**Relatório Trabalho Prático**

Football Data from Transfermarkt

14829, José Pedro Gomes

15708, Pedro Carvalho

15709, José Carreira

Barcelos, dezembro 2023

Índice

[1. Introdução 3](#_Toc155819690)

[1.1. Objetivos do Projeto 3](#_Toc155819691)

[1.2. Lista de recursos da Plataforma 4](#_Toc155819692)

[1.3. Volume de dados Plataforma 5](#_Toc155819693)

[2. Modelo de dados relacional — *SQL Server* 6](#_Toc155819694)

[2.1. Modelo de dados 6](#_Toc155819695)

[2.2. Importação de dados 8](#_Toc155819696)

[2.3. Ajustes e testes ao Modelo de dados 12](#_Toc155819697)

[2.4. Operações de manipulação de dados 17](#_Toc155819698)

[2.4.1. INSERT 17](#_Toc155819699)

[2.4.2. UPDATE 18](#_Toc155819700)

[2.4.3. DELETE 19](#_Toc155819701)

[3. Modelo de dados – *Mongo*DB 22](#_Toc155819702)

[3.1. Modelo de Dados 22](#_Toc155819703)

[3.1.1 Alteração de abordagem 23](#_Toc155819704)

[3.2. Importação de dados 24](#_Toc155819705)

[3.2.1. Novo modelo de dados 24](#_Toc155819706)

[3.3. Ajustes e testes ao Modelo de dados 25](#_Toc155819707)

[3.4. Operações de manipulação de dados 27](#_Toc155819708)

[3.4.1. INSERT 28](#_Toc155819709)

[3.4.2. UPDATE 29](#_Toc155819710)

[3.4.3. DELETE 30](#_Toc155819711)

[4. Modelo de dados — Neo4J 30](#_Toc155819712)

[4.1. Modelo de Dados 31](#_Toc155819713)

[4.2. Importação de dados 32](#_Toc155819714)

[4.2.1 Definição de Nós 34](#_Toc155819715)

[4.2.2 Definição de Relacionamentos 35](#_Toc155819716)

[4.3. Ajustes e testes ao Modelo de dados 37](#_Toc155819717)

[4.4. Operações de manipulação de dados 39](#_Toc155819718)

[4.4.1. INSERT 40](#_Toc155819719)

[4.4.2. UPDATE 41](#_Toc155819720)

[4.4.3. DELETE 41](#_Toc155819721)

[5. Considerações 43](#_Toc155819722)

[5.1. Dificuldades encontradas 43](#_Toc155819723)

[5.2. Discussão de resultados 44](#_Toc155819724)

[6. Conclusão 47](#_Toc155819725)

[Referências 47](#_Toc155819726)

[Contribuições 47](#_Toc155819727)

[Anexos 48](#_Toc155819728)

[Annex 01 – <<Nome do Annex 01>> 48](#_Toc155819729)

[Annex 02 – <<Nome do Annex 02>> 48](#_Toc155819730)

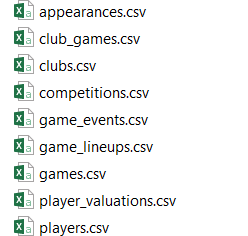
# Introdução

Este relatório é apresentado no âmbito do trabalho prático da unidade curricular de Bases de Dados Avançadas, do Mestrado de Engenharia Informática, lecionada pelo Professor Joaquim Silva. Esta UC tem como objetivo criar competências para o desenvolvimento e otimização de sistemas de armazenamento de dados relacionais e não relacionais, assim com análise de *performance* e persistência de dados. Também se pretende aumentar o conhecimento em relação a modelos e sistemas de dados orientados a documentos, *key/value* egrafos.

O desafio do trabalho prático consiste em criar bases de dados de tipologias diferentes: relacional, não relacional e grafos, modelar as bases de dados, fazer o *design*, implementar e monitorizar a sua *performance*. Pretende-se descrever o processo de cada base de dados, fazer comparações de resultados, conseguindo perceber as diferenças entre as BDs e quais seriam as melhores formas de otimizar cada uma delas, conseguindo identificar qual seria a melhor escolha para responder a determinadas perguntas.

## Objetivos do Projeto

O tema escolhido pelo grupo foi um *dataset* (constituído por vários ficheiros CSV) de estatísticas de futebol do Transfermarkt obtido no Kaggle (<https://www.kaggle.com/datasets/davidcariboo/player-scores>) com o nome “Football data from Transfermarkt”, com informação de várias competições nacionais e europeias desde 2012.

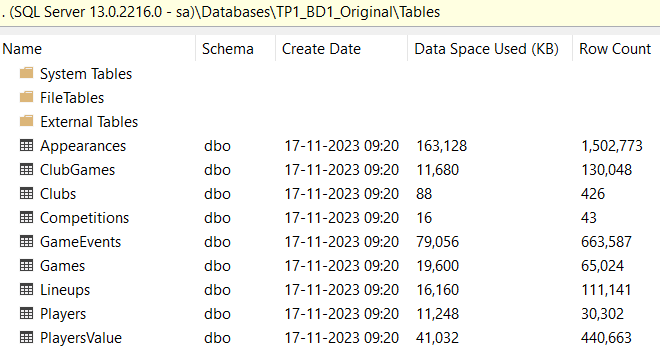


O objetivo passa por integrar todos os ficheiros *CSV,* pois cada ficheiro corresponde a determinada parte da informação, analisar o seu conteúdo e de seguida importar a informação para uma base de dados em *SQL*, para se conseguir ter melhor perceção do potencial, ou eventuais falhas, dos dados. Depois da análise de toda a informação disponível será normalizada a BD, serão validados e importados os dados e será testada a *performance* da BD, este trabalho também será efetuado em *MongoDB* e *Neo4j*.

## Lista de recursos da Plataforma

De seguida será explicada a estrutura de informação disponível, que deverá ser capaz de dar respostas às necessidades de uma plataforma de resultados, com informação detalhada dos jogos, jogadores e equipas.

A estrutura do *dataset* original foi importada para uma base de dados em *SQL* (“TP1\_BD1\_Original”), tendo sido criadas tabelas diretamente a partir de cada ficheiro *CSV* disponível. Como é visível na imagem abaixo, o *dataset* abrange informação variada em relação aos jogos, competições e intervenientes, contém alguma complexidade a relacionar as várias informações disponíveis e possui um volume de registos relativamente elevado, o que deverá permitir as análises pretendidas.



* *Appearances*, detalhe de cada jogador presente no jogo (golos, cartões, etc.);
* *ClubGames,* informação de cada jogo de determinada equipa;
* *Clubs,* informação de cada equipa;
* *Competitions,* competições disponíveis (campeonatos, taças nacionais, competições europeias, etc.);
* *GameEvents,* detalhe deeventos do jogo (golos, substituições, cartões, etc.);
* *Games,* informação de cada jogo;
* *Lineups,* onze inicial e substitutos que entraram no jogo;
* *Players,* informação relacionada com os jogadores;
* *PlayersValue,* detalhe das avaliações de cada jogador.

A partir desta informação pretende-se ser capaz efetuar análises detalhadas de competições, jogos, equipas, jogadores, treinadores, árbitros e estádios. O objetivo do grupo passará por estruturar de forma correta toda esta informação, para se conseguir desenvolver uma base de dados capaz de dar as respostas necessárias, de forma rápida e assertiva.

## Volume de dados Plataforma

O *dataset* depois de ser importado para *SQL* na BD “TP1\_BD1\_Original” permitiu ter a noção da dimensão de cada tabela. A análise da informação disponível permitiu chegar a algumas conclusões em relação aos totais de registos por época (tendo por base os registos disponíveis desde 2012):

* Cada época tem entre 5.5k e 6k registos de jogos;
* Os registos de eventos por jogo variam entre 46k e 73k registos;
* A informação relacionada com os detalhes do jogador em cada jogo varia entre 120k e 145k.

