Universidade do Minho
Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Letivo de 2016/2017

Comboio Académico

Bruno Rafael Lamas Corredoura Dantas, A74207
Daniel Camelo Rodrigues, A75655
Hugo Alves Carvalho, A74219
Luís Miguel da Cunha Lima, A74260

Janeiro, 2017

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

Comboio Académico

Bruno Rafael Lamas Corredoura Dantas, A74207

Daniel Camelo Rodrigues, A75655

Hugo Alves Carvalho, A74219

Luís Miguel da Cunha Lima, A74260

Janeiro, 2017

Resumo

O projeto apresentado neste relatório consiste na migração de um sistema de base de dados relacional, que foi implementado anteriormente através do SGBD MySQL, para um sistema de base de dados não relacional usando o SBD Neo4j orientada a grafos.

Inicialmente revemos e analisamos o modelo conceptual e o respetivo esquema lógico para termos uma ideia inicial do processo de migração.

Posteriormente exportamos os dados das tabelas da base de dados relacional para prosseguirmos com a implementação dos seus respetivos nodos através da importação de dados dos ficheiros ".csv". De seguida, identificamos algumas restrições de unicidade das propriedades dos nodos, e assim criámos os relacionamentos existentes entre estes.

Por fim, elaboramos um conjunto de *queries* na linguagem de interrogação do Neo4j (*cypher)* com o mesmo propósito das queries do sistema relacional para podermos comprovar a operacionalidade do sistema implementado e a consistência dos dados.

Área de Aplicação: Planeamento e execução de sistemas de bases de dados não relacionais relativo a reservas em comboios nacionais e internacionais.

Palavras-Chave: Estudante, Funcionário, Bilhete, Viagem, Comboio, Reserva, Lugar, Nodo, Relacionamento, Base de dados.

Índice

1. Introdução 1
1.1. Motivação e Objetivos 1
1.2. Justificação da Viabilidade do Projeto 1
1.3. Estrutura do Relatório 2
2. Esquema da Base de Dados Relacional 3
3. Migração de Dados 5
3.1. Sistema de Base da Dados não Relacional 5
3.2. Ficheiros CSV 5
3.3. Importação de Dados 6
3.3.1. Estudante 6
3.3.2. Funcionário 7
3.3.3. Bilhete 7
3.3.4. Viagem 8
3.3.5. Comboio 9
3.3.6. Lugar 10
3.4. *Unique Constraints* 11
3.5. Relacionamentos 13
3.5.1. Funcionário regista Bilhete 13
3.5.2. Estudante compra Bilhete 14
3.5.3. Viagem tem Bilhete 15
3.5.4. Comboio faz Viagem 15
3.5.5. Comboio tem Lugar 17
3.6. Modelo Neo4j 18
4. *Queries* 19
4.1. Quais o nome e número de cartão de cidadão dos estudantes que compraram bilhetes e estudam na Universidade do Minho? 19
4.2. Qual a hora de partida, hora de chegada e preço das viagens com partida em Braga e destino no Porto? 20
4.3. Quais as viagens que são realizadas pelo comboio número 1? 21
4.4. Quais os lugares ocupados para a viagem 1 no dia 20 de dezembro de 2016? 22

5. Conclusões 24

Índice de Figuras

Figura 1 – Tabela funcionario no MySQL 5
Figura 2 – Ficheiro “funcionario.csv” 5
Figura 3 – Import do ficheiro “estudante.csv” 6
Figura 4 – Neo4j – Nodos estudante 6
Figura 5 – Import do ficheiro “funcionario.csv” 7
Figura 6 – Neo4j – Nodos Funcionario 7
Figura 7 – Import do ficheiro “bilhete.csv” 8
Figura 8 – Neo4j – Nodos Bilhete 8
Figura 9 – Import do ficheiro “viagem.csv” 9
Figura 10 – Neo4j – Nodos Viagem 9
Figura 11 – Import do ficheiro “comboio.csv” 9
Figura 12 – Neo4j – Nodos Comboio 10
Figura 13 – Import do ficheiro “lugar.csv” 10
Figura 14 – Neo4j – Nodos Lugar 11
Figura 15 – *Unique Constraint* numero_cc (estudante) 11
Figura 16 – *Unique Constraint* idBilhete (bilhete) 12
Figura 17 – *Unique Constraint* idFuncionario (funcionario) 12
Figura 18 – Unique Constraint idViagem (viagem) 12
Figura 19 – *Unique Constraint* idComboio (comboio) 12
Figura 20 – Relacionamento Funcionário regista Bilhete 13
Figura 21 – Neo4j – Relacionamento Funcionário regista Bilhete 13
Figura 22 – Relacionamento Estudante compra Bilhete 14
Figura 23 – Neo4j – Relacionamento Estudante compra Bilhete 14
Figura 24 – Relacionamento Viagem tem Bilhete 15
Figura 25 – Neo4j – Relacionamento Viagem tem Bilhete 15
Figura 26 – Relacionamento Comboio faz Viagem 16
Figura 27 – Neo4j – Relacionamento Comboio faz Viagem 16
Figura 28 – Relacionamento Comboio tem Lugar 17
Figura 29 – Neo4j – Relacionamento Comboio tem Lugar 17
Figura 30 – Esquema Neo4j 18
Figura 31 – *Querie* 1 no Neo4j 19

Figura 32 – Querie 1 em SQL 20
Figura 33 – *Querie 2* no Neo4j 20
Figura 34 – *Querie 2* em SQL 20
Figura 35 – *Querie* 3 no Neo4j 21
Figura 36 – *Querie* 3 em SQL 22
Figura 37 – *Querie* 4 no Neo4j 22
Figura 38 – *Querie* 4 em SQL 23

1. Introdução

O presente relatório refere-se ao “Comboio Académico”, um sistema de base de dados que foi implementado pelo grupo, que inicialmente tinha sido implementado como um sistema de base de dados relacional.

Este projeto surge na necessidade de alterar o sistema anterior para um sistema de base de dados não relacional.

A apresentação do caso de estudo e a contextualização referentes a este trabalho podem ser consultados no relatório anterior do modelo relacional, uma vez que são idênticos.

1.1. Motivação e Objetivos

O principal objetivo do grupo com realização deste projeto é adquirir rotinas de planeamento e execução de sistemas de bases de dados não relacionais.

Após um aumento considerável do volume de dados no nosso SBD, a questão da eficiência começou a tornar-se muito importante. Como tal, foi necessário explorar linguagens NoSQL para tentar resolver este problema.

Decidimos optar por um SBD não relacional orientado por grafos, suportado pela aplicação Neo4j.

1.2. Justificação da Viabilidade do Projeto

Uma das principais vantagens em trocar um SBD relacional por uma solução noSQL, em particular orientado por grafos, é o seu escalonamento. Como este não possuí nenhum tipo de esquema pré-definido, o modelo possui maior flexibilidade o que favorece a inserção transparente de outros elementos.

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.

1.3. Estrutura do Relatório

Numa parte inicial do relatório é explicado o processo de migração de dados do SBDR para o SBD não relacional orientado por grafos e como se realizou a importação dos mesmos. Posteriormente, segue-se a explicação das restrições utilizadas para garantir a integridade dos dados após a sua inserção. Após este subcapítulo, é mostrado como se recriaram os relacionamentos para o novo SBD.

Por último, é apresentado o modelo final em Neo4j e as mesmas queries que foram realizadas em SQL no SBDR, desta vez realizadas em cypher.

2. Esquema da Base de Dados Relacional

Uma vez que este trabalho consiste na migração de um sistema de base de dados relacional para um modelo de dados não relacional, é necessário identificar as entidades e os seus atributos de modo a perceber como se comporta a base de dados.

Assim, na figura seguinte é mostrado o esquema conceptual do SBDR.

