

Distributed HDMR (Team name)

Dr. Bharatesh Chakravarthi

CSE512

27 November 2023

Part 1: Design and Implementation of a Distributed Database System

1. Introduction

This project "EcoSphere: Distributed Commerce Gateway" is a collaborative effort to design, implement, and optimize a distributed database system tailored for the E-commerce domain. It addresses inherent challenges of ensuring data consistency, availability, and performance across distributed nodes. The plan of action is divided into six parts, ranging from the design and implementation of a comprehensive database system to exploring fragmentation and replication techniques, query processing and optimization, distributed transaction management, and the implementation of a NoSQL database system using MongoDB. The ultimate goal is to create a robust and efficient distributed database solution that caters to the complex requirements of an online retail environment. This document presents a report on Part 1 of the project.

2. Implementation Tools and Languages

Tools used: VS code, pgAdmin, Git

Database systems used: PostgreSQL

Languages: Python, SQL

3. Design and Implementation of a Distributed Database System

3.1. Distributed Database Schema:

In the proposed E-commerce distributed database, the key tables include Users, Products, Categories, Orders, OrderDetails, Transactions, Reviews, Inventory, and Shipping, each serving a distinct purpose. While creating the schema, the normalization techniques (1NF, 2NF, 3NF) were taken into consideration. The Users table stores registered user information, while Products and Categories manage product details and classifications of those products ("Electronics", "Clothing", "Books", "Home & Garden", "Sports & Outdoors") respectively. The Orders and OrderDetails tables track

user orders and their contents. Transactions record payment details for each order. The Reviews table captures user feedback on products, storing their ratings as well as review text. Inventory table manages stock levels and locations, with information for the warehouse location. Finally the shipping monitors the delivery process for orders, the delivery dates and the status ("Shipped", "Processing", "Delivered"). The details of the column names, Primary and Foreign keys of the tables can be viewed from the ER diagram in Fig 1, as well as from the Queries screenshots in the next section. Relationships among these tables, such as the one-to-many connection between Users and Orders or between Products and Reviews, facilitate the representation of complex interactions and dependencies inherent to the E-commerce domain, providing a comprehensive view of the online retail environment. Following is the Entity-Relationship diagram corresponding to the above schema -

