

Universidade do Minho
Mestrado em Engenharia Informática

NoSQL

GRUPO:

Carlos Miguel Luzia de Carvalho - PG47092

Rui Emanuel Gomes Vieira - PG47635

Francisco Correia Franco - PG

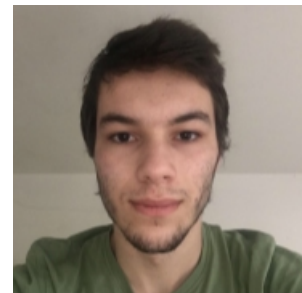
18 Abril 2022



PG47092



PG47635



PG47687

Índice

1	Introdução	1
2	Base de Dados Relacional	3
2.1	Queries	3
2.1.1	Query - Empregado com maior salário	3
2.1.2	Query - Departamento com mais Empregados	4
2.1.3	Query - Empregados que trabalharam em mais do que Departamento	4
2.1.4	Query - Diferentes departamentos na Europa	5
3	Base de Dados Documental	6
3.1	Criação de Collections	6
3.1.1	EMPLOYEES	6
3.1.2	DEPARTMENTS	7
3.1.3	JOB_HISTORY	8
3.2	Queries	8
3.2.1	Query - Empregado com maior salário	9
3.2.2	Query - Departamento com mais Empregados	9
3.2.3	Query - Empregados que trabalharam em mais do que Departamento	9
3.2.4	Query - Diferentes departamentos na Europa	9
4	Base de Dados Orientada a Grafos	10
4.1	Import dos Dados	10
4.2	Criação das relações	10
4.2.1	Relações Job_History	12
4.3	Queries	14
4.3.1	Query - Empregado com maior salário	14
4.3.2	Query - Departamento com mais Empregados	14
4.3.3	Query - Empregados que trabalharam em mais do que Departamento	15
4.3.4	Query - Diferentes departamentos na Europa	15
5	Comparações e Análise dos Resultados	16
5.1	OracleDB	16
5.2	MongoDB	16
5.3	Neo4j	16
6	Conclusão	18
7	Anexo	19
7.1	Queries SQL	19
7.2	MongoDB	20
7.3	Import Neo4j	21

1 Introdução

No âmbito da unidade Curricular de NoSQL foi nos proposto que realizássemos um trabalho de análise, planeamento, e implementação de um SGBD relacional e dois não relacionais. Para isso deveríamos utilizar a base de dados relacional HR que é disponibilizada no Oracle Database. Esta representa um esquema de uma aplicação de Recursos Humanos, criado pela Oracle, que tem como objetivo principal armazenar os dados dos empregados de uma organização, possuindo assim 7 tabelas:

- EMPLOYEES: Dados dos empregados, tais como: nome, departamento e cargo atual. Os empregados podem ou não estar vinculados a um departamento.
- DEPARTMENTS: Dados dos departamentos em que empregados podem trabalhar.
- REGIONS: Dados sobre as regiões em que a organização pode atuar. Ex.: America, Asia.
- 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.

Estando estas relacionadas da seguinte forma:

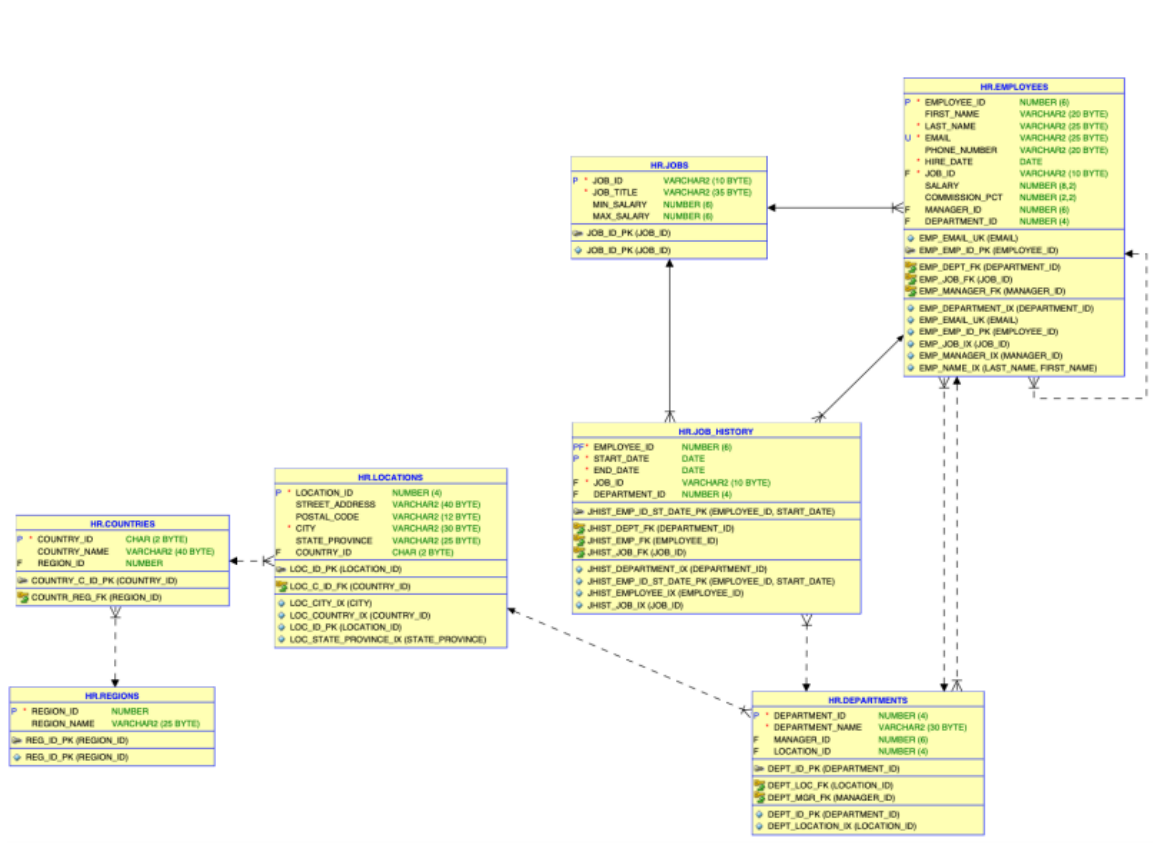


Figure 1: Modelo Lógico

2 Base de Dados Relacional

Relativamente ao modelo relacional utilizamos a o OracleDB que foi a ferramenta mais frequentemente utilizada durante as aulas da Unidade Curricular, á parte isso o uso de SQL é algo que nós temos vindo a utilizar ao longo do curso por isso não foi para nós um problema.

Os dados que importamos para o Oracle foram disponibilizados pelos docentes sendo assim somente necessário correr os dois scripts disponibilizados para importarmos estes dados para a base de dados, não se mostrando grande desafio porém isto levou a uma análise mais minuciosa sobre os dados e consequentemente a um maior entendimento sobre o modelo lógico disponibilizado.

2.1 Queries

Relativamente a queries procuramos enquanto grupo, encontrar algumas com vários tipos de abstração para posteriormente, identificarmos melhor diferenças e ou facilidades ou dificuldades na comparação dos diversos tipos de bases de dados.

