



Universidade do Minho

Mestrado Integrado em Engenharia Informática

Unidade Curricular de Bases de Dados NOSQL

Ano Letivo de 2020/2021

Recursos Humanos

Eduardo Conceição(a83870),

Gonçalo Esteves(a85731),

Gonçalo Veloso(pg42830),

João Linhares(a86618),

Rui Oliveira(a83610),

Janeiro, 2021

Data de Recepção	
Responsável	
Avaliação	
Observações	

Recursos Humanos

Eduardo Conceição(a83870),
Gonçalo Esteves(a85731),
Gonçalo Veloso(pg42830),
João Linhares(a86618),
Rui Oliveira(a83610),

Janeiro, 2021

Resumo

Neste trabalho, realizado no âmbito da cadeira de Bases de Dados *NoSQL*, do primeiro ano do Mestrado em Engenharia Informática na Universidade do Minho, foi-nos proposta a implementação de dois sistemas de Bases de Dados não relacionais, baseadas no modelo de bases de dados de Recursos Humanos da *Oracle*.

Área de Aplicação: Desenho e Arquitetura de Bases de Dados, Conversão de Bases de Dados, Modelação de Bases de Dados Não Relacionais

Palavras-Chave: Bases de Dados Relacionais, Bases de Dados Não Relacionais, *MongoDB*, *Neo4J*, *Oracle Database*, Gestão de Recursos Humanos, *NoSQL*

Índice

RESUMO	I
ÍNDICE DE FIGURAS	IV
ÍNDICE DE TABELAS	V
1. INTRODUÇÃO	1
1.1. CONTEXTUALIZAÇÃO	1
1.2. APRESENTAÇÃO DO CASO DE ESTUDO	1
1.3. MOTIVAÇÃO E OBJETIVOS.....	1
1.4. ESTRUTURA DO RELATÓRIO	2
2. BASE DE DADOS RELACIONAL – ORACLE	3
2.1. <i>QUERIES</i>	4
2.1.1. <i>Encontrar todos os empregados que trabalham no departamento IT</i>	5
2.1.2. <i>Obter o número de trabalhadores por departamento</i>	5
2.1.3. <i>Top 5 trabalhadores com maior salário</i>	6
2.1.4. <i>Soma de todos os salários mensais</i>	6
2.1.5. <i>Soma de todos os salários por região</i>	6
2.1.6. <i>Salário médio em cada cargo</i>	7
2.1.7. <i>Número de departamentos por país com empregados a operar nestes</i>	7
2.1.8. <i>Encontrar os empregados que trabalham no departamento de Purchasing e quais os seus cargos</i>	8
3. BASE DE DADOS DOCUMENTAL – MONGODB	9
3.1. AGREGAÇÃO DE DADOS	9
3.2. MIGRAÇÃO DOS DADOS	11
3.2.1. <i>Exportação</i>	11
3.2.2. <i>Importação</i>	12
3.3. <i>QUERIES</i>	13
3.3.1. <i>Encontrar todos os empregados que trabalham no departamento IT</i>	14
3.3.2. <i>Obter o número de trabalhadores por departamento</i>	14
3.3.3. <i>Top 5 trabalhadores com maior salário</i>	15
3.3.4. <i>Soma de todos os salários mensais</i>	15
3.3.5. <i>Soma de todos os salários por região</i>	15
3.3.6. <i>Salário médio em cada cargo</i>	16
3.3.7. <i>Número de departamentos por país com empregados a operar nestes</i>	16
3.3.8. <i>Encontrar os empregados que trabalham no departamento de Purchasing e quais os seus cargos</i>	16

3.4. VANTAGENS	17
4. BASE DE DADOS ORIENTADA A GRAFOS – NEO4J	18
4.1. AGREGAÇÃO DE DADOS	18
4.2. MIGRAÇÃO DOS DADOS	19
4.1.1. <i>Exportação</i>	20
4.1.2. <i>Importação</i>	20
4.3. <i>QUERIES</i>	20
4.3.1. <i>Encontrar todos os empregados que trabalham no departamento IT</i>	21
4.3.2. <i>Obter o número de trabalhadores por departamento</i>	22
4.3.3. <i>Top 5 trabalhadores com maior salário</i>	22
4.3.4. <i>Soma de todos os salários mensais</i>	23
4.3.5. <i>Soma de todos os salários por região</i>	23
4.3.6. <i>Salário médio em cada cargo</i>	24
4.3.7. <i>Número de departamentos por país com empregados a operar nestes</i>	24
4.3.8. <i>Encontrar os empregados que trabalham no departamento de Purchasing e quais os seus cargos</i>	25
4.4. VANTAGENS	25
5. COMPARAÇÃO ENTRE AS VÁRIAS VERSÕES.....	27
6. CONCLUSÕES E TRABALHO FUTURO	29
REFERÊNCIAS	30
LISTA DE SIGLAS E ACRÓNIMOS	31
ANEXOS	32
I. QUERIES ORACLE DB	33
II. QUERIES MONGODB	35
III. QUERIES NEO4J.....	37
IV. SCRIPT PARA EXPORTAR OS DADOS PARA UM FICHEIRO “.JSON”	38
V. SCRIPT PARA IMPORTAR OS DADOS A PARTIR DE UM FICHEIRO CSV UTILIZANDO CYPHER.....	40

Índice de Figuras

Figura 1 - Esquema Lógico da base de dados HR	3
Figura 2 - Query 2.1.1. e o seu resultado com <i>Oracle Database</i>	5
Figura 3 - Query 2.1.2. e o seu resultado com <i>Oracle Database</i>	5
Figura 4 - Query 2.1.3. e o seu resultado com <i>Oracle Database</i>	6
Figura 5 - Query 2.1.4. e o seu resultado com <i>Oracle Database</i>	6
Figura 6 - Query 2.1.5. e o seu resultado com <i>Oracle Database</i>	6
Figura 7 - Query 2.1.6. e o seu resultado com <i>Oracle Database</i>	7
Figura 8 - Query 2.1.7. e o seu resultado com <i>Oracle Database</i>	7
Figura 9 - Query 2.1.8. e o seu resultado com <i>Oracle Database</i>	8
Figura 10 - Modelo do documento em formato <i>json</i>	10
Figura 11 - Gráfico do modelo do documento	11
Figura 12 - Query de criação de <i>jsons</i>	12
Figura 13 - Estatística da base de dados em <i>MongoDB</i>	13
Figura 14 - Exemplo de um documento em <i>MongoDB</i>	13
Figura 15 - Query 3.3.1. e o seu resultado com <i>MongoDB</i>	14
Figura 16 - Query 3.3.2. e o seu resultado com <i>MongoDB</i>	14
Figura 17 - Query 3.3.3. e o seu resultado com <i>MongoDB</i>	15
Figura 18 - Query 3.3.4. e o seu resultado com <i>MongoDB</i>	15
Figura 19 - Query 3.3.5. e o seu resultado com <i>MongoDB</i>	15
Figura 20 - Query 3.3.6. e o seu resultado com <i>MongoDB</i>	16
Figura 21 - Query 3.3.7. e o seu resultado com <i>MongoDB</i>	16
Figura 22 - Query 3.3.8. e o seu resultado com <i>MongoDB</i>	16
Figura 23 - Gráfico da BD em <i>Neo4J</i>	19
Figura 24 - Resultado da Query 4.3.1. em <i>Neo4J</i>	21
Figura 25 - Resultado da Query 4.3.2. em <i>Neo4J</i>	22
Figura 26 - Resultado da Query 4.3.3. em <i>Neo4J</i>	22
Figura 27 - Resultado da Query 4.3.4. em <i>Neo4J</i>	23
Figura 28 - Resultado da Query 4.3.5. em <i>Neo4J</i>	23
Figura 29 - Resultado da Query 4.3.6. em <i>Neo4J</i>	24
Figura 30 - Resultado da Query 4.3.7. em <i>Neo4J</i>	24
Figura 31 - Resultado da Query 4.3.8. em <i>Neo4J</i>	25

Índice de Tabelas

Não foi encontrada nenhuma entrada do índice de ilustrações.

1. Introdução

Nesta secção iremos apresentar uma contextualização do trabalho que nos foi proposto e explicar sucintamente cada parte deste relatório.

1.1. Contextualização

Começemos por uma pequena contextualização do problema. Neste trabalho tivemos que converter uma base de dados relacional em duas bases de dados *NoSQL*, ou não relacionais, bases de dados estas que teriam de ser uma documental, e uma gráfica. O problema em questão ronda à volta de um sistema de gestão de Recursos Humanos, que tem como objetivo guardar as informações de empregados de uma organização.

1.2. Apresentação do Caso de Estudo

O sistema em questão deverá armazenar informação relativa aos empregados de uma dada instituição, sendo, portanto, um SGBD orientado a recursos humanos. O sistema deverá estar organizado de forma tal que deveremos ter informação sobre os dados dos empregados, dos departamentos em que os empregados podem trabalhar, as regiões (ex.: América, Ásia, África, etc.), os países e localizações em que a empresa atua/já atuou, bem como os vários empregos que um empregado pode exercer e os que realmente já exerceu. Tudo isto virá de um sistema implementado numa base de dados relacional, e o requisito principal é converter o mesmo num sistema não relacional.

1.3. Motivação e Objetivos

O principal objetivo deste trabalho, feito no âmbito da cadeira de Bases de Dados *NoSQL*, do primeiro ano de Mestrado em Engenharia Informática na Universidade do Minho, será a modelação e implementação de sistemas de bases de dados não relacionais, e entender melhor o processo de conversão entre um sistema relacional e um não relacional, seja ele orientado a documentos ou a grafos.

