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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2018/2019

Trabalho Prático

Grupo 49

Tiago Baptista, João Leal, Fábio Silva

Novembro, 2018

Data de Recepção	
Responsável	
Avaliação	
Observações	

LEIParking

Tiago Baptista, João Leal, Fábio Silva

Janeiro, 2019

Índice

1.Índice de Figuras

2. Resumo

Numa parte anterior deste projeto foi solicitada a elaboração de um modelo de base de dados relacional de acordo com as especificações do utilizador, neste caso, uma empresa de *Parking*. Nessa parte do projeto, utilizou-se a linguagem SQL.

Nesta segunda parte do projeto, realizou-se a análise, descrição e migração do modelo anterior para uma base de dados NoSQL. Neste processo, fez-se também o povoamento da base de dados e desenvolveram-se várias *queries* que para além de verificarem se a base de dados está de acordo com os requisitos do utilizador, servirão para se consulta de informação da base de dados criada.

Ao longo deste relatório, são mostradas algumas diferenças entre SQL e NoSQL e é feita uma descrição de MongoDB.

Área de aplicação: Desenho e arquitetura de Sistemas de Bases de Dados

Palavras-chave: Relacionamentos, NoSQL, SQL, MongoDB, migração, cliente, reserva, parque

3. Conceito de MongoDb

O MongoDB é a ferramenta que se utiliza no presente trabalho, é um tipo de base de dados orientado a documentos. É o mais famoso NoSQL do mercado open source e é escrito em C++. Este consegue providenciar uma boa performance na escalaridade, no que toca a ser necessário apenas um nó de base de dados ou dezenas deles. Além disso, é um modelo de dados intuitivo que consegue lidar com grandes estruturas de dados e permite ao administrador guardar os seus registos sem ter a preocupação de organizar estruturas de dados como o número de campos ou tipos de campos para armazenar valores.

Esta ferramenta guarda os dados em documentos que são alterados criando ou eliminando campos. Isto é útil para representar relações hierárquicas assim como para armazenar matrizes e outras estruturas.

4. Comandos Mongo utilizados

Ao longo desta parte do trabalho, foram utilizados vários comandos úteis para resolução de *queries* ou migração de dados, para a nova base de dados. Os comandos que utilizados são os abaixo descritos:

- **find()**

Este comando seleciona documentos numa coleção ou view e retorna um resultado para os documentos selecionados.

```
db.ReservaCollection.find({"Pagamento.multa": {$gt: 0}}).pretty()
```

Figura 1

Neste exemplo , do projeto realizado utilizou-se o comando find({“Pagamento.multa”: {$gt: 0}}) para encontrar na coleção Reserva os pagamentos onde a multa foi maior que 0.

- **pretty()**

Este comando faz com que o resultado do comando find() seja legível para o utilizador. U este comando a seguir ao comando find().

Figura 2

No mesmo exemplo, da Figura 2, encontra-se o comando *pretty* para que o resultado apareça formatado.

- **aggregate()**

Este comando agrega valores numa coleção ou view. O aggregate() pode ter dentro dele match, group, sort, project, collation, hint, entre outros.

Figura 3

Neste exemplo, utilizou-se o comando aggregate para agregar valores sem os juntar por grupos.

- **$project**

Este comando passa os documentos com os campos solicitados para o próximo estágio no pipeline. Os campos especificados podem ser campos existentes dos documentos de entrada ou campos recém-calculados. $project pega num documento que pode especificar a inclusão de campos, a supressão do campo _id, a adição de novos campos e a redefinição dos valores dos campos existentes.

Figura 4

No exemplo, usa-se $project para passar a informação dentro do comando para o próximo estágio.

- **$filter**

Este comando seleciona um subconjunto de um array e retorna um array com apenas os elementos que correspondem à condição especificada. Além disso, os elementos retornados estão na ordem original.

Figura 5

No exemplo acima, usou-se $filter para selecionar os lugares livres e retornar um *array* com todos os lugares livres.

- **$match**

Este comando filtra os documentos para transmitir somente os documentos que correspondem às condições especificadas para o próximo estágio do pipeline. $match recebe um documento que especifica as condições da consulta.

```
db.ParqueCollection.aggregate([{$match:{distrito:"Braga"}},{$project:{Lugares:{$filter:{input: "$Lugares",as: "lugar", cond: {$eq: ["$$lugar.ocupado",0]}}}}}]).pretty()
```

Figura 6

Neste exemplo, usou-se $match para restringir a nossa procura ao distrito de Braga.

