**ASSIGNMENT NO. 4**

**Aim**

Implement any one Partitioning technique in Parallel Databases

**Objective**

Understand and implement partition techniques in parallel databases.

**Theory**

Importance of I/O Parallelism

* Reduce the time required to retrieve relations from disk by partitioning the relations on multiple disks.
* Horizontal partitioning – tuples of a relation are divided among many disks such that each tuple resides on one disk.

Types of Partitioning Techniques :

**Round-robin**:

Send the *i*th tuple inserted in the relation to disk *i* mod *n*.

**Hash partitioning**:

* + Choose one or more attributes as the partitioning attributes.
  + Choose hash function *h* with range 0…*n* - 1
  + Let *i* denote result of hash function *h* applied to the partitioning attribute value of a tuple. Send tuple to disk *i*.

**Range partitioning:**

* + Choose an attribute as the partitioning attribute.
  + A partitioning vector [*v*o, *v*1, ..., *vn*-2] is chosen.
  + Let *v* be the partitioning attribute value of a tuple. Tuples such that *v*i ≤ *vi*+1 go to disk *I* + 1. Tuples with *v* < *v0* go to disk 0 and tuples with *v* ≥ *v*n-2 go to disk *n*-1.

E.g., with a partitioning vector [5,11], a tuple with partitioning attribute value of 2 will go to disk 0, a tuple with value 8 will go to disk 1, while a tuple with value 20 will go to disk2.

**Table used**

**cqlsh:tp1> select \* from emp;**

**emp\_id | emp\_name**

**--------+----------**

**1101 | AP**

**1001 | DIRECTOR**

**1003 | PRIN**

**1002 | JD**

**(4 rows)**

**Test2.java**

**import** com.datastax.driver.core.Cluster;

**import** com.datastax.driver.core.ResultSet;

**import** com.datastax.driver.core.Row;

**import** com.datastax.driver.core.Session;

**public** **class** test2 {

**public** **static** **void** main(String args[]){

Cluster cluster = Cluster.*builder*().addContactPoint("localhost").build();

System.***out***.println("Cluster Created.....");

//Creating Session object

Session session = cluster.connect("tp1");

System.***out***.println("Session Created.....");

String q4 = "select \* from emp;";

ResultSet result = session.execute(q4);

System.***out***.println("1. Round Robing (Assuning no of partitions 3...... ");

**for** (Row row : result) {

**int** p;

p=row.getInt("emp\_id")%3;

System.***out***.format("Partion number for record %d is %d \n", row.getInt("emp\_id"), p);

}

ResultSet result1 = session.execute(q4);

System.***out***.println("2. Range partiotion (assuming empid less than 1100 on disk 0 and remaining on disk1 ...... ");

**for** (Row row1 : result1) {

**int** p1;

p1=row1.getInt("emp\_id");

**if**(p1<1100)

System.***out***.format("Partion number for record %d is 0 \n", row1.getInt("emp\_id"));

**else**

System.***out***.format("Partion number for record %d is 1 \n", row1.getInt("emp\_id"));

}

ResultSet result2 = session.execute(q4);

System.***out***.println("3. Hash partiotion (assuming h(k)= k mod n ...... ");

**for** (Row row2 : result2) {

**int** p2;

p2=row2.getInt("emp\_id")%3;

System.***out***.format("Partion number for record %d is %d \n", row2.getInt("emp\_id"), p2);

}

session.close();

cluster.close();

}

}

Output

Cluster Created.....

Session Created.....

1. Round Robing (Assuning no of partitions 3......

Partion number for record 1101 is 0

Partion number for record 1001 is 2

Partion number for record 1003 is 1

Partion number for record 1002 is 0

2. Range partiotion (assuming empid less than 1100 on disk 0 and remaining on disk1 ......

Partion number for record 1101 is 1

Partion number for record 1001 is 0

Partion number for record 1003 is 0

Partion number for record 1002 is 0

3. Hash partiotion (assuming h(k)= k mod n ......

Partion number for record 1101 is 0

Partion number for record 1001 is 2

Partion number for record 1003 is 1

Partion number for record 1002 is 0