Bases de Dados NoSQL

Mestrado Integrado em Engenharia Informática
(2020/2021)

A84003 Beatriz Rocha

A85227 João Azevedo

A85729 Paulo Araújo

A83719 Pedro Machado

Braga,
janeiro 2021

Resumo

O presente trabalho prático foi realizado com o objetivo de criar competências na utilização de diferentes paradigmas de bases de dados e na sua aplicação, conceção e implementação. Este consistiu na análise, planeamento e implementação de dois modelos não relacionais (um orientado a grafos - Neo4j - e outro orientado a documentos - MongoDB), tendo como ponto de partida a base de dados relacional HR disponibilizada no Oracle Database.

Deste modo, ao longo deste documento, iremos descrever pormenorizadamente o trabalho elaborado, começando pela descrição da base de dados HR, seguida da apresentação dos modelos não relacionais implementados e do método utilizado para o seu desenvolvimento. Por fim, iremos apresentar o conjunto de interrogações implementadas, onde iremos fazer a comparação de desempenho dos vários modelos (relacional e não relacionais).

Área de Aplicação: Desenho, arquitetura, desenvolvimento e implementação de Sistemas de Gestão de Bases de Dados.

Palavras-Chave: Bases de Dados, Bases de Dados Relacionais, Bases de Dados Não Relacionais, SQL, NoSQL, Oracle, MongoDB, Neo4j, Migração de Dados.

Índice

1 Introdução **5**
1.1 Contextualização 5
1.2 Apresentação do Caso de Estudo 5
1.3 Estrutura do Relatório 6

2 Base de Dados Relacional **7**
2.1 Operacionalidade do Oracle 8

3 Bases de Dados NoSQL **11**
3.1 Base de Dados Orientada a Documentos - MongoDB 11
3.1.1 Migração de Dados 11
3.1.2 Operacionalidade do MongoDB 15
3.1.3 Modelo Relacional (Oracle) *vs* Modelo Não Relacional Documental (MongoDB) 20
3.2 Base de Dados Orientada a Grafos - Neo4j 20
3.2.1 Migração de Dados 21
3.2.2 Operacionalidade do Neo4j 24

4 Análise de Resultados **29**
4.1 Comparação Geral entre os Sistemas 29
4.2 Testes de Desempenho 29

5 Organização do Projeto **33**
5.1 Diretoria `migrate` 33
5.1.1 Ficheiro de Configuração 33
5.1.2 Subdiretoria `mongodb` 33
5.1.3 Subdiretoria `neo4j` 33
5.2 Diretoria `oracle` 34
5.3 Diretoria `performance` 34

6 Conclusões e Trabalho Futuro **35**

Índice de Figuras

2.1 Esquema da base de dados relacional 7

3.1 Esquema da base de dados não relacional (Neo4j) 21
3.2 Resultado da primeira consulta 25
3.3 Resultado da segunda consulta 25
3.4 Resultado da terceira consulta 26
3.5 Resultado da quinta consulta 27
3.6 Resultado da sexta consulta 28

4.1 Evolução do tempo de execução para a base de dados Oracle 30
4.2 Evolução do tempo de execução para a base de dados MongoDB 31
4.3 Evolução do tempo de execução para a base de dados Neo4j 32

Índice de Tabelas

1.1 SQL *vs* NoSQL 6

2.1 Resultado da primeira consulta 8
2.2 Resultado da segunda consulta 8
2.3 Resultado da terceira consulta 9
2.4 Resultado da quarta consulta 9
2.5 Resultado da quinta consulta 10
2.6 Resultado da sexta consulta 10

3.1 Resultado da quarta consulta 27

4.1 Tempos de execução médios para Oracle 30
4.2 Tempos de execução médios para MongoDB 30
4.3 Tempos de execução médios para Neo4j 31

1 Introdução

1.1 Contextualização

Atualmente, existe uma vasta quantidade de Sistemas de Gestão de Bases de Dados e, em muitas situações, é comum as empresas optarem pelo modelo não relacional devido às mudanças repentinas que os seus dados possam sofrer. Ora, nestas situações, a possibilidade de migrar os dados entre diferentes paradigmas de bases de dados (relacional e não relacional) torna-se bastante útil.

Neste trabalho prático, teremos de simular uma situação desse género e migrar os dados de uma base de dados relacional (HR do Oracle Database) para duas bases de dados não relacionais (Neo4j e MongoDB).

Deste modo, foi fundamental entender bem as várias entidades e relações da base de dados HR e conhecer bem as bases de dados não relacionais escolhidas para, assim, sermos capazes de desenvolver novos esquemas que respeitassem os dados iniciais, mas tendo sempre em conta as particularidades do novo paradigma.

Finalmente, de forma a salientar as maiores diferenças entre cada um dos motores e comparar o respetivo desempenho, implementámos um conjunto de operações que poderão ser consultadas mais à frente.

1.2 Apresentação do Caso de Estudo

O HR trata-se de um esquema de uma aplicação de Recursos Humanos criado pela Oracle cujo principal objetivo é armazenar os dados dos funcionários de uma organização.

Tal como já referimos anteriormente, com este trabalho, pretende-se migrar os dados de uma base de dados relacional para duas bases de dados não relacionais, mas, para isso, teremos de começar por assentar os conceitos acerca de ambos os paradigmas.

As principais diferenças a reter que serão úteis para o desenvolvimento do projeto apresentam-se na Tabela 1.1. [4]

	SQL	NoSQL
Tipo	Relacional	Não relacional
Dados	Dados estruturados armazenados em tabelas	Dados não estruturados armazenados em documentos, grafos ou tabelas chave-valor
Esquema	Estático	Dinâmico
Escalabilidade	Vertical	Horizontal
Linguagem	SQL	Sem linguagem de consulta padrão
Transações	Seguem os princípios do ACID	Não seguem os princípios do ACID

Tabela 1.1: SQL *vs* NoSQL

1.3 Estrutura do Relatório

O presente documento é composto por seis capítulos:

1. **Introdução** - Neste capítulo, é apresentado o caso de estudo acompanhado da contextualização do trabalho prático, onde damos a conhecer o problema proposto;

2. **Base de Dados Relacional** - Neste capítulo, apresentamos a base de dados Oracle que nos foi fornecida, bem como a descrição das suas entidades, atributos e relações. Ainda neste capítulo, apresentamos as várias interrogações implementadas que, mais à frente, servirão para comparar o desempenho dos vários modelos;

3. **Bases de Dados NoSQL** - Este capítulo contém uma breve explicação do motivo pelo qual optámos pelas bases de dados MongoDB e Neo4j. É aqui que descrevemos as estratégias implementadas para migrar os dados e apresentamos as interrogações adaptadas aos novos modelos;

4. **Análise de Resultados** - Aqui, comparamos as várias bases de dados em termos de desempenho, quer seja com a apresentação das vantagens e desvantagens da utilização de cada uma delas, quer seja com a apresentação de gráficos que comparam os respetivos tempos de execução;

5. **Organização do Projeto** - No penúltimo capítulo, explicamos toda a organização do projeto, ou seja, como estão organizadas as diretorias, subdiretorias e ficheiros para facilitar a pesquisa de um determinado documento;

6. **Conclusão e Trabalho Futuro** - Por último, neste capítulo, analisamos o trabalho realizado, tiramos conclusões acerca dos resultados obtidos e apresentamos algumas sugestões daquilo que poderia vir a ser implementado no futuro.

2 Base de Dados Relacional

O SGBD relacional HR é um sistema de base de dados relacional de Recursos Humanos, criado pela Oracle, que tem como objetivo principal armazenar os dados dos funcionários de uma organização.

Esta base de dados possui 7 tabelas:

- **`EMPLOYEES`:** Dados dos funcionários, tais como nome, departamento e cargo atual. Os funcionários podem ou não estar vinculados a um departamento;
- **`DEPARTMENTS`:** Dados dos departamentos em que funcionários podem trabalhar;
- **`REGIONS`:** Dados sobre os locais, endereços dos escritórios, armazéns ou locais de produção da organização;
- **`LOCATIONS`:** Dados sobre os locais ou endereços dos escritórios, armazéns ou locais de produção da organização;
- **`COUNTRIES`:** Dados sobre os países em que a organização atua;
- **`JOBS`:** Dados sobre os tipos de cargos que os funcionários podem ocupar;
- **`JOB_HISTORY`:** Histórico dos cargos anteriores ocupados pelos funcionários dentro da organização.

