Distributed Database Systems (CSE 512)

Group project

Task 1

Project Overview

Topic: Development of a distributed database system for efficient management of health information.
Context: The system aims to handle medical history, patient records, treatments, billing information, etc., in a seamless, efficient, and confidential manner.

Part 1: Design and Implementation of a Distributed Healthcare Database

Tables

We define and finalize a database schema for our distributed healthcare database project, compatible with PostgreSQL, we'll detail each table, its structure, and the relationships among them. The tables which include Patient Records, Doctor's Information, Appointments, Medical History, Medications, and Billing and Insurance.

1. **patient_records:**
   a. Description: Contains personal and medical records of patients.
   b. Columns:
      i. patient_id (INT, NOT NULL, PK): Unique identifier for the patient.
      ii. patient_name (VARCHAR(150), NOT NULL): Full name of the patient.
      iii. date_of_birth (DATE, NOT NULL): Birthdate of the patient.
      iv. gender (VARCHAR(10), NOT NULL): Gender of the patient.
      v. address (VARCHAR(100), NOT NULL): Residential address of the patient.
      vi. contact_number (VARCHAR(10), NOT NULL): Contact number of the patient.
      vii. email (VARCHAR(50), NOT NULL): Email address of the patient.
      viii. allergies (VARCHAR(50)): Known allergies of the patient.

2. **doctors_info**
    a. Description: Stores information about doctors
    b. Columns:
        i. doctor_id (INT, NOT NULL, PK): Unique identifier for the doctor.
        ii. name (VARCHAR(150), NOT NULL): Full name of the doctor.
        iii. specialization (VARCHAR(150), NOT NULL): Medical specialization of the doctor.
        iv. contact_number (VARCHAR(10), NOT NULL): Contact number of the doctor.
        v. email (VARCHAR(50), NOT NULL): Email address of the doctor.
        vi. availability (VARCHAR(150), NOT NULL): Availability hours of the doctor.

3. **appointments**
    a. Description: Records appointments scheduled with doctors.
    b. Columns:
        i. appointment_id (INT, NOT NULL, PK): Unique identifier for the appointment.
        ii. patient_id (INT, NOT NULL, FK): Identifier for the patient.
        iii. doctor_id (INT, NOT NULL, FK): Identifier for the doctor.
        iv. appointment_date (TIMESTAMP, NOT NULL): Date and time of the appointment.
        v. purpose (VARCHAR(100), NOT NULL): Purpose of the appointment.

4. **medical_history**
    a. Description: Keeps a log of patients' medical history.
    b. Columns:
        i. history_id (INT, NOT NULL, PK): Unique identifier for the medical history entry.
        ii. patient_id (INT, NOT NULL, FK): Identifier for the patient.
        iii. diagnosis (VARCHAR(200), NOT NULL): Diagnosis details.
        iv. treatment (VARCHAR(200), NOT NULL): Treatment details.
        v. date_of_visit (TIMESTAMP, NOT NULL): Date and time of the medical visit.

5. **medications**
    a. Description: Catalogs medications that can be prescribed.
    b. Columns:
        i. medication_id (INT, NOT NULL, PK): Unique identifier for the medication.
        ii. name (VARCHAR(20), NOT NULL): Name of the medication.
        iii. dosage (VARCHAR(20), NOT NULL): Dosage details for the medication.
        iv. manufacturer (VARCHAR(100), NOT NULL): Manufacturer of the medication.

6. **billing_insurance**
    a. Description: Manages billing and insurance information related to patients.
    b. Columns:
        i. billing_id (INT, NOT NULL, PK): Unique identifier for the billing record.
        ii. patient_id (INT, NOT NULL, FK): Identifier for the patient.
        iii. amount (DECIMAL(10,2), NOT NULL): The billed amount for services.
        iv. date (DATE, NOT NULL): Date of the billing.
        v. insurance_provider (VARCHAR(100), NOT NULL): Name of the insurance provider.
        vi. insurance_policy_number (VARCHAR(50), NOT NULL): Policy number of the patient's insurance.

