



**Universidade do Minho**

Mestrado Integrado em Engenharia Informática  
Licenciatura em Ciências da Computação

## **Unidade Curricular de Bases de Dados**

Ano Lectivo de 2018/2019

### **Loja BracaTECH**

**Filipa Correia Parente - A82145**

**José André Martins Pereira – A82880**

**Rafaela Maria Soares da Silva – A79034**

**Ricardo André Gomes Petronilho – A81744**





Data de	
Recepção	
Responsável	
Avaliação	
Observações	

## Loja BracaTECH

**Filipa Correia Parente - A82145**

**José André Martins Pereira – A82880**

**Rafaela Maria Soares da Silva – A79034**

**Ricardo André Gomes Petronilho – A81744**

Janeiro, 2019

<</opcional Dedicatória>>

## Resumo

Este relatório refere-se à segunda parte do projeto BracaTECH - migração de um Sistema de Base de Dados relacional para um Sistema de Base de Dados não relacional.

No âmbito da realização do projeto em *NoSQL*, o tipo de modelo de dados não relacional adoptado foi o *MongoDB*.

**Área de Aplicação:** <<Identificação da Área de trabalho. Por exemplo: Desenho e arquitetura de Sistemas de Bases de Dados.>>

**Palavras-Chave:** NoSQL, *MongoDB*

# Índice

1. Introdução	1
1.1. Contextualização	1
1.2. Apresentação do Caso de Estudo	1
1.3. Motivação e Objectivos	1
1.4. Estrutura do Relatório	1
2. Sugestões para Escrita do Relatório	2
2.1. Sugestões Gerais	2
2.2. Termos Estrangeiros	2
2.3. Tabelas e Figuras	2
2.4. Siglas e Acrónimos	3
2.5. Referências Bibliográficas	3
2.6. Tipo de Ficheiro	3
3. Conclusões e Trabalho Futuro	4
Bibliografia	5
Referências WWW	6
Lista de Siglas e Acrónimos	7
 Anexos	
I. Anexo 1	9

## **Índice de Figuras**

**Não foi encontrada nenhuma entrada do índice de ilustrações.**

## **Índice de Tabelas**

Tabela 1 - Ilustração de inserção de uma tabela e sua legenda.	3
--	---



# **1. Introdução**

## **1.1. Contextualização**

Como referido na primeira fase do projeto, a empresa BracaTECH tem como intuito destacar-se pela qualidade de serviço.

## **1.2. Apresentação do Caso de Estudo**

A administração do departamento de comunicação estratégica e do departamento financeiro, depois de optarem pela implementação de um Sistema de Base de Dados relacional para gerir informações relativas aos produtos, serviços, funcionários e clientes, decidiram migrar para um Sistema de Base de Dados não relacional.

A decisão surgiu pelo facto de que as consultas envolvem grandes volumes de dados, derivado à afluência de clientes nos últimos três meses.

Com as previsões de aumento de clientes no futuro próximo, visto que, está em curso a abertura de mais lojas BracaTECH, o processamento dos dados irá tornar-se pouco eficiente, pois a base de dados relacional não é adequada para o tratamento de grandes quantidades de dados.

Assim, as equipas dos departamentos decidiram migrar para um sistema NoSQL, com a finalidade de aumentar a rapidez do processamento das queries.

## **1.3. Motivação e Objectivos**

O departamento de comunicação estratégica necessita de dar primazia à rapidez do sistema, visto que tem de atuar o mais rápido possível para conseguir “fidelizar” os consumidores.

Nesse sentido, necessita de aceder às informações dos clientes e das vendas de forma rápida para conseguir extrair o padrão de gosto e evidenciar a aderência a certas tendências tecnológicas. Assim, terá mais hipótese de cativar constantemente o cliente com anúncios apelativos.

Já o departamento financeiro tem como objetivo otimizar o lucro da empresa, necessitando de ter acesso rápido ao que está a causar prejuízo. Devido à expansão da rede de lojas BracaTech, otimizar o lucro é essencial.

## **1.4. Estrutura do Relatório**

Neste relatório será apresentado o processo de migração, incluindo a definição da estrutura base, o mapeamento do processo e as fases do mesmo.

Será ainda apresentado as *queries* mais importantes realizadas na primeira fase do projeto.

