ESCOLA SUPERIOR DE TECNOLOGIA IPCA

ADVANCED DATABASES

Master on Computer Engineering, 2022-23

Project 1

Carlos Martins, 18836

João Azevedo, 18845

Barcelos, December 2022

Conteúdo

Índice de Tabelas

Índice de Códigos

Índice de Figuras

1. 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.

1.1. 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* onde os 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.

1.2. 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.

1.3. 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

2. Relational data model — SqlServer

2.1. 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).

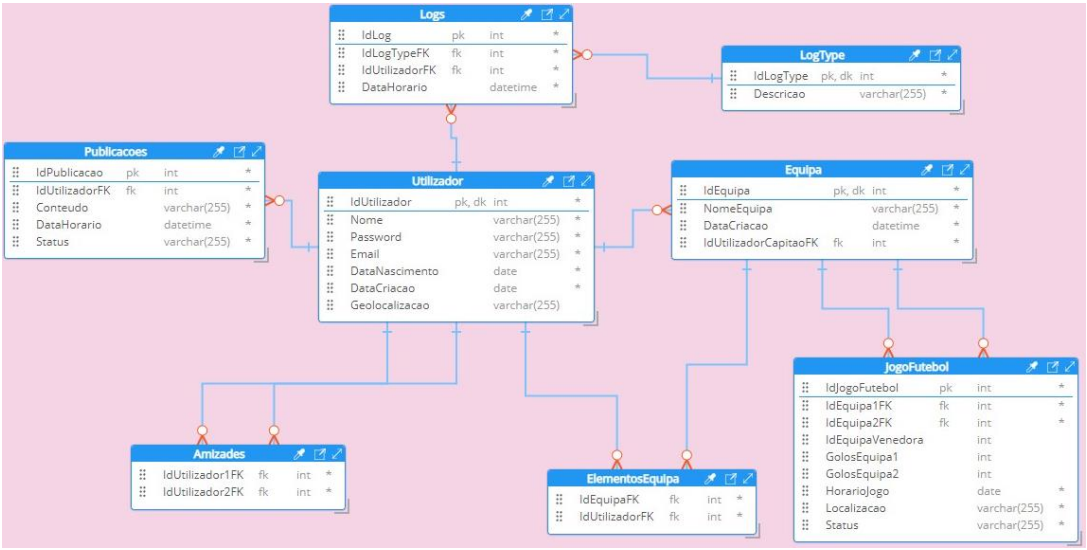


Figura 1 - Modelo de base de dados relacional

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 $n * equipa = linhas$ por equipa, onde n 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.

```
UPDATE JogoFutebol
SET IDEquipaVencedora =
    CASE
        WHEN GolosEquipa1 > GolosEquipa2 THEN IdEquipa1FK
        WHEN GolosEquipa2 > GolosEquipa1 THEN IdEquipa2FK
        ELSE NULL
    END
```

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.

2.2. 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*:

```
SELECT HorarioJogo, geq1+geq2 AS TotalGolos FROM (
SELECT HorarioJogo, SUM(GolosEquipa1) geq1, SUM(GolosEquipa2) geq2 FROM
JogoFutebol
GROUP BY HorarioJogo) jf
ORDER BY HorarioJogo ASC
```

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

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

	HorarioJogo	TotalGolos
1	2022-01-01	7672
2	2022-01-02	7424
3	2022-01-03	8443
4	2022-01-04	8832
5	2022-01-05	8703
...		
1456	2025-12-26	7715
1457	2025-12-27	7638
1458	2025-12-28	8192
1459	2025-12-29	7953
1460	2025-12-30	8855
1461	2025-12-31	8173

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:

```
CREATE INDEX NumeroGolosIndex ON JogoFutebol(HorarioJogo) INCLUDE (IdEquipa1FK, IdEquipa2FK, GolosEquipa1, GolosEquipa2)
```

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*.

2) Listagem de jogos em que um conjunto de utilizadores participou

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

```
SELECT jogos.IdJogoFutebol, eq1.NomeEquipa Equipa1, jogos.GolosEquipa1, jogos.GolosEquipa2,
eq2.NomeEquipa Equipa2,
CASE
        WHEN eqv.NomeEquipa IS NULL THEN 'Empate'
        ELSE eqv.NomeEquipa
END EquipaVencedora, jogos.HorarioJogo, jogos.Localizacao FROM (
SELECT * FROM JogoFutebol jf
WHERE jf.IdEquipa1FK IN ( SELECT eq.IdEquipa IdEquipa FROM Equipa eq, ElementosEquipa el
WHERE eq.IdEquipa = el.IdEquipaFK AND el.IdUtilizadorFK IN('10230', '98372', '52342',
'11837','22347','82468','55555','67822',
'12698','9878','52324','87632','1','2','3','4','5','6','7')) AND jf.Status = 'Finished'
OR jf.IdEquipa2FK IN ( SELECT eq.IdEquipa IdEquipa FROM Equipa eq, ElementosEquipa el
WHERE eq.IdEquipa = el.IdEquipaFK AND el.IdUtilizadorFK IN('10230', '98372', '52342',
'11837','22347','82468','55555','67822',
'12698','9878','52324','87632','1','2','3','4','5','6','7')) AND jf.Status = 'Finished') jogos
LEFT JOIN Equipa eq1 ON eq1.IdEquipa = jogos.IdEquipa1FK
LEFT JOIN Equipa eq2 ON eq2.IdEquipa = jogos.IdEquipa2FK
LEFT JOIN Equipa eqv ON eqv.IdEquipa = jogos.IDEquipaVencedora
```

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:

	IdJogoFutebol	Equipa1	GolosEquipa1	GolosEquipa2	Equipa2	EquipaVencedora	HorarioJogo	Localizacao
1	713	Nalpur FC	7	0	Adelanto FC	Nalpur FC	2022-11-09	5-kai, 5-19-19 Shinei 4 Jo, Kiyota Ward, Sapporo, Ja...
2	4889	Monreale FC	3	1	Morris FC	Monreale FC	2023-09-17	32F, 1-7-10 Omido, Higashiosaka, Osaka, Japan
3	6798	Torrinha FC	1	8	Fuyang FC	Fuyang FC	2022-07-28	Block 45, 378 Silver St, Newnham, Cambridge, CB3 ...
4	8031	Capão do Leão FC	6	6	Gloucester FC	Empate	2022-01-09	19F, 172 2nd Zhongshan Road, Yuexiu District, Guan...
5	8932	Ribadavia FC	2	3	Tazert FC	Tazert FC	2024-02-01	20F, 622 Lefeng 6th Rd, Zhongshan, China
6	10730	Aschau im Chiemgau FC	6	2	Ban Tha Phra FC	Aschau im Chiemgau FC	2025-10-19	573 Lark Street Apartment 41, Albany, NY 12210, Unit...
930	995193	Capão do Leão FC	11	16	Dubno FC	Dubno FC	2023-08-21	Room 13, CR Building, 161 Ganlan Rd, Pudong, Sha...
931	995598	Boa Esperança do Sul ...	18	2	Camerano FC	Boa Esperança do Sul ...	2023-11-09	214 West Market Street Apt 49, Akron, OH 44333, Uni...
932	996082	Canzo FC	14	20	Gonghe FC	Gonghe FC	2022-10-06	Flat 23, 214 Hanover St, Liverpool, L1 4AF, United Ki...
933	996754	Mandapeta FC	5	9	Albacete FC	Albacete FC	2025-12-05	No.25 building, 548 Daxin S Rd, Daxin Shangquan, ...
934	998364	Siemiatycze FC	10	15	Kampong Chh...	Kampong Chhnang FC	2023-12-14	Rm. 37, 2-1-2 Kaminopporo 1 Jo, Atsubetsu Ward, S...
935	999108	Gonghe FC	8	18	São Miguel do ...	São Miguel do Anta FC	2022-12-11	952 Ridgewood Road 3rd Floor, Akron, OH 44321, U...

