Universidade do Minho
Mestrado em Engenharia Informática

Unidade Curricular de
Bases de Dados NoSQL

Ano Letivo de 2020/2021

Trabalho Prático

Diogo Alexandre Rodrigues Lopes (PG42823)

Joel Costa Carvalho (PG42837)

Ana Margarida da Rocha Ferreira (PG44412)

Bruno Xavier Brás dos Santos (PG44414)

Grupo 7

Data de Recepção	
Responsável	
Avaliação	
Observações	

Trabalho Prático

Diogo Alexandre Rodrigues Lopes (PG42823)

Joel Costa Carvalho (PG42837)

Ana Margarida da Rocha Ferreira (PG44412)

Bruno Xavier Brás dos Santos (PG44414)

Grupo 7

Janeiro, 2021

Resumo

Com o surgimento do *Big Data* emergiram novos e rebuscados desafios relativamente ao seu armazenamento, tratamento e manipulação. Os sistemas de bases de dados relacionais não se apresentam como a solução indicada para este problema. Deste modo, foram desenvolvidas várias tecnologias para combater estes problemas, entre as quais se salienta o modelo de base de dados NoSQL. Este novo paradigma de armazenamento visa responder às necessidades de armazenamento e processamento de dados volumosos e heterogéneos. Assim, devido às mudanças repentinas dos requisitos do *software*, tornou-se necessário efetuar a migração dos dados de vários sistemas de bases de dados para outro sistema de base de dados não relacional, que permitisse colmatar as novas necessidades de armazenamento e processamento.

O presente trabalho prático foi realizado no âmbito da unidade curricular de Bases de Dados NoSQL e consistiu na análise, planeamento e implementação de um sistema de gestão de base de dados relacional e dois não relacionais. Para tal, adaptou-se a base de dados no modelo relacional HR, disponibilizada no Oracle, para dois modelos não relacionais, sendo uma das bases de dados NoSQL utilizadas orientada a documentos, a MongoDB, e a outra orientada a grafos, a Neo4j.

Área de Aplicação: Planeamento e implementação de Sistemas de Gestão de Bases de Dados relacionais e não relacionais.

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

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

O presente trabalho prático foi realizado no âmbito da unidade curricular de Bases de Dados NoSQL e consistiu na análise, planeamento e implementação de um sistema de gestão de base de dados relacional e de dois não relacionais. Nesta secção é feito o enquadramento do tema, a apresentação da motivação e objetivos, terminando com uma visão sobre a estrutura deste documento.

1.1. Contextualização

Com a elevada expansão da Internet, o número de utilizadores tem vindo a aumentar constantemente, e como tal, levou também a um crescimento do número de páginas da Internet e do número de utilizadores das redes sociais. Com um aumento tão drástico, quer os motores de busca, quer as redes sociais, tiveram de lidar com quantidades enormes de dados a processar, alcançando a ordem dos *Petabytes*. As bases de dados tradicionais, isto é, do modelo relacional não foram concebidas para lidar com um enorme volume de dados em pouco tempo, só sendo possível o aumento da sua capacidade de processamento através de reconfigurações. Assim, com vista a encontrar uma alternativa que conseguisse resolver estes problemas, foram desenvolvidas novas tecnologias, nomeadamente as bases de dados NoSQL.

Assim, dadas as mudanças repentinas dos requisitos de *software* dos sistemas de bases de dados, a possibilidade da migração de dados entre diferentes paradigmas de bases de dados é uma mais-valia.

1.2. Motivação e Objetivos

Para se efetuar a migração de dados entre diferentes paradigmas de base de dados, isto é, do modelo relacional para um modelo não relacional, é necessário analisar o esquema de dados e planear como os dados vão passar a ser armazenados nos sistemas de base de dados NoSQL.

O presente trabalho tem como finalidade a análise, planeamento e implementação de um sistema de gestão de base de dados relacional e de dois não relacionais, sendo um orientado a documentos e outro orientado a grafos, através da utilização da base de dados relacional HR, disponibilizada no Oracle.

Para a realização do projeto, inicialmente, foi necessária a seleção dos modelos de base de dados não relacionais a utilizar, tendo-se escolhido a MongoDB para base de dados orientada a documentos e a Neo4j para a base de dados orientada a grafos. De seguida, foi necessário planear o procedimento para migrar os dados fornecidos para os novos sistemas não relacionais. Após a migração dos dados, para demonstrar a operacionalidade dos sistemas implementados, definiu-se e implementou-se um conjunto

de *queries.* Por fim, é feita uma análise do trabalho, contemplando as vantagens e desvantagens de cada um dos SDB implementados.

1.3. Estrutura do Relatório

O presente relatório é constituído por sete secções. Na primeira secção, **Introdução**, é feito um enquadramento do tema do trabalho e enumeração dos objetivos do mesmo.

De seguida, na secção denominada **Metodologia** encontra-se uma explicação do procedimento seguido para a concretização deste projeto.

Posteriormente, nas secções **Base de Dados Relacional - Oracle**, **Base de Dados Orientada a Documentos - MongoDB** e **Base de Dados Orientada a Grafos - Neo4j**, encontram-se descritas as diferentes estratégias adotadas no desenvolvimento de cada um dos modelos, explicando o raciocínio aplicado na migração dos dados e apresentando o resultado de cada *query* selecionada.

Na secção **Discussão dos Resultados** é feita uma comparação entre os diferentes sistemas, apresentando as suas vantagens e desvantagens, e a respetiva implementação das *queries*.

Por fim, na **Conclusão**, é apresentada uma síntese e análise do trabalho realizado.

2. Metodologia

O conjunto de dados utilizado para a realização do presente projeto é denominado HR e encontra-se disponibilizado na base de dados Oracle. Este é um esquema de uma aplicação de Recursos Humanos, fornecido pela Oracle, que tem como objetivo principal armazenar os dados dos empregados de uma organização.

O HR possui 7 tabelas:

- **EMPLOYEES**: Dados dos empregados, tais como: nome, email, número de telefone, cargo atual, data de início do contrato, departamento e gerente.
- **DEPARTMENTS**: Dados dos departamentos em que empregados podem trabalhar, tais como: nome do departamento, gerente e localização.
- **REGIONS**: Dados sobre as regiões em que a organização pode atuar.
- **LOCATIONS**: Dados sobre os locais ou endereços dos escritórios, armazéns ou locais de produção da organização, tais como endereço, código postal, cidade, província e país.
- **COUNTRIES**: Dados sobre os países em que a organização atua, numa dada região.
- **JOBS**: Dados sobre os tipos de cargos que os empregados podem ocupar, tais como o título do cargo e salário mínimo e salário máximo possíveis.
- **JOB_HISTORY**: Histórico dos cargos anteriores ocupados pelos empregados dentro da organização.

Para realizar o presente projeto começou-se por selecionar as bases de dados a utilizar, para além da relacional Oracle. Para a base de dados orientada a documentos e para a orientada a grafos optou-se por utilizar a MongoDB e a Neo4j, respetivamente, uma vez que se adequam ao problema e o grupo encontra-se bastante familiarizado com as mesmas, devido à realização das fichas propostas.

De seguida, efetuou-se a análise do esquema relacional da base de dados HR fornecida e planeou-se o modelo de dados para a migração para os novos sistemas não relacionais, de modo a maximizar cada um dos seus paradigmas.

Por último, tendo em conta que um dos principais objetivos deste trabalho prático é demonstrar a operacionalidade dos sistemas de bases de dados não relacionais, através da implementação de um conjunto de *queries*, desenvolveu-se oito *queries* no modelo relacional e comprovou-se a solidez dos dois modelos NoSQL produzidos. Cada um dos SDB selecionados, nomeadamente, Oracle, MongoDB e Neo4j, possui diferentes características e finalidades, no entanto, nas *queries* implementadas, presentes nas secções 3.2., 4.3. e 5.3., teve-se o cuidado de retornar sempre o mesmo tipo e listagem de dados.

3. Base de Dados Relacional – Oracle

3.1. Modelo de Dados

O Oracle trata-se de um SDB segundo o modelo relacional. Neste modelo os dados são armazenados em tabelas, contendo colunas e linhas, que representam registos. As informações sobre uma entidade podem ser distribuídas por várias tabelas e é possível associar dados de diferentes tabelas através do estabelecimento do relacionamento entre tabelas pela utilização de chave estrangeiras. Além disso, nas bases de dados relacionais é necessária a criação de um esquema lógico antes de se inserir qualquer dado.

Para a base de dados em estudo, temos o seguinte esquema lógico.

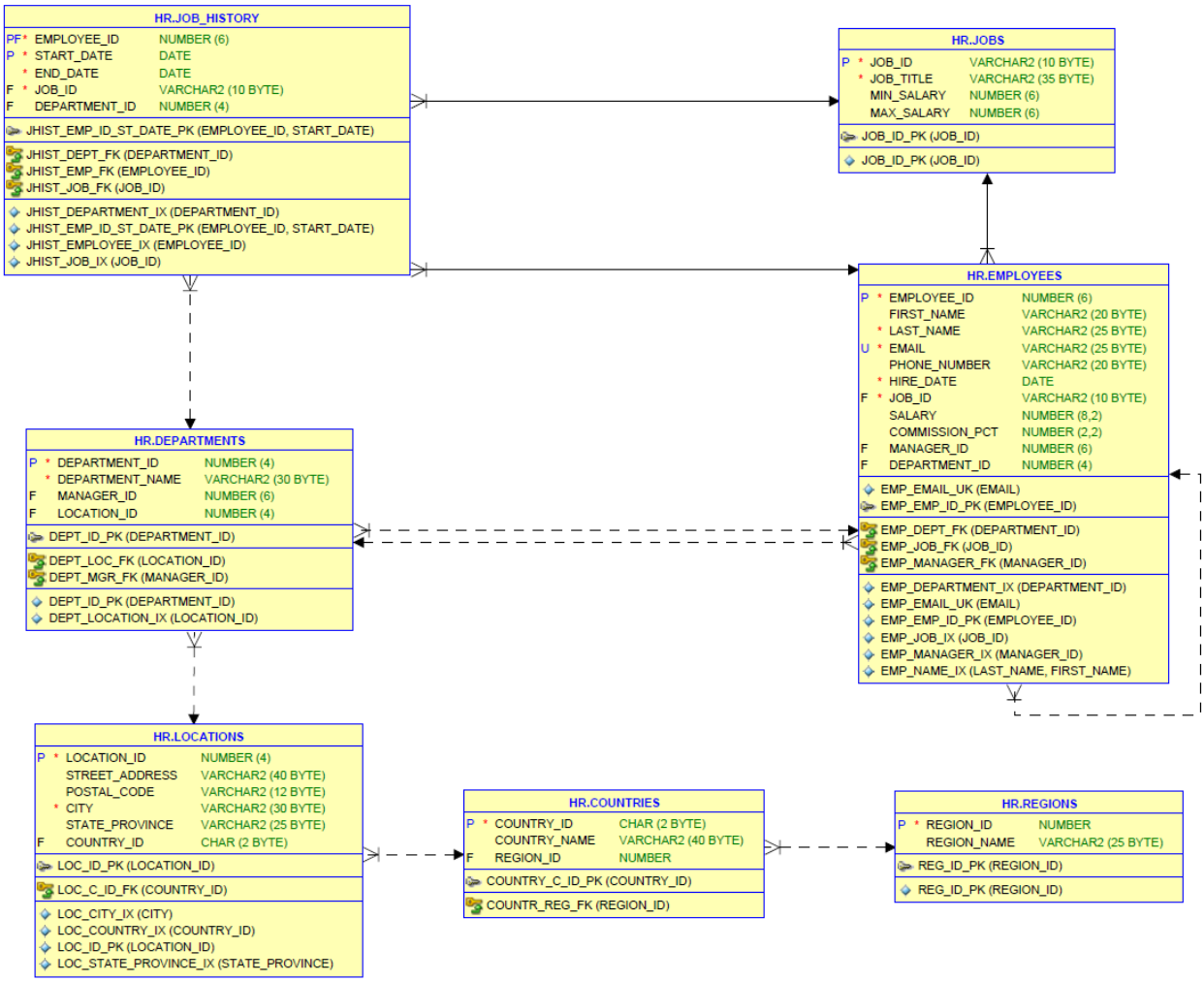

Figura 1. Esquema lógico da base de dados HR.