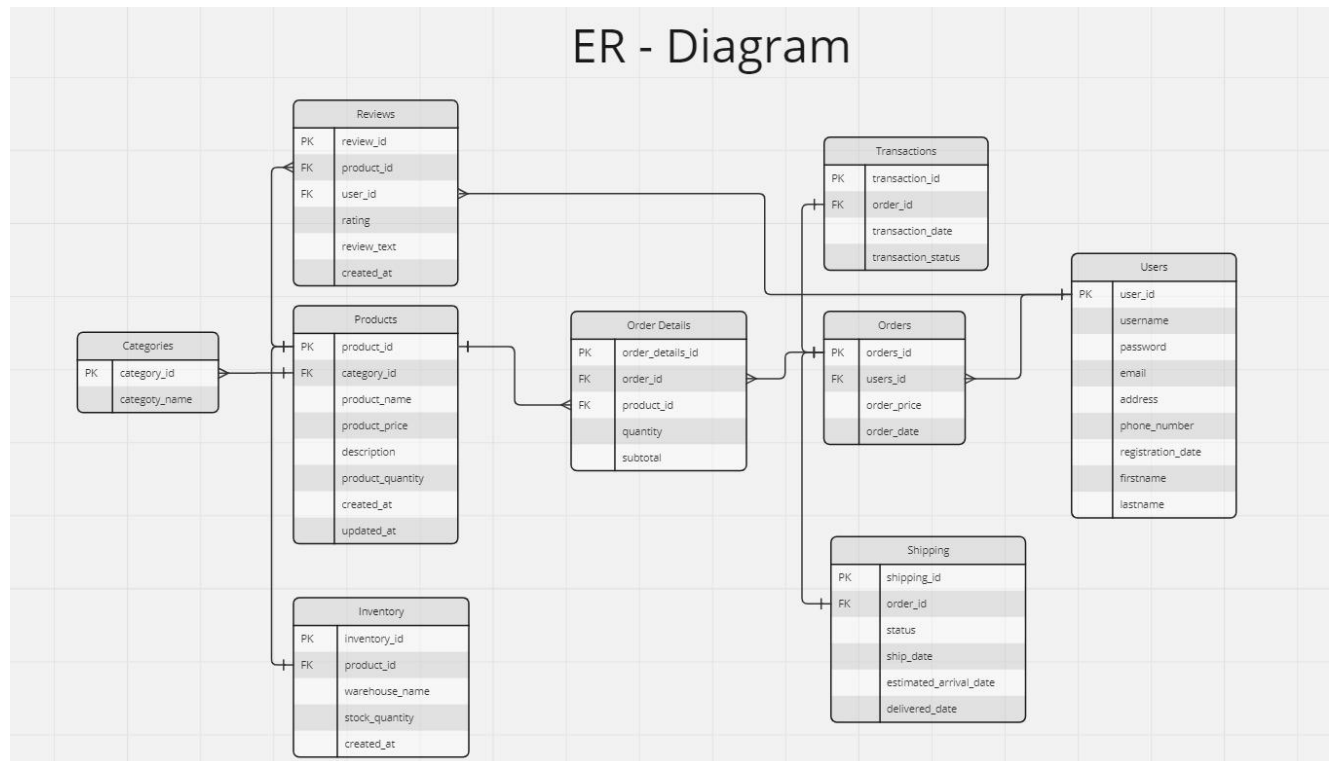


Fig 1: Entity-Relationship diagram for EcoSphere: Distributed Commerce Gateway

3.2. Database Tables:

We have used the postgres server to create our database named “e-commerce”, as shown in the code and the screenshot of the database from pgAdmin software as shown below -

```

final_project.py 3, M X task_5.ipynb
Part 1 > final_project.py > create_database

15
16 def connect_postgres(dbname):
17     """Connect to the PostgreSQL using psycopg2 with default database
18     | Return the connection"""
19     return psycopg2.connect(host=POSTGRES_CONFIG['HOST_NAME'], dbname=dbname, user=POSTGRES_CONFIG['USER_NAME'], password=
POSTGRES_CONFIG['PASSWORD'], port=POSTGRES_CONFIG['PORT'])
20
21
22 def create_database(dbname):
23     """Connect to the PostgreSQL by calling connect_postgres() function
24     | Create a database named 'dbname' passed in argument
25     | Close the connection"""
26     conn = None
27     cur = None
28     try:
29         conn = connect_postgres('postgres')
30         conn.autocommit = True
31         cur = conn.cursor()
32
33         # Check if the database already exists
34         cur.execute("SELECT 1 FROM pg_database WHERE datname = %s", (dbname,))
35         if cur.fetchone():
36             print(f"Database '{dbname}' already exists.")
37         else:
38             # Create the new database
39             cur.execute(f"CREATE DATABASE {dbname}")
40             print(f"Database '{dbname}' created successfully.")
41
42     except Exception as error:
43         print(error)
44
45     finally:
46         if cur is not None:
47             cur.close()
48         if conn is not None:
49             conn.close()
50

