

**CSE 512 Distributed Database Systems**

**Project Part - 2**

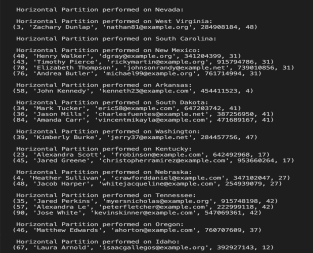
**Partitioning**

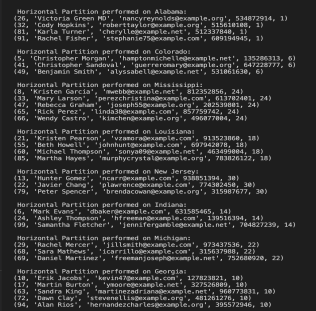
Database partitioning involves the segmentation of data within an application's database into distinct sections or partitions. These partitions can be stored, accessed, and administered independently. Implementing data partitioning can enhance the scalability and performance of the application. There are two types of partitioning: vertical and horizontal partitioning.

**Horizontal Partitioning**

Horizontal partitioning involves the division of a table based on rows, using a specified range or condition. This approach is advantageous when managing tables with a substantial number of rows, and when data can be logically grouped based on specific criteria. The primary goal of horizontal partitioning is to enhance query performance by reducing the volume of data that needs to be scanned for particular queries.

In the database we have performed horizontal partitioning based on each state name from the list of state names that were inserted into the database as shown in below figures .Users were partitioned based on their user\_id, full name, email address, phone number and state\_id as shown in Figure[1] and Figure[2].

Figure 1:Horizontal partitioning

Figure 2:Horizontal partitioning

**Vertical Partitioning**

Vertical partitioning is a database optimization strategy that entails splitting a table based on columns. This approach proves beneficial when dealing with tables containing numerous columns, especially when not all columns are regularly accessed together. The primary aim is to enhance query performance by minimizing I/O operations and facilitating more effective indexing of pertinent columns.

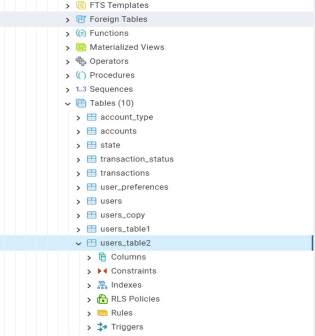
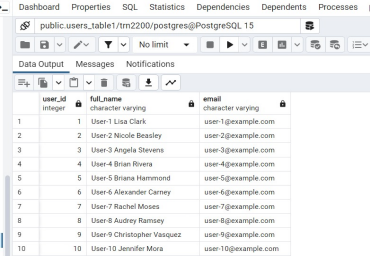
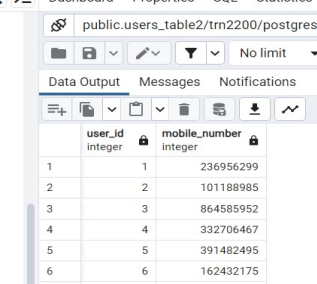
In the Vertical Partitioning scheme, we've partitioned the User table into two tables: User Table 1 and User Table 2. User Table 1 contains columns such as user\_id, full\_name, and email. While, User Table 2 includes user\_id and mobile\_number, as illustrated in Figure [3].The outcomes of the query for User Table 1 are exhibited in Figure [4], whereas the result set corresponding to User Table 2 is presented in Figure [5]. 

Figure 3:PgAdmin Vertical Partition

Figure 4:Result on Vertical Partition of User Table 1

Figure 5: Result on Vertical Partition of User Table

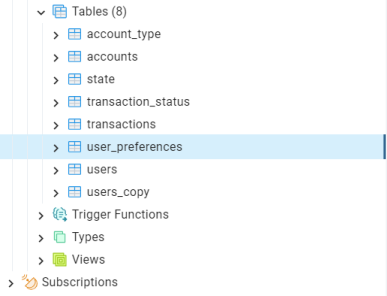
**Data Replication**

Data replication in a database management system (DBMS) involves creating and maintaining multiple copies of the same data across different locations, databases, or servers. Each copy of the data is referred to as a replica. Replication can occur within a single database, across multiple databases, or even across geographically distributed servers.

We have simulated a master-slave replication mechanism for data replication.

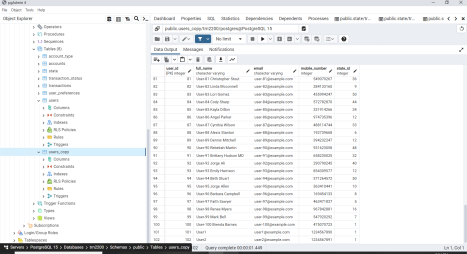
The master copy contains the primary data and takes care of all write operations. The slave is the replica and is used for read-only purposes. In the event of failure, the replica can be used for further reads. This serves as a failsafe mechanism.

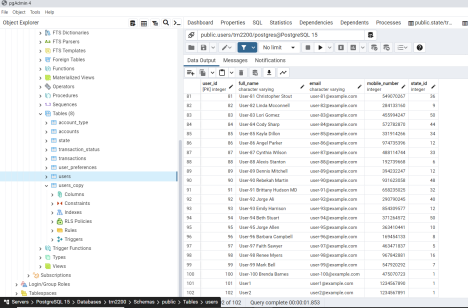
For the purpose of demonstrating data replication, we have created a copy of the user’s table and ensured that any and all writes on the table are reflected in the replica table. The replica table can be used for querying purposes alone and it cannot be used for logging transactions as shown in Figure 6.

Figure 6:Replica Table Creation

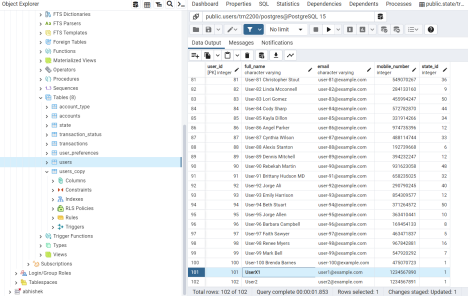
The initial design of the database had 7 tables created. Replicating a table would result in the addition of another new table. We worked on the Python script to create and connect to a database and create all necessary tables and the recovery or replica table for fail-safe handling.

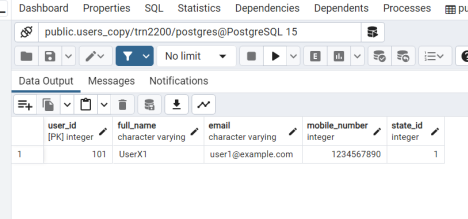
To demonstrate how it works, we have added two new rows in the accounts table. And let us compare the results of both tables as illustrated in Figure 7.

Figure 7: User’s copy Table

Figure 8:User Table

For instance, Let us rename User1 to UserX1 in the User table. Any write operations performed on the master table should be transmitted or reflected to the slave or replica as well.

Figure 9: Renaming the data in the User Table

Figure 10:Replicated Table reflecting the changes

Thus we have managed to create a table which inherits all changes made to the master table. In an ideal master-slave server, we do not make any writes to the replica. This is because modifying directly into the replica tables might result in inconsistencies or might overwrite the values stored elsewhere. We have shown a replication of a table. Similarly, it can be done for the whole database or a selection of tables, based on the requirement. Note: Triggers were used to ensure any and all changes to the master table are propagated to the replica table