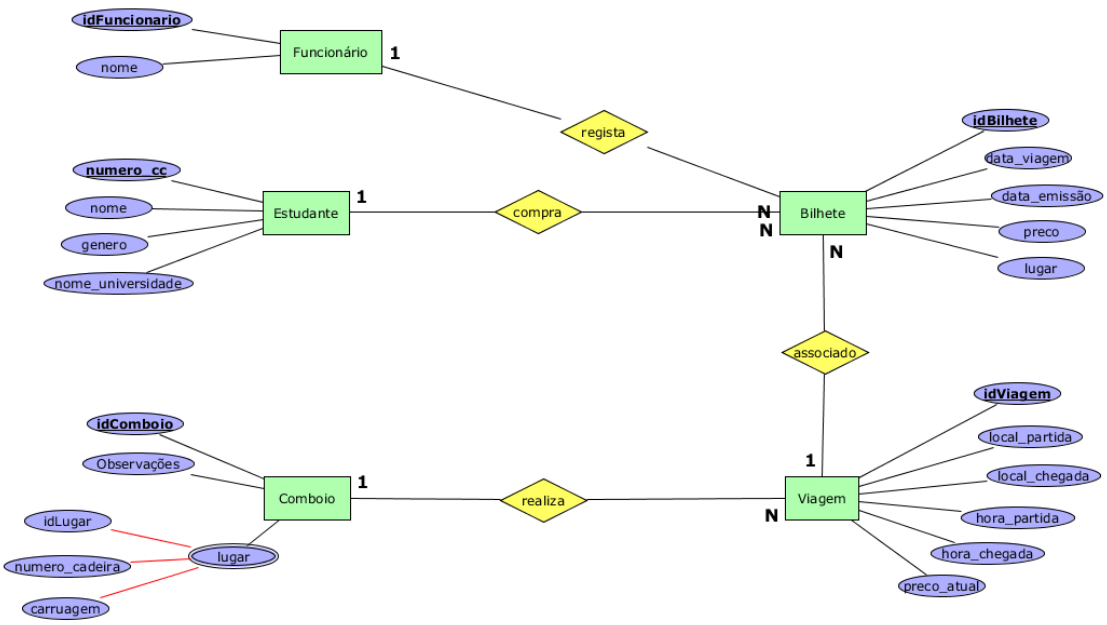

Figura 1 – Esquema conceptual do SBDR

Este modelo foi convertido no seu respetivo esquema lógico, onde para além da identificação de entidades e atributos, podemos verificar como é preservada a integridade dos dados.

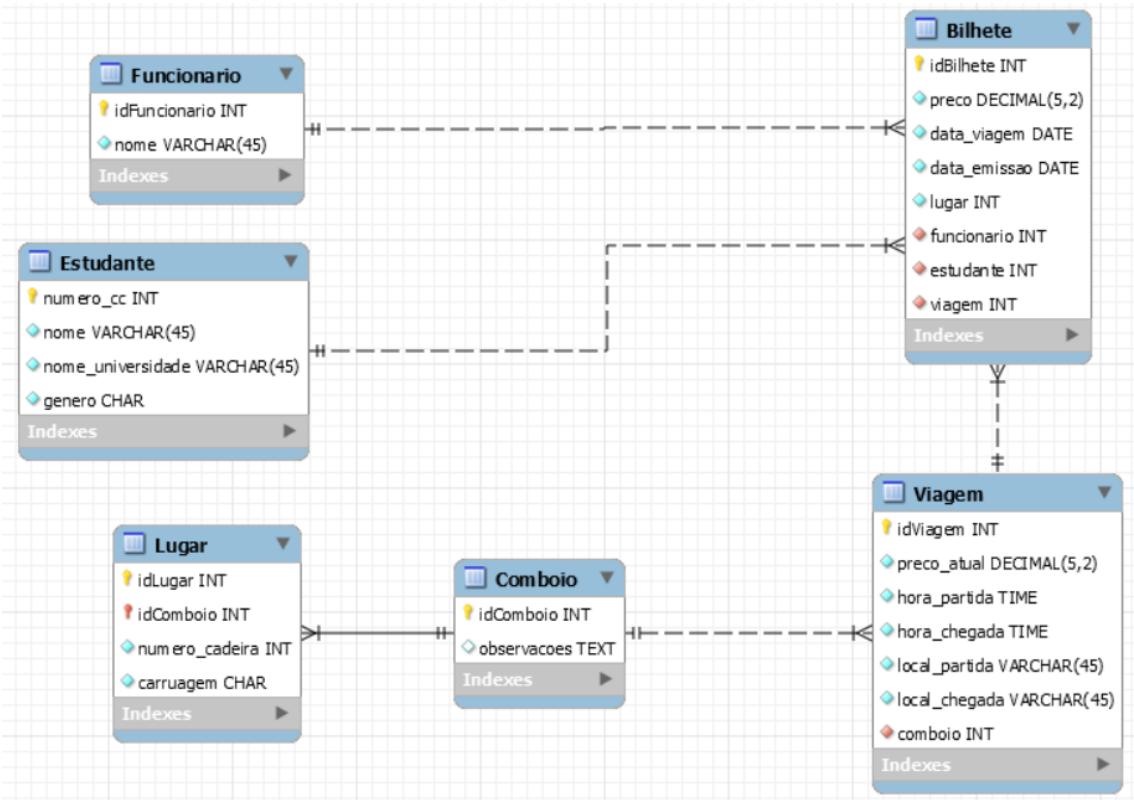

Figura 2 – Esquema lógico do SBDR

3. Migração de Dados

3.1. Sistema de Base da Dados não Relacional

Nesta parte do relatório iremos explicar como fizemos o processo de migração de dados do sistema de base de dados relacional apresentado no capítulo anterior, para um sistema de base de dados não relacional orientado por grafos (*Graphs Databases)*.

Para realizarmos esse processo, foi utilizada a ferramenta Neo4j.

3.2. Ficheiros CSV

O primeiro passo para inicializarmos o processo de migração de dados, foi exportar cada uma das tabelas do modelo relacional para o seu respetivo ficheiro ".csv".

De seguida exemplificamos este passo com a exportação de dados da tabela funcionário.

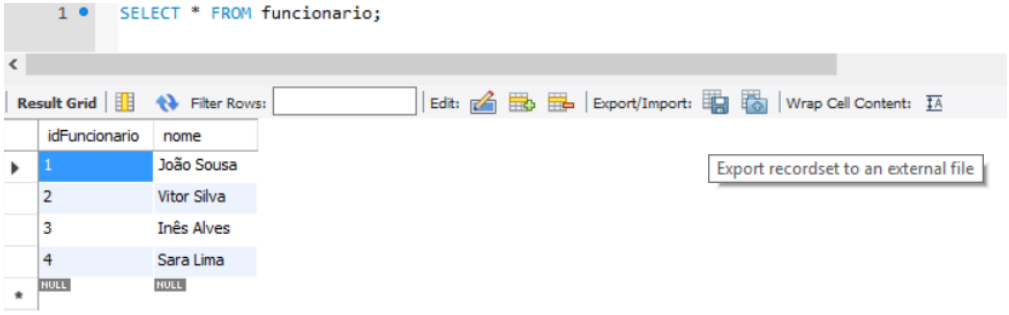

Figura 3 – Tabela funcionario no MySQL

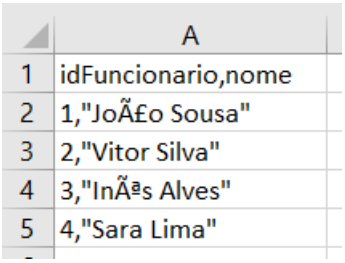

Figura 4 – Ficheiro "funcionario.csv"

3.3. Importação de Dados

Neste tópico iremos explicar como fizemos a importação de dados dos ficheiros ".csv" descritos no tópico anterior.

Para importarmos cada uma das linhas de cada ficheiro e assim criarmos os seus respetivos nodos, foram usados comandos específicos do *cypher* que serão também demonstrados.

3.3.1. Estudante

Como podemos verificar no modelo de dados relacional, a tabela estudante contém uma chave primária (numero_cc) e três atributos (nome, nome_universidade e género). Assim, ao fazer a importação de dados e criação de nodos, são importados todos os dados do ficheiro "estudante.csv".

```
LOAD CSV WITH HEADERS FROM 'file:///estudante.csv' AS row
CREATE (e: Estudante {numero_cc: toInt(row.numero_cc)})
SET e.nome = toString(row.nome),
    e.nome_universidade = toString(row.nome_universidade),
    e.genero = toString(row.genero)
RETURN e;
```

Figura 5 – Import do ficheiro "estudante.csv"

Como podemos comprovar na figura seguinte, os nodos estudante foram corretamente criados.

Figura 6 – Neo4j – Nodos estudante

3.3.2. Funcionário

Uma vez que a tabela contém uma chave primária (numero_cc) e um atributo (nome), ao fazer a importação de dados e criação de nodos, são importados todos os dados do ficheiro "funcionario.csv".

```
LOAD CSV WITH HEADERS FROM 'file:///funcionario.csv' AS row
CREATE (f: Funcionario {idFuncionario: toInt(row.idFuncionario)})
SET f.nome = toString(row.nome)
RETURN f;
```

Figura 7 – Import do ficheiro "funcionario.csv"

Como podemos comprovar na figura seguinte, os nodos funcionário foram corretamente criados.