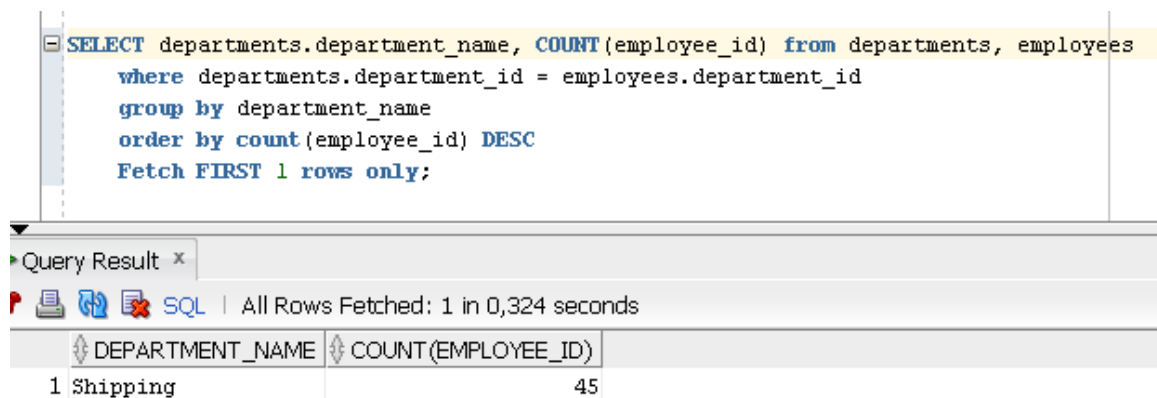
A baixo estarão disponíveis algumas destas queries e os seus respetivos resultados, para que estes também possam ser comparados.

2.1.1 Query - Empregado com maior salário

<

Figure 2: Empregado com maior salário

2.1.2 Query - Departamento com mais Empregados



The screenshot shows a SQL query in a text editor and its corresponding result in a query window. The query selects the department name and the count of employees, ordered by count in descending order, and fetches the first row. The result shows the 'Shipping' department with 45 employees.

```
SELECT departments.department_name, COUNT(employee_id) from departments, employees
where departments.department_id = employees.department_id
group by department_name
order by count(employee_id) DESC
Fetch FIRST 1 rows only;
```

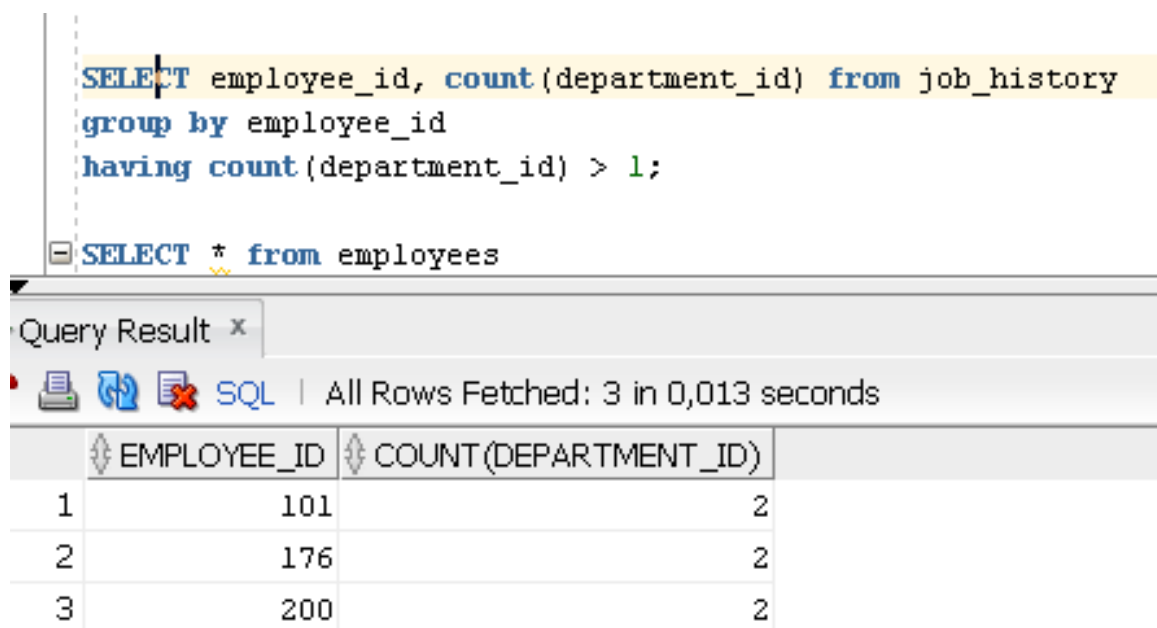
Query Result x

SQL | All Rows Fetched: 1 in 0,324 seconds

DEPARTMENT_NAME	COUNT(EMPLOYEE_ID)
1 Shipping	45

Figure 3: Departamento com mais Empregados

2.1.3 Query - Empregados que trabalharam em mais do que Departamento



The screenshot shows two SQL queries in a text editor and their corresponding results in a query window. The first query selects employee IDs and the count of departments they worked in, filtered by a count greater than 1. The second query selects all columns from the employees table. The result shows three employees (IDs 101, 176, and 200) who have worked in two departments each.

```
SELECT employee_id, count(department_id) from job_history
group by employee_id
having count(department_id) > 1;

SELECT * from employees
```