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:

```
CREATE INDEX JogosDeUsersIndex ON JogoFutebol(Status) INCLUDE (IdEquipa1FK,
IdEquipa2FK, GolosEquipa1, GolosEquipa2, IdEquipaVencedora,HorarioJogo,Localizacao)
```

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.

3) Listagem dos jogos com o respetivo nome das equipas e dos respetivos capitães

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

```
SELECT jf.IdJogoFutebol, eq1.NomeEquipa Equipa1, jf.GolosEquipa1,
eq2.NomeEquipa Equipa2, jf.GolosEquipa2, ut1.Nome CapitaoEquipa1, ut2.Nome
CapitaoEquipa2,
CASE
	WHEN eqv.NomeEquipa IS NULL THEN 'Empate'
	ELSE eqv.NomeEquipa
END EquipaVencedora, jf.HorarioJogo, jf.Localizacao
FROM JogoFutebol jf
RIGHT JOIN Equipa eq1 ON eq1.IdEquipa = jf.IdEquipa1FK
RIGHT JOIN Equipa eq2 ON eq2.IdEquipa = jf.IdEquipa2FK
LEFT JOIN Utilizador ut1 ON ut1.IdUtilizador = eq1.IdUtilizadorCapitaoFK
LEFT JOIN Utilizador ut2 ON ut2.IdUtilizador = eq2.IdUtilizadorCapitaoFK
LEFT JOIN Equipa eqv ON eqv.IdEquipa = jf.IDEquipaVencedora
WHERE jf.Status = 'Finished'
```

Código 6 - *Query* de listagem dos jogos com nomes de equipa e capitão

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

	IdJogoFutebol	Equipa1	GolosEquipa1	Equipa2	GolosEquipa2	CapitaoEquipa1	CapitaoEquipa2	EquipaVencedora	HorarioJogo	Localizacao
1	3	Leixlip FC	7	Salay FC	3	Jacqueline Guzman	Yue Tsz Ching	Leixlip FC	2022-10-14	154 Central Avenue A
2	12	Botolan FC	8	San Andrés del Rabanedo FC	7	Lau Chiu Wai	Angela Ferguson	Botolan FC	2022-04-22	3-kai, 2-1-12 Kaminc
3	13	San Juan FC	5	Kraslice FC	7	Yu Zhennan	Mak Sum Wing	Kraslice FC	2025-04-22	Flat 40, 343 Silver St
4	16	Itaqui FC	8	Anapoima FC	5	Dong Anqi	Duan Rui	Itaqui FC	2025-02-03	Room 9, 27 Middle H
5	19	Botelhos FC	2	Monóvar FC	5	Arthur Ellis	Ann Payne	Monóvar FC	2023-01-02	48 Central Avenue S
6	21	Hudsonville FC	5	Vitomirică FC	9	Masuda Eita	Jia Shihan	Vitomirică FC	2024-05-17	No.27 building, 871 S
7	25	Sumter FC	7	Brakpan FC	0	Zhang Lan	Ueda Ikki	Sumter FC	2022-08-02	Flat 26, 367 Wyngate
				...						
722210	999994	Tatoufet FC	15	Appenzell FC	12	Amber Johnson	Takada Hikaru	Tatoufet FC	2022-08-15	Room 31, 287 Kengu
722211	999995	Wamba FC	11	Culasi FC	11	Choi Wing Fat	Johnny Tucker	Empate	2022-07-04	Block 4, 860 New Str
722212	999996	Acque Dolci FC	13	Sant Sadurní d'Anoia FC	18	Tanaka Tsubasa	Shao Lu	Sant Sadurní d'Anoia FC	2025-04-03	178 Bank Street Apa
722213	999997	Sorsogon FC	12	Ketovo FC	10	Linda Lewis	Noguchi Shino	Sorsogon FC	2022-02-23	29F, 1-7-13 Omido, H
722214	999998	Bangar FC	11	Peta? Tiqwa FC	16	Choi Sau Man	Leslie Perez	Peta? Tiqwa FC	2024-10-13	703 Nostrand Ave Ap
722215	999999	Déchy FC	9	Körfez FC	4	Nakamura Kasumi	Kwok Wai Lam	Déchy FC	2022-01-13	3-kai, 4-9-12 Kamihi
722216	1000000	Eugenópolis FC	4	Doney Park FC	15	Han Xiaoming	Judy Hawkins	Doney Park FC	2023-02-06	397 Fifth Avenue Apt

Figura 4 - 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 3 - 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:

```
CREATE INDEX JogoFutebolNomesEquipasCapitaesIndex ON JogoFutebol(Status) INCLUDE (
IdEquipa1FK, IdEquipa2FK, IdEquipaVencedora, GolosEquipa1, GolosEquipa2,
HorarioJogo, Localizacao)
```

Código 7 - 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:

```
SELECT jf.IdJogoFutebol, eq1.NomeEquipa Equipa1, jf.GolosEquipa1,
eq2.NomeEquipa Equipa2, jf.GolosEquipa2, ut1.Nome CapitaoEquipa1,
ut2.Nome CapitaoEquipa2,
CASE
      WHEN eqv.NomeEquipa IS NULL THEN 'Empate'
      ELSE eqv.NomeEquipa
END EquipaVencedora, jf.HorarioJogo, jf.Localizacao
FROM JogoFutebol jf
INNER JOIN Equipa eq1 ON eq1.IdEquipa = jf.IdEquipa1FK
INNER JOIN Equipa eq2 ON eq2.IdEquipa = jf.IdEquipa2FK
INNER JOIN Utilizador ut1 ON ut1.IdUtilizador = eq1.IdUtilizadorCapitaoFK
INNER JOIN Utilizador ut2 ON ut2.IdUtilizador = eq2.IdUtilizadorCapitaoFK
LEFT JOIN Equipa eqv ON eqv.IdEquipa = jf.IDEquipaVencedora
WHERE jf.Status = 'Finished'
```

Código 8 - *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%.