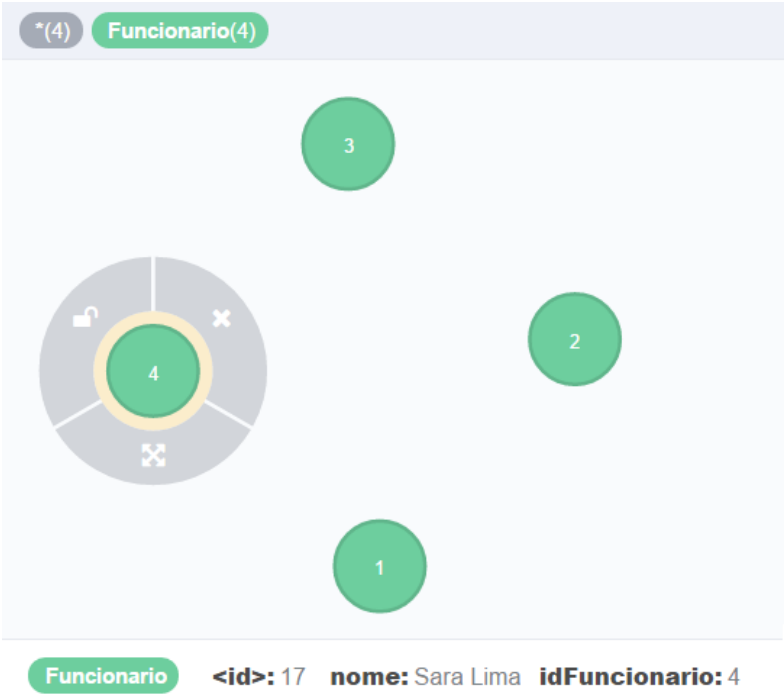

Figura 8 – Neo4j – Nodos Funcionario

3.3.3. Bilhete

Como podemos verificar no modelo de dados relacional, a tabela bilhete contém uma chave primária (idBilhete), quatro atributos (preço, data_viagem, data_emissão e lugar) e ainda três chaves estrangeiras (funcionário, estudante e viagem).

Uma vez que as chaves estrangeiras no modelo relacional servem para garantir a integridade referencial, e como no modelo não relacional esta é garantida pelos

relacionamentos, apenas precisamos de importar os dados idBilhete, preço, data_viagem, data_emissão e lugar do ficheiro “bilhete.csv”.

```
LOAD CSV WITH HEADERS FROM 'file:///bilhete.csv' AS row
CREATE (b: Bilhete {idBilhete: toInt(row.idBilhete)})
SET b.preco = toFloat(row.preco),
    b.data_viagem = toString(row.data_viagem),
    b.data_emissao = toString(row.data_emissao),
    b.lugar = toInt(row.lugar)
RETURN b;
```

Figura 9 – Import do ficheiro “bilhete.csv”

Como podemos comprovar na figura seguinte, os nodos bilhete foram corretamente criados.

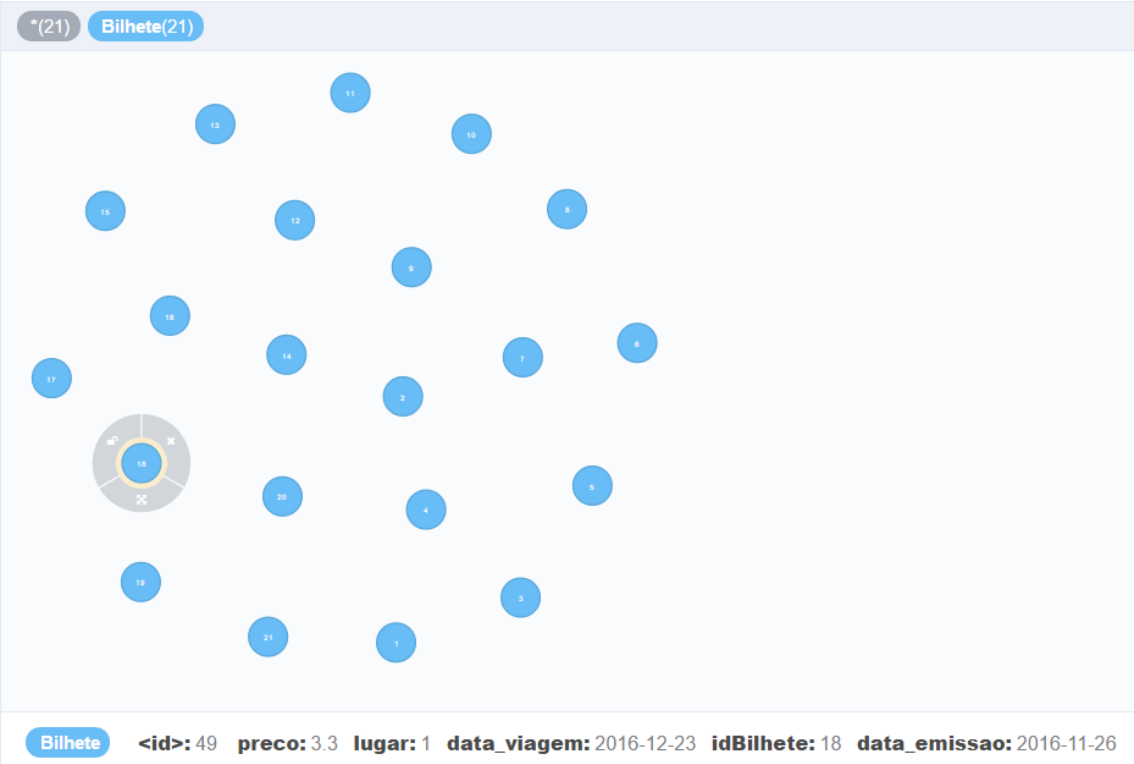

Figura 10 – Neo4j – Nodos Bilhete

3.3.4. Viagem

Como podemos verificar no modelo de dados relacional, a tabela viagem contém uma chave primária (idViagem), cinco atributos (preco_atual, hora_partida, hora_chegada, local_partida, local_chegada) e uma chave estrangeira (comboio).

Tal como referido na situação do bilhete do ponto anterior, apenas precisamos de importar idViagem, preço_atual, hora_partida, hora_chegada, local_partida e local_chegada do ficheiro “viagem.csv”.

```
LOAD CSV WITH HEADERS FROM 'file:///viagem.csv' AS row
CREATE (v: Viagem {idViagem: toInt(row.idViagem)})
SET v.preco_atual = toFloat(row.preco_atual), v.hora_partida = toString(row.hora_partida),
    v.hora_chegada = toString(row.hora_chegada), v.local_partida = toString(row.local_partida),
    v.local_chegada = toString(row.local_chegada)
RETURN v;
```

Figura 11 – Import do ficheiro “viagem.csv”

Como podemos comprovar na figura seguinte, os nodos viagem foram corretamente criados.

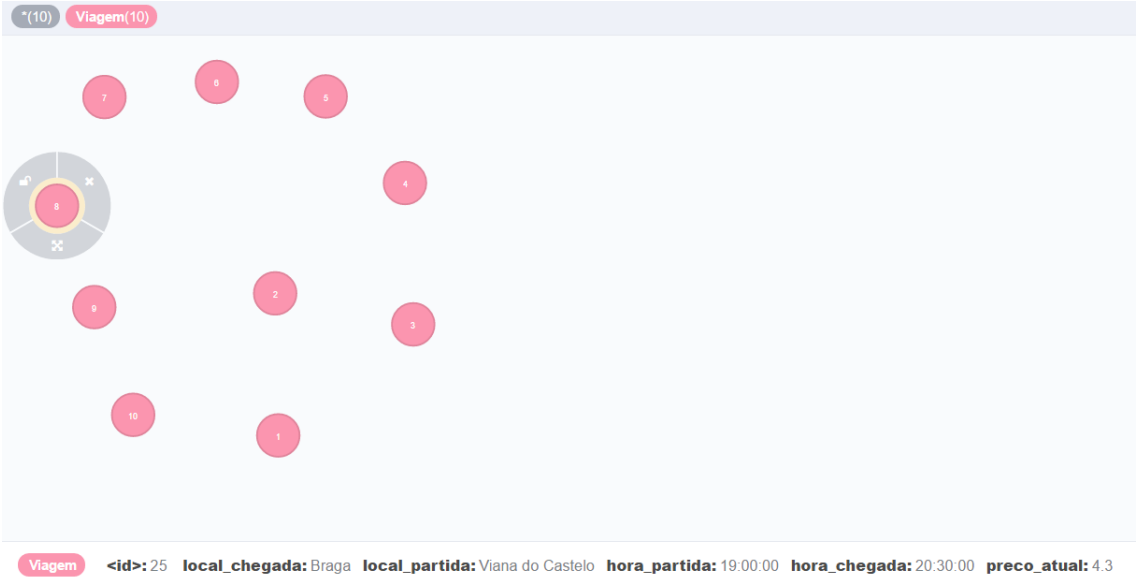

Figura 12 – Neo4j – Nodos Viagem

3.3.5. Comboio

Como podemos verificar no modelo de dados relacional, a tabela comboio contém uma chave primária (idComboio) e um atributo (observações). Assim, ao fazer a importação de dados e criação de nodos, são importados todos os dados do ficheiro “comboio.csv”.

```
LOAD CSV WITH HEADERS FROM 'file:///comboio.csv' AS row
CREATE (c: Comboio {idComboio: toInt(row.idComboio)})
SET c.observacoes = toString(row.observacoes)
RETURN c;
```