As operações mais frequentes deverão estar relacionadas com informação dos jogos, equipas, jogadores e eventos do jogo. Pelo que uma correta modelação da BD, armazenamento dos dados e a otimização no acesso a estes dados seja fundamental.

# Modelo de dados relacional — *SQL Server*

Depois de se analisar os dados chegou-se à conclusão de que havia muita redundância de informação e que seria necessário partir a informação em várias entidades, de forma a garantir a integridade e a normalização dos dados.

Foram identificadas as principais tabelas:

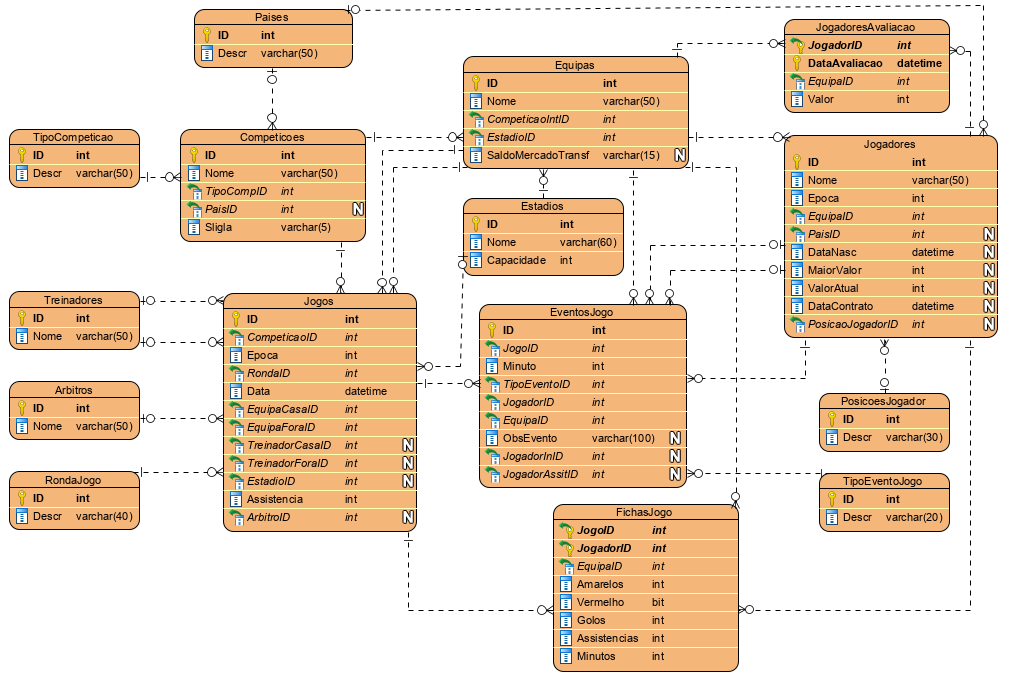
* Competições;
* Equipas;
* Jogos;
* Eventos do Jogo;
* Ficha de Jogo;
* Jogadores;
* Avaliação dos Jogadores;

Como tabelas auxiliares foram identificadas as seguintes tabelas:

* Árbitros;
* Estádios;
* Países;
* Posições de Jogador;
* Ronda/Jornada do Jogo;
* Tipo de Competição;
* Tipo de evento do Jogo.

## Modelo de dados

Foi efetuado o processo de normalização da estrutura do *dataset,* com o objetivo de eliminar redundância e eventuais anomalias do modelo de dados. O seguinte diagrama ER deu origem à base de dados “TP1\_BD1”, estruturada de forma a permitir ao modelo de dados dar as respostas necessárias ao negócio.



Foram definidas chaves primárias em todas as tabelas e algumas chaves estrangeiras. Alguns campos que não são obrigatórios foram definidos como *null*, houve especial atenção na definição do tipo de dados e tamanho dos campos *varchar,* para garantir a integridade e consistência de dados, e controlar o crescimento do tamanho de armazenamento da base de dados.

Em relação aos *IDs* das tabelas Equipas, Jogadores e Jogos, optou-se por continuar a utilizar o valor de *ID* (correspondente à informação em causa) disponível no *dataset* para se conseguir relacionar melhor a informação, em vez dos nomes/descrições, pois foram detetados casos em que o mesmo nome existe mais do que uma vez, aumentando significativamente a margem de erro no cruzamento da informação. Nas restantes tabelas os *IDs* são todos do tipo *int* com *identity increment* de1.

## Importação de dados

Como explicado no capítulo anterior, a informação disponível nos *CSV* foi importada para a BD “TP1\_BD1\_Original”, esta base de dados serviu como fonte de dados para a base de dados normalizada “TP1\_BD1”. Depois de profunda análise dos dados, identificou-se algumas particularidades nos registos:

* Jogadores com o mesmo nome (usou-se o *playerID* como ID de tabela);
* Mesmo estádio utilizado por mais do que uma equipa, sendo que as equipas mencionam capacidade diferente (usou-se a maior capacidade indicada);
* Estádios que apesar de não terem a capacidade disponível, têm a informação da afluência nos jogos (nesses casos utilizou-se o maior valor);
* Equipas que são mencionadas em jogos, mas não existem na tabela das equipas, provavelmente equipas de escalões inferiores em jogos de taça (estes jogos não foram importados);
* Competições com o mesmo nome em diferentes países (utilizou-se a sigla da competição para efetuar as comparações);
* Nem todos os jogadores têm avaliação disponível;
* Tabela *Lineups* só tem dados disponíveis desde 2021 (excluiu-se os dados desta tabela no modelo de dados final).

De seguida será explicado detalhadamente o processo de popular as tabelas da base de dados “TP1\_BD1”.

As primeiras tabelas a serem populadas logicamente foram as auxiliares:

* Paises



* TipoCompeticao



* Estadios



* Treinadores



* Arbitros



* RondaJogo



* PosicoesJogador



* TipoEventoJogo



*Queries* utilizadas na importação de dados para as tabelas principais:

* Competicao



* Equipas



* Jogadores



* JogadoresAvaliacao



* Jogos



* FichasJogo

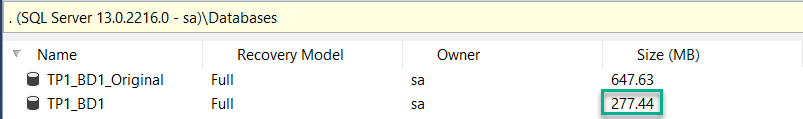


* EventosJogo



O processo de importação de dados ajudou a identificar problemas e oportunidades, daí surgiram pequenos ajustes na estruturação das tabelas: tipos de dados, tamanho máximo definido, campos não obrigatórios e em alguns casos (já explicados anteriormente) deixar de usar *ID* de auto incremento.

O que foi possível constatar de imediato, e que não foi surpresa para o grupo, foi a diferença de tamanho no armazenamento da informação entre as duas bases de dados, a original (após importação dos *CSV*) e a normalizada:



A base de dados “TP1\_BD1” ficou com 43% do tamanho em comparação com a BD original.

