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

Unidade Curricular de Bases de Dados

Ano Letivo de 2018/2019

ECOmboios

Diogo Sobral a82523

Henrique Pereira a80261

Pedro Moreira a82364

Pedro Ferreira a81135

2018

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

ECOmboios

Diogo Sobral a82523

Henrique Pereira a80261

Pedro Moreira a82364

Pedro Ferreira a81135

2018

Resumo

Este relatório foi realizado no âmbito da Unidade Curricular de Base de Dados, descrevendo a criação de uma base de dados para uma empresa imaginária de transportes ferroviários, a ECOmboios. Escolhemos implementar esta base de dados por se tratar de uma situação real e que consegue ser bem fundamentada, como iremos concluir ao longo deste relatório.

A primeira parte do relatório diz respeito à definição do sistema, onde contextualizamos e fundamentamos a implementação da base de dados relacional, analisando também a sua viabilidade para a empresa ECOmboios.

Numa segunda fase, apresentamos os requisitos que por nós foram levantados e classificados como requisitos de descrição, exploração e controlo.

De seguida, apresentamos o modelo concetual para a base de dados que queríamos construir, tendo sempre em conta os requisitos levantados previamente. Aqui identificamos e classificamos as entidades e os relacionamentos existentes entre estas. Uma fase fulcral desta fase foi a validação do modelo de dados com o utilizador, pois permitiu-nos avançar para a fase seguinte: a modelação lógica.

Como foi referido, a quarta parte do relatório diz respeito à modelação lógica da base de dados para a ECOmboios. Nesta fase, apresentamos o modelo por nós desenvolvido. Além disso, validamos o mesmo através da normalização, com interrogações do utilizador e com as transações estabelecidas.

A quinta fase do projeto é referente à elaboração do modelo físico da base de dados, ou seja, é aqui que selecionamos o sistema de gestão da base de dados relacionais e traduzimos o esquema lógico para o sistema escolhido. Por fim, traduzimos as interrogações do utilizador e as transações em código SQL.

A última fase do projeto corresponde à análise, planeamento, e implementação de um SBD não relacional. Começamos por apresentar uma justificação à utilização deste tipo de sistemas, seguida da estrutura de dados definida. Por fim, explicamos o processo de migração dos dados e expomos as interrogações definidas sobre este sistema.

Área de Aplicação: Desenho e arquitectura de Sistemas de Bases de Dados

Palavras-Chave: Bases de Dados, Bases de Dados Relacionais, Levantamento de Requisitos, Modelo Concetual, Modelo Lógico, Modelo Físico, brModelo, MySQL Workbench, SQL, NoSQL, MongoDB

Índice

Índice de Figuras

Índice de Tabelas

1. Definição do Sistema

A empresa ECOmboios, responsável por efetuar serviços de transporte ferroviário, necessita de implementar um sistema de base de dados para suportar a compra de bilhetes para as viagens disponibilizadas. A base de dados deverá guardar e relacionar a informação sobre os serviços existentes de forma a que, a cada momento, seja possível consultar o estado da rede de transportes e obter uma perspetiva sobre a forma como o negócio se está a desenvolver. Com essa finalidade, o sistema deve estar munido de um conjunto de meios que permita saber, por exemplo, quantos bilhetes foram vendidos para uma determinada viagem, o valor faturado num determinado período, entre outras informações.

O desenvolvimento do sistema supramencionado será faseado em diversas etapas, nomeadamente: definição do sistema, levantamento e análise de requisitos, modelação conceptual, modelação lógica, implementação física, povoamento, e por fim, exploração e monitorização da informação armazenada.

Neste primeiro capítulo, procede-se à definição do sistema através da exposição do contexto da aplicação assim como da fundamentação para a sua implementação. Finalmente, é apresentada uma análise de viabilidade do processo.

1.1. Contexto da aplicação do sistema

“Chegar mais longe em menos tempo”, este é o lema da ECOmboios, uma das empresas com maior representatividade, a nível mundial, na prestação de transporte sobre linhas férreas. Fundada em 1985, inicialmente sob o nome de Ferroviária Ilimitada, a empresa ferroviária foi pioneira a realizar deslocações em comboios de alta velocidade, a nível europeu. Lançou-se no mercado com a ligação de alta-velocidade entre Paris e Berlim, reduzindo o tempo de viagem de 15 horas para apenas 7, sem que o conforto dos seus clientes fosse afetado. Além de procurar satisfação dos seus clientes, sempre foi bastante visionária e, já por várias vezes, demonstrou ter um apurado sentido de responsabilidade social, procurando reduzir a poluição causada pelos serviços que disponibiliza. Denota-se o facto de ser a única companhia ferroviária, a nível mundial, a utilizar apenas eletricidade renovável para deslocar os seus comboios, tendo sido também pioneira no abandono da queima de combustíveis fósseis para a obtenção de energia elétrica.

A exploração das vias férreas portuguesas encontrava-se até 2015 sobre monopólio da empresa CP, Comboios de Portugal. Nesse ano, procedeu-se à abertura do mercado deste setor, sendo que, rapidamente, se registaram várias propostas para a construção de novas linhas férreas e sua exploração. De entre essas propostas, destaca-se a oferta realizada pela ECOmboios, que procurava concretizar um desejo expresso há já algum tempo por bastantes utilizadores dos comboios: ser possível viajar cómoda e rapidamente entre Braga, Porto e Lisboa. De facto, a CP já possibilitava a conexão entre estas cidades. Contudo, a lentidão e o elevado número de fatores que decrementavam a qualidade da viagem, como por exemplo o elevado número de paragens efetuadas durante o percurso, afastavam potenciais interessados neste serviço para outros tipos de deslocação, como o autocarro ou o carro pessoal. Em meados de 2017, concluídas as obras de construção das linhas férreas e das centrais de produção de energia elétrica renovável, a ECOmboios iniciou a sua participação no mercado português.

Decorridos quase dois anos desde o inicio da sua atividade em Portugal, a empresa decidiu terminar com a venda física de bilhetes para as viagens, limitando este processo a um meio eletrónico, através de uma aplicação. Para suportar esta nova funcionalidade, necessita de implementar um Sistema de Base de Dados.

1.2. Fundamentação da implementação da base de dados

Com a necessidade de reduzir a poluição causada pelos veículos pessoais assim como o congestionamento provocado pelo uso deste meio de deslocação, a utilização dos transportes públicos tem vindo a ser fortemente fomentada no quotidiano dos portugueses. Na tentativa de induzir uma mudança de hábitos na forma de deslocação, são múltiplas e de variado tipo as campanhas publicitárias cujo destaque são os transportes públicos, entre os quais se encontra o comboio. Efetivamente, estas campanhas têm tido um grau de aceitação bastante elevado junto da população portuguesa, o que se reflete no aumento significativo do número de utilizadores de transportes públicos. Os serviços disponibilizados pela ECOmboios registaram aumentos na procura que variam entre os 15% e os 45%.

Com o objetivo de tornar as suas viagens ainda mais amigas do ambiente, e desta forma atrair ainda mais clientes, a empresa pretende passar a vender bilhetes exclusivamente online, sendo o respetivo bilhete validado mediante a sua apresentação num telemóvel, já a bordo do comboio. Esta medida representa um importante passo no combate aos incómodos associados à compra de bilhetes, verificados até então. Por outro lado, integra uma estratégia de marketing que procura cativar clientes que realizam as suas escolhas com base em critérios ambientais e/ou tecnológicos. Apresenta ainda uma vertente económica, visto que permitirá reduzir, justificadamente, o número de postos de trabalho mantidos pela empresa, em específico, os que estão associados à venda física de bilhetes, assim como eliminar os gastos inerentes à exploração e manutenção dos pontos de venda. Por fim, aliviará o tratamento dos dados, que passarão a ser analisados de forma eletrónica. Desta maneira, a implementação de um sistema de gestão de base de dados torna-se uma medida necessária para a empresa melhor controlar

a venda de bilhetes para as viagens de comboio e para processar e armazenar as informações relativas a estes. Os clientes poderão, então, por exemplo, consultar os bilhetes por si adquiridos e o dinheiro gasto em viagens, consultar as viagens que pode fazer consoante o seu orçamento e visualizar a origem, destino, duração, percurso, preço e data da viagem associada ao seu bilhete.

1.3. Análise da viabilidade do processo

A ECOmboios assume-se como uma entidade responsável e socialmente ativa. Como tal, não pode deixar que a sua obrigação para com os clientes e para com o mundo desapareça. No entanto, é, acima de tudo, uma empresa, e, por isso, todos os seus projetos têm de ser devidamente analisados e justificados, de modo a assegurar o seu futuro.

A análise da viabilidade deste projeto foca-se no estudo da quantidade de recursos humanos e financeiros necessários à implementação da base de dados, assim como num levantamento das vantagens económicas que esta proporcionará. Do ponto de vista financeiro, destaca-se o facto de que o custo inicial da implementação será rapidamente recuperado, visto que o novo sistema de compra de bilhetes levará a uma redução nas despesas com os funcionários, traduzindo-se assim em retorno do investimento. Por outro lado, não só reduzirá o número de erros humanos associados ao atendimento, como também permitirá que vários clientes efetuem a compra de bilhetes simultaneamente, eliminando as filas de espera que frequentemente se observavam no procedimento antigo. Deste modo, a aplicação poderá cativar ainda mais clientes. Certamente, os custos operacionais da base de dados não superarão os antigos custos do atendimento e a curto prazo não será necessário um segundo investimento. Do ponto de vista dos Recursos Humanos, é de referir que o povoamento inicial não deverá ser muito grande, uma vez que a empresa está a pensar em inserir dados gerados apenas após a implementação da base de dados, ou seja, não serão registados, por exemplo, bilhetes anteriores ao lançamento do projeto. Uma vez que o povoamento inicial será pequeno, o tempo requerido para o processo de implementação será diminuto e não mobilizará muitos recursos humanos. Após a implementação do sistema, também não será necessário um grande número de pessoas para a sua manutenção.

Desta maneira, conclui-se que a construção deste sistema é uma mais valia para a empresa, sendo totalmente viável.

2. Levantamento e Análise de Requisitos

No capítulo anterior foi definido o contexto em que a base de dados irá ser implementada. Finda a apresentação do caso, foi necessário compreender, em maior detalhe, que informação é que a base de dados deverá ser capaz de guardar. Para tal, procedemos ao levantamento de requisitos, recolhendo e analisando informação sobre o *modus operandi* da empresa e sobre a participação do cliente na compra de bilhetes.

2.1. Método de levantamento e de análise de requisitos adotado

No processo de levantamento de requisitos, recorremos aos seguintes métodos: análise de documentos, observação direta e entrevista a pessoas.

Quanto à análise de documentos, consultamos bilhetes em formato digital e em papel, filtrando tudo o que estava no bilhete e era considerado informação útil para o cliente e para a empresa.

Na parte relativa à observação, tivemos a oportunidade de nos deslocar a um dos balcões de atendimento na estação de Braga, com a finalidade de detalhar o seu funcionamento e a forma como os clientes são abordados quando desejam comprar um bilhete. Para além do que foi previamente enunciado, efetuamos uma viagem de Braga para o Porto, procurando obter uma visão mais concisa sobre o funcionamento do serviço de transporte.

Finalmente, procedemos à realização de uma entrevista a dois utilizadores assíduos das linhas ferroviárias sobre possíveis mudanças que estes gostariam de ver e de toda a informação que do seu ponto de vista é relevante.

2.2. Requisitos levantados

Através da análise da informação recolhida, procedemos à definição dos requisitos que a base de dados deverá suportar. Procuramos dotar a informação de uma maior estruturação, delimitando o papel de cada uma das componentes que formam o ambiente de negócio da empresa. Deste processo resultou a diferenciação entre o ponto de vista do cliente e a perspetiva do administrador da aplicação. De facto, a visão e objetivos que estes têm sobre a base de dados divergem.

2.2.1. Requisitos de descrição

Cliente

Os clientes têm de estar registados no sistema para poderem ser identificados e comprar bilhetes. Os dados guardados sobre cada cliente incluem o seu número, nome, email, o seu Número de Identificação Fiscal e a password que permite o acesso à aplicação. Estes dados, à exceção do número, são fornecidos ao sistema aquando do registo do cliente na aplicação. A cada cliente é atribuído um número que é único e o permite identificar. Tendo efetuado o log in na aplicação, o cliente pode comprar bilhetes a qualquer hora do dia desde que seja para uma viagem que ainda não tenha começado. Para realizar uma viagem, o cliente tem de ter adquirido bilhete.

Bilhete

Um bilhete é adquirido por um cliente que pretende realizar uma determinada viagem. Assim, a um bilhete está associado uma e uma só viagem, e apenas um cliente. Os dados associados a cada bilhete, para além dos já referidos, são o seu custo e o lugar (classe e número) que o cliente deverá ocupar no comboio. O custo é calculado através do preço base da viagem e da classe à qual pertence o lugar identificado. A bilhetes cujo lugar pertença à classe Económica é aplicado o preço base, enquanto que no caso de pertencer à classe Premium, é aplicado um extra de 50% sobre preço base. Quando a compra é efetuada até uma semana antes da data de início da viagem, é efetuado um desconto de 25%.

Viagem

Cada viagem guarda dados sobre a sua origem, hora de partida, destino, hora de chegada, duração, preço base e número de bilhetes comprados, sendo identificada por um número único. Uma viagem é realizada num determinado comboio o que faz com que o número máximo de bilhetes comprados seja intrínseco ao comboio no qual esta se efetuará.

Comboio

O tipo de comboios utilizado na prestação das viagens tem duas categorias: Económica e Premium. O mesmo comboio pode realizar várias viagens. A informação guardada para cada comboio passa pelo registo do seu número de identificação e pelos lugares existentes, identificados pela sua classe e número.