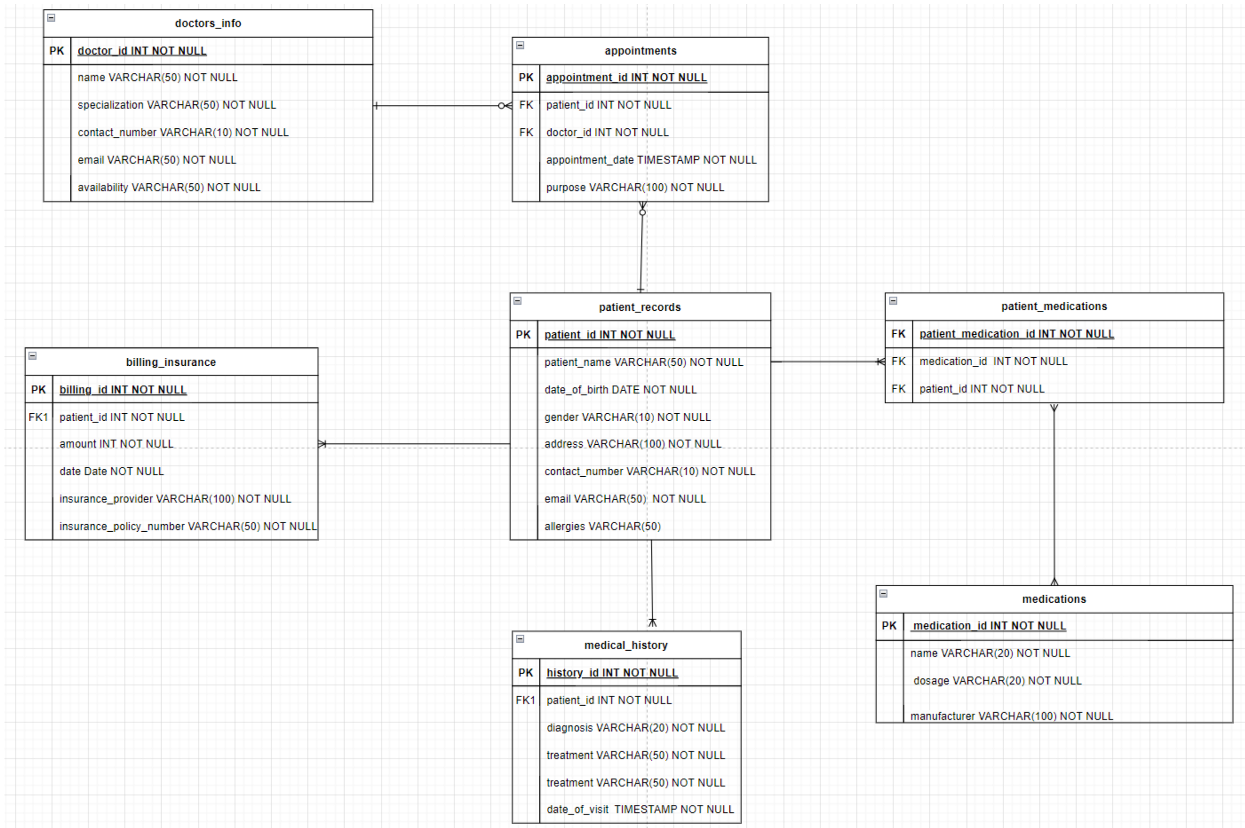
7. **medical_history**
    a. Description : Keeps a log of patients' medical history.
    b. Columns:
        i. history_id (INT, NOT NULL, PK): Unique identifier for the medical history entry.
        ii. patient_id (INT, NOT NULL, FK): Identifier for the patient.
        iii. diagnosis (VARCHAR(200), NOT NULL): Diagnosis details.
        iv. treatment (VARCHAR(200), NOT NULL): Treatment details.
        v. date_of_visit (TIMESTAMP, NOT NULL): Date and time of the medical visit.

8. **Patient_medications**
    a. Description : Junction table for many-to-many relationship between patients and medications.
    b. Columns:
        i. patient_id (INT, NOT NULL, FK): Identifier for the patient.
        ii. medication_id (INT, NOT NULL, FK): Identifier for the medication.

Relationships

➢ doctors_info to appointments: One-to-many. A doctor can have multiple appointments.
➢ patient_records to appointments: One-to-many. A patient can have multiple appointments.
➢ patient_records to medical_history: One-to-many. A patient can have multiple medical history entries.
➢ patient_records to billing_insurance: One-to-many. A patient can have multiple billing records.
➢ Patient_records to patient_medications: One-to-many. A patient can be prescribed multiple medications and also medication can be prescribed to multiple patients (via medications).

E-R Diagram



doctors_info
PK doctor_id INT NOT NULL
name VARCHAR(50) NOT NULL
specialization VARCHAR(50) NOT NULL
contact_number VARCHAR(10) NOT NULL
email VARCHAR(50) NOT NULL
availability VARCHAR(50) NOT NULL
appointments
PK appointment_id INT NOT NULL
FK patient_id INT NOT NULL
FK doctor_id INT NOT NULL
appointment_date TIMESTAMP NOT NULL
purpose VARCHAR(100) NOT NULL
patient_records
PK patient_id INT NOT NULL
patient_name VARCHAR(50) NOT NULL
date_of_birth DATE NOT NULL
gender VARCHAR(10) NOT NULL
address VARCHAR(100) NOT NULL
contact_number VARCHAR(10) NOT NULL
email VARCHAR(50) NOT NULL
allergies VARCHAR(50)
patient_medications
FK patient_medication_id INT NOT NULL
FK medication_id INT NOT NULL
FK patient_id INT NOT NULL
billing_insurance
PK billing_id INT NOT NULL
FK1 patient_id INT NOT NULL
amount INT NOT NULL
date Date NOT NULL
insurance_provider VARCHAR(100) NOT NULL
insurance_policy_number VARCHAR(50) NOT NULL
medications
PK medication_id INT NOT NULL
name VARCHAR(20) NOT NULL
dosage VARCHAR(20) NOT NULL
manufacturer VARCHAR(100) NOT NULL
medical_history
PK history_id INT NOT NULL
FK1 patient_id INT NOT NULL
diagnosis VARCHAR(20) NOT NULL
treatment VARCHAR(50) NOT NULL
treatment VARCHAR(50) NOT NULL
date_of_visit TIMESTAMP NOT NULL