- **$sum**

Este comando calcula e retorna a soma de valores numéricos. $sum ignora valores não numéricos.

```
db.ReservaCollection.aggregate([{$group:{_id:{}, Amount:{$sum: {$add: ["$Pagamento.preco","$Pagamento.multa"]}}}}])
```

Figura 7

No exemplo, utilizou-se $sum para somar os valores dos preços mais as multas.

- **$add**

Este comando adiciona números ou um número e uma data. Se um dos argumentos for uma data, $add trata os outros argumentos como milissegundos para adicionar à data.

```
db.ReservaCollection.aggregate([{$group:{_id:{}, Amount:{$sum: {$add: ["$Pagamento.preco","$Pagamento.multa"]}}}}])
```

Figura 8

No mesmo exemplo anteriormente referido, neste caso, utilizou-se $add para adicionar os valores do preço e da multa.

5. Razões para a utilização de NoSQL no caso de Estudo

Antes da segunda parte deste projecto analisaram-se as várias diferenças entre uma base de dados SQL e uma NoSQL.

Uma base de dados NoSQL, além de não ter interface SQL, é open source, pois estas bases de dados foram projetadas para satisfazer as necessidades que as bases de dados relacionais não satisfazem, como alta performance e capacidade de expansão. Existem 4 tipos diferentes de bases de dados NoSQL:

- Chave-Valor
- Grafos
- Colunas
- Documentos

O Neo4j trabalha à base de gráficos e o MongoDB trabalha à base de documentos.

MongoDB tem várias características. Algumas delas são:

- Alta performance e escalabilidade;
- Replicação (ferramenta que possibilita o armazenamento de dados e backups independentemente do local físico onde os dados estão armazenados);
- API simples para acesso aos dados;
- Raízes Open Source;
- Armazenamento de dados estruturados ou não estruturados (permite uma fácil aplicação da escalabilidade e um aumento na disponibilidade de dados);

Existem várias diferenças entre uma base de dados SQL e NoSQL a vários níveis:

- **Armazenamento de dados**

  SQL: armazenamento num modelo relacional, em tabelas.

  NoSQL: abrange uma série de dados, cada uma com a sua estrutura de armazenamento de dados.

- **Estrutura e flexibilidade**

  SQL: alterações feitas implicam a alteração de toda a base de dados.

  NoSQL: como as estruturas são dinâmicas, as informações podem ser adicionadas com facilidade e permitem haver entradas em colunas da tabela vazias.

- **Escalabilidade**

SQL: escalabilidade vertical, o que significa que com grandes quantidades de dados torna-se difícil e demorado o processo de escalar uma base de dados .

NoSQL: escalabilidade horizontal, o que se torna mais económico e permite ter um bom desempenho .

- **Propriedades ACID**

SQL: bancos de dados relacionais compatíveis com ACID.

NoSQL: varia de acordo com as tecnologias, mas, em alguns casos, a compatibilidade ACID é sacrificada para desempenho e escalaridade.

Com isto, reuniram-se todas as opiniões dos intervenientes no projeto com o intuito de avaliar as vantagens assim como as desvantagens da utilização de uma base de dados NoSQL chegando-se à conclusão que o mais adequado seria fazer uma base de dados NoSQL, uma vez que:

- Os dados estão disponíveis;
- O custo é mais reduzido comparadamente a uma base de dados relacional;
- É uma base de dados orientada a objetos flexíveis;
- É mais fácil de inserir novos dados;
- É uma boa solução para lidar com o problema de dados em massa, que é um dos problemas que poderiam surgir;

Dentro deste tipo de bases de dados, escolheu-se o MongoDB, pois este é um tipo de base de dados que não fornece relacionamentos restritos entre documentos, o que não acontece nas bases de dados relacionais, que estão dependentes de relacionamentos para normalizar o armazenamento de dados. Em vez de armazenar dados relacionados com uma área de armazenamento separada, em bases de dados de documentos, estes são integrados ao próprio documento. Isto permite ganhar bastante em desempenho. Além disto, MongoDB tem a caraterística de Sharding que será devidamente enunciada em seguida.