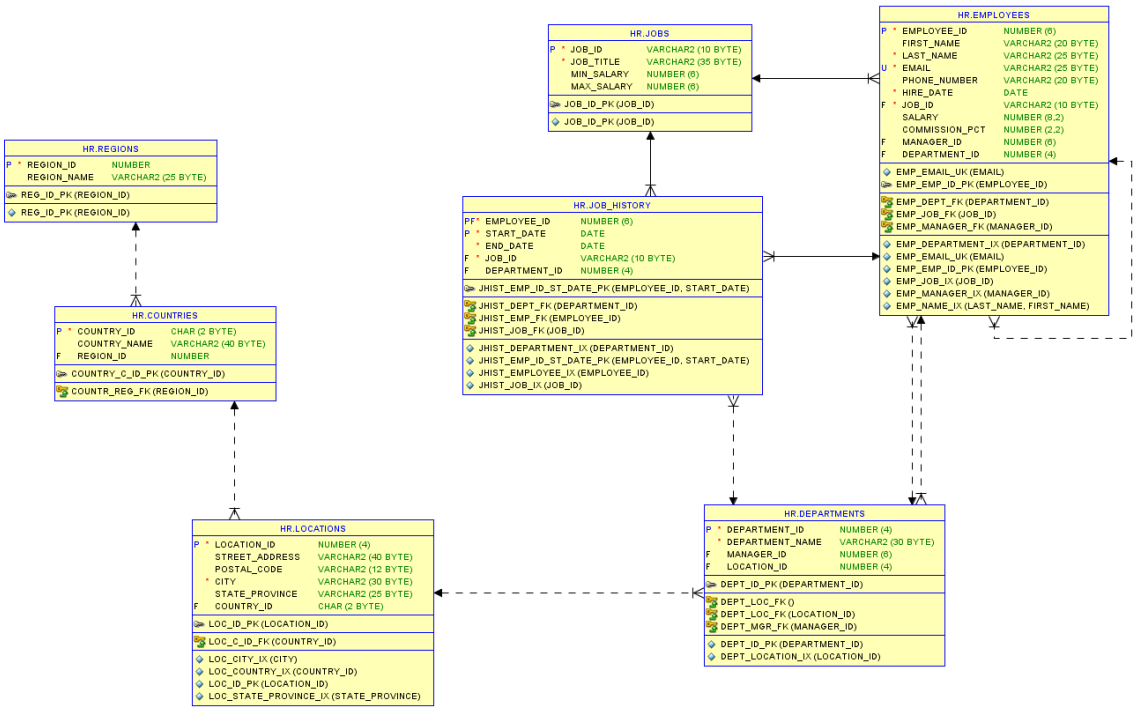

Figura 2.1: Esquema da base de dados relacional

2.1 Operacionalidade do Oracle

Com vista a demonstrarmos a operacionalidade deste SGBD relacional, desenvolvemos interrogações adequadas ao contexto em que este está inserido. Seguem-se, então, alguns exemplos:

1. **Número de funcionários que trabalham no Canadá:**

```
select count(*) as "Number of employees who work in
   Canada"
from countries c
    join locations l on c.country_id = l.country_id
        join departments d on l.location_id = d.
   location_id
            join employees e on d.department_id = e.
   department_id
    where country_name = 'Canada';
```

   Esta operação junta 3 tabelas distintas (`COUNTRIES`, `LOCATIONS`, `DEPARTMENTS` e `EMPLOYEES`) pelo que tem uma complexidade elevada e, como consequência, pode tornar-se pouco eficiente para grandes quantidades de dados. Segue-se o resultado da mesma:

	Number of employees who work in Canada
1	2

Tabela 2.1: Resultado da primeira consulta

2. **Primeiro e último nome das pessoas cujo emprego é Vice-Presidente da Administração:**

```
select first_name, last_name from employees e
    join jobs j on e.job_id = j.job_id
    where job_title = 'Administration Vice President';
```

   Com esta consulta, juntamos duas tabelas: `Jobs` e `Employees`. Desta forma, é natural que, neste modelo, obtenhamos um desempenho possivelmente inferior. Contudo, visto que só efetuamos a junção de duas tabelas, pode acontecer que o modelo relacional seja mais eficiente do que os não relacionais, dependendo das suas estruturas de dados. Observemos, então, o resultado obtido nesta operação:

	First Name	Last Name
1	Neena	Kochhar
2	Lex	De Haan

Tabela 2.2: Resultado da segunda consulta

3. **Funcionários que tenham tido mais do que um trabalho antes:**

```
select e.first_name, e.last_name from employees e
    join job_history j on e.employee_id = j.employee_id
        group by e.first_name, e.last_name
        having count(j.job_id) > 1;
```

Através da agregação de tabelas, seguido de um `group by`, esta operação tem um custo elevado a nível de desempenho da sua execução. Desta forma, é necessário observar se o uso de modelos não relacionais contornam isto, possibilitando uma abordagem mais eficiente. É possível notar os dados resultantes desta consulta de seguida:

	First Name	Last Name
1	Neena	Kochhar
2	Jonathon	Taylor
3	Jennifer	Whalen

Tabela 2.3: Resultado da terceira consulta

4. **Nome de cada departamento e média dos seus salários com duas casas decimais:**

```
select department_name, round(avg(salary), 2) as
"Mean of the department's salaries"
from departments d
    join employees e on d.department_id = e.
   department_id
        group by department_name;
```

Novamente, através de uma junção de tabelas e de uma agregação, é necessário comparar, numa fase posterior, o desempenho deste modelo relacional com os não relacionais. Esta operação retorna os seguintes dados:

	Department Name	Mean of the department's salaries
1	Administration	4400
2	Accounting	10154
3	Purchasing	4150
4	Human Resources	6500
5	IT	5760
...	...	...

Tabela 2.4: Resultado da quarta consulta

5. **Identificador e nome da rua dos funcionários cujo salário é superior a 10 mil unidades monetárias:**

```
select employee_id, street_address from employees e
    join departments d on e.department_id = d.
   department_id
        join locations l on d.location_id = l.
   location_id
        where salary > 10000;
```

Através desta operação, pretendemos analisar a eficiência da junção de três tabelas num modelo relacional e consequente comparação com os não relacionais desenvolvidos. Com isto, obtemos os seguintes campos como resultado da consulta:

	Employee Id	Street Address
1	100	2004 Charade Rd
2	101	2004 Charade Rd
3	102	2004 Charade Rd
4	108	2004 Charade Rd
5	114	2004 Charade Rd
...	...	...

Tabela 2.5: Resultado da quinta consulta

6. **Nome e número de departamentos da região que possui o maior número de departamentos:**

```
select region_name, count(*) as "Number of departments"
from regions r
    join countries c on r.region_id = c.region_id
        join locations l on l.country_id = c.country_id
            join departments d on d.location_id = l.
   location_id
            group by region_name
            order by "Number of departments" desc
            fetch first 1 rows only;
```

Por último, esta operação é a mais complexa para este modelo. Esta junta 4 tabelas e, de seguida, procede ao `group by` pelo nome da região. Deste modo, é interessante avaliarmos as diferenças de desenvolvimento desta consulta para as bases de dados NoSQL. Desta forma, conseguimos obter o resultado seguinte:

	Region Name	Number of departments
1	Americas	24

Tabela 2.6: Resultado da sexta consulta

3 Bases de Dados NoSQL

3.1 Base de Dados Orientada a Documentos - MongoDB

Uma base de dados orientada a documentos está desenhada para gerir dados semi-estruturados, ou seja, não é definido, *a priori*, um esquema fixo para os mesmos, pois os próprios dados é que definem a estrutura dos objetos e coleções lá armazenadas.

Devido a este facto, uma base de dados como esta não será indicada para guardar relações entre objetos, fazendo com que a definição das "regras de negócio" que incluem restrições de dados ou valores únicos, por exemplo, se mantenham no modelo relacional apropriado para isso mesmo.

A implementação de um conceito como este e, em particular no MongoDB, introduz a possibilidade de ignorar atributos que não têm valor, o que num modelo relacional faria com que se armazenasse um valor nulo.

A redundância dos dados e a flexibilidade de armazenamento centralizado num único objeto ao invés de múltiplas tabelas distribuídas, assim como as razões apresentadas anteriormente, definem a nossa escolha de, neste caso, uma base de dados baseada em documentos, como o MongoDB, permitindo muito melhor desempenho em consultas, desde que não se caia no erro de tentar estabelecer relações entre objetos/documentos.

