Distributed Database Systems (CSE 512)

Group project

Task 2

Horizontal Fragmentation

Horizontal fragmentation in a database context involves dividing a table into several smaller, disjoint subsets of tuples (rows) based on certain criteria. This is usually done to improve query performance, especially in a distributed database system where different fragments might reside on different servers closer to the user querying them.

Code:

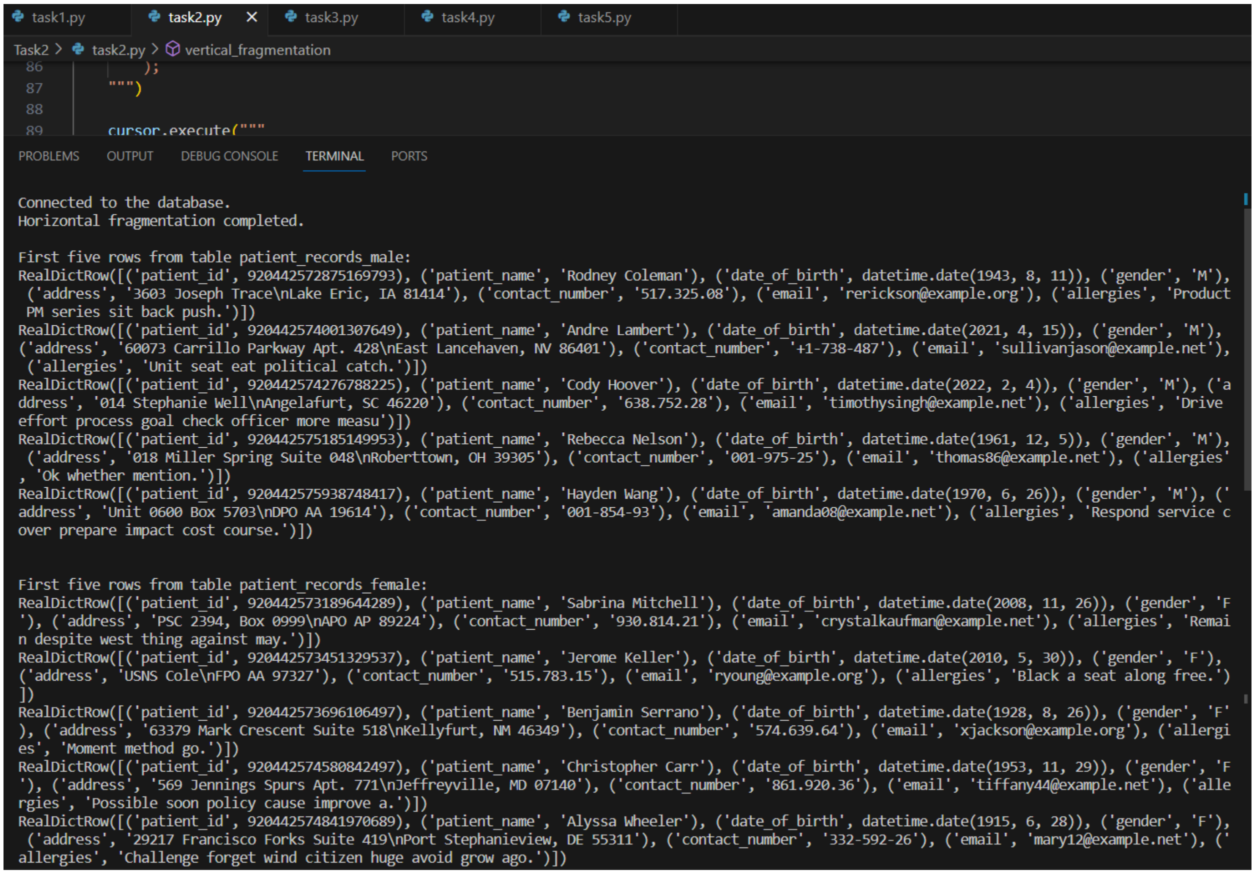
```
import psycopg2.extras
import uuid
from faker import Faker
import psycopg2
from psycopg2 import sql
import random
from datetime import datetime, timedelta


DATABASE_NAME = 'healthcare'

fake = Faker()

DB_URL =
"postgresql://shashank:z3L2HOT24J5yDJWdt3esqw@plain-koala-13452.5xj.cockro
achlabs.cloud:26257/healthcare?sslmode=verify-full"
```

```
def connect_db():
    try:
        conn = psycopg2.connect(DB_URL,
                                application_name="healthcare_app",

cursor_factory=psycopg2.extras.RealDictCursor)
        print("Connected to the database.")
        return conn
    except Exception as e:
        print("Database connection failed.")
        print(e)
        return None

def horizontal_fragmentation(conn):
    """Performs horizontal fragmentation on the patient_records table
based on gender."""
    cursor = conn.cursor()
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS patient_records_male AS
        SELECT * FROM patient_records WHERE gender = 'M';
    """)
    cursor.execute("""
        CREATE TABLE IF NOT EXISTS patient_records_female AS
        SELECT * FROM patient_records WHERE gender = 'F';
    """)

    # Insert data into fragment tables
    cursor.execute("""
        INSERT INTO patient_records_male
        SELECT * FROM patient_records WHERE gender = 'M';
    """)
    cursor.execute("""
        INSERT INTO patient_records_female
        SELECT * FROM patient_records WHERE gender = 'F';
    """)
    tables = ['patient_records_male', 'patient_records_female']

    print("Horizontal fragmentation completed.\n")

    for table in tables:
```

```
        print(f"First five rows from table {table}:")

        cursor.execute(sql.SQL("SELECT * FROM {} LIMIT 
5").format(sql.Identifier(table)))

        records = cursor.fetchall()

        for row in records:
            print(row)
        print("\n")
    conn.commit()
    cursor.close()
```

