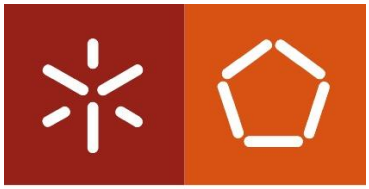

Universidade do Minho

Departamento de Informática

Bases de Dados NoSQL

Trabalho Prático

Ana Gil (A85266), Angélica Freitas (A83761), Paulo Lima (A89983), Rodrigo Pimentel (A83765)

29 de janeiro de 2021

Conteúdo

Capítulo 1

Introdução

Com a realização deste projeto, pretende-se a complementação de matérias da unidade curricular de Bases de Dados NoSQL na utilização de diferentes paradigmas de bases de dados e a sua aplicação na concepção e implementação de sistemas. Neste projeto iremos analisar profundamente as principais diferenças entre os paradigmas Relacional e NoSQL e modelos documentais, relacionais e orientados a grafo.

O ponto de partida será uma base de dados em modelo relacional e, a partir dessa, iremos realizar migrações e queries para os outros respetivos modelos e identificar diferenças, vantagens e desvantagens. No final deste trabalho pretendemos demonstrar como deve ser feita uma migração total (queries e dados) de um paradigma relacional para um paradigma documental e gráfico.

Capítulo 2

Desenvolvimento

2.1 Passos Iniciais

Após a execução dos scripts fornecidos (*1-database.sql* e *2-script.sql*) e o devido estudo do esquema de recursos humanos, considerámos quais as bases de dados que iriamos usar para cada modelo (relacional, documental e orientada a grafo). Para o modelo relacional decidimos utilizar a Oracle, pois tem uma boa performance em comparação com outras bases de dados quando os dados se tornam grandes, não costuma ter problemas com múltiplos joins e é dos software mais populares de momento. Para o modelo documental, recorremos ao MongoDB, porque permite construir uma topologia *clustered* de alta disponibilidade com replicação e *sharding*. O nosso sistema não apresenta tal topologia, porém é importante pois permite que essa evolução seja efetuada, o que é imprescindível para aplicações atuais e também por ser open source. Por fim, para o modelo orientado a grafo, escolhemos o *neo4j*, devido ao tamanho da sua comunidade, o que nos permitirá aceder a informação mais rapidamente, por apresentar aprendizagem fácil e rápida e por ter leituras e escritas rápidas.

Assim, escolhidas as bases de dados a utilizar, desenvolvemos um conjunto de queries para serem realizadas ao longo de todos os modelos:

1. Inserir um *Employee*
2. Procurar um *Employee* pelo seu *id*
3. Listar *Employees* por *Managers*
4. Listar os *Employees* que começaram a exercer num determinado ano
5. Atualizar o salário de um *Employee*
6. Aumentar o salário dos *Employees* que tenham salário mínimo num determinado *job* em 5%.
7. Listar a média do salário do departamento que tem mais *Employees* e a comissão é nula
8. Top 3 países com melhor média de salário
9. Terminar um *job* atual de um determinado *Employee*
10. Atribuir um novo *job* a um determinado *Employee*
11. Listar a média de salário por *job title*
12. *Employees* que não mudaram de carreira
13. *Employees* que já não trabalham na empresa

Com estas *queries*, pretendiamos não só demonstrar a operacionalidade dos sistemas implementados, mas também mostrar uma boa variedade de interrogações sobre os mesmos. Assim, quaisquer outras *queries* que não apresentadas, por exemplo top 3 países com pior média de salário, serão, à partida, permutações simples às apresentadas.

