CSE 512: Distributed Database Systems

Project Report: Query Processing and Optimization Techniques (Part 3)

Group Name: Data Dominators

1. Introduction

The aim of Part 3 of our project is to implement query processing and optimization techniques to enhance the performance of our distributed database system. Our focus is on query optimization and distributed indexing to ensure efficient data retrieval and processing across the distributed environment.

2. Implementation

We have chosen PostgreSQL for our database system and simulated a distributed database environment. Within this framework, we have implemented the following components:

2.1. **Query Optimization**: Utilizing EXPLAIN and EXPLAIN ANALYZE commands, we have analyzed and optimized queries for efficient data retrieval.

```python
#Optimization Script using EXPLAIN and EXPLAIN ANALYZE
def run_explain(query, conn):
    """Run EXPLAIN and EXPLAIN ANALYZE on a given query and print the results."""
    with conn.cursor() as cur:
        cur.execute(f"EXPLAIN {query}")
        explain_output = cur.fetchall()
        print("EXPLAIN Output:")
        for row in explain_output:
            print(row[0])
        cur.execute(f"EXPLAIN ANALYZE {query}")
        explain_analyze_output = cur.fetchall()
        print("\nEXPLAIN ANALYZE Output:")
        for row in explain_analyze_output:
            print(row[0])
```

2.2. **Distributed Indexing**: We have created indexes on simulated child databases, representing a distributed environment to demonstrate the impact of indexing strategies on query performance.

```python
def create_index(conn, table_name, column_name):
    """Create an index on a specified column of a table."""
    with conn.cursor() as cursor:
        cursor.execute(f"CREATE INDEX IF NOT EXISTS idx_{column_name} ON {table_name} (
        conn.commit()
        print(f"Created index on column '{column_name}' of table '{table_name}'.")
```

```
# Simulate a distributed query that fetches user profiles based on location
query = "SELECT * FROM userprofile WHERE location = %s;"
query_params = ("ndislvc niducsadxknccsakjnddxisjsx",)
```

2.3. **Query Optimization Techniques**: We have provided a script that uses PostgreSQL's native capabilities to run EXPLAIN and EXPLAIN ANALYZE on a set of queries.
Performance metrics are captured using Python's time library to measure query execution times before and after optimizations.

2.4. **Distributed Indexing Simulation**: Our Python script demonstrates the concept of distributed indexing by creating and querying indexes on each child database. By partitioning the dataset based on user ID and distributing it across child databases, we simulate a sharded environment.

3. Code Structure and Usage

The code is modularized into functions for database creation, connection, and execution of query optimization strategies.
We have included functions to generate random user profiles and batch insert them into the databases, simulating real-world usage.

4. Testing and Error Handling

The script includes robust error handling and logging to capture any issues during query execution. Performance improvements are logged to show the effectiveness of query optimizations.

Before applying the indexing, searching for any username or attribute in the table, would take some time to process and retrieve the results. After creating the index, it reduces the searching process within 0 seconds, that is negligible.

```
Query took 0.1261 seconds on database
total_time_without_index is 0.12609028816223145
Created index on column 'location' of table 'userprofile'.
Query took 0.0000 seconds on database
total_time_with_index is 0.0
```

Using the features of PostgreSQL like Joins, groupby, optimized the query processing by 50% better than the usual query for the data retrieval.

```
    original_query = """
    SELECT
      u.userid,
      u.username,
      u.location,
      u.email,
      (SELECT COUNT(*) FROM post p WHERE p.userid = u.userid) as post_count
    FROM
      userprofile u;
    """

    optimized_query = """
    SELECT
      u.userid,
      u.username,
      u.location,
      u.email,
      COUNT(p.postid) as post_count
    FROM
      userprofile u
    LEFT JOIN
      post p ON u.userid = p.userid
    GROUP BY
      u.userid, u.username, u.location, u.email;
    """
```

PROBLEMS 2 OUTPUT DEBUG CONSOLE TERMINAL PORTS

```
PS C:\Personals\ASU\Course\Distributed Database Systems\DDS_CSE512\DDS_CSE512\Project\Github repo\dds_project> python project_section_3.py
Query Time: 4.01 seconds
Query Time: 2.11 seconds
Original query duration: 4.01 seconds
Optimized query duration: 2.11 seconds
Performance improvement: 47.40%
```

5. Conclusion

Our implementation of query processing and optimization techniques has resulted in measurable improvements in query execution times.
The use of distributed indexing has demonstrated potential performance benefits in a distributed database system.

6. Future Enhancements

Future work may include integrating open-source tools such as Apache Calcite for further optimization and exploring additional indexing strategies.

7. Deliverables

The deliverables for this part include the provided Python script (project_section_3.py), documentation of the optimization process, and snapshots capturing the execution of the code.