create database company;  
use company;  
  
-- Create EMPLOYEE Table

CREATE TABLE employee (

ssn VARCHAR(10) PRIMARY KEY,

name VARCHAR(50) NOT NULL,

address VARCHAR(100),

sex CHAR(1),

salary INT,

superssn VARCHAR(10),

dno INT

);

-- Create DEPARTMENT Table

CREATE TABLE department (

dno INT PRIMARY KEY,

dname VARCHAR(50) NOT NULL,

mgrssn VARCHAR(10),

mgrstartdate DATE,

FOREIGN KEY (mgrssn) REFERENCES employee(ssn)

);

-- Create DLOCATION Table

CREATE TABLE dlocation (

dno INT,

dlocation VARCHAR(50),

PRIMARY KEY (dno, dlocation),

FOREIGN KEY (dno) REFERENCES department(dno)

);

-- Create PROJECT Table

CREATE TABLE project (

pno INT PRIMARY KEY,

pname VARCHAR(50) NOT NULL,

plocation VARCHAR(50),

dno INT,

FOREIGN KEY (dno) REFERENCES department(dno)

);

-- Create WORKS\_ON Table

CREATE TABLE works\_on (

essn VARCHAR(10),

pno INT,

hours INT,

PRIMARY KEY (essn, pno),

FOREIGN KEY (essn) REFERENCES employee(ssn),

FOREIGN KEY (pno) REFERENCES project(pno)

);

-- Insert into EMPLOYEE

INSERT INTO employee (ssn, name, address, sex, salary, superssn, dno) VALUES

('abc01','ben scott','bangalore','m', 450000, NULL, 3),

('abc02','harry smith','bangalore','m', 500000, 'abc03', 5),

('abc03','lean baker','bangalore','m', 700000, 'abc04', 5),

('abc04','martin scott','mysore','m', 500000, 'abc06', 5),

('abc05','neela sharma','mangalore','f', 650000, 'abc06', 5);

cp

-- Select from EMPLOYEE

SELECT \* FROM employee;

-- Insert into DEPARTMENT

INSERT INTO department (dno, dname, mgrssn, mgrstartdate) VALUES

(1, 'accounts', 'abc05', '2016-01-03'),

(2, 'it', 'abc02', '2017-02-04'),

(3, 'hr', 'abc01', '2016-04-05'),

(4, 'helpdesk', 'abc04', '2017-06-03'),

(5, 'sales', 'abc03', '2017-01-08');

-- Select from DEPARTMENT

SELECT \* FROM department;

-- Insert into DLOCATION

INSERT INTO dlocation (dno, dlocation) VALUES

(1, 'bengaluru'),

(2, 'bengaluru'),

(3, 'bengaluru'),

(4, 'mysore'),

(5, 'mysore');

-- Select from DLOCATION

SELECT \* FROM dlocation;

-- Insert into PROJECT

INSERT INTO project (pno, pname, plocation, dno) VALUES

(1000, 'iot', 'bengaluru', 5),

(1001, 'cloud', 'bengaluru', 5),

(1002, 'bigdata', 'bengaluru', 5),

(1003, 'sensors', 'bengaluru', 3),

(1004, 'bank management', 'bengaluru', 1);

-- Select from PROJECT

SELECT \* FROM project;

-- Insert into WORKS\_ON

INSERT INTO works\_on (essn, pno, hours) VALUES

('abc02', 1000, 4),

('abc02', 1001, 6),

('abc03', 1000, 10),

('abc05', 1000, 3),

('abc01', 1003, 7);

-- Select from WORKS\_ON

SELECT \* FROM works\_on;

**Explanation of the SQL Code**

This SQL script demonstrates how to optimize queries against the call\_detail table by partitioning its primary index using the **range partitioning** technique.

**Step 1: Creating and Using the Database**

create database if not exists prog6;

use prog6;

* **create database if not exists prog6;** → Creates a database named prog6 if it does not already exist.
* **use prog6;** → Selects prog6 as the current working database.