The function creates two new tables, patient_records_male and patient_records_female, which are fragments of the original patient_records table. It then inserts the corresponding data into these new tables based on the gender column.

Results:




```
Connected to the database.
Horizontal fragmentation completed.

First five rows from table patient_records_male:
RealDictRow([('patient_id', 920442572875169793), ('patient_name', 'Rodney Coleman'), ('date_of_birth', datetime.date(1943, 8, 11)), ('gender', 'M'),
 ('address', '3603 Joseph Trace\nLake Eric, IA 81414'), ('contact_number', '517.325.08'), ('email', 'rerickson@example.org'), ('allergies', 'Product
 PM series sit back push.')])
RealDictRow([('patient_id', 920442574001307649), ('patient_name', 'Andre Lambert'), ('date_of_birth', datetime.date(2021, 4, 15)), ('gender', 'M'),
('address', '60073 Carrillo Parkway Apt. 428\nEast Lancehaven, NV 86401'), ('contact_number', '+1-738-487'), ('email', 'sullivanjason@example.net'),
 ('allergies', 'Unit seat eat political catch.')])
RealDictRow([('patient_id', 920442574276788225), ('patient_name', 'Cody Hoover'), ('date_of_birth', datetime.date(2022, 2, 4)), ('gender', 'M'), ('a
ddress', '014 Stephanie Well\nAngelafurt, SC 46220'), ('contact_number', '638.752.28'), ('email', 'timothysingh@example.net'), ('allergies', 'Drive
effort process goal check officer more measu')])
RealDictRow([('patient_id', 920442575185149953), ('patient_name', 'Rebecca Nelson'), ('date_of_birth', datetime.date(1961, 12, 5)), ('gender', 'M'),
 ('address', '018 Miller Spring Suite 048\nRoberttown, OH 39305'), ('contact_number', '001-975-25'), ('email', 'thomas86@example.net'), ('allergies'
, 'Ok whether mention.')])
RealDictRow([('patient_id', 920442575938748417), ('patient_name', 'Hayden Wang'), ('date_of_birth', datetime.date(1970, 6, 26)), ('gender', 'M'), ('
address', 'Unit 0600 Box 5703\nDPO AA 19614'), ('contact_number', '001-854-93'), ('email', 'amanda08@example.net'), ('allergies', 'Respond service c
over prepare impact cost course.')])


First five rows from table patient_records_female:
RealDictRow([('patient_id', 920442573189644289), ('patient_name', 'Sabrina Mitchell'), ('date_of_birth', datetime.date(2008, 11, 26)), ('gender', 'F
'), ('address', 'PSC 2394, Box 0999\nAPO AP 89224'), ('contact_number', '930.814.21'), ('email', 'crystalkaufman@example.net'), ('allergies', 'Remai
n despite west thing against may.')])
RealDictRow([('patient_id', 920442573451329537), ('patient_name', 'Jerome Keller'), ('date_of_birth', datetime.date(2010, 5, 30)), ('gender', 'F'),
('address', 'USNS Cole\nFPO AA 97327'), ('contact_number', '515.783.15'), ('email', 'ryoung@example.org'), ('allergies', 'Black a seat along free.')
])
RealDictRow([('patient_id', 920442573696106497), ('patient_name', 'Benjamin Serrano'), ('date_of_birth', datetime.date(1928, 8, 26)), ('gender', 'F'
), ('address', '63379 Mark Crescent Suite 518\nKellyfurt, NM 46349'), ('contact_number', '574.639.64'), ('email', 'xjackson@example.org'), ('allergi
es', 'Moment method go.')])
RealDictRow([('patient_id', 920442574580842497), ('patient_name', 'Christopher Carr'), ('date_of_birth', datetime.date(1953, 11, 29)), ('gender', 'F
'), ('address', '569 Jennings Spurs Apt. 771\nJeffreyville, MD 07140'), ('contact_number', '861.920.36'), ('email', 'tiffany44@example.net'), ('alle
rgies', 'Possible soon policy cause improve a.')])
RealDictRow([('patient_id', 920442574841970689), ('patient_name', 'Alyssa Wheeler'), ('date_of_birth', datetime.date(1915, 6, 28)), ('gender', 'F'),
 ('address', '29217 Francisco Forks Suite 419\nPort Stephanieview, DE 55311'), ('contact_number', '332-592-26'), ('email', 'mary12@example.net'), ('
allergies', 'Challenge forget wind citizen huge avoid grow ago.')])
```

