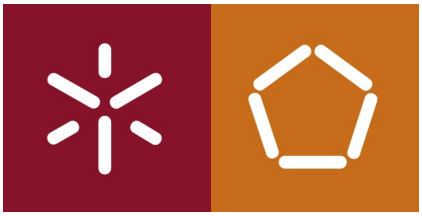

Universidade do Minho
Mestrado Integrado em Engenharia Informática
4ºano - 1º Semestre

Base de Dados NoSQL

Trabalho Prático

Grupo 01




a83899 – André Loureiro Morais
a76089 - Etienne da Silva Filipe Amado da Costa
a83732 – Gonçalo Rodrigues Pinto
a85954 – Luís Mário Macedo Ribeiro

29 de Janeiro de 2021

Conteúdo

1 Introdução **2**

2 Caraterização das Bases de Dados **3**
2.1 Bases de Dados Relacionais 3
2.2 Bases de Dados NoSQL 4
2.2.1 Bases de dados orientadas a documentos 4
2.2.2 Bases de dados orientadas a grafos 4

3 Contextualização **6**

4 Descrição do Problema **6**

5 Migração dos dados **7**
5.1 Base de dados orientado a documentos 7
5.2 Base de dados orientado a grafos 10

6 Operacionalidade das Bases de Dados **13**
6.1 Base de Dados Relacional 13
6.2 Base de Dados NoSQL 18
6.2.1 Base de dados orientada a documentos 18
6.2.2 Base de dados orientada a grafos 27

7 Comparação da Operacionalidade **32**

8 Análise Crítica **38**
8.1 BD Relacional *vs.* BD orientada a documentos 38
8.2 BD Relacional *vs.* BD orientada a grafos 39
8.3 BD orientada a documentos *vs.* BD orientada a grafos 40

9 Conclusão **42**

1 Introdução

No 1º semestre do 1º ano do Mestrado em Engenharia Informática da Universidade do Minho, existe uma unidade curricular complementar denominada por Bases de Dados NoSQL, que tem como objetivo a introdução aos sistemas de bases de dados de última geração, em particular no que diz respeito à sua administração e exploração.

O presente trabalho enquadra-se nesta unidade curricular e pretende desenvolver competências aos alunos na utilização de diferentes paradigmas de bases de dados e a sua aplicação na conceção e implementação de sistemas.

Neste trabalho pretende-se que cada grupo realizasse um trabalho de análise, planeamento, de implementação de um sistema de gestão de base de dados relacional (SGBD) e de dois sistemas não relacionais. Para esse fim, escolheu-se para o SGBD relacional o *Oracle SQL Developer* uma vez este, estar previamente instalado nas nossas máquinas, além disso, ao longo do semestre foi várias vezes explorado noutros contextos. Quanto aos sistemas de bases de dados não relacionais, foi requerido a utilização de uma base de dados orientada a documentos, para tal optou-se pela utilização do *MongoDB* dado que, já nos encontramos muito familiarizados com esta ferramenta. Além disso, foi requerido o uso de outro modelo não relacional, sendo este orientado a grafos, desse modo a escolha recaiu pela utilização do *NEO4J*, visto que, previamente tínhamos efetuado a instalação deste sistema, para explorar outros conceitos lecionados ao longo do semestre desta unidade curricular.

2 Caraterização das Bases de Dados

Para realizar a análise, planeamento, e implementação foi necessário efetuar um levantamento de características e estudo das mesmas, com o objetivo de perceber as diferenças entre cada modelo a utilizar, seguidamente apresentamos o estudo efetuado.

2.1 Bases de Dados Relacionais

No modelo relacional, todos os dados são logicamente estruturados dentro das relações (tabelas). Cada relação tem um nome e é composta por atributos (colunas) de dados, também identificados por um nome. Cada tuplo (linha) contém um valor por atributo.

Numa base de dados relacional, todos os dados, incluindo o próprio dicionário de dados, são representados de uma só forma em tabelas bidimensionais. Cada elemento de dados fica bem determinado pela combinação do nome da tabela onde está armazenado, o valor da chave primária e respetivo atributo. Os valores nulos são suportados para representar informação não disponível ou não aplicável, independentemente do domínio dos respetivos atributos. Apesar de um sistema relacional poder suportar várias linguagens, este suporta no mínimo manipulação de dados, definição de dados, definição de *views*, definição de restrições de integridade, definição de acessos e manipulação de transações. Permite a capacidade de tratar uma tabela como se fosse um simples operando, tanto em operações de consulta como de atualização.

Entre as principais vantagens do modelo relacional podemos referir a independência total dos dados, visão múltipla dos dados, maior segurança no acesso aos dados, maior agilidade para consulta/atualização e qualidade dos dados garantida por restrições de integridade.

Em síntese, o sistema de gestão de bases de dados utilizado, ***Oracle***, os seus dados são guardados em tabelas, que se encontram relacionadas com outras tabelas através de chaves estrangeiras e de tabelas intermédias.

2.2 Bases de Dados NoSQL

O termo **NoSQL** é muito amplo dado que não se refere a um modelo específico de bases de dados; refere-se a toda uma panóplia de diferentes modelos que têm como principal característica não se encaixarem no modelo relacional referido acima. Uma das principais razões pelas quais a abordagem NoSQL começou a ser adotada ocorreu devido ao *big data*.

Contudo as bases de dados NoSQL partilham algumas características, tais como, não seguir o modelo relacional acima descrito, de código aberto, sem esquema, escaláveis horizontalmente, não seguem os princípios *ACID* (Atomicidade, Consistência, Isolamento, Durabilidade) e sem uma linguagem de consulta padrão.

2.2.1 Bases de dados orientadas a documentos

As bases de dados documentais armazenam cada registo e todos os dados associados num único documento. Cada documento é composto por dados semiestruturados, dados esses que podem ser consultados usando várias ferramentas de consulta e análise do SGBD.

Nas bases de dados documentais os dados são armazenados como um valor que, por sua vez, é um documento e a chave associada é um identificador exclusivo para esse valor.

O sistema de gestão de bases de dados orientado a documentos utilizado, ***MongoDB***, foi concebido especialmente para os dados que podem ser representados num único documento, não sendo adequado para dados relacionados. Desta forma, o *MongoDB* é mais apropriado para realizar consultas.

2.2.2 Bases de dados orientadas a grafos

Uma base de dados orientada a grafos é uma base de dados que usa um modelo com recurso a grafos para representar e armazenar os dados sendo uma alternativa ao modelo relacional. Numa base de dados orientada a grafos, não há esquema predefinido, qualquer esquema é simplesmente um reflexo dos dados que foram inseridos, à medida que dados mais variados são inseridos, o esquema acompanha esse crescimento.

Esse esquema é composto pelos seguintes componentes:

- Nodos são as unidades fundamentais e podem conter propriedades, como pares de valores-chave;
- Relacionamentos são componentes importantes neste tipo de base de dados, cada relacionamento contém um nodo inicial e um nodo final;
- Propriedades são pares de valores-chave para descrever tanto os nodos como os relacionamentos;
- Etiquetas são rótulos que associam um nome comum a um conjunto de nodos ou relacionamentos, um nodo ou relacionamento pode conter um ou mais rótulos, é possível criar novos rótulos para nodos ou para relacionamentos já existentes como também remover já existentes.

As bases de dados com recurso a grafos são criadas e geridas por um sistema de gestão de bases de dados, onde algumas utilizam mecanismos de armazenamento relacionais, contudo, muitos sistemas de bases de dados NoSQL usam *tags* ou propriedades para definir relacionamentos entre nodos.

O sistema de gestão de bases de dados orientado a grafos utilizado, ***Neo4J***, é vocacionado para a representação de relacionamentos que existem entre os dados, funcionando bastante bem quando existem vários relacionamentos entre várias entidades, ao contrário de grande parte dos restantes sistemas de bases de dados. Como tal, as *queries* que melhor demonstram a operacionalidade deste sistema são as alusivas a pesquisas relacionais.