Estação

Uma viagem é efetuada entre duas estações, sendo que uma estação pode ser origem ou destino de várias viagens. Os dados guardados para estação são o seu nome e um número que permite identificá-la.

2.2.2. Requisitos de exploração

Do ponto de vista do cliente deve ser possível:

- (RE1) - Ver o histórico das suas viagens num dado período.
- (RE2) - Consultar o montante gasto num dado período.
- (RE3) - Visualizar a origem, destino, duração, preço, lugar e data da viagem associada ao seu bilhete.
- (RE4) - Pesquisar as viagens disponíveis num dado período entre duas estações.
- (RE5) - Consultar a lista de lugares livres para uma viagem.
- (RE6) - Consultar o horário específico duma determinada estação.
- (RE7) - Consultar lista de estações existentes.

Do ponto de vista do administrador da aplicação deve ser possível:

- (RE8) - Consultar as viagens realizadas por um determinado comboio num dado período.
- (RE9) - Saber quais os passageiros que viajaram entre duas estações num dado período.
- (RE10) - Saber quais os passageiros que participaram numa dada viagem.
- (RE11) - Verificar quantos bilhetes foram vendidos num determinado período.
- (RE12) - Calcular o valor total faturado num determinado período.

2.2.3. Requisitos de controlo

Do ponto de vista de um cliente, este deve conseguir:

- (RC1) - Inserir os dados para efetuar o registo.
- (RC2) - Inserir dados para a compra de um novo bilhete.
- (RC3) - Atualizar a informação inserida aquando do registo.

Do ponto de vista do administrador da aplicação, deve ser possível:

- (RC4) - Inserir uma nova viagem.
- (RC5) - Inserir dados sobre um novo comboio, assim como os lugares existentes neste.
- (RC6) - Adicionar novas estações.
- (RC7) - Atualizar os dados relativos a uma viagem.

2.3. Análise geral dos requisitos

Os requisitos de exploração registados revelam ser bastante descritivos do modo de funcionamento da ECOmboios, uma vez que englobam e descrevem todas as componentes que constituem o ambiente em que a empresa se encontra inserida e detalham as interações entre

estes. Com os requisitos de exploração e controlo torna-se evidente o papel de cada um dos utilizadores da base dados e o que se pretende com esta.

Podemos então concluir que os requisitos registados identificam corretamente os dados que serão guardados na base de dados e o que se pretende atingir com o sistema.

3. Modelo Conceptual

Uma vez concluída a definição dos requisitos e tendo já realizado uma análise detalhada dos mesmos, a próxima etapa da criação do sistema de base de dados da ECOmboios foi a modelação conceptual do mesmo. O modelo conceptual é independente dos detalhes de implementação e procura representar de forma fidedigna o modelo de informação usado pela empresa. Para facilitar a definição e interpretação do modelo, este é acompanhado de documentação, composta por um diagrama ER e um dicionário de dados sobre as entidades, relações e atributos identificados.

3.1. Apresentação da abordagem de modelação realizada

A abordagem de modelação utilizada na realização do modelo foi adaptada da metodologia de Connoly & Begg (2004), que orienta a modelação através de nove passos. Neste caso, o procedimento foi o seguinte:

1. Identificar os tipos de entidades existentes
2. Identificar os tipos de relacionamento
3. Identificar e associar atributos a cada tipo de entidade ou relacionamento
4. Determinar o domínio dos atributos
5. Determinar as chaves candidatas, primárias e estrangeiras
6. Considerar o uso de detalhe ou generalização de entidades
7. Verificar que o modelo conceptual suporta as transações necessárias
8. Validação do modelo de dados com o utilizador

Tendo isto em conta, foi possível definir o modelo conceptual da maneira que vamos apresentar nas secções que se seguem.

3.2. Identificação e caracterização das entidades

O primeiro passo efetuado na construção do modelo conceptual passou por definir os tipos de entidades existentes. Através de análise dos requisitos levantados conseguimos identificar as entidades necessárias. Desta forma, a primeira entidade identificada foi o Cliente, que será quem vai gerar a maior quantidade de dados a serem guardados na base de dados, efetuando compra de bilhetes online para as viagens que pretende realizar. Assim sendo, podemos estabelecer Bilhete e Viagem como entidades do modelo, servindo a primeira como forma de associação entre o cliente e uma ou mais ocorrências da segunda. Prosseguindo com a análise, surgem como entidades o Comboio e a Estação. De facto, o comboio proporciona a realização de uma viagem e, consequentemente, permite que o cliente atinja os seus objetivos. Por sua vez, a entidade Estação representa o local físico onde uma viagem tem início ou termina.

Findo o processo de identificação de entidades, pudemos sumariar os dados recolhidos na seguinte tabela:

Tabela 1 - Entidades

Entidade	Descrição	Aliases	Ocorrência
Cliente	Termo geral para descrever as pessoas registadas na aplicação da empresa.	Utilizador	Cada cliente pode ou não comprar bilhete(s) para uma ou mais viagens.
Bilhete	Termo geral que descreve os bilhetes comprados para as viagens existentes.	Reserva	Um bilhete é sempre associado a um cliente e a uma viagem. Para uma viagem podem ser comprados vários bilhetes, por diferentes clientes.
Viagem	Termo geral que detalha os serviços oferecidos pela empresa.	-	A uma viagem estão associados vários bilhetes.
Comboio	Termo geral referentes aos meios usados para realizar as viagens.	-	Um comboio tem vários lugares e realiza múltiplas viagens.
Estação	Termo geral representativo dos pontos físicos onde as viagens iniciam ou terminam.	-	Uma estação pode ser ponto de partida ou ponto de chegada de várias viagens.

3.3. Identificação e caracterização dos relacionamentos

No seguimento da construção do modelo conceptual, procedeu-se à identificação dos relacionamentos existentes entre as entidades previamente apontadas. Recorremos novamente aos requisitos registados para os estabelecer. Deste modo, enumeramos os seguintes relacionamentos:

Cliente – Bilhete

Este relacionamento representa a compra de um bilhete por parte de um cliente. O cliente pode optar por não comprar bilhetes para uma determina viagem, não participando nesta, ou então pode comprar um ou mais bilhetes. Cada bilhete comprado está sempre associado a um único cliente. Logo, estamos perante um relacionamento de 1 para N com participação opcional por parte do cliente e obrigatória do lado do bilhete.

Bilhete – Viagem

Cada bilhete é relativo a uma e uma só viagem (participação obrigatória), porém, para uma viagem, podem ser adquiridos nenhum ou vários bilhetes (participação opcional). Estamos, portanto, perante um relacionamento de N para 1.

Viagem – Estação

Uma viagem começa numa estação e termina noutra. Uma determinada estação pode ser a origem ou o destino de várias viagens. Ora, existem então dois relacionamentos entre as

entidades Viagem e Estação, ambos de N para 1. A participação de ambas as entidades nestes relacionamentos é obrigatória.

Viagem – Comboio

Cada viagem é feita por um comboio. Este pode efetuar várias viagens, desde que não sejam ao mesmo tempo. Assim, podemos afirmar que o relacionamento entre Viagem e Comboio é de N para 1. Do lado da viagem, a participação é obrigatória, pois uma viagem é sempre efetuada por um comboio. No entanto, um comboio pode não efetuar nenhuma viagem sendo a sua participação opcional.

Identificados os relacionamentos, obtivemos a seguinte tabela:

Tabela 2 - Relacionamentos

Entidade	Cardinalidade	Relacionamento	Cardinalidade	Entidade
Cliente	1..1	compra	0..n	Bilhete
Bilhete	0..n	relativo a	1..1	Viagem
Viagem	0..n	é feita por	1..1	Comboio
	1..n	origem	1..1	Estação
	1..n	destino	1..1	Estação

3.4. Identificação e caracterização das Associações dos Atributos com as Entidades e Relacionamentos

Para que as entidades identificadas pudessem efetivamente representar os objetos cujas informações se pretendem guardar no sistema, necessitavam de ser guarnecidas com os devidos atributos. Estes são facilmente identificados nos requisitos do sistema, onde a informação guardada para cada tipo de objeto foi suficientemente detalhada.

De seguida, apresenta-se o dicionário de dados relativo aos atributos estabelecidos:

Tabela 3 - Atributos

Entidade	Atributo	Descrição	Data Type	Null	Multivalorado	Derivado
Cliente	id	Identifica unicamente um cliente	INT	Não	Não	Não
	nome	Nome do cliente	VARCHAR(50)	Não	Não	Não
	email	Email do cliente	VARCHAR(30)	Não	Não	Não
	nif	NIF do cliente	INT	Não	Não	Não
	password	Password de acesso do cliente	VARCHAR(18)	Não	Não	Não
Bilhete	id	Identifica unicamente um bilhete	INT	Não	Não	Não

	preco	Preço do bilhete	FLOAT(5, 2)	Não	Não	Sim
	data_aquisicao	Data em que foi adquirido o bilhete	DATETIME	Não	Não	Não
	lugar					
	classe	Classe a que pertence o lugar	CHAR(1) (P ou E)		Não	Sim
	numero	Número do lugar	SMALLINT		Não	Sim
Viagem	id	Identifica unicamente uma viagem	INT	Não	Não	Não
	data_partida	Data de partida da viagem	DATETIME	Não	Não	Não
	data_chegada	Data de chegada da viagem	DATETIME	Não	Não	Não
	duracao	Tempo que demora a viagem	TIME	Não	Não	Sim
	preco_base	Preço base da viagem	FLOAT (5, 2)	Não	Não	Não
Estação	id	Identifica unicamente uma estação	INT	Não	Não	Não
	nome	Nome da estação	VARCHAR(20)	Não	Não	Não
Comboio	id	Identifica unicamente um comboio	INT	Não	Não	Não
	lugar				Sim	Não
	classe	Classe a que pertence o lugar	CHAR(1) (P ou E)	Não	Não	Não
	numero	Número do lugar	SMALLINT	Não	Não	Não

3.5. Detalhe ou generalização de entidades

Uma vez que as entidades identificadas representam objetos distintos com diferentes papéis no decorrer da atividade da empresa, não houve necessidade de proceder ao detalhe ou generalização das entidades. De facto, não é necessário utilizar a generalização, dado que os atributos associados a cada entidade variam, sendo que não existem duas entidades com um conjunto de atributos idênticos. Também não é necessário proceder ao detalhe de entidades, pois as ocorrências das entidades de um mesmo tipo terão caraterísticas semelhantes, acontecerão sobre contextos idênticos e, acima de tudo, representarão objetos também eles idênticos.

3.6. Apresentação e explicação do diagrama ER

Tendo definido as entidades, relacionamentos e atributos a utilizar, temos o seguinte diagrama ER:

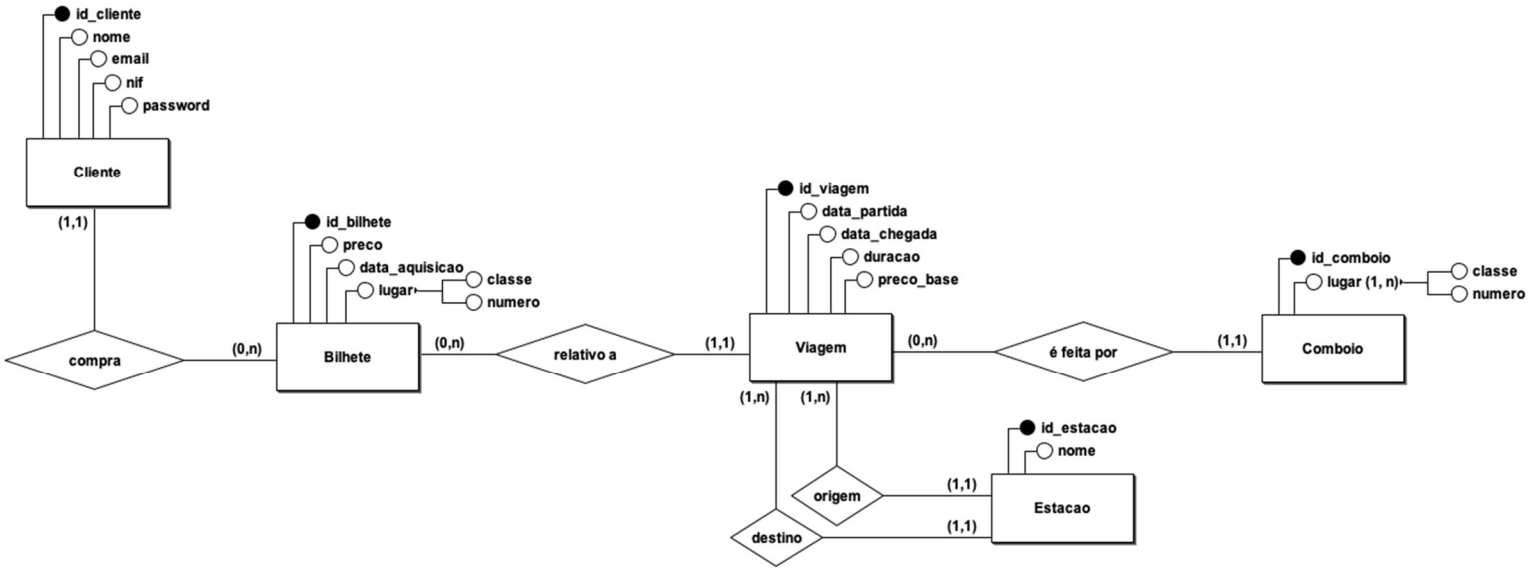

Figura 1 - Diagrama ER

Analisando o diagrama apresentado, constatamos que as entidades e os atributos que lhes estão associados se encontram em conformidade com os requisitos de exploração enunciados. Também os relacionamentos estabelecidos refletem o que é descrito nos requisitos. A justificação das multiplicidades encontra-se na secção 3.3. Assim sendo, resta explicar a escolha das chaves primárias de cada entidade.

