Part 3 : Query Processing and Optimization Techniques

1. Introduction:

The goal of this project is to improve the performance of a distributed database system through the implementation of query processing and optimization techniques. Two key areas of focus are query optimization and distributed indexing. In this report, we will discuss the techniques employed, tools utilized, and the achieved results.

2. Query Optimization:

Analysis and Optimization:

The project involved the analysis and optimization of SQL queries for efficient data retrieval. The chosen database system, PostgreSQL, was utilized to enhance the performance of queries.
The optimization process aimed to minimize unnecessary table joins and select relevant data efficiently.

Dynamically Generated Queries:

A Python script was developed to dynamically generate optimized queries based on the original queries.
The script extracted key components (SELECT, FROM, WHERE) from the original query and built a template for the JOIN part.
The optimized query focused on selecting relevant rows first and then performing joins, optimizing the data retrieval process.
Distributed Indexing:

Index Creation and Deletion:

Indexes play a crucial role in improving query performance. The script created and dropped indexes dynamically to assess their impact on query execution times.
Indexes were strategically placed on columns involved in WHERE clause conditions and join conditions.

3. Tools Used:

PostgreSQL:

PostgreSQL was chosen as the primary database system for this project.
The built-in query optimization capabilities of PostgreSQL were leveraged to enhance the efficiency of queries.

Python (Scripting):

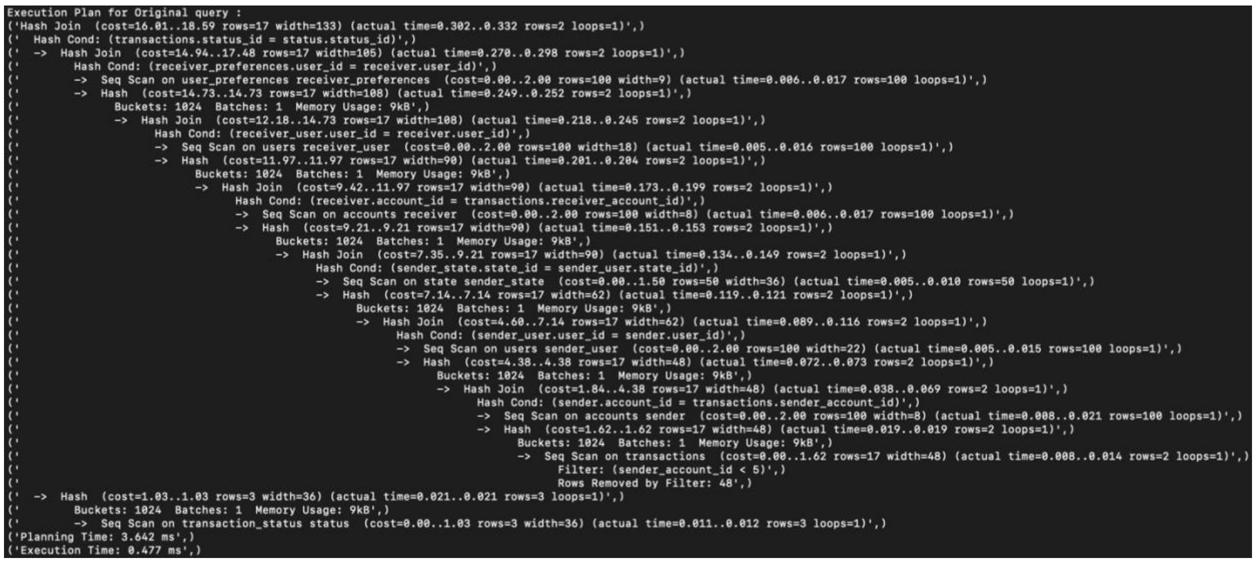
A Python script was developed to automate the query optimization process.
The script connected to the PostgreSQL database, executed queries, measured execution times, and dynamically generated optimized queries.
Deliverables:

Code/Script:

The Python script used for query optimization, index creation, and execution time measurement.
Snapshots:

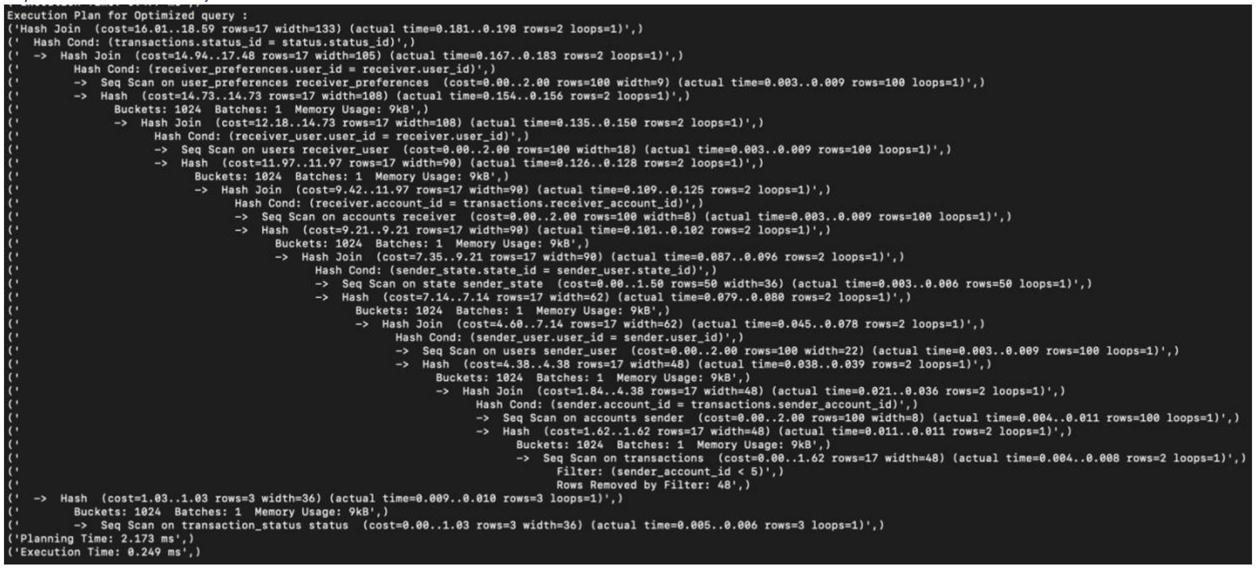
4. Snapshots of the execution plans generated using the EXPLAIN ANALYZE feature in PostgreSQL.

Original Query:



```
Execution Plan for Original query :
('Hash Join  (cost=16.01..18.59 rows=17 width=133) (actual time=0.302..0.332 rows=2 loops=1)',)
('  Hash Cond: (transactions.status_id = status.status_id)',)
('  ->  Hash Join  (cost=14.94..17.48 rows=17 width=105) (actual time=0.270..0.298 rows=2 loops=1)',)
('        Hash Cond: (receiver_preferences.user_id = receiver.user_id)',)
('        ->  Seq Scan on user_preferences receiver_preferences  (cost=0.00..2.00 rows=100 width=9) (actual time=0.006..0.017 rows=100 loops=1)',)
('        ->  Hash  (cost=14.73..14.73 rows=17 width=108) (actual time=0.249..0.252 rows=2 loops=1)',)
('              Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('              ->  Hash Join  (cost=12.18..14.73 rows=17 width=108) (actual time=0.218..0.245 rows=2 loops=1)',)
('                    Hash Cond: (receiver_user.user_id = receiver.user_id)',)
('                    ->  Seq Scan on users receiver_user  (cost=0.00..2.00 rows=100 width=18) (actual time=0.005..0.016 rows=100 loops=1)',)
('                    ->  Hash  (cost=11.97..11.97 rows=17 width=90) (actual time=0.201..0.204 rows=2 loops=1)',)
('                          Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                          ->  Hash Join  (cost=9.42..11.97 rows=17 width=90) (actual time=0.173..0.199 rows=2 loops=1)',)
('                                Hash Cond: (receiver.account_id = transactions.receiver_account_id)',)
('                                ->  Seq Scan on accounts receiver  (cost=0.00..2.00 rows=100 width=8) (actual time=0.006..0.017 rows=100 loops=1)',)
('                                ->  Hash  (cost=9.21..9.21 rows=17 width=90) (actual time=0.151..0.153 rows=2 loops=1)',)
('                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                      ->  Hash Join  (cost=7.35..9.21 rows=17 width=90) (actual time=0.134..0.149 rows=2 loops=1)',)
('                                            Hash Cond: (sender_state.state_id = sender_user.state_id)',)
('                                            ->  Seq Scan on state sender_state  (cost=0.00..1.50 rows=50 width=36) (actual time=0.005..0.010 rows=50 loops=1)',)
('                                            ->  Hash  (cost=7.14..7.14 rows=17 width=62) (actual time=0.119..0.121 rows=2 loops=1)',)
('                                                  Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                  ->  Hash Join  (cost=4.60..7.14 rows=17 width=62) (actual time=0.089..0.116 rows=2 loops=1)',)
('                                                        Hash Cond: (sender_user.user_id = sender.user_id)',)
('                                                        ->  Seq Scan on users sender_user  (cost=0.00..2.00 rows=100 width=22) (actual time=0.005..0.015 rows=100 loops=1)',)
('                                                        ->  Hash  (cost=4.38..4.38 rows=17 width=48) (actual time=0.072..0.073 rows=2 loops=1)',)
('                                                              Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                              ->  Hash Join  (cost=1.84..4.38 rows=17 width=48) (actual time=0.038..0.069 rows=2 loops=1)',)
('                                                                    Hash Cond: (sender.account_id = transactions.sender_account_id)',)
('                                                                    ->  Seq Scan on accounts sender  (cost=0.00..2.00 rows=100 width=8) (actual time=0.008..0.021 rows=100 loops=1)',)
('                                                                    ->  Hash  (cost=1.62..1.62 rows=17 width=48) (actual time=0.019..0.019 rows=2 loops=1)',)
('                                                                          Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                                          ->  Seq Scan on transactions  (cost=0.00..1.62 rows=17 width=48) (actual time=0.008..0.014 rows=2 loops=1)',)
('                                                                                Filter: (sender_account_id < 5)',)
('                                                                                Rows Removed by Filter: 48',)
('  ->  Hash  (cost=1.03..1.03 rows=3 width=36) (actual time=0.021..0.021 rows=3 loops=1)',)
('        Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('        ->  Seq Scan on transaction_status status  (cost=0.00..1.03 rows=3 width=36) (actual time=0.011..0.012 rows=3 loops=1)',)
('Planning Time: 3.642 ms',)
('Execution Time: 0.477 ms',)
```

