<<Theme / Title >>

<<Author1\_Name>>, << Author1\_StudentNr>>

<<Author2\_Name>>, << Author2\_ StudentNr >>

Barcelos, December 2022

Contents

[1. Introduction 2](#_Toc120893356)

[1.1. Project objectives 2](#_Toc120893357)

[1.2. List of platform features 2](#_Toc120893358)

[1.3. Platform workload 2](#_Toc120893359)

[2. Relational data model — PostgreSQL 2](#_Toc120893360)

[2.1. Data model design 2](#_Toc120893361)

[2.2. Data model tuning and testing 2](#_Toc120893362)

[3. Document-based data model — MongoDB 3](#_Toc120893363)

[3.1. Data model design 3](#_Toc120893364)

[3.2. Data model tuning and testing 3](#_Toc120893365)

[4. Graph-based data model — Neo4J 3](#_Toc120893366)

[4.1. Data model design 3](#_Toc120893367)

[4.2. Data model tuning and testing 3](#_Toc120893368)

[5. Discussion and conclusion 3](#_Toc120893369)

[References 4](#_Toc120893370)

[Contributions 4](#_Toc120893371)

[Annexes 5](#_Toc120893372)

[Annex 01 – <<Nome do Annex 01>> 5](#_Toc120893373)

[Annex 02 – <<Nome do Annex 02>> 5](#_Toc120893374)

# Introduction

Atualmente, em Portugal, a obesidade e o excesso de peso são problemas que se tem tornado mais frequentes com o decorrer dos anos. Estima-se que cerca de 50% da população portuguesa tem excesso de peso e desses 50%, 30 são consideradas obesas. Posto isto, o objetivo da nossa aplicação é promover o exercício físico fornecendo uma plataforma que agiliza o agendamento de uma atividade que a maioria dos portugueses gosta de praticar: o futebol.

## Project objectives

Dada a introdução feita no tópico anterior (1) pretendemos desenvolver uma aplicação que permita o agendamento de jogos de futebol com mais facilidade. Esta facilidade será garantida através das várias funcionalidades que a aplicação disponibilizará, das quais, a possibilidade de postar um conjunto de anúncios com o objetivo de anunciar a sua disponibilidade para jogar futebol, onde em cada anúncio, é apresentado o texto que o utilizador pretender, juntamente com o seu *username* e as informações temporais relativas à publicação.

Além disso, a aplicação também disponibilizará de um sistema de *friendslist* ondeos utilizadores poderão enviar pedidos de amizade uns aos outros e, com isto, podem então adicioná-los a uma equipa ou agendar um jogo de futebol entre equipas. Cada equipa terão um capitão que será o administrador da equipa e o mesmo poderá convidar novos utilizadores para fazerem parte da equipa, remover utilizadores e agendar jogos de futebol contra outras equipas. O agendamento de jogos será feito através de um menu onde o capitão da equipa terá a possibilidade de propor um jogo contra certa equipa e todos os jogadores de ambas as equipas serão notificadas da proposta.

## List of platform features

As funcionalidades da aplicação serão as seguintes:

* Autenticação – Login, registo e alteração de informações de utilizador.
* Agendamento e cancelamento de jogos de futebol.
* Atualização de informações sobre jogos de futebol agendados.
* Criação, edição e manutenção de equipas de futebol.
* Sistema de lista de amigos – Adição e remoção de amizades.
* Criação e gestão de torneios – Funcionalidade limitada a administradores da aplicação.
* Sistema de publicações – Criação de publicações, eliminação e edição.

## Platform workload

De modo a ter um conteúdo mais realista do conteúdo presente nas tabelas da base de dados, decidimos utilizar uma mistura do *software* para geração de dados disponibilizado pelo Navicat e, também, um conjunto de dados gerados por nós no Excel.

A utilização do Excel foi devida à falta de conteúdo online sobre informação de equipas de futebol, por isso descarregamos um CSV com milhares de nomes de cidade e concatenámos a *string* “FC” aos mesmos.