3 Contextualização

Neste trabalho foi nos facultado uma base de dados relacional denominada **HR** que é disponibilizada no *Oracle Database*, para a sua criação foi disponibilizado um *script* que permitiu a criação dos *tablespaces*, unidade fundamental numa base de dados da *Oracle*, permanentes e temporários, bem como, possibilitou a definição de um utilizador com respetivos acessos.

Posteriormente, criou-se uma conexão com o utilizador criado com a palavra passe definida na *pluggable database* onde nesta se executou um outro *script* também disponibilizado que permitiu a criação das tabelas e respetivo povoamento das mesmas.

4 Descrição do Problema

O esquema criado **HR** é uma aplicação de Recursos Humanos, criado pela *Oracle*, que tem como objetivo principal armazenar os dados dos empregados de uma organização.

O **HR** possui 7 tabelas:

- EMPLOYEES: Dados dos empregados, tais como: nome, departamento e cargo atual. Os empregados podem ou não estar vinculados a um departamento. Existe ainda funcionários que não possuem comissão, bem como, o presidente da empresa não possui gestor;
- DEPARTAMENTS: Dados dos departamentos em que empregados podem trabalhar. Os departamentos podem ou não ter um gestor;
- REGIONS: Dados sobre as regiões em que a organização pode atuar;
- LOCATIONS: Dados sobre os locais ou endereços dos escritórios, armazéns ou locais de produção da organização. Os locais podem não possuir província bem como não possuir qualquer departamento;
- COUNTRIES: Dados sobre os países em que a organização atua. Os países podem não ter locais associados;
- JOBS: Dados sobre os tipos de cargos que os empregados podem ocupar;
- JOB_HISTORY: Histórico dos cargos anteriores ocupados pelos empregados dentro da organização.

5 Migração dos dados

Após compreendido as particularidades do esquema criado, tal como, as características dos dados presentes, foi possível efetuar a migração da base de dados relacional para os outros modelos de bases de dados NoSQL.

Para esse efeito o grupo decidiu criar programas na linguagem de programação Java de forma a efetuar a migração, dado já estarmos familiarizados com esta linguagem, além de que, fornece diversas ferramentas como estruturas de dados e ainda permite facilmente efetuar a conexão às bases de dados da *Oracle*.

5.1 Base de dados orientado a documentos

Antes da migração para este sistema foi necessário definir a unidade fundamental que iria ser representada no documento.

Visto que o esquema criado é uma aplicação de Recursos Humanos, o grupo definiu que iria armazenar no documento uma lista de valores, em que cada valor representa um determinado funcionário, este valor é composto pelos atributos presentes na tabela "EMPLOYEES", assim o conjunto de chave / valor são os atributos de um determinado funcionário. Além disso, para representar as relações dos funcionários criou-se uma chave denominada "job"cujo valor é composto pelos atributos do cargo que ocupa e ainda uma outra chave designada "department"cujo valor associado é composto pelos diferentes atributos do departamento, pelo nome do gestor deste departamento, localização deste, país e região respetivamente. Criou-se ainda uma lista de valores associado ao funcionário que representa um cargo que desempenhou no passado.

Deste modo, a informação que não se encontra relacionada com um funcionário não é apresentada no documento criado pois de forma a preservar essa informação era necessário criar outros documentos, criação essa que tornaria o sistema não orientado a documentos como foi previamente caraterizado.

De seguida apresentamos um esboço de um valor que representa um funcionário, é de realçar que quando este não possui um determinado atributo ou relação esse não é descrito.

```
{   "_id": "",
    "first_name": "",
    "last_name": "",
    "email": "",
    "phone_number": "",
    "hire_date": "",
    "salary": "",
    "commission_pct": "",
    "job": {
        "title": "",
        "min_salary": "",
        "max_salary": ""
    },
    "department": {
        "department_id":"",
        "name": "",
    "manager_id": "",
        "manager": "",
        "street_address": "",
        "postal_code": "",
        "city": "",
        "state_province": "",
        "country_name": "",
        "region_name": ""
    },
    "history": [{
            "start_date": "",
            "end_date": "",
            "job_title": "",
            "job_min_salary": "",
            "job_max_salary": "",
            "department_name": "",
            "department_manager": "",
            "street_address": "",
            "postal_code": "",
            "city": "",
            "state_province": "",
            "country_name": "",
            "region_name": ""
        }]
}
```

Seguidamente, migrou-se os dados do modelo relacional para o *MongoDB*, para esse efeito efetuou-se uma conexão à base de dados da *Oracle* com o objetivo de ler a informação previamente inserida e a escrever num simples ficheiro JSON com o formato acima apresentado, para posteriormente se efetuar o carregamento deste ficheiro para o sistema escolhido, para tal foi necessário realizar as seguintes etapas:

Etapa 1 : Efetuar a conexão à *PDB* através do driver *oracle.jdbc.OracleDriver* com o utilizador criado previamente;

Etapa 2 : Criação de classes com respetivos atributos idênticos ao modelo relacional (por exemplo, a classe COUNTRY possui como atributos o id, o nome e a classe REGION respetiva), como também, coleções para armazenar estas mesmas classes;

Etapa 3 : Leitura de todas as tabelas a fim de povoar as coleções respetivas, para tal a leitura respeitou uma determinada ordem, de forma a que se uma classe dependesse de outra a que não depende é acedida primeiro, assim conseguiu-se povoar os atributos da classe que depende com o objeto respetivo (através do acesso às coleções devidas);

Etapa 4 : Criação do ficheiro JSON onde se escreve a lista de valores;

Etapa 5 : Iteração da coleção que armazenou os funcionários efetuando o conjunto de validações, de forma a verificar quais os atributos nulos, assim apenas são escritos os valores com significado na sintaxe devida. Para a representação das relações tirou-se proveito da estrutura de cada classe, no sentido de aceder ao objeto ao qual existe a relação (por exemplo, para descrever o cargo de um funcionário, basta aceder ao atributo JOB contido na classe EMPLOYEE).

Após executado o programa é então gerado o ficheiro que com auxílio do seguinte comando permitiu importar para a base de dados utilizada.

```
mongoimport --db BDNSQL
            --collection hr
            --drop
            --file scriptPovoamentoMongoDB.json
            --jsonArray
```

5.2 Base de dados orientado a grafos

Antes de efetuar a migração para o sistema orientado a grafos foi necessário identificar quais os dados que se tornam unidades fundamentais neste sistema, nodos, como também as suas relações e propriedades.

Desta forma, considerou-se como nodos:

- EMPLOYEE pois dado que é uma aplicação de Recursos Humanos é lógico que exista uma identidade que represente funcionários. As suas propriedades são o id, first_name, last_name, email, phone_number, hire_date, salary e aqueles que tem associado uma commission_pct;
- DEPARTMENT pelo mesmo motivo de cima como é aplicação de Recursos Humanos é coerente existir um local onde os funcionários trabalhem. As propriedades deste nodo são o id e name;
- JOB é apresentado como nodo pois é uma identidade relevante na área de Recursos Humanos de ser individualizada. As propriedades deste são o id, title, min_salary, max_salary.
- LOCATION também é apresentado como nodo pois existe localizações sem departamento e ao ser incluída noutros nodos perdia-se informação na migração. As propriedades deste tipo de nodo são o id, street_address, city, state_province e postal_code estes últimos apenas aos nodos que possuem.
- COUNTRY para garantir que informação não se perca, as suas propriedades são id, country_name e region_name;

Apresentados os nodos considerados criou-se os seguintes relacionamentos:

- CONTAINED_IN de LOCATION para COUNTRY, representa que uma determinada localização está contida num determinado país, não possui propriedades;
- HAVE de DEPARTMENT para LOCATION demonstra que um departamento possui uma localização, não possui propriedades;
- MANAGED_BY de DEPARTMENT para EMPLOYEE traduz qual o funcionário responsável pelo departamento, não possui propriedades;
- WORK_AT de EMPLOYEE para DEPARTMENT representa que o funcionário trabalha num departamento, não possui propriedades;
- DOES de EMPLOYEE para JOB demonstra qual o cargo que o funcionário desempenha, não possui propriedades;
- IS_MANAGED_BY de EMPLOYEE para EMPLOYEE traduz a hierarquia na empresa, não possui propriedades;
- DID de EMPLOYEE para JOB e IN de JOB para DEPARTMENT representa que o funcionário já efetuou um determinado cargo num determinado departamento, o primeiro relacionamento possui como propriedades a start_date e end_date e o segundo não possui propriedades;