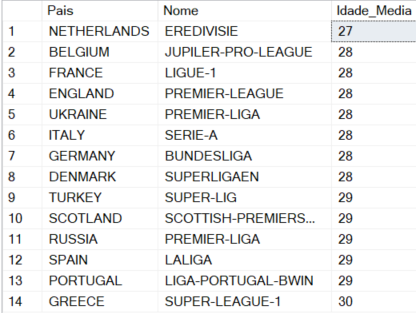
## Ajustes e testes ao Modelo de dados

O modelo de dados desenvolvido deverá ser capaz de dar resposta a uma plataforma de consulta de resultados, detalhes de jogos, equipa, jogadores, etc. daí terem sido testadas algumas *queries* para dar resposta a algumas questões que previsivelmente poderão ser colocadas por utilizadores, desde as questões mais simples às consultas mais criativas ou originais. Algumas das questões possíveis poderão ser:

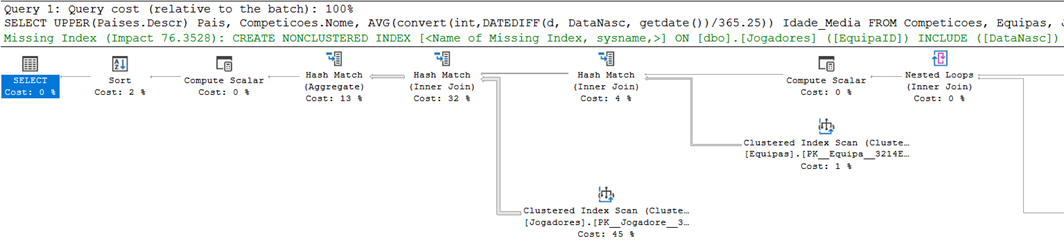
* Qual a idade média de jogadores das várias ligas nacionais? (Apresentar os resultados mais baixos primeiro)



Executando a *query* acima representada obtemos o seguinte resultado:



Ao analisar o *execution plan* desta *query* podemos verificar que indica a necessidade de criação de *index:*



Como sugerido no *execution plan* foi criado um *non clustered index:*

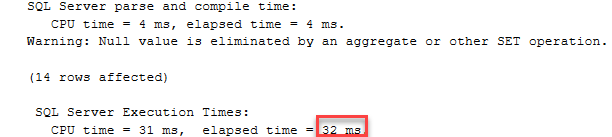


Para se conseguir verificar o tempo de execução da *query* em milissegundos, antes da sintaxe da *query* (para calcular a média de idades) colocou-se o seguinte comando:

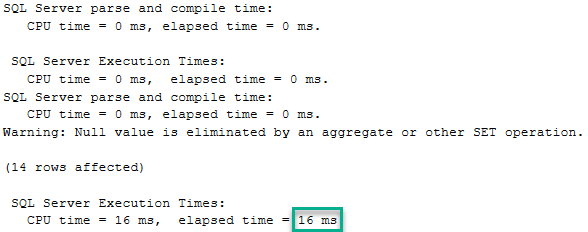


Isso permitiu obter um resultado mais detalhado em relação ao tempo de execução. Podemos abaixo verificar a diferença do tempo de execução das duas *queries*:

* Resultados de *query* **sem** *index* criado:



* Resultados de *query* **com** *index* criado:



Neste caso concreto a criação do *index* resultou num aumento de *performance* da *query*, pois o tempo de execução diminuiu (passou de 32 para 16 ms).

Outras possíveis questões que este modelo de dados deverá conseguir responder são:

* Da lista de 100 Jogadores com maior valor de mercado qual tem menos presenças em jogos nos últimos 2 anos?





* Por época qual o nº médio de golos marcado em campeonatos nacionais, desde a época 2020/2021?





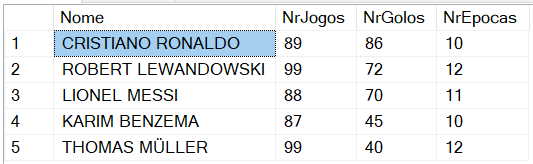
* Qual o arbitro (com pelos menos 10 jogos) que mostrou mais cartões vermelhos na Liga Europa?





* Qual o top 5 de jogadores com mais golos na *Champions* (mas que nunca jogaram o *qualifying* da *Champions*)?





* Qual o jogador que a entrar como substituto tem melhor média de golos (nos jogos em que entrou) por época?





* Qual o jogador que jogou mais jogos ao lado do Cristiano Ronaldo?





* Quais as competições onde a média de substituições por jogo é menor?





## Operações de manipulação de dados

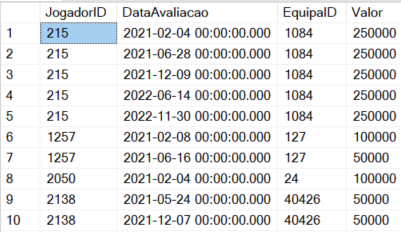
Além dos testes na execução de *queries*, também se testou as operações manipulação de dados: *Insert, Update* e *Delete*.

O desafio assumido pelo grupo foi efetuar estes testes sem afetar os dados disponíveis na base de dados (pois já estão validados e são fidedignos), pelo que se usou uma estratégia que passou por selecionar os registos da tabela “JogadoresAvalicao” as avaliações disponíveis referentes a 2021 e 2022 (a este intervalo de tempo estão alocados 90.149 registos).

*Query* desenvolvida para identificar os registos em causa:



Resultados obtidos:

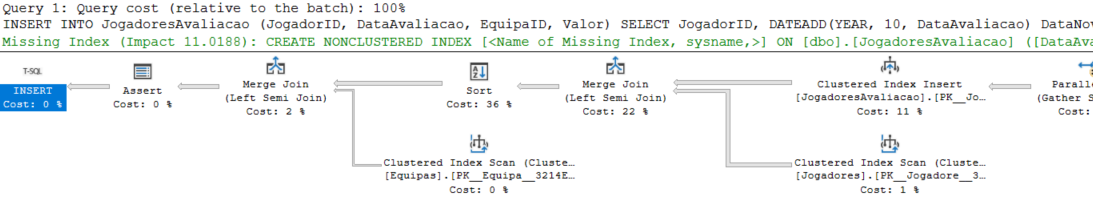


## INSERT

Inserir os registos identificados acima somando 10 anos à data de avaliação do registo, a *query* desenvolvida foi a seguinte:



A análise do *execution plan* da *query* identificou anecessidade de criar um *index*:

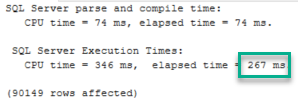


Sintaxe da criação do *index*:

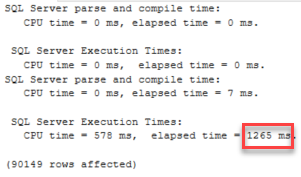


Foram ambas as situações testadas, *INSERT* sem *index* criadoe com *index* criado, de forma a conseguir-se perceber quais as diferenças de *performance*.

Sem *index* criado:



Com *index* criado:



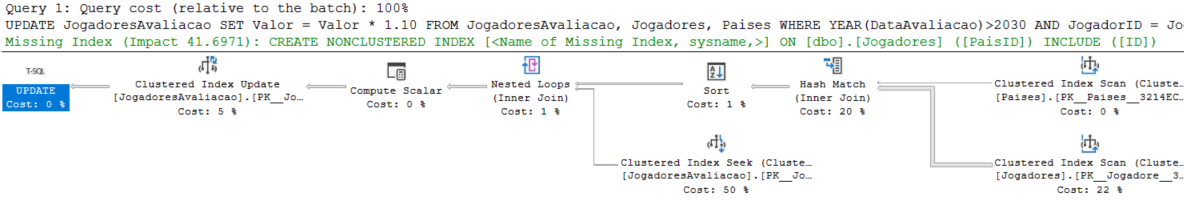
Perante os resultados obtidos pudemos comprovar que a inserção de registos tem tempos de execução diferentes com e sem *index*, como seria de esperar. Esta operação é mais rápida quando a tabela não tem *index* criado, o que já era expectável pois quando as tabelas têm *indexes,* além deinserir os dados, também tem de atualizar os *indexes* da tabela, para garantir a consistência de dados na tabela e em todos os *indexes* relacionados. No final desta operação foi eliminado o *index* para não afetar os restantes testes.