5.1 Sharding

Uma base de dados não relacional MongoDB é uma base de dados escalável. Isto deve-se ao facto do MongoDB apresentar *Sharding.* Este é um método para distribuir dados em várias máquinas. Sistemas de banco de dados com grandes conjuntos de dados ou aplicativos de alto rendimento podem desafiar a capacidade de um único servidor. Assim, o *Sharding* é uma solução para esse problema, pois assim o fluxo de dados é distribuído por várias máquinas, não havendo o risco de sobrecarga de um servidor. Apesar disto, este método só é evocado quando uma máquina não consegue suportar o armazenamento de

quantidades cada vez maiores de dados nem consegue efetuar operações *read/write* em tempo aceitável.

5.2 Mapeamento SQL-MongoDB

Base de Dados Relacional SQL	**Modelo NoSQL (MongoDB)**
Base de Dados	Base de Dados
Tabela	Coleção
Linha/Tuplo	Documento
Coluna	Campo
Índice	Índice
Join (Junção de Tabelas)	Documento Embebido
Chave Primária	Chave "_id" fornecida pelo próprio MongoDB *(por defeito)*
Servidor e Cliente (Base de Dados)	
Servidor: mysqld	**Servidor: mongod**
Cliente: mysql	**Cliente: mongo**

6. Processo de Migração

6.1 Modelação de dados

O mongoDb agrupa documentos em coleções, tal como referido anteriormente, não impondo um esquema restrito, sendo portanto *schema less.*Na realidade traduz-se na existência de documentos numa mesma coleção com estruturas diferentes e com os campos comuns entre eles, contendo diferentes tipos de dados.

Por outro lado no modelo relacional verificou-se que isto não ocorre, sendo que todos os registos que figuram numa mesma relação deverão ter a mesma estrutura e conter o mesmo tipo de dados.

O facto de o *mongo* não ter a imposição de uma definição rígida a nível da estrutura permite - lhe alguns aspectos vantajosos, entre os quais se destacam:

1. a estrutura dos dados ser determinada pelo código da aplicação que se suporta na base de dados, e não pela base de dados em si .
2. a fase inicial do desenvolvimento de uma aplicação que se suporta numa base de dados *mongoDB* é mais rápida, pois numa fase inicial ocorrem,tendencialmente,algumas alterações nos esquemas das bases de dados, sendo que o mongoDB não tem qualquer problema devido à característica *schema less.*
3. os esquemas flexíveis para os dados permitem modelá-los verdadeiramente sem necessidade de efetuar adaptações para encaixá-los num modelo mais rígido.

Tendo em conta o referido em cima começou-se por tentar identificar as coleções necessárias para a implementação. Para tal teve-se em conta que o esquema do modelo segue-se pelo esquema do modelo em SQL e que devia continuar a satisfazer os requisitos funcionais.De seguida identificou-se objetos que foram usados em conjunto, de maneira a incluí-los no mesmo documento de maneira a evitar possíveis operações equivalentes a join.Por fim após pesquisa, verificou-se que era aceitável duplicação de dados mas decidiu-se ao máximo evitá-la.

