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

Unidade Curricular de Bases de Dados

Ano Letivo de 2017/2018

Sistema de gestão de eventos sociais em *NoSQL*

David Kramer A77849, Joaquim Simões A77653,

José Menezes A79187, Rafael Costa A61799

Janeiro, 2018

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

Sistema de gestão de eventos sociais em *NoSQL*

David Kramer A77849, Joaquim Simões A77653,

José Menezes A79187, Rafael Costa A61799

Janeiro, 2018

Resumo

Durante este relatório apresentamos, com detalhe, os vários passos efetuados no processo de migração do nosso sistema de base de dados relacional para um sistema não relacional em *Neo4j*.

Inicialmente apresentamos o problema em questão e os principais motivos que nos levaram a efetuar este processo de migração. De seguida, passamos à fase do processo de migração propriamente dito, descrevendo a estrutura das bases de dados orientadas a grafos. Ilustramos, para uma melhor perceção, um esquema que carateriza a nossa base de dados não relacional, bem como as instruções necessárias em *MySQL* e em *Neo4j* para o processo de migração. Após estes passos comparamos algumas *queries* desenvolvidas em *Neo4j*, que consideramos importantes, com as mesmas desenvolvidas em *MySQL*.

Concluímos depois o trabalho com uma breve apreciação critica do projeto nos diferentes temas abordados.

Área de Aplicação: Conceção de Sistemas de Bases de Dados Não Relacionais.
Palavras-Chave: MySQL, NoSQL, Neo4J, Bases de Dados Relacionais, Bases de Dados não Relacionais.

Índice

Índice de Figuras

1. Introdução

1.1. Contextualização

Nos dias de hoje, com a crescente evolução da tecnologia e do número de utilizadores ligados entre si, os requisitos são cada vez de maior quantidade e complexidade para uma base de dados. Assim, de modo a que as bases de dados pudessem ser eficientes e suportar um crescimento acentuado do número de utilizadores, surgiu um paradigma não relacional de armazenamento de dados, mais conhecido como *NoSQL.*

As bases de dados *NoSQL,* relativamente ao paradigma relacional, têm a capacidade de responder a pedidos sobre grandes quantidades de dados com grande eficiência, apresentam um desenvolvimento simples e possuiem uma escalabilidade horizontal, isto é, a escalabilidade é aumentada através de um aumento de número de nós que contribuem para o processamento.

Deste modo, foi decidido usar *Neo4j* para a implementação de uma base de dados *NoSQL*.

1.2. Apresentação do Caso de Estudo

Decidiu-se construir uma base de dados que permita gerir e consultar eventos sociais realizados numa qualquer região. Esta ideia surgiu com o objetivo de permitir obter, de uma maneira cómoda, a informação detalhada de quaisquer eventos organizados numa dada data, verificar quem os organizou e as pessoas que, por sua vez, irão atender a esses eventos.

Os eventos são caracterizados por um nome, e possuem uma ou mais edições. Por sua vez uma edição tem a si associada, um preço, um título, uma data de início e final do evento, uma lotação máxima, uma breve descrição do que se irá realizar, uma morada (composta por país, cidade e rua). Além disso, uma edição de um evento fornece estatísticas que informam a quantidade de mulheres *versus* homens que a irão frequentar, bem como a quantidade de pessoas que pertencem às diferentes faixas etárias. De notar que as faixas etárias consideradas consistem em: pessoas com menos de 18 anos, entre 18 a 25 anos, entre 25 a 30 anos e com mais de 30 anos.

Os eventos são criados por uma qualquer entidade organizadora que pode ser um qualquer tipo de estabelecimento ou uma câmara municipal. Uma organização tem a si associada uma morada (composta por país, cidade e rua), um nome, um *e-mail*, um ou mais contactos (quer sejam eles, números de telefone, telemóveis ou *websites*) e mantém as estatísticas gerais de todos os eventos organizados.

Finalmente, as pessoas (que podem ser vistas como clientes) vão a uma ou mais edições de eventos influenciando as estatísticas destas, assim como as estatísticas das organizações envolvidas na produção dos eventos. Uma pessoa é definida através de um nome, *e-mail*, data de nascimento, sexo, e tem a si associada uma ou mais fotos.