3.1.1 Migração de Dados

Esta fase passou por tomar um conjunto de decisões de modo a garantir uma série de propriedades que achamos importantes quando migramos dados para sistemas baseados em documentos:

1. Flexibilidade na representação dos dados: Não devemos, de todo, esquecer este tópico, que inclui uma definição dos dados numa ótica de escalabilidade horizontal, ou seja, menos consistência e consequente redundância de dados sem estrutura fixa;

2. Os dados formam o esquema: A segunda questão que colocamos quando pensamos em migrar para um sistema como estes é: Que dados migrar? Todas as tabelas? Que informação é relevante? A verdade é que estamos a estudar um sistema relacional de dados acerca de Recursos Humanos e, portanto, pode não ser vantajoso guardar toda a informação (especialmente aquela que não é usada);

Com base numa frase do enunciado, conseguimos extrair o ponto principal da utilização desta base de dados, sendo este: "*(...) objetivo principal armazenar os dados dos empregados (...)*" e, portanto, focando o desenvolvimento neste facto, decidimos

proceder à definição de um esquema que extraísse toda a informação disponível de um funcionário e a compactasse num documento e, com os documentos conjugados, definimos a coleção `employees`. De seguida, apresentamos um exemplo de um objeto que conjuga toda a informação de um registo para um funcionário:

```
{
    _id: ObjectId('600f34ac75fc4ee1416b05e9'),
    job_title: 'Administration Vice President',
    job_min_salary: 15000,
    job_max_salary: 30000,
    first_name: 'Lex',
    last_name: 'De Haan',
    email: 'LDEHAAN',
    phone_number: '515.123.4569',
    hire_date: '13-01-2001',
    salary: 17000,
    comission_pct: null,
    curr_dep_manager_email: 'SKING',
    curr_dep_name: 'Executive',
    curr_dep_loc_st_addr: '2004 Charade Rd',
    curr_dep_loc_pt_code: '98199',
    curr_dep_loc_city: 'Seattle',
    curr_dep_loc_st_prov: 'Washington',
    curr_dep_country_name: 'United States of America',
    curr_dep_region_name: 'Americas',
    hist:
    [
        {
            hist_start_date: '13-01-2001',
            hist_end_date: '24-07-2006',
            hist_job_title: 'Programmer',
            hist_dep_name: 'IT',
            hist_dep_loc_st_addr: '2014 Jabberwocky Rd',
            hist_dep_loc_pt_code: '26192',
            hist_dep_loc_city: 'Southlake',
            hist_dep_loc_st_prov: 'Texas',
            hist_dep_country_name: 'United States of
   America',
            hist_dep_region_name: 'Americas'
        }
    ]
}
```

A partir deste exemplo conseguimos extrair a estrutura base de todos os documentos da nossa representação não relacional (documental) do modelo relacional inicial e acompanhamos o seguinte exemplo com excertos de pseudo-código extraído diretamente da nossa implementação em Python:

1. Estabelecer conexão ao Oracle e obter todas as tabelas:

```
c = create_connection(user, uri, ...).cursor()

# get all relevant tables & store in memory

employees = c.execute("select * from employees")
jobs = c.execute("select * from jobs")
# ...

for row in jobs:
    job_id = row[0] # first column == job id
    # create documents
    for emp in employees:
        # process this employee if
        # the current job matches
        DOCUMENT = {}

```

2. Para todos os empregos existentes, criar registos (objetos JSON), compactando toda a informação:

   (a) Registos de informação pessoal - todos os objetos começam por conter a informação pessoal existente na tabela em Oracle denominada por `EMPLOYESS` e a informação é obtida diretamente, sem consultar outras tabelas:

```
DOCUMENT["first_name"] = emp[1]
DOCUMENT["last_name"]  = emp[2]
DOCUMENT["email"]      = emp[3]
#...
```

   (b) Registos do emprego atual - esta próxima consulta envolve registar o emprego atual de um funcionário. Através do ciclo mais exterior obtivemos essa informação:

```
DOCUMENT["job_title"]       = jb[1]
DOCUMENT["job_min_salary"]  = jb[2]
#...
```

   (c) Registos dos departamentos e a sua localização, caso exista - neste caso, podemos ver uma desvantagem existente na nossa abordagem, que pode ser

até algo irrelevante, visto que adotámos um modelo onde damos importância aos registos de funcionários e não propriamente à "informação solta" na base de dados:

```
depart_id = emp[10]

if not (depart_id is None):
    # note: curr == current
    DOCUMENT["curr_dep_manager_email"] = ...
    DOCUMENT["curr_dep_name"]          = ...
    #...
    if not (country_id is None):
        # get country atributes
        #...
        if not (region_id is None):
            # get region atributes
            #...
```

Exemplos de informação solta podem incluir departamentos sem localização definida, sem país ou até que não pertençam a nenhuma região definida na base de dados. Quer isto dizer que toda a informação não ligada aos funcionários não é definida por nós como relevante para as consultas que foram pensadas e para o objetivo principal do modelo relacional apresentado (obter informações de Recursos Humanos).

(d) Registos de empregos antigos e todas as informações ligadas aos mesmos, maioritariamente, já analisadas anteriormente:

```
DOCUMENT['hist'] = []

for jb_hist in jobHistory:

    HIST_OBJ["hist_start_date"] = jb_hist[1]
    HIST_OBJ["hist_end_date"]   = jb_hist[2]
    # ...
    depart_id = jb_hist[4]
    if not (depart_id is None):

        # same process as in point (c)
```

O processamento do histórico só é feito se um dado funcionário tiver registos de empregos antigos na tabela do modelo relacional `JOB_HISTORY`, sendo este guardado sob a forma de um *array* de objetos. O exemplo apresentado inicialmente mostra um caso em que um funcionário com o primeiro nome `Lex` está a trabalhar atualmente no departamento `Executive`, mas que já trabalhou em `IT` como programador.

3. Migrar todos estes documentos gerados para o motor de bases de dados MongoDB:

```
client = MongoClient(uri, ...)

# switch to created db
db = client.hr_migrate

for doc in GEN_DOCUMENTS:
    db.employees.insert_one(doc)

    # close connections...
```

A migração dos dados propriamente dita é muito direta e passa por, simplesmente, inserir cada um dos documentos de dados compactados na base de dados `hr_migrate`, para a coleção `employees` criada.

3.1.2 Operacionalidade do MongoDB

Com vista a demonstrarmos a operacionalidade deste SGBD não relacional, desenvolvemos interrogações adequadas ao contexto em que este está inserido. Seguem-se, então, alguns exemplos:

1. **Número de funcionários que trabalham no Canadá:**

```
db.employees.find(
{
    "curr_dep_country_name": "Canada"
}).size()
```

   De forma genérica, percebemos que ter os dados compactados é, essencialmente, uma vantagem, visto que, aqui, a procura envolve filtrar todos os documentos por uma "chave" ao invés do modelo relacional, que conjuga 4 tabelas nesta pesquisa, sendo elas `LOCATIONS`, `COUNTRIES`, `DEPARTMENTS` e `EMPLOYEES` e que tem problemas evidentes de carregar todos estes dados para memória, balancear os custos de escolher um plano de interrogação X ou Y, calcular estatísticas e todas estas particularidades de um modelo relacional.

   O resultado da consulta anterior pode ser observado de seguida e equivale ao valor observado em Oracle:

```
> db.employees.find(<query1>).size()
2
```

2. **Primeiro e último nome das pessoas cujo emprego é Vice-Presidente da Administração:**

```
db.employees.find(
{
    "job_title": "Administration Vice President"
},
{
    "First Name": "$first_name",
    "Last Name": "$last_name",
    "_id": 0
}).pretty()
```

Em comparação com a versão relacional, esta consulta surge apenas como um exemplo que junta duas tabelas (`JOBS` e `EMPLOYEES`), mas onde, ainda assim, é muito mais vantajoso ter os dados compactados como em MongoDB para pesquisas eficientes.

O resultado desta consulta, da mesma forma que a anterior, produz o mesmo resultado que em Oracle:

```
> db.employees.find(<query2>).pretty()
{ "First Name" : "Neena", "Last Name" : "Kochhar" }
{ "First Name" : "Lex", "Last Name" : "De Haan" }
```

3. **Funcionários que tenham tido mais do que um trabalho antes:**

```
db.employees.find(
{
    $where: "this.hist.length > 1"
},
{
    "First Name": "$first_name",
    "Last Name": "$last_name",
    "_id": 0
}).pretty()
```

Neste cenário, temos uma consulta que, em geral, beneficia mais tanto deste modelo não relacional documental como de um baseado em grafos como o que apresentamos na próxima secção. Neste caso, por cada funcionário, guardamos uma lista de objetos/registos de empregos anteriores e, por isso, é muito mais simples apenas olhar para o tamanho do *array* armazenado do que juntar 2 ou 3 tabelas.

O resultado desta consulta, da mesma forma que as anteriores, produz o mesmo resultado que em Oracle:

```
> db.employees.find(<query3>).pretty()
{ "First Name" : "Neena", "Last Name" : "Kochhar" }
{ "First Name" : "Jennifer", "Last Name" : "Whalen" }
{ "First Name" : "Jonathon", "Last Name" : "Taylor" }
```

4. **Nome de cada departamento e média dos seus salários com duas casas decimais:**

```
db.employees.aggregate([
{
    $group:
    {
        _id: "$curr_dep_name",
        avgAmount:
        {
            $avg: "$salary"
        }
    }
},
{
    $project:
    {
        _id: 0,
        "Departament": "$_id",
        "Average Salary":
        {
            $round: ["$avgAmount", 2]
        }
    }
}
]).pretty()
```

A implementação desta consulta foi mais complicada do que as anteriores, pois, como todos os documentos armazenados estão definidos por funcionário e não por departamento, é preciso fazer um `group` inicial dos dados por nome de departamento e, depois, aplicar uma função de agregação `avg` ao atributo `salary` de um funcionário. Este processamento, em termos de custo total, pode trazer resultados parecidos ao modelo relacional e, portanto, não constitui um caso em que a diferença de desempenho seja tão evidente.

