Mestrado em Engenharia Informática

Bases de dados NoSQL

Relatório Trabalho Prático

Aluno/a:	*Número:*
Ana Filipa Pereira	PG46978
Ana Luísa Carneiro	PG46983
Ana Rita Peixoto	PG46988
Luís Miguel Pinto	PG47428

Junho 2022

Conteúdo

1 Introdução

Este trabalho prático foi desenvolvido no âmbito da unidade curricular de **Base de dados NoSQL**, e consistiu na conceção e implementação de três sistemas de gestão de bases de dados (SGBD) sendo um relacional e dois não relacionais. Neste projeto, os dados utilizados são referentes a aplicação de Recursos Humanos, além de que a base de dados relacional escolhida foi o **Oracle**, a BD não relacional orientada a documentos foi o **MongoDB** e a BD não relacional orientada a grafos foi o **Neo4j**. Posto isto, foram também desenvolvidas diversas *queries*, com diferentes níveis de complexidade, de maneira a avaliar a sua performance nestes paradigmas. Por fim, realizou-se uma análise crítica do trabalho desenvolvido, efetuando-se uma comparação entre os modelos e as funcionalidades implementadas.

2 Base de Dados Relacional - *Oracle*

Este trabalho prático teve por base um modelo lógico SQL de uma base de dados Oracle, representativo de uma aplicação de Recursos Humanos, que tem como objetivo armazenar os dados dos empregados de uma organização. Este modelo explicita a lógica de negócio do programa, os atributos de cada entidade e como se relacionam entre si. Através da sua análise foi possível observar como os dados estão organizados e como seria a melhor política de transição dos mesmos para modelos não relacionais.

3 Base de Dados Documental - *MongoDB*

A partir do modelo da base de dados relacional chegou-se a um modelo documental que maximiza este paradigma. Para isso, utilizou-se a base a dados documental **MongoDB**, visto que é aquela que tivemos mais contacto ao longo das aulas, tendo assim um conhecimento prévio e aprofundado do seu funcionamento.

O modelo documental implementado pretende representar em cada documento JSON toda a informação associada aos departamentos da empresa. Esta visão alto nível foi benéfica pois assim temos um balanceamento entre a informação contida em cada documento e o número de documentos presentes na coleção. Por exemplo, se cada um dos documentos JSON representasse a informação sobre um *employee*, então haveria muitos documentos com pouca informação associada, e se cada documento representasse a informação do país então havia poucos documentos com muita informação associada. Ambos os exemplos eram extremos e iriam ser limitantes tanto na implementação das *queries* como na obtenção de informação pertinente, quer seja por causa da falta de informação quer seja por ter muita informação *nested*. Para além desta razão, visto que a base de dados implementada tem como objetivo obter informações sobre uma empresa, achamos que o esquema estar orientado aos departamentos é o equilíbrio ideal e a escolha mais lógica para organizar as informação da mesma. Desta forma, foi implementada a seguinte estrutura:

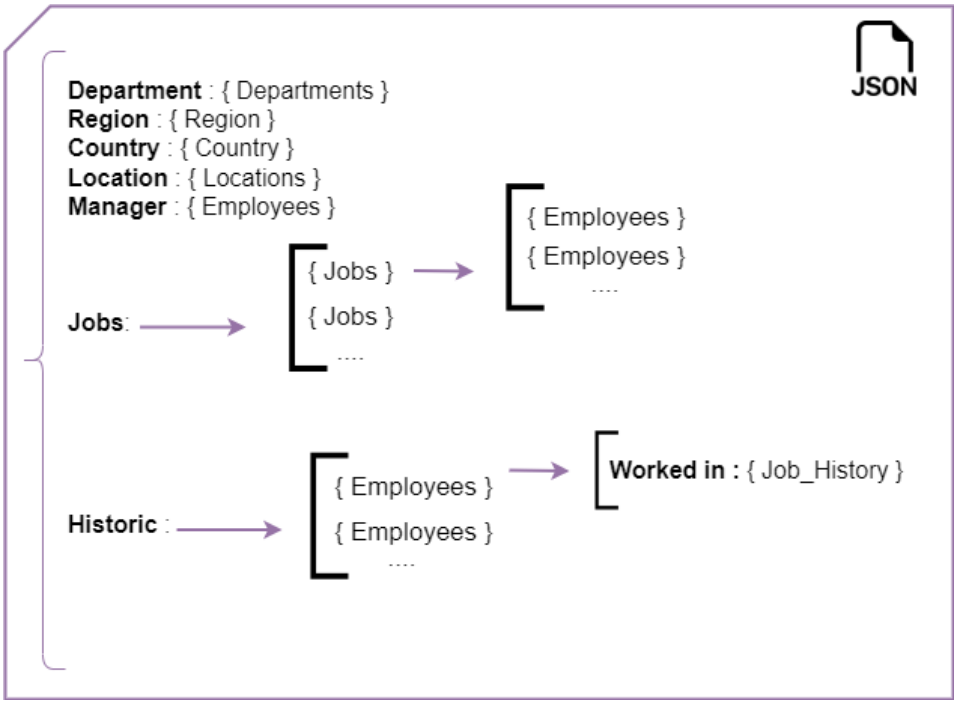


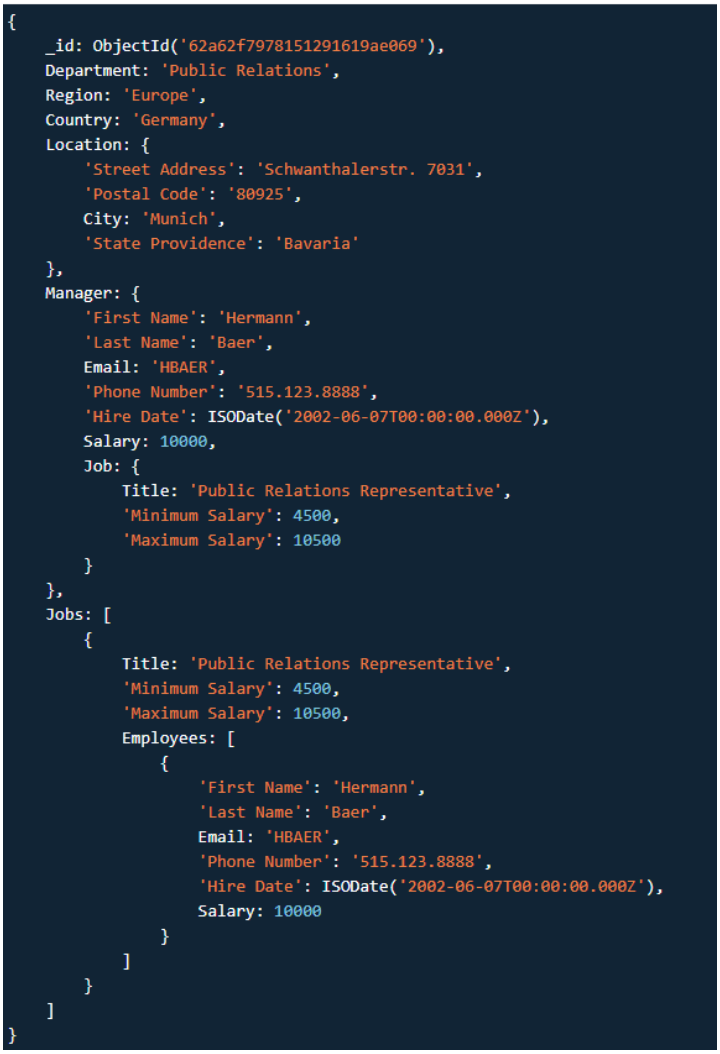
Figura 1.1: Esquema da arquitetura de um documento do MongoDB

Neste esquema conseguimos ver a estrutura para cada documento JSON, que contém a informação acerca do nome e da localização do departamento, assim como a informação sobre o *manager* desse departamento.

Além disso, também vamos armazenar em cada documento as profissões e respetivos *employees* associados ao departamento. Esta decisão foi tomada de forma a evitar dados repetidos e redundantes, visto que se listássemos os *employees* e associássemos a cada um deles a respetiva profissão iria haver muitos *employees* com profissões repetidas.

Finalmente, decidiu-se representar em cada documento a informação sobre o histórico de cada departamento, isto é, os *employees* que já estiveram naquele departamento e a respetiva profissão que desempenharam. Optamos por colocar o histórico no primeiro nível do ficheiro JSON e associado ao departamento por simplificação e eficiência. Apesar de poder existir alguma repetição de informação sobre os *employees* no histórico, esta implementação diminui os aninhamentos no ficheiro e aumenta a eficiência na procura por histórico.

