Distributed HDMR (Team name)

Dr. Bharatesh Chakravarthi

CSE512

27 November 2023

Part 2: Fragmentation and Replication Techniques

- **Introduction**

The second part of this project, "EcoSphere: Distributed Commerce Gateway", looks into the implementation of advanced database techniques—specifically, fragmentation and replication—to optimize the performance of our distributed database system. We will explore both horizontal and vertical fragmentation, aiming to tailor our database tables into more manageable and efficient subsets. Horizontal fragmentation will involve splitting tables into subsets based on specific criteria, making data queries and updates more efficient. Vertical fragmentation, on the other hand, will focus on dividing tables into smaller columns to optimize data retrieval and reduce unnecessary data transmission. Additionally, we will configure a master-slave replication model.

- **Implementation Tools and Languages**

  Tools used: VS code, pgAdmin, Git

  Database systems used: PostgreSQL

  Languages: Python, SQL

- **Fragmentation**

  1. **Horizontal Fragmentation:**

     We have implemented horizontal partitioning on the Users table based on the “registration_date” column. This partitioning has led to the creation of two distinct tables: users_new and users_old. The users_new table has recent registrants (from year 2016 to 2023), ensuring quick access and efficient handling of current user data, which is likely to be accessed more frequently. Conversely, the users_old table contains older registration records (from 2010 to 2015). This separation aligns with our data access patterns, allowing for more efficient query processing. We can see the code for creating the partition tables below -

```
USERS_QUERY = f"""
            CREATE TABLE IF NOT EXISTS {TABLE_NAMES[0]} (
    user_id SERIAL,
    username TEXT NOT NULL,
    first_name TEXT NOT NULL,
    last_name TEXT NOT NULL,
    email TEXT NOT NULL,
    password TEXT NOT NULL,
    address TEXT,
    phone_number TEXT,
    registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
    PRIMARY KEY (user_id, registration_date)
)
PARTITION BY RANGE (registration_date);
"""
```

```
# Fragment on Users - Range partitioning
def create_partition_table_users(conn):
    try:
        cursor = conn.cursor()
        # Creating partition tables
        cursor.execute(f"""CREATE TABLE IF NOT EXISTS Users_Old PARTITION OF Users
                          FOR VALUES FROM ('2010-01-01 00:00:00') TO ('2015-12-31 00:00:00');""")

        cursor.execute(f"""CREATE TABLE IF NOT EXISTS Users_New PARTITION OF Users
                          FOR VALUES FROM ('2016-01-01 00:00:00') TO ('2023-12-31 00:00:00');""")
        conn.commit()

        print("Tables with horizontal partitions created successfully!")

    except (Exception, psycopg2.Error) as error:
        print("Error creating partitions:", error)
```

Fig 1: Code showing Users SQL query (above) and creation of horizontal partitions (below)

This can also be seen in pgAdmin, with Users table marked with a “P” superscript and the partitions demonstrated as below -

