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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Reservas de viagens em comboios nacionais e internacionais

A75353 Dinis Peixoto

A74185 Ricardo Pereira

A75210 Marcelo Lima

A67638 Carlos Faria

Janeiro, 2017

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Reservas de viagens em comboios nacionais e internacionais

A75353 Dinis Peixoto

A74185 Ricardo Pereira

A75210 Marcelo Lima

A67638 Carlos Faria

Janeiro, 2016

Resumo

Este relatório é o resultado do segundo trabalho elaborado no âmbito da Unidade Curricular de Base de Dados, na qual nos foi sugerida a criação de uma base de dados em MongoDB (NoSQL) com o tema "Reservas de viagens em comboios nacionais e internacionais", sendo este o resultado da migração feita a partir da base de dados relacional anteriormente criada. Ao longo deste relatório serão apresentados todos os passos que tivemos de efetuar durante este processo de migração.

Inicialmente é apresentada uma breve introdução juntamente com uma pequena contextualização do nosso projeto, o caso de estudo em questão bem como a motivação do projeto e os seus principais objetivos.

De seguida, é feita uma pequena introdução àquele que é o novo paradigma dos Sistemas de Base de Dados, o NoSQL. Aqui é debatido o porquê do surgimento deste tipo de Base de dados, apontando os seus pontos fortes relativamente aos Sistemas de Base de Dados Relacionais. Para finalizar este capítulo, é apresentada uma breve descrição sobre o MongoDB, o modelo de dados orientado a documentos mais popular da atualidade, apresentando as suas respetivas vantagens e desvantagens.

Logo após, é abordado todo o processo de migração que nos levou à implementação da Base de Dados. Inicialmente, de forma a nos podermos enquadrar mais facilmente, é apresentado o esquema do modelo lógico da base de dados que será algo da respetiva migração. De seguida, passando para a parte mais prática, é apresentado o esquema das três coleções de documentos que decidimos implementar, assim como a descrição da respetiva estrutura de cada documento utilizado. Para terminar, é explicado como foi feita esta migração de dados e estrutura de MySQL para MongoDB. São também colocados em anexo três exemplos de *queries* realizadas em MongoDB sobre a base de dados criada, com a respetiva correspondência em MySQL.

Por último, são feitas as conclusões do trabalho realizado, apontando os seus pontos fortes e fracos relativamente ao projeto anterior.

Área de Aplicação: Desenho e Arquitetura de Sistemas de Base de Dados no âmbito de uma aplicação responsável por reservas para viagens de comboio em toda a Europa.

Palavras-Chave: Bases de Dados, Bases de Dados Relacionais, Sistemas de Gestão de Base de Dados, SQL, NoSQL, MongoDB

Índice

Índice de Figuras

1. Introdução

1.1. Contextualização

A *Euro-Train* é uma empresa que se encarrega por prestar serviços de transporte ferroviário de passageiros em toda a Europa. A empresa foi fundada em 1885 com o nome *Iberian-Train* e tinha como objetivo assegurar a ligação entre as principais cidades de toda a Península-Ibérica. A sua preferência em relação às empresas concorrentes permitiu que esta crescesse e começasse a fazer ligação entre capitais em toda a Europa, daí a mudança de nome para o atual.

Atualmente a *Euro-Train* garante não só viagens entre capitais, mas sim entre a maior parte das cidades turísticas de toda a Europa Ocidental, pretendendo alcançar o mesmo para toda a Europa num futuro não muito distante. Desde o início que a antiga *Iberian-Train* era conhecida pela qualidade de transporte que oferecia aos clientes. A segurança garantida em cada viagem e os diferentes tipos de preocupações com o ambiente em que se insere foram também fatores importantes no aumento do reconhecimento da empresa em relação às suas adversárias.

Quando a empresa tomou a decisão de se expandir para toda a Europa e mudar o nome para *Euro-Train*, o público aderiu naturalmente à ideia. Numa altura em que o turismo na Europa aumentou consideravelmente, proporcionando um desenvolvimento exponencial da empresa, que passado pouco tempo era capaz de garantir transporte entre qualquer cidade turística da Europa Ocidental.

O desenvolvimento de uma empresa passa também por acompanhar os desenvolvimentos adjacentes à mesma, a *Euro-Train* não é exceção. Como tal, o desenvolvimento da tecnologia móvel permitiu um aumento significativo do uso de *smartphones*, que por sua vez, aumentou consideravelmente o número de reservas efetuadas por dia através da aplicação móvel, algo que podia não ser necessariamente bom se a empresa não conseguisse segurar a estabilidade dos dados.

Assim, a equipa responsável pela gestão da base de dados da *Euro-Train*, que até então trabalhava com um sistema de gestão de base de dados relacional, sentiu-se obrigada a fazer algumas mudanças para combater o elevado número de reservas com que a empresa de transportes tinha agora de lidar. Após algum tempo debruçado sobre o assunto a equipa tomou

a decisão de mudar para a base de dados NoSQL MongoDB, que, à partida, conseguiria assegurar a estabilidade dos dados tanto na situação atual como para o futuro da empresa.

1.2. Apresentação do Caso de Estudo

A aplicação desenvolvida pela *Euro-Train* está disponível ao público, sendo livre de quaisquer custos para o utilizador, tanto na plataforma *Android* como *iOS*.

Após o utilizador fazer o *download* da aplicação está a um pequeno passo de poder desfrutar de todas as vantagens da mesma. Para isso necessita de criar uma nova conta, na qual terá de fornecer o seu nome completo, e-mail, telemóvel, endereço (cidade e país) e por último, nome de utilizador (*username*) e *password*, os quais terá de utilizar sempre que se pretender autenticar na aplicação. Todas estas informações são diretamente inseridas na base de dados da aplicação, após o registo.

Sendo o *login* efetuado com sucesso, o utilizador depara-se com uma interface bastante simples e intuitiva que lhe permite facilmente selecionar a origem e o destino pretendido, introduzindo a cidade e o país, e logo de seguida lhe mostra um horário com todas as viagens disponíveis e as informações das mesmas (hora de partida/chegada) para que este possa escolher a que mais lhe agrada. É de notar que a *Euro-Train* oferece as mesmas viagens independentemente do dia da semana, isto é, as viagens são as mesmas tanto para dias úteis, como feriados ou fins-de-semana. Além disto, uma das maiores preocupações da *Euro-Train* é a grande abrangência de viagens entre cidades de toda a Europa, posto isto, a existência de mais do que uma estação por cidade seria ineficaz, assim sendo, na construção da rede ferroviária essa opção foi completamente excluída.

De seguida, com o dia, hora e viagem selecionados, são apresentados os preços e o utilizador está apto a reservar um lugar, dos que ainda estejam disponíveis no respetivo comboio. Para isso, terá a possibilidade de selecionar o número do lugar pretendido de uma lista que contém todos os lugares disponíveis, em cada uma das carruagens, estando cada lugar caracterizado pela sua respetiva classe (conforto ou turística), tendo sempre em conta que o preço associado a uma viagem em lugar de classe conforto é superior.