4) Listagem do número de golos que cada equipa marcou

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

```
SELECT eq.NomeEquipa, GolosEquipa1 + GolosEquipa2 AS GolosTotais FROM
(SELECT IdEquipa, NomeEquipa FROM Equipa) eq,
(SELECT IdEquipa1FK, SUM(GolosEquipa1) GolosEquipa1 FROM JogoFutebol
WHERE Status = 'Finished'
GROUP BY IdEquipa1FK) jf1,
(SELECT IdEquipa2FK, SUM(GolosEquipa2) GolosEquipa2 FROM JogoFutebol
WHERE Status = 'Finished'
GROUP BY IdEquipa2FK) jf2
WHERE jf1.IdEquipa1FK = jf2.IdEquipa2FK
AND eq.IdEquipa = jf1.IdEquipa1FK
```

Código 9 - *Query* de listagem de golos por equipa

	NomeEquipa	GolosTotais
1	Tokyo FC	104
2	Jakarta FC	303
3	Delhi FC	285
4	Manila FC	198
5	São Paulo FC	277
6	Seoul FC	285
...		
42900	Agapa FC	299
42901	Tukchi FC	416
42902	Numto FC	235
42903	Nord FC	348
42904	Timmiarmiut FC	283
42905	Nordvik FC	158

Figura 5 - 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 4 - 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)*.

```
CREATE INDEX JogoFutebolGolosPorEquipaIndex ON JogoFutebol(Status)
INCLUDE (IdEquipa1FK, IdEquipa2FK, GolosEquipa1, GolosEquipa2)
```

Código 10 - 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.

5) 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 5 - 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 6 - 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.

6) 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:

```
UPDATE JogoFutebol
SET IDEquipaVencedora = CASE
                        WHEN GolosEquipa1 > GolosEquipa2 THEN IdEquipa1FK
                        WHEN GolosEquipa2 > GolosEquipa1 THEN IdEquipa2FK
                        ELSE NULL
                        END
```

Código 11 - *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 7 - 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 de realizar um *Delete* seguido de um *Insert* daí os tempos de atualização serem afetados mais severamente pela utilização de índices.

3. Document-based data model — MongoDB

3.1. Data model design

Para o desenvolvimento de uma base de dados baseada em documentos, decidimos utilizar MongoDB.

Este tipo de base de dados possui um conjunto de coleções (tabelas) onde cada uma possui um ou muitos documentos (linhas).

A grande vantagem do MongoDB é o facto de ser uma base de dados não relacional o que promove a escalabilidade e uma melhoria em certos tipos de *query.* As *queries* com maior *performance* tratam-se de *queries* que não fazem agregações de dados de outras coleções, porque durante o desenvolvimento deste trabalho notámos que isso era um problema neste tipo de base de dados.

Posto isto, tivemos de alterar o nosso modelo de base de dados de modo a servir as capacidades do MongoDB.

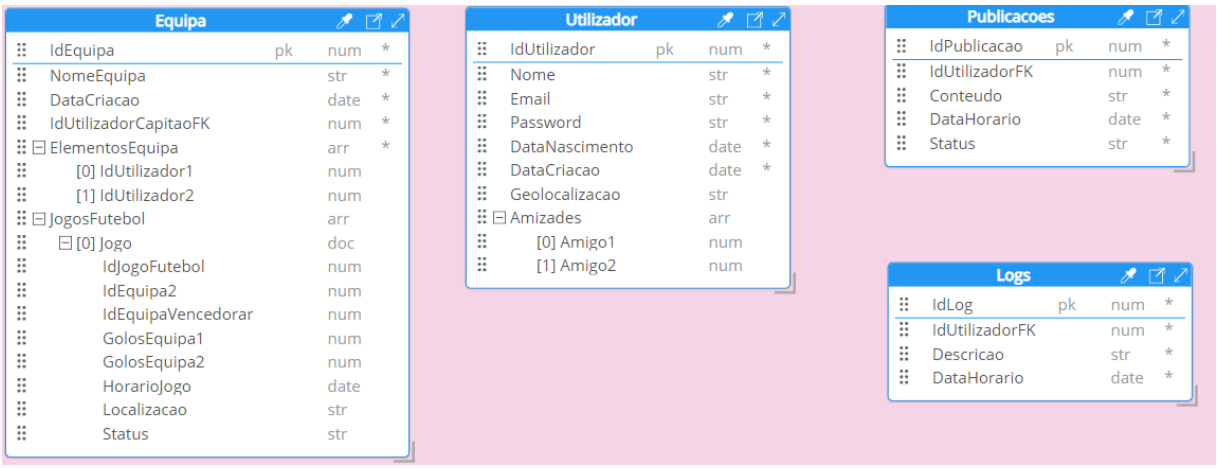


Figura 6- Modelo de base de dados baseada em documentos

Como é possível verificar, a figura apresentada anteriormente (Figura 6), possui um conjunto de diferenças relativamente ao modelo relacional. A principal diferença para o modelo anterior foi a junção de várias tabelas numa só como, por exemplo, a coleção equipa, agora, armazena todos os utilizadores da equipa, juntamente com todos os jogos de futebol associados à mesma. Esta alteração foi realizada de modo a reduzir o número de *queries* de agregação entre diferentes coleções.

Neste modelo, como foi dito anteriormente, o número de entidades foi reduzido porque algumas foram juntas numa só.

Posto isto, as entidades presentes neste tipo de base de dados são as seguintes:

- Utilizadores – onde informações do utilizador e respetivas amizades estão presentes
- Equipas – onde informações de equipas, jogos e elementos da equipa estão presentes
- Publicações
- *Logs*

Relativamente à preparação dos dados, tivemos algumas dificuldades. Inicialmente, exportámos todos os dados do modelo relacional em formato CSV e carregamo-los para coleções respetivas aos mesmos. Depois disso realizámos algumas *queries*, mas os resultados não eram os pretendidos. Um exemplo de um resultado obtido através de *queries* de agregação para chegar à estrutura final é o seguinte:


```
    "IdUtilizador": NumberInt("459"),
    "Nome": "Kwong Ka Keung",
    "Password": "wq3y6Uh9MV",
    "Email": "kakeung1026@gmail.com",
    "DataNascimento": ISODate("1999-09-22T00:00:00.000Z"),
    "DataCriacao": ISODate("2025-02-21T00:00:00.000Z"),
    "Geolocalizacao": "-90, \t\t7.0",
    "Amizades": [
        {
            "amigo": NumberInt("68871")
        }
    ]
}
```


Figura 7 - Exemplo de resultado *query* de agregação

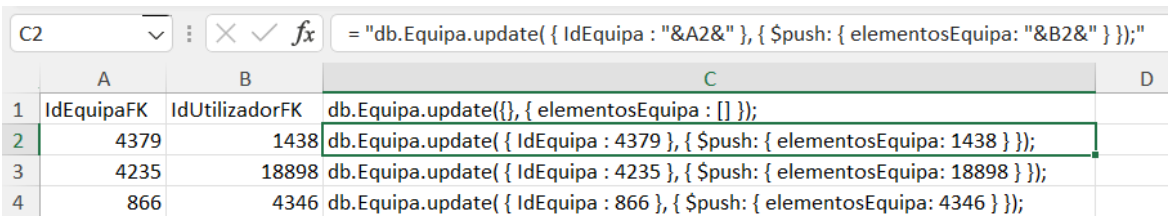
A *query* responsável pela geração resultado anterior pode ser encontrado no código 12.

```
db.Utilizador.aggregate([
    {
        $lookup: {
            from: "Amizades",
            localField: "IdUtilizador",
            foreignField: "IdUtilizador1FK",
            as: "amizades"
        }
    },
    {
        $unwind: "$amizades"
    },
    {
        $group: {
            _id: {
                IdUtilizador: "$IdUtilizador",
                Nome: "$Nome",
                Password: "$Password",
                Email: "$Email",
                DataNascimento: "$DataNascimento",
                DataCriacao: "$DataCriacao",
                Geolocalizacao: "$Geolocalizacao"
            },
            amizades: {
                $push: {
                    amigo: "$amizades.IdUtilizador2FK"
                }
            }
        }
    }
])
```

Código 12 - *Query* de agregação

Como é possível verificar pela figura 7, o *array* amizades possuía um objeto referente a cada amizade que o utilizador tinha em vez de guardar apenas um número inteiro.

