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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Projeto de um Sistema de Bases de Dados não Relacional

Bruno Miguel Salgado Machado (A74941)

Fábio Luís Baião da Silva (A75662)

João Rui de Sousa Miguel (A74237)

Luís Manuel Leite Costa (A74819)

Janeiro, 2017

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Projeto de um Sistema de Bases de Dados não Relacional

Bruno Miguel Salgado Machado (A74941)

Fábio Luís Baião da Silva (A75662)

João Rui de Sousa Miguel (A74237)

Luís Manuel Leite Costa (A74819)

Janeiro, 2017

Resumo

Este projeto tem como objetivo a migração da Base de Dados Relacional desenvolvida no primeiro trabalho prático para uma Base de Dados não Relacional (orientada a documentos - MongoDB).

O método para a migração da Base de Dados irá ser composto por três fases: a definição de um modelo de dados para a base de dados do novo sistema, o desenvolvimento de um processo para a migração de dados do sistema relacional para o sistema não relacional e a implementação das *queries* criadas para o modelo relacional, no modelo não relacional.

Área de Aplicação: Desenho, arquitetura e migração de Sistemas de Bases de Dados.
Palavras-Chave: Bases de Dados Relacionais e não Relacionais, JAVA, MongoDB.

Índice

Índice de Figuras

1. Introdução

No âmbito da disciplina de Base de Dados foi-nos proposto, a tarefa de transacionarmos o projeto feito anteriormente em MySQL, para um ambiente não relacional nomeadamente *Document Stores* com o programa MongoDB. Foi também proposto a criação de *queries* em MongoDB equivalente às realizadas anteriormente em MySQL.

Nos capítulos seguintes encontram-se explicações sucintas dos passos tomados para a realização da migração da Base Dados com auxilio de JAVA, para um ambiente não relacional, bem como a realização de novas *queries*.

2. Definição do Modelo de Dados

A definição do modelo de dados para o sistema de base de dados não relacional foi feita a partir do modelo conceptual construído no primeiro trabalho prático.

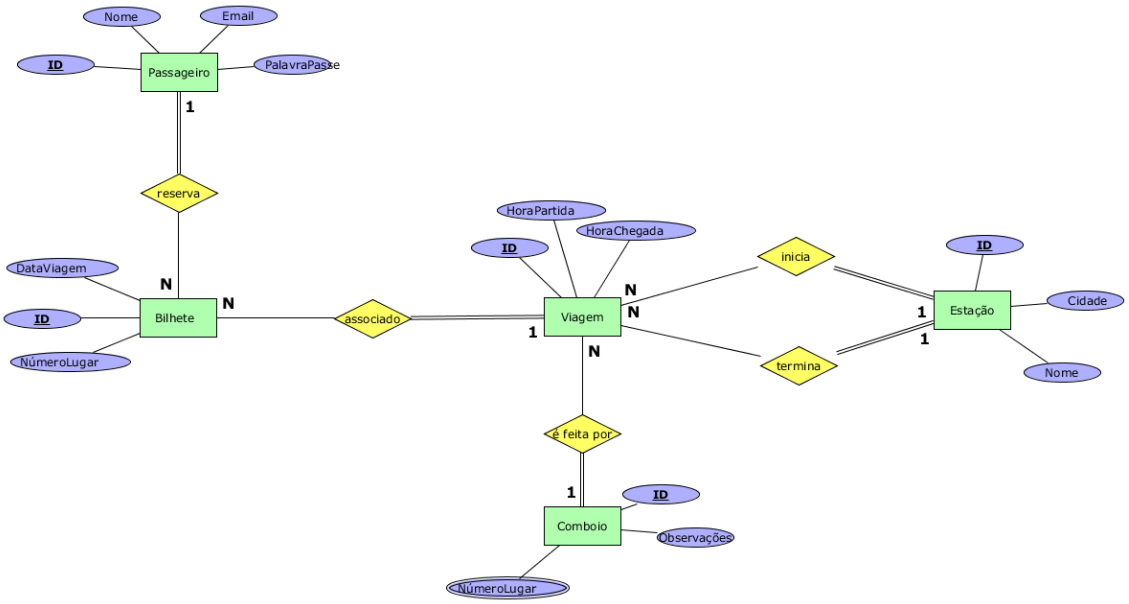

Figura 1 - Modelo Conceptual

O modelo não relacional que vamos definir é orientado a documentos. Neste tipo de modelos uma base de dados é composta por coleções. Cada coleção é constituída por um conjunto de documentos. Os documentos são estruturas de dados do tipo campo-valor.