Posto isto, para cada entidade o tamanho dos dados é aproximadamente:

* 150 000 utilizadores
* 2 000 000 de amizades, sendo que cada utilizador tem em média 13 amizades
* 1 000 000 de jogos de futebol, onde 722 000 foram finalizados, 167 000 estão pendentes, 56000 foram recusados e 55 500 foram aceites
* 200 000 publicações
* 43 000 equipas
* 1 000 000 de logs

De modo a realizar uma comparação de performance entre os diferentes tipos de bases de dados para a realização deste trabalho prático, foram escolhidos os seguintes processos:

1. Listagem do número de golos por dia
2. Listagem de jogos em que um conjunto de utilizadores participou
3. Listagem dos jogos com o respetivo nome das equipas e o nome dos respetivos capitães
4. Listagem do número de golos que cada equipa marcou
5. Inserção de um conjunto de jogos terminados
6. Atualização da equipa vencedora dos jogos terminados

# Relational data model — SqlServer

<<Include here the tasks performed and its results. >>

## Data model design

A primeira estrutura de base de dados trabalhada é a que temos mais experiência a trabalhar tendo em conta que, atualmente, é a mais comum foi a base de dados relacional que optámos por realizar utilizando SqlServer.

Tal como foi explicado na introdução (**1.1 – Project Objectives**), o objetivo principal da nossa aplicação seria fornecer uma plataforma onde os seus utilizadores pudessem agendar jogos de futebol contra outros utilizadores. Posto isto, para a base de dados relacional, foi definido o seguinte modelo de base de dados (Figura 1).

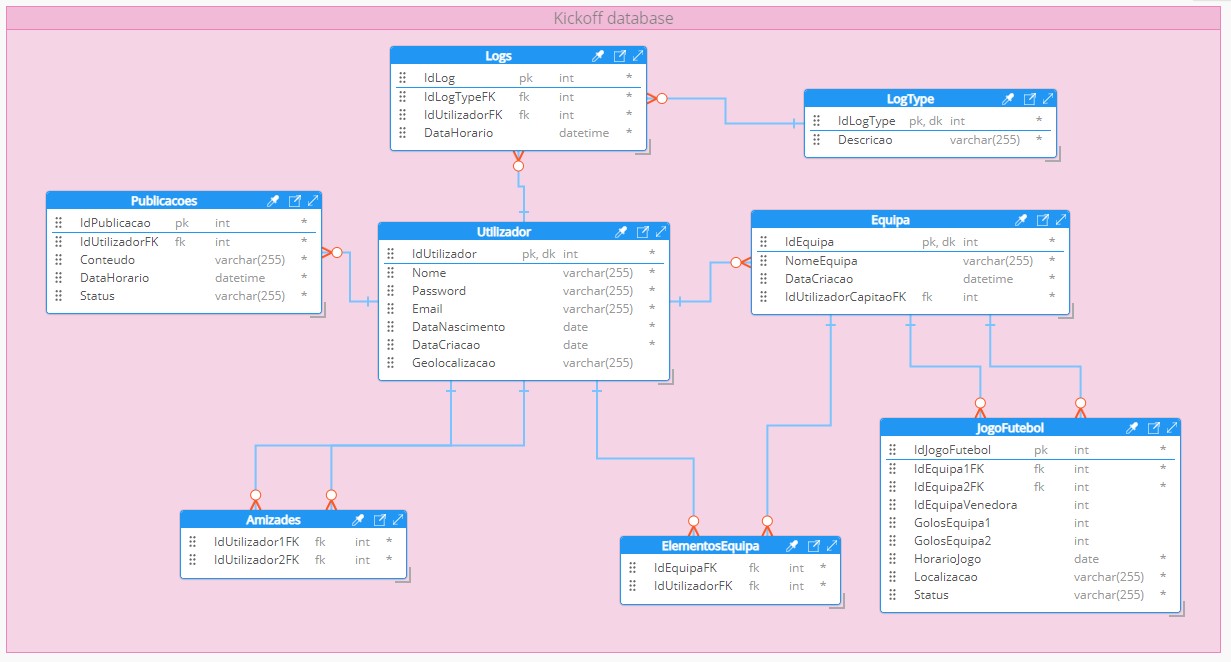


Figura 1 - Modelo de base de dados relacional

No modelo apresentado acima é possível identificar as seguintes entidades:

* Utilizadores
* Jogos de futebol
* Equipas
* Publicações
* Logs

Existem um conjunto de tabelas cujo seu propósito é auxiliar outras, por exemplo, não fazia sentido guardar que utilizadores pertenciam a uma respetiva equipa na própria tabela responsável pelo armazenamento de equipas, porque isto implicaria a que o grão da tabela fosse por equipa, onde seria o número de utilizadores que estariam associados a uma equipa. Posto isto, foi criada a tabela ElementosEquipa que possui o mesmo grão referido anteriormente, mas armazena apenas o Id da equipa e o utilizador respetivo. O mesmo acontece para a tabela LogType que possui a descrição de cada *log* em vez desta ser armazenada e repetida múltiplas vezes na tabela Log.

