Distributed Database Systems (CSE 512)

Group project

Task 3

Query Optimization

Query optimization is the process of altering a query to improve its execution performance. The goal is to reduce the resources required to execute the query, and the time it takes to return the results, without changing the output of the query. Query optimization often requires a deep understanding of both the specific database system being used and the structure of the data it holds. It's an iterative process that involves identifying slow queries, understanding why they are slow, making changes to improve performance, and then verifying that those changes have had the desired effect.

Code:

```
def optimize_queries(conn):
    """Optimizes queries for efficient data retrieval."""
    cursor = conn.cursor()

    cursor.execute("CREATE INDEX IF NOT EXISTS idx_patient_name ON
patient_records(patient_name);")
    cursor.execute("CREATE INDEX IF NOT EXISTS idx_doctor_specialization
ON doctors_info(specialization);")
    cursor.execute("CREATE INDEX IF NOT EXISTS idx_appointments_date ON
appointments(appointment_date);")

    # Explain a query to see the execution plan
    cursor.execute("EXPLAIN SELECT * FROM patient_records WHERE
patient_name = 'John Doe';")
    plan = cursor.fetchall()
```

```
    print("Execution plan for patient_records query:")
    for row in plan:
        print(row)

    # Optimize a query using JOIN and WHERE clauses
    cursor.execute("""
        EXPLAIN SELECT p.patient_name, a.appointment_date, d.name
        FROM appointments a
        JOIN patient_records p ON a.patient_id = p.patient_id
        JOIN doctors_info d ON a.doctor_id = d.doctor_id
        WHERE p.patient_name = 'John Doe' AND a.appointment_date > NOW() -
INTERVAL '1 year';
    """)
    plan = cursor.fetchall()
    print("\nExecution plan for appointments query:")
    for row in plan:
        print(row)

    # Ensure you commit the creation of indexes if outside of a
transaction
    conn.commit()
    cursor.close()

    print("\nQuery optimization completed.\n")

def analyze_complex_queries(conn):
    with conn.cursor() as cursor:
        complex_query = """
        SELECT d.name, COUNT(a.appointment_id) AS appointment_count
        FROM doctors_info d
        JOIN appointments a ON d.doctor_id = a.doctor_id
        GROUP BY d.name
        ORDER BY appointment_count DESC;
        """

        # Execute and explain the complex query
        cursor.execute("EXPLAIN " + complex_query)
        explain_result = cursor.fetchall()
        print("Execution Plan for the Complex Query:")
        for row in explain_result:
```

```
            print(row)

        # Optionally, you can also use EXPLAIN ANALYZE for more detailed
analysis
        cursor.execute("EXPLAIN ANALYZE " + complex_query)
        explain_analyze_result = cursor.fetchall()
        print("\nDetailed Execution Analysis for the Complex Query:")
        for row in explain_analyze_result:
            print(row)
```

The code above is the execution plan generated by the “EXPLAIN” command for two different queries against our PostgreSQL database ‘Project1’.

Execution Plan for patient_records Query:

The plan indicates that PostgreSQL is using an Index Scan on the patient_records table using the “idx_patient_name” index to efficiently find rows where patient_name is 'John Doe'. This is a good result because it means the database is using the index which we created and is more efficient than scanning the entire table.

These two lines describe the Indexing:

“Index Scan using idx_patient_name on patient_records (cost=0.14..8.16 rows=1 width=140)
Index Cond: ((patient_name)::text = 'John Doe'::text)”

The cost in the above line indicates an arbitrary unit of work that PostgreSQL uses to estimate how expensive the query will be to run, with the numbers before and after the dots representing the start-up cost and total cost, respectively. rows=1 indicates that the database expects to find one row matching the condition.

Execution Plan for appointments Query:

This more complex query involves joining the appointments, patient_records, and doctors_info tables.

The database performs a Nested Loop which means it will iterate over the result of one operation for each row returned by another.
Inside the nested loop, a Hash Join is used, which means it's building a hash table in memory based on the patient_id from the patient_records table to efficiently join with the appointments table. A Seq Scan on the appointments table is used, which means it's scanning every row. It's

doing this because the database doesn't expect many rows to match the filter (rows=200 is just an estimate), or there may not be an index it can use to speed up this part of the query.
The Index Scan is used again on patient_records to find 'John Doe', and the result is hashed for the join.
Finally, an Index Scan on the doctors_info table uses the primary key index to find matching rows for the doctor_ID obtained from the appointments table.