2.1 Entidades

As entidades do modelo conceptual são tabelas no modelo físico de um sistema relacional. Num sistema não relacional orientado a documentos as entidades passam a ser coleções. Assim, as coleções da base de dados não relacional que estamos a definir são: Passageiro, Bilhete, Viagem, Comboio e Estação.

2.2 Atributos

Os atributos simples de cada entidade são colunas da respetiva tabela no modelo físico de um sistema relacional. No sistema não relacional que estamos a definir os atributos simples estão representados nos campos dos documentos. Por exemplo, os atributos Nome, Email e Password da entidade Passageiro encontram-se nos campos dos documentos da coleção Passageiro. Para as outras entidades o processo é exatamente o mesmo.

A chave primária nos sistemas não relacionais é sempre o campo _id dos documentos, independentemente da coleção a que pertencem. Assim, na definição do modelo de dados para este novo sistema não definimos explicitamente as chaves primárias.

A entidade Comboio tem um atributo multi-valor que representa os números dos lugares que contém. Num sistema relacional os atributos muti-valor são representados numa tabela. No sistema não relacional orientado a documentos que estamos a construir é possível que os campos dos documentos sejam *arrays*. Assim, os documentos da entidade Comboio para além de terem o campo Observações, têm ainda o campo lugares que será um *array*.

2.3 Relacionamentos

Os relacionamentos do modelo conceptual construído na primeira parte do trabalho são todos de 1:N. Num sistema relacional estes relacionamentos são representados através de chaves estrangeiras. No entanto, no sistema não relacional que estamos a desenvolver não existem chaves estrangeiras. Existem, porém, duas alternativas: *embedding* e referenciamento.

O *embedding* consiste em incluir o documento do lado n do relacionamento num campo *array* do documento do lado 1. Por exemplo, cada documento da coleção Passageiro teria um campo Bilhetes onde cada elemento do *array* seria um documento Bilhete, eliminando a coleção Bilhete. No entanto, cada documento Viagem também teria de ter um *array*, com os bilhetes com os quais se relaciona. Isto iria provocar uma replicação de documentos nas várias coleções, o que não é recomendável. Esta situação também se verifica no caso das viagens.

Assim, é necessário recorrer à segunda alternativa: referenciamento. Este método de representar os relacionamentos, assemelha-se às chaves estrangeiras dos sistemas relacionais. É adicionado um campo no documento do lado n do relacionamento que referencia (ou seja, com o valor da chave primária) do documento do lado 1.

Conclui-se, nesta matéria, que os documentos da coleção Bilhete terão campos que referenciam os documentos das coleções Passageiro e Viagem; e os documentos terão campos que referenciam os documentos das coleções Comboio e Estação.

2.4 Esquema

A partir da informação dos três tópicos anteriores construímos o seguinte esquema para a base de dados do novo sistema.