Seguidamente, migrou-se os dados do modelo relacional para o *NEO4J*, para esse efeito efetuou-se uma conexão à base de dados da *Oracle* com o objetivo de ler a informação previamente inserida e a escrever num simples ficheiro de texto com a sintaxe que permite a criação dos nodos e relacionamentos apresentados, para posteriormente se efetuar o carregamento deste ficheiro para o sistema escolhido, para tal foi necessário realizar as seguintes etapas:

Etapa 1 : Efetuar a conexão à *PDB* através do driver *oracle.jdbc.OracleDriver* com o utilizador criado previamente;

Etapa 2 : Criação de classes com respetivos atributos idênticos ao modelo relacional (por exemplo, uma classe COUNTRY cujo atributos são o id, o nome e a classe REGION respetiva) como também coleções para armazenar estas mesmas classes;

Etapa 3 : Leitura de todas as tabelas a fim de povoar as coleções respetivas, para tal a leitura respeitou uma determinada ordem, de forma a que se uma classe dependesse de outra a que não depende é consultada primeiro, assim conseguiu-se povoar os atributos da classe que depende com o objeto respetivo (através do acesso às coleções devidas);

Etapa 4 : Criação do ficheiro onde se escreve os nodos, propriedades e relacionamentos;

Etapa 5 : Iteração das coleções que vão representar nodos, efetuando a escrita na sintaxe devida (exemplo, "CREATE (DE: COUNTRY { id:'DE', country_name:'Germany', region_name:'Europe'})"), para esse efeito utilizou-se etiquetas de forma a identificar os nodos;

Etapa 6 : Iteração novamente das coleções de forma a escrever os relacionamentos entre nodos (exemplo, "CREATE (l1600)-[:CONTAINED_IN {}]->(US)") recorrendo para isso às etiquetas previamente escritas.

Após executado o programa é então gerado o ficheiro que permitiu a criação da base de dados orientada a grafos com as características acima referidas.

6 Operacionalidade das Bases de Dados

De forma a demonstrar a operacionalidade dos sistemas de gestão de bases de dados foi implementado um conjunto de consultas selecionadas pelo grupo, que por um lado, demonstra a utilidade das mesmas no processo de implementação de uma futura aplicação (por exemplo, a apresentação dos países onde os funcionários trabalhem), por outro lado pretende salientar as principais diferenças entre cada sistema para a implementação da mesma consulta.

6.1 Base de Dados Relacional

A linguagem utilizada neste modelo foi o *SQL* que é a linguagem padrão para trabalhar com bases de dados relacionais, utilizando principalmente a componente designada de *"Data Query Language"*, DQL, foi possível manipular os dados de forma a responder às consultas definidas, o principal comando desta componente e utilizado foi o SELECT.

Assim, através dos comandos da componente DQL foi possível realizar as seguintes consultas:

1. **Apresentar os detalhes dos departamentos ordenados pelo nome de forma ascendente:**

```
SELECT * FROM departments d ORDER BY d.department_name ASC ;
```

2. **Apresentar os nomes dos departamentos e os responsáveis por estes:**

```
SELECT d.department_name, e.first_name, e.last_name
    FROM departments d, employees e
    WHERE d.manager_id=e.employee_id ;
```

3. **Apresentar os nomes dos departamentos e o número de funcionários de cada um:**

```
SELECT department_name, COUNT(*) AS tot_employees
    FROM employees e, departments d
    WHERE e.department_id=d.department_id
    GROUP BY department_name;
```

4. **Apresentar os identificadores dos gestores e o número de funcionários geridos por cada um:**

```
SELECT manager_id, COUNT(*) AS tot_employees
    FROM employees
    GROUP BY manager_id ;
```

5. **Apresentar os detalhes dos funcionários cuja percentagem de comissão seja nula, o salário esteja entre 10000 e 15000 e o gestor possua o identificador 101:**

```
SELECT *
    FROM employees
    WHERE commission_pct IS NULL AND salary>=10000
        AND salary<=15000 AND manager_id=101;
```

6. **Apresentar os funcionários cujo primeiro nome e último nome começam pela letra G:**

```
SELECT first_name, last_name
    FROM employees
    WHERE SUBSTR(first_name,0,1)='G' AND SUBSTR(last_name,0,1)='G';
```

7. **Apresentar todos os funcionários que entraram para a empresa após 19 de Dezembro de 2007:**

```
SELECT * FROM employees e
    WHERE e.hire_date > to_date('19-12-2007','DD-MM-YYYY');
```

8. **Apresentar o nome dos funcionários que foram contratados em 2008:**

```
SELECT first_name, last_name, hire_date
    FROM employees
    WHERE EXTRACT(YEAR FROM hire_date)=2008
    ORDER BY hire_date;
```

9. **Apresentar quantos funcionários foram admitidos em cada mês do ano 2008:**

```
SELECT COUNT(*) as tot, EXTRACT(MONTH FROM hire_date) AS MONTH
    FROM employees
    WHERE EXTRACT(YEAR FROM hire_date) = 2008
    GROUP BY (EXTRACT(MONTH FROM hire_date))
    ORDER BY EXTRACT(MONTH FROM hire_date) ASC;
```

10. **Apresentar o nome dos funcionários que iniciaram funções antes do seu gestor:**

```
SELECT e1.first_name, e1.last_name FROM employees e1
    JOIN employees e2 ON (e1.manager_id=e2.employee_id)
    WHERE e1.hire_date < e2.hire_date;
```

11. **Apresentar o cargo, o número de funcionários, o somatório dos salários e a diferença entre o maior salário e o menor dos funcionários desse cargo:**

```
SELECT job_id, COUNT(*) as total_employees,
   SUM(salary) AS tot_salaries, MAX(salary)-MIN(salary) AS diff_salary
    FROM employees
    GROUP BY job_id
    ORDER BY diff_salary desc;
```

12. **Apresentar o nome dos funcionários que ocupam o cargo de *Programmer* ou *President*:**

```
SELECT e.first_name, e.last_name, j.job_title
    FROM employees e,jobs j
    WHERE e.job_id=j.job_id AND
        (j.job_title='Programmer' OR j.job_title='President');
```

13. **Apresentar o nome do cargo e a média dos salários:**

```
SELECT e.first_name, e.last_name, j.job_title
    FROM employees e,jobs j
    WHERE e.job_id=j.job_id AND
        (j.job_title='Programmer' OR j.job_title='President');
```

14. **Apresentar simultaneamente o primeiro nome do funcionário, o seu cargo e a sua experiência, ordenando do mais experiente para o menos experiente:**

```
SELECT e.first_name, e.hire_date,
   FLOOR((SYSDATE-e.hire_date)/365) AS experience, j.job_title
    FROM employees e, jobs j
    WHERE e.job_id = j.job_id
    ORDER BY experience DESC;
```

15. **Apresentar os cargos cujo salário máximo seja menor ou igual que 6000:**

```
SELECT * FROM jobs WHERE max_salary <= 6000;
```

16. **Apresentar o cargo e a diferença entre o salário máximo e o salário mínimo desse cargo, em que o salário máximo esteja entre 6000 e 10000:**

```
SELECT j.job_title, (j.max_salary-j.min_salary) AS diff
    FROM jobs j
    WHERE (j.max_salary>=6000 AND j.max_salary<=10000);
```

17. **Apresentar o identificador dos funcionários que tiveram mais que um cargo no passado:**

```
SELECT employee_id FROM job_history
    GROUP BY employee_id HAVING COUNT(*) > 1;
```

18. **Apresentar os detalhes dos cargos que tenham sido executados por funcionários que atualmente recebem mais que 10000 de salário:**

```
SELECT jh.* FROM job_history jh, employees e
    WHERE (jh.employee_id = e.employee_id) AND salary > 10000;
```