Figura 13 – Import do ficheiro “comboio.csv”

Como podemos comprovar na figura seguinte, os nodos comboio foram corretamente criados.

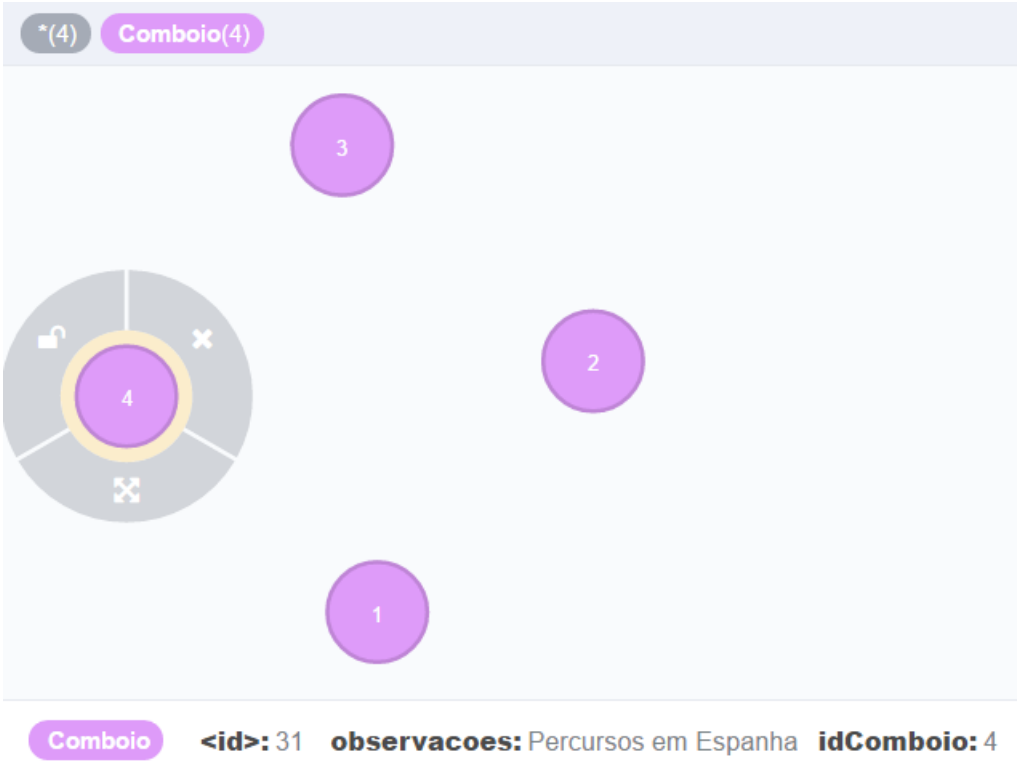

Figura 14 – Neo4j – Nodos Comboio

3.3.6. Lugar

Como podemos verificar no modelo de dados relacional, a tabela lugar contém uma chave primária composta por dois atributos (idLugar e idComboio), e dois atributos (numero_ cadeira e carruagem). No entanto, o atributo idComboio é uma chave estrangeira.

Assim, ao contrário das situações do bilhete e da viagem em que não precisávamos de importar a chave estrangeira, neste caso temos que importá-la pois ela compõe a chave primária, sendo fundamental para identificar cada lugar de cada comboio.

Ao fazer a importação de dados e criação de nodos, são importados todos os dados do ficheiro “lugar.csv”.

```
LOAD CSV WITH HEADERS FROM 'file:///lugar.csv' AS row
CREATE (l: Lugar {idLugar: toInt(row.idLugar)})
SET l.idComboio = toInt(row.idComboio),
    l.numero_cadeira = toInt(row.numero_cadeira),
    l.carruagem = toString(row.carruagem)
RETURN l;
```

Figura 15 – Import do ficheiro “lugar.csv”

Como podemos comprovar na figura seguinte, os nodos viagem foram corretamente criados.

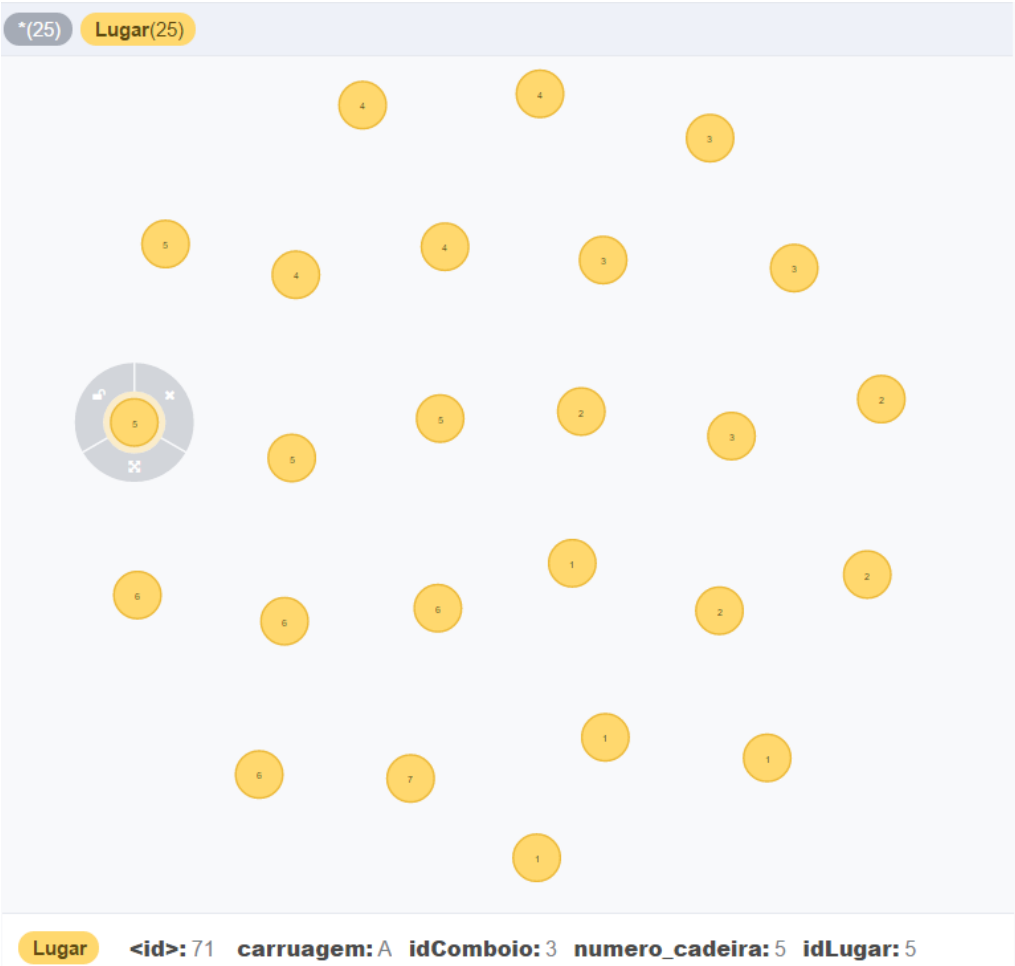

Figura 16 – Neo4j – Nodos Lugar

3.4. *Unique Constraints*

Para uma correta migração do sistema, e de modo a impedir possíveis erros futuros de inserção de dados, foram criadas algumas *Unique Constraints* para garantir que determinados valores só podem existir uma única vez em cada nodo.