3.2. *Queries*

3.2.1 *Query* 1 – Lista de Cidades por Departamento

A *query* seguinte retorna uma listagem de cidades por departamento. No total existem 7 cidades distintas por departamento.

```
/*Query 1 - Lista de Cidades dos Departamentos*/
select distinct city
from departments
        join locations on locations.location_id = departments.location_id;
```

Figura 2. Comando SQL da query 1.

CITY
London
Seattle
Munich
South San Francisco
Toronto
Southlake
Oxford

Figura 3. Resultado da query 1 na base de dados Oracle.

3.2.2 *Query* 2 – Lista de Empregados com Mudanças de Emprego

A *query* seguinte retorna a listagem de empregados que tiveram mudanças de emprego. No total existem 7 empregados que mudaram de emprego, sendo que a "Neena", o "Jonathon" e a "Jennifer" mudaram 2 vezes de cargo e os restantes mudaram apenas uma vez.

```
/*Query 2 - Lista de Empregados com mudanças de Empregos*/
select employees.first_name, count(*) as ocorrencias
from job_history
        join employees on employees.employee_id = job_history.employee_id
group by employees.first_name
order by ocorrencias desc;
```

Figura 4. Comando SQL da query 2.

FIRST_NAME	OCORRENCIAS
Neena	2
Jonathon	2
Jennifer	2
Den	1
Payam	1
Michael	1
Lex	1

Figura 5. Resultado da query 2 na base de dados Oracle.

3.2.3 *Query* 3 – Lista Geral do Conteúdo da Base de Dados

A *query* seguinte retorna a listagem de todos os dados relacionados com todos os empregados. Dada a extensão do conteúdo da base de dados, o resultado desta *query* encontra-se na pasta de submissão do trabalho.

```
/*Query 3 - Conteudo Geral da BD*/
select *
from employees
        join departments on departments.department_id = employees.department_id
        join job_history on job_history.job_id = employees.job_id
        join jobs on jobs.job_id = employees.job_id
        join locations on locations.location_id = departments.location_id
        join countries on countries.country_id = locations.country_id
        join regions on regions.region_id = countries.region_id;
```

Figura 6. Comando SQL da query 3.

3.2.4 *Query* 4 – Lista de Empregados Cujo Nome Inicia com a Letra "J"

A *query* seguinte retorna uma listagem com informação referente a cada empregado, cujo nome começa com a letra "J".

```
/*Query 4 - Lista de Empregados cujo Nome inicia com a letra J*/
select *
from employees
where first_name like 'J%'
```

Figura 7. Comando SQL da query 4.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
110	John	Chen	JCHEN	515.124.4269	05.09.28	FI_ACCOUNT	8200	(null)	108	100
186	Julia	Dellinger	JDELLING	650.509.3876	06.06.24	SH_CLERK	3400	(null)	121	50
189	Jennifer	Dilly	JDILLY	650.505.2876	05.08.13	SH_CLERK	3600	(null)	122	50
181	Jean	Fleaur	JFLEAUR	650.507.9877	06.02.23	SH_CLERK	3100	(null)	120	50
156	Janette	King	JKING	011.44.1345.429268	04.01.30	SA_REP	10000	0,35	146	80
127	James	Landry	JLANDRY	650.124.1334	07.01.14	ST_CLERK	2400	(null)	120	50
177	Jack	Livingston	JLIVINGS	011.44.1644.429264	06.04.23	SA_REP	8400	0,2	149	80
133	Jason	Mallin	JMALLIN	650.127.1934	04.06.14	ST_CLERK	3300	(null)	122	50
131	James	Marlow	JAMRLOW	650.124.7234	05.02.16	ST_CLERK	2500	(null)	121	50
125	Julia	Nayer	JNAYER	650.124.1214	05.07.16	ST_CLERK	3200	(null)	120	50
140	Joshua	Patel	JPATEL	650.121.1834	06.04.06	ST_CLERK	2500	(null)	123	50
145	John	Russell	JRUSSEL	011.44.1344.429268	04.10.01	SA_MAN	14000	0,4	100	80
139	John	Seo	JSEO	650.121.2019	06.02.12	ST_CLERK	2700	(null)	123	50
176	Jonathon	Taylor	JTAYLOR	011.44.1644.429265	06.03.24	SA_REP	8600	0,2	149	80
112	Jose Manuel	Urman	JMURMAN	515.124.4469	06.03.07	FI_ACCOUNT	7800	(null)	108	100
200	Jennifer	Whalen	JWHALEN	515.123.4444	03.09.17	AD_ASST	4400	(null)	101	10

Figura 8. Resultado da query 4 na base de dados Oracle.

3.2.5 *Query* 5 – Lista de Todos os Presidentes

A *query* seguinte retorna uma listagem com informação, nomeadamente o nome, a data de contratação e o respetivo salário, de todos Presidentes. Neste caso em concreto, apenas o "Steven" tem o cargo de Presidente.

```
/*Query 5 - Listagem de todos os Presidentes*/
select employees.first_name as "Name",
            employees.hire_date as "HireDate",
            employees.salary as "Salary"
from employees
    join jobs on jobs.job_id = employees.job_id
where jobs.job_title = 'President'
```

Figura 9. Comando SQL da query 5.

Name	HireDate	Salary
Steven	03.06.17	24000

Figura 10. Resultado da query 5 na base de dados Oracle.

3.2.6 *Query* 6 – *Top* 5 dos Funcionários Mais Bem Pagos

A *query* seguinte retorna a informação relativamente ao *top* 5 dos funcionários mais bem pagos. A pessoa com o salário mais elevado é o "Steven", com um valor de 24000.

```
/*Query 6 - TOP 5 do mais bem Pagos*/
select *
from employees
order by salary desc
FETCH FIRST 5 ROWS ONLY;
```

Figura 11. Comando SQL da query 6.

EMPLOYEE_ID	FIRST_NAME	LAST_NAME	EMAIL	PHONE_NUMBER	HIRE_DATE	JOB_ID	SALARY	COMMISSION_PCT	MANAGER_ID	DEPARTMENT_ID
100	Steven	King	SKING	515.123.4567	03.06.17	AD_PRES	24000	(null)	(null)	90
101	Neena	Kochhar	NKOCHHAR	515.123.4568	05.09.21	AD_VP	17000	(null)	100	90
102	Lex	De Haan	LDEHAAN	515.123.4569	01.01.13	AD_VP	17000	(null)	100	90
145	John	Russell	JRUSSEL	011.44.1344.429268	04.10.01	SA_MAN	14000	0,4	100	80
146	Karen	Partners	KPARTNER	011.44.1344.467268	05.01.05	SA_MAN	13500	0,3	100	80

Figura 12. Resultado da query 6 na base de dados Oracle.

3.2.7 *Query* 7 – Lista dos Departamentos por País e Código Postal

A *query* seguinte retorna uma listagem com os nomes distintos de departamentos, organizados através do código postal e país. No total existem 27 departamentos distribuídos por 4 países diferentes, onde os Estados Unidos são o país mais representado.

```
/*Query 7 - Lista do Nome dos Departamentos Por País e Código Postal*/
select distinct d.department_name, l.postal_code, c.country_name
from departments d
        inner join locations l on l.location_id = d.location_id
        inner join countries c on c.country_id = l.country_id
order by department_name;
```

Figura 13. Comando SQL da query 7.

DEPARTMENT_NAME	POSTAL_CODE	COUNTRY_NAME
Accounting	98199	United States of America
Administration	98199	United States of America
Benefits	98199	United States of America
Construction	98199	United States of America
Contracting	98199	United States of America
Control And Credit	98199	United States of America
Corporate Tax	98199	United States of America
Executive	98199	United States of America
Finance	98199	United States of America
Government Sales	98199	United States of America
Human Resources	(null)	United Kingdom
IT	26192	United States of America
IT Helpdesk	98199	United States of America
IT Support	98199	United States of America
Manufacturing	98199	United States of America
Marketing	M5V 2L7	Canada
NOC	98199	United States of America
Operations	98199	United States of America
Payroll	98199	United States of America
Public Relations	80925	Germany
Purchasing	98199	United States of America
Recruiting	98199	United States of America
Retail Sales	98199	United States of America
Sales	OX9 9ZB	United Kingdom
Shareholder Services	98199	United States of America
Shipping	99236	United States of America
Treasury	98199	United States of America

Figura 14. Resultado da query 7 na base de dados Oracle.

3.2.8 *Query* 8 – *Top* de Salário Médio por Departamento e Cidade

A *query* seguinte retorna uma listagem com o salário médio de cada *job title* distinto, organizada através da cidade e departamento. O cargo de presidente na cidade de Seattle do departamento Executivo é o que tem o salário médio mais elevado e os *Stock Clerk* da cidade de São Francisco do departamento de *Shipping* são os que têm o salário médio mais baixo.

```
/*Query 8 - TOP de Salário Médio por Departemento e Cidade*/
select distinct j.job_title, l.city, d.department_name,
            ((j.max_salary + j.min_salary)/2) as salario_medio
from departments d
        inner join employees e on e.department_id = d.department_id
        inner join jobs j on j.job_id = e.job_id
        inner join locations l on l.location_id = d.location_id
order by salario_medio desc;
```

Figura 15. Comando SQL da query 8.

JOB_TITLE	CITY	DEPARTMENT_NAME	SALARIO_MEDIO
President	Seattle	Executive	30040
Administration Vice President	Seattle	Executive	22500
Sales Manager	Oxford	Sales	15040
Accounting Manager	Seattle	Accounting	12100
Finance Manager	Seattle	Finance	12100
Marketing Manager	Toronto	Marketing	12000
Purchasing Manager	Seattle	Purchasing	11500
Sales Representative	Oxford	Sales	9004
Public Relations Representative	Munich	Public Relations	7500
Stock Manager	South San Francisco	Shipping	7000
Programmer	Southlake	IT	7000
Public Accountant	Seattle	Accounting	6600
Accountant	Seattle	Finance	6600
Human Resources Representative	London	Human Resources	6500
MarketingRepresentative	Toronto	Marketing	6500
Administration Assistant	Seattle	Administration	4500
Purchasing Clerk	Seattle	Purchasing	4000
Shipping Clerk	South San Francisco	Shipping	4000
Stock Clerk	South San Francisco	Shipping	3504

Figura 16. Resultado da query 8 na base de dados Oracle.