2.2 Modelo Relacional: Oracle

2.2.1 Queries

1. Inserir um Employee

```
INSERT INTO employees (employee_id, last_name, email, hire_date,
    job_id, manager_id, department_id)
VALUES
(&employee_id, '&last_name', '&email', '&hire_date', '&job_id',
    &manager_id, &department_id);
```

2. Procurar Employee pelo *ID*

```
SELECT *
FROM employees e
WHERE e.employee_id=&id;
```

3. Listar Employees geridos por *Manager*

```
SELECT CONCAT(CONCAT(CONCAT(m.first_name, m.last_name), ',
    #'),m.employee_id) "Manager",
         e.employee_id, e.first_name, e.last_name,
         e.email, e.phone_number, e.hire_date, e.job_id,
         e.salary, e.commission_pct, e.department_id
FROM employees e
INNER JOIN employees m
ON e.manager_id=m.employee_id
ORDER BY e.manager_id;
```

4. Lista de *Employees* que começaram a trabalhar num determinado ano

```
DEFINE s_year = &start_year;
SELECT e.*
FROM employees e
WHERE extract(year FROM e.hire_date)=&s_year;
```

5. Atualizar o salário de um *Employee*

```
CREATE OR REPLACE TRIGGER check_update_salary
BEFORE UPDATE OF salary on employees
FOR EACH ROW
DECLARE
    min_s NUMBER;
    max_s NUMBER;
BEGIN
    SELECT max_salary into max_s FROM jobs where
       job_id=:old.job_id;
    SELECT min_salary into min_s FROM jobs where
       job_id=:old.job_id;

    IF (:new.salary > max_s or :new.salary < min_s)
    THEN
        RAISE_APPLICATION_ERROR (
            num => -01438,
            msg => 'Salary not allowed'
        );
    END IF;
```

```
END;
/

UPDATE employees e
SET e.salary=&new_salary
WHERE e.employee_id=&employee_id;
```

6. Aumentar 5% o salário de um Employee caso este tenha o salário mais baixo

```
UPDATE employees e
SET e.salary = 1.05*e.salary
WHERE e.salary = (SELECT min_salary FROM jobs WHERE
   job_id=&job_id);
```

7. Média dos salários do departamento com mais employees e sem commission

```
SELECT d.department_name, "TOTAL".n_employees, "TOTAL".avg_salary
   from departments d
INNER JOIN (
    SELECT d.department_id, count(1) as n_employees, avg(e.salary)
       as avg_salary
    FROM employees e
    INNER JOIN departments d
    ON e.department_id=d.department_id
    WHERE e.commission_pct is null
    GROUP BY d.department_id
) "TOTAL"
ON "TOTAL".department_id=d.department_id
ORDER BY "TOTAL".n_employees desc
OFFSET 0 ROWS FETCH NEXT 1 ROWS ONLY;
```

8. Top 3 de Países com maior média de salário

```
SELECT co.country_id, ROUND(sum(e.salary)/count(*),2) as ratio,
   min(e.salary) "min salary", max(e.salary) "max salary"
FROM countries co, locations l, departments d, employees e
WHERE co.country_id=l.country_id
AND l.location_id=d.location_id
AND d.department_id=e.department_id
GROUP BY co.country_id;
```

9. Terminar o *job* atual de um *employee*

```
CREATE OR REPLACE PROCEDURE end_job(id_employee NUMBER)
IS
    e_start_date DATE;
    e_job_id VARCHAR2(10);
    e_department_id NUMBER ;
    aux number;
    aux1 number;
BEGIN
    select count(*) into aux from employees e, job_history jh
            where e.employee_id = jh.employee_id
            and not(jh.end_date is null)
            and jh.employee_id=id_employee;

    select count(*) into aux1 from employees e, job_history jh
```

```
            where e.employee_id = jh.employee_id
            and e.hire_date>jh.end_date
            and jh.employee_id=id_employee;
    IF (aux=0 OR aux1>0)
    THEN
        SELECT hire_date INTO e_start_date from employees where
            employee_id=id_employee;
        SELECT job_id INTO e_job_id from employees where
            employee_id=id_employee;
        SELECT department_id INTO e_department_id from employees
            where employee_id=id_employee;

        INSERT INTO job_history (employee_id, start_date,
            end_date, job_id, department_id)
        VALUES (id_employee,e_start_date,(SELECT CURRENT_DATE FROM
            DUAL),e_job_id,e_department_id);
    END IF;
END;
/
```