Por exemplo, nos nodos de Estudante foi criada uma *Unique Constraint* que garante que não podem existir números de cartão de cidadão repetidos. Tal como pode ser comprovado pela figura seguinte, a *constraint* foi criada corretamente.

```
$ CREATE CONSTRAINT ON (e: Estudante) ASSERT e.numero_cc IS UNIQUE;
```

Rows

Added 1 constraint, statement completed in 175 ms.

Figura 17 – *Unique Constraint* numero_cc (estudante)

É considerado que não podem existir valores repetidos de idBilhete, foi criada a seguinte *constraint:*

```
$ CREATE CONSTRAINT ON (b:Bilhete) ASSERT b.idBilhete IS UNIQUE;
```

Rows

Added 1 constraint, statement completed in 93 ms.

Figura 18 – *Unique Constraint* idBilhete (bilhete)

Como diferentes funcionários não podem ter o mesmo idFuncionario, foi criada a seguinte *constraint:*

```
$ CREATE CONSTRAINT ON (f: Funcionario) ASSERT f.idFuncionario IS UNIQUE;
```

Rows

Added 1 constraint, statement completed in 77 ms.

Figura 19 – *Unique Constraint* idFuncionario (funcionario)

Como diferentes viagens não podem ter o mesmo idViagem, foi criada a seguinte *constraint:*

```
$ CREATE CONSTRAINT ON (v: Viagem) ASSERT v.idViagem IS UNIQUE;
```

Rows

Added 1 constraint, statement completed in 91 ms.

Figura 20 – Unique Constraint idViagem (viagem)

Como diferentes comboios não podem ter o mesmo idComboio, foi criada a seguinte *constraint:*

```
$ CREATE CONSTRAINT ON (c: Comboio) ASSERT c.idComboio IS UNIQUE;
```

Rows

Added 1 constraint, statement completed in 45 ms.

Figura 21 – *Unique Constraint* idComboio (comboio)

3.5. Relacionamentos

Em seguida, será explicado como criámos os diferentes relacionamentos existentes no sistema, com a demonstração dos respetivos comandos utilizados no *cypher,* bem como o comprovativo da correta criação dos relacionamentos.

3.5.1. Funcionário regista Bilhete

Para criar o relacionamento entre o funcionário e bilhete, é necessário procurar no ficheiro "bilhete.csv" os números de funcionário que aparecem na coluna funcionário (correspondente à chave estrangeira do modelo relacional).

```
Load CSV WITH HEADERS FROM 'file:///bilhete.csv' as row
MATCH(f: Funcionario {idFuncionario: toInt(row.funcionario)})
MATCH(b: Bilhete  {idBilhete: toInt(row.idBilhete)})
Create (f)-[:REGISTA]->(b);
```

Figura 22 – Relacionamento Funcionário regista Bilhete

Como podemos verificar na figura seguinte, o relacionamento foi criado corretamente.

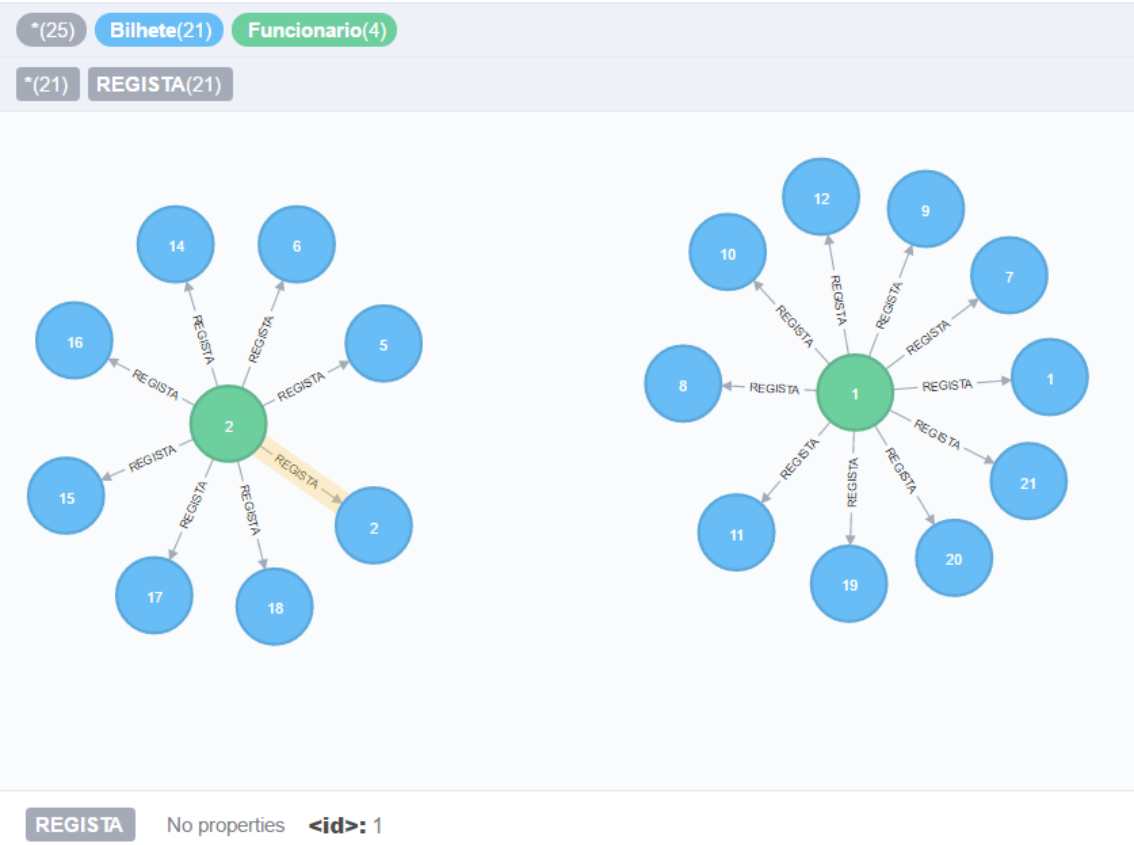

Figura 23 – Neo4j – Relacionamento Funcionário regista Bilhete

3.5.2. Estudante compra Bilhete

Para criar o relacionamento entre o estudante e bilhete, é necessário procurar no ficheiro “bilhete.csv” os números de estudante que aparecem na coluna estudante (correspondente à chave estrangeira do modelo relacional).

```
Load CSV WITH HEADERS FROM 'file:///bilhete.csv' as row
MATCH(e: Estudante {numero_cc: toInt(row.estudante)})
MATCH(b: Bilhete  {idBilhete: toInt(row.idBilhete)})
Create (e)-[:COMPRA]->(b);
```

Figura 24 – Relacionamento Estudante compra Bilhete

Como podemos verificar na figura seguinte, o relacionamento foi criado corretamente.