Após efetuar o pagamento, a reserva fica registada contendo toda a informação sobre a viagem e o lugar selecionado, em conjunto com um número único que a identifica e um código, a que só o utilizador em questão terá acesso e que deverá apresentar momentos antes da viagem, de modo a validar o seu bilhete.

Assim, a base de dados da aplicação terá também de guardar informações relativas a cada comboio, como o seu tipo, número total de lugares, uma lista de lugares (contendo informação sobre cada lugar) e uma breve descrição.

1.3. Motivação e Objetivos

Na realização deste projeto foi-nos proposta a implementação de uma BD NoSQL com base num tema de trabalho facultado, reservas de viagens em comboios nacionais e internacionais. Este tema apresenta alguma complexidade a nível de implementação do SBD tornando o desafio do projeto ainda mais aliciante. Reserva de viagens de comboios é sempre sinónimo de perda de algum tempo por parte do utente quer na escolha da reserva da viagem que pretende quer na fila de espera para a escolha da mesma.

Com o desenvolvimento deste projeto pretendemos simplificar todo o processo desde que o utente efetua a sua reserva da viagem de comboio até quando chega ao destino pretendido. As reservas efetuam-se via *online,* facilitando a visualização de todas as viagens existentes por parte do cliente, assim como a existência de lugares disponíveis no comboio pertencente a viagem que o cliente pretende reservar. A aplicação irá corresponder a todas a exigências do cliente no momento em que pretende efetuar a reserva numa determinada viagem.

Tratando-se esta de uma base de dados resultante de um processo de migração da base de dado relacional anteriormente criada, o objetivo é obter um Sistema de Gestão de Base de Dados capaz de aceitar um elevado tráfego de utilizadores ao mesmo tempo que assegura a performance pretendida, sendo esta uma das principais vantagens dos SGBD NoSQL, o qual será mais à frente aprofundado.

Assim sendo, construímos uma base de dados simples, mas mantendo-nos sempre focados na realidade e nos problemas que surgem frequentemente aos utilizadores deste tipo de viagens. Podemos assim dar entender como funciona um sistema de reservas de viagens desta natureza, facilitando a gestão dos dados e todo o processo das reservas efetuadas *online*, bem como o armazenamento de toda a informação numa base de dados. Em suma, o cliente irá poupar um dos recursos mais valiosos nos dias que correm, tempo.

1.4. Estrutura do Relatório

Após termos apresentado o caso de estudo escolhido e a motivação que está por detrás deste, nos seguintes capítulos deste projeto serão abordadas todas as etapas que tivemos de percorrer até chegar ao resultado final. No segundo capítulo é explicado o porquê deste processo de migração para uma base de dados NoSQL, o MongoDB, assim como as vantagens e desvantagens trazidas pela mesma. O terceiro capítulo aborda a parte mais prática, ou seja, as decisões tomadas ao nível da implementação da própria base de dados e o porquê das mesmas. No quarto e último capítulo é feita uma apreciação crítica ao trabalho realizado, tendo por base o projeto anteriormente elaborado. Por fim, em anexo, são apresentados exemplos de *queries* realizadas em MongoDB com a respetiva correspondência em MySQL.

2. Introdução ao novo paradigma NoSQL

2.1. Opção por Sistemas NoSQL na *Euro-Train*

Como dito anteriormente, este trabalho consiste na criação de uma base de dados em MongoDB (NoSQL) para a empresa de comboios *Euro-Train*, sendo este o resultado da migração feita a partir da base de dados relacional anteriormente criada no primeiro projeto.

Desta forma, é importante saber o que realmente são base de dados NoSQL e em que medidas estas nos trazem vantagens em relação aos SGBDR.

Como sabemos, Sistemas de Base de Dados Relacionais têm dominado o mercado desde há bastante tempo, tendo se tornando o padrão para a maioria dos Sistemas de Gestão de Base de Dados (SGBD). Isto deve-se em grande parte ao modelo na qual estas se baseiam, o modelo ACID, que preservam a integridade de uma dada transação através de Atomicidade, Consistência, Isolamento e Durabilidade.

Estas caraterísticas permitiram manter os SGBDs Relacionais sempre numa posição de predominância entre os restantes, mas ao mesmo tempo, não impediu o aparecimento de determinados problemas, grande parte devido ao elevado crescimento do volume de dados presente na base de dados de algumas organizações. Temos o exemplo do *Facebook* que, como sabemos, é um caso real de como este crescimento se tem rapidamente expandido. No caso destes tipos de organizações, a utilização dos SGBDs relacionais tem se mostrado muito problemática e não tão eficiente.

As bases de dados relacionais são extremamente eficientes no armazenamento de informação estruturada e consistente, porém, em situações distintas estas podem não ser as características mais relevantes no armazenamento de informação, daí o aparecimento de soluções alternativas capazes de armazenar e consultar dados não estruturados.

As bases de dados relacionais apresentam uma estrutura muito rígida, que deve ser assegurada durante toda a utilização destas. Assim, a necessidade de aplicar alterações na estrutura destas podem ser demoradas, o que nem sempre é o mais conveniente. As bases de dados não relacionais apresentam flexibilidade neste ponto, uma vez que não apresentando uma estrutura rígida permitem que alterações na estrutura de armazenamento de dados da mesma seja muito menos demorada que nas bases de dados relacionais.

Assim, os principais problemas encontrados com a utilização do Modelo Relacional estão principalmente na dificuldade de conciliar o tipo de modelo com a procura do aumento da escalabilidade que está cada vez mais frequente.

Desta forma, surge assim um novo paradigma que são as Base de Dados NoSQL (*Not only* SQL) que vêm de certa forma tentar solucionar diversos problemas relacionados com a escalabilidade, performance e disponibilidade dos SGBDRs. Promovem assim uma alternativa de alto armazenamento com velocidade e grande disponibilidade, procurando livrar-se de certas regras e estruturas que caracterizam o Modelo Relacional. A proposta dos modelos NoSQL na realidade não é extinguir o Modelo Relacional, mas utilizá-lo em casos onde é necessária uma maior flexibilidade na estruturação da base de dados.

As bases de dados NoSQL usam diversos modelos de dados, entre eles documentos, grafos, *key-value* e *column oriented*. Como dito anteriormente, o SGBD que iremos utilizar será o MongoDB.

Assim sendo, com esta migração pretendemos obter um Sistema de Gestão de Base de Dados capaz de aceitar um elevado tráfego de utilizadores ao mesmo tempo que assegura a performance pretendida.

2.2. MongoDB

Uma das formas mais populares para armazenar dados em bases de dados não relacionais são os modelos de dados orientados a documentos, onde cada registro e os seus respetivos dados são considerados como um "documento". O MongoDB, o que iremos utilizar, é um modelo *open-source* sendo dos mais desenvolvidos e explorados no mercado na atualidade.