10. Atribuir um novo job a um determinado employee

```
CREATE OR REPLACE PROCEDURE add_job(id_employee NUMBER, new_job_id
    VARCHAR2)
IS
    adate DATE;
BEGIN
    SELECT CURRENT_DATE INTO adate FROM DUAL;

    UPDATE employees
    SET HIRE_DATE = adate, JOB_ID = new_job_id
    WHERE employee_id=id_employee;
END;
/
```

11. Média de salários por *job*

```
select js.job_title "job title", "avg_salary"."avg_s" "average
    salary"
from jobs js
inner join (
SELECT j.job_id, avg(e.salary) "avg_s"
FROM employees e, jobs j
WHERE e.job_id=j.job_id
group by j.job_id) "avg_salary"
on "avg_salary".job_id=js.job_id;
```

12. *Employees* que não mudaram de carreira

```
select e.* from job_history j
right join employees e
on j.employee_id=e.employee_id
where j.employee_id is null;
```

13. *Employees* que já não trabalham para a empresa

```
select e.*
from employees e, job_history jh
where e.employee_id=jh.employee_id
and e.hire_date=jh.start_date;
```

2.2.2 Características principais

O modelo relacional tem como características principais:

- Dados representados por tabelas e colunas.
- Obedecer aos princípios ACID.
- Desenhado para escabilidade vertical.
- Linguagem de consulta universal SQL.
- Modelos de negócio fechados. Normalmente não são "open source".
- Ótimo modelo para quem pretente abdicar um pouco da "performance"por consistência e estabilidade de dados.

2.3 Modelo Documental: MongoDB

2.3.1 Migração do Modelo Relacional para Documental

Para migrar os dados na base de dados *Oracle* para um modelo não relacional orientado a documentos foi necessário ter em conta as caraterísticas deste tipo de base de dados, neste caso o *MongoDB*.

As bases de dados relacionais armazenam dados em tabelas, cada tabela contém colunas e cada linha representa um registo. As informações sobre uma entidade podem ser espalhadas por várias tabelas, sendo que os dados de tabelas diferentes podem estar associados quando estabelecemos um relacionamento entre as tabelas. Contrariamente, as bases de dados documentais não usam tabelas. Os dados de uma determinada entidade são armazenados num único documento e todos os dados associados são armazenados dentro desse documento. Tendo em conta esta informação, alteramos as tabelas de forma a que os dados fossem organizados num documento com informações que se relacionam com um *Employee*.

A realização da migração dos dados implicou a separação do processo em duas fases essenciais: a recolha de dados provenientes da Base de Dados *Oracle*, e o processamento desses mesmos dados.

Todo o processo de obtenção de dados e processamento foi feito em *Python*. Para a primeira fase, foi utilizada a biblioteca *cx_Oracle* que permite fazer a ligação à *PDB* e a partir daí executar os comandos SQL necessários para obter a informação necessária.

Todas as *queries* foram feitas de forma a que o resultado obtido fosse retornado em *JSON*, esta pratica facilitou o processo de migração visto que foi que alterar o formato de *JSON* para *dict* em Python e bastante fácil. Desta forma, o processamento de dados para o formato desejado foi acessível, sendo que o documento JSON criado para fazer a importação no MongoDB partiu da gestão dos *dict* criados.

```
SELECT JSON_OBJECT ('COUNTRY_ID' VALUE COUNTRY_ID,
                    'COUNTRY_NAME' VALUE COUNTRY_NAME,
                    'REGION_ID' VALUE REGION_ID)
                    FROM COUNTRIES
```