## 2. Processo de migração

### 2.1. Estrutura em *MongoDB*

Para a estruturação da coleção Funcionário, decidiu-se apresentar os seguintes campos:

```
{ "id": 1,
  "nome_completo": "Ricardo Andre Gomes Petronilho",
  "tipo": "n",
  "sexo": "M",
  "data_nascimento": "1998-06-29",
  "data_registo_perfil": "2017-10-28 14:50:38",
  "valor_total_vendas": 0,
  "salario": 1850,
  "contacto": { "email": "rpetronilho@gmail.com",
    "telefone": [{"tipo": "p", "numero": 912345678}]
  },
  "endereco": { "cidade": "Braga", "freguesia": "Fraiao", "rua": "Travessa da Quinta", "codigo_postal": "4715-338" }
}
```

Figura 1 – Estrutura do Funcionário

Decidiu-se manter todos os atributos que estavam implementados na entidade Funcionário em MySQL.

Nesse intuito, como a entidade endereço é um atributo composto (em MySQL), decidiu-se que o campo endereco contém a cidade, freguesia, rua e código postal.

Como a entidade telefones é um atributo multivalorado composto pelo tipo de telefone (em MySQL), decidiu-se que o campo contacto iria conter o email(entidade simples em MySQL) e o telefone(array de tópos de dois elementos – o primeiro relativo ao tipo e o segundo ao número de telefone), de forma a simplificar a apresentação.

A estrutura da coleção Cliente é semelhante à do Funcionário, tendo disso mantido todos os atributos e a opção pelo contacto e endereço decida da forma atrás referida.

```
{
  "nif": 748637829,
  "nome_completo": "Miguel Nuno Martins Oliveira",
  "sexo": "M",
  "valor_total_descontos": 0,
  "valor_total_gasto": 0,
  "data_nascimento": "1997-11-22",
  "data_registo_perfil": "2016-5-31",
  "contacto": {
    "email": "mnunooliveira@gmail.com",
    "telefone": [{ "tipo": "p", "numero": "701223730" }, { "tipo": "t", "numero": "154488636" } ]
  },
  "profissao": "Canalizador",
  "endereco": {
    "codigo_postal": "4715-500",
    "cidade": "Braga",
    "freguesia": "Gualtar",
    "rua": "Rua da Borga"
  },
  "classificacao": 0
}
```

Figura 2 – Estrutura do Cliente

A Venda é a entidade que corresponde à venda quer de produtos quer de serviços.

Anteriormente (no modelo relacional), os Produtos estavam relacionados com a Venda através de uma relação de muitos para muitos (N para N), resultava assim numa nova tabela, a VendaProduto, correspondente a esta relação.

Em NoSQL resolvemos esta relação embebendo a tabela VendaProduto na coleção Venda. Assim quando produtos são vendidos, no documento correspondente a essa venda existe um array de Produtos associados tal como se pode verificar na seguinte figura.

▼ (1) ObjectId("5c3cd825eaa35205783457d6")	{ 8 fields }
_id	ObjectId("5c3cd825eaa35205783457d6")
id	1
data	2018-11-25 00:00:00.000Z
preco_total	31366.2
desconto_total	151.8
▶ Cliente	{ 2 fields }
▶ Funcionario	{ 2 fields }
▼ Produtos	[ 3 elements ]
▼ [0]	{ 7 fields }
id_produto	1
designacao	Macbook Air - Edição 2018
quantidade	10
preco_unitario_final	1500.0
preco_total	15000.0
desconto_unitario	0.0
desconto_total	0.0
▶ [1]	{ 7 fields }
▶ [2]	{ 7 fields }

Figura 3 - Array de produtos no documento de uma Venda

Por outro lado, quando serviços são vendidos, aparece o campo Serviço associado ao documento dessa venda, tal como se pode verificar na seguinte figura.

▼ (3) ObjectId("5c3cd825eaa35205783457d8")	{ 8 fields }
_id	ObjectId("5c3cd825eaa35205783457d8")
# id	3
data	2017-06-23 23:00:00.000Z
# preco_total	100.0
# desconto_total	0.0
▶ Cliente	{ 2 fields }
▶ Funcionario	{ 2 fields }
▼ Serviço	{ 3 fields }
# id_servico	1
data_inicio	2017-06-22 23:00:00.000Z
data_fim	2017-06-23 23:00:00.000Z

Figura 4 – Campo Serviço no documento de uma Venda

O facto de cada documento poder variar a sua estrutura na mesma coleção é uma vantagem nas bases de dados não relacionais.

