose status is “B”

```
db.orderinfo.updateMany(
  { status: "B" },
  { $set: { cust_name: "updated_customer" } }
)
```

✅ To verify:

```
db.orderinfo.find()
```

7☐ Drop the created index (if needed)

```
db.orderinfo.dropIndex({ status: 1, cust_name: 1,
price: -1 })
```

Chit 22:

Collection “orderinfo“ which contains the documents given as below(Perform on Mongo Terminal) {

cust_id:123

cust_name:”abc”,

status:”A”,

price:250

}

i. Display the name of the customer having the price between 250 and 450

ii. Increment the price by 10 for cust_id: 123 and decrement the price by 5 for cust_id: 124 iii. Remove any one of the field from the orderinfo collection.
iv. Find the name of the customer whose status is either A or price is 250 or both.

Here’s the **complete MongoDB solution for Chit 22** — with proper commands that you can directly run in the **Mongo shell (`mongosh`)** 👇

1☐ Create database and collection

```
use orderDB

db.orderinfo.insertMany([
  { cust_id: 123, cust_name: "abc", status: "A", price: 250 },
  { cust_id: 124, cust_name: "xyz", status: "B", price: 400 },
  { cust_id: 125, cust_name: "pqr", status: "C", price: 500 },
  { cust_id: 126, cust_name: "mno", status: "A", price: 350 }
])
```

2☐ Display the name of the customer having the price between 250 and 450

```
db.orderinfo.find(
  { price: { $gte: 250, $lte: 450 } },
  { cust_name: 1, _id: 0 }
)
```

✅ **Output Example:**

```
{ cust_name: "abc" }
{ cust_name: "xyz" }
{ cust_name: "mno" }
```

3☐ Increment the price by 10 for cust_id: 123 and decrement by 5 for cust_id: 124

```
db.orderinfo.updateOne(
  { cust_id: 123 },
  { $inc: { price: 10 } }
)

db.orderinfo.updateOne(
  { cust_id: 124 },
  { $inc: { price: -5 } }
)
```

✅ **Check updated data:**

```
db.orderinfo.find()
```

4☐ Remove any one field from the orderinfo collection (e.g., remove "status")

```
db.orderinfo.updateMany({}, { $unset: { status: "" } })
```

✅ **Verify:**

```
db.orderinfo.find()
```

Now documents will no longer contain the `status` field.

5☐ Find the name of the customer whose status is either "A" or price is 250 or both

(If you removed `status` in the previous step, reinsert it for this query demo)

```
db.orderinfo.updateOne({ cust_id: 123 }, { $set: {
status: "A" } })
db.orderinfo.updateOne({ cust_id: 126 }, { $set: {
status: "A" } })
```

Then run:

```
db.orderinfo.find(
  { $or: [ { status: "A" }, { price: 250 } ] },
  { cust_name: 1, _id: 0 }
)
```

✅ **Output Example:**

```
{ cust_name: "abc" }
{ cust_name: "mno" }
```

Chit 23:

> **Using MySQL and JAVA conncetivity (Two Tier) perform the following queries**
>
> i. Create a table of employee details , Employee(SSN, Ename,state, salary)
>
> ii. Insert Records into Employee table
>
> iii. Retrieve the details based on Social Security Number(SSN).
>
> iv. Update the employee state from 'MH' to 'TN'
>
> v. Delete all the employees from ‘Gujrat’

ChatGPT said:

Here’s the **complete solution for Chit 23** — showing both **MySQL queries** and **Java (JDBC Two-Tier Application)** code for connectivity and CRUD operations 👇

🗄️ Step 1: MySQL Queries

1☐ Create the Database and Table

```
CREATE DATABASE companyDB;
USE companyDB;

CREATE TABLE Employee (
  SSN INT PRIMARY KEY,
  Ename VARCHAR(50) NOT NULL,
  state VARCHAR(50),
  salary DECIMAL(10,2)
);
```

2☐ Insert Records

```
INSERT INTO Employee VALUES
(101, 'Adarsh', 'MH', 45000),
(102, 'Rohit', 'Gujrat', 50000),
(103, 'Neha', 'MH', 48000),
(104, 'Sahil', 'TN', 55000);
```

3☐ Retrieve Details Based on SSN

```
SELECT * FROM Employee WHERE SSN = 102;
```

4☐ Update Employee State from 'MH' to 'TN'

```
UPDATE Employee
SET state = 'TN'
WHERE state = 'MH';
```

5☐ Delete All Employees from ‘Gujrat’

```
DELETE FROM Employee
WHERE state = 'Gujrat';
```

💻 Step 2: Java Code (JDBC Connectivity – Two-Tier Architecture)

