Indexing and Optimization

Advanced Databases/Databases Technologies

2024/2025 Project – 2nd Checkpoint

1. Group Details:

Group number: 8

Members: Afonso Baptista 58213; Miguel Borges 58187; Miguel Dinis 58198; Rafael Correia 58256

Every member of the group contributed equally to the development of this project.

- Afonso Baptista:
  - Data insertion and report development.
- Miguel Borges:
  - Rewrite queries and time comparison and report development.
- Miguel Dinis:
  - SQL – creation of optimized database and applying indexes to improve performance.
- Rafael Correia:
  - MongoDB – tried to create a better and optimized version and applied indexes for both MongoDB databases.
  - A few corrections in SQL queries.

2. Project Description

2.1. Changes Description

SQL

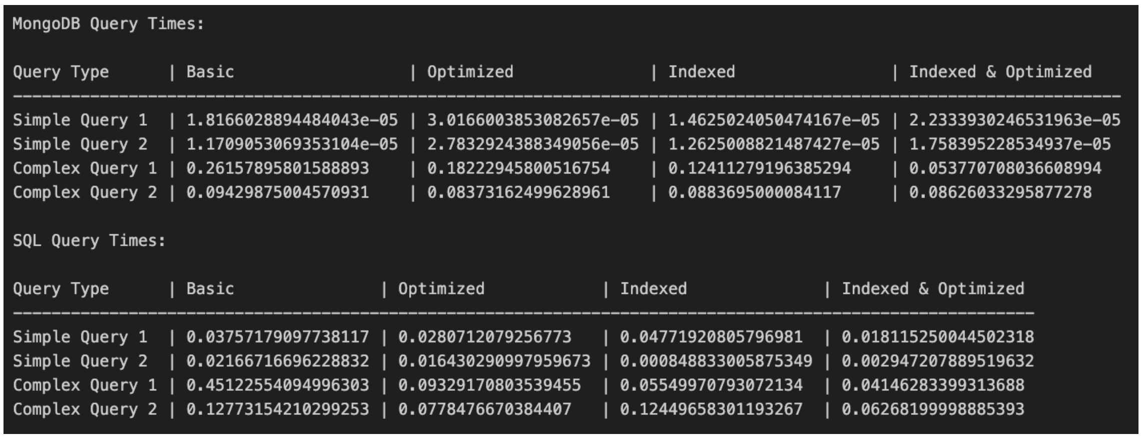
- Created a *locations* table, with unique(city,country).
- Created a *details* table, including *tournament*, *location_id* and *neutral* entries for each match.
- Edited *matches* table to have *date*, *home_team*, *away_team*, *home_score*, *away_score* and *details_id*.
- The goalscorers and shootouts tables are the same as before.

Our database design follows the Third Normal Form (3NF), ensuring the elimination of redundancies and maintaining data integrity. The locations table has a composite unique key (city, country) to ensure each location is uniquely identified. The details table links to the locations table via location_id, and the matches table is structured to reference the details table through detail_id, avoiding unnecessary data duplication. This normalization improves query performance and simplifies data management, while maintaining clear relationships between matches, their details, and associated locations.

<u>MongoDB</u>

We optimized our MongoDB schema by creating other two collections, one for goalscorers and another for shootouts that reference the matches collection. This way our queries are faster because in the queries we don't use the goalscorers and shootouts data, then instead of reading every time 11 columns we reduce for 9 columns, it's not a lot, but it's something.