De forma a evitar agregações de coleções, armazenamos o nome, tanto do **Funcionário** como do **Cliente** no próprio documento da **Venda**, como se pode verificar na seguinte figura.

▼ (7) ObjectId("5c3cd825eaa35205783457dc")	{ 8 fields }
_id	ObjectId("5c3cd825eaa35205783457dc")
# id	7
data	2017-08-28 23:00:00.000Z
# preco_total	300.0
# desconto_total	0.0
▼ Cliente	{ 2 fields }
# nif	214512313
# nome_completo	Gabriela Maria Soares
▼ Funcionario	{ 2 fields }
# id_funcionario	3
# nome_funcionario	Maria Moreira
▶ Serviço	{ 3 fields }

Figura 5 – Campo Cliente e Funcionario no documento de uma Venda

Além do nome também se armazenam as respetivas chaves estrangeiras do Cliente e Funcionario, o nif e id\_funcionario, caso seja necessário procurar mais informações.

Tal como na primeira fase do projeto em MySQL, a entidade Servico corresponde ao conjunto de serviços que a empresa BracaTECH prestou, ou ainda está a prestar, no caso de uma venda correspondente a um serviço.

```
{
  "_id" : ObjectId("5c3ddd6cb86eab4444210d98"),
  "id" : 7,
  "descricao" : "Limpeza interna cooler e chassis",
  "data_inicio" : "2017-09-28 19:35:12.0",
  "data_fim" : null,
  "estado Equipamento" : "Ecra rachado no canto superior direito",
  "id_funcionario" : "1"
}
```

Figura 6 – Estrutura do Servico

É de realçar que, como os atributos “data\_inicio” e “data\_fim” correspondem, respetivamente à data de inicio e de fim da realização desse serviço, “data\_fim” : null significa que o serviço ainda não está terminado.

A entidade Produto corresponde ao conjunto de produtos que a loja dispõe para a venda ao público.

```
{
  "_id" : ObjectId("5c3ddd6bb86eab4444210d8e"),
  "id" : 5,
  "designacao" : "RAM HYPREX",
  "descricao" : "Especificações: Read: 500 MB/s; Write: 500 MB/s",
  "categoria" : "Armazenamento",
  "stock" : 500,
  "preco_unitario" : 70,
  "desconto" : 0.05000000074505806
}
```

Figura 7 – Estrutura do Produto

Como se pode constatar, à semelhança da entidade Servico, a entidade Produto também manteve a mesma estrutura da primeira fase do projeto.

## 2.2. Implementação do processo de Migração

A migração dos dados da base de dados relacional para NoSQL foi realizada elaborando uma aplicação na linguagem JAVA.

A aplicação foi definida para se conectar à base de dados do SQL definida na primeira fase do trabalho, utilizando a API **mysql-connector-java-5.1.47**:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

public class Connect {
    public static Connection connect() throws SQLException, ClassNotFoundException {
        Class.forName("com.mysql.jdbc.Driver");

        return DriverManager.getConnection( url: "jdbc:mysql://localhost/BracaTECH?user=root&password=password");
    }

    public static void close(Connection c) {
        try {
            if(c!=null && !c.isClosed()) {
                c.close();
            }
        } catch (Exception e) {
            e.printStackTrace();
        }
    }
}
```

Figura 8 – Conexão à base de dados SQL

Para a conexão à base de dados MongoDB usou-se as **drives oficiais** da versão **3.4.1**:

```
import com.mongodb.MongoClient;
import com.mongodb.client.MongoDatabase;

import java.sql.Connection;

public class FazerMigracao {

    public static void main(String[] args) throws ClassNotFoundException {
        MongoClient mongo = new MongoClient( host: "localhost", port: 27017);
        MongoDatabase db = mongo.getDatabase( dbName: "project");

        Connection con = null;