## UPDATE

Utilizando os mesmos registos gerados no ponto anterior (registos inseridos com datas de 2031 e 2032) foi testada a operação *UPDATE*, desta vez cruzando com a tabela dos Jogadores e Países, pois o objetivo será alterar (para os jogadores provenientes de Portugal) o valor da avaliação e aumentá-lo em 10%:



A análise do *execution plan* da *query* identificou anecessidade de criar um *index*:

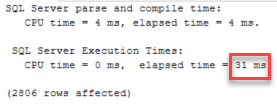


Sintaxe da criação do *index:*

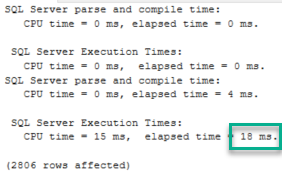


Testou-se novamente a execução da *query* com e sem a criação do *index*:

Sem *index* criado:



Com *index* criado:



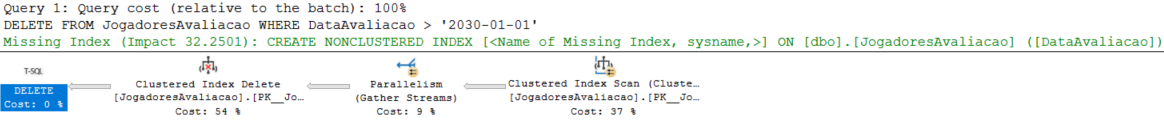
A criação do *index* resultou numa melhoria de *performance* da *query,* passando de 31 para 18 milissegundos após a criação do *index*. No final desta operação foi eliminado o *index* para não afetar os restantes testes.

## DELETE

Novamente utilizando os registos obtidos no ponto 2.4.1, desta vez o objetivo será testar a operação *DELETE*, de forma a remover os registos inseridos e posteriormente alterados, para que a tabela volte a ter os dados originais. Sintaxe utilizada para eliminar os registos:



A análise do *execution plan* da *query* identificou anecessidade de criar um *index*:

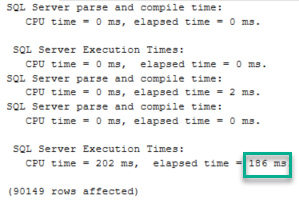


Sintaxe da criação do *index*:

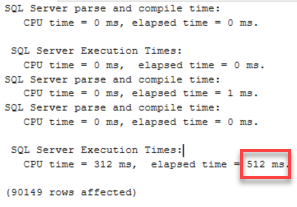


A execução da *query* foi testada com e sem a criação do *index*:

Sem *index* criado:



Com *index* criado:



A obtenção destes resultados permitiu comprovar que a eliminação de registos tem tempos de execução diferentes com e sem *index*, sem *index* tem melhor *performance* do que com.

Enquanto que se a tabela não tiver *index* criado, o SQL apenas procede à eliminação dos registos. Enquanto que se a tabela tiver *index* criado, além da eliminação dos registos também os *indexes* terão de ser atualizados, de forma a garantir a consistência de dados na tabela e em todos os *indexes* relacionados. Isso pode fazer com que a operação seja mais lenta pois existe o trabalho adicional de reorganização de *indexes*.

# Modelo de dados — *Mongo*DB

## Modelo de Dados

Em *MongoDB*, os dados são armazenados em documentos no formato *BSON* (*Binary JSON*), que são estruturas de dados semelhantes a *JSON*. Cada documento contém um *id* único criado pela BD e pertence a uma *collection*. Uma vez que foram utilizadas várias *collections* alguns ids foram mantidos para ser possível relacionar as diferentes *collections*.

Optou-se por criar as seguintes *collections*:

* Jogos;
* Jogadores;
* EventosJogo;
* FichasJogo;
* JogadoresAvaliacao.

Na imagem seguinte está representada a estrutura do documento para a *collection* Jogos.

Uma imagem com texto, captura de ecrã, software, Página web

Descrição gerada automaticamente

Os campos desta *collection* são:

* \_id: Campo criado automaticamente que referência o documento.
* Competicao: Campo do tipo subdocumento que contem campos com informações da competição (Nome, Tipo, Sigla e Pais).
* Estadio: Campo do tipo subdocumento que que contem campos com informação do estádio (Nome e capacidade).
* JogoId: Campo simples que contem um id de jogo para ser possível relacionar com as outras *collections*.
* Ronda: Campo simples.
* Epoca: Campo simples.
* Arbitro: Campo simples.
* Assistencia: Campo simples.
* Treinadores: Campo do tipo subdocumento que que contem campos com informação dos treinadores (casa e fora).
* “Equipas”: Campo do tipo subdocumento que que contem campos com informação das equipas (casa e fora).

## 3.1.1 Alteração de abordagem

Após todos os testes efetuados nos pontos anteriores o desempenho das *querys* **não**correspondia à nossa expectativa face aos resultados obtidos no modelo relacional. Depois de análise mais detalhada sobre o modo de funcionamento de bases de dados não relacionais, entendeu-se que o modo como os dados estavam relacionados e o tipo de *queries* utilizadas eram fatores determinantes na *performance* de bases de dados *NOSQL*.

Sendo o *MongoDB* uma base de dados orientada a documentos decidiu-se alterar o modo como os dados estavam armazenadas e o tipo de consultas realizadas.

Foi decidido reduzir o número de *collections* e adicionar os dados de forma “aninhada” em cada *collection*. A imagem abaixo representada mostra as *collections* que resultaram das operações de tratamento e aninhamento realizadas. Este processo será explicado em mais detalhe na seção 3.2.

Uma imagem com texto, Tipo de letra, captura de ecrã, design

Descrição gerada automaticamente

## Importação de dados

Uma vez que houve tratamento de dados no modelo relacional, foi utilizada a base de dados relacional como fonte de origem para a importação, para que os dados utilizados nas diferentes bases de dados sejam consistentes assim como os resultados obtidos fiáveis.

Foi utilizada a ferramenta *Navicat* para a importação direta, a base de dados ficou com o nome “BDA1\_Original”.

A ferramenta *NaviCat* permitiu configurar várias origens de dados diferentes. Neste caso foi configurada uma ligação ao *SQL* e uma outra ao *MongoDB*. Após estarem as duas ligações configuradas foi utilizada a ferramenta de importação direta de *SQL* para *Mongo*. Esta importação direta transforma cada tabela da base de dados de origem (*SQL*) numa *collection* na base de dados de destino (*MongoDB*).

*Uma imagem com captura de ecrã, texto, software, Software de multimédia

Descrição gerada automaticamente*

## Novo modelo de dados

A nova abordagem teve como objetivo evitar carregar e consumir os dados de forma “*bulk*”, mas fazer algum tratamento aos mesmos quando os importamos para *collections*.

A *collection* “Competicao” foi carregada a partir do *script* em anexo ao relatório com o nome “*Competicao.txt*”. Na estrutura de dados foram adicionados dois campos calculados, a média de idade dos jogadores e a média de golos por competição (de modo a reduzir o processamento necessário para obter estes resultados)

A *collection* “Jogos” passou a ter mais dados “aninhados”, foram adicionados os eventos dos jogos assim como as fichas de jogo. O *script* utilizado para a carga dos dados está em anexo com o nome “Jogos.txt”.

A *collection* “Jogadores” passou a conter também dados como equipa, pais e dados de avaliação. O *script* utilizado para carga de dados está em anexo com o nome “Jogadores.txt”.

Uma imagem com texto, captura de ecrã, Tipo de letra, número

Descrição gerada automaticamente

## Ajustes e testes ao Modelo de dados

Para testar a *performance* da base de dados utilizamos as mesmas *queries* que foram utilizadas para o modelo relacional como termo de comparação. Foram escolhidas as primeiras 3 questões:

* Qual a idade média de jogadores das várias ligas nacionais? (Apresentar os resultados mais baixos primeiro)

Para responder à pergunta executamos a *query* abaixo (em anexo ao relatório com o nome *Pergunta1\_v2.txt*).

Uma imagem com texto, captura de ecrã, software, Software de multimédia

Descrição gerada automaticamente

Após a execução podemos verificar que tempo passou de 42 segundos (abordagem anterior) para 0.26 segundos. Uma diferença **bastante** significativa face à abordagem inicial, o que demonstra novamente que em bases de dados *NOSQL* a maneira que os dados estão organizados tem um impacto profundo na sua velocidade de obtenção.

* Da lista de 100 Jogadores com maior valor de mercado qual tem menos presenças em jogos nos últimos 2 anos?

