

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 -

```

17  USERS_QUERY = f"""
18      CREATE TABLE IF NOT EXISTS {TABLE_NAMES[0]} (
19          user_id SERIAL,
20          username TEXT NOT NULL,
21          first_name TEXT NOT NULL,
22          last_name TEXT NOT NULL,
23          email TEXT NOT NULL,
24          password TEXT NOT NULL,
25          address TEXT,
26          phone_number TEXT,
27          registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
28          PRIMARY KEY (user_id, registration_date)
29      )
30      PARTITION BY RANGE (registration_date);
31  """

```

```

98  # Fragment on Users - Range partitioning
99  def create_partition_table_users(conn):
100      try:
101          cursor = conn.cursor()
102          # Creating partition tables
103          cursor.execute(f"""CREATE TABLE IF NOT EXISTS Users_Old PARTITION OF Users
104              FOR VALUES FROM ('2010-01-01 00:00:00') TO ('2015-12-31 00:00:00');""")
105
106          cursor.execute(f"""CREATE TABLE IF NOT EXISTS Users_New PARTITION OF Users
107              FOR VALUES FROM ('2016-01-01 00:00:00') TO ('2023-12-31 00:00:00');""")
108          conn.commit()
109
110          print("Tables with horizontal partitions created successfully!")
111
112      except (Exception, psycopg2.Error) as error:
113          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 -

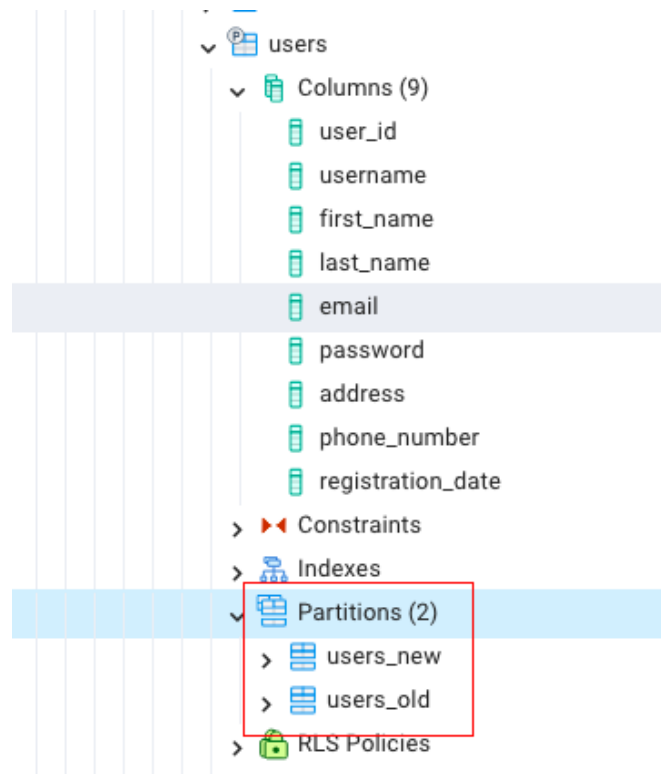


Fig 2: pgAdmin screenshot showing horizontal partitions of Users table

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-

public.users/e\_commerce/postgres@postgres

Query: `SELECT * FROM public.users_new ORDER BY user_id ASC, registration_date ASC`

Data Output

	user_id [PK] integer	username text	first_name text	last_name text	email text	password text	address text	phone_number text	registration_date [PK] timestamp without time zone
1	251	jason95	John	Barker	jonathanh...	3\$h7Ba9ecE	USS Robe...	(957)313-71...	2021-11-10 09:01:15.846
2	252	glopez	Teresa	Ryan	pauljohns...	#9Dh7Kqisx	65665 Mo...	702.325.7376	2021-04-21 17:31:34.925
3	253	peter26	Paul	Myers	debra16...	PB\$#0*Wz...	0950 Jac...	762.474.0510	2023-02-16 19:28:39.584
4	254	nberg	James	Chung	kevin68@...	@y3JRdRF...	Unit 3614...	+1-669-771...	2022-09-23 15:44:16.184
5	255	stevenso...	Joshua	Sanchez	dannywhi...	)4rPRQdNV(	634 Andr...	001-575-26...	2022-09-22 19:55:19.941
6	256	williamsa...	David	Gonzales	wweaver...	\$ysIAo@0...	310 Philli...	268.672.63...	2022-06-11 23:19:15.697
7	257	paulabish...	Kaitlyn	Washingt...	hansonvi...	#H00BRrp...	PSC 0441...	861-449-2241	2020-08-21 19:13:14.222
8	258	rodriguezj...	Daniel	Best	helen81@...	F_2LmKxitA	387 Free...	+1-851-949...	2020-04-20 09:09:27.915
9	259	nicholasa...	Samuel	Green	xweber@...	_s01W2ox*\$	333 Murp...	(649)935-93...	2021-03-14 05:42:11.453
10	260	robert03	Russell	Mcintosh	brian72@...	Mt%8clda#e	92669 Sh...	820.677.61...	2023-06-05 21:45:14.499
11	261	gregory76	Seth	Chapman	meghan2...	3sVKg*\$t&G	USNS Whi...	909-500-10...	2023-02-04 16:49:31.679
12	262	james67	Kerry	Martinez	mmiller@...	kV%IOJdLz4	Unit 2619...	001-677-40...	2023-03-30 13:32:42.304
13	263	corey75	Melissa	Powell	michaelc...	f\$7u9Qga...	702 Sulliv...	001-825-81...	2020-12-19 08:29:28.066