1.4. Estrutura do Relatório

Nesta secção iremos expor a estrutura geral do nosso relatório. Em primeiro lugar, iremos explicar o sistema que nos foi fornecido, baseado num sistema da *Oracle*. De seguida, iremos expor quais os passos que tomamos para fazer a conversão para dois sistemas não relacionais: primeiro, um sistema orientado a documentos (*MongoDB*) e, em segundo lugar, um sistema orientado a grafos (*Neo4J*). Iremos também explicar a nossa razão para escolhermos estes sistemas, bem como explorar as várias vantagens que estes sistemas oferecem em relação ao modelo relacional que a *Oracle Database* segue. Para além disto, iremos também explicar algumas interrogações ao sistema que implementamos (*queries*), juntamente com uma comparação entre a sua implementação nos três tipos diferentes de bases de dados.

2. Base de Dados Relacional – Oracle

Com o objetivo de realizar um trabalho de análise, planeamento e implementação de SGBD não relacionais através da base de dados relacional HR foi necessário, em primeiro lugar, implementar a mesma num SGBD relacional.

Como esta base de dados é disponibilizada pela *Oracle Database* e como este SGBD foi lecionada nas aulas práticas foi utilizado este modelo relacional tanto para a criação da própria BD, o seu povoamento e, posteriormente, para migração de dados para os modelos não relacionais.

Foram fornecidos dois *scripts* da BD que foram corridos utilizando o *SQLDeveloper*: um deles cria o *tablespace* e um utilizador com permissões sobre este e o outro cria todas as tabelas, dados, sequências, índices e efetua o povoamento. O primeiro *script* (*1-database.sql*) foi corrido com o *user system* para criar o esquema HR e o utilizador HR, posteriormente foi corrido o segundo (*2-script.sql*) sobre o utilizador HR que criou os restantes objetos da BD e foi feito *commit* para tornar as alterações permanentes.

No final obtivemos a BD HR totalmente funcional com *Oracle Database* com o seguinte esquema lógico:

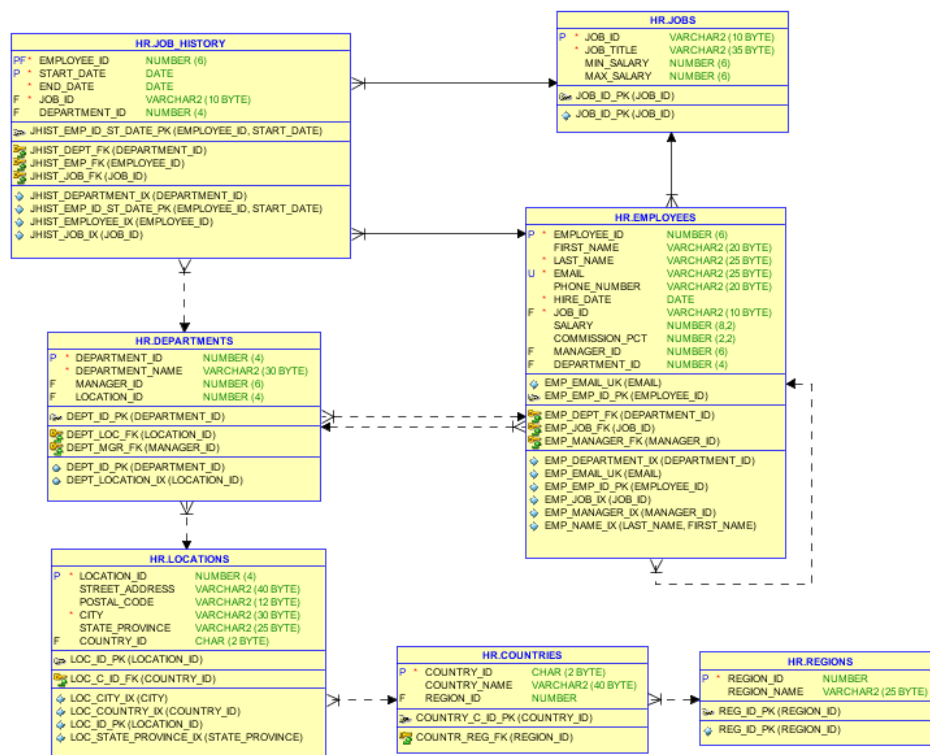


Figura 1 - Esquema Lógico da base de dados HR

Para facilitar a futura migração de dados para os modelos não relacionais foi feita uma análise do esquema da BD em questão, tal como as suas tabelas e o relacionamento entre estas.

O HR é um esquema de uma aplicação de Recursos Humanos, criado pela *Oracle*, que tem como objetivo principal armazenar os dados dos empregados de uma organização. Como se pode reparar na figura 1 o HR possui 7 tabelas:

- **EMPLOYEES:** Dados dos empregados, tais como: nome, contactos, departamento e cargo atual. Os empregados podem ou não estar vinculados a um departamento.
- **DEPARTAMENTOS:** Dados dos departamentos em que empregados podem trabalhar.
- **REGIONS:** Dados sobre as regiões em que a organização pode atuar. Ex.: América, Ásia.
- **LOCATIONS:** Dados sobre os locais ou endereços dos escritórios, armazéns ou locais de produção da organização.
- **COUNTRIES:** Dados sobre os países em que a organização atua.
- **JOBS:** Dados sobre os tipos de cargos que os empregados podem ocupar.
- **JOB_HISTORY:** Histórico dos cargos anteriores ocupados pelos empregados dentro da organização.

Uma vez estudadas as tabelas, foi necessário compreender as relações estabelecidas entre elas de forma a perceber como toda a informação se encontra associada através das chaves primárias e estrangeiras de cada tabela. O esquema lógico da BD (figura 1) foi bastante importante devido à facilidade em que as relações são representadas, o que nos levou a concluir que a tabela dos empregados está diretamente relacionada com a tabela dos cargos e do departamento e está também relacionada consigo mesma que corresponde ao gerente do mesmo empregado. A tabela de departamentos está novamente relacionada com a tabela *EMPLOYEES* que indica o gerente do departamento e também existe uma sequência de relações entre as tabelas *LOCATIONS*, *COUNTRIES* e *REGIONS* que fornecem a informação da localização que o departamento atua. Por fim existe a tabela *JOB_HISTORY* que se relaciona com *EMPLOYEES*, *JOBS* e *DEPARTAMENTOS* para guardar o histórico de um cargo posterior de um empregado.

2.1. Queries

Para podermos fazer alguns acessos ilustrativos à base de dados, é necessário estabelecer um pequeno número de interrogações a fazer, ditas *queries*. Para que possamos validar a nossa implementação em cada um dos tipos de bases de dados, e conveniente que as *queries* implementadas sejam as mesmas em cada um dos SGBDs. Como tal, optamos por implementar o seguinte conjunto de interrogações:

1. Encontrar todos os empregados que trabalham no departamento *IT*

2. Obter o número de trabalhadores por departamento
3. Top 5 trabalhadores com maior salário
4. Soma de todos os salários mensais
5. Soma de todos os salários por região
6. Salário médio em cada cargo
7. Número de departamentos por país com empregados a operar nestes
8. Encontrar os empregados que trabalham no departamento de *Purchasing* e quais os seus cargos

2.1.1. Encontrar todos os empregados que trabalham no departamento *IT*

```
-- Encontrar todos os empregados que trabalham no departamento IT

select e.first_name "Primeiro Nome", e.last_name "Último Nome" from employees e
join departments d on e.department_id = d.department_id
where d.department_name = 'IT';
```

	Primeiro Nome	Último Nome
1	Alexander	Hunold
2	Bruce	Ernst
3	David	Austin
4	Valli	Pataballa
5	Diana	Lorentz

Figura 2 - Query 2.1.1. e o seu resultado com *Oracle Database*

2.1.2. Obter o número de trabalhadores por departamento

```
-- Obter o número de trabalhadores por departamento

select count(e.employee_id) "Número de Empregados", d.department_name "Departamento" from employees e
join departments d on e.department_id = d.department_id
group by d.department_name;
```

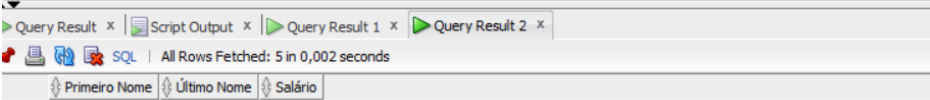
	Número de Empregados	Departamento
1	1	Administration
2	2	Accounting
3	6	Purchasing
4	1	Human Resources
5	5	IT
6	1	Public Relations
7	3	Executive
8	45	Shipping
9	34	Sales
10	6	Finance
11	2	Marketing

Figura 3 - Query 2.1.2. e o seu resultado com *Oracle Database*

2.1.3. Top 5 trabalhadores com maior salário

```
-- Top 5 trabalhadores com maior salário

select e.first_name "Primeiro Nome", e.last_name "Último Nome",e.salary "Salário" from employees e
order by e.salary DESC
FETCH FIRST 5 ROWS ONLY;
```



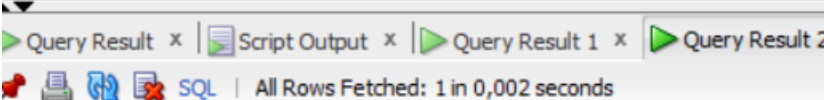
	Primeiro Nome	Último Nome	Salário
1	Steven	King	24000
2	Neena	Kochhar	17000
3	Lex	De Haan	17000
4	John	Russell	14000
5	Karen	Partners	13500

Figura 4 - Query 2.1.3. e o seu resultado com Oracle Database

2.1.4. Soma de todos os salários mensais

```
-- Soma de todos os salários mensais

select sum(e.salary) Total from employees e;
```



