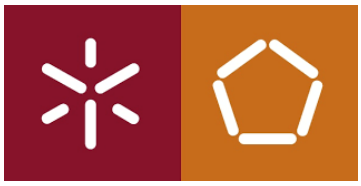

Universidade do Minho

Departamento de Informática

Trabalho Prático
Bases de dados NoSQL

Carlos Ferreira (PG47087)

Joel Martins (PG47347)

Júlio Alves (PG47390)

Nuno Silva (PG42645)

12 de junho de 2022

Conteúdo

Capítulo 1

Introdução

Este projeto foi realizado no âmbito da unidade curricular de Base de Dados *NoSQL*, de forma a consolidar a matéria lecionada nas aulas. Foi então desenvolvido um trabalho de análise, planeamento e implementação de um sistema de gestão de base de dados relacional disponibilizada pela equipa docente. O ficheiro fornecido tem o nome de **HR** e é disponibilizado no *Oracle Database*. **HR** é um esquema de uma aplicação de Recursos Humanos, criado pela Oracle, tem como objetivo armazenar os dados dos empregados de uma organização. Pretende-se explicar os processos de migração para outros sistemas não relacionais, de modo a maximizar cada um dos paradigmas. Além disso, existe um conjunto de *queries* feitos em cada base de dados, de forma a conseguir demonstrar a operacionalidade dos sistemas implementados. De forma a concluir o mesmo, foi pedido uma análise crítica do trabalho realizado, comparando, sempre que possível, os modelos e as funcionalidades implementadas, com as disponibilizadas.

Capítulo 2

Base de dados relacional - Oracle

Os dados fornecidos pelos docentes estão escritos em *query language*, por isso através do *software* ***DataGrip***, fizemos uma nova conexão **Oracle** e corremos as *queries* que nos criaram sete tabelas:

2.1 Esquema Lógico da Base de dados HR

- **HR.EMPLOYEES** - Esta tabela trata toda a informação, relativa aos funcionários da empresa, incluíndo o seu nome, *email* , salário, entre outros.
- **HR.JOBS** - Esta tabela trata de toda a informação relativa às funções de cada funcionário. Como por exemplo, o salário máximo e mínimo permitido para um trabalho.
- **HR.JOBHISTORY** - Esta tabela funcioná como um histórico de funções de cada funcionário, incluí a data que os colaborades iniciaram e terminaram as suas funções.
- **HR.DEPARTMENTS** - Contém todos os departamentos da empresa.
- **HR.LOCATIONS** - Contém todas localizações dos departamentos da empresa.
- **HR.COUNTRIES** - Contém todos os países onde a empresa opera.
- **HR.REGIONS** - Contém todas as regiões onde a empresa opera.

