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

Unidade Curricular de Bases de Dados

Vacinação em Portugal

Neo4J

João Monteiro

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Vacinação em Portugal

João Monteiro, Andreia Alves

Novembro, 2017

Resumo

O presente relatório tem como objetivo fazer uma apresentação da abordagem do grupo à implementação de um sistema não relacional de bases de dados para um caso de estudo específico.

O grupo pretende modificar uma base de dados relacional(MySQL), do plano de vacinação nacional, para uma base de dados não relacional. Nesta fase, o grupo começou por estudar o modelo lógico da base de dados relacional e planear a transferência desta para o modelo de grafos usado pelo Neo4J.

Para a migração dos dados usou-se os componentes etl do Neo4J, povoando a base de dados. Após a migração definiu-se e implementou-se algumas queries em Cypher para o teste da base de dados.

Finalmente, fez-se uma análise crítica do trabalho.

Área de Aplicação: Arquitetura e Migração de Sistemas de Bases de Dados Não Relacional.
Palavras-Chave: Base de Dados, MySQL, Neo4J, Migração de Dados, Cypher.

Índice

Índice de Figuras

1. Introdução

Tal como foi referido no resumo anteriormente apresentado, o grupo pretende implementar um sistema de base de dados não relacional do plano nacional de vacinação. Neste capitulo serão apresentados os objetivos deste projeto.

Este trabalho consiste no estudo dos modelos não relacionais e as suas diferenças para os modelos relacionais. Assim, criamos um modelo equivalente entre as duas bases de dados e aplicamos a migração dos dados do modelo relacional (MySQL) para o modelo não relacional (Neo4J). Com implementação de algumas queries para o teste deste no final.

No segundo capitulo, falaremos do modelo usado para a implementação da base de dados no Neo4J, as suas diferenças e semelhanças com o modelo MySQL.

No terceiro capitulo, apresentaremos as ferramentas utilizadas para a migração de dados entre a base de dados MySQL e o Neo4J.

No quarto capitulo, mostramos alguns exemplos de queries implementados no Neo4J para o teste da Base de Dados.

Finalmente, terminamos com a análise crítica do projeto apresentado.

2. Modelo de dados em Neo4J

O modelo de dados em Neo4j, orientado a grafos, foi concebido a partir do modelo conceptual e do modelo lógico relacional obtidos no trabalho anterior. Na Figura 1 encontra-se o modelo de dados lógico relacional.