Uma imagem com captura de ecrã, texto, Software de multimédia, software

Descrição gerada automaticamentePara responder a esta pergunta, foi criada uma tabela com os 100 jogadores mais valiosos que depois foi cruzada com a restante *query* (em anexo com o nome *Pergunta2\_v2.txt*).

Após executar a *query* o tempo foi de 4.7 segundos.

Foram criados índices utilizando os seguintes comandos:



Uma imagem com texto, software, Software de multimédia, Software gráfico

Descrição gerada automaticamenteVoltou-se a correr a *query* e tempo reduziu para 3.4 segundos.

Antes da alteração de abordagem o tempo obtido para responder a esta pergunta era de 6.4 segundos.

* Por época qual o nº médio de golos marcado em campeonatos nacionais, desde a época 2020/2021?

O *script* para responder a esta pergunta está em anexo com nome “Pergunta3\_v2.txt”, após correr o *script*, o tempo de execução foi de 0.249 segundos. Também aqui a criação de indexes não melhora a *performance*.

Uma imagem com texto, captura de ecrã, software, ecrã

Descrição gerada automaticamente

## Operações de manipulação de dados

Além dos testes na execução de *queries*, também se testou as operações manipulação de dados: *Insert, Update* e *Delete* tal como no modelo relacional, de forma a testar a velocidade destas operações.

Para garantir que o volume de dados testado é igual ao modelo relacional foi utilizada uma estratégia que passou por selecionar da tabela “Jogadores” as avaliações disponíveis referentes a 2021 e 2022 (a este intervalo de tempo estão alocados 90.149 registos, o mesmo numero de registos obtidos no modelo relacional).

Uma imagem com software, Software de multimédia, texto, eletrónica

Descrição gerada automaticamenteNa imagens abaixo representadas podemos ver a *query* desenvolvida para obter o volume de dados considerado e os resultados obtidos.

## INSERT

Inserir os registos identificados acima somando 10 anos à data de avaliação do registo, a *query* desenvolvida foi a seguinte:

Uma imagem com texto, software, Software de multimédia, Software gráfico

Descrição gerada automaticamente

O tempo de execução foi de 83 segundos. De seguida foi criado um *índex* para verificar as diferenças de *performance*.

Sintaxe de criação do *index*:

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Após a criação do *index* voltou-se a fazer o *INSERT* e desta vez o tempo de execução da tarefa aumentou para 104 segundos. O mesmo comportamento já tinha sido verificado no modelo relacional, o aumento de tempo é devido ao trabalho extra que a base de dados tem após inserir dados de manter o *index* atualizado.

Uma imagem com texto, software, Software de multimédia, ecrã

Descrição gerada automaticamente

No final desta operação foi eliminado o *index* para não afetar os restantes testes.

## UPDATE

Utilizando os mesmos registos gerados no ponto anterior (registos inseridos com datas de 2031 e 2032) foi testada a operação *UPDATE*. O objetivo será alterar (para os jogadores com nacionalidade Portuguesa) o valor da avaliação e aumentá-lo em 10%:

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteFoi criada uma *collection* temporária com os dados filtrados já por país. O *script* utilizado para construir a mesma foi o seguinte:

Apos estar criada a *collection* foi executado o *update,* no primeiro caso sem *index* criado, e no segundo com *index* criado. Os tempos de execução são muito idênticos, com vantagem para o *update* com *index.*

Sintaxe de criação de *index*:

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

Uma imagem com captura de ecrã, Software de multimédia, software, texto

Descrição gerada automaticamenteSem *index* criado:

Com *index* criado:

Uma imagem com texto, captura de ecrã, Software de multimédia, software

Descrição gerada automaticamente

## DELETE

Novamente utilizando os registos obtidos no ponto 3.4.1, desta vez o objetivo será testar a operação *DELETE* (de forma a remover os registos inseridos e posteriormente alterados) para que a *collection* volte a ter os dados originais. A execução da *query* foi testada com e sem a criação do *índex*.

Uma imagem com texto, Software de multimédia, software, captura de ecrã

Descrição gerada automaticamenteSem *index* criado:

Uma imagem com texto, Software de multimédia, software, captura de ecrã

Descrição gerada automaticamenteCom *index* criado:

A obtenção destes resultados permitiu comprovar que a eliminação de registos tem tempos de execução diferentes com e sem *índex*. Verificou-se que sem *index* é possível obter melhor *performance* do que com *index*.

A operação com *index* criado é mais lenta pois existe o trabalho adicional de manter o mesmo atualizado aquando da eliminação de registos.

# Modelo de dados — Neo4J

Depois da modelação em SQL Server e MongoDB, iniciou-se o mesmo processo, mas utilizando uma Base de Dados de Grafos, NEO4J. A Base de Dados armazena os dados em forma de grafos, composta por nós e relacionamentos. Para este efeito utilizou-se o NEO4J Desktop 1.5.9, para se conseguir obter resultados válidos, tendo em conta que irão correr localmente e não em ambiente web.

A estratégia utilizada, também para tirar partido do motor da base de dados, passou por juntar alguma da informação que tinha sido normalizada na modelação de SQL, assumindo alguma redundância da informação. Mas tirando partido da performance na obtenção de dados disponíveis em nós com relacionamentos.

Como entidades que dão origem a nós foram identificadas as seguintes:

* Competicoes;
* Equipas;
* EventosJogo;
* FichasJogo;
* Jogadores;
* JogadoresAvaliacao;
* Jogos.

Como relações foram identificadas as seguintes:

* COMPETE (relação entre Equipas e Competições);
* CONTEM (relação entre FichasJogo e Jogos);
* ESTEVE\_PRESENTE (relação entre Jogadores e FichasJogo);
* JOGA (relação entre Jogadores e Equipas);
* PERTENCE (relação entre Jogos e Competições)

## Modelo de Dados

O processo de modelação em NEO4J foi um desafio devido ao desconhecimento existente, no grupo, em relação a este modelo de Bases de Dados. O maior desafio passou pela correta identificação dos nós (e as propriedades a considerar), de forma a identificar corretamente as principais entidades que o modelo iria gerir, e das relações existentes entre os nós disponíveis.

Na identificação dos nós, propriedades e relações foram tidas em consideração as questões a que o modelo deveria ser capaz de responder. Foi criada a base de dados “BDA1”, onde toda a informação está alojada e que deverá permitir suportar o processo de negócio.

Uma imagem com texto, captura de ecrã, Tipo de letra

Descrição gerada automaticamenteUma imagem com texto, captura de ecrã, menu, Tipo de letra

Descrição gerada automaticamenteA sintaxe utilizada para consultar dados, executar comandos e efetuar operações em NEO4J é “*cypher*”, a utilização desta linguagem também representou um desafio para o grupo, mas revelou-se bastante interessante e depois de alguma utilização, e entendimento de princípios básicos, a própria sintaxe torna-se mais intuitiva e utilização mais acessível.

Na imagem anterior é possível verificar os nós e as relações disponíveis na base de dados “BDA1”. Em comparação com *SQL* as entidades são diferentes, pois a estratégia passou por juntar os dados das Equipas, Jogos, Jogadores e Competições, de forma que fosse possível obter respostas com a *performance* pretendida.

## Importação de dados

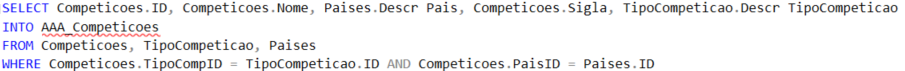
A estratégia utilizada na importação de dados passou pela seleção da informação disponível na base de dados em SQL, através de *queries.* Foram criadas tabelas “temporárias”, a partir das quais se efetuou a exportação para ficheiros *CSV*.

Uma imagem com texto, Tipo de letra, algebra

Descrição gerada automaticamente

A imagem acima permite verificar as tabelas que foram criadas para servir de suporte à exportação de dados para os ficheiros CSV.

