



Universidade do Minho

Mestrado Integrado em Engenharia Informática
Licenciatura em Ciências da Computação

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Sistema de Reservas de Viagens

**Bruno Pereira (a75135), Diogo Silva (a76407), João
Gonçalves (a76290), Maria Ana de Brito (a73580)**

Janeiro, 2017

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Sistema de Reservas de Viagens

Bruno Pereira (a75135), Diogo Silva (a76407), João Gonçalves (a76290), Maria Ana de Brito (a73580)

Janeiro, 2017

Resumo

Nesta segunda fase do projeto prático, foi-nos proposto fazer a migração da base de dados relacional criada anteriormente para uma base de dados não relacional, mais especificamente em MongoDB. Em relação ao teorema CAP das BD não relacionais, o MongoDB assume-se mais orientado para o CP, isto é, “Consistency” e “Partition Tolerance”, uma vez que todas as “queries” em si realizadas apresentam ou a informação mais recente ou um erro. Além disso se uma das partições falhar, o sistema não falha por completo.

Primeiramente, o grupo fez o planeamento do esquema da base de dados não relacional, pois durante a migração teriam de ocorrer algumas mudanças na forma como os dados são armazenados, uma vez que agora estamos a operar num paradigma diferente.

De seguida, foi criado um “script” para realizar a migração dos dados de uma BD relacional (SQL) para uma BD não relacional (NoSQL) orientada aos documentos.

Por fim, foi definido um conjunto de “queries” que mostram a operacionalidade da nova base de dados.

Área de Aplicação: Desenvolvimento e implementação de um sistema de base de dados não relacional.

Palavras-Chave: JAVA, MongoDB, BD, NoSQL, Document Stores, MySQL, Não Relacional

Índice

1. Introdução	1
1.1. Contextualização	1
1.2. Apresentação do Caso de Estudo	1
1.3. Motivação e Objectivos	2
1.3.1 Motivação	2
1.3.2 Objectivos	2
1.4. Planeamento	3
1.5. Estrutura do Relatório	3
2. Desenvolvimento do Projeto	4
2.1. Desenvolvimento do Esquema para a Base de Dados	4
2.1.1 Documento para Cliente	5
2.1.2 Documento para Comboio	6
2.1.3 Documento para Viagem	7
2.2. Migração dos Dados	8
2.2.1 Script de Migração	8
2.3. Conjunto de Queries	13
2.3.1 Query 1	13
2.3.2 Query 2	14
2.3.3 Query 3	15
2.3.4 Query 4	16
2.3.5 Query 5	17
2.3.6 Query 6	17
2.3.7 Query 7	18
2.3.8 Query 8	19
3. Conclusões	20

Anexos

I. Anexo 1

Erro!

Marcador não definido.

Índice de Figuras

Figura 1 - Esquema da Base de Dados Não Relacional	4
Figura 2 - Documento Cliente	5
Figura 3 - Documento Comboio	6
Figura 4 - Documento Viagem	7
Figura 5 - Query 1	13
Figura 6 - Query 2	14
Figura 7 - Query 3	15
Figura 8 - Query 4	16
Figura 9 - Query 5	17
Figura 10 - Query 6	17
Figura 11 - Query 7	18
Figura 12 - Query 8	19

Índice de Tabelas

Tabela 1 - Planeamento

3

1. Introdução

Neste capítulo, está definido o problema que nos foi proposto, bem como a sua contextualização. Além disso, também referimos a motivação para a implementação da base de dados não relacional e as suas vantagens relativamente à relacional.

1.1. Contextualização

A *PolkaTerra* é uma empresa ferroviária checa, que se pretende sediar em Portugal. Parte dos seus serviços serão a realização de viagens expressas nacionais como internacionais. É, pois, uma empresa de grande dimensão, que possui vários comboios ao seu dispor e efetua trajetos diretos.

1.2. Apresentação do Caso de Estudo

A direção da empresa *PolkaTerra* pretende criar uma plataforma para que os seus clientes possam reservar as viagens, sem se terem de deslocar à estação mais próxima. Assim, chegaram à conclusão que precisavam de armazenar os dados relativos a essas reservas numa base de dados.

A empresa possui um vasto conjunto de comboios que podem efetuar vários trajetos, nacionais ou internacionais, ao longo do dia, várias vezes ao dia. A BD contém vários documentos, sendo o principal o documento que identifica a viagem. Este documento terá todos os dados relativos à mesma, tais como, por exemplo, o local de partido, o destino, e a hora, e ainda, para uma determinada data, o comboio que a irá realizar, e por fim uma lista com todas as reservas efetuadas nessa viagem. Estas reservas ainda contêm alguns dados sobre os clientes que as efetuaram. Como é possível existirem clientes sem reservas e comboios sem viagens, foi necessário criar dois novos tipos de documentos (um para os clientes e outro para os comboios), de forma a guardar toda a informação relativa aos mesmos.