O resultado desta consulta produz o seguinte resultado, onde foram omitidas algumas entradas do mesmo:

```
> db.employees.find(<query4>).pretty()
{"Departament": null, "Average Salary": 7000}
{"Departament": "Finance", "Average Salary": 8601.33}
{"Departament": "Purchasing", "Average Salary": 4150}
{"Departament": "IT", "Average Salary": 5760}
# ...other results
```

Neste caso, o resultado indica o valor das médias de salários dos funcionários que pertencem aos departamentos indicados. No entanto, como resultado extra, indica-nos também a média de salários dos funcionários que não têm departamento definido, valor esse que deixámos ficar por mera curiosidade, já que em Oracle o `join` é feito considerando entradas não nulas.

5. **Identificador e nome da rua dos funcionários cujo salário é superior a 10 mil unidades monetárias:**

```
db.employees.find(
{
    "salary":
    {
        $gt: 10000
    }
},
{
    "E-mail": "$email",
    "Street Address": "$curr_dep_loc_st_addr",
    _id: 0
}).pretty()
```

Este exemplo é outro que constitui uma enorme vantagem em relação a modelos relacionais[1], visto que, aqui, apenas é utilizado um filtro, que ainda poderia ser mais otimizado se optássemos por definir índices para a coluna `salary`. O resultado obtido foi o seguinte:

```
> db.employees.find(<query5>).pretty()
{"E-mail":"SKING","Street Address":"2004 Charade Rd"}
{"E-mail":"NKOCHHAR","StreetAddress":"2004 Charade Rd"}
{"E-mail":"LDEHAAN","Street Address":"2004 Charade Rd"}
# ...other results
```

[1]No modelo relacional ocorre a junção de 3 tabelas em simultâneo

6. **Nome e número de departamentos da região que possui o maior número de departamentos:**

```
db.employees.aggregate([
{
    $group:
    {
        _id: "$curr_dep_region_name",
        dep_name:
        {
            $addToSet: "$curr_dep_name"
        }
    }
}]).pretty()
```

Esta última operação afirma-se como o principal fator comparativo entre os nossos modelos não relacionais e constitui um exemplo em que o resultado obtido é mais reduzido do que recorrendo a um modelo por grafos (ou até mesmo a um modelo relacional), devido às escolhas adotadas na esquematização genérica, que excluem informação que não esteja ligada aos funcionários. Assim, o resultado obtido foi o seguinte[2]:

```
> db.employees.find(<query6>).pretty()
{
  "_id" : "Europe",
  "dep_name" : [
    "Sales",
    "Human Resources",
    "Public Relations"
  ]
}
{ "_id" : null, "dep_name" : [ ] }
{
  "_id" : "Americas",
  "dep_name" : [
    "Administration",
    "IT",
    "Shipping",
    "Accounting",
    "Marketing",
    "Executive",
    "Purchasing",
    "Finance"
  ]
}
```

[2]De notar que o resultado apresentado para Oracle é apenas do *top* 1. Todavia, achámos por bem colocar a enumeração dos departamentos para explicar a diferença de resultados

Deste modo, os departamentos apresentados são apenas 11 dos 27 existentes. A região `Americas` continua a ser a que contém mais departamentos, no entanto, em Oracle obtivemos 24 ao invés dos 8 apresentados aqui.

3.1.3 Modelo Relacional (Oracle) *vs* Modelo Não Relacional Documental (MongoDB)

Como já foi referido ao longo da explicação de cada consulta, as principais diferenças entre o modelo documental adotado e o relacional estão na representação dos dados, sem estrutura fixa, sendo o esquema formado pelos dados/documentos que armazenam toda a informação dos funcionários. Desta forma, outras informações menos relevantes (como as regiões, países, entre outros) só existem se, no fundo, estiverem ligadas a funcionários, o que pode ou não ser visto como uma desvantagem neste modelo, mas que, a nosso ver, faz cumprir o objetivo principal da base de dados HR.

3.2 Base de Dados Orientada a Grafos - Neo4j

Uma base de dados orientada a grafos é uma base de dados desenhada para tratar das relações entre os dados e os próprios dados com a mesma importância. Destina-se a conter dados sem restringi-los a um modelo predefinido (tal como já foi referido na Tabela 1.1). Em vez disso, os dados são armazenados tal como os extraímos, mostrando como cada entidade individual se liga ou relaciona com as outras.

Enquanto as bases de dados relacionais calculam relações a partir de operações de `join` bastante custosas em termos de desempenho, uma base de dados de grafos armazena as relações juntamente com os dados no modelo. Assim sendo, aceder a nodos e relações numa base de dados de grafos torna-se eficiente e permite atravessar milhões de ligações por segundo por *core*.

No nosso caso de estudo, optámos por escolher Neo4j como a base de dados orientada a grafos que iremos utilizar, pelas seguintes razões:

- Possui uma linguagem de consulta declarativa (Cypher) otimizada para grafos semelhante a SQL, o que facilita bastante o processo de familiarização com a sintaxe;
- Executa travessias em tempo constante em grandes grafos tanto para profundidade como para largura, devido à representação eficiente dos nodos e relações;
- Possui um esquema de grafos flexível capaz de se adaptar ao longo do tempo, tornando possível adicionar novas relações quando a empresa sofrer mudanças;
- Permite escalonar biliões de nodos em *hardware* moderado;
- Possui *drivers* para linguagens de programação populares, incluindo Python (que foi a linguagem adotada por nós).

3.2.1 Migração de Dados

Para que fosse possível migrar a base de dados HR de um modelo relacional para um modelo não relacional (orientado a grafos) tirando proveito das funcionalidades do último, foi necessário fazer algumas alterações no esquema inicial.[3]

Tal como podemos observar na Figura 2.1, no esquema inicial existe uma tabela `JOB_HISTORY` que armazena os empregos passados dos funcionários. Contudo, neste novo esquema adaptado ao Neo4j, optámos por não representar essa tabela como um nodo e, em vez disso, decidimos colocar os atributos relevantes da mesma (`start_date` e `end_date`) na relação entre os nodos que representam as tabelas `JOB` e `EMPLOYEES`, ou seja, a partir dessa relação, podemos saber a data em que um funcionário começou a/terminou de desempenhar um determinado emprego. Claro está que o atributo `end_date` apenas se encontrará preenchido para os empregos passados, pois, nos outros casos, encontrar-se-á a `null`. Para além disso, e visto que um mesmo emprego pode pertencer a departamentos distintos, teremos também de colocar o atributo `department_id` nessa relação, de modo a podermos saber a qual departamento pertence um emprego passado.

Na Figura 3.1, pode ser visto o esquema adaptado ao Neo4j. De notar que todas as restantes tabelas e relações foram convertidas diretamente para nodos e arcos, respetivamente.

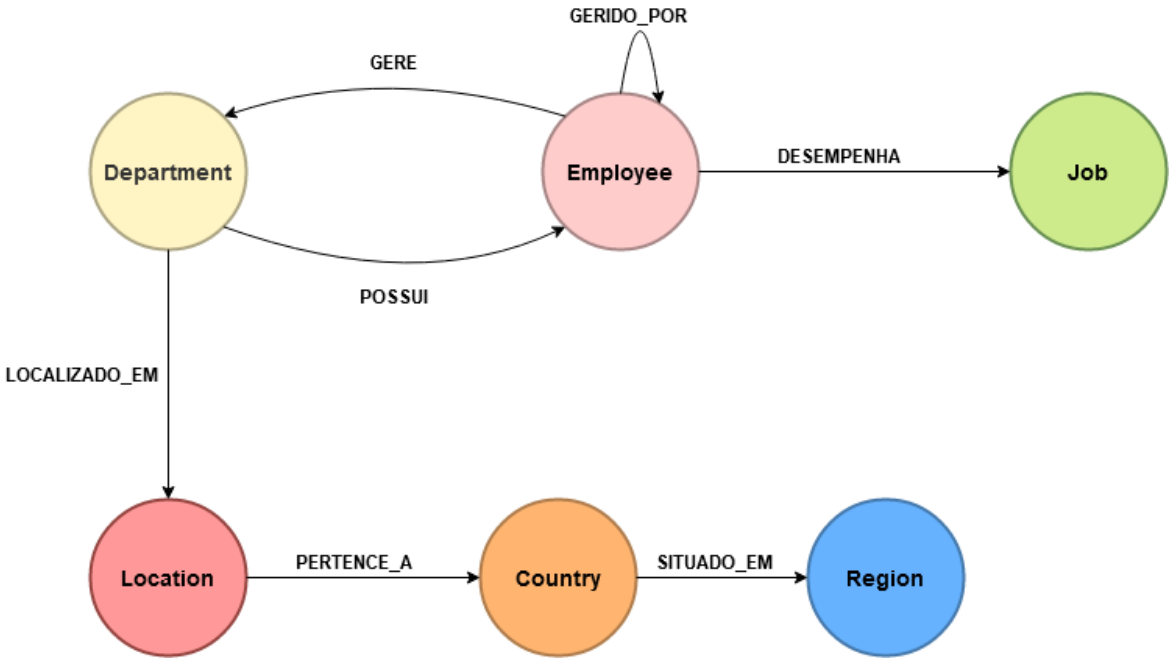

Figura 3.1: Esquema da base de dados não relacional (Neo4j)

