

**Experiment No.2**

**Title:** Implementation of Distributed Database.

Page No:

**Batch: A2 Roll No.:16010421073 Experiment No.: 2**

**Aim: To** Implement Distributed Database.

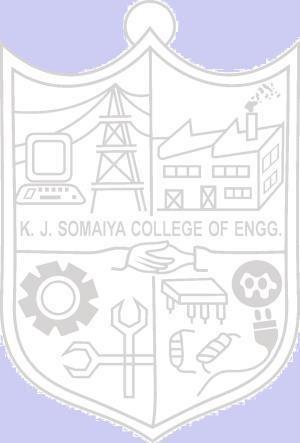


**Resources needed:** PostgreSQL 9.3



# 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.

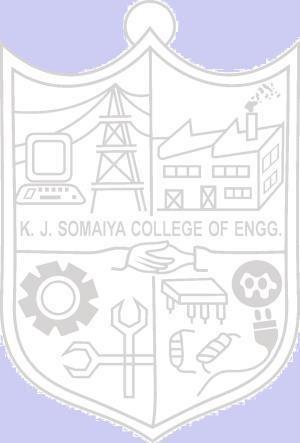
After creating the partition, **database link (DBLINK)** is used to create a connection of the host database server with the client database.

Page No:

# 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 sql

A connection info string, as previously described for dblink\_connect.

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%';

Page No:

# Procedure:

**Implementing distributed database:**

# Create the parent table.

Create the parent table.

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

# 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);

# 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);

1. **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');

# 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;

# 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 .*

# 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;

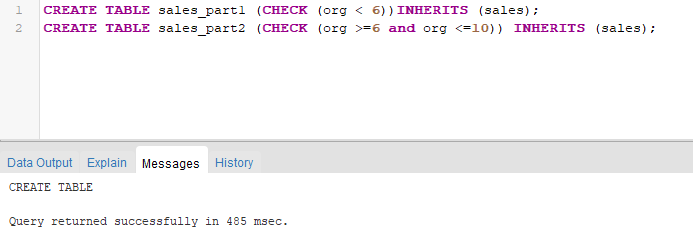


# Results: (Program printout with output)

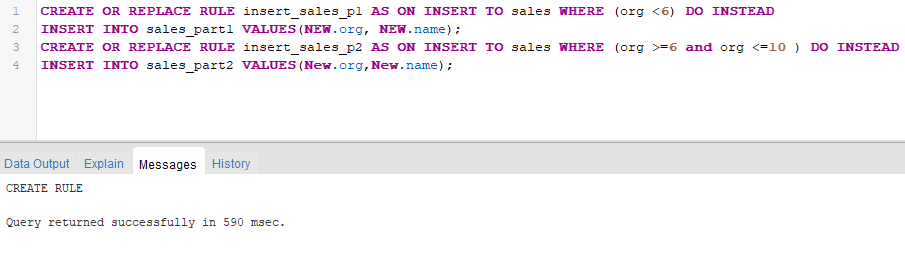
# Step1: Create the parent table.

# 

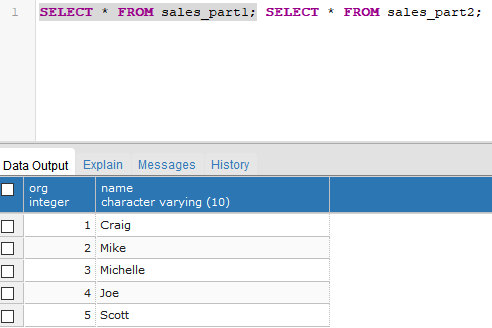
# Step2: Create the child (partitioned) tables