1.3. Motivação e Objectivos

1.3.1 Motivação

O administrador deparou-se com vários problemas relativos à base de dados relacional. Com o passar do tempo e o crescimento da empresa, a BD relacional deixou de ser a opção ideal, sendo necessário arranjar alternativas. Os problemas que surgiram foram:

- Crescimento exponencial dos dados.
- Necessidade de redução de tempo de pesquisas extensivas.
- Dificuldade de alterar o esquema da BD e manter os dados consistentes.
- O acesso à BD é comprometido com a falha na infraestrutura de apoio.

Assim, pretende-se efetuar a migração dos dados da BD relacional para uma BD não relacional sem perder qualquer tipo de informação.

1.3.2 Objectivos

Com base nos problemas apresentados na motivação, foi escolhido uma BD baseada em “Document Stores”, para acolher os dados da BD relacional. Assim, o MongoDB apresenta soluções para os problemas apresentados no ponto anterior. O MongoDB utiliza o conceito de “Big Data”, que se refere a tecnologias que envolvem dados que são demasiado diversos e sujeitas a muitas alterações. O “Big Data” apresenta características muito vantajosas tais como o aumento do volume, variedade e velocidade.

Relativamente ao crescimento dos dados, o MongoDB tem maior capacidade e simplicidade de armazenamento.

Relativamente às pesquisas, o MongoDB permite um tempo de execução inferior, mesmo com um maior volume de dados, ao modelo relacional.

Como o esquema da BD relacional é muito pouco flexível, qualquer alteração na BD implica uma alteração na maior parte dos dados e se a BD não estiver normalizada, implica uma inconsistência dos dados. Assim, o MongoDB, como não possui esquema fixo, é permitida a inserção de documentos de tipos diferentes na mesma BD, tal como dados não relacionados no mesmo documento. Assim, os documentos não necessitam de manter qualquer tipo de estrutura, sendo isto muito vantajoso se for necessário alterar a BD.

O SGBD anteriormente usado poderia ficar inacessível se uma das partes constituintes da estrutura de apoio ficar indisponível. Desta forma, o MongoDB permite

o acesso parcial aos dados que não estão assegurados pela parte indisponível, permitindo assim a manutenção parcial das funções da BD.

1.4. Planeamento

Semana	Planeamento
11/01 – 13/01	Discussão e realização do esquema da base de dados não relacional.
16/01 – 22/01	Realização do “script” de migração, das “queries”. e do relatório.

Tabela 1 - Planeamento

1.5. Estrutura do Relatório

Este relatório encontra-se ordenado pela ordem cronológica de realização das tarefas.

Primeiramente, encontra-se o desenvolvimento do projeto onde se encontra o esquema da BD não relacional (onde é explicado o nosso esquema e decisões que tomamos), o script de migração de dados, e as “queries” que decidimos implementar (tal como o resultado delas).

Por fim, temos as conclusões, onde fazemos a avaliação do trabalho e da abordagem não relacional que utilizamos.

