1. **Create table EMPLOYEE with attributes EMPID, ENAME, DESIGNATION, HIREDATE, BASICSAL, DEPTID.**
2. **Insert 5 rows into EMPLOYEE table.**
3. **Write SQL command to display the records inserted.**
4. **Alter the table EMPLOYEE by including the new attributes DA, HRA, PF, GROSSSALARY and NETSALARY. Calculate DA, HRA, PF, GROSSSALARY, NETSALARY by using the formula and update the table.**
5. **Display details of employees whose net salary is greater than 20000.**
6. **Display ENAME, DESIGNATION and GROSSSALARY of employees whose PF is greater than 3000.**

**Create database COMPANY**

mysql> create database company;

Query OK, 1 row affected (0.15 sec)

**Use database**

mysql> use company;

Database changed

**Create table**

mysql> CREATE TABLE EMPLOYEE (EMPID VARCHAR(10), ENAME VARCHAR(20), DESIGNATION VARCHAR(20), HIREDATE DATE, BASICSAL FLOAT, DEPTID SMALLINT, CONSTRAINT PK\_EMPLOYEE PRIMARY KEY (EMPID));

1. **Insert 5 rows into EMPLOYEE table.**

mysql> INSERT INTO EMPLOYEE VALUES ('E100', 'SANTHOSH', 'MANAGER', '2016-08-22', 25000, 1);

mysql> INSERT INTO EMPLOYEE VALUES ('E101', 'ANAND', 'SALES EXECUTIVE', '2018-07-14', 34000, 2);

mysql> INSERT INTO EMPLOYEE VALUES ('E102', 'DEEPTHI', 'SUPERVISOR', '2015-08-04', 34000, 2);

mysql> INSERT INTO EMPLOYEE VALUES ('E103', 'RAJU', 'CLERK', '2015-05-19', 11000, 3);

mysql> INSERT INTO EMPLOYEE VALUES ('E104', 'KIRAN', 'HELP DESK', '2019-09-25', 10000, 1);

1. **Write SQL command to display the records inserted.**

**mysql> SELECT \* FROM EMPLOYEE;**

**ALTER TABLE**

alter table employee add DA FLOAT, ADD HRA FLOAT, ADD PF FLOAT, ADD GROSSSALARY FLOAT, ADD NETSALARY FLOAT;

1. **Alter the table EMPLOYEE by including the new attributes DA, HRA, PF, GROSSSALARY and NETSALARY. Calculate DA, HRA, PF, GROSSSALARY, NETSALARY by using the formula and update the table.**

**CALCULATE DA, HRA, PF**

**mysql>** UPDATE EMPLOYEE SET DA=0.8 \* BASICSAL, HRA=0.12 \* BASICSAL, PF=0.1 \* BASICSAL;

**CALCULATE GROSSSALARY**

**mysql>** UPDATE EMPLOYEE SET GROSSSALARY= BASICSAL + DA + HRA + PF;

**CALCULATE NETSALARY**

**mysql>** UPDATE EMPLOYEE SET NETSALARY= GROSSSALARY - PF;

1. **Display details of employees whose net salary is greater than 20000.**

mysql> SELECT \* FROM EMPLOYEE WHERE NETSALARY > 20000;

1. **Display the ENAME, DESIGNATION, GROSSSALARY and PF whose PF is greater than 3000.**

mysql> SELECT ENAME, DESIGNATION, GROSSSALARY, PF FROM EMPLOYEE

WHERE PF > 3000;

**2. Create table EMPLOYEE with fields EMPID, ENAME, DESIGNATION, HIREDATE, SALARY, DEPTID, write SQL commands for the following**

**a. Display the name and salary of the employees whose salary is greater than or equal to 40000 and less than or equal to 50000.**

**b. List the employee names and their designation who have joined after 30th June**

**2018 and before 31st December 2018.**

**c) Display the names of employees working in department 1, 2, or 4.**

**d) Display the details of employees in ascending order based on EMPID.**

**e) Display the details of employees in descending order based on SALARY**

**f) Display the employee names where employee name starts with S, has got S, and ends with S from EMPLOYEE table. Write separate commands for each case.**

**g) Search for employee names starting with 'A' or 'B':**

1. **Search for employee names containing 'ann' or 'anne':**

**Create table**

mysql> CREATE TABLE EMPLOYEE (EMPID VARCHAR(10), ENAME VARCHAR(20), DESIGNATION VARCHAR(20), HIREDATE DATE, SALARY FLOAT, DEPTID INT, CONSTRAINT PK\_EMPLOYEE PRIMARY KEY (EMPID));