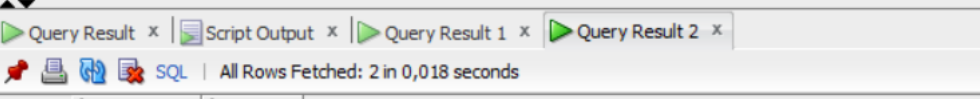
	TOTAL
1	691416

Figura 5 - Query 2.1.4. e o seu resultado com Oracle Database

2.1.5. Soma de todos os salários por região

```
-- Soma de todos os salários por região

select sum(e.salary) "Total Salário", r.region_name "Região" from employees e
join departments d on e.department_id = d.department_id
join locations l on l.location_id = d.location_id
left join countries c on c.country_id = l.country_id
left join regions r on r.region_id = c.region_id
group by r.region_name;
```

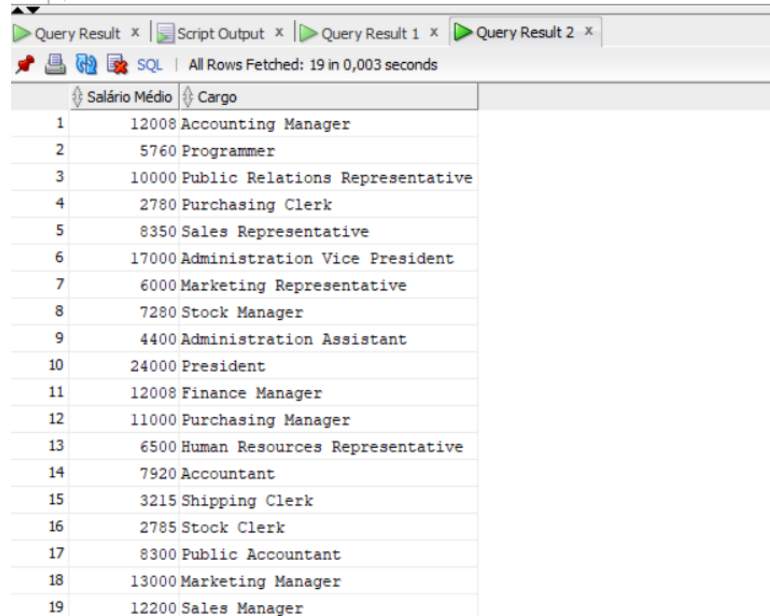


	Total Salário	Região
1	321000	Europe
2	363416	Americas

Figura 6 - Query 2.1.5. e o seu resultado com Oracle Database

2.1.6. Salário médio em cada cargo

```
-- Salário médio em cada cargo
select avg(e.salary) "Salário Médio", j.job_title "Cargo" from employees e
join jobs j on j.job_id = e.job_id
group by j.job_title;
```



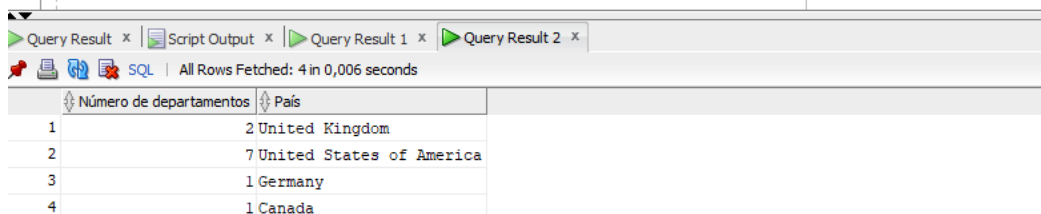
The screenshot shows the Oracle SQL Developer interface. The top pane contains the SQL query. The bottom pane shows the query result with two columns: 'Salário Médio' and 'Cargo'. The results are listed in a table with 19 rows.

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

Figura 7 - Query 2.1.6. e o seu resultado com Oracle Database

2.1.7. Número de departamentos por país com empregados a operar nestes

```
-- Número de departamentos por país com empregados a operar nestes
select count(distinct(d.department_id)) "Número de departamentos", c.country_name "País" from employees e
join departments d on e.department_id = d.department_id
join locations l on l.location_id = d.location_id
left join countries c on c.country_id = l.country_id
group by c.country_name;
```



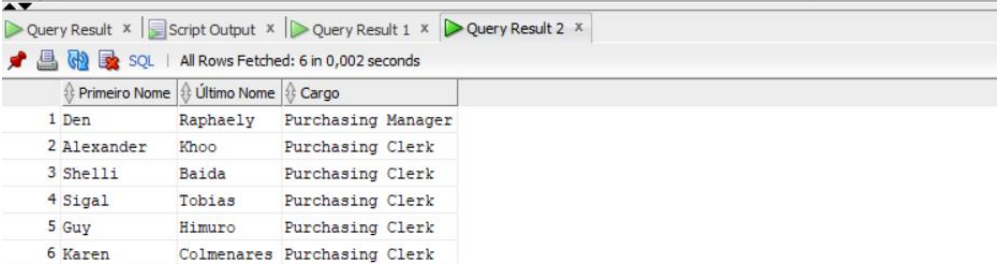
The screenshot shows the Oracle SQL Developer interface. The top pane contains the SQL query. The bottom pane shows the query result with two columns: 'Número de departamentos' and 'País'. The results are listed in a table with 4 rows.

	Número de departamentos	País
1	2	United Kingdom
2	7	United States of America
3	1	Germany
4	1	Canada

Figura 8 - Query 2.1.7. e o seu resultado com Oracle Database

2.1.8. Encontrar os empregados que trabalham no departamento de *Purchasing* e quais os seus cargos

```
-- Encontrar os empregados que trabalham no departamento de Purchasing e quais as suas profissões
select e.first_name "Primeiro Nome", e.last_name "Último Nome", j.job_title "Cargo" from employees e
join departments d on e.department_id = d.department_id
join jobs j on j.job_id = e.job_id
where d.department_name = 'Purchasing';
```



The screenshot shows the Oracle SQL Developer interface. The top pane contains the SQL query. The bottom pane shows the query result with columns 'Primeiro Nome', 'Último Nome', and 'Cargo'. The result is a table with 6 rows of employee data from the Purchasing department.

	Primeiro Nome	Último Nome	Cargo
1	Den	Raphaely	Purchasing Manager
2	Alexander	Khoo	Purchasing Clerk
3	Shelli	Baida	Purchasing Clerk
4	Sigal	Tobias	Purchasing Clerk
5	Guy	Himuro	Purchasing Clerk
6	Karen	Colmenares	Purchasing Clerk

Figura 9 - Query 2.1.8. e o seu resultado com *Oracle Database*

3. Base de Dados Documental – *MongoDB*

Um dos paradigmas *NoSQL* ao qual teríamos de implementar a BD HR da *Oracle* é o paradigma documental ao qual foi utilizado o *MongoDB* durante este trabalho prático.

Foi escolhido o *MongoDB* pois apesar de ser a SGBD mais utilizada nas aulas práticas para este paradigma, apresenta várias vantagens como alta flexibilidade nos documentos, fornece fragmentação automática para escala horizontal e suporta alta disponibilidade em *racks* e centros de dados.

As principais vantagens em trocar um sistema de base de dados relacional por uma solução *NoSQL*, em particular uma base de dados de armazenamento de documentos, são a escalabilidade e os relacionamentos.

Como este não possui nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade porquanto podemos simplesmente carregar todos os dados sem usar qualquer esquema predefinido.

A ideia do modelo documental é que todos os dados associados a um registo são armazenados no mesmo documento, a necessidade de estabelecer um relacionamento no modelo documental não tem o mesmo impacto que numa base de dados relacional, isto resultará em repetição de dados, porém essa repetição não é crítica pois não implica uma redução na performance.

Outro fator fundamental do sucesso desse modelo é a sua disponibilidade. A grande distribuição dos dados leva a que um maior número de solicitações aos dados seja executado pelo sistema num menor intervalo de tempo.

3.1. Agregação de dados

Antes de efetuar qualquer migração dos dados para o *MongoDB* foi primeiro necessário estudar e compreender o modelo relacional que pretendemos efetuar a migração como os seus dados, como são guardados e o objetivo da própria BD. Esta análise foi fundamental para podermos executar o processo de adaptação de um modelo relacional para um modelo orientado a documentos.

Como o objetivo principal da BD HR é armazenar os dados dos empregados de uma organização foi decidido criar uma coleção em que cada um dos seus documentos vai conter toda a informação de um empregado de uma organização.

Devido ao facto de este paradigma ter como base os documentos, teve de ser feito um planeamento de que dados devem ser guardados e como agregar as várias tabelas em um só documento para representar a informação de um empregado.

Como cada documento representa a informação de cada funcionário foi apenas mantido o id deste sendo que os restantes ids utilizados no modelo relacional não fazem parte do esquema do documento. As restantes componentes exceto as chaves estrangeiras da tabela *EMPLOYEES* fazem parte do documento.

Para representar a informação do cargo foi criado um documento aninhado com a chave *job* que contém a informação toda da tabela *JOBS* exceto o id. Foi também criado outro documento aninhado para representar a informação do gerente, como o gerente também é um empregado e para não estar a fazer recursividade dos dados todos dele apenas foi guardado o seu id, os nomes e formas de contacto. Relativamente à informação do departamento foi criado outro documento aninhado que contém o nome, o id do seu gerente e outro documento aninhado dentro com a chave *location* que contém os campos de todas as tabelas *LOCATIONS*, *COUNTRIES* e *REGIONS* exceto os seus ids. Por fim, para representar a informação do histórico de cargos foi utilizada a chave *job_history* que representa um *array* de objetos em que cada um tem as datas de início e fim, a informação toda do cargo e do departamento antigos tal como os documentos aninhados no documento principal, pois como é um histórico estes elementos têm de ser estáticos no tempo e caso a informação dos departamentos ou cargos mude a informação do histórico é perdida.