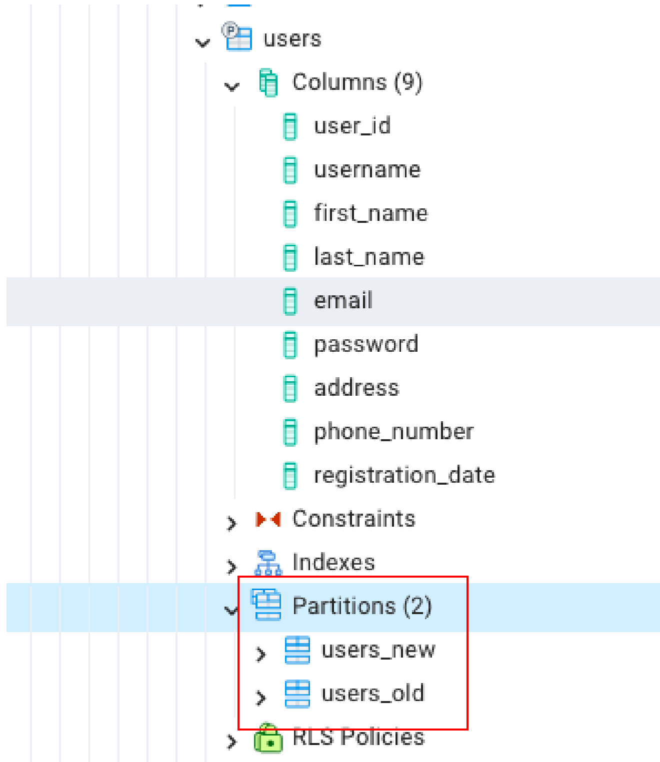
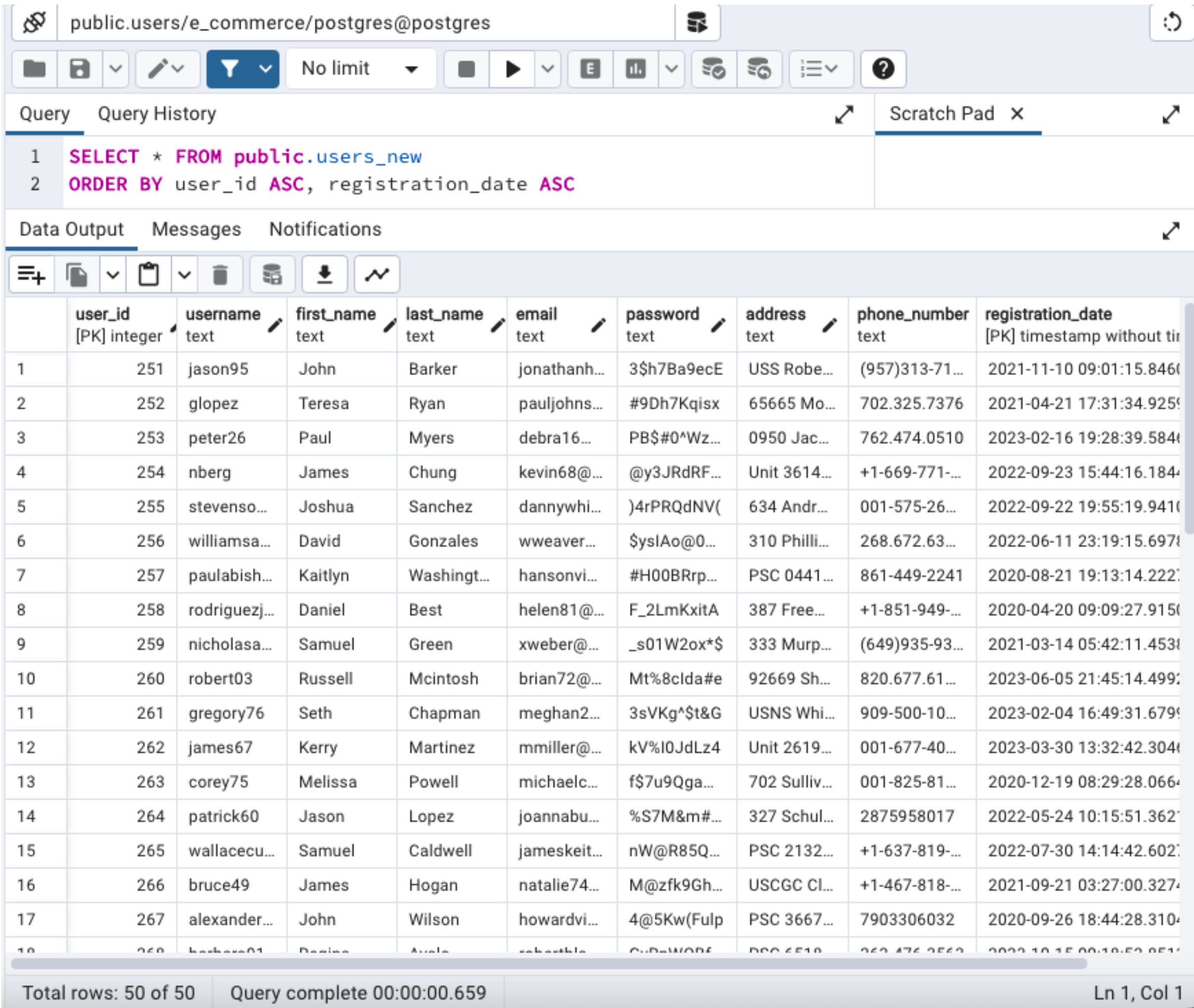


Fig 2: pgAdmin screenshot showing horizontal partitions of Users table

The initial mock data that was inserted into the Users table in Part 1, consisted of user registration date from the current decade (*registration_date = fake.date_time_this_decade()*). Hence, the Users_old partition table should have been empty and the Users_new partition table should contain all the 50 rows of data inserted, as can be confirmed from the images below-