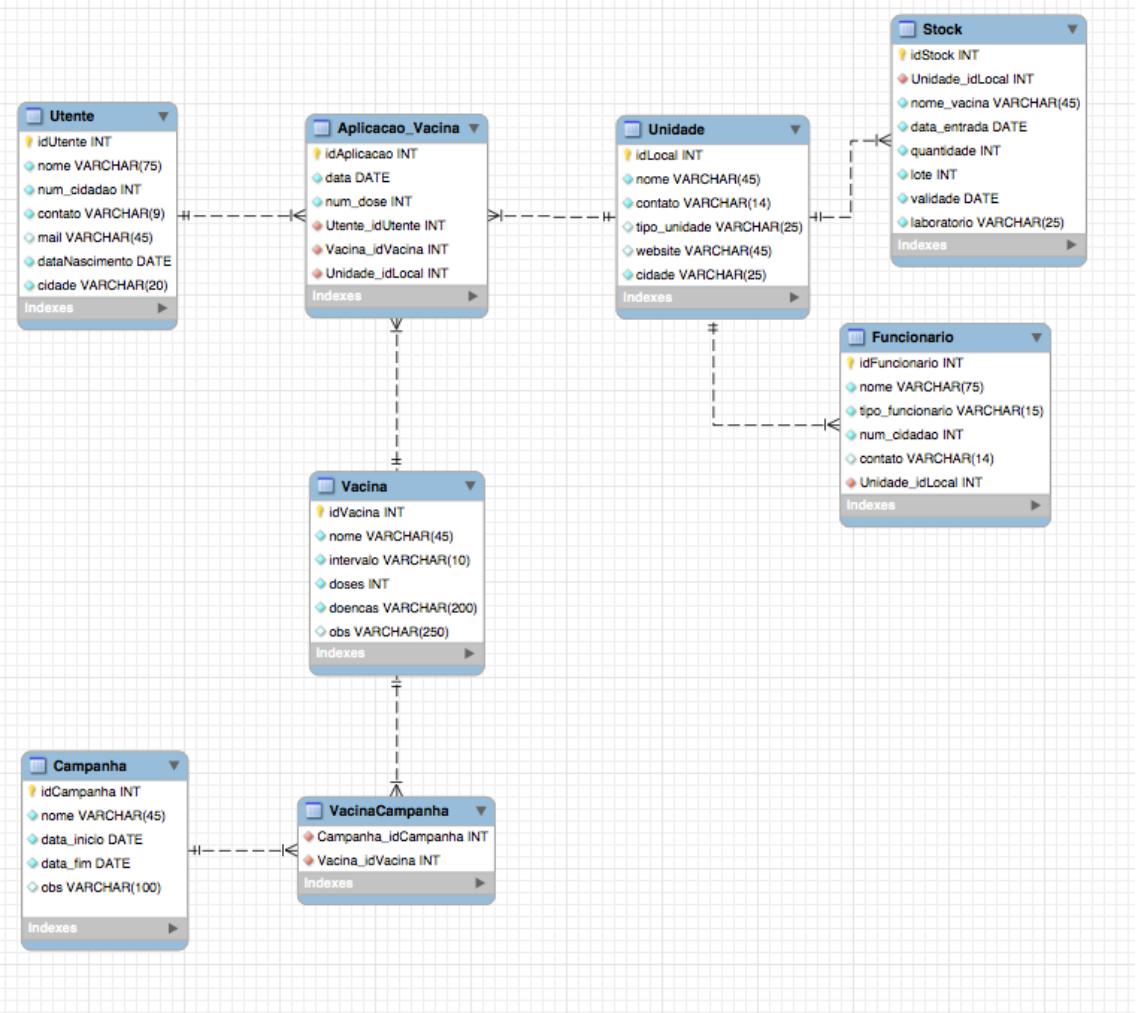

Figura 1 – Diagrama do Modelo Lógico Relacional

Em Neo4j é importante referir:

- **Nodos ou Nós:** são os registos (correspondentes às linhas das tabelas no relacional);
- ***Labels:*** são conjunto agrupados de nodos
- **Relacionamentos:** são as conexões entre os nodos
- **Propriedades:** são definições de um nodo (visto como os atributos no relacional)

2.1. Identificar os Nodos e os Relacionamentos

Todas as informações são armazenadas em tabelas => No Neo4j, armazenam-se as informações em nodos com propriedades. As relações entre itens de informações armazenados (linhas em tabelas) são indicadas por chaves que unem as tabelas => No Neo4j, armazenam-se as relações entre nodos de maneira explícita como relacionamentos. As **Chaves primárias**: identificam registros exclusivos em cada tabela. As **Chaves Estrangeiras** identificam Relacionamentos entre Tabelas. Cada chave estrangeira tem uma fonte como a tabela de início e o alvo como a tabela final, em outras palavras, o nodo inicial e o nodo final de um relacionamento no Neo4j.

A estrutura de algumas tabelas representa uma tabela *JOIN*, em que uma tabela é unida com outra tabela através de uma tabela provisória. Isso geralmente é feito para representar um relacionamento de muitos para muitos ou às vezes feito como um exercício de normalização.

Por exemplo, VacinaCampanha é uma tabela *JOIN* para significar uma associação entre Vacina e Campanha. Se seguíssemos as regras que anteriormente estabelecemos, acabaríamos com três nodos: Vacina, VacinaCampanha e Campanha. Mas no Neo4j, os relacionamentos são entidades de primeira classe, então podemos ignorar o nodo interino e, em vez disso, podemos importá-lo como: *(Vacina)-[:REFERENTE]->(Campanha)*.

Há tabelas que aparecem como tabelas *JOIN*, mas que unem mais de duas tabelas, ou seja, elas contêm mais de duas chaves estrangeiras. A tabela Aplicacao_Vacina pode ser vista como uma tabela intermediária que conecta as tabelas Utente, Unidade e Vacina. No Neo4j, essa tabela é importada como um nodo. Muitas vezes, essas são entidades ou conceitos "faltantes" em seu domínio. As chaves estrangeiras são transformadas em relacionamentos como esperado e a tabela *JOIN* é importada como um nodo intermediário.

Resumidamente:

- Todas as informações são armazenadas em tabelas => No Neo4j, armazenam-se informações em nodos com propriedades.
- As relações entre itens armazenados de informações (linhas em tabelas) são indicadas por restrições que vinculam tabelas em conjunto => No Neo4j, armazenam-se relações entre nodos de forma explícita como relacionamentos.
- Interprete-se as restrições como *JOINs* ou *JoinTables*.
- Armazena-se as *JOINs* como relacionamentos.
- *JoinTables* que têm exatamente duas chaves estrangeiras são armazenados como relacionamentos.
- Tabelas que correspondem ao caso do nodo intermédio (mais de duas Chaves Estrangeiras) são importadas como nodos e as *JOINs* para as outras tabelas são armazenados como relacionamentos.