Listing 2.1: Recolha de dados em formato JSON

2.3.2 Queries

1. Inserir um Employee

```
var employee = {
    "EMPLOYEE_ID": 12345,
    "FIRST_NAME": "Steven",
    "LAST_NAME": "King",
    "EMAIL": "SKING",
    "PHONE_NUMBER": "515.123.4567",
    "HIRE_DATE": "2003-06-17T00:00:00",
    "SALARY": 24000,
    "COMMISSION_PCT": "null",
    "DEPARTMENT": {
        "DEPARTMENT_ID": 90,
        "DEPARTMENT_NAME": "Executive",
        "MANAGER_ID": 100,
        "LOCATION_ID": 1700,
        "LOCATION": {
            "LOCATION_ID": 1700,
            "STREET_ADDRESS": "2004 Charade Rd",
            "POSTAL_CODE": "98199",
            "CITY": "Seattle",
            "STATE_PROVINCE": "Washington",
            "COUNTRY_ID": "US",
            "COUNTRY": {
                "COUNTRY_ID": "US",
                "COUNTRY_NAME": "United States of America",
                "REGION_ID": 2,
                "REGION": {
                    "REGION_ID": 2,
                    "REGION_NAME": "Americas"
                }
            }
        }
    },
    "JOB": {
        "JOB_ID": "AD_PRES",
        "JOB_TITLE": "President",
        "MIN_SALARY": 20080,
        "MAX_SALARY": 40000
    },
    "JOB_HISTORY": [],
    "MANAGER": "null"
}
db.hrs.insert(employee);
```

2. Procurar Employee pelo seu *ID*

```
db.hrs.find({EMPLOYEE_ID: 100}).pretty();
```

3. Listar Employees por *Managers*

```
db.hrs.aggregate([
    {$project : {"MANAGER.EMPLOYEE_ID": 1 ,EMPLOYEE_ID: 1,
       FIRST_NAME:1, SALARY:1, _id:0}},
    {$sort: {"MANAGER.EMPLOYEE_ID": 1}}
]);
```

4. Listar os *Employees* que comecaram a exercer num determinado ano

```
db.hrs.find({HIRE_DATE: {$regex: /2006-.*/}}).pretty();
```

5. Atualizar o salário de um *Employee*

```
db.hrs.updateOne({EMPLOYEE_ID: 100},{$set: {SALARY: 23999}});
```

6. Aumentar o salário dos *Employees* que tenham salário mínimo num determinado *job* em 5%

```
db.hrs.update({ $expr: {$eq: ["$SALARY","$JOB.MIN_SALARY"]
   },"JOB.JOB_ID": "SH_CLERK" }, { $mul: { SALARY: 1.05 } },
   {multi: true})
```

7. Listar a média do salário do departamento que tem mais *Employees* e a comissão é nula

```
db.hrs.aggregate([
    {$group:
        {
            _id: "$DEPARTMENT.DEPARTMENT_NAME",
            salary: {$sum: "$SALARY"},
            employees: {$sum: 1}
        }
    },
    {
        $project: {_id:1,employees:1,avg: {$divide:
           ["$salary","$employees"]}}
    },
    {
        $sort: {employees: -1}
    },
    {
        $limit: 1
    }
]);
```

8. Top 3 de Países com maior média de salário

```
db.hrs.aggregate([
    {$group:
        {
            _id: "$DEPARTMENT.LOCATION.COUNTRY.COUNTRY_ID",
            salary: {$sum: "$SALARY"},
            num: {$sum: 1}
        }
    },
    {
        $project: {ratio: {$divide: ["$salary","$num"]}}
    },
    {
        $sort: {ratio: -1}
    },
    {
        $limit: 3
    }
]);
```

9. Terminar o *job* atual de um determinado *Employee*

```
var h_date = db.hrs.findOne({EMPLOYEE_ID:
    103},{HIRE_DATE:1,_id:0});
var j_id = db.hrs.findOne({EMPLOYEE_ID:
    103},{"JOB.JOB_ID":1,_id:0});
var d_id = db.hrs.findOne({EMPLOYEE_ID:
    103},{"DEPARTMENT.DEPARTMENT_ID":1,_id:0});

db.hrs.update(
    {EMPLOYEE_ID:103},
    {
        "$push": {JOB_HISTORY:{
                EMPLOYEE_ID: 103,
                START_DATE: hdate,
                END_DATE: "2021-02-12",
                JOB_ID: j_id,
                DEPARTMENT_ID: d_id}
        }
    }
)
```