19. **Apresentar os detalhes dos cargos atuais dos funcionários que trabalharam como *IT_PROG* no passado:**

```
SELECT * FROM jobs WHERE job_id IN
    (SELECT job_id FROM employees WHERE employee_id IN
        (SELECT employee_id FROM job_history
            WHERE job_id='IT_PROG'));
```

20. **Apresentar os países e o número de cidades existentes associadas a cada país:**

```
SELECT country_id,  COUNT(*) as total_cities
    FROM locations
    GROUP BY country_id;
```

21. **Apresentar o nome da região, o nome do país, o nome da cidade, o nome do departamento e do gestor deste, bem como do seu cargo:**

```
SELECT r.region_name, c.country_name, l.city,
   d.department_name, e.first_name, e.last_name, j.job_title
    FROM countries c, departments d, employees e,
       jobs j, locations l, regions r
    WHERE r.region_id = c.region_id AND c.country_id=l.country_id
        AND l.location_id=d.location_id AND d.manager_id=e.employee_id
        AND e.job_id=j.job_id;
```

22. **Apresentar o nome do funcionário e o respetivo país onde se encontra a trabalhar:**

```
SELECT e.first_name, c.country_name
    FROM employees e, departments d, locations l, countries c
    WHERE d.department_id = e.department_id
        AND l.location_id=d.location_id AND l.country_id=c.country_id;
```

23. **Apresentar o nome do país, a cidade e o número de departamentos que possuem mais de 5 funcionários:**

```
SELECT country_name, city, COUNT(department_id) AS num_dep
    FROM countries c, locations l, departments d
    WHERE c.country_id=l.country_id AND d.location_id=l.location_id
        AND department_id IN
            (SELECT department_id FROM employees
                GROUP BY department_id HAVING COUNT(department_id)>5)
    GROUP BY country_name, city;
```

6.2 Base de Dados NoSQL

Como foi apresentado previamente NoSQL é o termo utilizado para bases de dados não relacionais, onde geralmente não é utilizado o SQL como linguagem de consulta, neste tipo de bases de dados não existe uma linguagem de consulta padrão. Assim cada tipo de base de dados utiliza uma sintaxe devida.

6.2.1 Base de dados orientada a documentos

Para demonstrar a operacionalidade da base de dados orientada a documentos como também perceber as vantagens da sua utilização, decidiu-se proceder de forma análoga ao que foi feito na secção anterior (6.1), implementar as mesmas consultas.

Como é conhecido a grande parte dos sistemas de gestão de bases de dados que utilizam documentos possuem uma linguagem de consulta própria, o sistema utilizado para implementar este tipo de bases de dados foi *MongoDB*, este utiliza uma sintaxe própria como linguagem de consulta, assim todas as consultas abaixo apresentadas encontram-se nessa linguagem.

1. **Apresentar os detalhes dos departamentos ordenados pelo nome de forma ascendente:**

```
db.hr.distinct("department").sort()
```

2. **Apresentar os nomes dos departamentos e os responsáveis por estes:**

```
db.hr.aggregate([
    { $group: {
            _id: null,
            departments: {
                $addToSet: {
                    _id: "$department.department_id",
                    department: "$department.name",
                    manager: "$department.manager"
                }
            }}
    }, { $project: {_id:0}}
])
```

3. **Apresentar os nomes dos departamentos e o número de funcionários de cada um:**

```
db.hr.aggregate([
    { $group: {
        _id: "$department.name",
        tot_employees: {$sum: 1}
    }},
    { $match: {_id: {$ne: null}}}
])
```

4. **Apresentar os identificadores dos gestores e o número de funcionários geridos por cada um:**

```
db.hr.aggregate([
    { $group: {
        _id: "$manager_id",
        tot_employees: {$sum: 1}
    }}
])
```

5. **Apresentar os detalhes dos funcionários cuja percentagem de comissão seja nula, o salário esteja entre 10000 e 15000 e o gestor possua o identificador 101:**

```
db.hr.find(
    { $and: [
        {salary: {$gte: 10000}},
        {salary: {$lte: 15000}},
        {manager_id: 101}
    ]}
)
```

6. **Apresentar os funcionários cujo primeiro nome e último nome começam pela letra G:**

```
db.hr.find(
    { $and: [
        {first_name: {$regex: "G"}},
        {last_name: {$regex: "G"}}
    ]}
)
```

7. **Apresentar todos os funcionários que entraram para a empresa após 19 de Dezembro de 2007:**

```
db.hr.find(
    { $and: [
        {hire_date: {$gte: "2007-12-19"}}
    ]}
)
```

8. **Apresentar o nome dos funcionários que foram contratados em 2008:**

```
db.hr.find(
    {hire_date: {$regex: /2008/}},
    {first_name: 1, last_name: 1, hire_date: 1}
)
```

9. **Apresentar quantos funcionários foram admitidos em cada mês do ano 2008:**

```
db.hr.aggregate([
    { $match: {
        $and: [
            {hire_date: {$gte: "2008-01-01"}},
            {hire_date: {$lte: "2008-12-31"}}
        ]
    }},
    { $addFields: {
        mes: {
            $arrayElemAt: [
                { $split: ["$hire_date", "-"]}
                ,1
            ]
        }
    }},
    { $group: {
        _id: "$mes",
        funcionarios: {$sum:1}
    }}
])
```

10. **Apresentar o nome dos funcionários que iniciaram funções antes do seu gestor:**

    Para concretizar esta consulta é necessário uma das duas possibilidades:

    - Percorrer duas vezes o documento, porque para cada funcionário, teria que voltar a percorrer o documento para comparar o *hire_date* entre o identificador do gestor e o identificador do funcionário. O *MongoDB* não nos permite executar operações desse tipo.
    - Adicionar nos campos do documento a informação relativa ao gestor de cada funcionário, o que não é ideal pois iria implicar um enorme aumento no tamanho de cada registo.

    Por isso, decidimos que esta consulta não seria ideal para representar neste tipo de base de dados.

11. **Apresentar o cargo, o número de funcionários, o somatório dos salários e a diferença entre o maior salário e o menor dos funcionários desse cargo:**

```
db.hr.aggregate([
    { $group: {
        _id:"$job.title",
        total_employees: {$sum:1},
        tot_salaries: {$sum: "$salary"},
        salaries: {$push: "$salary"}
    }},
    { $addFields: {
        diff_salary: {
            $subtract: [
                {$max: "$salaries"},
                {$min: "$salaries"}
            ]
        }
    }}
])
```

12. **Apresentar o nome dos funcionários que ocupam o cargo de *Programmer* ou *President*:**

```
db.hr.find(
    { $or: [
        {"job.title":"President"},
        {"job.title":"Programmer"}
    ]},
    {first_name: 1, last_name: 1, "job.title": 1}
)
```

13. **Apresentar o nome do cargo e a média dos salários:**

```
db.hr.aggregate([
    { $group: {
        _id:"$job.title",
        avgSalary: {$avg: "$salary"}
    }}
])
```

14. **Apresentar simultaneamente o primeiro nome do funcionário, o seu cargo e a sua experiência, ordenando do mais experiente para o menos experiente:**

```
db.hr.aggregate([
    { $project: {
        _id:1,
        name: {$concat: ["$first_name"," ", "$last_name"]},
        hire_date:1
    }},
    {$sort: {hire_date:1}}
])
```

15. **Apresentar os cargos cujo salário máximo seja menor ou igual que 6000:**

```
db.hr.find(
    {"job.max_salary": {$lte: 6000}}
)
```

16. **Apresentar o cargo e a diferença entre o salário máximo e o salário mínimo desse cargo, em que o salário máximo esteja entre 6000 e 10000:**

```
db.hr.aggregate([
    { $match: {
        $and: [
            {"job.max_salary": {$gte: 6000}},
            {"job.max_salary": {$lte: 10000}}
        ]
    }},
    { $project: {
        _id:"$job.title",
        max_salary: "$job.max_salary",
        min_salary: "$job.min_salary",
        diff: {
            $subtract: [
                "$job.max_salary",
                "$job.min_salary"
            ]
        } }} ])
```