Com isto obteve-se:

- **7 nodos** no modelo de dados Neo4j: (Utente, Aplicacao_Vacina, Unidade, Vacina, Campanha, Stock e Funcionario), com as respetivas propriedades (atributos no modelo relacional);

- **6 relacionamentos**:
  - (Aplicacao_Vacina)-[:RECEBIDA_POR]->(Utente)
  - (Aplicacao_Vacina)-[:APLICADA_EM]->(Unidade)
  - (Aplicacao_Vacina)-[:DE_UMA]->(Vacina)
  - (Funcionario)-[:TRABALHA_EM]->(Unidade)
  - (Stock)-[:DE_CADA]->(Unidade)
  - (vacina)-[:REFERENTE]->(Campanha)

Assim, tem-se:

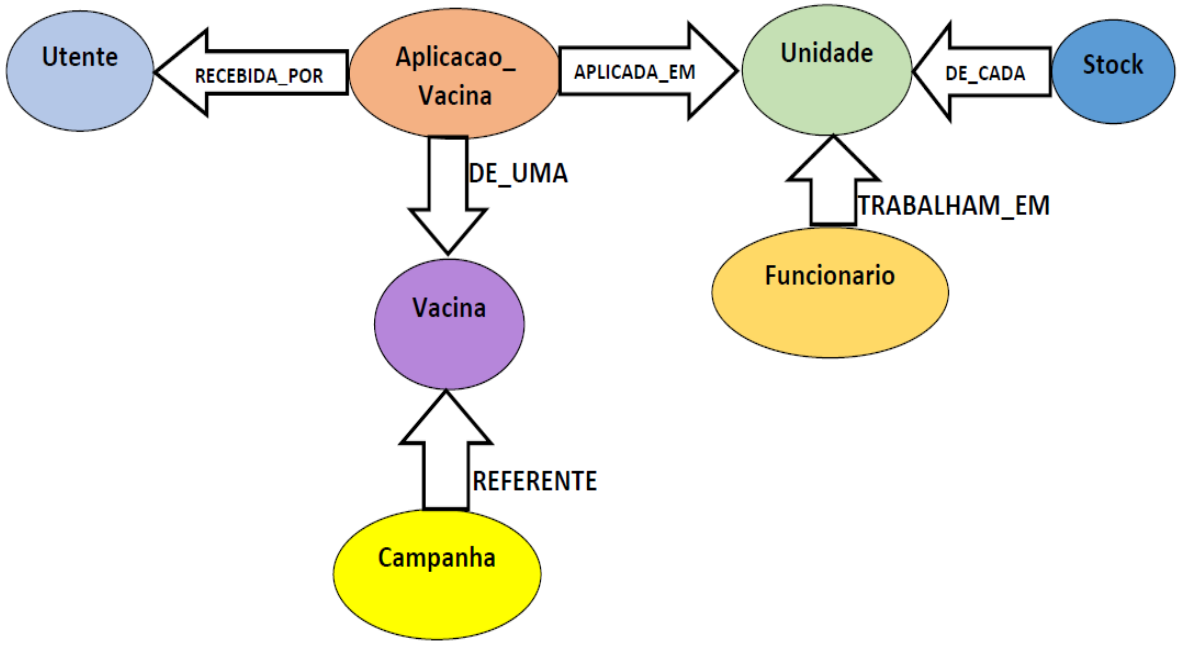

Figura 2 – Diagrama em grafos equivalente ao modelo relacional

3. Processo de migração de Dados do MySQL para o Neo4J

Após a definição do esquema da base de dados *NoSQL*, o próximo passo foi proceder à migração dos dados da base de dados relacional para o novo sistema não relacional.