4. Base de Dados Orientada a Documentos – MongoDB

4.1. Modelo de Dados

O MongoDB corresponde a uma base de dados documental que é um tipo de base de dados não relacional, projetado para armazenar e consultar dados como documentos do tipo JSON. Neste caso, não são utilizadas tabelas e as informações de uma determinada entidade e dados associados são armazenados num único documento, não existindo o conceito de chaves estrangeiras. Ao contrário das bases de dados relacionais, não há necessidade de criar um esquema lógico que albergue os dados, sendo possível simplesmente carregar todos os dados sem um esquema predefinido. Além disso, as bases de dados documentais podem ser dimensionadas horizontalmente, enquanto as relacionais são mais adequadas a um dimensionamento vertical.

Para adquirir um modelo que permita albergar o conjunto de dados presente na base de dados Oracle no modelo documental foi necessário planear o modelo de dados no formato JSON, de modo a implementá-lo, tendo em conta as características do SDB MongoDB.

Várias abordagens foram tentadas, porém concluiu-se que o método mais apropriado para agregar toda a informação em duas coleções, separando os dados relativos aos empregados e uma outra relativamente aos departamentos.

A **coleção empregados**, visível na imagem abaixo, comtempla a informação associada às tabelas *Employees*, *Jobs*, *Jobs_history* e o id do departamento do emprego atual. Este *id* do departamento permite a ligação para realizar a agrupação dos dados entre as duas coleções.

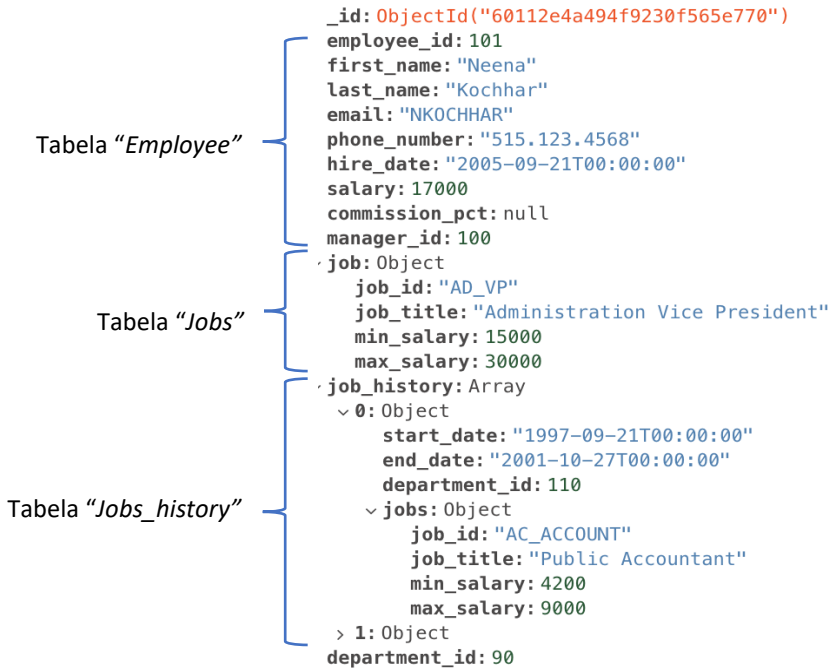

Figura 17. Documento da coleção empregados.

Relativamente à **coleção departamentos**, visível na imagem abaixo, contempla a informação associada às tabelas *Departments*, *Locations*, *Countries* e *Jobs_regions.*

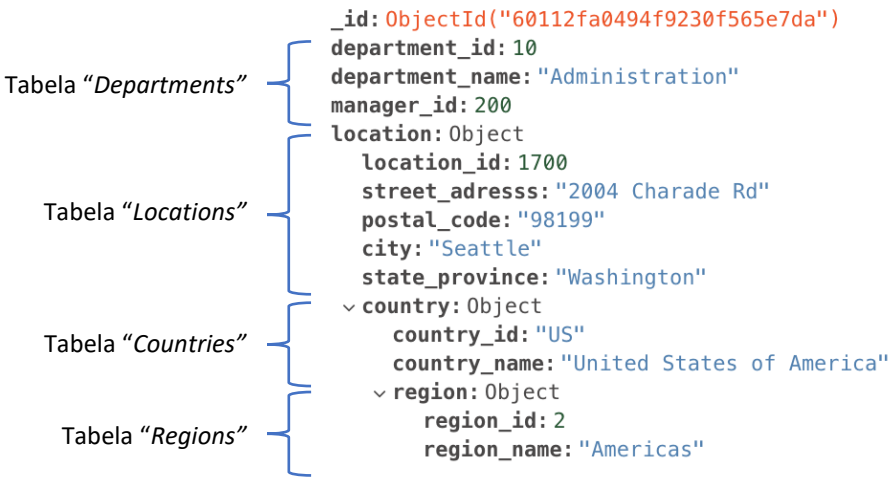

Figura 18. Documento da coleção departamentos.

Posto isto, possui-se toda a informação compacta e de fácil visualização. Precedentemente, tentou-se mais dois formatos de modelos JSON, mas mediante uma análise profunda detetou-se que o mais conveniente seria o que referido acima.

4.1.1 Abordagem 1

Uma coleção Departamentos, como a base de dados contém 27 departamentos consequentemente havia 27 documentos. Esta abordagem tornava-se custosa porque para realizar *queries* relacionadas com os empregados era necessário efetuar agregações com alguma complexidade.

- Informação da Tabela "*Departments*"
    - Informação da Tabela "*Locations*"
    - Informação da Tabela "*Countries*"
    - Informação da Tabela "*Regions*"
- *Array employees*
    - Informação da Tabela "*Employees*"
        - Informação da Tabela "*Jobs*"
        - Informação da Tabela "*Jobs_history*"

4.1.2 Abordagem 2

Uma coleção Empregados, como a base de dados contém 107 empregados consequentemente havia 107 documentos. Esta abordagem trazia pouca vantagem porque o conteúdo de cada departamento

surgia repetido consoante o número de empregados do mesmo, com o isto o tamanho da base de dados praticamente duplicava comparativamente com o formato final.

- Informação da Tabela *Employees*
    - Informação da Tabela *Jobs*
    - Informação da Tabela *Jobs_history*
- Informação da Tabela *Departments*
    - Informação da Tabela *Locations*
    - Informação da Tabela C*ountries*
    - Informação da Tabela *Regions*

4.2. Migração de Dados

A migração de dados contidos na base de dados relacional OracleDB para o modelo documental MongoDB foi realizada entre várias etapas. Após uma pesquisa detetamos que o OracleDB fornece funções que utilizadas em *queries* permitem construir um JSON mediante a forma que o utilizador necessitar. Através destas, foi possível gerar documentos JSON complexos e hierárquicos aninhados, as subconsultas aninhadas podem gerar coleções JSON que representam relacionamentos um-para-muitos.

Observando a Tabela 1, encontra-se evidenciado como se torna fácil a junção do primeiro e último nome do empregado e a transição para JSON.

QUERY
`select json_object('name' value first_name || ' ' || last_name) as full_name from employees;`
RESULTADO

`{"name": "Ellen Abel"}`
`{"name": "Sundar Ande"}`
`{"name": "David Austin"}`

Tabela 1. JSON_OBJECT transição MongoDB.

Posto isto e tirando partido destas funções, estruturou-se o JSON da forma que se necessitava. Esta tarefa foi realizada de forma acelerada, visto que a base de dados é pequena e as *queries* de estruturação do JSON eram relativamente simples.

Apesar de não ser necessária a realização de nenhum *script* para automatizar o processo, concluiu-se que seria mais funcional. Deste modo, efetuou-se um *script* em *Python* (Anexo 1 ou Ficheiro

ScriptMongo.py), em que a *query* com a estrutura JSON é executada e os resultados são inseridos na devida coleção.

Seguidamente na Tabela 2 e na Tabela 3, apresentou-se de forma reduzida as *queries* SQL que permitem gerar a coleção *employees* e a coleção *departments,* respetivamente, através das funções **json_object()** e **json_arrayagg().**

- **json_object() -** Constrói objetos JSON a partir de pares nome-valor, em que o nome é textual e valor pode ser qualquer expressão SQL .
- **json_arrayagg()** - Constrói uma array JSON agregando informações de várias linhas agrupadas. A ordem dos elementos reflete na ordem dos resultados da consulta, esta pode ser trabalhada mediante a utilização da order_by na *query.*

```
select json_object (
    'employee_id' value e.employee_id,
    (...),
    'job' value (
        select json_object(
              'job_id' value j.job_id,
              (...)
            )
        from jobs j
        where j.job_id = e.job_id
    ),
    'job_history' value (
        select json_arrayagg(
            json_object(
              (...)
              'jobs' value (
                    select json_object(
                          'job_id' value j.job_id,
                          (...)
                        )
                    from jobs j
                    where j.job_id = jh.job_id
                )
            )
        )
        from job_history jh
        where jh.employee_id = e.employee_id
    ),
    'department_id' value e.department_id
)
from employees e;
```

Tabela 2. SQL para a coleção employees.

```
select json_object (
    'department_id' value d.department_id,
    (...),
    'location' value (
        select json_object(
              'location_id' value l.location_id,
              (...)
              'country' value (
                    select json_object(
                          'country_id' value c.country_id,
                          'country_name' value c.country_name,
```

```
                        'region' value (
                              select json_object(
                                    'region_id' value r.region_id,
                                    'region_name' value r.region_name
                                )
                              from regions r
                              where r.region_id = c.region_id
                          )
                      )
                  from countries c
                  where c.country_id = l.country_id
              )
          )
        from locations l
        where l.location_id = d.location_id
    )
)
from departments d;
```

Tabela 3. SQL para a coleção departments.

Sendo assim, com as *queries* preparadas apenas foi necessário introduzi-las no *script* de modo a realizar a inserção automática.

4.3. *Queries*

De acordo com o referido anteriormente, implementou-se 8 *queries* em MongoDB, que retornam os mesmos registos que as *queries* feitas no SDB Oracle e Neo4j, para comprovar a solidez dos 2 modelos NoSQL produzidos.

4.3.1 *Query* 1 – Lista de Cidades por Departamento

De forma semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem de cidades por departamento. Através do *output* obtido é possível confirmar que no total existem 7 cidades distintas por departamento.

```
db.departments.distinct("location.city");
```

```
[
        "London",
        "Munich",
        "Oxford",
        "Seattle",
        "South San Francisco",
        "Southlake",
        "Toronto"
]
```

Figura 19. Resultado da query 1 em MongoDB.