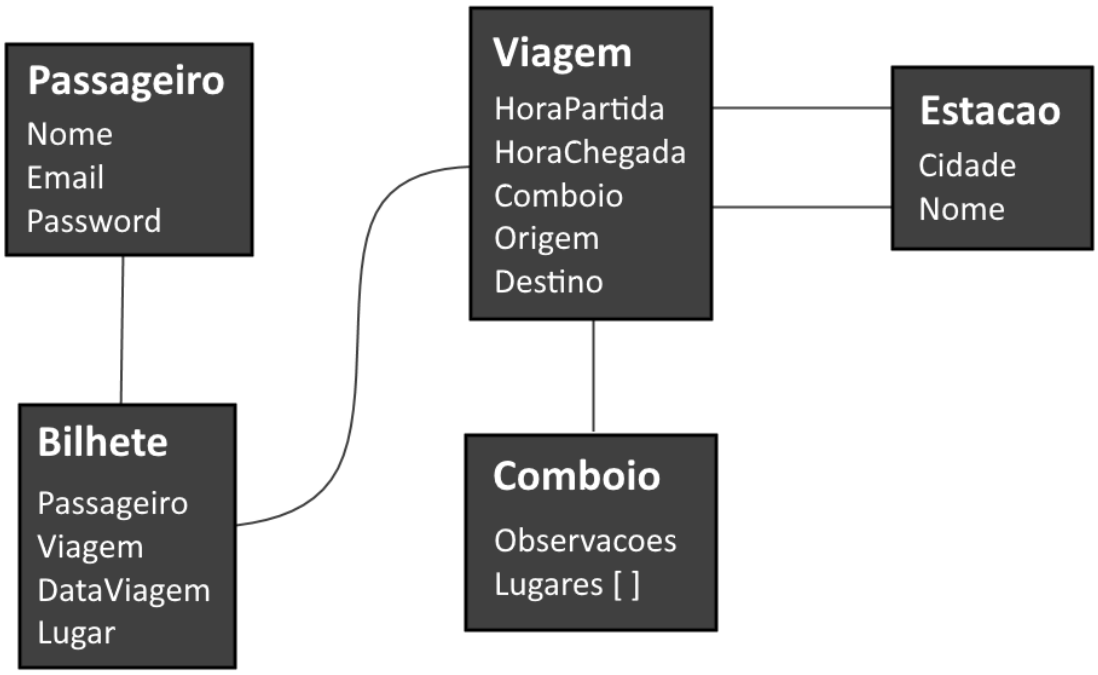

Figura 2 - Esquema para a Base de Dados não Relacional

3. Processo para a migração de dados

Para realizar a migração dos dados contidos no sistema relacional para o novo sistema, decidimos utilizar a linguagem Java. O processo utilizado foi muito simples: para cada tabela é aberta uma conexão ao MySQL sendo extraídos todos os dados dessa tabela; de seguida os dados extraídos são tratados um a um (criando um documento para cada linha) para, no final, serem inseridos na coleção respetiva do MongoDB.

Apesar da simplicidade fazemos, de seguida, uma explicação pormenorizada da migração dos dados de algumas tabelas.

3.1 Passageiros

A migração dos dados dos passageiros é feita da seguinte forma:

- Com o método `connect` da classe `Connect` (código em anexo) é aberta uma conexão ao MySQL.
- De seguida são obtidas todas as linhas presentes na tabela Passageiro (através da *query*: `SELECT * FROM Passageiro`) e guardadas num `ResultSet`.
- Para cada linha obtida é criado um `ObjectId` (a chave primária por defeito no MongoDB) e é guardado um mapeamento entre o `id` do MySQL e o `ObjectId` gerado para o MongoDB (para futura utilização nos documentos da coleção Bilhete como referência). É ainda criado um documento com os dados respetivos (Nome, Email e Password).
- Quando todas as linhas tiverem sido percorridas e todos os documentos tiverem sido criados é aberta uma conexão ao MongoDB e são inseridos todos os documentos na coleção Passageiro.

```java
private static Map<Integer, ObjectId> migratePassageiros(){
    Connection conn = null;
    Map<Integer, ObjectId> idsPassageiros = new HashMap<>();
    try{
        conn = Connect.connect();
        Statement stm = conn.createStatement();
        ResultSet rs = stm.executeQuery("SELECT * FROM Passageiro");
        List<Document> documents = new ArrayList<>();
        while (rs.next()){
            int id = rs.getInt("id");
            ObjectId o = new ObjectId();
            idsPassageiros.put(id, o);
            Document d = new Document("_id", o)
                    .append("nome", rs.getString("nome"))
                    .append("email", rs.getString("email"))
                    .append("password", rs.getString("palavrapasse"));

            documents.add(d);
        }
        MongoClient mongoClient = new MongoClient();
        MongoDatabase database = mongoClient.getDatabase("comboios");
        MongoCollection<Document> collection = database.getCollection("passageiro");
        collection.insertMany(documents);
        mongoClient.close();
    }
    catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
    }
    finally{
        Connect.close(conn);
    }
    return idsPassageiros;
}
```

Figura 3 - Código para a migração dos dados dos passageiros

O processo utilizado para a os dados das estações é idêntico.