Depois de alguma pesquisa, concluiu-se que existem várias formas para realizar a migração de dados:

- utilizando a ferramenta *etl-tool* do Neo4j, que com um único comando da *shell* faz a importação e mapeamento das tabelas do relacional para o modelo de dados do Neo4j;

- usando o *LOAD CSV* que faz a exportação das tabelas do modelo relacional para o formato CSV, para depois usar os mesmos ficheiros ".csv" para fazer a importação dos dados para o Neo4j, usando os comandos da linguagem de interrogação *Cypher* (utilizada pelo Neo4j). De notar que cada tabela é exportada para um ficheiro CSV individual correspondente;

- usando o JDBC Driver para conectar com a base de dados do MySQL e a linguagem de interrogação *Cypher* para importar diretamente as tabelas em SQL para Neo4j:

  ➔ faz-se o registo do JDBC Driver com o comando *Cypher* no *browser* Neo4j:

  CALL apoc.load.driver("com.mysql.jdbc.Driver");

  ➔ começar a importar as tabelas do modelo relacional, convertidos para o modelo de dados do neo4j, onde cria-se os nodos e depois os relacionamentos. Um exemplo:

  //Criação do nodo Utente
  CALLapoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password,
  "Utente") YIELD row
  CREATE (:Utente {idUtente: toInteger(row.idUtente), nome: row.nome, contato: row.contato, mail: row.mail, dataNascimento: row.dataNascimento,
  cidade: row.cidade});

4. Queries

Primeiramente, criou-se os nodos

```
//Utente
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Utente") YIELD row
CREATE (:Utente {idUtente: toInteger(row.idUtente), nome: row.nome, contato: row.contato, mail: row.mail,
dataNascimento: toString(row.dataNascimento), cidade: row.cidade});
```

```
//Unidade
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Unidade") YIELD row
CREATE (:Unidade {idLocal: toInteger(row.idLocal), nome: row.nome, contato: row.contato, tipo_unidade:
row.tipo_unidade, website: row.website, cidade:row.cidade});
```

```
//Vacina
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Vacina") YIELD row
CREATE (:Vacina {idVacina: toInteger(row.idVacina), nome: row.nome, intervalo: row.intervalo, doses:
toInteger(row.doses), doencas: row.doencas, obs: row.obs});
```

```
//Aplicacao_Vacina
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Aplicacao_Vacina") YIELD row
CREATE (:Aplicacao_Vacina {idAplicacao: toInteger(row.idAplicacao), data: toString(row.data), contato:
row.contato, num_dose: toInteger(row.num_dose)});
```

```
//Campanha
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Campanha") YIELD row
CREATE (:Campanha {idCampanha: toInteger(row.idCampanha), nome: row.nome, data_inicio:
toString(row.data_inicio), data_fim: toString(row.data_fim), obs: row.obs});
```

```
//Stock
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Stock") YIELD row
CREATE (:Stock {idStock: toInteger(row.idStock), nome_vacina: row.nome_vacina, data_entrada:
toString(row.data_entrada), quantidade: toInteger(row.quantidade), lote: toInteger(row.lote), validade:
toString(row.validade), laboratorio: row.laboratorio});


//Funcionario
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Funcionario") YIELD row
CREATE (:Funcionario {idFuncionario: toInteger(row.idFuncionario), nome: row.nome, tipo_funcionario:
row.tipo_funcionario, num_cidadao: toInteger(row.num_cidadao), contato: row.contato});
```

Seguiu-se depois para a criação dos relacionamentos:

```
//relação funcionario -> unidade
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Funcionario") YIELD row
MATCH (f:Funcionario {idFuncionario: row.idFuncionario})
MATCH (un:Unidade {idLocal: row.idLocal})
MERGE (:Funcionario)-[:TRABALHA_EM]->(:Unidade);

//realção stock -> unidade
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Stock") YIELD row
MATCH (s:Stock {idStock: row.idStock})
MATCH (un:Unidade {idLocal: row.idLocal})
MERGE (s)-[:DE_CADA]->(un);


 //relação aplicacao -> utente
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Aplicacao_Vacina") YIELD row
MATCH (u:Utente {idUtente: row.idUtente})
MATCH (ap:Aplicacao_Vacina {idAplicacao: row.idAplicacao})
MERGE (ap)-[:RECEBIDA_POR]->(u);

//relação aplicacao -> vacina
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Aplicacao_Vacina") YIELD row
MATCH (v:Vacina {idVacina: row.idVacina})
MATCH (ap:Aplicacao_Vacina {idAplicacao: row.idAplicacao})
MERGE (ap)-[:DE_UMA]->(v);
```

```
//relação aplicação -> unidade
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","Aplicacao_Vacina") YIELD row
MATCH (un:Unidade {idLocal: toInteger(row.idLocal)})
MATCH (ap:Aplicacao_Vacina {idAplicacao: toInteger(row.idAplicacao)})
MERGE (ap)-[:APLICADA_EM]->(un);

 //relação campanha -> vacina
CALL apoc.load.jdbc("jdbc:mysql://localhost:3306/vacinas?user=root&password","VacinaCampanha") YIELD row
MATCH (v:Vacina {idVacina: row.idVacina})
MATCH (c:Campanha {idCampanha: row.idCampanha})
MERGE (c)-[:REFERENTE]->(v);
```