Output:

task1.py task2.py task3.py task4.py task5.py

Task3 > task3.py > create_distributed_indexes

```
114     cursor.execute(query2)
115     plan2 = cursor.fetchall()
116     for row in plan2:
117         print("Example 2 : ",row)
118
```

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell - Task3

```
Connected to the database.
Execution Plan for the Complex Query:
RealDictRow([('info', 'distribution: local')])
RealDictRow([('info', 'vectorized: true')])
RealDictRow([('info', '')])
RealDictRow([('info', '• sort')])
RealDictRow([('info', '│ estimated row count: 13')])
RealDictRow([('info', '│ order: -count_rows')])
RealDictRow([('info', '│')])
RealDictRow([('info', '└── • group (hash)')])
RealDictRow([('info', '    │ estimated row count: 13')])
RealDictRow([('info', '    │ group by: name')])
RealDictRow([('info', '    │')])
RealDictRow([('info', '    └── • hash join')])
RealDictRow([('info', '        │ estimated row count: 20')])
RealDictRow([('info', '        │ equality: (doctor_id) = (doctor_id)')])
RealDictRow([('info', '        │ left cols are key')])
RealDictRow([('info', '        │')])
RealDictRow([('info', '        ├── • scan')])
RealDictRow([('info', '        │     estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the future)')])
RealDictRow([('info', '        │     table: doctors_info@doctors_info_pkey')])
RealDictRow([('info', '        │     spans: FULL SCAN')])
RealDictRow([('info', '        │')])
RealDictRow([('info', '        └── • scan')])
RealDictRow([('info', '              estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the future)')])
RealDictRow([('info', '              table: appointments@appointments_pkey')])
RealDictRow([('info', '              spans: FULL SCAN')])
RealDictRow([('info', '')])
RealDictRow([('info', 'index recommendations: 1')])
RealDictRow([('info', '1. type: index creation')])
RealDictRow([('info', '   SQL command: CREATE INDEX ON healthcare.public.appointments (doctor_id);')])

Detailed Execution Analysis for the Complex Query:
RealDictRow([('info', 'planning time: 560µs')])
RealDictRow([('info', 'execution time: 15ms')])
```

task1.py task2.py task3.py task4.py task5.py

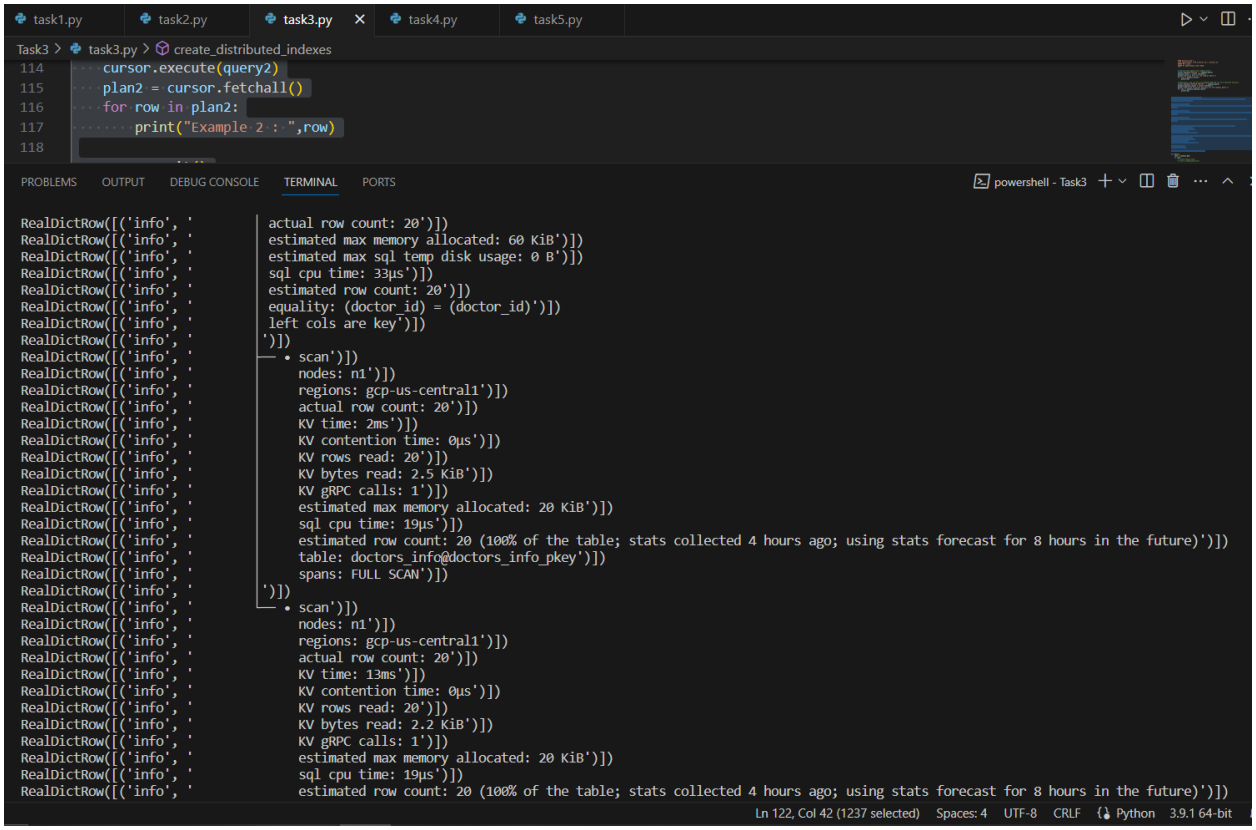
Task3 > task3.py > create_distributed_indexes