Posto isto, de modo a ultrapassar as dificuldades geramos *queries* de acordo com os dados presentes no CSV utilizando ferramentas do Excel.



	A	B	C	D
1	IdEquipaFK	IdUtilizadorFK	db.Equipa.update({}, { elementosEquipa : [] });	
2	4379	1438	db.Equipa.update({ IdEquipa : 4379 }, { $push: { elementosEquipa: 1438 } });	
3	4235	18898	db.Equipa.update({ IdEquipa : 4235 }, { $push: { elementosEquipa: 18898 } });	
4	866	4346	db.Equipa.update({ IdEquipa : 866 }, { $push: { elementosEquipa: 4346 } });	

Figura 8 - Geração de *queries* no Excel

3.2. Data model tuning and testing

Inicialmente, tentamos realizar o carregamento e preparação das coleções utilizando o mesmo *dataset* que utilizamos no modelo relacional, mas as *queries* de agregação que realizámos para construir as coleções estavam a demorar imenso tempo a finalizar. Por isso, não conseguimos utilizar a mesma quantidade de dados utilizada no modelo relacional.

Numa primeira fase com o auxílio do Studio 3T foi feita uma migração inicial de 1000 rows por entidade tendo como ideia a realização de uma fase de testes das querys de mongo. Após a fase de teste foi realizado um aumento do número de dados.

Posto isto, o número total de documentos em cada *collection* é o seguinte:

- Equipas – Aproximadamente 6300 equipas, com uma média de 5 elementos por equipa e 16 mil jogos de futebol.
- Utilizadores – Aproximadamente 20 mil utilizadores com uma média de 2 amizades.
- Publicações – Aproximadamente 10 mil publicações.
- *Logs* – Aproximadamente 40 mil *logs*.

Além disso, também tivemos dificuldade em replicar a *query* do número de golos por dia, por isso substituímo-la por uma *query* que lista o utilizador vencedor de cada jogo que é apresentada de seguida.

1) Listagem de vencedores dos jogos terminados

A *query* utilizada para listagem destes vencedores no MongoDB foi a seguinte:

```
db.Equipa.aggregate([
  { $match : { "jogosFutebol.Status" : "Finished" } },
 {
         $project:
         {
             IdEquipa: "$IdEquipa",
             NomeEquipa: "$NomeEquipa",
             EquipaVencedora:
             {
                 $switch: {
                     branches: [
                         { case: { $eq: [ "$jogosFutebol.GolosEquipa1",
"$jogosFutebol.GolosEquipa2" ] }, then: 0 },
                         { case: { $lt: [ "$jogosFutebol.GolosEquipa1",
"$jogosFutebol.GolosEquipa2" ] }, then: "$jogosFutebol.IdEquipa2" },
                         { case: { $gt: [ "$jogosFutebol.GolosEquipa1",
"$jogosFutebol.GolosEquipa2" ] }, then: "$IdEquipa" }
                     ],
                      default: 0
                 }
             }
         }
     }
 ]);
```

Código 13 - *Query* para listagem dos vencedores dos jogos MongoDB

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


```
{
    "_id": ObjectId("63b196c8035013959908 54d5"),
    "IdEquipa": NumberInt("2"),
    "NomeEquipa": "Jakarta FC",
    "EquipaVencedora": [
        1087,
        4111,
        3025
    ]
}
```


Figura 9 - Resultado da *query* para listagem de vencedores MongoDB

A *query* apresentada possui alguns problemas que não conseguimos identificar como, por exemplo, inicialmente começamos por definir que queremos apenas os jogos com a coluna Status com o valor "*Finished*" e não são esses os valores que estão a ser utilizados pela restante parte da *query*.

Além disso, os valores obtidos através das comparações entre os golos de cada equipa também não estão corretos. Testámos escrever as expressões por extenso em vez de utilizar as funções de equivalência do MongoDB mas também não obtivemos os resultados corretos.

Tempos de execução:

	1ª execução	2ª execução	3ª execução	Pior tempo	Melhor tempo	Tempo médio
S/Índice	0.080s	0.092s	0.079s	0.092s	0.079s	0.084s
C/Índice	0.100s	0.088s	0.107s	0.107s	0.088s	0.098s

Tabela 8 - Tempos de execução *query* de listagem de vencedores MongoDB

Apesar da *query* ser relativamente rápida, de modo a otimizá-la criamos um índice no campo Status, porque queríamos apenas utilizar os valores cujo status fosse "Finished".

Como o campo *status* é um valor encapsulado, ou seja, é acedido através de: Coleção Equipa -> Campo JogosFutebol -> Campo Status, criámos um índice com a *flag "*{background:1 }, mas como é possível ver através da tabela anterior, a criação do índice prejudicou ligeiramente a performance da *query.*

2) Listagem de jogos em que um conjunto de utilizadores participou

A *query* desenvolvida para esta operação foi a seguinte:

```
db.Equipa.find(
   { elementosEquipa: { $elemMatch: { $in: [1411, 1212, 1, 223, 17231, 8231, 6353, 7263, 9212, 772, 213, 667]} } }
)
```

Código 14 - *Query* para de listagem de jogos de utilizadores MongoDB

O resultado para a *query* apresentada anteriormente é o seguinte:


```
{
    "_id": ObjectId("63b196c803501395990854da"),
    "IdEquipa": NumberInt("7"),
    "NomeEquipa": "Mumbai FC",
    "DataCriacao": ISODate("2023-04-23T00:00:00.000Z"),
    "IdUtilizadorCapitaoFK": NumberInt("14514"),
    "jogosFutebol": [
        {
            "IdJogoFutebol": 14553,
            "IdEquipa2": 4125,
            "IdEquipaVencedora": null,
            "GolosEquipa1": null,
            "GolosEquipa2": null,
            "HorarioJogo": "42787",
            "Localizacao": "No.46 building, No.568, Dongsan Road, Erxianqiao, Chenghua District, Chengdu, China",
            "Status": "Refused"
        },
        {
            "IdJogoFutebol": 8071,
            "IdEquipa2": 2670,
            "IdEquipaVencedora": 7,
            "GolosEquipa1": 6,
```


Figura 10 - Resultado da *query* de listagem de jogos de utilizadores MongoDB

Como é possível ver no resultado da *query* todos os jogos onde o utilizador marcou presença ou está associado aparecem dentro do *array* jogosFutebol.

Tempos de execução:

	1ª execução	2ª execução	3ª execução	Pior tempo	Melhor tempo	Tempo médio
S/Índice	0.019s	0.018s	0.020s	0.020s	0.018s	0.019s
C/Índice	0.022s	0.19s	0.018s	0.022s	0.018s	0.020s

Tabela 9 - Tempos de execução listagem de jogos de utilizadores MongoDB

De modo a prejudicar o tempo de execução da query decidimos acrescentar mais utilizadores, mas reparámos que isso não trazia problema nenhum nem impactava a velocidade do mongoDB consideravelmente.