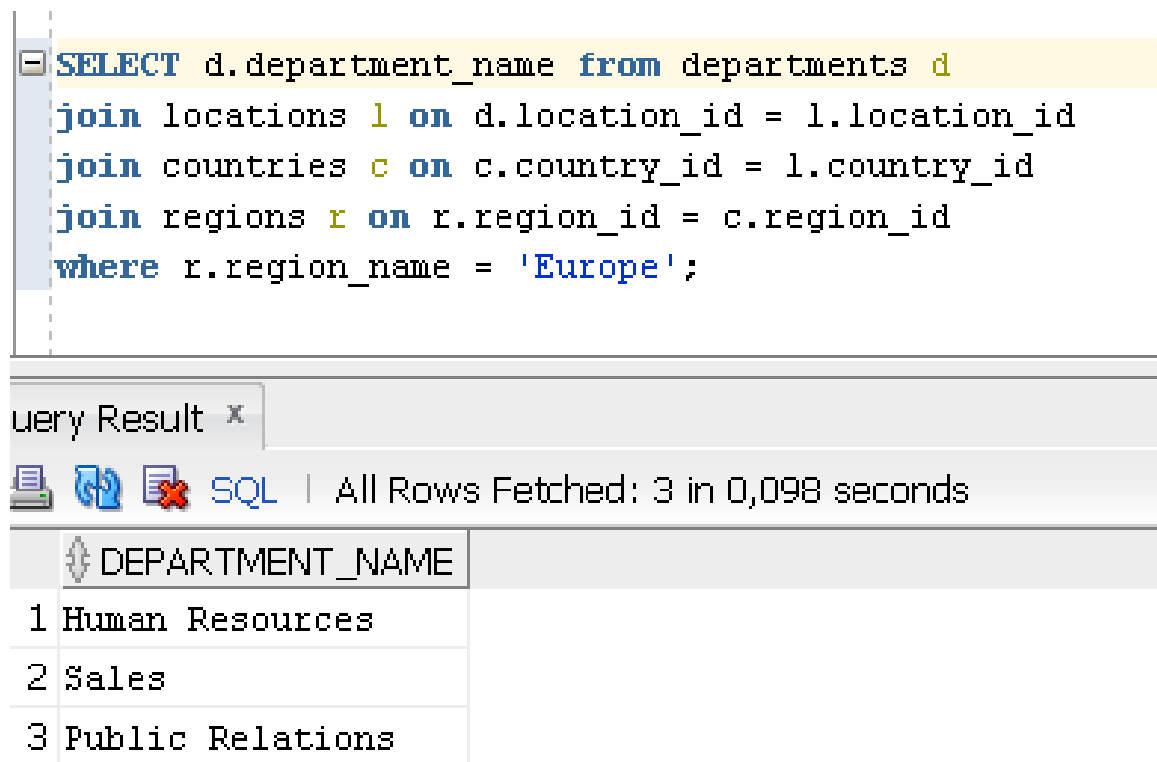
Query Result x

SQL | All Rows Fetched: 3 in 0,013 seconds

	EMPLOYEE_ID	COUNT(DEPARTMENT_ID)
1	101	2
2	176	2
3	200	2

Figure 4: Empregados que trabalharam em mais do que Departamento

2.1.4 Query - Diferentes departamentos na Europa



The screenshot shows a SQL query in a text editor and its results in a query result window. The query is as follows:

```
SELECT d.department_name from departments d
join locations l on d.location_id = l.location_id
join countries c on c.country_id = l.country_id
join regions r on r.region_id = c.region_id
where r.region_name = 'Europe';
```

The query result window shows the following results:

	DEPARTMENT_NAME
1	Human Resources
2	Sales
3	Public Relations

Figure 5: Diferentes departamentos na Europa

3 Base de Dados Documental

Relativamente á base de dados documental escolhida foi o MongoDB, uma vez que esta a par do OracleDB foi aquela com a qual trabalhamos mais no decorrer da Unidade Curricular.

Apesar de tudo, estando o modelo dado mais direcionado para uma base de dados relacional não houve grandes complicações com a criação de **Collections** uma vez que fizemos uso de ferramentas como o Studio3T.

3.1 Criação de Collections

Como referimos já a cima de forma a facilitar o processo de criar as **Collections** utilizamos o Studio3T que nos permitiu interligar o OracleDB ao MongoDB, e posteriormente, a partir das tabelas criadas em SQL e das suas relações (*foreign Keys*), permitiu-nos criar e organizar da melhor forma as nossas Collections.

Assim, a nosso ver numa perspetiva de conseguir utilizar da melhor forma o paradigma documental decidimos criar 3 **Collections**, assentando estas sobre os **EMPLOYEES**, **DEPARTMENTS** e **JOB_HISTORY**.

3.1.1 EMPLOYEES

Relativamente a esta Collection, e numa prespetiva de maximizar as vantagens do modelo Documental, inserimos nela todos os aspetos relevantes de um Employee, desde o seu trabalho, o departamento e respetiva localização do mesmo, e caso exista o seu histórico dentro da empresa. Como podemos ver na imagem a baixo.


```

    "_id" : NumberLong(198),
    "EMPLOYEE_ID" : NumberLong(198),
    "FIRST_NAME" : "Donald",
    "LAST_NAME" : "OConnell",
    "EMAIL" : "DOCONNEL",
    "PHONE_NUMBER" : "650.507.9833",
    "HIRE_DATE" : ISODate("2007-06-21T00:00:00Z"),
    "SALARY" : NumberDecimal("2600"),
    "COMMISSION_PCT" : null,
    "MANAGER_ID" : NumberLong(124),
    "JOBS" : {
      "JOB_ID" : "SH_CLERK",
      "JOB_TITLE" : "Shipping Clerk",
      "MIN_SALARY" : NumberLong(2500),
      "MAX_SALARY" : NumberLong(5500)
    },
    "DEPARTMENTS" : {
      "DEPARTMENT_ID" : NumberLong(50),
      "DEPARTMENT_NAME" : "Shipping",
      "MANAGER_ID" : NumberLong(121),
      "LOCATION_ID" : NumberLong(1500)
    },
    "JOB_HISTORY" : {
    },
    "LOCATIONS" : {
      "LOCATION_ID" : NumberLong(1500),
      "STREET_ADDRESS" : "2011 Interiors Blvd",
      "POSTAL_CODE" : "99236",
      "CITY" : "South San Francisco",
      "STATE_PROVINCE" : "California"
    },
    "COUNTRIES" : {
      "COUNTRY_ID" : "US",
      "COUNTRY_NAME" : "United States of America"
    },
    "REGIONS" : {
      "REGION_ID" : NumberLong(2),
      "REGION_NAME" : "Americas"
    }
  }

```

Figure 6: Collection Employees

3.1.2 DEPARTMENTS

Relativamente à Collection Departments, seguimos a mesma abordagem, procurando armazenar nesta todas as características relevantes a um departamento, assim temos um exemplo apresentado a baixo.

```

    "_id" : ObjectId("6296532b6dd7f36a2990b254"),
    "DEPARTMENT_ID" : NumberLong(200),
    "DEPARTMENT_NAME" : "Operations",
    "EMPLOYEES" : {
    },
    "LOCATIONS" : {
      "LOCATION_ID" : NumberLong(1700),
      "STREET_ADDRESS" : "2004 Charade Rd",
      "POSTAL_CODE" : "98199",
      "CITY" : "Seattle",
      "STATE_PROVINCE" : "Washington"
    },
    "COUNTRIES" : {
      "COUNTRY_ID" : "US",
      "COUNTRY_NAME" : "United States of America"
    },
    "REGIONS" : {
      "REGION_ID" : NumberLong(2),
      "REGION_NAME" : "Americas"
    }
  }

```

Figure 7: Collection Departments

3.1.3 JOB_HISTORY

A Collection Job_history, foi algo que optamos também criar uma vez que apesar de já estar bem explícita dentro da Collection Employees, desta forma conseguimos ter um mais rápido acesso á informação presente no que seria o histórico de um trabalhador, desde o seu antigo trabalho (salário, título de trabalho), ao departamento onde trabalhava.

```
"_id" : ObjectId("6296546a6dd7f36a2990b289"),
"START_DATE" : ISODate("2002-07-01T00:00:00Z"),
"END_DATE" : ISODate("2006-12-31T00:00:00Z"),
"EMPLOYEES" : {
  "EMPLOYEE_ID" : NumberLong(200),
  "FIRST_NAME" : "Jennifer",
  "LAST_NAME" : "Whalen",
  "EMAIL" : "JWHALEN",
  "PHONE_NUMBER" : "515.123.4444",
  "HIRE_DATE" : ISODate("2003-09-17T00:00:00Z"),
  "JOB_ID" : "AD_ASST",
  "SALARY" : NumberDecimal("4400"),
  "COMMISSION_PCT" : null,
  "MANAGER_ID" : NumberLong(101),
  "DEPARTMENT_ID" : NumberLong(10)
},
"JOBS" : {
  "JOB_ID" : "AC_ACCOUNT",
  "JOB_TITLE" : "Public Accountant",
  "MIN_SALARY" : NumberLong(4200),
  "MAX_SALARY" : NumberLong(9000)
},
"DEPARTMENTS" : {
  "DEPARTMENT_ID" : NumberLong(90),
  "DEPARTMENT_NAME" : "Executive",
  "MANAGER_ID" : NumberLong(100),
  "LOCATION_ID" : NumberLong(1700)
}
```

Figure 8: Collection Job_History

3.2 Queries

Relativamente ás queries realizadas em MongoDB, como poderemos ver a baixo no corrente relatório, não encontramos um grande acrescento de dificuldade, apesar de notarmos em grande parte que o mongo não estava tão preparado para trabalhar os dados que nos foram apresentados, muito talvez devido ao facto por estes dados virem de um modelo relacional.

