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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Reservas de viagens em comboios nacionais e internacionais

A75353 Dinis Peixoto

A74185 Ricardo Pereira

A75210 Marcelo Lima

A67638 Carlos Faria

Novembro, 2016

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Reservas de viagens em comboios nacionais e internacionais

A75353 Dinis Peixoto

A74185 Ricardo Pereira

A75210 Marcelo Lima

A67638 Carlos Faria

Novembro, 2016

Resumo

Este relatório é o resultado do trabalho elaborado no âmbito da Unidade Curricular de Base de Dados, na qual nos foi sugerida a criação de uma base de dados com o tema "Reservas de viagens em comboios nacionais e internacionais". Ao longo deste relatório serão apresentados todos os passos que tivemos de efetuar desde a criação do Modelo Conceptual até à implementação do Modelo Físico.

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

De seguida, é feita uma análise da viabilidade do projeto bem como uma análise de requisitos, onde são expostas as entidades que serão essenciais para a construção do projeto. É ainda feita uma análise de transações de dados, onde são expostas possíveis funcionalidades de interação tanto com o utilizador como o gestor.

Na apresentação do Modelo Conceptual são descritas as entidades e relacionamentos envolvidos, bem como os respetivos atributos. Para cada entidade, é apresentada informação sobre a sua descrição e ocorrência e é identificada a chave primária bem como as chaves alternativas. Para cada atributo é apresentado o domínio de valores possíveis. Termina com a descrição das transações de informação que um gestor e utilizador poderão realizar.

Na transição do Modelo Conceptual para Lógico é abordado novamente os relacionamentos e o domínio de atributos, mas desta vez de forma mais aprofundada, agora com a identificação das chaves estrangeiras. É abordada a normalização das tabelas, onde é justificada a normalização do nosso modelo até à terceira forma normal. Para terminar, são apresentadas algumas das transações mais importantes do sistema através de Mapas de Transação.

Passando para o Modelo Físico, é apresentada a tradução do Modelo Lógico para um esquema que já pode realmente ser usado como um SGBD, todo ele em código SQL. É feita uma estimativa dos requisitos de espaço em disco que poderá ser ocupado pela base de dados. Termina com uma análise de transações bem como uma definição das vistas dos utilizadores e regras de acesso.

Por último, são apresentadas as conclusões do trabalho realizado, apontando os seus pontos fortes e fracos e são também mencionadas algumas sugestões para estender o trabalho realizado num futuro próximo.

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

Palavras-Chave: Bases de Dados, Bases de Dados Relacionais, Análise de Requisitos, Modelo Conceptual, Modelo Lógico, Modelo Físico, brModelo, MySQL Workbench, SQL

Índice

Anexos

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1. Contextualização

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

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

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

O desenvolvimento de uma qualquer empresa passa também por acompanhar os desenvolvimentos adjacentes à mesma, a *Euro-Train* não é exceção. Como tal, o desenvolvimento da tecnologia móvel permitiu um aumento significativo do uso de *smartphones*. Tal como muitas outras empresas, a *Euro-Train* viu assim a possibilidade de lançar uma aplicação *online* para *smartphones*, que permitisse a reserva de bilhetes para viagens tanto a nível nacional como a nível internacional (em toda a Europa).

1.2. Apresentação do Caso de Estudo

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

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

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

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

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

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

1.3. Motivação e Objetivos

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

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

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

1.4. Estrutura do Relatório

Após termos apresentado o caso de estudo escolhido e a motivação que está por detrás deste, nos seguintes capítulos deste projeto será exposta a análise e a construção do Modelo Conceptual da base de dados, a transição deste para o seu esquema Lógico e por último a implementação do Físico. Estas etapas serão abordas em pormenor nos próximos capítulos do relatório.

Na análise do caso de estudo dá-se a conhecer uma breve descrição das entidades e dos relacionamentos entre elas, bem como os atributos que as caracterizam. A maneira como um utilizador irá interagir com a informação da base de dados é descrita sob a forma de possíveis operações de manipulação e consulta, através de exemplos práticos.

Logo depois, é exposta a análise do Modelo Conceptual da base de dados, onde são apresentadas de forma tabular as entidades e relacionamentos envolvidos, bem como, os respetivos atributos. Para cada entidade, é apresentada informação sobre a sua descrição e ocorrência, enquanto que, para os relacionamentos é caracterizada a sua multiplicidade entre as entidades envolvidas. Para os atributos, é identificado o tipo de dados e o seu tamanho, bem como se estes podem ser nulos, multivalorados ou derivados, após isto, determina-se

também o domínio de valores possíveis. Apresenta-se para cada entidade as chaves candidatas, onde é identificada a chave primária bem como as chaves alternativas. Segue ainda a descrição das transações de informação que um gestor e utilizador poderão realizar. Por fim, fechando esta secção é apresentada a representação gráfica do Modelo Conceptual.

Passando para o Modelo Lógico, primeiramente é feita a especificação das entidades e diferentes tipos de relacionamentos entre elas, identificando-se agora as chaves estrangeiras. É abordada a normalização das tabelas, onde é justificada a normalização do nosso modelo até à terceira forma normal. Logo após, são apresentadas algumas das transações mais importantes do sistema através de Mapas de Transação que mostram a sequência de ações que são necessárias para executar a transação com sucesso, validando, por consequência, a mesma.

De seguida, é apresentado um retrato do Modelo Lógico, proveniente do *MySQL Workbench*, onde é possível identificar os vários relacionamentos anteriormente descritos. É verificada a integridade do modelo, identificando informação sobre os atributos, nomeadamente se são de preenchimento obrigatório e quais as restrições ao domínio de valores possíveis de atribuir. Estuda-se a multiplicidade dos relacionamentos através da contextualização das chaves primárias e estrangeiras. Justifica-se, também, a integridade referencial entre entidades, ou seja, apresenta-se as regras que mantêm a consistência de informação através das tabelas que partilham uma chave estrangeira. Para finalizar esta secção, é apresentado um pequeno estudo sobre a viabilidade do crescimento futuro da base de dados, onde é sugerida a implementação de novas funcionalidades.

Finalmente, é apresentado o Modelo Físico da base de dados. A descrição detalhada de atributos e relacionamentos já segue, desta forma, uma semântica aproximada da linguagem SQL. É feita uma estimativa dos requisitos do espaço em disco que poderá ser ocupado pela base de dados. Termina com uma análise de transações bem como uma definição das vistas dos utilizadores e regras de acesso. Por último, são apresentadas as conclusões do trabalho realizado bem como alguns anexos importantes.

2. Análise da viabilidade do projeto

Este projeto consiste na implementação de um Sistema de Base de Dados (SBD), mais especificamente num Sistema de Base de Dados Relacional (SBDR). No entanto, será esta implementação realmente necessária e trará vantagens consigo associadas? Antes de mais, é preciso entender o que é uma base de dados e qual o seu propósito. Ora, de um ponto de vista teórico, podemos afirmar que uma base de dados é um conjunto estruturado de informação, partilhável e sujeita a um controlo central.

Uma base de dados relacional é composta por um conjunto de tabelas e associações entre estas. A associação entre os dados é o ponto forte dos sistemas relacionais. Estas tabelas são formadas por linhas e colunas onde se encontram os dados, que numa base de dados relacional são representados como valores nas colunas das tabelas.

A implementação deste tipo de Sistema de Base de Dados, proporciona várias vantagens e ganhos operacionais que tornam este projeto bastante viável e oportuno. Uma vantagem importante das tabelas resulta do facto de uma tabela poder ter mais do que uma finalidade e dos seus dados poderem ser vistos com diferentes formas e formatos, ao contrário de, por exemplo, um ficheiro.

Relativamente à implementação da base de dados podemos destacar as seguintes cinco vantagens mais relevantes:

- **Resposta rápida aos pedidos de informação**

Um dos ganhos a nível operacional é a resposta rápida aos pedidos de informação. Como os dados estão integrados numa única estrutura (a base de dados) a resposta a questões complexas processa-se mais rapidamente.