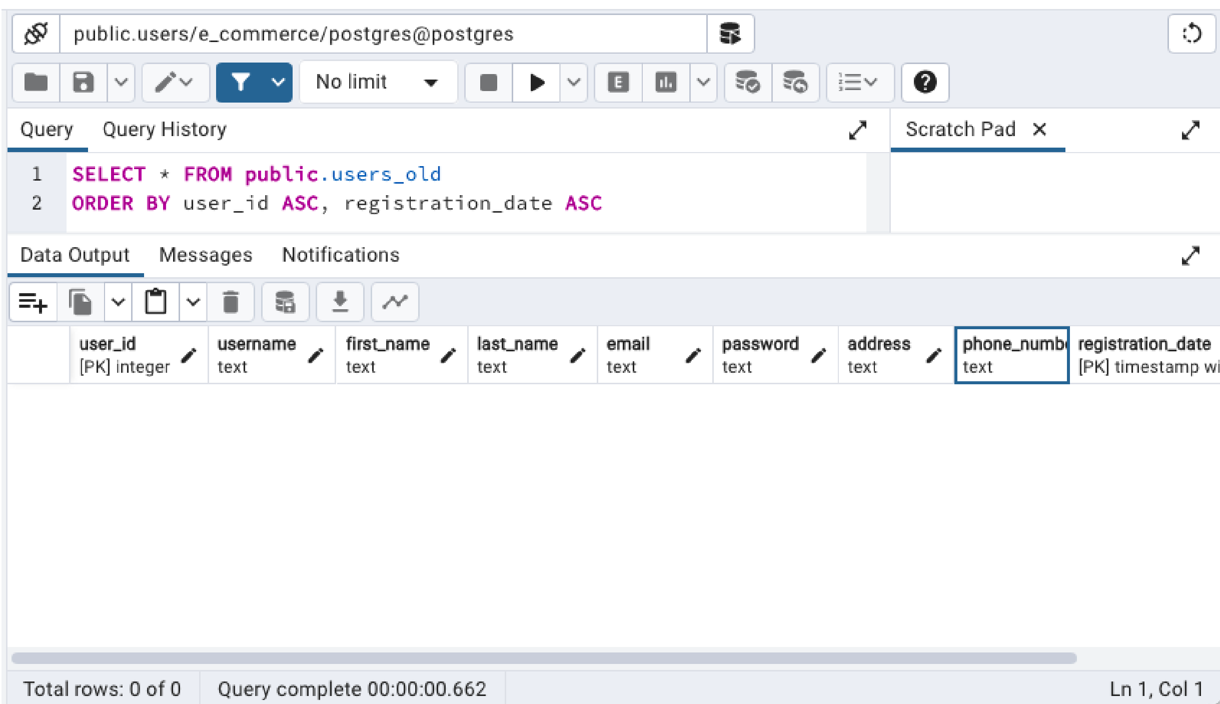


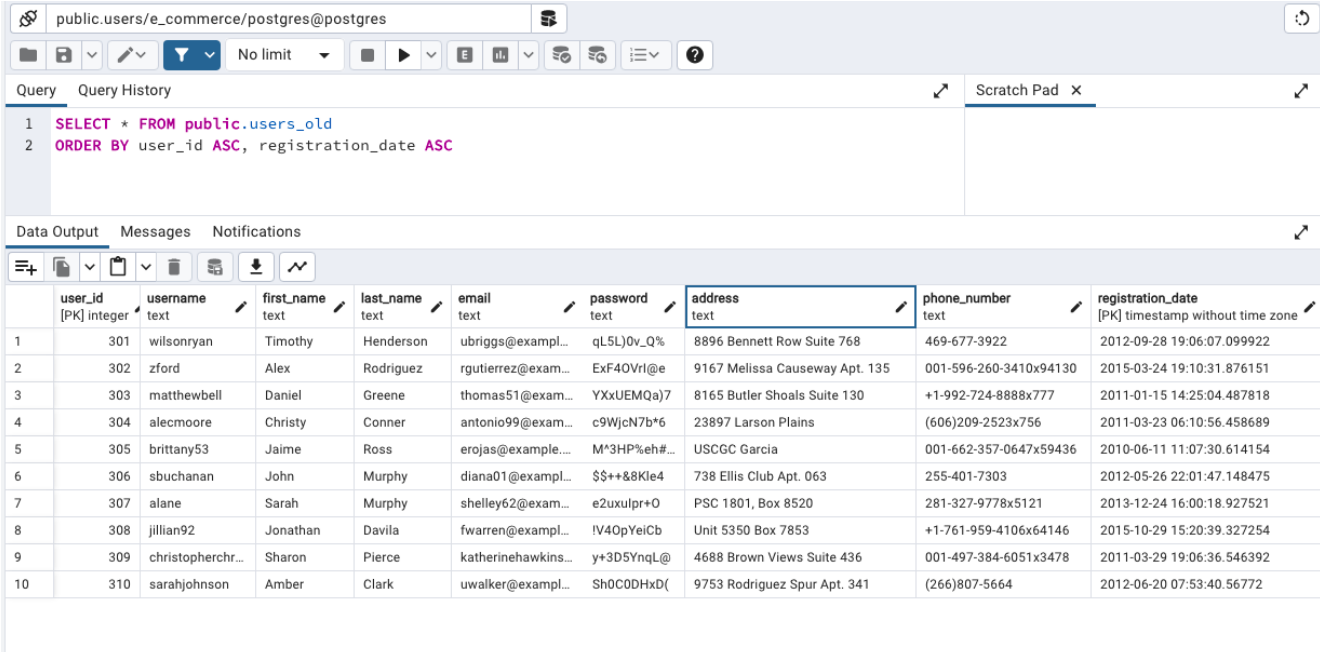
Fig 3: Screenshot showing data in Users_old and Users_new