**Insert 5 records**

**mysql>** INSERT INTO EMPLOYEE VALUES ('E100', 'SANTHOSH', 'MANAGER', '2016-08-22', 25000, 1);

**mysql>** INSERT INTO EMPLOYEE VALUES ('E101', 'ANAND', 'SALES EXECUTIVE', '2018-07-14', 34000, 2);

**mysql>** INSERT INTO EMPLOYEE VALUES ('E102', 'DEEPTHI', 'SUPERVISOR', '2015-08-04', 64000, 2);

**mysql>** INSERT INTO EMPLOYEE VALUES ('E103', 'RAJU', 'CLERK', '2015-05-19', 41000, 3);

**mysql>** INSERT INTO EMPLOYEE VALUES ('E104', 'KIRAN', 'HELP DESK', '2019-09-25', 30000, 4);

**mysql>** SELECT \* FROM EMPLOYEE;

1. **Display the name and salary of the employees whose salary is greater than or equal to 20000 and less than or equal to 30000.**

**mysql>** SELECT ENAME, SALARY

-> FROM EMPLOYEE

-> WHERE SALARY BETWEEN 20000 AND 30000;

**b) List the employee names and their designation who have joined on or after 30th June**

**2018 and on or before 31st December 2018.**

**mysql>** SELECT ENAME, DESIGNATION

-> FROM EMPLOYEE

-> WHERE HIREDATE BETWEEN '2018-06-30' AND '2018-12-31';

**c) Display the names of employees and their department ids working in departments ‘1’, ‘2’, and ‘3’.**

**mysql>** SELECT ENAME, DEPTID FROM EMPLOYEE WHERE DEPTID IN (1,2,3);

**d) Display the details of employees in ascending order based on EMPID**

**mysql> SELECT \* FROM EMPLOYEE ORDER BY EMPID ASC;**

1. **Display the details of employees in descending order based on SALARY**

**mysql> SELECT \* FROM EMPLOYEE ORDER BY SALARY DESC;**

1. **Display the employee names, department id where employee name starts with ‘S’, has got ‘S’, and ends with ‘S’ from EMPLOYEE table. Write separate commands for each case.**

**Display the employee names, department id where employee name starts with ‘S’**

**mysql>** SELECT ENAME,DEPTID FROM EMPLOYEE WHERE ENAME LIKE 'S%';

**Display the employee names, department id where employee name has got ‘S’**

**mysql>** SELECT ENAME,DEPTID FROM EMPLOYEE WHERE ENAME LIKE '%S%';

**Display the employee names, DEPTID where employee name ends with “S’**

**mysql>** SELECT ENAME, DEPTID FROM EMPLOYEE WHERE ENAME LIKE '%S';

**g) Search for employee names starting with 'A' or 'B':**

**mysql>** SELECT \* FROM EMPLOYEE

WHERE ENAME REGEXP '^[AB]';

1. **Search for employee names containing 'ann' or 'anne':**

**mysql>** SELECT \* FROM EMPLOYEE

WHERE ENAME REGEXP 'ann(e)?';

1. **Create table EMPLOYEE with fields EMPID, ENAME, DESIGNATION, HIREDATE, SALARY, DEPTID, write SQL commands for the following.**
2. **Display the total salary of all the employees.**
3. **Display the average, maximum and minimum salary of employees.**
4. **Display the total salary of all employees whose salary is greater than 30000.**
5. **Display the count of ‘MANAGERS’ in the organisation.**
6. **Retrieve the total number of employees in the company.**
7. **For each department, retrieve the department id, the number of employees in the department, and their average salary.**
8. **Find the total salary by department.**
9. **Display the number of employees in each department which have five or more employees sorted in the descending order of number of employees.**
10. **Find departments with a total salary greater than 100,000.**

**Note: For query 3 use the EMPLOYEE TABLE CREATED IN Query 2.**

1. **Display the total salary of all the employees.**

SELECT SUM(SALARY) AS TOTAL\_SALARY FROM EMPLOYEE;

1. **Display the average, maximum and minimum salary of employees.**

SELECT AVG(SALARY) AS AVERAGE\_SALARY, MIN(SALARY) AS MINIMUM\_SALARY, MAX(SALARY) AS MAXIMUM\_SALARY

From Employee;

1. **Display the total salary of all employees whose salary is greater than 30000**