17. **Apresentar o identificador dos funcionários que tiveram mais que um cargo no passado:**

```
db.hr.find(
    {'history.1': { $exists: true}},
    {_id:1, first_name:1, last_name:1, history:1}
)
```

18. **Apresentar os detalhes dos cargos que tenham sido executados por funcionários que atualmente recebem mais que 10000 de salário:**

```
db.hr.aggregate([
    { $match: {
        salary: {$gte: 10000}
    }},
    { $project: {
        _id:1,
        name: {$concat: ["$first_name"," ", "$last_name"]},
        salary: 1,
        history: 1
    }}
])
```

19. **Apresentar os detalhes dos cargos atuais dos funcionários que trabalharam como *IT_PROG* no passado:**

```
db.hr.find(
    {"history": {
        $elemMatch: {job_title:"Programmer"}
    }},
    {}
)
```

20. **Apresentar os países e o número de cidades existentes associadas a cada país:**

```
db.hr.aggregate([
    { $group: {
        _id:'$department.country_name',
        cities: {$addToSet: '$department.city'}
    }},
    { $addFields: {
            total_cities: {$size: "$cities"}
    }}
])
```

21. **Apresentar o nome da região, o nome do país, o nome da cidade, o nome do departamento e do gestor deste, bem como do seu cargo:**

```
db.hr.find(
    {},
    {
        _id: 1,
        first_name: 1,
        last_name: 1,
        "job.title": 1,
        "department.name": 1,
        "department.manager": 1,
        "department.city": 1,
        "department.country_name": 1,
        "department.region_name":1
    }
)
```

22. **Apresentar o nome do funcionário e o respetivo país onde se encontra a trabalhar:**

```
db.hr.find(
    {},
    {
        _id: 1,
        first_name: 1,
        last_name: 1,
        "department.country_name": 1
    }
)
```

23. **Apresentar o nome do país, a cidade e o número de departamentos que possuem mais de 5 funcionários:**

```
db.hr.aggregate([
    { $group: {
        _id: "$department.name",
        country: {$addToSet: "$department.country_name"},
        city: {$addToSet: "$department.city"},
        funcionarios: {$sum: 1}
    }},
    { $match: {
        funcionarios: {$gt:5}
    }}
])
```

6.2.2 Base de dados orientada a grafos

De forma a demonstrar a operacionalidade da base de dados orientada a grafos como também perceber as vantagens da sua utilização, procedeu-se de modo análogo ao que foi na seções anteriores (6.1 e 6.2.1) e implementar as mesmas consultas.

Como sabemos a grande parte dos sistemas de gestão de bases de dados que utilizam grafos possuem uma linguagem de consulta própria, o sistema utilizado para implementar este tipo de bases de dados foi *Neo4J*, este utiliza o *Cypher* como linguagem de consulta, assim todas as consultas abaixo apresentadas encontram-se nessa linguagem.

1. **Apresentar os detalhes dos departamentos ordenados pelo nome de forma ascendente:**

```
MATCH (d:DEPARTMENT) RETURN d ORDER BY d.name ASC
```

2. **Apresentar os nomes dos departamentos e os responsáveis por estes:**

```
MATCH (e:EMPLOYEE)<-[:MANAGED_BY]-(d:DEPARTMENT)
    RETURN d.name AS Department_Name, e.first_name, e.last_name
```

3. **Apresentar os nomes dos departamentos e o número de funcionários de cada um:**

```
MATCH (e:EMPLOYEE)-[:WORK_AT]->(d:DEPARTMENT)
    WITH d,COUNT(e) AS total_employees
    RETURN d.name AS Department_Name, total_employees
```

4. **Apresentar os identificadores dos gestores e o número de funcionários geridos por cada um:**

```
MATCH (employee:EMPLOYEE)-[:IS_MANAGED_BY]->(manager:EMPLOYEE)
    WITH manager, COUNT(employee) AS total_employees
    RETURN manager.id, total_employees
```

5. **Apresentar os detalhes dos funcionários cuja percentagem de comissão seja nula, o salário esteja entre 10000 e 15000 e o gestor possua o identificador 101:**

```
MATCH (e:EMPLOYEE)-[:IS_MANAGED_BY]->(manager:EMPLOYEE{id:101})
    WHERE e.salary >= 10000 AND e.salary <=15000
    RETURN e
```

6. **Apresentar os funcionários cujo primeiro nome e último nome começam pela letra G:**

```
MATCH (e:EMPLOYEE)
    WHERE (e.first_name STARTS WITH 'G' AND e.last_name STARTS WITH 'G')
    RETURN e.first_name, e.last_name
```

7. **Apresentar todos os funcionários que entraram para a empresa após 19 de Dezembro de 2007:**

```
MATCH (e:EMPLOYEE)
    WHERE e.hire_date > date({year:2007, month:12, day:19})
    RETURN e.first_name, e.hire_date
```

8. **Apresentar o nome dos funcionários que foram contratados em 2008:**

```
MATCH (e:EMPLOYEE) WHERE date(e.hire_date).year = 2008
    RETURN e.first_name, e.last_name, e.hire_date
```

9. **Apresentar quantos funcionários foram admitidos em cada mês do ano 2008:**

```
MATCH (e:EMPLOYEE) WHERE date(e.hire_date).year = 2008
    RETURN COUNT(date(e.hire_date).month) AS tot,
    date(e.hire_date).month AS month
    ORDER BY month ASC
```

10. **Apresentar o nome dos funcionários que iniciaram funções antes do seu gestor:**

```
MATCH (employee:EMPLOYEE)-[:IS_MANAGED_BY]->(boss:EMPLOYEE)
    WHERE employee.hire_date < boss.hire_date
    RETURN employee.first_name, employee.last_name
```

11. **Apresentar o cargo, o número de funcionários, o somatório dos salários e a diferença entre o maior salário e o menor dos funcionários desse cargo:**

```
MATCH (e:EMPLOYEE)-[:DOES]->(j:JOB)
    RETURN j.title AS Title, COUNT(e.id) AS Employees,
    sum(e.salary) AS `Total Salary`,
    (max(e.salary)-min(e.salary)) AS `Max & Min Salary Difference`
    ORDER BY `Max & Min Salary Difference` DESC
```

12. **Apresentar o nome dos funcionários que ocupam o cargo de *Programmer* ou *President*:**

```
MATCH (e:EMPLOYEE)-[:DOES]->(j:JOB)
    WHERE (j.title = 'President' OR j.title = 'Programmer')
    RETURN e.first_name, e.last_name, j.title
```

13. **Apresentar o nome do cargo e a média dos salários:**

```
MATCH (e:EMPLOYEE)-[:DOES]->(j:JOB)
    RETURN j.title AS Title, AVG(e.salary) AS `Average Salary`
    ORDER BY `Average Salary` ASC
```

14. **Apresentar simultaneamente o primeiro nome do funcionário, o seu cargo e a sua experiência, ordenando do mais experiente para o menos experiente:**

```
MATCH (e:EMPLOYEE)-[:DOES]->(j:JOB)
    RETURN e.first_name AS Name, j.title AS Title,
        duration.inDays(e.hire_date,date()).days AS Experience
    ORDER BY Experience DESC
```

15. **Apresentar os cargos cujo salário máximo seja menor ou igual que 6000:**

```
MATCH (e:EMPLOYEE)-[:DOES]->(j:JOB)
    RETURN e.first_name AS Name, j.title AS Title,
        duration.inDays(e.hire_date,date()).days AS Experience
    ORDER BY Experience DESC
```

16. **Apresentar o cargo e a diferença entre o salário máximo e o salário mínimo desse cargo, em que o salário máximo esteja entre 6000 e 10000:**

```
MATCH (j:JOB) WHERE j.max_salary >= 6000 AND j.max_salary <= 10000
    RETURN j.title AS Title,
        (j.max_salary-j.min_salary) AS ‘Max & Min Salary Difference‘
```