14	264	patrick60	Jason	Lopez	joannabu...	%S7M&m#...	327 Schul...	2875958017	2022-05-24 10:15:51.362
15	265	wallacecu...	Samuel	Caldwell	jameskeit...	nW@R85Q...	PSC 2132...	+1-637-819...	2022-07-30 14:14:42.602
16	266	bruce49	James	Hogan	natalie74...	M@zfk9Gh...	USCGC Cl...	+1-467-818...	2021-09-21 03:27:00.327
17	267	alexander...	John	Wilson	howardvi...	4@5Kw(Fulp	PSC 3667...	7903306032	2020-09-26 18:44:28.310
18	268	harkness01	Danica	Avila	seabattle...	CuDaWQDF...	PSC 6518...	962.476.2562	2022-10-15 00:18:52.051

Total rows: 50 of 50    Query complete 00:00:00.659    Ln 1, Col 1

public.users/e\_commerce/postgres@postgres

Query: `SELECT * FROM public.users_old ORDER BY user_id ASC, registration_date ASC`

Data Output

	user_id [PK] integer	username text	first_name text	last_name text	email text	password text	address text	phone_number text	registration_date [PK] timestamp without time zone
--	-------------------------	------------------	--------------------	-------------------	---------------	------------------	-----------------	----------------------	---

Total rows: 0 of 0    Query complete 00:00:00.662    Ln 1, Col 1

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
21
22 def demonstrate_H_partition(conn):
23     print("\n-----HORIZONTAL PARTITIONING -----\\n")
24     fake = Faker()
25     try:
26         cursor = conn.cursor()
27         if conn and cursor:
28             for _ in range(10):
29                 username = fake.user_name()
30                 first_name = fake.first_name()
31                 last_name = fake.last_name()
32                 email = fake.email()
33                 password = fake.password()
34                 address = fake.address()
35                 phone_number = fake.phone_number()
36                 registration_date = fake.date_time_between_dates(datetime_start=datetime(2010,1,1), datetime_end=datetime(2015,12,31))
37
38                 cursor.execute(f"""
39                     INSERT INTO {table_names[0]} (username, first_name, last_name, email, password, address, phone_number, registration_date)
40                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
41                     """, (username, first_name, last_name, email, password, address, phone_number, registration_date))
42             for _ in range(5):
43                 username = fake.user_name()
44                 first_name = fake.first_name()
45                 last_name = fake.last_name()
46                 email = fake.email()
47                 password = fake.password()
48                 address = fake.address()
49                 phone_number = fake.phone_number()
50                 registration_date = fake.date_time_this_decade()
51
52                 cursor.execute(f"""
53                     INSERT INTO {table_names[0]} (username, first_name, last_name, email, password, address, phone_number, registration_date)
54                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
55                     """, (username, first_name, last_name, email, password, address, phone_number, registration_date))
56     except:
57         conn.rollback()
58     conn.commit()
```