Na entidade “Cliente” verifica-se a existência de três chaves candidatas: “id_cliente”, “email” e “nif”. Efetivamente, um cliente pode ser identificado pelo seu NIF, que é único, contudo a sua definição encontra-se fora do domínio do sistema e, em comparação com o atributo “id_cliente”, o seu valor numérico será certamente superior. Uma vez que por cada registo na aplicação é usado um email e este se encontra associado a uma só conta, poderíamos usar este atributo como chave da entidade “Cliente”, contudo o cliente pode alterar de email o que afasta este atributo da seleção para chave primária. Assim, definimos o “id_cliente” como chave primária. Na entidade “Bilhete” apenas o seu número identificador serve como chave candidata, sendo automaticamente selecionado para chave primária. Tal como a entidade “Bilhete”, uma “Viagem” apenas pode ser identificada pelo seu atributo “id_viagem”. Já na entidade “Estacao”, tanto “nome” como “id_estacao” são chaves candidatas. Escolhemos como chave primária “id_estacao” pois apresenta um caráter mais permanente e é definido pelo sistema. A empresa poderia, por algum motivo, alterar o nome da estação. Por fim, a entidade “Comboio” apenas pode ser identificada pelo seu atributo “id_comboio”.

3.7. Validação do modelo de dados com o utilizador

Uma vez concluído o processo de modelação concetual da base de dados, foi necessário validar o modelo de dados com o utilizador. Desta forma, foi realizada uma reunião com o corpo administrativo da ECOmboios. Nesse encontro, procurámos explicar, passo a passo, a maneira como seria possível satisfazer cada um dos requisitos, previamente aprovados pela empresa. Ou seja, no que toca ao cliente, este deveria poder consultar o histórico das viagens que realizou num dado período e o montante gasto no mesmo, visualizar as informações relativas a um bilhete seu, pesquisar as viagens disponíveis num dado período entre duas estações, consultar a lista de lugares livres para uma viagem e consultar a lista de estações existentes e os seus horários (com as viagens que têm como origem ou destino cada estação). Tendo isto em conta, na reunião explicitamos o seguinte:

- Para podermos consultar o histórico das viagens realizadas, necessitamos das entidades Viagem e Bilhete. Como temos que cada Bilhete é relativo a uma Viagem e que cada Bilhete é comprado por um Cliente, este pode consultar o histórico das viagens por si realizadas num dado período de tempo (cada Viagem tem uma data de partida e de chegada) consultando os bilhetes por si adquiridos para esse período.
- Como cada Cliente pode aceder aos bilhetes por si adquiridos, e como estes são relativos às viagens por si realizadas, é possível verificar qual o montante gasto num período de tempo somando os preços que estão associados a cada Bilhete desse Cliente entre as datas dadas.
- O Cliente pode consultar o preço e lugar de um Bilhete seu (pois são seus atributos) e a duração e as datas de chegada e partida são atributos da Viagem que está relacionada com o Bilhete referido. Uma vez que cada Viagem tem origem numa Estação e destino noutra, é possível consultar também, portanto, qual a origem e destino da Viagem à qual o Bilhete é relativo.
- Cada Viagem tem uma Estação como origem e outra como destino. Assim sendo, como cada Viagem tem uma data de chegada e de partida, é possível pesquisar as viagens existentes num dado período de tempo entre duas estações.
- Cada Viagem é feita por um Comboio e cada Comboio tem uma lista de lugares. Desta feita, como cada bilhete tem o lugar que lhe está reservado, é possível listar quais os lugares livres para uma dada Viagem, cruzando os lugares de um comboio com o lugar atribuído a cada um dos bilhetes relativos a essa Viagem. Assim, os lugares que ficarem de fora deste cruzamento são os lugares livres para a Viagem requerida.
- Como cada Viagem tem como origem uma Estação e como destino outra, é possível em cada Estação listar os horários das viagens (que têm uma data de partida e outra de chegada) que daí partem ou que lá chegam.
- Existindo uma entidade Estação, podemos consultar as estações existentes.

No que toca ao ponto de vista do administrador, demonstramos de que forma este seria capaz de consultar as viagens que um dado comboio realizou num certo período de tempo, saber quais os passageiros que viajaram entre duas estações num dado período ou numa dada viagem, verificar quantos bilhetes foram vendidos num certo intervalo de tempo e calcular o valor total faturado num determinado período. Assim sendo, o corpo administrativo da ECOmboios pediu para explicarmos como é que o nosso modelo concetual conseguia responder a tais requisitos. Para isso, afirmamos o seguinte:

- Como cada Viagem é realizada por um Comboio, e tem uma data de partida e uma data de chegada. Desta forma, é possível listar as viagens que um dado comboio realizou entre certas datas.
- Cada Bilhete é comprado por um Cliente e é relativo a uma Viagem. Ora, cada Viagem tem uma data de partida e outra de chegada e está associada a uma Estação de origem e a uma Estação de destino. Assim sendo, podemos saber quais os clientes que realizaram uma certa viagem entre duas estações e os que realizaram viagens entre essas estações num dado um período de tempo, através dos bilhetes comprados para tais viagens.
- Cada Bilhete tem uma data de aquisição, logo é possível verificarmos quantos bilhetes foram adquiridos entre duas datas dadas.
- Cada Bilhete tem um preço e uma data de aquisição. Desta maneira, podemos calcular o valor total faturado num dado período de tempo somando os valores do preço de todos os bilhetes adquiridos nesse período.

No fim da apresentação, o corpo administrativo da empresa ficou bastante satisfeito com as justificações dadas e validou o modelo de dados que apresentamos, uma vez que permite satisfazer todas as necessidades dos utilizadores relativas à base de dados da ECOmboios.

4. Modelação lógica

Após o modelo conceptual ter sido validado pelo cliente, prosseguimos à definição do modelo lógico do sistema. A sua construção foi orientada pelo modelo de dados relacional.

Neste capítulo, procede-se à descrição do processo de construção do modelo, sendo o resultado final apresentado de seguida. Sobre o resultado final são explicadas as devidas validações, nomeadamente: validação através da normalização, validação com as interrogações do utilizador e validação com as transações estabelecidas. Por fim, é sumariada a revisão do modelo obtido com o cliente.

4.1. Construção e validação do modelo de dados lógico

Procurando identificar e estabelecer as relações necessárias para que o modelo lógico fosse capaz de representar as entidades, relacionamentos e atributos necessários, procedemos a uma análise faseada do modelo conceptual, sendo que em cada etapa identificamos ocorrências de certos elementos no modelo, derivando de imediato a sua representação no modelo lógico. A ordem de identificação/derivação foi a seguinte:

1º. Entidades fortes.
2º. Entidades fracas.
3º. Relacionamentos binários de um-para-muitos.
4º. Relacionamentos de um-para-um.
5º. Relacionamentos recursivos de um-para-um
6º. Relacionamentos do tipo superclasse/subclasse
7º. Relacionamentos de muitos-para-muitos
8º. Relacionamentos complexos
9º. Atributos multivalorados.

Em relação à primeira etapa, concluímos que todas as entidades representadas no modelo são entidades fortes. Assim, para cada uma delas, criamos uma relação que incluísse todos os atributos simples dessa entidade, identificando ainda a chaves primária e eventuais chaves alternativas. Deste modo, temos:

- **Cliente** (id_cliente, nome, email, nif, password)
  Chave Primária - id_cliente
  Chave Alternativa – nif, email

- **Bilhete** (id_bilhete, preco, data_aquisicao, classe, numero)
  Chave Primária - id_bilhete

- **Viagem** (id_viagem, data_partida, data_chegada, duração, preco_base)
  Chave Primária - id_viagem

- **Estacao** (id_estacao, nome)
  Chave Primária - id_estacao

- **Comboio** (id_comboio)
  Chave Primária - id_comboio

Dado o facto de todas as entidades serem fortes, na segunda etapa do processo não foram derivadas novas relações. Já no terceiro passo, identificamos cinco relacionamentos de um-para-muitos. Este tipo de relacionamentos dá origem a uma relação da qual faz parte a chave primária da entidade que se encontra do lado do um e os restantes atributos simples da entidade que se posiciona do lado do N. É de notar que as entidades "Bilhete" e "Viagem" participam em vários relacionamentos deste tipo. Colocando várias chaves estrangeiras na mesma relação conseguimos derivar duas relações que cobrem os cinco relacionamentos existentes. Estas são:

- **Bilhete** (id_bilhete, preco, data_aquisicao, classe, numero, cliente, viagem)
  Chave Primária - id_bilhete
  Chave Estrangeira - cliente **referencia** Cliente(id_cliente)
  Chave Estrangeira - viagem **referencia** Viagem(id_viagem)

- **Viagem** (id_viagem, data_partida, data_chegada, duração, preco_base, comboio, origem, destino)
  Chave Primária - id_viagem
  Chave Estrangeira - comboio **referencia** Comboio(id_comboio)
  Chave Estrangeira - origem **referencia** Estacao(id_estacao)
  Chave Estrangeira - destino **referencia** Estacao(id_estacao)

Até ao nono passo, não foi derivada mais nenhuma relação pois nenhum dos outros tipos de relacionamentos enumerados se encontram presentes no modelo conceptual. No entanto, é possível identificar um atributo multivalorado na entidade Comboio: o lugar. Este é composto pela classe e pelo número do lugar. Assim, conseguimos derivar mais uma relação, representativa dos lugares do comboio. Uma vez que a mesma associação lugar-classe se encontra repetida em vários comboios, um lugar só pode ser identificado unicamente se soubermos qual o comboio a que pertence. Dito isto, podemos afirmar que a chave primária desta nova relação é uma chave composta. De facto, temos:

- **Lugar** (classe, numero, comboio)
  Chave Primária – (classe, numero, comboio)
  Chave Estrangeira - comboio **referencia** Comboio(id_comboio)

4.2. Desenho do modelo lógico

Com as relações derivadas no ponto anterior, surgiu o seguinte esquema para o modelo lógico:

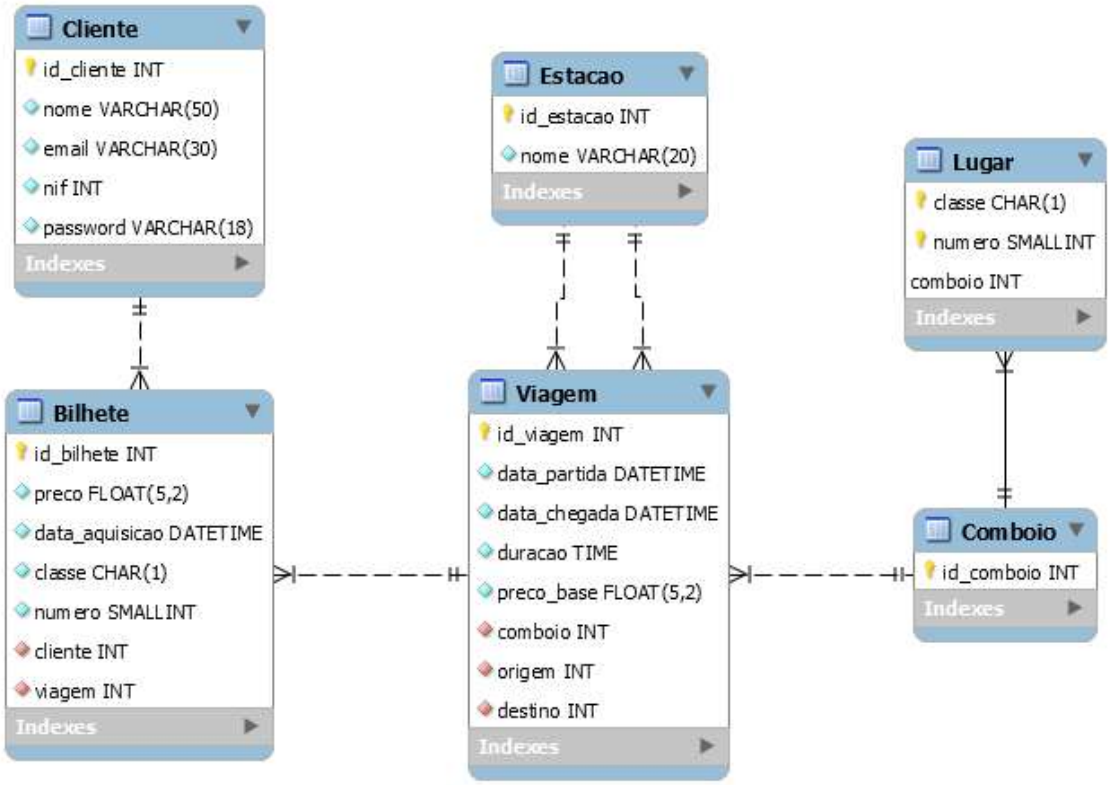

Figura 2 - Modelo lógico

4.3. Validação do modelo através da normalização

A normalização é muito importante no modelo relacional de uma base de dados, uma vez que garante que as relações tenham um número mínimo, mas necessário, de atributos, de modo a que todos os requisitos de dados da empresa sejam suportados. Além disso, permite que a redundância de dados seja a mínima possível.