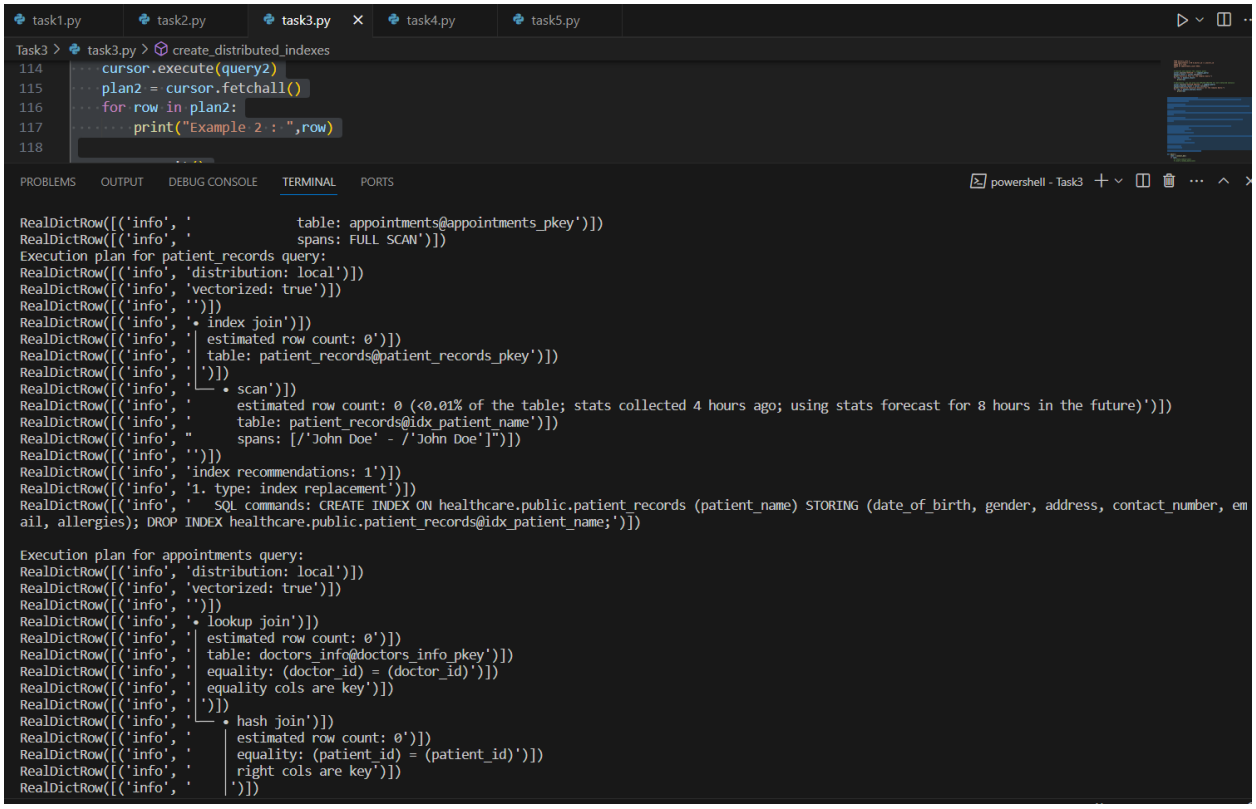
```
114     cursor.execute(query2)
115     plan2 = cursor.fetchall()
116     for row in plan2:
117         print("Example 2 : ",row)
118
```

