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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Reserva de viagens em comboios nacionais e internacionais (Neo4J)

Carlos Valente (a73929), Gustavo Andrez (a27748), Rogério Moreira (a74634), Samuel Ferreira (a76507)

Janeiro, 2017

D

Data de Recepção	
Responsável	
Avaliação	
Observações	

Reserva de viagens em comboios nacionais e internacionais

Carlos Valente (a73929), Gustavo Andrez (a27748), Rogério Moreira (a74634), Samuel Ferreira (a76507)

janeiro, 2017

Resumo

O projeto apresentado ao grupo consiste na implementação de uma base de dados em Neo4j a partir de uma já existente em MySQL.

O primeiro passo consistiu na exportação dos dados para ficheiro e posterior importação para no sistema NoSQL.

O segundo passo foi proceder à otimização, nomeadamente a eliminação de campos importados que deixam de ser necessários na nova estrutura e a criação de restrições.

O terceiro passo consistiu na experimentação da base de dados. Assim, é explorada a introdução de dados na BD bem como a elaboração de queries de consulta da informação.

Índice

Índice de Figuras

1. Introdução

O objetivo deste trabalho é a migração de uma base de dados relacional em MySQL para um sistema de bases de dados não relacional orientado por grafos - Neo4j.

A principal característica deste tipo de bases de dados é que não usam SQL, não seguindo o modelo relacional. As bases de dados não estão assim “presas” a uma rede de tabelas interligadas por chaves, são muito mais flexíveis, podendo adicionar dados mais facilmente, sem necessidade de reestruturar todas as tabelas.

Este novo sistema não relacional tem algumas características interessantes para o cliente e, por isso, a pedido, a base de dados foi migrada para este sistema. Algumas das qualidades de um sistema não relacional são:

- Facilidade em adicionar novos dados que não seguem uma estrutura rigorosa;
- Dados sempre disponíveis;
- Excelente maneira de lidar com o problema do Big Data;
- Menos manutenção, e consequentemente menos despesas (importante para o cliente).

2. Implementação da Base de Dados em Neo4j

Começámos o processo de migração por procurar casos de estudo anteriores de migrações de base de dados relacionais para Neo4j, incluindo a consulta da vasta documentação que o Neo4J disponibiliza para termos noção das boas práticas de migração entre sistemas de gestão de base de dados.

Daqui retiramos algumas regras que teríamos que manter durante toda a migração:

- Uma linha dá origem a um nodo
- Uma tabela dá origem a um tipo de nodo

O primeiro passo da migração passou por isso na exportação de todos os dados da base de dados em MySQL. Isto foi feito usando métodos de exportação da linguagem SQL. Em seguida apresentamos um exemplo de uma das queries utilizadas para exportar uma das tabelas:

```
SELECT * FROM Comboio
INTO OUTFILE '/tmp/comboio.csv'
FIELDS ENCLOSED BY '"' TERMINATED BY ',' ESCAPED BY ''
LINES TERMINATED BY '\r\n';
```

Contudo, tivemos alguns problemas devido às restrições de segurança do MySQL Server e a existência da variável *--secure-file-priv* que nos impedia de exportar dados das tabelas. Para isso tivemos que desativar esta variável no ficheiro de configurações pessoais do MySQL Server. Depois disto todas as tabelas da base de dados relacional foram exportadas para ficheiros *.csv*. Cada registo na base de dados relacional deu origem uma linha no ficheiro .csv da respetiva entidade. Quando a exportação foi terminada demos por terminada a 1º fase do processo de migração.

Em seguida demos início à fase da importação dos dados para o novo sistema. Em primeiro lugar importamos cada uma das tabelas nos ficheiros *.csv* usando comandos da linguagem Cypher que o Neo4J utiliza. De realçar que, devido ao facto de o Neo4J não ter o tipo *Date* nativo, todas as datas foram importadas na forma de *String*. em seguida apresentamos um exemplo de uma das queries utilizadas:

```
LOAD CSV FROM 'file:///reserva.csv' AS line
CREATE (reserva:Reserva {id: TOINT(line[0]) })
SET reserva.Custo=TOFLOAT(line[1]),
    reserva.Classe=TOINT(line[2]),
    reserva.Lugar=TOINT(line[3]),
    reserva.Carruagem=TOINT(line[4]),
```

```
    reserva.Nome=line[5],
    reserva.CheckIn=line[6],
    reserva.Utilizador_Email=line[7],
    reserva.Viagem_id=TOINT(line[8])
RETURN reserva;
```

A partir do procedimento acima descrito, importamos para o Neo4J todos os dados de todas as tabelas da base de dados relacional. Cada linha deu origem a um nodo e ficaram a existir 4 tipos de nodos diferentes: ***Comboio***, ***Reserva***, ***Viagem*** e ***Utilizador***.