```

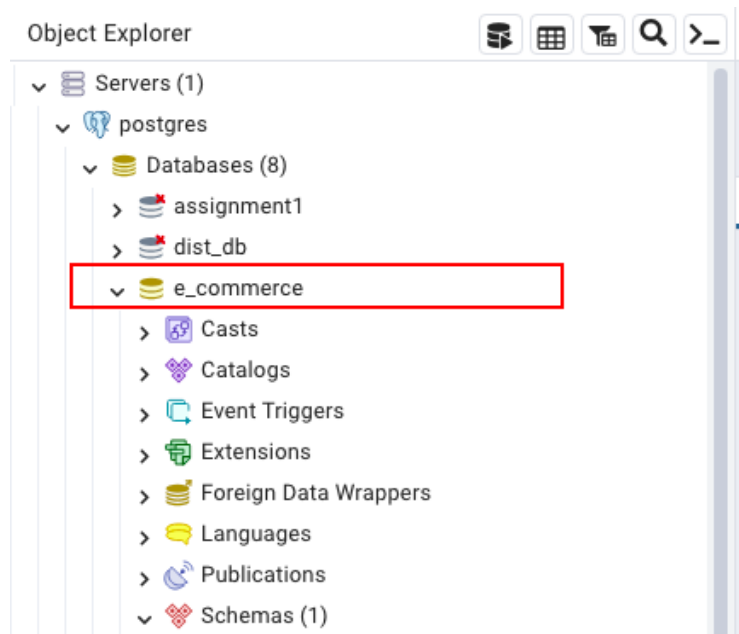


Fig 2: Code showing database creation (above) and the screenshot of database e-commerce created in pgAdmin (below)

There are a total of 9 tables created for the given distributed database schema. The details of the columns, their data types, primary and foreign keys can be gathered from the screenshots of the queries as shown below -

```

final_project.py 3, M  constants.py x
Part 1 > constants.py > ...
13
14 # Tables Names
15 TABLE_NAMES = ["Users", "Categories", "Products", "Orders", "OrderDetails", "Transactions", "Reviews", "Inventory",
16 "Shipping"]
17
18 USERS_QUERY = f"""
19     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[0]} (
20         user_id SERIAL,
21         username TEXT NOT NULL,
22         first_name TEXT NOT NULL,
23         last_name TEXT NOT NULL,
24         email TEXT NOT NULL,
25         password TEXT NOT NULL,
26         address TEXT,
27         phone_number TEXT,
28         registration_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
29         PRIMARY KEY (user_id, registration_date)
30     )
31     PARTITION BY RANGE (registration_date);
32     """
33
34 CATEGORIES_QUERY = f"""
35     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[1]} (
36         category_id SERIAL PRIMARY KEY,
37         category_name TEXT NOT NULL
38     );
39     """
40
41 PRODUCTS_QUERY = f"""
42     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[2]} (
43         product_id SERIAL PRIMARY KEY,
44         product_name TEXT NOT NULL,
45         description TEXT,
46         product_price DECIMAL NOT NULL,
47         category_id INT REFERENCES Categories(category_id),
48         product_quantity INT DEFAULT 0,
49         created_at date DEFAULT CURRENT_DATE,
50         updated_at date DEFAULT CURRENT_DATE
51     );
52     """

```

```
53 ORDERS_QUERY = f"""
54     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[3]} (
55         order_id SERIAL PRIMARY KEY,
56         user_id INT,
57         user_registration_date TIMESTAMP,
58         order_price DECIMAL NOT NULL,
59         order_date date DEFAULT CURRENT_DATE,
60         FOREIGN KEY (user_id, user_registration_date) REFERENCES Users(user_id, registration_date)
61     );
62 """
63
64 ORDER_DETAILS_QUERY = f"""
65     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[4]} (
66         order_detail_id SERIAL PRIMARY KEY,
67         order_id INT REFERENCES Orders(order_id),
68         product_id INT REFERENCES Products(product_id),
69         quantity INT,
70         subtotal DECIMAL
71     );
72 """
73
74 TRANSACTIONS_QUERY = f"""
75     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[5]} (
76         transaction_id SERIAL PRIMARY KEY,
77         order_id INT REFERENCES Orders(order_id),
78         transaction_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
79         transaction_status TEXT
80     );
81 """
```

```

82
83 REVIEWS_QUERY = f"""
84     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[6]} (
85         review_id SERIAL PRIMARY KEY,
86         product_id INT REFERENCES Products(product_id),
87         user_id INT,
88         user_registration_date TIMESTAMP,
89         rating INT CHECK (rating >= 1 AND rating <= 5),
90         review_text TEXT,
91         created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
92         FOREIGN KEY (user_id, user_registration_date) REFERENCES Users(user_id, registration_date)
93     );
94 """
95
96 INVENTORY_QUERY = f"""
97     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[7]} (
98         inventory_id SERIAL PRIMARY KEY,
99         product_id INT REFERENCES Products(product_id),
100        warehouse_name TEXT,
101        stock_quantity INT DEFAULT 0,
102        created_at TIMESTAMP DEFAULT CURRENT_TIMESTAMP
103    );
104 """
105
106 SHIPPING_QUERY = f"""
107     CREATE TABLE IF NOT EXISTS {TABLE_NAMES[8]} (
108         shipping_id SERIAL PRIMARY KEY,
109         order_id INT REFERENCES Orders(order_id),
110         status TEXT,
111         ship_date TIMESTAMP DEFAULT CURRENT_TIMESTAMP,
112         estimated_arrival_date TIMESTAMP,
113         delivered_date TIMESTAMP
114     );
115 """