Porém, apesar disso, tendo as nossas Collections desta forma estruturada conseguimos dar resposta sem grandes dificuldades ás queries propostas. Apesar de termos tido alguma dificuldade em operações como **Count**.

3.2.1 Query - Empregado com maior salário

```
> db.EMPLOYEES.find().sort({SALARY: -1}).limit(1).pretty()
{
  "_id" : NumberLong(100),
  "EMPLOYEE_ID" : NumberLong(100),
  "FIRST_NAME" : "Steven",
  "LAST_NAME" : "King",
  "EMAIL" : "SKING",
  "PHONE_NUMBER" : "515.123.4567",
  "HIRE_DATE" : ISODate("2003-06-17T00:00:00Z"),
  "SALARY" : NumberDecimal("24000"),
  "COMMISSION_PCT" : null,
  "MANAGER_ID" : null,
  "JOBS" : {
    "JOB_ID" : "AD_PRES",
    "JOB_TITLE" : "President",
    "MIN_SALARY" : NumberLong(20080),
    "MAX_SALARY" : NumberLong(40000)
  },
  "DEPARTMENTS" : {
    "DEPARTMENT_ID" : NumberLong(90),
    "DEPARTMENT_NAME" : "Executive",
    "MANAGER_ID" : NumberLong(100),
    "LOCATION_ID" : NumberLong(1700)
  },
  "JOB_HISTORY" : {
  },
  "LOCATIONS" : {
    "LOCATION_ID" : NumberLong(1700),
    "STREET_ADDRESS" : "2004 Charade Rd",
    "POSTAL_CODE" : "98199",
    "CITY" : "Seattle",
    "STATE_PROVINCE" : "Washington"
  },
  "COUNTRIES" : {
    "COUNTRY_ID" : "US",
    "COUNTRY_NAME" : "United States of America"
  },
  "REGIONS" : {
    "REGION_ID" : NumberLong(2),
    "REGION_NAME" : "Americas"
  }
}
```

Figure 9: Empregado com maior salário

3.2.2 Query - Departamento com mais Empregados

```
> db.EMPLOYEES.aggregate({$unwind: "$EMPLOYEE_ID"}, {$group: {"_id": "$DEPARTMENTS.DEPARTMENT_NAME", "COUNT": {$sum:1}}}, {$sort: {"COUNT": -1}}, {$limit: 1})
{ "_id" : "Shipping", "COUNT" : 45 }
```

Figure 10: Departamento com mais Empregados

3.2.3 Query - Empregados que trabalharam em mais do que Departamento

```
> db.JOB_HISTORY.aggregate({$unwind: "$EMPLOYEES.DEPARTMENT_ID"}, {$group: {"_id": "$EMPLOYEES.EMPLOYEE_ID", "COUNT": {$sum:1}}}, {$match: {"COUNT": {"$gt": 1}}})
{ "_id" : NumberLong(101), "COUNT" : 2 }
{ "_id" : NumberLong(200), "COUNT" : 2 }
{ "_id" : NumberLong(176), "COUNT" : 2 }
```

Figure 11: Empregados que trabalharam em mais do que Departamento

3.2.4 Query - Diferentes departamentos na Europa

```
> db.DEPARTMENTS.find({"REGIONS.REGION_NAME": "Europe"},{_id: 0, DEPARTMENT_NAME: 1}).pretty()
{ "DEPARTMENT_NAME" : "Human Resources" }
{ "DEPARTMENT_NAME" : "Public Relations" }
{ "DEPARTMENT_NAME" : "Sales" }
```

Figure 12: Diferentes departamentos na Europa

4 Base de Dados Orientada a Grafos

Relativamente á Base de dados Orientada a grafos, assim como no caso das restantes optamos pela qual fomos tendo mais contacto no decorrer da Unidade Curricular, sendo esta o Neo4j.

Durante a familiarização com esta ferramenta tivemos alguns percalços que vamos abordar mais á frente no corrente relatório, porém uma vez ultrapassados, esta mostrou-se ser uma ferramenta e uma base de dados bastante potente.

4.1 Import dos Dados

De forma a transpor os dados que tínhamos na OracleDB tivemos que passar toda a nossa informação nas tabelas para documentos com o formato *.csv* e colocar estes dentro da pasta imports no container do neo4j, para isso recorremos ás ferramentas do *SQLdeveloper* que através da invocação de queries como *SELECT * FROM employees* nos permitiu esta transposição das tabelas.

De seguida, fomos então procurar dar LOAD destes CSVs para nodos do Neo4j, a baixo é mostrado um exemplo de como fizemos para o caso dos JOBS.

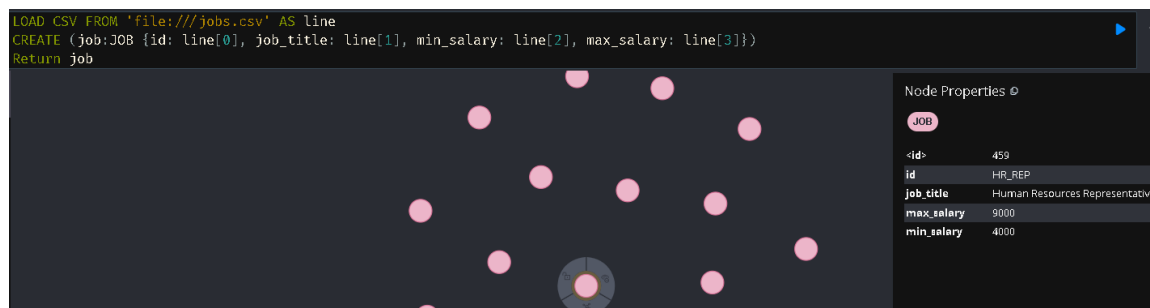


Figure 13: LOAD JOBS

4.2 Criação das relações

Depois de dado o import de todos os CSVs, relativos ás tabelas da base de dados relacional, foi necessário criar as relações entre estas, fazendo uso das *foreign keys*, de notar que vimos-nos obrigados a utilizar este método uma vez que tivemos vários erros ao tentar importar estas relações diretamente dos CSVs como queríamos idealmente ter feito. Apesar disso, foi desta forma mais fácil, fundir as nossas tabelas em relações utilizando o processo que está demonstrado nas figuras a baixo.

