**Check Default Database:**

SHOW DATABASE;

**Create Database:**

CREATE DATABASE IF NOT EXISTS mydatabase;

**Describe Database:**

DESCRIBE DATABASE mydatabase;

**Create Internal Table:**

CREATE TABLE mydatabase.mytable (id INT, name STRING) COMMENT 'My Table Description';

**Describe Table:**

DESCRIBE FORMATTED mydatabase.mytable;

**Add Comments to Table Columns:**

CREATE TABLE mytable (id INT COMMENT 'Unique Identifier', name STRING COMMENT 'Name of the Person');

**Create Table If Not Exists:**

CREATE TABLE IF NOT EXISTS mydatabase.mytable AS SELECT \* FROM existing\_table;

**Load File from Local File System (CSV):**

LOAD DATA LOCAL INPATH '/path/to/local/file.csv' INTO TABLE mytable;

**Alter Table - Rename Table:**

ALTER TABLE old\_table\_name RENAME TO new\_table\_name;

**Alter Table - Add Columns:**

ALTER TABLE mytable ADD COLUMNS (new\_column1 INT, new\_column2 STRING);

**Alter Table - Alter Column:**

ALTER TABLE mytable CHANGE COLUMN old\_column\_name new\_column\_name INT;

**Alter Table - Delete/Replace Column:**

ALTER TABLE mytable REPLACE COLUMNS (new\_column1 INT, new\_column2 STRING);

**Static Partitioning - Creating Files to Store Students' Data:**

INSERT OVERWRITE TABLE student\_partition PARTITION (course='Math') SELECT \* FROM student WHERE course='Math';

**Dynamic Partitioning - Set Partition Values:**

SET hive.exec.dynamic.partition=true;

SET hive.exec.dynamic.partition.mode=nonstrict;

**Dynamic Partitioning - Create Dummy Table and Load Data:**

CREATE TABLE dummy\_table (id INT, name STRING, course STRING);

INSERT INTO TABLE dummy\_table VALUES (1, 'John', 'Math'), (2, 'Alice', 'English');

**Dynamic Partitioning - Create Partition Table and Insert Data:**

CREATE TABLE partitioned\_table (id INT, name STRING) PARTITIONED BY (course STRING);

INSERT OVERWRITE TABLE partitioned\_table PARTITION (course) SELECT id, name, course FROM dummy\_table;

**Retrieve Data by Partition:**

SELECT \* FROM partitioned\_table WHERE course='Math';

### HiveQL Operators:

#### Select the Database:

USE mydatabase;

#### Arithmetic Operators:

* Increase the salary of each employee by 50:

 UPDATE employee SET salary = salary + 50;

 Decrease the salary of each employee by 50:

 UPDATE employee SET salary = salary - 50;

 Find out the 10% salary of each employee:

* SELECT name, salary \* 0.1 AS ten\_percent\_salary FROM employee;

#### Relational Operators:

* Fetch the details of employees with salary >= 40000:

 SELECT \* FROM employee WHERE salary >= 40000;

 Fetch the details of employees with salary < 40000:

* SELECT \* FROM employee WHERE salary < 40000;

### HiveQL Functions:

#### Select the Database and Load Data:

USE mydatabase;

CREATE TABLE employee (id INT, name STRING, salary INT);

LOAD DATA LOCAL INPATH '/path/to/local/employee\_data.csv' INTO TABLE employee;

#### Mathematical Functions:

* Fetch the square root of each employee's salary:
* SELECT name, salary, SQRT(salary) AS sqrt\_salary FROM employee;

#### Aggregate Functions:

* Fetch the maximum salary of an employee:

 SELECT MAX(salary) FROM employee;

 Fetch the minimum salary of an employee:

* SELECT MIN(salary) FROM employee;

#### Built-in Functions:

* Fetch the name of each employee in uppercase:

 SELECT name, UPPER(name) AS upper\_name FROM employee;

 Fetch the name of each employee in lowercase:

SELECT name, LOWER(name) AS lower\_name FROM employee;

### Group By and Having Clause:

#### Select the Database and Load Data:

USE mydatabase;

CREATE TABLE employee (empid INT, empname STRING, salary INT, deptid INT);

LOAD DATA LOCAL INPATH '/path/to/local/employee\_data.csv' INTO TABLE employee;

#### Fetch the Sum of Employee Salaries Department Wise:

SELECT deptid, SUM(salary) AS total\_salary FROM employee GROUP BY deptid;

#### Having Clause - Sum of Employee Salaries by Department (Sum >= 40000):

SELECT deptid, SUM(salary) AS total\_salary FROM employee GROUP BY deptid HAVING SUM(salary) >= 40000;

### Order By and Sort By Clause:

#### Order By Clause - Ascending Order:

SELECT \* FROM employee ORDER BY salary ASC;

#### Order By Clause - Descending Order:

SELECT \* FROM employee ORDER BY salary DESC;

#### Sort By Clause - Descending Order:

SELECT \* FROM employee SORT BY salary DESC;

### HiveQL Join:

#### Create Table Employee1:

CREATE TABLE employee1 (empid INT, empname STRING, deptid INT) ROW FORMAT DELIMITED FIELDS TERMINATED BY ',';

#### Inner Join:

SELECT \* FROM employee e INNER JOIN employee1 e1 ON e.empid = e1.empid;

#### Left Outer Join:

SELECT \* FROM employee e LEFT OUTER JOIN employee1 e1 ON e.empid = e1.empid;

#### Right Outer Join:

SELECT \* FROM employee e RIGHT OUTER JOIN employee1 e1 ON e.empid = e1.empid;

#### Full Outer Join:

SELECT \* FROM employee e FULL OUTER JOIN employee1 e1 ON e.empid = e1.empid;

#### Drop View:

DROP VIEW IF EXISTS sal;

Note: Replace placeholders like mydatabase, /path/to/local/employee\_data.csv with your actual values.