- **Acesso múltiplo**

O acesso múltiplo é outra das vantagens de um SBDR. O *software* de gestão de base de dados permite que os dados sejam acedidos de diversas maneiras. Nomeadamente, os dados podem ser visualizados através de pesquisas sobre qualquer um dos campos da tabela.

- **Flexibilidade**

Outra das grandes vantagens é a flexibilidade. Em consequência da independência entre dados e programas, qualquer alteração num desses elementos não implica modificações drásticas no outro.

- **Integridade da informação**

A integridade da informação é também um dos motivos que justifica a viabilidade deste projeto. Dada a absoluta exigência de não permitir a redundância, as modificações de dados são feitas num só sítio, evitando-se assim possíveis conflitos entre diferentes versões da mesma informação.

- **Melhor gestão da informação**

Por último, um SBDR permite-nos uma melhor gestão da informação. Em consequência da localização central dos dados, sabe-se sempre como e onde está a informação.

Desta forma, podemos concluir que a implementação da base de dados tem realmente o objetivo de facilitar e melhorar a forma como toda a informação é processada e armazenada e deste modo contribuir para uma melhor gestão da mesma.

3. Análise do caso de estudo

3.1. Análise de requisitos

Esta implementação de base de dados pretende armazenar as reservas de viagens de comboio nacionais e internacionais, para melhorar e facilitar a gestão e o processo de reservas, efetuadas via *online*.

- **Cliente**

  Cada cliente pode efetuar várias reservas, mas para isso precisa primeiro de se registar na aplicação, onde terá de fornecer um username e uma password, que mais tarde usará sempre que se pretender autenticar. É também necessário saber o seu nome, endereço (cidade e país) e contactos (e-mail e telemóvel), com o intuito de poder entrar em contacto com este, em caso de necessidade. A cada cliente está associado um número identificador único.

- **Reserva**

  A reserva terá de possuir a data da viagem, especificar um lugar que o cliente irá ocupar, fazendo referência ao número do lugar, carruagem e a classe a que este lugar corresponde, assim como o respetivo preço. Necessita também de possuir um código que o cliente utilizará para validar a reserva, no comboio. Cada reserva tem um número identificador único.

- **Comboio**

  Um comboio pode realizar várias viagens. A caracterização de um comboio é feita com base no seu tipo, numa breve descrição do mesmo, na sua capacidade, e na disposição dos lugares, isto é, no número de cada lugar, na carruagem e na classe em que se insere. A distinção de comboios é feita por um número único que cada um apresenta.

- **Viagem**

  Uma viagem é efetuada por um comboio apenas e contém várias reservas associadas. Para cada viagem, para além da duração desta, é necessário

saber informações relativamente ao ponto de partida e ao ponto de chegada, tais como a hora, a cidade e o país. A distinção de viagens entre dois locais é feita por um número identificador único.

3.2. Análise de transações de dados

- **Utilizador**

  Um utilizador corresponde a um cliente da Euro-Train, como tal é capaz de executar as seguintes ações:

  - Inserção de dados
    - Inserir os dados de um novo cliente, para o seu registo na aplicação – tal como o nome António Fernandes, username antoniofernandes123, password helloworld, e-mail antoniofernandes@mail.com, número de telemóvel 987654321 e endereço Braga, Portugal.
    - Inserir dados de uma nova reserva, por exemplo, uma reserva no dia 2016-11-15 para uma viagem na classe conforto, lugar 85 e na carruagem nº3. Com o código da reserva B012D31A e um preço de 35€.

  - Atualização/Remoção de dados
    - Atualizar dados do próprio utilizador.
    - Atualizar/Remover dados de uma reserva.

  - Queries sobre os dados
    - Consultar as suas reservas efetuadas.
    - Consultar horário de viagens entre duas cidades.
    - Consultar lista de lugares disponíveis numa determinada viagem.
    - Apresentar lista de viagens entre duas cidades ordenado pelas viagens com menor duração.

- **Gestor**

  Um gestor corresponde a um funcionário com acesso a mais privilégios na base de dados da Euro-Train, este é capaz de executar todas as ações de um utilizador e ainda as seguintes:

  - Inserção de dados
    - Inserir dados de um novo comboio – tal como os detalhes de um alfa, a sua descrição e todos os lugares presentes neste, identificando para cada, o número, a carruagem e a classe pertencente.

        - Inserir dados de uma nova viagem – tal como uma viagem com origem em Lisboa, Portugal e destino em Madrid, Espanha, adicionando as respetivas horas de partida/chegada, como por exemplo, com partida às 10:00:00 e chegada às 12:00:00. Ao adicionar uma viagem deve também fazer-se referência ao comboio que a vai realizar.

- Atualização de dados
    - Atualizar dados de um comboio.
    - Atualizar dados de uma viagem.

- Queries sobre os dados
    - Consultar lista de utilizadores/viagens/comboios presentes na aplicação.
    - Apresentar todas as reservas de uma dada viagem, por ordem alfabética do nome do titular da reserva.
    - Apresentar lista de todas as viagens, por ordem crescente da duração.
    - Apresentar lista de todos os comboios, por ordem decrescente da capacidade.

4. Modelo Conceptual de Dados

4.1. Diagrama do Modelo Conceptual

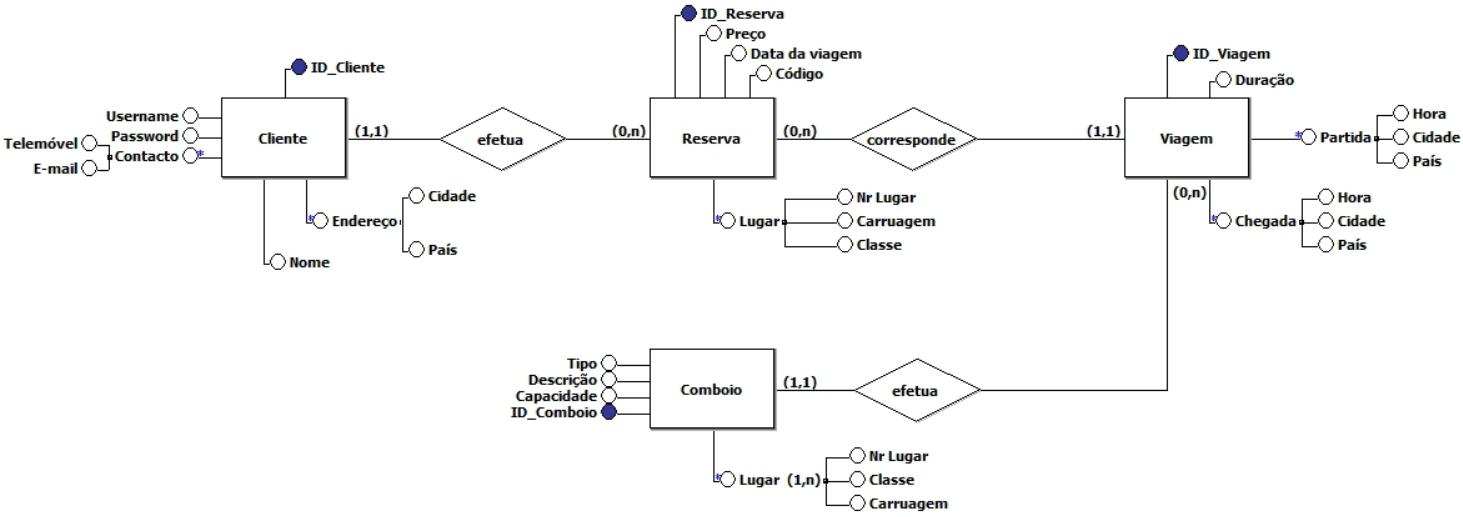

Figura 1 - Diagrama do Modelo Conceptual

4.2. Identificar entidades

Entidade	Descrição	Aliases	Ocorrência
Cliente	Termo geral que descreve todos os clientes registados no sistema.	-	Cada cliente pode efetuar várias reservas para uma determinada viagem.
Reserva	Termo geral que descreve todas as reservas registadas no sistema.	-	Cada reserva está associada a apenas uma viagem, e corresponde a um único cliente.
Viagem	Termo geral que descreve todas as viagens registadas no sistema.	-	Cada viagem é realizada por um único comboio e contém várias reservas.
Comboio	Termo geral que descreve todos os comboios registados no sistema.	-	Cada comboio pode realizar várias viagens.