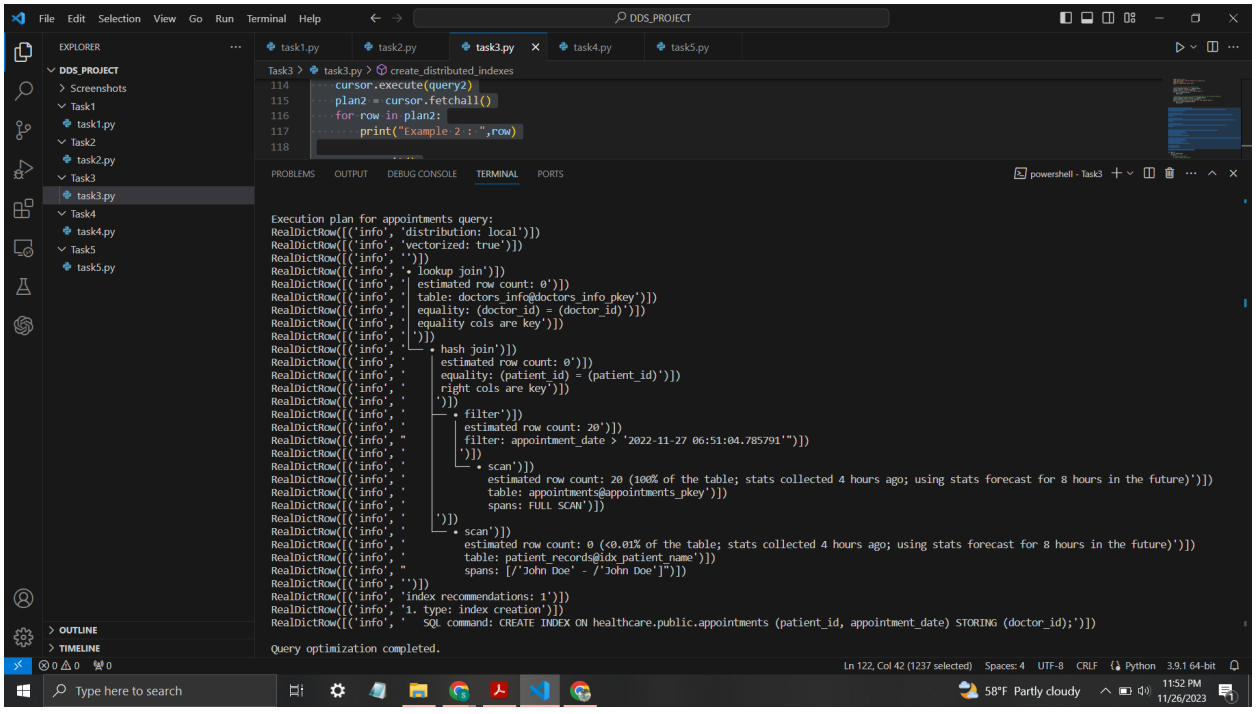
PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell - Task3

```
Detailed Execution Analysis for the Complex Query:
RealDictRow([('info', 'planning time: 560µs')])
RealDictRow([('info', 'execution time: 15ms')])
RealDictRow([('info', 'distribution: local')])
RealDictRow([('info', 'vectorized: true')])
RealDictRow([('info', 'rows read from KV: 40 (4.8 KiB, 2 gRPC calls)')])
RealDictRow([('info', 'cumulative time spent in KV: 14ms')])
RealDictRow([('info', 'maximum memory usage: 180 KiB')])
RealDictRow([('info', 'network usage: 0 B (0 messages)')])
RealDictRow([('info', 'regions: gcp-us-central1')])
RealDictRow([('info', 'sql cpu time: 125µs')])
RealDictRow([('info', 'estimated RUs consumed: 0')])
RealDictRow([('info', '')])
RealDictRow([('info', '• sort')])
RealDictRow([('info', '│ nodes: n1')])
RealDictRow([('info', '│ regions: gcp-us-central1')])
RealDictRow([('info', '│ actual row count: 15')])
RealDictRow([('info', '│ estimated max memory allocated: 10 KiB')])
RealDictRow([('info', '│ estimated max sql temp disk usage: 0 B')])
RealDictRow([('info', '│ sql cpu time: 11µs')])
RealDictRow([('info', '│ estimated row count: 13')])
RealDictRow([('info', '│ order: -count_rows')])
RealDictRow([('info', '│')])
RealDictRow([('info', '└── • group (hash)')])
RealDictRow([('info', '    │ nodes: n1')])
RealDictRow([('info', '    │ regions: gcp-us-central1')])
RealDictRow([('info', '    │ actual row count: 15')])
RealDictRow([('info', '    │ estimated max memory allocated: 70 KiB')])
RealDictRow([('info', '    │ estimated max sql temp disk usage: 0 B')])
RealDictRow([('info', '    │ sql cpu time: 44µs')])
RealDictRow([('info', '    │ estimated row count: 13')])
RealDictRow([('info', '    │ group by: name')])
RealDictRow([('info', '    │')])
RealDictRow([('info', '    └── • hash join')])
```