Further, we are inserting 10 rows of data into the users table with registration year before 2015, and 5 rows of data with registration date in the current decade as shown below -

```
Part 2 > fragmentation.py > demonstrate_H_partition

def demonstrate_H_partition(conn):
    print("\n---------HORIZONTAL PARTITIONING ------------\n")
    fake = Faker()
    try:
        cursor = conn.cursor()
        if conn and cursor:
            for _ in range(10):
                username = fake.user_name()
                first_name = fake.first_name()
                last_name = fake.last_name()
                email = fake.email()
                password = fake.password()
                address = fake.address()
                phone_number = fake.phone_number()
                registration_date = fake.date_time_between_dates(datetime_start=datetime(2010,1,1), datetime_end=datetime(2015,12,31))

                cursor.execute(f"""
                    INSERT INTO {table_names[0]} (username, first_name, last_name, email, password, address, phone_number, registration_date)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """, (username, first_name, last_name, email, password, address, phone_number, registration_date))
            for _ in range(5):
                username = fake.user_name()
                first_name = fake.first_name()
                last_name = fake.last_name()
                email = fake.email()
                password = fake.password()
                address = fake.address()
                phone_number = fake.phone_number()
                registration_date = registration_date = fake.date_time_this_decade()

                cursor.execute(f"""
                    INSERT INTO {table_names[0]} (username, first_name, last_name, email, password, address, phone_number, registration_date)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
                """, (username, first_name, last_name, email, password, address, phone_number, registration_date))
            conn.commit()
```

Fig 4: Code showing insertion of data into Users to demonstrate partiotioning

It is expected that the data should automatically be partitioned into the Users_old table and Users_new table, making it easier to handle and query for large databases. This can be verified from the below image -