O MongoDB armazena os dados em documentos JSON com um esquema dinâmico, isto é, os dados armazenados não têm de seguir um esquema rígido, podem seguir uma qualquer estrutura.

Vantagens

Documentos como unidades independentes:

O aumento do volume de dados presente nas bases de dados proporcionou a distribuição dos dados por várias máquinas (*clusters*), o conceito *Sharding*. Com os documentos a comportarem-se como unidades independentes, permitiu que estes fossem distribuídos pelos vários *clusters* melhorando assim consideravelmente o desempenho e permitindo também maior escalabilidade.

Dados não estruturados são mais facilmente armazenados:

As inserções numa base de dados orientada a documentos apresentam um desempenho consideravelmente melhor que numa base de dados relacional, uma vez que esta não apresenta um *schema* e como tal não existe uma série de considerações a serem verificadas.

Consultas e transações mais simples:

A inexistência de relacionamentos entre documentos implica a inexistência de transações e *joins*. Para combater estas inexistências, um documento possibilita todas as informações necessárias. Assim sendo, as consultas/transações são mais simples e fáceis de ajustar.

Desvantagens

Redundância de dados:

Este é um fator muito comum nestes tipos de bases de dados, o facto de não existirem relacionamentos entre os documentos implica que existam muitos dados repetidos.

Inconsistência:

Ao contrário dos SGBDR, o MongoDB não garante a consistência dos dados presentes, cabe ao programador da aplicação garantir que todos as inserções ou alterações nos dados os mantêm consistentes.

3. Processo de migração

3.1. Diagrama do Modelo Lógico anterior

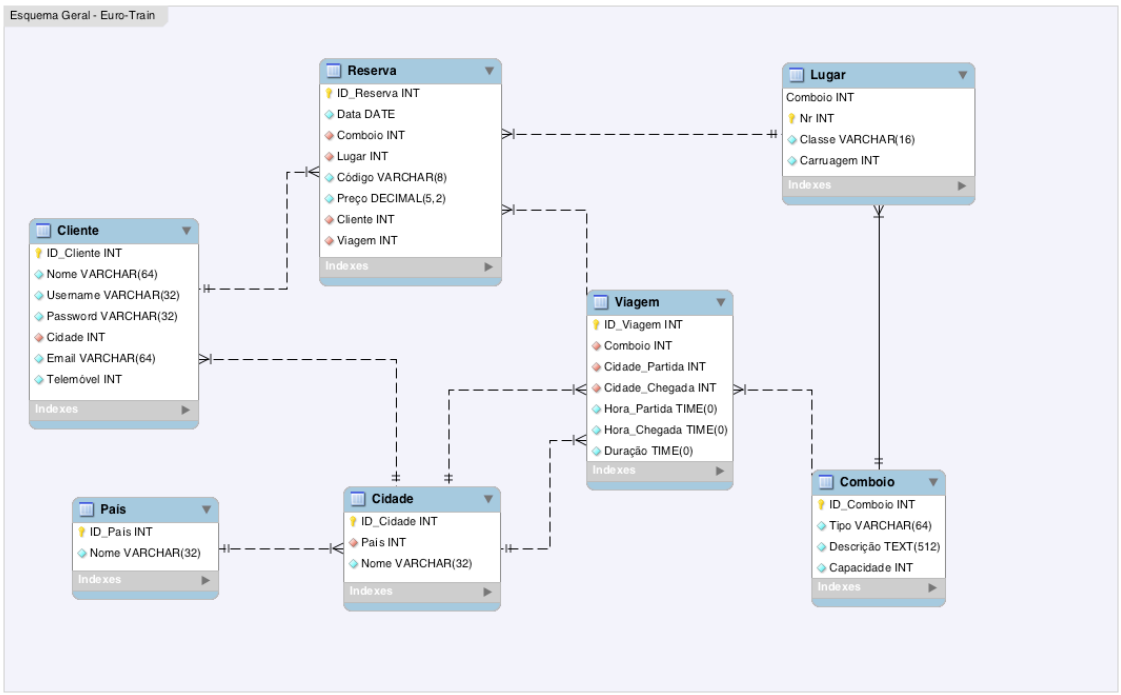

Figura 1 – Diagrama do Modelo Lógico.