O processo de migração, neste caso, é feito em duas fases:

1. Criação dos ficheiros de dados para o modelo:

   Para esta fase, definimos o ficheiro `init.sql` que contém todas as relações do modelo original representadas sob a forma de consultas que dão origem aos CSV finais, através de uma estruturação do ficheiro deste modo:

```
select * from ... | csv_file
```

Em Python, mais uma vez, procedemos à leitura deste ficheiro e geramos, para cada consulta, o ficheiro CSV respetivo com o nome indicado a seguir ao separador `|`, como podemos ver através do seguinte algoritmo:

```
file = open(init.sql,"r") # read
for line in file.readlines():
    if not (line.startswith("\n") or len(line)==0):
        parts = line.split("|")
        query = parts[0]
        csv   = parts[1]
        create_csv(query, csv) # generate csv file
```

Cada um dos ficheiros gerados corresponde aos nodos estabelecidos e a relações entre os mesmos para o modelo Neo4j apresentado anteriormente. No entanto, devemos dar uma especial atenção à última consulta presente nesse ficheiro, com o nome de ficheiro CSV resultante `EMP_RELATION`, como se pode ver de seguida:

```
select
    e.EMPLOYEE_ID,
    e.JOB_ID as "CurrentJob",
    j.JOB_ID as "PreviousJob",
    e.HIRE_DATE as "HireDate",
    j.START_DATE as "HistStartDate",
    j.END_DATE as "HistEndDate",
    e.DEPARTMENT_ID as "CurrentDep",
    j.DEPARTMENT_ID as "PreviousDep"
from EMPLOYEES e
full outer join JOB_HISTORY j
on e.EMPLOYEE_ID = j.EMPLOYEE_ID | EMP_RELATION
```

O ficheiro de dados final correspondente a esta consulta representa um ponto de partida para a geração do ficheiro `employees_relationship.csv` que contém todas as ligações entre os nodos `Job` e os nodos `Employee`, onde, por exemplo, cada ligação representa um trabalho atual ou passado (colocando o atributo `end_date` a `null` ou não, respetivamente) e especificando o `department_id` na relação, para saber em que departamento trabalha/trabalhou. No entanto, devido à utilização de um `full outer join`, temos algum trabalho de tratamento de dados a fazer:

```
# manage employee relationship
for row in EMP_REL:
    # check if end date is null
    if not (row[5] is None):
        R["end_date"] = row[5]
    else:
        R["end_date"] = "null"
```

```
    # manage jobs
    if (row[2] is None):
        # this row represents the current job
        R["JOB_ID"] = row[1]
         # hire_date form EMPLOYEE
        R["START_DATE"] = row[4]
        # ...
    else:
        # ...previous job
        R["JOB_ID"] = row[2]
         # start_date from JOB_HISTORY
        R["START_DATE"] = row[4]
        # ...
```

O processamento envolve, para além do indicado no excerto acima, atribuir o identificador do departamento na relação. No entanto, temos dois cenários possíveis: no caso de ser um trabalho atual, usamos o da relação com `EMPLOYEE`, já no outro caso, usamos o da relação com `JOB_HISTORY`. Após a geração desta estrutura (em memória), resta-nos apenas converter o dicionário para uma estrutura CSV, como foi feito até agora.

2. Migração dos dados para o sistema Neo4j:

   Esta segunda fase torna-se simples, visto que temos todos os ficheiros de dados criados. Contudo, é necessário estabelecer as relações em Cypher que "dão vida" aos dados gerados para os nodos. Assim sendo, começamos por definir o ficheiro `import.cypher` que define essas regras. Tomemos como exemplo a relação de funcionários com trabalhos atuais/antigos explicada anteriormente:

```
# syntax barely changed for better comprehension
LOAD CSV WITH HEADERS FROM ... AS csvLine
MATCH (
    e:Employee {employee_id: csvLine.EMPLOYEE_ID}),
    (j:Job {job_id: csvLine.JOB_ID}
)
CREATE
    (e)-[:DESEMPENHA {start_date: csvLine.START_DATE,
    end_date: csvLine.END_DATE,
    department_id: csvLine.DEPARTMENT_ID}]->(j);
```

   Deste modo, conseguimos indicar ao Neo4j que queremos conectar os nodos `Employee` e `Job` através da relação `DESEMPENHA` que contém como atrbutos o identificador do departamento e as datas de início e fim, sendo a final nula caso se trate do emprego atual.

A migração dos dados propriamente dita passa por ler o ficheiro referido anteriormente e executar cada uma das consultas Cypher:

```
# create connection
client = GraphDatabase.driver(uri, ...)

# process import.cypher
# ...open, read, parse, extract queries

with client.session() as session:
    for query in queries:
        session.run(query)

# close connection
client.close()
```

3.2.2 Operacionalidade do Neo4j

Com vista a demonstrarmos a operacionalidade deste SGBD não relacional, desenvolvemos interrogações adequadas ao contexto em que este está inserido.

O elevado desempenho deste tipo de base de dados deve-se à existência de um nodo conter direta e fisicamente uma lista de registos de relacionamento que representam os seus relacionamentos com outros nodos. Portanto sempre que é feita uma operação equivalente ao `join`, a base de dados orientada a grafos apenas usa essa lista e tem acesso direto aos nós conectados, eliminando a necessidade de um cálculo custoso de pesquisa e correspondência.

Seguem-se, então, alguns exemplos:

1. **Número de funcionários que trabalham no Canadá:**

```
match (c: Country)<-[: PERTENCE_A]-(l: Location)<-[:
   LOCALIZADO_EM]-(d: Department)-[: POSSUI]->(e:
   Employee)
    where c.country_name = 'Canada'
        return count(e) as 'Number of employees who
   work in Canada'
```

   Como já foi referido anteriormente, como cada nodo contém uma lista de relações para a execução desta consulta, é apenas necessário encontrar os diferentes caminhos percorrendo as relações desde um nodo `Employee` até ao nodo `Country` cujo nome é Canadá. Isto pode ser observado em seguida:

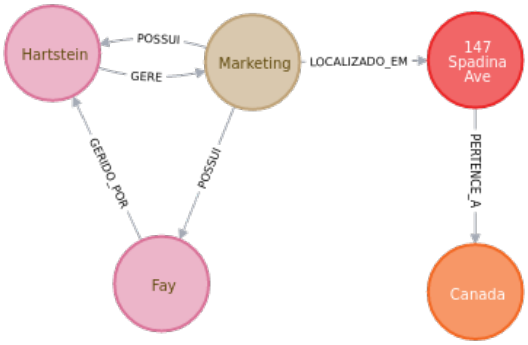

Figura 3.2: Resultado da primeira consulta

Feito isto, é apenas necessário contar os nodos de funcionários existentes. Apesar de parecerem ser feitas as mesmas visitas em relação à base de dados Oracle, o desempenho para um grande volume de dados será drasticamente superior.

2. **Primeiro e último nome das pessoas cujo emprego é Vice-Presidente da Administração:**

```
match (e: Employee)-[d: DESEMPENHA]->(j: Job)
    where d.end_date is NULL
        and j.job_title = 'Administration Vice President'
            return e.employee_first_name, e.employee_last_name
```

Este exemplo é muito semelhante ao anterior para a implementação desta base de dados, a única diferença agora é que se procura os percursos desde o nodo `Employee` até ao nodo `Job` que contém o nome Vice-Presidente da Administração. O resultado da consulta pode ser observado de seguida:

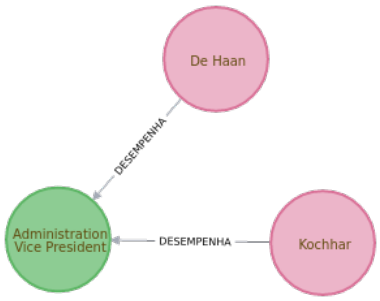

Figura 3.3: Resultado da segunda consulta

3. **Funcionários que tenham tido mais do que um trabalho antes:**

```
match (e: Employee)-[d: DESEMPENHA]->(j: Job)
    where d.end_date is not null
        with e.employee_first_name as first_name,
```

```
4               e.employee_last_name as last_name,
5               count(*) as n_jobs
6                   where n_jobs > 1
7                       return first_name, last_name
```

Este já é um exemplo mais interessante, visto que, em relação à base de dados Oracle, foi suprimida a tabela `JOB_HISTORY`. A informação necessária para a realização desta consulta encontra-se, agora, na relação entre o nodo `Job` e o nodo `Employee`. Portanto para a resolução da mesma é apenas necessário encontrar as relações existentes entre estes dois nodos e verificar quais aqueles que têm mais do que uma ocorrência de um trabalho realizado, ou seja, o atributo `end_date` necessita de ter o valor diferente de `null` nesta relação. O resultado obtido pode ser observado em seguida:

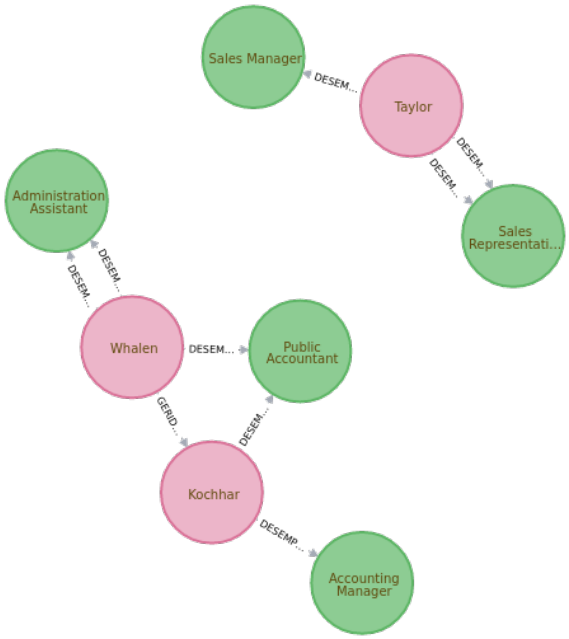

Figura 3.4: Resultado da terceira consulta

4. **Nome de cada departamento e média dos seus salários com duas casas decimais:**

```
1 match (e: Employee)<-[: POSSUI]-(d: Department)
2     return d.department_name, round(10^2 * avg(
    toInteger(e.employee_salary)))/10^2 as 'Mean of the
    department's salaries'
```

Para este exemplo é necessário encontrar todos os percursos existentes entre um nodo `Employee` e um nodo `Department` e fazer uma média dos seus salários pelo departamento a que estes pertencem. Os resultados são apresentados de seguida:

d.department_name	Mean of the department's salaries
"Administration"	4400.0
"Marketing"	9500.0
"Purchasing"	4150.0
"Human Resources"	6500.0
"Shipping"	3475.56
...	...

Tabela 3.1: Resultado da quarta consulta

5. **Identificador e nome da rua dos funcionários cujo salário é superior a 10 mil unidades monetárias:**

```
match (l: Location)<-[: LOCALIZADO_EM]-(d: Department)
   -[: POSSUI]->(e: Employee)
    where toInteger(e.employee_salary) > 10000
        return e.employee_id, l.street_address
```

Agora, é necessário encontrar todos os percursos entre o nodo `Employee` e o nodo `Location`, com um filtro para o nodo dos funcionários, ou seja, mantendo apenas as relações e os nodos dos funcionários com um salário superior a 10 mil unidades monetárias. Com isto, é apenas necessário apresentar os atributos `employee_id` e `street_address`. O resultado pode ser observado em seguida:

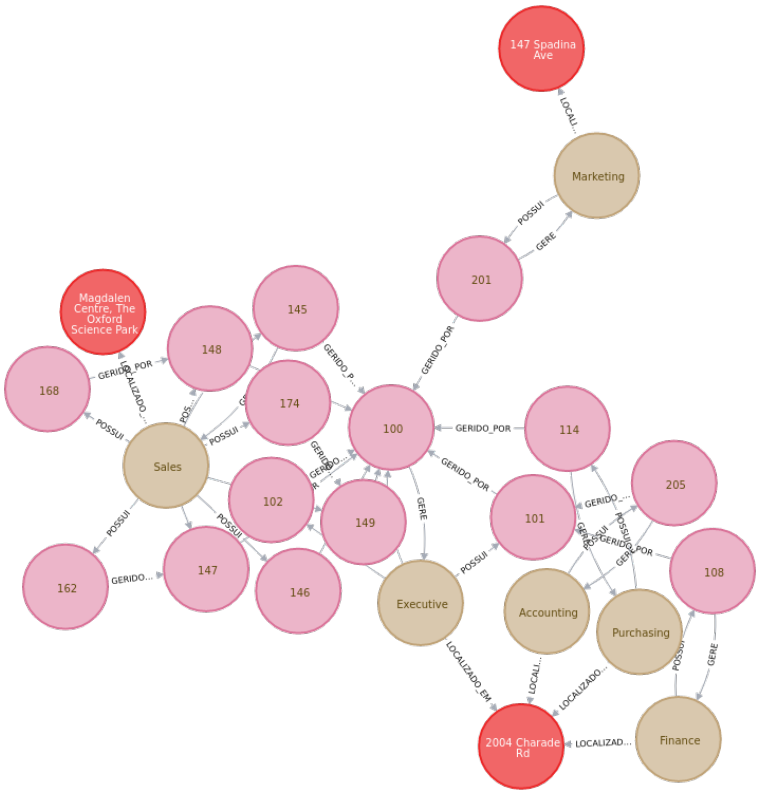

Figura 3.5: Resultado da quinta consulta

6. **Nome e número de departamentos da região que possui o maior número de departamentos:**

```
match (r: Region)<-[: SITUADO_EM]-(c: Country)<-[:
   PERTENCE_A]-(l: Location)<-[: LOCALIZADO_EM]-(d:
   Department)
    return r.region_name, count(d) as 'Number of
   departments'
        order by count(d) desc limit 1
```

Idêntico aos exemplos anteriores, neste caso, é necessário encontrar todos os percursos existentes entre o nodo `Region` e o nodo `Department` e apresentar o nome da região e o número de departamentos que cada uma tem. Depois, é apenas necessário apresentar a região com mais departamentos e o seu resultado pode ser observado em seguida:

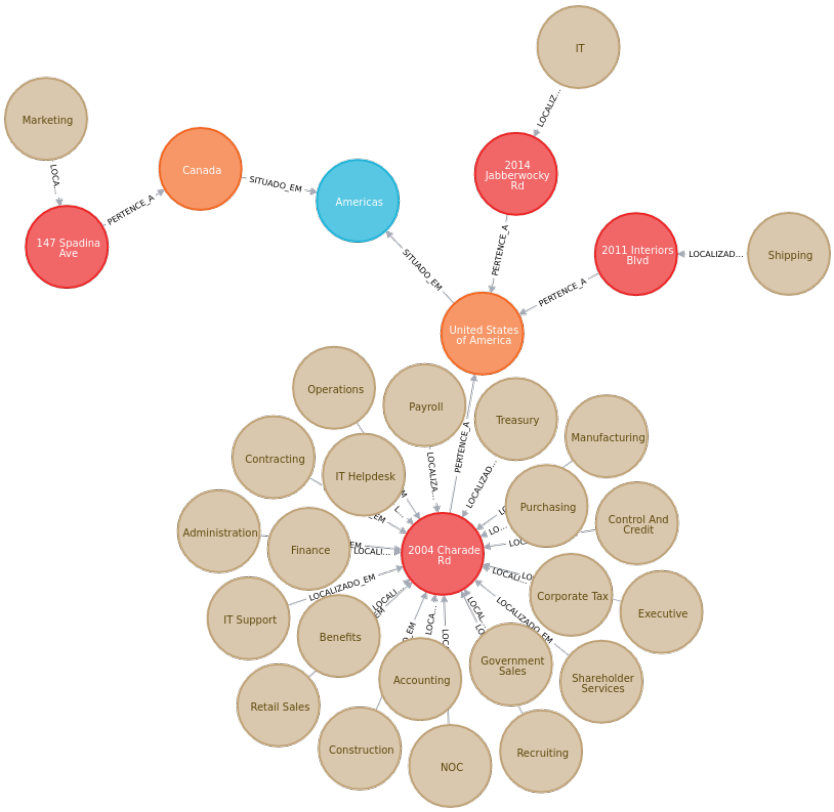

Figura 3.6: Resultado da sexta consulta

4 Análise de Resultados

4.1 Comparação Geral entre os Sistemas

A implementação de um SGBD relacional obriga à criação de um modelo conceptual e lógico, fazendo com que os dados sejam inseridos de forma estruturada. Para tal, estes sistemas recorrem a tabelas e a relações entre as mesmas, garantindo os princípios da filosofia ACID.

O facto de este tipo de bases de dados garantir estas características faz com que se tornem rapidamente ineficientes para um grande número de tabelas. Esta ineficiência deriva da necessidade de percorrer várias tabelas e muitas linhas para a realização de uma só operação.

Mais concretamente, na migração que foi realizada para a base de dados documental MongoDB, foi notória a melhoria do desempenho de algumas consultas e uma grande redução da sua complexidade. O único "problema", se assim se pode chamar, é que, nesta migração, não foram mantidos dados de tabelas que não se relacionavam com nenhum outro elemento, ou seja, dados que consideramos irrelevantes. Neste exemplo em concreto, foram removidos dados de departamentos que não se encontravam relacionados com funcionários, querendo isto dizer que apenas são mantidos dados que se relacionem de forma direta ou indireta com a tabela `EMPLOYEES`.

Quanto à migração para a base de dados orientada a grafos Neo4j, como esta suporta e respeita as transações ACID, a sua "estrutura" ficou muito semelhante à existente no modelo relacional, com algumas alterações quanto à existência de alguns dados associados a algumas tabelas. É percetível que, neste SGBD, a tabela `JOB_HISTORY` do modelo relacional deixou de existir, passando a ser representada por uma relação.