Desta forma, procedemos à verificação do modelo através da normalização, ou seja, verificamos se o modelo estava de acordo com a 1ª Forma Normal, a 2ª Forma Normal e a 3ª Forma Normal:

1. Todas as relações estão na 1ª Forma Normal, pois nenhuma apresenta atributos multivalor nem grupos repetidos, ou seja, a interseção de cada linha e cada coluna apresenta apenas um único valor.
2. Todas as relações estão na 1ª Forma Normal e todos os seus atributos são totalmente dependentes da chave primária (não há dependências parciais). Como apenas existe uma chave primária composta (na tabela Lugar) e não existem atributos nessa tabela apenas dependentes de uma das chaves primárias, o modelo respeita a 2ª Forma Formal.

3. Todas as relações estão na 1ª e 2ª Forma Normal e nenhum dos seus atributos apresenta dependências transitivas. Como nenhum atributo de nenhuma relação depende de outro atributo que não seja a chave primária, o modelo respeita a 3ª Forma Normal.

Em suma, como todas as relações estão de acordo com a 3ª Forma Normal, concluímos que o modelo está normalizado.

4.4. Validação do modelo com interrogações do utilizador

Depois da construção e da normalização do modelo lógico, tivemos que verificar se o modelo que construímos conseguia responder aos requisitos do utilizador e, por isso, recorremos à validação através de interrogações por parte do mesmo.

Assim sendo, no que toca aos clientes da ECOmboios, demonstramos que o modelo lógico consegue responder às questões apresentadas da seguinte forma:

- Posso ver o histórico das minhas viagens num dado período?

Inicialmente, selecionamos da tabela dos clientes o Cliente que pretendemos e juntamos o resultado com a tabela dos bilhetes. De seguida, selecionamos da tabela das viagens apenas aquelas que estão no intervalo que pretendemos e, por fim, juntamos o resultado desta etapa com o da primeira, obtendo desta forma o histórico das viagens dum cliente num dado período.

- Posso consultar o montante gasto num intervalo de tempo?

Para responder a esta interrogação, efetuamos, inicialmente, o mesmo processo que foi executado para a primeira interrogação. Após já termos a junção das tabelas iniciais, somamos o valor do preço dos bilhetes desse cliente, obtendo assim o montante gasto num intervalo de tempo.

- Posso visualizar a origem, destino, duração, preço, lugar e data da viagem associada ao meu bilhete?

Em primeiro lugar, vamos à tabela dos bilhetes e selecionamos o bilhete que queremos. Juntando o resultado com a tabela das viagens e depois com a tabela das estações, podemos obter as informações que pretendemos, ou seja, a origem, destino, duração, preço, lugar e data da viagem associados ao bilhete escolhido.

- **Posso pesquisar as viagens disponíveis num dado período entre duas estações?**

Para tal, começamos por selecionar da tabela das viagens apenas as que estão no período pretendido. Da tabela das estações, selecionamos as duas que queremos. Por fim, juntamos as duas tabelas resultantes, obtendo, desta forma, as viagens disponíveis num dado período entre duas estações.

- **Posso consultar a lista de lugares livres para uma viagem?**

Para consultar a lista de lugares livres para uma viagem, selecionamos, da tabela das viagens, a viagem que pretendemos. De seguida, juntamos a tabela resultante com a tabela dos comboios e, logo a seguir, juntamos com a tabela dos lugares do comboio. Da tabela resultante, ficamos apenas com os lugares e subtraímos a tabela conseguinte de juntar a viagem com o bilhete, ficando, assim, só com os lugares livres para a viagem pretendida.

- Posso consultar o horário específico duma determinada estação?

Inicialmente, selecionamos a estação que queremos da tabela das estações e juntamos com a tabela das viagens. Da tabela resultante, selecionamos as viagens com a data que pretendemos.

- Posso consultar a lista de estações existentes?

Para tal, basta consultar a tabela das estações.

Do ponto de vista do administrador da aplicação, surgiram as seguintes questões, às quais respondemos da forma que segue:

- Posso consultar as viagens realizadas por um determinado comboio num dado período?

Começamos por ir à tabela dos comboios e selecionamos o que pretendemos. Na tabela das viagens, selecionamos apenas as que estão no intervalo que queremos. Por fim, juntamos as duas tabelas resultantes, resultando esta junção na lista de viagens realizadas por o comboio escolhido no período selecionado.

- Posso saber quais os passageiros que viajaram entre duas estações num dado período?

Na tabela das estações, selecionamos apenas as estações que queremos. De seguida, selecionamos da tabela das viagens apenas as que estão no intervalo pretendido. Juntamos as duas tabelas resultantes. Após este processo, juntamos a tabela com a dos bilhetes e depois com a dos clientes, sabendo assim quais os passageiros que viajaram entre as duas estações num dado período.

- Posso saber quais os passageiros que participaram numa dada viagem?

Na tabela das viagens, selecionamos a viagem que queremos. Juntando com a tabela dos bilhetes e depois com a dos clientes, obtemos os passageiros que participaram na viagem escolhida.

- Posso verificar quantos bilhetes foram vendidos num determinado período?

Para isto, selecionamos na tabela dos bilhetes todos aqueles cuja data de aquisição está no intervalo pretendido e contamos quantos são.

4.5. Validação do modelo com as transações estabelecidas

Uma vez validado o modelo lógico com interrogações do utilizador, chegou o momento de o validar com as transações estabelecidas. Estas transações dizem respeito aos requisitos de controlo por nós recolhidos. Assim sendo, temos que o modelo consegue responder a cada um deles, relativos ao cliente, uma vez que:

- podemos inserir os dados de um cliente para efetuar o registo, pois temos uma tabela de Clientes, onde são guardadas as suas informações (ID, nome, email, NIF e password);
- podemos inserir dados para compra de um novo bilhete, dado que a tabela dos Bilhetes apresenta como atributo e chave estrangeira o ID do cliente que o adquiriu (existindo, portanto, uma relação entre as duas tabelas);
- as informações do cliente podem ser atualizadas, pois estão armazenadas na tabela dos Clientes.

Relativamente às transações que dizem respeito ao administrador, temos que:

- podemos inserir uma nova viagem, pois estas estão representadas na tabela Viagem;
- podemos inserir dados sobre um novo comboio, uma vez que temos uma tabela para os Comboios. Os lugares deste podem também ser introduzidos, dado que existe uma tabela Lugar que está diretamente relacionada com a tabela Comboio, sendo o ID deste uma chave primária da dos Lugares;
- podemos adicionar novas estações, pois temos uma tabela Estação, onde armazenamos as suas informações;
- podemos atualizar os dados relativos a uma viagem, visto que temos uma tabela para as Viagens, estando esta associada à dos Comboios e à das Estações.

Desta maneira, pudemos concluir que o modelo lógico desenvolvido é válido, tendo em conta as transações estabelecidas.

4.6. Revisão do modelo lógico com o utilizador

Na fase final da modelação lógica da base de dados, foi necessário validá-la com o corpo administrativo da ECOmboios. Assim sendo, foi marcada mais uma reunião, onde apresentamos e justificamos a estrutura que desenvolvemos.

Por fim, a empresa validou o modelo lógico, pois este respondia a todos os requisitos da aplicação.

5. Implementação Física

A fase final do processo de construção do sistema de base dados passou pela sua implementação física, seguida do povoamento, exploração e monitorização da informação armazenada. Começamos por traduzir o esquema lógico, apresentado no capítulo anterior, para o SGBD escolhido. Uma vez implementado o esquema físico, procedemos à tradução das interrogações do utilizador e das transações estabelecidas para SQL. De seguida, definimos índices e vistas de utilização. Por fim, implementamos mecanismos de segurança com o objetivo de restringir o acesso dos utilizadores a certas partes da base de dados.

Neste capítulo, para além de uma descrição detalhada de cada um dos processos referidos anteriormente, é também apresentada uma estimativa do espaço que a base de dados ocupará em disco e é sumariada a revisão do sistema implementado junto da empresa ECOmboios.

5.1. Seleção do sistema de gestão de bases de dados

A fim de desenvolver o esquema físico da base de dados tivemos, inicialmente, de escolher qual o SGBD a utilizar. Uma vez que nos encontramos no domínio do modelo de dados relacional, optamos por usar o *MySQL* pois este disponibiliza mecanismos de controlo de concorrência, nomeadamente as transações, permitindo eliminar aquilo que seria um grave problema no contexto da aplicação: vários clientes comprarem simultaneamente bilhete para o mesmo lugar na mesma viagem. A favor da sua escolha encontra-se também o facto do MySQL permitir a definição de perfis de utilização, útil para delimitar o acesso de cada tipo de utilizador a apenas uma parte da base dados que seja suficiente para satisfazer os seus requisitos.

5.2. Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL

No processo de construção do esquema físico começamos por identificar as relações base, produzidas aquando da definição do esquema lógico. Desta forma, temos:

- **Relação Cliente**

Domínio ID Cliente:	inteiro
Domínio Nome:	string de comprimento variável, comprimento 50
Domínio Email:	string de comprimento variável, comprimento 30
Domínio Número Identificação Fiscal:	inteiro
Domínio Password:	string de comprimento variável, comprimento 18

Cliente(

id_cliente	ID Cliente	NOT NULL,
nome	Nome	NOT NULL,
email	Email	NOT NULL,
nif	Número Identificação Fiscal	NOT NULL,
password	Password	NOT NULL,

CHAVE PRIMÁRIA (id_cliente)
CHAVE ALTERNATIVA (nif)
CHAVE ALTERNATIVA (email));

Transpondo para código SQL, temos:

```
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Cliente` (
  `id_cliente` INT NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(50) NOT NULL,
  `email` VARCHAR(30) NOT NULL,
  `nif` INT NOT NULL,
  `password` VARCHAR(18) NOT NULL,
  UNIQUE (`email`),
  UNIQUE (`nif`),
  PRIMARY KEY (`id_cliente`))
ENGINE = InnoDB;
```

Figura 3 - Código SQL para criação da tabela Cliente

- **Relação Bilhete**

Domínio ID Bilhete: inteiro
Domínio Preço: valor monetário, entre 000.00 – 999.99
Domínio Data Aquisição: temporal
Domínio Classe: caracter (‘P’ ou ‘E’)
Domínio Número: inteiro
Domínio Cliente: inteiro
Domínio Viagem: inteiro

Bilhete(

id_bilhete	ID Bilhete	NOT NULL,
preco	Preço	NOT NULL,
data_aquisicao	Data Aquisição	NOT NULL,
classe	Classe	NOT NULL,
numero	Número	NOT NULL,
cliente	Cliente	NOT NULL,
viagem	Viagem	NOT NULL,

CHAVE PRIMÁRIA (id_bilhete),
CHAVE ESTRANGEIRA (cliente) REFERENCIA Cliente(id_cliente)
ON UPDATE CASCADE
ON DELETE NO ACTION,
CHAVE ESTRANGEIRA (viagem) REFERENCIA Viagem(id_viagem)
ON UPDATE CASCADE
ON DELETE NO ACTION);

Transpondo para código SQL, temos:

```
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Bilhete` (
  `id_bilhete` INT NOT NULL AUTO_INCREMENT,
  `preco` FLOAT(5,2) NOT NULL,
  `data_aquisicao` DATETIME NOT NULL,
  `classe` CHAR(1) NOT NULL,
  `numero` SMALLINT NOT NULL,
  `cliente` INT NOT NULL,
  `viagem` INT NOT NULL,
  PRIMARY KEY (`id_bilhete`),
  INDEX `fk_Bilhete_Cliente1_idx` (`cliente` ASC),
  INDEX `fk_Bilhete_Viagem1_idx` (`viagem` ASC),
  CONSTRAINT `fk_Bilhete_Cliente1`
    FOREIGN KEY (`cliente`)
    REFERENCES `ECOmboios`.`Cliente` (`id_cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Bilhete_Viagem1`
    FOREIGN KEY (`viagem`)
    REFERENCES `ECOmboios`.`Viagem` (`id_viagem`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 4 - Código SQL para criação da tabela Bilhete

- **Relação Viagem**

Domínio ID Viagem: inteiro
Domínio Data Partida: temporal
Domínio Data Chegada: temporal
Domínio Duração: temporal
Domínio Preço Base: valor monetário, entre 000.00 – 999.99
Domínio Comboio: inteiro
Domínio Origem: inteiro
Domínio Destino: inteiro

Viagem(

id_viagem	ID Viagem	NOT NULL,
data_partida	Data Partida	NOT NULL,
data_chegada	Data Chegada	NOT NULL,
duração	Duração	NOT NULL,
preco_base	Preço Base	NOT NULL,
comboio	Comboio	NOT NULL,
origem	Origem	NOT NULL,
destino	Destino	NOT NULL,

CHAVE PRIMÁRIA (id_viagem),
CHAVE ESTRANGEIRA (comboio) REFERENCIA Comboio(id_comboio)
ON UPDATE CASCADE
ON DELETE NO ACTION,
CHAVE ESTRANGEIRA (origem) REFERENCIA Estacao(id_estacao)
ON UPDATE CASCADE
ON DELETE NO ACTION,

CHAVE ESTRANGEIRA (destino) REFERENCIA Estacao(id_estacao)
ON UPDATE CASCADE
ON DELETE NO ACTION);

Transpondo para código SQL, temos:

```
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Viagem` (
  `id_viagem` INT NOT NULL AUTO_INCREMENT,
  `data_partida` DATETIME NOT NULL,
  `data_chegada` DATETIME NOT NULL,
  `duracao` TIME NOT NULL,
  `preco_base` FLOAT(5,2) NOT NULL,
  `comboio` INT NOT NULL,
  `origem` INT NOT NULL,
  `destino` INT NOT NULL,
  PRIMARY KEY (`id_viagem`),
  INDEX `fk_Viagem_Comboio1_idx` (`comboio` ASC),
  INDEX `fk_Viagem_Estacao1_idx` (`origem` ASC),
  INDEX `fk_Viagem_Estacao2_idx` (`destino` ASC),
  CONSTRAINT `fk_Viagem_Comboio1`
    FOREIGN KEY (`comboio`)
    REFERENCES `ECOmboios`.`Comboio` (`id_comboio`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Estacao1`
    FOREIGN KEY (`origem`)
    REFERENCES `ECOmboios`.`Estacao` (`id_estacao`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Estacao2`
    FOREIGN KEY (`destino`)
    REFERENCES `ECOmboios`.`Estacao` (`id_estacao`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 5 - Código SQL para a criação da tabela Viagem