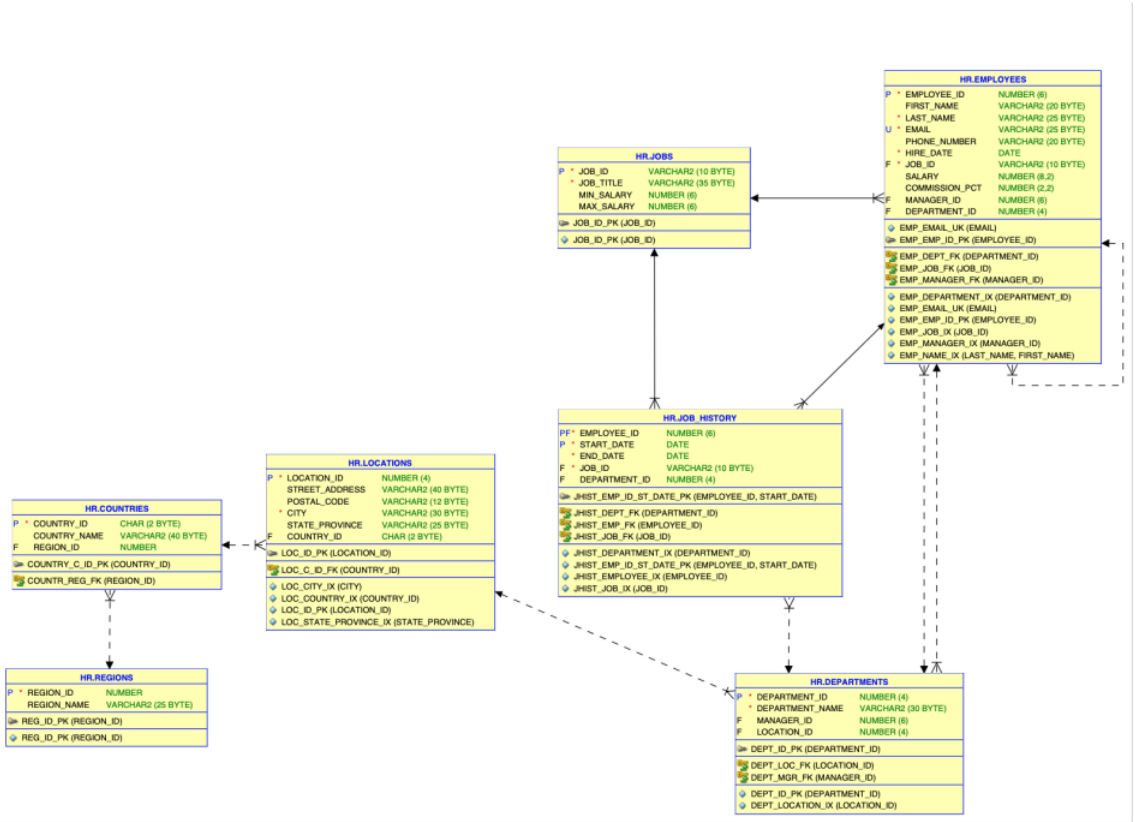

Figura 2.1: Esquema Lógico da base de dados HR.

Capítulo 3

Migração dos Dados para Base de Dados documental - MongoDB

De forma a migrar os dados das tabelas *SQL* do Oracle, usamos o software Studio 3T (antigo Robo3t). Este software facilitou no processo de migração, pois permitiu-nos gerar um único documento json, que contém toda a informação relevante para MongoDB.

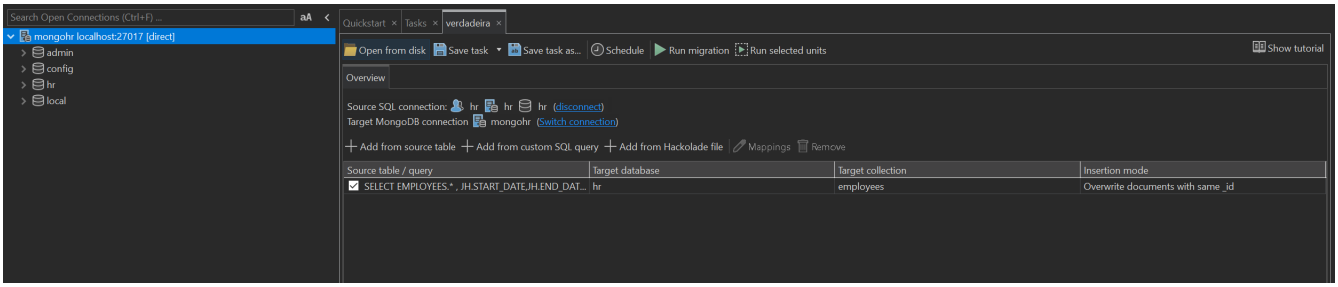

Figura 3.1: Migração para MongoDB no Studio3t.

Este processo de migração pode ser dividido em três fases, descritas seguidamente.

3.1 Definir ponto inicial de migração

[8]

O objetivo desta migração era juntar a informação relevante num único documento. Considerou-se que especificamente na tabela **JobHistory**, podiam ser removidas todas as colunas à exceção da ***Start Date*** , ***End Date*** e ***JOB ID***, visto que estas colunas se referem a funções antigas.

Para essa informação ficar associada a um *employee*, foi usado uma das funcionalidades da tarefa de migração chamada ***Add Source from Custom SQL query***, que permite buscar a informação às tabelas *SQL* através de uma *query*:

```
SELECT EMPLOYEES.* , JH.START_DATE,JH.END_DATE,JH.JOB_ID PREVJOB FROM EMPLOYEES
left join JOB_HISTORY JH on EMPLOYEES.EMPLOYEE_ID = JH.EMPLOYEE_ID
```

Usando o ***Left Join*** é selecionada toda a informação da tabela ***Employees*** e acresce-se as tabelas em cima mencionadas. Caso os *employees* não tenham funções antigas é atribuído o valor ***null*** nos respetivos campos.