3.2. Esquema das Coleções

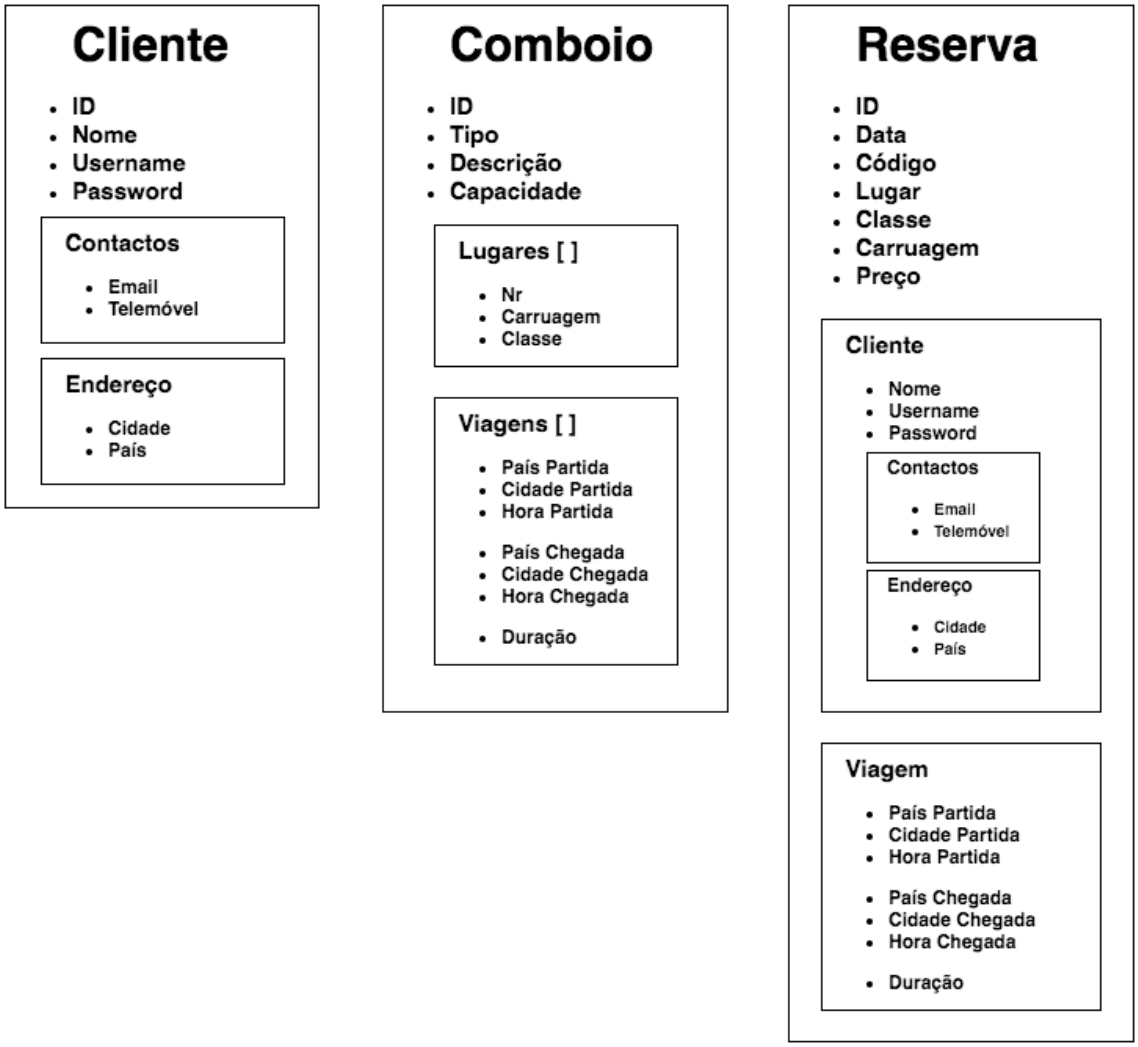

Figura 2 – Esquema das coleções.

Para implementar a nossa Base de Dados em MongoDB decidimos criar três coleções distintas: Cliente, Comboio e Reserva. O Cliente possui os campos ID, Nome, Username, Password, Email, Telemóvel, Cidade e País. O Comboio possui os campos ID, Tipo, Descrição, Capacidade, Lista de Lugares, os quais possuem um Nr, Carruagem e Classe, e por último uma lista de Viagens, as quais possuem um País Partida, Cidade Partida, Hora Partida, País Chegada, Cidade Chegada, Hora Chegada e Duração. Por ultimo, a Reserva possui um ID, Data, Código, Lugar, Classe, Carruagem, Preço, Cliente (já descrito anteriormente) e uma Viagem (já descrito anteriormente).

3.3. Estrutura dos documentos

- **Cliente**

{

"_id" : ID do Cliente (gerado automaticamente pelo MongoDB),
"Nome" : Nome do Cliente,
"Username" : Username do Cliente,
"Password" : Password do Cliente,
"Contactos" : {
"Email" : Email do Cliente,
"Telemovel" : Número de telemóvel do Cliente,
},
"Endereço" : {
"Cidade" : Cidade do Cliente,
"Pais" : País do Cliente.
}

}

- **Comboio**

{

"_id" : ID do Comboio (gerado automaticamente pelo MongoDB),
"Tipo" : Tipo do Comboio (Alfa, Intercidades ou Internacional),
"Descrição" : Breve descrição do Comboio,
"Capacidade" : Capacidade do Comboio,
"Lugares" : [
{
"Nr" : Número do Lugar,
"Carruagem" : Número da Carruagem do respetivo Lugar,
"Classe" : Classe do Lugar,
},
],
"Viagens" : [
{
"País Partida" : País de partida da Viagem,
"Cidade Partida" : Cidade de partida da Viagem,
"Hora Partida" : Hora de partida da Viagem,
"País Chegada" : País de chegada da Viagem.
"Cidade Chegada" : Cidade de chegada da Viagem,
"Hora Chegada" : Hora de chegada da Viagem,

"Duração" : Duração da Viagem,

},

]

}

- **Reserva**

{

"_id" : ID da Reserva (gerado automaticamente pelo MongoDB),

"Data" : Data da Viagem a que corresponde a Reserva,

"Código" : Código de verificação da Reserva,

"Lugar" : Número do Lugar reservado,

"Classe" : Classe do Lugar reservado,

"Carruagem" : Número da Carruagem onde se encontra o Lugar reservado,

"Preço" : Preço da Viagem,

"Cliente" : {

"Nome" : Nome do Cliente que efetuou a Reserva,

"Username" : *Username* do Cliente que efetuou a Reserva,

"Password" : Password do Cliente que efetuou a Reserva,

"Contactos" : {

"Email" : Email do Cliente que efetuou a Reserva,

"Telemovel" : Número de telemóvel do Cliente que efetuou a Reserva,

},

"Endereço" : {

"Cidade" : Cidade do Cliente que efetuou a Reserva,

"Pais" : País do Cliente que efetuou a Reserva,

}

},

"Viagem" : {

"País Partida" : País de Partida da Viagem a que corresponde a Reserva,

"Cidade Partida" : Cidade de Partida da Viagem a que corresponde a Reserva,

"Hora Partida" : Hora de Partida da Viagem a que corresponde a Reserva,

"País Chegada" : País de Chegada da Viagem a que corresponde a Reserva,

"Cidade Chegada" : Cidade de Chegada da Viagem a que corresponde a Reserva,

"Hora Chegada" : Hora de Chegada da Viagem a que corresponde a Reserva,

"Duração" : Duração da Viagem a que corresponde a Reserva.

}

}

3.4. Migração dos dados de MySQL para MongoDB

O processo de migração de uma BD relacional (MySQL), para uma BD NoSQL (MongoDB), possui várias opções de importação de dados, sendo necessário a seleção de uma. A opção tomada foi a implementação de uma aplicação que na sua essência recolhe os dados de uma BD presente em MySQL, e armazena esta no MongoDB. A aplicação foi codificada em Java, linguagem de programação em que a equipa técnica da *Euro-Train* se sentia mais confortável, e para tal foi necessário o uso dos drivers que permitiam a conexão a ambos os SGBDs (MySQL e MongoDB), permitindo assim efetuar a recolha e inserção dos dados.

No processo de recolha de uma BD e inserção para outra, é necessário ter em consideração o esquema que delineámos inicialmente em relação à BD em MongoDB, isto é, através deste esquema possuímos um guia, na forma de construção e inserção das *Collections* e seus respetivos documentos. Este aspeto é de grande importância dado que a BD em MongoDB apresenta uma grande flexibilidade, como já referido anteriormente, que permite um documento possuir uma estrutura qualquer.

Numa visão mais completa, o processo de recolha dos dados da base de dados relacional consiste em 3 operações. Cada operação corresponde a uma *Collection*, isto é, cada operação é responsável por recolher os dados referente aos documentos que constituem uma Collection na base de dados NoSQL. Para tal foi necessário combinar os dados necessários no MySQL, através de operações de SELECT e INNER JOIN. As *Collections* obtidas são o Cliente, o Comboio e a Reserva.

Quanto ao processo de inserção, este também é constituído por 3 operações, que são intercaladas com as operações de recolha anteriormente referidas. Como os dados são recolhidos de forma a obter os dados necessários para um documento de uma certa *Collection*, as operações de inserção em MongoDB tornam-se mais simples. Limitam-se, portanto, partindo dos dados recolhidos, formar o documento pretendido, e inserir o mesmo na BD. É neste passo que garantimos a criação do esquema de cada *Collection*.