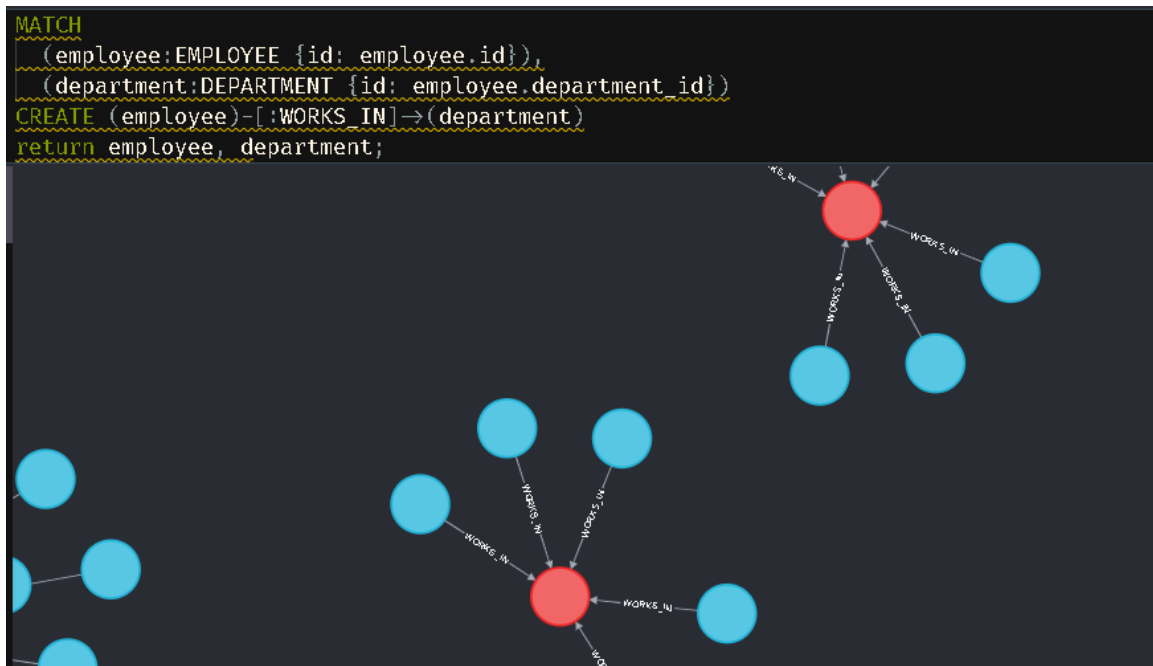


Figure 14: Relações employee - department

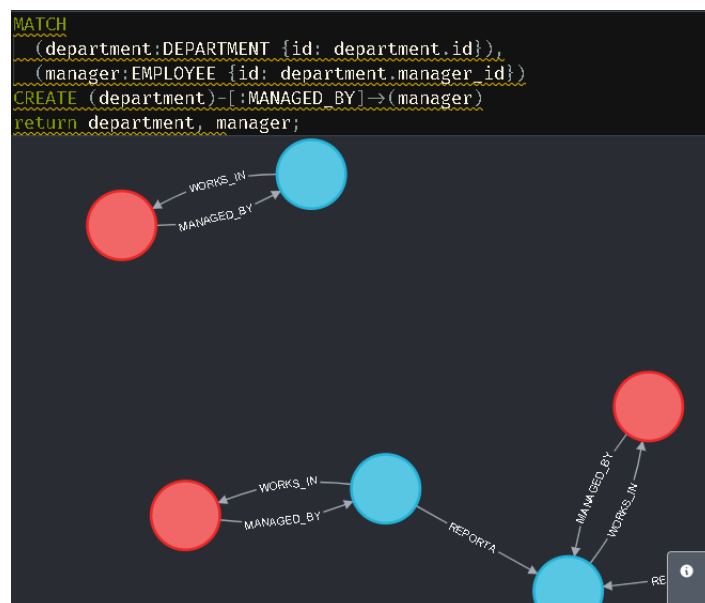


Figure 15: Relações Department- Department Manager

4.2.1 Relações Job_History

Relativamente às relações relativas ao Histórico dos trabalhadores, tivemos de ter especial cuidado, uma vez que não existe com tanta facilidade como nos modelos anteriormente apresentados uma forma tão fácil de representar e armazenar este tipo de relação.

Assim numa fase inicial partimos por relacionar o employee com o seu histórico de trabalho, tendo assim o exemplo na imagem que se segue.

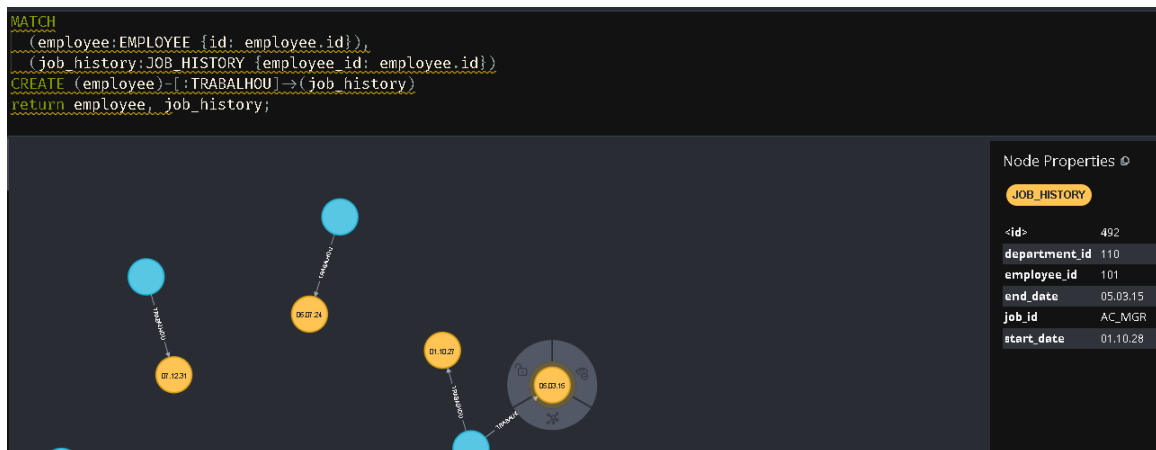


Figure 16: Relação Employee - Job_History

De seguida tendo já o Employee relacionado com o seu Histórico, fomos relacionar este conjunto (EMPLOYEE -> JOB_HISTORY) com o departamento associado a este Histórico. Por fim fomos relacionar o conjunto (EMPLOYEE -> JOB_HISTORY) com o Trabalho ou função desempenhada.

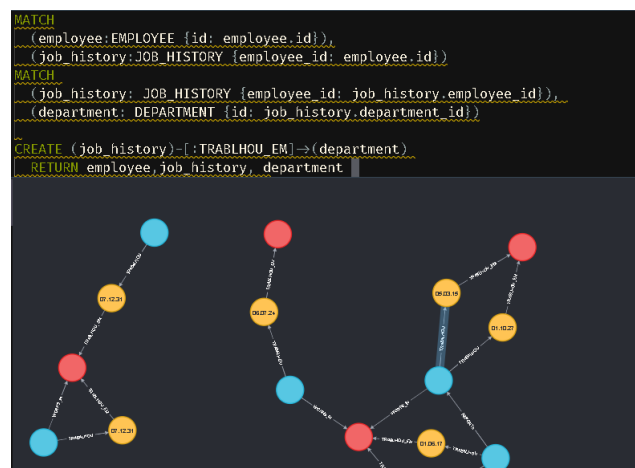


Figure 17: Relação Employee - Job_History - Department

Abaixo temos também uma amostra de como ficou o nosso Grafo final com todas as suas relações.