Nesta parte do processo o documento *JSON* estava estruturado da seguinte forma:

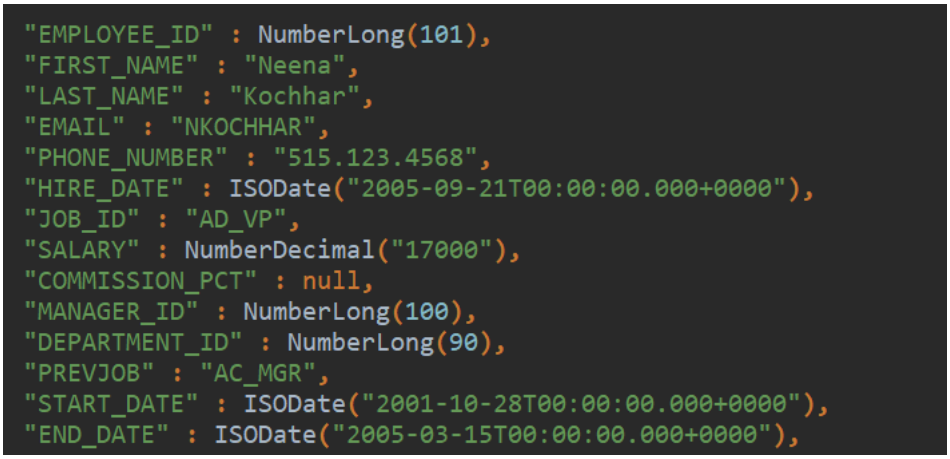

Figura 3.2: Estrutura Inicial do Documento.

Como se pode ver na imagem acima, que demonstra a informação de um funcionário da empresa, neste caso as datas *Start Date* e *End Date* são nulas. Caso o funcionário tivesse várias funções antigas, o funcionário aparecia duas vezes no documento com os campos respetivos a alterarem consoante a função, a data de ínicio da sua realização assim como a data de finalização da função.

3.2 Agregar informação através das chaves estrangeiras

Após findar a primeira etapa, usamos as chaves estrangeiras da tabela *employee* para ir agregando a informação de todas as outras tabelas.

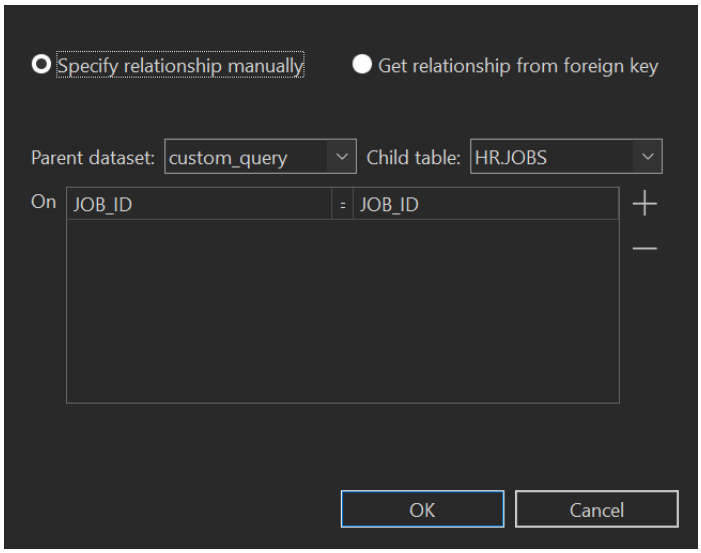

Figura 3.3: Agregar informação ao documento através das chaves estrangeiras.

Este processo foi repetido para as seguintes tabelas : ***JOBS***, ***DEPARTMENTS*** , ***LOCATIONS*** , ***COUNTRIES*** e ***REGIONS***.

A estrutura final do documento é a que se pode ver na figura abaixo:

Figura 3.4: Estrutura Final do Documento para criar a *database* no MONGODB

Como se pode observar, cada tabela foi dividida em *arrays* com o nome da respetiva tabela, e com toda a informação relativa à mesma.