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

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 -

public.users/e\_commerce/postgres@postgres

Query Query History Scratch Pad

```

1 SELECT * FROM public.users_old
2 ORDER BY user_id ASC, registration_date ASC

```

Data Output Messages Notifications

	user_id [PK] integer	username text	first_name text	last_name text	email text	password text	address text	phone_number text	registration_date [PK] timestamp without time zone
1	301	wilsonryan	Timothy	Henderson	ubriggs@exampl...	qL5L)0v_Q%	8896 Bennett Row Suite 768	469-677-3922	2012-09-28 19:06:07.099922
2	302	zford	Alex	Rodriguez	rgutierrez@exam...	ExF40Vrl@e	9167 Melissa Causeway Apt. 135	001-596-260-3410x94130	2015-03-24 19:10:31.876151
3	303	matthewbell	Daniel	Greene	thomas51@exam...	YXxUEMQa)7	8165 Butler Shoals Suite 130	+1-992-724-8888x777	2011-01-15 14:25:04.487818
4	304	alecmoore	Christy	Conner	antonio99@exam...	c9WjcN7b*6	23897 Larson Plains	(606)209-2523x756	2011-03-23 06:10:56.458689
5	305	brittany53	Jaime	Ross	erojas@example...	M*3HP%eh#...	USCGC Garcia	001-662-357-0647x59436	2010-06-11 11:07:30.614154
6	306	sbuchanan	John	Murphy	diana01@exampl...	\$S++&8Kle4	738 Ellis Club Apt. 063	255-401-7303	2012-05-26 22:01:47.148475
7	307	alane	Sarah	Murphy	shelley62@exam...	e2uxulpr+O	PSC 1801, Box 8520	281-327-9778x5121	2013-12-24 16:00:18.927521
8	308	jillian92	Jonathan	Davila	fwarren@exampl...	IV4OpYeiCb	Unit 5350 Box 7853	+1-761-959-4106x64146	2015-10-29 15:20:39.327254
9	309	christopherchr...	Sharon	Pierce	katherinehawkins...	y+3D5YnqL@	4688 Brown Views Suite 436	001-497-384-6051x3478	2011-03-29 19:06:36.546392
10	310	sarahjohnson	Amber	Clark	uwalker@exampl...	Sh0C0DHxD(	9753 Rodriguez Spur Apt. 341	(266)807-5664	2012-06-20 07:53:40.56772

public.users/e\_commerce/postgres@postgres

Query Query History Scratch Pad

```

1 SELECT * FROM public.users_new
2 ORDER BY user_id ASC, registration_date ASC

```

Data Output Messages Notifications

	user_id [PK] integer	username text	first_name text	last_name text	email text	password text	address text
1	251	jason95	John	Barker	jonathanhill@example.com	3\$h7Ba9ecE	USS Roberts
2	252	glopez	Teresa	Ryan	pauljohnson@example.com	#9Dh7Kqisx	65665 Monica Route A
3	253	peter26	Paul	Myers	debra16@example.com	PB\$#0*Wzb&	0950 Jacob Shoal
4	254	nberg	James	Chung	kevin68@example.net	@y3JRdRF...	Unit 3614 Box 9232
5	255	stevensonrobert	Joshua	Sanchez	dannywhite@example.org	)4rPRQdNV(	634 Andrew Gardens
6	256	williamsaudrey	David	Gonzales	ww Weaver@example.net	\$ysIAo@07m	310 Phillips Island
7	257	paulabishop	Kaitlyn	Washington	hansonvictor@example.com	#H00BRp...	PSC 0441, Box 1962
8	258	rodriguezjulie	Daniel	Best	helen81@example.com	F_2LmKxitA	387 Freeman Locks A