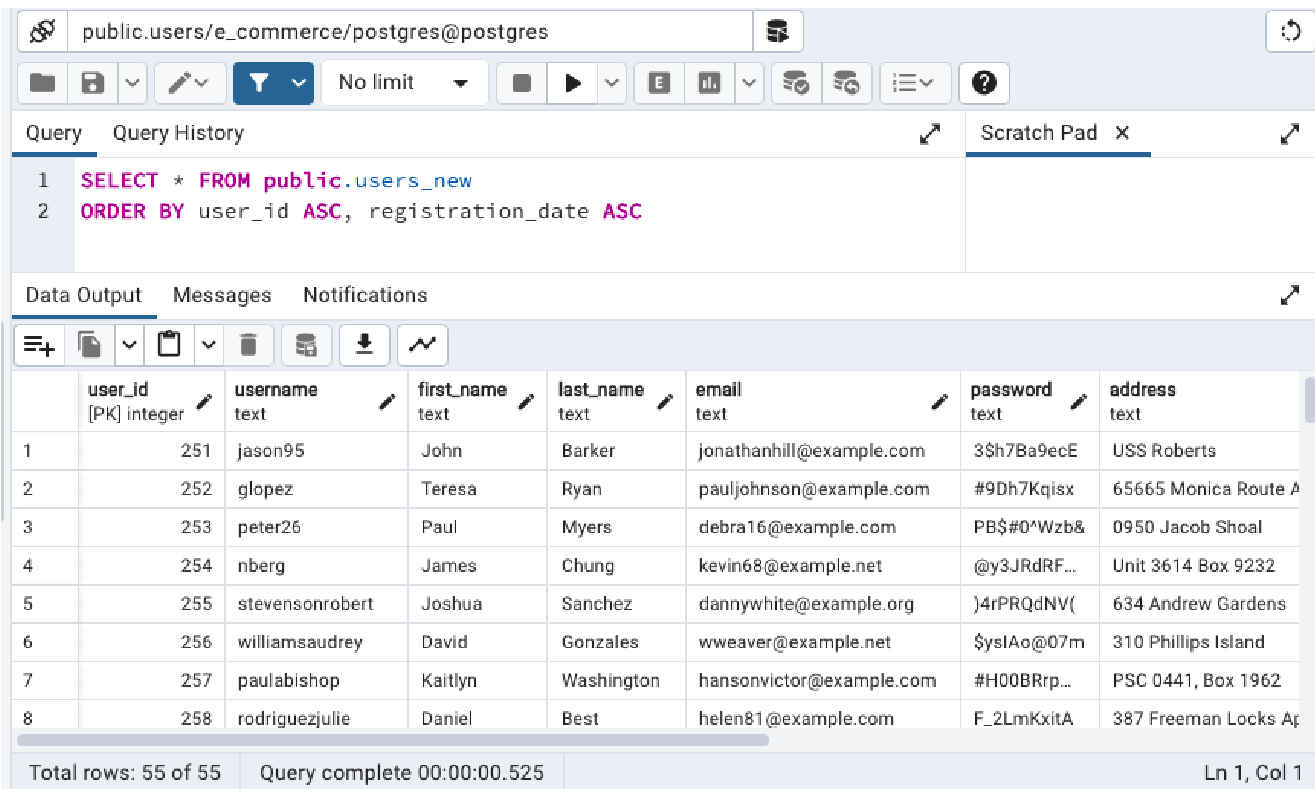


Fig 5: Data retrieval on pgAdmin showing successful horizontal partitioning

As we can see in the above figures, the 15 rows of new that were inserted into the Users table got partitioned into the users_old and users_new tables, which are partitions of the main table. Thus we have 10 rows of old data in users_old and 55 rows of new data in users_new.

●.2. Vertical fragmentation:

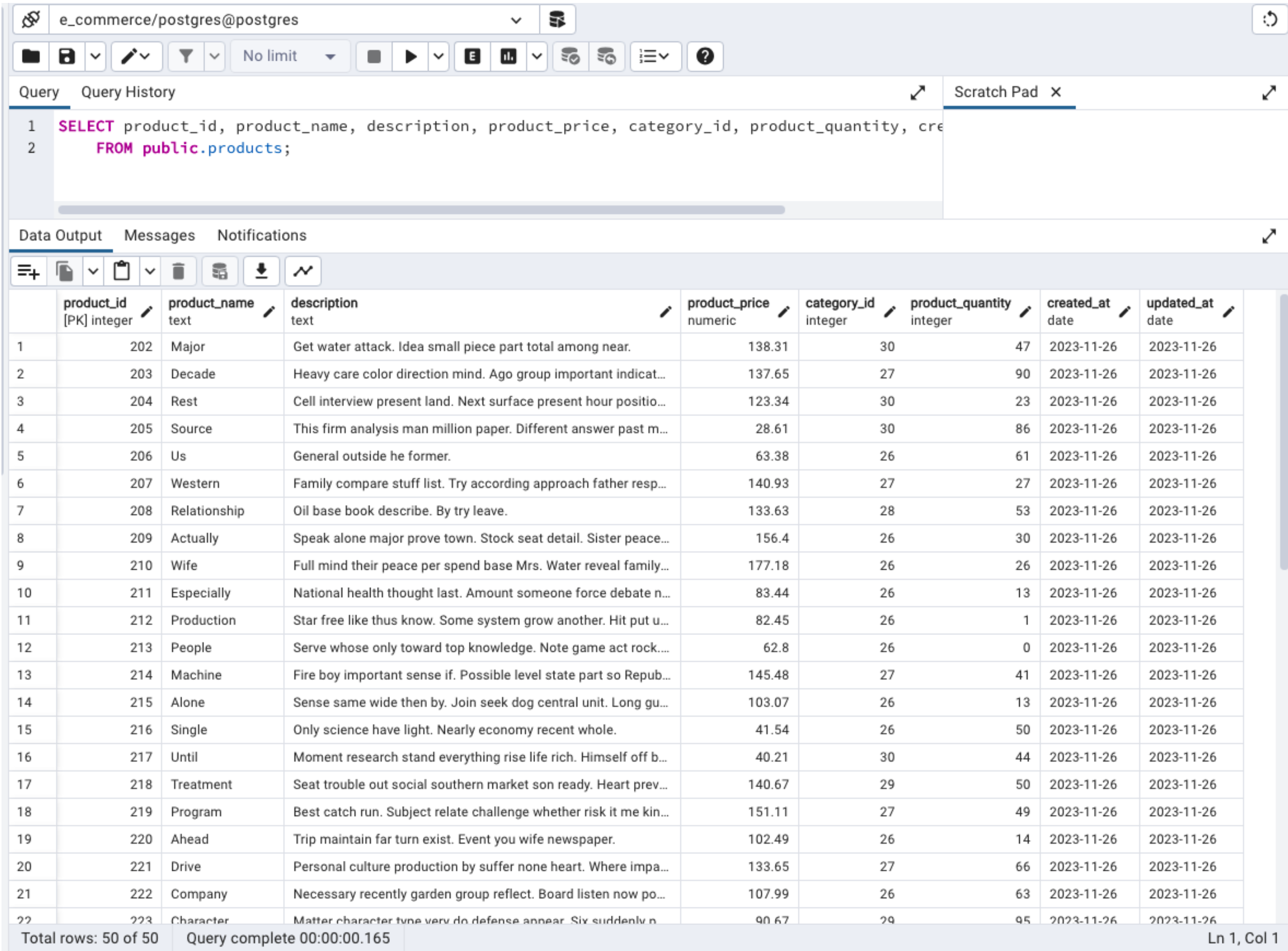
We have implemented vertical fragmentation on the Products table, which resulted in the creation of two specialized tables: Products_Info and Products_Details. The Products_Info table now contains key product attributes such as product_id, product_name, description, product_price, category_id, and product_quantity. This table is tailored to serve queries primarily focused on product information and pricing, streamlining data retrieval for these common requests. On the other hand, the Products_Details table has product_id, created_at, and updated_at, for tracking and managing the product lifecycle and update history. Since postgres does not directly support vertical fragmentation, we have inserted the data into the fragments using the data from the main Products table. Following is the code for showing vertical fragmentation -

```python
def vertical_partitioning(conn):
    try:
        cursor = conn.cursor()
        # Creating partition tables
        cursor.execute("CREATE TABLE IF NOT EXISTS Products_Info AS SELECT product_id, product_name, description,
        product_price, category_id, product_quantity FROM Products;")
        cursor.execute("CREATE TABLE IF NOT EXISTS Products_Details AS SELECT product_id, created_at, updated_at FROM
        Products;")

        conn.commit()

        print("Tables with vertical partitions created successfully!")

    except (Exception, psycopg2.Error) as error:
        print("Error creating partitions:", error)
```

```python
def demonstrate_V_partition(conn):
    print("\n---------VERTICAL PARTITIONING ------------\n")
    fake = Faker()
    try:
        cursor = conn.cursor()
        if conn and cursor:
            cat_ids = []
            cursor.execute("SELECT category_id FROM Categories")
            cat_ids = [row[0] for row in cursor.fetchall()]
            for _ in range(5):
                product_name = fake.word().capitalize()
                description = fake.text(max_nb_chars=100)
                product_price = round(random.uniform(5, 200), 2)
                category_id = random.choice(cat_ids)
                product_quantity = random.randint(0, 100)
                cursor.execute(f"""
                    INSERT INTO {table_names[2]} (product_name, description, product_price, category_id,
                    product_quantity)
                    VALUES (%s, %s, %s, %s, %s)
                """, (product_name, description, product_price, category_id, product_quantity))

            conn.commit()
```