No final obtemos o modelo que pode ser visto ou em formato *json* ou em gráfico nas figuras seguintes.

```
{
  "id": integer,
  "first_name": string,
  "last_name": string,
  "email": string,
  "phone_number": string,
  "hire_date": string,
  "job": {
    "title": string,
    "min_salary": integer,
    "max_salary": integer
  },
  "salary": integer,
  "manager": {
    "manager_id": integer,
    "manager_first_name": string,
    "manager_last_name": string,
    "manager_email": string,
    "manager_phone_number": string
  },
  "department": {
    "name": string,
    "manager_id": integer,
    "location": {
      "street_address": string,
      "postal_code": string,
      "city": string,
      "state_province": string,
      "country": string,
      "region": string
    }
  },
  "job_history": [{
    "start_date": string,
    "end_date": string,
    "department": {
      "name": string,
      "manager_id": integer,
      "location": {
        "street_address": string,
        "postal_code": string,
        "city": string,
        "state_province": string,
        "country": string,
        "region": string
      }
    },
    "job": {
      "title": string,
      "min_salary": integer,
      "max_salary": integer
    }
  }
}]
}
```

Figura 10 - Modelo do documento em formato *json*

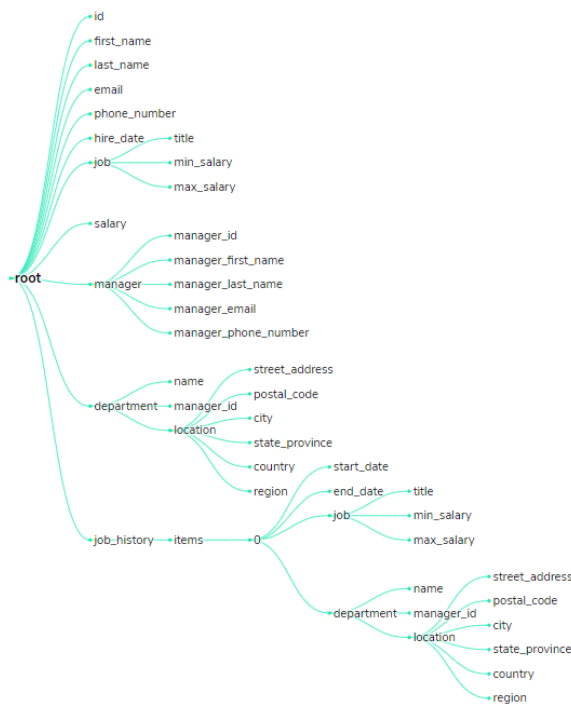


Figura 11 - Gráfico do modelo do documento

3.2. Migração dos Dados

A migração dos dados foi dividida em duas partes, a exportação dos dados e a importação dos mesmos para o *MongoDB*.

3.2.1. Exportação

Após ter definido o modelo de cada documento é necessário arranjar uma forma de obter os dados e exportá-los para o formato desejado. Este problema foi resolvido com apenas um *script SQL* chamado “*HR.sql*” em *Oracle SQL Developer* que contém uma *query* na *Oracle Database* que junta a informação toda no formato pretendido e no fim guarda o resultado obtido num ficheiro “*.json*” ao qual é um processo simples e rápido ao qual demorou apenas 0.15 s para gerar o ficheiro todo.

Este *script* envolve uma *query*, a qual pode ser vista na figura seguinte, com processos de *joins* nas tabelas para juntar a informação e criação de objetos *json* para guardar a informação nesse formato. Durante a criação desta *query* foi tomado o cuidado para caso uma chave esteja com valor a *null* ou caso um objeto não contenha informação nenhuma, a chave ou objeto não aparecerá no documento.

Para guardar os *jsons* num ficheiro foi utilizado o *spool* do *Oracle Database* que redireciona o *output* para um ficheiro chamado “*hr.json*” que vai ter um objeto em *json* para cada empregado. Foi necessário desativar algumas configurações para não obter espaços em branco e informação sem interesse no nosso ficheiro às quais foram ativadas novamente no fim do *script*.

```

select json_object(
  '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,
  'job' VALUE json_object('title' VALUE j.job_title, 'min_salary' VALUE j.min_salary, 'max_salary' VALUE j.max_salary FORMAT JSON ASSENT ON NULL),
  'salary' VALUE e.salary,
  'commission_pct' VALUE e.commission_pct,
  'manager' VALUE case when (m.employee_id IS NULL) then null
else json_object('manager_id' VALUE m.employee_id, 'manager_first_name' VALUE m.first_name, 'manager_last_name' VALUE m.last_name, 'manager_email' VALUE m.email, 'manager_phone_number' VALUE m.phone_number)
end FORMAT JSON,
  'department' VALUE case when (d.department_id IS NULL) THEN NULL
else json_object('name' VALUE d.department_name, 'manager_id' VALUE d.manager_id,
  'location' VALUE json_object('street_address' VALUE l.street_address,
    'postal_code' VALUE l.postal_code,
    'city' VALUE l.city,
    'state_province' VALUE l.state_province,
    'country' VALUE c.country_name,
    'region' VALUE r.region_name
    ASSENT ON NULL)
  end FORMAT JSON ASSENT ON NULL)
end FORMAT JSON,
  'job_history' VALUE (select json_arrayagg(
    json_object('start_date' VALUE jha.start_date, 'end_date' VALUE jha.end_date,
      'job' VALUE json_object('title' VALUE ja.job_title, 'min_salary' VALUE ja.min_salary, 'max_salary' VALUE ja.max_salary FORMAT JSON ASSENT ON NULL),
      'department' VALUE case when (da.department_id IS NULL) THEN NULL
      else json_object('name' VALUE da.department_name, 'manager_id' VALUE da.manager_id,
        'location' VALUE json_object('street_address' VALUE la.street_address,
          'postal_code' VALUE la.postal_code,
          'city' VALUE la.city,
          'state_province' VALUE la.state_province,
          'country' VALUE ca.country_name,
          'region' VALUE ra.region_name
          ASSENT ON NULL)
        end FORMAT JSON ASSENT ON NULL)
      end FORMAT JSON)) from job_history jha
    Inner Join jobs ja on jha.job_id = ja.job_id
    LEFT Join departments da on jha.department_id = da.department_id
    LEFT Join locations la on da.location_id = la.location_id
    LEFT Join countries ca on la.country_id = ca.country_id
    LEFT Join regions ra on ca.region_id = ra.region_id
    where e.employee_id = jha.employee_id)
  end FORMAT JSON
  ASSENT ON NULL
) from employees e
Inner Join jobs j on e.job_id = j.job_id
LEFT Join employees m on e.manager_id = m.employee_id
LEFT Join departments d on e.department_id = d.department_id
LEFT Join locations l on d.location_id = l.location_id
LEFT Join countries c on l.country_id = c.country_id
LEFT Join regions r on c.region_id = r.region_id;

```

Figura 12 - Query de criação de *jsons*

3.2.2. Importação

Já obtido o ficheiro “*hr.json*” falta apenas importa-lo para o *MongoDB*.

Em primeiro lugar foi transferido o ficheiro desde a nossa máquina para a máquina virtual que tem o *container* do *MongoDB* a correr. De seguida foi transferido o ficheiro “*hr.json*” desde a *VM* para o *container* através do comando ***docker cp hr.json <id-container-mongo>:/***. Por fim após aceder o *container* com o comando ***docker exec -it <id-container-mongo> bash*** foi utilizado o *mongoimport* para fazer a importação do ficheiro para o *MongoDB* ao qual basta introduzir na linha de comandos:

mongoimport --db hr --collection humanresources --drop --file hr.json

Após a importar os dados, como se pode ver na figura seguinte, tanto a coleção como os dados foram inseridos com sucesso. O objetivo de criar um documento para cada empregado

foi realizado com sucesso pois existem no total 107 entradas na tabela *EMPLOYEES* e na nossa coleção existem 107 objetos.

```
> db.stats()
{
  "db" : "hr",
  "collections" : 1,
  "views" : 0,
  "objects" : 107,
  "avgObjSize" : 685.5420560747664,
  "dataSize" : 73353,
  "storageSize" : 32768,
  "indexes" : 1,
  "indexSize" : 20480,
  "totalSize" : 53248,
  "scaleFactor" : 1,
  "fsUsedSize" : 5532782592,
  "fsTotalSize" : 28968488960,
  "ok" : 1
}
```

Figura 13 - Estatística da base de dados em *MongoDB*

```
> db.humanresources.findOne()
{
  "_id" : ObjectId("60133fd6e6e07518f7a6116a"),
  "id" : 206,
  "first_name" : "William",
  "last_name" : "Gietz",
  "email" : "WGIEZT",
  "phone_number" : "515.123.8181",
  "hire_date" : "2002-06-07T00:00:00",
  "job" : {
    "title" : "Public Accountant",
    "min_salary" : 4200,
    "max_salary" : 9000
  },
  "salary" : 8300,
  "manager" : {
    "manager_id" : 205,
    "manager_first_name" : "Shelley",
    "manager_last_name" : "Higgins",
    "manager_email" : "SHIGGINS",
    "manager_phone_number" : "515.123.8080"
  },
  "department" : {
    "name" : "Accounting",
    "manager_id" : 205,
    "location" : {
      "street_address" : "2004 Charade Rd",
      "postal_code" : "98199",
      "city" : "Seattle",
      "state_province" : "Washington",
      "country" : "United States of America",
      "region" : "Americas"
    }
  }
}
```

Figura 14 - Exemplo de um documento em *MongoDB*

3.3. Queries

O conjunto de *queries* implementadas para este SGBD orientado a documentos é o mesmo que para o modelo relacional, mas utilizando a linguagem de consulta própria do *MongoDB*, chamada *MongoDB Query Language*.

Nesta secção iremos mostrar os resultados obtidos com as *queries* em *MongoDB*, de modo a podermos fazer uma comparação entre os vários modelos de base de dados. Como vamos aferir, os resultados são iguais, o que implica uma implementação correta das interrogações e do modelo orientado a documentos.