3.3 Gerar *database* e coleção no MONGODB

É importante referir que devemos conectar o Studio 3t à nossa base de dados MongoDB de maneira a que quando executarmos a tarefa de migração , o software crie automaticamente a *database* e a coleção.

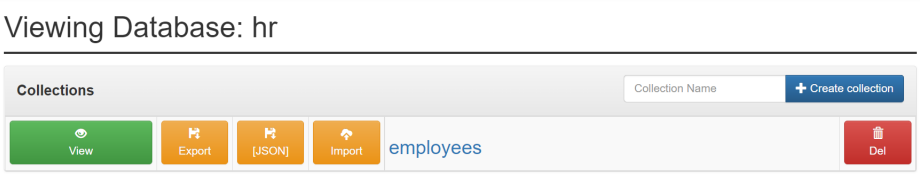

Figura 3.5: Base de dados e coleção criada através da migração

Capítulo 4

Migração de dados para Base de Dados de grafos - *Neo4j*

De forma a migrar os dados contidos na base de dados relacional para uma base de dados de grafos [7], neste caso o *NEO4J*, foram realizadas queries no Oracle que conjugassem campos relacionados entre si de várias tabelas, de forma a não só diminuir o número de nodos criados, como diminuir o número de ligações entre nodos. Desta forma, conseguimos obter informação mais facilmente e obtemos uma performance mais elevada do *Neo4j*, que sofre bastante com o aumentar de informação na base de dados. As queries efetuadas foram:

```
SELECT EMPLOYEES.EMPLOYEE_ID,EMPLOYEES.FIRST_NAME,Employees.LAST_NAME,EMPLOYEES.EMAIL,
EMPLOYEES.PHONE_NUMBER,EMPLOYEES.HIRE_DATE,EMPLOYEES.SALARY,EMPLOYEES.COMMISSION_PCT,
EMPLOYEES.MANAGER_ID,DEPARTMENTS.DEPARTMENT_ID,DEPARTMENTS.DEPARTMENT_NAME,
JOBS.JOB_TITLE
FROM EMPLOYEES
LEFT JOIN DEPARTMENTS  ON EMPLOYEES.DEPARTMENT_ID=DEPARTMENTS.DEPARTMENT_ID
LEFT JOIN JOBS ON EMPLOYEES.JOB_ID = JOBS.JOB_ID

SELECT JOB_HISTORY.EMPLOYEE_ID,JOB_HISTORY.START_DATE,JOB_HISTORY.END_DATE,
DEPARTMENTS.DEPARTMENT_ID,DEPARTMENTS.DEPARTMENT_NAME
FROM JOB_HISTORY
LEFT JOIN DEPARTMENTS ON JOB_HISTORY.DEPARTMENT_ID = DEPARTMENTS.DEPARTMENT_ID

SELECT * FROM DEPARTMENTS;

SELECT LOCATIONS.LOCATION_ID,LOCATIONS.STREET_ADDRESS,LOCATIONS.POSTAL_CODE,
LOCATIONS.CITY,LOCATIONS.STATE_PROVINCE,COUNTRIES.COUNTRY_NAME,REGIONS.REGION_NAME
FROM LOCATIONS
INNER JOIN COUNTRIES ON LOCATIONS.COUNTRY_ID = COUNTRIES.COUNTRY_ID
INNER JOIN REGIONS ON COUNTRIES.REGION_ID = REGIONS.REGION_ID
```

Para este fim, foram obtidos quatro ficheiros csv através do resultado das queries efetuadas no Oracle, sendo eles:

- Departamentos - Foi considerado que a tabela dos departamentos era relevante e não poderia ser conjugada com outra, pelo que este ficheiro representa unicamente a tabela

Departments.

- Job_History - Pelo mesmo motivo que a tabela de departamentos, este ficheiro corresponde à tabela de Job_History
- Employees - Nesta tabela foram utilizados os dados presentes na tabela dos Employees conjuntamente com a informação do trabalho desse mesmo employee, que está na tabela de Jobs, sendo que apenas consideramos o campo do Job_title e retiramos a chave estrangeira do Job_id de forma a reduzir o número de relações.
- Locations - Para esta tabela, conjugamos a informação presente em três tabelas, sendo elas Locations, Regions e Countries, utilizando a tabela de Locations como base e acrescentado os campos relativos ao nome do país e da região, diminuindo bastante as relações, uma vez que passamos de três nodos para apenas um.