Distribution Strategy Overview

We have chosen a sharding approach with a combination of horizontal partitioning and replication to distribute our data. This strategy allows us to distribute data across multiple servers while keeping a copy of the data on different nodes to ensure high availability and fault tolerance.

1. Doctors Info and Appointments :
    a. Sharding Key: doctor_id
    b. Partitioning Scheme: By specialization or geographical location (if applicable)
    c. Rationale: Grouping appointments by doctor and specialization can improve query performance for appointment scheduling and doctor searches.

2. Patient Records, Medical History, and Billing
    a. Sharding Key: patient_id
    b. Partitioning Scheme: By patient's last name or ZIP code (for geographical distribution)
    c. Rationale: Co-locating a patient's records, medical history, and billing information optimizes for transactions related to individual patients.

3. Medications
    a. Sharding Key: medication_id
    b. Partitioning Scheme: Alphabetically by name or by manufacturer
    c. Rationale: Distributing medication data helps balance the load and improves search performance based on medication name or manufacturer.

4. Replication Strategy
    a. Method: Synchronous or Asynchronous Replication
    b. Rationale: To ensure data is consistently backed up across nodes, enhancing data availability and durability.
    c. Details:
        i. Primary-secondary replication setup.
        ii. Each shard will have one primary node and multiple secondary nodes.

Failover and Recovery Plan

- Automatic Failover: In case the primary node fails, one of the secondary nodes will be promoted to primary automatically.
- Data Recovery: Regular backups and transaction logs will be maintained for data recovery in case of system failures.

Load Balancing

- Read-Write Splitting: Read operations are distributed across primary and secondary nodes to balance the load.
- Connection Pooling: Implemented at the application layer to manage and optimize connections to the database nodes.

Monitoring and Maintenance

- Regular monitoring of query performance, node health, and data distribution.
- Periodic evaluation and rebalancing of data distribution to accommodate changes in data size and access patterns.

This data distribution plan aims to provide a robust framework for the healthcare database to ensure it is scalable, resilient, and performant. Regular assessments will be conducted to adapt to the changing needs of the system and to incorporate improvements in data distribution technologies.

Data Insertion Mechanism:

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 create_tables(conn):
    with conn.cursor() as cur:
        cur.execute('''
            CREATE TABLE IF NOT EXISTS doctors_info (
                doctor_id SERIAL PRIMARY KEY,
                name VARCHAR(50) NOT NULL,
                specialization VARCHAR(50) NOT NULL,
                contact_number VARCHAR(10) NOT NULL,
                email VARCHAR(50) NOT NULL UNIQUE,
                availability VARCHAR(50) NOT NULL
            );
            CREATE TABLE IF NOT EXISTS patient_records (
                patient_id SERIAL PRIMARY KEY,
                patient_name VARCHAR(50) NOT NULL,
                date_of_birth DATE NOT NULL,
                gender VARCHAR(10) NOT NULL,
                address VARCHAR(100) NOT NULL,
                contact_number VARCHAR(10) NOT NULL UNIQUE,
                email VARCHAR(50) NOT NULL UNIQUE,
                allergies VARCHAR(50)
            );
            CREATE TABLE IF NOT EXISTS appointments (
```

```
            appointment_id SERIAL PRIMARY KEY,
            patient_id INT NOT NULL,
            doctor_id INT NOT NULL,
            appointment_date TIMESTAMP NOT NULL,
            purpose VARCHAR(100) NOT NULL,
            FOREIGN KEY (patient_id) REFERENCES patient_records
(patient_id),
            FOREIGN KEY (doctor_id) REFERENCES doctors_info
(doctor_id)
        );
        CREATE TABLE IF NOT EXISTS medical_history (
            history_id SERIAL PRIMARY KEY,
            patient_id INT NOT NULL,
            diagnosis VARCHAR(200) NOT NULL,
            treatment VARCHAR(200) NOT NULL,
            date_of_visit TIMESTAMP NOT NULL,
            FOREIGN KEY (patient_id) REFERENCES patient_records
(patient_id)
        );
        CREATE TABLE IF NOT EXISTS medications (
            medication_id SERIAL PRIMARY KEY,
            name VARCHAR(20) NOT NULL,
            dosage VARCHAR(20) NOT NULL,
            manufacturer VARCHAR(100) NOT NULL
        );
        CREATE TABLE IF NOT EXISTS billing_insurance (
            billing_id SERIAL PRIMARY KEY,
            patient_id INT NOT NULL,
            amount DECIMAL(10, 2) NOT NULL,
            date DATE NOT NULL,
            insurance_provider VARCHAR(100) NOT NULL,
            insurance_policy_number VARCHAR(50) NOT NULL,
            FOREIGN KEY (patient_id) REFERENCES patient_records
(patient_id)
        );
        CREATE TABLE IF NOT EXISTS patient_medications (
            patient_id INT NOT NULL,
            medication_id INT NOT NULL,
            PRIMARY KEY (patient_id, medication_id),
```

