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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Comboios Flavienses

Pedro Bragança(A70886), Daniel Vieira(a73974),

Rafael Silva(A74264)

Janeiro de 2017

Data de Recepção	
Responsável	
Avaliação	
Observações	

D

Comboios Flavienses

Pedro Bragança(A70886), Daniel Vieira(a73974),

Rafael Silva(A74264)

Janeiro de 2017

<</opcional Dedicatória>>

Resumo

Este projeto insere-se na unidade curricular de Base de Dados, e consiste em levar a prática o planeamento e execução de projetos de Sistemas de Base de Dados não Relacionais.
No trabalho1 foi nos proposto um trabalho em que tivéssemos que fazer uma gestão de reservas de bilhete de comboios nacionais quer internacionais.
Neste 2ºtrabalho foi pretendido que cada grupo realizasse um trabalho de análise, planeamento, e implementação de um SBD não relacional, tendo como base o tema de trabalho anterior.
Para que este relatório seja completo vamos detalhar cada passo que façamos para a apresentação e resolução do problema citado na frase anterior.
No final iremos apresentar uma analise crítica, objetiva, sucinta do nosso trabalho, evidenciando aquilo que tivemos mais ou menos dificuldades, isto é, os obstáculos que percorremos no desenrolar deste trabalho, bem como anexos e tabelas para demonstrar de uma forma mais clara o processo de elaboração deste problema que nos foi dado.

Área de Aplicação: Planeamento e execução de Sistemas de Bases de Dados não Relacionais.
Palavras-Chave: Bases de Dados não Relacionais, Java, MongoDB, Documentos.

Índice

Índice de Figuras

1. Introdução

1.1. Contextualização

Este projeto que está inserido na unidade curricular de Base de Dados, unidade curricular esta que insere-se no Mestrado Integrado de Engenharia Informática teve, ao contrário dos anos anteriores, uma tema que nos foi dado pelo professor. Consiste na gestão de reservas de bilhetes de comboios nacionais e internacionais.

1.2. Apresentação do Caso de Estudo

A tecnologia está ao nosso dispor como um bem para a sociedade, certo?
Logo para quê usar arquivos em papel, com grandes gastos em termos de horários de trabalhos bem como uma utilização de espaços reais tanto maior quanto aquilo que armazenamos, quando podemos usar tudo informaticamente, isto é, à base de uma plataforma eletrónica que podemos aceder a qualquer hora?
É aqui que Base de Dados entra, isto é, é uma forma de poupar dinheiro quer nas horas de trabalho de cada funcionário que se iriam precisar para arquivar ficheiros, bem como o espaço que lhe e concedido para guardar esses ficheiros arquivados. É tudo feito através de uma plataforma eletrónica, como já dissemos anteriormente, logo haverá evidentemente menos custos associados a horas de trabalho bem como ao espaço real.
Mas a principal vantagem é a analise de dados e para isso apresentamos queries, isto é, “programas” que façam perguntas em relação aos dados recolhidos para que depois possamos fazer observações, estudos sobre eles mesmos.
Logo para isso, representaremos um sistema de gestão de reservas de bilhetes de comboios de viagens nacionais e internacionais, como nos foi pedido, organizados consoante o levantamento de requisitos.

1.3 Motivação e Objectivos

Nas grandes aplicações web é cada vez mais comum a quantidade de informações ser cada vez maior, e o pior é que não acaba,vito que irá ser cada vez mais.

Haverá sempre mais dados para armazenar.

Como lidamos com isso de maneira eficiente?

A grande motivação para NoSQL foi precisamente essa, uma maior eficencia com uma grande quantidade de dados.

Neste caso usando o MongoDB que é uma aplicação de código orientado a Documentos, usando conjunto de documentos em JSON, iremos “transformar” a base de dados do 1ºtrabalho em MongoDB.

Usando o MongoDB, iremos modelar informação de modo mais natural, visto que estes dados irão alinhados em hierarquias complexas o que leva a que essas mesmas informações sejam fáceis de ser buscadas.

1.4 Estrutura do Relatório

Com este relatório iremos incidir evidentemente nas matérias dadas na aulas teóricas e práticas desde a apresentação de scripts com uma breve explicação de como migramos os dados relacionados ao Sistema relacional para Sistema nao relacional, alem de umas queries.
No final iremos apresentar uma analise crítica do nosso trabalho fazendo uma breve comparacão entre o modelo do 1ºtrabalho e o modelo deste.