2. Desenvolvimento do Projeto

2.1. Desenvolvimento do Esquema para a Base de Dados

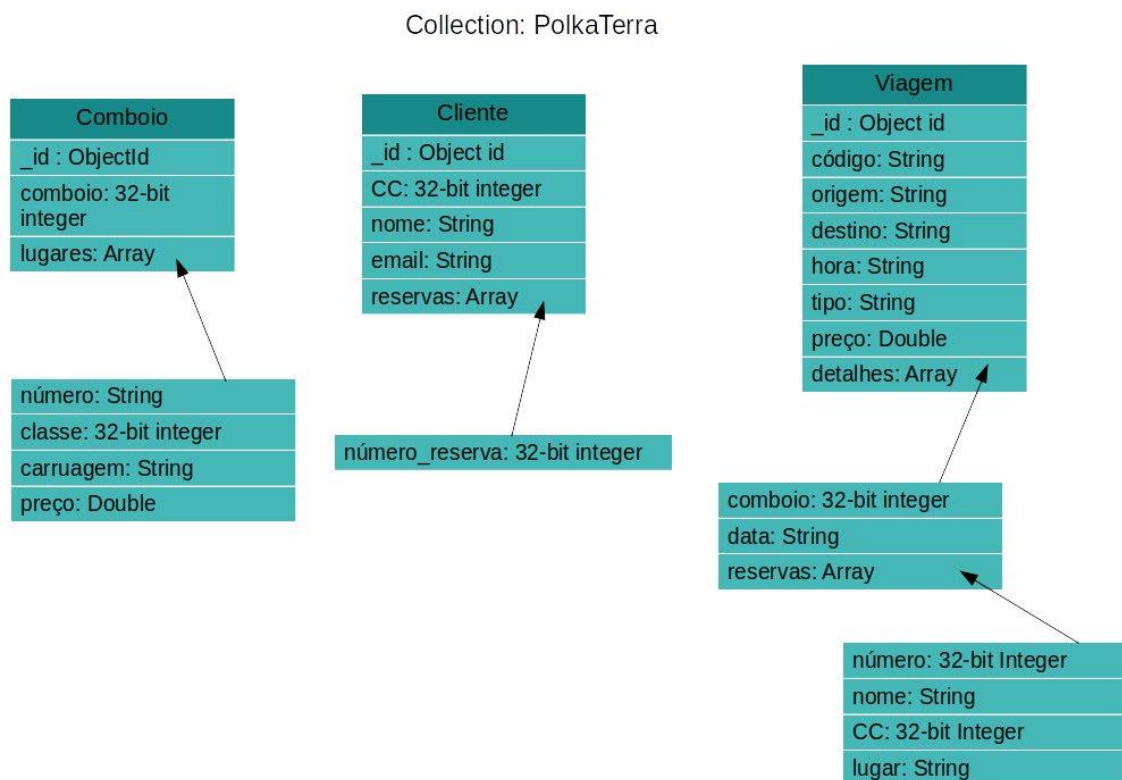


Figura 1 - Esquema da Base de Dados Não Relacional

Como primeira abordagem ao paradigma não relacional, o grupo realizou uma pesquisa sobre as possíveis estruturas que o MongoDB pode abranger dentro de uma base de dados. Conclui-se que uma coleção aceita documentos com estruturas distintas, o que permite uma grande flexibilidade no armazenamento dos dados.

Como se tratam de paradigmas diferentes, relacional e não relacional, não se pode assumir que a tradução entre eles seja literal, isto é, a migração dos dados não é completamente linear, uma vez que uma tabela do relacional não equivale necessariamente a

um documento no não relacional. Assim, foi necessário proceder-se a uma análise de como seria feito o armazenamento dos dados da BD relacional criada na primeira fase do projeto prático. Distinguiu-se, portanto, três tipos de documentos diferentes: documentos para os clientes, documentos para os comboios e documentos para as viagens. Estes documentos farão parte da mesma coleção, polkaterra, uma vez que não se viu a necessidade de usar coleções diferentes, tirando partido da característica polimórfica que os documentos podem tomar. Desta maneira, foram definidos estes três tipos de documentos, pois, tanto os clientes, como os comboios e as viagens, são independentes entre si, isto é, existem por si só (ao contrário das reservas, que estão associadas a uma viagem e a um cliente).

2.1.1 Documento para Cliente

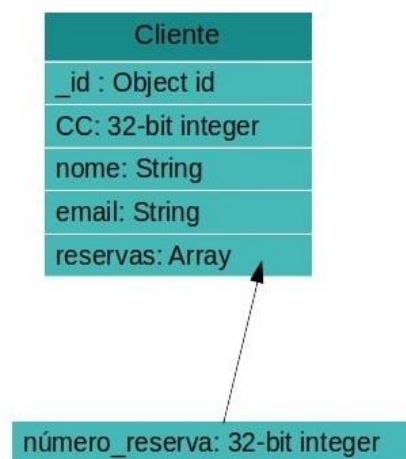


Figura 2 - Documento Cliente

Relativamente ao cliente, são necessários guardar os seguintes dados:

- Número de CC (CC): identificador principal do cliente.
- Nome (nome): indica o nome e apelido do cliente.
- E-mail (email): indica o e-mail do cliente.
- “Array” de reservas (reservas): contém todos os identificadores das reservas efetuadas pelo cliente.

A necessidade de existir um documento que identifica só os clientes, surge do facto de poderem existir clientes que não possuem reservas, logo, se não existissem documentos para os clientes, na base de dados apenas estariam armazenados clientes com reservas efetuadas.