        Migracao m = new Migracao(con, db);

        m.drop();
        m.migrarResultado();
        m.migrarResultado();
        m.migrarResultado();
        m.migrarResultado();
        m.migrarResultado();
    }
}
```

Figura 9 – Conexão à base de dados NoSQL

Tal como se pode observar na imagem acima, definiu-se um método para a migração dos dados para cada coleção e o *drop*, que remove as mesmas.

Em relação à coleção *Cliente*, tal como já foi dito anteriormente, esta irá guardar as informações normais, bem como um documento denominado por *Contacto* com o email e uma lista de telefones e um documento *Endereco* com as respetivas informações. Deste modo, inicialmente obtém-se as informações normais, usando a *query* abaixo na aplicação:

```
Select C.nif, C.nome_completo, C.sexo, C.valor_total_descontos,
       C.valor_total_gasto, C.data_nascimento, C.data_registo_perfil,
       C.email, C.profissao, C.codigo_postal, C.cidade, C.freguesia, C.rua, C.classificacao
FROM Cliente AS C
```

Figura 10 – Query usada no código abaixo



```

this.con = (Connection) Connect.connect();
String sql = "" + "Select C.nif, C.nome_completo, C.sexo, C.valor_total_descontos, C.valor_total_gasto, C.data_nasci
PreparedStatement ps = this.con.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
MongoCollection<BasicDBObject> collection = this.db.getCollection( s: "Cliente", BasicDBObject.class);

while(rs.next()){
    nif = rs.getInt( columnLabel: "C.nif");
    nome_completo = rs.getString( columnLabel: "C.nome_completo");
    sexo = rs.getString( columnLabel: "C.sexo");
    valor_total_descontos = rs.getDouble( columnLabel: "C.valor_total_descontos");
    valor_total_gasto = rs.getDouble( columnLabel: "C.valor_total_gasto");
    data_nascimento = rs.getDate( columnLabel: "C.data_nascimento");
    data_registo_perfil = rs.getDate( columnLabel: "C.data_registo_perfil");
    email = rs.getString( columnLabel: "C.email");
    profissao = rs.getString( columnLabel: "C.profissao");
    codigo_postal = rs.getString( columnLabel: "C.codigo_postal");
    cidade = rs.getString( columnLabel: "C.cidade");
    freguesia = rs.getString( columnLabel: "C.freguesia");
    rua = rs.getString( columnLabel: "C.rua");
    classificacao = rs.getInt( columnLabel: "C.classificacao"); // classificação é um int

    BasicDBObject document = new BasicDBObject();

```

Figura 11 – Busca dos dados do Cliente à base de dados SQL

Após a recolha dos dados, cria-se o documento, e adiciona-se os dados normais ao mesmo, da seguinte forma:

```

BasicDBObject document = new BasicDBObject();

// ADIÇÃO DAS INFOS INDEPENDENTES DO CLIENTE

document.put("nif", nif);
document.put("nome_completo", nome_completo);
document.put("sexo", sexo);
document.put("valor_total_descontos", valor_total_descontos);
document.put("valor_total_gasto", valor_total_gasto);
document.put("data_nascimento", data_nascimento);
document.put("data_registo_perfil", data_registo_perfil);
document.put("profissao", profissao);
document.put("classificacao", classificacao);

```

Figura 12 – Inserção dos dados no documento Cliente

De seguida, cria-se o documento contacto, ao qual adiciona-se o email, que já foi obtido, na *query* acima, e uma lista com os números de telefone, que são obtidos usando a seguinte *query*:

```

SELECT CT.nr_telefone, CT.tipo
FROM ClienteTelefone AS CT
WHERE CT.nif_cliente = ?

```

Figura 13 – Query usada no código abaixo

```

BasicDBObject documentoContato = new BasicDBObject();
documentoContato.put("email", email); // adição do email

PreparedStatement ps2 = this.con.prepareStatement( sql: "SELECT CT.nr_telefone, CT.tipo FROM ClienteTelefone AS CT W
ps2.setInt( parameterIndex: 1,nif);
ResultSet rs2 = ps2.executeQuery();
BasicDBList listaTelefones = new BasicDBList(); // lista dos telefones deste cliente
// LISTA DE TELEFONES

```

Figura 14 – Adição do campo email e criação da lista para os números de telefone