1.3. Fundamentação da implementação da base de dados

Uma das bases de dados orientadas a grafos mais utilizadas na atualidade é o *Neo4j.* Ao contrário de uma base de dados relacional, esta permite uma maior facilidade e simplicidade no seu desenvolvimento por não necessitar de tabelas ou de uma maior preocupação no tipo de dados de cada entidade. Torna-se muito útil também quando criamos uma base de dados que tenha bastantes relacionamentos, pois este processo pode ser feito em qualquer etapa e é tão simples como criar uma relação que "aponte" uma entidade para outra, não sendo mais uma vez necessário construir tabelas (para relacionamentos "*many-to-many"*), proceder à utilização de *foreign keys,* etc. ao contrário do que acontece numa base de dados relacional.

Por fim, temos talvez um dos maiores pontos positivos em utilizar este tipo de base de dados que são as consultas. Numa base de dados relacional, estas por vezes podem tornar-se excessivamente complexas e custosas que precisam de ser constantemente otimizadas. Enquanto que numa base de dados orientada a grafos é quase irrelevante o numero de nós que iremos considerar para uma determinada consulta (devido ao aumento da escalabilidade quanto maior o numero de nós). É por este motivo que este modelo é o mais adequado para dados com vários relacionamentos entre nós, como por exemplo redes sociais (um conceito muito idêntico ao explorado neste projeto).

2. Migração para Base de Dados em *Neo4j*

2.1. Estrutura da Base de Dados orientada a Grafos

O *Neo4j* é uma base de dados não relacional orientada a grafos. As bases de dados orientadas a grafos caracterizam-se por guardar os diferentes relacionamentos (incluindo a direção) entre as diferentes entidades por oposição a outros tipos de bases de dados.

Neste tipo de base de dados as diferentes entidades podem ser vistas como nós que contêm uma *label* (representa o nome de uma tabela numa base de dados em *SQL*) e um conjunto de propriedades, que representam os diferentes atributos de uma entidade. Estas diferentes propriedades são armazenadas seguindo o esquema "chave-valor" em que a "chave" identifica o nome do atributo e o "valor" representa a informação do atributo.

Os relacionamentos traduzem uma ligação entre dois nós distintos e possuem sempre uma direção, um tipo, um nó de início e um nó de fim e podem, tal como os nós, ter associadas a si um conjunto de propriedades distintas. Estes relacionamentos podem ser navegados independentemente das suas direções.

O facto de este tipo de base de dados manter a informação de como dos nós se relacionam entre si permite que certas propriedades em *SQL* sejam desnecessárias, tais como a criação de chaves estrangeiras e tabelas auxiliares que traduzem um relacionamento "*many-to-many"*.

2.2. Esquema da Base de Dados

A seguinte figura ilustra o esquema que traduz a nossa base de dados em *Neo4j*.