Make sure you have the MySQL Connector JAR (e.g., `mysql-connector-j-8.0.xx.jar`) added to your classpath.

```
import java.sql.*;

public class EmployeeDB {
    public static void main(String[] args) {
        String url = 
"jdbc:mysql://localhost:3306/companyDB";
        String user = "root";  // your MySQL username
        String password = "root";  // your MySQL 
password

        try {
            // 1☐ Load Driver
            Class.forName("com.mysql.cj.jdbc.Driver");

            // 2☐ Establish Connection
            Connection con = 
DriverManager.getConnection(url, user, password);
            System.out.println("✅ Connected to MySQL 
Database!");

            Statement stmt = con.createStatement();

            // 3☐ Insert Data
            String insert = "INSERT INTO Employee 
VALUES (105, 'Priya', 'MH', 47000)";
            stmt.executeUpdate(insert);
            System.out.println("✅ Record Inserted!");

            // 4☐ Retrieve Employee by SSN
            int searchSSN = 102;
            ResultSet rs = stmt.executeQuery("SELECT * 
FROM Employee WHERE SSN=" + searchSSN);
            while (rs.next()) {
                System.out.println("\nSSN: " +
rs.getInt("SSN"));
                System.out.println("Name: " +
rs.getString("Ename"));
                System.out.println("State: " +
rs.getString("state"));
                System.out.println("Salary: " +
rs.getDouble("salary"));
            }

            // 5☐ Update Employee State from 'MH' to
'TN'
            String update = "UPDATE Employee SET
state='TN' WHERE state='MH'";
            stmt.executeUpdate(update);
            System.out.println("\n✅ Updated Employee
State from MH to TN!");

            // 6☐ Delete Employees from 'Gujrat'
            String delete = "DELETE FROM Employee WHERE
state='Gujrat'";
            stmt.executeUpdate(delete);
            System.out.println("✅ Deleted Employees
from Gujrat!");

            // 7☐ Close connection
            con.close();
            System.out.println("\n🔒 Connection
Closed.");

        } catch (Exception e) {
            System.out.println("❌ Error: " +
e.getMessage());
        }
    }
}
```

✅ Expected Output

```
✅ Connected to MySQL Database!
✅ Record Inserted!

SSN: 102
Name: Rohit
State: Gujrat
Salary: 50000.0

✅ Updated Employee State from MH to TN!
✅ Deleted Employees from Gujrat!
🔒 Connection Closed.
```

Would you like me to add **PreparedStatement** versions (safer for user input and SQL injection prevention) instead of plain `Statement`? It's often required in lab practicals.

Chit 24:

(Perform on MYSQL Terminal)

Emp(emp_id,ename, street, city)

works(emp_id,c_id,ename, cname, sal)

Company(c_id,cname, city)

Manager(mgr_id, mgrname)

i. Modify the database so that a particular company (eg. ABC) now in Pune

ii. Give all managers of Mbank a 10% raise. If salary is >20,000, give only 3% raise.

iii. Find out the names of all the employees who works in 'Bosch' company in city Pune iv. Delete all records in the works table for employees of a particular company (Eg, SBC Company) whose salary>50,000.

Here’s the **complete MySQL solution for Chit 24**, with all tables, sample data, and the required queries 👇

🧩 Step 1: Create Tables

```
CREATE DATABASE companyDB2;
USE companyDB2;

CREATE TABLE Emp (
  emp_id INT PRIMARY KEY,
  ename VARCHAR(50),
  street VARCHAR(100),
  city VARCHAR(50)
);

CREATE TABLE Company (
  c_id INT PRIMARY KEY,
  cname VARCHAR(50),
  city VARCHAR(50)
);

CREATE TABLE Manager (
  mgr_id INT PRIMARY KEY,
  mgrname VARCHAR(50)
);

CREATE TABLE Works (
  emp_id INT,
  c_id INT,
  ename VARCHAR(50),
  cname VARCHAR(50),
  sal DECIMAL(10,2),
  FOREIGN KEY (emp_id) REFERENCES Emp(emp_id),
  FOREIGN KEY (c_id) REFERENCES Company(c_id)
);
```

🧩 Step 2: Insert Sample Data

```
INSERT INTO Emp VALUES
(1, 'Adarsh', 'MG Road', 'Pune'),
(2, 'Rohit', 'FC Road', 'Mumbai'),
(3, 'Neha', 'JM Road', 'Pune'),
(4, 'Amit', 'Shivaji Nagar', 'Nashik');

INSERT INTO Company VALUES
(101, 'ABC', 'Mumbai'),
(102, 'Bosch', 'Pune'),
(103, 'Mbank', 'Delhi'),
(104, 'SBC', 'Pune');

INSERT INTO Manager VALUES
(201, 'Ravi'),
(202, 'Sonia'),
(203, 'Aman');

INSERT INTO Works VALUES
(1, 101, 'Adarsh', 'ABC', 25000),
(2, 102, 'Rohit', 'Bosch', 18000),
(3, 103, 'Neha', 'Mbank', 22000),
(4, 104, 'Amit', 'SBC', 55000);
```

⚙️ Step 3: Queries

i. Modify the database so that a particular company (e.g., ABC) is now in Pune

```
UPDATE Company
SET city = 'Pune'
WHERE cname = 'ABC';
```

✅ *ABC company's city will now be updated to Pune.*

---

ii. Give all managers of Mbank a 10% raise. If salary is >20,000, give only 3% raise.

