



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].

```

Horizontal Partition performed on Nevada:

Horizontal Partition performed on West Virginia:
(3, 'Zachary Dunlap', 'nathan81@example.org', 284908184, 48)

Horizontal Partition performed on South Carolina:

Horizontal Partition performed on New Mexico:
(40, 'Henry Walker', 'dgray@example.org', 341204399, 31)
(43, 'Timothy Pierce', 'rickymartin@example.org', 915794786, 31)
(70, 'Elizabeth Thompson', 'johnsonrandy@example.net', 739010856, 31)
(76, 'Andrea Butler', 'michael99@example.org', 761714994, 31)

Horizontal Partition performed on Arkansas:
(58, 'John Kennedy', 'kenneth23@example.com', 454411523, 4)

Horizontal Partition performed on South Dakota:
(34, 'Mark Tucker', 'eric58@example.com', 647203742, 41)
(36, 'Jason Mills', 'charlesfuentes@example.net', 387256950, 41)
(84, 'Amanda Carr', 'vincentmikayla@example.com', 471689167, 41)

Horizontal Partition performed on Washington:
(39, 'Kimberly Burke', 'jerry37@example.net', 284457756, 47)

Horizontal Partition performed on Kentucky:
(23, 'Alexandra Scott', 'frobinson@example.com', 642492968, 17)
(45, 'Jared Greene', 'christopherramirez@example.com', 953660264, 17)

Horizontal Partition performed on Nebraska:
(4, 'Heather Sullivan', 'crawforddaniel@example.com', 347102047, 27)
(48, 'Jacob Harper', 'whitejacqueline@example.com', 254939079, 27)

Horizontal Partition performed on Tennessee:
(35, 'Jared Perkins', 'myersnicholas@example.org', 915748198, 42)
(57, 'Alexandra Le', 'peterfletcher@example.com', 222999118, 42)
(90, 'Jose White', 'kevinskinner@example.com', 547069361, 42)

Horizontal Partition performed on Oregon:
(46, 'Matthew Edwards', 'ahorton@example.com', 760707609, 37)

Horizontal Partition performed on Idaho:
(67, 'Laura Arnold', 'isaacgallegos@example.org', 392927143, 12)

```

Figure 1:Horizontal partitioning

```

Horizontal Partition performed on Alabama:
(26, 'Victoria Green MD', 'nancyreynolds@example.org', 534872914, 1)
(32, 'Cody Hopkins', 'robertttaylor@example.org', 515610108, 1)
(81, 'Karla Turner', 'cherylle@example.net', 512337840, 1)
(91, 'Rachel Fisher', 'stephanie75@example.com', 609194945, 1)

Horizontal Partition performed on Colorado:
(5, 'Christopher Morgan', 'hamptonmichelle@example.net', 135286313, 6)
(41, 'Christopher Sandoval', 'guerrerrromary@example.org', 647228777, 6)
(49, 'Benjamin Smith', 'alysabell@example.net', 531061630, 6)

Horizontal Partition performed on Mississippi:
(8, 'Kristen Garcia', 'mwebb@example.net', 812352856, 24)
(33, 'Mary Larson', 'perezchristina@example.com', 613702401, 24)
(47, 'Rebecca Graham', 'joseph55@example.org', 202539801, 24)
(66, 'Rick Perez', 'linda38@example.com', 857759742, 24)
(65, 'Wendy Castro', 'kimchen@example.org', 496077004, 24)

Horizontal Partition performed on Louisiana:
(21, 'Kristen Pearson', 'vzamora@example.com', 913523860, 18)
(55, 'Beth Howell', 'johnhunt@example.com', 697942078, 18)
(60, 'Michael Thompson', 'sonya09@example.net', 463499004, 18)
(85, 'Martha Hayes', 'murphycrystal@example.org', 783826122, 18)

Horizontal Partition performed on New Jersey:
(13, 'Hunter Gomez', 'ncarr@example.com', 938851394, 30)
(22, 'Javier Chang', 'plawrence@example.com', 774302450, 30)
(79, 'Peter Spencer', 'brendacowan@example.org', 315987677, 30)

Horizontal Partition performed on Indiana:
(6, 'Mark Evans', 'dbaker@example.com', 631585465, 14)
(24, 'Ashley Thompson', 'hfreeman@example.com', 139516394, 14)
(99, 'Samantha Fletcher', 'jennifergamble@example.net', 704827239, 14)

Horizontal Partition performed on Michigan:
(29, 'Rachel Mercer', 'jillsmith@example.com', 973437536, 22)
(68, 'Sara Mathews', 'icarrillo@example.com', 315637988, 22)
(69, 'Daniel Martinez', 'freemanjoseph@example.net', 752680920, 22)

Horizontal Partition performed on Georgia:
(10, 'Erik Jacobs', 'kevin47@example.com', 127823821, 10)
(17, 'Martin Burton', 'ymoore@example.net', 327526809, 10)
(63, 'Sandra King', 'martinezadriana@example.net', 960773831, 10)
(72, 'Dawn Clay', 'stevenellis@example.org', 481261276, 10)
(94, 'Alan Rios', 'hernandezcharles@example.org', 395572946, 10)