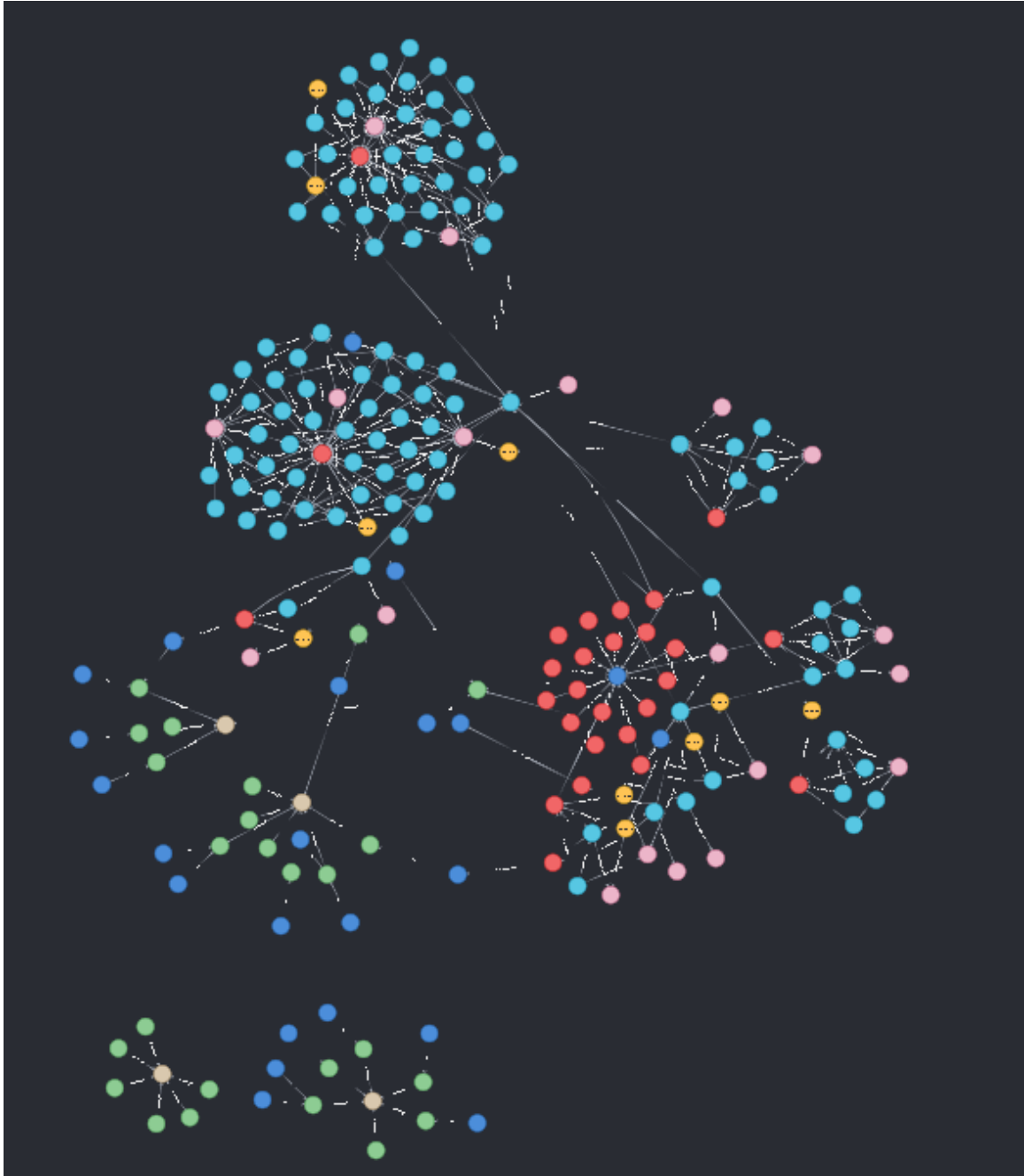


Figure 18: Grafo Final

4.3 Queries

Relativamente às Queries trabalhadas em Neo4j, também não tivemos grandes dificuldades em conseguir encontrar soluções, apesar de em alguns aspetos reconhecermos que existem algumas queries onde os valores ou nodos retornados tem uma leitura mais complicada para a compreensão.

De forma a melhor comparar o Neo4j com os restantes modelos de base de dados vamos mostrar nesta secção as queries demonstradas a cima para os restantes.

4.3.1 Query - Empregado com maior salário

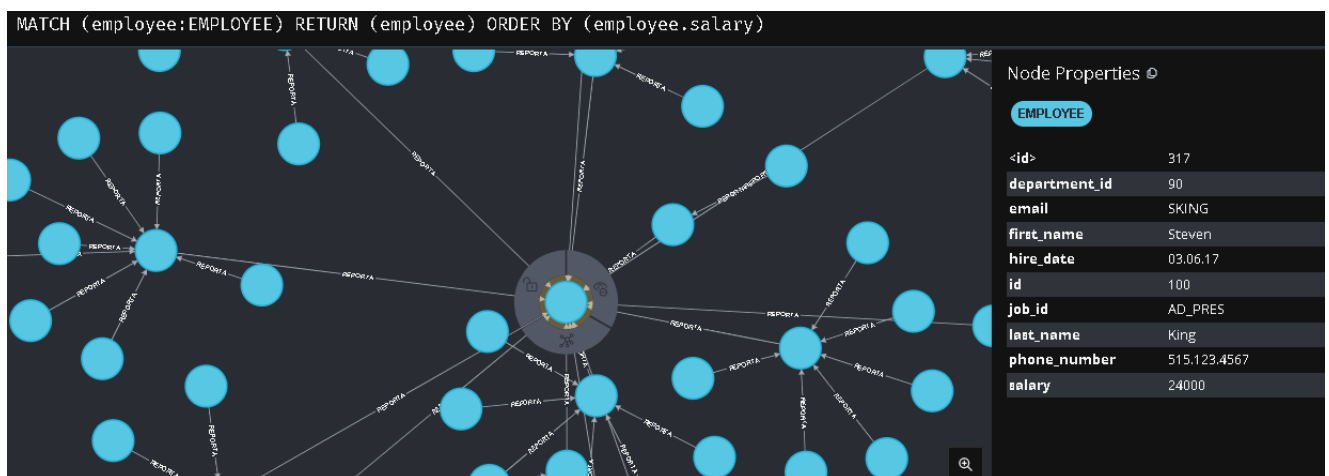


Figure 19: Empregado com maior salário

4.3.2 Query - Departamento com mais Empregados

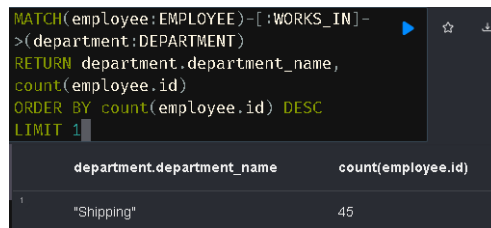
```
MATCH(jobh:JOB_HISTORY)
WITH jobh, count(jobh.employee_id) as cnt
WHERE cnt > 1
RETURN jobh.department_id, cnt as total
```

(no changes, no records)

Figure 20: Departamento com mais Empregados

Nesta Query deparamo-nos com um problema no que toca ao agrupamento intrínseco deste tipo de bases de dados. No output não é executado o `count` juntando as diversas ocorrências do mesmo `department_id`, deste modo nunca conseguimos alcançar os resultados de ter departamentos com mais que um empregado, como seria de esperar.