SELECT SUM(SALARY) AS TOTAL\_SALARY FROM EMPLOYEE

WHERE SALARY > 30000;

1. **Display the count of ‘MANAGERS’ in the organisation.**

SELECT COUNT(DESIGNATION) FROM EMPLOYEE

WHERE DESIGNATION='MANAGER';

1. **Retrieve the total number of employees in the company**

SELECT COUNT (\*)

FROM EMPLOYEE;

Or

SELECT COUNT (EMPID)

FROM EMPLOYEE;

1. **For each department, retrieve the department number, the number of employees in the department, and their average salary.**

SELECT Dno, COUNT (\*), AVG (Salary)

FROM EMPLOYEE

GROUP BY Deptid;

1. **Find the total salary by department:**

SELECT DNO, SUM(SALARY) AS Total\_Salary

FROM EMPLOYEE

GROUP BY Deptid;

1. **Display the number of employees in each department which have five or more employees sorted in the descending order of number of employees.**

SELECT DNO, COUNT(EMPID) AS NO\_OF\_EMPLOYEES

FROM EMPLOYEE

GROUP BY DNO

HAVING COUNT(EMPID) >= 5

ORDER BY COUNT(EMPID) DESC;

1. **Find departments with a total salary greater than 100,000.**

SELECT DNO, SUM(SALARY) AS Total\_Salary

FROM EMPLOYEE

GROUP BY DNO

HAVING SUM(SALARY) > 100000;

1. Create table BORROWER with attributes BORROWER\_ID, CUSTSOMER\_NAME, and LOAN\_NUMBER and insert few records.

Create table LOAN with attributes LOAN\_NUMBER, BRANCH\_NAME, AMOUNT, and BORROWER\_ID and insert few records.

Write queries to demonstrate:

* 1. Inner join.
  2. Left outer join.
  3. Right outer join.
  4. Full outer join.

**Creating BORROWER and LOAN tables and inserting records.**

CREATE TABLE BORROWER(BORROWER\_ID VARCHAR(20), CUSTSOMER\_NAME VARCHAR(50), LOAN\_NUMBER VARCHAR(20), CONSTRAINT PK\_BORROWER PRIMARY KEY(BORROWER\_ID));

INSERT INTO BORROWER VALUES (‘B301’,'SOWMYA','L102');

INSERT INTO BORROWER VALUES (‘B100’,'BABU','L100');

INSERT INTO BORROWER VALUES (‘B401’,'RAGHU','L105');

CREATE TABLE LOAN(LOAN\_NUMBER VARCHAR(20), BRANCH\_NAME VARCHAR(20), AMOUNT FLOAT, BORROWER\_ID VARCHAR(20), CONSTRAINT PK\_KEY PRIMARY KEY(LOAN\_NUMBER), CONSTRAINT FK\_LOAN FOREIGN KEY(BORROWER\_ID) REFERENCES BORROWER(BORROWER\_ID));

INSERT INTO LOAN VALUES ('L100','JAYANAGAR', 600000, ‘B100’);

INSERT INTO LOAN VALUES ('L108','BTM LAYOUT', 800000, null);

INSERT INTO LOAN VALUES ('L102','ATTIBELE', 300000, ‘B301’);

1. **Inner join.**

SELECT \*

FROM LOAN L JOIN

BORROWER B

ON L.LOAN\_NUMBER = B.LOAN\_NUMBER;

1. **Left outer join.**

SELECT \*

FROM LOAN L left JOIN

BORROWER B

ON L.LOAN\_NUMBER = B.LOAN\_NUMBER;

1. **Right outer join.**

SELECT \*

FROM LOAN L right JOIN

BORROWER B

ON L.LOAN\_NUMBER = B.LOAN\_NUMBER;

1. **Full outer join.**

SELECT \*

FROM LOAN L left JOIN

BORROWER B

ON L.LOAN\_NUMBER = B.LOAN\_NUMBER

UNION

SELECT \*

FROM LOAN L right JOIN

BORROWER B

ON L.LOAN\_NUMBER = B.LOAN\_NUMBER;

1. **Python program to demonstrate Database access using Python.**
2. Establish connection to MYSQL database using mysql.connector.connect().
3. Create a Cursor object.
4. Create table **Employee** with the fields Ename, Empid, Designation, Hiredate, Salary, Dno and insert few records.
5. Fetch all rows from the Employee table.
6. Fetch employee details whose salary is greater than Rs. 60000.
7. **Count the number of employees in a specific department (say** Dno = 101).

