**1.LIBRARY DB  
1 Consider the following schema for a Library Database: BOOK (Book\_id, Title, Publisher\_Name, Pub\_Year) BOOK\_AUTHORS (Book\_id, Author\_Name) PUBLISHER (Name, Address, Phone) BOOK\_COPIES (Book\_id, Branch\_id, No-of\_Copies) BOOK\_LENDING (Book\_id, Branch\_id, Card\_No, Date\_Out, Due\_Date) LIBRARY\_BRANCH (Branch\_id, Branch\_Name, Address) Write SQL queries to:**

1. **Retrieve the details of all books in the library – id, title, name of publisher, authors, number of copies in each branch, etc.**
2. **Get the particular borrowers who have borrowed more than 3 books from Jan 2020 to Jun 2022.**
3. **Delete a book in BOOK table and Update the contents of other tables using DML statements.**
4. **Create the view for BOOK table based on year of publication and demonstrate its working with a simple query.**
5. **Create a view of all books and its number of copies which are currently available in the Library.**
6. **Demonstrate the usage of view creation**

Here are the SQL queries for your requirements:

**1. Retrieve book details (id, title, publisher, authors, copies, etc.)**

SELECT

B.Book\_id,

B.Title,

P.Name AS Publisher\_Name,

B.Pub\_Year,

GROUP\_CONCAT(A.Author\_Name) AS Authors,

C.Branch\_id,

C.No\_of\_Copies,

L.Branch\_Name,

L.Address

FROM BOOK B

JOIN PUBLISHER P ON B.Publisher\_Name = P.Name

JOIN BOOK\_AUTHORS A ON B.Book\_id = A.Book\_id

JOIN BOOK\_COPIES C ON B.Book\_id = C.Book\_id

JOIN LIBRARY\_BRANCH L ON C.Branch\_id = L.Branch\_id

GROUP BY B.Book\_id, C.Branch\_id;

**2. Get borrowers who borrowed more than 3 books between Jan 2020 to Jun 2022**

SELECT Card\_No, COUNT(Book\_id) AS Books\_Borrowed

FROM BOOK\_LENDING

WHERE Date\_Out BETWEEN '2020-01-01' AND '2022-06-30'

GROUP BY Card\_No

HAVING COUNT(Book\_id) > 3;

**3. Delete a book and update other tables**

**Delete book from BOOK table**

DELETE FROM BOOK WHERE Book\_id = 'B001'; -- Replace 'B001' with the actual Book\_id

**Update BOOK\_COPIES**

DELETE FROM BOOK\_COPIES WHERE Book\_id = 'B001';

**Update BOOK\_AUTHORS**

DELETE FROM BOOK\_AUTHORS WHERE Book\_id = 'B001';

**Update BOOK\_LENDING**

DELETE FROM BOOK\_LENDING WHERE Book\_id = 'B001';

**4. Create a view based on year of publication**

**Create View**

CREATE VIEW Book\_By\_Year AS

SELECT Book\_id, Title, Publisher\_Name, Pub\_Year

FROM BOOK;

**Demonstrate View**

SELECT \* FROM Book\_By\_Year WHERE Pub\_Year = 2020;

**5. Create a view of available books and their copies**

**Create View**

CREATE VIEW Available\_Books AS

SELECT B.Book\_id, B.Title, SUM(C.No\_of\_Copies) AS Total\_Copies

FROM BOOK B

JOIN BOOK\_COPIES C ON B.Book\_id = C.Book\_id

GROUP BY B.Book\_id, B.Title;

**Demonstrate View**

SELECT \* FROM Available\_Books WHERE Total\_Copies > 0;

**6. Demonstrate the usage of views**

**Example: Use Book\_By\_Year view to get books published after 2015**

SELECT \* FROM Book\_By\_Year WHERE Pub\_Year > 2015;

**Example: Use Available\_Books view to find books with more than 5 copies**

SELECT \* FROM Available\_Books WHERE Total\_Copies > 5;

These queries should help you retrieve, update, and manage the Library Database effectively! Let me know if you need any modifications. 😊📚

**2.ORDER DATABASE  
Consider the following schema for Order Database: SALESMAN (Salesman\_id, Name, City, Commission) CUSTOMER (Customer\_id, Cust\_Name, City, Grade,Salesman\_id) 6 91 ORDERS (Ord\_No, Purchase\_Amt, Ord\_Date, Customer\_id, Salesman\_id) Write SQL queries to:**