```

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].

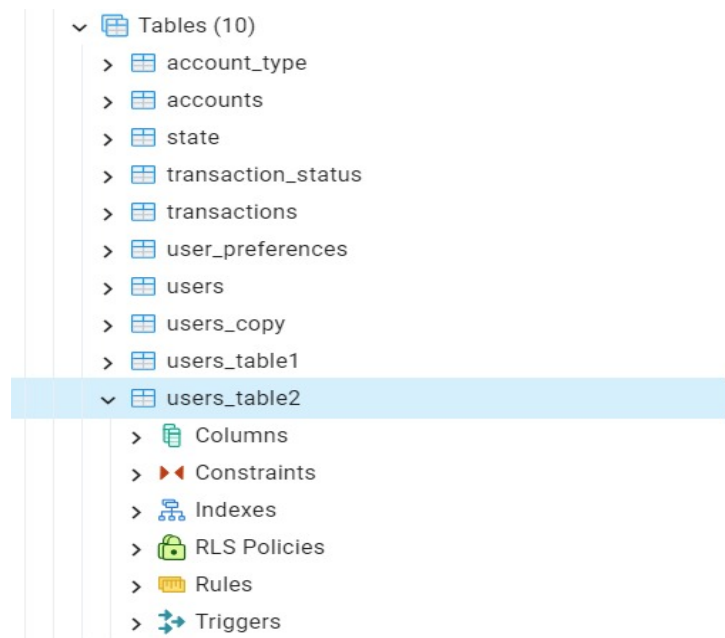


Figure 3:PgAdmin Vertical Partition

Dashboard Properties SQL Statistics Dependencies Dependents Processes

public.users_table1/trn2200/postgres@PostgreSQL 15

Data Output Messages Notifications

	user_id integer	full_name character varying	email character varying
1	1	User-1 Lisa Clark	user-1@example.com
2	2	User-2 Nicole Beasley	user-2@example.com
3	3	User-3 Angela Stevens	user-3@example.com
4	4	User-4 Brian Rivera	user-4@example.com
5	5	User-5 Briana Hammond	user-5@example.com
6	6	User-6 Alexander Carney	user-6@example.com
7	7	User-7 Rachel Moses	user-7@example.com
8	8	User-8 Audrey Ramsey	user-8@example.com
9	9	User-9 Christopher Vasquez	user-9@example.com
10	10	User-10 Jennifer Mora	user-10@example.com

Figure 4:Result on Vertical Partition of User Table 1

Dashboard Properties SQL Statistics

public.users_table2/trn2200/postgres

Data Output Messages Notifications

	user_id integer	mobile_number integer
1	1	236956299
2	2	101188985
3	3	864585952
4	4	332706467
5	5	391482495
6	6	162432175

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.