10. Atribuir um novo *job* a um determinado *Employee*

```
var h_datee = db.hrs.findOne({EMPLOYEE_ID:
    103},{HIRE_DATE:1,_id:0});
var j_idd = db.hrs.findOne({EMPLOYEE_ID:
    103},{"JOB.JOB_ID":1,_id:0});
var d_idd = db.hrs.findOne({EMPLOYEE_ID:
    103},{"DEPARTMENT.DEPARTMENT_ID":1,_id:0});

db.hrs.update(
    {EMPLOYEE_ID:103},
    {
        "$push": {JOB_HISTORY:{
                EMPLOYEE_ID: 103,
                START_DATE: hdatee,
                END_DATE: "2021-02-12",
                JOB_ID: j_idd,
                DEPARTMENT_ID: d_idd}
        }
    }
)

db.hrs.update({EMPLOYEE_ID:103},{$set:{HIRE_DATE:"2021-02-12",
    "JOB.JOB_ID":"AD_PRES"}, "DEPARTMENT.DEPARTMENT_ID":10})
```

11. Listar a média de salário por *job title*

```
db.hrs.aggregate([
    {$group: {_id: "$JOB.JOB_TITLE", avg: {$avg: "$SALARY"}}}
]);
```

12. *Employees* que não mudaram de carreira

```
db.hrs.find({JOB_HISTORY:{ $exists: true, $eq:
    []}},{_id:0}).pretty();
```

13. *Employees* que já não trabalham para a empresa

```
db.hrs.aggregate([
     {$match: {JOB_HISTORY:{$ne: []}}},
    {
       $project: {
        _id:0,
        EMPLOYEE_ID:1,
          res: {
             $filter: {
                input: "$JOB_HISTORY",
                as: "jh",
                cond: {$eq: [ "$$jh.START_DATE", "$HIRE_DATE" ] }
             }
          }
       }
    },
    {$match: {res: {$ne: []}}}
 ]);
```

2.3.3 Características principais

O modelo documental tem como características principais:

- Dados representados em colecções de documentos (JSON,XML,...).
- Não obedece aos princípios ACID.
- Desenhado para escabilidade horizontal.
- Possui diversas linguagens de consulta, por vezes até SQL.
- Alta disponibilidade de dados e *Big Data*.
- *Open source*.

2.3.4 Vantagens/Desvantagens em relação ao modelo relacional

Após a realização deste projeto, as principais diferenças entre o modelo relacional e documental foram notáveis. Como o último não segue os princípios ACID (referido na secção 2.3.3), isto é, por exemplo, não há restrições de integridade, que são associados à Consistência (como por exemplo *foreign keys*). Por um lado, isto permite que toda a informação referente à entidade se encontre imediatamente disponível, ao contrário de modelos relacionais que é necessário fazer *lookups* adicionais. Por outro lado, devido à mesma razão, é possível a inserção de dados incoerentes, por exemplo, seria permitido a inserção de um *employee* com um *job title* inexistente. Assim, algumas das *queries* realizadas na secção 2.3.2 são sujeitas a erro humano, que é o caso da *query* 9. Também se observou um bom desempenho em *MongoDB* em relação à *Oracle* e, devido ao esquema livre que apresenta, o processo de migração de dados foi facilitado significamente. Como mencionado anteriormente, as bases de dados documentais foram desenhadas para escabilidade horizontal, que ao contrário de bases de dados relacionais, é o mais desejado para aplicações atuais.

2.4 Modelo Gráfico: Neo4J

2.4.1 Migração do Modelo Relacional para Gráfico

Para migrar os dados da *Oracle Database* previamente disponibilizada de um modelo relacional para um modelo não relacional orientado a grafos, foi necessário ter em conta as caraterísticas deste tipo de base de dados, neste caso o Neo4J.

Visto que uma base de dados orientada a grafos é constituída tanto por nós como por ligações, que representam o relacionamento entre tabelas, torna-se redundante cada nó conter o valor das suas chaves estrangeiras. Assim, no processo de migração, estas foram usadas para estabelecer e criar as ligações entre os nós porém não constam como atributo do nó em específico.

Nesta fase, começamos por, através das ferramentas disponibilizadas pelo *Oracle SQL Developer* (File - Data Modeler - Export - To CSV), exportar cada uma das tabelas para ficheiros *CSV*, sendo este um processo simples e instantâneo.