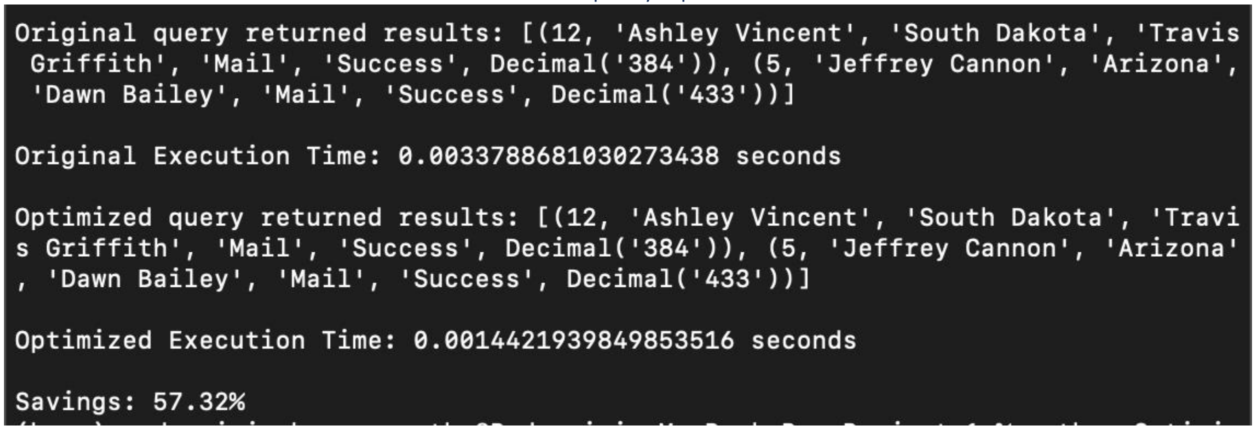
Optimized Query:



```
Execution Plan for Optimized query :
('Hash Join  (cost=16.01..18.59 rows=17 width=133) (actual time=0.181..0.198 rows=2 loops=1)',)
('  Hash Cond: (transactions.status_id = status.status_id)',)
('  ->  Hash Join  (cost=14.94..17.48 rows=17 width=105) (actual time=0.167..0.183 rows=2 loops=1)',)
('        Hash Cond: (receiver_preferences.user_id = receiver.user_id)',)
('        ->  Seq Scan on user_preferences receiver_preferences  (cost=0.00..2.00 rows=100 width=9) (actual time=0.003..0.009 rows=100 loops=1)',)
('        ->  Hash  (cost=14.73..14.73 rows=17 width=108) (actual time=0.154..0.156 rows=2 loops=1)',)
('              Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('              ->  Hash Join  (cost=12.18..14.73 rows=17 width=108) (actual time=0.135..0.150 rows=2 loops=1)',)
('                    Hash Cond: (receiver_user.user_id = receiver.user_id)',)
('                    ->  Seq Scan on users receiver_user  (cost=0.00..2.00 rows=100 width=18) (actual time=0.003..0.009 rows=100 loops=1)',)
('                    ->  Hash  (cost=11.97..11.97 rows=17 width=90) (actual time=0.126..0.128 rows=2 loops=1)',)
('                          Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                          ->  Hash Join  (cost=9.42..11.97 rows=17 width=90) (actual time=0.109..0.125 rows=2 loops=1)',)
('                                Hash Cond: (receiver.account_id = transactions.receiver_account_id)',)
('                                ->  Seq Scan on accounts receiver  (cost=0.00..2.00 rows=100 width=8) (actual time=0.003..0.009 rows=100 loops=1)',)
('                                ->  Hash  (cost=9.21..9.21 rows=17 width=90) (actual time=0.101..0.102 rows=2 loops=1)',)
('                                      Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                      ->  Hash Join  (cost=7.35..9.21 rows=17 width=90) (actual time=0.087..0.096 rows=2 loops=1)',)
('                                            Hash Cond: (sender_state.state_id = sender_user.state_id)',)
('                                            ->  Seq Scan on state sender_state  (cost=0.00..1.50 rows=50 width=36) (actual time=0.003..0.006 rows=50 loops=1)',)
('                                            ->  Hash  (cost=7.14..7.14 rows=17 width=62) (actual time=0.079..0.080 rows=2 loops=1)',)
('                                                  Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                  ->  Hash Join  (cost=4.60..7.14 rows=17 width=62) (actual time=0.045..0.078 rows=2 loops=1)',)
('                                                        Hash Cond: (sender_user.user_id = sender.user_id)',)
('                                                        ->  Seq Scan on users sender_user  (cost=0.00..2.00 rows=100 width=22) (actual time=0.003..0.009 rows=100 loops=1)',)
('                                                        ->  Hash  (cost=4.38..4.38 rows=17 width=48) (actual time=0.038..0.039 rows=2 loops=1)',)
('                                                              Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                              ->  Hash Join  (cost=1.84..4.38 rows=17 width=48) (actual time=0.021..0.036 rows=2 loops=1)',)
('                                                                    Hash Cond: (sender.account_id = transactions.sender_account_id)',)
('                                                                    ->  Seq Scan on accounts sender  (cost=0.00..2.00 rows=100 width=8) (actual time=0.004..0.011 rows=100 loops=1)',)
('                                                                    ->  Hash  (cost=1.62..1.62 rows=17 width=48) (actual time=0.011..0.011 rows=2 loops=1)',)
('                                                                          Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('                                                                          ->  Seq Scan on transactions  (cost=0.00..1.62 rows=17 width=48) (actual time=0.004..0.008 rows=2 loops=1)',)
('                                                                                Filter: (sender_account_id < 5)',)
('                                                                                Rows Removed by Filter: 48',)
('  ->  Hash  (cost=1.03..1.03 rows=3 width=36) (actual time=0.009..0.010 rows=3 loops=1)',)
('        Buckets: 1024  Batches: 1  Memory Usage: 9kB',)
('        ->  Seq Scan on transaction_status status  (cost=0.00..1.03 rows=3 width=36) (actual time=0.005..0.006 rows=3 loops=1)',)
('Planning Time: 2.173 ms',)
('Execution Time: 0.249 ms',)
```