2.1.2 Documento para Comboio

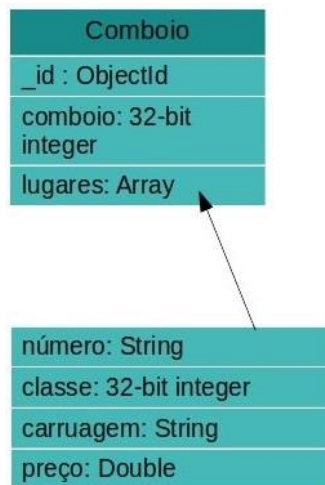


Figura 3 - Documento Comboio

Relativamente ao comboio, será necessário armazenar os seguintes dados:

- Id do Comboio (comboio): indica o identificador do comboio.
- “Array” de lugares (lugares): indica todos os lugares do comboio e a sua informação.

O “array” lugares contém os seguintes dados:

- Número (número): indica o número do lugar.
- Classe (classe): indica a classe do lugar.
- Carruagem (carruagem): indica a carruagem do lugar.
- Preço (preço): indica o preço do lugar.

Os documentos que contêm os comboios foram criados devido à necessidade de armazenar todos os comboios, tanto os que realizam viagens, como os que não realizam nenhuma viagem.

2.1.3 Documento para Viagem

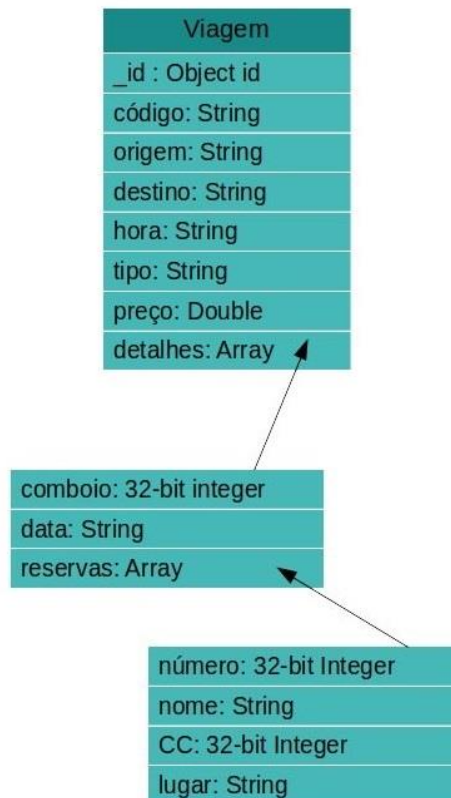


Figura 4 - Documento Viagem

O documento Viagem contém os seguintes dados:

- Código (código): código que identifica a viagem.
- Origem (origem): da viagem.
- Destino (destino): da viagem.
- Hora de partida (hora): da viagem.
- Tipo (tipo): indica se a viagem é nacional ou internacional.
- Preço (preço): da viagem.
- “Array” (detalhes).

O “array” detalhes contém a seguinte informação:

- Identificador do comboio (comboio).
- Data de partida da viagem (data).
- “Array” de reservas (reservas).

O “array” reservas possui os seguintes dados:

- Identificador da reserva (número).
- Nome e apelido do cliente (nome).
- Número de CC (CC).
- Número de lugar (lugar).

Para armazenar os dados de uma viagem e as suas reservas, decidiu-se recorrer ao conceito de documentos aninhados (“embedded documents”), de forma a juntar toda a informação relacionada sobre uma determinada viagem num só documento.

Deste modo, à semelhança do modelo relacional, uma viagem é vista como se estivesse num “placard” de uma estação de comboios, isto é, como característica universal é apresentada, entre outras, a sua hora de partida. Contudo, esta viagem, com o mesmo horário, pode ser feita em diversos dias. É por esta razão que o campo hora não se encontra no subdocumento aninhado juntamente com a data de partida, pois entende-se que a hora da viagem está num nível hierárquico acima da data. Assim, a hora define um itinerário com uma origem e um destino, enquanto que a data especifica a viagem realizada nesse dia.

Além disso, é possível uma viagem não possuir nenhuma reserva. Neste caso, o array reservas será vazio. No subdocumento das reservas, adicionou-se o campo do nome do cliente, que é redundante, para facilitar a definição das “queries” sobre a base de dados. Como no MongoDB, a redundância de dados é encorajada em certos casos semelhantes a este, o grupo decidiu que seria uma decisão correta.

2.2. Migração dos Dados

Para fazer a migração dos dados da BD relacional para a BD não relacional orientada a documentos foram-nos dadas duas opções: usar um “script” numa linguagem de programação à nossa escolha ou usar uma ferramenta. Neste seguimento, o grupo decidiu usar um “script” de migração na linguagem JAVA da sua autoria. Este “script” respeita o esquema acima mencionado que o grupo, no planeamento da base de dados não relacional, definiu.