#Establish connection to MYSQL database

import mysql.connector

try:

conn = mysql.connector.connect(

host="localhost",

user="root",

password="SamAtt@1",

database="company"

)

#Create a Cursor object

if conn.is\_connected():

cursor = conn.cursor()

# SQL query to create the Employee table

create\_table\_query = """

CREATE TABLE IF NOT EXISTS Emp (

Ename VARCHAR(100),

Empid VARCHAR(10) PRIMARY KEY,

Designation VARCHAR(50),

Hiredate DATE,

Salary DECIMAL(10, 2),

Dno INT

)

"""

# Execute the query to create the table

cursor.execute(create\_table\_query)

# Commit the changes

conn.commit()

# Insert data into the Employee table

insert\_query = """

INSERT INTO Emp (Ename, Empid, Designation, Hiredate, Salary, Dno)

VALUES (%s, %s, %s, %s, %s, %s)

"""

# Sample data to insert

employee\_data = [

('John Doe', 'E001', 'Manager', '2020-06-15', 70000, 101),

('Jane Smith', 'E002', 'Developer', '2019-09-23', 60000, 102),

('Mike Johnson', 'E003', 'Analyst', '2021-02-01', 50000, 101)

]

# Insert each employee record

for employee in employee\_data:

cursor.execute(insert\_query,employee)

# Commit the transaction

conn.commit()

# Query to select all rows

select\_query = "SELECT \* FROM Emp"

# Execute the query

cursor.execute(select\_query)

# Fetch all the results

result = cursor.fetchall()

print('Employee details')

# Print the results

for row in result:

print(row)

# Query to filter employees with salary greater than 60000

filter\_query = "SELECT \* FROM Emp WHERE Salary > 60000"

# Execute the query

cursor.execute(filter\_query)

# Fetch the results

result = cursor.fetchall()

print('Employees whose salary is more than 60000')

# Print the results

for row in result:

print(row)

# Query to count employees in department 101

count\_query = "SELECT COUNT(\*) FROM Emp WHERE Dno = 101"

# Execute the query

cursor.execute(count\_query)

# Fetch the result

result = cursor.fetchone()

# Print the count

print("Number of employees in Department 101:", result[0])

except mysql.connector.Error as e:

print(f"Error: '{e}'")

finally:

if conn.is\_connected():

cursor.close()

conn.close()

print("MySQL connection is closed.")

1. **Create a collection called “posts” containing fields such as title, body, category, likes, tags, date. Create documents using insertOne() and insertMany() methods. Write queries for the following:**
2. **To retrieve all documents**
3. **Find all documents where the value for "category" is “Database”**
4. **Find all documents with more than 150 likes**
5. **Find all posts where “likes” is between 100 and 300**
6. **Find documents where category is "Database" or "Programming"**
7. **To retrieve documents where either the likes are more than 100 or the post has a tag of "JavaScript"**
8. **Find posts in the "Programming" category with more than 100 Likes**

**Create documents using insertOne() and insertMany()**

>db.posts.insertOne({

title: "Introduction to MongoDB",

body: "MongoDB is a NoSQL database that stores data in JSON-like format...",

category: "Database",

likes: 120,

tags: ["MongoDB", "NoSQL", "Database"],

date: new Date("2024-01-15")

});

>db.posts.insertMany([

{

title: "Understanding Indexes in MongoDB",

body: "Indexes are special data structures in MongoDB...",

category: "Database",

likes: 85,

tags: ["MongoDB", "Indexes", "Database"],

date: new Date("2024-01-20")

},

{

title: "JavaScript for Beginners",

body: "JavaScript is a versatile language for web development...",

category: "Programming",

likes: 200,

tags: ["JavaScript", "Web Development", "Programming"],

date: new Date("2024-01-22")

},

{

title: "Machine Learning Basics",

body: "Machine learning is a field of artificial intelligence...",

category: "AI",

likes: 150,

tags: ["Machine Learning", "AI", "Data Science"],

date: new Date("2024-01-25")

}

]);

1. **To retrieve all documents**

>db.posts.find();

1. **Find all documents where the value for "category" is “Database”**

**>** db.posts.find({category:"Database"});

1. **Find all documents with more than 150 likes**

**>** db.posts.find({likes: { $gt: 150 }});

1. **Find all posts where “likes” is between 100 and 300**

**>** db.posts.find({"likes" : {"$lt" : 300, "$gt" : 100}})

1. **Find documents where category is "Database" or "Programming"**

**>** db.posts.find({ $or: [{ category: "Database" }, { category: "Programming" }] });

1. **To retrieve documents where either the likes are more than 100 or the post has a tag of "JavaScript"**