Tabela 1 – Identificação de entidades.

4.3. Identificar relacionamentos

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Cliente	1..1	efetua	0..N	Reserva
Reserva	0.. N	corresponde	1..1	Viagem
Comboio	1..1	efetua	0..N	Viagem

Tabela 2 – Identificação de relacionamentos.

Durante o processo de criação do Modelo Conceptual deparamo-nos com várias alternativas, que ao início nos pareceram bastante aceitáveis. No entanto, de forma a garantir a melhor acessibilidade e organização dos dados, chegamos a este modelo que foi o que nos pareceu mais assertivo e adequado à base de dados em questão.

Um dos requisitos que nos deu um pouco mais trabalho foi a necessidade de identificar individualmente cada lugar existente em cada um dos comboios, de forma a termos a informação relativa à disponibilidade de cada um dos lugares no momento em que um Cliente faz uma Reserva. No entanto, ultrapassamos este obstáculo adicionando o Lugar como um atributo multi-valor do Comboio.

Relativamente à Viagem, no início ponderamos colocar a data de partida e chegada como atributos da mesma. Uma vez que estas são realizadas todos os dias e às mesmas horas, não fazia sentido colocar a data de partida e chegada como atributos, isso só iria sobrecarregar a base de dados, uma vez que seria necessário adicionar múltiplas viagens com a mesma origem e destino para os diversos dias do ano. E desta forma, à medida que os dias passavam, as viagens teriam de ser descartadas da base de dados, o que não seria, de todo, correto.

Os relacionamentos binários que ligam a Reserva ao Cliente e à Viagem mantêm o registo atualizado dos clientes que fazem uma determinada reserva para uma dada viagem. Por outro lado, os relacionamentos binários que ligam a Viagem à Reserva e ao Comboio permitem-nos manter o registo atualizado dos vários lugares de um comboio que estão reservados para uma determinada viagem.

Observando a cardinalidade dos relacionamentos, podemos verificar que um cliente pode fazer tantas reservas quantas desejar, no entanto podem haver clientes registados sem efetuar nenhuma reserva. Da mesma maneira que uma viagem poderá ter várias reservas ou nenhuma. E por último, um comboio pode efetuar várias viagens, podendo haver comboios que não efetuam qualquer viagem.

4.4. Identificar e associar atributos

Entidade	Atributos	Descrição	Data type & length	Nulls	Multi-valor	Derivado	VP
Cliente	ID_Cliente	Identificador do Cliente	INT	Não	Não	Não	-
	Nome	Nome do Cliente	VARCHAR(64)	Não	Não	Não	-
	Username	Username do Cliente	VARCHAR(32)	Não	Não	Não	-
	Password	Password do Cliente	VARCHAR(32)	Não	Não	Não	-
	Contacto						
	E-mail	E-mail do Cliente	VARCHAR(64)	Não	Não	Não	-
	Telemovel	Número de telemóvel do Cliente	INT	Não	Não	Não	-
	Endereço						
	País	País do Cliente	VARCHAR(32)	Não	Não	Não	-
	Cidade	Cidade do Cliente	VARCHAR(32)	Não	Não	Não	-
Reserva	ID_Reserva	Identificador da Reserva	INT	Não	Não	Não	-
	Lugar						
	Lugar	Número do Lugar	INT	Não	Não	Não	-
	Carruagem	Número da Carruagem do Lugar	INT	Não	Não	Não	-
	Classe	Classe do Lugar	VARCHAR(16)	Não	Não	Não	-
	Data da Viagem	Data da Viagem reservada	DATE	Não	Não	Não	-
	Código	Código de verificação da Reserva	VARCHAR(8)	Não	Não	Não	-
	Preço	Preço da Viagem	DECIMAL(5,2)	Não	Não	Não	-
Comboio	ID_Comboio	Identificador do Comboio	INT	Não	Não	Não	-
	Descrição	Descrição do Comboio	TEXT(512)	Não	Não	Não	-
	Tipo	Tipo do Comboio	VARCHAR(64)	Não	Não	Não	-
	Capacidade	Capacidade do Comboio	INT	Não	Não	Sim	-
	Lugar						
	Carruagem	Carruagem do Lugar	INT	Não	Sim	Não	-
	Classe	Classe do Lugar	INT	Não	Sim	Não	-
	Nr Lugar	Número do Lugar	INT	Não	Sim	Não	-
Viagem	ID_Viagem	Identificador da Viagem	INT	Não	Não	Não	-
	Duração	Duração da Viagem	TIME	Não	Não	Sim	-
	Partida						
	Hora	Hora de partida da Viagem	TIME	Não	Não	Não	-
	Cidade	Cidade de partida da Viagem	VARCHAR(32)	Não	Não	Não	-
	País	País de partida da Viagem	VARCHAR(32)	Não	Não	Não	-
	Chegada						
	Hora	Hora de chegada da Viagem	TIME	Não	Não	Não	-
	Cidade	Cidade de chegada da Viagem	VARCHAR(32)	Não	Não	Não	-
	País	País de chegada da Viagem	VARCHAR(32)	Não	Não	Não	-

Tabela 3 – Identificação e associação de atributos.

4.5. Determinar domínio dos atributos

Cada atributo definido tem um domínio de ocorrência de valores que lhe podem ser atribuídos. Neste capítulo segue a listagem de todos os atributos da nossa Base de Dados, com o respetivo domínio de valores.

- **Cliente**
  - *ID_Cliente*
    Qualquer número inteiro positivo.
  - *Username*
    Qualquer sequência alfanumérica.
  - *Password*
    Qualquer sequência alfanumérica.
  - *Nome*
    Sequência de palavras.
  - *E-mail*
    Terá de respeitar as regras de validação de endereços de e-mail em vigor, onde temos uma sequência de caracteres alfanuméricos, seguidos do caracter "@" e do domínio do provedor de serviço de e-mail.
  - *Telemóvel*
    Sequência de números positivos.
  - *País*
    Corresponde a um país presente no mapa do continente europeu.
  - *Cidade*
    Corresponde a uma cidade registada nos mapas topográficos dos países europeus.

- **Reserva**
  - *ID_Reserva*
    Qualquer número inteiro positivo.
  - *Nr Lugar*
    Qualquer número inteiro positivo.
  - *Carruagem*
    Qualquer número inteiro positivo.
  - *Classe*
    Pode ter um dos seguintes valores pré-definidos: Conforto ou Turística.
  - *Data da Viagem*
    Segue o formato ISSO 8601 (aaaa-mm-dd).
  - *Código*
    Sequência de caracteres alfanuméricos.

- Preço
  Qualquer número decimal positivo.

- **Comboio**
  - *ID_Comboio*
    Qualquer número inteiro positivo.
  - *Descrição*
    Sequência de palavras.
  - *Tipo*
    Pode ter um dos seguintes valores pré-definidos: Alfa, Intercidades, Internacional.
  - *Capacidade*
    Qualquer número inteiro positivo.
  - *Nr Lugar*
    Qualquer número inteiro positivo.
  - *Carruagem*
    Qualquer número inteiro positivo.
  - *Classe*
    Pode ter um dos seguintes valores pré-definidos: Conforto ou Turística.