```

BasicDBList listaTelefones = new BasicDBList(); // lista dos telefones deste cliente
// LISTA DE TELEFONES

while(rs2.next()){
    nr_telefone = rs2.getInt( columnLabel: "CT.nr_telefone");
    tipo_telefone = rs2.getString( columnLabel: "CT.tipo");

    BasicDBObject documentoTelefone = new BasicDBObject();

    documentoTelefone.put("nr_telefone", nr_telefone);
    documentoTelefone.put("tipo", tipo_telefone);

    listaTelefones.add(documentoTelefone);
}

documentoContato.put("Telefones", listaTelefones); // adição da lista de telefones ao Documento Contacto
document.put("Contacto",documentoContato); // adição do documento Contacto

```

Figura 15 –Adição dos números de telefone ao documento Contacto e adição deste ao documento Cliente

No ciclo que está no retângulo verde, percorremos os dados resultantes da query acima, ou seja, os números de telefones de um Cliente, e adicionamos à listaTelefones, a mesma é adicionada ao documentoContato, e de seguida este é adicionado ao documento do Cliente.

```

// ADIÇÃO DO ENDEREÇO
BasicDBObject documentoEndereco = new BasicDBObject();

documentoEndereco.put("codigo_postal", codigo_postal);
documentoEndereco.put("cidade",cidade);
documentoEndereco.put("freguesia",freguesia);
documentoEndereco.put("rua", rua);

document.put("Endereco", documentoEndereco);
collection.insertOne(document); // adição do documento

```

Figura 16 – Criação do documento Endereço, embebido no documento Cliente

Por fim, cria-se o documento Endereco, e adiciona-se as respetivas informações que foram obtidas na primeira query, de seguida insere-se o documento Endereco no documento do Cliente (seta azul) e por fim adiciona-se o

documento do Cliente à coleção de Clientes (seta vermelha), e repete-se este processo, descrito anteriormente, para cada cliente da base de dados.

A migração para a coleção Funcionario é muito semelhante à do Cliente, portanto não será necessário explicar em detalhe. Do mesmo modo, a coleção do Produto e Servico, contêm apenas as informações básicas.

Por fim, em relação ao método migrarVenda, este é composto como já foi referenciado anteriormente por documentos embebidos ao documento Venda, assim desta forma, inicialmente busca-se os dados normais com a seguinte query e insere-se no documento:

```
Select V.id, V.data_venda, V.preco_total, V.valor_desconto
FROM Venda AS V
```

Figura 17 – Query usada no código abaixo

```
this.con = (Connection) Connect.connect();
String sql = "" + "Select V.id, V.data_venda, V.preco_total, V.valor_desconto FROM Venda AS V \n";
PreparedStatement ps = this.con.prepareStatement(sql);
ResultSet rs = ps.executeQuery();
MongoCollection<BasicDBObject> collection = this.db.getCollection("Venda", BasicDBObject.class);

while(rs.next()){
    id = rs.getInt( columnLabel: "V.id");
    data = rs.getDate( columnLabel: "V.data_venda");
    preco_total = rs.getDouble( columnLabel: "V.preco_total");
    desconto_total = rs.getDouble( columnLabel: "V.valor_desconto");

    BasicDBObject document = new BasicDBObject();

    document.put("id", id);
    document.put("data", data);
    document.put("preco_total", preco_total);
    document.put("desconto_total", desconto_total);
}
```

Figura 18 – Criação do documento Venda

De seguida, é necessário incorporar no documento Venda, dois documentos embebidos com a informação do cliente e outro com a do funcionário, usando as seguintes queries e código:

```
SELECT C.nif, C.nome_completo
FROM Venda AS V
INNER JOIN Cliente AS C
ON C.nif = V.nif_cliente
WHERE V.id = ?
```

Figura 19 – Query usada no código abaixo

```

PreparedStatement ps2 = this.con.prepareStatement( sql: "SELECT C.nif, C.nome_co
ps2.setInt( parameterIndex: 1, id);
ResultSet rs2 = ps2.executeQuery();
while (rs2.next()){
    nif_cliente = rs2.getInt( columnLabel: "C.nif");
    nome_completo_cliente = rs2.getString( columnLabel: "C.nome_completo");