4.3.2 *Query* 2 – Lista de Empregados com Mudanças de Emprego

De forma semelhante à *query* implementada no modelo relacional, esta também retorna a listagem de empregados que tiveram mudanças de emprego. Foi possível verificar que no total existem 7 empregados que mudaram de emprego, sendo que a "Neena", o "Jonathon" e a "Jennifer" mudaram 2 vezes de cargo e os restantes mudaram apenas uma vez.

```
db.employees.aggregate([{ '$project': { '_id': 0, 'first_name': 1, 'ocorrencias': {
'$size': { '$ifNull': [ '$job_history', [] ]}}}}, {'$match': { 'ocorrencias': {'$gt':
                    0 }}}, { '$sort': { 'ocorrencias': -1 }}]);
```

```
{ "first_name" : "Neena", "ocorrencias" : 2 }
{ "first_name" : "Jonathon", "ocorrencias" : 2 }
{ "first_name" : "Jennifer", "ocorrencias" : 2 }
{ "first_name" : "Lex", "ocorrencias" : 1 }
{ "first_name" : "Den", "ocorrencias" : 1 }
{ "first_name" : "Payam", "ocorrencias" : 1 }
{ "first_name" : "Michael", "ocorrencias" : 1 }
```

Figura 20. Resultado da query 2 em MongoDB.

4.3.3 *Query* 3 – Lista Geral do Conteúdo das Base de Dados

De forma semelhante à *query* implementada no modelo relacional, esta também retorna a listagem de todos os dados relacionados com todos os empregados. É de notar que, dada a extensão da base de dados, a Figura 21 apresenta apenas parte dos resultados obtidos.

```
db.employees.aggregate([{$lookup: { from: 'departments', localField: 'department_id',
foreignField: 'department_id', as: 'departments'}}]).pretty();
```

Figura 21. Resultado da query 3 em MongoDB.

4.3.4 *Query* 4 – Lista de Empregados Cujo Nome Inicia com a Letra "J"

De forma semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem reduzida com informação referente a cada empregado, cujo nome começa com a letra "J". Mais uma vez, foi possível verificar que listagem era igual à obtida no SDB Oracle.

```
db.employees.find({"first_name": /^J.*/}, {_id: 0, job: 0, job_history: 0,
department: 0}).pretty();
```

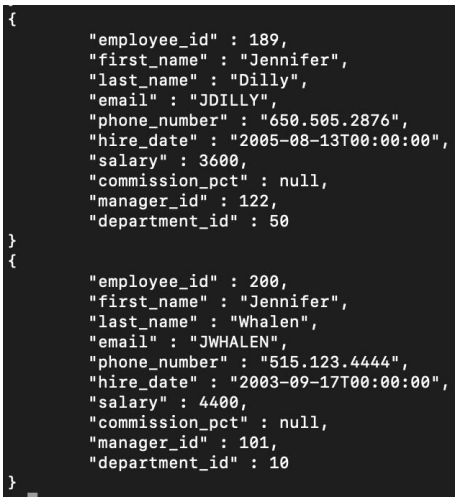

Figura 22. Resultado da query 4 em MongoDB.

4.3.5 *Query* 5 – Lista de Todos os Presidentes

De forma semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem com diversa informação, nomeadamente o nome, a data de contratação e o respetivo salário, de todos Presidentes. Verificamos que apenas o "Steven" tem o cargo de Presidente.

```
db.employees.find({"job.job_title": "President"}, {_id: 0, employee_id: 0, last_name:
0, email: 0, phone_number: 0, job_id: 0, commission_pct: 0, manager_id: 0, job: 0,
job_history: 0, department: 0}).pretty();
```

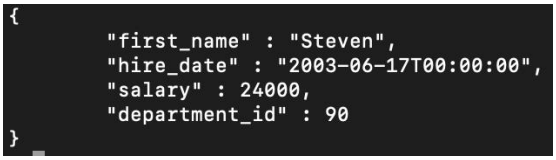

Figura 23. Resultado da query 5 em MongoDB.

4.3.6 *Query* 6 – *Top* 5 dos Funcionários Mais Bem Pagos

De forma semelhante à *query* implementada no modelo relacional, esta também retorna a informação relativamente ao *top* 5 dos funcionários mais bem pagos. Desta forma comprovou-se que a pessoa com o salário mais elevado é o "Steven", com um valor de 24000.

```
db.employees.find({}, {_id: 0, job: 0, job_history: 0, department: 0}).sort({salary:-
1}).limit(5).pretty();
```

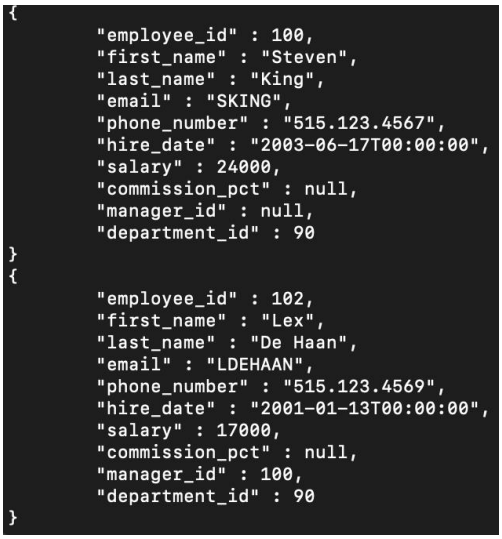

Figura 24. Resultado da query 6 em MongoDB.

4.3.7 *Query* 7 – Lista dos Departamentos por País e Código Postal

De forma semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem reduzida com os nomes distintos de departamentos, organizados através do código postal e país. Verificou-se que no total existem 27 departamentos distribuídos por 4 países diferentes, onde os Estados Unidos são o país mais representado.

```
db.departments.find({}, {_id: 0, "location.postal_code": 1, department_name: 1,
                "location.country.country_name": 1}).pretty();
```

Figura 25. Resultado da query 7 em MongoDB.

4.3.8 *Query* 8 – *Top* de Salário Médio por Departamento e Cidade

De forma semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem com o salário médio de cada *job title* distinto, organizada através da cidade e departamento. O cargo de presidente na cidade de Seattle do departamento Executivo é o que tem o salário médio mais elevado e os *Stock Clerk* da cidade de São Francisco do departamento de *Shipping* são os que têm o salário médio mais baixo.

```
db.departments.aggregate([{
        '$lookup': {
            'from': 'employees',
            'localField': 'department_id',
            'foreignField': 'department_id',
            'as': 'employees'}
    }, {
        '$unwind': {'path': '$employees', 'includeArrayIndex': 'index'}
    }, {
        '$addFields': {'soma_salario': {'$sum': ['$employees.job.max_salary',
'$employees.job.min_salary']}}
    }, {
        '$addFields': {'salario_medio': {'$divide': ['$soma_salario', 2]}
        }
    }, {
        '$project': {'_id': 0, 'department_name': 1, 'location.city': 1,
'salario_medio': 1}
    }, {
        '$unwind': {'path': '$department_name'}
    }, {
        '$group': {'_id': null,
            'resultado': {
                '$addToSet': {'department_name': '$department_name', 'city':
'$location.city', 'salario_medio': '$salario_medio'}}}
    }, {
        '$project': {'resultado': 1, '_id': 0}
    }, {
        '$unwind': {'path': '$resultado'}
    }, {
        '$sort': {'resultado.salario_medio': -1}
    }
]);
```


```
{ "resultado" : { "department_name" : "Executive", "city" : "Seattle", "salario_medio" : 30040 } }
{ "resultado" : { "department_name" : "Executive", "city" : "Seattle", "salario_medio" : 22500 } }
{ "resultado" : { "department_name" : "Sales", "city" : "Oxford", "salario_medio" : 15040 } }
{ "resultado" : { "department_name" : "Finance", "city" : "Seattle", "salario_medio" : 12100 } }
{ "resultado" : { "department_name" : "Accounting", "city" : "Seattle", "salario_medio" : 12100 } }
{ "resultado" : { "department_name" : "Marketing", "city" : "Toronto", "salario_medio" : 12000 } }
{ "resultado" : { "department_name" : "Purchasing", "city" : "Seattle", "salario_medio" : 11500 } }
{ "resultado" : { "department_name" : "Sales", "city" : "Oxford", "salario_medio" : 9004 } }
{ "resultado" : { "department_name" : "Public Relations", "city" : "Munich", "salario_medio" : 7500 } }
{ "resultado" : { "department_name" : "Shipping", "city" : "South San Francisco", "salario_medio" : 7000 } }
{ "resultado" : { "department_name" : "IT", "city" : "Southlake", "salario_medio" : 7000 } }
{ "resultado" : { "department_name" : "Accounting", "city" : "Seattle", "salario_medio" : 6600 } }
{ "resultado" : { "department_name" : "Finance", "city" : "Seattle", "salario_medio" : 6600 } }
{ "resultado" : { "department_name" : "Human Resources", "city" : "London", "salario_medio" : 6500 } }
{ "resultado" : { "department_name" : "Marketing", "city" : "Toronto", "salario_medio" : 6500 } }
{ "resultado" : { "department_name" : "Administration", "city" : "Seattle", "salario_medio" : 4500 } }
{ "resultado" : { "department_name" : "Shipping", "city" : "South San Francisco", "salario_medio" : 4000 } }
{ "resultado" : { "department_name" : "Purchasing", "city" : "Seattle", "salario_medio" : 4000 } }
{ "resultado" : { "department_name" : "Shipping", "city" : "South San Francisco", "salario_medio" : 3504 } }
```


Figura 26. Resultado da query 8 em MongoDB.

5. Base de Dados Orientada a Grafos – Neo4j

5.1. Modelo de Dados

Com o objetivo de obter um modelo que permita guardar o conjunto de dados presente na BD segundo o modelo relacional no Oracle, no modelo orientado a grafos, foi necessário repensar o esquema de BD, de modo a que fosse possível implementar o mesmo tendo em conta as características do SDB Neo4j.

Ao derivar um modelo gráfico de um modelo relacional, algumas diretrizes devem ser mantidas, como por exemplo, uma linha adapta-se num nó, o nome de uma tabela corresponde a um *label* e as chaves estrangeiras correspondem a relacionamentos. Com base nestes princípios, foi possível mapear o modelo relacional requerido no modelo gráfico escolhido seguindo algumas etapas.