4.2 Testes de Desempenho

Consideramos importante, para qualquer sistema que desenvolvemos, estudar os seus limites, sejam eles teóricos ou práticos, e este projeto não será exceção. Procedemos à realização de testes de desempenho para cada uma das consultas desenvolvidas nos diferentes sistemas adotados com vista a tentar perceber se a nossa solução de migração tem efeitos positivos no tempo de execução destas consultas.

O padrão de testes é simples: para cada sistema de base de dados executamos cada consulta N vezes e, para cada vez, medimos o tempo de resposta e apresentamos também o tempo médio de execução de cada consulta naquele sistema.

Comecemos então pelo sistema relacional, Oracle, onde temos os seguintes tempos medidos para 50 execuções de cada consulta:

N.º da consulta	1	2	3	4	5	6
Tempo médio (ms)	0.336	0.431	1.849	1.396	1.879	2.168

Tabela 4.1: Tempos de execução médios para Oracle

Se analisarmos os tempos médios de execução obtidos, podemos verificar que as consultas mais rápidas são a 1 e 2 que, apesar de envolverem junções de 2 ou 3 tabelas, apresentam tempos de execução baixos, devido ao facto de a quantidade de dados não ser suficiente para produzir testes mais pesados. A consulta 6 é a mais demorada, envolvendo também ordenação de dados, mas limitando o iterador do resultado para apenas uma entrada e, assim, otimiza-se o tempo de execução pedindo apenas o *top* 1.

Apresentamos também um gráfico através de um *script* em Python desenvolvido especificamente para estes testes, que mostra, para cada uma das 50 execuções, a evolução do tempo de resposta, em milissegundos:

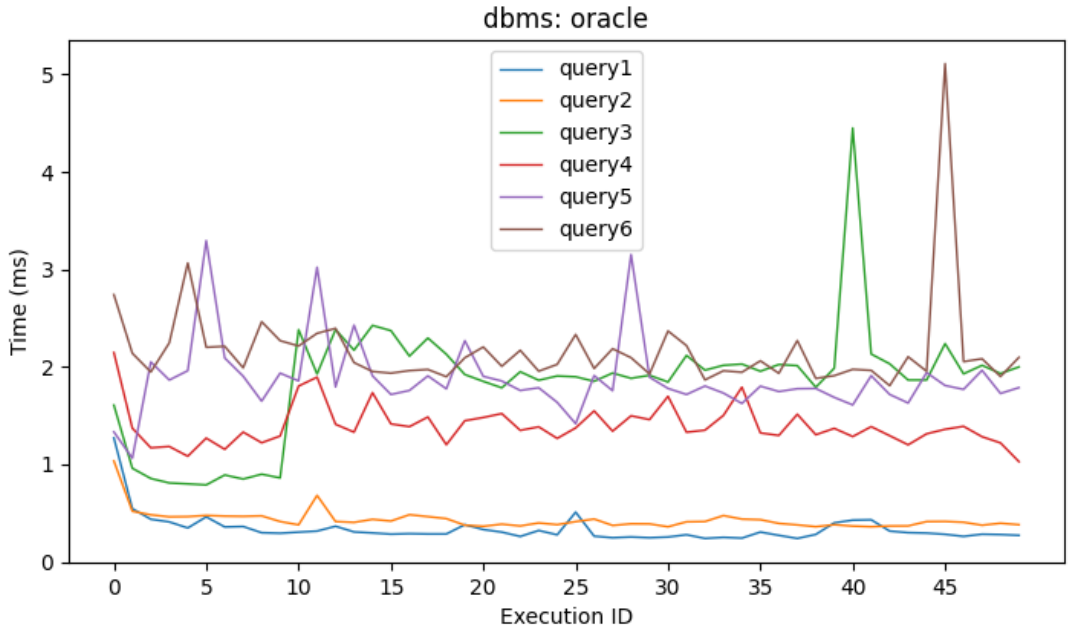

Figura 4.1: Evolução do tempo de execução para a base de dados Oracle

De seguida, para estabelecer como fator comparativo, procedemos ao desenvolvimento dos mesmos testes para o sistema não relacional MongoDB, onde verificámos a seguinte média de tempos:

N.º da consulta	1	2	3	4	5	6
Tempo médio (ms)	0.060	0.037	0.037	1.156	0.041	0.837

Tabela 4.2: Tempos de execução médios para MongoDB

Se excluirmos a consulta 4 e a 6, podemos ver uma melhoria evidente no tempo de execução das demais, muito pela estratégia adotada na compactação dos dados num

objeto só. O problema está quando analisamos aquelas que envolvem agregação de dados em MongoDB, ou seja, aquelas que envolvem `group`, sendo elas as consultas 4 e 6, algo que não é, de todo, o ponto forte de um sistema não relacional documental e, por isso, os tempos são mais elevados, mas, ainda assim, melhores do que na Oracle.

Os tempos médios apresentados anteriormente são referentes ao gráfico que vamos apresentar de seguida, onde a variação dos tempos acaba por ser mais constante:

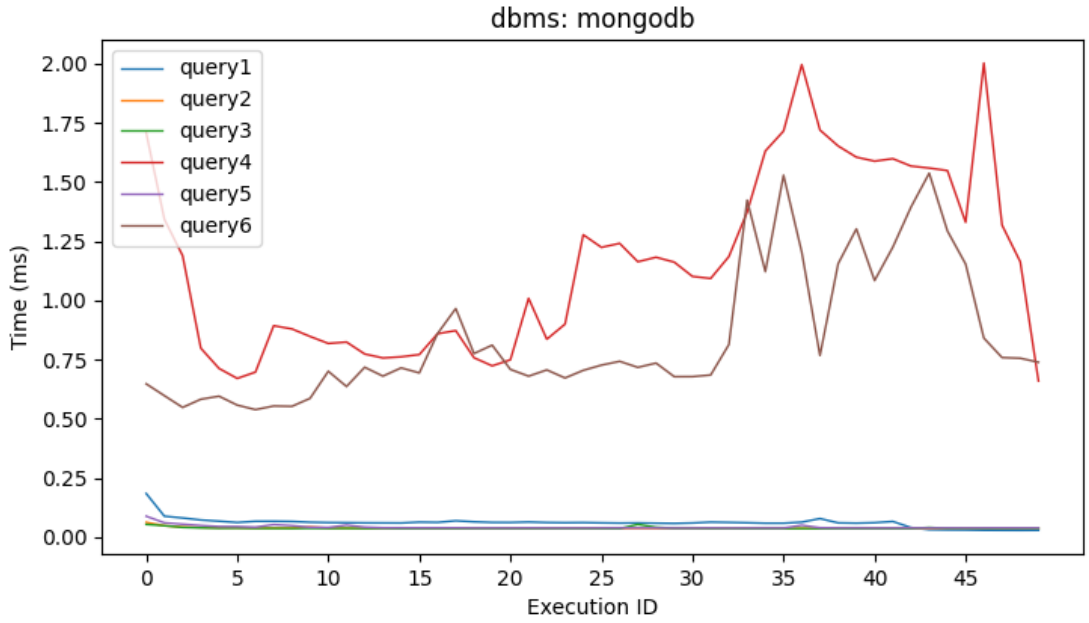

Figura 4.2: Evolução do tempo de execução para a base de dados MongoDB

Por fim, realizámos o mesmo processo para o sistema baseado em grafos, no entanto, este foi aquele que não apresentou os resultados, na sua maioria, que esperávamos:

N.º da consulta	1	2	3	4	5	6
Tempo médio (ms)	0.704	0.646	1.097	1.724	1.856	0.813

Tabela 4.3: Tempos de execução médios para Neo4j

Aqui, os tempos dispersam um pouco e temos, por exemplo, as consultas 1, 2 e 4 com um peso um pouco maior neste sistema, pois envolve fazer filtragem de dados não compactados e agregações dos mesmos. No entanto, a explicação para estes valores pode ser outra:

1. Instabilidade: Na realização de testes, muitos fatores podem explicar a variação dos tempos de execução, como a gestão de conexões ao sistema Neo4j, tipo de comunicação e protocolos utilizados, entre outros;

2. Granularidade dos dados: Por mais eficiente que seja o sistema ou as consultas desenvolvidas, é necessário perceber que os dados que temos para trabalhar são

muito poucos para podermos justificar, com clareza, que o sistema X é melhor que o sistema Y, na medida em que um sistema baseado em grafos apenas admite diferenças significativas se a quantidade dos dados for muito maior, pelo que se trata de uma solução mais escalável.