```
                FOREIGN KEY (patient_id) REFERENCES patient_records
(patient_id),
                FOREIGN KEY (medication_id) REFERENCES medications
(medication_id)
            );
        ''')
        print("Tables created successfully.")
        conn.commit()


def insert_random_data(conn):
    """Insert random data into tables"""
    cursor = conn.cursor()

    for _ in range(20):
        cursor.execute(
            """
            INSERT INTO doctors_info(name, specialization, contact_number,
email, availability)
            VALUES (%s, %s, %s, %s, %s);
            """,
            (fake.name(),
             fake.job()[:150],
             fake.phone_number()[:10],
             fake.email(),
             fake.day_of_week())
        )
    # Insert data into patient_records
    for _ in range(20):
        cursor.execute(
            """
            INSERT INTO patient_records (patient_name, date_of_birth,
gender, address, contact_number, email, allergies)
            VALUES (%s, %s, %s, %s, %s, %s, %s);
            """,
            (
                fake.name(),
                fake.date_of_birth(minimum_age=0, maximum_age=115),
                random.choice(['M', 'F']),
                fake.address()[:100],
```

```
                fake.phone_number()[:10],
                fake.email()[:50],
                fake.sentence()[:50]
            )
        )

    # Insert data into medications
    for _ in range(20):
        cursor.execute(f"INSERT INTO medications(name, dosage,
manufacturer) VALUES (%s, %s, %s);",(fake.word(), f"{random.randint(1,
500)} mg", fake.company()))


    conn.commit()

    # Retrieve IDs for foreign key relationships
    cursor.execute(f"SELECT doctor_id FROM doctors_info;")
    doctor_ids = [row['doctor_id'] for row in cursor.fetchall()]

    cursor.execute(f"SELECT patient_id FROM patient_records;")
    patient_ids = [row['patient_id'] for row in cursor.fetchall()]

    cursor.execute(f"SELECT medication_id FROM medications;")
    medication_ids = [row['medication_id'] for row in cursor.fetchall()]

    for _ in range(20):
        # appointments
        cursor.execute(f"INSERT INTO appointments(patient_id, doctor_id,
appointment_date, purpose) VALUES (%s, %s, %s,
%s);",(random.choice(patient_ids), random.choice(doctor_ids),
fake.date_time_this_month(), fake.sentence()))

        # medical_history
        cursor.execute(f"INSERT INTO medical_history(patient_id,
diagnosis, treatment, date_of_visit) VALUES (%s, %s, %s, %s);",
(random.choice(patient_ids), fake.sentence(), fake.sentence(),
fake.date_time_this_month()))