task1.py
task2.py
task3.py
task4.py
task5.py
Task3 > task3.py > create_distributed_indexes
114 cursor.execute(query2)
115 plan2 = cursor.fetchall()
116 for row in plan2:
117 print("Example 2 : ",row)
118
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell - Task3
RealDictRow([('info', '
actual row count: 20')])
estimated max memory allocated: 60 KiB')])
estimated max sql temp disk usage: 0 B')])
sql cpu time: 33µs')])
estimated row count: 20')])
equality: (doctor_id) = (doctor_id)')])
left cols are key')])
')])
└─ • scan')])
nodes: n1')])
regions: gcp-us-central1')])
actual row count: 20')])
KV time: 2ms')])
KV contention time: 0µs')])
KV rows read: 20')])
KV bytes read: 2.5 KiB')])
KV gRPC calls: 1')])
estimated max memory allocated: 20 KiB')])
sql cpu time: 19µs')])
estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the future)')])
table: doctors_info@doctors_info_pkey')])
spans: FULL SCAN')])
')])
└─ • scan')])
nodes: n1')])
regions: gcp-us-central1')])
actual row count: 20')])
KV time: 13ms')])
KV contention time: 0µs')])
KV rows read: 20')])
KV bytes read: 2.2 KiB')])
KV gRPC calls: 1')])
estimated max memory allocated: 20 KiB')])
sql cpu time: 19µs')])
estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the future)')])
Ln 122, Col 42 (1237 selected)
Spaces: 4
UTF-8
CRLF
Python
3.9.1 64-bit





The query optimizer has decided to use these methods based on the available indexes and the statistics it has about the tables. From this output, it seems the query is optimized well because it's using indexes. However, the Seq Scan on the appointments table could potentially be improved with an index on appointment_date, especially if the table grows large.

Distributed Indexing

Distributed indexing is a strategy used to improve the performance of queries in a distributed database system. In such a system, data is stored across multiple locations, and indexes need to be managed in a way that takes into account the distribution of the data.

Code:

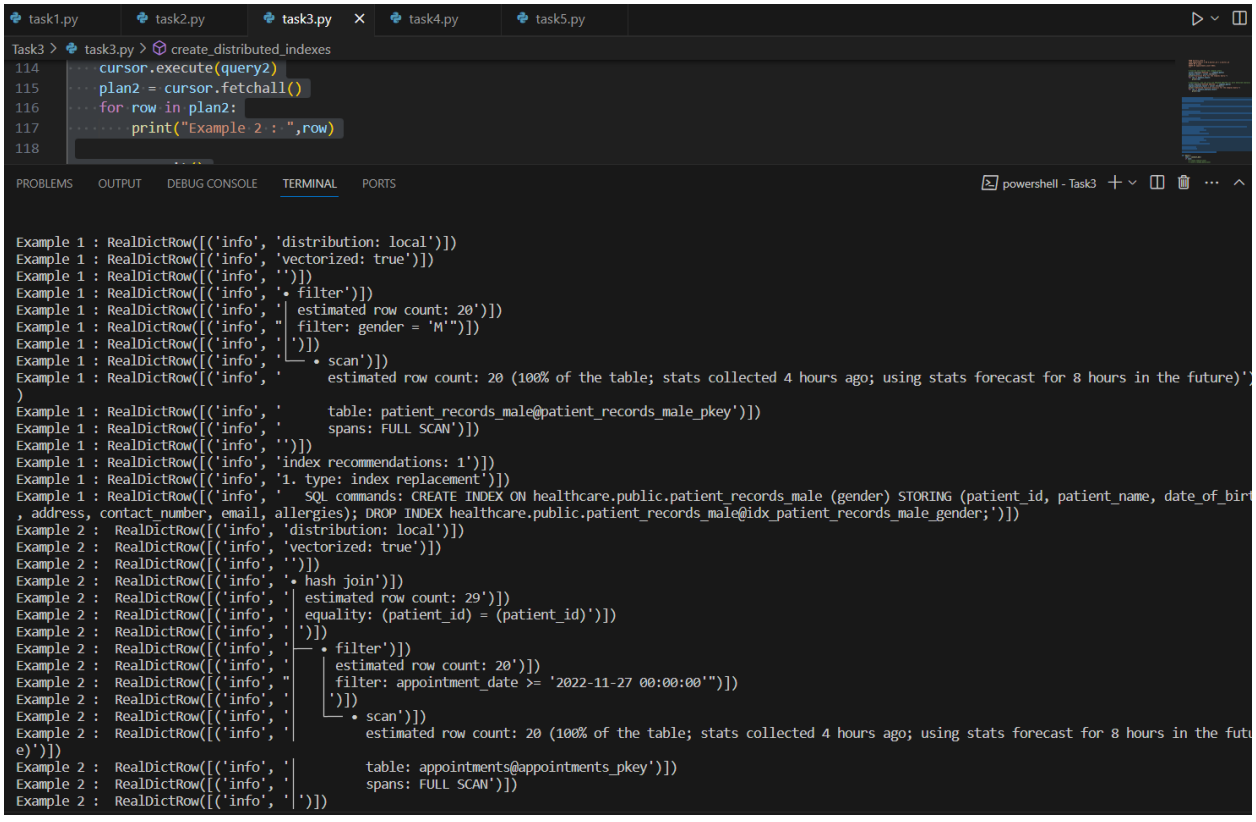
```
def create_distributed_indexes(conn):
    """Creates distributed indexes on partitioned tables in a distributed environment."""
    cursor = conn.cursor()

    cursor.execute("""
        CREATE INDEX IF NOT EXISTS idx_patient_records_male_gender ON patient_records_male(gender);
    """)

    cursor.execute("""
        CREATE INDEX IF NOT EXISTS idx_patient_records_female_gender ON patient_records_female(gender);
    """)

    cursor.execute("""
        CREATE INDEX IF NOT EXISTS idx_appointments_date ON appointments(appointment_date);
    """)


    query1 = "EXPLAIN SELECT * FROM patient_records_male WHERE gender = 'M';"
    cursor.execute(query1)
    plan1 = cursor.fetchall()
    for row in plan1:
```