4.3.3 Query - Empregados que trabalharam em mais do que Departamento

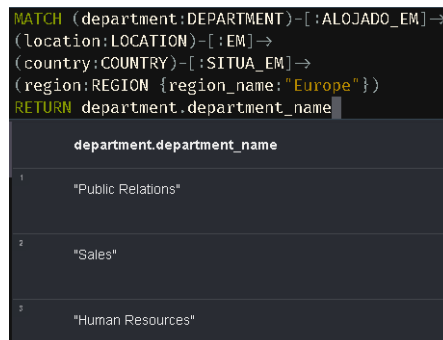


```
MATCH (employee:EMPLOYEE)-[:WORKS_IN]->
(department:DEPARTMENT)
RETURN department.department_name,
count(employee.id)
ORDER BY count(employee.id) DESC
LIMIT 1
```

	department.department_name	count(employee.id)
1	"Shipping"	45

Figure 21: Empregados que trabalharam em mais do que Departamento

4.3.4 Query - Diferentes departamentos na Europa



```
MATCH (department:DEPARTMENT)-[:ALOJADO_EM]->
(location:LOCATION)-[:EM]->
(country:COUNTRY)-[:SITUA_EM]->
(region:REGION {region_name:"Europe"})
RETURN department.department_name
```

	department.department_name
1	"Public Relations"
2	"Sales"
3	"Human Resources"

Figure 22: Diferentes departamentos na Europa

5 Comparações e Análise dos Resultados

Apesar de termos no decorrer do presente relatório vindo a abordar algumas das diferenças e comparações que encontramos entre os vários tipos de Bases de Dados utilizadas neste projeto, vamos nesta secção explicitar de melhor forma as características que para nós foram mais evidentes, fazendo uma comparação entre estes.

5.1 OracleDB

Tendo o modelo fornecido, estar enquadrado para responder da melhor forma a uma Base de Dados relacional, não identificamos grandes problemas no armazenamento dos dados neste modelo, porém relativamente á construção de Queries, já não podemos dizer o mesmo, apesar de já termos algum hábito a trabalhar com Base de Dados Relacionais, as pesquisas entre relações são sempre mais complexas e dão sempre mais algum trabalho, algo que não acontece no caso do MongoDB onde a informação tende a estar toda presente no mesmo documento, não necessitando de grande trabalho a conjugar e compreender as relações entre entidades e no caso do Neo4j onde estas relações estão já estabelecidas sendo muito fácil encontrar a informação que se pretenda destas.

5.2 MongoDB

Relativamente ao MongoDB, todo o mecanismo de *import* dos dados e criação de Collections foi bastante fácil com o uso das ferramentas certas, porém, sentimos que as Collections tinham de ser bastante direcionadas para o tipo de Queries a que procurávamos dar resposta, uma vez que, caso assim não fosse, poderíamos encontrar-nos numa situação em obter essas respostas não fosse tão fácil assim, como o caso das Queries que implicavam uma contagem.

Podendo perder em relação á base de dados relacional onde as relações e ligações entre entidades estão muito bem definidas e é muito fácil com o uso de chaves estrangeiras navegar entre o conteúdo de toda a base de dados. Ou seja, apercebemos-nos que apesar de ser muito fácil encontrar todas as relações com uma entidade num único documento podemos acabar por ter documentos com quantidades de informação muito grandes e ou várias collections com informação repetida, como acabou por ser o caso onde a informação relativa aos Departments está repetida dentro da Collection de Departments e de Employees.

5.3 Neo4j

Por fim, relativamente ao Neo4j, como dito anteriormente, tivemos alguns problemas na importação dos dados para a Base de dados, revelou-se uma operação algo mais custosa "*Hard Code*" uma vez que tivemos de selecionar de cada tabela os campos que queiramos migrar, e consequentemente criar todas as relações entre estes, este foi um processo um pouco demorado, porém podemos interpretar esta dificuldade como consequência dos dados virem preparados para uma base de dados Relacional.

Comparando a questão de acessos á informação e a resposta a Queries, em grande maioria o Neo4j, mostrou-se uma base de dados bastante capaz. E que em concordância com a base

de dados Relacional e em contrapartida com o MongoDB é capaz de mostrar todas as relações existentes, sem a possibilidade de perda de informação entre documentos, como é o caso do que consideramos poder acontecer com o MongoDB, porém relativamente às bases de dados relacionais também se mostrou mais capacitada para mostrar mais facilmente estas relações.

Apesar de tudo e dos resultados das nossas Queries não ser o mais exato em todas elas, podendo isto dever-se aos CSVs ou mesmo ao facto de nós não estarmos tão familiarizados com o funcionamento do Neo4j esta base de dados orientada a grafos, vem de certa forma interligar o potencial das duas bases de dados anteriormente referidas.

6 Conclusão

Este trabalho foi proposto no âmbito da Unidade Curricular Bases de Dados de NoSQL, foi nos então proposto que realizasse-mos um trabalho de análise, planeamento e implementação de um SGBD relacional e dois não relacionais. Para isso deveríamos utilizar a base de dados relacional HR que é disponibilizada no Oracle Database. Esta representa um esquema de uma aplicação de Recursos Humanos, criado pela Oracle.

Relativamente ao trabalho realizado consideramos que conseguimos cumprir o objetivo do projeto, este permitiu-nos por em prática os conhecimentos adquiridos na Unidade Curricular acerca e bases de dados relacionais e não relacionais.

Conseguimos observar como os mesmos dados têm de ser organizados nas diferentes bases de dados de modo a não perdermos nenhuma informação e ainda para conseguirmos responder às mesmas queries em qualquer uma delas. Foi possível ainda comprovar as diferentes características de cada uma delas em termos de tempo e eficiência na procura da informação.