        # billing_insurance
```

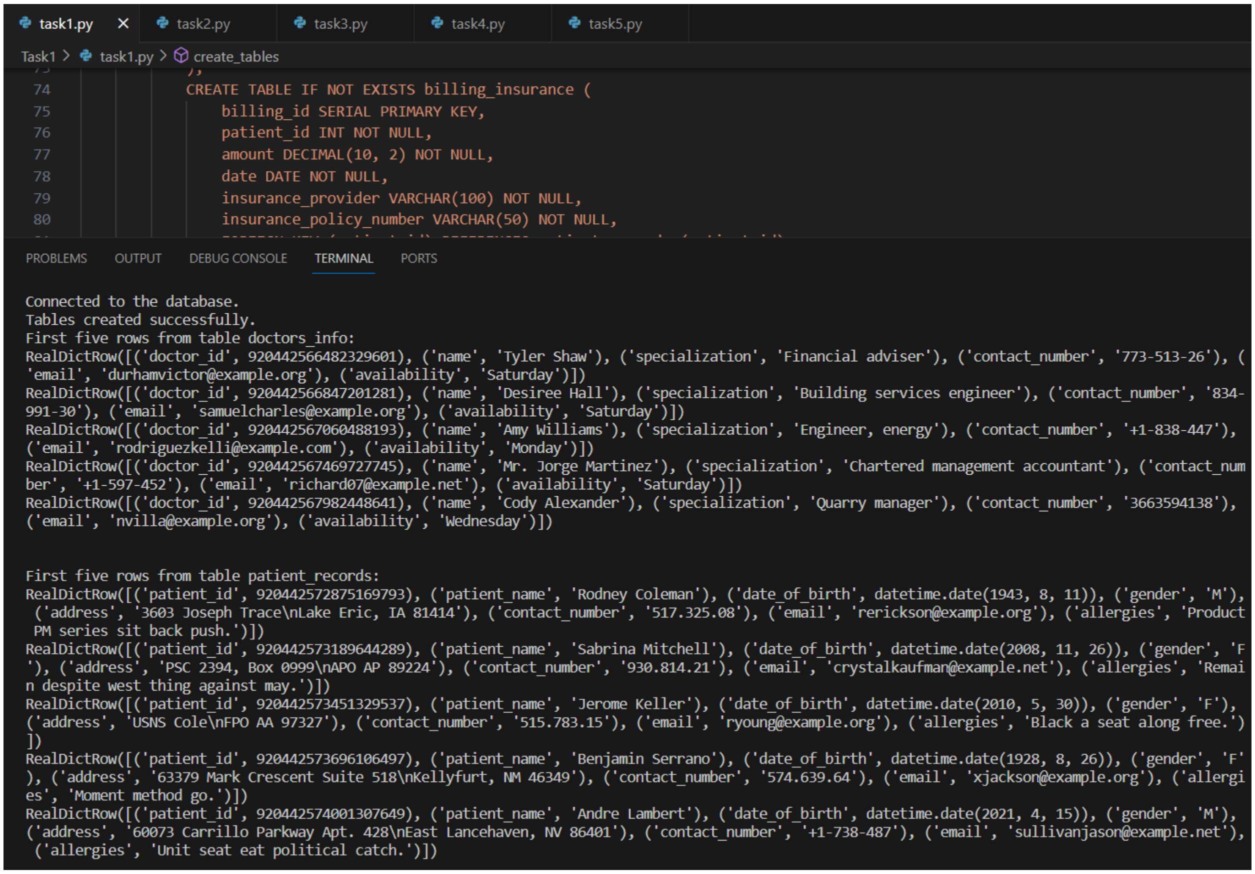
```
        cursor.execute(f"INSERT INTO billing_insurance(patient_id, amount,
date, insurance_provider, insurance_policy_number) VALUES (%s, %s, %s, %s,
%s);",(random.choice(patient_ids), round(random.uniform(100, 10000), 2),
fake.date_this_year(), fake.company(),
fake.bothify(text='????-########')))

        # patient_medications
    used_combinations = set()
    while len(used_combinations) < 20:
        patient_id_choice = random.choice(patient_ids)
        medication_id_choice = random.choice(medication_ids)
        if (patient_id_choice, medication_id_choice) not in
used_combinations:
            try:
                cursor.execute(
                    """
                    INSERT INTO patient_medications(patient_id,
medication_id)
                    VALUES (%s, %s);
                    """,
                    (patient_id_choice, medication_id_choice)
                )
                used_combinations.add((patient_id_choice,
medication_id_choice))
            except psycopg2.errors.UniqueViolation:
                continue
            except psycopg2.DatabaseError as error:
                print(f"An error occurred: {error}")
                conn.rollback()
                break
    conn.commit()

def select_table_data(conn):
    """Prints the first five rows of all tables in the database"""
    cursor = conn.cursor()

    tables = ['doctors_info', 'patient_records', 'appointments',
'medical_history', 'medications', 'billing_insurance',
'patient_medications']
```

```
    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")

    cursor.close()

def main():
    conn = connect_db()
    if conn:
        create_tables(conn)
        insert_random_data(conn)
        select_table_data(conn)
        conn.close()


if __name__ == "__main__":
    main()
```

The data is inserted in each respective table using the Faker() library in Python which is used to generate fake but realistic-looking data. It's particularly useful when you need to populate a database with test data that resembles actual user information. The library can create a wide range of data types, including but not limited to names, addresses, phone numbers, emails, dates.

Data Retrieval

Results:




```
    CREATE TABLE IF NOT EXISTS billing_insurance (
        billing_id SERIAL PRIMARY KEY,
        patient_id INT NOT NULL,
        amount DECIMAL(10, 2) NOT NULL,
        date DATE NOT NULL,
        insurance_provider VARCHAR(100) NOT NULL,
        insurance_policy_number VARCHAR(50) NOT NULL,
```

```
Connected to the database.
Tables created successfully.
First five rows from table doctors_info:
RealDictRow([('doctor_id', 920442566482329601), ('name', 'Tyler Shaw'), ('specialization', 'Financial adviser'), ('contact_number', '773-513-26'), ('email', 'durhamvictor@example.org'), ('availability', 'Saturday')])
RealDictRow([('doctor_id', 920442566847201281), ('name', 'Desiree Hall'), ('specialization', 'Building services engineer'), ('contact_number', '834-991-30'), ('email', 'samuelcharles@example.org'), ('availability', 'Saturday')])
RealDictRow([('doctor_id', 920442567060488193), ('name', 'Amy Williams'), ('specialization', 'Engineer, energy'), ('contact_number', '+1-838-447'), ('email', 'rodriguezkelli@example.com'), ('availability', 'Monday')])
RealDictRow([('doctor_id', 920442567469727745), ('name', 'Mr. Jorge Martinez'), ('specialization', 'Chartered management accountant'), ('contact_number', '+1-597-452'), ('email', 'richard07@example.net'), ('availability', 'Saturday')])
RealDictRow([('doctor_id', 920442567982448641), ('name', 'Cody Alexander'), ('specialization', 'Quarry manager'), ('contact_number', '3663594138'), ('email', 'nvilla@example.org'), ('availability', 'Wednesday')])