**>** db.posts.find({

$or: [

{ likes: { $gt: 100 } },

{ tags: "JavaScript" }

]

});

1. **Find posts in the "Programming" category with more than 100 Likes**

**>** db.posts.find({

$and: [

{ category: "Programming" },

{ likes: { $gt: 100 } }

]

});

1. **Create a collection called “posts” containing fields such as title, author, content, category, likes, tags, date. Create a compound index on the posts collection using the fields author (ascending) and date (descending). Write a query to demonstrate how this index improves performance for filtering by author and sorting by date.**
2. **Create the ‘posts’ Collection and Insert Data**

>db.posts.insertMany([

{

title: " Understanding Indexes in MongoDB ",

author: "Sowmya",

content: " Indexes are special data structures in MongoDB...",

category: "Tech",

likes: 50,

tags: ["JavaScript", "MongoDB"],

date: new Date("2024-11-01")

},

{

title: " JavaScript for Beginners ",

author: "Ram",

content: " JavaScript is a versatile language for web development...",

category: "Health",

likes: 30,

tags: ["Fitness", "Wellness"],

date: new Date("2024-11-02")

},

{

title: " Machine Learning Basics ",

author: "Raghu",

content: " Machine learning is a field of artificial intelligence...",

category: "Tech",

likes: 70,

tags: ["Programming", "Database"],

date: new Date("2024-11-03")

},

{

title: "Post 4",

author: "Charlie",

content: "Content of post 4",

category: "Food",

likes: 40,

tags: ["Cooking", "Recipes"],

date: new Date("2024-01-04")

}

]);

1. **Query for Filtering by author and Sorting by date (Before creating index)**

**>** db.posts.find({ author: "Charlie" }).sort({ date: -1 });

1. **Query for Filtering by ‘author’ and sorting by ‘date’ (before creating index)**

**>** db.posts.find({ author: "Charlie" }).sort({ date: -1 }).explain("executionStats");

1. **Create a compound index on the ‘posts’ collection using the fields ‘author’ (ascending) and ‘date’ (descending)**

**>**db.posts.createIndex({ author: 1, date: -1 });

1. **Query for Filtering by ‘author’ and sorting by ‘date’ (After creating index)**

> db.posts.find({ author: "Charlie" }).sort({ date: -1 }).explain("executionStats");

**Use the .explain() method to verify that the query uses the compound index.**

This will show the **winning plan** and indicate the use of the compound index. Look for:

* + stage: "IXSCAN" (Index Scan).
  + Reduced totalKeysExamined and documentsExamined compared to a collection scan.

**Explanation of Performance Improvement:**

* Without the index, MongoDB would perform a **COLLSCAN** (collection scan), examining all documents in the collection.
* With the compound index, MongoDB uses an **IXSCAN**, filtering efficiently by ‘author’ and retrieving documents in the desired ‘date’ order without needing an additional sort stage.
* This reduces query execution time and resource usage, particularly for large datasets.

1. **Create a collection called “posts” containing fields such as title, author, content, category, likes, tags, date. Calculate the total number of likes for each author in the posts collection using MapReduce.**
   1. **Create the ‘posts’ Collection and Insert Data (Refer Query 7)**
   2. **Define the Map Function**: The map function emits the ‘author’ as the key and the ‘likes’ as the value.

**const mapFunction = function () {**

**emit(this.author, this.likes);**

**};**

* 1. **Define the Reduce Function**: The reduce function sums up the ‘likes’ for each ‘author’.

const reduceFunction = function (key, values) {

return Array.sum(values);

};

* 1. **Run MapReduce**: Use the mapReduce method to apply the map and reduce functions.

db.posts.mapReduce(

mapFunction,

reduceFunction,

{

out: "author\_likes\_totals" // Output the results to a new collection.

}

);

* 1. **View the Results**: The results are stored in a new collection named author\_likes\_totals. Query this collection to see the output:

db.author\_likes\_totals.find();

### Alternative with Aggregation Framework:

MapReduce is powerful but slower than the aggregation framework and is also deprecated in MongoDB. The equivalent aggregation query is:

db.posts.aggregate([

{ $group: { \_id: "$author", totalLikes: { $sum: "$likes" } } }

]);