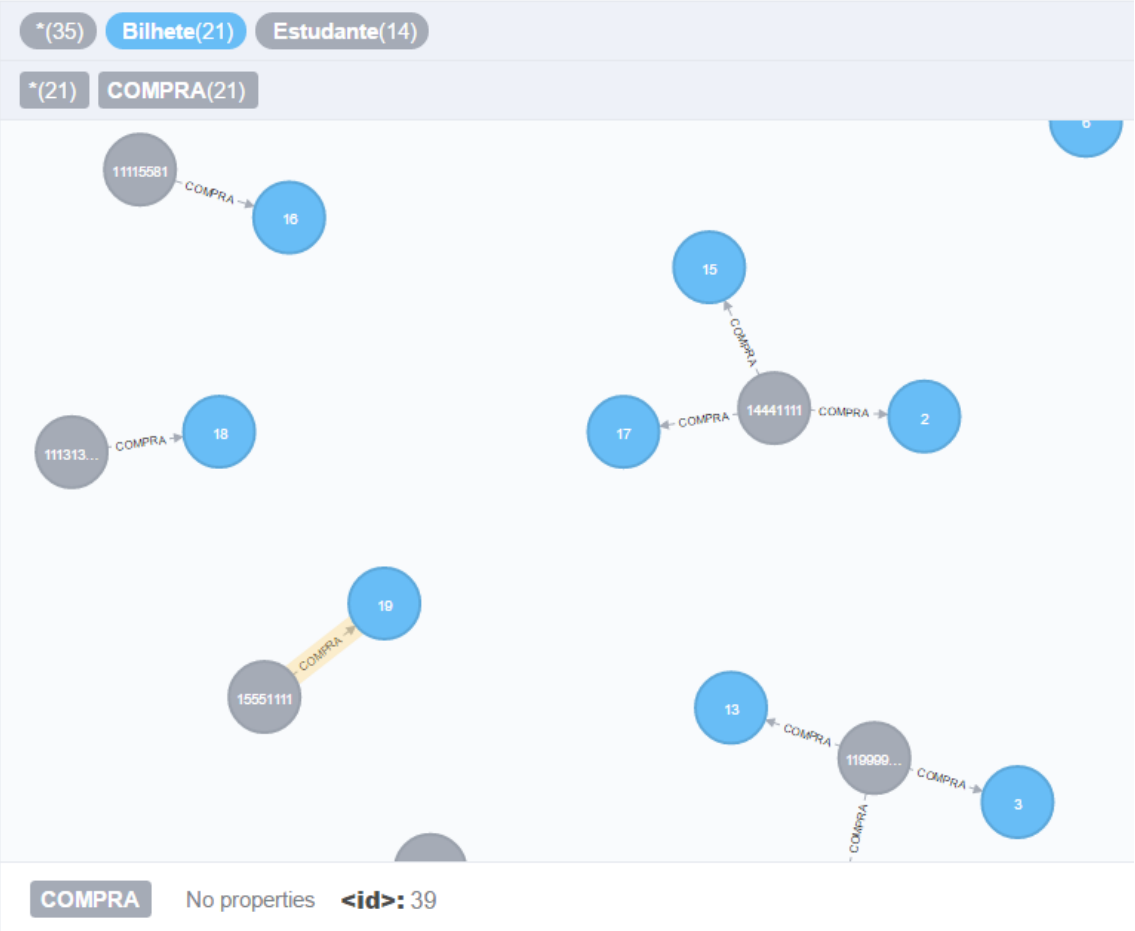

Figura 25 – Neo4j – Relacionamento Estudante compra Bilhete

3.5.3. Viagem tem Bilhete

Para criar o relacionamento entre viagem e bilhete, é necessário procurar no ficheiro “bilhete.csv” os números de viagem que aparecem na coluna viagem (correspondente à chave estrangeira do modelo relacional).

```
Load CSV WITH HEADERS FROM 'file:///bilhete.csv' as row
MATCH(v: Viagem {idViagem: toInt(row.viagem)})
MATCH(b: Bilhete  {idBilhete: toInt(row.idBilhete)})
Create (v)-[:TEM_BILHETE]->(b);
```

Figura 26 – Relacionamento Viagem tem Bilhete

Como podemos verificar na figura seguinte, o relacionamento foi criado corretamente.

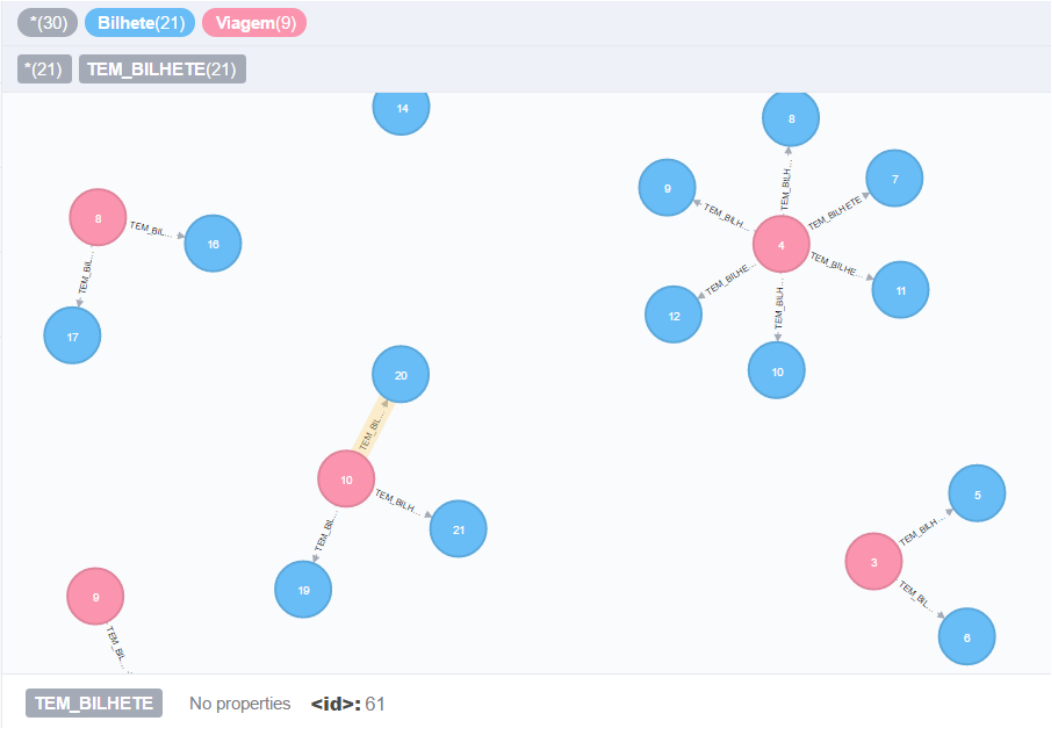

Figura 27 – Neo4j – Relacionamento Viagem tem Bilhete

3.5.4. Comboio faz Viagem

Para criar o relacionamento entre comboio e viagem, é necessário procurar no ficheiro “viagem.csv” os números de comboio que aparecem na coluna comboio (correspondente à chave estrangeira do modelo relacional).

```
Load CSV WITH HEADERS FROM 'file:///viagem.csv' as row
MATCH(c: Comboio {idComboio: toInt(row.comboio)})
MATCH(v: Viagem  {idViagem: toInt(row.idViagem)})
Create (c)-[:FAZ_VIAGEM]->(v);
```

Figura 28 – Relacionamento Comboio faz Viagem

Como podemos verificar na figura seguinte, o relacionamento foi criado corretamente.