Após a obtenção dos dados necessários, passamos para o *Neo4j* onde iremos dar uso aos mesmos. Para criar um nodo, utiliza-se o seguinte comando: [5]

```
    LOAD CSV WITH HEADERS FROM "file:///A1.csv" AS row
    CREATE (employee:EMPLOYEE {EMPLOYEE_ID: row.EMPLOYEE_ID})
    SET employee.FIRST_NAME = row.FIRST_NAME
    SET employee.LAST_NAME = row.LAST_NAME
    SET employee.EMAIL = row.EMAIL
    SET employee.PHONE_NUMBER = row.PHONE_NUMBER
    SET employee.HIRE_DATE = row.HIRE_DATE
    SET employee.SALARY = toInteger(row.SALARY)
    SET employee.COMMISSION_PCT = row.COMMISSION_PCT
    SET employee.DEPARTMENT_NAME = row.DEPARTMENT_NAME
    SET employee.JOB_TITLE = row.JOB_TITLE
    SET employee.FIRST_NAME = row.FIRST_NAME
RETURN employee;
```

Temos de ter em atenção dois pormenores ao criar um nodo que são, ao criar o nodo inicialmente, temos de ignorar as chaves estrangeiras até termos todas as tabelas criadas, uma vez que sem todas as tabelas criadas, essas chaves não se conseguirão relacionar com nada e o outro pormenor é que todos os dados são carregados para o *Neo4j* como string, pelo que caso tenhamos interesse em modificar o tipo de um campo, temos de o especificar, como no caso do salário apresentado em cima, em que é dado cast para integer.

Após termos todas as tabelas criadas, vamos então estabelecer as relações entre as mesmas

```
    LOAD CSV WITH HEADERS FROM 'file:///A1.csv' AS row
    MATCH (employee:EMPLOYEE {EMPLOYEE_ID: row.EMPLOYEE_ID})
    MATCH (manager:EMPLOYEE {EMPLOYEE_ID: row.MANAGER_ID})
    MATCH (department:DEPARTMENT{DEPARTMENT_ID:row.DEPARTMENT_ID})
    CREATE (employee)-[:SUPERVISED_BY]->(manager)
    CREATE (employee)-[:WORKS_AT]->(department)

    RETURN employee,manager,department;
```

As relações que estamos a estabelecer são entre Employee e Manager (que também é um employee) e entre Employee e Department, sendo que ao criar as ligações estamos a dizer que o employee é supervisionado pelo manager e que o employee trabalha no departamento.