3.2 Comboios

O método para a migração dos comboios é, também, semelhante ao método usado para a migração dos passageiros e estações. No entanto, como foi explicado na secção 2.2, a informação dos números dos lugares de cada comboio vai ser guardada num *array* no documento do comboio, enquanto que no MySQL estes estão guardados numa tabela à parte.

Assim, para cada linha identificando um comboio, é necessário obter-se os números dos lugares referentes a esse comboio, sendo depois introduzidos no documento da mesma forma que os restantes dados.

```java
private static List<Integer> migrateLugares(int id){
    List<Integer> lugares = new ArrayList<>();
    Connection conn = null;
    try{
        conn = Connect.connect();
        PreparedStatement stm = conn.prepareStatement("SELECT * FROM LugarComboio WHERE comboio_id = ?");
        stm.setInt(1, id);
        ResultSet rs = stm.executeQuery();
        while(rs.next()){
            lugares.add(rs.getInt("NumeroLugar"));
        }

    } catch (SQLException | ClassNotFoundException ex) {
        System.out.println(ex.getMessage());
    }
    finally{
        Connect.close(conn);
    }
    return lugares;
}
```

Figura 4 - Código para obter os números dos lugares de um comboio

3.3 Viagens

Na migração dos dados referentes às viagens oferecidas pela empresa, a diferença em relação às outras migrações já abordadas, está na utilização dos mapeamentos entre o `id` e o `ObjectId` construídos nessas migrações.

Ao criar cada documento, nos campos em que é necessário colocar uma referência, é obtida a chave estrangeira respetiva e, através do mapeamento respetivo também, é obtido o `ObjectId`, sendo colocado no campo correspondente.

```java
private static Map<Integer, ObjectId> migrateViagens(Map<Integer, ObjectId> idsComboios, Map<Integer, ObjectId> idsEstacoes){
    Connection conn = null;
    Map<Integer, ObjectId> idsViagens = new HashMap<>();
    try{
        conn = Connect.connect();
        Statement stm = conn.createStatement();
        ResultSet rs = stm.executeQuery("SELECT * FROM Viagem");
        List<Document> documents = new ArrayList<>();
        while (rs.next()){
            int id = rs.getInt("id");
            ObjectId o = new ObjectId();
            idsViagens.put(id, o);
            Document d = new Document("_id", o)
                    .append("comboio", idsComboios.get(rs.getInt("comboio_id")))
                    .append("horapartida", rs.getTime("horapartida"))
                    .append("horachegada", rs.getTime("horachegada"))
                    .append("origem", idsEstacoes.get(rs.getInt("origem")))
                    .append("destino", idsEstacoes.get(rs.getInt("destino")));

            documents.add(d);
        }
        MongoClient mongoClient = new MongoClient();
        MongoDatabase database = mongoClient.getDatabase("comboios");
        MongoCollection<Document> collection = database.getCollection("viagem");
        collection.insertMany(documents);
        mongoClient.close();
    }
    catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
    }
    finally{
        Connect.close(conn);
    }
    return idsViagens;
}
```

Figura 5 - Código para a migração dos dados das viagens

A única diferença, em termos lógicos, da migração das viagens e dos bilhetes, é que nos bilhetes não é criado nenhum mapeamento, ao contrário das viagens em que é criado para posterior utilização na migração dos bilhetes.

4. Implementação de *Queries*

Numa migração de sistemas de bases de dados, para além da construção de um modelo de dados compatível com o novo sistema e da migração dos dados, é igualmente importante reconstruir as *queries* na linguagem específica desse mesmo novo sistema.

As *queries* que foram reimplementadas são:

- Consultar viagens: dada uma cidade de origem e uma de destino são apresentadas todas as viagens que são realizadas entre essas duas cidades.
- Consultar lugares: dada uma viagem e uma data são apresentados todos os lugares ainda disponíveis do comboio que realiza a viagem.
- Consultar bilhetes: dado um email e uma password são apresentados todos os bilhetes reservados pelo passageiro correspondente.