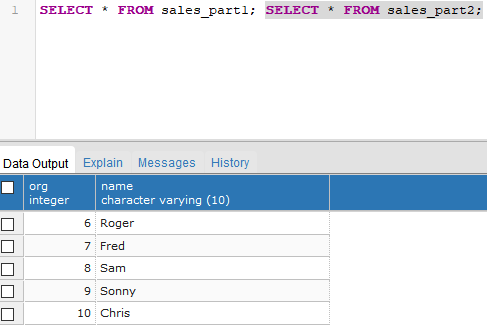


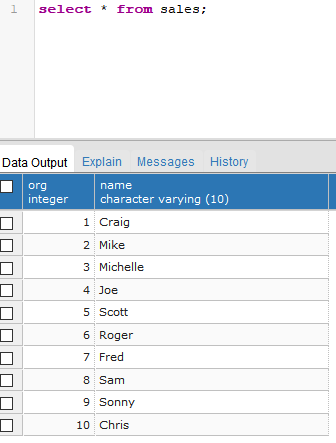
**Step3: Crating Rules**

**Step4: Inserting values in sales**

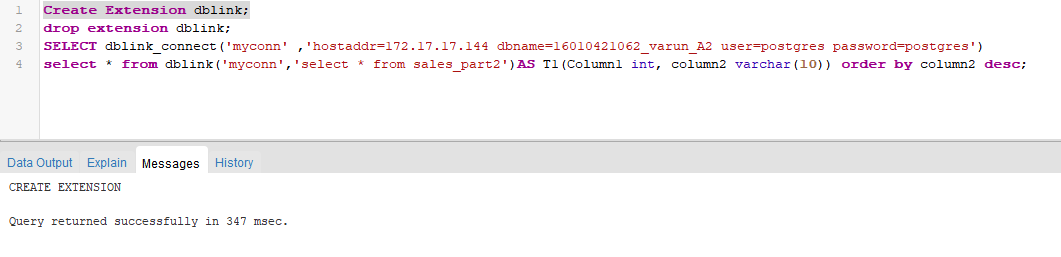
**Step5: Checking data in tables**

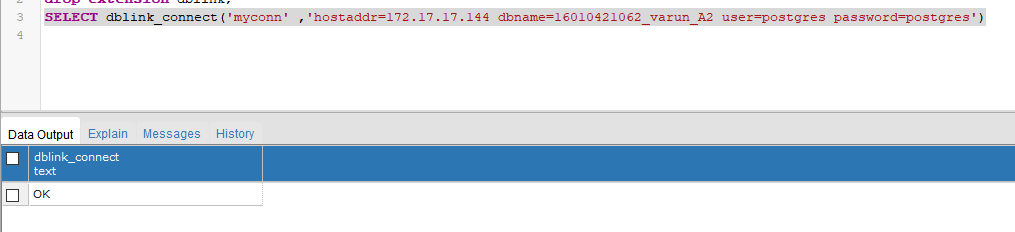




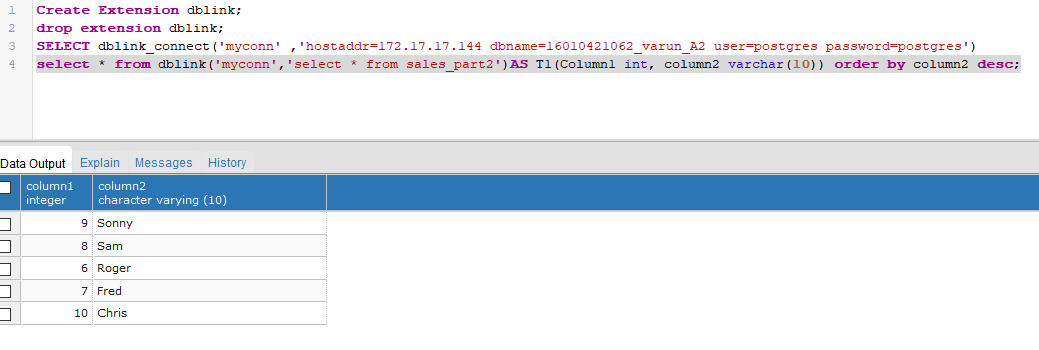