```

Fig 3: Screenshots for the SQL queries for 9 tables creation

```

51
52 def execute_query(conn, query):
53     """
54     Executes a given SQL query using the provided database connection.
55
56     :param conn: Database connection object
57     :param query: SQL query string to be executed
58     """
59     try:
60         # Create a new cursor
61         with conn.cursor() as cur:
62             # Execute the query
63             cur.execute(query)
64             # Print success message
65             print("Query executed successfully")
66             # Commit the transaction
67             conn.commit()
68     except (Exception, psycopg2.Error) as error:
69         # Rollback in case of error
70         conn.rollback()
71         print("Error executing query:", error)
72         raise
73
74 def create_tables(conn):
75     try:
76         execute_query(conn, CREATE_QUERIES[0])
77         printStatements("Succesfully created User Table")
78         execute_query(conn, CREATE_QUERIES[1])
79         printStatements("Succesfully created Categories Table")
80         execute_query(conn, CREATE_QUERIES[2])
81         printStatements("Succesfully created Products Table")
82         execute_query(conn, CREATE_QUERIES[3])
83         printStatements("Succesfully created Orders Table")
84         execute_query(conn, CREATE_QUERIES[4])
85         printStatements("Succesfully created Order Details Table")
86         execute_query(conn, CREATE_QUERIES[5])
87         printStatements("Succesfully created Transactions Table")
88         execute_query(conn, CREATE_QUERIES[6])
89         printStatements("Succesfully created Reviews Table")
90         execute_query(conn, CREATE_QUERIES[7])
91         printStatements("Succesfully created Inventory Table")
92         execute_query(conn, CREATE_QUERIES[8])
93         printStatements("Succesfully created Shipping Table")
94         create_partition_table_users(conn)
95     except (Exception, psycopg2.Error) as error:
96         print("Error creating tables:", error)
97

```

Fig 4: Screenshot for Python code for table creation

Following is a screenshot of the tables that are successfully created in the database. As an example, a query is run in pgAdmin, fetching the details for User table, and we can see that the table is empty for now, with the schema as per the queries described above.