1.5 Levantamento e analise de requisitos

-Na reserva há varias viagens no qual cliente tem que escolher a que lhe mais convém. Em cada reserva só se reserve 1 viagem,logo se quiser 2 viagens tem que fazer uma reserve de novo.

- Na viagem há varias informações importantes como por exemplo: Local da partida, Local de chegada, Hora de partida, Hora de chegada bem como o preço(toda esta informações consegue-se saber no momento em que se faz a reserva)

- Para fazer a reserva é preciso ser cliente e para ser cliente é preciso dar os seus dados pessoas como por exemplo, nome, NIF, género, telemóvel, email, senha.
- NIF, email, telemóvel são únicos, logo vai haver sempre diferentes.
- Poderá haver evidentemente clientes com o mesmo nome.
- Ao fazer a reserva, irá lhe ser dado um lugar que varia consoante o comboio, isto é, se for da 1ºClasse é um número de 1 até 9 e caso seja Económica vem do 10 ate ao 25. É impossível fazer a reserva sem escolher a classe que quer.
- Cada comboio terá no máximo 25 lugares.
- Há 2 classes em cada comboio, logo haverá 2 preço sendo que o mais barato é da classe Económica e o mais caro é a 1ºClasse.
- Cada estação pode ter como partido ou destino varias viagens.
- E impossível existir uma viagem sem partida, destino bem como a hora de partida e hora de data.
- Os comboios pelo menos fazem 1 viagem, isto é, o comboio 1 pode ir de Lisboa-Porto e depois Chaves-Manchester.

1.6 Justificação do Trabalho

Visto que este trabalho se trata de um planeamento e execução de projetos de Sistemas de Base de Dados não Relacionais,e como neste sistema não usamos modelo conceptual, não achamos bem apresentar neste relatório porque achamos que não faz parte do objetivo do trabalho(apesar de ser feito em relação ao 1ºtrabalho que tem modelo conceptual).

Vamos nos restringir unicamente ao trabalho que nos foi proposto,logo tudo o que foi feito para haver uma migração dos dados contidos no sistema relacional implementado anteriormente para o novo sistema não relacional, alem das querys, estará apresentado neste relatório.