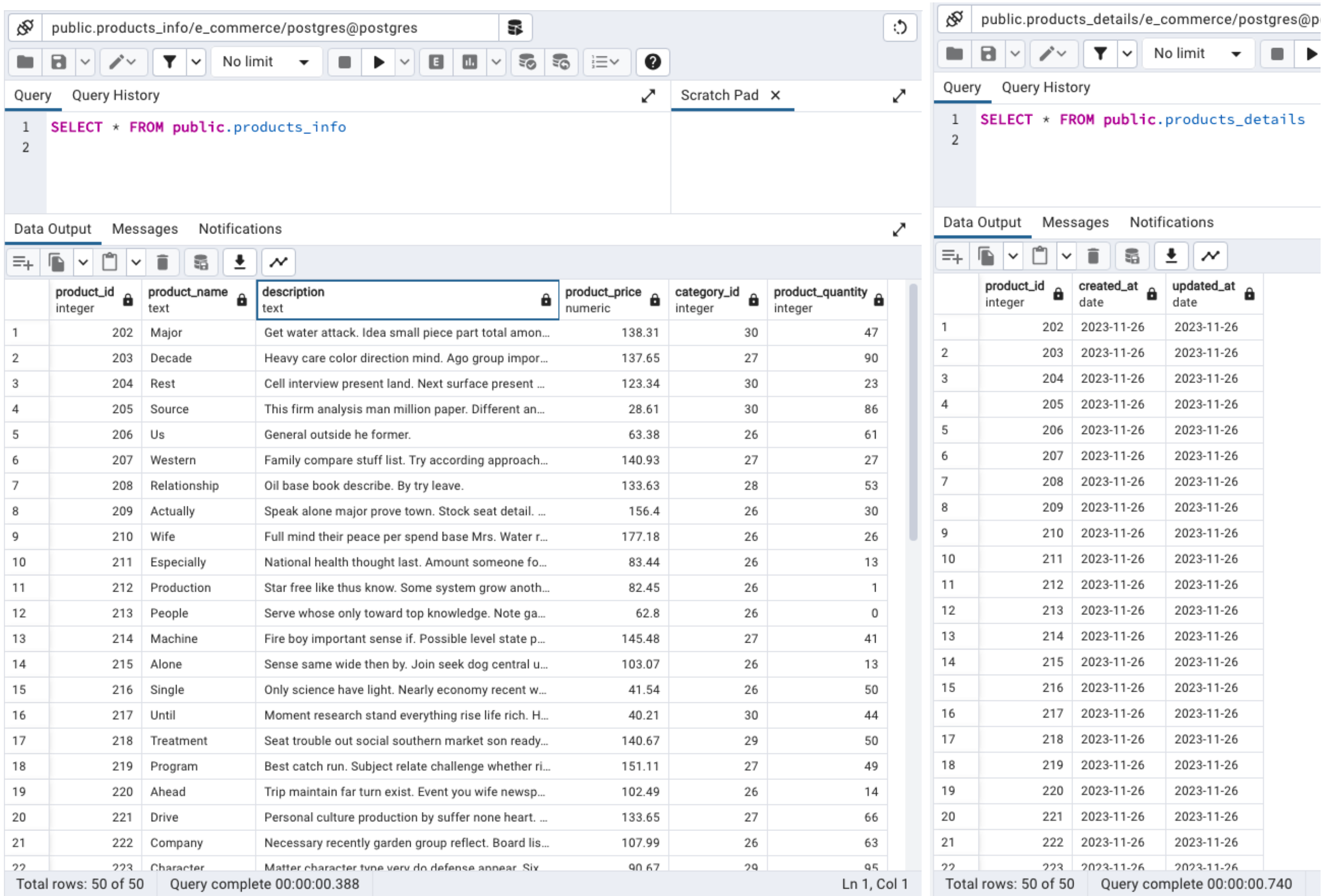
Fig 6: Code screenshot showing vertical fragmentation