3.5. Migração do esquema de MySQL para MongoDB

A principal característica na migração de um esquema de MySQL para MongoDB é a remoção dos relacionamentos entre tabelas, presentes no primeiro caso. Apesar da inexistência de relacionamentos no esquema de dados presentes no MongoDB, os dados precisam de ser armazenados de maneira a fazer corresponder todas as necessidades que eram anteriormente tidas no modelo relacional, uma vez que o propósito da empresa se mantém.

Inicialmente era necessário fazer a distinção das *collections* que iríamos utilizar e apenas posteriormente abordar os diferentes documentos que estas iriam conter.

Seria inevitável não ter uma *collection* para a informação dos clientes uma vez que a nossa aplicação tem de lidar com centenas de registos de novos utilizadores por dia, o que implica centenas de inserções. A mesma razão deu origem à *collection* Reserva. Desta forma, é possível certificar que a nossa base de dados apresenta um elevado desempenho, tanto quanto a registos de novos clientes como a reservas por parte destes.

Por fim, consideramos necessária uma terceira *collection*, Comboio, capaz de armazenar os comboios disponibilizados pela *Euro-Train*, assim como os lugares que cada um destes possuí e as viagens que estão encarregados de fazer, a existência desta *collection* é também necessária uma vez que, apesar de não existirem muitas inserções de comboios, seria necessária, por exemplo, uma listagem de todas as viagens que a *Euro-Train* proporciona aos seus clientes, mesmo que uma determinada viagem ainda não tenha reservas efetuadas, não permitindo assim que esta informação fosse contida na *collection* Reserva.

Com a distinção das *collections* realizada, seria agora o momento indicado para abordar os diferentes documentos que cada uma destas deveria conter.

A *collection* Cliente deverá conter toda a informação que o utilizador fornece aquando o registo na aplicação: Nome, *Username*, Password, Contactos e Endereço. A *collection* Comboio deverá conter toda a informação do respetivo comboio, desde o seu Tipo, Descrição, Capacidade, à identificação de cada lugar e carruagem que contém, apresentando o número do lugar, a carruagem em que se encontra e a classe deste. Deverá também conter a lista de viagens que lhe estão atribuídas, contendo toda a informação de cada uma das viagens: País, Cidade, Hora de Partida/Chegada e Duração.

A *collection* Reserva, e a mais completa em relação às restantes, deverá conter todas as informações relativas à reserva efetuada (Data, Código, Lugar, Carruagem, Classe, Preço) ao Cliente (Nome, *Username*, Password, Contactos e Endereço) e mesmo à Viagem (País, Cidade e Hora de Partida/Chegada e respetiva Duração).

Como tal, o novo esquema deve garantir que as reservas são efetivamente armazenadas corretamente, permitindo um elevado desempenho nas inserções e consultas mais simples comparativamente ao modelo relacional, algo que só é conseguido através de redundância de dados, algo que podemos verificar neste novo esquema, muito predominante na *collection* Reserva.

4. Conclusão e apreciação crítica

Após a conclusão do projeto proposto é nos agora possível fazer uma apreciação crítica sobre o mesmo, tendo também em conta o resultado do projeto anteriormente realizado. Começando pelas desvantagens, embora esta migração tenha sido proposta com o intuito de melhorar a conceção da Base de Dados, é possível apontar vários pontos negativos que com ela surgiram.

- ***Tiggers* do MySQL**

O facto do SGBD MySQL possuir *triggers* garante que a inserção dos lugares no comboio fique mais simplificada, pois atualiza a capacidade no momento da inserção, comparada ao MongoDB, onde é necessário executar um *update* para atualizar a capacidade do comboio.

- **Combinação de dados**

Uma base de dados SQL permite-nos combinar os vários dados, de tabelas diferentes, através dos relacionamentos existentes (*Foreign keys*), sendo necessário o uso de *JOINs*, para tal. No entanto, no MongoDB, não é possível tal mecanismo, sendo necessário por isso garantir a combinação dos dados no momento da inserção, o que pode originar redundância de dados. Este facto, sucede-se na *Collection* Comboio, onde encontramos uma lista de documentos embebidos de Viagens, e na *Collection* Reserva, que encontramos um documento Cliente e um documento Viagem embebidos. Para além disso, pode acontecer não ser possível combinar os dados no momento da inserção, sendo necessário recorrer, a uma segunda *query*, para recolher os dados pretendidos. Este problema acontece quando pretendemos obter os lugares livres numa viagem, onde é necessário combinar os lugares do comboio com os lugares já reservados, sendo isto apenas possível ao nível da aplicação, enquanto que no modelo relacional era facilmente conseguido através da utilização de JOINs.

- **Inexistência de restrições**

Uma base de dados em MongoDB, não apresenta um esquema rígido como uma base de dados em MySQL, onde é necessário definir previamente um esquema e estrutura, para a construção da base de dados. Para além disso, possui as restrições de integridade, de entidade, de domínio e ainda gerais, que tornam o esquema mais rígido. Pelo contrário, o MongoDB não apresenta qualquer restrição, possuindo desta forma um esquema extremamente flexível, o que pode provocar a inconsistência da base de dados. É por isso mesmo, necessário, um trabalho adicional por parte do programador na parte da aplicação para garantir que qualquer inserção ou modificação feita na base de dados irá manter esta consistente.

No entanto, como é obvio, esta migração também trouxe vantagens, que de certa forma compensam os pontos negativos anteriormente referidos.

- **Elevado desempenho nas inserções**

A escolha pela utilização do MongoDB na situação em que a Euro-Train se encontrava, foi, no entanto, a ideal. A explicação para isto é muito simples, a Euro-Train encontrava-se numa posição em que não conseguia lidar com os milhares de registos e reservas por dia, o demasiado volume de dados com necessidade de serem armazenados implicava um desempenho muito baixo para a aplicação. A migração para MongoDB veio resolver isto na medida que, com a utilização deste tipo de base de dados, as inserções em qualquer uma das *collections* são realizadas com um desempenho muito melhor que na situação anterior, uma vez que nesta não são verificadas inúmeras restrições de integridade para garantir a consistência da BD após a inserção.

- **Facilidade quanto a alterações no esquema da BD**

No modelo relacional a base de dados apresentava uma estrutura muito rígida, e por isso, pouco flexível, o que originava muito tempo perdido pela equipa de gestão de BDs da *Euro-Train* quando, por qualquer razão, uma mudança na estrutura da BD fosse necessária. O mesmo não se verifica com a utilização do MongoDB.

Nesta situação na base de dados sem esquema (*no schema*), logo sem as características rígidas que uma BD do modelo anterior apresenta, o processo de modificação do esquema implica muito menos tempo gasto pela equipa de gestão da *Euro-Train*, podendo este ser utilizado na manutenção da consistência da BD por parte dos programadores, ao nível da própria aplicação.

Finalizando, para percebermos melhor como é que as BDs NoSQL se distinguem das BDs Relacionais vamos expor aquele que é o Teorema CAP e aplica-lo ao nosso caso prático do MongoDB.