Na imagem abaixo é possível verificar a sintaxe que deu origem a uma dessas tabelas:



Através da ferramenta disponibilizada em SQL foi possível efetuar a exportação do conteúdo das tabelas para os respetivos ficheiros CSV, exemplo na imagem abaixo:

Uma imagem com texto, captura de ecrã, software, Página web

Descrição gerada automaticamente

Depois de se gerar os ficheiros CSV e da instalação do NEO4J Desktop, passou-se para a modelação dos dados. Criou-se o projeto “NEO4J” e configurou-se o DBMS (Database Management System) “TrabalhoBDA1”, depois disso foi criada a base de dados “BDA1” onde toda a informação ficará armazenada. Na pasta do projeto também estão disponíveis os CSV’s disponíveis (imagem abaixo).

Uma imagem com texto, captura de ecrã, software, Página web

Descrição gerada automaticamente

Depois da base de dados e dos ficheiros CSV estarem disponíveis, iniciou-se o processo de importação dos nós e relações. O NEO4J permite através da utilização de comandos ou *cypher* uma modelação “automática” da base de dados. Pois este processo transforma as linhas e colunas do CSV em nós, relacionamentos e propriedades, criando uma representação estruturada das entidades e suas interconexões.

### 4.2.1 Definição de Nós

Criação de nós através da importação, utilizando *cypher*, de ficheiros CSV para NEO4J:

* Competicoes

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

* Equipas

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

* EventosJogo

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

* FichasJogo

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

* Jogadores

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

* JogadoresAvaliacao

Uma imagem com texto, Tipo de letra, captura de ecrã

Descrição gerada automaticamente

* Jogos

Uma imagem com texto, Tipo de letra, file, captura de ecrã

Descrição gerada automaticamente

### 4.2.2 Definição de Relacionamentos

Criação de relacionamentos através da importação, utilizando *cypher*, de ficheiros CSV para NEO4J:

* COMPETE (relação entre Equipas e Competições)

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

* CONTEM (relação entre FichasJogo e Jogos)

Uma imagem com texto, captura de ecrã

Descrição gerada automaticamente

* ESTEVE\_PRESENTE (relação entre Jogadores e FichasJogo)

Uma imagem com texto, captura de ecrã, diagrama

Descrição gerada automaticamente

* JOGA (relação entre Jogadores e Equipas)

Uma imagem com texto, captura de ecrã, diagrama, design

Descrição gerada automaticamente

* PERTENCE (relação entre Jogos e Competições)

Uma imagem com mapa, texto, captura de ecrã, diagrama

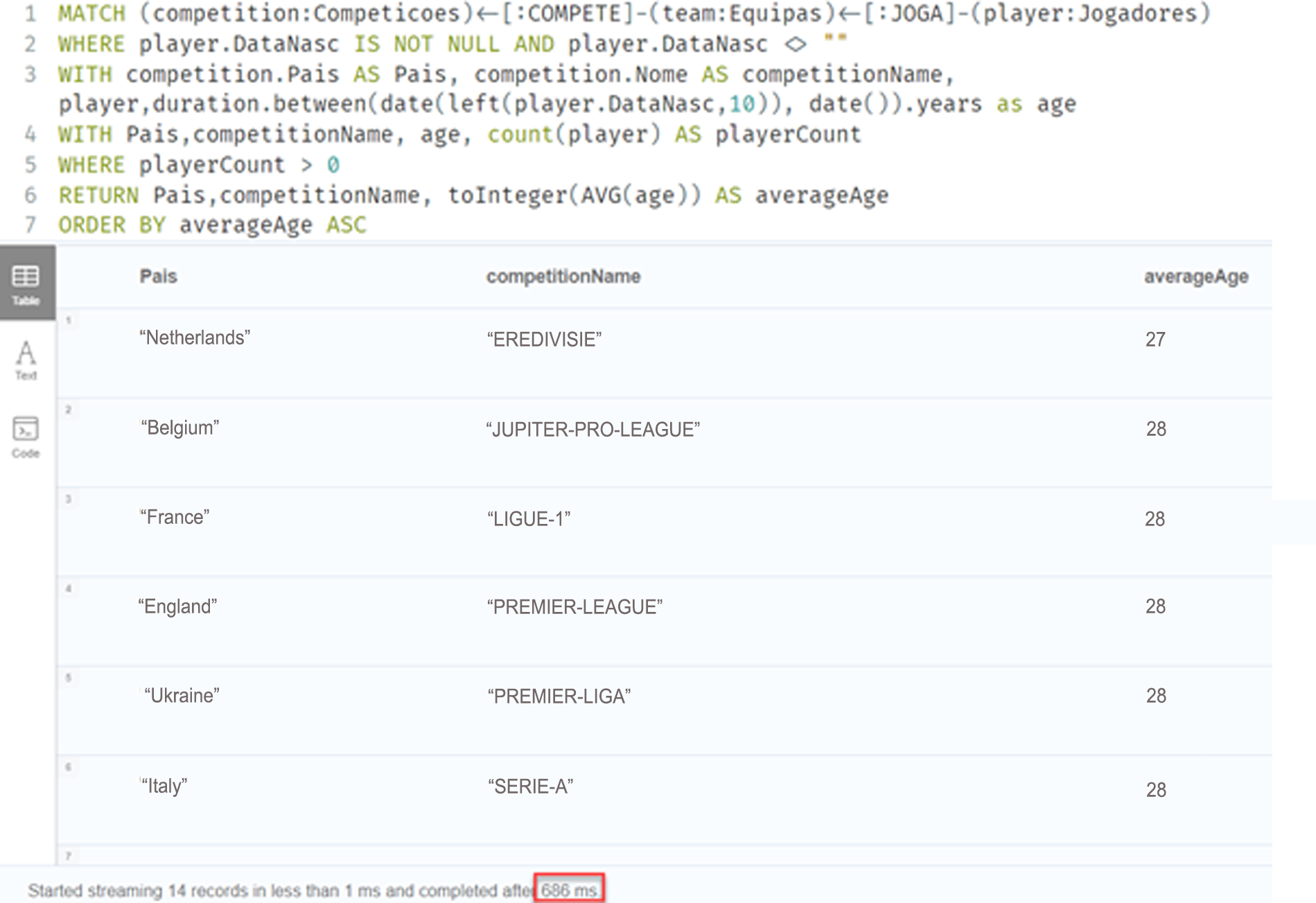
Descrição gerada automaticamente

## Ajustes e testes ao Modelo de dados

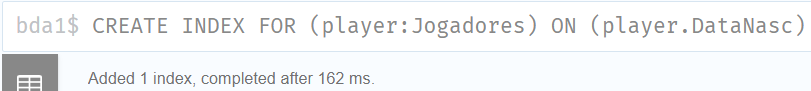
O modelo de dados foi estruturado desta forma em NEO4J com o objetivo de ser capaz de dar resposta às necessidades do negócio, respondendo às questões identificadas. Novamente com a utilização de *cypher,* foram testadas algumas *queries* para dar resposta às questões. Este processo foi moroso pois esta linguagem apesar de poderosa e expressiva, confere algumas dificuldades na aprendizagem, obrigando a um esforço adicional.