**Step6:**

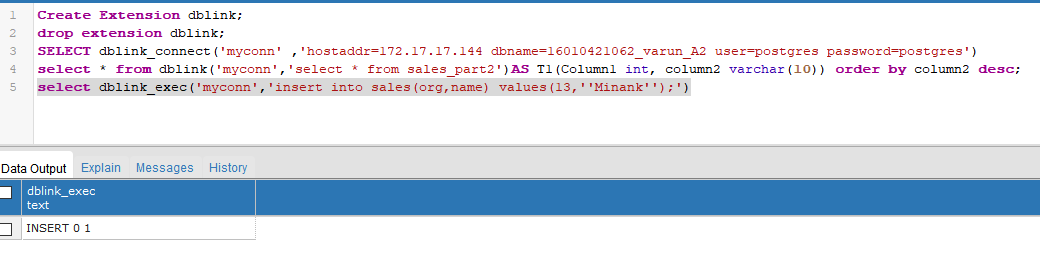
**Creating Link**

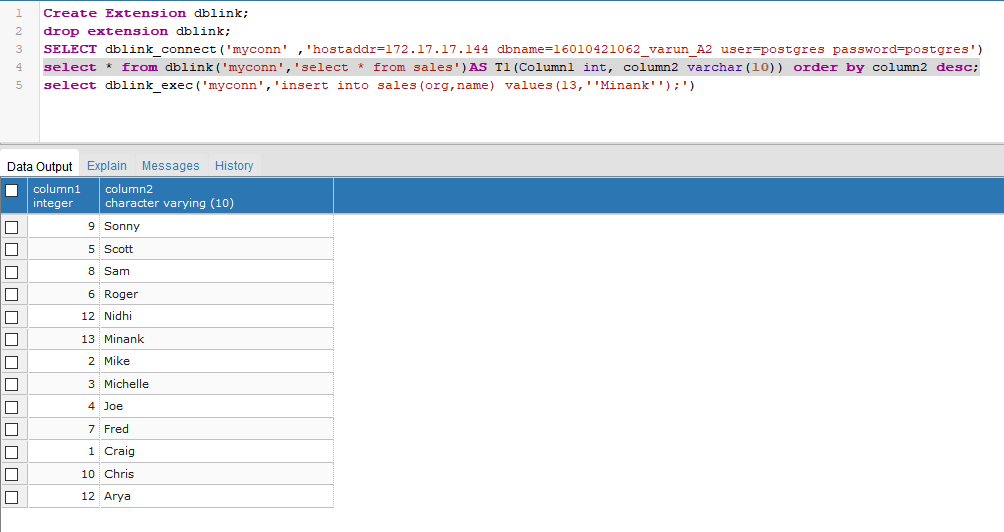


**Step7: Displaying table from host machine**

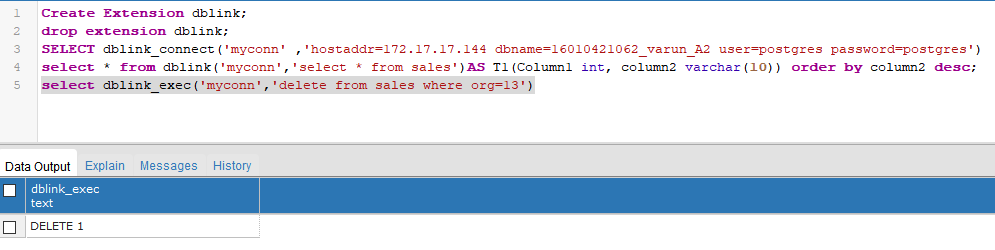


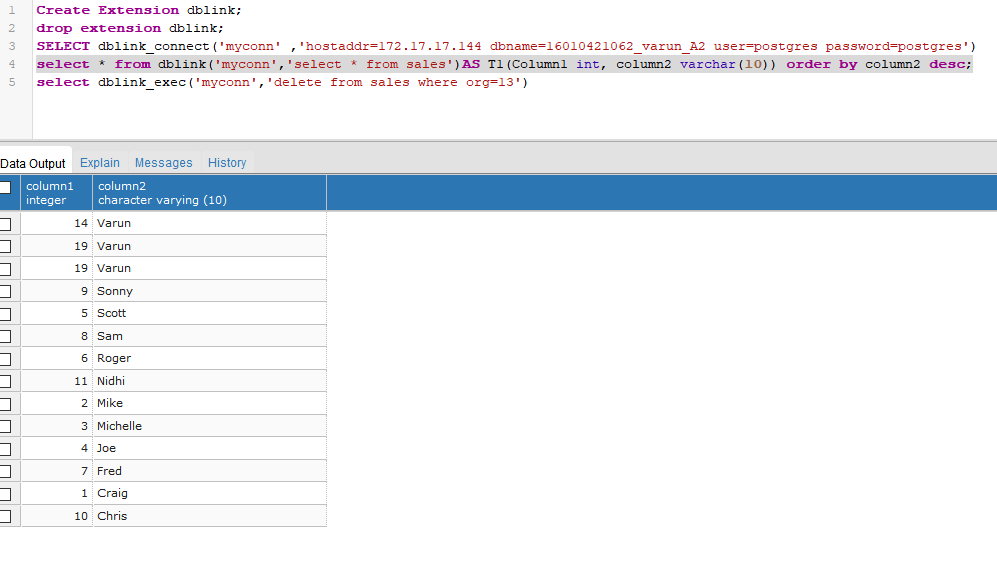
**Step8: Insertion from remote machine:**

