Universidade de Lisboa
Faculdade de Ciências
Departamento de Informática

Phase 2 Report

Martim Emauz 58668 Miguel Martins 58661 Rúben Torres 62531

Bases de Dados Avançadas

Mestrado em Engenharia Informática

2024

1 Project Contribution

Martim Emauz (33%) - Indexes and benchmark comparison

Miguel Martins (33%) - Indexes and optimization in MySQL

Rúben Torres (33%) - Indexes and optimization in MongoDB

2 Main changes

The code was refactored to better define and show the operations done with the dataframes, and the dabase initialization.
The queries were refactored, in each of the databases, to better mirror each other's actions and essentially minimize any differences between them.
Indexes were added to the queries to optimize their performance.

3 Comparative Analysis

3.1 MySQL

- Before changes:

```
# mysql results

Query: simple_query_1, Average Execution Time: 0.0017889745235443115
    seconds
Query: top_performing_players, Average Execution Time:
    0.003777108907699585 seconds
```

- After changes:

```
# optimized mysql results

Query: simple_query_1, Average Execution Time: 0.0017039170265197754
    seconds
Query: top_performing_players, Average Execution Time:
    0.0038099508285522462 seconds
```

Indexing led to a minor performance improvement for simple_query_1, though the more complex top_performing_players query showed a small degradation. This implies that indexing in MySQL is helpful for simpler queries but may not always lead to improvements for more complex queries, especially if the indexing strategy isn't fully optimized for that query's data access pattern.
Both systems benefit from indexing, but the results are query-dependent. For simpler queries, indexing provides clear benefits, but for more complex queries, the optimization is less pronounced. Therefore, the choice of indexing strategy and its application to the specific query patterns in use will determine the extent of performance improvements in both MySQL and MongoDB, although in MongoDB we saw bigger performance improvements.

3.2 MongoDB

- Before changes:

```
# mongodb results

Query: mongodb_iberian_players_count, Average Execution Time:
    0.004534037828445435 seconds
Query: mongodb_top_performing_players, Average Execution Time:
    0.06852202701568604 seconds
```

- After changes:

```
# optimized mongodb results

Query: mongodb_iberian_players_count, Average Execution Time:
    0.0029003067016601563 seconds
Query: mongodb_top_performing_players, Average Execution Time:
    0.0628510115146637 seconds
```

Indexing had a clear positive impact on the performance of mongodb_iberian_players_count, significantly reducing the execution time. However, the performance improvement for mongodb_top_performing_players was smaller, highlighting that not all queries benefit equally from indexing, especially in MongoDB where query patterns and index structure can dramatically impact performance.

4 Optimization and Indexes

Indexes were added to optimize the query result speed on both platforms. The following indexes were added to the MySQL queries:

```
# Indexes MySQL

def create_indexes():

    create_indexes = [
    # Simple query
    "CREATE INDEX country_index ON player(country);",
    # Complex query
    # Indexes for JOIN
    "CREATE INDEX idx_player_stats_player_id ON player_stats(player_id);",
    "CREATE INDEX idx_player_id ON player(id);",
    # Indexes for WHERE
    "CREATE INDEX idx_player_matches_range ON player(matches);",
    # Composite indexes for GROUP BY and ORDER BY
    "CREATE INDEX idx_player_grouping ON player(id, name);",
    "CREATE INDEX idx_player_stats_contribution ON player_stats(player_id, goals,
        assists);"
    ]
```

```
    for query in create_indexes:
        mycursor.execute(query)
        mydb.commit()
```

The following indexes were added to the MongoDB queries:

```
# Indexes MongoDB

def create_indexes_mongodb():
    # Access collections using dictionary-style syntax
    player_collection = db["player"]
    player_stats_collection = db["player_stats"]

    # Simple query: Index on 'country' field in 'player' collection
    player_collection.create_index([("country", 1)], name="country_index")

    # Indexes for JOIN: Create an index on 'player_id' field in 'player_stats'
        collection
    player_stats_collection.create_index(
        [("player_id", 1)], name="idx_player_stats_player_id"
    )

    # Indexes for JOIN: Create an index on 'id' field in 'player' collection
    player_collection.create_index([("id", 1)], name="idx_player_id")

    # Indexes for WHERE: Create an index on 'matches' field in 'player' collection
    player_collection.create_index([("matches", 1)], name="idx_player_matches_range")

    # Composite Indexes for GROUP BY and ORDER BY: 'id' and 'name' fields in 'player'
        collection
    player_collection.create_index([("id", 1), ("name", 1)],
        name="idx_player_grouping")

    # Composite Indexes for GROUP BY and ORDER BY: 'player_id', 'goals', 'assists'
        fields in 'player_stats' collection
    player_stats_collection.create_index(
        [("player_id", 1), ("goals", 1), ("assists", 1)],
        name="idx_player_stats_contribution",
    )
```

5 Additional Features

No additional features were added.

6 Undone Features

No features were left undone.

7 Known Errors

No errors or bugs were found.