O teorema de CAP assenta em três requisitos que afetam os sistemas distribuídos. Esses requisitos são a Consistência (*Consistency*), Disponibilidade (*Availability*) e tolerância ao particionamento (*Partition Tolerance*). Segundo a hipótese de Brewer (2000), num sistema distribuído é desejável obter os três requisitos apresentados anteriormente. Porém, no mundo real, essas propriedades são impossíveis de serem obtidas ao mesmo tempo. De acordo com o teorema, um sistema distribuído só pode garantir apenas duas dessas propriedades simultaneamente.

Para que um sistema seja considerado consistente, deve-se garantir que, após a realização de uma determinada transação, o sistema deve manter a consistência dos dados, ou seja, uma vez escrito um registo, este fica disponível para ser utilizado imediatamente. Um sistema de dados distribuído consistente é classificado como fortemente consistente (por exemplo, ACID) ou como tendo alguma forma fraca de consistência (por exemplo, BASE).

O modelo de consistência eventual BASE (*Basically Available, Soft-state, Eventual consistency*), é utilizado por bases de dados NoSQL. A consistência fraca é fornecida sob a forma de consistência eventual, o que significa que a base de dados pode atingir um estado consistente, incluindo normalmente as bases de dados em que os dados são replicados. Apenas existe a garantia que a versão mais recente de um registo consistente num nó, sendo que versões antigas deste registo podem ainda existir noutros nós, eventualmente todos os nós têm a última versão.

A disponibilidade que, segundo Browne (2009), refere-se à conceção e implementação de um sistema de modo que seja assegurado que este permanece ativo durante um determinado período de tempo. Basicamente, mesmo que um nó da rede esteja desligado, todos os clientes podem encontrar, sempre que pretenderem, pelo menos uma cópia dos dados que necessitam.

A tolerância ao particionamento refere-se ao sistema ser implementado em servidores diferentes, e continuar com os seus dados, propriedades e características, sempre transparentes para o cliente.

Então, como só é possível escolher dois dos três requisitos, e de acordo com (Brewer, 2000) é possível obter as seguintes combinações: CA, CP e AP. Neste trabalho apenas vamos abordar as combinações CA e CP, pois estas correspondem às bases de dados implementadas.

Quando consideramos a combinação Consistência e Disponibilidade (CA), abdicamos da tolerância ao particionamento, podendo o sistema inteiro ficar indisponível até o membro do cluster voltar. Para além disso, com o aumento dos dados, o sistema não consegue responder de forma desejada. Estão associados a base de dados relacionais, nomeadamente MySQL.

Por outro lado, uma base de dados MongoDB, apresenta uma consistência forte e tolerância ao particionamento, abrindo mão da disponibilidade. O MongoDB apresenta uma consistência forte por padrão, isto é, se fizermos uma escrita e, em seguida, uma leitura,

assumindo que a escrita foi bem-sucedida, será sempre possível ler o resultado da escrita. Isso acontece porque o MongoDB é um sistema que apresenta um nó primário (master) e todas as leituras realizam-se neste nó, por padrão. Quando opcionalmente, ativamos a leitura a nós secundários, o MongoDB, torna-se eventualmente consistente onde é possível ler resultados desatualizados.

Em suma, uma das principais vantagens na migração de MySQL para MongoDB, é da base de dados passar a suportar a tolerância ao particionamento, o que se torna muito conveniente, quando a quantidade de dados se torna considerável, e é necessário implementar a partição dos mesmos. No entanto, é de considerar que a disponibilidade é sacrificada, podendo alguns dados estar inacessíveis, mas os restantes estão consistentes.

Referências

1. MongoDB, 2015. RDBMS to MongoDB Migration Guide. [pdf] Disponível em: <https://webassets.mongodb.com/_com_assets/collateral/RDBMStoMongoDBMigration.pdf?_ga=1.130066959.1443285478.1481552884>

2. Paulo Fagundes, 2014. Quando usar MongoDB ao invés de MySQL ou outro RDBMS. [online] Disponível em: < https://mongodbwise.wordpress.com/2014/05/29/quando-usar-mongodb-ao-inves-de-mysql-ou-outro-rdbms/ >

3. MongoDB, 2015. What is a MongoDB ? [online] Disponível em: <https://www.mongodb.com/what-is-mongodb>

4. IMB, 2011. Explore o MongoDB. [online] Disponível em: <https://www.ibm.com/developerworks/br/library/os-mongodb4/ >

Lista de Siglas e Acrónimos

BD Base de Dados

SQL Structured Query Language

SGBD Sistema de Gestão de Base de Dados

SBDR Sistema de Base de Dados Relacional

SBD Sistema de Base de Dados

NoSQL Not only SQL

Anexos

1. Exemplos de queries em MongoDB.
2. Aplicação de migração dos dados de MySQL para MongoDB.

I. Anexo 1 - Exemplos de *queries* em MongoDB

Neste anexo são apresentados exemplos de resoluções de algumas queries em MongoDB com a respetiva correspondência em SQL, assim como os seus resultados.

1. **Query que apresenta a lista de reservas, com o respetivo cliente, ordenado pelo preço.**

```sql
USE EuroTrain;

SELECT Cliente.Nome AS Nome, Reserva.Preço AS Preço FROM Reserva
INNER JOIN Cliente ON Cliente.ID_Cliente = Reserva.Cliente
ORDER BY Preço DESC;
```

Figura 3 - Resolução SQL da *query* 1.

```
// Querie 1 //
db.Reserva.find({},{_id:0,"Cliente.Nome":1,Preço:1}).sort({Preço:-1})
```

Figura 4 - Resolução MongoDB da *query* 1.

```
> db.Reserva.find({},{_id:0,"Cliente.Nome":1,Preço:1}).sort({Preço:-1})
{ "Preço" : 39.9, "Cliente" : { "Nome" : "Joana" } }
{ "Preço" : 39.9, "Cliente" : { "Nome" : "Müller" } }
{ "Preço" : 33.9, "Cliente" : { "Nome" : "Müller" } }
{ "Preço" : 18.9, "Cliente" : { "Nome" : "José" } }
{ "Preço" : 18.9, "Cliente" : { "Nome" : "Ana" } }
{ "Preço" : 17.9, "Cliente" : { "Nome" : "José" } }
{ "Preço" : 17.9, "Cliente" : { "Nome" : "Ana" } }
{ "Preço" : 17.9, "Cliente" : { "Nome" : "Ana" } }
{ "Preço" : 14.9, "Cliente" : { "Nome" : "Patrick" } }
{ "Preço" : 14.9, "Cliente" : { "Nome" : "Patrick" } }
```

Figura 5 - Resultados da query 1.