6.2 Esquematização das Coleções

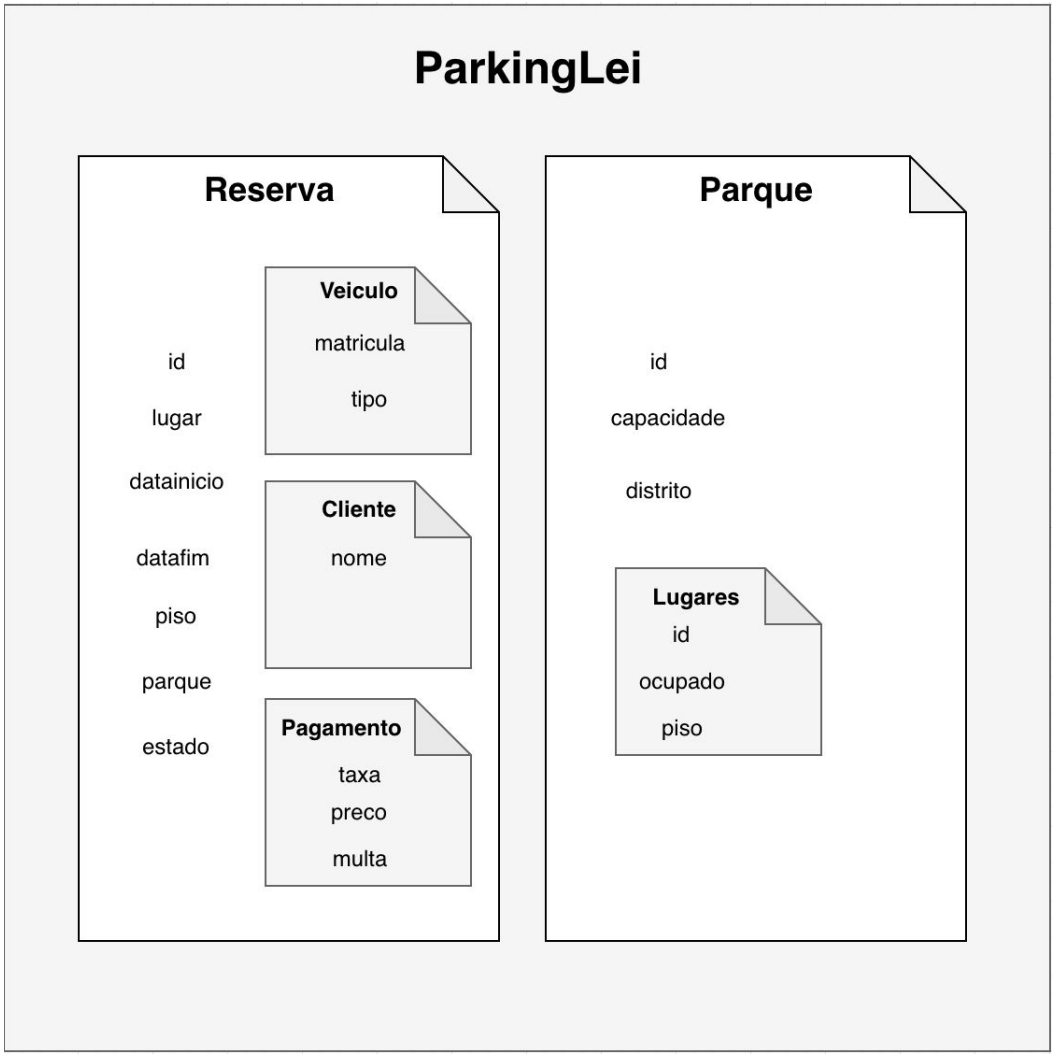

Figura 9

Analisando as tabelas existentes no modelo relacional SQl e comprando-as com as coleções criadas para o mongoDB denota-se claramente que,segundo, o segundo modelo os dados se encontram mais agrupados. Isto deve-se ao facto de em mongo não existirem joins, pelo menos explicitamente, o que leva à necessidade de manter alguns dados juntos.

Após análise detalhada, chegamos ao esquema apresentado acima, decidiu-se por dividir em 2 coleções : a Reserva e o Parque.

Verificou-se, após novo olhar sobre os requisitos do cliente, que as operações realizadas sobre as reservas necessitavam de informação sobre o cliente, os seus veículos e pagamentos efetuados pelas Reservas, razão pela qual incluímos informações sobre todos estes na coleção Reserva.

Quanto à coleção Parque, garantiu -se maior facilidade para aceder informações relativas a um Parque.

6.2.1 Coleção Reserva

Esta coleção visa representar toda a informação necessária para cada Reserva e contém os seguintes campos:

- **id** : identificador da reserva;
- **lugar** : indica o lugar da reserva;
- **data inicio** : indica a data de início da reserva;
- **data fim** : indica a data de fim da reserva;
- **piso**:indica o piso de uma reserva;
- **parque** : indica o parque na qual a reserva é afeta;
- **estado**: indica o estado da reserva, ativa ou inativa;
- **Veiculo** : indica todas as informações relativas ao veículo para o qual foi efetuado a reserva;
- **Cliente** : indica todas as informações relativas ao cliente para o qual foi efetuado a reserva;
- **Pagamento** : indica todas as informações sobre o pagamento;

6.2.2 Coleção Parque

Esta coleção visa representar toda a informação necessária para cada Parque e contém os seguintes campos:

- **id:**identificador da reserva;
- **capacidade** : indica a capacidade do parque;
- **Lugares:** possui todas as informações sobre os lugares afetos a um parque;

6.3 Script de Migração

Para realizar a transição de *MySQL* para MongoDB decidiu-se usar um script escrito em Python, que permitiu a migração dos dados que se encontravam na base de dados relacional para a nova em MongoDB. Existe necessidade de realizar o processo desta maneira pois permite agilizar o processo de transição que de outra maneira seria efetuada de forma manual e custosa a nível de tempo.