Since manager salaries are stored in the `works` table (via company name `Mbank`):

```
UPDATE Works
SET sal = CASE
    WHEN sal > 20000 THEN sal * 1.03
    ELSE sal * 1.10
END
WHERE cname = 'Mbank';
```

✅ *Mbank managers (or employees) get salary updates based on condition.*

---

iii. Find out the names of all the employees who work in ‘Bosch’ company in city Pune

```
SELECT e.ename
FROM Emp e
JOIN Works w ON e.emp_id = w.emp_id
JOIN Company c ON w.c_id = c.c_id
WHERE c.cname = 'Bosch' AND c.city = 'Pune';
```

✅ *Displays names of employees in Bosch, Pune.*

---

iv. Delete all records in the works table for employees of a particular company (e.g., SBC) whose salary > 50,000

```
DELETE FROM Works
WHERE cname = 'SBC' AND sal > 50000;
```

✅ *Deletes records matching SBC with high salary.*

🧾 Optional: View Data

```
SELECT * FROM Emp;
SELECT * FROM Company;
SELECT * FROM Works;
```

Would you like me to also include the **trigger** to automatically log deleted employee data from the `works` table into an audit table (like a continuation to this chit)?

Chit 25:

(Perform on MYSQL Terminal)

Empl(e_no, e_name, post, pay_rate)

Position(pos_no, post)

Duty-alloc (pos_no, e_no, month,year, shift)

Implement the following SQL queries

i. Get duty allocation details for e_no 123 for the first shift in the month of April 2003 ii. Get the employees whose rate of pay is > or equal rate of pay of employees 'Sachin'. iii. Create a view for displaying minimum, maximum and average salary for all the posts. iv. Get count of different employees on each shift having post ‘manager’.

Here’s the **complete MySQL solution for Chit 25** with table creation, sample data, and all required queries 👇

🗄️ Step 1: Create Tables

```
CREATE DATABASE IF NOT EXISTS dutyDB;
USE dutyDB;

CREATE TABLE Empl (
    e_no INT PRIMARY KEY,
    e_name VARCHAR(50),
    post VARCHAR(50),
    pay_rate DECIMAL(10,2)
);

CREATE TABLE Position (
    pos_no INT PRIMARY KEY,
    post VARCHAR(50)
);
```

```
CREATE TABLE Duty_alloc (
    pos_no INT,
    e_no INT,
    month VARCHAR(20),
    year INT,
    shift VARCHAR(20),
    FOREIGN KEY (pos_no) REFERENCES Position(pos_no),
    FOREIGN KEY (e_no) REFERENCES Empl(e_no)
);
```

📝 Step 2: Insert Sample Data

```
INSERT INTO Empl VALUES
(123, 'Sachin', 'Manager', 50000),
(124, 'Rohit', 'Manager', 55000),
(125, 'Neha', 'Clerk', 30000),
(126, 'Amit', 'Clerk', 32000),
(127, 'Priya', 'Manager', 60000);

INSERT INTO Position VALUES
(1, 'Manager'),
(2, 'Clerk');

INSERT INTO Duty_alloc VALUES
(1, 123, 'April', 2003, 'First'),
(1, 124, 'April', 2003, 'Second'),
(2, 125, 'April', 2003, 'First'),
(2, 126, 'April', 2003, 'Second'),
(1, 127, 'April', 2003, 'First');
```

⚙️ Step 3: Queries

i. Get duty allocation details for e_no 123 for the first shift in April 2003

```
SELECT *
FROM Duty_alloc
WHERE e_no = 123 AND shift = 'First' AND month =
'April' AND year = 2003;
```

---

ii. Get employees whose pay_rate is >= pay_rate of employee 'Sachin'

```
SELECT *
FROM Empl
WHERE pay_rate >= (
    SELECT pay_rate FROM Empl WHERE e_name = 'Sachin'
);
```

---

iii. Create a view for displaying min, max, and average salary for all posts

```
CREATE VIEW Salary_Stats AS
SELECT post,
       MIN(pay_rate) AS Min_Salary,
       MAX(pay_rate) AS Max_Salary,
       AVG(pay_rate) AS Avg_Salary
FROM Empl
GROUP BY post;
```

✅ To view the results:

```
SELECT * FROM Salary_Stats;
```

---

iv. Get count of different employees on each shift having post 'Manager'

```
SELECT d.shift, COUNT(DISTINCT d.e_no) AS
Employee_Count
FROM Duty_alloc d
JOIN Empl e ON d.e_no = e.e_no
WHERE e.post = 'Manager'
GROUP BY d.shift;
```

This covers all **four queries** for Chit 25.