4.1 Consultar Viagens

A *query* para realizar a consulta das viagens usa a operação `$lookup` duas vezes para fazer o *join* entre os documentos das coleções viagem e estação (origem e destino). De seguida é feito um `$match` para selecionar apenas as viagens que tenham como cidade das estações origem e destino as cidades indicadas. Por fim, é utilizada a operação `$project` para apresentar apenas os dados essenciais.

Exemplo de consulta para viagens com origem em Lisboa e destino Madrid:

```
db.viagem.aggregate([
{
        $lookup: {
                from: "estacao",
                localField: "origem",
                foreignField: "_id",
                as: "origem"
        }
},
{
        $lookup: {
                from: "estacao",
                localField:"destino",
                foreignField: "_id",
                as: "destino"
        }
},
{
        $match: {
                "origem.cidade": "Lisboa",
                "destino.cidade": "Madrid"
        }
},
{
        $project: {
                "horapartida" : {
                        $dateToString: {
                                format: "%H:%M", date: "$horapartida"
                        }
                },
                "horachegada" : {
                        $dateToString: {
                                format: "%H:%M", date: "$horachegada"
                        }
                },
                "origem": {
                        $arrayElemAt: ["$origem", 0]
                },
                "destino": {
                        $arrayElemAt: ["$destino", 0]
                }
        }
},
{
        $project: {
                "origem._id": 0,
                "destino._id": 0
        }
}
]).pretty()
```

Figura 6 - *Query* para consultar viagens

4.2 Consultar Lugares

Para consultar os lugares disponíveis do comboio, começa por ser feito o $match da viagem que se pretende consultar. De seguida é utilizada a operação `$lookup` duas vezes para saber qual o comboio que realiza a viagem pretendida e os bilhetes que foram reservados para essa viagem. No passo seguinte é feito um `$filter` para filtrar os bilhetes que foram reservados para a viagem escolhida no dia pretendido. Por fim, é feito mais um `$filter` para serem apenas apresentados os lugares do comboio que ainda não foram reservados.

Exemplo de consulta de lugares para a viagem com `ObjectId` = “587fe83863a09e69a2adac30” para a dia 6 de Janeiro de 2017:

```
db.viagem.aggregate([
{
        $match:            {
                "_id": ObjectId("587fe83863a09e69a2adac30")
        }
},
{
        $lookup: {
                from: "comboio",
                localField: "comboio",
                foreignField: "_id",
                as: "comboio"
        }
},
{
        $lookup: {
                from: "bilhete",
                localField: "_id",
                foreignField: "viagem",
                as: "bilhetes"
        }
},
{
        $project: {
                "comboio": {
                        $arrayElemAt: ["$comboio", 0]
                },
                "bilhetes": {
                        $filter: {
                                input: "$bilhetes",
                                as: "bilhete",
                                cond: {
                                        $eq : ["$$bilhete.viagem", Date("2017-01-06")]
                                }
                        }
                },
                "ocupados": "$bilhetes.lugar"
        }
},
{
        $project: {
                "_id": 0,
                "lugares": {
                        $filter: {
                                input: "$comboio.lugares",
                                as: "num",
                                cond: {
                                        $not: [{$in: ["$$num", "$ocupados"]}]
                                }
                        }
                }
        }
}
]).pretty()
```

Figura 7 - *Query* para consultar lugares

4.3 Consultar Bilhetes

A *query* que permite a um passageiro consultar os bilhetes reservados pelo próprio, apesar de comprida, é simples. Em primeiro lugar é utilizada a operação `$lookup` fazendo a correspondência de cada bilhete com o respetivo passageiro. De seguida, são selecionados

apenas os bilhetes que tenham sido reservados pelo passageiro com o email e password fornecidos, usando a operação `$match`. Os passos seguintes consistem em `$lookup` e `$project`, que apenas servem para apresentar dados relevantes de cada bilhete.