Como exemplo, podemos ver o ficheiro JSON do departamento *Public Relations* que segue a estrutura apresentada em cima. É de notar que neste ficheiro não consta a secção *historic*, pois não existe essa informação.



```
{
    _id: ObjectId('62a62f7978151291619ae069'),
    Department: 'Public Relations',
    Region: 'Europe',
    Country: 'Germany',
    Location: {
        'Street Address': 'Schwanthalerstr. 7031',
        'Postal Code': '80925',
        City: 'Munich',
        'State Providence': 'Bavaria'
    },
    Manager: {
        'First Name': 'Hermann',
        'Last Name': 'Baer',
        Email: 'HBAER',
        'Phone Number': '515.123.8888',
        'Hire Date': ISODate('2002-06-07T00:00:00.000Z'),
        Salary: 10000,
        Job: {
            Title: 'Public Relations Representative',
            'Minimum Salary': 4500,
            'Maximum Salary': 10500
        }
    },
    Jobs: [
        {
            Title: 'Public Relations Representative',
            'Minimum Salary': 4500,
            'Maximum Salary': 10500,
            Employees: [
                {
                    'First Name': 'Hermann',
                    'Last Name': 'Baer',
                    Email: 'HBAER',
                    'Phone Number': '515.123.8888',
                    'Hire Date': ISODate('2002-06-07T00:00:00.000Z'),
                    Salary: 10000
                }
            ]
        }
    ]
}
```

Figura 1.2: Ficheiro JSON do departamento *Public Relations*

Para a passagem dos dados do Oracle para os documentos JSON utilizou-se o programa `migration_mongodb.py` que vai efetuar a conexão com a base de dados Oracle, de modo a permitir a conversão para os ficheiros JSON do modelo documental. Este programa foi criado para facilitar a implementação dos dados em MongoDB de forma a que seja possível uma transferência eficiente e que se adaptasse à alteração dos dados em Oracle. De seguida, podemos ver parte do programa implementado que vai percorrer todos os departamentos e a cada um deles armazenar a respetiva informação (Localização, Trabalhos, Historio) num dicionário que será passado para um ficheiro em JSON.

```
try:
    con = cx_Oracle.connect(user, pwd,
        '{}:{}/{}'.format(host,portno,service_name))
    print('Successful connection!')
    cursor = con.cursor()
    query = "SELECT * FROM DEPARTMENTS"
    cursor.execute(query)
    result = cursor.fetchall()

    #Percorre todos os departamentos e para cada um constroi um ficheiro em
        JSON
    for department in result:

        #Atributos do departamento
        IdDepartment = department[0]
        departmentName = department[1]
        managerID = department[2]
        locationID = department[3]

        #Dicionrio que vai conter o conteudo JSON que depois ser escrito no
            ficheiro
        dictFile = {}
        # Criao do nome do ficheiro a ser utilizado
        fileName = "Department-" + departmentName + "-" + str(IdDepartment) +
            ".json"
        # Ficheiro JSON
        jsonDepartment = open(fileName, "w")

        #Caso o departamento tenha uma localizao atribuida
        if locationID != None:
            dictFile = buildDepartmentLocation(dictFile)

        #Caso o departamento tenha um manager atribuido
        if managerID != None:
           dictFile = buildDepartmentManger(dictFile)

        #Selecionar os empregos e employees associados ao departamento
        dictFile = buildDepartmentJobs(dictFile)
        #Selecionar o histrico associado ao departamento
        dictFile = buildDepartmentHistoric(dictFile)
        #Retira do conteudo JSON os campos a null
        dictFile = deleteNullFromDict(dictFile)

        #Escreve o conteudo JSON (dicionrio) no ficheiro
        json.dump(dictFile, jsonDepartment, indent = 6,default=str)
        jsonDepartment.close()

except cx_Oracle.DatabaseError as er:
    print('There is an error in the Oracle database:', er)
```

4 Base de Dados por Grafos - *Neo4j*

A partir do modelo relacional inicial, implementou-se também um modelo de base de dados não relacional e orientado a grafos, utilizando-se, para o efeito, o **Neo4j**. É de realçar que a escolha desta base de dados em específico deveu-se ao facto de esta ter sido abordada em contexto de aula, havendo, por isso um maior conhecimento e destreza na sua utilização.

Assim sendo, o primeiro passo deste processo de transição consistiu no desenvolvimento de um *script* em *Python*, `migration_neo4j.py`, no qual se efetuou a conexão à base de dados relacional, e, recorrendo a um cursor, foram obtidos os vários registos desta BD. De seguida, e partindo destes dados, as *queries* em *cypher* necessárias para adaptar o esquema em *Oracle*, foram escritas num ficheiro de texto. Estes comandos são depois copiados para a *bash* do *Neo4j*, criando-se a estrutura final em grafo.

Tal como é possível observar na figura 1.5, as tabelas no *Oracle* referentes a entidades, tais como *Employee* ou *Location*, deram origem a nodos, enquanto que as *foreign keys* são representadas pelas relações entre estes nodos. É importante salientar que para os nodos *Employee* e *Department* foram criadas duas relações, sendo que a relação *Work* existe para representar o departamento no qual um colaborador trabalha no presente, enquanto que a relação *Worked* representa o departamento no qual um dado colaborador trabalhou no passado. Esta relação *Worked* possui como atributos os vários dados do *Job History* do respetivo trabalhador, tais como o nome do *Job* que efetuou e ainda as datas em que começou e terminou esse *Job*. Na figura 1.3 é possível ver uma pequena secção do grafo, gerado pela *query* da figura 1.4, em *Neo4j*.

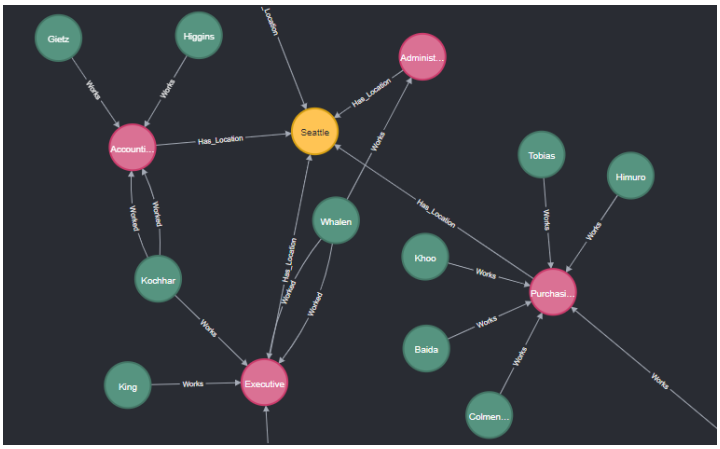


Figura 1.3: Exemplo de parte do grafo do *Neo4j*

```
MATCH (e:Employee)-[w:Works]→(d:Department)-[r:Has_Location]→(l:Location)
-[r1:Has_Country]→(c:Country)
RETURN c.name AS Country,w,r,r1,l,d, e AS Employee
```

Figura 1.4: Query que deu origem à secção da figura anterior

Por outro lado, e de forma a otimizar a pesquisa neste paradigma orientado a grafos, tivemos em conta a direção das relações entre nodos. Deste modo, no esquema em *Oracle* as relações 1:N foram transpostas em *Neo4j* como relações entre origem e destino, cujo os nodos de origem são as entidades com multiplicidade N e os seus respetivos nodos

destino serão a entidade com multiplicidade 1. Esta decisão baseou-se no facto de que assim conseguimos simular múltiplas relações 1:1, tirando o melhor partido da eficiência deste paradigma de grafos.

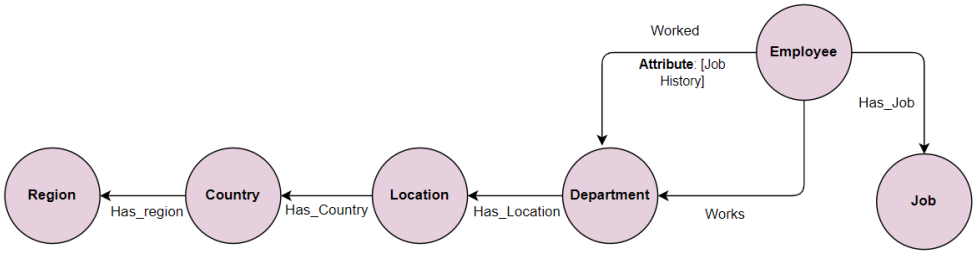


Figura 1.5: Esquema da arquiterura do Neo4j

Além disso, é também de realçar que de forma a facilitar o processo de criação de relações entre nodos, a partir dos dados na BD relacional, foram usados os *ids* das respetivas entidades, ou seja, os nodos possuem inicialmente um identificador no seu conjunto de atributos. No entanto, assim que todas relações e nodos são criados, estes *ids* deixam de ser necessários pelo que são removidos.