2.2.1 Script de Migração

```
public class Script
{
    private          static          final          String          DB_URL          =
    "jdbc:mysql://localhost/PolkaTerra";
    private static final String USER = "root";
    private static final String PASS = "";
    private static Connection conn = null;
    private static PreparedStatement stmt = null;

    /*
    Cria um BasicDBObject através dos parâmetros dados.
```

```

    */
    public static BasicDBObject toObjectLugar(String n, int cl, String
cr, float p)
    {
        BasicDBObject obj = new BasicDBObject();

        obj.append("número", n)
            .append("classe", cl)
            .append("carruagem", cr)
            .append("preço", p);

        return obj;
    }

```

```

    /*
    Retorna o nome do cliente com o número de cartão de cidadão igual
ao do parâmetro
    */
    public static String getNome(int n)
    {
        String nome = null;
        String apelido = null;
        try {
            String sql = "select nome, apelido from Cliente where
num_cc = " + n;
            Statement s = conn.createStatement();
            ResultSet res = s.executeQuery(sql);
            if(res.next())
            {
                nome = res.getString("nome");
                apelido = res.getString("apelido");
            }
        } catch (SQLException ex) {

            Logger.getLogger(Script.class.getName()).log(Level.SEVERE, null, ex);
        }
    }

```

```

        return nome + " " + apelido;
    }

    public static void main(String[] args)
    {
        try{
            Class.forName("com.mysql.jdbc.Driver");
            conn = DriverManager.getConnection(DB_URL, USER, PASS);
            Mongo mongo = new Mongo("localhost", 27017);
            DB db = mongo.getDB("polkaterra");
            DBCollection coll = db.getCollection("polkaterra");

            ResultSet rs, result;
            BasicDBObject doc;
            PreparedStatement stm, st;

            stmt = conn.prepareStatement("SELECT * FROM Comboio");
            rs = stmt.executeQuery();
            while(rs.next())
            {
                doc = new BasicDBObject();
                int id_comboio = rs.getInt("id_comboio");
                doc.append("comboio", id_comboio);
                stm = conn.prepareStatement("SELECT * from Lugar where
id_comboio =?");
                stm.setInt(1, id_comboio);
                result = stm.executeQuery();
                List<BasicDBObject> lugares = new ArrayList<>();
                while(result.next())
                {
                    String numero = result.getString("número");
                    int classe = result.getInt("classe");
                    String carruagem = result.getString("carruagem");
                    float preco = result.getFloat("preço");
                    lugares.add(toObjectLugar(numero, classe, carruagem,
preco));
                }
                doc.append("lugares", lugares);
                coll.insert(doc);
            }
            stmt = conn.prepareStatement("SELECT * from Cliente");

```



```

rs = stmt.executeQuery();

while(rs.next())
{
    doc = new BasicDBObject();
    int numcc = rs.getInt("num_cc");
    String email = rs.getString("email");
    String nome = rs.getString("nome");
    String apelido = rs.getString("apelido");
    doc.append("CC", numcc)
        .append("nome", nome + " " + apelido)
        .append("email", email);

    stm = conn.prepareStatement("SELECT id_reserva from
Reserva where id_cliente =?");
    stm.setInt(1, numcc);
    result = stm.executeQuery();
    List<BasicDBObject> reservas = new ArrayList<>();
    while(result.next())
    {
        reservas.add(new BasicDBObject("número_reserva",
result.getInt("id_reserva")));
    }
    doc.append("reservas", reservas);
    coll.insert(doc);
}

stmt = conn.prepareStatement("SELECT * from Viagem");
rs = stmt.executeQuery();

while(rs.next())
{
    doc = new BasicDBObject();
    String viagem = rs.getString("id_viagem");
    float preço = rs.getFloat("preço");
    String origem = rs.getString("origem");
    String destino = rs.getString("destino");
    String hora = rs.getTime("hora").toString();
    String tipo = rs.getString("tipo");
    doc.append("código", viagem)
        .append("origem", origem)
        .append("destino", destino)

```

```

        .append("hora", hora)
        .append("tipo", tipo)
        .append("preço", Math.round(preço*100d)/100d);
List<BasicDBObject> datas = new ArrayList<>();
stm = conn.prepareStatement("select * from ViagemComboio
where id_viagem = ?");
stm.setString(1, viagem);
result = stm.executeQuery();
while(result.next())
{
    int comboio = result.getInt("id_comboio");
    String data = result.getDate("data").toString();
    st = conn.prepareStatement("select * from Reserva
where id_viagem = ? and data = ?");
    st.setString(1, viagem);
    st.setString(2, data);
    ResultSet set = st.executeQuery();
    List<BasicDBObject> reservas = new ArrayList<>();
    while(set.next())
    {
        int reserva = set.getInt("id_reserva");
        String lugar = set.getString("lugar");
        int numcc = set.getInt("id_cliente");
        String nome = getNome(numcc);
        BasicDBObject obj = new BasicDBObject();
        obj.append("número", reserva)
            .append("nome", nome)
            .append("CC", numcc)
            .append("lugar", lugar);
        reservas.add(obj);
    }
    BasicDBObject o = new BasicDBObject();
    o.append("comboio", comboio)
      .append("data", data)
      .append("reservas", reservas);
    datas.add(o);
}
doc.append("detalhes", datas);
coll.insert(doc);
}