2.2. Comparative Operation Analysis



```
MongoDB Query Times:

Query Type      | Basic                  | Optimized              | Indexed                | Indexed & Optimized
-------------------------------------------------------------------------------------------------------------------
Simple Query 1  | 1.8166028894484043e-05 | 3.0166003853082657e-05 | 1.4625024050474167e-05 | 2.2333930246531963e-05
Simple Query 2  | 1.1709053069353104e-05 | 2.7832924388349056e-05 | 1.2625008821487427e-05 | 1.758395228534937e-05
Complex Query 1 | 0.26157895801588893    | 0.18222945800516754    | 0.12411279196385294    | 0.053770708036608994
Complex Query 2 | 0.09429875004570931    | 0.08373162499628961    | 0.0883695000084117     | 0.08626033295877278

SQL Query Times:

Query Type      | Basic               | Optimized            | Indexed              | Indexed & Optimized
----------------------------------------------------------------------------------------------------------
Simple Query 1  | 0.03757179097738117 | 0.0280712079256773   | 0.04771920805796981  | 0.018115250044502318
Simple Query 2  | 0.02166716696228832 | 0.016430290997959673 | 0.000848833005875349 | 0.002947207889519632
Complex Query 1 | 0.45122554094996303 | 0.09329170803539455  | 0.05549970793072134  | 0.04146283399313688
Complex Query 2 | 0.12773154210299253 | 0.0778476670384407   | 0.12449658301193267  | 0.06268199998885393
```

We took some conclusions about the time each query takes with each optimized or non -optimized type, as we can see in this print.

The Basic type is the type from phase one, without any kind of optimization. As we can see some queries take a bit longer than others and the SQL queries take more time than de MongoDB ones. Now we are going to compare the values between the basic type and each of the others.

From the Optimized type, mentioned before in this report, we can conclude it was a bit worse for the simple queries in MongoDB, but it was better in the complex ones, even though the difference between values is not significant. For SQL times we can see that it was efficient as the values are all lower than the basic ones.

From the Indexed type, also mentioned before, we can say that, in MongoDB, the values are slightly better except for the simple query 2, but also with non-significant differences. For SQL queries we can see that the first query time takes a bit longer but the rest of them have lower times and the second simple query is the one with the biggest time difference.

Putting these two types together in Indexed & Optimized column, we can see that for MongoDB the simple queries take a bit longer, but the complex ones take shorter. For SQL, all the queries took shorter times than with the Basic type.

To conclude this section we can say that, in general, using both optimization types, it was very efficient, even though some might take a bit longer.

3. Discussion

<u>Indexes</u>

After testing a lot with all types of indexes, we found that the best indexes that optimized the read operations and didn't slow down the write operations were the indexes created for the teams playing against each other and the indexes created for the scores. These indexes were the most effective in reducing the query times for both the basic and optimized databases. The idea behind these indexes is to create an index for the columns that are most frequently used in the queries. In our case, the columns "home_team", "away_team", "home_score", and "away_score" are used in almost all queries, so creating indexes for these columns significantly improved the query performance. The indexes for the teams playing against each other helped in the queries that involved filtering matches based on the teams playing against each other, while the indexes for the scores helped in the queries that involved filtering matches based on the scores. These indexes reduced the query times by a significant margin, making the read operations faster and more efficient.

Optimization

The simple queries we couldn't optimize them more, because they are too simple.

In the first SQL complex query

"Change the neutral field to True for the matches that have more than 5 goals scored and that both teams have played with each other at least 100 times"

Instead of looping through the team pairs and updating the matches refered to those team pairs, making a lot of queries, we made a single query that updates all the matches that match the team pairs and the condition of the sum of the scores being greater than 5. This way we reduced the number of queries and the time to execute the query.

In the second SQL complex query

"Get the 100 matches with the most difference between the home_score and away_score and add 100 matches with the scores flipped, without shootouts and goalscores"

Instead of fetching the details for each match by a for loop, we made a single query that fetches all the details for the matches with the most difference in goals. This way we reduced the number of queries and the time to execute the query.

On mongo we only optimized the complex query 1, by the same logic as the SQL query, we made a single query that updates all the matches that match the team pairs and the condition of the sum of the scores being greater than 5. This way we reduced the number of queries and the time to execute the query.

Original dataset's link: https://www.kaggle.com/datasets/martj42/international-football-results-from-1872-to-2017