Exemplo de consulta de Bilhetes para o passageiro com o email joaorui@email.com e password joaorui:

```
db.bilhete.aggregate([
{
        $lookup: {
                from: "passageiro",
                localField: "passageiro",
                foreignField: "_id",
                as: "passageiro"
        }
},
{
        $match: {
                "passageiro.email": "joaorui@email.com",
                "passageiro.password": "joaorui"
        }
},
{
        $lookup: {
                from: "viagem",
                localField: "viagem",
                foreignField: "_id",
                as: "viagem"
        }
},
{
        $project: {
                "_id": 0,
                "lugar": 1,
                "viagem": {
                        $arrayElemAt: ["$viagem", 0]
                },
                "dataviagem": {
                        $dateToString: {
                                format: "%Y-%m-%d", date: "$dataviagem"
                        }
                }
        }
},
{
        $lookup: {
                from: "estacao",
                localField: "viagem.origem",
                foreignField: "_id",
                as: "origem"
        }
},
{
        $lookup: {
                from: "estacao",
                localField: "viagem.destino",
                foreignField: "_id",
                as: "destino"
        }
},
{
        $project: {
                "viagem.horapartida" : {
                        $dateToString: {
                                format: "%H:%M", date: "$viagem.horapartida"
                        }
                },
                "viagem.horachegada" : {
                        $dateToString: {
                                format: "%H:%M", date: "$viagem.horachegada"
                        }
                },
                "origem": {
                        $arrayElemAt: ["$origem", 0]
                },
                "destino": {
                        $arrayElemAt: ["$destino", 0]
                },
                "lugar": 1,
                "dataviagem": 1
        }
},
{
        $project: {
                "origem._id": 0,
                "destino._id": 0
        }
}
]).pretty();
```

Figura 8 - *Query* para consultar bilhetes

5. Análise crítica

Um dos objetivos em termos de desempenho na utilização do MongoDB é a possibilidade de obter a mesma informação de apenas um documento, quando no modelo relacional era necessário fazer *JOIN* de várias tabelas. Para isso, representa-se os relacionamentos fazendo *embedding* dos documentos. No entanto, como foi explicado anteriormente, no modelo de dados definido para este sistema (equivalente ao modelo definido no primeiro trabalho) é necessário usar referências ao invés de “embeber” documentos. Com as referências, a necessidade de utilizar *JOINs* não desaparece e, assim, o objetivo não é atingido.

Para além de em termos de desempenho não haver melhorias, ainda existe a questão do desenvolvimento das *queries*, que se revelaram mais complicadas de fazer (provavelmente por desconhecimento da notação) em relação às *queries* em SQL.

6. Conclusões e Trabalho Futuro

Este trabalho foi concluído com alguma facilidade. Desde cedo percebemos que o modelo de dados para esta segunda fase seria semelhante ao da fase anterior, tal como seria de esperar, para que se pudesse proceder à migração dos dados.

Optámos por fazer o script de migração em java, uma vez que esta fornece Drivers para ligações tanto com SQL como MongoDB, o que facilita todo o processo.

A nosso ver a implementação desta base de dados, num modelo não relacional, não trará qualquer vantagem dada a existência de relacionamentos. Estes relacionamentos fazem com que seja necessário recorrer a métodos de referenciação. Caso não utilizássemos estes métodos, ocuparíamos muito espaço desnecessariamente, uma vez que, se optássemos por embutir documentos dentro de outros documentos, correríamos o risco de existir muita informação duplicada.

Referências

https://docs.mongodb.com/manual/
https://mongodb.github.io/mongo-java-driver/3.4/

Anexos

I. Método `main`

```java
public static void main(String[] args){
    Map<Integer, ObjectId> idsPassageiros = migratePassageiros();
    Map<Integer, ObjectId> idsComboios = migrateComboios();
    Map<Integer, ObjectId> idsEstacoes = migrateEstacoes();
    Map<Integer, ObjectId> idsViagens = migrateViagens(idsComboios, idsEstacoes);
    migrateBilhetes(idsViagens, idsPassageiros);
}
```