```

```

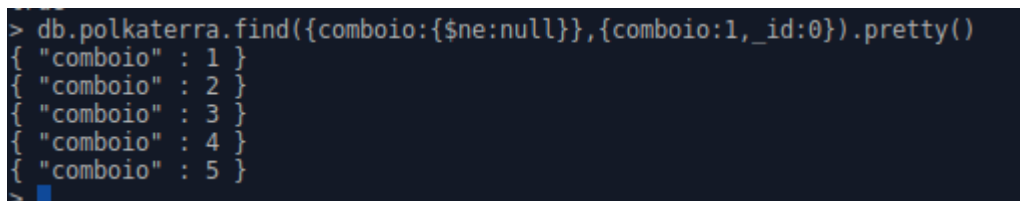
    }
    catch(SQLException | ClassNotFoundException se)
    {
        System.out.println(se.getMessage());
    }
    finally
    {
        try
        {
            if(stmt!=null)
                conn.close();
        }
        catch(SQLException se){}
    }
}
}

```

De modo a migrar os dados da BD relacional para o novo sistema não relacional, foi preciso ler os dados da base de dados SQL e agrupá-los de acordo com o esquema definido previamente. De seguida, foram adicionados, sob a forma de um documento, para a coleção criada na base de dados NoSQL. Todo este processo pode ser verificado no código JAVA apresentado acima.

2.3. Conjunto de Queries

2.3.1 Query 1



```

> db.polkaterra.find({comboio:{$ne:null}}, {comboio:1, _id:0}).pretty()
{ "comboio" : 1 }
{ "comboio" : 2 }
{ "comboio" : 3 }
{ "comboio" : 4 }
{ "comboio" : 5 }
>

```

Figura 5 - Query 1

Esta “query” mostra todos os números dos comboios existentes na base de dados.

2.3.2 Query 2

```
> db.polkaterra.find({email:{$ne:null}}, {email:true, _id:0})
{ "email" : "ruicosta@live.com.pt" }
{ "email" : "martadfernandes@live.com.pt" }
{ "email" : "princess@gmail.com" }
{ "email" : "jonas@sapo.pt" }
{ "email" : "calibri@gmail.com" }
{ "email" : "maestrovitorino@live.com.pt" }
{ "email" : "anastbarros@hotmail.com" }
{ "email" : "special@one.co.uk" }
{ "email" : "carlota_12@sapo.pt" }
{ "email" : "di_ana@sapo.pt" }
{ "email" : "disilva@live.com.pt" }
{ "email" : "mari-500@hotmail.com" }
{ "email" : "anabela@sapo.pt" }
{ "email" : "andre12021996@gmail.com" }
{ "email" : "tochaves@live.com.pt" }
{ "email" : "ruteruterute97@gmail.com" }
{ "email" : "armandoMarta@live.com.pt" }
{ "email" : "luisboamorte@sapo.pt" }
{ "email" : "carlosSilva@hotmail.com" }
{ "email" : "manelchaves@gmail.com" }
Type "it" for more
> 
```

Figura 6 - Query 2

Esta “query” mostra todos os “e-mails” dos clientes que estão armazenados na base de dados.

2.3.3 Query 3

```
> db.polkaterra.find({código:"POPA1155"},{"detalhes.reservas.nome":1,_id:0}).pretty()
{
  "detalhes" : [
    {
      "reservas" : [
        {
          "nome" : "João Amorim"
        },
        {
          "nome" : "Manuel Chaves"
        },
        {
          "nome" : "Bruno Pereira"
        },
        {
          "nome" : "Victorino de Almeida"
        },
        {
          "nome" : "Carlos da Maia"
        },
        {
          "nome" : "Maria Brito"
        },
        {
          "nome" : "Diogo Silva"
        },
        {
          "nome" : "João Gonçalves"
        },
        {
          "nome" : "Carlos Silva"
        },
        {
          "nome" : "Armando Marta"
        },
        {
          "nome" : "Carla Trafaria"
        },
        {
          "nome" : "Rute Castro"
        },
        {
          "nome" : "Marta Fernandes"
        }
      ]
    }
  ]
}
```

Figura 7 - Query 3

Esta “query” mostra os nomes dos clientes que fizeram reservas numa dada viagem (neste caso, é a viagem com código POPA1155).