3.3.1. Encontrar todos os empregados que trabalham no departamento *IT*

```
> db.humanresources.find({"department.name":"IT"},
... {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name"}
... )
{ "Primeiro Nome" : "Alexander", "Último Nome" : "Hunold" }
{ "Primeiro Nome" : "Bruce", "Último Nome" : "Ernst" }
{ "Primeiro Nome" : "Diana", "Último Nome" : "Lorentz" }
{ "Primeiro Nome" : "Valli", "Último Nome" : "Pataballa" }
{ "Primeiro Nome" : "David", "Último Nome" : "Austin" }
```

Figura 15 - Query 3.3.1. e o seu resultado com MongoDB

3.3.2. Obter o número de trabalhadores por departamento

```
> db.humanresources.aggregate([
... {"$match": {
... "department.name": {
... "$exists": true,
... "$ne": null
... }
... }},
... {"$group": {_id:"$department.name", "Número de Empregados":{$sum:1}}},
... {"$project": {_id: 0, "Departamento" : "$_id", "Número de Empregados": 1}}
... ])
{ "Número de Empregados" : 1, "Departamento" : "Administration" }
{ "Número de Empregados" : 6, "Departamento" : "Finance" }
{ "Número de Empregados" : 5, "Departamento" : "IT" }
{ "Número de Empregados" : 6, "Departamento" : "Purchasing" }
{ "Número de Empregados" : 34, "Departamento" : "Sales" }
{ "Número de Empregados" : 45, "Departamento" : "Shipping" }
{ "Número de Empregados" : 1, "Departamento" : "Human Resources" }
{ "Número de Empregados" : 3, "Departamento" : "Executive" }
{ "Número de Empregados" : 2, "Departamento" : "Marketing" }
{ "Número de Empregados" : 1, "Departamento" : "Public Relations" }
{ "Número de Empregados" : 2, "Departamento" : "Accounting" }
```

Figura 16 - Query 3.3.2. e o seu resultado com MongoDB

3.3.3. Top 5 trabalhadores com maior salário

```
> db.humanresources.find({
... {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name", "Salário": "$salary"}
... }).sort({salary:-1}).limit(5)
{ "Primeiro Nome" : "Steven", "Último Nome" : "King", "Salário" : 24000 }
{ "Primeiro Nome" : "Neena", "Último Nome" : "Kochhar", "Salário" : 17000 }
{ "Primeiro Nome" : "Lex", "Último Nome" : "De Haan", "Salário" : 17000 }
{ "Primeiro Nome" : "John", "Último Nome" : "Russell", "Salário" : 14000 }
{ "Primeiro Nome" : "Karen", "Último Nome" : "Partners", "Salário" : 13500 }
```

Figura 17 - Query 3.3.3. e o seu resultado com MongoDB

3.3.4. Soma de todos os salários mensais

```
> db.humanresources.aggregate([
... {"$group": { _id: null, "Total": {$sum: "$salary"} }},
... {"$project": { _id: 0, "Total": 1}}
... ])
{ "Total" : 691416 }
```

Figura 18 - Query 3.3.4. e o seu resultado com MongoDB

3.3.5. Soma de todos os salários por região

```
> db.humanresources.aggregate([
... {"$match": {
... "department.location.region": {
... "$exists": true,
... "$ne": null
... }
... }},
... {"$group": { _id: "$department.location.region", "Total Salário": {$sum: "$salary"} },
... {"$project": { _id: 0, "Região" : "$_id", "Total Salário": 1}}
... ])
{ "Total Salário" : 321000, "Região" : "Europe" }
{ "Total Salário" : 363416, "Região" : "Americas" }
```

Figura 19 - Query 3.3.5. e o seu resultado com MongoDB

3.3.6. Salário médio em cada cargo

```
> db.humanresources.aggregate([
... {"$group": {_id: "$job.title", "Salário Médio": {$avg: "$salary"}}},
... {"$project": {_id: 0, "Cargo": "$_id", "Salário Médio": 1}}
... ])
{ "Salário Médio" : 12008, "Cargo" : "Accounting Manager" }
{ "Salário Médio" : 12008, "Cargo" : "Finance Manager" }
{ "Salário Médio" : 11000, "Cargo" : "Purchasing Manager" }
{ "Salário Médio" : 2785, "Cargo" : "Stock Clerk" }
{ "Salário Médio" : 6500, "Cargo" : "Human Resources Representative" }
{ "Salário Médio" : 12200, "Cargo" : "Sales Manager" }
{ "Salário Médio" : 2780, "Cargo" : "Purchasing Clerk" }
{ "Salário Médio" : 10000, "Cargo" : "Public Relations Representative" }
{ "Salário Médio" : 8350, "Cargo" : "Sales Representative" }
{ "Salário Médio" : 7920, "Cargo" : "Accountant" }
{ "Salário Médio" : 3215, "Cargo" : "Shipping Clerk" }
{ "Salário Médio" : 5760, "Cargo" : "Programmer" }
{ "Salário Médio" : 7280, "Cargo" : "Stock Manager" }
{ "Salário Médio" : 24000, "Cargo" : "President" }
{ "Salário Médio" : 13000, "Cargo" : "Marketing Manager" }
{ "Salário Médio" : 17000, "Cargo" : "Administration Vice President" }
{ "Salário Médio" : 4400, "Cargo" : "Administration Assistant" }
{ "Salário Médio" : 6000, "Cargo" : "Marketing Representative" }
{ "Salário Médio" : 8300, "Cargo" : "Public Accountant" }
```

Figura 20 - Query 3.3.6. e o seu resultado com MongoDB

3.3.7. Número de departamentos por país com empregados a operar nestes

```
> db.humanresources.aggregate([
... {"$match": {
... "department.location.country": {
... "$exists": true,
... "$ne": null
... }
... }},
... {"$group": {_id: "$department.location.country", arrayset: {$addToSet: "$department.name"}}},
... {"$project": {_id: 0, "Número de Departamentos": {"$size": "$arrayset"}, "País": "$_id"}}
... ])
{ "Número de Departamentos" : 1, "País" : "Canada" }
{ "Número de Departamentos" : 7, "País" : "United States of America" }
{ "Número de Departamentos" : 2, "País" : "United Kingdom" }
{ "Número de Departamentos" : 1, "País" : "Germany" }
```

Figura 21 - Query 3.3.7. e o seu resultado com MongoDB

3.3.8. Encontrar os empregados que trabalham no departamento de *Purchasing* e quais os seus cargos

```
> db.humanresources.find({"department.name": "Purchasing"},
... {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name", "Cargo": "$job.title"}
... )
{ "Primeiro Nome" : "Shelli", "Último Nome" : "Baida", "Cargo" : "Purchasing Clerk" }
{ "Primeiro Nome" : "Alexander", "Último Nome" : "Khoo", "Cargo" : "Purchasing Clerk" }
{ "Primeiro Nome" : "Sigal", "Último Nome" : "Tobias", "Cargo" : "Purchasing Clerk" }
{ "Primeiro Nome" : "Guy", "Último Nome" : "Himuro", "Cargo" : "Purchasing Clerk" }
{ "Primeiro Nome" : "Karen", "Último Nome" : "Colmenares", "Cargo" : "Purchasing Clerk" }
{ "Primeiro Nome" : "Den", "Último Nome" : "Raphaely", "Cargo" : "Purchasing Manager" }
```

Figura 22 - Query 3.3.8. e o seu resultado com MongoDB

3.4. Vantagens

A mudança de um modelo relacional para um não relacional tem as suas vantagens. No entanto, nesta secção focar-nos-emos nas vantagens relativas à utilização de uma base de dados no modelo orientado a documentos, das quais podemos destacar as seguintes:

- **Flexibilidade** – Bases de Dados *NoSQL*, por regra, não apresentam esquema. Isto permite um desenvolvimento muito ágil do sistema de bases de dados, o que permite prototipagem de aplicações muito eficiente, e leva a que a estrutura dos dados na BD possa ser alterada conforme as necessidades no futuro.
- **Paginação** - A forma como os dados são consultados em *Document Stores* facilita a agregação de um certo número de dados, de modo a que a consulta a um sub-grupo de uma resposta a uma interrogação é muito mais simples, leve e rápido do que no caso das bases de dados relacionais.
- **Escalabilidade Horizontal** – Devido a não haver dependência entre os vários documentos, este sistema de base de dados permite-nos obter uma elevada escalabilidade horizontal que consiste em dividir o *dataset* e a sua carga por múltiplos servidores. Assim nos permite executar *queries* em paralelo ao que pode potencialmente fornecer uma melhor eficiência do que um único serviço de alta velocidade e alta capacidade como no caso de modelos relacionais. Com isto, ajuda no problema do futuro elevado crescimento de dados ao que basta adicionar servidores adicionais conforme necessário, o que pode ser um custo geral mais baixo do que uma melhoria no *hardware* de ponta para uma única máquina que teria que ser feito em modelos relacionais.

4. Base de Dados Orientada a Grafos – *Neo4j*

Nesta secção iremos explicar como procedemos na modelação da base de dados não relacional orientada a grafos, nomeadamente como proceder à agregação dos dados, à importação e exportação dos mesmos, como ficou o sistema no final e explicar as vantagens que este sistema tem em relação a outros.

4.1. Agregação de dados

Tal como aconteceu anteriormente na versão de MongoDB, foi necessário agrupar os dados relacionados por forma a utilizar mais eficientemente as capacidades que as bases de dados orientadas a grafos disponibilizam. No entanto, ao contrário do que aconteceu com a referida, esta agregação foi realizada após a importação dos dados para Neo4J.

Ao manter os diferentes *ids* que relacionavam os elementos da base de dados relacional, aquando da importação da informação, somos capazes de dotar o grafo gerado de relações, estabelecendo as mesmas a partir dos *ids*.