De seguida, podemos ver parte do programa implementado, que vai escrever num ficheiro de texto as *queries* em *cypher* necessárias para criar os diversos nodos e relações já explicadas do sistema.

```
try:
    con = cx_Oracle.connect(user, pwd,
        '{}:{}/{}'.format(host,portno,service_name))
    print('Sucessful connection!')
    cursor = con.cursor()

    #  Criao  do nome do ficheiro a ser utilizado
    fileName = "Script" + ".txt"
    # Ficheiro
    cyph = open(fileName, "w")

    ### Criar nodos Region , Country e Location ###
    createRegionNodes(cursor,cyph)
    createCountryNodes(cursor,cyph)
    createLocationNodes(cursor,cyph)
    createDepartmentNodes(cursor,cyph)
    createEmployeeNodes(cursor,cyph)
    createJobNodes(cursor,cyph)

    # Criar relacao entre location e Country
    create_country_rel(cursor,cyph)
    # Criar relacao entre Country e Region
    create_region_rel(cursor,cyph)
     # Criar relacao entre Departament e Location
    create_dep_rel(cursor,cyph)
    # Criar relacao Employee e Job
    create_job_rel(cursor,cyph)
    # Criar relacao Employee e Department onde trabalha atualmente
    create_emp_dep_rel(cursor,cyph)
    # Criar relacao Employee e Department Onde trabalhou na passado (job
        history)
    create_emp_dep_history_rel(cursor,cyph)

    #Remover os ids dos nodos
    remove_ids(cyph)
```

5 Desenvolvimento e Performance das Queries

Nesta secção está apresentado um conjunto de *queries* estratégicas, escolhidas de forma a tirar partido das implementações e funcionalidades de cada base de dados (seja em *Oracle*, *MongoDB* ou *Neo4J*). Desta forma, visto que existem estruturas diferentes de armazenamento dos dados, é possível testar e comparar as diferentes performances para cada *query*.

Os valores do tempo de execução de cada *query* foram obtidos numa só máquina de forma a obter valores realistas e consistentes para todas as implementações de queries. Como forma de obter tempos de execução mais próximos da realidade, registaram-se 5 medições sendo o tempo de execução da *query* dado pela média destas 5 medições. Por uma questão de consistência dos resultados, ignoramos a primeira execução da *query* uma vez que este valor é bastante discrepante dos demais. Para obtenção das medições para o Oracle e para o Neo4j, após a execução da *query* estas BDs já devolvem o tempo de execução da *query*. No entanto, para o Oracle, para além do tempo de execução, esta também apresenta o tempo necessário para realizar o *print* do resultado (*fetching*), assim apenas contabilizou-se o tempo de execução sem o *fecthing*. Para a obtenção das medições para o MongoDB, foi utilizada a função apresentada abaixo que permite fazer a diferença entre os tempos no inicio do *query* e no fim da execução desta.

```
function time(command) {
    const t1 = new Date();
    const result = command();
    const t2 = new Date();
    print("time: " + (t2 - t1) + "ms");
    return result;
}
```

Esta metodologia rigorosa e consistente foi nos benéfica para a obtenção de tempos de execução mais próximos do real e mais estáveis.

Em cada subsecção seguinte será apresentada a implementação das *queries* em *Oracle*, *MongoDB* e *Neo4j*, e o respetivo tempo de execução obtido em milissegundos (ms). O resultado de cada uma das implementações das querys encontra-se nos Anexos na secção 8.

5.1 *Query* 1 - Listar o número de departamentos em cada país

Nas secções em baixo encontram-se as implementações da *query* 1 nas várias BDs que tem como objetivo percorrer todos departamentos e listar o número de departamentos que estão em cada país. Desta forma analisamos a performance de percorrer os departamentos para cada BD.

Oracle

```
SELECT COUNTRY_NAME, COUNT( DISTINCT DEPARTMENT_ID ) AS NR_DEPARTMENTS
FROM COUNTRIES C, DEPARTMENTS D, LOCATIONS L
WHERE D.LOCATION_ID = L.LOCATION_ID AND L.COUNTRY_ID = C.COUNTRY_ID
group by COUNTRY_NAME ORDER BY NR_DEPARTMENTS desc;
```

Listing 1.1: Implementação em Oracle da *query* 1

MongoDB

```
db.departments.aggregate( [{$group: {_id : "$Country",'count' : {$sum:1}}},
    { $sort: {count : -1} }])
```

Listing 1.2: Implementação em MongoDB da query 1

Neo4j

```
MATCH
    (d:Department)-[r:Has_Location]->(l:Location)-[r1:Has_Country]->(c:Country)
RETURN c.name AS Country, count(d) AS TOTAL
ORDER BY TOTAL DESC
```

Listing 1.3: Implementação em Neo4j da *query* 1

Análise de Resultados

Após efetuar os testes de performance desta *query* para os diferentes modelos, podemos efetuar uma análise crítica dos resultados. Podemos observar que esta *query* possui menor tempo de execução para o MongoDB, seguida do Oracle e por fim o Neo4J. Isto deve-se à estrutura dos dados em cada modelo, ou seja, visto que o MongoDB possui um documento por departamento e esta *query* contabiliza o número de departamentos por país, é de esperar que seja bastante eficiente neste modelo documental. Por outro lado, nos outros modelos o acesso não é tão direto, o que pode levar a que ocorra maior *delay*. No Neo4J é necessário percorrer várias relações e vários nodos para poder efetuar esta *query*, o que leva a um maior tempo de execução. No Oracle é também necessário passar por várias relações e entidades o que faz que demore mais tempo em relação ao MongoDB.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	7	5	10	6	9	**7.4**
MongoDB	2	3	4	2	3	**2.8**
Neo4J	8	10	6	21	5	**10**

Tabela 1.1: Tempo de execução para a *query* 1

5.2 *Query* 2 - Listar o número de *employees* em cada país

Nas secções em baixo encontram-se as implementações da *query* 2 nas várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura por todos os trabalhadores da empresa em cada país.

Oracle

```
SELECT COUNTRY_NAME, COUNT( DISTINCT EMPLOYEE_ID ) AS NR_EMPLOYEES
FROM COUNTRIES C, DEPARTMENTS D, LOCATIONS L, EMPLOYEES E
WHERE D.LOCATION_ID = L.LOCATION_ID AND L.COUNTRY_ID = C.COUNTRY_ID
  AND E.DEPARTMENT_ID = D.DEPARTMENT_ID
group by COUNTRY_NAME ORDER BY NR_EMPLOYEES desc;
```

Listing 1.4: Implementação em Oracle da *query* 2

MongoDB

```
db.departments.aggregate([
{
  "$project":{
    Country : 1,
    "employees_count":{
      "$sum":{
        "$map":{
          "input":"$Jobs",
          "in":{"$size": {"$ifNull": ["$$this.Employees", []] }}
}}}},
{ "$group": {_id : "$Country", 'count' : {$sum:"$employees_count"}}
},
{ $sort: {count : -1} }])
```

Listing 1.5: Implementação em MongoDB da *query* 2

Neo4j

```
MATCH (e:Employee)-[w:Works]->(d:Department)-[r:Has_Location]->(l:Location)
                                            -[r1:Has_Country]->(c:Country)
RETURN c.name AS Country, count(e) AS TOTAL
ORDER BY TOTAL DESC
```

Listing 1.6: Implementação em Neo4j da *query* 2

Análise de Resultados

Nesta *query* obtivemos um melhor tempo de execução para o modelo por grafos (Neo4J), seguido do MongoDB e, por fim, o Oracle. Podemos concluir que, para o MongoDB, houve uma certa penalização no tempo de execução visto que foi necessário percorrer a lista de *employees* que se encontra *nested* na lista de *Jobs*. No caso do Oracle, o facto de ser necessário atravessar o modelo através de várias junções de tabelas também se mostrou causador de algum *delay* na execução desta *query*. Por fim, o Neo4J foi o modelo que demonstrou melhor desempenho sem haver tanta penalização na sua travessia.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	11	49	11	13	11	**19**
MongoDB	7	5	3	13	36	**12.8**
Neo4J	20	7	4	8	6	**9**

Tabela 1.2: Tempo de execução para a *query* 2

5.3 *Query* 3 - Listar o *employees* que mais ganha na empresa

Na secção em baixo encontram-se as implementações da *query* 3 nas várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura por *employee* da empresa e por salário.

Oracle

```
SELECT FIRST_NAME, LAST_NAME , SALARY, JOB_TITLE
    FROM EMPLOYEES E
        JOIN JOBS J on E.JOB_ID = J.JOB_ID
            WHERE E.SALARY =(SELECT MAX(SALARY) FROM EMPLOYEES);
```

Listing 1.7: Implementação em Oracle da *query* 3

MongoDB

É importante realçar que no MongoDB, a implementação da *query* foi feita em duas partes, sendo que em primeiro lugar foi necessário obter uma lista de todos os salários existentes e depois obter aquele que seria o mais **alto**.

```
listaSalarios = db.departments.distinct("Jobs.Employees.Salary")
maxSalary = listaSalarios[listaSalarios.length-1]
db.departments.aggregate([
    {$unwind : "$Jobs"},
    {$unwind : "$Jobs.Employees"},
    {$match : {"Jobs.Employees.Salary": maxSalary}},
    {$group:
        {_id:{ First_name: "$Jobs.Employees.First Name",
               Last_Name:"$Jobs.Employees.Last Name",
               Job:"$Jobs.Title",
               Salary : "$Jobs.Employees.Salary"}}])
.pretty();
```

Listing 1.8: Implementação em MongoDB da *query* 3

Neo4j

```
MATCH (e:Employee)
WITH MAX (e.salary) AS max
MATCH (e:Employee)-[:Has_Job]->(j:Job) WHERE e.salary = max
RETURN e.first_name AS FIRST_NAME, e.last_name AS LAST_NAME,e.salary AS
    SALARY, j.title AS JOB_TITLE
```

Listing 1.9: Implementação em Neo4j da *query* 3

Análise de Resultados

Tal como se pode observar pela tabela abaixo, esta *query* possui melhor performance no *Neo4j*. No *Oracle*, de forma a devolver o *Job* do trabalhador com o salário mais alto, foi necessário realizar uma operação *join* o que contribuiu para o elevado tempo de execução. Por outro lado, no *MongoDB*, a pesquisa por *employees* é custosa pois estes estão inseridos em documentos *nested*, dentro do *Job*. Desta forma, no *Neo4j*, havendo uma relação direta entre *Employee* e *Job*, faz com que esta pesquisa seja mais eficiente.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	12	7	7	9	16	**10.2**
MongoDB	11	5	8	5	8	**7.4+4***
Neo4J	4	14	6	4	5	**6.6**

Tabela 1.3: Tempo de execução para a *query* 3

5.4 *Query* 4 - Listar o *employee* que menos ganha na empresa

Na secção em baixo encontram-se as implementações da *query* 4 nas várias BDs. Há semelhança da *query* anterior, o objetivo consistiu em analisar a performance das bases de dados quando é feita uma procura por *employee* da empresa e por salário. Visto que a *query* 3 e a *query* 4 são praticamente iguais, os seus tempos de execução não serão muito diferentes.