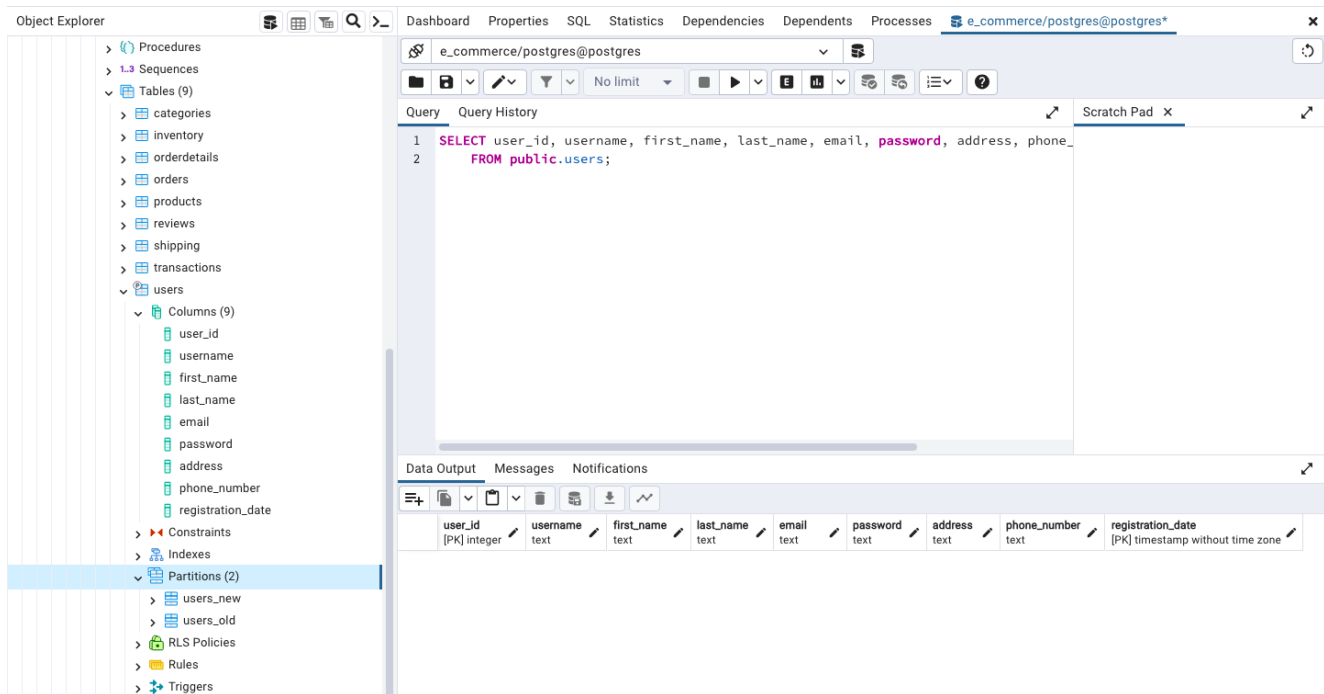


Fig 5: Screenshot of pgAdmin showing the tables created with the partitions and empty data

3.3. Data Distribution Plan:

After thorough analysis and consideration, the chosen data distribution strategy for this project centers around a partitioned and replicated database model. This approach involves fragmenting data into distinct partitions, or shards, each managed independently. These partitions can be distributed across multiple servers or nodes, allowing for load balancing and reducing the risk of single points of failure. The partitioning logic takes into account transaction volume, data access patterns, and geographical distribution of users. The details of partitioning will be discussed in the Part 2 report. To further enhance data availability and fault tolerance, a replication mechanism is also integrated. The data is replicated across another node and it is scalable so as to allow replication across multiple nodes, ensuring that in the event of a server failure, a backup is available to maintain uninterrupted service. For large amounts of data, replication also can aid in load balancing, particularly for read-heavy operations, by allowing read requests to be served by multiple nodes. The combined use of partitioning and replication in our data distribution plan aims to provide a robust, scalable solution, catering to the demands of high-volume, distributed applications while maintaining data integrity and consistency.