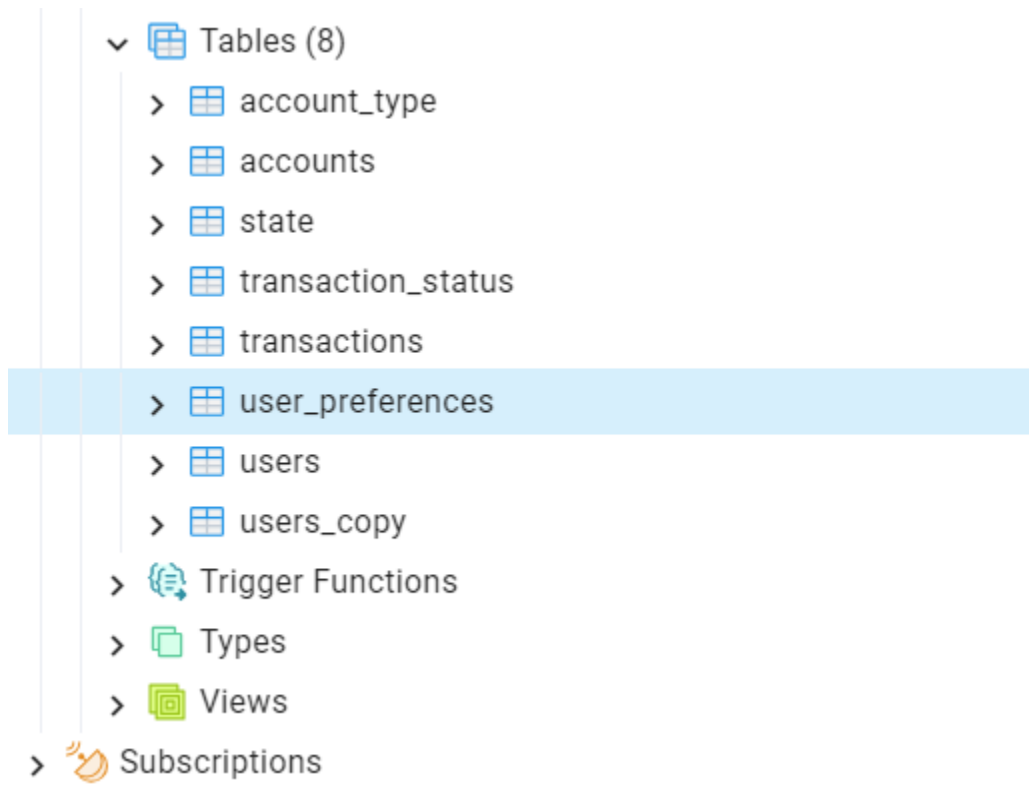
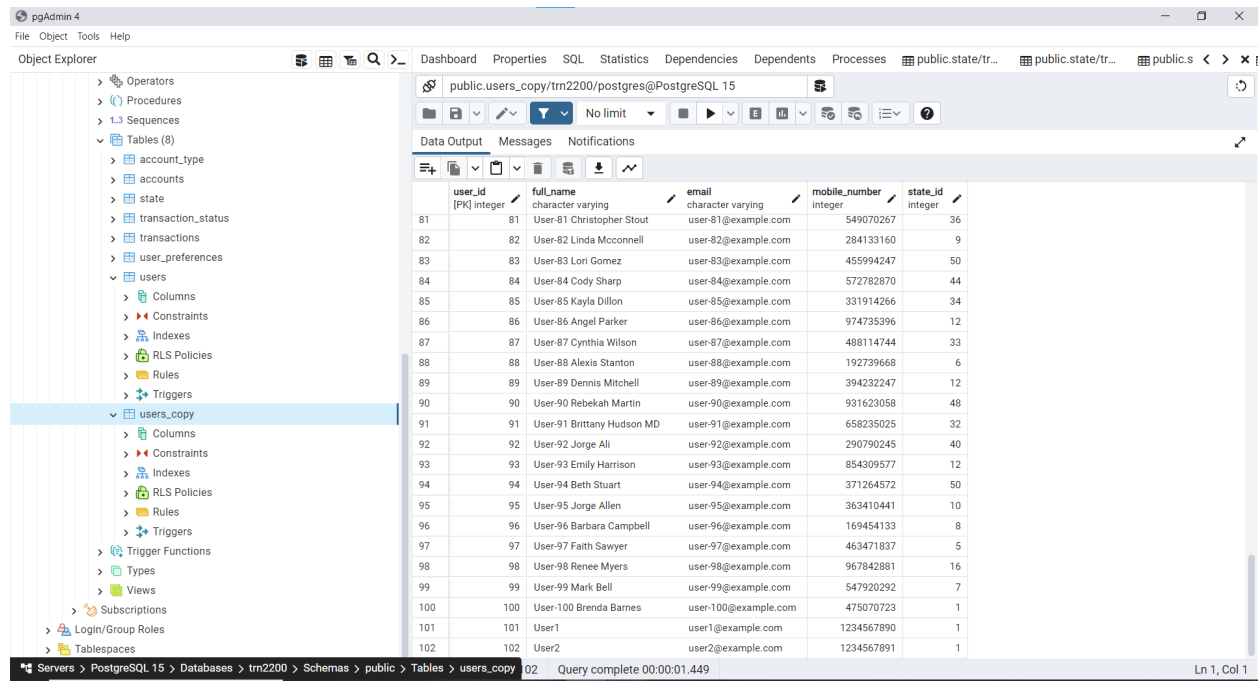