Oracle

```
SELECT FIRST_NAME, LAST_NAME , SALARY, JOB_TITLE
    FROM EMPLOYEES E
        JOIN JOBS J on E.JOB_ID = J.JOB_ID
            WHERE E.SALARY =(SELECT MIN(SALARY) FROM EMPLOYEES);
```

Listing 1.10: Implementação em Oracle da *query* 4

MongoDB

É importante realçar que no MongoDB, a implementação da *query* foi feita de forma análoga à *query* 3, sendo que em primeiro lugar foi necessário obter uma lista de todos os salários existentes e depois obter aquele que seria o mais **baixo**. Tendo-se este valor efetua-se a *query* principal. Na tabela de medição abaixo apresentada, o valor com um asterisco corresponde ao tempo que demorou a obter este salário mínimo na BD.

```
listaSalarios = db.departments.distinct("Jobs.Employees.Salary")
minSalary = listaSalarios[0]
db.departments.aggregate([
    {$unwind : "$Jobs"},
    {$unwind : "$Jobs.Employees"},
    {$match : {"Jobs.Employees.Salary": minSalary}},
    {$group:{_id:{
        First_name: "$Jobs.Employees.First Name",
        Last_Name:"$Jobs.Employees.Last Name",
        Job:"$Jobs.Title",
        Salary : "$Jobs.Employees.Salary"}}}])
.pretty();
```

Listing 1.11: Implementação em MongoDB da *query* 4

Neo4j

```
MATCH (e:Employee)
WITH MIN (e.salary) AS min
MATCH (e:Employee)-[:Has_Job]->(j:Job) WHERE e.salary = min
RETURN e.first_name AS FIRST_NAME, e.last_name AS LAST_NAME,e.salary AS
    SALARY, j.title AS JOB_TITLE
```

Listing 1.12: Implementação em Neo4j da *query* 4

Análise de resultados

Relativamente à análise de resultados, a justificação é análoga à da *query* 3.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	12	8	9	8	7	**8.8**
MongoDB	3	4	6	4	3	**4+4***
Neo4J	9	5	5	9	4	**6.4**

Tabela 1.4: Tempo de execução para a *query* 4

5.5 *Query* 5 - Listar o *employee* que está a trabalhar há mais tempo

Na secção em baixo encontram-se as implementações da *query* 5 nas várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura por *employees* da empresa e por data de contratação.

Oracle

```
SELECT FIRST_NAME , LAST_NAME , HIRE_DATE ,floor(months_between(
    CURRENT_DATE , HIRE_DATE ) /12) AS NR_YEARS_WORKING
    FROM EMPLOYEES
        WHERE HIRE_DATE =(SELECT MIN(HIRE_DATE) FROM EMPLOYEES);
```

Listing 1.13: Implementação em Oracle da *query* 5

MongoDB

É importante realçar que no MongoDB, a implementação desta *query* foi feita em duas partes, sendo que em primeiro lugar foi necessário obter uma lista de todas datas de contratação existentes e depois obter aquela que seria a mais **antiga**. Tendo-se este valor efetua-se a *query* principal. Na tabela de medição abaixo apresentada, o valor com um asterisco corresponde ao tempo que demorou a obter, na BD, a data de contratação

mais antiga.

```
listaDatas = db.departments.distinct("Jobs.Employees.Hire Date")
oldest = listaDatas[0]
 db.departments.aggregate([
    {$unwind : "$Jobs"},
    {$unwind : "$Jobs.Employees"},
    {$match : {"Jobs.Employees.Hire Date": oldest}},
    {$group:
        {_id:{
            First_name: "$Jobs.Employees.First Name",
            Last_Name:"$Jobs.Employees.Last Name",
            Job:"$Jobs.Title", "Hire Date" : "$Jobs.Employees.Hire Date"}}},
    {$addFields:{ YearsWorking : {$subtract:["$$NOW",new
        Date("$Jobs.Employees.Hire Date")] }}} ])
.pretty();
```

Listing 1.14: Implementação em MongoDB da *query* 5

Neo4j

```
MATCH (e:Employee)
WITH MIN (e.date_hired) AS min
MATCH (e:Employee) WHERE e.date_hired = min
RETURN e.first_name as FIRST_NAME , e.last_name as LAST_NAME , e.date_hired
    AS DATE_HIRED,duration.inMonths(e.date_hired, date()).months/12 AS
    NR_WORKING
```

Listing 1.15: Implementação em Neo4j da *query* 5

Análise de Resultados

Tal como se pode observar pela tabela abaixo, esta *query* possui melhor performance no *Neo4j*. No *MongoDB*, a pesquisa por *employees* é custosa pois estes estão inseridos em documentos *nested*, dentro do *Job*. Relativamente, ao *Oracle*, este não possui uma elevada performance, apesar de a *query* ser semelhante à implementada em *Neo4j*. Isto poderá dever-se ao facto de que a contabilização do número de anos de trabalho de um *employee* é mais custosa no *Oracle*, comparativamente ao *Neo4j*.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	13	5	13	13	6	**10**
MongoDB	6	5	6	10	4	**6.2+2***
Neo4J	4	2	2	5	2	**3**

Tabela 1.5: Tempo de execução para a *query* 5

5.6 *Query* 6 - Listar o *employees* que está a trabalhar há menos tempo

Na secção em baixo encontram-se as implementações da *query* 6 nas várias BDs. Há semelhança da *query* anterior, o objetivo consistiu em analisar a performance das bases de dados quando é feita uma procura por *employee* da empresa e por data de contratação. Visto que a *queries* 5 e 6 são praticamente iguais, os seus tempos de execução não serão muito diferentes.

Oracle

```
SELECT FIRST_NAME , LAST_NAME , HIRE_DATE ,floor(months_between(
    CURRENT_DATE , HIRE_DATE ) /12) AS NR_YEARS_WORKING
    FROM EMPLOYEES
        WHERE HIRE_DATE =(SELECT MAX(HIRE_DATE) FROM EMPLOYEES);
```

Listing 1.16: Implementação em Oracle da *query* 6

MongoDB

É importante realçar que no MongoDB, a implementação desta *query* foi feita de forma análoga à *query* 5, sendo que em primeiro lugar foi necessário obter uma lista de todas datas de contratação existentes e depois obter aquela que seria a mais **recente**. Tendo-se este valor efetua-se a *query* principal. Na tabela de medição abaixo apresentada, o valor com um asterisco corresponde ao tempo que demorou a obter, na BD, a data de contratação mais recente.

```
listaDatas = db.departments.distinct("Jobs.Employees.Hire Date")
recent = listaDatas[listaDatas.length-1]
db.departments.aggregate([
    {$unwind : "$Jobs"},
    {$unwind : "$Jobs.Employees"},
    {$match : {"Jobs.Employees.Hire Date": recent}},
    {$group:
        {_id:{
            First_name: "$Jobs.Employees.First Name",
            Last_Name:"$Jobs.Employees.Last Name",
            Job:"$Jobs.Title", "Hire Date" : "$Jobs.Employees.Hire Date"}}},
    {$addFields:{ YearsWorking : {$subtract:["$$NOW",new
        Date("$Jobs.Employees.Hire Date")]}}}])
.pretty();
```

Listing 1.17: Implementação em MongoDB da *query* 6

Neo4j

```
MATCH (e:Employee)
WITH MAX (e.date_hired) AS max
MATCH (e:Employee) WHERE e.date_hired = max
RETURN e.first_name as FIRST_NAME , e.last_name as LAST_NAME , e.date_hired
    AS DATE_HIRED,duration.inMonths(e.date_hired, date()).months/12 AS
    NR_WORKING
```

Listing 1.18: Implementação em Neo4j da *query* 6

Análise de Resultados

Relativamente à análise de resultados, a justificação é análoga à da *query* 5.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	24	12	13	7	7	**12.6**
MongoDB	3	2	6	7	3	**4.2+2***
Neo4J	5	5	5	4	5	**4.8**

Tabela 1.6: Tempo de execução para a *query* 6

5.7 *Query* 7 - Ordenar as várias profissões pelo número de *employees*

Nas secções em baixo encontram-se as implementações da *query* 7 para as várias BDs, de forma a analisar a performance das mesmas quando é feita uma ordenação por profissão com procura por *employees*.