- **Viagem**
  - *ID_Viagem*
    Qualquer número inteiro positivo.
  - Duração
    Segue o formato ISSO 8601 (hh:mm:ss).
  - *Hora (Partida/Chegada)*
    Segue o formato ISSO 8601 (hh:mm:ss).
  - *País (Partida/Chegada)*
    Corresponde a um país presente no mapa do continente europeu.
  - *Cidade (Partida/Chegada)*
    Corresponde a uma cidade registada nos mapas topográficos dos países europeus.

4.6. Determinar candidatos a Chaves Primárias

Pretende-se averiguar os atributos passíveis de serem chaves candidatas para cada entidade. Estas chaves candidatas identificam unicamente uma ocorrência das entidades, podendo existir várias que satisfaçam este requisito. No entanto, apesar de existirem diversas chaves candidatas, apenas uma será selecionada para ser chave primária e, a partir desse momento, as demais chaves tornam-se chaves alternativas.

Para cada entidade existente realizou-se um estudo das chaves candidatas, identificando a chave primária e alternativas, quando existentes, ordenando-as por ordem de relevância na identificação incontestável de cada ocorrência na base de dados.

- **Cliente**
  - Chave Primária: ID_Cliente
    Identifica unicamente o cliente. Por uma questão de simplicidade, face às chaves candidatas existentes, opta-se por criar um código identificador único para cada cliente dentro da própria BD.

  - Chave Alternativa: username
    O username é característico de cada cliente, não podendo existir usernames iguais entre diferentes clientes, no entanto, como é uma *String* não satisfaz as nossas necessidades como chave primária, uma vez que é difícil a utilização de chaves deste género.

  - Chave Alternativa: E-mail
    O endereço de e-mail identifica unicamente o seu titular. Normalmente um e-mail é único e intransmissível, embora possam ocorrer situações de partilha, o que poderia gerar situações de conflito na identificação de um cliente.

  - Chave Alternativa: Telemóvel
    O número de telemóvel de um cliente, identifica de forma inequívoca o cliente. No entanto, o número de telemóvel pode sofrer alterações ou ser atribuído a outra pessoa, deixando de identificar o cliente.

- **Reserva**
  - Chave Primária: ID_Reserva
    Por uma questão de simplicidade e inexistência de chaves candidatas, opta-se por criar um código identificador único para cada Reserva.

- **Comboio**
  - Chave Primária: ID_Comboio
    A inexistência de chaves candidatas obriga à criação de um código identificador único para cada comboio.

- **Viagem**
  - Chave Primária: ID_Viagem
    Uma vez mais, dada a falta de chaves candidatas, associa-se um código identificador único para cada viagem.

4.7. Validação com as transações do utilizador

Dado o Modelo Conceptual é necessário verificar que é possível concretizar as transações de dados efetuadas, tanto pelos utilizadores como pelos gestores. Estas transações são nada mais que as queries referidas anteriormente, as quais se pretendem que a aplicação seja capaz de dar resposta e que deverão ser suportadas pelo Modelo Conceptual, de forma a confirmar a sua validade.

Assim, ao longo desta secção, será apresentada a descrição de cada uma destas, confirmando, individualmente, a validade de cada uma, elucidando como é que as diferentes Entidades, Atributos e Relacionamentos do nosso Modelo Conceptual interagem de forma a obter o resultado pretendido em cada uma destas transações.

- **Transações do utilizador**

    - *Consultar reservas feitas por um determinado cliente.*
      As reservas feitas por um cliente podem ser consultadas através do relacionamento entre as entidades Cliente e Reserva.

    - *Consultar horário de viagens entre duas cidades.*
      Um utilizador pode consultar o horário de viagens entre duas cidades, introduzindo estas na aplicação. Assim, fazendo uma seleção nos atributos compostos Partida e Chegada da entidade Viagem, em que as cidades de partida e chegada sejam correspondentes com as dadas pelo utilizador, é possível fazer uma listagem do atributo Hora, no atributo composto Partida, na mesma entidade, obtendo assim o horário das viagens com o percurso pretendido.

    - *Consultar lista de lugares disponíveis numa determinada viagem.*
      Dada uma determinada viagem é possível consultar a lista de lugares disponíveis, primeiramente através do relacionamento entre a entidade Reserva e a entidade Viagem. Através da consulta do atributo do Lugar em Reserva, podemos obter uma lista dos lugares ocupados nessa mesma viagem.
      O relacionamento entre as entidades Viagem e Comboio é também necessário para esta consulta. O atributo Nr Lugar, no composto Lugar, presente em Comboio, permite-nos obter uma lista dos lugares do comboio que efetua a viagem considerada. Assim, com a lista de lugares do comboio em conjunto com a lista de lugares ocupados pelas reservas já realizadas pelos clientes podemos obter a lista de lugares disponíveis na viagem considerada.

- *Apresentar lista de viagens entre duas cidades ordenado pelas viagens com menor duração.*

  Um utilizador pode consultar a lista de viagens entre dois locais, ordenado pelas de menor duração, fazendo a seleção das viagens entre dois locais, através dos atributos compostos (Partida e Chegada) presentes na entidade Viagem, tal como faria para consultar o horário de viagens entre duas cidades. Desta vez, no entanto, ao apresentar a lista de viagens é necessário ter em conta o atributo Duração, presente na mesma entidade, de maneira a que a lista esteja ordenada pelas viagens de menor duração.

- **Transações do gestor**

  - *Consultar lista de utilizadores/viagens/comboios presentes na aplicação.*

    A lista com todos os clientes, viagens ou comboios presentes na aplicação pode ser obtida através das entidades Cliente, Viagem ou Comboio, respetivamente. Esta lista será apresentada ordenadamente pelo atributo ID associado a cada uma das entidades.

  - *Apresentar todas as reservas de uma dada viagem, por ordem alfabética do nome do titular da reserva.*

    De maneira a ser possível apresentar as reservas de uma dada viagem por ordem alfabética do nome do titular da reserva, é necessário ter em conta o relacionamento entre as entidades Reserva e Viagem, de maneira a saber todas as reservas efetuadas para uma determinada viagem, e ainda, o relacionamento entre as entidades Cliente e Reserva, pois através do atributo Nome na entidade Cliente conseguimos obter o nome do titular da reserva, sendo assim capazes de construir uma lista ordenada alfabeticamente.

  - *Apresentar lista de todas as viagens, por ordem crescente da duração.*

    Para apresentar a lista de todas as viagens, por ordem crescente da duração, necessitamos obrigatoriamente de aceder ao atributo Duração da entidade Viagem. A Duração é um atributo derivado, isto é, podemos obtê-lo através da diferença dos atributos Hora em cada um dos atributos compostos da entidade Viagem (Partida e Chegada). Desta forma, a lista pretendida é obtida ordenando as viagens pela sua duração.

- *Apresentar lista de todos os comboios, por ordem decrescente da capacidade.*
  Os detalhes dos comboios relativos às capacidades são obtidos através do atributo multi-valor Lugar, existente na entidade Comboio. A capacidade, como é um atributo derivado, o seu valor é obtido contabilizando os lugares existentes no comboio. Assim, determinamos a capacidade de um comboio, e podemos efetivamente fazer uma lista de todos os comboios existentes, ordenados pela sua capacidade.