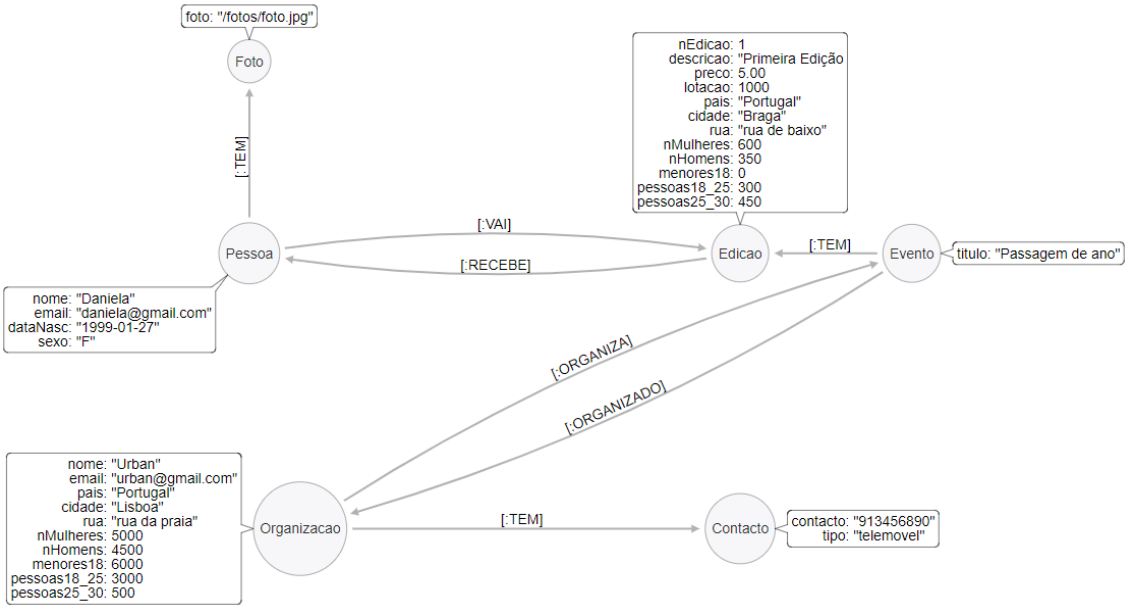

Figura 1 - Esquema da Base de Dados

O esquema ilustra a maneira como as diferentes entidades (nós) se relacionam entre si, e os seus diferentes atributos que representam as propriedades dos nós. A título ilustrativo atribuíram-se valores às diferentes propriedades presentes no esquema.

2.3. Processo de Migração

O processo de migração foi dividido em várias fases. Começou-se por guardar todos os registos presentes associados às diferentes tabelas do nosso esquema desenvolvido em *SQL* em ficheiros *CSV*. Para acelerar a construção das diferentes associações entre os diversos nós, criaram-se índices em certas propriedades chave dos nós. Após a criação dos índices procedeu-se ao povoamento da base de dados em *NoSQL*, começando por guardar a informação dos diversos nós e, no final, guardar o modo como os diferentes nós se relacionam entre si. As secções seguintes descrevem, com detalhe, cada uma das diferentes fases envolvidas no processo de migração.

2.3.1 Criação dos Ficheiros *CSV*

O *Neo4j* apresenta suporte para o povoamento da base de dados através de ficheiros com *CSV*. Neste processo, cada linha corresponde a um nó que contém um diverso conjunto de propriedades.

A criação dos diversos ficheiros *CSV* é bastante simples em *MySQL*. A seguinte figura ilustra as diversas instruções em *MySQL* necessárias para a conversão de todos os registos associados à tabela "Pessoa" para um ficheiro *CSV*:

```
# Verificar o caminho dos ficheiros
select @@GLOBAL.secure_file_priv;


# Pessoas
select 'id', 'email', 'nome', 'sexo', 'dataNasc'
union all
select * from pessoa
into outfile 'C:/ProgramData/MySQL/MYSQL Server 5.7/Uploads/persons.csv'
fields terminated by ','
enclosed by '"'
lines terminated by '\n';
```

Figura 2 - Criação do Ficheiro *CSV*

O ficheiro *CSV* produzido após esta instrução contém todos os registos associados à tabela "Pessoa" (segundo *select*) com um cabeçalho que contém os nomes dos diferentes atributos presentes no esquema da tabela (primeiro *select*). Este cabeçalho é de grande utilidade no povoamento da base de dados não relacional. Relativamente à localização do ficheiro de *output*, o *MySQL* apresenta uma regra de segurança que dita que os ficheiros de *output* produzidos devem ser guardados numa certa diretoria específica para o seu efeito. Neste caso foi indicado o caminho absoluto para essa mesma diretoria "C:/ProgramData/MySQL/MYSQK Server 5.7/Uploads/persons.csv". A instrução "select @@GLOBAL.secure_file_priv" apenas serviu para obtermos o caminho absoluto dessa diretoria. Este processo foi efetuado para todas as diferentes tabelas presentes no nosso antigo modelo.

2.3.2 Criação de Índices

A criação de índices sobre certas propriedades chave dos diversos nós serviu para uma maior performance na criação dos diversos relacionamentos entre os nós. Estes índices foram apenas temporários, já que estes apenas serviram para tornar a procura dos nós mais eficiente. A criação de todos os índices é dada pelas seguintes expressões:

```
CREATE INDEX ON :Pessoa(id);
CREATE INDEX ON :Foto(pessoaID);
CREATE INDEX ON :Evento(id);
CREATE INDEX ON :Organizacao(id);
CREATE INDEX ON :Contacto(orgID);
CREATE INDEX ON :Edicao(eventoID);
CREATE INDEX ON :Edicao(nEdicao);
```

Figura 3 - Criação dos Índices

2.3.3 Criação dos Nós

A criação dos diversos nós, caracteriza-se pela leitura, linha a linha, de cada um dos diferentes ficheiros *CSV* criados. O processo de povoamento relativo à entidade "Pessoa" é dado pelas seguintes expressões:

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///persons.csv" AS line
CREATE (p:Pessoa {id: toInteger(line.id), email: line.email,
    nome: line.nome, sexo: line.sexo, dataNasc: line.dataNasc});
```

Figura 4 - Criação dos Nós

O ficheiro é carregado com o cabeçalho que identifica os nomes dos diversos atributos envolvidos o que se torna bastante útil na criação das diferentes propriedades do nó. De notar que todas diferentes propriedades são armazenadas como *Strings* por defeito. Posto isto recorreu-se às funções *toInteger() e toFloat()* que convertem, respetivamente, os valores das propriedades para inteiros e números reais.

2.3.4 Criação dos Relacionamentos entre Nós

A criação dos relacionamentos foi efetuada após a criação dos diversos nós, já que um relacionamento entre quaisquer nós só pode ser feito após a criação desses mesmos nós. Os relacionamentos foram definidos através de um nome (o mais intuitivo possível) e de uma direção. As seguintes expressões ilustram a criação dos relacionamentos envolvidos entre os nós que caracterizam uma "Pessoa" e os nós que caracterizam uma "Edição":

```
USING PERIODIC COMMIT
LOAD CSV WITH HEADERS FROM
"file:///personsEditions.csv" AS line
MATCH (p:Pessoa {id: toInteger(line.pessoaID)})
MATCH (ed:Edicao {eventoID: toInteger(line.eventoID), nEdicao: toInteger(line.nEdicao)})
CREATE (p)-[:VAI]->(ed), (ed)-[:RECEBE]->(p);
```

Figura 5 - Relacionamento entre uma Pessoa e uma Edição

Na criação dos relacionamentos começa-se por verificar se os diferentes nós envolvidos existem (instrução "*MATCH*"). O relacionamento é criado através da sintaxe *no1-[Relacionamento]->no2*. Por exemplo, na figura ilustrada temos que um nó qualquer "p" tem um relacionamento com um nó "ed" que é chamado "VAI" e que a direção do relacionamento é do nó "p" para o nó "ed". De notar que foi também criado um relacionamento no sentido inverso ("ed RECEBE p"). Este caso ilustra um relacionamento "*many-to-many*".

2.3.5 Remoção de Atributos desnecessários

Os passos descritos nas secções anteriores foram os necessários para a migração da nossa base de dados em *MySQL* para a base de dados em *Neo4j.* No entanto teve-se em consideração a remoção de informação redundante de modo a reduzir a quantidade de memória ocupada pelos dados envolvidos no nosso sistema de base de dados. Tal como foi dito, todos os índices criados apenas serviram para aumentar a performance do processo de criação dos diversos relacionamentos entre os nós. Ora, como todas os relacionamentos já foram criados, estes índices podem ser removidos. Adicionalmente, certos atributos (como as chaves primárias caracterizadas por identificadores necessárias no modelo relacional) deixam de ser necessários e podem também ser eliminados. Este processo de remoção é caracterizado pela seguinte figura:

```
MATCH (n)
REMOVE n.id;

MATCH (n)
remove n.eventoID;

MATCH (n)
remove n.pessoaID;

MATCH (n)
remove n.orgID;
```

Figura 6 - Remoção dos Atributos e dos Índices desnecessários

De notar que a primeira instrução remove, de uma maneira bastante minimalista, todos os atributos de nome "id" de qualquer nó presente no sistema de base de dados.

3. *Queries* em *Neo4j*

A realização das *queries* em *Neo4j* foi mais simples do que em MySQL. Nas passagens seguintes apresentamos três *queries* que representam as diferenças entre o *MySQL* e o *Neo4j,* nomeadamente a maior simplicidade desta base de dados não relacional.

3.1. Eventos que decorram num dado intervalo

Neo4j:

```
MATCH (ev:Evento)-[:TEM]->(ed:Edicao)
WHERE (split(ed.fim, " ")[0] >= "2017-10-01" AND split(ed.fim, " ")[0] <= "2018-05-31") OR
    (split(ed.inicio, " ")[0] >= "2017-10-01" AND split(ed.inicio, " ")[0] <= "2018-05-31")
RETURN ev, ed;
```

Figura 7- *Neo4j*- primeira *query*

Através da operação *match* determinamos os eventos e suas edições, enquanto que através do *where* filtramos essas edições e eventos para um dado intervalo. No fim, a operação *return* devolve os eventos e edições num dado intervalo.

MySQL:

```
select *
from Evento
join Edicao on Evento.id = Edicao.eventoID
where (date(Edicao.fim) >= '2017-10-01'  and date(Edicao.fim) <= '2018-05-31') or
(date(Edicao.inicio) >= '2017-10-01' and date(Edicao.inicio) <= '2018-05-31');
```

Figura 8- *MySQL*- primeira *query*

3.2. Pesquisar por eventos e suas edições segundo um título

Neo4j:

```
MATCH (ev:Evento)-[:TEM]->(ed:Edicao)
WHERE ev.titulo CONTAINS "Noite"
RETURN ev, ed;
```

Figura 9- Neo4j- segunda query

Através da operação *match* determinamos os eventos e suas edições, enquanto que através do *where* filtramos os eventos que contêm no seu título a *string* “Noite”. No fim, a operação *return* devolve os eventos calculados e suas edições.

MySQL:

```
select *
from Edicao
INNER JOIN Evento on Edicao.eventoID=Evento.id;
```

Figura 10- *MySQL*- segunda *query*

3.3. Listar as pessoas (e as suas fotos) que vão a um evento e as suas organizações

Neo4j:

```
MATCH (o:Organizacao)-[:ORGANIZA]->(ev:Evento)-[:TEM]->(ed:Edicao)
    -[:RECEBE]->(p:Pessoa)-[:TEM]->(f:Foto)
WHERE ev.titulo CONTAINS "Noite"
RETURN ev, ed, p, f, o;
```

Figura 11- *Neo4j*- terceira *query*

Através da operação *match* determinamos as organizações que organizam eventos, eventos estes que contêm edições às quais pessoas (com fotos) vão, enquanto que através do *where* filtramos as edições que contenham no seu título a *string* “Noite”. No fim, a operação *return* devolve os eventos calculados e a Organização, edição e pessoas (com foto) correspondentes.

MySqI:

```
select  *
from OrganizacaoEvento
JOIN Organizacao on Organizacao.id= OrganizacaoEvento.orgID
JOIN Evento on Evento.id= OrganizacaoEvento.eventoID
JOIN Edicao on Edicao.eventoID= Evento.id
JOIN PessoaVaiEdicao on PessoaVaiEdicao.eventoID= Edicao.eventoID
JOIN Pessoa on Pessoa.id = PessoaVaiEdicao.pessoaID
JOIN Foto on Foto.pessoaID= PessoaVaiEdicao.pessoaID;
```

Figura 12- *MySQL*- terceira *query*

4. Conclusões e Análise Crítica

Com a realização deste trabalho exploramos as características que definem as bases de dados não relacionais e verificamos as vantagens da sua utilização, nomeadamente face à escalabilidade de um projeto.

Estudamos os conceitos e a estrutura de uma base de dados orientada a grafos e, verificamos que este tipo de base de dados tem vindo a ser cada vez mais utilizada por grandes empresas, sendo uma enorme motivação para o nosso projeto.

Relativamente ao processo de migração para uma base de dados em *Neo4j*, descobrimos que este se tornava bastante simples com recurso a ficheiros *CSV.* Isto fez com que, através de algumas instruções em *MySQL*, o processo de criação destes ficheiros fosse efetuado de uma maneira bastante cómoda e simples. Tentamos, sem sucesso, tornar essas instruções genéricas para todas as relações contidas no nosso esquema relacional sem ter que se fornecer o caminho absoluto do ficheiro de *output.* Através dos ficheiros *CSV* produzidos a partir dos nossos dados, o processo de povoamento da base de dados em *Neo4j* foi efetuado sem quaisquer dificuldades.

Finalmente, tendo concluído o processo de migração com sucesso, decidimos comparar certas instruções de procura, inserção e remoção em *Neo4j* com as mesmas desenvolvidas em *MySQL*. Verificamos então, que o *Neo4j* fornece um conjunto de instruções bastante mais poderoso e intuitivo que o *MySQL* principalmente no que toca a consultas mais complexas que envolvem a junção de múltiplas tabelas.

Referências

Connoly, T.M. and Begg, C.E., 2015. *Database Systems, A Pratical Approach to Design, Implementation, and Management.* 6th ed. Pearson.

Referências WWW

https://neo4j.com

https://www.couchbase.com/resources/why-nosql

https://dev.mysql.com/doc/refman/5.7/en/tutorial.html

Lista de Siglas e Acrónimos

BD Base de Dados

DW Data Warehouse

SQL Structured Query Language

NoSQL Not Only Structured Query Language