Oracle

```
SELECT JOB_TITLE, COUNT( DISTINCT EMPLOYEE_ID ) AS NR_EMPLOYEES
FROM JOBS J, EMPLOYEES E
WHERE J.JOB_ID = E.JOB_ID
group by JOB_TITLE ORDER BY NR_EMPLOYEES desc;
```

Listing 1.19: Implementação em Oracle da *query* 7

MongoDB

```
db.departments.aggregate([
{$unwind:"$Jobs"},
{
 $group: {
   _id : "$Jobs.Title",
   'count' :
         {"$sum":{"$size": {"$ifNull": ["$Jobs.Employees", []] }}}
}
},
{ $sort: {count : -1} }
])
```

Listing 1.20: Implementação em MongoDB da *query* 7

Neo4j

```
MATCH (e:Employee)-[r:Has_Job]->(j:Job)
RETURN j.title AS Job, count(e) AS TOTAL
ORDER BY TOTAL DESC
```

Listing 1.21: Implementação em Neo4j da *query* 7

Análise de Resultados

Podemos observar que esta *query* teve um tempo de execução menor para o Neo4J, seguido do MongoDB e Oracle. No caso do Oracle, foi necessário efetuar a junção de uma tabela e uma contagem, o que pode adicionar alguma penalização nos tempos de execução. Para o MongoDB o tempo foi mais rápido ligeiramente visto que a procura era mais direta no documento. No entanto, a necessidade de fazer *unwind* da lista de Jobs adicionou alguma complexidade que não existiu no Neo4J, conferindo a este último o melhor tempo de execução.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	11	8	10	7	7	**8.6**
MongoDB	5	13	4	2	3	**5.4**
Neo4J	2	3	10	5	3	**4.6**

Tabela 1.7: Tempo de execução para a query 7

5.8 *Query* 8 - Ordenar cada departamento pelo número de *employees* que lá trabalham

Nas secções em baixo encontram-se as implementações da *query* 8 para as várias BDs, de forma a analisar a performance das mesmas quando é feita uma ordenação por departamentos com procura por *employees*.

Oracle

```
SELECT D."DEPARTMENT_NAME" , COUNT (E.EMPLOYEE_ID) Nr_Empregados
    FROM EMPLOYEES E
        RIGHT JOIN DEPARTMENTS D ON D."DEPARTMENT_ID" = E."DEPARTMENT_ID"
            GROUP BY D.DEPARTMENT_NAME
                ORDER BY Nr_Empregados DESC;
```

Listing 1.22: Implementação em Oracle da query 8

MongoDB

```
db.departments.aggregate( [
    { $project :
        { _id: 0, department: "$Department", JobsEmployee:
                                { $reduce:
                                    { input: "$Jobs.Employees.Email",
                                        initialValue: [ ], in: { $concatArrays:
                                        [ "$$value", "$$this" ] } } } } },
    { $group: { _id: "$department", numberEmployees: { $sum: { $size: {
        $ifNull: ["$JobsEmployee", []] } } } } },
    { $sort: {numberEmployees: -1} }
])
```

Listing 1.23: Implementação em MongoDB da *query* 8

Neo4j

```
MATCH ()-[w:Works*0..1]->(d:Department)
RETURN DISTINCT d.name AS DEPARTMENT_NAME, COUNT(*)-1 AS NR_EMPLOYEES
ORDER BY NR_EMPLOYEES DESC
```

Listing 1.24: Implementação em Neo4j da *query* 8

Análise de Resultados

Tal como podemos ver na tabela, os tempos de execução do MongoDB são mais baixo do que nas outras *queries*, pois os documentos em MongoDB estão ordenados por cada departamentos sendo por isso mais fácil e mais rápido obter informação associada aos departamentos. Já para o Oracle obtemos tempos de execução mais elevados, devido à necessidade de ter fazer *JOINs* para associar a informação presente na tabela *Departments* com a informação presente em *Employees*.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	12	13	13	11	9	**11.6**
MongoDB	4	9	4	3	3	**4.6**
Neo4J	14	9	5	4	3	**7**

Tabela 1.8: Tempo de execução para a *query* 8

5.9 *Query* 9 - Listar o salário médio por departamento

Nas secções em baixo encontram-se as implementações da *query* 9 para as várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura pelo o salário de cada *employee*.

Oracle

```
SELECT DEPARTMENTS.DEPARTMENT_NAME Departamento, AVG(SALARY) Media FROM
    EMPLOYEES, DEPARTMENTS
WHERE DEPARTMENTS.DEPARTMENT_ID = EMPLOYEES.DEPARTMENT_ID
GROUP BY EMPLOYEES.DEPARTMENT_ID, DEPARTMENTS.DEPARTMENT_NAME
ORDER BY Media ASC;
```

Listing 1.25: Implementação em Oracle da query 9

MongoDB

```
db.departments.aggregate([
    {$project: { _id :0, department : "$Department", salaries:
        { $reduce: { input: "$Jobs.Employees.Salary", initialValue: [ ], in:
            { $concatArrays: [ "$$value", "$$this" ] } } }}
    },
    {$project:
        { _id :0 , Departamento : "$department", Media : { $avg: "$salaries"
            } }
    },
    {$sort: {Media: 1} }, {"$match": { "Media" :{ "$ne" : null } } } ])
```

Listing 1.26: Implementação em MongoDB da *query* 9

Neo4j

```
MATCH (e:Employee)-[:Works]->(d:Department)
RETURN d.name as Department, AVG(e.salary) as Salary
ORDER BY Salary ASC
```

Listing 1.27: Implementação em Neo4j da *query* 9

Análise de Resultados

Conseguimos aferir que as base de dados não relacionais tiveram um melhor desempenho. No caso do MongoDB, a justificação advém do facto de que a estrutura dos documentos é orientada aos departamentos e trata-se também de uma *query* orientada ao departamento. Por sua vez, o Neo4j consegue obter um desempenho ainda melhor, isto pode ser devido a uma combinação de fatores quer pela complexidade reduzida da *query*, quer pela a estrutura do modelo de grafos que permite uma pesquisa mais rápida. Relativamente ao Oracle, podemos também explicar a redução do desempenho devido a complexidade da *query* que afeta o desempenho da mesma.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	15	12	13	10	13	**12.6**
MongoDB	6	7	8	4	6	**6.2**
Neo4J	2	5	5	4	6	**4.4**

Tabela 1.9: Tempo de execução para a *query* 9

5.10 *Query* 10 - Listar o histórico de cada *employee*

Nas secções em baixo encontram-se as implementações da *query* 10 para as várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura pelo histórico de cada *employee*.

Oracle

```
SELECT E.FIRST_NAME EMPLOYEE_FIRST, E.LAST_NAME EMPLOYEE_LAST, J.START_DATE,
    J.END_DATE, JB.JOB_TITLE JOB, D.DEPARTMENT_NAME DEPARTMENT FROM
    JOB_HISTORY J
    JOIN DEPARTMENTS D on D.DEPARTMENT_ID = J.DEPARTMENT_ID
        JOIN EMPLOYEES E on E.EMPLOYEE_ID = J.EMPLOYEE_ID
            JOIN JOBS JB on JB.JOB_ID = J.JOB_ID;
```

Listing 1.28: Implementação em Oracle da query 10

MongoDB

```
db.departments.aggregate( [
    {$unwind: "$Historic"},
    { $project :
        { _id: 0,
        "First Name": "$Historic.First Name",
        "Last Name": "$Historic.Last Name",
        "Department where the employee worked": "$Department",
        "Past Jobs": "$Historic.Worked in"}}
]).pretty()
```

Listing 1.29: Implementação em MongoDB da *query* 10

Neo4j

```
MATCH (e:Employee)-[w:Worked]->(d:Department)
RETURN e.first_name AS Employee_Name, e.last_name As Employee_Surname,
    w.start_date AS Start_Date, w.end_date AS End_Date, w.job_name AS Job,
    d.name AS Department
```

Listing 1.30: Implementação em Neo4j da *query* 10

Análise de Resultados

Na tabela abaixo podemos ver que a BD que dá menor tempos de execução é a Neo4j pois para obter a informação sobre o histórico de cada *employee* basta verificarmos quais os *employees* que participam na relação *Worked* entre o *employee* e o *department*, isto é, basta verificar a aresta *worked* do grafo implementado. Para o Oracle, o tempo de execução é maior devido à implementação da *query* necessitar vários JOINs para obter a informação que faz com que a execução seja mais lenta.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	8	8	8	11	10	**9**
MongoDB	3	3	5	3	2	**3.2**
Neo4J	3	2	3	2	3	**2.6**

Tabela 1.10: Tempo de execução para a *query* 10

5.11 *Query* 11 - Listar o salário médio por profissão

Finalmente, nas secções em baixo encontram-se as implementações da *query* 11 para as várias BDs, de forma a analisar a performance das mesmas quando é feita uma procura pelo o salário de cada profissão.