O gráfico seguinte[1] mostra essa variação de forma nítida e, inclusive, retrata a instabilidade que achamos que possa estar a acontecer:

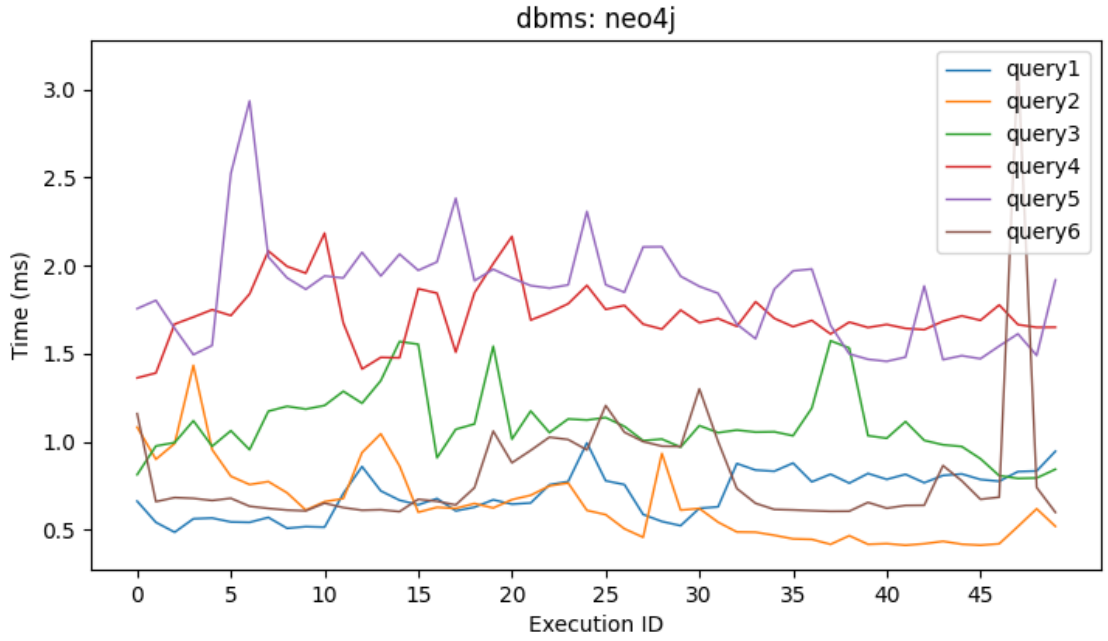

Figura 4.3: Evolução do tempo de execução para a base de dados Neo4j

Por outro lado, a consulta 6 apresenta uma melhoria que pensamos ser significativa (o seu tempo de execução é quase 3 vezes menor do que em Oracle), pois envolve a junção de muitas tabelas pelo que, num sistema orientado a grafos, esta operação é extremamente eficiente, bastando verificar os nodos adjacentes que já se encontram relacionados com outros, enquanto no modelo relacional é preciso juntar as tabelas primeiro e só depois tratar os dados.

Concluímos, então, que os sistemas não relacionais diferem em dois pontos muito específicos, sendo o MongoDB melhor para filtragem de grandes quantidades de dados pois o seu "esquema" é, em geral, um objeto compactado com toda a informação. No entanto, quando queremos percorrer mais tabelas (`join`) e descobrir, por exemplo, quais são todos os nodos que estão relacionados com outros e, de seguida, percorrer as associações desses com outros quaisquer, sob a forma de árvore, o Neo4j apresenta uma solução mais eficiente, ou seja, menos redundância e consequentes pesquisas rápidas. Contudo, apenas com base numa grande quantidade de dados, o chamado *Big Data*, conseguiríamos explorar as vantagens da utilização de bases de dados não relacionais, sendo que, neste trabalho, isto é algo que não conseguimos verificar.

[1] De notar que a escala é diferente dos demais gráficos.

5 Organização do Projeto

Este projeto está dividido em duas grandes partes, a implementação do SGBD relacional e *scripts* de geração de dois não relacionais (MongoDB e Neo4j). É importante referir que ainda existe uma parte relativa às consultas implementadas para cada tipo de base de dados. A estrutura é a seguinte:

```
...
\migrate
\oracle\hr
\queries
```

5.1 Diretoria `migrate`

Nesta diretoria, podem ser observados os *scripts* efetuados para a realização das duas migrações (MongoDB e Neo4j) e, ainda, um ficheiro de configurações (`configuration.toml`).

5.1.1 Ficheiro de Configuração

Este ficheiro contém todas as informações necessárias à realização das conexões a cada uma das base de dados. Para tal, utilizámos o formato TOML, um formato de arquivo de configuração que suporta a utilização de dados chave-valor muito útil para a nossa implementação em específico. Em seguida, podemos observar os dados necessários para a conexão à base de dados Neo4j:

```
[neo4j]
uri      = bolt://localhost:7687
user     = neo4j
passwd   = test
```

5.1.2 Subdiretoria `mongodb`

Aqui, é possível encontrar um ficheiro Python que faz toda a migração do SGBD relacional para a base de dados orientada a documentos MongoDB. Para tal, é apenas necessário executar o seguinte comando:

```
python3 migrate-mongodb.py
```

5.1.3 Subdiretoria `neo4j`

À semelhança do que foi referido anteriormente, mas agora relativamente à base de dados orientada a grafos (Neo4j), é possível encontrar tudo o que é necessário para esta migração. Uma vez que se torna um pouco mais complexa em termos de estrutura, esta migração é efetuada em dois passos:

1. Geração de todos os ficheiros CSV para a diretoria `gen_csv`. Para isso, é necessário executar o ficheiro `init-csv.py` que recorre à leitura do ficheiro `init.sql`;

2. Execução da migração propriamente dita, fazendo a conexão à base de dados Neo4j e importando todos os ficheiros gerados anteriormente através da execução do ficheiro `migrate.py`. Este ficheiro recorre à leitura de outro ficheiro já referenciado (`import.cypher`) que contém todos os comandos necessários para a criação de todos os nodos e relações entre eles.

Deste modo, podem-se correr os *scripts* referentes aos pontos anteriores através do seguinte comando:

```
python3 init-csv.py && python3 migrate-py
```

5.2 Diretoria `oracle`

Aqui, podem ser encontrados os ficheiros disponibilizados pelos docentes para a implementação do SGBD relacional Oracle. O ficheiro `1-database.sql` contém a informação relativa à criação do *schema* e do utilizador para esta base de dados. Já o ficheiro `2-script.sql` faz a criação de todas as tabelas e o povoamento das mesmas desta aplicação de Recursos Humanos.

5.3 Diretoria `performance`

Nesta diretoria, podemos encontrar tudo o que foi necessário para a realização dos testes de desempenho. O ficheiro `performance.py` permite-nos gerar os testes de forma automática recorrendo aos ficheiros com as consultas presentes na diretoria `queries`. Este ficheiro gera os resultados relativos a cada uma das bases de dados e armazena os mesmos na subdiretoria `results`, procedendo, de seguida, à apresentação de um gráfico onde consta a evolução do tempo de resposta.

À semelhança dos ficheiros anteriores, para a execução deste será apenas necessário executar o comando:

```
# Executes 30 times each query for each dmbs
python3 performance.py 30
```

6 Conclusões e Trabalho Futuro

A realização deste trabalho permitiu a assimilação de conceitos aprendidos nas aulas teóricas no que toca às diferenças de paradigmas entre bases de dados relacionais e não relacionais.

A escolha das bases de dados NoSQL, como Neo4j e MongoDB, permitiram a utilização de modelos não relacionais diferentes, sendo que conseguimos perceber na prática as vantagens de usarmos uma base de dados documental ou de grafos.

Isto também permitiu analisarmos as diferenças da eficiência de cada modelo distinto para diversas operações. Apesar da documentação do desempenho e justificação do procedimento, teria sido interessante explorar melhor a eficiência de cada modelo com uma maior quantidade de dados, pois o paradigma NoSQL é mais adequado para essas situações. Desta forma, gostaríamos de comparar o seu desempenho numa situação mais plausível para o uso deste paradigma.

Assim, pensamos que os objetivos propostos pela equipa docente foram cumpridos, pelo que levamos deste trabalho bastante conhecimento relativo às grandes diferenças dos dois paradigmas, bem como à migração de dados para duas bases de dados com modelos diferentes.

Referências

[1] Harisson G., Next Generation Databases: NoSQL and Big Data, Apress, 2015. Mullins, C., Database Administration: The Complete Guide to Practices and Procedures, Addison Wesley Pub, 2002.

[2] MongoDB. 2021. NoSQL vs SQL Databases. [online] Available at: https://www.mongodb.com/nosql-explained/nosql-vs-sql.

[3] Neo4j Graph Database Platform. 2021. Import Relational Data Into Neo4j - Developer Guides. [online] Available at: https://neo4j.com/developer/guide-importing-data-and-etl/.

[4] Docs.microsoft.com. 2021. Relational vs. NoSQL data. [online] Available at: https://docs.microsoft.com/en-us/dotnet/architecture/cloud-native/relational-vs-nosql-data.

Lista de Siglas e Acrónimos

SGBD	Sistema de Gestão de Bases de Dados
SQL	*Structured Query Language*
ACID	Atomicidade, Consistência, Isolamento, Durabilidade
CSV	*Comma-separated values*
JSON	*JavaScript Object Notation*
TOML	*Tom's Obvious, Minimal Language*
HR	*Human Resources*