Mapeamento dos registos em nós e dos nomes das tabelas em *labels:*

1. Cada linha da tabela "*Employees*" no OracleDB é mapeada num nodo no modelo gráfico. Os atributos presentes na tabela são convertidos em propriedades do referido nodo e o nome da tabela corresponde a um *label* (*Label*: **Employee**).
2. Cada registo da tabela "*Jobs*" é mapeado num nodo, sendo as suas propriedades os atributos presentes na referida tabela. O nome da tabela corresponde a um (*Label*: **Job**).
3. Cada registo da tabela "*Departments*" corresponde a um nodo. De igual modo, os atributos e o nome da tabela correspondem a, respetivamente, às propriedades e *label* do nodo (*Label*: **Department**).
4. Como a tabela *"Job_History*" apenas apresenta os identificadores do departamento e do *job,* foi necessário fazer uma junção entre as tabelas "*Department"*, "*Jobs"* e "*Job_History" para ob*ter as informações referentes aos identificadores. A tabela resultante apresenta, em vez dos identificadores, os respetivos nomes. A partir da tabela resultante, cada registo é mapeado num nodo e as propriedades e o *label* do nodo correspondem aos atributos e o nome da tabela resultante (*Label*: **Job_History**).
5. Na tabela "*Locations*", cada registo, os seus atributos e o nome da tabela correspondem a, respetivamente, um nodo, as propriedades do nodo e o seu *label* (*Label*: **Location**).
6. Na tabela "*Countries*" o procedimento foi idêntico, ou seja, cada registo corresponde a um nodo, sendo as propriedades do nodo os atributos da tabela e o seu *label* o nome da tabela.
7. Na tabela "*Regions*", cada registo corresponde a um nodo, sendo as suas propriedades os atributos presentes na tabela e o *label* o nome da tabela (*Label*: **Region**).

Na figura seguinte é possível observar alguns tipos de nodos e como eles estão ligados através dos relacionamentos. O nodo central a verde corresponde ao continente Europeu e conectados a ele estão vários países (castanho claro). A cada país estão ligadas várias cidades onde a presente empresa opera (rosa). A cada cidade podem estar ligados vários departamentos, como por exemplo, a cidade de Oxford contém o departamento *Sales* (azul escuro) e conectados a este departamento estão alguns funcionários (azul claro).

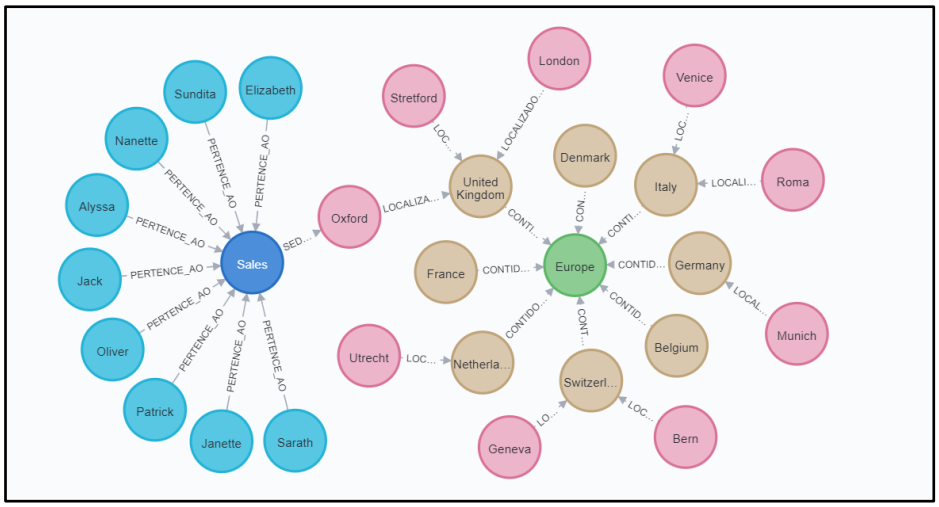

Figura 27. Localizações disponíveis no continente europeu.

No sentido de melhorar a performance, aumentar a disponibilidade dos dados e reduzir a complexidade nas consultas na presente base de dados gráfica, implementou-se vários relacionamentos entre os nodos. O objetivo foi que sempre que se fizesse uma pesquisa por um nodo, outros nodos relacionados com o nodo pesquisado aparecessem apresentando informações relacionadas. A ligação entre o nodo pesquisado e os nodos relacionados é feita com recurso a setas que denotam o tipo de relacionamento existente.

Relacionamentos entre os nodos:

1. Cada *employee* tem o seu *job* atual. Desta forma, entre os nodos do tipo *Employee* e *Job* foi implementado uma relação denotada **TRABALHA_EM.**
2. Como cada *employee* provavelmente já teve *jobs* antigos, entre os nodos do tipo *Employee* e *Job_History* implementamos a relação **TRABALHOU_EM**. A presente relação tem duas propriedades, startDate e endDate.
3. Como existem *employees* que são gerentes de outros *employees*, implementamos a relação **GERIDO_POR**. Este relacionamento corresponde à relação unária existente na tabela *Employees* no modelo relacional.
4. Ainda em relação aos nodos do tipo *Employee*, como cada *employee* trabalha num *department*, implementamos a relação **PERTENCE_AO** entre nodos do tipo *Employee* e Department.

5. Cada departamento possui um gerente. Sendo assim, entre cada nodo do tipo *Department* e o nodo do tipo *Employee* cujo funcionário é o gerente, implementamos o relacionamento **GERENTE_DO**.
6. Cada departamento está situado numa localização. Deste forma, entre os nodos do tipo *Department* e *Location* implementamos o relacionamento **SEDIADO_EM**.
7. Um país contém localizações onde vários departamentos estão situados, posto isso, entre os nodos do tipo *Location* e *Country* concebemos o relacionamento **LOCALIZADO_EM**.
8. Como uma região pode conter vários países, entre os nodos do tipo *Country* e *Region* implementamos o relacionamento **CONTIDO_EM**.

A título de exemplo, na figura seguinte podemos ver que o *employee* Steven é gerente (manager) de 14 *employee*, pertence ao departamento *Executive* na qual é o respetivo gerente e atualmente tem o cargo de Presidente.

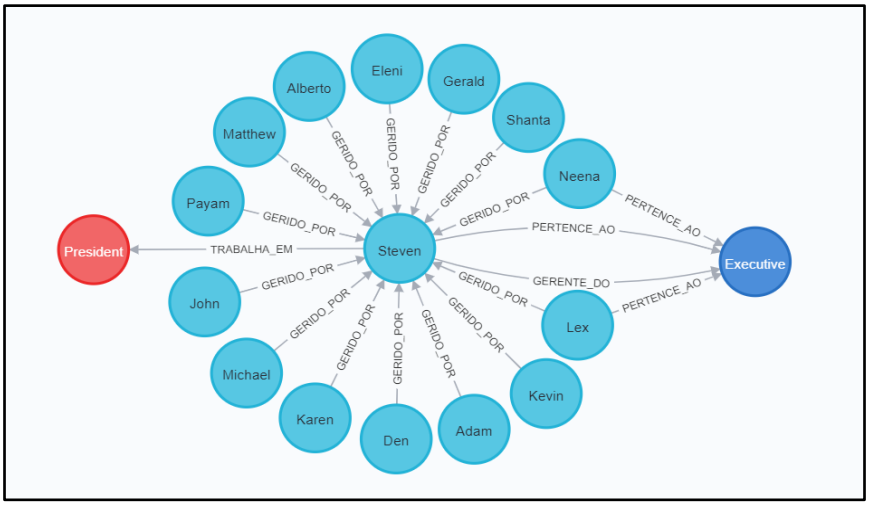

Figura 28. Relacionamentos do presidente Steven.

Na figura seguinte é apresentada uma visão geral dos nodos e relacionamentos concebidos na base de dados Neo4j. Tal como demonstrado na figura, a base de dados Neo4j possui 215 nodos divididos em 7 *labels* e 414 relacionamentos divididos em 8 tipos.

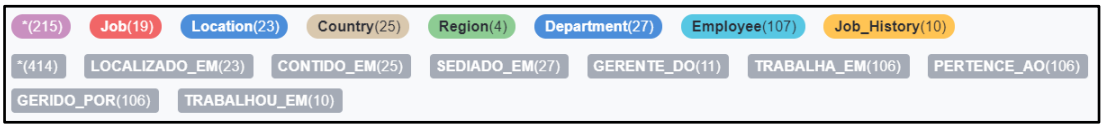

Figura 29. Nodos e relacionamentos no SBD Neo4j.

Na figura seguinte é apresentado o esquema da base de dados gráfica. Neste é possível ver cada tipo de nodo (cores diferentes) e de relacionamento implementados.

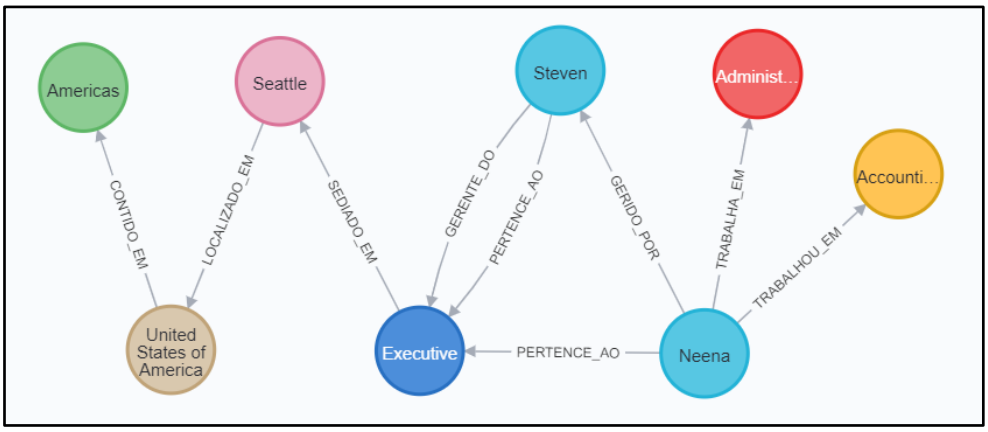

Figura 30. Esquema da base de dados Neo4j.

5.2. Migração de Dados

A migração dos dados contidos na base de dados relacional OracleDB para o modelo não relacional Neo4j foi dividida em duas fases. A primeira fase teve como cerne converter os dados presentes nas tabelas da base de dados relacional num formato que fosse fácil de carregar para o Neo4j para que na segunda fase esses dados fossem inseridos com sucesso na base de dados.

Como explicito anteriormente, na primeira fase convertemos os dados presentes nas tabelas do modelo relacional para ficheiros no formato .csv. Para tal foi concebido um *script* em *Python* (Anexo 2 ou ficheiro *exportDataCSV.py*). Após o término da execução do *script*, são concebidos oito ficheiros no formato .csv. Sete ficheiros correspondem aos registos presentes nas sete tabelas do modelo relacional em que alguns atributos das tabelas correspondem aos cabeçalhos dos respetivos dos ficheiros. Como retratado no subtópico 5.1 foi necessário fazer a junção de três tabelas. Sendo assim, o último ficheiro .csv contém os registos da junção das referidas três tabelas.

A segunda fase representou o carregamento e a construção da base de dados gráfica através dos ficheiros .csv concebidos na primeira fase. Para tal foi construído um *script* em *Cypher* (Anexo 3 ou ficheiro *importNeo4J.cypher*). Na primeira parte do *script* são inseridas as informações gerais na base de dados Neo4j, isto é, para cada linha de cada ficheiro .csv é construído um nodo em que as suas propriedades são os campos da respetiva linha separados por uma vírgula. Na figura seguinte é apresentado o código que gerou todos os nodos do tipo *Location*. Foi lido o ficheiro *locations.csv* e para cada linha definimos um nodo e as suas propriedades.

```
59  // Informações das localizacoes
60  LOAD CSV WITH HEADERS FROM 'file:///locations.csv' AS row
61  CREATE (location:Location {idLocation: row.LOCATION_ID})
62  SET location.streerAddress= row.STREET_ADDRESS,
63      location.postalCode = row.POSTAL_CODE,
64      location.city = row.CITY,
65      location.stateProvince = row.STATE_PROVINCE,
66      location.idCountry = row.COUNTRY_ID
67  RETURN location;
```

Figura 31. Génese dos nodos Location.