No caso dos nodos do tipo *COUNTRY*, apenas foi necessário relacionar cada um com o correspondente nodo *REGION*, por forma a definir a região em que cada país se encontra, utilizando para isso os *region_ids* encontrados em cada um e a relação *Located_In*.

Para os nodos *LOCATION* foi semelhante, relacionando, no entanto, cada um com um nó *COUNTRY*, simbolizando o país em que uma localização se encontra, utilizando para isso também a relação *Located_In* mas agora os *country_ids*.

Em relação aos nodos *DEPARTMENT*, foi necessário estabelecer não só a sua localização mas também o seu gerente. Para isso, utilizamos o *manager_id* de cada *DEPARTMENT* por forma a relacioná-lo com o *EMPLOYEE* cujo *employee_id* fosse igual, recorrendo à relação *Managed_By*. Quanto à localização, relacionou-se cada *DEPARTMENT* com a *LOCATION* correspondente, utilizando os *location_ids* e, novamente, a relação *Located_In*.

Passando para os nodos *EMPLOYEE*, era necessário relacioná-los com nodos *JOB*, *EMPLOYEE* e *DEPARTMENT*. Quanto aos primeiros, estabelecem a profissão exercida por cada funcionário, de tal forma que recorremos à relação *Works_As* para ligar nodos *EMPLOYEE* e *JOB* com *job_ids* semelhantes. Os segundos eram necessários para definir o patrão para quem trabalha cada funcionário, como tal, utilizamos a relação *Works_To* para relacionar *EMPLOYEES* cujo *manager_id* fosse igual ao *employee_id* de um *EMPLOYEE*. Por fim, era necessário estabelecer em que departamento trabalha um funcionário, e para isso utilizamos a relação *Works_In*, por forma a ligar nodos *EMPLOYEE* e *DEPARTMENT* em que os *department_id* fossem iguais.

Por último, os nodos *JOBHISTORY* relacionam-se com nodos *EMPLOYEE*, *JOB* e *DEPARTMENT*. Por forma a definir as profissões passadas de um trabalhador, utilizamos a relação *Worked_As* para ligar nodos *EMPLOYEE* e *JOBHISTORY* cujos *employee_ids* fossem iguais. Uma vez que cada profissão antiga era de um dado tipo e ocorria num determinado lugar,

ligou-se cada nodo *JOBHISTORY* a um nodo *Job* e um nodo *DEPARTMENT*, através das relações *Profession* (que se estabelecia quando os *job_ids* eram iguais) e *Place* (que se estabelecia quando os *department_ids* eram iguais), respetivamente.

Posto tudo isto, procedeu-se à remoção de todos os *ids* uma vez que a informação contida nestes era agora redundante.

O resultante aspeto da Base de Dados em *Neo4J* é o seguinte:

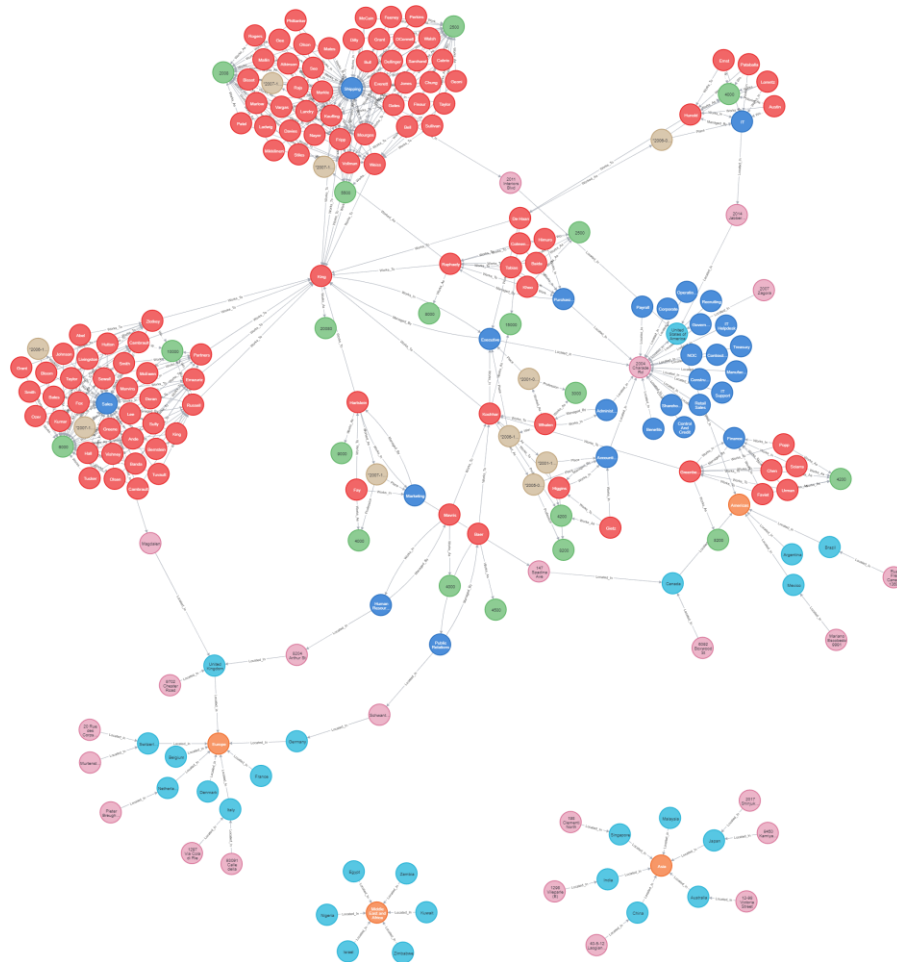


Figura 23 - Gráfico da BD em *Neo4J*

4.2. Migração dos Dados

Como ocorreu para a importação para *MongoDB*, a importação dos dados para *Neo4J* tem duas fases, uma de exportação dos dados e uma seguinte de importação para a base de dados não relacional.

4.1.1. Exportação

O primeiro passo consiste na exportação dos dados para um formato de texto com o qual seja conveniente trabalharmos. Para esse efeito, utilizamos a ferramenta de exportação para CSV do *SQLDeveloper*, que nos permite exportar os valores guardados nas tabelas para um ficheiro CSV. Para este efeito, cada uma das tabelas foi guardada no ficheiro CSV diferente, de modo a podermos distinguir cada um dos componentes na importação.

4.1.2. Importação

Os ficheiros produzidos vão depois ser importados para *Neo4J* utilizando um *script* escrito na linguagem de consulta deste tipo de base de dados: *Cypher*. Este *script* pode ser encontrado em anexo.

Ao fazer a importação para um modelo não relacional, optamos por remover os identificadores que eram utilizados como chave estrangeira. Esta decisão deveu-se ao facto que este tipo de informação é desnecessário neste tipo de bases de dados, uma vez que as ligações são feitas através dos nodos, o que torna estes *ids* inúteis.

4.3. Queries

O conjunto de *queries* implementadas para este último sistema de gestão de bases de dados é o mesmo que para os outros dois, mas utilizando a linguagem de consulta própria do *Neo4J*, chamada *Cypher*.

Nesta secção iremos mostrar os resultados obtidos com as *queries* em *Cypher*, de modo a podermos comparar com os resultados das outras anteriormente demonstradas. Como vamos aferir, os resultados são iguais, o que implica uma implementação correta das interrogações e do modelo orientado a grafos.

Em anexo poderemos encontrar o código fonte de cada uma.

4.3.1. Encontrar todos os empregados que trabalham no departamento *IT*

Employee
"Bruce Ernst"
"Alexander Hunold"
"Valli Pataballa"
"David Austin"
"Diana Lorentz"

Figura 24 - Resultado da Query 4.3.1. em Neo4J

4.3.2. Obter o número de trabalhadores por departamento

Department	NumberOfEmployees
"Administration"	1
"Marketing"	2
"Purchasing"	6
"Human Resources"	1
"Shipping"	45
"IT"	5
"Public Relations"	1
"Sales"	34
"Executive"	3
"Finance"	6
"Accounting"	2

Figura 25 - Resultado da Query 4.3.2. em Neo4J

4.3.3. Top 5 trabalhadores com maior salário

Employee	Job	Salary
"Steven King"	"President"	24000
"Neena Kochhar"	"Administration Vice President"	17000
"Lex De Haan"	"Administration Vice President"	17000
"John Russell"	"Sales Manager"	14000
"Karen Partners"	"Sales Manager"	13500

Figura 26 - Resultado da Query 4.3.3. em Neo4J

4.3.4. Soma de todos os salários mensais

NumberOfEmployees	TotalOfSalaries
107	691416

Figura 27 - Resultado da Query 4.3.4. em Neo4J

4.3.5. Soma de todos os salários por região

Region	NumberOfEmployees	TotalOfSalaries
"Europe"	36	321000
"Americas"	70	363416

Figura 28 - Resultado da Query 4.3.5. em Neo4J

4.3.6. Salário médio em cada cargo

Job	NumberOfEmployees	MinimumSalary	MaximumSalary	AverageSalary
"President"	1	24000	24000	24000.0
"Administration Vice President"	2	17000	17000	17000.0
"Marketing Manager"	1	13000	13000	13000.0
"Sales Manager"	5	10500	14000	12200.0
"Finance Manager"	1	12008	12008	12008.0
"Accounting Manager"	1	12008	12008	12008.0
"Purchasing Manager"	1	11000	11000	11000.0
"Public Relations Representative"	1	10000	10000	10000.0
"Sales Representative"	30	6100	11500	8350.0
"Public Accountant"	1	8300	8300	8300.0
"Accountant"	5	6900	9000	7920.0
"Stock Manager"	5	5800	8200	7280.0
"Human Resources Representative"	1	6500	6500	6500.0
"Marketing Representative"	1	6000	6000	6000.0
"Programmer"	5	4200	9000	5760.0
"Administration Assistant"	1	4400	4400	4400.0
"Shipping Clerk"	20	2500	4200	3214.9999999999995
"Stock Clerk"	20	2100	3600	2785.0
"Purchasing Clerk"	5	2500	3100	2780.0