Above are the first five rows of two columns patient_records_male and patient_records_female after horizontal fragmentation.

Vertical Fragmentation

Vertical fragmentation involves dividing a table into sub-tables where each sub-table contains a subset of columns from the original table. This approach can be particularly useful when different applications or different parts of an application frequently access only a subset of columns.

In the below function, it performs vertical fragmentation on the patient_records table based on a hypothetical use-case where one application mostly accesses personal information and another deals with contact information.

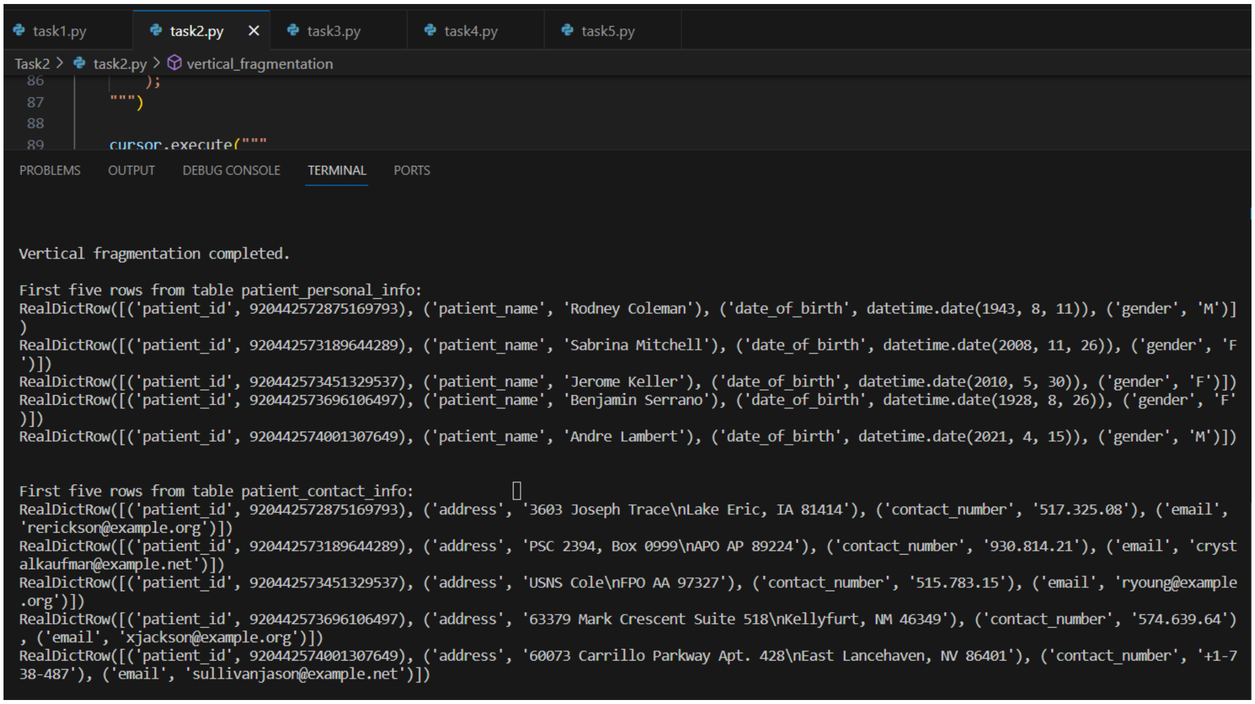
Code:

```
def vertical_fragmentation(conn):
    cursor = conn.cursor()

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS patient_personal_info (
            patient_id INT PRIMARY KEY,
            patient_name VARCHAR(150) NOT NULL,
            date_of_birth DATE NOT NULL,
            gender VARCHAR(10) NOT NULL
        );
    """)

    cursor.execute("""
        CREATE TABLE IF NOT EXISTS patient_contact_info (
            patient_id INT PRIMARY KEY,
            address VARCHAR(100) NOT NULL,
            contact_number VARCHAR(15) NOT NULL UNIQUE,
            email VARCHAR(50) NOT NULL UNIQUE
        );
    """)

    cursor.execute("""
        INSERT INTO patient_personal_info (patient_id, patient_name,
date_of_birth, gender)
        SELECT patient_id, patient_name, date_of_birth, gender FROM
patient_records
        ON CONFLICT (patient_id) DO NOTHING;
    """)
    cursor.execute("""
        INSERT INTO patient_contact_info (patient_id, address,
contact_number, email)
        SELECT patient_id, address, contact_number, email FROM
patient_records
        ON CONFLICT (patient_id) DO NOTHING;
    """)
    print("Vertical fragmentation completed.\n")
```

```
    tables = ['patient_personal_info', 'patient_contact_info']

    for table in tables:
        print(f"First five rows from table {table}:")

        cursor.execute(sql.SQL("SELECT * FROM {} LIMIT
5").format(sql.Identifier(table)))

        # Fetch first five records from the cursor
        records = cursor.fetchall()

        for row in records:
            print(row)
        print("\n")
    conn.commit()
    cursor.close()
```