Oracle

```
SELECT JOBS.JOB_TITLE Profisso, AVG(SALARY) Media FROM EMPLOYEES, JOBS
WHERE JOBS.JOB_ID = EMPLOYEES.JOB_ID
GROUP BY EMPLOYEES.JOB_ID, JOB_TITLE
ORDER BY Media ASC;
```

Listing 1.31: Implementação em Oracle da query 11

MongoDB

```
db.departments.aggregate([
    {$unwind : "$Jobs"},
    {$project: { _id :1 , Profissao : "$Jobs.Title" , Media : { $avg :
        "$Jobs.Employees.Salary"}}}, { $sort: {Media: 1} }
])
```

Listing 1.32: Implementação em MongoDB da *query* 11

Neo4j

```
MATCH (e:Employee)-[]->(j:Job)
RETURN j.title as Job, AVG(e.salary) as Salary
ORDER BY Salary ASC
```

Listing 1.33: Implementação em Neo4j da *query* 11

Análise de Resultados

Numa análise comparativa podemos observar que o melhor modelo foi o documental, o MongoDB, em seguida o modelo relacional, o Oracle, e por último o modelo orientado a gráfos, o Neo4j. Podemos relatar que o modelo documental comportou-se melhor uma vez que não existem profissões iguais em departamentos diferentes, devido a estrutura departamental adotada nesta base de dados e a simplicidade da *query* foi possível obter os melhores resultados. No que diz respeito ao Oracle, o seu desempenho está dependente das consultas as tabelas *employees* e *jobs*, apesar de ser uma consulta direta os agrupamentos contribuem para um maior tempo de execução. No Neo4j, uma vez que teremos de iterar por todos os *employees* e seus correspondentes salários e apenas após isso poderemos agrupar os salários médios de cada profissão a *query* torna-se mais custosa comparativamente aos outros dois modelos.

Medição	M1	M2	M3	M4	M5	**Média (ms)**
Oracle	9	7	6	8	9	**7.8**
MongoDB	9	8	4	4	5	**6**
Neo4J	3	12	7	12	12	**9.2**

Tabela 1.11: Tempo de execução para a *query* 11

6 Análise Crítica e Comparação de Modelos

Após implementar as *queries* nas diferentes BDs podemos agora analisar os resultados obtidos, bem como efetuar uma análise crítica dos mesmos.

Primeiramente, podemos observar que o número de *queries* implementadas é relevante, tendo sido selecionadas 11 *queries* com variados propósitos. Desde analisar salários, melhores funcionários, questões geográficas, históricos dos trabalhadores, entre outras das mais diversificadas áreas de estudo com interesse para análises de Recursos Humanos da empresa em questão.

Em adição, podemos ainda realçar a existência de complexidades variáveis na implementação das diferentes *queries* nos modelos não relacionais (MongoDB e Neo4j) e relacional (Oracle). Tal como pode ser evidenciado pela implementação das *queries* na secção 5, a implementação em MongoDB exigiu uma complexidade relativamente superior quando comparada com a implementação em Oracle, que por sua vez também tem uma complexidade relativa superior as *queries* implementadas em Neo4j. Isto deve-se ao facto do MongoDB usar uma *query language* mais complexa, de estarmos mais familiarizados com o uso de SQL e da facilidade de implementação de *queries* em Neo4j através da linguagem Cypher.

Tal como fora referido anteriormente, um fator importante para o sucesso deste trabalho foi a escolha das arquiteturas para cada modelo. No caso da base de dados documental *MongoDB*, tiramos partido de sintetizar a informação sobre cada departamento num ficheiro JSON. Desta forma, conseguimos ter um conjunto reduzido de ficheiros com a informação representativa do esquema implementado pelo *Oracle*. Comparativamente ao Oracle, a estrutura de dados implementada em MongoDB evita que haja acesso a dados externos quer estes estejam em tabelas quer estejam em documentos, permitindo a redução do tempo de execução. Foi tido em consideração na implementação do modelo MongoDB a redução de conjuntos de dados *nested*, tentando obter um documento simples, de fácil acesso e procura.

Para o caso do *Neo4j*, a estrutura de dados foi implementada de forma a tirar partido da utilização de relações entre os diferentes nodos do grafo, criando assim um estrutura que seja representativa do esquema que estava a ser implementado no *Oracle*. Comparativamente com o modelo do *Oracle*, o modelo do *Neo4j* implementado é benéfico pois, evita a utilização de JOINs custosos no desenvolvimento das *queries*, bastando apenas referir as relações entre nodos que queremos obter em cada *query*. Desta forma, tal como podemos verificar na análise dos resultados da secção 5, para a maioria das *queries* desenvolvidas no *Neo4j* os valores do tempo de execução tendem a ser menores quando temos informações diretamente obtidas através das relações entre os nodos.

Desta forma, elaboraram-se, com sucesso, duas base de dados não relacionais, uma

documental e outra orientada a grafos, tendo-se tirado o melhor partido destes dois paradigmas. Além disso, apesar de existirem algumas diferenças nos tempos de execução *MongoDB* e *Neo4J*, estas não são muito significativas, pelo que permite concluir que, na generalidade, os modelos de bases de dados não relacionais apresentam melhor performance do que os modelos relacionais (neste caso, o *Oracle*). No entanto, numa perspetiva mais detalhada, é possível observar que a BD *Neo4j* é a que possui melhor performance a nível geral, revelando-se a mais indicada para o caso de estudo em questão.

7 Conclusão

Dado por concluído o trabalho prático faz sentido apresentar uma visão crítica, refletida e ponderada do trabalho realizado.

No espetro positivo, consideramos relevante destacar o facto de termos implementado estruturas para as base dados *MongoDB* e *Neo4j* que maximizam os paradigmas de cada uma. Além disso, é importante realçar que o conjunto de *queries* implementadas é completo e devolve informação pertinente sobre a empresa, além de conseguir avaliar de forma consistente as várias estruturas implementadas. Finalmente, com o presente documento é possível ter uma visão completa e representativa das razões e dos aspetos mais importantes para as implementações de cada BD.

Por outro lado, em termos de trabalho futuro ou melhorias no nosso projeto, consideramos que seria benéfico adicionar mais dados à BD e efetuar mais testes de performance, de maneira a analisar o comportamento das várias estruturas desenvolvidas com elevados volumes de dados. Por outro lado, seria interessante utilizar outras métricas de avaliação tais como o CPU ou RAM gastos durante a execução de cada *query*.

Em suma, consideramos que o balanço do trabalho é positivo, as dificuldades sentidas foram superadas e os requisitos propostos foram cumpridos.

Bibliografia

[1] "The Neo4j Graph Platform"[online]. Disponível em: https://neo4j.com/ [Acedido em junho de 2022].

[2] "Neo4j Aggregating functions"[online]. Disponível em: https://neo4j.com/docs/cypher-manual/current/functions/aggregating/ [Acedido em junho de 2022].

[3] "Group and count relationships in cypher neo4j"[online]. Disponível em: https://stackoverflow.com/questions/27536615/how-to-group-and-count-relationships-in-cypher-neo4j [Acedido em junho de 2022].

[4] "MongoDB - Query an Array"[online]. Disponível em: https://www.mongodb.com/docs/manual/tutorial/query-arrays/ [Acedido em junho de 2022].

[5] "MongoDB - dateDiff (aggregation)"[online]. Disponível em: https://www.mongodb.com/docs/manual/reference/operator/aggregation/dateDiff/ [Acedido em junho de 2022].

[6] "Calculate date difference in year, month, day"[online]. Disponível em: https://stackoverflow.com/questions/47236343/calculate-date-difference-in-year-month-day [Acedido em junho de 2022].

[7] "How to analyze MongoDB RAM usage"[online]. Disponível em: https://www.mongodb.com/community/forums/t/how-to-analyze-mongodb-ram-usage/12108/8 [Acedido em junho de 2022].

[8] "MongoDB - Analyze Query Performance"[online]. Disponível em: https://www.mongodb.com/docs/manual/tutorial/analyze-query-plan/ [Acedido em junho de 2022].

[9] "Conditional Cypher Execution"[online]. Disponível em: https://neo4j.com/labs/apoc/4.1/cypher-execution/conditionals/ [Acedido em junho de 2022].

[10] "Database SQL Reference - CURRENTTIMESTAMP"[online]. Disponível em: https://docs.oracle.com/cd/B19306$_0$1/$server$.102/b14200/$functions$037.htm

[Acedido em junho de 2022].