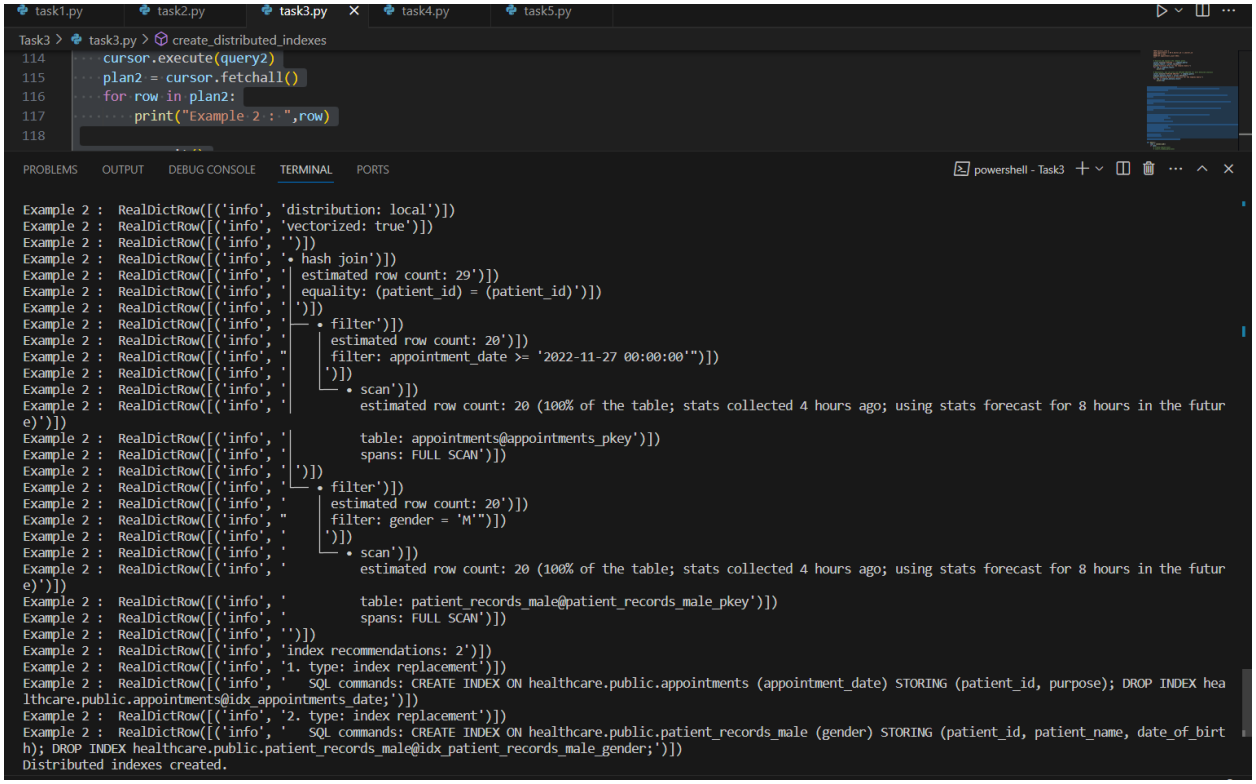
```
        print("Example 1 :",row)

    query2 = "EXPLAIN SELECT pr.patient_name, pr.date_of_birth, pr.gender,
a.appointment_date, a.purpose FROM appointments a JOIN
patient_records_male pr ON a.patient_id = pr.patient_id WHERE pr.gender =
'M' AND a.appointment_date >= CURRENT_DATE - INTERVAL '1 year';"
    cursor.execute(query2)
    plan2 = cursor.fetchall()
    for row in plan2:
        print("Example 2 : ",row)

    conn.commit()
    cursor.close()

    print("Distributed indexes created.")
```

The function executes an EXPLAIN command on a query that selects all columns from the patient_records_male table where the gender is 'M'. This EXPLAIN statement is used to show the execution plan for the query, which outlines how the database would execute this query without actually running it.
It prints out the execution plan, which will typically indicate whether the query planner intends to use the index just created “idx_patient_records_male_gender” to perform an index scan for this query.

The second query executes an EXPLAIN command on a more complex query that joins the appointments table with the patient_records_male table on patient_id and filters records based on gender and appointment_date.
It prints out the execution plan for this query, which would show if the database is planning to use the appropriate indexes for both the join operation and the filter conditions.

Output :