No final da migração, obtemos a seguinte base de dados

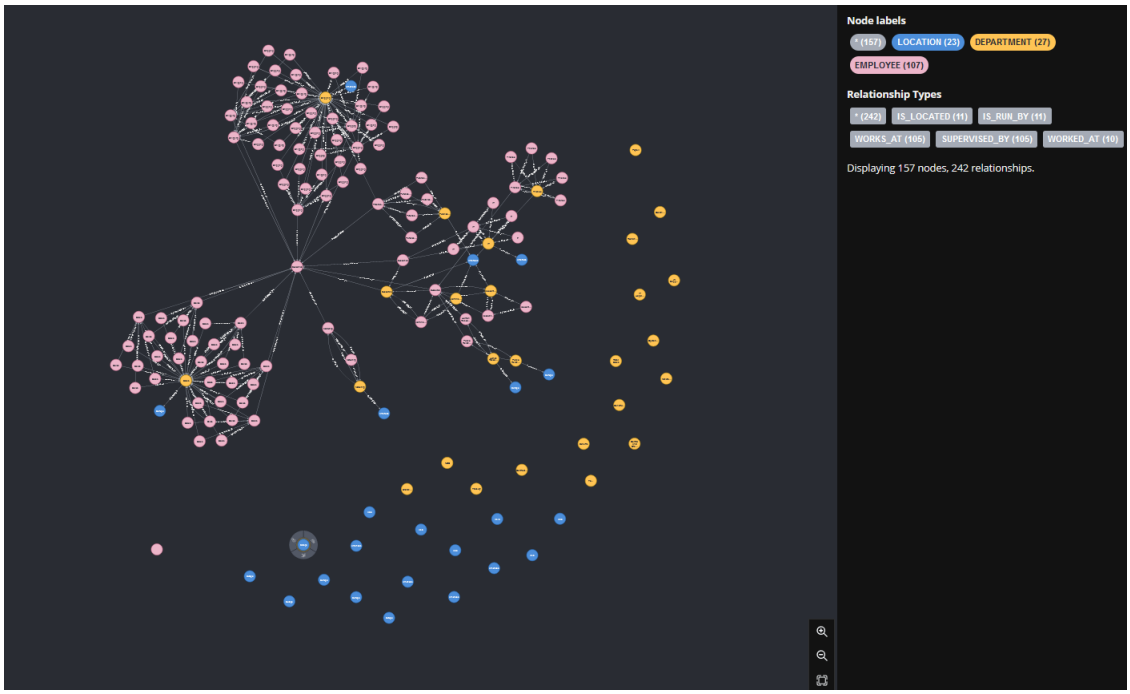

Figura 4.1: Grafo Neo4J

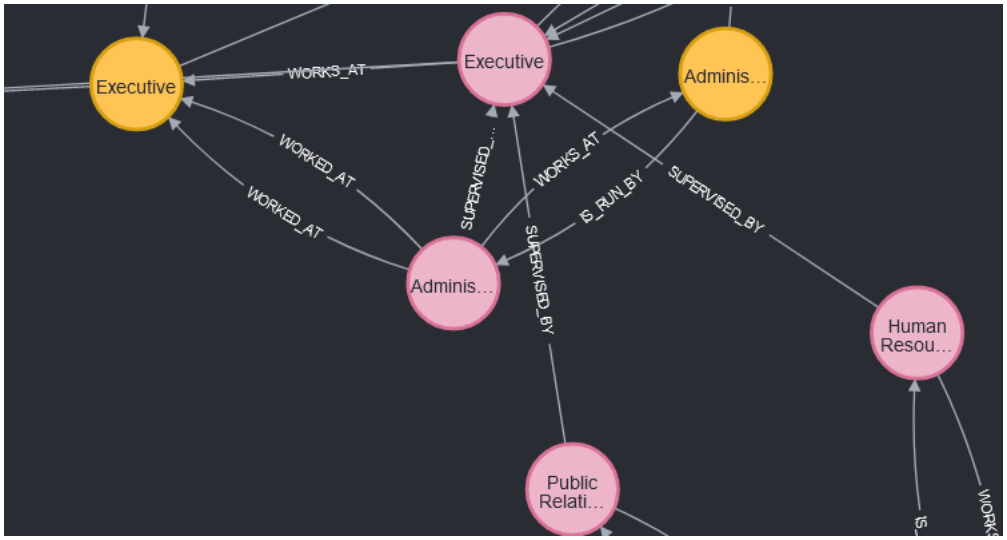

Figura 4.2: Relações entre nodos Neo4J

Para a tabela relativa à JOB_HISTORY foi utilizada uma abordagem diferente. Uma vez que a informação relativa a esta tabela não tinha qualquer utilidade se estivesse isolada como

um nodo, foi criada uma relação entre os nodos dos Employees e o nodo dos Departments que continha a informação relativa ao histórico de trabalho de um employee. Desta forma, se um dado employee trabalhou num dado departamento, existirá uma relação denominada "WORKED_AT"entre os dois nodos em questão. Essa relação foi criada da seguinte forma:

```
LOAD CSV WITH HEADERS FROM 'file:///A2.csv' AS row
MATCH (employee:EMPLOYEE {EMPLOYEE_ID: row.EMPLOYEE_ID})
MATCH (department:DEPARTMENT {DEPARTMENT_ID: row.DEPARTMENT_ID})
CREATE (employee)-[r:WORKED_AT {employee_id:row.EMPLOYEE_ID,start_date:date(row.START_DAT
RETURN employee,department;
```

[6]

Capítulo 5

Métricas de avaliação

5.1 *Queries*

Foi decidido começar por definir as *queries* em *query language*, esta escolha tem por base a experiência dos elementos do grupo em trabalhar com esta linguagem. As *queries* definidas foram as seguintes:

- **Quantos funcionários existem?**
- **Quantos postos de trabalho diferentes existem?**
- **Que postos de trabalho existem?**
- **Quantos departamentos diferentes existem?**
- **Que departamentos existem?**
- **Qual é o salário médio de um funcionário?**
- **Em quantos países estão os departamentos da empresa?**
- **Funcionários que estão há mais tempo na empresa?**
- **Que funcionários ganham o máximo para a sua função e qual a sua função?**

5.2 *Tempo de Execução*

De forma a haver uma comparação de execução nos diferentes softwares, tabelou-se os resultados de execução de cada *query* nos diferentes motores de base de dados. Vale a pena referir que existem variáveis externas que podem prejudicar ou melhorar este tempo, como por exemplo o hardware.