17. **Apresentar o identificador dos funcionários que tiveram mais que um cargo no passado:**

```
MATCH (e:EMPLOYEE)-[:DID]->(j:JOB)
    WITH e, count(e) AS ‘Past Jobs‘
    WHERE ‘Past Jobs‘ > 1
    RETURN e.id AS ID, e.first_name AS Name, ‘Past Jobs‘
```

18. **Apresentar os detalhes dos cargos que tenham sido executados por funcionários que atualmente recebem mais que 10000 de salário:**

```
MATCH (e:EMPLOYEE)-[:DID]->(history:JOB)
    WHERE e.salary > 10000
    RETURN e, history
```

19. **Apresentar os detalhes dos cargos atuais dos funcionários que trabalharam como *IT_PROG* no passado:**

```
MATCH (job:JOB)<-[:DOES]-(e:EMPLOYEE)-[:DID]->(history:JOB{id:'IT_PROG'})
    RETURN job, e, history
```

20. **Apresentar os países e o número de cidades existentes associadas a cada país:**

```
MATCH (l:LOCATION)-[:CONTAINED_IN]->(c:COUNTRY)
    RETURN c.country_name AS Country, count(l.city) AS Cities
    ORDER BY Cities DESC
```

21. **Apresentar o nome da região, o nome do país, o nome da cidade, o nome do departamento e do gestor deste, bem como do seu cargo:**

```
MATCH (country:COUNTRY)<-[:CONTAINED_IN]-(location:LOCATION)~
    <-[:HAVE]-(department:DEPARTMENT)
    -[:MANAGED_BY]->(manager:EMPLOYEE)-[:DOES]->(job:JOB)
        RETURN country.region_name, country, location,
            department, manager, job
```

22. **Apresentar o nome do funcionário e o respetivo país onde se encontra a trabalhar:**

```
MATCH (country:COUNTRY)<-[:CONTAINED_IN]-(location:LOCATION)
    <-[:HAVE]-(department:DEPARTMENT)
    <-[:WORK_AT]-(employee:EMPLOYEE)
        RETURN country.country_name, employee.first_name
```

23. **Apresentar o nome do país, a cidade e o número de departamentos que possuem mais de 5 funcionários:**

```
MATCH (country:COUNTRY)<-[:CONTAINED_IN]-(location:LOCATION)
    <-[:HAVE]-(department:DEPARTMENT)
    <-[:WORK_AT]-(employee:EMPLOYEE)
        WITH country, location, department,
            COUNT(employee) AS total_employees
        WHERE total_employees > 5
        RETURN country, location, department;
```

7 Comparação da Operacionalidade

Após a demonstração da operacionalidade de cada uma das bases de dados implementadas, tal como foi apresentado, executou-se um conjunto de consultas nos diferentes modelos.

Nesta seção iremos abordar as principais diferenças encontradas no processo de implementação de cada consulta, bem como salientar o objetivo de cada *query* quando considerada relevante.

Queremos também mencionar que todos os tempos que irão ser referidos são na ordem de milissegundos e segundos, unidades quase impercetíveis, contudo poderá ser importante caso os dados cresçam exponencialmente. No sistema de gestão de bases de dados utilizado relacional e o orientado a grafos, em cada consulta é exibido a quantidade de tempo demorada; no orientado a documentos foi necessário acrescentar a extensão, *.explain("executionStats")*, às consultas apresentadas, de forma a obter o tempo que cada consulta demora neste sistema.

1. **Apresentar os detalhes dos departamentos ordenados pelo nome de forma ascendente:** [VER: SQL=1;Doc.=1;Graf.=1]
   Na presente consulta o processo de implementação foi muito semelhante nas diferentes bases de dados utilizadas.

2. **Apresentar os nomes dos departamentos e os responsáveis por estes:** [VER: SQL=2;Doc.=2;Graf.=2]
   Na atual *query* a principal diferença encontrada foi que, no modelo relacional efetuou-se a junção de duas tabelas, enquanto que base de dados com recurso a grafos apenas se consultou o relacionamento correspondente (MANAGED_BY) entre as entidades funcionários e departamentos, o que levou a que esta fosse executada em metade do tempo. Em relação à base de dados orientada a documentos dada a estrutura do documento criado, simplesmente se retirou as informações essenciais da chave *department*.

3. **Apresentar os nomes dos departamentos e o número de funcionários de cada um:** [VER: SQL=3;Doc.=3;Graf.=3]
   Nesta consulta o processo de elaboração foi muito idêntico, contudo no modelo relacional e no modelo com documentos foi necessário agrupar os departamentos por nome, processo esse que não foi aplicado no de grafos.

Para este modelo foi necessário recorrer à cláusula *WITH* para efetuar a contagem de funcionários o permitiu que esta consulta seja encadeada, assim foi possível apresentar os mesmos resultados o que implicou que a consulta fosse aproximadamente 10 milissegundos mais lenta, no entanto a utilização de documentos foi a mais eficiente.

4. **Apresentar os identificadores dos gestores e o número de funcionários geridos por cada um:** [VER: SQL=4;Doc.=4;Graf.=4]
   Como foi previamente apresentado a execução desta *query* tanto no modelo relacional como não relacional com o uso do documento foi muito idêntico dado que, houve a necessidade de agrupar pelos gestores enquanto que, no modelo com grafos retirou-se partido da relação criada.

5. **Apresentar os detalhes dos funcionários cuja percentagem de comissão seja nula, o salário esteja entre 10000 e 15000 e o gestor possua o identificador 101:** [VER: SQL=5;Doc.=5;Graf.=5]
   Na presente consulta o processo de implementação foi muito semelhante enquanto que, na base de dados relacional acedeu-se às colunas respetivas; na bases de dados orientada a documentos acedeu-se aos pares chave/valor devidos; na base de dados orientada a grafos acedeu-se às propriedades dos nodos funcionários. Como pode ser observado todas as bases de dados retornam a mesma informação só que o modo de acesso varia consoante as propriedades de cada uma.

6. **Apresentar os funcionários cujo primeiro nome e último nome começam pela letra G:** [VER: SQL=6;Doc.=6;Graf.=6]
   Na atual *query* a principal diferença que quisemos demonstrar foi a manipulação de texto e observou-se que a utilização de documentos obteve um excelente tempo de resposta em comparação à base de dados orientada a grafos.

7. **Apresentar todos os funcionários que entraram para a empresa após 19 de Dezembro de 2007:** [VER: SQL=7;Doc.=7;Graf.=7]
   Tal como a consulta anterior pretendeu-se destacar nesta, a forma como lidar com datas. Registou-se que na base de dados relacional se obteve uma excelente resposta. Um outro fator observado foi que para o sistema de gestão de base de dados com uso de documentos é possível manipular uma data como se fosse um simples número.

8. **Apresentar o nome dos funcionários que foram contratados em 2008:** [VER: SQL=8;Doc.=8;Graf.=8]
   No seguimento da mesma manipulação quisemos confirmar se esta superioridade era mantida apenas quando é comparado com um determinado ano. Chegámos à conclusão de que sim, a base de dados relacional fornece excelentes ferramentas para manipular datas quando comparado com a utilização de uma base de dados assente num documento onde foi necessário aplicar uma expressão regular a fim de encontrar o ano de contratação.

9. **Apresentar quantos funcionários foram admitidos em cada mês do ano 2008:** [VER: SQL=9;Doc.=9;Graf.=9]
   Na atual *query* pretendeu-se manipular datas, mas agrupando a informação apesar das diferenças notórias no processo de escrita entre as bases de dados. Nas bases de dados com grafos não foi necessário agrupar a informação, neste caso por mês, mesmo assim demorou ligeiramente mais tempo a ser executada. Um outro fator notado foi que dada a estrutura do documento criado a *query* para responder tornou-se demasiada complexa neste sistema.

10. **Apresentar o nome dos funcionários que iniciaram funções antes do seu gestor:** [VER: SQL=10;Doc.=10;Graf.=10]
    Na presente consulta pretendeu-se realçar as vantagens que a base de dados com grafos possui ao representar os relacionamentos entre os nodos comparado aos documentos cujo objetivo não é esse, logo este tipo de consultas torna-se demasiado complexo com a utilização de documentos.