    BasicDBObject documentoCliente = new BasicDBObject();

    documentoCliente.put("nif",nif_cliente);
    documentoCliente.put("nome_completo", nome_completo_cliente);

    document.put("Cliente", documentoCliente); // adição do cliente à venda
}

```

Figura 20 – Criação do documento Cliente embebido ao documento Venda

```

SELECT F.id, F.nome_completo
FROM Venda AS V
INNER JOIN Funcionario AS F
ON F.id = V.id_funcionario
WHERE V.id = ?

```

Figura 21 – Query usada no código abaixo

```

PreparedStatement ps3 = this.con.prepareStatement( sql: ""+"SELECT F.id, F.nome
ps3.setInt( parameterIndex: 1,id);
ResultSet rs3 = ps3.executeQuery();
while (rs3.next()){
    id_funcionario = rs3.getInt( columnLabel: "F.id");
    nome_completo_funcionario = rs3.getString( columnLabel: "F.nome_completo");

    BasicDBObject documentoFuncionario = new BasicDBObject();

    documentoFuncionario.put("id_funcionario", id_funcionario);
    documentoFuncionario.put("nome_funcionario", nome_completo_funcionario);

    document.put("Funcionario", documentoFuncionario);
}

```

Figura 22 – Criação do documento Funcionário embebido ao documento Venda

Após a adição destes dois documentos é necessário adicionar outros dois documentos embebidos ao documento Venda, que neste caso só um dos dois ocorre, pois, a venda ou é referente a um serviço ou a um conjunto de produtos.

Deste modo, criou-se uma flag (seta vermelha, imagem abaixo) de controlo, para verificar a que se refere a Venda. Assim caso a Venda seja de produtos, faz-se o seguinte código e a seguinte query:

```
SELECT P.id, P.designacao, VP.quantidade,
       VP.preco_unitario_final, VP.preco_total,
       VP.desconto_unitario, VP.desconto_total
FROM Produto AS P
INNER JOIN VendaProduto AS VP
ON P.id = VP.id_produto
INNER JOIN Venda AS V
ON VP.id_venda = V.id
WHERE V.id = ?
```

Figura 23 – Query usada no código abaixo

```
PreparedStatement ps4 = this.con.prepareStatement( sql: ""+SELECT P.id, P.designacao, \
ps4.setInt( parameterIndex: 1,id);

ResultSet rs4 = ps4.executeQuery();

BasicDBList lista_produtos = new BasicDBList();

int flag = 0;

while(rs4.next()){
    flag++; // para garantir que não adiciona produtos caso não haja;
    id_produto = rs4.getInt( columnLabel: "P.id");
    designacao_produto = rs4.getString( columnLabel: "P.designacao");
    quantidade_produto = rs4.getInt( columnLabel: "VP.quantidade");
    preco_unitario_final_produto = rs4.getFloat( columnLabel: "VP.preco_unitario_final");
    preco_total_produto = rs4.getFloat( columnLabel: "VP.preco_total");
    desconto_unitario_produto = rs4.getFloat( columnLabel: "VP.desconto_unitario");
    desconto_total_produto = rs4.getFloat( columnLabel: "VP.desconto_total");

    BasicDBObject documento_produto = new BasicDBObject();

    documento_produto.put("id_produto", id_produto);
    documento_produto.put("designacao", designacao_produto);
    documento_produto.put("quantidade", quantidade_produto);
    documento_produto.put("preco_unitario_final", preco_unitario_final_produto);
    documento_produto.put("preco_total", preco_total_produto);
    documento_produto.put("desconto_unitario", desconto_unitario_produto);
    documento_produto.put("desconto_total", desconto_total_produto);

    lista_produtos.add(documento_produto); // lista dos produtos
}

if(flag > 0)document.put("Produtos", lista_produtos); // adição dos produtos
```

Figura 24 – criação do documento produto e a respetiva lista de produtos

Por outro lado, se a Venda se refere a um serviço, então faz-se a seguinte query e código e lembrando, como se trata do último documento a adicionar ao documento Venda.

Por fim, adiciona-se o documento Venda à coleção (seta azul) e repete-se este processo para todas as vendas da base de dados SQL:

```

SELECT S.id, S.data_inicio, S.data_fim
FROM Servico AS S
INNER JOIN Venda AS V
ON S.id_venda = V.id
WHERE V.id = ?

```

Figura 25 – Query usada no código abaixo

