Experiment No.2

Title: Implementation of Distributed Database.

Batch: A2 **Roll No.: 16010421059** **Experiment No.: 2**

Aim: To Implement Distributed Database.

Resources needed: PostgreSQL 9.3

Theory

A distributed database system allows applications to access and manipulate data from local and remote databases. It partitions the data and stores at different physical locations. Partitioning refers to splitting what is logically one large table into smaller physical pieces. Partitioning can provide several benefits:

- Query performance can be improved dramatically for certain kinds of queries.
- Update performance can be improved too, since each piece of the table has indexes smaller than an index on the entire data set would be. When an index no longer fits easily in memory, both read and write operations on the index take progressively more disk accesses.
- Bulk deletes may be accomplished by simply removing one of the partitions, if that requirement is planned into the partitioning design. `DROP TABLE` is far faster than a bulk `DELETE`, to say nothing of the ensuing `VACUUM` overhead.
- Seldom-used data can be migrated to cheaper and slower storage media.

Partitioning enhances the performance, manageability, and availability of a wide variety of applications and helps reduce the total cost of ownership for storing large amounts of data. Partitioning allows tables, indexes, and index-organized tables to be subdivided into smaller pieces, enabling these database objects to be managed and accessed at a finer level of granularity.

The benefits will normally be worthwhile only when a table would otherwise be very large. The exact point at which a table will benefit from partitioning depends on the application, although a rule of thumb is that the size of the table should exceed the physical memory of

The following forms of partitioning can be implemented in PostgreSQL:

Range Partitioning

The table is partitioned into "ranges" defined by a key column or set of columns, with no overlap between the ranges of values assigned to different partitions. For example one might partition by date ranges, or by ranges of identifiers for particular business objects.

List Partitioning

The table is partitioned by explicitly listing which key values appear in each partition.

After creating the partition, **database link (DBLINK)** is used to create a connection of the host database server with the client database.

Database Links:

The central concept in distributed database systems is a **database link**. A database link is a connection between two physical database servers that allows a client to access them as one logical database. Database link is a pointer that defines a one-way communication path from one Database server to another database server. The link pointer is actually defined as an entry in a data dictionary table. To access the link, you must be connected to the local database that contains the data dictionary entry.

A database link connection is one-way in the sense that a client connected to local database A can use a link stored in database A to access information in remote database B, but users connected to database B cannot use the same link to access data in database A. If local users on database B want to access data on database A, then they must define a link that is stored in the data dictionary of database B.

A database link connection allows local users to access data on a remote database. For this connection to occur, each database in the distributed system must have a unique global database name in the network domain. The global database name uniquely identifies a database server in a distributed system.

dblink executes a query (usually a SELECT, but it can be any SQL statement that returns rows) in a remote database.

When two text arguments are given, the first one is first looked up as a persistent connection's name; if found, the command is executed on that connection. If not found, the first argument is treated as a connection info string as for dblink_connect, and the indicated connection is made just for the duration of this command.

Arguments

conname

Name of the connection to use; omit this parameter to use the unnamed connection.

connstr A connection info string, as previously described for dblink_connect.

sql The SQL query that you wish to execute in the remote database, for example select * from foo.

fail_on_error

If true (the default when omitted) then an error thrown on the remote side of the connection causes an error to also be thrown locally. If false, the remote error is locally reported as a NOTICE, and the function returns no rows.

Return Value

The function returns the row(s) produced by the query. Since dblink can be used with any query, it is declared to return record, rather than specifying any particular set of columns. This means that you must specify the expected set of columns in the calling query — otherwise PostgreSQL would not know what to expect. Here is an example:

```
SELECT *
  FROM dblink('dbname=mydb', 'select proname, prosrc from pg_proc')
    AS t1(proname name, prosrc text)
  WHERE proname LIKE 'bytea%';
```

Procedure:

Implementing distributed database:

1. **Create the parent table.**

Create the parent table.

```
CREATE TABLE sales(org int, name varchar(10));
```

2. **Create the child (partitioned) tables**

```
CREATE TABLE sales_part1
(CHECK (org < 6))
INHERITS (sales);

CREATE TABLE sales_part2
(CHECK (org >=6 and org <=10))
INHERITS (sales);
```

3. **Create the rules**

```
CREATE OR REPLACE RULE insert_sales_p1
AS ON INSERT TO sales
WHERE (org <6)
DO INSTEAD
INSERT INTO sales_part1 VALUES(NEW.org, NEW.name);

CREATE OR REPLACE RULE insert_sales_p2
AS ON INSERT TO sales
WHERE (org >=6 and org <=10 )
DO INSTEAD
INSERT INTO sales_part2 VALUES(New.org,New.name);
```

4. **Add sample data to the new table.**

```
INSERT INTO sales VALUES(1,'Craig');
INSERT INTO sales VALUES(2,'Mike');
INSERT INTO sales VALUES(3,'Michelle');
INSERT INTO sales VALUES(4,'Joe');
INSERT INTO sales VALUES(5,'Scott');
INSERT INTO sales VALUES(6,'Roger');
INSERT INTO sales VALUES(7,'Fred');
INSERT INTO sales VALUES(8,'Sam');
INSERT INTO sales VALUES(9,'Sonny');
INSERT INTO sales VALUES(10,'Chris');
```

5. **Confirm that the data was added to the parent table and the partition tables**

```
SELECT * FROM sales;

SELECT * FROM sales_part1;
SELECT * FROM sales_part2;
```

6. **Create a dblink_connect to create a connection string to use.**

   Access the file : pg_hba.conf file under C:\Program Files\PostgreSQL\9.3\data and make the following entry , stating that the host machine accessible to other machines.