- **Relação Estação**

Domínio ID Estação inteiro
Domínio Nome string de comprimento variável, comprimento 20
Estacao(
id_estacao ID Estação NOT NULL,
nome Nome NOT NULL,
CHAVE PRIMÁRIA (id_estacao)
CHAVE ALTERNATIVA(nome));

Transpondo para código SQL, temos:

```
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Estacao` (
 `id_estacao` INT NOT NULL AUTO_INCREMENT,
 `nome` VARCHAR(20) NOT NULL,
 UNIQUE (`nome`),
 PRIMARY KEY (`id_estacao`))
ENGINE = InnoDB;
```

Figura 6 - Código SQL para a criação da tabela Estacao

- **Relação Comboio**

Domínio ID Comboio inteiro
Comboio(
id_comboio ID Comboio NOT NULL,
CHAVE PRIMÁRIA (id_comboio));

Transpondo para código SQL, temos:

```
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Comboio` (
  `id_comboio` INT NOT NULL AUTO_INCREMENT,
  PRIMARY KEY (`id_comboio`))
ENGINE = InnoDB;
```

Figura 7 - Código SQL para a criação da tabela Comboio

- **Relação Lugar**

Domínio Classe: carater ('P' ou 'E')
Domínio Numero: inteiro
Domínio Comboio: inteiro
Lugar(
classe Classe NOT NULL,
numero Número NOT NULL,
comboio Comboio NOT NULL,
CHAVE PRIMÁRIA (classe, numero, comboio),
CHAVE ESTRANGEIRA (comboio) REFERENCIA Comboio(id_comboio)
ON UPDATE CASCADE
ON DELETE NO ACTION);

Transpondo para código SQL, temos:

```sql
CREATE TABLE IF NOT EXISTS `ECOmboios`.`Lugar` (
  `classe` CHAR(1) NOT NULL,
  `numero` SMALLINT NOT NULL,
  `comboio` INT NOT NULL,
  INDEX `fk_Lugar_Comboio1_idx` (`comboio` ASC),
  PRIMARY KEY (`numero`, `comboio`, `classe`),
  CONSTRAINT `fk_Lugar_Comboio1`
    FOREIGN KEY (`comboio`)
    REFERENCES `ECOmboios`.`Comboio` (`id_comboio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 8 - Código SQL para a criação da tabela Lugar

Após a identificação das relações base procedemos à representação dos atributos derivados definindo a forma de os calcular. Em conformidade com os requisitos levantados, temos que o preço de um bilhete é calculado com através do preço base da viagem, ao qual pode ser aplicado uma taxa extra se o lugar pertencer à classe Premium, e um desconto, caso seja comprado uma semana antes do inicio da viagem. Para calcular o preço do bilhete, desenvolvemos o seguinte *trigger*:

```sql
DELIMITER $$
CREATE TRIGGER preco_bilhete BEFORE INSERT ON bilhete
FOR EACH ROW
BEGIN
    DECLARE preco FLOAT(5, 2);
    SET preco = (SELECT preco_base FROM viagem WHERE id_viagem = new.viagem);
    -- Verifica se é Premium
    IF new.classe = 'P'
    THEN
        SET preco = preco * 1.5;
    END IF;

    -- Verifica se adquiriu uma semana antes da partida
    IF new.data_aquisicao < date_sub((SELECT data_partida FROM viagem WHERE id_viagem = new.viagem), INTERVAL 1 WEEK)
    THEN
        SET preco = preco * 0.75;
    END IF;

    -- Regista preco
    SET new.preco = preco;

END $$
```

Figura 9 - Código SQL para calcular o preço de um bilhete

Por fim, para calcular a duração de uma viagem, estabelecemos o seguinte *trigger*:

```sql
DELIMITER $$
CREATE TRIGGER duracao BEFORE INSERT ON viagem
FOR EACH ROW
BEGIN
    SET new.duracao = TIMEDIFF(new.data_chegada, new.data_partida);
END $$
```

Figura 10 - Código SQL para o cálculo da duração de uma viagem

5.3. Tradução das interrogações do utilizador para SQL

Nesta secção serão apresentados alguns exemplos de código SQL que permitem obter respostas às interrogações do utilizador, expressas pelos requisitos de exploração.

- **(RE1) – Consultar o histórico de viagens realizadas por um cliente num certo período**

```
DELIMITER $$
CREATE PROCEDURE historico_viagens(IN id_cliente INT, data_inicio DATETIME, data_fim DATETIME)
BEGIN
    SELECT v.data_partida AS 'Data de Partida', eo.nome AS 'Origem', v.data_chegada  AS 'Data de Chegada', ed.nome AS 'Destino'
    FROM bilhete AS b INNER JOIN viagem AS v
                      ON b.viagem = v.id_viagem
                            INNER JOIN estacao AS eo
                            ON v.origem = eo.id_estacao
                            INNER JOIN estacao AS ed
                            ON v.destino = ed.id_estacao
    WHERE b.cliente = id_cliente AND v.data_partida >= data_inicio AND v.data_chegada <= data_fim
    ORDER BY v.data_partida ASC;
END $$
```

Figura 11 - Código SQL para consultar o histórico de viagens realizadas por um cliente num certo período

- **(RE2) – Consultar o montante gasto por um cliente num dado período**

```
DELIMITER $$
CREATE FUNCTION total_gasto(id_cliente INT, data_inicio DATETIME, data_fim DATETIME)
       RETURNS FLOAT DETERMINISTIC
BEGIN
    RETURN (SELECT sum(b.preco)
            FROM bilhete as b
            WHERE b.cliente = id_cliente AND b.data_aquisicao >= data_inicio AND b.data_aquisicao <= data_fim);

END $$
```

Figura 12 - Código SQL para consultar o montante total gasto por um cliente num dado período

- **(RE4) – Pesquisar as viagens existentes entre duas estações num certo período**

```
DELIMITER $$
CREATE PROCEDURE viagens_between(IN id_estacao_o INT, id_estacao_d INT, data_inicio DATETIME, data_fim DATETIME)
BEGIN
   SELECT v.id_viagem AS 'ID', v.data_partida AS 'Data de Partida', v.data_chegada AS 'Data de Chegada', v.duracao AS 'Duração'
   FROM viagem AS v
   WHERE v.data_partida > data_inicio AND v.data_chegada < data_fim
      AND v.origem = id_estacao_o AND v.destino = id_estacao_d;
END $$
```

Figura 13 - Código SQL para consultar as viagens existentes entre duas estações num certo período

- **(RE5) - Consultar a lista de lugares livres para uma viagem**

```
DELIMITER $$
CREATE PROCEDURE lugares_livres(IN id_viagem INT)
BEGIN
   SELECT l.classe AS 'Classe', l.numero AS 'Lugar'
   FROM Viagem AS v INNER JOIN Comboio AS c
            ON v.comboio = c.id_comboio
              INNER JOIN Lugar AS l
              ON c.id_comboio = l.comboio
   WHERE (l.classe, l.numero) NOT IN (
      SELECT l.classe, l.numero
      FROM Viagem AS v
          INNER JOIN Bilhete as b
          ON v.id_viagem = viagem
             INNER JOIN lugar AS l
             ON (b.numero = l.numero AND b.classe = l.classe))
       AND v.id_viagem = id_viagem;
END $$
```

Figura 14 - Código SQL para saber os lugares livres numa viagem

- (**RE9) – Saber que clientes viajaram entre duas estações num certo período**

```
DELIMITER $$
CREATE PROCEDURE clientes_between_estacoes(IN id_estacao_o INT, id_estacao_d INT, data_inicio DATETIME, data_fim DATETIME)
BEGIN
   SELECT c.id_cliente, c.nome, count(c.id_cliente) AS 'Número de bilhetes'
   FROM cliente AS c INNER JOIN bilhete AS b
            ON c.id_cliente = b.cliente
              INNER JOIN viagem AS v
              ON b.viagem = v.id_viagem
   WHERE v.data_partida > data_inicio AND v.data_chegada < data_fim
       AND v.origem = id_estacao_o AND v.destino = id_estacao_d
   GROUP BY c.id_cliente;
END $$
```

Figura 15 - Código SQL para saber que clientes viajaram entre duas estações num dado período

- **(RE10) – Saber que clientes participaram numa determinada viagem**

```
DELIMITER $$
CREATE PROCEDURE clientes_na_viagem(IN id_viagemp INT)
BEGIN
   SELECT c.nome AS Nome, c.id_cliente AS ID
      FROM Bilhete b, Cliente c
      WHERE b.viagem = id_viagemp
         AND b.cliente = c.id_cliente;
END $$
```

Figura 16 - Código SQL para saber que clientes participaram numa determinada viagem

- **(RE11) – Calcular o total faturado num determinado período**

```
DELIMITER $$
CREATE FUNCTION total_faturado(d_inicio DATETIME, d_fim DATETIME)
    RETURNS FLOAT DETERMINISTIC
BEGIN
   RETURN (SELECT sum(b.preco)
        FROM bilhete as b
        WHERE b.data_aquisicao >= d_inicio
            AND b.data_aquisicao <= d_fim);
END $$
```

Figura 17 - Código SQL para calcular o total faturado num determinado período

5.4. Tradução das transações estabelecidas para SQL

Tendo em conta as transações estabelecidas, definimos código em SQL para as executar. De seguida, apresentam-se alguns exemplos de transações.

- **(RC1) – Registar um novo cliente**

```
DELIMITER $$
CREATE PROCEDURE adiciona_cliente(IN nome VARCHAR(50), email VARCHAR(30), nif INT, password VARCHAR(18))
BEGIN
   INSERT INTO cliente(nome, email, nif, password)
   VALUES (nome, email, nif, password);
END $$
```

Figura 18 - Código SQL para registar um novo cliente

- **(RC2) – Inserir dados para a compra de um novo bilhete**

```
DELIMITER $$
CREATE PROCEDURE adiciona_bilhete(IN id_cliente INT, classe CHAR(1), numero INT, id_viagem INT)
BEGIN
   DECLARE r INT;
   SET AUTOCOMMIT = OFF;

   START TRANSACTION READ WRITE;
      SET r = lugar_livre(classe, numero, id_viagem);
      IF (r = 1)
      THEN
         INSERT INTO bilhete(data_aquisicao, classe, numero, cliente, viagem)
         VALUES (now(), classe, numero, id_cliente, id_viagem);
      END IF;
   COMMIT;
END $$
```

Figura 19 - Código SQL para adicionar um bilhete à base de dados

- **(RC5) – Adicionar lugares a um comboio**

```
DELIMITER $$
CREATE PROCEDURE adiciona_lugares(IN id_comboio INT)
BEGIN
    DECLARE i INT DEFAULT 1;

    WHILE (i <= 50) DO
        INSERT INTO lugar(classe, numero, comboio)
        VALUES ('P', i, id_comboio), ('E', i, id_comboio);
        SET i = i+1;
    END WHILE;

    WHILE (i <= 200) DO
        INSERT INTO lugar(classe, numero, comboio)
        VALUES ('E', i, id_comboio);
        SET i = i+1;
    END WHILE;
END $$
```

Figura 20 - Código SQL para adicionar lugares a um comboio

5.5. Escolha, definição e caracterização de índices em SQL

Uma vez respondidas as interrogações do nosso cliente, é necessário pensar que, para uma empresa como a ECOmboios, não basta qualquer tipo de solução. A rapidez com que os clientes conseguem aceder à informação tem um papel central no sistema desenvolvido, sendo que quanto mais rápido este for, mais apelativo será. Nesse sentido, decidimos estudar possíveis maneiras de aumentar a velocidade da queries, através do uso de índices. Após análise cuidada do problema, pudemos constatar que grande parte dos objetivos que foram traçados para a base de dados, passavam por só considerar valores em determinados intervalos de tempo. No entanto, não tínhamos nada que ajudasse a chegar rapidamente a estes valores.

A tabela que mais vezes é consultada quando são precisos intervalos de tempo é a das viagens. Assim sendo, decidimos que as colunas que guardam a data de chegada e a data de partida deviam ser acedidas de forma mais rápida, de forma a não prejudicar o desempenho. Como tal, atribuímos-lhes um índice.

```
CREATE INDEX idx_data_partida ON Viagem(data_partida);
CREATE INDEX idx_data_chegada ON Viagem(data_chegada);
```

Figura 21 - Código SQL para a criação dos índices sobre as colunas data_partida e data_chegada

No que toca à tabela dos bilhetes, tanto o cliente como o administrador usam várias vezes o valor da data de aquisição para os intervalos e, seguindo a mesma lógica, atribuímos um índice à coluna que guarda este atributo.