**Step 2: Creating the Partitioned Table**

create table call\_detail (

phone\_number decimal(10) not null,

call\_start timestamp,

call\_duration integer,

call\_description varchar(30),

primary key (phone\_number, call\_start)

)

partition by range(unix\_timestamp(call\_start)) (

partition p0 values less than (unix\_timestamp('2023-01-22 00:00:00')),

partition p1 values less than (unix\_timestamp('2023-01-23 00:00:00'))

);

**Table Schema Explanation**

* **phone\_number DECIMAL(10) NOT NULL** → Stores the phone number, ensuring it is **not null**.
* **call\_start TIMESTAMP** → Stores the start time of the call.
* **call\_duration INTEGER** → Stores the duration of the call (in seconds or minutes).
* **call\_description VARCHAR(30)** → Describes the call.

**Partitioning Explanation**

* The **primary key** is defined as (phone\_number, call\_start), which allows indexing and efficient retrieval based on these two columns.
* **partition by range(unix\_timestamp(call\_start))** → The call\_start column is partitioned based on the Unix timestamp (converting call\_start into an integer for easier range comparisons).
* **Partition Definitions**:
  + **Partition p0** → Contains records where call\_start is **before** '2023-01-22 00:00:00'.
  + **Partition p1** → Contains records where call\_start is **before** '2023-01-23 00:00:00' but **after** '2023-01-22 00:00:00'.

💡 **Why partitioning?**  
Partitioning helps in efficient query execution. Queries searching for call records within specific date ranges will scan only relevant partitions instead of the entire table.

**Step 3: Describing Table Structure**

desc call\_detail;

* Displays the structure of the call\_detail table, showing column names, data types, and constraints.

**Step 4: Creating an Index**

create index idx on call\_detail(phone\_number, call\_start);

* Creates an **index named idx** on the phone\_number and call\_start columns.
* This helps in speeding up queries that filter or sort based on these columns.

**Step 5: Showing Available Indexes**

show index from call\_detail;

* Lists all indexes defined on the call\_detail table, including the primary key and any additional indexes.

**Step 6: Inserting Sample Data**

insert into call\_detail values

(122434231, '2023-01-20 00:00:00', 50, 'FDF'),

(122343243, '2023-01-22 10:00:00', 40, 'FFR'),

(122434230, '2023-01-11 00:00:00', 50, 'FDF'),

(122343242, '2023-01-10 10:00:00', 40, 'FFR'),

(123232323, '2022-01-11 10:00:00', 30, 'eed');

* Inserts **5 records** into the call\_detail table.
* Based on the **partitioning rules**, the data will be stored as follows:
  + **Partition p0**:
    - 122434231, '2023-01-20 00:00:00', 50, 'FDF'
    - 122434230, '2023-01-11 00:00:00', 50, 'FDF'
    - 122343242, '2023-01-10 10:00:00', 40, 'FFR'
    - 123232323, '2022-01-11 10:00:00', 30, 'eed'
  + **Partition p1**:
    - 122343243, '2023-01-22 10:00:00', 40, 'FFR'

**Step 7: Querying Data**

select \* from call\_detail;

* Retrieves **all records** from the call\_detail table, scanning both partitions.

**Step 8: Querying Specific Partitions**

select \* from call\_detail partition (p0);

* Retrieves **only the records stored in partition p0**.

select \* from call\_detail partition (p1);

* Retrieves **only the records stored in partition p1**.

**Benefits of Partitioning**

1. **Improved Query Performance**:
   * Instead of scanning the whole table, queries only scan relevant partitions.
   * Example: A query searching for calls before '2023-01-22' will scan **only partition p0**.
2. **Efficient Indexing**:
   * The index is created within each partition, reducing lookup time.
3. **Better Data Management**:
   * If old records are no longer needed, entire partitions can be dropped instead of deleting row by row.

**Final Thoughts**

This example demonstrates **range partitioning** based on call\_start timestamps. Partitioning helps optimize queries by:

* Storing data efficiently.
* Reducing query execution time.
* Allowing partition pruning (automatically selecting only relevant partitions).

Would you like a query example that benefits from this partitioning? 😊