Figura 29 - Resultado da Query 4.3.6. em Neo4J

4.3.7. Número de departamentos por país com empregados a operar nestes

Country	NumberOfDepartments
"United States of America"	7
"Canada"	1
"United Kingdom"	2
"Germany"	1

Figura 30 - Resultado da Query 4.3.7. em Neo4J

4.3.8. Encontrar os empregados que trabalham no departamento de *Purchasing* e quais os seus cargos

Employee	Job
"Alexander Khoo"	"Purchasing Clerk"
"Shelli Baida"	"Purchasing Clerk"
"Guy Himuro"	"Purchasing Clerk"
"Karen Colmenares"	"Purchasing Clerk"
"Sigal Tobias"	"Purchasing Clerk"
"Den Raphaely"	"Purchasing Manager"

Figura 31 - Resultado da Query 4.3.8. em Neo4J

4.4. Vantagens

Como aconteceu com a utilização de bases de dados documentais, a passagem de uma base de dados relacional para uma não relacional orientada a grafos ocorreu pois esta apresenta algumas vantagens, nomeadamente:

- **Performance** - Bases de dados baseadas em SQL mostram uma *performance* degradada quando temos consultas que utilizam muitas tabelas recursivamente, com operações pesadas como *JOINS*. Nas bases de dados orientadas a grafos, o mesmo não acontece porque, em regra, consultas recursivas vão aceder apenas a partes do grafo que estão próximas umas das outras, ou seja, acedem a um sub-grafo cujo diâmetro é muito menor do que o grafo que o contém.
- **Flexibilidade** - Uma característica que define o paradigma não relacional é a não existência de um *schema*. No modelo relacional, nós temos de criar uma estrutura rígida em que os dados inseridos têm de se enquadrar, o que dificulta o trabalho de modelação caso, no futuro, o modelo tenha de ser alterado por uma qualquer razão. Bases de dados orientadas a grafos, não se tendo de cingir por um esquema, apresentam uma estrutura

muito flexível que permite que surjam novos relacionamentos no futuro, o que acaba por facilitar o crescimento incremental de uma aplicação.

5. Comparação entre as várias Versões

O projeto consistiu em implementar a mesma base de dados em modelos diferentes. Isto permitiu-nos, no final, ter uma noção de que vantagens um modelo apresenta sobre os outros.

Em primeiro lugar, podemos comparar a versão original, em *Oracle Database*, que é uma implementação que utiliza o modelo relacional, com as duas implementações não relacionais de um modo muito genérico. O modelo não relacional apresenta as seguintes vantagens em relação ao modelo relacional:

- Em termos de **escalabilidade**, o modelo não relacional, em ambos os casos, mostra-se apto para trabalhar num ambiente tipo *cluster*, em que temos, não uma base de dados centralizada que pode escalar mais facilmente com o aumento da capacidade de *storage* num nodo de um sistema, mas sim um SGBD que pode mais facilmente ser distribuído por vários nodos de um SD, pelo que são em geral **horizontalmente escaláveis**.
- As bases de dados relacionais normalmente forçam a existência de um esquema, ou seja, uma estrutura fixa que os dados que são inseridos na BD terão de seguir. No entanto, o paradigma *NoSQL*, em particular o caso das bases de dados orientadas a grafos e a documentos não têm esta restrição, sendo denominadas **schemaless**. Isto permite uma enorme flexibilidade na adição de dados no futuro a uma base de dados, uma vez que, se estes não seguirem o mesmo tipo de estrutura que os anteriores, poderão ser adicionados na mesma sem comprometer a estrutura dos dados já inseridos.
- Estes tipos de bases de dados não apresentam relações explícitas entre os vários dados (daí a denominação de **não relacionais**). Isto leva a que certas operações de consulta que são consideradas computacionalmente pesadas, como *JOINS*, não sejam implementadas e as operações de consulta que existem são, de facto, mais leves e rápidas.
- Ambos SGBD são considerados **open-source** (ou, no mínimo, no caso do *MongoDB*, *source available*), o que leva potencialmente a inovação mais rápida e tecnologia mais flexível.

Tendo isto em conta, podemos definir as vantagens que justificaram o salto do modelo relacional para o seu contra-parte.

Para além das vantagens que definimos anteriormente em cada um dos dois sistemas não relacionais que utilizamos, podemos também expor algumas vantagens entre o modelo orientado a grafos e o orientado a documentos:

- Em *MongoDB*, o padrão que o sistema segue é baseado em JSON, que é um tipo de serialização de informação que é extremamente comum e com elevados níveis de suporte, o que leva a que seja mais fácil a aplicação dos dados guardados no mesmo, por exemplo, em aplicações que utilizam o estilo *REST*.

Em contrapartida, *Neo4J* não apresenta esta vantagem, uma vez que utiliza uma sintaxe própria para o armazenamento e representação de informação.

- Em contrapartida, um problema que *MongoDB* apresenta é a utilização de dados redundantes. Isto deve-se ao facto que, por estarmos a trabalhar com um sistema baseado em JSON, não existem formas óbvias de tratar da informação que se encontrava nas relações entre os elementos na versão *SQL*. No entanto, esta replicação de dados não acontece em *Neo4J*, uma vez que, no grafo que constitui a BD, as relações mantêm-se de certa forma, através das ligações entre os vários nodos do grafo.

6. Conclusões e Trabalho Futuro

Através da análise do esquema relacional da base de dados HR, fomos capazes de realizar a conversão dos dados de um sistema relacional para sistemas não relacionais como *MongoDB* e *Neo4j*. Este processo foi constituído, para cada sistema não relacional, por um processo de exportação dos dados nos formatos pretendidos e pela importação dos mesmos nos seguintes paradigmas. Seguidamente, a execução de *queries* nos dados de modo a mostrar os mesmos e a demonstrar que a conversão para vários paradigmas não afetou a qualidade, consistência e veracidade dos dados foi realizada com sucesso.

Deste modo, a interpretação dos vários paradigmas foi executada com sucesso, tornando capaz, também, uma análise dos vários modelos e das suas vantagens.

Referências

<https://docs.oracle.com/en/database/oracle/oracle-database/index.html>

<https://docs.mongodb.com/guides/>

<https://neo4j.com/developer/get-started/>

Lista de Siglas e Acrónimos

BD	Base de Dados
HR	<i>Human Resources</i>
SGBD	Sistema de Gestão de Base de Dados
SQL	<i>Structured Query Language</i>
NoSQL	<i>Not only SQL</i>
VM	<i>Virtual Machine</i>
CSV	<i>Comma Separated Values</i>
JSON	<i>JavaScript Oriented Notation</i>
REST	<i>Representational State Transfer</i>
SD	Sistema Distribuído

Anexos

I. <i>QUERIES ORACLE DB</i>	33
II. <i>QUERIES MONGODB</i>	35
III. <i>QUERIES NEO4J</i>.....	37
IV. <i>SCRIPT PARA EXPORTAR OS DADOS PARA UM FICHEIRO “.JSON”</i>	38
V. <i>SCRIPT PARA IMPORTAR OS DADOS A PARTIR DE M FICHEIRO CSV UTILIZANDO CYPHER</i>	40

I. Queries Oracle DB

-- Encontrar todos os empregados que trabalham no departamento IT

```
select e.first_name "Primeiro Nome", e.last_name "Último Nome" from employees e
join departments d on e.department_id = d.department_id
where d.department_name = 'IT';
```

-- Obter o número de trabalhadores por departamento

```
select count(e.employee_id) "Número de Empregados", d.department_name "Departamento" from employees e
join departments d on e.department_id = d.department_id
group by d.department_name;
```

-- Top 5 trabalhadores com maior salário

```
select e.first_name "Primeiro Nome", e.last_name "Último Nome", e.salary "Salário" from employees e
order by e.salary DESC
FETCH FIRST 5 ROWS ONLY;
```

-- Soma de todos os salários mensais

```
select sum(e.salary) Total from employees e;
```

-- Soma de todos os salários por região

```
select sum(e.salary) "Total Salário", r.region_name "Região" from employees e
join departments d on e.department_id = d.department_id
join locations l on l.location_id = d.location_id
left join countries c on c.country_id = l.country_id
left join regions r on r.region_id = c.region_id
group by r.region_name;
```

-- Salário médio em cada cargo

```
select avg(e.salary) "Salário Médio", j.job_title "Cargo" from employees e
join jobs j on j.job_id = e.job_id
group by j.job_title;
```

-- Número de departamentos por país

```
select count(distinct(d.department_id)) "Número de departamentos", c.country_name "País" from employees e
join departments d on e.department_id = d.department_id
join locations l on l.location_id = d.location_id
left join countries c on c.country_id = l.country_id
group by c.country_name;
```

-- Encontrar os empregados que trabalham no departamento de Purchasing e quais os seus cargos

```
select e.first_name "Primeiro Nome", e.last_name "Último Nome", j.job_title "Cargo" from employees e
join departments d on e.department_id = d.department_id
join jobs j on j.job_id = e.job_id
```

```
where d.department_name = 'Purchasing';
```

II. Queries MongoDB

Encontrar todos os empregados que trabalham no departamento IT

...

```
db.humanresources.find({"department.name":"IT"},
    {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name"}
)
```

...

Obter o número de trabalhadores por departamento

...

```
db.humanresources.aggregate([
    {"$match": {
        "department.name": {
            "$exists": true,
            "$ne": null
        }
    }},
    {"$group": {_id:"$department.name", "Número de Empregados":{$sum:1}}},
    {"$project": {_id: 0, "Departamento" : "$_id", "Número de Empregados": 1}}
])
```

...