task1.py
task2.py
task3.py
task4.py
task5.py
Task3 > task3.py > create_distributed_indexes
cursor.execute(query2)
plan2 = cursor.fetchall()
for row in plan2:
print("Example 2 : ",row)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell - Task3
Example 1 : RealDictRow([('info', 'distribution: local')])
Example 1 : RealDictRow([('info', 'vectorized: true')])
Example 1 : RealDictRow([('info', '')])
Example 1 : RealDictRow([('info', '• filter')])
Example 1 : RealDictRow([('info', '│ estimated row count: 20')])
Example 1 : RealDictRow([('info', "│ filter: gender = 'M'")])
Example 1 : RealDictRow([('info', '└')])
Example 1 : RealDictRow([('info', '└── • scan')])
Example 1 : RealDictRow([('info', '   estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the future)')
)
Example 1 : RealDictRow([('info', '   table: patient_records_male@patient_records_male_pkey')])
Example 1 : RealDictRow([('info', '   spans: FULL SCAN')])
Example 1 : RealDictRow([('info', '')])
Example 1 : RealDictRow([('info', 'index recommendations: 1')])
Example 1 : RealDictRow([('info', '1. type: index replacement')])
Example 1 : RealDictRow([('info', '   SQL commands: CREATE INDEX ON healthcare.public.patient_records_male (gender) STORING (patient_id, patient_name, date_of_birt
, address, contact_number, email, allergies); DROP INDEX healthcare.public.patient_records_male@idx_patient_records_male_gender;')])
Example 2 : RealDictRow([('info', 'distribution: local')])
Example 2 : RealDictRow([('info', 'vectorized: true')])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', '• hash join')])
Example 2 : RealDictRow([('info', '│ estimated row count: 29')])
Example 2 : RealDictRow([('info', '│ equality: (patient_id) = (patient_id)')])
Example 2 : RealDictRow([('info', '│')])
Example 2 : RealDictRow([('info', '├── • filter')])
Example 2 : RealDictRow([('info', '│   │ estimated row count: 20')])
Example 2 : RealDictRow([('info', "│   │ filter: appointment_date >= '2022-11-27 00:00:00'")])
Example 2 : RealDictRow([('info', '│   │')])
Example 2 : RealDictRow([('info', '│   └── • scan')])
Example 2 : RealDictRow([('info', '│   estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the futu
e)')])
Example 2 : RealDictRow([('info', '│   table: appointments@appointments_pkey')])
Example 2 : RealDictRow([('info', '│   spans: FULL SCAN')])
Example 2 : RealDictRow([('info', '')])


task1.py
task2.py
task3.py
task4.py
task5.py
Task3 > task3.py > create_distributed_indexes
cursor.execute(query2)
plan2 = cursor.fetchall()
for row in plan2:
print("Example 2 : ",row)
PROBLEMS
OUTPUT
DEBUG CONSOLE
TERMINAL
PORTS
powershell - Task3
Example 2 : RealDictRow([('info', 'distribution: local')])
Example 2 : RealDictRow([('info', 'vectorized: true')])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', '• hash join')])
Example 2 : RealDictRow([('info', '│ estimated row count: 29')])
Example 2 : RealDictRow([('info', '│ equality: (patient_id) = (patient_id)')])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', '├── • filter')])
Example 2 : RealDictRow([('info', '│ estimated row count: 20')])
Example 2 : RealDictRow([('info', "│ filter: appointment_date >= '2022-11-27 00:00:00'")])
Example 2 : RealDictRow([('info', '│')])
Example 2 : RealDictRow([('info', '└── • scan')])
Example 2 : RealDictRow([('info', ' estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the futur
e)')])
Example 2 : RealDictRow([('info', ' table: appointments@appointments_pkey')])
Example 2 : RealDictRow([('info', ' spans: FULL SCAN')])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', '└── • filter')])
Example 2 : RealDictRow([('info', ' estimated row count: 20')])
Example 2 : RealDictRow([('info', " filter: gender = 'M'")])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', ' └── • scan')])
Example 2 : RealDictRow([('info', ' estimated row count: 20 (100% of the table; stats collected 4 hours ago; using stats forecast for 8 hours in the futur
e)')])
Example 2 : RealDictRow([('info', ' table: patient_records_male@patient_records_male_pkey')])
Example 2 : RealDictRow([('info', ' spans: FULL SCAN')])
Example 2 : RealDictRow([('info', '')])
Example 2 : RealDictRow([('info', 'index recommendations: 2')])
Example 2 : RealDictRow([('info', '1. type: index replacement')])
Example 2 : RealDictRow([('info', '   SQL commands: CREATE INDEX ON healthcare.public.appointments (appointment_date) STORING (patient_id, purpose); DROP INDEX hea
lthcare.public.appointments@idx_appointments_date;')])
Example 2 : RealDictRow([('info', '2. type: index replacement')])
Example 2 : RealDictRow([('info', '   SQL commands: CREATE INDEX ON healthcare.public.patient_records_male (gender) STORING (patient_id, patient_name, date_of_birt
h); DROP INDEX healthcare.public.patient_records_male@idx_patient_records_male_gender;')])
Distributed indexes created.