2. **Query que apresenta a lista de viagens de cada comboio, com a respetiva duração.**

```sql
USE EuroTrain;

SELECT Comboio.ID_Comboio AS Comboio, CP.Nome AS Partida, CC.Nome AS Chegada, Viagem.Duração FROM Viagem
    INNER JOIN Comboio
    ON Comboio.ID_Comboio = Viagem.Comboio
        INNER JOIN Cidade AS CP
        ON CP.ID_Cidade = Viagem.Cidade_Partida
            INNER JOIN Cidade AS CC
            ON CC.ID_Cidade = Viagem.Cidade_Chegada;
```

Figura 6 - Resolução SQL da query 2.

```
// Querie 2 //
db.Comboio.find({},{"Viagens.Cidade Partida":1,"Viagens.Cidade Chegada":1,"Viagens.Duração":1}).pretty()
```

Figura 7 - Resolução MongoDB da query 2.

```
> db.Comboio.find({},{"Viagens.Cidade Partida":1,"Viagens.Cidade Chegada":1,"Viagens.Duração":1}).pretty()
{
        "_id" : ObjectId("58813abf83ba0b7645058292"),
        "Viagens" : [
                {
                        "Cidade Partida" : "Braga",
                        "Cidade Chegada" : "Porto",
                        "Duração" : "01:20:00"
                },
                {
                        "Cidade Partida" : "Porto",
                        "Cidade Chegada" : "Lisboa",
                        "Duração" : "03:30:00"
                },
                {
                        "Cidade Partida" : "Lisboa",
                        "Cidade Chegada" : "Porto",
                        "Duração" : "03:30:00"
                },
                {
                        "Cidade Partida" : "Porto",
                        "Cidade Chegada" : "Braga",
                        "Duração" : "01:20:00"
                }
        ]
}
{
        "_id" : ObjectId("58813abf83ba0b7645058293"),
        "Viagens" : [
                {
                        "Cidade Partida" : "Braga",
                        "Cidade Chegada" : "Madrid",
                        "Duração" : "04:00:00"
                },
                {
                        "Cidade Partida" : "Madrid",
                        "Cidade Chegada" : "Braga",
                        "Duração" : "04:00:00"
                }
        ]
}
```

Figura 8 - Resultados da query 2.

3. **Query que apresenta a lista dos usernames dos clientes que efetuaram reservas, ordenada pelo respetivo número de reservas efetuadas.**

```sql
USE EuroTrain;

SELECT Cliente.Username, COUNT(*) AS `Número de Reservas` FROM Reserva
    INNER JOIN Cliente ON Cliente.ID_Cliente = Reserva.Cliente
GROUP BY Cliente.Username
ORDER BY `Número de Reservas` DESC;
```

Figura 9 - Resolução SQL da query 5.

```
// Querie 5 //
   db.Reserva.aggregate([{"$group":{_id:"$Cliente.Username","Número de Reservas":{$sum:1}}},{$sort:{"Número de Reservas":-1}}])
OU db.Reserva.aggregate([{"$group":{_id:{Username: "$Cliente.Username"},"Número de Reservas":{$sum:1}}},{$sort:{"Número de Reservas":-1}}])
```

Figura 10 - Resolução MongoDB da query 5.

```
> db.Reserva.aggregate([
...     {"$group" : {_id: {Username: "$Cliente.Username"}, "Número de Reservas": { $sum:1 }}},
...     {$sort:{"Número de Reservas":-1}}
... ])
{ "_id" : { "Username" : "ana22" }, "Número de Reservas" : 3 }
{ "_id" : { "Username" : "muller23" }, "Número de Reservas" : 2 }
{ "_id" : { "Username" : "patrick11" }, "Número de Reservas" : 2 }
{ "_id" : { "Username" : "jose33" }, "Número de Reservas" : 2 }
{ "_id" : { "Username" : "ju213" }, "Número de Reservas" : 1 }
```

Figura 11 - Resultados da query 5.

II. Anexo 2 – Aplicação de migração dos dados

- Classe EuroTrain

```
public class EuroTrain {

    public static void main(String[] args) {
        MongoClient mongo = new MongoClient("localhost",27017);
        MongoDatabase db = mongo.getDatabase("EuroTrain");

        Connection con = null;
        Migrate m = new Migrate(con, db);

        m.drop();
        m.migrateClient();
        m.migrateTrain();
        m.migrateReserve();
    }
}
```

