-- Create a new database if not exists

CREATE DATABASE IF NOT EXISTS mydatabase;

-- Switch to the created database

USE mydatabase;

-- Create a table named 'employee' with columns id, name, dept\_no, and salary

CREATE TABLE IF NOT EXISTS employee (

id INT,

name STRING,

dept\_no INT,

salary INT

);

-- Display the table structure

DESCRIBE employee;

**Q.Hive Partitioning**

**Step 1: Create a Database (if not already created)**

CREATE DATABASE IF NOT EXISTS mydatabase;

USE mydatabase;

**Step 2: Create a Partitioned Table**

CREATE TABLE IF NOT EXISTS employee\_partitioned (

id INT,

name STRING,

salary INT

)

PARTITIONED BY (dept\_no INT);

In this example, the employee\_partitioned table is created with columns id, name, and salary. The table is partitioned by the dept\_no column.

**Step 3: Load Data into the Partitioned Table**

-- Load data into partition with dept\_no = 1

INSERT INTO TABLE employee\_partitioned PARTITION (dept\_no=1) VALUES (1, 'John', 50000);

-- Load data into partition with dept\_no = 2

INSERT INTO TABLE employee\_partitioned PARTITION (dept\_no=2) VALUES (2, 'Alice', 60000);

You can load data into specific partitions using the PARTITION clause.

**Step 4: Query Data from the Partitioned Table**

-- Query data from partition with dept\_no = 1

SELECT \* FROM employee\_partitioned WHERE dept\_no = 1;

-- Query data from partition with dept\_no = 2

SELECT \* FROM employee\_partitioned WHERE dept\_no = 2;

**Q. Hive Built-In Operators**

### Assume the Employee Table (emp) is Created as Follows:

CREATE TABLE IF NOT EXISTS emp (

id INT,

name STRING,

salary INT,

role STRING

);

-- Sample data

INSERT INTO emp VALUES

(1, 'John', 60000, 'Manager'),

(2, 'Alice', 50000, 'Developer'),

(3, 'Bob', 70000, 'Manager'),

(4, 'Eve', 55000, 'Tester');

### Arithmetic Operators:

#### Increase the Salary of Each Employee by 10%:

SELECT id, name, salary, salary \* 1.1 AS increased\_salary FROM emp;

#### Decrease the Salary of Each Employee by 5000:

SELECT id, name, salary, salary - 5000 AS decreased\_salary FROM emp;

### Relational Operators:

#### Fetch the Details of Employees with Salary Greater Than or Equal to 60000:

SELECT \* FROM emp WHERE salary >= 60000;

#### Fetch the Details of Employees with Salary Less Than 60000:

SELECT \* FROM emp WHERE salary < 60000;

### Built-In Functions:

#### Fetch the Uppercase Role for Each Employee:

SELECT id, name, salary, UPPER(role) AS uppercase\_role FROM emp;

#### Fetch the Length of Each Employee's Name:

SELECT id, name, salary, LENGTH(name) AS name\_length FROM emp;

#### Concatenate Name and Role for Each Employee:

SELECT id, name, salary, CONCAT(name, ' - ', role) AS full\_info FROM emp;

### Q.Built-In Functions:

#### Fetch the Square Root of Each Employee's Salary:

sql

SELECT id, name, salary, SQRT(salary) AS sqrt\_salary FROM emp;

#### Fetch the Minimum Salary Among Employees:

sql

SELECT MIN(salary) AS min\_salary FROM emp;

#### Fetch the Maximum Salary Among Employees:

sql

SELECT MAX(salary) AS max\_salary FROM emp;

#### Fetch the Uppercase Role for Each Employee:

sql

SELECT id, name, salary, UPPER(role) AS uppercase\_role FROM emp;

**Q. . Hive Views and Indexes**

### Views:

#### Create a View:

sql

CREATE VIEW employee\_view AS SELECT id, name, salary FROM emp WHERE salary >= 60000;

#### Display the Contents of the View:

sql

SELECT \* FROM employee\_view;

### Order By Clause:

#### Arrange Data in Ascending Order of Salary:

sql

SELECT \* FROM emp ORDER BY salary ASC;

#### Arrange Data in Descending Order of Salary:

sql

SELECT \* FROM emp ORDER BY salary DESC;

### Group By Clause:

#### Fetch the Total Salary by Role:

sql

SELECT role, SUM(salary) AS total\_salary FROM emp GROUP BY role;

### Joins:

#### Inner Join:

sql

CREATE TABLE dept (

dept\_id INT,

dept\_name STRING

);

-- Sample data

INSERT INTO dept VALUES

(1, 'HR'),

(2, 'Engineering');

-- Inner Join

SELECT emp.id, emp.name, emp.salary, dept.dept\_name

FROM emp

JOIN dept ON emp.dept\_id = dept.dept\_id;

#### Left Outer Join:

sql

-- Left Outer Join

SELECT emp.id, emp.name, emp.salary, dept.dept\_name

FROM emp

LEFT OUTER JOIN dept ON emp.dept\_id = dept.dept\_id;

### Indexes

#### Create Index (Not a real index in Hive, just illustrative):

sql

CREATE INDEX emp\_salary\_index ON TABLE emp(salary) AS 'COMPACT' WITH DEFERRED REBUILD;

#### Show Index (Not applicable in Hive; illustrative):

sql

SHOW INDEX ON emp;

#### Drop Index (Not applicable in Hive; illustrative):

sql

DROP INDEX emp\_salary\_index ON emp;