```
CREATE INDEX idx_data_aquisicao ON Bilhete(data_aquisicao);
```

Figura 22 - Código SQL para a criação de um índice sobre a coluna data_aquisicao

A criação destes índices permite, assim, que a base de dados tenha um melhor desempenho nas queries que utilizam estes atributos para selecionar os valores pretendidos.

5.6. Estimativa do espaço em disco da base de dados e taxa de crescimento anual

De seguida, apresenta-se uma estimativa do espaço que a base de dados ocupará em disco. Para precaver qualquer cenário possível, a análise feita ao crescimento da base de dados teve por base a análise do seu pior caso, ou seja, foi assumido que todas as viagens realizadas têm lotação esgotada. Quanto aos atributos do tipo *VARCHAR,* assumimos a utilização do tamanho máximo declarado.

Para facilitar o cálculo da estimativa pretendida, começamos por determinar quanto é que uma ocorrência de cada tipo de entidade ocuparia em disco. Desta forma, obtivemos a seguinte tabela:

Tabela 4 - Tamanho de cada entrada na respetiva tabela

Relação	Atributo	Data Type	Tamanho	Total
Cliente	id_cliente	INT	4 bytes	106 bytes
	nome	VARCHAR(50)	50 bytes	
	email	VARCHAR(30)	30 bytes	
	nif	INT	4 bytes	
	password	VARCHAR(18)	18 bytes	
Bilhete	id_bilhete	INT	4 bytes	27 bytes
	preco	FLOAT(5, 2)	4 bytes	
	data_aquisicao	DATETIME	8 bytes	
	classe	CHAR(1)	1 byte	
	numero	SMALLINT	2 bytes	
	cliente	INT	4 bytes	
	viagem	INT	4 bytes	
Viagem	id_viagem	INT	4 bytes	39 bytes
	data_partida	DATETIME	8 bytes	
	data_chegada	DATETIME	8 bytes	
	duração	TIME	3 bytes	

	preco_base	FLOAT(5, 2)	4 bytes	
	comboio	INT	4 bytes	
	origem	INT	4 bytes	
	destino	INT	4 bytes	
Comboio	id_comboio	INT	4 bytes	4 bytes
Estação	id_estacao	INT	4 bytes	24 bytes
	nome	VARCHAR(20)	20 bytes	
Lugar	classe	CHAR(1)	1 byte	7 bytes
	numero	SMALLINT	2 bytes	
	comboio	INT	4 bytes	

Apesar da entidade Cliente ser a que requer maior quantidade de memória por ocorrência, não será a que ocupará mais no sistema. A quantidade de dados mais significativa será gerada pela entidade Bilhete, pois a por cada lugar existente numa determinada viagem estará associado um bilhete.

Pelo Anexo 1, podemos verificar que existem 72 viagens diárias, ocupando estas 72*39 = **2808 bytes/dia** no sistema. Uma vez que existem 6 comboios com 250 lugares (50 premium + 200 económico), o espaço necessário para representar todos os lugares existentes é (250*7) * 6 = **10500 bytes**. Posto isto, a cada dia serão adicionadas 72*250 = 18000 entradas na tabela Bilhete, perfazendo um total de 18000*27 = **486000 bytes/dia**. Somando tudo e prevendo um registo de 50000 clientes por ano teremos então 50000*106 + 486000*365 + 2808*365 + 10500 = **175,2 MBytes/ano**.

5.7. Definição e caracterização das vistas de utilização em SQL

Em SQL, uma *view* é o resultado de uma ou mais operações relacionais sobre a base de dados. Ainda que, para o utilizador, pareça uma tabela como as outras, uma view é uma tabela dinâmica resultante de várias operações entre tabelas ou outras *views*. Este mecanismo providencia uma série de vantagens com a sua utilização e como tal decidimos tirar proveito das mesmas com os seguintes exemplos.

```
CREATE VIEW proxima_viagens AS
SELECT o.nome as 'Origem', v.data_partida as 'Data Partida', d.nome as 'Destino', v.data_chegada as 'Data Chegada'
FROM viagem as v INNER JOIN estacao as o
        ON v.origem = o.id_estacao
        INNER JOIN Estacao as d
        ON v.destino = d.id_estacao
WHERE v.data_partida >= now()
ORDER BY v.data_partida ASC;
```

Figura 23 - Vista que mostra a informação relativa a todas as viagens agendadas

```
CREATE VIEW numero_passagueiro_naviagem AS
SELECT v.id_viagem AS 'ID Viagem', count(v.id_viagem) AS 'Número de Bilhetes Vendidos', v.data_partida AS 'Data de Partida'
    FROM Viagem v , Bilhete b
    WHERE v.id_viagem = b.viagem
    GROUP BY v.id_viagem
    ORDER BY v.data_partida;
```

Figura 24 - Vista que mostra o número de bilhetes vendidos associados à viagem em questão

```
CREATE VIEW numero_passagueiros_estacoes AS
SELECT O.nome AS Origem , D.nome AS Destino, COUNT(b.id_bilhete) AS 'Número de Passageiros'
FROM Bilhete AS B RIGHT JOIN Viagem AS V
            ON B.viagem = V.id_viagem
            INNER JOIN Estacao AS O
            ON V.origem = O.id_estacao
            INNER JOIN Estacao AS D
            ON V.destino = D.id_estacao
GROUP BY O.id_estacao , D.id_estacao
ORDER BY O.nome ASC;
```

Figura 25 - Vista que calcula o número total de bilhete vendidos em viagens entre duas estações.

5.8. Definição e caracterização dos mecanismos de segurança em SQL

Para regular o acesso à base de dados de modo a evitar qualquer tipo de fraude ou ameaça para o sistema foram criados três perfis de utilização: administrador, gestor e utilizador.

O administrador não tem qualquer tipo de restrição podendo, assim, realizar qualquer operação sobre a base de dados ou aceder a qualquer informação nela guardada.

```
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'admin';
GRANT ALL PRIVILEGES ON ecomboios.* TO 'admin'@'localhost';
```

Figura 26 - Código SQL para a criação do perfil Admin

A função do gestor é regular e assegurar a existência de viagens assim como de comboios. Ao adicionar um novo comboio tem também de assegurar a existência dos respetivos lugares. O gestor tem também a capacidade de adicionar novas estações.

```
CREATE USER 'gestor'@'localhost' IDENTIFIED BY 'gestor';
GRANT SELECT (id_cliente, nome, email, nif) ON ecomboios.cliente TO 'gestor'@'localhost';
GRANT SELECT ON ecomboios.bilhete TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON ecomboios.viagem TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON ecomboios.estacao TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON ecomboios.comboio TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON ecomboios.lugar TO 'gestor'@'localhost';
```

Figura 27 - Código SQL para a criação do perfil Gestor

O utilizador pode efetuar o seu registo, podendo posteriormente alterar os dados inseridos, visualizar as viagens e lugares existentes, assim como as estações. Para além disso, pode ainda comprar bilhetes.

```
CREATE USER 'utilizador'@'localhost' IDENTIFIED BY 'utilizador';
GRANT INSERT, UPDATE ON ecomboios.cliente TO 'utilizador'@'localhost';
GRANT SELECT, INSERT ON ecomboios.bilhete TO 'utilizador'@'localhost';
GRANT SELECT ON ecomboios.viagem TO 'utilizador'@'localhost';
GRANT SELECT ON ecomboios.estacao TO 'utilizador'@'localhost';
GRANT SELECT ON ecomboios.lugar TO 'utilizador'@'localhost';
```

Figura 28 - Código SQL para a criação do perfil Utilizador

5.9. Revisão do sistema implementado com o utilizador

Uma vez concluída a última etapa da construção do sistema pretendido pela ECOmboios, procedemos à apresentação do produto obtido. Para o efeito, reunimos com o corpo administrativo da empresa. Nesse encontro, exploramos as várias funcionalidades desenvolvidas, explicando o seu propósito e procurando exemplifica-las.

O produto final foi aceite com bastante agrado uma vez que cumpria todos os requisitos pretendidos.

6. Conclusões e Trabalho Futuro

O desenvolvimento do sistema de base de dados apresentado foi devidamente faseado o que contribuiu para uma maior robustez e estruturação da solução encontrada. De facto, a existência de uma metodologia de trabalho revelou-se essencial para a diminuição da ocorrência de erros durante o processo de desenvolvimento, aumentando assim a produtividade. Embora a construção de um sistema de base de dados seja um processo iterativo, o facto de termos encarado e aprofundado com o devido cuidado cada uma das fases de implementação fez com que quase não fosse preciso retroceder a uma fase anterior.

Analisando o trabalho desenvolvido e refletindo sobre cada uma das etapas, podemos afirmar que os grandes alicerces da solução encontrada são, sem dúvida, as fases de definição do problema e de desenvolvimento do modelo conceptual. Efetivamente, quanto mais aprofundado for o estudo do comportamento da empresa, maior será a qualidade dos requisitos levantados, o que se refletirá em respostas mais assertivas às interrogações do utilizador. Por outro lado, quanto mais detalhado for o conhecimento em relação ao que se pretende representar na base de dados e ao *modus operandi* da empresa, mais fiel será a reprodução dos dados no sistema. É ainda de referir que do modelo conceptual se deriva o modelo lógico, que, posteriormente, dá origem ao modelo físico. Um mau modelo conceptual resultará num sistema obsoleto.

No decorrer do trabalho, a fase que mais tempo demorou a estar concluída e mais duvidas nos suscitou foi a modelação conceptual. Desenvolvemos vários modelos conceptuais incorretos devido a uma fraca definição do problema. Através de sucessivos refinamentos dos objetivos que se esperavam alcançar com a implementação do sistema de base de dados, o desenho do modelo conceptual surgiu de forma natural.

No futuro, poderíamos incluir na base de dados informação relativa aos funcionários da empresa, nomeadamente os maquinistas e os revisores. Desta forma, a base de dados possibilitaria também analisar a afetação dos funcionários a cada serviço.

Em suma, podemos concluir que os objetivos foram cumpridos, sendo o resultado deste processo uma base de dados totalmente funcional capaz de cumprir os propósitos que levaram à sua implementação.

7. Abordagem não relacional

7.1. Justificação da utilização de um sistema NoSQL

Passados uns meses após a implementação da base de dados da ECOmboios, o conceito inovador de viagens de comboio ecológicas causou uma enorme afluência às viagens proporcionadas pela empresa. Ora, dado o elevado número de clientes e tendo em conta todas as viagens diárias efetuadas pelos comboios ininterruptamente, a base de dados cresceu de uma forma exponencial, sobrecarregando a base de dados relacional e tornando as interrogações mais lentas. Perante várias queixas de clientes, o quadro administrativo da empresa agendou uma reunião para procurar soluções para o problema exposto. A alternativa encontrada foi migrar os dados da base relacional que desenvolvemos em SQL para uma baseada em documentos, ou seja, para um modelo NoSQL, utilizando para tal o MongoDB. Esta proposta foi suportada pelos seguintes argumentos, que a administração da ECOmboios aceitou de bom grado e demonstrou o seu entusiasmo em conseguir prestar um serviço com melhor qualidade aos seus clientes:

- As bases de dados não relacionais (NoSQL), neste caso, o MongoDB, permitem uma maior escalabilidade face às relacionais.
- Sendo os requisitos da base de dados na sua maioria de natureza simples, o MongoDB terá uma eficiência maior, apresentando os resultados em tempo inferior aos do MySQL.
- Face o enorme crescimento do número de clientes da empresa, esta pode querer modificar o tipo de comboios a utilizar ou, por exemplo, alterar as informações a guardar dos clientes, existindo a possibilidade de criação de passes mensais ou de cartões *premium*. Ora, com o SQL não seria possível modificar a base de dados já existente introduzindo novos tipos de dados. Porém, o MongoDB é flexível e dinâmico, ou seja, é permitida a introdução de novas colunas ou campos na base, sem prejudicar os dados já existentes.
- Sendo a base de dados utilizada por vários utilizadores ao mesmo tempo (vários clientes podem querer comprar bilhetes no mesmo momento), o MongoDB apresenta também melhor performance no que toca ao acesso concorrente à base em questão.

Com isto, ficou definido que iríamos proceder à migração dos dados em MySQL para uma nova base de dados em MongoDB, ou seja, num modelo não relacional baseada em documentos.

7.2. Identificação e descrição dos objetivos da base de dados

A ECOmboios verificou, então, que, tal como inicialmente previsto, o número de bilhetes e de viagens representam, cada vez mais, uma maior percentagem no número de elementos do sistema de base de dados atual. Sendo estas duas entidades fulcrais para o correto funcionamento do sistema, o seu crescimento acentuado é traduzido num aumento do tempo de computação necessário para que um utilizador comum possa tirar proveito do leque de serviços que a ECOmboios oferece. Este aumento coloca em risco não só o conforto, mas também o grau de satisfação que sempre foram prometidos.

Com a implementação da base de dados MongoDB, a empresa teceu como seu principal objetivo, atenuar o impacto que o crescimento dos bilhetes e das viagens têm na qualidade do serviço prestado aos seus clientes. Através do uso de uma base de dados, focada para responder às interrogações necessárias para o uso comum, é possível que o tempo de computação gasto reduza quando comparado ao sistema atual. Assim, a ECOmboios prevê que o grande crescimento atual não tenha um impacto tão notório.

7.3. Tipo de questões a realizar sobre o sistema de dados NoSQL

As questões que serão realizadas sobre o sistema de dados NoSQL incidem sobre os requisitos de exploração apresentados previamente. Isto porque foram estas as operações cujo desempenho se foi deteriorando com o crescimento da base de dados relacional, como por exemplo saber o total gasto por um cliente num determinado período. Assim, mais especificamente, podemos enumerar as questões realizadas sobre o sistema (dividindo novamente as questões pelo ponto de vista de quem as irá utilizar, ou seja, do cliente ou do administrador):

Do ponto de vista do cliente deve ser possível:

- (RE1) - Ver o histórico das suas viagens num dado período.
- (RE2) - Consultar o montante gasto num dado período.
- (RE3) - Visualizar a origem, destino, duração, preço, lugar e data da viagem associada ao seu bilhete.
- (RE4) - Pesquisar as viagens disponíveis num dado período entre duas estações.
- (RE5) - Consultar a lista de lugares livres para uma viagem.
- (RE6) - Consultar o horário específico duma determinada estação.

Do ponto de vista do administrador da aplicação deve ser possível:

- (RE8) - Consultar as viagens realizadas por um determinado comboio num dado período.
- (RE9) - Saber quais os passageiros que viajaram entre duas estações num dado período.
- (RE10) - Saber quais os passageiros que participaram numa dada viagem.

- (RE11) - Verificar quantos bilhetes foram vendidos num determinado período.
- (RE12) - Calcular o valor total faturado num determinado período.

Como pode ser notado, não referimos a questão relativa à listagem das estações existentes, uma vez que consideramos não haver necessidade de proceder a tal *query*, tendo que, para tal, guardar em documentos apenas o "id" e o nome das estações. Os requisitos de controlo não foram aqui referidos por a sua realização ser de fácil natureza e de ser necessário apenas um comando básico para os realizar, ainda que tivessem que ser adaptados para a estrutura base NoSQL que implementamos.

A implementação em MongoDB das questões enumeradas serão apresentadas numa das secções que se seguem.

7.4. Estrutura base para o sistema de dados NoSQL

Perante os requisitos apresentados e tendo em conta o modelo lógico da base de dados relacional, decidimos definir a estrutura base para o sistema de dados NoSQL da seguinte maneira:

- Coleção "Cliente", cujos documentos contêm as informações relativas a cada um dos clientes (nome, email, NIF e password), apresentando ainda uma lista de documentos *embedded* relativos aos bilhetes desse mesmo cliente. Cada documento "Bilhete" contem o seu preço, data de aquisição, classe, número do lugar, data de partida, data de chegada, duração, origem e destino. Ambos os tipos de documentos contêm ainda o seu id.
- Coleção "Viagem", cujos documentos apresentam o preço base da viagem, a data de partida, a data de chegada, a duração, o id do comboio e as estações de origem e de destino. Além disso, os documentos apresentam também o seu id e documentos *embedded* relativos a todos os bilhetes emitidos para essa viagem, quer já tenham sido adquiridos ou não.

Desta forma, conseguimos resolver a maior parte das questões sem recorrer a *joins*. Isto porque anexamos todos os bilhetes de um cliente ao documento respetivo e todos os bilhetes relativos aos lugares do comboio na dita viagem. Não utilizamos portanto nenhuma coleção para os bilhetes isoladamente, pois estes existem sempre no contexto de uma viagem ou de um cliente (caso este o tenha adquirido). Não utilizamos também nenhuma coleção ou documento para as estações e para os comboios, pois no MongoDB apenas precisamos de saber o nome das estações de origem e destino de uma viagem (não necessitando de criar um documento apenas para podermos guardar o seu "id") e, no que toca aos comboios, tendo também todos os bilhetes associados a cada viagem com o respetivo lugar, torna-se também desnecessário guardar propositadamente o seu "id".

Os esquemas que descrevemos acima são então os seguintes:

Figura 29 - Estrutura dos documentos de cada coleção

7.5. Identificação dos objetos de dados no sistema SQL utilizados para alimentar o novo sistema

Uma vez definida a estrutura base para o sistema NoSQL, tivemos que identificar quais os objetos de dados do sistema SQL que iriamos utilizar para alimentar (povoar) o novo sistema.

Desta forma, vamos, em primeiro lugar, relembrar muito sinteticamente a nova estrutura:

1. Conjunto de documentos “Cliente”, com lista dos bilhetes por si adquiridos, id, nome, email, NIF e password.
2. Os bilhetes de cada cliente apresentam o seu id, o preço, a data de aquisição, a classe e número do lugar, data de partida, data de chegada, duração, origem e destino.
3. Conjunto de documentos “Viagem”, com lista de todos os bilhetes associados a essa viagem, o id da mesma, a data de partida, a data de chegada, a duração, o número do comboio, origem e destino.
4. Os bilhetes de cada viagem incluem o seu id, preço e cliente (caso não tenha sido adquirido, estes três campos serão iguais a 0), data de aquisição, classe e número do lugar.

Para o primeiro e segundo pontos, temos que utilizar os objetos de dados da tabela “Cliente” do sistema SQL e, além disso, dos objetos da tabela “Bilhete”, para podermos inserir a lista destes que está associada a cada cliente. Para obtermos o nome das estações de origem e chegada associadas ao bilhete tivemos que as obter a partir da tabela “Estação”.

Para o terceiro, precisamos dos objetos relativos à tabela “Viagem” (pois lá temos as

informações básicas destas) e, novamente, dos objetos relativos à tabela “Estação”, para podermos guardar os nomes da origem e da chegada.

Para o quarto e último ponto, precisamos dos dados guardados na tabela “Lugar”, uma vez que, como vamos guardar todos os bilhetes existentes numa viagem (caso tenham sido adquiridos ou não), necessitamos de saber quais os lugares disponíveis para tal. Além disso, precisamos também da tabela “Bilhete”, para podermos determinar que bilhetes já foram vendidos.

7.6. Processo de conversão de dados

Em relação à tabela Cliente, mais especificamente a cada entrada, é guardado o id do cliente (que serve também como identificador), o seu nome, email, NIF, password e uma lista de todos os bilhetes adquiridos por si. Em cada bilhete presente nesta lista encontra-se o seu id, preço, data de aquisição, classe, número, data de partida, data de chegada, duração, origem e destino. Para a construção desta lista começamos por filtrar os bilhetes emitidos para obtermos os que foram adquiridos pelo cliente em específico. Para tratar do processo de conversão de dados criamos objetos auxiliares (cada um representativo de um cliente e dos seus respetivos bilhetes), que são introduzidos individualmente, de forma a evitar problemas de falta de memória.

Em relação à tabela Viagem, novamente relativo a cada *row*, é guardado o id da viagem, o seu preço base, datas de partida e chegada, duração, o comboio pela qual é realizada, origem e destino, e uma lista de bilhetes. Esta lista, ao contrário do que se observa nos dados que serão migrados relativos a cada Cliente, não representa todos os bilhetes comprados para a viagem em causa, mas sim uma lista de todos os possíveis bilhetes. Esta lista é determinada recorrendo ao Comboio que realizará a viagem, acedendo aos lugares existentes neste. O que diferencia os bilhetes comprados dos não comprados é o valor do campo “data_aquisicao” ser *null*. O processo de conversão de cada entrada é semelhante ao explicado no caso de Cliente.

7.7. Processo de migração de dados

O processo de migração foi suportado por dois pares de métodos escritos em JAVA, que de seguida se apresentam e explicam: `loadViagem()` e `mongoAddViagem(Viagem v, List<Bilhete> bilhetes)`, `loadCliente()` e `mongoAddCliente(Cliente c)`.