5. Modelo Lógico de Dados

5.1. Diagrama do Modelo Lógico

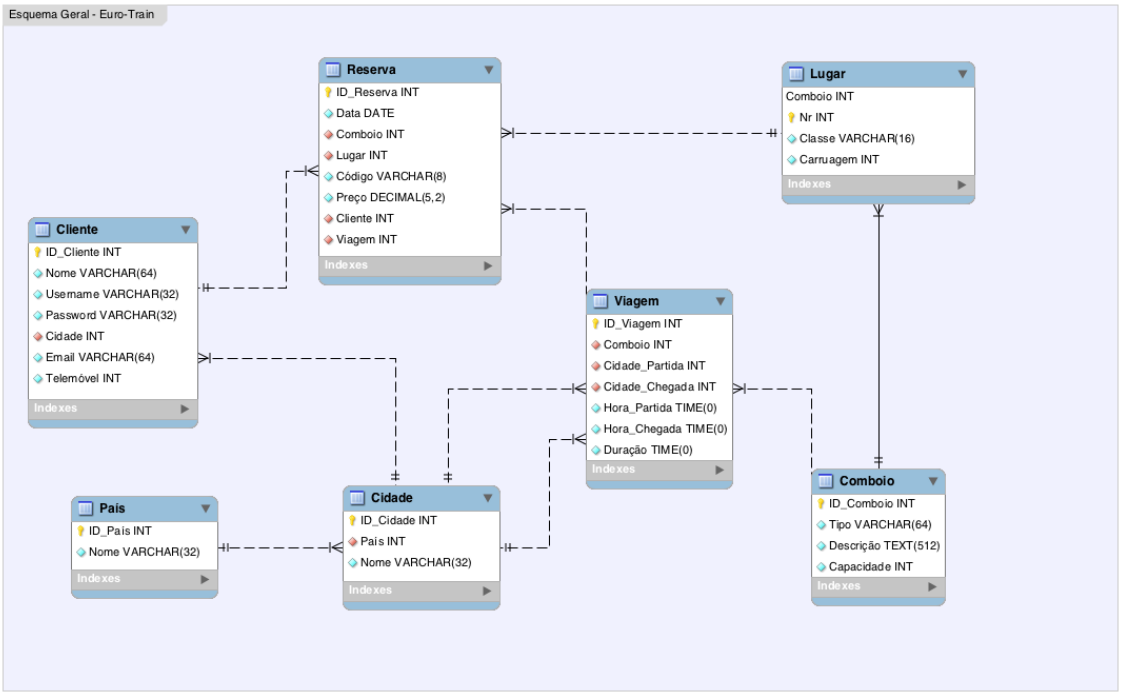

Figura 2 – Diagrama do Modelo Lógico.

5.2. Transição do Modelo Conceptual para o Lógico

Ao longo deste capítulo abordaremos as diferentes decisões tomadas na transição do Modelo Conceptual, apresentado no capítulo anterior, para o Modelo Lógico. Esta transição foi relativamente simples para o grupo, cumprindo com as regras essenciais à mesma em praticamente todas as situações.

Houve, no entanto, duas situações em que a tradução do Modelo Conceptual para o Lógico não foi direta, tendo acontecido nos atributos Cidade e País, presentes nas entidades Cliente e Viagem e no atributo multi-valor Lugar da entidade Comboio.

Considerando que os dois primeiros atributos têm uma óbvia relação, uma vez que um país é formado por um conjunto de diversas cidades, e ainda com o objetivo de manter os dados o máximo consistentes possível, isto é, evitando diversas formas de identificar uma cidade ou um país, optamos por transformar estes em tabelas País e Cidade, adicionando um relacionamento 1:N entre estas. Esta situação obrigou à existência de um relacionamento das tabelas que incluíam os atributos em questão, com a tabela Cidade, que por si já possui um relacionamento com a tabela País.

A segunda situação foi na entidade Comboio que, contendo um atributo multi-valor, obrigou à criação de uma tabela capaz de representar este atributo, o que não apresenta nada fora do regular. No entanto, o facto de na entidade Reserva estarem presentes atributos indicando o lugar, classe e carruagem onde o cliente iria desfrutar da sua viagem, fez com que, de maneira a simplificar o modelo e ainda a economizar uns meros *bytes* por reserva, fosse tomada a decisão de criar um relacionamento 1:N entre a tabela Lugar e a tabela Reserva.

Apesar destas duas alterações realizadas na transição dos modelos, o propósito do Modelo Conceptual foi mantido no Modelo Lógico.

5.3. Derivação de Relacionamentos

- **Entidades Fortes**

Uma Entidade Forte é caracterizada por possuir uma chave primária que a identifica inequivocamente e não apresenta dependência de outras chaves. Todos os atributos simples são considerados e incluídos no relacionamento. Para atributos compostos, considera-se apenas os atributos simples que os constituem.

Cliente (ID_Cliente, Nome, Username, Password, E-mail, Telemóvel)
Chave Primária: ID_Cliente

Reserva (ID_Reserva, Preço, Data, Código)
Chave Primária: ID_Reserva

Viagem (ID_Viagem, Duração, Hora_Partida, Hora_Chegada)
Chave Primária: ID_Viagem

Comboio (ID_Comboio, Tipo, Descrição, Capacidade)
Chave Primária: ID_Comboio

- **Relacionamentos Um-Para-Muitos (1:N)**

De um relacionamento 1:N é derivada uma cópia da chave primária da Entidade de menor cardinalidade e colocada na Entidade de maior cardinalidade. Esta cópia é designada por chave estrangeira, garantido assim a integridade dos dados referenciais.

Cliente (ID_Cliente, Nome, Username, Password, Cidade, E-mail, Telemóvel)
Chave Primária: ID_Cliente
Chave Estrangeira Cidade **referência** Cidade (ID_Cidade)

Reserva (ID_Reserva, Data, Comboio, Código, Preço, Lugar, Cliente, Viagem)
Chave Primária: ID_Reserva
Chave Estrangeira Cliente **referência** Cliente (ID_Cliente)
Chave Estrangeira Viagem **referência** Viagem (ID_Viagem)
Chave Estrangeira Comboio, Lugar **referência** Lugar (Comboio, Nr)

Viagem (ID_Viagem, Duração, Hora_Partida, Hora_Chegada, Comboio, Cidade_Partida, Cidade_Chegada)
Chave Primária ID_Viagem
Chave Estrangeira Comboio **referência** Comboio (ID_Comboio)
Chave Estrangeira Cidade_Partida **referência** Cidade(ID_Cidade)
Chave Estrangeira Cidade_Chegada **referência** Cidade(ID_Cidade)

Cidade (ID_Cidade, Nome, País)
Chave Primária ID_Cidade
Chave Estrangeira País **referência** País (ID_País)

Lugar (Comboio, Nr, Classe, Carruagem)
Chave Primária: Comboio, Nr
Chave Estrangeira Comboio **referência** Comboio (ID_Comboio)

- **Atributos Multi-Valor**

Um atributo multi-valor numa Entidade origina um novo relacionamento de forma a representar o atributo em questão e a chave primária da respetiva Entidade. Deste relacionamento obtemos uma chave primária composta pela chave estrangeira proveniente da Entidade e o próprio atributo multi-valor. Caso o atributo multi-valor seja composto os atributos simples que o constituem é que são considerados no relacionamento.

Lugar (Nr_Lugar, Comboio, Classe, Carruagem)
Chave Primária Nr_Lugar, Comboio

Chave Estrangeira Comboio **referência** Comboio (id_Comboio)

5.4. Validação através da Normalização

Para fazer uso da normalização é necessário que em cada relacionamento, se identifiquem as dependências funcionais existentes entre os atributos. Um relacionamento designa-se de dependência funcional quando um atributo determina exclusivamente outro atributo. Desta forma, é possível indicar as seguintes dependências funcionais:

- **Cliente**

  ID_Cliente → Nome, Username, Password, Cidade, E-mail, Telemóvel

- **Reserva**

  ID_Reserva → Data, Comboio, Lugar, Código, Preço, Cliente, Viagem

- **Viagem**

  ID_Viagem → Cidade_Partida, Hora_Partida, Cidade_Chegada, Hora_Chegada, Comboio

- **Comboio**

  ID_Comboio → Tipo, Descrição, Capacidade

Para além disso, este processo de normalização é constituído por um conjunto sequencial de passos que têm como finalidade verificar se os atributos estão ou não em conformidade com as formas normais, sendo que estas formas são como orientações para a representação de bons relacionamentos.

- **Primeira Forma Normal (1FN):**

  Um relacionamento está na Primeira Forma Normal, se os valores de todos os atributos forem atómicos, isto é, se não for possível decompô-los, e não existir conjuntos de atributos repetidos que descrevam a mesma característica.