Figure 6: Replica Table Creation

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.



user_id [PK] integer	full_name character varying	email character varying	mobile_number integer	state_id integer
81	User-81 Christopher Stout	user-81@example.com	549070267	36
82	User-82 Linda Mcconnell	user-82@example.com	284133160	9
83	User-83 Lori Gomez	user-83@example.com	455994247	50
84	User-84 Cody Sharp	user-84@example.com	572762870	44
85	User-85 Kayla Dillon	user-85@example.com	331914266	34
86	User-86 Angel Parker	user-86@example.com	974735396	12
87	User-87 Cynthia Wilson	user-87@example.com	468114744	33
88	User-88 Alexis Stanton	user-88@example.com	192739668	6
89	User-89 Dennis Mitchell	user-89@example.com	394232247	12
90	User-90 Rebekah Martin	user-90@example.com	931623058	48
91	User-91 Brittany Hudson MD	user-91@example.com	658235025	32
92	User-92 Jorge Ali	user-92@example.com	290790245	40
93	User-93 Emily Harrison	user-93@example.com	854309577	12
94	User-94 Beth Stuart	user-94@example.com	371264572	50
95	User-95 Jorge Allen	user-95@example.com	363410441	10
96	User-96 Barbara Campbell	user-96@example.com	169454133	8
97	User-97 Faith Sawyer	user-97@example.com	463471837	5
98	User-98 Renee Myers	user-98@example.com	967842881	16
99	User-99 Mark Bell	user-99@example.com	547920292	7
100	User-100 Brenda Barnes	user-100@example.com	475070723	1
101	User1	user1@example.com	1234567890	1
102	User2	user2@example.com	1234567891	1

Figure 7: User's copy Table

pgAdmin 4

File Object Tools Help

Object Explorer

- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (8)
 - account_type
 - accounts
 - state
 - transaction_status
 - transactions
 - user_preferences
 - users**
 - users_copy
 - Columns
 - Constraints
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
 - Trigger Functions
 - Types
 - Views
- Subscriptions
- Login/Group Roles
- Tablespaces

Dashboard Properties SQL Statistics Dependencies Dependents Processes public.state/tr...