11. **Apresentar o cargo, o número de funcionários, o somatório dos salários e a diferença entre o maior salário e o menor dos funcionários desse cargo:** [VER: SQL=11;Doc.=11;Graf.=11]
    Na atual *query* pretendeu-se destacar o comportamento de cada base de dados com mais que uma operação aritmética, bem como, a múltipla apresentação dos mesmos. Observou-se uma maior eficácia da base de dados orientada a documentos apesar da complexidade.

12. **Apresentar o nome dos funcionários que ocupam o cargo de *Programmer* ou *President*:** [VER: SQL=12;Doc.=12;Graf.=12]
    Nesta consulta quisemos salientar principalmente, a elaboração de dados relacionados, tal como era espetável a melhor resposta obtida foi na base de dados orientada a grafos.

13. **Apresentar o nome do cargo e a média dos salários:** [VER: SQL=13;Doc.=13;Graf.=13]
    Na corrente *query* pretendeu-se realçar dados relacionados com operações aritméticas envolvidas no processo de desenvolvimento, o que levou a um decréscimo da prestação da base de dados orientada a grafos.

14. **Apresentar simultaneamente o primeiro nome do funcionário, o seu cargo e a sua experiência, ordenando do mais experiente para o menos experiente:** [VER: SQL=14;Doc.=14;Graf.=14]
    Na presente consulta, o objetivo foi verificar qual dos sistemas era mais eficiente na ordenação de datas. Observámos que cada sistema tem uma sintaxe diferente e consequentemente isso influência a prestação.

15. **Apresentar os cargos cujo salário máximo seja menor ou igual que 6000:** [VER: SQL=15;Doc.=15;Graf.=15]
    Nesta *query* não se pretendeu mostrar a complexidade ou diferenças entre cada sistema, mas sim apresentar as similaridades dos sistemas utilizados na resolução de certos problemas.

16. **Apresentar o cargo e a diferença entre o salário máximo e o salário mínimo desse cargo, em que o salário máximo esteja entre 6000 e 10000:** [VER: SQL=16;Doc.=16;Graf.=16]
    Uma *query* simples realizada numa base de dados relacional pode tornar-se demasiado complexa, ao nível de escrita numa base de dados orientada a documentos, logo por vezes é necessário ter em consideração a estrutura do documento.

17. **Apresentar o identificador dos funcionários que tiveram mais que um cargo no passado:** [VER: SQL=17;Doc.=17;Graf.=17]
    Na corrente consulta quisemos demonstrar a clareza que a base de dados orientada a grafos permite na construção de *queries* com a representação explícita de cada relacionamento entre entidades.

18. **Apresentar os detalhes dos cargos que tenham sido executados por funcionários que atualmente recebem mais que 10000 de salário:** [VER: SQL=18;Doc.=18;Graf.=18]
    Na presente *query* dado a representação da informação através de um grafo, os resultados obtidos são mais facilmente compreendidos do que a informação seja devolvida através de um documento.

19. **Apresentar os detalhes dos cargos atuais dos funcionários que trabalharam como *IT_PROG* no passado:**[VER:SQL=19;Doc.=19;Graf.=19]
Na atual consulta tal como se apresentou a base de dados com recurso a grafos expressa claramente as relações entre as diversas entidades, neste caso a entidade funcionário estabelece duas relações com a mesma entidade, cargo. Contudo estas traduzem significados totalmente diferentes, uma representa a atualidade e a outra o passado; isto é, relevante na medida que é necessário conhecer o significado de cada relacionamento neste tipo de base de dados. Comparado com a base de dados que utiliza documentos a informação não é tão explicativa, mas é mais prática.

20. **Apresentar os países e o número de cidades existentes associadas a cada país:** [VER: SQL=20;Doc.=20;Graf.=20]
A base de dados documental armazenou a informação toda num único documento onde cada valor representa a entidade funcionário, o que levou a que alguma informação ficasse contida dentro deste. Esta abordagem levanta problemas pois da forma como foi estruturada o documento, o resultado obtido nesta consulta utilizando o *MongoDB* foi incompleto comparado com o obtido utilizando o SQL, porque como foi descrito no problema, existem cidades que não possuem departamentos e consequentemente estas não se encontram relacionadas com a entidade base do documento, funcionário.

21. **Apresentar o nome da região, o nome do país, o nome da cidade, o nome do departamento e do gestor deste, bem como do seu cargo:** [VER: SQL=21;Doc.=21;Graf.=21]
Na corrente *query* pretendeu-se principalmente relacionar todas as entidades (descritas previamente), esperava-se que a base de dados orientada a grafos fosse a melhor na representação da *query* como no resultado e assim foi. Esta é claramente uma base de dados perfeita para representar informação relacionada pois permite termos uma noção clara do que estamos a relacionar, sem o risco de esquecer relações que comprometem o resultado. Dada a representação através de um grafo conseguimos além de observar o resultado desta consulta, visualizar quais as cidades que não possuem departamentos e consequentemente gestores.

22. **Apresentar o nome do funcionário e o respetivo país onde se encontra a trabalhar:** [VER: SQL=22;Doc.=22;Graf.=22]
    Na presente *query* quisemos salientar que para uma *query* simples entre duas entidades é necessário estabelecer na base de dados relacional e na base de dados orientada a grafos várias relações. Nesta última, se o objetivo for apenas obter o nome do funcionário e o respetivo país, uma abordagem com grafos não ajuda, na medida que apresenta demasiada informação, para esta situação uma abordagem tabelar facilita a interpretação.

23. **Apresentar o nome do país, a cidade e o número de departamentos que possuem mais de 5 funcionários:** [VER: SQL=23;Doc.=23;Graf.=23]
    Por fim, na atual *query* observou-se que na base de dados orientada a grafos os resultados obtidos são mais claros, intuitivos e percetíveis quando comparados às outras bases de dados utilizadas.

8 Análise Crítica

8.1 BD Relacional *vs.* BD orientada a documentos

Após implementado os dois tipos de bases de dados é possível constatar várias diferenças entre estes dois modelos.

Como foi dito previamente a base de dados relacional armazenou os dados em tabelas, cada tabela contém colunas e cada linha representa um registo, ou seja, as informações sobre uma entidade, por exemplo, um funcionário encontra-se espalhada por várias tabelas e para tal, foi necessário estabelecer relacionamentos entre as tabelas.

Comparando com a base de dados documental implementada esta não usa tabelas, os dados da mesma entidade são armazenados num único documento e todos os dados associados são armazenados dentro desse documento, o que acarretou algumas desvantagens como a perda de informação dos registos que não estavam relacionados.

No entanto, também trouxe várias vantagens interessantes, tais como:

Flexibilidade : Nas bases de dados relacionais, apesar de ter sido dado o esquema foi necessário efetuar um modelo lógico antes da inserção de qualquer dado. Na base de dados documental, não exigiu esse requisito tal como foi apresentado simplesmente carregou-se todos os dados sem usar qualquer esquema predefinido.

Relacionamentos : Os dados armazenados nas bases de dados documental não possuem chaves estrangeiras, sendo estas apenas usadas na base de dados relacional para impor relacionamentos entre tabelas. A ideia por trás do modelo de documental foi que todos os dados associados a um registo (funcionário) são armazenados no mesmo documento. A necessidade de estabelecer um relacionamento no modelo documental não possui o mesmo impacto que numa base de dados relacional.

Escalabilidade : A bases de dados de documental pode ser dimensionada horizontalmente, isto é, os dados podem ser armazenados em milhares de computadores e o sistema terá sempre um bom desempenho - *sharding*. Para a base de dados relacional é mais adequado um dimensionamento vertical (por exemplo, adicionar mais memória, espaço de disco, etc.). Contudo é importante salientar que existe um limite para os recursos de uma máquina, acabando a escalabilidade horizontal poder ser a única opção.