De seguida iniciámos a fase da importação das relações. Num sistema de base de dados relacional as relações são feitas recorrendo a chaves estrangeiras o que não acontece num sistema de grafos não relacional como é o caso. Por isso, para criarmos as relações existentes recorrendo às chaves estrangeiras também importadas. As queries utilizadas foram:

```
MATCH (c:Comboio),
(v:Viagem)
WHERE c.id=v.Comboio_id
CREATE (c)-[:FAZ_VIAGEM]->(v);

MATCH (v:Viagem),
(r:Reserva)
WHERE v.id=r.Viagem_id
CREATE (r)-[:DE_UMA]->(v);

MATCH (u:Utilizador),
(r:Reserva)
WHERE u.email=r.Utilizador_Email
CREATE (u)-[:EFETUA_RESERVA]->(r);
```

A partir do momento em que as relações foram também importadas deixou de fazer sentido existirem chaves estrangeiras nos nodos e por isso todas as chaves estrangeiras nos diferentes tipos de nodos foram removidas. Além disso, os *ids* importados do modelo relacional foram também removidos uma vez que o próprio Neo4J atribui a cada nodo um *id* único. O utilizador foi a única entidade que manteve a sua chave primária, uma vez que, para além de representar a chave, corresponde ao *email* do utilizador. A decisão de eliminar os *ids* do modelo relacional deve-se ao facto de o Neo4J fazer toda a gestão dos *ids*, não sendo necessário incrementar em cada iteração esta label, nem de fazer a gestão de *ids* repetidos. Foram também removidas todas as *labels* dos diferentes nodos que apresentavam valores NULL, uma vez que neste modelo não relacional os nodos não têm todos que obedecer à mesma estrutura fixa sendo possível mudar a estrutura de apenas um nodo para adicionar uma label, por isto não faz sentido guardar valores NULL. Quando for necessário adicionar estes valores é criada uma nova label no respectivo nodo e o valor é guardado.

3. Apresentação da Base de Dados em Neo4j

Por forma a visualizarmos o grafo com todos os nodos armazenados, efetuamos o seguinte comando:

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

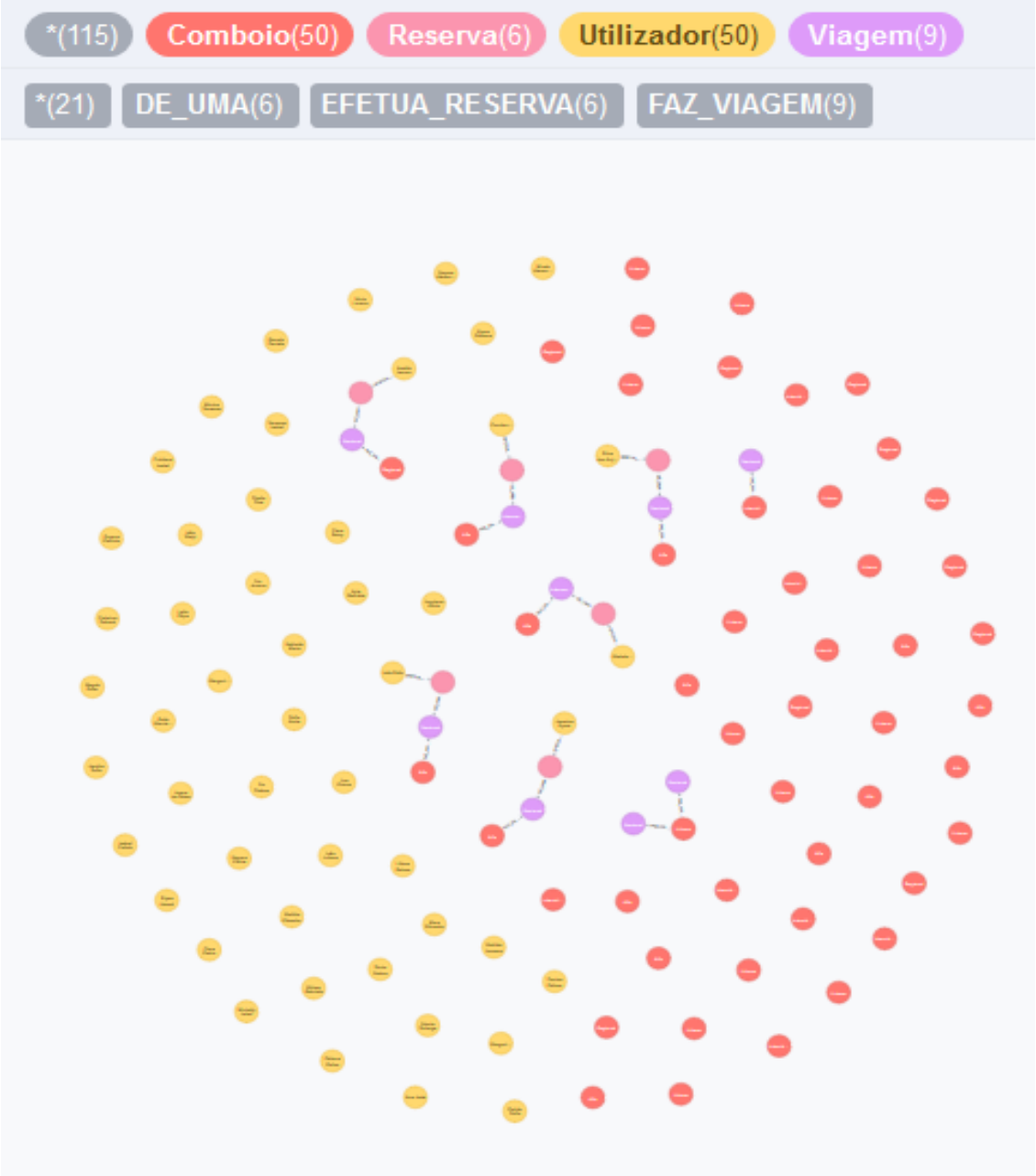

Figura 1 – Vista dos nodos e arcos da DB em Neo4j como grafo

Figura 2 – Vista dos nodos e arcos da DB em Neo4j como tabelas

De seguida apresentamos as operações feitas sobre a BD por forma a adicionar um nodo “Reserva” e respectivas relações:

- Adição de um nodo referente a uma reserva:

```
$ CREATE (ee:Reserva { Classe: "2", Lugar: "2", CheckIn:
  NULL, Custo:"24.25", Nome:"Samuel Costa", Carruagem:"1"})
```

Com esta query foi criado o nodo com id 115 com a informação da reserva.

- Adição de uma relação entre uma reserva e uma viagem:

```
$ match (n:Reserva), (m:Viagem) where ID(n)=115 and
  ID(m)=52 create (n)-[:DE_UMA]->(m)
```

Com esta query, foi criada a relação entre a reserva introduzida acima e a viagem associada (viagem com id 52).

- Adição de uma relação entre uma reserva e um utilizador:

```
$ match (n:Reserva), (m:Utilizador) where ID(n)=115 and
  ID(m)=102 create (m)-[:EFETUA_RESERVA]->(n)
```

Com esta query, foi criada a relação entre a reserva introduzida acima e o utilizador associado (utilizador com id 52).

Depois da adição deste nodo e das respetivas relações, obteve-se a seguinte árvore em que o nodo introduzido é a reserva do canto inferior esquerdo.

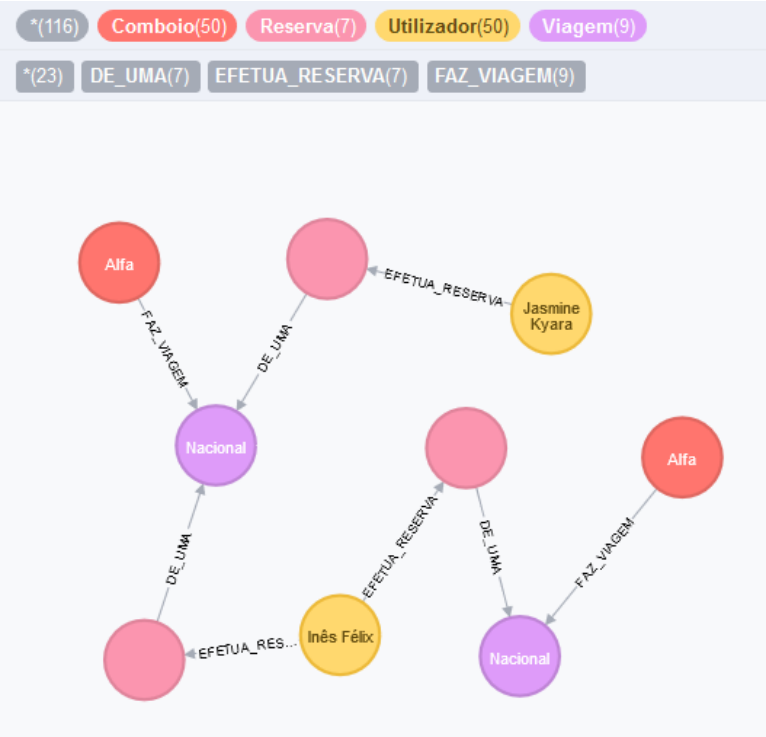

Figura 3 – Conjunto de nodos e suas relações

De seguida apresentamos a operação feita sobre a BD por forma adicionar uma restrição, como falado anteriormente:

- adição da restrição que obriga a que o email do utilizador seja único na BD:
-

```
$ CREATE CONSTRAINT ON (user:Utilizador) ASSERT user.email
  IS UNIQUE
```

Deste modo, não será possível a inserção de novos utilizadores com email já registados.