7 Anexo

7.1 Queries SQL

```
SELECT * FROM employees ORDER BY salary ;
```

```
Select employees.department_id , COUNT (*) as total  
FROM employees  
GROUP BY employees.department_id;
```

```
SELECT * FROM departments where department_id = '50';
```

```
SELECT departments.department_name, COUNT(employee_id) from departments, employees  
where departments.department_id = employees.department_id  
group by department_name  
order by count(employee_id) DESC  
Fetch FIRST 1 rows only;
```

```
SELECT first_name, last_name, count(job_id) from employees  
group by employees.first_name, employees.last_name;
```

```
SELECT employee_id, count(department_id) from job_history  
group by employee_id  
having count(department_id) > 1;
```

```
SELECT * from employees  
where employees.employee_id IN (SELECT employee_id from job_history  
group by employee_id  
having count(department_id) > 1);
```

```
Select * from employees e  
order by salary DESC  
FETCH first 1 rows only;
```

```
SELECT * from departments d  
join locations l on d.location_id = l.location_id  
where l.city = 'Seattle';
```

```

SELECT d.department_name from departments d
join locations l on d.location_id = l.location_id
join countries c on c.country_id = l.country_id
join regions r on r.region_id = c.region_id
where r.region_name = 'Europe';

```

```

SELECT * FROM employees e
where e.manager_id = '100';

```

```

SELECT * FROM employees e
where e.manager_id=100 AND e.department_id =90;

```

--QUEM ESTA AINDA A TRABALHAR NA EMPRESA

```

SELECT * from employees e
full outer join job_history jh on jh.employee_id = e.employee_id
where jh.employee_id IS NULL OR e.employee_id IS NULL;

```

```

SELECT * FROM employees;
SELECT * FROM departments;
SELECT * FROM jobs;

```

7.2 MongoDB

```

1 -> db.EMPLOYEES.find({}).sort({SALARY: 1}).pretty()
2 -> db.EMPLOYEES.aggregate({$unwind: "$DEPARTMENTS.DEPARTMENT_ID"}, {$group: {"
    _id": "$DEPARTMENTS.DEPARTMENT_ID", "COUNT": {$sum:1}}})
3 -> db.DEPARTMENTS.find({DEPARTMENT_ID: 50}).pretty()
4 -> db.EMPLOYEES.aggregate({$unwind: "$EMPLOYEE_ID"},
    {$group: {"_id": "$DEPARTMENTS.DEPARTMENT_NAME", "COUNT": {$sum:1}}}, {
    $sort: {"COUNT": -1}}, {$limit: 1})
5 -> db.EMPLOYEES.aggregate({$unwind: "$JOBS.JOB_ID"},
    {$group: {"_id": {$concat:["$FIRST_NAME", " ", "$LAST_NAME"]}, "COUNT": {
    $sum:1}}}, {$sort: {"COUNT": 1}})
6 -> db.JOB_HISTORY.aggregate({$unwind: "$EMPLOYEES.DEPARTMENT_ID"},
    {$group: {"_id": "$EMPLOYEES.EMPLOYEE_ID", "COUNT": {$sum:1}}},{$match: {"
    COUNT": {"$gt": 1}}})
7 -> db.EMPLOYEES.aggregate({$unwind: "$JOB_HISTORY.DEPARTMENT_ID"},
    {$group: {"_id": "$EMPLOYEE_ID", "COUNT": {$sum:1}, "DATA": {"$addToSet": "
    $$ROOT"}}},{$match: {"COUNT": {"$gt": 1}}})
8 -> db.EMPLOYEES.find({}).sort({SALARY: -1}).limit(1).pretty()
9 -> db.DEPARTMENTS.find({"LOCATIONS.CITY": "Seattle").pretty()
10-> db.DEPARTMENTS.find({"REGIONS.REGION_NAME": "Europe"},{_id: 0,
    DEPARTMENT_NAME: 1}).pretty()

```

```

11-> db.EMPLOYEES.find({MANAGER_ID: 100}).limit(1).pretty()
12-> db.EMPLOYEES.find({$and: [{MANAGER_ID: 100}, {"DEPARTMENTS.DEPARTMENT_ID":
    90}]}).pretty()

```

7.3 Import Neo4j

```

//employees
LOAD CSV FROM 'file:///emp.csv' AS line
CREATE (employee:EMPLOYEE {id: line[0], first_name: line[1], last_name: line[2],
    email: line[3], phone_number: line[4], hire_date: line[5], job_id: line[6],
    salary: line[7], comission_pct: line[8], manager_id: line[9], department_id:
    line[10]})
Return employee

//departments
LOAD CSV FROM 'file:///departments.csv' AS line
CREATE (department:DEPARTMENT {id: line[0], department_name: line[1], manager_id:
    line[2], location_id: line[3]})
Return department

//jobs
LOAD CSV FROM 'file:///jobs.csv' AS line
CREATE (job:JOB {id: line[0], job_title: line[1], min_salary: line[2], max_salary:
    line[3]})
Return job

//LOCATIONS
LOAD CSV FROM 'file:///locations.csv' AS line
CREATE (location:LOCATION {id: line[0], street_address: line[1], postal_code: line
    [2], city: line[3], state_providence: line[4], country_id: line[5] })
Return location

//COUNTRIES
LOAD CSV FROM 'file:///countries.csv' AS line
CREATE (country:COUNTRY {id: line[0], country_name: line[1], region_id: line[2]})
Return country

//REGION
LOAD CSV FROM 'file:///regions.csv' AS line
CREATE (region:REGION {id: line[0], region_name: line[1]})
Return region

//JOB_HISTORY

```

```

LOAD CSV FROM 'file:///job_history.csv' AS line
CREATE (job_history:JOB_HISTORY {employee_id: line[0],start_date: line[1],
    end_date: line[2],job_id: line[3], department_id: line[4] })
Return job_history

//employees with departments

MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (department:DEPARTMENT {id: employee.department_id})
CREATE (employee)-[:WORKS_IN]->(department)
return employee, department;

//employees with managers
MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (manager:EMPLOYEE {id: employee.manager_id})
CREATE (employee)-[:REPORTA]->(manager)

//employees with jobs
MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (job:JOB {id: employee.job_id})
CREATE (employee)-[:FAZ_TRABALHo]->(job)
return employee, job;

//country region

MATCH
    (country:COUNTRY {id: country.id}),
    (region:REGION {id: country.region_id})
CREATE (country)-[:SITUA_EM]->(region)
return country, region;

//location country

MATCH
    (location:LOCATION {id: location.id}),
    (country:COUNTRY {id: location.country_id})
CREATE (location)-[:EM]->(country)
return location, country;

//department location

MATCH
    (department:DEPARTMENT {id: department.id}),

```



```

    (location:LOCATION {id: department.location_id})
CREATE (department)-[:ALOJADO_EM]->(location)
return department, location;

// department manager
MATCH
    (department:DEPARTMENT {id: department.id}),
    (manager:EMPLOYEE {id: department.manager_id})
CREATE (department)-[:MANAGED_BY]->(manager)
return department, manager;

//job history employee
MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (job_history:JOB_HISTORY {employee_id: employee.id})
CREATE (employee)-[:TRABALHOU]->(job_history)
return employee, job_history;

//job_history with department
MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (job_history:JOB_HISTORY {employee_id: employee.id})
MATCH
    (job_history: JOB_HISTORY {employee_id: job_history.employee_id}),
    (department: DEPARTMENT {id: job_history.department_id})

CREATE (job_history)-[:TRABLHOU_EM]->(department)
RETURN employee,job_history, department

//job_history with jobs
MATCH
    (employee:EMPLOYEE {id: employee.id}),
    (job_history:JOB_HISTORY {employee_id: employee.id})
MATCH
    (job_history: JOB_HISTORY {employee_id: job_history.employee_id}),
    (job: JOB {id: job_history.job_id})

CREATE (job_history)-[:TRABLHOU_EM]->(job)
RETURN employee,job_history, job

```