5. Results:

Results for the before and after scenarios for query optimization.



```
Original query returned results: [(12, 'Ashley Vincent', 'South Dakota', 'Travis
 Griffith', 'Mail', 'Success', Decimal('384')), (5, 'Jeffrey Cannon', 'Arizona',
 'Dawn Bailey', 'Mail', 'Success', Decimal('433'))]

Original Execution Time: 0.0033788681030273438 seconds

Optimized query returned results: [(12, 'Ashley Vincent', 'South Dakota', 'Travi
s Griffith', 'Mail', 'Success', Decimal('384')), (5, 'Jeffrey Cannon', 'Arizona'
, 'Dawn Bailey', 'Mail', 'Success', Decimal('433'))]

Optimized Execution Time: 0.0014421939849853516 seconds

Savings: 57.32%
```

- The original and optimized queries were executed, and their execution plans were analyzed.
- The optimized query demonstrated a significant reduction in execution time compared to the original query.
- Percentage savings in execution time were calculated, showcasing the effectiveness of the optimization techniques.

6. Conclusion:

The project successfully implemented query optimization and distributed indexing techniques to enhance the performance of a distributed database system. The use of PostgreSQL and a dynamic query optimization approach demonstrated substantial improvements in query execution times. The script, snapshots, and this documentation serve as valuable resources for understanding the optimization process and its impact on distributed database performance.