Posto isto, tentámos, então, aumentar a *performance* da *query* criando um índice, mas os resultados não foram animadores, porque não obtivemos diferença praticamente nenhuma no tempo de execução como é possível verificar através da tabela 9.

Através do MongoDB Compass verificamos que a razão pela qual os resultados foram idênticos era simplesmente pelo facto do índice não estar a ser utilizado.

3) Listagem dos jogos com o respetivo nome das equipas e o nome dos respetivos capitães

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

```
db.Equipa.aggregate([
    {
        $lookup: {
            from: "Utilizador",
            localField: "IdUtilizadorCapitaoFK",
            foreignField: "IdUtilizador",
            as: "Utilizador"
            }
     },
     {
          $unwind: "$Utilizador"
     },
        {
            $project: {
            IdEquipa: "$IdEquipa",
            NomeEquipa: "$NomeEquipa",
            IdUtilizadorCapitao:"$Utilizador.Nome",
            JogoFutebol: "$jogosFutebol"
         }
      }
])
```

Código 15 - *Query* para listagem dos jogos com nomes de equipa e capitao MongoDB

O resultado da execução da *query* anterior é o seguinte:


```
{
    "_id": ObjectId("63b196c803501395990854d5"),
    "IdEquipa": NumberInt("2"),
    "NomeEquipa": "Jakarta FC",
    "IdUtilizadorCapitao": "Kono Kenta",
    "JogoFutebol": [
        {
            "IdJogoFutebol": 14007,
            "IdEquipa2": 1087,
            "IdEquipaVencedora": null,
            "GolosEquipa1": null,
            "GolosEquipa2": null,
            "HorarioJogo": "36773",
            "Localizacao": "Unit 31, Oxford Eco Centre, 914 Whitehouse Lane, Huntingdon Rd, Cambridge, CB3 0LX, United Kingdom",
            "Status": "Pending"
        },
        {
            "IdJogoFutebol": 5246,
            "IdEquipa2": 4111,
            "IdEquipaVencedora": 4111,
            "GolosEquipa1": 0,
            "GolosEquipa2": 9,
            "HorarioJogo": "38884",
            "Localizacao": "Unit 14, Oxford Eco Centre, 531 Regent Street, London, W1B 2LX, United Kingdom",
            "Status": "Finished"
```


Figura 11 - Listagem dos jogos com nomes de equipa e capitão MongoDB

Como é possível verificar pela figura 9, a figura apresenta apenas o nome da equipa 1 e o seu respetivo capitão.

Tendo em conta que esta *query* utiliza o método *aggregate* e *lookup* do MongoDB era esperado que a mesma demorasse algum tempo, mesmo estando a trabalhar com um *dataset* reduzido.

Como foi referido anteriormente, as operações que, no fundo, fazem *Joins* tal como a de agregação são bastante custosas para o MongoDB e, neste caso, decidimos criar um índice no campo no IdUtilizador.

Relativamente aos tempos de execução:

	1ª execução	2ª execução	3ª execução	Pior tempo	Melhor tempo	Tempo médio
S/Índice	54.099s	51.729s	48.626s	54.099s	48.626s	51.485s
C/Índice	0.476s	0.476s	0.495s	0.495s	0.476s	0.482s

Tabela 10 - Tempos de execução da listagem dos jogos com Equipa e capitão MongoDB

Pela tabela 8 observámos que esta operação foi muito mais custosa que a mesma realizada no SqlServer apesar do *dataset* do MongoDB ser substancialmente menor que o do SqlServer.

Não tínhamos dúvidas que, com a criação de um índice, obtivéssemos uma resposta mais rápida, mas não estávamos à espera de uma redução de, aproximadamente, 99% no tempo médio de resposta.

O índice criado consiste num índice do tipo *unique* e está associado à coluna IdUtilizador da coleção Utilizador, porque é a coluna responsável pelo *lookup* que associa um identificador a um nome.

```
db.Utilizador.ensureIndex( { IdUtilizador : 1 }, { unique : true})
```

Código 16 - Código para criação do indíce IdUtilizador MongoDB

4) Listagem do número de golos que cada equipa marcou

Para realização desta operação foi desenvolvida a seguinte query:

```
db.Equipa.aggregate([
  {
      $project: {
          IdEquipa: "$IdEquipa",
          NomeEquipa: "$NomeEquipa",
          Golos: { $sum:  "$jogosFutebol.GolosEquipa1" }
    }}
])
```

Código 17 - *Query* para listagem de número de golos por equipa MongoDB

O resultado da execução da *query* anterior é o seguinte:

```
{
    "_id": ObjectId("63b196c803501395990854d4"),
    "IdEquipa": NumberInt("1"),
    "NomeEquipa": "Tokyo FC",
    "Golos": 2
}

// 2
{
    "_id": ObjectId("63b196c803501395990854d5"),
    "IdEquipa": NumberInt("2"),
    "NomeEquipa": "Jakarta FC",
    "Golos": 4
}
```

Figura 12 - Resultado da execução da *query* número de golos por equipa MongoDB

Tempos de execução:

	1ª execução	2ª execução	3ª execução	Pior tempo	Melhor tempo	Tempo médio
S/Índice	0.083s	0.082s	0.084s	0.084s	0.082s	0.083s
C/Índice	0.085s	0.081s	0.079s	0.079s	0.085s	0.082s

Tabela 11 - Tempos de execução do número de golos por equipa MongoDB

Comparativamente à base de dados relacional, a operação é mais rápida, mas não conseguimos afirmar se é pelo *dataset* ser menor ou pelo modelo de base dados, porque todos os jogos de uma equipa vão estar sempre no documento respetivo à mesma.

Os tempos de execução da *query* já eram relativamente rápidos, mas para tentar otimizá-la decidimos criar um índice nos campos IdEquipa e NomeEquipa, mas não obtivemos melhoria nenhuma como é possível verificar.

Além disso, através do MongoCompass notámos que os índices não estavam a ser utilizados pelo MongoDB e não conseguimos entender o porquê.

5) Inserção de um conjunto de jogos terminados

Neste processo serão inseridos um conjunto de jogos terminados a partir de *queries* de inserção. Estas *queries* foram geradas no Excel através de dados gerados pelo Navicat.

Como desta vez os dados têm de ser trabalhados no Excel de modo a gerar as *queries* de inserção, o tempo contabilizado será apenas o da inserção e não da geração dos dados.

Tempos de execução:

	1ª execução (15000)	2ª execução (15000)	3ª execução (15000)
S/Índice	9.997s	10.120s	9.873s
C/Índice	4.913s	4.444s	4.837s

Tabela 12 - Tempos de execução para inserção de dados MongoDB

Como é possível ver pela tabela 11, a criação de um índice resultou na diminuição de, aproximadamente, 50% no tempo de inserção de dados.

6) Atualização de um conjunto de publicações

Decidimos substituir a query que atualizava os vencedores de cada jogo, porque no carregamento já estavam a ser inseridos os vencedores dos mesmos.