Query	Oracle	MongoDB	NEO4J
Q1	97ms	545ms	1ms
Q2	148ms	473ms	15ms
Q3	107ms	760ms	1ms
Q4	74ms	740ms	6ms
Q5	81ms	495ms	1ms
Q6	55ms	379ms	1ms
Q7	36ms	505ms	11ms
Q8	82ms	474ms	2ms
Q9	229ms	371ms	N/A

Como se pode ver na tabela acima, os tempos de execução dos comandos em MongoDB foram os maiores. Um fator que pode afetar estes tempos pode ser a composição do documento, como a informação está toda agregada num único documento , a pesquisa pode demorar. Algo que não se verifica no *SQL* que como é um modelo relacional tem a informação separada, sendo mais rápido o seu acesso.

Podemos constatar que o Neo4j é o mais rápido, por exemplo, em comparação com base de dados relacionais, em relação a dados conectados, o que tem como consequência que a latência das queries numa base de dados de grafos é proporcional a quanto do grafo vamos explorar numa query e não é proporcional à quantidade de dados armazenados. [1]

Capítulo 6

Resultados

6.1 *Queries*

[4]

```
//Quantos funcionários existem?

ORACLE - SELECT COUNT(*) from EMPLOYEES;
MONGO - db.employees.distinct('EMPLOYEE_ID').length
NEO4J - match (e:EMPLOYEE) with count (distinct e.EMPLOYEE_ID) as total
        return total

//Quantos postos de trabalho diferentes existem?

ORACLE - SELECT COUNT(*) from JOBS;
MONGO - db.employees.distinct("JOBS.JOB_TITLE").length
NEO4J - match (e:EMPLOYEE) with count (distinct e.JOB_TITLE) as total
        return total

//Que postos de trabalho existem?

ORACLE - SELECT * from JOBS;
MONGO - db.employees.distinct("JOBS.JOB_TITLE")
NEO4J - match (e:EMPLOYEE) return distinct(e.JOB_TITLE)

//Quantos departamentos diferentes existem?

ORACLE - SELECT COUNT( DISTINCT DEPARTMENT_ID) FROM EMPLOYEES;
MONGO - db.employees.distinct("DEPARTMENTS.DEPARTMENT_ID").length
NEO4J - match (d:DEPARTMENT) with count (distinct d.DEPARTMENT_ID) as total
        return total

//Que departamentos existem?

ORACLE - SELECT DEPARTMENTS.DEPARTMENT_NAME from DEPARTMENTS;
```

```
MONGO - db.employees.distinct('DEPARTMENTS.DEPARTMENT_NAME');
NEO4J - match (d:DEPARTMENT) return distinct (d.DEPARTMENT_NAME)

// Qual é o salário médio de um funcionário?

ORACLE - select round(avg(SALARY),2) from EMPLOYEES;
MONGO - db.employees.aggregate([{$group:{"_id":"_id",
        ValorMedio: { $avg: "$SALARY" }}}])
NEO4J - match(e:EMPLOYEE) return avg(e.SALARY)

// em quantos paises estão os departamentos da empresa?

ORACLE - SELECT COUNT(DISTINCT countries.COUNTRY_ID) FROM COUNTRIES
INNER JOIN LOCATIONS L on COUNTRIES.COUNTRY_ID = L.COUNTRY_ID
INNER JOIN DEPARTMENTS D on L.LOCATION_ID = D.LOCATION_ID
INNER JOIN EMPLOYEES E on D.DEPARTMENT_ID = E.DEPARTMENT_ID

MONGO - db.employees.distinct("COUNTRIES.COUNTRY_ID").length
NEO4J - MATCH (d:DEPARTMENT)-[:IS_LOCATED]->(l:LOCATION)
        WITH l.COUNTRY_NAME as country,count(l) as total
        return country,total

//Quais os paises que trabalham os funcionarios da empresa?

ORACLE -SELECT DISTINCT COUNTRIES.COUNTRY_NAME FROM COUNTRIES
INNER JOIN LOCATIONS L on COUNTRIES.COUNTRY_ID = L.COUNTRY_ID
INNER JOIN DEPARTMENTS D on L.LOCATION_ID = D.LOCATION_ID
INNER JOIN EMPLOYEES E on D.DEPARTMENT_ID = E.DEPARTMENT_ID

MONGO -db.employees.distinct("COUNTRIES.COUNTRY_NAME")

NEO4J - MATCH (e:EMPLOYEE)-[:WORKS_AT]->(d:DEPARTMENT)-[:IS_LOCATED]->(l:LOCA
        WITH l.COUNTRY_NAME as country,count(l) as total
        return country,total

// funcionário que estão há mais tempo na empresa?

ORACLE - select FIRST_NAME, LAST_NAME,max(HIRE_DATE) tempo
        from EMPLOYEES LEFT JOIN JOB_HISTORY JH
        on EMPLOYEES.EMPLOYEE_ID = JH.EMPLOYEE_ID
where END_DATE IS NULL
group by FIRST_NAME,LAST_NAME
order by tempo desc;

MONGO-db.employees.find({END_DATE:null}).sort({"HIRE_DATE":1}).limit(5);

NEO4J - Não fomos capazes de realizar esta query pois, como era necessário ut
```