Para dar resposta às questões foram desenvolvidas as seguintes *queries*:

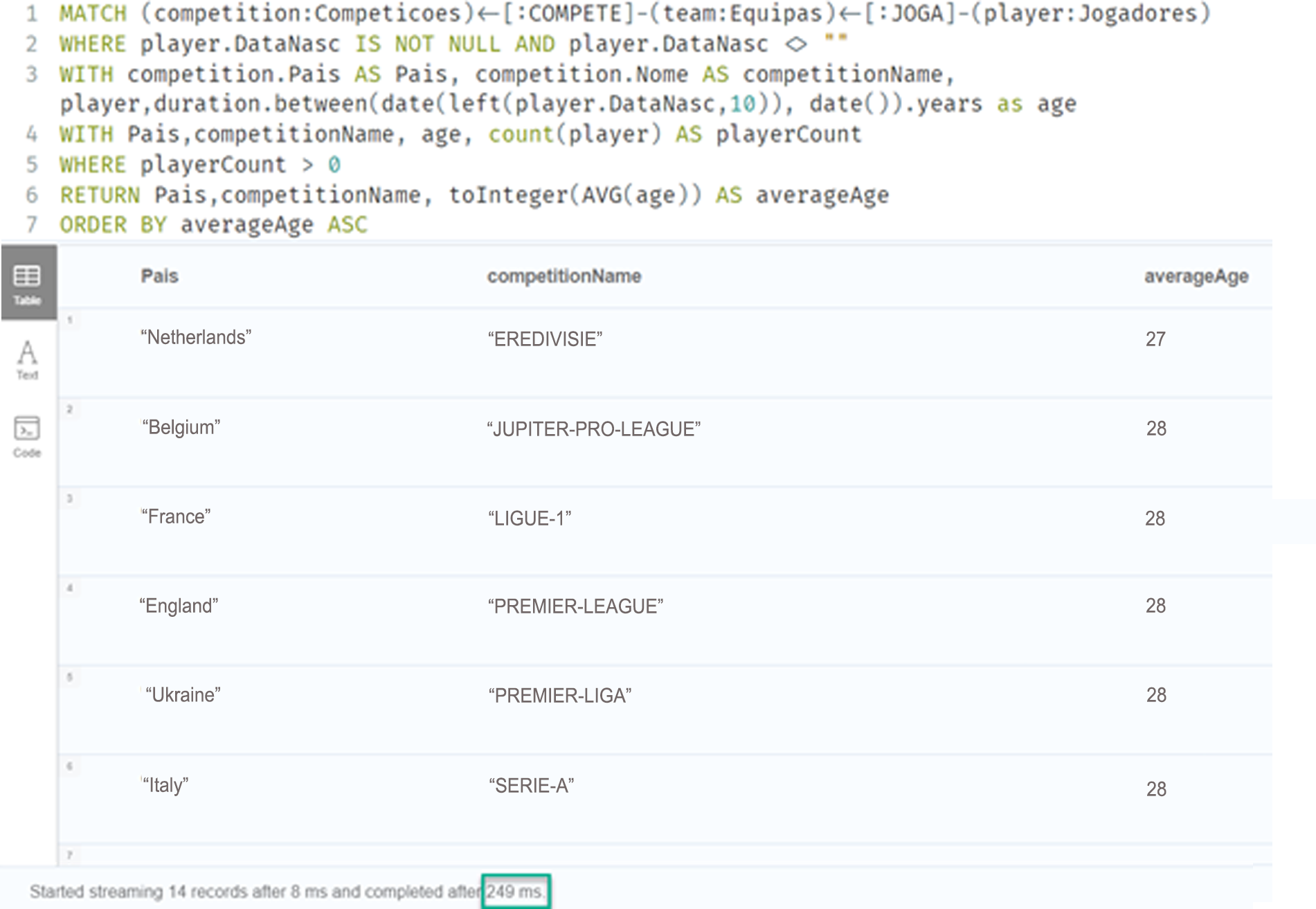
* Qual a idade média de jogadores das várias ligas nacionais? (Apresentar os resultados mais baixos primeiro)



Como podemos observar na imagem anterior, a *query* demorou 686 milissegundos a processar, para devolver os 14 registos. Pelo que se testou a criação de um índice para a propriedade “DataNAsc”:



De seguida foi novamente testada a *query* e os resultados foram significativamente melhores, passando o tempo de processamento para 249 milissegundos, menos de metade que o tempo da execução inicial (disponível na imagem abaixo).



* Da lista de 100 Jogadores com maior valor de mercado qual tem menos presenças em jogos nos últimos 2 anos?

Uma imagem com texto, captura de ecrã, Tipo de letra, software

Descrição gerada automaticamente

* Por época qual o nº médio de golos marcado em campeonatos nacionais, desde a época 2020/2021?

Uma imagem com texto, captura de ecrã, número, Tipo de letra

Descrição gerada automaticamente

## Operações de manipulação de dados

Também em NEO4J foram testadas as operações manipulação de dados: *Insert*, *Update* e *Delete*. A estratégia utilizada para a execução destas operações foi semelhante aos outros modelos: selecionar os registos da entidade “JogadoresAvalicao” as avaliações disponíveis referentes a 2021 e 2022 (a este intervalo de tempo estão alocados 90.149 registos).

*Query* desenvolvida para identificar os registos em causa e os resultados obtidos:

Uma imagem com texto, captura de ecrã, círculo

Descrição gerada automaticamente

## INSERT

Para inserir os registos identificados acima somando 10 anos à data de avaliação do registo, a *query* desenvolvida foi a seguinte:

Uma imagem com texto, captura de ecrã, Tipo de letra, ecrã

Descrição gerada automaticamente

Tempo de execução sem *index* criado:



De seguida foi testada a criação de um *index* para tentar melhorar a *performance* desta operação, sintaxe da criação do *index*:



Tempo de execução da *query*, depois da criação do *index*:



Perante os resultados obtidos pudemos comprovar que a inserção de registos apresenta tempos de execução diferentes com e sem *index*. Esta operação foi um pouco mais rápida quando a tabela tem *index* criado, ao contrário dos outros modelos analisados, o que pode ser justificado pelo volume de dados existente. É expectável que com aumento de dados a presença do *index* possa aumentar os tempos de execução de *inserts*. No final desta operação foi eliminado o *index* para não afetar os restantes testes.

## UPDATE

Utilizando os mesmos registos gerados no ponto anterior (registos inseridos com datas de 2031 e 2032) foi testada a operação *UPDATE*, o objetivo será alterar (para os jogadores com nacionalidade Portuguesa) o valor da avaliação e aumentá-lo em 10%:

Uma imagem com texto, Tipo de letra, captura de ecrã, file

Descrição gerada automaticamente

Tempo de execução da *query* indicada antes da criação do *index*:



De seguida foi testada a criação de um *index* para tentar melhorar a *performance* desta operação, sintaxe da criação do *index*:



Tempo de execução da *query*, depois da criação do *index*:



Perante os resultados obtidos pudemos comprovar que a atualização de registos origina tempos de execução diferentes com e sem *index*. A criação do *index* resultou numa melhoria de *performance* da *query,* passando de 294 para 185 milissegundos após a criação do *index*. No final desta operação foi eliminado o *index* para não afetar os restantes testes.

## DELETE

Novamente utilizando os registos obtidos no ponto 4.4.1, desta vez o objetivo será testar a operação DELETE, de forma a remover os registos inseridos e posteriormente alterados, para que a tabela volte a ter os dados originais.

Sintaxe utilizada para eliminar os registos (sem *index* criado):

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

De seguida foi testada a criação de um *index* para tentar melhorar a *performance* desta operação, sintaxe da criação do *index*:



Tempo de execução da *query*, depois da criação do *index*:

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

Neste caso os tempos de execução obtidos são diferentes com e sem *index*, curiosamente é um pouco mais rápido com *index* do que sem *index.* Mais uma vez, esta situação poderá estar relacionada com o volume de dados e com a adaptação do modelo ao *index*. Prevê-se que, com o aumento da informação alojada na base de dados, estes tempos de resposta poderão mudar e que a presença do *index,* nesta operação específica, poderá diminuir a *performance*.

# Considerações

Ao longo do processo de modelação dos três modelos desenvolvidos (*SQL Server, MongoDB* e *NEO4J*) foram sentidas algumas dificuldades, que levaram a alterações de estratégia, aprendizagem de ferramentas e linguagens novas, assim como obrigaram a um esforço adicional para a obtenção dos resultados pretendidos.

## Dificuldades encontradas

* **Seleção do *Dataset***

A escolha de um *dataset* com informação interessante, válida, atualizada e que cumprisse os requisitos de volume (centenas de milhares de linhas), foi a primeira dificuldade encontrada. O primeiro *dataset* escolhido (Vendas de Veículos Elétricos) não cumpria totalmente os requisitos, nomeadamente o volume de entidades era reduzido e existiam dados inconsistentes.