1.7 Script De Migração Dos Dados

```
public List<Cliente> getCliente() throws ClassNotFoundException{
    ResultSet cli=null;
    PreparedStatement stCL=null;
    ArrayList<Cliente> clientes = new ArrayList<>();
    try {
        stCL= con.prepareStatement("SELECT * FROM Comboios_Flavienses.cliente;");
        cli=stCL.executeQuery();
        while(cli.next()){
            String Nome = cli.getString ("nome");
            int nif = cli.getInt ("nif");
            String genero = cli.getString("genero");
            int telemovel = cli.getInt("telemovel");
            String emai = cli.getString("email");
            String senha = cli.getString("senha");
            clientes.add( new Cliente (Nome,nif,genero,telemovel,emai,senha));
        }
    }
     catch (SQLException ex) {
        Logger lgr = Logger.getLogger(Version.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
     }
     finally {
        try {
            if (cli != null) {
              cli.close();
            }
            if (stCL != null) {
                stCL.close();
            }
        }
        catch (SQLException ex) {
            Logger lgr = Logger.getLogger(Version.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }
    return clientes;
}
```

```java
public List<Comboio> getComboio () throws ClassNotFoundException{
    ResultSet comboio=null;
    ResultSet lc=null;
    PreparedStatement stC=null;
    PreparedStatement stLC=null;
    List<Comboio> com = new ArrayList<>();
    try {
        stC= con.prepareStatement("SELECT * FROM Comboios_Flavienses.Comboio;");
        comboio=stC.executeQuery();
        while(comboio.next()){
            int id = comboio.getInt("id_Comboio");
            String tipo = comboio.getString ("Tipo_Comboio");
            int Lotacao_Primeira_Classe = comboio.getInt ("Lotaçao_Primeira_Classe");
            int Lotacao_Classe_Economica = comboio.getInt ("Lotaçao_Economica");
            int Capacidade = comboio.getInt ("Capacidade");
            stLC=con.prepareStatement("SELECT * FROM comboios_flavienses.Lugar where Comboio_id_Comboio= "+id);
            List<Lugar> lugares = new ArrayList<>();
            lc=stLC.executeQuery();
            while (lc.next()){
                int N = lc.getInt ("N_Lugar");
                String Classe = lc.getString ("Classe");
                lugares.add(new Lugar (N,Classe));
            }
            com.add (new Comboio (id, tipo, Lotacao_Primeira_Classe, Lotacao_Classe_Economica, Capacidade, lugares));
        }

    }
```

```
        catch (SQLException ex) {
          Logger lgr = Logger.getLogger(Version.class.getName());
          lgr.log(Level.SEVERE, ex.getMessage(), ex);
        }
        finally {
          try {
              if (lc != null) {
                  lc.close();
              }
              if (comboio != null) {
                comboio.close();
              }
              if (stLC != null) {
                  stLC.close();
              }
              if (stC != null) {
                  stC.close();
              }
          }
          catch (SQLException ex) {
              Logger lgr = Logger.getLogger(Version.class.getName());
              lgr.log(Level.WARNING, ex.getMessage(), ex);
          }
      }
      return com;
}
```

```
public List<Reserva> getReserva () throws ClassNotFoundException{
    ResultSet reserva=null;
    PreparedStatement stR=null;
    List<Cliente> cliente = new ArrayList<>();
    List<Reserva> bt = new ArrayList<>();
    List<Viagem> viagem = new ArrayList<>();
    try {
        stR= con.prepareStatement("SELECT * FROM comboios_flavienses.Reserva");
        reserva=stR.executeQuery();
        while (reserva.next()){
            int idR = reserva.getInt ("id_Reserva");
            int idC = reserva.getInt ("Cliente_id_Cliente");
            float Custo = reserva.getInt ("Custo");
            int lugar = reserva.getInt ("Lugar");
            String Classe = reserva.getString ("Classe");
            String DR = reserva.getString("Data_Reserva");
            cliente = getCliente();
            viagem = getViagem();
            bt.add (new Reserva (idR, Custo, lugar, Classe, DR, cliente, viagem));
        }
      }
      catch (SQLException ex) {
        Logger lgr = Logger.getLogger(Version.class.getName());
        lgr.log(Level.SEVERE, ex.getMessage(), ex);
      }
      finally {
        try {
            if (reserva != null) {
                reserva.close();
            }
            if (stR != null) {
```

```
        catch (SQLException ex) {
            Logger lgr = Logger.getLogger(Version.class.getName());
            lgr.log(Level.WARNING, ex.getMessage(), ex);
        }
    }
    return bt;
}
```

```
    public List<Viagem> getViagem () throws ClassNotFoundException{
        ResultSet V =null;
        PreparedStatement stV=null;
        List<Viagem> viagem = new ArrayList<>();
        List<Comboio> comboio = new ArrayList<>();
        try {
            stV= con.prepareStatement("SELECT * FROM comboios_flavienses.Viagem");
            V=stV.executeQuery();
            while(V.next()){
                int idV = V.getInt ("id_Viagem");
                int idC = V.getInt ("Comboio_id_Comboio");
                float Preco_Primeira_Classe = V.getFloat ("Preço_Primeira_classe");
                float Preco_Economica = V.getFloat ("Preço_Economica");
                int Ocupacao_Primeira = V.getInt ("Ocupaçao_Primeira_Classe");
                int Ocupacao_Economica = V.getInt ("Ocupaçao_Classe_Economica");
                String HP = V.getString("Hora_Partida");
                String HD = V.getString("Hora_Destino");
                String Partida = V.getString("Partida");
                String Destino = V.getString("Destino");
                comboio = getComboio();
                viagem.add (new Viagem (Preco_Primeira_Classe, Preco_Economica, Ocupacao_Primeira,
                                        Ocupacao_Economica, HP, HD, Partida, Destino, comboio));
            }
          }
          catch (Exception ex) {
            Logger lgr = Logger.getLogger(Version.class.getName());
            lgr.log(Level.SEVERE, ex.getMessage(), ex);
          }
```

```
          finally {
            try {
                if (V != null) {
                    V.close();
                }
                if (stV != null) {
                    stV.close();
                }
            }
            catch (SQLException ex) {
                Logger lgr = Logger.getLogger(Version.class.getName());
                lgr.log(Level.WARNING, ex.getMessage(), ex);
            }
        }
        return viagem;
    }
}
```

```java
public class SQL {

    public static void main(String[] args) throws Exception{
      SQLreader sql = new SQLreader();

      List<Cliente> vCliente = new ArrayList<>();
      List<Comboio> vComboio = new ArrayList<>();
      List<Reserva> vReserva = new ArrayList<>();
      List<Viagem> vViagem = new ArrayList<>();

      FileWriter clienteF;
      FileWriter comboioF;
      FileWriter reservaF;
      FileWriter viagemF;

      ObjectMapper cli = new ObjectMapper();
      cli.enable(SerializationConfig.Feature.INDENT_OUTPUT);
      cli.writerWithDefaultPrettyPrinter();

      ObjectMapper com = new ObjectMapper();
      com.enable(SerializationConfig.Feature.INDENT_OUTPUT);
      com.writerWithDefaultPrettyPrinter();

      ObjectMapper rs = new ObjectMapper();
      rs.enable(SerializationConfig.Feature.INDENT_OUTPUT);
      rs.writerWithDefaultPrettyPrinter();

      ObjectMapper vi = new ObjectMapper();
      vi.enable(SerializationConfig.Feature.INDENT_OUTPUT);
      vi.writerWithDefaultPrettyPrinter();
```

```
sql.iniciarC();

vCliente = sql.getCliente();
vComboio = sql.getComboio();
vReserva = sql.getReserva();
vViagem = sql.getViagem();

clienteF = new FileWriter("cliente.json");
cli.writeValue(clienteF,vCliente);
clienteF.close();

comboioF = new FileWriter("comboio.json");
com.writeValue(comboioF,vComboio);
comboioF.close();

reservaF = new FileWriter("reserva.json");
rs.writeValue(reservaF,vReserva);
reservaF.close();

viagemF = new FileWriter("viagem.json");
vi.writeValue(viagemF,vViagem);
viagemF.close();

sql.finalizarC();

System.out.println("Successfully Copied JSON Object to File cliente");
System.out.println("Successfully Copied JSON Object to File comboio");
System.out.println("Successfully Copied JSON Object to File viagem");
System.out.println("Successfully Copied JSON Object to File reserva");
System.out.println("JSON Object: " + cli);
System.out.println("JSON Object: " + com);
System.out.println("JSON Object: " + rs);
System.out.println("JSON Object: " + vi);
```