Relativamente às relações entre as diversas tabelas da base de dados, as mesmas são:

* Um utilizador pode realizar 0 ou muitas publicações.
* Um utilizador pode estar associado a 0 ou muitas equipas.
* Um utilizador pode ser capitão de 0 ou muitas equipas.
* Um utilizador pode ter 0 ou muitas amizades.
* Um utilizador pode estar associado a 0 o muitos *logs.*
* Uma equipa pode estar associada a 0 ou muitos jogos de futebol.
* Um tipo de *log* pode estar associado a 0 ou muitos *logs.*

Por fim, a população das tabelas para este modelo foi realizada utilizando a ferramenta *Data Generation* disponibilizada pelo Navicat e como foi referido no capítulo (**1.3 – Platform Workload**) foi utilizado também o Excel para geração da tabela de equipas, porque o navicat não conseguia garantir nomes de equipa únicos. Além disso também foram executadas algumas *queries* para garantir a autenticidade e integridade dos dados como, por exemplo, foi executada a seguinte *query* cujo objetivo foi atualizar o campo IdEquipaVencedora da tabela equipas.



Código 1 - *Query* de atualização dos vencedores

Além desta *query* também foram executadas outras cujo objetivo foi para remover registos onde uma certa equipa jogou contra ela mesma e um utilizador era amigo dele mesmo.

## Data model tuning and testing

Neste capítulo serão apresentados os resultados da lista de operações apresentada no capítulo **1.3 – Platform Workload.**

De modo a obter resultados conclusivos cada operação será realizada 3 vezes antes e 3 vezes depois das tentativas de melhoria realizadas**.**

Começámos pela operação responsável pela listagem do número de golos por dia.

1. **Listagem do número de golos por dia**

Para obter esta listagem foi realizada a seguinte *query*:



Código 2 - *Query* de listagem do número de golos/dia

O resultado da execução da *query* apresentada acima é o seguinte:

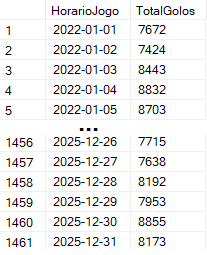


Figura 2 - Listagem do número de golos/dia

Tempos de execução:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1ª execução | 2ª execução | 3ª execução | Pior tempo | Melhor tempo | Tempo médio |
| S/ Índice | 0.359s | 0.349s | 0.507s | 0.507s | 0.349s | 0.405s |
| C/ Índice | 0.171s | 0.197s | 0.175s | 0.197s | 0.171s | 0.181s |

Tabela 1 - Tempos de execução listagem de golos/dia

Como é possível ver pela tabela apresentada anteriormente (Tabela 1) a criação de um índice melhorou o tempo de execução da *query* substancialmente. A diferença entre a execução mais rápida e a mais lenta foi de aproximadamente 55%.

O índice utilizado para esta *query* foi o seguinte:



Código 3 - Índice número de golos/dia

Para esta *query* decidimos criar um índice na coluna HorarioJogo, porque é a coluna responsável pela identificação do dia em que cada jogo foi realizado e, com isto, é feito um particionamento/ordenação na base de dados por dia, o que resulta no aumento da performance da *query*.

1. **Listagem de jogos em que um conjunto de utilizadores participou**

Para obter esta listagem foi realizada a seguinte *query*:



Código 4 - *Query* de listagem de jogos que certos utilizadores participaram

Nota: Para a realização desta *query* tivemos de escolher um conjunto de utilizadores, mas foram completamente aleatórios.