3.4. Data Insertion Mechanism:

For testing purposes, the project uses *Faker* library in python to generate and insert mock data into the database. This approach ensures a streamlined and efficient mechanism for populating the database with realistic data, taking into account the respective data types, for various testing scenarios that we have tested in later parts of the project. The insertion process is automated, with scripts designed to batch-insert this mock data, ensuring both time efficiency and consistency in test data creation. The following are the screenshots for the batch data insertion code -

```
Part 1 > final_project.py > insert_mock_data
117 def insert_mock_data(conn):
118     fake = Faker()
119     try:
120         cursor = conn.cursor()
121         if conn and cursor:
122             for _ in range(50):
123                 username = fake.user_name()
124                 first_name = fake.first_name()
125                 last_name = fake.last_name()
126                 email = fake.email()
127                 password = fake.password()
128                 address = fake.address()
129                 phone_number = fake.phone_number()
130                 registration_date = fake.date_time_this_decade()
131
132                 cursor.execute(f"""
133                 INSERT INTO {TABLE_NAMES[0]} (username, first_name, last_name, email, password, address, phone_number,
134                 registration_date)
135                 VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
136                 """, (username, first_name, last_name, email, password, address, phone_number, registration_date))
137
138             categories = ["Electronics", "Clothing", "Books", "Home & Garden", "Sports & Outdoors"]
139             for category_name in categories:
140                 cursor.execute(f"""
141                 INSERT INTO {TABLE_NAMES[1]} (category_name) VALUES (%s)
142                 """, (category_name,))
143
144 cat_ids = []
145 cursor.execute("SELECT category_id FROM Categories")
146 cat_ids = [row[0] for row in cursor.fetchall()]
147 for _ in range(50):
148     product_name = fake.word().capitalize()
149     description = fake.text(max_nb_chars=100)
150     product_price = round(random.uniform(5, 200), 2)
151     category_id = random.choice(cat_ids)
152     product_quantity = random.randint(0, 100)
153     cursor.execute(f"""
154     INSERT INTO {TABLE_NAMES[2]} (product_name, description, product_price, category_id, product_quantity)
155     VALUES (%s, %s, %s, %s, %s)
156     """, (product_name, description, product_price, category_id, product_quantity))
157
158 users = []
159 cursor.execute("SELECT user_id, registration_date FROM Users")
160 users = cursor.fetchall()
161
162 for _ in range(50):
163     user_id, registration_date = random.choice(users)
164     order_price = round(random.uniform(10, 1000), 2)
165     order_date = fake.date_between(start_date=registration_date, end_date="today")
166     cursor.execute(f"""
167     INSERT INTO {TABLE_NAMES[3]} (user_id, user_registration_date, order_price, order_date)
168     VALUES (%s, %s, %s, %s)
169     """, (user_id, registration_date, order_price, order_date))
170
```

```

173     ord_ids = []
174     cursor.execute("SELECT order_id FROM Orders")
175     ord_ids = [row[0] for row in cursor.fetchall()]
176     prod_ids = []
177     cursor.execute("SELECT product_id FROM Products")
178     prod_ids = [row[0] for row in cursor.fetchall()]
179     for _ in range(50):
180         order_id = random.choice(ord_ids)
181         product_id = random.choice(prod_ids)
182         quantity = random.randint(1, 10)
183         subtotal = round(random.uniform(10, 100), 2)
184         cursor.execute(f"""
185             INSERT INTO {TABLE_NAMES[4]} (order_id, product_id, quantity, subtotal)
186             VALUES (%s, %s, %s, %s)
187             """, (order_id, product_id, quantity, subtotal))
188
189
190     for _ in range(50):
191         order_id = random.choice(ord_ids)
192         transaction_date = fake.date_time_between(start_date="-1y", end_date="now")
193         transaction_status = random.choice(["Success", "Pending", "Failed"])
194         cursor.execute(f"""
195             INSERT INTO {TABLE_NAMES[5]} (order_id, transaction_date, transaction_status)
196             VALUES (%s, %s, %s)
197             """, (order_id, transaction_date, transaction_status))
198

```

```

200     for _ in range(50):
201         product_id = random.choice(prod_ids)
202         user_id, user_registration_date = random.choice(usrs)
203         rating = random.randint(1, 5)
204         review_text = fake.paragraph()
205         created_at = fake.date_time_between(start_date="-365d", end_date="now")
206
207         cursor.execute(f"""
208             INSERT INTO {TABLE_NAMES[6]} (product_id, user_id, user_registration_date, rating, review_text,
209             created_at)
210             VALUES (%s, %s, %s, %s, %s, %s)
211             """, (product_id, user_id, user_registration_date, rating, review_text, created_at))
212
213
214     for _ in range(50):
215         product_id = random.choice(prod_ids)
216         warehouse_name = fake.company()
217         stock_quantity = random.randint(0, 1000)
218         created_at = fake.date_time_between(start_date="-30d", end_date="now")
219         cursor.execute(f"""
220             INSERT INTO {TABLE_NAMES[7]} (product_id, warehouse_name, stock_quantity, created_at)
221             VALUES (%s, %s, %s, %s)
222             """, (product_id, warehouse_name, stock_quantity, created_at))
223