Com isto, já pode-se fazer algumas ***queries*** pretendidas sobre a base de dados:

- **Listar toda a informação de um utente específico**

```
MATCH (u:Utente)
WHERE idUtente(u) = 1
RETURN u;
```

- **Listar todos os utentes que sejam de Braga**

```
MATCH (u:Utente)
WHERE cidade(u) = “Braga”
RETURN u;
```

- **Listar as aplicações de vacinas tomadas por um cliente específico identificado pelo seu id, e as unidades de saúde onde foram feitas**

```
MATCH (u:Utente {u.IdUtente})
MATCH (ap:Aplicacao_Vacina {ap.IdAplicacao})
MATCH (un:Unidade {un.IdLocal})
MERGE (ap)-[r1:APLICADA_EM]-(un)
MERGE (ap)-[r2:RECEBIDA_POR]-(u)
RETURN  r1, r2;
```

- **Atualizar a propriedade de um dado nodo ou inserir se não existir**

```
MATCH (un:Unidade)
WHERE un.idLocal = 1
SET un.website = “www.privadodebraga.pt”
```

- **Listar todas as vacinas disponíveis na base de dados**

```
MATCH (v:Vacina)
RETURN v;
```

5. Análise Crítica do Trabalho Realizado

Após realizado o trabalho de modelação e implementação da base de dados na forma relacional no MySQL e na forma não relacional em Neo4j, analisar-se-à e comparar-se-à as duas abordagens em paralelo.

No que toca ao modelo conceptual, verificamos que este se adequa às duas abordagens, traduzindo os requisitos e conceitos do trabalho de forma adequada. Dessa forma, foi mais fácil definir o modelo lógico
tanto na sua forma relacional, como na sua forma não relacional.

De acordo com o que foi definido na secção anterior sobre o esquema da base de dados não relacional, podemos verificar que os dois modelos em MySQL e Neo4j são quase iguais, aplicando as regras de conversão de cada tabela relacional à nodos e relacionamentos referidos na Secção 2.1 do presente relatório.

A migração entre os dois sistemas de dados requeriu maior atenção e tempo dsipendido devido à problemas de configuração do Neo4j no sistema operativo utilizado (MAC OS X). Depois de várias tentativas usando as formas possíveis de migração de dados referidas anteriormente, optou-se por utilizar o JDBC Driver para o mesmo, pois foi o que mais se aproximou dos objetivos propostos. Uma dificuldade foi o facto de que no Cypher não ter um equivalente para dados do tipo Date dos atributos do relacional, onde esse campo era migrado como sendo null. Para ultrapassar isso, decidiu-se invocar a função toString() do Cypher e tratar as datas como sendo texto.

Relativamente às queries, foi utilizado o CQL (Cypher Query Language) que é uma linguagem declarativa, inspirada em SQL, para descrever visualmente os padrões em grafos. Isso permite-nos indicar o que queremos selecionar, inserir, atualizar ou excluir dos dados do grafo sem que precisemos descrever exatamente como fazê-lo. Queries simples que apenas necessita-se de obter informação dentro de um dado nodo ou label, são fáceis de fazer. Por outro lado, queries para apresentar as JOINS das tabelas do relacional pode ser uma operação mais custosa.

Sendo o SQL na sua generalidade uma linguagem usada por todos os sistemas de base de dados relacionais, torna-se mais fácil a sua migração. No mundo não relacional, parece ser normal cada sistema adotar a sua própria linguagem de “query”, também devido ao facto de haver vários modelos de dados diferentes, como modelos de chave-valor, documentos, etc.

Referências

- https://neo4j.com/developer/get-started/
- https://www.tutorialspoint.com/neo4j/
- https://neo4j.com/developer/cypher-query-language/
- http://www.developeradvocate.com/2017/03/rdbms-to-graph-database/
- https://neo4j.com/developer/graph-db-vs-rdbms/
- https://dzone.com/refcardz/from-relational-to-graph-a-developers-guide

Lista de Siglas e Acrónimos

JDBC Java Database Connectivity

SQL Structured Query Language

NoSQL Not Only SQL

CQL Cypher Query Language