O resultado da execução da *query* apresentada acima é o seguinte:



Figura 3 - Listagem dos jogos do conjunto de utilizadores

Tempos de execução:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1ª execução | 2ª execução | 3ª execução | Pior tempo | Melhor tempo | Tempo médio |
| S/ Índice | 0.592s | 0.670s | 0.575s | 0.670s | 0.575s | 0.612s |
| C/ Índice | 0.401s | 0.447s | 0.414s | 0.401s | 0.447s | 0.421s |

Tabela 2 - Tempos de execução listagem de jogos de conjunto de utilizadores

Como é possível ver pela tabela anterior (Tabela 2), a criação de um índice melhorou a *performance* da *query.* Com a criação do mesmo, o tempo de execução médio reduziu em, aproximadamente, 31%.

O índice realizado para a melhoria do tempo de execução da *query* foi o seguinte:



Código 5 - Índice *query* listagem de jogos de certos utilizadores

Como nesta *query* o objetivo era listar os jogos finalizados de certos utilizadores, decidimos criar um índice na coluna status o que permitia ao motor de base de dados obter logo uma listagem dos jogos finalizados.

1. **Listagem dos jogos com o respetivo nome das equipas e dos respetivos capitães**

Para obter a listagem pretendida foi utilizada a *query*:



Código - *Query* de listagem dos jogos com nomes de equipa e capitão

O resultado da *query* anterior pode ser visualizado na figura apresentada abaixo:



Figura - Listagem dos jogos com nomes de equipa e capitão

Tempos de execução:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1ª execução | 2ª execução | 3ª execução | Pior tempo | Melhor tempo | Tempo médio |
| S/ Índice e alteração | 6.523s | 8.785s | 7.888s | 8.785s | 6.523s | 7.732s |
| C/Índice | 6.068s | 6.765s | 7.003s | 7.003s | 6.068s | 6.612s |
| C/ alteração | 5.478s | 6.190s | 6.589s | 6.589s | 5.478s | 6.086s |
| C/Índice e alteração | 5.558s | 6.689s | 6.321s | 6.689s | 5.558s | 6.189s |

Tabela - Tempos de execução da listagem dos jogos com nomes de equipa e capitão

Para a otimização da *performance* desta *query* também realizamos a criação de um índice na coluna Status e IdEquipa1FK da tabela JogoFutebol. O índice criado foi o seguinte:



Código - Índice para listagem dos jogos com nomes de equipa e capitão

Além disso também fizemos ligeiras alterações na *query*, mais propriamente nos *Joins* ficando a mesma com a seguinte estrutura:



Código - *Query* otimizada

Como é possível verificar através da tabela 3, os melhores resultados foram obtidos através da *query* com os *Joins* modificados. Além disso é possível verificar que, apesar de ter sido criado um índice, os melhores tempos obtidos foram para execuções sem índice. Em teoria, isto podia ser explicado pela má seleção de um plano de execução por parte do motor do SqlServer, mas não conseguimos chegar a nenhuma conclusão.

Por fim, pelos dados obtidos podemos afirmar que a alteração no tipo de *Joins* utilizados na *query* reduziu o tempo de execução em, aproximadamente, 28%.

1. **Listagem do número de golos que cada equipa marcou**

Para obter a listagem pretendida foi utilizada a *query*:



Código - *Query* de listagem de golos por equipa

Uma imagem com mesa

Descrição gerada automaticamente

Figura - Resultado da *query* de golos por equipa

Tempos de execução:

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
|  | 1ª execução | 2ª execução | 3ª execução | Pior tempo | Melhor tempo | Tempo médio |
| S/Índice | 0.801s | 0.733s | 0.865s | 0.800s | 0.733s | 0.800s |
| C/Índice | 0.450s | 0.488s | 0.490s | 0.490s | 0.450s | 0.476s |

Tabela - Tempos de execução para a listagem de golos por equipa

Como é visível na anterior (Tabela 4), a criação de um índice reduziu substancialmente o tempo de execução da respetiva *query*. O índice utilizado para a mesma foi criado na coluna Status da tabela dos jogos de futebol, porque na *query* queremos apenas os jogos que já terminaram (com *status Finished)*.



Código - Índice para a listagem dos golos por equipa

Com a criação do índice apresentado acima (Código 10) foi obtida uma redução de, aproximadamente, 41% no tempo de execução.