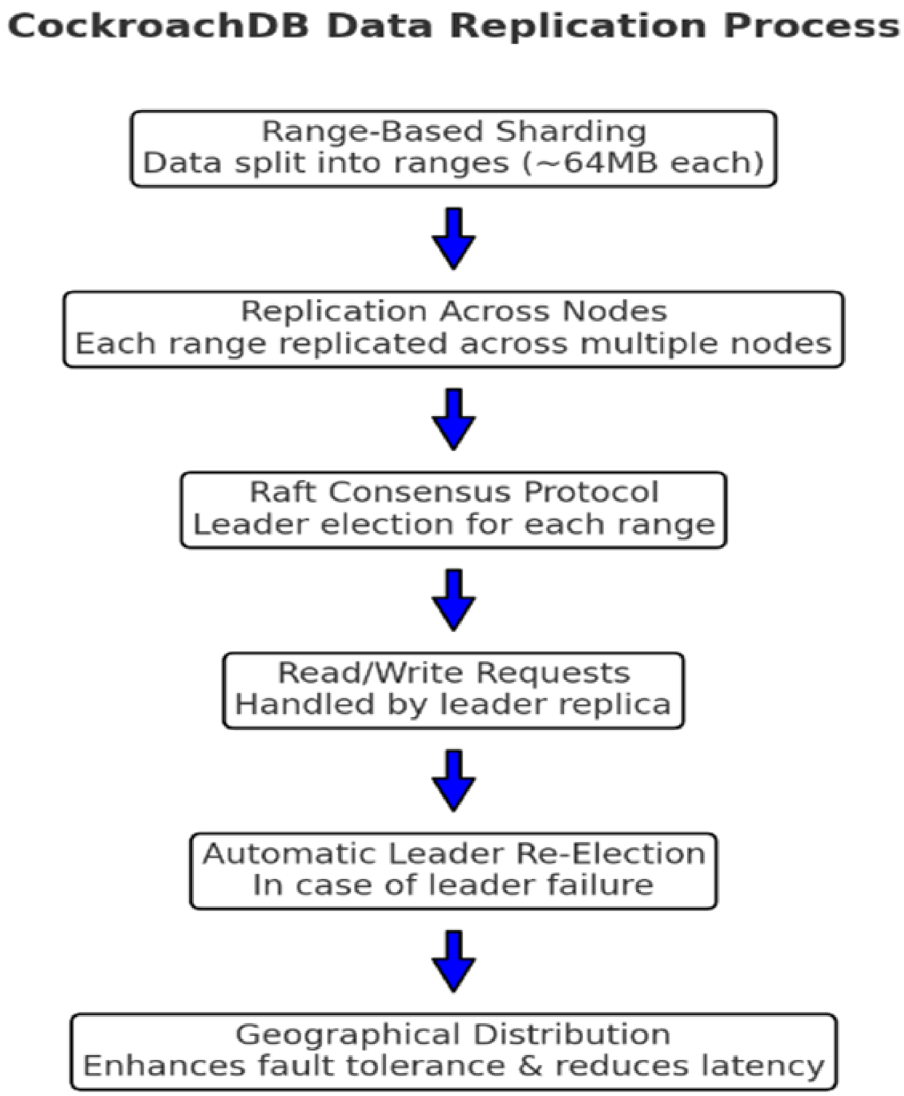
This function assumes the patient_records table has already been created and populated with data. It creates two new tables: patient_personal_info and patient_contact_info. Each new table includes the patient_id column, which serves as a foreign key back to the original patient_records table, ensuring referential integrity.

Results:



The patient_personal_info table is intended for applications that only need to access personal details about patients (excluding contact information), while the patient_contact_info table is designed for applications that only need to access contact details after vertical fragmentation.

.Replication Setup



CockroachDB handles data replication automatically through a sophisticated and distributed architecture, ensuring high availability and consistency of data across multiple nodes. At the heart of this system is its range-based sharding mechanism, where the database automatically splits data into smaller units called ranges, typically around 64MB each. These ranges are dynamically adjusted as data grows, allowing for efficient distribution and management. Once the data is divided into ranges, CockroachDB replicates each range across different nodes in the cluster. By default, each range is replicated three times, but this number can be adjusted for higher redundancy based on the fault tolerance requirements. This replication strategy is fundamental in providing both data redundancy and high availability.

The orchestration of these replicas is managed through the Raft consensus protocol, a robust system for ensuring data consistency across distributed systems. In this protocol, each range elects a leader replica through a distributed consensus process. This leader is then responsible for handling all read and write requests for that range, ensuring that all data operations are centrally coordinated and consistent. The other replicas act as followers, receiving updates from the leader. Should the leader replica become unavailable, say due to a node failure, Raft ensures that a new leader is automatically elected from the remaining replicas. This feature is crucial in maintaining continuous service and data integrity, even in the face of hardware or network failures. Moreover, CockroachDB's architecture allows for the geographical distribution of replicas. This not only enhances the fault tolerance of the system by spreading risk across different locations but also optimizes performance by allowing data to be located closer to users, reducing latency.Through this automatic replication mechanism, CockroachDB provides a highly resilient, consistent, and distributed database solution.