* **Modelação**

Em MongoDB e NEO4J, devido à falta de contacto do grupo com estes tipos de Bases de Dados, foi difícil definir uma estrutura de dados (baseado na desnormalizarão do modelo relacional para os modelos de *NoSQL*) que fosse capaz responder às necessidades do negócio. Como referido no capítulo 3, a modelação inicialmente definida resultava em tempos de execução bastante elevados, o que levou a perceber que seria necessário melhorar o modelo desenvolvido.

* ***Queries***

Enquanto em *SQL* foi relativamente simples desenvolver as *queries* necessárias, em *MongoDB* as *queries* de agregação representaram uma dificuldade adicional pois a sintaxe é muito própria e pouco intuitiva, também o relacionamento entre *collections* foi um desafio (entretanto ultrapassado).

Em *NEO4J*, a linguagem utilizada para a execução de *queries* é *cypher*, linguagem bastante poderosa e expressiva, mas também de aprendizagem complexa e um pouco lenta. O facto de normalmente não retornar erros de sintaxe representou uma grande dificuldade, *queries* que cruzam vários nós são bastante complexas, conversão de tipos de dados nem sempre funciona corretamente.

* **Importação de dados**

O modelo definido inicialmente em *MongoDB* (base de dados “BDA1\_Original”) utilizava *queries* de agregação para gerar *collections*. A execução destas *queries* apresentava um desempenho bastante aquém da espectativa.

Em *NEO4J*, também devido à utilização de *NEO4J Desktop* para correr localmente (de forma a ser comparável com os restantes modelos, que correm também localmente), a importação dos ficheiros *CSV* de FichasJogos (com 1.4 milhões de registos) retornava erro de memória, o que originou um bloqueio inicial neste processo. Para resolver esta situação foi necessário editar parâmetros nas *settings* do *DBMS*, nomeadamente os seguintes valores:

Uma imagem com texto, captura de ecrã, Tipo de letra, file

Descrição gerada automaticamente

Somente depois da alteração destes parâmetros foi possível executar as *queries* de importação (provavelmente outras *queries* de criação de relacionamentos e de operações de manipulação de dados, podem ter sido também “resolvidas” com estas alterações).

## Discussão de resultados

A elaboração deste trabalho foi um desafio bastante interessante para o grupo, principalmente devido à utilização de Bases de Dados em MongoDB e NEO4J, pois não eram de todo familiares aos elementos do grupo.

Neste capítulo serão abordados o processo de modelação e os resultados obtidos nos vários modelos de dados desenvolvidos.

* **Modelos de Dados**

No desenvolvimento dos modelos de dados apresentados para os três tipos de Bases de Dados, foram tidas em consideração as especificações do negócio, de forma que os modelos fossem capazes de dar resposta às necessidades identificadas.

Pode ser discutível se os modelos desenvolvidos são a única forma de estruturar a informação disponível, mas de acordo com os testes efetuados e a *performance* obtida na consulta de dados e nas operações de manipulação, conclui-se que estes modelos são capazes de dar as respostas que o negócio precisa (em todas as tipologias de Bases de Dados apresentadas).

* **Eficiência e Desempenho da Importação**

A importação de dados para *SQL Server*, depois da BD ser normalizada, decorreu de forma normal e relativamente rápida, também devido à experiência do grupo com a utilização deste tipo de Base de Dados.

Em relação a *MongoDB*, o problema detetado foi causado por uma definição errada da estruturação do Modelo, que levou a uma alteração de estratégia, conforme foi descrito no capítulo correspondente. Através do *Navicat* foi possível importar de forma rápida e simples a informação.

No caso do NEO4J, conforme foi descrito no ponto 5.1 foram sentidas dificuldades na importação de ficheiros CSV com elevado número de linhas, o que levou à necessidade de ajustar configurações, nos restantes ficheiros decorreu sem problemas. A importação foi efetuada utilizando NEO4J Desktop e foi possível através de *queries* utilizando linguagem *cypher.*

* **Tratamento de dados**

Ao analisar o *dataset* que foi utilizado na importação para *SQL* foram detetados alguns casos em que a informação não estava correta/completa. Foram detetados registos de eventos de jogo com jogadores a *null (*estes registos não foram considerados na importação*),* estádioscom capacidade a “0” (assumiu-se como capacidade o número de espetadores presentes no jogo com maior assistência) e jogadores sem registos de avaliação (neste caso não se procedeu a qualquer correção).

* **Análise de *Performance***

A correta estruturação dos Modelos é fundamental para a obtenção de uma boa *performance* na interação com a Base de Dados (independentemente da tipologia), mas em alguns casos é possível melhorar a *performance*. A criação de *indexes* foi fundamental na melhoria, conforme foi demonstrado em cada um dos capítulos correspondentes a cada Modelo de dados.

Em *SQL Server* aoanalisar a execução de uma *query,* através do *Execution Plan* é possível verificar se existe algum *index* identificado para criação, de forma a otimizar a execução da *query*. Isso simplificou bastante o processo de indexação e otimização das *queries* que se pretendia executar.

Mas em *MongoDB* e *NEO4J* não existe um utilitário semelhante, pelo que a identificação dos *indexes* a criar foi um desafio, obrigou a uma análise muito detalhada das *queries* e a uma correta identificação dos campos/propriedades utilizados como condições na *query,* depois de identificados procedeu-se à criação dos *indexes* e a nova medição dos tempos de execução.

* **Avaliação de Desempenho**

Como referido anteriormente, a estruturação dos Modelos desenvolvidos permitiu responder às necessidades do negócio. Cada um dos modelos apresentados tem as suas próprias características, vantagens/desvantagens e casos de uso onde a sua utilização é mais recomendada:

*- SQL Server,* excelente para dados estruturados e relações complexas entre diferentes conjuntos de dados;

- *MongoDB,* escalável e flexível principalmente para dados não estruturados, a sua flexibilidade permite uma fácil evolução/alteração do modelo;

- *NEO4J*, ideal para dados altamente “ligados”, onde as relações são tao importantes como os próprios dados. Bastante eficaz para responder a consultas que exploram padrões e relações complexas.

Ou seja, não é uma questão de escolher qual é o “melhor” ou o “pior”, mas sim de identificar qual é o negócio, o tipo de respostas que o modelo tem de ser capaz de responder e a eventuais necessidades específicas identificadas para o desenvolvimento do projeto/aplicação.

# Conclusão

A elaboração deste trabalho proporcionou, aos elementos do grupo, uma visão mais abrangente sobre a escolha e implementação de diferentes tipologias de Bases de Dados, modelo relacional e *NoSQL,* como *SQL Server*, *MongoDB* e *NEO4J.* Cada tipologia ofereceu abordagens distintas para a representação e manipulação de dados, demonstrando algumas vantagens e desvantagens em relação à informaçãodisponível no *dataset* original.

Cada um dos modelos demonstrou seu valor em diferentes aspetos: o *SQL Server* foi simples e eficaz para consultas analíticas e na agregação de dados, *MongoDB* destacou-se pela flexibilidade e escalabilidade, enquanto Neo4j demonstrou bastante potencial com flexibilidade de criar relações novas e consultas em profundidade.

Em última análise, a discussão dos resultados destaca a importância de alinhar a escolha do tipo de Base de Dados com os objetivos específicos do projeto. Cada abordagem oferece vantagens distintas, e a decisão deve ser baseada nas necessidades do negócio e nas respostas que a Base de Dados deve ser capaz de assegurar.

# Referências

# Contribuições

O desenvolvimento deste trabalho foi efetuado por todos os elementos do grupo, ao longo das aulas e através de sessões noturnas de *Teams* fomos discutindo possibilidades e debatendo ideias que iam surgindo.

Em termos de participação, todos os elementos contribuíram nas várias etapas do desenvolvimento. As ideias foram sempre discutidas em grupo, as possibilidades foram testadas e aprovadas/excluídas por todos.

# Anexos

<< 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.