Uma vez realizada a exportação dos dados, passamos para o processo de importação dos mesmos no Neo4J. Para efeitos de demonstração seguem-se de seguida partes de código presentes no *script* da migração dos dados, nomeadamente referentes à tabela *Employee* e à ligação representante da conectividade entre um *Employee* e um *Job*.

Tanto ao nível de importação dos dados como da implementação das queries, apresentadas na secção seguinte, foi utilizada a linguagem *Cypher*.

```
LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS csvLine
CREATE (e:Employees {
employee_id: toInteger(csvLine.EMPLOYEE_ID),
first_name: csvLine.FIRST_NAME,
last_name: csvLine.LAST_NAME,
email: csvLine.EMAIL,
phone_number: csvLine.PHONE_NUMBER,
hire_date: csvLine.HIRE_DATE,
salary: toFloat(csvLine.SALARY),
commission_pct: toFloat(csvLine.COMMISSION_PCT)});
```

Listing 2.2: Importação dos nós Employee

```
LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS csvLine
MATCH (e:Employees {
employee_id: toInteger(csvLine.EMPLOYEE_ID)}),
(j:Job {job_id: csvLine.JOB_ID})
CREATE (e)-[:Works_In]->(j);
```

Listing 2.3: Ligação Employee-Job

De modo semelhante foram criados os restantes nós e ligações.

2.4.2 Queries

1. Inserir um *Employee*

```
MATCH (j:Job),(d:Department)
WHERE j.job_id='AD_ASST' and d.department_id=80
CREATE (d)-[:Has]->(e:Employees {employee_id: 101 ,
    first_name: 'Neena', email: 'Kochhar', last_name:
    'NKOCHHAR', phone_number: '515.123.4568', hire_date:
    '05-09-21', salary: 17000})-[:Works_AS]->(j)
```

2. Procurar um *Employee* pelo seu *id*

```
MATCH (e: Employees {employee_id: 142})
RETURN e
```

3. Listar *Employees* por *Managers*

```
MATCH (m {employee_id: 100})-[:Leads]->(e)
RETURN e
```

4. Listar os *Employees* que comecaram a exercer num determinado ano

```
MATCH (jh:Job_History)-[:Of]->(e:Employees)
WHERE substring(jh.start_date,0,2)= substring('2006',2,4)
RETURN jh
```

5. Atualizar o salário de um *Employee*

```
MATCH (e: Employees {employee_id: 142})-[:Works_As]->(j:Job)
SET e.salary = CASE e.salary  WHEN 1200<j.min_salary
THEN j.min_salary ELSE (
    CASE e.salary
    WHEN  1200>j.max_salary
    THEN  max_salary
    ELSE 1200
    END
)
END
RETURN *
```

6. Aumentar o salário dos *Employees* que tenham salário mínimo num determinado *job* em 5%.

```
MATCH (e: Employees)-[:Works_In]->(j:Job)
WHERE e.salary=j.min_salary and j.job_id="SH_CLERK"
SET
   e.salary=toFloat(e.salary+(0.05*e.salary)),j.min_salary=e.salary
RETURN e
```

7. Listar a média do salário do departamento que tem mais *Employees* e a comissão é nula

```
MATCH (d:Department )-[:Has]->(e:Employees)
WHERE not exists(e.commission_pct)
RETURN d as Department ,count(e) as nEmployees
ORDER BY count(e) DESC
LIMIT 1
```

8. Top 3 países com maior média de salário

```
MATCH (c:Country)<-[:Is_in]-(l:Location)<-
[:Is_located]-(d:Department)-[:Has]->(e:Employees)
RETURN c as Country ,avg(e.salary)as AverageSalary
ORDER BY avg(e.salary) DESC
LIMIT 3
```

9. Terminar um *job* atual de um determinado *Employee*

```
MATCH (e:Employees)-[r:Works_As]->(j:Job)
WHERE e.employee_id=101
DELETE r
```

10. Atribuir um novo *job* a um determinado *Employee*

```
MATCH (e:Employees)-[r:Works_As]->(j:Job),(jj:Job)
WHERE e.employee_id=101 and jj.job_id="AD_ASST"
MERGE (e)-[:Works_As]->(jj)
DELETE r
```