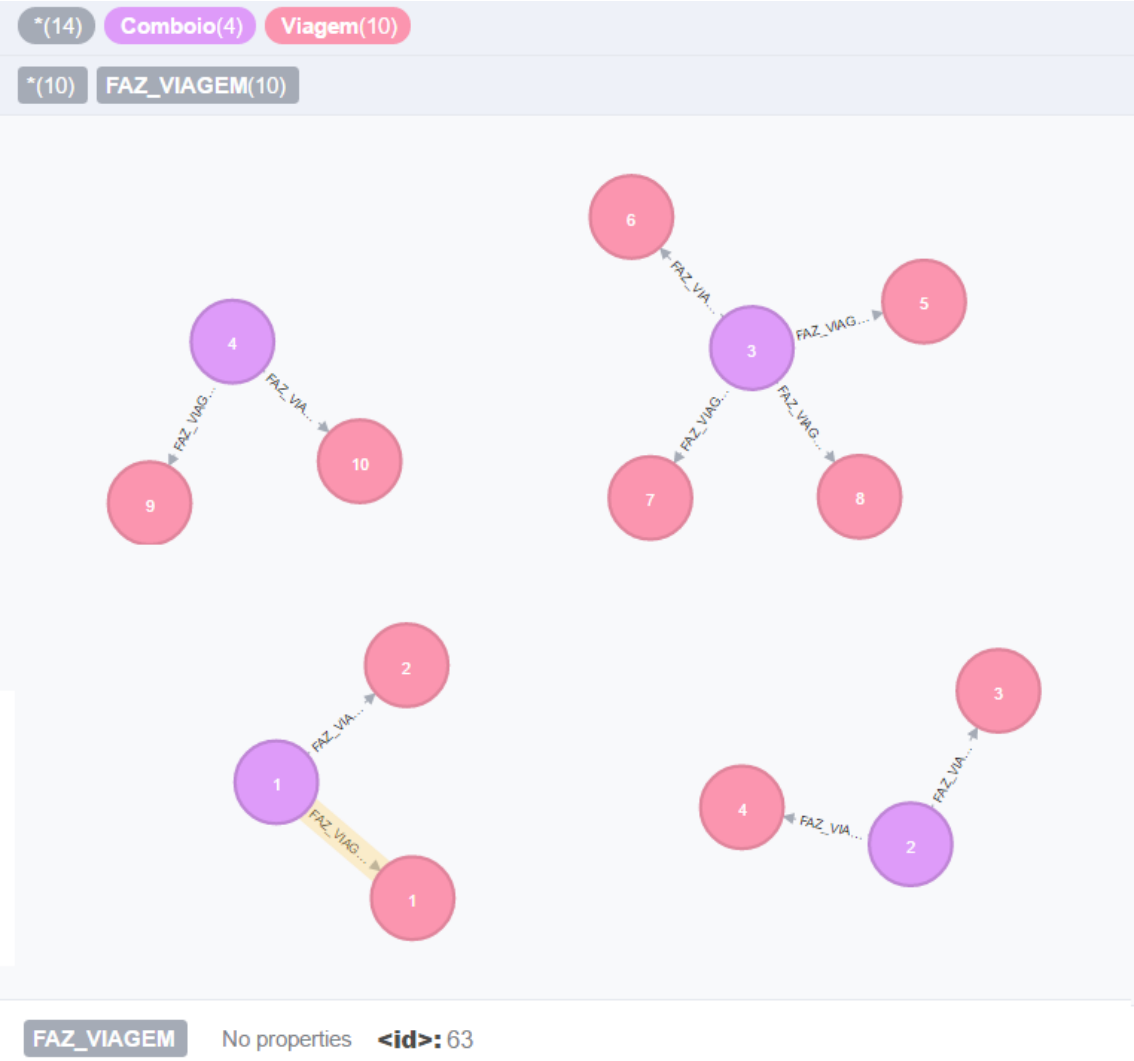

Figura 29 – Neo4j – Relacionamento Comboio faz Viagem

3.5.5. Comboio tem Lugar

Para criar o relacionamento entre comboio e lugar, ao contrário dos exemplos anteriores, não é necessário procurar no ficheiro “lugar.csv” uma vez que o número de comboio de cada lugar já foi importado anteriormente.

```
MATCH (l: Lugar),(c: Comboio)
WHERE l.idComboio = c.idComboio
CREATE (c)-[:TEM_LUGAR]->(l);
```

Figura 30 – Relacionamento Comboio tem Lugar

Como podemos verificar na figura seguinte, o relacionamento foi criado corretamente.