Total rows: 55 of 55 Query complete 00:00:00.525 Ln 1, Col 1

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 -

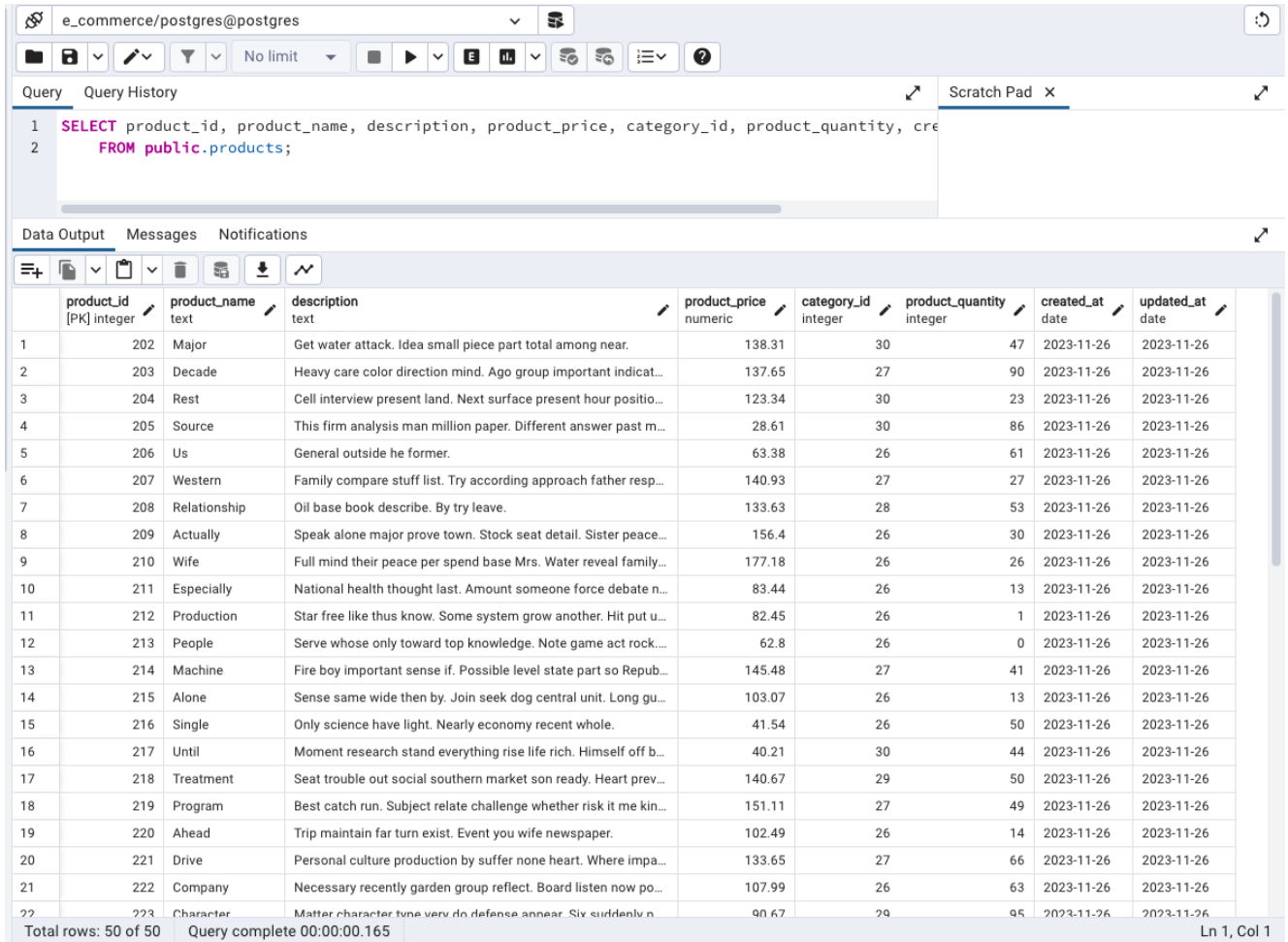
```

88  def vertical_partitioning(conn):
89      try:
90          cursor = conn.cursor()
91          # Creating partition tables
92          cursor.execute("CREATE TABLE IF NOT EXISTS Products_Info AS SELECT product_id, product_name, description,
93                          product_price, category_id, product_quantity FROM Products;")
94          cursor.execute("CREATE TABLE IF NOT EXISTS Products_Details AS SELECT product_id, created_at, updated_at FROM
95                          Products;")
96
97          conn.commit()
98
99          print("Tables with vertical partitions created successfully!")
100
101      except (Exception, psycopg2.Error) as error:
102          print("Error creating partitions:", error)
103
104
105  def demonstrate_V_partition(conn):
106      print("\n-----VERTICAL PARTITIONING -----\\n")
107      fake = Faker()
108      try:
109          cursor = conn.cursor()
110          if conn and cursor:
111              cat_ids = []
112              cursor.execute("SELECT category_id FROM Categories")
113              cat_ids = [row[0] for row in cursor.fetchall()]
114              for _ in range(5):
115                  product_name = fake.word().capitalize()
116                  description = fake.text(max_nb_chars=100)
117                  product_price = round(random.uniform(5, 200), 2)
118                  category_id = random.choice(cat_ids)
119                  product_quantity = random.randint(0, 100)
120                  cursor.execute(f"""
121                      INSERT INTO {table_names[2]} (product_name, description, product_price, category_id,
122                      product_quantity)
123                      VALUES (%s, %s, %s, %s, %s)
124                      """, (product_name, description, product_price, category_id, product_quantity))
125
126          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 -



The screenshot shows a database client interface with a query editor and a results pane. The query editor contains the following SQL query:

```
1 SELECT product_id, product_name, description, product_price, category_id, product_quantity, cre
2 FROM public.products;
```

The results pane displays a table with 10 columns: product\_id, product\_name, description, product\_price, category\_id, product\_quantity, created\_at, and updated\_at. The table contains 22 rows of data, with the first row being the header. The data is as follows:

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

The interface also shows a status bar at the bottom indicating "Total rows: 50 of 50" and "Query complete 00:00:00.165".

Fig 7: Screenshot of data from Products table



The image shows two side-by-side screenshots of a database query tool. The left screenshot shows a query for 'public.products\_info' with a 'SELECT \* FROM public.products\_info' statement. The right screenshot shows a query for 'public.products\_details' with a 'SELECT \* FROM public.products\_details' statement. Both screenshots show a 'Data Output' tab with a table of results.

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

product_id	created_at	updated_at
1	202	2023-11-26
2	203	2023-11-26
3	204	2023-11-26
4	205	2023-11-26
5	206	2023-11-26
6	207	2023-11-26
7	208	2023-11-26
8	209	2023-11-26
9	210	2023-11-26
10	211	2023-11-26
11	212	2023-11-26
12	213	2023-11-26
13	214	2023-11-26
14	215	2023-11-26
15	216	2023-11-26
16	217	2023-11-26
17	218	2023-11-26
18	219	2023-11-26
19	220	2023-11-26
20	221	2023-11-26
21	222	2023-11-26
22	223	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](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](http://www.postgresql.org/docs/current/protocol-replication.html).