1.8-Explicação dos scripts

O script acede a base de dados para obter o conteúdo guardado em cada tabela através dos 4 gets anexados em foto. De seguida ira transferir essa informação para um ficheiro json. Para isso será necessário criar as respetivas classes, Cliente, Reserva, Viagem, Comboio e Lugar, com os respetivos métodos java. Por conseguinte no SQLReader iremos juntar toda a informação com os respetivos Gets, convergindo tudo para uma coleção em MongoDB.

Por fim através dos FileWriters, vão ser criados os ficheiros aonde vai ser guadada toda a informação obtida e transformada, sob o formato de ArrayList.

1.9-Queries

```
> db.reserva.find( {} , {"viagem":"Chaves",viagem:"Porto"})
```

Ilustração 1: Mostra as reservas feitas para essa viagem

2. Conclusões e Trabalho Futuro

Depois de acabado este trabalho, cabe-nos fazer uma análise crítica, justa em relação a este 2ºtrabalho.

No nosso entender, houve varias dificuldades em como começar, isto é, como migrava os dados de umas base de dados para a outra. Depois de muito pesquisar, vimos que tínhamos que usar um script que migra a base de dados em Mysql para MongoDB.

E começou o nosso problema, visto que nos estava a dar um erro em datetime, visto que o Java não tem um tipo igual a esse. Depois de muito procurar chegamos a conclusão que não dava, logo fizemos por outra maneira, que também não deu resultado.

Mas lá conseguimos fazer.

Acabado de fazer um resumo das nossas tentativas de fazer o trabalho, temos que admitir que tivemos muitas dificuldades a fazer o script, visto que tínhamos que usar linguagem Java, mas la conseguimos perceber.

Acabado estes 2º trabalho, podemos evidenciar que há varias diferenças entre Modelo de Base de Dados Relacional e Não Relacional.

Para começar, podemos dizer que o Modelo de Base de Dados Relacional recupera, manipula e armazena dados estruturados em forma de tabelas enquanto que no modelo de Não Relacional há coleção de par chave-valor, Documentos(o nosso trabalho),grafos, etc.

Verificamos isto tudo através da realização destes 2 trabalhos visto que no 1º foi tudo baseado em tabelas(entidades) e suas manipulações enquanto que aqui, neste 2ºtrabalho foi tudo à base de Documentos que contém todos as informações importantes.

Mas a maior diferença é que o modelo não relacional permite partição dos dados, o que faz muito diferença para grandes volumes de dados, enquanto que no outro não, o que leva a que haja uma melhor performance.

Para terminar, queremos salientar que estes 2 trabalhos foram muito uteis para uma melhor compreensão dos modelo de Base de Dados quer Relacional(1ºtrabalho) quer não Relacional(2ºtrabalho).