- Classe Migrate

```
public class Migrate {
    private Connection con;
    private MongoDatabase db;

    public Migrate(Connection con, MongoDatabase db){
        this.db = db;
        this.con = con;
    }

    public void migrateClient(){
        String nome = null, username = null, password = null, email = null, cidade = null, pais =
null;
        int telemovel = 0;
        try{
```

```
        con = Connect.connect();
        PreparedStatement ps = con.prepareStatement(""
            + "SELECT * FROM Cliente \n"
            + "INNER JOIN Cidade \n"
            + "ON Cidade.ID_Cidade = Cliente.Cidade \n"
            + "INNER JOIN País \n"
            + "ON País.ID_País = Cidade.País \n");
        ResultSet rs = ps.executeQuery();

        MongoCollection<BasicDBObject> collection = this.db.getCollection("Cliente",
BasicDBObject.class);

        while(rs.next()){
           nome = rs.getString("Cliente.Nome");
           username = rs.getString("Cliente.Username");
           password = rs.getString("Cliente.Password");
           email = rs.getString("Cliente.Email");
           telemovel = rs.getInt("Cliente.Telemóvel");
           cidade = rs.getString("Cidade.Nome");
           pais = rs.getString("País.Nome");

           BasicDBObject document = new BasicDBObject();
           document.put("Nome", nome);
           document.put("Username", username);
           document.put("Password", password);
           document.put("Contactos",new BasicDBObject("Email", email)
                                .append("Telemovel",telemovel));
           document.put("Endereço",new BasicDBObject("Cidade", cidade)
                               .append("Pais", pais));
           collection.insertOne(document);
        }
      }
      catch(SQLException e){
        System.out.printf(e.getMessage());
      }
      finally{
        try{
          Connect.close(con);
        }
        catch(Exception e){
```

```
            System.out.printf(e.getMessage());
        }
    }
}

public void migrateTrain(){
    String tipo = null, descricao = null, classe = null, cidadeP = null, cidadeC = null, duracao =
null, paisC = null, paisP = null;
    int capacidade = 0, id = 0, lugar = 0, carruagem = 0;
    java.sql.Time horaP = null, horaC = null;

    try{
        con = Connect.connect();
        PreparedStatement ps = con.prepareStatement(""
            + "SELECT * FROM Comboio \n");
        ResultSet rs = ps.executeQuery();

        MongoCollection<BasicDBObject> collection = this.db.getCollection("Comboio",
BasicDBObject.class);

        while(rs.next()){
            id = rs.getInt("ID_Comboio");
            tipo = rs.getString("Tipo");
            descricao = rs.getString("Descrição");
            capacidade = rs.getInt("Capacidade");

            BasicDBObject document = new BasicDBObject();
            document.put("Tipo", tipo);
            document.put("Descrição",descricao);
            document.put("Capacidade",capacidade);

            PreparedStatement ps2 = con.prepareStatement(""
              + "SELECT * FROM Lugar \n"
              + "WHERE Comboio = ?");
            ps2.setString(1, Integer.toString(id));
            ResultSet rs2 = ps2.executeQuery();

            BasicDBList lista = new BasicDBList();

            while(rs2.next()){
```

```
    lugar = rs2.getInt("Nr");
    carruagem = rs2.getInt("Carruagem");
    classe = rs2.getString("Classe");

    BasicDBObject docLugar = new BasicDBObject();
    docLugar.put("Nr",lugar);
    docLugar.put("Carruagem", carruagem);
    docLugar.put("Classe", classe);

    lista.add(docLugar);
}

document.put("Lugares",lista);

PreparedStatement ps3 = con.prepareStatement(""
   + "SELECT * FROM Viagem \n"
   + "INNER JOIN Cidade AS CP \n"
   + "ON CP.ID_Cidade = Viagem.Cidade_Partida \n"
   + "INNER JOIN Cidade AS CC \n"
   + "ON CC.ID_Cidade = Viagem.Cidade_Chegada \n"
   + "INNER JOIN País AS PP \n"
   + "ON CP.País = PP.ID_País\n"
   + "INNER JOIN País AS PC \n"
   + "ON CC.País = PC.ID_País \n"
   + "WHERE Comboio = ?");
ps3.setString(1, Integer.toString(id));
ResultSet rs3 = ps3.executeQuery();

BasicDBList listaV = new BasicDBList();

while(rs3.next()){
    cidadeP = rs3.getString("CP.Nome");
    cidadeC = rs3.getString("CC.Nome");
    paisP = rs3.getString("PP.Nome");
    paisC = rs3.getString("PC.Nome");
    horaP = rs3.getTime("Viagem.Hora_Partida");
    horaC = rs3.getTime("Viagem.Hora_Chegada");
    duracao = rs3.getString("Viagem.Duração");

    BasicDBObject docViagem = new BasicDBObject();
```

```
                docViagem.put("País Partida",paisP);
                docViagem.put("Cidade Partida", cidadeP);
                docViagem.put("Hora Partida", horaP);
                docViagem.put("País Chegada",paisC);
                docViagem.put("Cidade Chegada", cidadeC);
                docViagem.put("Hora Chegada", horaC);
                docViagem.put("Duração",duracao);

                listaV.add(docViagem);
            }
            document.put("Viagens",listaV);
            collection.insertOne(document);
        }
    }
    catch(SQLException e){
      System.out.printf(e.getMessage());
    }
    finally{
        try{
          Connect.close(con);
        }
        catch(Exception e){
          System.out.printf(e.getMessage());
        }
    }
  }

  public void migrateReserve(){
      String codigo = null, nome = null, username = null, password = null, email = null, cidade =
null, pais = null;
      String classe = null, cidadeP = null, cidadeC = null, horaP = null, horaC = null, duracao =
null, paisC = null, paisP = null;
      int id = 0, lugar = 0, carruagem = 0, telemovel = 0;
      double preco = 0;
      java.sql.Date data = null;

      try{
          con = Connect.connect();
          PreparedStatement ps = con.prepareStatement(""
                + "SELECT * FROM Reserva AS R \n"
```

```
            + "INNER JOIN Lugar AS L \n"
            + "ON L.Comboio = R.Comboio AND L.Nr = R.Lugar  \n"
            + "INNER JOIN Cliente AS C \n"
            + "ON C.ID_Cliente = R.Cliente \n"
            + "INNER JOIN Cidade AS Ci \n"
            + "ON Ci.ID_Cidade = C.Cidade \n"
            + "INNER JOIN País AS P \n"
            + "ON P.ID_País = Ci.País \n"
            + "INNER JOIN Viagem AS V \n"
            + "ON V.ID_Viagem = R.Viagem \n"
            + "INNER JOIN Cidade AS CP \n"
            + "ON CP.ID_Cidade = V.Cidade_Partida \n"
            + "INNER JOIN Cidade AS CC \n"
            + "ON CC.ID_Cidade = V.Cidade_Chegada \n"
            + "INNER JOIN País AS PP \n"
            + "ON CP.País = PP.ID_País\n"
            + "INNER JOIN País AS PC \n"
            + "ON CC.País = PC.ID_País \n");
        ResultSet rs = ps.executeQuery();

        MongoCollection<BasicDBObject>  collection  =  this.db.getCollection("Reserva",
BasicDBObject.class);

        while(rs.next()){
            data = rs.getDate("R.Data");
            codigo = rs.getString("R.Código");
            lugar = rs.getInt("L.Nr");
            classe = rs.getString("L.Classe");
            carruagem = rs.getInt("L.Carruagem");
            preco = rs.getDouble("R.Preço");

            BasicDBObject document = new BasicDBObject();
            document.put("Data", data);
            document.put("Código", codigo);
            document.put("Lugar", lugar);
            document.put("Classe",classe);
            document.put("Carruagem",carruagem);
            document.put("Preço",preco);

            nome = rs.getString("C.Nome");
```

```
        username = rs.getString("C.Username");
        password = rs.getString("C.Password");
        email = rs.getString("C.Email");
        telemovel = rs.getInt("C.Telemóvel");
        cidade = rs.getString("C.Nome");
        pais = rs.getString("P.Nome");

        BasicDBObject docCliente = new BasicDBObject();
        docCliente.put("Nome", nome);
        docCliente.put("Username", username);
        docCliente.put("Password", password);
        docCliente.put("Contactos",new BasicDBObject("Email", email)
                                .append("Telemovel",telemovel));
        docCliente.put("Endereço",new BasicDBObject("Cidade", cidade)
                               .append("Pais", pais));

        document.put("Cliente",docCliente);

        cidadeP = rs.getString("CP.Nome");
        cidadeC = rs.getString("CC.Nome");
        paisP = rs.getString("PP.Nome");
        paisC = rs.getString("PC.Nome");
        horaP = rs.getString("V.Hora_Partida");
        horaC = rs.getString("V.Hora_Chegada");
        duracao = rs.getString("V.Duração");

        BasicDBObject docViagem = new BasicDBObject();
        docViagem.put("País Partida",paisP);
        docViagem.put("Cidade Partida", cidadeP);
        docViagem.put("Hora Partida", horaP);
        docViagem.put("País Chegada",paisC);
        docViagem.put("Cidade Chegada", cidadeC);
        docViagem.put("Hora Chegada", horaC);
        docViagem.put("Duração",duracao);

        document.put("Viagem",docViagem);

        collection.insertOne(document);
    }
}
```

```
        catch(SQLException e){
          System.out.printf(e.getMessage());
        }
        finally{
           try{
             Connect.close(con);
           }
           catch(Exception e){
             System.out.printf(e.getMessage());
           }
        }
    }

    public void drop (){

        MongoCollection<BasicDBObject>      collection      =      this.db.getCollection("Reserva",
BasicDBObject.class);
        collection.drop();

        collection = this.db.getCollection("Cliente", BasicDBObject.class);
        collection.drop();

        collection = this.db.getCollection("Comboio", BasicDBObject.class);
        collection.drop();
    }
}
```