2.3.4 Query 4

```
> db.polkaterra.update({CC:15017191},{ $set:{email:"new_email_joe@sapo.pt"}})
WriteResult({ "nMatched" : 1, "nUpserted" : 0, "nModified" : 1 })
> db.polkaterra.find({CC:15017191},{}).pretty()
{
  "_id" : ObjectId("587e47e688dbd21029c17d36"),
  "CC" : 15017191,
  "nome" : "João Amorim",
  "email" : "new_email_joe@sapo.pt",
  "reservas" : [
    {
      "número_reserva" : 1
    },
    {
      "número_reserva" : 73
    },
    {
      "número_reserva" : 74
    }
  ]
}
```

Figura 8 - Query 4

Esta “query” atualiza o campo “email” de um determinado cliente na base de dados (neste caso, é o cliente cujo número de cartão de cidadão é 15017191). Consegue-se verificar a mudança efetuada quando se realiza uma nova consulta nesse cliente.

2.3.5 Query 5

```
> db.polkaterra.find({origem:"Braga",destino:"Lisboa"},{_id:0,"detalhes.reservas":0}).pretty()
{
  "código" : "BRLB0630",
  "origem" : "Braga",
  "destino" : "Lisboa",
  "hora" : "06:30:00",
  "tipo" : "N",
  "preço" : 49.5,
  "detalhes" : [
    {
      "comboio" : 4,
      "data" : "2017-12-02"
    }
  ]
}
{
  "código" : "BRLB1112",
  "origem" : "Braga",
  "destino" : "Lisboa",
  "hora" : "11:12:00",
  "tipo" : "N",
  "preço" : 49.5,
  "detalhes" : [
    {
      "comboio" : 5,
      "data" : "2017-03-03"
    }
  ]
}
> □
```

Figura 9 - Query 5

Esta “query” mostra os vários campos das viagens, à exceção das suas reservas, que contêm a origem e o destino especificados. Desta maneira, vai mostrar os seus códigos, origem, destino, hora, tipo e preço, bem como a data de partida e o número do comboio que efetuou/efetuará essa viagem.

2.3.6 Query 6

```
> db.polkaterra.find({reservas:[]},{CC:true,nome:1,email:true,_id:0})
{ "CC" : 12354687, "nome" : "Pedro Pinto", "email" : "calibri@gmail.com" }
{ "CC" : 13654789, "nome" : "Jose Mourinho", "email" : "special@one.co.uk" }
{ "CC" : 15665974, "nome" : "Vale Santos", "email" : "panda@panda.pt" }
> □
```

Figura 10 - Query 6

Esta “query” mostra os detalhes dos clientes (número de cartão de cidadão, nome e “e-mail”) que não efetuaram nenhuma reserva de viagens.

2.3.7 Query 7

```
> db.polkaterra.find({comboio:1},{lugares:true,_id:0}).pretty()
{
  "lugares" : [
    {
      "número" : "1A",
      "classe" : 1,
      "carruagem" : "A",
      "preço" : 9
    },
    {
      "número" : "1B",
      "classe" : 2,
      "carruagem" : "B",
      "preço" : 5
    },
    {
      "número" : "1C",
      "classe" : 3,
      "carruagem" : "C",
      "preço" : 0
    },
    {
      "número" : "2A",
      "classe" : 1,
      "carruagem" : "A",
      "preço" : 9
    },
    {
      "número" : "2B",
      "classe" : 2,
      "carruagem" : "B",
      "preço" : 5
    },
    {
      "número" : "2C",
      "classe" : 3,
      "carruagem" : "C",
      "preço" : 0
    },
    {
      "número" : "3A",
      "classe" : 1,
      "carruagem" : "A",
      "preço" : 9
    }
  ],
}
```

Figura 11 - Query 7

Esta “query” mostra todos os lugares (número, classe, carruagem em que se encontra e preço definido) de um determinado comboio (neste caso, do comboio com número de identificação 1).