1. **Count the customers with grades above Bangalor’s average.**
2. **Find the name and numbers of all salesmen who had more than one customer.**
3. **List all salesmen and indicate those who have and don’t have customers in their cities (Use UNION operation).**
4. **Create a view that finds the salesman who has the customer with the highest order of a day.**
5. **Demonstrate the DELETE operation by removing salesman with id 1000. All his orders must also be deleted.**
6. **Create an index on ( Customer (id) ) to demonstrate the usage.**

Here are the SQL queries for your Order Database schema:

1. Count the customers with grades above Bangalore's average

SELECT COUNT(\*) AS Customer\_Count

FROM CUSTOMER

WHERE Grade > (SELECT AVG(Grade) FROM CUSTOMER WHERE City = 'Bangalore');

This query calculates the average grade of customers in Bangalore and counts how many customers have a grade above that.

2. Find salesmen who had more than one customer

SELECT S.Name, S.Salesman\_id, COUNT(C.Customer\_id) AS Num\_Customers

FROM SALESMAN S

JOIN CUSTOMER C ON S.Salesman\_id = C.Salesman\_id

GROUP BY S.Salesman\_id, S.Name

HAVING COUNT(C.Customer\_id) > 1;

This finds salesmen with more than one customer by grouping customers based on their assigned salesman.

3. List all salesmen and indicate if they have customers in their cities (Using UNION)

SELECT S.Salesman\_id, S.Name, S.City, 'Has Customers' AS Status

FROM SALESMAN S

JOIN CUSTOMER C ON S.Salesman\_id = C.Salesman\_id AND S.City = C.City

UNION

SELECT S.Salesman\_id, S.Name, S.City, 'No Customers' AS Status

FROM SALESMAN S

WHERE S.Salesman\_id NOT IN (SELECT DISTINCT Salesman\_id FROM CUSTOMER WHERE City = S.City);

This query uses UNION to combine two result sets: one for salesmen with customers in their city and another for those without.

4. Create a view for salesmen handling the highest order of the day

CREATE VIEW Top\_Salesman AS

SELECT O.Ord\_Date, O.Ord\_No, O.Purchase\_Amt, S.Salesman\_id, S.Name

FROM ORDERS O

JOIN SALESMAN S ON O.Salesman\_id = S.Salesman\_id

WHERE O.Purchase\_Amt = (

SELECT MAX(Purchase\_Amt)

FROM ORDERS

WHERE Ord\_Date = O.Ord\_Date

);

This view finds the salesman who handled the highest order amount for each day.

Demonstrating View Usage

SELECT \* FROM Top\_Salesman WHERE Ord\_Date = '2025-03-24';

5. Delete salesman with ID 1000 and remove their orders

Delete orders associated with the salesman first

DELETE FROM ORDERS WHERE Salesman\_id = 1000;

Now, delete the salesman

DELETE FROM SALESMAN WHERE Salesman\_id = 1000;

This ensures referential integrity by deleting dependent records first.

6. Create an index on Customer ID and demonstrate its usage

Creating an index

CREATE INDEX idx\_customer\_id ON CUSTOMER(Customer\_id);

Demonstrating usage: Fast lookup

EXPLAIN SELECT \* FROM CUSTOMER WHERE Customer\_id = 101;

Using EXPLAIN shows whether the index is being used in query execution.

These queries cover all requested operations efficiently. Let me know if you need modifications! **🚀**

**3.MOVIE DATABASE  
Consider the schema for Movie Database: ACTOR (Act\_id, Act\_Name, Act\_Gender) DIRECTOR (Dir\_id, Dir\_Name, Dir\_Phone) MOVIES (Mov\_id, Mov\_Title, Mov\_Year, Mov\_Lang, Dir\_id) MOVIE\_CAST (Act\_id, Mov\_id, Role) RATING (Mov\_id, Rev\_Stars) Write SQL queries to**

1. **List the titles of all movies directed by ‘Hitchcock’.**
2. **Find the movie names where one or more actors acted in two or more movies.**
3. **List all actors who acted in a movie before 2000 and also in a movie after 2020 (use JOIN operation).**
4. **Find the title of movies and number of stars for each movie that has at least one rating and find the highest number of stars that movie received. Sort the result by movie title.**
5. **Update rating of all movies directed by ‘Steven Spielberg’ to 5.**

Here are the SQL queries for your Movie Database schema:

**1. List the titles of all movies directed by ‘Hitchcock’**

SELECT M.Mov\_Title

FROM MOVIES M

JOIN DIRECTOR D ON M.Dir\_id = D.Dir\_id

WHERE D.Dir\_Name = 'Hitchcock';

This query finds all movies directed by Hitchcock by joining the MOVIES and DIRECTOR tables.