```

```

224
225     for _ in range(1, 51):
226         order_id = random.choice(ord_ids)
227         status = random.choice(["Shipped", "Processing", "Delivered"])
228         ship_date = fake.date_time_between(start_date="-30d", end_date="now")
229         estimated_arrival_date = ship_date + timedelta(days=random.randint(1, 10))
230         delivered_date = estimated_arrival_date + timedelta(days=random.randint(1, 5)) if status == "Delivered" else
                None
231
232         cursor.execute(f"""
233             INSERT INTO {TABLE_NAMES[8]} (order_id, status, ship_date, estimated_arrival_date, delivered_date)
234             VALUES (%s, %s, %s, %s, %s)
235             """, (order_id, status, ship_date, estimated_arrival_date, delivered_date))
236
237
238         conn.commit()
239
240         print(f"Inserted data!")
241
242     except (Exception, psycopg2.Error) as error:
243         print("Error inserting data:", error)

```

Fig 6: Screenshots of the code for data insertion

3.5. Data Retrieval Proof:

The above that has been inserted can be retrieved from the database using select SQL queries as shown below -

The screenshot shows a Jupyter Notebook interface with a code editor and a terminal output. The code defines a function `select_data(conn)` that iterates through `TABLE_NAMES` and executes a SQL query to select all data from each table. The output shows the data for the 'Shipping' table, which includes columns: `shipping_id`, `order_id`, `status`, `ship_date`, `estimated_arrival_date`, and `delivered_date`. The data is displayed in a table format with 14 rows.

```

245 def select_data(conn):
246     try:
247         cursor = conn.cursor()
248         if conn and cursor:
249             for table_name in TABLE_NAMES:
250                 cursor.execute(f"SELECT * FROM {table_name};")
251                 rows = cursor.fetchall()
252                 headers = [desc[0] for desc in cursor.description]
253                 print(f"Data from Table: {table_name}")
254                 print(tabulate(rows, headers=headers, tablefmt="grid"))
255                 print("\n" + "-" * 50 + "\n") # Separator between tables
256
257     except (Exception, psycopg2.Error) as error:
258         print("Error selecting data:", error)

```

shipping_id	order_id	status	ship_date	estimated_arrival_date	delivered_date
1	198	Processing	2023-11-23 19:51:45.177006	2023-11-25 19:51:45.177006	
2	169	Shipped	2023-11-13 14:02:23.395444	2023-11-21 14:02:23.395444	
3	174	Processing	2023-11-15 18:01:54.345766	2023-11-22 18:01:54.345766	
4	187	Delivered	2023-11-16 14:09:17.212365	2023-11-26 14:09:17.212365	2023-12-01 14:09:17.212365
5	185	Processing	2023-10-30 17:22:39.278301	2023-11-08 17:22:39.278301	
6	172	Shipped	2023-11-21 12:19:27.782103	2023-12-01 12:19:27.782103	
7	177	Shipped	2023-11-16 19:07:55.598897	2023-11-22 19:07:55.598897	
8	173	Delivered	2023-11-18 10:21:05.610372	2023-11-19 10:21:05.610372	2023-11-21 10:21:05.610372
9	179	Shipped	2023-11-10 22:12:11.412046	2023-11-17 22:12:11.412046	
10	190	Processing	2023-11-17 08:11:12.878829	2023-11-26 08:11:12.878829	
11	181	Delivered	2023-11-23 14:35:52.205136	2023-11-27 14:35:52.205136	2023-12-02 14:35:52.205136
12	168	Processing	2023-11-21 10:18:50.421653	2023-11-29 10:18:50.421653	
13	155	Processing	2023-10-31 17:04:45.705347	2023-11-01 17:04:45.705347	
14	167	Processing	2023-11-11 03:03:09.022673	2023-11-16 03:03:09.022673	

Fig 7: Screenshot of data retrieval code and result for Shipping table on the console

The image displays two screenshots of a database management interface, likely DBeaver, showing SQL queries and their results for the 'users' and 'products' tables.