Following is the data from the main Products table as well from the two vertical fragments Products_Info and Products_Details -



	product_id [PK] integer	product_name text	description text	product_price numeric	category_id integer	product_quantity integer	created_at date	updated_at date
1	202	Major	Get water attack. Idea small piece part total among near.	138.31	30	47	2023-11-26	2023-11-26
2	203	Decade	Heavy care color direction mind. Ago group important indicat...	137.65	27	90	2023-11-26	2023-11-26
3	204	Rest	Cell interview present land. Next surface present hour positio...	123.34	30	23	2023-11-26	2023-11-26
4	205	Source	This firm analysis man million paper. Different answer past m...	28.61	30	86	2023-11-26	2023-11-26
5	206	Us	General outside he former.	63.38	26	61	2023-11-26	2023-11-26
6	207	Western	Family compare stuff list. Try according approach father resp...	140.93	27	27	2023-11-26	2023-11-26
7	208	Relationship	Oil base book describe. By try leave.	133.63	28	53	2023-11-26	2023-11-26
8	209	Actually	Speak alone major prove town. Stock seat detail. Sister peace...	156.4	26	30	2023-11-26	2023-11-26
9	210	Wife	Full mind their peace per spend base Mrs. Water reveal family...	177.18	26	26	2023-11-26	2023-11-26
10	211	Especially	National health thought last. Amount someone force debate n...	83.44	26	13	2023-11-26	2023-11-26
11	212	Production	Star free like thus know. Some system grow another. Hit put u...	82.45	26	1	2023-11-26	2023-11-26
12	213	People	Serve whose only toward top knowledge. Note game act rock....	62.8	26	0	2023-11-26	2023-11-26
13	214	Machine	Fire boy important sense if. Possible level state part so Repub...	145.48	27	41	2023-11-26	2023-11-26
14	215	Alone	Sense same wide then by. Join seek dog central unit. Long gu...	103.07	26	13	2023-11-26	2023-11-26
15	216	Single	Only science have light. Nearly economy recent whole.	41.54	26	50	2023-11-26	2023-11-26
16	217	Until	Moment research stand everything rise life rich. Himself off b...	40.21	30	44	2023-11-26	2023-11-26
17	218	Treatment	Seat trouble out social southern market son ready. Heart prev...	140.67	29	50	2023-11-26	2023-11-26
18	219	Program	Best catch run. Subject relate challenge whether risk it me kin...	151.11	27	49	2023-11-26	2023-11-26
19	220	Ahead	Trip maintain far turn exist. Event you wife newspaper.	102.49	26	14	2023-11-26	2023-11-26
20	221	Drive	Personal culture production by suffer none heart. Where impa...	133.65	27	66	2023-11-26	2023-11-26
21	222	Company	Necessary recently garden group reflect. Board listen now po...	107.99	26	63	2023-11-26	2023-11-26
22	223	Character	Matter character type very do defense appear. Six suddenly n...	90.67	29	95	2023-11-26	2023-11-26