**2. Find movie names where one or more actors acted in two or more movies**

SELECT DISTINCT M.Mov\_Title

FROM MOVIES M

JOIN MOVIE\_CAST MC ON M.Mov\_id = MC.Mov\_id

WHERE MC.Act\_id IN (

SELECT Act\_id

FROM MOVIE\_CAST

GROUP BY Act\_id

HAVING COUNT(DISTINCT Mov\_id) >= 2

);

This finds movies where at least one actor has appeared in two or more different movies.

**3. List all actors who acted in a movie before 2000 and also in a movie after 2020**

SELECT DISTINCT A.Act\_Name

FROM ACTOR A

JOIN MOVIE\_CAST MC1 ON A.Act\_id = MC1.Act\_id

JOIN MOVIES M1 ON MC1.Mov\_id = M1.Mov\_id

JOIN MOVIE\_CAST MC2 ON A.Act\_id = MC2.Act\_id

JOIN MOVIES M2 ON MC2.Mov\_id = M2.Mov\_id

WHERE M1.Mov\_Year < 2000 AND M2.Mov\_Year > 2020;

This ensures the actor acted in at least one movie before 2000 and another after 2020.

**4. Find movie titles, number of stars for each movie, and highest rating received**

SELECT M.Mov\_Title, R.Rev\_Stars,

MAX(R.Rev\_Stars) OVER (PARTITION BY M.Mov\_id) AS Highest\_Stars

FROM MOVIES M

JOIN RATING R ON M.Mov\_id = R.Mov\_id

WHERE R.Rev\_Stars IS NOT NULL

ORDER BY M.Mov\_Title;

This retrieves the movie titles, their ratings, and the highest rating received, sorted by title.

**5. Update rating of all movies directed by ‘Steven Spielberg’ to 5**

UPDATE RATING

SET Rev\_Stars = 5

WHERE Mov\_id IN (

SELECT Mov\_id

FROM MOVIES

WHERE Dir\_id = (SELECT Dir\_id FROM DIRECTOR WHERE Dir\_Name = 'Steven Spielberg')

);

This updates all ratings to 5 for movies directed by Steven Spielberg.

Let me know if you need any modifications! 🎬🍿

**4.COLLEGE DATABASE**

**Consider the schema for College Database: STUDENT (USN, SName, Address, Phone, Gender) SEMSEC (SSID, Sem, Sec) CLASS (USN, SSID) SUBJECT (Subcode, Title, Sem, Credits) IAMARKS (USN, Subcode, SSID, Test1, Test2, Test3, FinalIA) Write SQL queries to**

1. **List all the student details studying in fifth semester ‘B’ section.**
2. **Compute the total number of male and female students in each semester and in each section.**
3. **Create a view of Event 1 marks of student USN ‘01JST IS ’ in all subjects.**
4. **Calculate the Final IA (average of best two test marks) and update the corresponding table for all students.**

**Categorize students based on the following criterion: If Final IA = 17 to 20 then CAT =‘Outstanding’ If Final IA = 12 to 16 then CAT = ‘Average’ If Final IA< 12 then CAT = ‘Weak’ Give these details only for 8th semester A, B, and C section students.**  
  
Here are the SQL queries for your College Database schema:

**1. List all student details studying in the fifth semester 'B' section**

SELECT S.\*

FROM STUDENT S

JOIN CLASS C ON S.USN = C.USN

JOIN SEMSEC SS ON C.SSID = SS.SSID

WHERE SS.Sem = 5 AND SS.Sec = 'B';

This query retrieves student details for those enrolled in the **5th semester, section B**.