Posto isto, decidimos desenvolver uma nova *query* que atualizava o estado de todos os registos anteriores à data atual.

A *query* em questão é a seguinte:

```
db.Publicacoes.updateMany({
    DataHorario: {
        $lt: new Date()
    }
},
{
    $set: {
        Status: 'Deleted'
    }
}
)
```

Código 18 - *Query* de atualização de publicações MongoDB

O *output* após a execução da *query* é o seguinte:

Message | Summary | Result

acknowledged	matchedCount	modifiedCount
true	2586	2586

Figura 13 - Resultado da execução da *query* de atualização de publicações MongoDB

Tempos de execução:

	1ª execução	2ª execução	3ª execução
S/Índice	0.057s	0.044s	0.042s

Tabela 13 - Tempos de Execução da query de atualização de publicações MongoDB

4. Graph-based data model — Neo4J

4.1. Data model design

Para o desenvolvimento de uma base de dados orientada a grafos, foi decidida a utilização do Neo4j.

Este tipo de base de dados possui um conjunto de nós e relações. A base de dados orientada a grafos é um modelo "menos genérico" que o modelo relacional, que proporciona uma modelagem mais simples com o objetivo de obter uma maior performance. Isto é obtido porque não são utilizadas operações com grandes custos como os *joins*.

O modelo de dados torna-se assim mais simples, as entidades (nós) estão ligadas por relacionamentos com uma determinada direção.

Na próxima figura (Figura 12) é apresentada a nossa implementação para este modelo de base de dados.

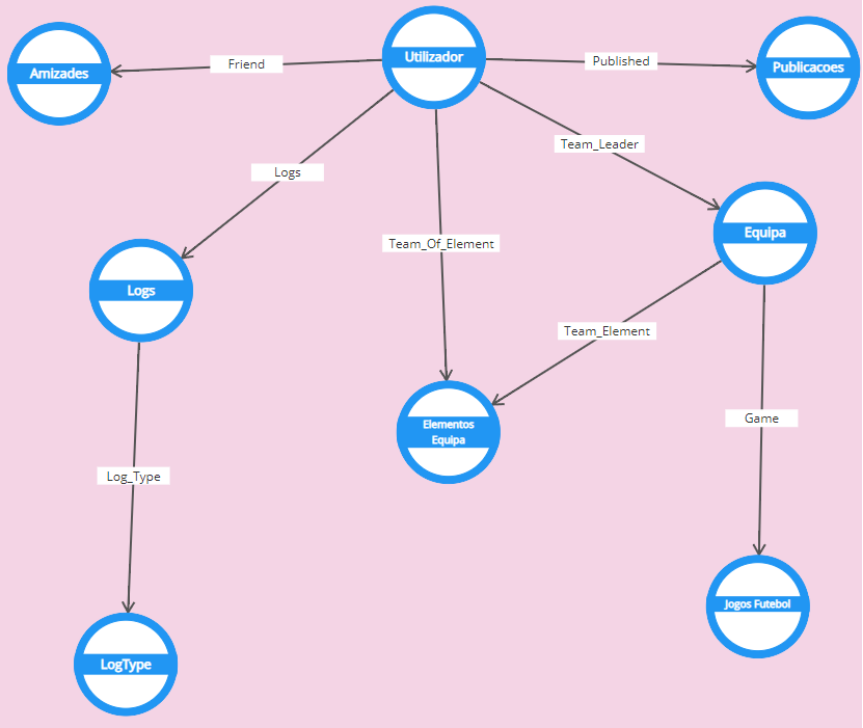


Figura 14 - Modelo de base de dados baseada em grafos

Como é possível verificar, a figura apresentada anteriormente (Figura 12), possui um conjunto de diferenças relativamente aos modelos apresentados previamente.

As entidades foram substituídas por nós e as ligações entre as tabelas substituídas por relações com a sua respetiva direção e cada uma tem o nome que a caracteriza de modo a torná-lo única.

Posto isto, os nós presentes na nossa implementação deste modelo de base de dados são:

- Utilizadores
- Jogos de futebol
- Equipas
- Publicações
- Logs

Relativamente à preparação dos dados foram utilizados CSVs para o processo de importação para o Neo4j.

Para que este processo fosse efetuado foi necessário guardar os ficheiros numa diretoria específica dentro do Docker como mostra a Figura 13. Tendo estes CSVs armazenados no interior da diretoria foi possível avançar para o próximo passo, que consistiu na utilização dos seguintes códigos (Código 15) para a realização do *import* dos dados para o Neo4j.

```
C:\Users\jpint>docker container ls
CONTAINER ID   IMAGE     COMMAND                  CREATED        STATUS         PORTS                                                      NAMES
2b12fe51c1f3   neo4j     "tini -g -- /startup…"   29 hours ago   Up 4 seconds   0.0.0.0:7474->7474/tcp, 7473/tcp, 0.0.0.0:7687->7687/tcp   neo4j

C:\Users\jpint>docker exec -ti 2b12fe51c1f3 bash
root@2b12fe51c1f3:/var/lib/neo4j# cd import/
root@2b12fe51c1f3:/var/lib/neo4j/import# ls
Amizades.csv  ElementosEquipa.csv  Equipa.csv  JogoFutebol.csv  LogType.csv  Logs.csv  Publicacoes.csv  Utilizador.csv
root@2b12fe51c1f3:/var/lib/neo4j/import# exit
exit

C:\Users\jpint>docker cp OneDrive\Desktop\csvFiles\SmallerData\Utilizador.csv 2b12fe51c1f3:/var/lib/neo4j/import
```

Figura 15 - Comandos Docker para Upload dos Ficheiros csv

```
--- Load Amizades
:auto LOAD CSV WITH HEADERS from "file:///Amizades.csv" AS line
CALL {
    with line
    CREATE (:Amizade {IdUtilizador1FK: toInteger(line.IdUtilizador1FK), IdUtilizador2FK:
toInteger(line.IdUtilizador2FK)})
} IN TRANSACTIONS OF 5000 ROWS

--- Load Elementos Equipa
:auto LOAD CSV WITH HEADERS from "file:///ElementosEquipa.csv" AS line
CALL {
    with line
    CREATE (:ElementosEquipa {IdEquipaFK: toInteger(line.IdEquipaFK), IdUtilizadorFK :
toInteger(line.IdUtilizadorFK)})
} IN TRANSACTIONS OF 5000 ROWS

--- Load LogType
:auto LOAD CSV WITH HEADERS from "file:///LogType.csv" AS line
CALL {
    with line
    CREATE (:LogType {IdLogType: toInteger(line.IdLogType), Descricao: line.Descricao})
} IN TRANSACTIONS OF 5000 ROWS

--- Load Logs
:auto LOAD CSV WITH HEADERS from "file:///Logs.csv" AS line
CALL {
    with line
    CREATE (:Logs {IdLog: toInteger(line.IdLog), IdLogTypeFK: toInteger(line.IdLogTypeFK),
IdUtilizadorFK: toInteger(line.IdUtilizadorFK), DataHorario: line.DataHorario})
} IN TRANSACTIONS OF 5000 ROWS

-- Load Publicacoes
:auto LOAD CSV WITH HEADERS from "file:///Publicacoes.csv" AS line
CALL {
    with line
    CREATE (:Publicacoes {IdPublicacao: toInteger(line.IdUtilizador), IdUtilizadorFK: toIn-
teger(line.IdUtilizadorFK), Conteudo: line.Conteudo, DataHorario: line.DataHorario, Status:
line.Status})
} IN TRANSACTIONS OF 5000 ROWS

-- Load Utilizador
:auto LOAD CSV WITH HEADERS from "file:///Utilizador.csv" AS line
CALL {
    with line
    CREATE (:Utilizador {IdUtilizador: toInteger(line.IdUtilizador), Nome: line.Nome, Pas-
sword: line.Password, Email: line.Email, DataNascimento: line.DataNascimento, DataCriacao:
line.DataCriacao, Geolocalizacao: line.Geolocalizacao})
} IN TRANSACTIONS OF 5000 ROWS
```