Fig 7: Screenshot of data from Products table



	product_id integer	product_name text	description text	product_price numeric	category_id integer	product_quantity integer
1	202	Major	Get water attack. Idea small piece part total amon...	138.31	30	47
2	203	Decade	Heavy care color direction mind. Ago group impor...	137.65	27	90
3	204	Rest	Cell interview present land. Next surface present ...	123.34	30	23
4	205	Source	This firm analysis man million paper. Different an...	28.61	30	86
5	206	Us	General outside he former.	63.38	26	61
6	207	Western	Family compare stuff list. Try according approach...	140.93	27	27
7	208	Relationship	Oil base book describe. By try leave.	133.63	28	53
8	209	Actually	Speak alone major prove town. Stock seat detail. ...	156.4	26	30
9	210	Wife	Full mind their peace per spend base Mrs. Water r...	177.18	26	26
10	211	Especially	National health thought last. Amount someone fo...	83.44	26	13
11	212	Production	Star free like thus know. Some system grow anoth...	82.45	26	1
12	213	People	Serve whose only toward top knowledge. Note ga...	62.8	26	0
13	214	Machine	Fire boy important sense if. Possible level state p...	145.48	27	41
14	215	Alone	Sense same wide then by. Join seek dog central u...	103.07	26	13
15	216	Single	Only science have light. Nearly economy recent w...	41.54	26	50
16	217	Until	Moment research stand everything rise life rich. H...	40.21	30	44
17	218	Treatment	Seat trouble out social southern market son ready...	140.67	29	50
18	219	Program	Best catch run. Subject relate challenge whether ri...	151.11	27	49
19	220	Ahead	Trip maintain far turn exist. Event you wife newsp...	102.49	26	14
20	221	Drive	Personal culture production by suffer none heart. ...	133.65	27	66
21	222	Company	Necessary recently garden group reflect. Board lis...	107.99	26	63
22	223	Character	Matter character type very do defense appear. Six...	90.67	29	95

	product_id integer	created_at date	updated_at date
1	202	2023-11-26	2023-11-26
2	203	2023-11-26	2023-11-26
3	204	2023-11-26	2023-11-26
4	205	2023-11-26	2023-11-26
5	206	2023-11-26	2023-11-26
6	207	2023-11-26	2023-11-26
7	208	2023-11-26	2023-11-26
8	209	2023-11-26	2023-11-26
9	210	2023-11-26	2023-11-26
10	211	2023-11-26	2023-11-26
11	212	2023-11-26	2023-11-26
12	213	2023-11-26	2023-11-26
13	214	2023-11-26	2023-11-26
14	215	2023-11-26	2023-11-26
15	216	2023-11-26	2023-11-26
16	217	2023-11-26	2023-11-26
17	218	2023-11-26	2023-11-26
18	219	2023-11-26	2023-11-26
19	220	2023-11-26	2023-11-26
20	221	2023-11-26	2023-11-26
21	222	2023-11-26	2023-11-26
22	223	2023-11-26	2023-11-26

Fig 8: Screenshot of data from the two fragments Product_Details and Product_Info table

By dividing the Users table into users_new and users_old based on registration dates, and segmenting the Products table into Products_Info and Products_Details, we have effectively tailored our database structure to align with specific data access patterns and usage scenarios. This optimization not only enhances query performance in case of large databases by reducing data load and improving retrieval times but also helps in efficient resource management.

- **Replication**

  The goal of this project was to establish a robust, real-time replication mechanism that ensures high data availability and disaster recovery. Streaming replication in PostgreSQL was chosen for its efficiency, low latency, and the ability to provide a near real-time copy of data from the primary to the replica node.

System Architecture:

The system consists of two nodes: the **primary node**, which handles all the database write operations, and the **replica node**, which mirrors the primary node's data. The replica node operates in a read-only mode until a failover is triggered.

Configuration of PostgreSQL Streaming Replication:

- Configuration involved setting up the primary node with the necessary parameters in **postgresql.conf and pg_hba.conf** files to enable replication.
- On the replica node, similar configuration changes were made, and the node was set up to stream data from the primary node, using PostgreSQL's built-in replication functionality.

Testing and Validation:

- Methods used to test the replication process, including scenarios such as creating, updating, and deleting records in the primary node and verifying these changes in the replica node in real-time.
- Testing failover scenarios to ensure the replica node can seamlessly take over as the primary node in case of a failure.

Conclusion

In this project, we have successfully implemented PostgreSQL streaming replication within a two-node architecture and fragmentation, comprising a primary node and a replica node. This setup has proven to be an efficient and robust solution for achieving real-time data replication, thereby significantly enhancing data availability and providing a solid foundation for disaster recovery. The meticulous configuration and validation of the replication process ensured that the system adhered to the key principles of data integrity and consistency. Our choice of streaming replication was instrumental in maintaining near real-time synchronization between the primary and replica nodes, with minimal replication lag and high fidelity of data.

The testing phase of the project, which included various scenarios of data manipulation and failover, further underscored the resilience and reliability of our system. The replica node’s seamless takeover capabilities in failover situations ensured uninterrupted database service, thus maintaining continuous data accessibility. This successful deployment of streaming replication in a PostgreSQL environment showcases the importance of strategic planning and execution in database management, particularly in scenarios that demand high availability and data consistency. T

References

[1] *Streaming replication*. Streaming Replication - PostgreSQL wiki. (n.d.). https://wiki.postgresql.org/wiki/Streaming_Replication

[2] "55.4. Streaming Replication Protocol." *PostgreSQL Documentation*, 9 Nov. 2023, www.postgresql.org/docs/current/protocol-replication.html.