```
private static void loadClientes() throws Exception {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost/ecomboios", "root", "12345");
    PreparedStatement st;
    st = con.prepareStatement("SELECT * FROM cliente;");
    ResultSet rs = st.executeQuery();
    while(rs.next()) {
        Cliente c = new Cliente();
        c.id = rs.getInt("id_cliente");
        c.nome = rs.getString("nome");
        c.email = rs.getString("email");
        c.nif = rs.getInt("nif");
        c.password = rs.getString("password");

        PreparedStatement aux = con.prepareStatement("SELECT b.*, v.*, eo.nome AS 'origem_nome', ed.nome AS 'destino_nome' " +
                                                     "FROM cliente AS c INNER JOIN bilhete AS b " +
                                                                       "ON c.id_cliente = b.cliente " +
                                                                       "INNER JOIN viagem as v " +
                                                                       "ON b.viagem = v.id_viagem " +
                                                                       "INNER JOIN estacao AS eo " +
                                                                       "ON v.origem = eo.id_estacao " +
                                                                       "INNER JOIN estacao AS ed " +
                                                                       "ON v.destino = ed.id_estacao " +
                                                     "WHERE b.cliente = ?;");
        aux.setInt(1, c.id);

        ResultSet rs_aux = aux.executeQuery();
        while (rs_aux.next()){
            Bilhete b = new Bilhete();
            b.id = rs_aux.getInt("id_bilhete");
            b.preco = (float) rs_aux.getDouble("preco");
            b.aquisicao = rs_aux.getTimestamp("data_aquisicao").toLocalDateTime();
            b.classe = rs_aux.getString("classe").charAt(0);
            b.numero = rs_aux.getInt("numero");
            b.viagem = rs_aux.getInt("viagem");
            b.partida = rs_aux.getTimestamp("data_partida").toLocalDateTime();
            b.chegada = rs_aux.getTimestamp("data_chegada").toLocalDateTime();
            b.duracao = rs_aux.getTime("duracao").toString();
            b.origem = rs_aux.getString("origem_nome");
            b.destino = rs_aux.getString("destino_nome");

            c.bilhetes.add(b);
        }
        mongoAddCliente(c);
    }
    con.close();
}
```

Figura 30 - Método 'loadClientes()'

Para extrairmos os dados relativos aos clientes, necessitamos de usar duas *queries*. A primeira *query* presente na figura serve para selecionar todos os clientes registados na base de dados. O 'ResultSet rs' representa tal resultado e iterando sobre ele podemos ir construindo a informação relativa a cada Cliente. A segunda *query* é utilizada para construir a lista dos bilhetes associados a um cliente. Esta *query* retorna informação de cada um dos bilhetes associados ao cliente em causa, assim como informação relativa à viagem a que o bilhete pertence para contemplar toda a informação necessária à execução correta das *queries* sobre esta coleção Cliente, sem ter necessidade de recorrer a outra. Construída a informação relativa a um cliente é então necessária a sua introdução na nova base de dados. Este processo é feito através do método `mongoAddCliente(Cliente c)` que é explicado de seguida.

```
private static void mongoAddCliente(Cliente c) {
    DB db = mongoClient.getDB("ecomboios");
    DBCollection coll = db.getCollection("Cliente");

    List<BasicDBObject> bilhetes = new ArrayList<>();
    for(Bilhete b : c.bilhetes){
        BasicDBObject obj = new BasicDBObject("_id", b.id)
                                        .append("preco", b.preco)
                                        .append("data_aquisicao", b.aquisicao)
                                        .append("classe", b.classe)
                                        .append("numero", b.numero)
                                        .append("data_partida", b.partida)
                                        .append("data_chegada", b.chegada)
                                        .append("duracao", b.duracao)
                                        .append("origem", b.origem)
                                        .append("destino", b.destino);

        bilhetes.add(obj);
    }

    BasicDBObject doc = new BasicDBObject("_id", c.id)
                                .append("nome", c.nome)
                                .append("email", c.email)
                                .append("nif", c.nif)
                                .append("password", c.password)
                                .append("bilhetes", bilhetes);

    coll.insert(doc);
}
```

Figura 31 - Método 'mongoAddCliente(Cliente c)'

De forma a inserir os dados relativos a um Cliente, é necessário inicialmente obter ligação com a base de dados Mongo e posteriormente obter a coleção relativa aos clientes. Para introduzir dados é necessário primeiro convertê-los e isso é feito introduzindo-os num “BasicBDObject”. O processo é terminado pela inserção do objeto criado, ou seja, o novo documento a ser introduzida através do método “insert(doc)” que é passado à coleção anteriormente obtida.

```
private static void loadViagem() throws SQLException {
    Connection con = DriverManager.getConnection("jdbc:mysql://localhost/ecomboios", "root", "12345");
    PreparedStatement st;
    st = con.prepareStatement("SELECT * " +
                              "FROM viagem AS v INNER JOIN estacao AS eo " +
                                              "ON v.origem = eo.id_estacao " +
                                              "INNER JOIN estacao AS ed " +
                                              "ON v.destino = ed.id_estacao;");

    ResultSet rs = st.executeQuery();
    while(rs.next()) {
        Viagem v = new Viagem();
        v.id = rs.getInt("id_viagem");
        v.partida = rs.getTimestamp("data_partida").toLocalDateTime();
        v.chegada = rs.getTimestamp("data_chegada").toLocalDateTime();
        v.duracao = rs.getTime("duracao").toString();
        v.preco_base = rs.getFloat("preco_base");
        v.comboio = rs.getInt("comboio");
        v.origem_nome = rs.getString(10);
        v.destino_nome = rs.getString(12);

        List<Bilhete> bilhetes = new ArrayList<>();
        PreparedStatement st_aux = con.prepareStatement(
            "SELECT aux2.id_bilhete, aux2.preco, aux2.data_aquisicao, aux1.classe, aux1.numero, aux2.cliente " +
            "FROM (SELECT v.id_viagem, l.classe, l.numero " +
                  "FROM viagem AS v INNER JOIN lugar AS l  " +
                                   "ON v.comboio = l.comboio  " +
                  "WHERE v.id_viagem = ?) AS aux1 " +
                                  "LEFT JOIN (SELECT * " +
                                             "FROM bilhete AS b " +
                                             "WHERE b.viagem = ?) AS aux2 " +
                                  "ON aux1.classe = aux2.classe AND aux1.numero = aux2.numero;");
        st_aux.setInt(1, v.id);
        st_aux.setInt(2, v.id);

        ResultSet rs_aux = st_aux.executeQuery();
        while(rs_aux.next()){
            Bilhete b = new Bilhete();
            b.id = rs_aux.getInt("id_bilhete");
            b.preco = rs_aux.getFloat("preco");
            try{
                b.aquisicao =  rs_aux.getTimestamp("data_aquisicao").toLocalDateTime();
            }
            catch (NullPointerException e){
                b.aquisicao = null;
            }
            b.classe = rs_aux.getString("classe").charAt(0);
            b.numero = rs_aux.getInt("numero");
            b.cliente = rs_aux.getInt("cliente");

            bilhetes.add(b);
        }
        mongoAddViagem(v, bilhetes);
    }
}
```

Figura 32 - Método 'loadViagem()'

Os dados relativos à viagem requerem o uso de duas *queries*. A primeira *query* produz uma tabela com todas as viagens e as respetivas origens e destinos. Após a execução da *query*, é produzida um conjunto com os respetivos resultados e é iniciada uma iteração sobre o mesmo. Esta parte é responsável por atribuir os valores corretos ao nosso objeto Viagem. A segunda *query* calcula todos os bilhetes possíveis para uma dada viagem, sendo que, esta resulta da junção de duas *sub-queries*. A primeira *sub-query* consiste na junção da viagem pretendida com

os lugares do comboio que a efetua, ficando assim com uma tabela com todos os lugares e o id da viagem. A segunda *sub-query* calcula todos os bilhetes vendidos para a viagem pretendida. Por fim, fazemos um *left outter join* entre os bilhetes vendidos e todos os lugares, originando uma tabela com todos os bilhetes possíveis, tendo em conta que os que não foram vendidos ficam com alguns campos a *null*. A segunda *query* é utilizada para preencher os bilhetes na respetiva viagem.