Capítulo 7

Discussão de Resultados

A migração dos dados entre as diferentes tecnologias tem de ser adaptada ao respetivo esquema e às suas caracteristicas. O uso do Studio 3T simplificou muito das nossas tarefas, porque permitiu fazer este tipo de processos de forma integrada. Inicialmente não tínhamos feito desta forma, decidiu-se gerar ficheiros *JSON* e depois através de um *script* desenvolvido em *python* criar as coleções no MONGODB.

Com a migração da base de dados relacional para uma base de dados de grafos, constatamos que o motor da base de dados para Neo4j se tornou lento, fazendo com que por vezes fosse difícil trabalhar no mesmo, o que nos leva a crer que, com um volume de dados maior, se torne impraticável a utilização da versão Community do Neo4j. Relativamente à execução de queries, e também devido ao "esquema"que escolhemos, constatamos que as mesmas também eram simples de fazer. No entanto, quando quisemos fazer queries que dependiam de dados presentes nas relações entre nodos, as mesmas tornaram-se bastante mais dificeis.

Capítulo 8

Conclusão

Cada tecnologia de base de dados tem as suas próprias características , o utilizador deve ter isso em conta no momento de escolher qual o motor de base de dados a usar. O Oracle, é um sistema de base de dados semelhante ao *SQL Server*, *MySql*, que segue o modelo relacional. Enquanto que o Mongo é uma boa opção para consultar estruturas de um documento relativamente pequeno. Já o Neo4j tem melhor performance em pesquisas relacionais.

[2] [3]

Bibliografia

[1] URL: https://stackoverflow.com/questions/13046442/comparison-of-relational-databases-and-graph-databases.

[2] Geeks for Geeks. *MongoDB: An introduction*. URL: https://www.geeksforgeeks.org/mongodb-an-introduction/.

[3] Geeks for Geeks. *Neo4j Introduction*. URL: https://www.geeksforgeeks.org/neo4j-introduction/.

[4] MongoDB. *MongoDb Documentação*. URL: https://www.mongodb.com/docs/.

[5] neo4j. *Importing CSV Data into Neo4j*. URL: https://neo4j.com/developer/guide-import-csv/.

[6] neo4j. *Neo4j documentation*. URL: https://neo4j.com/docs/.

[7] neo4j. *Tutorial: Import Relational Data Into Neo4j*. URL: https://neo4j.com/developer/guide-importing-data-and-etl/.

[8] Kathryn Vargas. *How To Merge Multiple SQL Tables Into One MongoDB Collection*. URL: https://studio3t.com/knowledge-base/articles/merge-multiple-sql-tables-to-mongodb/#treating-duplicate-fields.