Top 5 trabalhadores com maior salário

...

```
db.humanresources.find({},
    {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name", "Salário": "$salary"}
).sort({salary:-1}).limit(5)
```

...

Soma de todos os salários mensais

...

```
db.humanresources.aggregate([
    {"$group": { _id: null,"Total": {$sum: "$salary"}}},
    {"$project": { _id: 0, "Total": 1}}
])
```

...

Soma de todos os salários por região

...

```
db.humanresources.aggregate([
    {"$match": {
        "department.location.region": {
            "$exists": true,
            "$ne": null
        }
    }},
    {"$group": {_id:"$department.location.region", "Total Salário":{$sum: "$salary"}}},
    {"$project": {_id: 0, "Região" : "$_id", "Total Salário": 1}}
])
```

...

Salário médio em cada cargo

...

```
db.humanresources.aggregate([
    {"$group": {_id:"$job.title", "Salário Médio":{$avg: "$salary"}}},
```

```

        {"$project": {_id: 0, "Cargo": "$_id", "Salário Médio": 1}}
    ])
    ...

Número de departamentos por país
...

db.humanresources.aggregate([
    {"$match": {
        "department.location.country": {
            "$exists": true,
            "$ne": null
        }
    }},
    {"$group": {_id: "$department.location.country", arrayset: {"$addToSet": "$department.name"}}},
    {"$project": {_id: 0, "Número de Departamentos": {"$size": "$arrayset"}, "País": "$_id"}}
])
...

Encontrar os empregados que trabalham no departamento de Purchasing e quais os seus cargos
...

db.humanresources.find({"department.name": "Purchasing"},
    {_id: 0, "Primeiro Nome": "$first_name", "Último Nome": "$last_name", "Cargo": "$job.title"}
)
...

```

III. Queries Neo4J

```
//Encontrar toda a gente que trabalha no departamento de IT
match (e:Employee)-[:Works_In]->(d:Department)
where d.department_name = "IT"
return (e.first_name + " " + e.last_name) as Employee;
```

```
//Número de trabalhadores por departamento:
match (e:Employee)-[:Works_In]->(d:Department)
return d.department_name As Department, count(r) as NumberOfEmployees;
```

```
//Top 5 com maiores salarios:
match (e:Employee)-[:Works_As]->(j:Job)
return (e.first_name + " " + e.last_name) as Employee, j.job_title as Job, e.salary as Salary
order by Salary desc limit 5;
```

```
//Soma de todos os salários:
match (e:Employee)
return count(e) as NumberOfEmployees, sum(e.salary) as TotalOfSalaries;
```

```
//Soma salários por região:
match (e:Employee)-[:Works_In]->(d:Department)-[:Located_In]->(l:Location)-[:Located_In]->(c:Country)-[:Located_In]->(r:Region)
return r.region_name as Region, count(e) as NumberOfEmployees, sum(e.salary) as TotalOfSalaries;
```

```
//Salario medio por profissao
match (e:Employee)-[:Works_As]->(j:Job)
return j.job_title as Job, count(e) as NumberOfEmployees, min(e.salary) as MinimunSalary, max(e.salary) as MaximumSalary, avg(e.salary) as AverageSalary
order by AverageSalary desc;
```

```
//Numero de departamentos por pais, com funcionários a operar nestes
match (:Employee)-[:Works_In]->(d:Department)-[:Located_In]->(l:Location)-[:Located_In]->(c:Country)
return c.country_name as Country, count(distinct d) as NumberOfDepartments;
```

```
//Quais as profissoes de quem trabalha no departamento de Purchasing
match (e:Employee)-[:Works_As]->(j:Job)
match (e:Employee)-[:Works_In]->(d:Department)
where d.department_name = "Purchasing"
return (e.first_name + " " + e.last_name) as Employee, j.job_title as Job;
```

IV. Script para exportar os dados para um ficheiro “.json”

```
set feedback off
set echo off
set heading off
set trimspool on
set newpage 0
set verify off
spool hr.json
select json_object(
    '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,
    'job' VALUE json_object('title' VALUE j.job_title, 'min_salary' VALUE j.min_salary, 'max_salary' VALUE j.max_salary
FORMAT JSON ABSENT ON NULL),
    'salary' VALUE e.salary,
    'commission_pct' VALUE e.commission_pct,
    'manager' VALUE case when (m.employee_id IS NULL) then null
    else json_object('manager_id' VALUE m.employee_id, 'manager_first_name' VALUE m.first_name,
'manager_last_name' VALUE m.last_name , 'manager_email' VALUE m.email, 'manager_phone_number' VALUE
m.phone_number)
    end FORMAT JSON,
    'department' VALUE case when (d.department_id IS NULL) THEN NULL
    else json_object('name' VALUE d.department_name, 'manager_id' VALUE d.manager_id,
        'location' VALUE json_object('street_address' VALUE l.street_address,
            'postal_code' VALUE l.postal_code,
            'city' VALUE l.city,
            'state_province' VALUE l.state_province,
            'country' VALUE c.country_name,
            'region' VALUE r.region_name
            ABSENT ON NULL)
        FORMAT JSON ABSENT ON NULL)
    end FORMAT JSON,
    'job_history' VALUE (select json_arrayagg(
        json_object('start_date' VALUE jha.start_date, 'end_date' VALUE jha.end_date,
            'job' VALUE json_object('title' VALUE ja.job_title, 'min_salary' VALUE ja.min_salary, 'max_salary' VALUE
ja.max_salary FORMAT JSON ABSENT ON NULL),
            'department' VALUE case when (da.department_id IS NULL) THEN NULL
            else json_object('name' VALUE da.department_name, 'manager_id' VALUE da.manager_id,
                'location' VALUE json_object('street_address' VALUE la.street_address,
                    'postal_code' VALUE la.postal_code,
                    'city' VALUE la.city,
                    'state_province' VALUE la.state_province,
                    'country' VALUE ca.country_name,
                    'region' VALUE ra.region_name
                    ABSENT ON NULL)
                FORMAT JSON ABSENT ON NULL)
            end FORMAT JSON)) from job_history jha
```

```

        Inner Join jobs ja on jha.job_id = ja.job_id
        LEFT Join departments da on jha.department_id = da.department_id
        LEFT Join locations la on da.location_id = la.location_id
        LEFT Join countries ca on la.country_id = ca.country_id
        LEFT Join regions ra on ca.region_id = ra.region_id
        where e.employee_id = jha.employee_id)
    FORMAT JSON
    ABSENT ON NULL
) from employees e
Inner Join jobs j on e.job_id = j.job_id
LEFT Join employees m on e.manager_id = m.employee_id
LEFT Join departments d on e.department_id = d.department_id
LEFT Join locations l on d.location_id = l.location_id
LEFT Join countries c on l.country_id = c.country_id
LEFT Join regions r on c.region_id = r.region_id;
spool off;
set feedback on
set echo on
set heading on
set trimspool off
set newpage none
set verify on

```

V. *Script* para importar os dados a partir de um ficheiro CSV utilizando *Cypher*

```
LOAD CSV FROM 'file:///REGIONS_DATA_TABLE.csv' AS line
CREATE(r: Region {region_id: toInteger(line[0]), region_name: line[1]});
```

```
DROP INDEX ON :Region(region_id);
CREATE INDEX FOR (n:Region) ON (n.region_id);
```

```
LOAD CSV FROM 'file:///COUNTRIES_DATA_TABLE.csv' AS line
CREATE(c: Country {country_id: line[0], country_name: line[1], region_id: toInteger(line[2])});
```

```
DROP INDEX ON :Country(country_id);
CREATE INDEX FOR (n:Country) ON (n.country_id);
```

```
LOAD CSV FROM 'file:///LOCATIONS_DATA_TABLE.csv' AS line
CREATE(l: Location {location_id: toInteger(line[0]), street_address: line[1], postal_code: line[2], city: line[3],
state_province: line[4], country_id: line[5]});
```

```
DROP INDEX ON :Location(location_id);
CREATE INDEX FOR (n:Location) ON (n.location_id);
```

```
LOAD CSV FROM 'file:///DEPARTMENTS_DATA_TABLE.csv' AS line
CREATE(d: Department {department_id: toInteger(line[0]), department_name: line[1], manager_id:
toInteger(line[2]), location_id: toInteger(line[3])});
```

```
DROP INDEX ON :Department(department_id);
CREATE INDEX FOR (n:Department) ON (n.department_id);
```

```
LOAD CSV FROM 'file:///EMPLOYEES_DATA_TABLE.csv' AS line
CREATE(e: Employee {employee_id: toInteger(line[0]), first_name: line[1], last_name: line[2], email: line[3],
phone_number: line[4], hire_date: date(line[5]), job_id: line[6], salary: toInteger(line[7]), commission_pct:
toFloat(line[8]), manager_id: toInteger(line[9]), department_id: toInteger(line[10])});
```

```
DROP INDEX ON :Employee(employee_id);
CREATE INDEX FOR (n:Employee) ON (n.employee_id);
```



```
LOAD CSV FROM 'file:///JOB_HISTORY_DATA_TABLE.csv' AS line
CREATE(jh: JobHistory {employee_id: toInteger(line[0]), start_date: date(line[1]), end_date: date(line[2]),
job_id: line[3], department_id: toInteger(line[4])});
```

```
LOAD CSV FROM 'file:///JOBS_DATA_TABLE.csv' AS line
CREATE(j: Job {job_id: line[0], job_title: line[1], min_salary: toInteger(line[2]), max_salary: toInteger(line[3])});
```

```
DROP INDEX ON :job(job_id);
CREATE INDEX FOR (n:Job) ON (n.job_id);
```