8 Anexos

8.1 *Query* 1: Listar o número de departamentos em cada país

	COUNTRY_NAME	NR_DEPARTMENTS
1	United States of America	23
2	United Kingdom	2
3	Canada	1
4	Germany	1

Figura 1.6: Resultado da query 1 em Oracle

```
{ "_id" : "United States of America", "count" : 23 }
{ "_id" : "United Kingdom", "count" : 2 }
{ "_id" : "Canada", "count" : 1 }
{ "_id" : "Germany", "count" : 1 }
```

Figura 1.7: Resultado da query 1 em MongoDB

Country	TOTAL
"United States of America"	23
"United Kingdom"	2
"Germany"	1
"Canada"	1

Figura 1.8: Resultado da query 1 em Neo4j

8.2 *Query* 2: Listar o número de *employees* em cada país

	COUNTRY_NAME	NR_EMPLOYEES
1	United States of America	68
2	United Kingdom	35
3	Canada	2
4	Germany	1

Figura 1.9: Resultado da query 2 em Oracle

```
{ "_id" : "United States of America", "count" : 68 }
{ "_id" : "United Kingdom", "count" : 35 }
{ "_id" : "Canada", "count" : 2 }
{ "_id" : "Germany", "count" : 1 }
```

Figura 1.10: Resultado da query 2 em MongoDB

Country	TOTAL
"United States of America"	68
"United Kingdom"	35
"Canada"	2
"Germany"	1

Figura 1.11: Resultado da query 2 em Neo4j

8.3 *Query* 3: Listar o *employees* que mais ganha na empresa

	FIRST_NAME	LAST_NAME	SALARY	JOB_TITLE
1	Steven	King	24000.00	President

Figura 1.12: Resultado da query 3 em Oracle

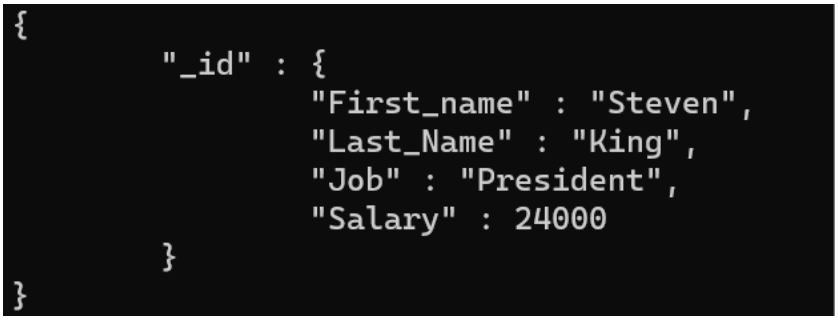


Figura 1.13: Resultado da query 3 em MongoDB

	FIRST_NAME	LAST_NAME	SALARY	JOB_TITLE
1	"Steven"	"King"	24000.0	"President"

Figura 1.14: Resultado da query 3 em Neo4j

8.4 *Query* 4: Listar o *employees* que menos ganha na empresa

	FIRST_NAME	LAST_NAME	SALARY	JOB_TITLE
1	TJ	Olson	2100.00	Stock Clerk

Figura 1.15: Resultado da query 4 em Oracle

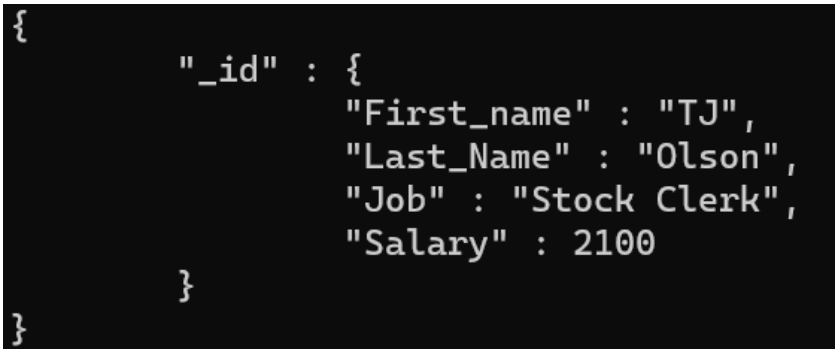


Figura 1.16: Resultado da query 4 em MongoDB

	FIRST_NAME	LAST_NAME	SALARY	JOB_TITLE
1	"TJ"	"Olson"	2100.0	"Stock Clerk"

Figura 1.17: Resultado da query 4 em Neo4j

8.5 *Query* 5: Listar o *employee* que está a trabalhar há mais tempo

	FIRST_NAME	LAST_NAME	HIRE_DATE	NR_YEARS_WORKING
1	Lex	De Haan	2001-01-13	21

Figura 1.18: Resultado da query 5 em Oracle


```
{
        "_id" : {
                "First_name" : "Lex",
                "Last_Name" : "De Haan",
                "Job" : "Administration Vice President",
                "Hire Date" : "2001-01-13 00:00:00"
        },
        "YearsWorking" : 21
}
```


Figura 1.19: Resultado da query 5 em MongoDB

FIRST_NAME	LAST_NAME	DATE_HIRED	NR_WORKING
"Lex"	"De Haan"	"2001-01-13"	21

Figura 1.20: Resultado da query 5 em Neo4j

8.6 *Query* 6: Listar o *employee* que está a trabalhar há menos tempo

	FIRST_NAME	LAST_NAME	HIRE_DATE	NR_YEARS_WORKING
1	Amit	Banda	2008-04-21	14
2	Sundita	Kumar	2008-04-21	14

Figura 1.21: Resultado da query 6 em Oracle


```
{
        "_id" : {
                "First_name" : "Amit",
                "Last_Name" : "Banda",
                "Job" : "Sales Representative",
                "Hire Date" : "2008-04-21 00:00:00"
        },
        "YearsWorking" : 14
}
{
        "_id" : {
                "First_name" : "Sundita",
                "Last_Name" : "Kumar",
                "Job" : "Sales Representative",
                "Hire Date" : "2008-04-21 00:00:00"
        },
        "YearsWorking" : 14
}
```


Figura 1.22: Resultado da query 6 em MongoDB

	FIRST_NAME	LAST_NAME	DATE_HIRED	NR_WORKING
1	"Amit"	"Banda"	"2008-04-21"	14
2	"Sundita"	"Kumar"	"2008-04-21"	14

Figura 1.23: Resultado da query 6 em Neo4j

8.7 *Query* 7: Ordenar as várias profissões pelo número de *employees*

	JOB_TITLE	NR_EMPLOYEES
1	Sales Representative	30
2	Stock Clerk	20
3	Shipping Clerk	20
4	Programmer	5
5	Purchasing Clerk	5
6	Sales Manager	5
7	Accountant	5
8	Stock Manager	5
9	Administration Vice President	2
10	Administration Assistant	1
11	Finance Manager	1
12	Purchasing Manager	1
13	Human Resources Representative	1
14	Public Relations Representative	1
15	Accounting Manager	1
16	Marketing Representative	1
17	Public Accountant	1
18	President	1
19	Marketing Manager	1

Figura 1.24: Resultado da query 7 em Oracle


```
{ "_id" : "Sales Representative", "count" : 30 }
{ "_id" : "Stock Clerk", "count" : 20 }
{ "_id" : "Shipping Clerk", "count" : 20 }
{ "_id" : "Purchasing Clerk", "count" : 5 }
{ "_id" : "Sales Manager", "count" : 5 }
{ "_id" : "Stock Manager", "count" : 5 }
{ "_id" : "Programmer", "count" : 5 }
{ "_id" : "Accountant", "count" : 5 }
{ "_id" : "Administration Vice President", "count" : 2 }
{ "_id" : "Public Relations Representative", "count" : 1 }
{ "_id" : "President", "count" : 1 }
{ "_id" : "Finance Manager", "count" : 1 }
{ "_id" : "Purchasing Manager", "count" : 1 }
{ "_id" : "Administration Assistant", "count" : 1 }
{ "_id" : "Marketing Representative", "count" : 1 }
{ "_id" : "Accounting Manager", "count" : 1 }
{ "_id" : "Public Accountant", "count" : 1 }
{ "_id" : "Human Resources Representative", "count" : 1 }
{ "_id" : "Marketing Manager", "count" : 1 }
```


Figura 1.25: Resultado da query 7 em MongoDB

	Job	TOTAL
1	"Sales Representative"	30
2	"Stock Clerk"	20
3	"Shipping Clerk"	20
4	"Accountant"	5
5	"Sales Manager"	5
6	"Purchasing Clerk"	5
7	"Stock Manager"	5
8	"Programmer"	5
9	"Administration Vice President"	2

Figura 1.26: Resultado Parcial da query 7 em Neo4j

8.8 *Query* 8: Ordenar cada departamento pelo número de *employees* que lá trabalham

	DEPARTMENT_NAME	NR_EMPREGADOS
1	Shipping	45
2	Sales	34
3	Finance	6
4	Purchasing	6
5	IT	5
6	Executive	3
7	Marketing	2
8	Accounting	2
9	Public Relations	1
10	Administration	1
11	Human Resources	1
12	Control And Credit	0
13	Shareholder Services	0
14	IT Helpdesk	0
15	Operations	0
16	Payroll	0
17	Recruiting	0
18	Retail Sales	0
19	NOC	0
20	Contracting	0
21	Corporate Tax	0
22	Benefits	0
23	Government Sales	0
24	Construction	0
25	Manufacturing	0
26	IT Support	0
27	Treasury	0