11. Listar a média de salário por *job title*

```
MATCH (j:Job)<-[:Works_As]-(e:Employees)
RETURN j.job_title as Job ,avg(e.salary) as AverageSalary
ORDER BY avg(e.salary) DESC
```

12. *Employees* que não mudaram de carreira

```
MATCH (e:Employees)
WHERE NOT (:Job_History)-[:Of]->(e)
RETURN e;
```

13. *Employees* que já não trabalham para a empresa

```
MATCH (j:Job_History)-[r:Of]->(e:Employees)
WHERE j.start_date= e.hire_date
RETURN e
```

Devido ao facto de estarmos perante um modelo orientado a grafos e que tem um componente visual, achamos pertinente acrescentar algumas queries de modo a dar aproveito desta componente.

14. Apresentar todos os Departamentos existentes na região "Americas"

```
MATCH (d:Department)<-[:Is_located]-(l:Location)-[i:Is_in]
->(c:Country)-[b:Belongs]->(r:Region)
WHERE r.region_name="Americas"
RETURN *
```

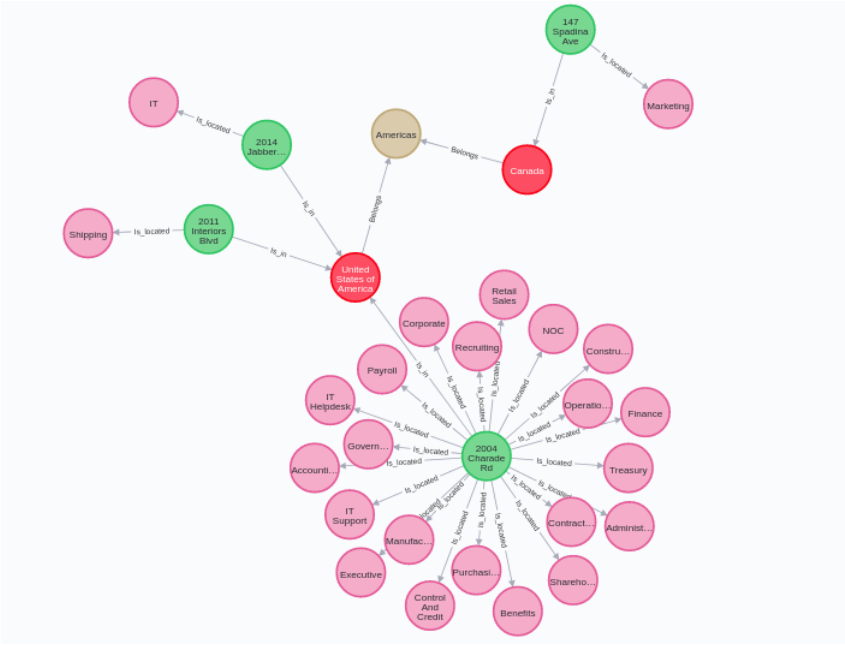

Figura 2.1: Resultado visual da *query* 14

15. Listar *Employees* existentes no Departamento *Shipping*

```
Match (d:Department)-[:Has]->(e:Employees)
WHERE d.department_name="Shipping"
RETURN *
```