Depois de feita uma tentativa de inserção para testar a restrição, apareceu a seguinte mensagem de erro:

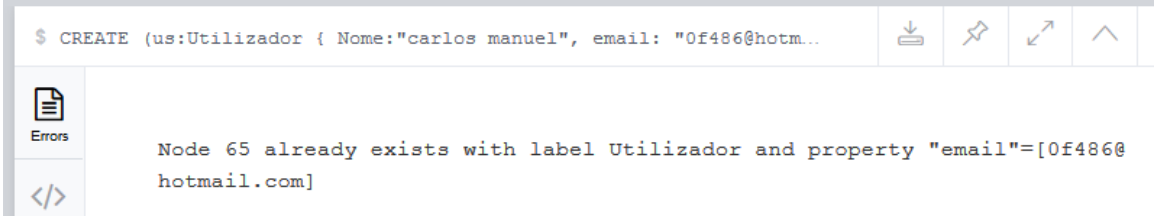

Figura 4 – Mensagem de erro aquando introdução de utilizador

4. Queries

Depois de implementada a base de dados em Neo4j foram elaboradas as seguintes queries:

1 - Nome dos utilizadores que compraram bilhetes para viagens com um certo destino anterior a uma determinada data (neste caso destino=Lisboa e data=00h do dia 4 de janeiro de 2017):

```
match (n1:Utilizador)-[rel:EFETUA_RESERVA]->(n2:Reserva)-[rel1:DE_UMA]->
(n3:Viagem)
where n3.Destino = "Lisboa" and n3.Chegada < "2017-01-04 00:00:00"
return n1.Nome
```

2 - Número total de reservas feitas para um determinado comboio (neste caso ID do comboio=1):

```
match (n1:Comboio)-[rel:FAZ_VIAGEM]->(n2:Viagem),(n3:Reserva)-[rel1:DE_UMA]-
>(n2:Viagem)
where ID(n1)=1
return count(n3) as soma
```

3 - Número total de viagens realizadas por um determinado comboio (neste caso ID do comboio=6) anterior a uma determinada data (data=00h do dia 4 de janeiro de 2017):

```
$ match (n1:Comboio)-[rel:FAZ_VIAGEM]->(n2:Viagem) where ID(n1)=6 and
  n2.Partida < "2017-01-04 00:00:00" return count(n1) as soma
```

4 - Qual o comboio que realizou certa viagem e qual o número de reservas associadas (neste caso viagem com id 52):

```
$ match (n1:Comboio)-[rel1:FAZ_VIAGEM]->(n2:Viagem), (n3:Reserva)-
  [rel2:DE_UMA]->(n2:Viagem) where ID(n2)=52 return count(n3) as
  reservas, ID(n1) as comboio
```

5. Comentários

A nova DB implementada é caracterizada por ser facilmente escalável e previsivelmente com melhor performance, característico das DB's não relacionais.

A linguagem utilizada na definição da queries pareceu-nos acessível e flexível.

Inevitavelmente, características como coerência e robustez da DB original foram postas em risco aquando da implementação da nova DB em Neo4j. Atendendo a este aspecto, consideramos positivo termos desenvolvido com sucesso uma restrição para evitar a repetição de emails na base de dados.

Com vista à implementação de queries com manipulação de datas, procurámos soluções para ultrapassar o facto de o Neo4j não possuir um tipo nativo de datas. A solução encontrada foi o uso de uma biblioteca externa – Apoc, no entanto não tivemos sucesso no uso desta biblioteca. Este foi também um dos pontos que menos gostamos no Neo4J uma vez que, em geral, nas bases de dados as datas são um tipo de dados importante e o facto do Neo4J não o resulta em que seja logo posto de parte aquando da escolha do sistema de bases de dados mais adequado para um determinado projeto.

6. Conclusão

Para este projeto em específico consideramos que a implementação BD relacional é a opção mais adequada devido a:

- O Neo4j não ter um tipo nativo de datas o que é bastante importante num projeto como é um sistema de reservas de bilhetes de comboios já que a gestão de datas e horas é uma parte crucial.
- à coerência e robustez da DB ser maior no modelo relacional;

Terminado este projecto consideramos que a DB desenvolvida cumpre os objetivos definidos, incluindo os registos iniciais bem como as relações existentes na DB original. Além disso é possível de uma forma simples, a manipulação e visualização dos dados.

Como trabalho futuro sugerimos:

- desenvolvimento de outras constraints por forma a promover a correção de

dados inseridos;

- povoamento da BD;
- desenvolvimento de mais queries;

Pelo referido, consideramos que os objetivos do trabalho foram cumpridos.