- **Segunda Forma Normal (2FN):**

  Um relacionamento está na Segunda Forma Normal, se estiver na Primeira Forma Normal e se todos os atributos não-chave dependem da totalidade da chave primária.

- **Terceira Forma Normal (3FN):**

  Um relacionamento está na Terceira Forma Normal, se está na Segunda Forma Normal e nenhum atributo não-chave é transitivamente dependente da chave primária.

No Modelo Conceptual em estudo, os relacionamentos multi-valor usados para guardar os registos de números dos lugares de um comboio, poderiam ir contra a Primeira Forma Normal, caso se mantivessem nas mesmas tabelas que as entidades aos quais estes correspondem. De forma a normalizar estas situações, criou-se a tabela Lugar evitando, deste modo, as repetições de atributos para descrever as várias entradas dos lugares de um comboio.

Com todos os relacionamentos a respeitarem a Primeira Forma Normal, pode-se verificar também se respeitam a Segunda Forma Normal. Neste caso, verifica-se que para todos os relacionamentos existe uma dependência total dos atributos simples em relação às chaves primárias, isto é, não existem dependências parciais de chaves candidatas que possa causar redundância de informação.

Para os relacionamentos respeitarem a Terceira Forma Normal, tem que se verificar que não existem dependências transitivas de atributos não-chave em relação à chave primária. O nosso modelo Lógico não apresenta nenhuma dependência transitiva. Um exemplo que podia tornar um caso de desrespeito à Terceira Forma Normal, eram os atributos derivados. No entanto, para o caso do atributo *duração*, a dependência existente é unicamente uma dependência de cálculo, o que não corresponde a uma dependência funcional. Para além disso, o atributo duração apresenta uma dependência funcional, para a chave primária ID_Viagem. Outro exemplo, é o atributo derivado *capacidade*, que corresponde ao mesmo caso anterior. Desta forma, não existe um desrespeito da Terceira Forma Normal.

Uma vez que, para evitar problemas de redundância de dados é apenas necessário que cada relacionamento esteja na Terceira Forma Normal, não é obrigatório que as demais formas normais sejam verificadas.

5.5. Validação com as Transações do Utilizador

Ao nível do Modelo Lógico, validamos as transações do utilizador através de mapas de transações. Desta forma, é possível verificar visualmente as interações entre as diferentes Entidades, Atributos e Relacionamentos de forma a obter o resultado pretendido.

- **Inserção de um Lugar**

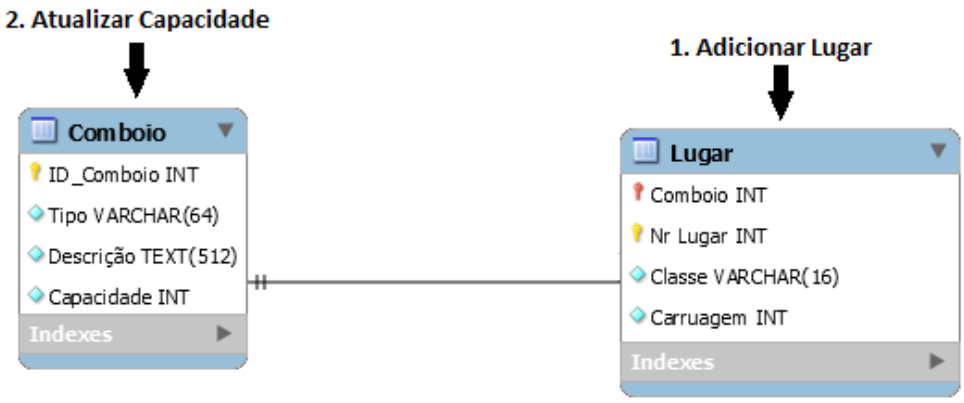

Figura 3 - Mapa de transação de uma inserção de um Lugar.

Quando é adicionado um lugar ao comboio, para além de ser necessário fazer inserção deste na tabela Lugar, é também necessário atualizar o atributo Capacidade presente na tabela Comboio.

- **Remoção de um Lugar**

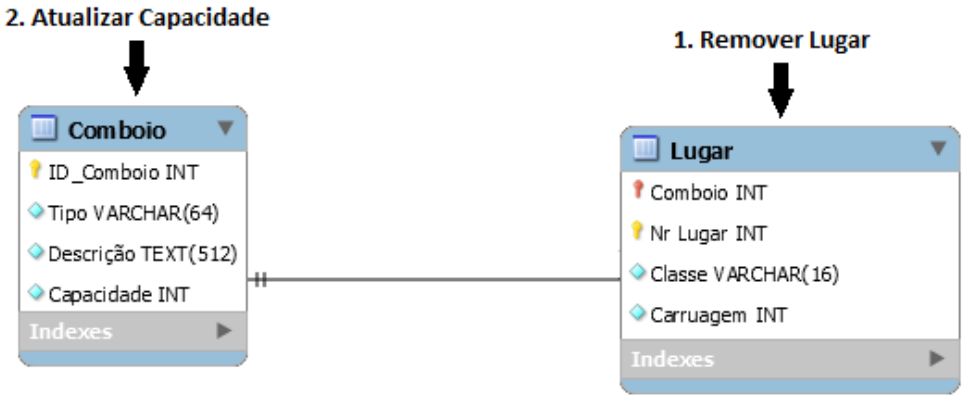

Figura 4 - Mapa de transação de uma remoção de um Lugar.

Quando é removido um lugar do comboio, para além de ser necessário fazer remoção deste na tabela Lugar é, tal como no exemplo anterior, necessário atualizar o atributo Capacidade presente na tabela Comboio.

5.6. Verificação da Integridade do Modelo Lógico

Com o objetivo de proteger a BD por forma a que esta não se torne inconsistente, incompleta ou incoerente, é necessário que todas as seguintes restrições de integridade sejam verificadas no Modelo Lógico.

- **Dados Necessários**

Numa Base de Dados, na inserção de registos, alguns campos são de preenchimento obrigatório e devem sempre conter um valor válido. As chaves primárias são um exemplo de registos que não podem conter valor nulos embora possam existir mais alguns exemplos, dependendo do contexto do problema e da sua importância.

Nomeadamente, na nossa Base de Dados, todos os campos são de preenchimento obrigatório, não existindo nenhum caso de exceção.

- **Restrições do Domínio de Atributos**

Todos os atributos da Base de Dados têm um domínio de valores possíveis, dentro do contexto do problema. As restrições definidas no Modelo Conceptual transitam para o Modelo Lógico, com as restrições adicionais dos tipos de dados inerentes à linguagem em que a Base de Dados é descrita.

Nesta secção, descreve-se os tipos de dados associados aos campos de valores das tabelas resultantes da conversão do Modelo Conceptual em Lógico.

Cliente

- ID_Cliente - Valor do tipo INT.
- Nome - Valor do tipo VARCHAR com limite de tamanho até 64 caracteres.
- Username - Valor do tipo VARCHAR com limite de tamanho até 32 caracteres.
- Password - Valor do tipo VARCHAR com limite de tamanho até 32 caracteres.
- Cidade - Valor do tipo INT.
- E-mail - Valor do tipo VARCHAR com limite de tamanho até 64 caracteres.
- Telemóvel - Valor do tipo INT.

Reserva

- ID_Reserva - Valor do tipo INT
- Data - Valor do tipo DATE.

- Lugar - Valor do tipo INT.
- Comboio - Valor do tipo INT.
- Código - Valor do tipo VARCHAR com limite de tamanho até 8 caracteres.
- Preço - Valor do tipo DECIMAL com limite de 2 casas decimais, e 3 dígitos para a parte inteira.
- Cliente - Valor do tipo INT.
- Viagem - Valor do tipo INT.