Na segunda parte do *script* são concebidos os relacionamentos entre os nodos. Na figura seguinte é apresentado o código que está na génese dos relacionamentos entre os nodos do tipo *Country* e *Region*. O código mostra que é percorrida cada linha do ficheiro *regions.csv* (que contém a lista de todos as regiões) e em cada linha é procurado o nodo do tipo *Country* com a propriedade *idRegion* que faz match com o campo REGION_ID da respetiva linha e procurado o nodo do tipo *Region* com a propriedade *idRegion* que faz match com o campo REGION_ID da mesma linha. Dispondo de ambos os nodos, de seguida é concebida a relação CONTIDO_EM entre os referidos nodos.

```
108  // Relação countries -> regions
109  LOAD CSV WITH HEADERS FROM 'file:///regions.csv' AS row
110  MATCH (country:Country {idRegion: row.REGION_ID})
111  MATCH (region:Region {idRegion: row.REGION_ID})
112
113  CREATE (country)-[:CONTIDO_EM]->(region)
114  RETURN country,region;
```

Figura 32. Génese dos relacionamentos CONTIDO_EM.

5.3. *Queries*

De acordo com o referido anteriormente, implementou-se 8 *queries* em Neo4j, que retornam os mesmos registos que as *queries* feitas no SDB Oracle e MongoDB, para comprovar a solidez dos 2 modelos NoSQL produzidos.

5.3.1 *Query* 1 – Lista de Cidades por Departamento

Semelhante à *query* implementada no modelo relacional, a seguinte retorna igualmente uma listagem única de cidades por departamento. É possível comprovar que de facto existem as 7 cidades distintas por cada departamento.

```
1 MATCH (d:Department) -[r:SEDIADO_EM]→ (l:Location)
2 RETURN DISTINCT l
```

Figura 33. Comando da query 1.

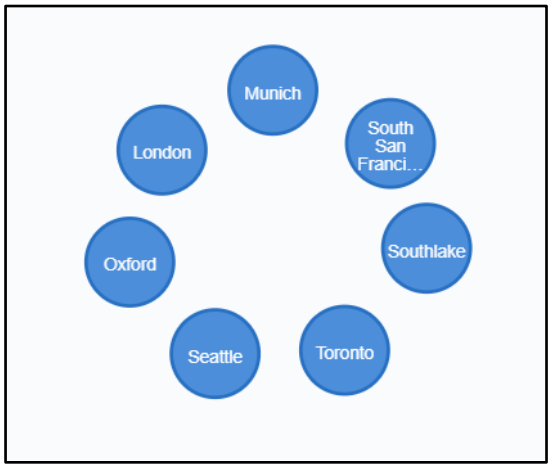

Figura 34. Resultado da query 1 na base de dados Neo4j.

5.3.2 *Query* 2 – Lista de Empregados com Mudanças de Emprego

Semelhante à *query* implementada no modelo relacional, esta retorna a listagem de empregados que tiveram mudanças de emprego. Foi possível verificar que no total existem 7 empregados que mudaram de emprego, sendo que a "Neena", o "Jonathon" e a "Jennifer" mudaram 2 vezes de trabalho, tendo os restantes mudado apenas uma vez.

```
MATCH (e:Employee) -[r:TRABALHOU_EM]→ (a:Job_History)
RETURN e.firstName AS First_Name, COUNT(r) AS Ocorrencias
ORDER BY Ocorrencias DESC
```

Figura 35. Comando da query 2.

"First_Name"	"Ocorrencias"
"Jonathon"	2
"Jennifer"	2
"Neena"	2
"Michael"	1
"Den"	1
"Payam"	1
"Lex"	1

Figura 36. Resultado da query 2 na base de dados Neo4j.

5.3.3 *Query* 3 – Lista Geral do Conteúdo das Base de Dados

Semelhante à *query* implementada no modelo relacional, esta retorna a listagem de todos dados relacionados com todos os empregados. Dada a extensão do conteúdo da base de dados, a Figura 38 apresenta apenas parte dos resultados obtidos.

```
MATCH (n)
RETURN n
```

Figura 37. Comando da query 3.

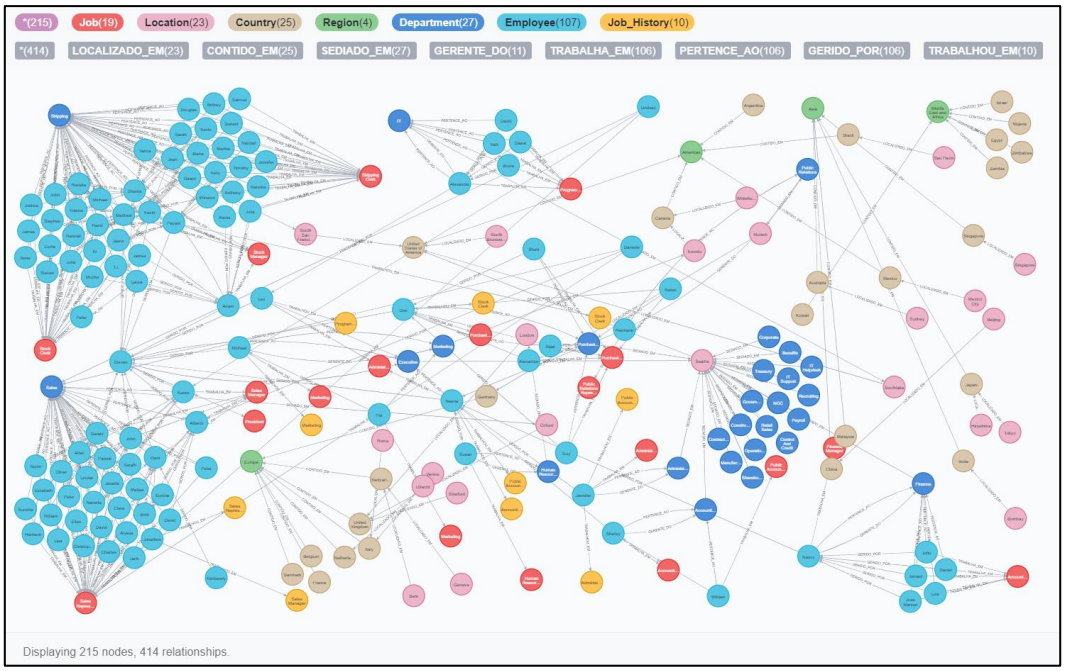

Figura 38. Resultado da query 3 na base de dados Neo4j.

5.3.4 *Query* 4 – Lista de Empregados Cujo Nome Inicia com a Letra "J"

Semelhante à *query* implementada no modelo relacional, esta retorna uma listagem com informação referente a cada empregado, cujo nome começa com a letra "J" e vem mais uma vez validar a implementação do Modelo de Dados no Neo4j.

```
1 MATCH (e:Employee)
2 WHERE e.firstName =~'J.*'
3 RETURN e
```

Figura 39. Comando da query 4.

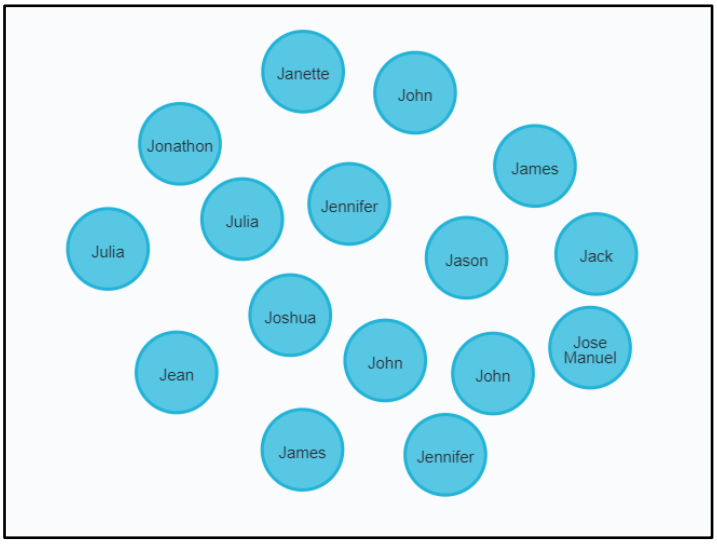

Figura 40. Resultado da query 4 na base de dados Neo4j.

5.3.5 *Query* 5 – Lista de Todos os Presidentes

Semelhante à *query* implementada no modelo relacional, esta retorna uma listagem com informação, nomeadamente o nome, a data de contratação e o respetivo salário, de todos Presidentes. Neste caso em concreto, apenas o "Steven" tem um *job title* de Presidente.

```
MATCH (n:Employee) -[:TRABALHA_EM]→ (:Job{jobTitle:"President"})
RETURN n.firstName AS Name, n.hireDate AS HireDate, n.salary AS Salary
```

Figura 41. Comando da query 5.

"Name"	"HireDate"	"Salary"
"Steven"	"2003-06-17 00:00:00"	"24000.0"

Figura 42. Resultado da query 5 na base de dados Neo4j.

5.3.6 *Query* 6 – *Top* 5 dos Funcionários Mais Bem Pagos

Semelhante à *query* implementada no modelo relacional, esta retorna informação sobre o Top 5 dos funcionários mais bem pagos estando de acordo com os dados apresentas na *query* segundo o Modelo Relacional. Novamente verificamos que o mais bem pago é o "Steven" com um salário de "24000".

```
MATCH (e:Employee)
RETURN e.firstName AS Name, toInteger(e.salary) AS Salary
ORDER BY toInteger(e.salary) DESC
LIMIT 5;
```

Figura 43. Comando da query 6.

"Name"	"Salary"
"Steven"	24000
"Lex"	17000
"Neena"	17000
"John"	14000
"Karen"	13500

Figura 44. Resultado da query 6 na base de dados Neo4j.

5.3.7 *Query* 7 – Lista dos Departamentos por País e Código Postal

Semelhante à *query* implementada no modelo relacional, esta retorna igualmente uma listagem com os distintos nomes de departamentos, organizados através do código postal e país. No total temos 27 departamentos distribuídos por 4 países diferentes, onde os Estados Unidos são o país mais representado. No caso da evidência do resultado apresentado, dada a extensão do *output*, no relatório apresentou-se apenas os 10 primeiros departamentos.

```
MATCH (d:Department) -[:SEDIADO_EM]→(l:Location) -[:LOCALIZADO_EM]→ (c:Country)
RETURN d.nameDepartmant AS NameDepartment, toInteger(l.postalCode)
       AS PostalCode, c.nameCountry AS Country
ORDER BY NameDepartment
```

Figura 45. Comando da query 7.