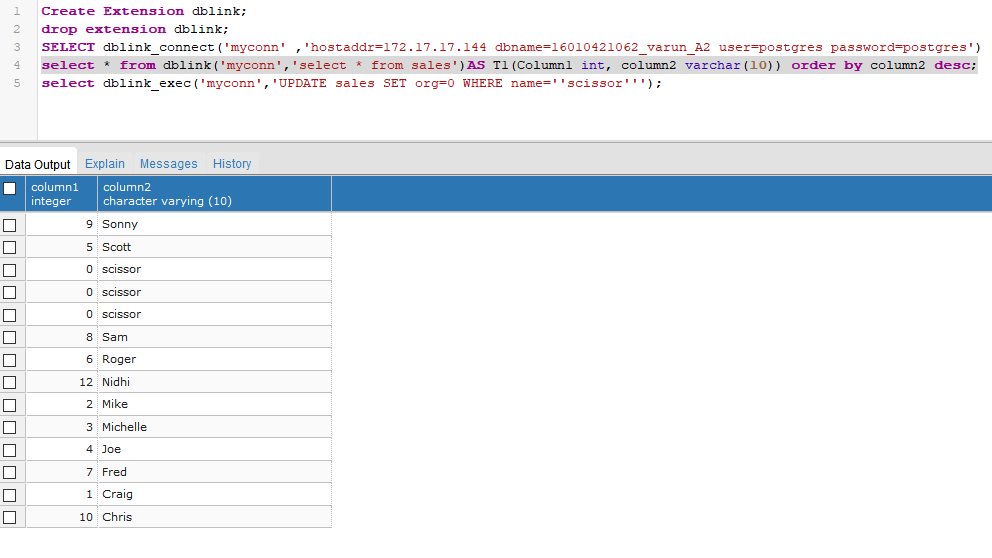


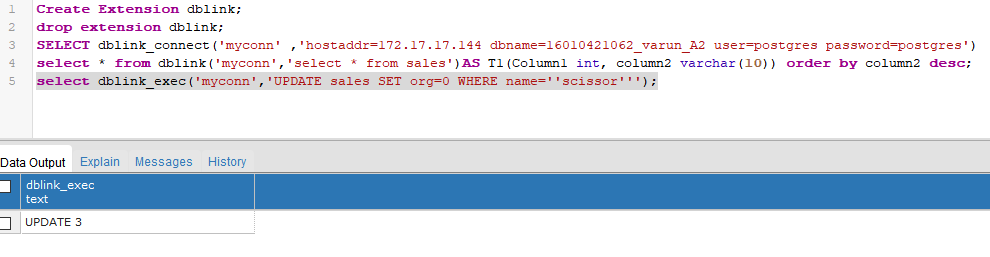


**Step 9: Remove from remote machine**





**Step10: Updation**



**Questions:**

# What are the different types of distributed database systems.

**Ans:**

A distributed database is a database that is not limited to one computer system.

It is like a database that consists of two or more files located in different computers or sites either on the same network or on an entirely different network.

There are two types of distributed databases:

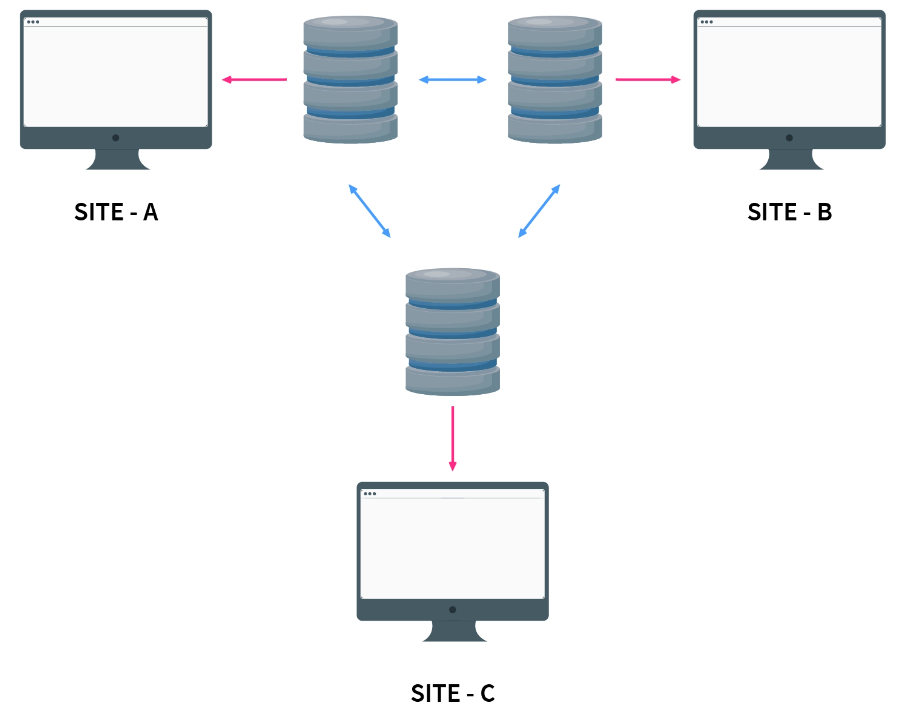
*Homogenous distributed database.*

*Heterogeneous distributed database.*

**Homogenous Distributed Database**

* A Homogenous distributed database is a network of identical databases stored on multiple sites.
* All databases stores data identically, the operating system, DDBMS and the data structures used – all are same at all sites, making them easy to manage.

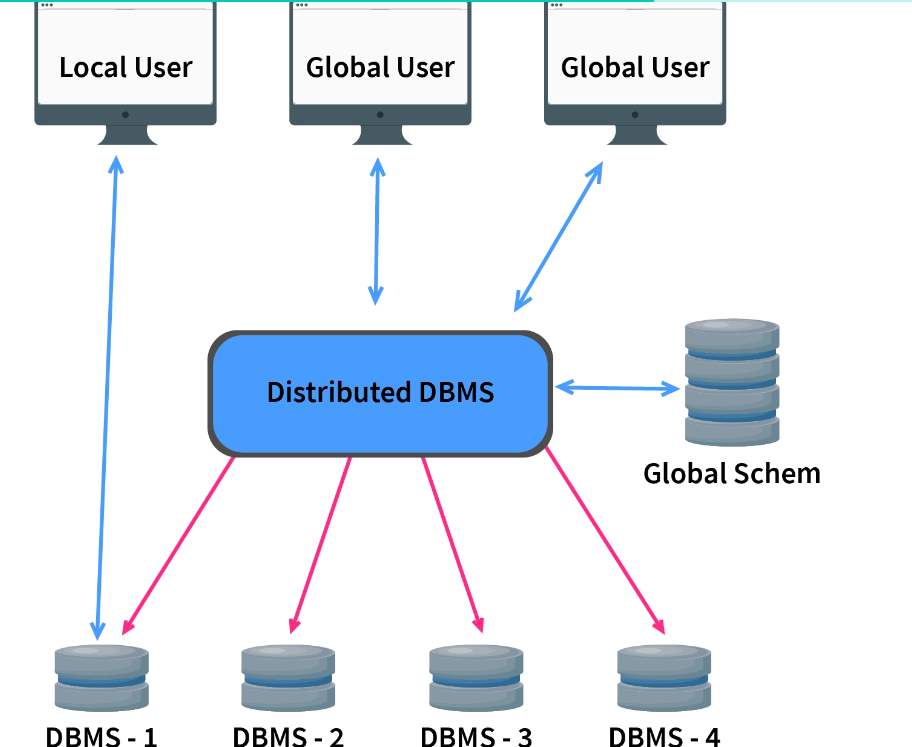
Below is a diagram for the same,



**Heterogeneous Distributed Database**

* It is the opposite of a Homogenous distributed database.
* It uses different schemas, operating systems, DDBMS, and different data models causing it difficult to manage.
* In the case of a Heterogeneous distributed database, a particular site can be completely unaware of other sites.
* This causes limited cooperation in processing user requests, this is why translations are required to establish communication between sites.

Below is a diagram for the same,



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

**Ans:** In order to insert and delete records in a remote table using a database link, you need to follow these steps:

* Establish a database link: Before you can insert or delete records in a remote table, you need to establish a database link that connects the local database to the remote database.
* You can create a database link using the CREATE DATABASE LINK statement in SQL.
* **Insert records:** To insert records in a remote table, you can use the INSERT INTO statement and specify the name of the remote table and the database link. For example:

**INSERT INTO <remote\_table> SELECT \* FROM <local\_table> WHERE <condition>;**

* To **delete records** from a remote table, you can use the following syntax:
* **DELETE FROM <remote\_table> WHERE <condition>;** where remote\_table is the name of the remote table, and condition is a clause that specifies the conditions for which records should be deleted.
* These operations require that the user executing them has sufficient privileges on both the local and remote databases.



**Outcomes:**

**CO1 :** Design advanced database systems using Parallel, Distributed and In-memory

Databases and its implementation.

# Conclusion: (Conclusion to be based on the outcomes achieved):

# Thus we successfully learned about implementation of distributed databases.



**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.post](http://www.postgresql.org/docs/)g[resql.org/docs/](http://www.postgresql.org/docs/)