A ligação à base de dados relacional efetua-se através do *mysqlconnetor* enquanto que a ligação ao mongo é efetuada pelo *pymongo*. De seguida realizamos as *queries* necessárias para obter a informação, organizamos os dados nos documentos que são exportados em formato JSON e posteriormente automaticamente carregados para o endereço da base de dados pretendido.

Para corre o script basta correr o ficheiro python e a migração é efetuada automaticamente.

```python
import mysql
import mysql.connector as con
from mysql.connector import errorcode
from pymongo import MongoClient
from bson.decimal128 import Decimal128
# -*- coding: utf-8 -*-

def connectToSQL():
    try:
        connection = con.connect(user="root", database="parking",password="@Tiagob12")
        print("Ligado a parking")
        return connection
    except mysql.connector.Error as err:
        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Username ou Password errados")
            return

        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("Erro na Base de Dados\n")
            return
        else:
            print(err)
            return

def connectToMongo():
    client = MongoClient("localhost", 27017)
    mongodb = client["parking"]

    if "parking" in client.list_database_names():
        print("Base de Dados Mongo ja existia, dropped para nova criacao\n")
        # SE JA EXISTIA A BD AO EXECUTAR O SCRIPT, FAZ DROP DELA
        client.drop_database("parking")
    return mongodb



def insertReservas(mysqld, mongodb):
    cursor = mysqld.cursor()
    MongoDoc = getReservas(cursor)
    InsertResult = mongodb.reserva.insert_many(MongoDoc)
    print("     Id's inseridos: ", InsertResult.inserted_ids, "\n")


def getReservas(cursor):
    reservaQuery = "SELECT R.idReserva, R.data_inicio, R.data_fim, R.ativa,P.preco,P.taxa, P.multa, L.id_lugar,L.piso,L.Parque_idParque, C.nome, V.matricula,V.tipo " \
                   "FROM Reserva AS R " \
                   "INNER JOIN LugaresParque AS L ON R.LugaresParque_lugar = L.id_lugar " \
                   "INNER JOIN Pagamento AS P ON R.Pagamento_idPagamento = P.idPagamento " \
                   "INNER JOIN Veiculo AS V ON R.Veiculo_matricula = V.matricula " \
                   "INNER JOIN Cliente AS C ON V.Cliente_idCliente = C.idCliente "
```

Figura 10

```
    cursor.execute(reservaQuery)
    reservaRes = cursor.fetchall()

    resList = []
    numberReservas = 0

    for reserva in reservaRes:
        ID = reserva[0]
        Inicio = reserva[1]
        Fim = reserva[2]
        Estado = reserva[3]
        Preco = reserva[4]
        Taxa = reserva[5]
        Multa = reserva[6]
        Lugar = reserva[7]
        Piso = reserva[8]
        Parque = reserva[9]
        Cliente = reserva[10]
        Matricula = reserva[11]
        Tipo = reserva[12]

        ResDoc = {"_id": ID,
                  "datainicio": Inicio,
                  "datafim": Fim,
                  "estado": Estado,
                  "lugar": Lugar,
                  "piso": Piso,
                  "parque": Parque,
                  "Cliente": {"nome":Cliente},
                  "Veiculo":{"matricula":Matricula,"tipo":Tipo},
                  "Pagamento":{"preco":Preco,"multa":Multa,"taxa":Taxa}
                  }
        resList.append(ResDoc)

    return resList


def insertParques(mysqld, mongodb):
    cursor = mysqld.cursor()
    MongoDoc = getParques(cursor)
    InsertResult = mongodb.parque.insert_many(MongoDoc)
    print("     Id's inseridos: ", InsertResult.inserted_ids, "\n")


def getParques(cursor):
    parqueQuery = "SELECT P.idParque, P.distrito, P.cidade, P.capacidade " \
                  "FROM Parque AS P "


    cursor.execute(parqueQuery)
    parqueRes = cursor.fetchall()

    parqueList = []
```