Viagem

- ID_Viagem - Valor do tipo INT.
- Hora_Partida - Valor do tipo TIME com 0 dígitos para a parte fracionaria dos segundos.
- Hora_Chegada - Valor do tipo TIME com 0 dígitos para a parte fracionaria dos segundos.
- Comboio - Valor do tipo INT.
- Cidade_Partida - Valor do tipo INT.
- Cidade_Chegada - Valor do tipo INT.

Comboio

- ID_Comboio - Valor do tipo INT.
- Tipo - Valor do tipo VARCHAR com limite de tamanho de 32 caracteres. Só admite as seguintes sequências de caracteres: 'Alfa', 'Intercidades', 'Internacionais'.
- Descrição - Valor do tipo TEXT com tamanho até 512 caracteres.

Lugar

- Nr_Lugar - Valor do tipo INT.
- Comboio - Valor do tipo INT.
- Classe - Valor do tipo VARCHAR com limite de tamanho até 16 caracteres. Só admite as seguintes sequencias de caracteres: 'Conforto' e 'Turística'.
- Carruagem - Valor do tipo INT.

- **Multiplicidade**

Com a passagem do Modelo Conceptual para o Lógico, a multiplicidade deve ser respeitada e mantida, nos diversos relacionamentos que compõem a Base de Dados.

No caso do relacionamento entre Cliente e Reserva, um Cliente pode efetuar várias reservas, mas uma Reserva é apenas efetuada por um Cliente. No Modelo Lógico, este

relacionamento é garantido através do uso de uma chave estrangeira Cliente na tabela Reserva.

O mesmo acontece no relacionamento entre a Reserva e a Viagem. Uma viagem tem correspondência a várias reservas, no entanto uma reserva apenas corresponde a uma viagem. Este relacionamento é garantido, da mesma forma como o anterior, isto é, com o uso de uma chave estrangeira Viagem na tabela Reserva.

Quanto ao relacionamento entre a Reserva e o Lugar, uma Reserva possui um lugar, mas um lugar pode estar associado a várias Reservas, logo o relacionamento cria duas chaves estrangeiras, Comboio e Lugar (chave primária composta da tabela Lugar), na tabela Reserva.
Para o relacionamento entre o País e a Cidade, um País possui várias cidades, no entanto uma Cidade existe apenas em um País. Este relacionamento cria uma chave estrangeira País na tabela Cidade.

No relacionamento entre a Viagem e a Cidade, uma Viagem possui uma cidade de partida e uma cidade de chegada, mas cada cidade pode ter várias viagens associadas a ela. Assim, para garantir o relacionamento, insere-se duas chaves estrangeiras, Cidade_Partida e Cidade_Chegada, na tabela Viagem.

Quanto ao relacionamento entre o Comboio e a Viagem, um Comboio pode realizar várias viagens, mas uma Viagem é efetuada por um Comboio apenas. Da mesma forma que as relações anteriores, através do uso de uma chave estrangeira Comboio na tabela Viagem, garantimos o relacionamento.

- **Integridade das Entidades**

Todas as entidades têm necessidade de serem identificadas inequivocamente, sendo tal possível através da utilização de chaves primárias. Para além disso, é de referir que as chaves primárias são obrigadas a respeitar a restrição de apenas aceitar valores diferentes de nulo, de forma a que seja possível estabelecer relações entre as tabelas da Base de Dados.

Desta forma, o nosso Modelo Lógico assegura o cumprimento das regras de integridade das Entidades e, transitivamente, da informação guardada nas tabelas correspondentes.

- **Integridade Referencial**

Uma chave estrangeira é um atributo ou conjunto de atributos, cujos valores aparecem necessariamente na chave primária de uma tabela, permitindo a existência de relacionamentos entre tabelas. A integridade referencial consiste em garantir a existência de um valor na chave estrangeira de modo a que não se perca o relacionamento com a tabela ao qual está ligada.

É necessário, então, definir restrições de existência que explicitem o valor da chave estrangeira quando ocorre uma ação – inserção, atualização e remoção – no registo ao qual esta está associada.

Cliente (ID_Cliente, Nome, Username, Password, E-mail, Telemóvel, Cidade)

Chave Primária: ID_Cliente

Chave Estrangeira Cidade **referência** Cidade (ID_Cidade)

ON UPDATE CASCADE ON DELETE NO ACTION

Reserva (ID_Reserva, Preço, Data, Código, Lugar, Carruagem, Classe, Cliente, Viagem)

Chave Primária: ID_Reserva

Chave Estrangeira Cliente **referência** Cliente (ID_Cliente)

ON UPDATE CASCADE ON DELETE NO ACTION

Chave Estrangeira Viagem **referência** Viagem (ID_Viagem)

ON UPDATE CASCADE ON DELETE NO ACTION

Chave Estrangeira Comboio, Lugar **referência** Lugar (Comboio, Nr)

ON UPDATE CASCADE ON DELETE NO ACTION

Viagem (ID_Viagem, Duração, Hora_Partida, Hora_Chegada, Comboio, Cidade_Partida, Cidade_Chegada)

Chave Primária ID_Viagem

Chave Estrangeira Comboio **referência** Comboio (ID_Comboio)

ON UPDATE CASCADE ON DELETE NO ACTION

Chave Estrangeira Cidade_Partida **referência** Cidade(ID_Cidade)

ON UPDATE CASCADE ON DELETE NO ACTION

Chave Estrangeira Cidade_Chegada **referência** Cidade(ID_Cidade)

ON UPDATE CASCADE ON DELETE NO ACTION

Cidade (ID_Cidade, Nome, País)

Chave Primária ID_Cidade

Chave Estrangeira País **referência** País (ID_País)

ON UPDATE CASCADE ON DELETE CASCADE

Lugar (Nr_Lugar, Comboio, Classe, Carruagem)

Chave Primária Nr_Lugar, Comboio

Chave Estrangeira Comboio **referência** Comboio (id_Comboio)

ON UPDATE CASCADE ON DELETE CASCADE

5.7. Viabilidade de Crescimento Futuro

A longevidade de um sistema de bases de dados depende, sobretudo, da sua capacidade em se adaptar a novos requisitos funcionais que surjam. Situações onde um Modelo Lógico só consegue suportar os requisitos para o qual foi projetado podem tornar o sistema rapidamente obsoleto ou fazer com que o custo de implementação das alterações necessárias seja demasiado dispendioso.

O modelo apresentado neste trabalho está limitado ao âmbito da Reserva de viagens em comboios nacionais e internacionais. Contudo, é possível imaginar novos requisitos, dentro do contexto em estudo que seriam facilmente implementáveis e tornariam a base de dados ainda mais útil.

A Reserva de viagens não tem de ser restrita apenas a clientes, podendo abranger também os funcionários. Ou seja, para além de guardar o registo de clientes que fazem uma determinada viagem num determinado comboio, a base de dados também seria capaz de registar os funcionários que estariam em serviço durante essa mesma viagem e as zonas do comboio pelas quais estes seriam responsáveis. Os funcionários abrangem tanto os condutores como os revisores.

6. Modelo Físico de Dados

6.1. Tradução do Modelo Lógico para um SGBD

O processo de modelação do esquema físico de uma BD envolve a tradução dos relacionamentos definidos no Modelo Lógico, num pseudocódigo que possa ser implementado no SGBD. Assim, procederemos à representação das relações base, dispondo informação que diga respeito a domínios, valores por defeito, valores nulos e ainda à representação de dados derivados.