**2. Compute the total number of male and female students in each semester and section**

SELECT SS.Sem, SS.Sec, S.Gender, COUNT(S.USN) AS Total\_Students

FROM STUDENT S

JOIN CLASS C ON S.USN = C.USN

JOIN SEMSEC SS ON C.SSID = SS.SSID

GROUP BY SS.Sem, SS.Sec, S.Gender

ORDER BY SS.Sem, SS.Sec, S.Gender;

This query groups students by **semester, section, and gender**, counting the number of students in each category.

**3. Create a view of Event 1 marks of student USN ‘01JST IS’ in all subjects**

CREATE VIEW Event1\_Marks AS

SELECT I.USN, I.Subcode, S.Title, I.Test1

FROM IAMARKS I

JOIN SUBJECT S ON I.Subcode = S.Subcode

WHERE I.USN = '01JST IS';

This view stores **Test 1 (Event 1) marks** for a specific student in all subjects.

**Demonstrating View Usage**

SELECT \* FROM Event1\_Marks;

**4. Calculate Final IA (average of best two test marks) and update the table**

UPDATE IAMARKS

SET FinalIA = (Test1 + Test2 + Test3 - LEAST(Test1, Test2, Test3)) / 2;

This query calculates the **Final IA** as the average of the best two test marks and updates the IAMARKS table.

**5. Categorize students based on Final IA for 8th semester A, B, and C sections**

SELECT S.USN, S.SName, SS.Sem, SS.Sec, I.FinalIA,

CASE

WHEN I.FinalIA BETWEEN 17 AND 20 THEN 'Outstanding'

WHEN I.FinalIA BETWEEN 12 AND 16 THEN 'Average'

WHEN I.FinalIA < 12 THEN 'Weak'

ELSE 'Unknown'

END AS Category

FROM STUDENT S

JOIN CLASS C ON S.USN = C.USN

JOIN SEMSEC SS ON C.SSID = SS.SSID

JOIN IAMARKS I ON S.USN = I.USN

WHERE SS.Sem = 8 AND SS.Sec IN ('A', 'B', 'C');

This categorizes students based on their **Final IA** scores for **8th semester, sections A, B, and C**.

**5.COMPANY DB  
Consider the schema for Company Database: EMPLOYEE (SSN, Name, Address, Sex, Salary, SuperSSN, DNo) DEPARTMENT (DNo, DName, MgrSSN, MgrStartDate) DLOCATION (DNo,DLoc) PROJECT (PNo, PName, PLocation, DNo) WORKS\_ON (SSN, PNo, Hours) Write SQL queries to**

1. **Make a list of all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.**
2. **Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise. 6 92**
3. **Find the sum of the salaries of all employees of the ‘Accounts’ department, as well as the maximum salary, the minimum salary, and the average salary in this department**

**Retrieve the name of each employee who works on all the projects controlled by department number. 5 (use NOT EXISTS operator). For each department that has more than five employees, retrieve the department number and the number of its employees who are making more than Rs.6, 00,000.**Here are the SQL queries for your Company Database schema:

1. List all project numbers for projects that involve an employee whose last name is ‘Scott’, either as a worker or as a manager of the department that controls the project.

SELECT DISTINCT W.PNo

FROM WORKS\_ON W

JOIN EMPLOYEE E ON W.SSN = E.SSN

WHERE E.Name LIKE '%Scott'

UNION

SELECT DISTINCT P.PNo

FROM PROJECT P

JOIN DEPARTMENT D ON P.DNo = D.DNo

JOIN EMPLOYEE E ON D.MgrSSN = E.SSN

WHERE E.Name LIKE '%Scott';

The first part retrieves project numbers where any employee (worker) has 'Scott' as their last name.

The second part retrieves project numbers where Scott is the manager of the department controlling the project.

2. Show the resulting salaries if every employee working on the ‘IoT’ project is given a 10 percent raise.

SELECT E.SSN, E.Name, E.Salary AS Old\_Salary,

E.Salary \* 1.10 AS New\_Salary

FROM EMPLOYEE E

JOIN WORKS\_ON W ON E.SSN = W.SSN

JOIN PROJECT P ON W.PNo = P.PNo

WHERE P.PName = 'IoT';

This retrieves the salaries before and after a 10% increase for employees working on the IoT project.

3. Find the sum of salaries, maximum salary, minimum salary, and average salary of all employees in the ‘Accounts’ department.