II. Classe Connect

```java
public class Connect {

    private static final String URL = "localhost";
    private static final String DB = "despesas";
    private static final String USERNAME = "root"; //TODO: alterar
    private static final String PASSWORD = "root"; //TODO: alterar

    /**
     * Estabelece ligação à base de dados
     * @return
     * @throws SQLException
     * @throws ClassNotFoundException
     */
    public static Connection connect() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.jdbc.Driver");
        //cliente deve fechar conexão!
        return DriverManager.getConnection("jdbc:mysql://"+URL+"/"+DB+"?user="+USERNAME+"&password="+PASSWORD);
    }

    /**
     * Fecha a ligação à base de dados, se aberta.
     * @param c
     */
    public static void close(Connection c) {
        try {
            if(c!=null && !c.isClosed()) {
                c.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

III. Método `migrateEstacoes()`

```java
private static Map<Integer, ObjectId> migrateEstacoes(){
    Connection conn = null;
    Map<Integer, ObjectId> idsEstacoes = new HashMap<>();
    try{
        conn = Connect.connect();
        Statement stm = conn.createStatement();
        ResultSet rs = stm.executeQuery("SELECT * FROM Estacao");
        List<Document> documents = new ArrayList<>();
        while (rs.next()){
            int id = rs.getInt("id");
            ObjectId o = new ObjectId();
            idsEstacoes.put(id, o);
            Document d = new Document("_id", o)
                    .append("cidade", rs.getString("cidade"))
                    .append("nome", rs.getString("nome"));

            documents.add(d);
        }
        MongoClient mongoClient = new MongoClient();
        MongoDatabase database = mongoClient.getDatabase("comboios");
        MongoCollection<Document> collection = database.getCollection("estacao");
        collection.insertMany(documents);
        mongoClient.close();
    }
    catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
    }
    finally{
        Connect.close(conn);
    }
    return idsEstacoes;
}
```

IV. Método `migrateComboios()`

```
private static Map<Integer, ObjectId> migrateComboios(){
    Connection conn = null;
    Map<Integer, ObjectId> idsComboios = new HashMap<>();
    try{
        conn = Connect.connect();
        Statement stm = conn.createStatement();
        ResultSet rs = stm.executeQuery("SELECT * FROM Comboio");
        List<Document> documents = new ArrayList<>();
        while (rs.next()){
            int id = rs.getInt("id");
            List<Integer> lugares = migrateLugares(id);
            ObjectId o = new ObjectId();
            idsComboios.put(id, o);
            Document d = new Document("_id", o)
                    .append("observacoes", rs.getString("observacoes"))
                    .append("lugares", lugares);

            documents.add(d);
        }
        MongoClient mongoClient = new MongoClient();
        MongoDatabase database = mongoClient.getDatabase("comboios");
        MongoCollection<Document> collection = database.getCollection("comboio");
        collection.insertMany(documents);
        mongoClient.close();
    }
    catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
    }
    finally{
        Connect.close(conn);
    }
    return idsComboios;
}
```

V. Método `migrateBilhetes()`

```java
private static void migrateBilhetes(Map<Integer, ObjectId> idsViagens, Map<Integer, ObjectId> idsPassageiros){
    Connection conn = null;
    try{
        conn = Connect.connect();
        Statement stm = conn.createStatement();
        ResultSet rs = stm.executeQuery("SELECT * FROM Bilhete");
        List<Document> documents = new ArrayList<>();
        while (rs.next()){
            int id = rs.getInt("id");
            ObjectId o = new ObjectId();
            Document d = new Document("_id", o)
                    .append("passageiro", idsPassageiros.get(rs.getInt("passageiro_id")))
                    .append("viagem", idsViagens.get(rs.getInt("viagem_id")))
                    .append("dataviagem", rs.getDate("dataviagem"))
                    .append("lugar", rs.getInt("lugarcomboio_numerolugar"));

            documents.add(d);
        }
        MongoClient mongoClient = new MongoClient();
        MongoDatabase database = mongoClient.getDatabase("comboios");
        MongoCollection<Document> collection = database.getCollection("bilhete");
        collection.insertMany(documents);
        mongoClient.close();
    }
    catch (ClassNotFoundException | SQLException e){
        System.out.println(e.getMessage());
    }
    finally{
        Connect.close(conn);
    }
}
```