Feita uma reflexão sobre qual a base de dados mais adequada para o trabalho proposto o nosso grupo considerou que tudo depende do propósito final das aplicações a desenvolver. Ambos os modelos possuem várias vantagens interessantes de serem exploradas. Supondo que a nossa base de dados principal é com base no uso de documentos, contudo pretendemos utilizar ferramentas de *business intelligence* para analisar um conjunto específico de dados, podemos ter necessidade de usar uma segunda base de dados relacional que armazena apenas os dados a serem analisados. Por outro lado, podemos ter uma situação oposta em que utilizamos a base de dados relacional como principal, contudo pretendemos armazenar outros dados que possam vir a ser importantes, não existindo ainda a necessidade de uma estrutura bem definida, e para tal podemos ter uma segunda base de dados secundária orientada a documentos, como um tipo de *data lake*.

8.2 BD Relacional *vs.* BD orientada a grafos

Após implementado os dois tipos de bases de dados foi possível concluir que o modelo com recursos a grafos é uma alternativa ao modelo relacional apesar da representação dos dados através de grafos contrastar com a estrutura tabelar.

A principal diferença notada foi que no SGBD orientado a grafos recorreu-se à utilização de etiquetas e propriedades para definir os relacionamentos entre nodos, este processo ajudou no retorno de grandes quantidades de dados relacionados, sem a necessidade de usar junções em várias tabelas, como foi necessário fazer no modelo relacional com o SQL.

Em suma, as bases de dados orientadas a grafos têm muitos benefícios sobre outros tipos de bases de dados, principalmente sobre as bases de dados relacionais, destacamos as seguintes:

Perfomance : As bases de dados com grafos potenciam grandes ganhos de desempenho em relação às bases de dados relacionais, principalmente quando se trata de consultas grandes em dados relacionados. Como foi demonstrado para algumas consultas na base de dados relacional envolveu consultas que juntou todas as tabelas (por exemplo, a consulta 21), consultas com tantas tabelas podem ser extremamente lentas, especialmente à medida que os dados crescem. Para os mesmos dados que foram processados e armazenados, algumas consultas foram mais simples e mais rápidas na base de dados com grafos, isso porque as consultas são localizadas numa parte do grafo, ou seja, isto significa que o tempo de execução de cada consulta é proporcional apenas ao tamanho da parte do grafo percorrido para satisfazer as condições.

Flexibilidade : No que diz respeito a este item não houve necessidade de definir um esquema, existiu total flexibilidade sobre o crescimento da base de dados. No modelo relacional, apesar de o modelo já ter sido dado foi necessário mapear os requisitos em detalhe antes de criar a base de dados tal como, previamente prever qualquer possível mudança nos requisitos e criar uma solução que atenda a todos os cenários futuros possíveis, o que nem sempre é possível. Caso os requisitos de negócios aumentem ou mudem significativamente, a estrutura da base de dados relacional poderá precisar mudar significativamente de forma a responder aos mesmos levando à possibilidade de refazer todo o modelo. Por outro lado, a mesma situação de possibilidade de reformulação não acontece numa base de dados orientada a grafos na medida que não há esquema predefinido como tal, qualquer esquema é simplesmente um reflexo dos dados que foram inseridos.

Feita uma reflexão sobre qual a base de dados mais adequada para o trabalho proposto o nosso grupo considerou que tudo depende do propósito final das consultas e das necessidades. Ambos os modelos possuem várias vantagens interessantes de serem exploradas. O modelo com grafos não é necessariamente o modelo ideal para todos os cenários. Para a quantidade de dados utilizada a estrutura tabelar das bases de dados relacionais foi adequada. No entanto podemos afirmar que o modelo com grafos foi excelente para consultar grandes conjuntos de dados relacionados.

8.3 BD orientada a documentos *vs.* BD orientada a grafos

Cada tipo de bases de dados NoSQL implementado apresenta um ponto de vista de abstração diferente para os mesmos dados.

Em contraste com a base de dados orientada a grafos, a tecnologia de base de dados de documentos já é conhecida há algum tempo. Tal como foi previamente referido as bases de dados NoSQL, tanto as bases de dados orientadas a grafos como a orientadas a documentos, permitem que as empresas resolvam problemas rapidamente, apesar de um dilúvio de informações que mudam rapidamente. Contudo o desafio prende-se em descobrir qual tipo de base de dados a utilizar, pois uma fornece uma visão dos dados muito diferente da outra tal como foi retratado no trabalho desenvolvido. Esta escolha depende muito da estrutura do negócio em causa e dos problemas que irão ser tratados.

Em suma, ambas as bases de dados apresentam várias vantagens, contudo é importante realçar a principal diferença entre os modelos:

- **Diferentes tipos de abstração** :
  - Uma base de dados orientada a grafos tal como referido foi melhor para descobrir como os funcionários se relacionam. A base de dados com grafos descreveu o comportamento das entidades e as relações entre elas. Conforme os dados são adicionados ou modificados, a base de dados orientada a grafos adapta-se e atualiza essas informações em tempo real. Esta tecnologia encontra melhor as informações onde a estrutura de dados inicial é desafiada e existe uma distribuição distorcida dos mesmos, ou seja, é ideal para lidar com dados inter-relacionados e/ou caso haja muitas conexões entre os diversos nodos.
  - Por outro lado, a base de dados orientada a documentos fornece uma flexível taxonomia que não é limitada pelo contexto. A hierarquia pode ser reorganizada e cada documento pode ter um conjunto diferente de valores. Os dados aparecem numa estrutura parecida a uma árvore onde caminhos ou ramificações conectam os valores dos dados/folhas. Uma propriedade interessante da base de dados com documentos é que facilmente se adapta a diferentes requisitos de formação dos dados o que leva a uma boa manipulação de novos tipos de dados a serem usados e até ter uma capacidade de resposta em caso de falhas.

Feita uma reflexão sobre qual a base de dados mais adequada para o trabalho proposto o nosso grupo considerou que ambas as bases de dados oferecem poderes muito interessantes para lidar com o futuro. As bases de dados orientada a grafos podem mostrar as conexões entre novos departamentos e os funcionários que provavelmente trabalharam aí. Se quisermos algo mais automático, que atualiza os funcionários na hora os salários dos funcionários, isso pode acontecer quase imediatamente, usando uma base de dados orientada a documentos. Resumindo, ambas as bases de dados oferecem diferentes perspetivas e poder para resolver novos problemas.

9 Conclusão

O presente relatório descreveu, de forma sucinta, o trabalho de análise, planeamento e implementação de um SGBD relacional, *Oracle SQL Developer* e dois não relacionais, orientado a documentos (*MongoDB*) e orientado a grafos (*NEO4J)*, a partir de um modelo relacional fornecido.

Durante a elaboração deste trabalho surgiram algumas dificuldades que, com o esforço e entusiasmo do grupo pelo resultado, acabaram por ser ultrapassadas. Entre as quais destacam-se o desafio que foi a realização de algumas consultas utilizam o *MongoDB* por não ser um sistema em que os elementos do grupo se sentissem muito confortáveis dado o percurso académico até ao momento.

Numa perspetiva futura, considera-se que seria interessante a migração dos dados fornecidos para outros sistemas de gestão de base de dados (por exemplo, para o *Elasticsearch* ou para o *ArangoDB*), bem como a realização das consultas que o grupo implementou nestes novos sistemas.

Após a realização deste trabalho, o grupo ficou consciente dos diferentes paradigmas de bases de dados utilizados como também a sua aplicação na conceção e implementação de sistemas.

Consideramos que os objetivos propostos com a realização deste trabalho foram cumpridos, bem como a consolidação dos conhecimentos em sistemas de bases de dados da última geração em particular na administração e exploração dos mesmos.

Por fim, o grupo espera que os conhecimentos obtidos e consolidados sejam de enorme utilidade tendo uma perspetiva futura.

Referências

[1] Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management , Addison-Wesley, 4a Edição, 2004.

[2] Luc Perkins, Eric Redmond, Jim Wilson, Seven Databases in Seven Weeks - A Guide to Modern Databases and the NoSQL Movement, Pragmatic Bookshelf, 2018.

[3] Data Topics. Domain-Specific Languages. Knight, Michelle (26 de Fevereiro de 2019). Acedido em 25 de Janeiro de 2021, em: https://www.dataversity.net/graph-database-vs-document-database-different-levels-of-abstraction/#