Figura 11

```
    parqueList = []
    numberParques = 0

    for parque in parqueRes:
        ID = parque[0]
        Distrito = parque[1]
        Cidade = parque[2]
        Capacidade = parque[3]
        Lugares = getLugares(cursor,ID)


        ResDoc = {"id": ID,
                  "distrito": Distrito,
                  "cidade": Cidade,
                  "capacidade": Capacidade,
                  "Lugares": Lugares}

        parqueList.append(ResDoc)

    return parqueList

def getLugares(cursor,Parqueid):
    lug = "SELECT L.id_lugar, L.piso, L.ocupado " \
          "FROM LugaresParque AS L " \
          "WHERE L.Parque_idParque="+str(Parqueid)

    cursor.execute(lug)
    lugares = cursor.fetchall()

    lugList = []

    for lugar in lugares:
        Lugar = lugar[0]
        Piso = lugar[1]
        Ocupado = lugar[2]

        lugaresDoc = {"ID_Lugar": Lugar,
                      "piso": Piso,
                      "ocupado": Ocupado
                      }

        lugList.append(lugaresDoc)


    return lugList

def main():
    # CONECTAR BASE DE DADOS SQL
    mysqld = connectToSQL()

    # CONECTAR BASE DE DADOS MONGO
    mongodb = connectToMongo()

    # ADICIONAR RESERVAS A COLECAO EM MONGO
    insertReservas(mysqld, mongodb)

    # ADICIONAR PARQUES A COLECAO EM MONGO
    insertParques(mysqld,mongodb)
```

Figura 12

De referir que é necessário ter ambos os motores de base de dados *(MySQL* e *MongoDB*) a correr antes de executar o *scipt*. Para a execução do script é necessário a instalação de uma versão do *python* e o *mysqlconnector* do *python*.De seguida passa-se a mostrar como executar o script de povoamento.

```
MBP-de-Tiago:Desktop macz$ python sqlmongo.py
Ligado a parking
Base de Dados Mongo ja existia, dropped para nova criacao

("     Id's inseridos: ", [1, 15, 16, 2, 18, 3, 4, 5, 7, 8, 17, 9, 10, 11, 12, 13, 14, 6], '\n')
("     Id's inseridos: ", [ObjectId('5c3e513b45e339741c30aceb'), ObjectId('5c3e513b45e339741c30acec'), ObjectId('5c3e513b45e339741c30aced')], '\n')
```

Figura 13

De seguida apresentam-se um exemplo de um documento de cada coleção da base de dados.

```
> db.reserva.find().pretty()
{
        "_id" : 1,
        "lugar" : 1,
        "datafim" : ISODate("2018-11-25T18:30:00Z"),
        "datainicio" : ISODate("2018-11-25T18:20:00Z"),
        "piso" : 0,
        "Veiculo" : {
                "matricula" : "03-03-LD",
                "tipo" : 1
        },
        "parque" : 1,
        "estado" : 1,
        "Cliente" : {
                "nome" : "Marcela Lima"
        },
        "Pagamento" : {
                "taxa" : 0.2,
                "multa" : 0,
                "preco" : 0.5
        }
```

Figura 14

Figura 15

De seguida apresenta-se um comando de verificação da base de dados criada.

Figura 16

7. Backup

Um *backup* de uma base de dados em **mongoDB** pode ser feito através do comando ***mondogump*****,** que se traduz, primeiramente, numa conexão ao servidor e, posteriormente, numa descarga de todos os dados dentro desse servidor (*MongoDatabase)* para a diretoria específica */bin/dump*. Este comando é executado na *shell***,** suportando também as especificações de opções que o utilizador deseje impor.

```
➜ Desktop mongodump
2019-01-15T21:52:36.624+0000    writing admin.system.version to
2019-01-15T21:52:36.626+0000    done dumping admin.system.version (1 document)
2019-01-15T21:52:36.626+0000    writing parking.ReservaCollection to
2019-01-15T21:52:36.626+0000    writing parking.ParqueCollection to
2019-01-15T21:52:36.628+0000    done dumping parking.ReservaCollection (18 documents)
2019-01-15T21:52:36.628+0000    done dumping parking.ParqueCollection (3 documents)
```