2.3.8 Query 8

```
> db.polkaterra.find({preço:{$lt:100}},{_id:0}).pretty()
{
  "código" : "BRC00930",
  "origem" : "Braga",
  "destino" : "Coimbra",
  "hora" : "09:30:00",
  "tipo" : "N",
  "preço" : 39.5,
  "detalhes" : [
    {
      "comboio" : 1,
      "data" : "2017-10-09",
      "reservas" : [
        {
          "número" : 72,
          "nome" : "Anabela Santos",
          "CC" : 14312090,
          "lugar" : "3A"
        }
      ]
    }
  ]
}
{
  "código" : "BRC01135",
  "origem" : "Braga",
  "destino" : "Coimbra",
  "hora" : "11:35:00",
  "tipo" : "N",
  "preço" : 39.5,
  "detalhes" : [
    {
      "comboio" : 2,
      "data" : "2017-12-09",
      "reservas" : [ ]
    }
  ]
}
{
  "código" : "BRLB0630",
  "origem" : "Braga",
  "destino" : "Lisboa",
  "hora" : "06:30:00",
  "tipo" : "N",
  "preço" : 49.5,
```

Figura 12 - Query 8

Esta “query” mostra todos os detalhes das viagens (código, origem, destino, hora de partida, tipo de viagem, preço, número de comboio a si designada, data de partida e especificações das reservas efetuadas) cujos preços são inferiores a um certo valor (neste caso, todas as viagens com um preço inferior a 100).

3. Conclusões

Na segunda fase do projeto prático, fizemos a mudança de uma BD relacional para uma BD não relacional, tendo em conta todas as alterações necessárias para a realização da migração dos dados e a operacionalidade das “queries” definidas.

Deste modo, esta migração de dados foi muito útil, uma vez que o sistema de gestão de bases de dados não relacional usado foi o MongoDB, que prima pela grande adaptabilidade e escalabilidade com um grande volume de dados, bem como pela variedade de estruturas de dados (documentos) que podem existir dentro de uma dada coleção de uma base de dados. Assim, não foi necessário criar estruturas previamente definidas para efetuar o armazenamento dos dados.

Uma das grandes diferenças do MongoDB em relação ao sistema de gestão de bases de dados usado na primeira fase do projeto é que, neste caso, não é feita a verificação e garantia da integridade dos dados inseridos na base de dados, isto é, podemos inserir num cliente uma reserva que, de facto, não existe e, como não há uma verificação de integridade, a inserção é feita com sucesso. Outra diferença crucial é a inexistência de “joins” que, no modelo relacional, existem em abundância. Devido a este facto, foi necessário proceder a um “merge” de vários tipos de dados que estão relacionados uns com os outros através de documentos aninhados, de modo a facilitar a operacionalidade das “queries” definidas. Portanto, em comparação ao modelo relacional, a variedade de “queries” é mais reduzida, uma vez que as possibilidades de operar sobre diferentes tipos de dados independentes são menores. Logo, estes dois aspetos têm de ser tratados a nível aplicacional e não a nível da base de dados, o que vai além dos objetivos deste projeto prático.

Como também não existe o conceito de transações no modelo não relacional, então qualquer inserção ou atualização que, devido à redundância que foi necessária introduzir em NoSQL, é preciso ser tomada como uma operação atómica tem, igualmente, de ser tratada com cuidado noutra nível, que não o da base de dados.

A maior dificuldade que o grupo enfrentou no desenvolvimento desta fase do projeto prático foi a definição estrutural dos documentos que iriam fazer parte da coleção polkaterra. Além disso, teve que ser decidido se existiriam várias coleções ou apenas uma. O grupo definiu, então, que apenas existiria uma única coleção na base de dados, pois tanto os clientes, os participantes nas viagens, e os comboios, os

agentes que realizam as viagens, estão relacionados com as viagens, estando tudo na mesma lógica de negócio.

Em suma, chegou-se à conclusão que este sistema de reserva de viagens seria mais adequado ao sistema de base de dados relacional, visto que este sistema lida com as transações complexas e questões de integridade que são fundamentais para este tipo de sistema de reservas.

Referências

MongoDB | FOR GIANT IDEAS, 2017. MongoDB and MySQL Compared. [online] Available at: < <https://www.mongodb.com/compare/mongodb-mysql> > [Accessed 18 January 2017].

DZone / Big Data Zone, 2017. Better explaining the CAP Theorem. [online] Available at: <<https://dzone.com/articles/better-explaining-cap-theorem>> [Accessed 16 January 2017].

MongoDB | FOR GIANT IDEAS, 2017. Big Data Explained. [online] Available at: < <https://www.mongodb.com/big-data-explained> > [Accessed 18 January 2017].

MongoDB | FOR GIANT IDEAS, 2017. 6 Rules of Thumb for MongoDB Schema Design: Part 1. [online] Available at: <<https://www.mongodb.com/blog/post/6-rules-of-thumb-for-mongodb-schema-design-part-1>> [Accessed 17 January 2017]

Lista de Siglas e Acrónimos

BD Base de Dados

SGBD Sistema de Gestão de Bases de Dados