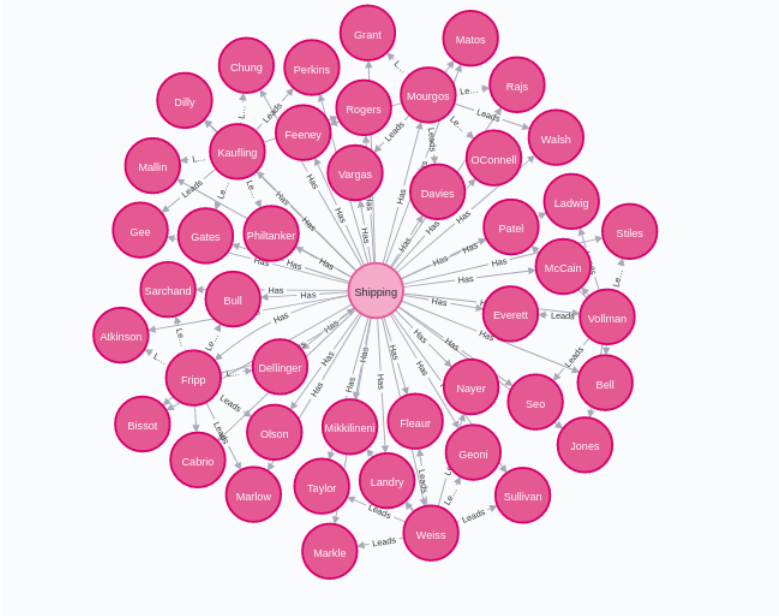

Figura 2.2: Resultado visual da *query* 15

2.4.3 Características principais

O modelo gráfico tem como características principais:

- Dados representados por grafos. Arestas simbolizam relações e nodos entidades.
- Não obedece aos princípios ACID.
- Desenhado para escabilidade horizontal.
- Possui diversas linguagens de consulta, no caso do Neo4J utiliza o *Cypher*.
- Dados orientados às relações.
- Ótimo para inferir conhecimento.

2.4.4 Vantagens/Desvantagens em relação ao modelo relacional

As base de dados orientadas a grafos têm muitos benefícios sobre outros tipos de base de dados, principalmente sobre as base de dados relacionais, sobretudo quando se trata de consultas de grandes dimensões, ou seja, que envolvem um grande número de tabelas, em dados relacionados. Nestes casos, se os mesmos dados forem armazenados em base de dados gráficas, as consultas seriam mais simples e mais rápidas.

Tal como a maior parte das base de dados não relacionais, não houve a necessidade de definir um esquema, permitindo assim obter uma total flexibilidade sobre o crescimento da base de dados, para além de simplificar bastante o processo inicial de migração de dados. Esta propriedade permite acompanhar, por exemplo, o crescimento de uma empresa, bastando acrescentar novos nodos e novos relacionamentos sem perturbar as consultas e as funcionalidades da camada aplicacional.

Na implementação deste trabalho estas caraterísticas não foram evidentemente verificadas uma vez que as dimensões dos dados não o permitiram, não havendo assim diferenças significativas entre o uso de ambas, quer a nível de performance quer a nível de flexibilidade. No entanto, o uso deste tipo de base de dados mostrou-se ser bastante simples e prático.

Capítulo 3

Conclusão

Durante o desenvolvimento do projeto procuramos encontrar as melhores soluções para agilizar os principais entraves nos problemas que surgiam. Procuramos também orientar o projeto, de forma a este incidir em funcionalidades com mais potencial para facilitar a visualização de toda a informação disponibilizada. Esta conceptualização foi essencial para delinear um trajeto orientado aos nossos objetivos.

Inicialmente foi necessário perceber quais as bases de dados mais apropriadas para o nosso projeto e quais formas corretas de implementar a recolha de dados na Base de Dados Oracle a ser utilizada. Este passo foi fundamental para percebermos qual o formato mais indicado para demonstrar os dados no sistema final.

O plano de desenvolvimento, passou pela delineação de fases de desenvolvimento, entre as quais, geração de Bases de Dados, migração de dados, e queries ao sistema gerado, esta organização permitiu que o desenvolvimento do projeto fosse de acordo com o proposto.

Ao longo deste projeto, deparamo-nos com poucas dificuldades, denominadamente na realização de algumas *queries* no MongoDB mais "complexas", que nos levou à utilização de agregações. Porém, após efetuar pesquisa na documentação do *MongoDB*, facilmente ultrapassamos essas dificuldades.

Tendo em conta tudo o que foi dito, estamos satisfeitos com o nosso trabalho e conseguimos compreender melhor o comportamento, características e vantagens e desvantagens dos diferentes paradigmas e modelos mencionados, tal como a migração de dados entre as mesmas.

Capítulo 4

Webgrafia

https://oracle.github.io/python-cx_Oracle/
https://oracle.github.io/odpi/doc/installation.html#linux/
https://docs.mongodb.com/manual/
https://neo4j.com/developer/cypher/querying/ https://docs.oracle.com/en/