Figura 17

Em contrapartida, o utilizador pode restaurar a base de dados encontrados na diretoria previamente referida, através do comando *mongorestore*.

```
➜ Desktop mongorestore
2019-01-15T21:52:47.007+0000    using default 'dump' directory
2019-01-15T21:52:47.007+0000    preparing collections to restore from
2019-01-15T21:52:47.008+0000    reading metadata for parking.ReservaCollection from dump/parking/ReservaCollection.metadata.json
2019-01-15T21:52:47.009+0000    reading metadata for parking.ParqueCollection from dump/parking/ParqueCollection.metadata.json
2019-01-15T21:52:47.103+0000    restoring parking.ParqueCollection from dump/parking/ParqueCollection.bson
2019-01-15T21:52:47.190+0000    restoring parking.ReservaCollection from dump/parking/ReservaCollection.bson
2019-01-15T21:52:47.192+0000    no indexes to restore
2019-01-15T21:52:47.192+0000    finished restoring parking.ParqueCollection (3 documents)
2019-01-15T21:52:47.194+0000    no indexes to restore
2019-01-15T21:52:47.194+0000    finished restoring parking.ReservaCollection (18 documents)
2019-01-15T21:52:47.194+0000    done
```

Figura 18

8. Validação com o cliente

Tal como na primeira parte deste projecto, foi necessária a validação do modelo com perguntas ao cliente,em prol de garantir que a fiabilidade da base de dados de forma a que esta cumpra os requisitos exigidos. Assim sendo, após essa validação, é necessário a certificação da mesma, agora num modelo não relacional. Sendo assim, delineamos as *queries* que se seguem, com a sua respetiva codificação, juntamente com o resultado obtido:

→ Query 01: Nomes dos Clientes e respectivas Matrículas de Veículos, que usaram os Parques da empresa, entre duas Datas

```
> use parking
switched to db parking
> db.ReservaCollection.find({"data inicio": {$gte: ISODate("2018-11-26")},"data fim":{$lt: ISODate("2018-11-27")}},{Veiculo:1,Cliente:1}).pretty()
{
        "_id" : 4,
        "Cliente" : {
                "nome" : "Tiago Tombado"
        },
        "Veiculo" : {
                "matricula" : "11-46-QJ",
                "tipo" : 1
        }
}
{
        "_id" : 5,
        "Cliente" : {
                "nome" : "Ricardo Furacão"
        },
        "Veiculo" : {
                "matricula" : "16-16-FC",
                "tipo" : 1
        }
}
{
        "_id" : 6,
        "Cliente" : {
                "nome" : "Bruno Ferrero"
        },
        "Veiculo" : {
                "matricula" : "47-41-IP",
                "tipo" : 1
        }
}
```

Figura 19

→ Query 02: Reservas com multa

```
> use parking
switched to db parking
> db.ReservaCollection.find({"Pagamento.multa": {$gt: 0}}).pretty()
{
        "_id" : 6,
        "data inicio" : ISODate("2018-11-26T01:30:00Z"),
        "data fim" : ISODate("2018-11-26T02:35:00Z"),
        "estado" : 0,
        "lugar" : 13,
        "piso" : 3,
        "parque" : 1,
        "Cliente" : {
                "nome" : "Bruno Ferrero"
        },
        "Veiculo" : {
                "matricula" : "47-41-IP",
                "tipo" : 1
        },
        "Pagamento" : {
                "preco" : 11.4,
                "multa" : 2,
                "taxa" : 0.2
        }
}
```

Figura 20

→ Query 03: Lucro total da Empresa (Preço + Multa)

```
> use parking
switched to db parking
>
> db.ReservaCollection.aggregate([{$group:{_id:{}, Amount:{$sum: {$add: ["$Pagamento.preco","$Pagamento.multa"]}}}}])
{ "_id" : {  }, "Amount" : 61.15 }
```

Figura 21

→ Query 04: Lugares livres de um Parque numa dada Cidade (“Braga”)