1. **Inserção de um conjunto de jogos terminados**

Para a realização e geração de um conjunto de dados respetivos a jogos terminados, foi utilizado o navicat.

Realisticamente, neste processo será contabilizado o tempo de geração e inserção de dados, onde o tempo de geração ocupará grande percentagem do tempo total do processo. Posto isto, será necessária gerar uma grande quantidade de dados para conseguir observar diferenças no tempo de execução.

Tempos de execução:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1ª execução (300000) | 2ª execução  (600000) | 3ª execução  (900000) |
| S/Índice | 1.53.42m | 3.39.45m | 6.04.16m |
| C/Índice | 1.59.22m | 3.56.90m | 6.33.64m |

Tabela - Tempos de execução para a inserção de dados

Na seguinte tabela é possível encontrar a percentagem de aumento do tempo de execução de inserção de dados com índices relativamente à inserção de dados sem índices.

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1ª execução | 2ª execução | 3ª execução |
| Aumento | 5% | 7% | 7% |

Tabela - Aumento percentual do tempo de execução

Por fim, apesar de não serem muito significativas no nosso volume de dados, em volumes de dados maiores os índices afetam drasticamente tempo de execução de inserções de forma negativa, pelo simples de facto de para além de inserirem novos dados, também terem de fazer atualizações nos índices.

1. **Atualização da equipa vencedora dos jogos terminados**

Para a realização da tarefa de atualização de um grande volume de dados, decidimos realizar uma *query* cujo objetivo é atualizar a coluna que identifica a equipa vencedora. Esta *query* compara o número de golos de cada equipa e se uma equipa tiver golos superior à outra, será armazenado o seu identificador na respetiva coluna, se não é inserido *NULL.*

A *query* utilizada foi a seguinte:



Código - *Query* para atualização de jogos

Tempos de execução:

|  |  |  |  |
| --- | --- | --- | --- |
|  | 1ª execução (1.3M) | 2ª execução  (1.6M) | 3ª execução  (1.9M) |
| S/Índice | 1.130s | 1.401s | 1.921s |
| C/Índice | 27.454s | 34.486s | 40.831s |

Tabela - Tempos de execução para atualização de dados

Como é possível observar na tabela 7, a utilização de índices afeta drasticamente os tempos de execução da *query*. Esta perda *performance* deve-se ao facto de que para cada linha, será necessário atualizar as informações dos índices para além de fazer a comparação e alteração dos dados de cada linha, tal como na inserção de dados.

A grande diferença é que o comando de *Update* pode ter que realizar um *Delete* seguido de um *Insert* daí os tempos de atualização serem afetados mais severamente pela utilização de índices.

# Document-based data model — MongoDB

<<Include here the tasks performed and its results. >>

## Data model design

<< Identify the entities and their relationships.. Implement each data model and populate the databases with meaningful data. The database should have enough size to draw conclusions about the performance and suitability of the data model.>>

## Data model tuning and testing

<<Test and evaluate some scenarios for each database system. Perform the execution analysis of operations previously described, and implement solutions to improve their execution, e. g., the creation of indexes.>>

# Graph-based data model — Neo4J

<<Include here the tasks performed and its results. >>

## Data model design

<< Identify the entities and their relationships.. Implement each data model and populate the databases with meaningful data. The database should have enough size to draw conclusions about the performance and suitability of the data model.>>

## Data model tuning and testing

<<Test and evaluate some scenarios for each database system. Perform the execution analysis of operations previously described, and implement solutions to improve their execution, e. g., the creation of indexes.>>

# Discussion and conclusion

<< Taking into account the results and conclusions, please provide a grounded recommendation for the database model that should be adopted. >>

# References

<< In this section, should be presented in APA format the list of bibliographic sources that were relevant the execution of the work. All the references must be cited on the report.>>

# Contributions

<<Here must be reported the **individual contributions** of each student. For each project task, identify the students that have actively participated.>>

# Annexes

<< The annexes should be used for the inclusion of additional information needed for a better understanding of the report or to complement it. The annexes created should be numbered, start at the top of a new page and be cited in the body text of the report.>>

## Annex 01 – <<Nome do Annex 01>>

## Annex 02 – <<Nome do Annex 02>>

Etc.