"NameDepartment"	"PostalCode"	"Country"
"Accounting"	98199	"United States of America"
"Administration"	98199	"United States of America"
"Benefits"	98199	"United States of America"
"Construction"	98199	"United States of America"
"Contracting"	98199	"United States of America"
"Control And Credit"	98199	"United States of America"
"Corporate Tax"	98199	"United States of America"
"Executive"	98199	"United States of America"
"Finance"	98199	"United States of America"
"Government Sales"	98199	"United States of America"

Figura 46. Resultado da query 7 na base de dados Neo4j.

5.3.8 *Query* 8 – *Top* de Salário Médio por Departamento e Cidade

Semelhante à *query* implementada no modelo relacional, esta também retorna uma listagem com o salário médio de cada *job title* único, organizada através da cidade e departamento. Assim foi possível comprovar que os Presidentes da cidade de Seattle do departamento Executivo são quem têm o salário médio mais alto, e os *Stock Clerk* da cidade de São Francisco do departamento de *Shipping* são os que têm o salário médio mais baixo.

```
MATCH (j:Job) <-[:TRABALHA_EM]-(E:Employee)-[:PERTENCE_AO]->
      (d:Department)-[:SEDIADO_EM]->(l:Location)
RETURN DISTINCT j.jobTitle AS Job, l.city AS City, d.nameDepartmant
       AS Department, (toInteger(j.maxSalary) + toInteger(j.minSalary))/2
       AS Salary
ORDER BY Salary DESC
```

Figura 47. Comando da query 8.

"Job"	"City"	"Department"	"Salary"
"President"	"Seattle"	"Executive"	30040
"Administration Vice President"	"Seattle"	"Executive"	22500
"Sales Manager"	"Oxford"	"Sales"	15040
"Finance Manager"	"Seattle"	"Finance"	12100
"Accounting Manager"	"Seattle"	"Accounting"	12100
"Marketing Manager"	"Toronto"	"Marketing"	12000
"Purchasing Manager"	"Seattle"	"Purchasing"	11500
"Sales Representative"	"Oxford"	"Sales"	9004
"Public Relations Representative"	"Munich"	"Public Relations"	7500
"Programmer"	"Southlake"	"IT"	7000

Figura 48. Resultado da query 8 na base de dados Neo4j.

6. Discussão dos Resultados

Cada um dos sistemas de gestão de base de dados selecionados, nomeadamente, Oracle, MongoDB e Neo4j, possui diferentes características e finalidades. Ao migrar os dados das bases de dados do modelo relacional para os dois modelos não relacionais e ao implementar as mesmas *queries* em todos os sistemas foi possível identificar os pontos fortes e pontos fracos de cada um deles. Claro está que estas estão relacionadas com as nossas escolhas, outras abordagens teriam outras conclusões.

	Vantagens	**Desvantagens**
OracleDB	1. **Fácil exportação** das consultas; 2. **Consultas** fáceis.	1. **Esquema** rígido com relações; 2. **Consultas** com necessidade de realizar *joins;* 3. **Informação espalhada** por várias tabelas.
MongoDB	1. Sem **esquema rígido;** 2. **Carregamento ágil**; 3. **Rapidez;** 4. Linguagem **JSON.**	1. **Consultas** com projeções alguma complexidade;
Neo4j	1. **Consultas** rápidas e intuitivas; 2. **Carregamento ágil**; 3. **Rapidez** e **Flexibilidade;**	1. **Difícil** visualização com uma grande carga de dados; 2. **Estudo** da linguagem Cypher.

Tabela 4. Vantagens e desvantagens de cada um dos SDB utilizados.

7. Conclusão

O presente trabalho prático consistiu na análise, planeamento e implementação de um SGBD relacional e dois não relacionais, através da adaptação da base de dados relacional HR disponibilizada no Oracle. Com a sua realização foi possível consolidar a aprendizagem na unidade curricular de Bases de Dados NoSQL, nomeadamente na seleção de metodologias apropriadas para modelar sistemas de dados, assim como na manipulação de dados em bases de dados SQL e NoSQL.

Ao nível de dificuldades encontradas no desenvolvimento deste trabalho prático, a maior delas esteve na implementação do SDB MongoDB, uma vez que a elaboração das *queries* revelou-se um processo bastante complexo comparativamente ao Oracle e ao Neo4J. Além disso a passagem de dados de uma base de dados relacional para uma base de dados documental revelou-se também um desafio, visto que, geralmente, ao passar esta informação se fica com dados redundantes. No entanto, o grupo de trabalho pensa ter ultrapassado com sucesso esta situação com o modelo desenvolvido.

Relativamente à migração dos dados da base de dados relacional para a base de dados orientada a grafos, este foi um processo mais intuitivo, uma vez que se fez a associação entre elementos do modelo relacional e do orientado a grafos, isto é, os registos correspondem a nodos, os nomes das tabelas correspondem a *labels* e chaves estrangeiras ou *joins* correspondem a relacionamentos.

Com a realização do trabalho foi possível perceber que cada SBD implementado foi desenvolvido para uma finalidade diferente. No caso da Oracle, que segue o modelo relacional é uma base de dados *Enterprise*, com suporte comercial. O MongoDB, por sua vez, é melhor para consultas de informações que estejam estruturadas num pequeno documento, sem informação aninhada. Relativamente ao Neo4j, tem o melhor desempenho no que toca a pesquisas relacionais.

Deste modo, concluiu-se que na escolha de um sistema de gestão de base de dados deve-se ter atenção às características que são privilegiadas e quais serão as *queries* mais frequentes, para se obter o melhor desempenho possível.

Em suma e reiterando, considera-se que foi possível aplicar todo o conhecimento adquirido ao longo do semestre e planear e implementar um SGBD relacional e dois não relacionais.

Referências

Cardoso, R. M. (2012). *Bases de Dados NoSQL.* Instituto Superior de Engenharia do Porto, Departamento de Informática, Porto. Obtido de https://core.ac.uk/download/pdf/302865845.pdf

Neo4j. (2021). *Tutorial: Import Relational Data Into Neo4j.* Obtido de Neo4j Developer: https://neo4j.com/developer/guide-importing-data-and-etl/

Pereira, D. J. (2014). *Armazéns de Dados em Bases de Dados NoSQL.* Instituto Superior de Engenharia do Porto, Departamento de Informática, Porto. Obtido de https://recipp.ipp.pt/bitstream/10400.22/5681/1/DM_DanielPereira_2014_MEI.pdf

Lista de Siglas e Acrónimos

BD Base de Dados

SDB *System Database* (Sistema de Base de Dados)

HR *Human Resources* (Recursos Humanos)

Anexos

Esta secção contém os *scripts* criados para a migração dos dados do modelo relacional para os dois modelos não relacionais escolhidos.

I. Anexo 1 – *Script* de Migração dos Dados para MongoDB (*Python*)

```
import cx_Oracle
import json
from sshtunnel import SSHTunnelForwarder
import pymongo
import sys

# variables
MONGO_HOST = ('localhost', 2222)
MONGO_USER = "root"
MONGO_PASS = "uminho!2020"
MONGO_DB = "HR"
MONGO_COLLECTION1 = "employees"
MONGO_COLLECTION2 = "departments"

# define ssh tunnel
server = SSHTunnelForwarder(
    MONGO_HOST,
    ssh_username = MONGO_USER,
    ssh_password = MONGO_PASS,
    remote_bind_address = ('127.0.0.1', 27017)
)

employees = '''select
            JSON_OBJECT (
                'employee_id' VALUE e.employee_id,
                'first_name' VALUE e.first_name,
                'last_name' VALUE e.last_name,
                'email' VALUE e.email,
                'phone_number' VALUE e.phone_number,
                'hire_date' VALUE e.hire_date,
                'salary' VALUE e.salary,
                'commission_pct' VALUE e.commission_pct,
                'manager_id' VALUE e.manager_id,
                'job' VALUE (
                    SELECT JSON_OBJECT(
                        'job_id' VALUE j.job_id,
                        'job_title' VALUE j.job_title,
                        'min_salary' VALUE j.min_salary,
                        'max_salary' VALUE j.max_salary
                        )

                    from jobs j
                    where j.job_id = e.job_id
                ),
                'job_history' VALUE (
                    SELECT JSON_ARRAYAGG(
                        JSON_OBJECT(
                        'start_date' VALUE jh.start_date,
                        'end_date' VALUE jh.end_date,
                        'department_id' VALUE jh.department_id,
                        'jobs' VALUE (
                                SELECT JSON_OBJECT(
                                    'job_id' VALUE j.job_id,
                                    'job_title' VALUE j.job_title,
                                    'min_salary' VALUE j.min_salary,
                                    'max_salary' VALUE j.max_salary
                                    )
```

```
                                from jobs j
                                where j.job_id = jh.job_id
                            )
                        )
                    )
                    from job_history jh
                    where jh.employee_id = e.employee_id
                ),
                'department_id' VALUE e.department_id
            )
            from employees e'''

departments = '''select
            JSON_OBJECT (
                'department_id' VALUE d.department_id,
                'department_name' VALUE d.department_name,
                'manager_id' VALUE d.manager_id,
                'location' VALUE (
                    SELECT
                        JSON_OBJECT(
                        'location_id' VALUE l.location_id,
                        'street_adresss' VALUE l.street_address,
                        'postal_code' VALUE l.postal_code,
                        'city' VALUE l.city,
                        'state_province' VALUE l.state_province,
                        'country' VALUE (
                                SELECT JSON_OBJECT(
                                    'country_id' VALUE c.country_id,
                                    'country_name' VALUE c.country_name,
                                    'region' VALUE (
                                            SELECT JSON_OBJECT(
                                                'region_id' VALUE r.region_id,
                                                'region_name' VALUE r.region_name
                                                )
                                            from regions r
                                            where r.region_id = c.region_id
                                        )
                                    )
                                from countries c
                                where c.country_id = l.country_id
                            )
                        )

                    from locations l
                    where l.location_id = d.location_id
                )
            )
            from departments d'''

def oracle2Mongo(conn, collection, option, string):
    try:

        cur = conn.cursor()
        cur.execute(option)

        for row in cur:
            data = json.loads(row[0])
            collection.insert_one(data)

        print("Completed collection %s!" %(string))

    except Exception as err:
        error = "Error while inserting collection " + string + ": " + err
        print(error)
        sys.exit(1)
```

```
    finally:
        cur.close()

if __name__ == '__main__':

    try:
        # start ssh tunnel
        server.start()
        connection = pymongo.MongoClient('127.0.0.1', 27017)
        db = connection[MONGO_DB]
        collection1 = db[MONGO_COLLECTION1]
        collection2 = db[MONGO_COLLECTION2]

        try:
            cx_Oracle.init_oracle_client(lib_dir=r"C:\Users\Bruno-
Santos\Desktop\BDNSQL\BlackBoard\ficha2\instantclient_19_9")
            conn = cx_Oracle.connect("hr", "hr", "localhost:1521/orclpdb1.localdomain
", encoding="UTF-8")
            oracle2Mongo(conn, collection1, employees, "employees")
            oracle2Mongo(conn, collection2, departments, "departments")

        except Exception as err:
            print('Error while connecting to OracleDB: ', err)
            sys.exit(2)

        finally:
            conn.close()
            sys.exit(0)

    except  Exception as err:
            print('Error while connecting to MongoDB: ', err)
            sys.exit(3)
```