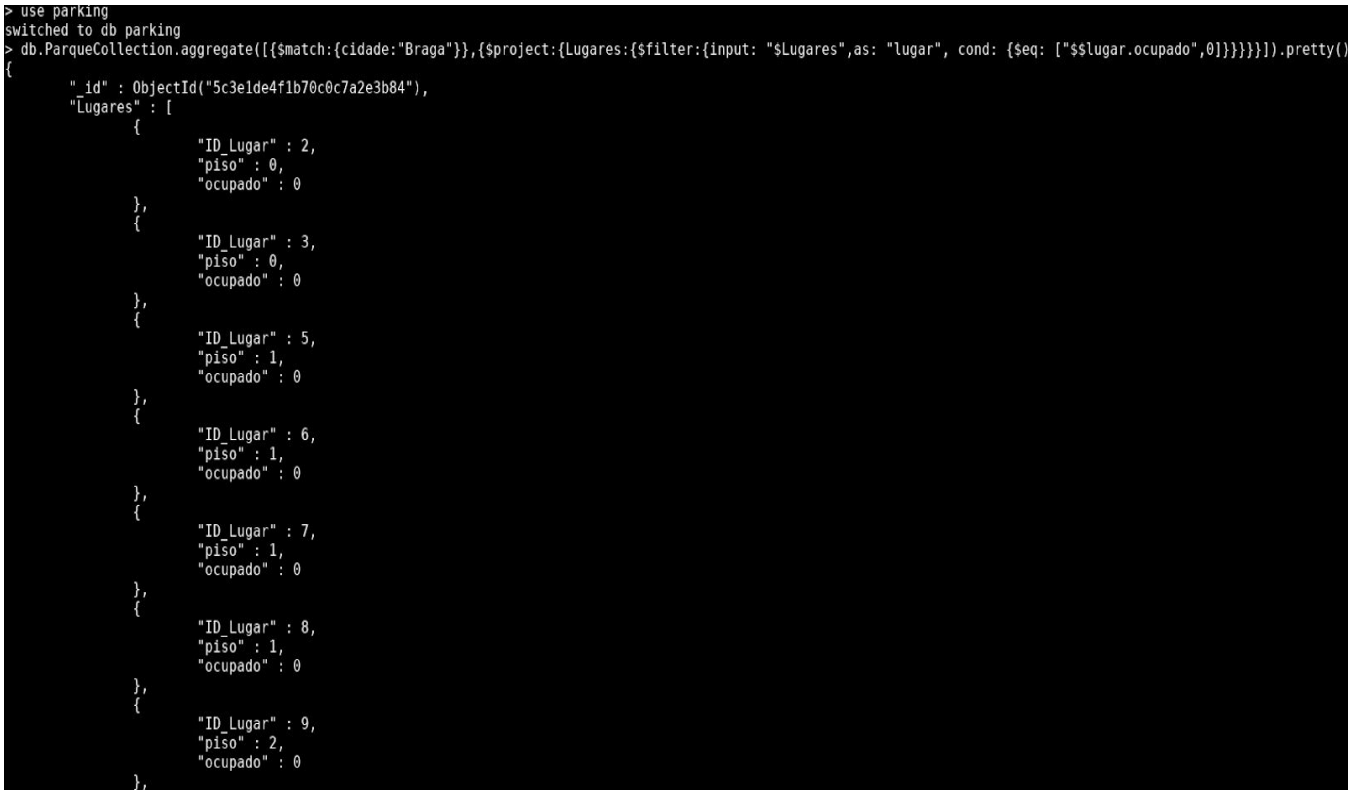

Figura 22

9.Conclusão

Após a conclusão do projeto acabámos com duas bases de dados implementadas,funcionais e a cumprir os requisitos traçados pelo cliente na fase inicial. Uma com base relacional em *MySQ*L e outra *NoSQL* em *MongoDB.* Comparativamente com a primeira fase, notou-se maior facilidade e mais fácil implementação da base de dados não relacional.Conclui-se também que o paradigma NoSQL acaba por ser mais flexível e facilitar bastante no desenvolvimento de uma aplicação.

A partir deste ponto fica ainda trabalho por fazer, a monitorização da base de dados, do seu crescimento assim como garantir backups frequentes de maneira a manter a integridade dos dados armazenados.

10. Bibliografia

1. https://docs.mongodb.com/manual/# , Consultado em: 09/01/2019
2. https://www.w3schools.com/python/python_mongodb_getstarted.asp, Consultado em : 10/01/2019