Figura 1.27: Resultado da query 8 em Oracle

```
{ "_id" : "Shipping", "numberEmployees" : 45 }
{ "_id" : "Sales", "numberEmployees" : 34 }
{ "_id" : "Finance", "numberEmployees" : 6 }
{ "_id" : "Purchasing", "numberEmployees" : 6 }
{ "_id" : "IT", "numberEmployees" : 5 }
{ "_id" : "Executive", "numberEmployees" : 3 }
{ "_id" : "Marketing", "numberEmployees" : 2 }
{ "_id" : "Accounting", "numberEmployees" : 2 }
{ "_id" : "Public Relations", "numberEmployees" : 1 }
{ "_id" : "Administration", "numberEmployees" : 1 }
{ "_id" : "Human Resources", "numberEmployees" : 1 }
{ "_id" : "Construction", "numberEmployees" : 0 }
{ "_id" : "Contracting", "numberEmployees" : 0 }
{ "_id" : "Operations", "numberEmployees" : 0 }
{ "_id" : "Control And Credit", "numberEmployees" : 0 }
{ "_id" : "Benefits", "numberEmployees" : 0 }
{ "_id" : "Corporate Tax", "numberEmployees" : 0 }
{ "_id" : "IT Support", "numberEmployees" : 0 }
{ "_id" : "Treasury", "numberEmployees" : 0 }
{ "_id" : "IT Helpdesk", "numberEmployees" : 0 }
Type "it" for more
```

Figura 1.28: Resultado da query 8 em MongoDB

	DEPARTMENT_NAME	NR_EMPLOYEES
1	"Shipping"	45
2	"Sales"	34
3	"Purchasing"	6
4	"Finance"	6
5	"IT"	5
6	"Executive"	3
7	"Marketing"	2

Figura 1.29: Resultado Parcial da query 8 em Neo4j

8.9 *Query* 9: Listar o salário médio por departamento

	DEPARTAMENTO	MEDIA
1	Shipping	3475.5555555555555555555555555555555556
2	Purchasing	4150
3	Administration	4400
4	IT	5760
5	Human Resources	6500
6	Finance	8601.3333333333333333333333333333333333
7	Sales	8955.882352941176470588235294117647058824
8	Marketing	9500
9	Public Relations	10000
10	Accounting	10154
11	Executive	19333.333333333333333333333333333333333

Figura 1.30: Resultado da query 9 em Oracle

```
{ "Departamento" : "Shipping", "Media" : 3475.5555555555557 }
{ "Departamento" : "Purchasing", "Media" : 4150 }
{ "Departamento" : "Administration", "Media" : 4400 }
{ "Departamento" : "IT", "Media" : 5760 }
{ "Departamento" : "Human Resources", "Media" : 6500 }
{ "Departamento" : "Finance", "Media" : 8601.333333333334 }
{ "Departamento" : "Sales", "Media" : 8955.882352941177 }
{ "Departamento" : "Marketing", "Media" : 9500 }
{ "Departamento" : "Public Relations", "Media" : 10000 }
{ "Departamento" : "Accounting", "Media" : 10154 }
{ "Departamento" : "Executive", "Media" : 19333.333333333332 }
```

Figura 1.31: Resultado da query 9 em MongoDB

	Department	Salary
	"Shipping"	3475.555555555555
2	"Purchasing"	4150.0
3	"Administration"	4400.0
4	"IT"	5760.0
5	"Human Resources"	6500.0
6	"Finance"	8601.333333333334
7	"Sales"	8955.882352941177

Figura 1.32: Resultado parcial da query 9 em Neo4j

8.10 *Query* 10: Listar o histórico de cada *employee*

	EMPLOYEE_FIRST	EMPLOYEE_LAST	START_DATE	END_DATE	JOB	DEPARTMENT
1	Neena	Kochhar	1997-09-21	2001-10-27	Public Accountant	Accounting
2	Neena	Kochhar	2001-10-28	2005-03-15	Accounting Manager	Accounting
3	Lex	De Haan	2001-01-13	2006-07-24	Programmer	IT
4	Den	Raphaely	2006-03-24	2007-12-31	Stock Clerk	Shipping
5	Payam	Kaufling	2007-01-01	2007-12-31	Stock Clerk	Shipping
6	Jonathon	Taylor	2007-01-01	2007-12-31	Sales Manager	Sales
7	Jonathon	Taylor	2006-03-24	2006-12-31	Sales Representative	Sales
8	Jennifer	Whalen	2002-07-01	2006-12-31	Public Accountant	Executive
9	Jennifer	Whalen	1995-09-17	2001-06-17	Administration Assistant	Executive
10	Michael	Hartstein	2004-02-17	2007-12-19	Marketing Representative	Marketing

Figura 1.33: Resultado da query 10 em Oracle


```
{
        "First Name" : "Payam",
        "Last Name" : "Kaufling",
        "Department where the employee worked" : "Shipping",
        "Past Jobs" : [
                {
                        "Start Date" : "2007-01-01 00:00:00",
                        "End Date" : "2007-12-31 00:00:00",
                        "Job" : {
                                "Title" : "Stock Clerk",
                                "Minimum Salary" : 2008,
                                "Maximum Salary" : 5000
                        }
                }
        ]
}
{
        "First Name" : "Lex",
        "Last Name" : "De Haan",
        "Department where the employee worked" : "IT",
        "Past Jobs" : [
                {
                        "Start Date" : "2001-01-13 00:00:00",
                        "End Date" : "2006-07-24 00:00:00",
                        "Job" : {
                                "Title" : "Programmer",
                                "Minimum Salary" : 4000,
                                "Maximum Salary" : 10000
                        }
                }
```


Figura 1.34: Resultado parcial da query 10 em MongoDB

	Employee_Name	Employee_Surname	Start_Date	End_Date	Job	Department
1	"Michael"	"Hartstein"	"2004-02-17"	"2007-12-19"	"Marketing Representative"	"Marketing"
2	"Payam"	"Kaufling"	"2007-01-01"	"2007-12-31"	"Stock Clerk"	"Shipping"
3	"Den"	"Raphaely"	"2006-03-24"	"2007-12-31"	"Stock Clerk"	"Shipping"
4	"Lex"	"De Haan"	"2001-01-13"	"2006-07-24"	"Programmer"	"IT"
5	"Jonathon"	"Taylor"	"2007-01-01"	"2007-12-31"	"Sales Manager"	"Sales"
6	"Jonathon"	"Taylor"	"2006-03-24"	"2006-12-31"	"Sales Representative"	"Sales"
7						

Figura 1.35: Resultado parcial da query 10 em Neo4j

8.11 Query 11: Listar o salário médio por profissão

	PROFISSÃO	MEDIA
1	Purchasing Clerk	2780
2	Stock Clerk	2785
3	Shipping Clerk	3215
4	Administration Assistant	4400
5	Programmer	5760
6	Marketing Representative	6000
7	Human Resources Representative	6500
8	Stock Manager	7280
9	Accountant	7920
10	Public Accountant	8300
11	Sales Representative	8350
12	Public Relations Representative	10000
13	Purchasing Manager	11000
14	Finance Manager	12008
15	Accounting Manager	12008
16	Sales Manager	12200
17	Marketing Manager	13000
18	Administration Vice President	17000
19	President	24000

Figura 1.36: Resultado da query 11 em Oracle


```
{ "_id" : ObjectId("629c9d5ab80cb5907ee82603"), "Profissao" : "Purchasing Clerk", "Media" : 2780 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82605"), "Profissao" : "Stock Clerk", "Media" : 2785 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82605"), "Profissao" : "Shipping Clerk", "Media" : 3215 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82601"), "Profissao" : "Administration Assistant", "Media" : 4400 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82606"), "Profissao" : "Programmer", "Media" : 5760 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82602"), "Profissao" : "Marketing Representative", "Media" : 6000 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82604"), "Profissao" : "Human Resources Representative", "Media" : 6500 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82605"), "Profissao" : "Stock Manager", "Media" : 7280 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee8260a"), "Profissao" : "Accountant", "Media" : 7920 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee8260b"), "Profissao" : "Public Accountant", "Media" : 8300 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82608"), "Profissao" : "Sales Representative", "Media" : 8396.551724137931 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82607"), "Profissao" : "Public Relations Representative", "Media" : 10000 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82603"), "Profissao" : "Purchasing Manager", "Media" : 11000 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee8260a"), "Profissao" : "Finance Manager", "Media" : 12008 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee8260b"), "Profissao" : "Accounting Manager", "Media" : 12008 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82608"), "Profissao" : "Sales Manager", "Media" : 12200 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82602"), "Profissao" : "Marketing Manager", "Media" : 13000 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82609"), "Profissao" : "Administration Vice President", "Media" : 17000 }
{ "_id" : ObjectId("629c9d5ab80cb5907ee82609"), "Profissao" : "President", "Media" : 24000 }
```


Figura 1.37: Resultado da query 11 em MongoDB

	Job	Salary
1	"Purchasing Clerk"	2780.0
2	"Stock Clerk"	2784.9999999999995
3	"Shipping Clerk"	3214.9999999999995
4	"Administration Assistant"	4400.0
5	"Programmer"	5760.0
6	"Marketing Representative"	6000.0
7		

Figura 1.38: Resultado Parcial da query 11 em Neo4j