II. Anexo 2 – *Script* de Exportação de Dados para Migração em Neo4j (*Python*)

```
import requests
import cx_Oracle
import csv
import json
import sys

countries = '''SELECT country_id AS COUNTRY_ID,
                      country_name AS COUNTRY_NAME,
                      region_id AS REGION_ID
           FROM countries'''

regions = '''SELECT region_id AS REGION_ID,
                    region_name AS REGION_NAME
           FROM REGIONS'''

departments = '''SELECT department_id AS DEPARTMENT_ID,
                        department_name AS DEPARTMENT_NAME,
                        manager_id AS MANAGER_ID,
                        location_id AS LOCATION_ID
           FROM departments'''

employees = '''SELECT employee_id AS EMPLOYEE_ID,
                      first_name AS FIRST_NAME,
                      last_name AS LAST_NAME,
                      email AS EMAIL,
                      phone_number AS PHONE_NUMBER,
                      hire_date AS HIRE_DATE,
                      job_id AS JOB_ID,
                      salary AS SALARY,
                      commission_pct AS COMISSION_PCT,
                      manager_id AS MANAGER_ID,
                      department_id AS DEPARTMENT_ID
                FROM employees'''

job_history = '''SELECT employee_id AS EMPLOYEE_ID,
                        start_date AS START_DATE,
                        end_date AS END_DATE,
                        job_id AS JOB_ID,
                        department_id AS DEPARTMENT_ID
           FROM job_history'''

jobs = '''SELECT job_id AS JOB_ID,
                 job_title AS JOB_TITLE,
                 min_salary AS MIN_SALARY,
                 max_salary AS MAX_SALARY
           FROM jobs'''

locations = '''SELECT location_id AS LOCATION_ID,
                      street_address AS STREET_ADDRESS,
                      postal_code AS POSTAL_CODE,
                      city AS CITY,
                      state_province AS STATE_PROVINCE,
                      country_id AS COUNTRY_ID
           FROM locations'''

locations = '''SELECT location_id AS LOCATION_ID,
                      street_address AS STREET_ADDRESS,
                      postal_code AS POSTAL_CODE,
```

```
                    city AS CITY,
                    state_province AS STATE_PROVINCE,
                    country_id AS COUNTRY_ID
          FROM locations'''

departments_employees_jobs = '''
                              SELECT *
                              FROM employees
                              INNER JOIN departments ON employees.department_id=dep
artments.department_id
                              INNER JOIN jobs ON employees.job_id=jobs.job_id
                            '''

# join entre o job_history, department e jobs
# para tirar as informaçoes do job_history
job_history_departments_jobs = '''SELECT employee_id AS EMPLOYEE_ID,
                                         start_date AS START_DATE,
                                         end_date AS END_DATE,
                                         department_name AS DEPARTMENT_NAME,
                                         manager_id AS MANAGER_ID,
                                         job_title AS JOB_TITLE,
                                         min_salary AS MIN_SALARY,
                                         max_salary AS MAX_SALARY
                              FROM job_history
                              INNER JOIN departments  ON job_history.department_id=
departments.department_id
                              INNER JOIN jobs ON job_history.job_id=jobs.job_id
                            '''

manager = '''SELECT DISTINCT manager_id AS MANAGER_ID
             FROM employees
             WHERE manager_id IS NOT NULL
             '''

def oracle2csv(query, table, conn):

        try:
            cursor = conn.cursor()
            cursor.execute(query)
            name = "C:/Users/Bruno-Santos/AppData/Local/Neo4j/Relate/Data/dbmss/dbms-
0d045cb1-7d0a-46b0-a188-86b7a0841246/import/"+table+".csv"
            columns = [i[0] for i in cursor.description]
            rows = cursor.fetchall()
            # write oracle data to the csv file
            csv_file = open(name, mode='w')
            writer = csv.writer(csv_file, delimiter=',', lineterminator="\n", quoting
=csv.QUOTE_NONNUMERIC)
            # write column headers to csv file
            writer.writerow(columns)

            for row in rows:
                writer.writerow(row)   ## write rows to csv file

            csv_file.close()

        except Exception as err:
            print('Error while fill the file ' + table + ' ', err)

        else:
            print('complete '+table+'.csv file')

        finally:
            cursor.close()

if __name__ == '__main__':
```

```
    try:
        cx_Oracle.init_oracle_client(lib_dir=r"C:\Users\Bruno-
Santos\Desktop\BDNSQL\ficha2\instantclient_19_9")
        conn = cx_Oracle.connect("hr", "hr", "localhost:1521/orclpdb1.localdomain", e
ncoding="UTF-8")

        oracle2csv(countries,"countries", conn)
        oracle2csv(departments,"departments", conn)
        oracle2csv(employees,"employees", conn)
        oracle2csv(job_history,"job_history", conn)
        oracle2csv(jobs,"jobs", conn)
        oracle2csv(locations,"locations", conn)
        oracle2csv(regions,"regions", conn)
        oracle2csv(job_history_departments_jobs,"job_history_departments_jobs", conn)
        oracle2csv(manager,"manager", conn)

    except Exception as err:

        print('Error while creating the connection ', err)
        sys.exit(1)

    finally:
        conn.commit()
        conn.close()
```

III. Anexo 3 – *Script* de Importação de Dados para Migração em Neo4j (*Cypher*)

```
////////// 1. INSERIR INFORMACOES GERAIS. //////////////

// Informações dos paises
LOAD CSV WITH HEADERS FROM 'file:///countries.csv' AS row
CREATE (country:Country {idCountry: row.COUNTRY_ID})
SET country.nameCountry= row.COUNTRY_NAME,
    country.idRegion = row.REGION_ID
RETURN country;

// Informações das regioes
LOAD CSV WITH HEADERS FROM 'file:///regions.csv' AS row
CREATE (region:Region {nameRegion: row.REGION_NAME})
SET region.idRegion= row.REGION_ID
RETURN region;

// Informações dos departamentos
LOAD CSV WITH HEADERS FROM 'file:///departments.csv' AS row
CREATE (department:Department {idDepartment: row.DEPARTMENT_ID})
SET department.nameDepartmant= row.DEPARTMENT_NAME,
    department.idManager = row.MANAGER_ID,
    department.idLocation = row.LOCATION_ID
RETURN department;

// Informações dos funcionarios
LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
CREATE (employee:Employee {firstName: row.FIRST_NAME})
SET employee.lastName= row.LAST_NAME,
    employee.email = row.EMAIL,
    employee.phoneNumber = row.PHONE_NUMBER,
    employee.hireDate = row.HIRE_DATE,
    employee.idJob = row.JOB_ID,
    employee.salary = row.SALARY,
    employee.comission_pct = row.COMISSION_PCT,
    employee.idManager = row.MANAGER_ID,
    employee.idDepartment = row.DEPARTMENT_ID,
    employee.idEmployee = row.EMPLOYEE_ID
RETURN employee;

// Informações do historial dos trabalhos
LOAD CSV WITH HEADERS FROM 'file:///job_history_departments_jobs.csv' AS row
CREATE (job_history:Job_History {idEmployee: row.EMPLOYEE_ID})
SET job_history.department_name = row.DEPARTMENT_NAME,
    job_history.manager_id = row.MANAGER_ID,
    job_history.job_title= row.JOB_TITLE,
    job_history.min_salary = row.min_salary,
    job_history.max_salary = row.max_salary
RETURN job_history;

// Informações dos jobs
LOAD CSV WITH HEADERS FROM 'file:///jobs.csv' AS row
CREATE (job:Job {idJob: row.JOB_ID})
SET job.jobTitle= row.JOB_TITLE,
    job.minSalary = row.MIN_SALARY,
    job.maxSalary = row.MAX_SALARY
RETURN job;

// Informações das localizacoes
LOAD CSV WITH HEADERS FROM 'file:///locations.csv' AS row
```

```
CREATE (location:Location {idLocation: row.LOCATION_ID})
SET location.streerAddress= row.STREET_ADDRESS,
    location.postalCode = row.POSTAL_CODE,
    location.city = row.CITY,
    location.stateProvince = row.STATE_PROVINCE,
    location.idCountry = row.COUNTRY_ID
RETURN location;

////////// 2. INSERIR RELAÇÕES ENTRE NODOS. //////////////

// Relação Employee -> Job
// Relação Employee -> Department
LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
MATCH (employee:Employee {idEmployee: row.EMPLOYEE_ID})
MATCH (job:Job {idJob: row.JOB_ID})
MATCH (department:Department {idDepartment: row.DEPARTMENT_ID})
CREATE (employee)-[:TRABALHA_EM]->(job)
CREATE (employee)-[:PERTENCE_AO]->(department)
RETURN employee, job, department;

// Relação Employee -> job_history_departments_jobs
LOAD CSV WITH HEADERS FROM 'file:///employees.csv' AS row
MATCH (job_history:Job_History {idEmployee: row.EMPLOYEE_ID})
MATCH (employee:Employee {idEmployee: row.EMPLOYEE_ID})
MERGE (employee)-[t:TRABALHOU_EM]->(job_history)
ON CREATE SET t.startDate = row.START_DATE,
              t.endDate = row.END_DATE;

// Relação Department -> Locations
LOAD CSV WITH HEADERS FROM 'file:///locations.csv' AS row
MATCH (department:Department {idLocation: row.LOCATION_ID})
MATCH (location:Location {idLocation: row.LOCATION_ID})
CREATE (department)-[:SEDIADO_EM]->(location)
RETURN department,location;

// Relação locations -> countries
LOAD CSV WITH HEADERS FROM 'file:///countries.csv' AS row
MATCH (country:Country {idCountry: row.COUNTRY_ID})
MATCH (location:Location {idCountry: row.COUNTRY_ID})
CREATE (location)-[:LOCALIZADO_EM]->(country)
RETURN country,location;

// Relação countries -> regions
LOAD CSV WITH HEADERS FROM 'file:///regions.csv' AS row
MATCH (country:Country {idRegion: row.REGION_ID})
MATCH (region:Region {idRegion: row.REGION_ID})
CREATE (country)-[:CONTIDO_EM]->(region)
RETURN country,region;

// Relação employee -> manager
LOAD CSV WITH HEADERS FROM "file:///employees.csv" AS row
MATCH (employee:Employee {idEmployee: row.EMPLOYEE_ID})
MATCH (manager:Employee {idEmployee: row.MANAGER_ID})
MERGE (employee)-[:GERIDO_POR]->(manager);

// Relação department -> manager
LOAD CSV WITH HEADERS FROM "file:///departments.csv" AS row
MATCH (departments:Department {idDepartment: row.DEPARTMENT_ID})
MATCH (employee:Employee {idEmployee: row.MANAGER_ID})
MERGE (employee)-[:GERENTE_DO]->(departments);
```