   ```
   host    all         all         all         trust
   ```

Create Extension dblink;
SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.103 dbname=postgres user=postgres password=postgres')
172.17.17.103 *is the host address that has the database and the partitions .*

7. **Use dblink command on the remote machine to access the partitions present in the host machine.**
   Access the file : pg_hba.conf file under C:\Program Files\PostgreSQL\9.3\data and make the following entry , stating that the remote machine needs to access the host machine:
   host postgres postgres 172.17.17.103/32 md5
   And the client can execute the following command, in the SQL Query window:
   sample:

   Create Extension dblink;

   SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.103 dbname=sachin user=postgres password=postgres')

   select * from dblink('myconn','select * from sales_part2')AS T1(Column1 int, column2 varchar(10)) order by column2 desc;

   Inserting data into the table remotely

   select dblink_exec('myconn','insert into sales values(12,"John")')
   select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2 varchar(10)) order by column1 asc;

   Delete data from table remotely

   select dblink_exec('myconn','delete from sales org=3')
   select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2 varchar(10)) order by column1 asc;

Results:

CREATE TABLE sales(org int, name varchar(10));

CREATE TABLE sales_part1
(CHECK (org < 6))
INHERITS (sales);

CREATE TABLE sales_part2
(CHECK (org >=6 and org <=10))
INHERITS (sales);

CREATE OR REPLACE RULE insert_sales_p1
AS ON INSERT TO sales
WHERE (org <6)
DO INSTEAD
INSERT INTO sales_part1 VALUES(NEW.org, NEW.name);

```
CREATE OR REPLACE RULE insert_sales_p2
AS ON INSERT TO sales
WHERE (org >=6 and org <=10 )
DO INSTEAD
INSERT INTO sales_part2 VALUES(New.org,New.name);


INSERT INTO sales VALUES(1,'Craig');
INSERT INTO sales VALUES(2,'Mike');
INSERT INTO sales VALUES(3,'Michelle');
INSERT INTO sales VALUES(4,'Joe');
INSERT INTO sales VALUES(5,'Scott');
INSERT INTO sales VALUES(6,'Roger');
INSERT INTO sales VALUES(7,'Fred');
INSERT INTO sales VALUES(8,'Sam');
INSERT INTO sales VALUES(9,'Sonny');
INSERT INTO sales VALUES (10,'Chris');
```

select*from sales;

org integer	name character varying (10)
1	Craig
2	Mike
3	Michelle
4	Joe
5	Scott
6	Roger
7	Fred
8	Sam
9	Sonny
10	Chris

select*from sales_part2;

org integer	name character varying (10)
6	Roger
7	Fred
8	Sam
9	Sonny
10	Chris

select*from sales_part1;

org integer	name character varying (10)
6	Roger
7	Fred
8	Sam
9	Sonny
10	Chris

Create Extension dblink;
SELECT dblink_connect('myconn1' ,'hostaddr=172.17.17.144 dbname=EXP2 user=postgres password=postgres')

select * from dblink('myconn1','select * from sales_part2')AS T1(Column1 int, column2 varchar(10)) order by column2 desc

column1 integer	column2 character varying (10)
9	Sonny
8	Sam
6	Roger
7	Fred
10	Chris

select dblink_exec('myconn1','insert into sales values(12,"John")')

dblink_exec text
INSERT 0 1

select * from dblink('myconn1','select * from sales')AS T1(Column1 int, column2 varchar(10)) order by column1 asc;

column1 integer	column2 character varying (10)
1	Craig
2	Mike
3	Michelle
4	Joe
5	Scott
6	Roger
7	Fred
8	Sam
9	Sonny
10	Chris
12	John
12	John
12	John
12	John
12	John
12	John
12	John

Select dblink_exec('myconn1','delete from sales where org=3')

dblink_exec text
DELETE 0

select * from dblink('myconn1','select * from sales')AS T1(Column1 int, column2 varchar(10)) order by column1 asc;

column1 integer	column2 character varying (10)
1	Craig
2	Mike
4	Joe
5	Scott
6	Roger
7	Fred
8	Sam
9	Sonny
10	Chris
12	John
12	John
12	John
12	John
12	John
12	John
12	John
12	John

Questions:

1. **What are the different types of distributed database systems.**

a) Homogeneous Database:

In a homogeneous database, all different sites store database identically. The operating system, database management system, and the data structures used – all are the same at all sites. Hence, they're easy to manage.

b) Heterogeneous Database:

In a heterogeneous distributed database, different sites can use different schema and software that can lead to problems in query processing and transactions. Also, a particular site might be completely unaware of the other sites. Different computers may use a different operating system, different database application. They may even use different data models for the database. Hence, translations are required for different sites to communicate.

2. **Give steps to insert and delete records in remote table.**

Step1:

Create Extension dblink;
SELECT dblink_connect('myconn' ,'hostaddr=172.17.17.103 dbname=postgresuser=postgres password=postgres')

Step2:

Inserting data into the table remotely

select dblink_exec('myconn','insert into sales values(12,"John")')
select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2varchar(10)) order by column1 asc;

Step3:

Delete data from table remotely

select dblink_exec('myconn','delete from sales org=3')
select * from dblink('myconn','select * from sales')AS T1(Column1 int, column2varchar(10)) order by column1 asc;

Outcomes: CO1 Design advanced database systems using Parallel, Distributed and In-memory Databases and its implementation.

.

Conclusion: Understood the working and the concepts of a distributed database and applied it on our own database to check and get the optimal results.

Grade: AA / AB / BB / BC / CC / CD /DD

Signature of faculty in-charge with date

References:

Books/ Journals/ Websites:

1. Elmasri and Navathe, “Fundamentals of Database Systems”, Pearson Education
2. https://www.postgresql.org/docs/