public.users/trn2200/postgres@PostgreSQL 15

Data Output Messages Notifications

	user_id [PK] integer	full_name character varying	email character varying	mobile_number integer	state_id integer
81	81	User-81 Christopher Stout	user-81@example.com	549070267	36
82	82	User-82 Linda Mcconnell	user-82@example.com	284133160	9
83	83	User-83 Lori Gomez	user-83@example.com	455994247	50
84	84	User-84 Cody Sharp	user-84@example.com	572782870	44
85	85	User-85 Kayla Dillon	user-85@example.com	331914266	34
86	86	User-86 Angel Parker	user-86@example.com	974735396	12
87	87	User-87 Cynthia Wilson	user-87@example.com	488114744	33
88	88	User-88 Alexis Stanton	user-88@example.com	192739668	6
89	89	User-89 Dennis Mitchell	user-89@example.com	394232247	12
90	90	User-90 Rebekah Martin	user-90@example.com	931623058	48
91	91	User-91 Brittany Hudson MD	user-91@example.com	658235025	32
92	92	User-92 Jorge Ali	user-92@example.com	290790245	40
93	93	User-93 Emily Harrison	user-93@example.com	854309577	12
94	94	User-94 Beth Stuart	user-94@example.com	371264572	50
95	95	User-95 Jorge Allen	user-95@example.com	363410441	10
96	96	User-96 Barbara Campbell	user-96@example.com	169454133	8
97	97	User-97 Faith Sawyer	user-97@example.com	463471837	5
98	98	User-98 Renee Myers	user-98@example.com	967842881	16
99	99	User-99 Mark Bell	user-99@example.com	547920292	7
100	100	User-100 Brenda Barnes	user-100@example.com	475070723	1
101	101	User1	user1@example.com	1234567890	1
102	102	User2	user2@example.com	1234567891	1

Servers > PostgreSQL 15 > Databases > trn2200 > Schemas > public > Tables > users 2 of 102 Query complete 00:00:01.853

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.

Object Explorer

- FTS Dictionaries
- FTS Parsers
- FTS Templates
- Foreign Tables
- Functions
- Materialized Views
- Operators
- Procedures
- Sequences
- Tables (8)
 - account_type
 - accounts
 - state
 - transaction_status
 - transactions
 - user_preferences
 - users**
 - users_copy
 - Columns
 - Constraints
 - Indexes
 - RLS Policies
 - Rules
 - Triggers
 - Trigger Functions
 - Types
 - Views
- Subscriptions
- Login/Group Roles
- Tablespaces
- abhishek

Dashboard Properties SQL Statistics Dependencies Dependents Processes public.state/tr...

public.users/trn2200/postgres@PostgreSQL 15

No limit

Data Output Messages Notifications

	user_id [PK] integer	full_name character varying	email character varying	mobile_number integer	state_id integer
81	81	User-81 Christopher Stout	user-81@example.com	549070267	36
82	82	User-82 Linda Mcconnell	user-82@example.com	284133160	9
83	83	User-83 Lori Gomez	user-83@example.com	455994247	50
84	84	User-84 Cody Sharp	user-84@example.com	572782870	44
85	85	User-85 Kayla Dillon	user-85@example.com	331914266	34
86	86	User-86 Angel Parker	user-86@example.com	974735396	12
87	87	User-87 Cynthia Wilson	user-87@example.com	488114744	33
88	88	User-88 Alexis Stanton	user-88@example.com	192739668	6
89	89	User-89 Dennis Mitchell	user-89@example.com	394232247	12
90	90	User-90 Rebekah Martin	user-90@example.com	931623058	48
91	91	User-91 Brittany Hudson MD	user-91@example.com	658235025	32
92	92	User-92 Jorge Ali	user-92@example.com	290790245	40
93	93	User-93 Emily Harrison	user-93@example.com	854309577	12
94	94	User-94 Beth Stuart	user-94@example.com	371264572	50
95	95	User-95 Jorge Allen	user-95@example.com	363410441	10
96	96	User-96 Barbara Campbell	user-96@example.com	169454133	8
97	97	User-97 Faith Sawyer	user-97@example.com	463471837	5
98	98	User-98 Renee Myers	user-98@example.com	967842881	16
99	99	User-99 Mark Bell	user-99@example.com	547920292	7
100	100	User-100 Brenda Barnes	user-100@example.com	475070723	1
101	101	UserX1	user1@example.com	1234567890	1
102	102	User2	user2@example.com	1234567891	1

Total rows: 102 of 102 Query complete 00:00:01.853 Rows selected: 1 Changes staged: Updated: 1

Figure 9: Renaming the data in the User Table

Dashboard Properties SQL Statistics Dependencies Dependents Processes pu

public.users_copy/trn2200/postgres@PostgreSQL 15

No limit

Data Output Messages Notifications

	user_id [PK] integer	full_name character varying	email character varying	mobile_number integer	state_id integer
1	101	UserX1	user1@example.com	1234567890	1

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