SELECT SUM(E.Salary) AS Total\_Salary,

MAX(E.Salary) AS Max\_Salary,

MIN(E.Salary) AS Min\_Salary,

AVG(E.Salary) AS Avg\_Salary

FROM EMPLOYEE E

JOIN DEPARTMENT D ON E.DNo = D.DNo

WHERE D.DName = 'Accounts';

This calculates total, maximum, minimum, and average salary for employees in the Accounts department.

4. Retrieve the name of each employee who works on all the projects controlled by department number 5 (use NOT EXISTS).

SELECT E.Name

FROM EMPLOYEE E

WHERE NOT EXISTS (

SELECT P.PNo

FROM PROJECT P

WHERE P.DNo = 5

EXCEPT

SELECT W.PNo

FROM WORKS\_ON W

WHERE W.SSN = E.SSN

);

EXCEPT finds projects that belong to department 5 but an employee has not worked on.

NOT EXISTS ensures only employees who worked on all department 5 projects are retrieved.

5. For each department with more than five employees, retrieve the department number and the number of employees earning more than Rs. 6,00,000.

SELECT E.DNo, COUNT(E.SSN) AS Num\_Employees

FROM EMPLOYEE E

WHERE E.Salary > 600000

GROUP BY E.DNo

HAVING COUNT(E.SSN) > 5;

This retrieves departments with more than 5 employees who earn above ₹6,00,000.

These queries cover all the required operations in the Company Database. Let me know if you need any modifications! 😊

**6.CALL DETAIL SCHEMA DB  
Consider the schema of the call detail table to partitioned primary index:   
CREATE TABLE calldetail   
( phone\_number DECIMAL(10) NOT NULL,  
 call\_start TIMESTAMP, call\_duration INTEGER,  
 call\_description VARCHAR(30))  
 PRIMARY INDEX (phone\_number, call\_start);  
 Demonstrate the query against this table be optimized by partitioning its primary index using partitioning techniques.**

To optimize queries on the calldetail table using partitioning techniques, we can use Range Partitioning or Hash Partitioning based on the phone\_number and call\_start columns.

1. Creating a Partitioned Primary Index

We can partition the table by RANGE on call\_start to improve query performance for time-based searches.

CREATE TABLE calldetail (

phone\_number DECIMAL(10) NOT NULL,

call\_start TIMESTAMP NOT NULL,

call\_duration INTEGER,

call\_description VARCHAR(30),

PRIMARY KEY (phone\_number, call\_start)

)

PARTITION BY RANGE (YEAR(call\_start)) (

PARTITION p1 VALUES LESS THAN (2020),

PARTITION p2 VALUES LESS THAN (2022),

PARTITION p3 VALUES LESS THAN (2024),

PARTITION p4 VALUES LESS THAN (2026)

);

📌 Explanation:

* This partitions the table by year based on call\_start, so queries filtering by time range are faster.
* Each partition contains data for a specific year range, reducing the number of rows scanned.

2. Optimizing Queries Using Partitioning

Optimized Query to Retrieve Calls Between 2022 and 2024

SELECT \* FROM calldetail

WHERE call\_start BETWEEN '2022-01-01' AND '2024-12-31';

👉 This query scans only relevant partitions (p2 and p3) instead of the entire table.

3. Hash Partitioning for Load Balancing

If queries are frequently performed on phone\_number, we can use HASH Partitioning instead:

CREATE TABLE calldetail (

phone\_number DECIMAL(10) NOT NULL,

call\_start TIMESTAMP NOT NULL,

call\_duration INTEGER,

call\_description VARCHAR(30),

PRIMARY KEY (phone\_number, call\_start)

)

PARTITION BY HASH (phone\_number) PARTITIONS 4;

📌 Benefits:

* Distributes phone\_number values across 4 partitions, reducing contention on a single partition.
* Useful for queries where phone\_number is frequently used in WHERE conditions.

4. Checking Partition Usage

To verify that partitioning improves performance, run:

EXPLAIN SELECT \* FROM calldetail WHERE call\_start BETWEEN '2022-01-01' AND '2024-12-31';

* This will show whether only specific partitions are scanned instead of the full table.

Conclusion

By using RANGE Partitioning (for time-based queries) or HASH Partitioning (for balancing loads across phone numbers), queries can be optimized for faster execution and better performance.

Would you like a different partitioning strategy based on query patterns? 😊