Código 19 - Código do carregamento dos ficheiros para o Neo4j

Após o carregamento dos dados, isto é, a criação dos nós, foi, então, necessário criar as relações entre os mesmos. Para chegar a esse resultado executamos os comandos presente no código 16 de modo a atingir as relações apresentadas na figura 12.

O resultado como mostra na Figura 14 foi idêntico ao pretendido e definido na modelo de dados.

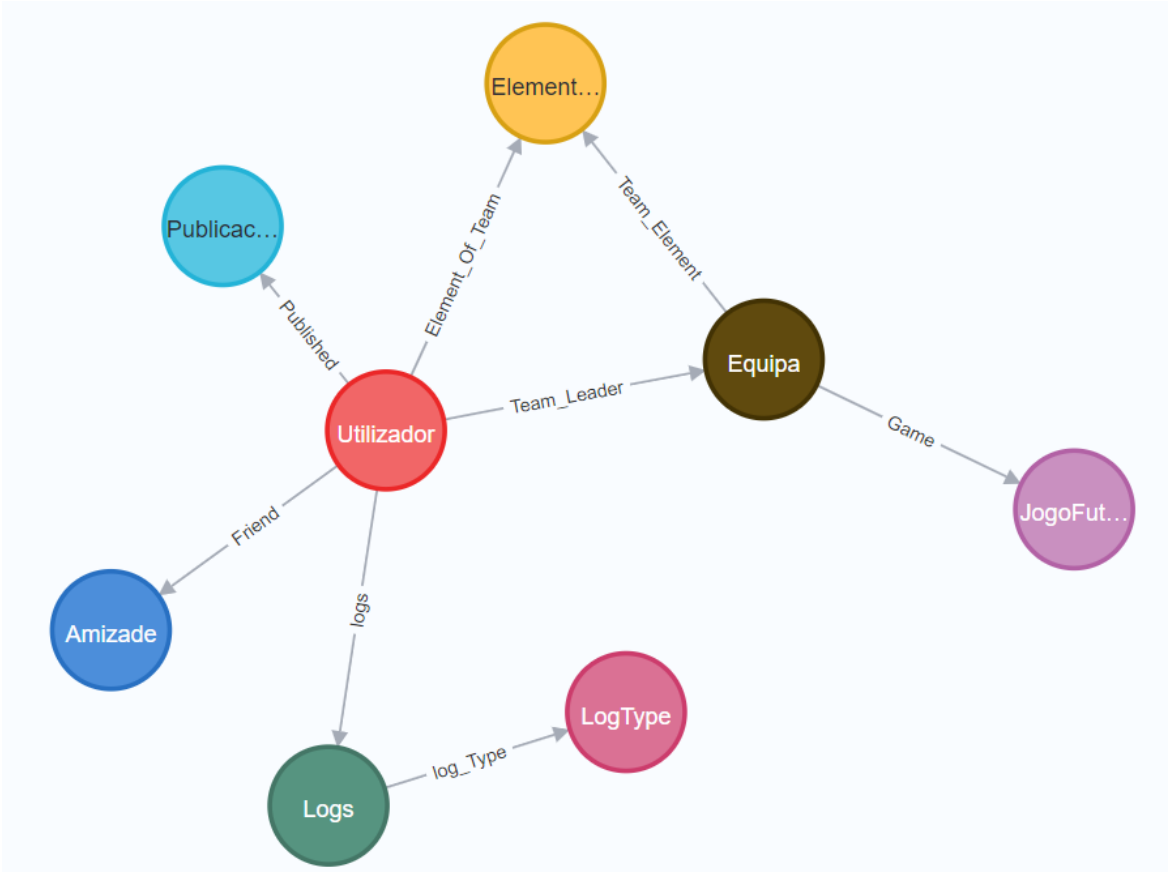


Figura 16 - Base de Dados Neo4j

4.2. Data model tuning and testing

Após o *import* dos dados foram realizadas algumas *queries.* A análise das *queries* não possuem os tempos de execução pois o Neo4j não fornece essa funcionalidade.

1) Jogos terminados de certos utilizadores

Para realização desta operação foi desenvolvida a seguinte *query*:

```
match (u:Utilizador{IdUtilizador: 19250})-[:Team_Leader]-(e:Equipa)-[:Game]-
(j:JogoFutebol) return u,j
```

Código 20 - Query para listar Jogos de determinados Utilizadores Neo4j

O resultado obtido foi o seguinte:

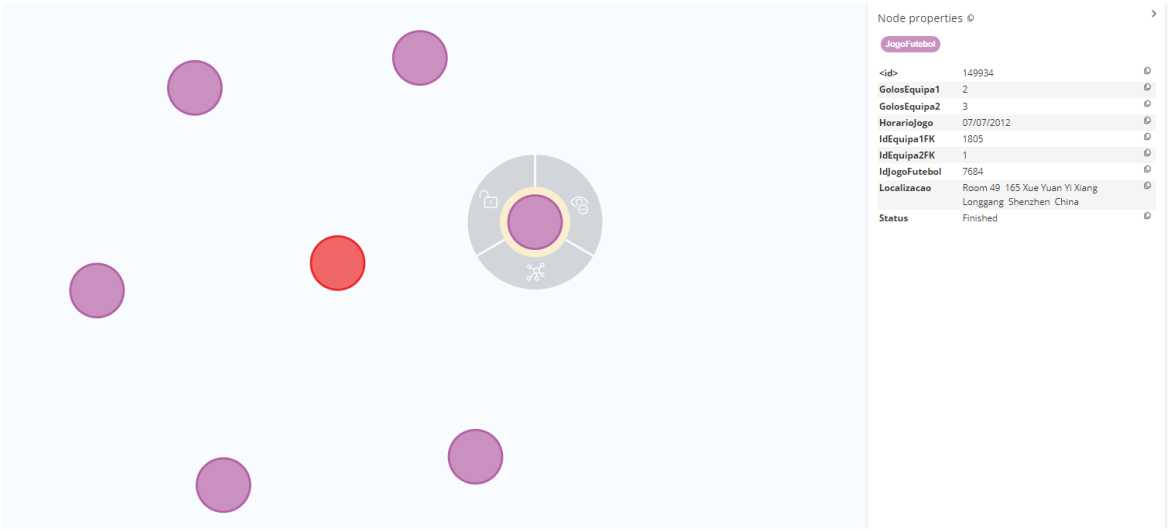


Figura 17 - Resultado da query Jogos terminados de certos Utilizadores

2) Obter Jogos, Capitão e Equipas

Para realização desta operação foi desenvolvida a seguinte *query*:

```
match (u:Utilizador)-[:Team_Leader]-(e:Equipa)-[:Game]-(j:JogoFutebol) return *
```