```
private static void mongoAddViagem(Viagem v, List<Bilhete> bilhetes){
    DB db = mongoClient.getDB("ecomboios");
    DBCollection coll = db.getCollection("Viagem");


    List<BasicDBObject> bilhetes_obj = new ArrayList<>();
    for(Bilhete b : bilhetes){
        BasicDBObject x = new BasicDBObject("_id", b.id)
                                    .append("preco", b.preco)
                                    .append("data_aquisicao", b.aquisicao)
                                    .append("classe", b.classe)
                                    .append("numero", b.numero)
                                    .append("cliente", b.cliente);

        bilhetes_obj.add(x);
    }

    BasicDBObject obj = new BasicDBObject("_id", v.id)
                                   .append("preco_base", v.preco_base)
                                   .append("data_partida", v.partida)
                                   .append("data_chegada", v.chegada)
                                   .append("duracao", v.duracao)
                                   .append("comboio", v.comboio)
                                   .append("origem", v.origem_nome)
                                   .append("destino", v.destino_nome)
                                   .append("bilhetes", bilhetes_obj);

    coll.insert(obj);
}
```

Figura 33 - Método 'mongoAddViagem()'

O processo de inserção dos dados na coleção Viagem é semelhante ao explicado anteriormente para a coleção Cliente e por isso não necessita de explicação.

7.8. Interrogações no sistema NoSQL

- (RE1) - Ver o histórico das suas viagens num dado período.

```
db.Cliente.find(
    {
        "$and" : [
            {
                "_id" : id
            },
            {
                "bilhetes.data_partida" : {
                    "$gte" : ISODate(data_partida),
                    "$lte" : ISODate(data_chegada)
                }
            }
        ]
    },
    {
        "bilhetes.origem" : 1.0,
        "bilhetes.destino" : 1.0,
        "bilhetes.data_partida" : 1.0,
        "bilhetes.data_chegada" : 1.0
    }
)
```

Figura 34 - Query 'historico_viagens'

- (RE2) - Consultar o montante gasto num dado período.

```
db.Cliente.aggregate(
    [
        {
            $unwind: "$bilhetes"
        },
        {
            $match: {
                "$and" : [
                    {"bilhetes.data_aquisicao":  {"$gte" : ISODate(data_inicio)}},
                    {"bilhetes.data_aquisicao":  { "$lte": ISODate(data_fim)}},
                    {"_id" : id}]
                }
        },
        {
            $group: {
                _id: "$nome",
                Valor: {
                    $sum: "$bilhetes.preco"
                }
            }
        }
    ]
)
```

Figura 35 - Query 'total_gasto'

- (RE3) - Visualizar a origem, destino, duração, preço, lugar e data da viagem associada ao seu bilhete.

```
db.Cliente.find(
    {
        "bilhetes._id" : id
    },
    {
        "bilhetes.$" : 1,
        "_id" : 0,
        "bilhetes._id" : 1.0,
        "bilhetes.origem" : 1.0,
        "bilhetes.destino" : 1.0,
        "bilhetes.duracao" : 1.0,
        "bilhetes.preco" : 1.0,
        "bilhetes.numero" : 1.0,
        "bilhetes.classe" : 1.0,
        "bilhetes.data_partida" : 1.0
    }
)
```

Figura 36 - Query 'detalhes_viagem'

- (RE4) - Pesquisar as viagens disponíveis num dado período entre duas estações.

```
db.Viagem.find(
    {
        "$and" : [
            {
                "data_partida" : {
                    "$gte" : ISODate(data_inicio)
                }
            },
            {
                "data_chegada" : {
                    "$lte" : ISODate(data_fim)
                }
            },
            {
                "origem" : origem
            },
            {
                "destino" : destino
            }
        ]
    },
    {
        "preco_base" : 1.0,
        "data_partida" : 1.0,
        "data_chegada" : 1.0,
        "duracao" : 1.0,
        "origem" : 1.0,
        "destino" : 1.0
    }
)
```

Figura 37 - Query 'viagens_between'

- (RE5) - Consultar a lista de lugares livres para uma viagem.

```
db.Viagem.aggregate(
    [
        {
            $match: {
                "_id": id_viagem
            }
        },
        {
            $unwind:
                "$bilhetes"
        },
        {
            $match: {
                "bilhetes.data_aquisicao": null
            }
        },
        {
            $project: {
                "bilhetes.classe":1,
                "bilhetes.numero":1
            }
        }
    ]
)
```

Figura 38 - Query 'lugares_livres'

- (RE6) - Consultar o horário específico duma determinada estação.

```
db.getCollection("Viagem").find(
    {
        "$and" : [
            {
                "origem" : origem
            },
            {
                "data_partida" : {
                    "$gte" : ISODate(data_partida)
                }
            }
        ]
    },
    {
        "bilhetes" : 0.0
    }
)
```

Figura 39 - Query 'horario_partida_estacao'

```
db.getCollection("Viagem").find(
    {
        "$and" : [
            {
                "destino" : destino
            },
            {
                "data_chegada" : {
                    "$gte" : ISODate(data_chegada)
                }
            }
        ]
    },
    {
        "bilhetes" : 0.0
    }
)
```

Figura 40 - Query 'horario_chegada_estacao'

Do ponto de vista do administrador da aplicação deve ser possível:

- (RE8) - Consultar as viagens realizadas por um determinado comboio num dado período.

```
db.Viagem.find(
    {
        "$and" : [
            {
                "comboio" : comboio
            },
            {
                "data_partida" : {
                    "$gte" : ISODate(data_inicio)
                }
            },
            {
                "data_chegada" : {
                    "$lte" : ISODate(data_fim)
                }
            }
        ]
    },
    {
        "comboio" : 0.0,
        "bilhetes" : 0.0
    }
)
```

Figura 41 - Query 'viagens_comboio_between'

- (RE9) - Saber quais os passageiros que viajaram entre duas estações num dado período.

```
db.Cliente.find(
    {
        "$and" : [
            {
                "bilhetes.origem" : origem
            },
            {
                "bilhetes.destino" : destino
            },
            {
                "bilhetes.data_partida" : {
                    "$gte" : ISODate(data_partida),
                    "$lte" : ISODate(data_chegada)
                }
            }
        ]
    },
    {
        "nome" : 1.0
    }
)
```

Figura 42 - Query 'clientes_between_estacoes'

- (RE10) - Saber quais os passageiros que participaram numa dada viagem.

```
db.Viagem.aggregate(
    [
        {
            $match: {
                "_id": id_viagem
            }
        },
        {
            $unwind:
                "$bilhetes"
        },
        {
            $match: {"bilhetes.data_aquisicao": {$ne: null}}
        },
        {
            $project: {
                "bilhetes.cliente":1
            }
        }
    ]
)
```

Figura 43 - Query 'clientes_na_viagem'

- (RE11) - Verificar quantos bilhetes foram vendidos num determinado período.

```
db.Viagem.aggregate(
    [
        {
            $unwind: "$bilhetes"
        },
        {
            $match: {"bilhetes.data_aquisicao": {$ne: null}}
        },
        {
            $match: {
                "bilhetes.data_aquisicao": {
                    $gte: ISODate(data_inicio),
                    $lte: ISODate(data_fim)}
            }
        },
        {
            $group: {
                _id: null,
                num_bilhetes: {
                    $sum: 1 }
                }
        }
    ]
)
```

Figura 44 - Query 'total_bilhetes_vendidos'

- (RE12) - Calcular o valor total faturado num determinado período.

```
db.Cliente.aggregate(
    [
        {
            $unwind: "$bilhetes"
        },
        {
            $match: {
                "$and" : [
                    {"bilhetes.data_aquisicao":  {"$gte" : ISODate(data_inicio)}},
                    {"bilhetes.data_aquisicao":  { "$lte": ISODate(data_fim)}}]
            }
        },
        {
            $group: {
                _id: null,
                Valor: {
                    $sum: "$bilhetes.preco"
                }
            }
        }
    ]
)
```

Figura 45 - Query 'total_faturado'

7.9. Conclusões

Uma vez terminada a implementação do sistema NoSQL, podemos fazer uma abordagem crítica sobre o mesmo. Comparando com o sistema relacional, verificamos que o aumento de performance se deve em grande parte à inclusão de todos os dados necessários ao processamento de uma interrogação na Coleção sobre onde esta se executa. No entanto, esta condição leva à existência de dados repetidos, algo que não acontecia no sistema relacional. Atendendo à forma como as interrogações são construídas é também possível verificar a ausência do acesso a diferentes coleções na mesma interrogação. Já no modelo relacional temos de aceder a diferentes tabelas e efetuar operações de *joins* para chegarmos ao resultado obtido. A leitura das interrogações feitas sobre o MongoDB é também mais simples que as *queries* escritas em SQL. Porém, os resultados apresentados pelas interrogações em SQL são de mais fácil leitura comparativamente aos resultados que o MongoDB apresenta.

Refletindo sobre o processo realizado, podemos concluir que o tempo requerido para o seu desenvolvimento foi despendido homogeneamente pelas diferentes fases. É essencial estabelecer à partida que interrogações queremos satisfazer no sistema não relacional, uma vez que todos as junções devem ser realizadas no processo de migração, montando, de forma correta, os documentos que povoam as coleções.

Em suma, podemos concluir que os objetivos foram cumpridos, sendo o resultado deste processo uma base de dados totalmente funcional capaz de responder às interrogações para as quais foi desenhada.

8. Referências Bibliográficas (em formato Harvard)

1. CONNOLLY, T. M., & BEGG, C. E. (2014). Database systems: a practical approach to design, implementation, and management. Harlow, England, Addison-Wesley.

Lista de Siglas e Acrónimos

BD Base de Dados

SGBD Sistema de Gestão de Base de Dados

Anexos

1. Horários

Braga – Porto

Tabela 5 - Horário Braga-Porto

Braga	Porto	Porto	Braga
Hora Partida	**Hora Chegada**	**Hora Partida**	**Hora Chegada**
00:00:00	00:20:00	00:10:00	00:30:00
07:00:00	07:20:00	07:10:00	07:30:00
08:00:00	08:20:00	08:10:00	08:30:00
09:00:00	09:20:00	09:10:00	09:30:00
10:00:00	10:20:00	10:10:00	10:30:00
11:00:00	11:20:00	11:10:00	11:30:00
12:00:00	12:20:00	12:10:00	12:30:00
13:00:00	13:20:00	13:10:00	13:30:00
14:00:00	14:20:00	14:10:00	14:30:00
15:00:00	15:20:00	15:10:00	15:30:00
16:00:00	16:20:00	16:10:00	16:30:00
17:00:00	17:20:00	17:10:00	17:30:00
18:00:00	18:20:00	18:10:00	18:30:00
19:00:00	19:20:00	19:10:00	19:30:00
20:00:00	20:20:00	20:10:00	20:30:00
21:00:00	21:20:00	21:10:00	21:30:00
22:00:00	22:20:00	22:10:00	22:30:00
23:00:00	23:20:00	23:10:00	23:30:00

Porto – Lisboa

Tabela 6 - Horário Porto-Lisboa

Porto	Lisboa	Lisboa	Porto
Hora Partida	**Hora Chegada**	**Hora Partida**	**Hora Chegada**
01:00:00	02:25:00	01:10:00	02:55:00
07:00:00	08:25:00	07:10:00	08:55:00
09:00:00	10:25:00	09:10:00	10:55:00
11:00:00	12:25:00	11:10:00	12:55:00
13:00:00	14:25:00	13:10:00	14:55:00
15:00:00	16:25:00	15:10:00	16:55:00
17:00:00	18:25:00	17:10:00	18:55:00
19:00:00	20:25:00	19:10:00	20:55:00
21:00:00	22:25:00	21:10:00	22:55:00
23:00:00	00:25:00	23:10:00	00:55:00

Braga – Lisboa

Tabela 7 - Horário Braga-Lisboa

Braga	Lisboa	Lisboa	Braga
Hora Partida	**Hora Chegada**	**Hora Partida**	**Hora Chegada**
08:10:00	09:55:00	08:10:00	09:55:00
10:10:00	11:55:00	10:10:00	11:55:00
12:10:00	13:55:00	12:10:00	13:55:00
14:10:00	15:55:00	14:10:00	15:55:00
16:10:00	17:55:00	16:10:00	17:55:00
18:10:00	19:55:00	18:10:00	19:55:00
20:10:00	21:55:00	20:10:00	21:55:00
22:10:00	23:55:00	22:10:00	23:55:00