First five rows from table patient_records:
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', 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', 'Remain 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'), ('allergies', 'Moment method go.')])
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.')])
```

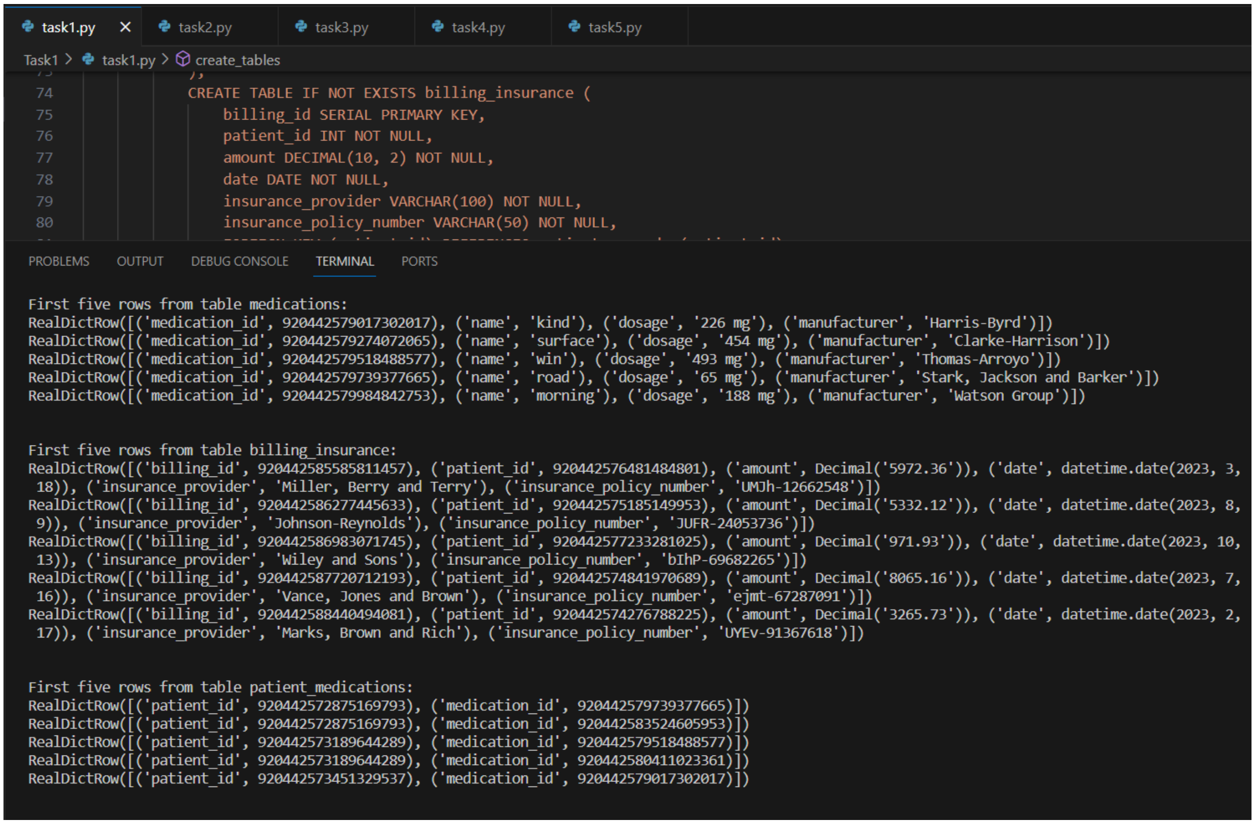
task1.py × | task2.py | task3.py | task4.py | task5.py

Task1 > task1.py > create_tables

```sql
            CREATE TABLE IF NOT EXISTS billing_insurance (
                billing_id SERIAL PRIMARY KEY,
                patient_id INT NOT NULL,
                amount DECIMAL(10, 2) NOT NULL,
                date DATE NOT NULL,
                insurance_provider VARCHAR(100) NOT NULL,
                insurance_policy_number VARCHAR(50) NOT NULL,
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