Código 21 - Query para Obter Jogos, Capitães e Equipas Neo4j

O resultado obtido foi o seguinte:

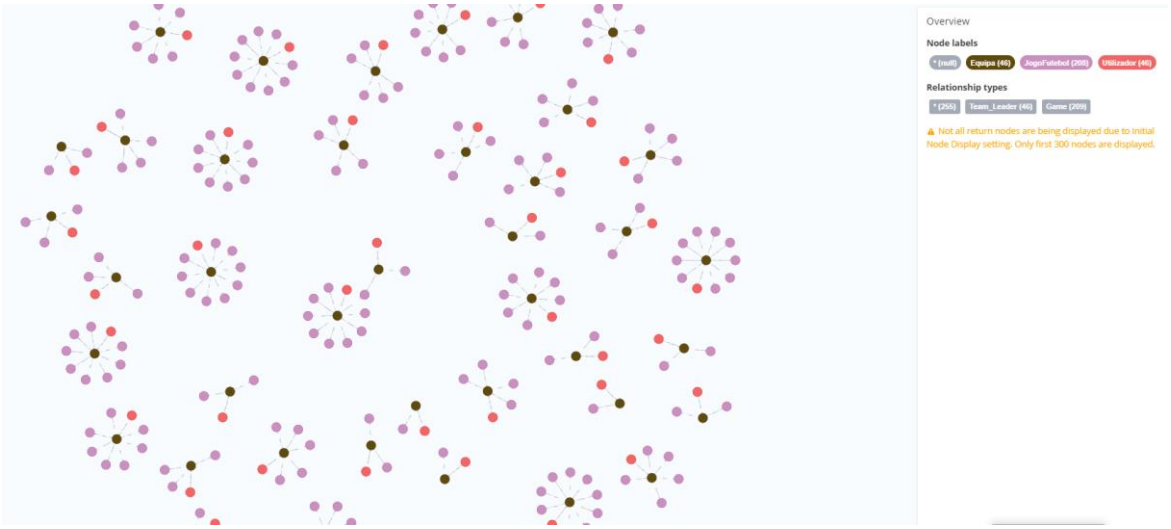


Figura 18 - Resultado da query Obter Jogos, Capitão e Equipas

3) Golos totais por equipa

Para realização desta operação foi desenvolvida a seguinte *query*:

```
match (e:Equipa)-[:Game]-(j:JogoFutebol{Status: "Finished"}) where e.IdE-
quipa = j.IdEquipa1FK return  e.NomeEquipa, SUM(j.GolosEquipa1)
match (e:Equipa)-[:Game]-(j:JogoFutebol{Status: "Finished"}) where e.IdE-
quipa = j.IdEquipa2FK return   e.NomeEquipa, SUM(j.GolosEquipa2)
```

Código 22 - Query Golos totais por equipa Neo4j

Para o desenvolvimento desta *query* tivemos alguns problemas, dos quais, a dificuldade em somar os golos de uma respetiva equipa quando esta estivesse no lugar de Equipa1 (A equipa que joga em casa) e Equipa2 (A equipa que joga fora). De modo a ultrapassar esta dificuldade decidimos realizar duas *queries* onde uma delas soma os golos de uma respetiva equipa quando esta joga em casa e outra para realizar o oposto.

O resultado obtido foi o seguinte:

	e.NomeEquipa	SUM(j.GolosEquipa1)
1	"Ciudad General Escobedo FC"	12
2	"Negapatam FC"	14
3	"Vantaa FC"	2
4	"Mestre FC"	32
5	"Preston FC"	18
6	"Alicante FC"	5
7		

Figura 19 - Resultado da query Golos Totais por Equipa1

	e.NomeEquipa	SUM(j.GolosEquipa2)
1	"Bijapur FC"	9
2	"Okazaki FC"	20
3	"Kecskemét FC"	24
4	"Lianzhou FC"	47
5	"McAllen FC"	34
6	"Temara FC"	16

Figura 20 - Resultado da query Golos Totais por Equipa2

4) Contar quantos amigos tem cada Utilizador

Para realização desta operação foi desenvolvida a seguinte *query*:

```
match (u:Utilizador)-[:Friend]-(a:Amizade) where u.IdUtilizador = a.IdUtilizador1FK
return  u.Nome, count(a)
```

Código 23 - Query para contar quantos amigos tem cada Utilizador Neo4j

O resultado obtido foi o seguinte:

	u.Nome	count(a)
1	"Brenda Young"	1
2	"Xue Xiaoming"	8
3	"Nakagawa Sakura"	9
4	"Lam Ming"	7
5	"Yin Chun Yu"	8
6	"Tang Xiaoming"	15

Figura 21 - Resultado da *query* que conta quantos amigos tem cada Utilizador

5. Discussion and conclusion

Tendo em conta a análise efetuada nos capítulos anteriores aos diferentes tipos de base de dados notamos que a base de dados ideal para o nosso modelo de negócio seria a base de dados relacional. Isto deve-se ao facto de o nosso modelo possuir várias relações, além disso, também é um modelo de base de dados que temos mais experiência.

Apesar de todos os constrangimentos que tivemos durante a realização deste trabalho prático especialmente no desenvolvimento da base de dados MongoDB e Neo4j conseguimos reconhecer as capacidades e utilidades destes modelos base de dados não usuais. Além disso, também tivemos algumas restrições temporais, porque começamos o desenvolvimento do trabalho relativamente tarde.

Posto isto, concluímos que foi um trabalho interessante, mas custoso e com a realização do mesmo adquirimos novos conhecimentos e ficamos mais preparados para o futuro.

References

[Sql Server]

https://learn.microsoft.com/en-us/sql/?view=sql-server-ver16

[MongoDb]

https://docs.docker.com/engine/reference/commandline/cli/

https://docs.mongodb.com/

https://studio3t.com/knowledge-base/articles/mongodb-import-json-csv-bson/

https://www.mongodb.com/docs/database-tools/mongoimport/

https://docs.mongodb.com/manual/reference/operator/aggregation

https://docs.mongodb.com/manual/reference/operator/aggregation/out/

https://www.digitalocean.com/community/tutorials/how-to-use-aggregations-in-mongodb

[Neo4j]

https://neo4j.com/docs/cypher-manual/current/

https://neo4j.com/blog/data-profiling-holistic-view-neo4j/

https://neo4j.com/developer/guide-import-csv/

https://neo4j.com/docs/getting-started/current/data-import/relational-to-graph-import/#import-relational

https://neo4j.com/developer/guide-import-csv/

https://neo4j.com/docs/cypher-manual/current/clauses/load-csv/

https://neo4j.com/docs/getting-started/current/cypher-intro/load-csv/

https://neo4j.com/labs/apoc/4.1/import/load-csv/

Contributions

Carlos Martins – 18836 João Azevedo – 18845

Ambos os elementos do grupo tiveram uma participação ativa e similar no trabalho desenvolvido.