Top Screenshot: Users Table

The left sidebar shows the 'users' table selected under the 'public' schema. The main pane displays the following SQL query:

```
1 SELECT user_id, username, first_name, last_name, email, password, address, phone_
2 FROM public.users;
```

The 'Data Output' tab shows the results of the query, displaying 17 rows of user data. The columns are: user_id (PK integer), username text, first_name text, last_name text, email text, password text, and address text.

user_id	username	first_name	last_name	email	password	address
1	jason95	John	Barker	jonathanhill@example.com	3\$7Ba9ecE	USS Roberts
2	glopez	Teresa	Ryan	pauljohnson@example.com	#9Dh7Kqisx	65665 Monica Route Apt. 828
3	peter26	Paul	Myers	debra16@example.com	PB\$#0*Wz...	0950 Jacob Shoal
4	nberg	James	Chung	kevin68@example.net	@y3JrdRF...	Unit 3614 Box 9232
5	stevensonrobert	Joshua	Sanchez	dannywhite@example.org)4rPRQdNV(634 Andrew Gardens
6	williamsaudrey	David	Gonzales	wweaver@example.net	\$ysIAo@0...	310 Phillips Island
7	paulabishop	Kaitlyn	Washington	hansonvictor@example.com	#H00BRrp...	PSC 0441, Box 1962
8	rodriguezjulie	Daniel	Best	helen81@example.com	F_2LmKxitA	387 Freeman Locks Apt. 116
9	nicholasalvarez	Samuel	Green	xweber@example.net	_s01W2ox*\$	333 Murphy Harbors
10	robert03	Russell	Mcintosh	brian72@example.net	Mt%8clda#e	92669 Shannon Junction
11	gregory76	Seth	Chapman	meghan21@example.com	3sVKg*5iG	USNS White
12	james67	Kerry	Martinez	mmiller@example.net	kV%i0JdLz4	Unit 2619 Box 1115
13	corey75	Melissa	Powell	michaalcervantes@example...	f\$7u9Qga...	702 Sullivan Crescent Suite 008
14	patrick60	Jason	Lopez	joannaburton@example.org	%S7M&m#...	327 Schultz Falls Suite 076
15	wallacecurtis	Samuel	Caldwell	jameskeith@example.net	nW@R85Q...	PSC 2132, Box 6522
16	bruce49	James	Hogan	natalie74@example.org	M@zfk9Gh...	USCGC Clayton
17	alexanderpamela	John	Wilson	howardvictoria@example.net	4@5Kw(Fulp	PSC 3667, Box 6193

Bottom Screenshot: Products Table

The left sidebar shows the 'products' table selected under the 'public' schema. The main pane displays the following SQL query:

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

The 'Data Output' tab shows the results of the query, displaying 21 rows of product data. The columns are: product_id (PK integer), product_name text, description text, product_price numeric, category_id integer, product_quantity integer, and created_at date.

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

Total rows: 50 of 50 Query complete 00:00:00.244 Ln 1, Col 1

Fig 8: Screenshot for data retrieval proof for Users table (above) and Products table (below)

4. Conclusion

In Part 1 of the project, we successfully designed and implemented a distributed database system tailored to handle real-time data of an e-commerce system. Key deliverables include a distributed database schema with entity-relationship diagram, creation scripts for database tables with appropriate attributes, keys, and constraints, and a robust data distribution plan, an efficient data insertion mechanism, as well as evidence of successful data retrieval.