```

PreparedStatement ps5 = this.con.prepareStatement( sql: "SEL

ps5.setInt( parameterIndex: 1,id);

ResultSet rs5 = ps5.executeQuery();

while(rs5.next()){
    id_servico = rs5.getInt( columnLabel: "S.id");
    data_inicio = rs5.getDate( columnLabel: "S.data_inicio");
    data_fim = rs5.getDate( columnLabel: "S.data_fim");

    BasicDBObject documentoServico = new BasicDBObject();

    documentoServico.put("id_servico", id_servico);
    documentoServico.put("data_inicio", data_inicio);
    documentoServico.put("data_fim",data_fim);

    document.put("Servico",documentoServico);
}

collection.insertOne(document);

```




Figura 26 – Criação do documento Serviço embebido ao documento Venda e  
adição do documento Venda à coleção

Decidiu-se apresentar em *NoSQL* as *queries* que consideramos mais importantes para o nosso sistema.

----> **Query 3 - N produtos com menos stock**  
db.getCollection('Produto').find({},{"\_id":1, "designacao":1,  
"stock":1}).sort({"stock": 1}).limit(N)

----> **Query 6 - Número de serviços realizados por um funcionário num intervalo de datas**  
db.Venda.aggregate([  
{\$match : {Servico: { \$exists: true }, data:{\$gte: ISODate("2000-01-01T00:00:00.000Z"),\$lt: ISODate("2018-12-31T00:00:00.000Z")}} },

```

    { $group: { "_id": {idFuncionario:"$Funcionario.id_funcionario"}, "Nome
do funcionario": { "$first": "$Funcionario.nome_funcionario"}, "Numero
de servicos feitos": { $sum: 1 } } },
  ])
---> Query 9 - Classificação geral
db.Cliente.aggregate([
  { $group: { _id: null, avg: { $avg: "$classificacao" } } }
])
----> Query 11 - Valor total em salários
db.Funcionario.aggregate([
  { $group: { "_id": "", "Valor total em salarios": { $sum: "$salario" }, },
])
----> Query 12 - Procurar cliente por telefone
db.getCollection('Cliente').find({"Contacto.Telefones.nr_telefone":NumeroTelefo
ne},{_id:0})
-----> Query 13 – Produtos comprados por cada cliente por ordem
crescente de preço
db.Venda.aggregate([
{$unwind:"$Produtos"},
{$match:{$and:[{"Cliente.nif": 748637829},{ "Produtos":{" $exists: true }}}}},
{$sort:{"Produtos.preco_total":1}},
{$project: {idProduto:"$Produtos"}}])
---> Query 14 – Apresentar Informação dos produtos comprados
var idArg = 1;
db.Venda.aggregate([
  { $match : { id:idArg } },
  { $lookup:
    {
      from: "Produto",
      localField : "Produtos.id_produto",
      foreignField : "id",
      as : "Produtos comprados"
    }
  },
  { $project: { "Produtos comprados":1 } }
])

```

Tabela 1 - Ilustração de inserção de uma tabela e sua legenda.

## Conclusões e Trabalho Futuro

A adaptação ao modelo de dados não relacional MongoDB exigiu recorrência a diversas pesquisas, visto que não estávamos tão familiarizados quanto em MySQL.

No que concerne à definição da estrutura para o sistema de base de dados NoSQL, esta envolveu uma análise crítica de tempo significativo, o que causou, de certa forma, um pequeno atraso na inicialização da migração.

A migração dos dados do SQL para NoSQL teve inicialmente alguns problemas, pois não se conseguiu encontrar imediatamente as drives corretas para a conexão com a base de dados do MongoDB. Importante referir também, que para a migração dos dados, foram utilizadas todas as tabelas da base de dados relacional.

Relativamente à representação das transações realizadas inicialmente em MySQL para NoSQL, estas não foram implementadas, visto que o *software* que tiver a incumbência de “controlar o *MongoDB*” *necessita* de estar preparado para o implementar.



## Referências

<https://docs.mongodb.com/>

<https://dev.mysql.com/doc/>

## **Lista de Siglas e Acrónimos**

<b>BD</b>	Base de Dados
<b>NoSQL</b>	Not only Structed Query Language

