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

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.

1. **Implementation**

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

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

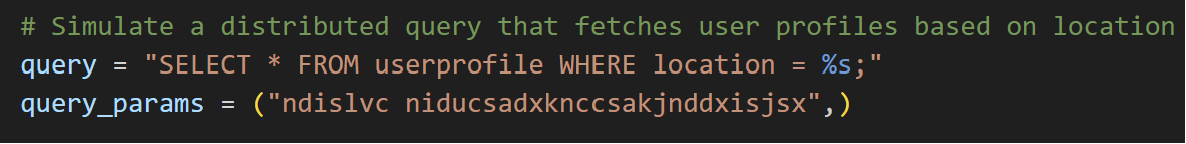
A screen shot of a computer program

Description automatically generated

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

A screenshot of a computer code

Description automatically generated



* 1. **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.

* 1. **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.

1. **Code Structure and Usage**

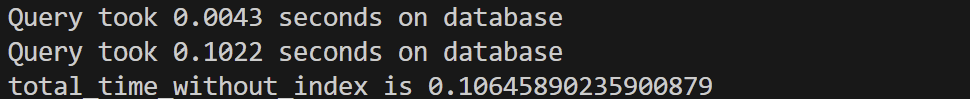
The code is modularized into functions for database creation, connection, and execution of query optimization strategies.

We have included functions to generate random user profiles and batch insert them into the databases, simulating real-world usage.

1. **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 by 3 times.

A black screen with white text

Description automatically generated

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

A screenshot of a computer

Description automatically generated

1. **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.

1. **Future Enhancements**

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

1. **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.