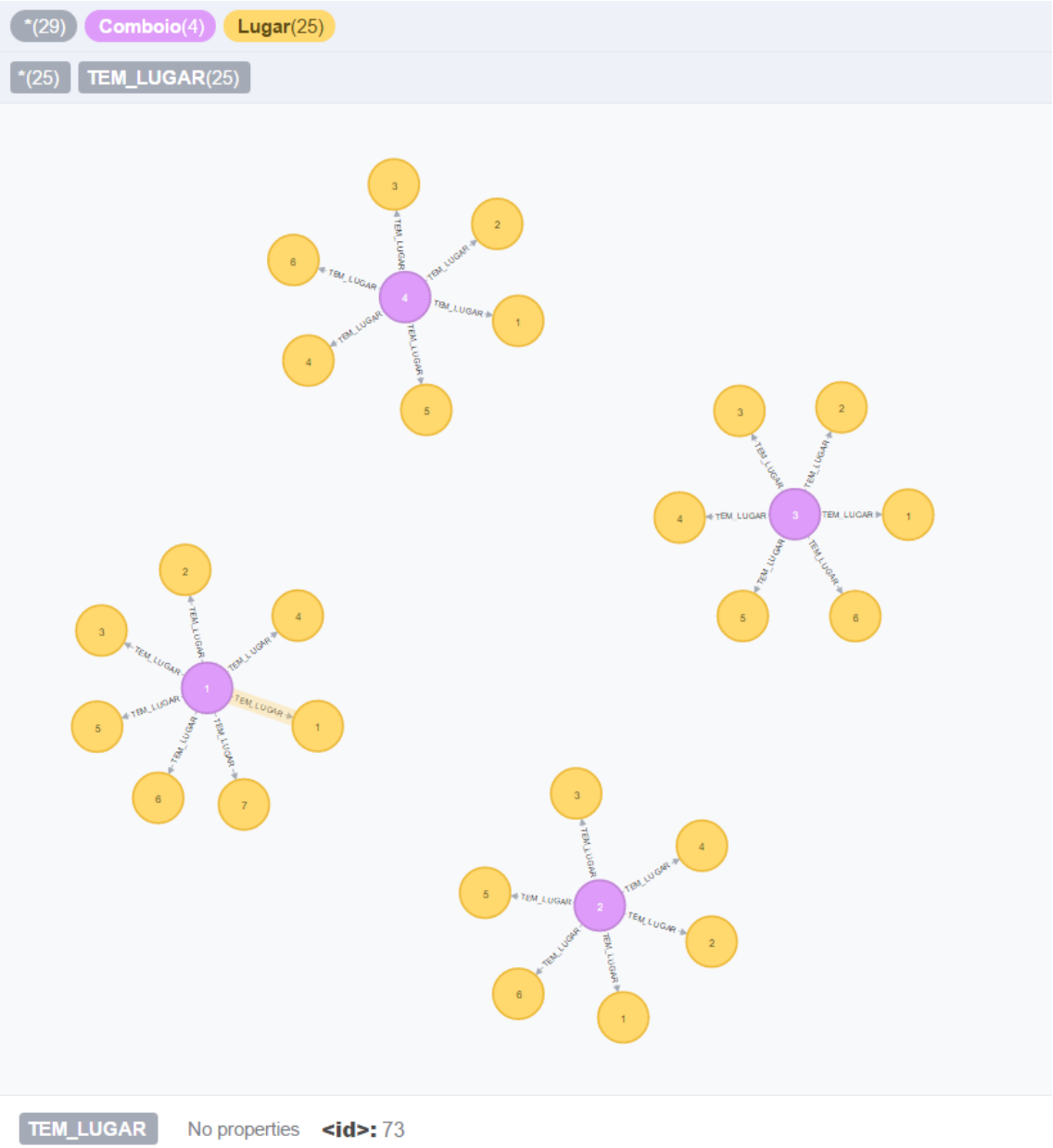

Figura 31 – Neo4j – Relacionamento Comboio tem Lugar

3.6. Modelo Neo4j

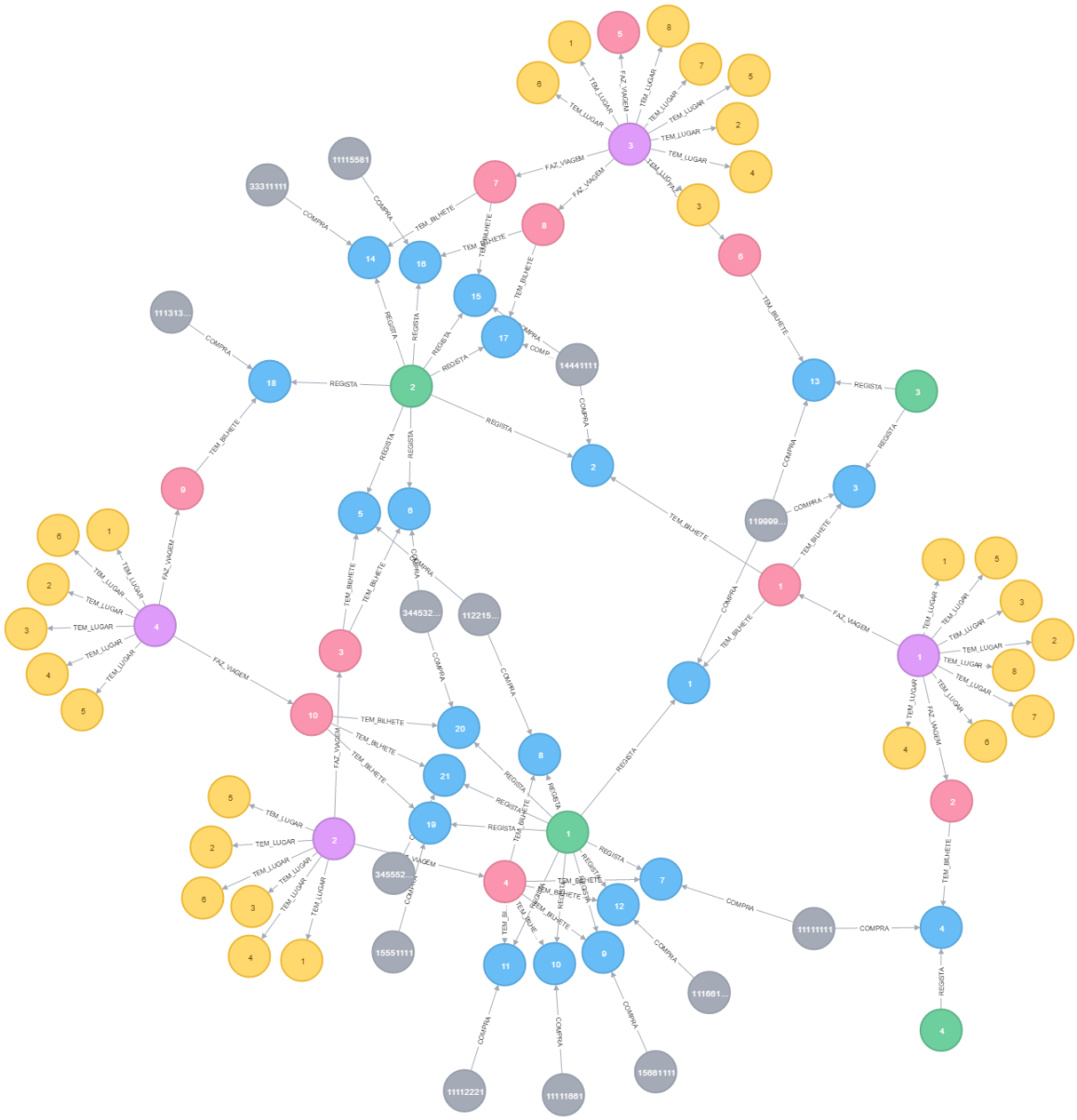

Figura 32 – Esquema Neo4j

4. *Queries*

De modo a demonstrar a operacionalidade do sistema implementado, serão exemplificadas de seguida um conjunto de *queries*, utilizado a linguagem de interrogação do Neo4j (*cypher*).

Por outro lado, é demonstrado também uma comparação com os resultados da mesma *querie* no modelo relacional.

4.1. Quais o nome e número de cartão de cidadão dos estudantes que compraram bilhetes e estudam na Universidade do Minho?

```
1 MATCH (e: Estudante)-[:COMPRA]->(b: Bilhete)
2 WHERE e.nome_universidade = 'Universidade do Minho'
3 RETURN DISTINCT e.numero_cc, e.nome;
4
```

$ MATCH (e: Estudante)-[:COMPRA]->(b: Bilhete) WHERE e.nome_universidade = 'Universidade do Minho' RETURN DISTINCT...

Rows | Text | Code

e.numero_cc	e.nome
11111111	Gonçalo Lopes
11111661	Bruno Rodrigues
11115581	Andreia Cunha
33311111	Leticia Sousa

Figura 33 – *Querie* 1 no Neo4j

```
/*Quais  os estudantes da Universidade do Minho que compraram bilhetes?*/
SELECT numero_cc, nome FROM estudante
WHERE nome_universidade = 'Universidade do Minho'
      AND numero_cc IN (SELECT estudante FROM bilhete);
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

numero_cc	nome
11111111	Gonçalo Lopes
11111661	Bruno Rodrigues
11115581	Andreia Cunha
33311111	Leticia Sousa
NULL	NULL

Figura 34 – Querie 1 em SQL

Como podemos verificar, o resultado da *querie* é o exatamente o mesmo, tal como era pretendido.

4.2. Qual a hora de partida, hora de chegada e preço das viagens com partida em Braga e destino no Porto?

```
MATCH (v: Viagem)
WHERE v.local_partida = 'Braga' AND v.local_chegada = 'Porto'
RETURN v.hora_partida, v.hora_chegada, v.preco_atual;
```

$ MATCH (v: Viagem) WHERE v.local_partida = 'Braga' AND v.local_chegada = 'Porto' RETURN v.hora_partida, v.hora_ch...

Rows | Text

v.hora_partida	v.hora_chegada	v.preco_atual
10:00:00	11:00:00	2
12:00:00	13:00:00	2

Figura 35 – *Querie 2* no Neo4j

```
/*Qual a hora de partida, hora de chegada e preço das viagens Braga - Porto? */
SELECT hora_partida, hora_chegada, preco_atual
FROM Viagem
WHERE local_partida = 'Braga' AND local_chegada = 'Porto';
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

hora_partida	hora_chegada	preco_atual
10:00:00	11:00:00	2.00
12:00:00	13:00:00	2.00

Figura 36 – *Querie 2* em SQL

Como podemos verificar, o resultado da *querie* é o exatamente o mesmo, tal como era pretendido.

4.3. Quais as viagens que são realizadas pelo comboio número 1?

```
1 MATCH (c: Comboio)-[:FAZ_VIAGEM]->(v: Viagem)
2 WHERE c.idComboio = 1
3 RETURN v;
4
```

```
$ MATCH (c: Comboio)-[:FAZ_VIAGEM]->(v: Viagem) WHERE c.idComboio = 1 RETURN v;
```

Graph | Rows | Text | Code

v

local_chegada	Barcelos
local_partida	Viana do Castelo
hora_partida	09:30:00
hora_chegada	10:15:00
preco_atual	2.5
idViagem	1

local_chegada	Viana do Castelo
local_partida	Barcelos
hora_partida	11:00:00
hora_chegada	11:45:00
preco_atual	2.5
idViagem	2

Started streaming 2 records after 2 ms and completed after 3 ms.

Figura 37 – *Querie* 3 no Neo4j

```sql
/* Quais as viagens que o comboio 1 realiza? */
SELECT *
FROM Viagem
WHERE comboio = 1;
```

Result Grid | Filter Rows: | Edit: | Export/Import: | Wrap Cell Content:

idViagem	preco_atual	hora_partida	hora_chegada	local_partida	local_chegada	comboio
1	2.50	09:30:00	10:15:00	Viana do Castelo	Barcelos	1
2	2.50	11:00:00	11:45:00	Barcelos	Viana do Castelo	1
NULL	NULL	NULL	NULL	NULL	NULL	NULL

Figura 38 – *Querie* 3 em SQL

Como podemos verificar, o resultado da *querie* é o exatamente o mesmo, tal como era pretendido.

4.4. Quais os lugares ocupados para a viagem 1 no dia 20 de dezembro de 2016?

```cypher
MATCH (v: Viagem)-[:TEM_BILHETE]->(b: Bilhete)
WHERE v.idViagem = 1 AND b.data_viagem = '2016-12-20'
RETURN b.lugar;
```

$ MATCH (v: Viagem)-[:TEM_BILHETE]->(b: Bilhete) WHERE v.idViagem = 1 AND b.data_vi

Rows | Text

b.lugar
3
2

Figura 39 – *Querie* 4 no Neo4j

```
11  /*Quais os lugares ocupados para viagem 1 no dia 2016-12-20 */
12  SELECT lugar FROM Bilhete
13  WHERE data_viagem = '20161220' AND viagem = 1;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

lugar
2
3

Figura 40 – *Querie* 4 em SQL

Como podemos verificar, o resultado da *querie* é o exatamente o mesmo, tal como era pretendido.

5. Conclusões

Este trabalho foi bastante importante e enriquecedor pois permitiu que os membros do grupo compreendessem e assimilassem melhor os conceitos da matéria relativos a execução de projetos de Sistemas Bases de Dados (SBD) não relacionais.

Inicialmente identificamos as entidades os seus atributos do sistema relacional, de maneira a compreender o comportamento da base de dados. Assim, conseguimos interpretar melhor o problema de modo a desenvolver o novo sistema não relacional que fosse equivalente ao esquema relacional implementado anteriormente.

De seguida, explicamos e definimos todo o processo de migração dos dados do SGB relacional para o não relacional, utilizando os ficheiros ".csv" e os comandos da linguagem de interrogação do Neo4j (*cypher)*. Foi nesta fase que foram evidenciadas algumas dificuldades em garantir a integridade dos dados, pois no modelo não relacional estas são garantidas pelos relacionamentos.

Por fim, foram implementadas e exemplificadas algumas *queries*, na linguagem especifica do Neo4j, de modo a demonstrar a execução do sistema, equiparando sempre com os resultados da mesma *querie* no sistema relacional.

Como podemos comprovar com a realização deste trabalho, um SBD não relacional é muito eficaz e rápido na organização de dados e ainda muito flexível. Por outro lado, a realização deste projeto permitiu-nos consolidar os conhecimentos adquiridos na unidade curricular, bem como identificar algumas lacunas que temos que corrigir futuramente.

Referências

[1] http://neo4j.com/docs/stable/tutorials.html

[2] http://www.quackit.com/neo4j/tutorial/

Lista de Siglas e Acrónimos

BD Base de Dados

SBD Sistema de Base de Dados

SGBD Sistema Gestor de Base de Dados

SBDR Sistema de Base de Dados Relacional