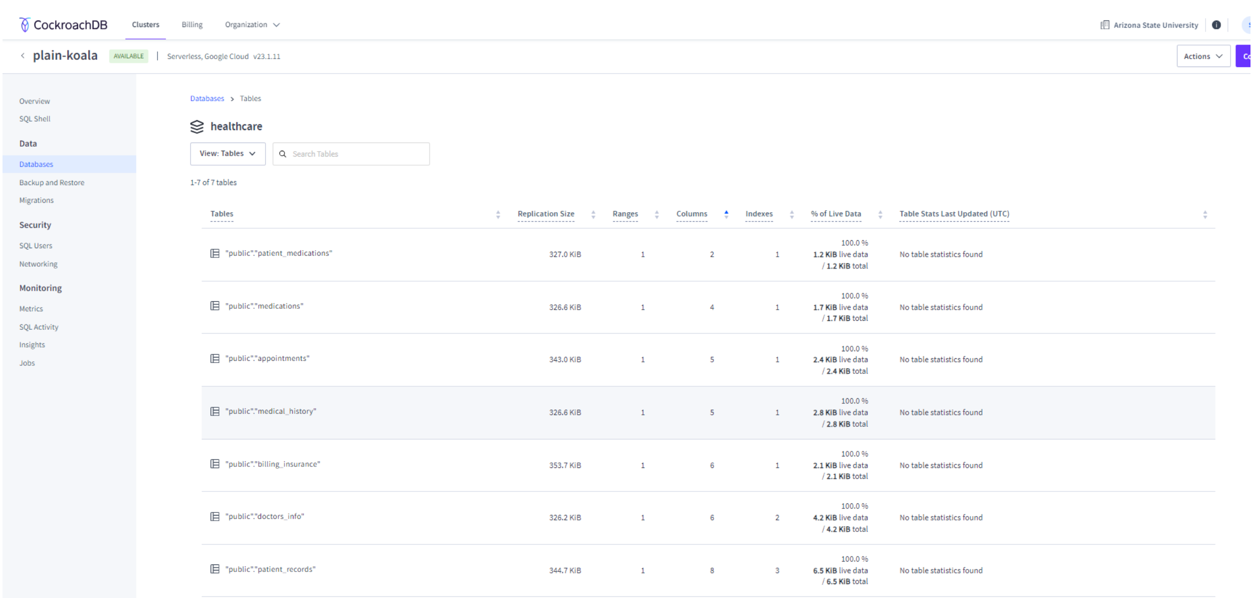
```
 ('allergies', 'Unit seat eat political catch.')])


First five rows from table appointments:
RealDictRow([('appointment_id', 920442584948015105), ('patient_id', 920442576184868865), ('doctor_id', 920442569531981825), ('appointment_date', dat
etime.datetime(2023, 11, 8, 18, 21, 19)), ('purpose', 'Card article picture reach writer amount.')])
RealDictRow([('appointment_id', 920442585822658561), ('patient_id', 920442576856514561), ('doctor_id', 920442566847201281), ('appointment_date', dat
etime.datetime(2023, 11, 2, 19, 16, 28)), ('purpose', 'Doctor kind international product investment TV sign result.')])
RealDictRow([('appointment_id', 920442586523664385), ('patient_id', 920442574001307649), ('doctor_id', 920442572285902849), ('appointment_date', dat
etime.datetime(2023, 11, 2, 14, 57, 22)), ('purpose', 'Day probably pattern compare response.')])
RealDictRow([('appointment_id', 920442587244232705), ('patient_id', 920442573451329537), ('doctor_id', 920442570646519809), ('appointment_date', dat
etime.datetime(2023, 11, 11, 22, 3, 15)), ('purpose', 'Weight tough suddenly identify western.')])
RealDictRow([('appointment_id', 920442587964538881), ('patient_id', 920442578755223553), ('doctor_id', 920442568878555137), ('appointment_date', dat
etime.datetime(2023, 11, 13, 23, 44, 40)), ('purpose', 'Ground because pattern reality relationship care system glass.')])


First five rows from table medical_history:
RealDictRow([('history_id', 920442585225658369), ('patient_id', 920442577506664449), ('diagnosis', 'Consumer benefit because man can something.'), (
'treatment', 'Community put series relate money natural.'), ('date_of_visit', datetime.datetime(2023, 11, 14, 13, 46, 56))])
RealDictRow([('history_id', 920442586047905793), ('patient_id', 920442574001307649), ('diagnosis', 'Matter smile trade car.'), ('treatment', 'Apply
trial property effect same more.'), ('date_of_visit', datetime.datetime(2023, 11, 15, 10, 35, 1))])
RealDictRow([('history_id', 920442586752548865), ('patient_id', 920442578232541185), ('diagnosis', 'Stage make might huge.'), ('treatment', 'List fu
ll particularly message per fire.'), ('date_of_visit', datetime.datetime(2023, 11, 6, 9, 0, 23))])
RealDictRow([('history_id', 920442587473051649), ('patient_id', 920442577233281025), ('diagnosis', 'Both again budget.'), ('treatment', 'Economic ra
te remember trouble lead.'), ('date_of_visit', datetime.datetime(2023, 11, 17, 15, 38, 12))])
RealDictRow([('history_id', 920442588227076097), ('patient_id', 920442573451329537), ('diagnosis', 'Lot expect it order at.'), ('treatment', 'Studen
t such although fight field garden.'), ('date_of_visit', datetime.datetime(2023, 11, 16, 3, 6, 45))])


First five rows from table medications:
RealDictRow([('medication_id', 920442579017302017), ('name', 'kind'), ('dosage', '226 mg'), ('manufacturer', 'Harris-Byrd')])
RealDictRow([('medication_id', 920442579274072065), ('name', 'surface'), ('dosage', '454 mg'), ('manufacturer', 'Clarke-Harrison')])
```




```
    CREATE TABLE IF NOT EXISTS billing_insurance (
        billing_id SERIAL PRIMARY KEY,
        patient_id INT NOT NULL,
        amount DECIMAL(10, 2) NOT NULL,
        date DATE NOT NULL,
        insurance_provider VARCHAR(100) NOT NULL,
        insurance_policy_number VARCHAR(50) NOT NULL,
```

```
First five rows from table medications:
RealDictRow([('medication_id', 920442579017302017), ('name', 'kind'), ('dosage', '226 mg'), ('manufacturer', 'Harris-Byrd')])
RealDictRow([('medication_id', 920442579274072065), ('name', 'surface'), ('dosage', '454 mg'), ('manufacturer', 'Clarke-Harrison')])
RealDictRow([('medication_id', 920442579518488577), ('name', 'win'), ('dosage', '493 mg'), ('manufacturer', 'Thomas-Arroyo')])
RealDictRow([('medication_id', 920442579739377665), ('name', 'road'), ('dosage', '65 mg'), ('manufacturer', 'Stark, Jackson and Barker')])
RealDictRow([('medication_id', 920442579984842753), ('name', 'morning'), ('dosage', '188 mg'), ('manufacturer', 'Watson Group')])


First five rows from table billing_insurance:
RealDictRow([('billing_id', 920442585585811457), ('patient_id', 920442576481484801), ('amount', Decimal('5972.36')), ('date', datetime.date(2023, 3,
 18)), ('insurance_provider', 'Miller, Berry and Terry'), ('insurance_policy_number', 'UMJh-12662548')])
RealDictRow([('billing_id', 920442586277445633), ('patient_id', 920442575185149953), ('amount', Decimal('5332.12')), ('date', datetime.date(2023, 8,
 9)), ('insurance_provider', 'Johnson-Reynolds'), ('insurance_policy_number', 'JUFR-24053736')])
RealDictRow([('billing_id', 920442586983071745), ('patient_id', 920442577233281025), ('amount', Decimal('971.93')), ('date', datetime.date(2023, 10,
 13)), ('insurance_provider', 'Wiley and Sons'), ('insurance_policy_number', 'bIhP-69682265')])
RealDictRow([('billing_id', 920442587720712193), ('patient_id', 920442574841970689), ('amount', Decimal('8065.16')), ('date', datetime.date(2023, 7,
 16)), ('insurance_provider', 'Vance, Jones and Brown'), ('insurance_policy_number', 'ejmt-67287091')])
RealDictRow([('billing_id', 920442588440494081), ('patient_id', 920442574276788225), ('amount', Decimal('3265.73')), ('date', datetime.date(2023, 2,
 17)), ('insurance_provider', 'Marks, Brown and Rich'), ('insurance_policy_number', 'UYEv-91367618')])


First five rows from table patient_medications:
RealDictRow([('patient_id', 920442572875169793), ('medication_id', 920442579739377665)])
RealDictRow([('patient_id', 920442572875169793), ('medication_id', 920442583524605953)])
RealDictRow([('patient_id', 920442573189644289), ('medication_id', 920442579518488577)])
RealDictRow([('patient_id', 920442573189644289), ('medication_id', 920442580411023361)])
RealDictRow([('patient_id', 920442573451329537), ('medication_id', 920442579017302017)])
```

After creating the database and creating all the tables, the cockroachDB cluster looks like below screenshot:



Tables	Replication Size	Ranges	Columns	Indexes	% of Live Data	Table Stats Last Updated (UTC)
"public"."patient_medications"	327.0 KiB	1	2	1	100.0 % 1.2 KiB live data / 1.2 KiB total	No table statistics found
"public"."medications"	326.6 KiB	1	4	1	100.0 % 1.7 KiB live data / 1.7 KiB total	No table statistics found
"public"."appointments"	343.0 KiB	1	5	1	100.0 % 2.4 KiB live data / 2.4 KiB total	No table statistics found
"public"."medical_history"	326.6 KiB	1	5	1	100.0 % 2.8 KiB live data / 2.8 KiB total	No table statistics found
"public"."billing_insurance"	353.7 KiB	1	6	1	100.0 % 2.1 KiB live data / 2.1 KiB total	No table statistics found
"public"."doctors_info"	326.2 KiB	1	6	2	100.0 % 4.2 KiB live data / 4.2 KiB total	No table statistics found
"public"."patient_records"	344.7 KiB	1	8	3	100.0 % 6.5 KiB live data / 6.5 KiB total	No table statistics found