Relações base

- **Relação Cliente**

  Domínio ID_Cliente: Inteiro
  Domínio Nome: String de tamanho variável, tamanho 64
  Domínio Username: String de tamanho variável, tamanho 32
  Domínio Password: String de tamanho variável, tamanho 32
  Domínio Email: String de tamanho variável, tamanho 64
  Domínio Cidade: Inteiro
  Domínio Telemóvel: Inteiro

  Cliente(

ID_Cliente	ID_Cliente	NOT NULL,
Nome	Nome	NOT NULL,
Username	Username	NOT NULL,
Password	Password	NOT NULL,
Email	Email	NOT NULL,
Cidade	Cidade	NOT NULL,
Telemóvel	Telemóvel	NOT NULL,

  PRIMARY KEY (ID_Cliente),
  FOREIGN KEY (Cidade) **REFERENCES** Cidade(ID_Cidade)
  ON UPDATE CASCADE ON DELETE NO ACTION);

- **Relação Reserva**

  Domínio ID_Reserva: Inteiro
  Domínio Data: Data, formato: *AAAA-MM-DD*
  Domínio Código: String de tamanho fixo, tamanho 8
  Domínio Preço: Decimal
  Domínio Cliente: Inteiro
  Domínio Viagem: Inteiro
  Domínio Comboio: Inteiro
  Domínio Lugar: Inteiro

  Reserva(

ID_Reserva	ID_Reserva	NOT NULL,
Data	Data	NOT NULL,
Codigo	Codigo	NOT NULL,
Preço	Preço	NOT NULL,
Cliente	Cliente	NOT NULL,
Viagem	Viagem	NOT NULL,
Comboio	Comboio	NOT NULL,
Lugar	Lugar	NOT NULL,

  PRIMARY KEY (ID_Reserva),
  FOREIGN KEY (Cliente) **REFERENCES** Cliente(ID_Cliente)
  ON UPDATE CASCADE ON DELETE NO ACTION,
  FOREIGN KEY (Viagem) **REFERENCES** Viagem(ID_Viagem)
  ON UPDATE CASCADE ON DELETE NO ACTION,
  FOREIGN KEY (Comboio, Lugar) **REFERENCES** Lugar (Comboio, Nr)
  ON UPDATE CASCADE ON DELETE NO ACTION);

- **Relação Viagem**

  Domínio ID_Viagem: Inteiro
  Domínio Comboio: Inteiro
  Domínio Cidade_Partida: Inteiro
  Domínio Cidade_Chegada: Inteiro
  Domínio Hora_Partida: Hora, formato *HH:MM:SS*
  Domínio Hora_Chegada: Hora, formato *HH:MM:SS*

  Viagem(

ID_Viagem	ID_Viagem	NOT NULL,
Comboio	Comboio	NOT NULL,
Cidade_Partida	Cidade_Partida	NOT NULL,

Cidade_Chegada Cidade_Chegada NOT NULL,
Hora_Partida Hora_Partida NOT NULL,
Hora_Chegada Hora_Chegada NOT NULL,
PRIMARY KEY (ID_Viagem),
FOREIGN KEY (Comboio) **REFERENCES** Comboio(ID_Comboio),
ON UPDATE CASCADE ON DELETE NO ACTION,
FOREIGN KEY (Cidade_Partida**) REFERENCES** Cidade(ID_Cidade),
ON UPDATE CASCADE ON DELETE NO ACTION,
FOREIGN KEY (Cidade_Chegada) **REFERENCES** Cidade(ID_Cidade),
ON UPDATE CASCADE ON DELETE NO ACTION);

- **Relação Comboio**

Domínio ID_Comboio: Inteiro
Domínio Tipo: String de tamanho variável, tamanho 64
Domínio Descrição: Texto de tamanho variável, tamanho 512
Domínio Capacidade: Inteiro

Comboio(
ID_Comboio ID_Comboio NOT NULL,
Tipo Tipo NOT NULL,
Descrição Descrição NOT NULL,
Capacidade Capacidade NOT NULL,
PRIMARY KEY (ID_Comboio));

- **Relação Lugar**

Domínio Comboio: Inteiro
Domínio Nr: Inteiro
Domínio Classe: String de tamanho variável, tamanho 16
Domínio Carruagem: Inteiro

Lugar(
Comboio Comboio NOT NULL,
Nr Nr NOT NULL,
Classe Classe NOT NULL,
Carruagem Carruagem NOT NULL,
PRIMARY KEY(Comboio),
PRIMARY KEY(Nr)
FOREIGN KEY (Comboio) **REFERENCES** Comboio(ID_Comboio),
ON UPDATE CASCADE ON DELETE CASCADE);

- **Relação Cidade**

  Domínio ID_Cidade: Inteiro
  Domínio País: Inteiro
  Domínio Nome: String de tamanho variável, tamanho 32

  Cidade(
  ID_Cidade ID_Cidade NOT NULL,
  País País NOT NULL,
  Nome Nome NOT NULL,
  PRIMARY KEY(ID_Cidade),
  FOREIGN KEY(País) **REFERENCES** País(ID_País)
  ON UPDATE CASCADE ON DELETE CASCADE);

- **Relação País**

  Domínio ID_País: Inteiro
  Domínio Nome: String de tamanho variável, tamanho 32

  País(
  ID_País ID_País NOT NULL,
  Nome Nome NOT NULL,
  PRIMARY KEY(ID_País));

Representação dos dados derivados

Dados derivados correspondem a atributos cujo valor pode ser calculado através de outros atributos já existentes. No nosso SGBD, tal como no Modelo Lógico, existem apenas dois destes atributos (*Duração* e *Capacidade*). Vamos considerar, no entanto, apenas o cálculo efetuado pelo sistema no atributo Capacidade, uma vez que para o atributo Duração é a própria aplicação que o faz, sendo o cálculo exterior ao SGBD.

O cálculo da Capacidade é realizado sempre que há adição ou remoção de um lugar de um determinado comboio.

- Adição de um lugar

```
DELIMITER $$
CREATE TRIGGER ActCapacidadeInc
    AFTER INSERT ON Lugar
    FOR EACH ROW
BEGIN
    UPDATE Comboio
        SET Capacidade = Capacidade + 1
        WHERE ID_Comboio = NEW.Comboio;
END $$
```

Figura 5 - Script do Trigger de atualização da Capacidade (inserção).

- Remoção de um lugar

```
DELIMITER $$
CREATE TRIGGER ActCapacidadeDec
    AFTER DELETE ON Lugar
    FOR EACH ROW
BEGIN
    UPDATE Comboio
        SET Capacidade = Capacidade - 1
        WHERE ID_Comboio = OLD.Comboio;
END $$
```

Figura 6 - Script do Trigger de atualização da Capaciade (remoção).

6.2. Criação do Modelo Físico

Para gerar o esquema físico foi utilizada a funcionalidade *Forward Engineer* da ferramenta *MySQL Workbench*. Desta maneira, o código gerado corresponde ao Modelo Lógico anteriormente elaborado.

De seguida será apresentado o código equivalente às tabelas mais relevantes. O script de criação completo encontra-se em anexo.

```sql
-- -----------------------------------------------------
-- Table `EuroTrain`.`Cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Cliente` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Cliente` (
  `ID_Cliente` INT NOT NULL,
  `Nome` VARCHAR(64) NOT NULL,
  `Username` VARCHAR(32) NOT NULL,
  `Password` VARCHAR(32) NOT NULL,
  `Cidade` INT NOT NULL,
  `Email` VARCHAR(64) NOT NULL,
  `Telemóvel` INT NOT NULL,
  PRIMARY KEY (`ID_Cliente`),
  INDEX `fk_Cliente_Cidade1_idx` (`Cidade` ASC),
  CONSTRAINT `fk_Cliente_Cidade1`
    FOREIGN KEY (`Cidade`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 7 - Criação da tabela Cliente em SQL.

```sql
-- -----------------------------------------------------
-- Table `EuroTrain`.`Comboio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Comboio` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Comboio` (
  `ID_Comboio` INT NOT NULL,
  `Tipo` VARCHAR(64) NOT NULL,
  `Descrição` TEXT(512) NOT NULL,
  `Capacidade` INT NOT NULL,
  PRIMARY KEY (`ID_Comboio`))
ENGINE = InnoDB;


-- -----------------------------------------------------
```

Figura 8 - Criação da tabela Comboio em SQL.

```sql
-- -----------------------------------------------------
-- Table `EuroTrain`.`Lugar`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Lugar` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Lugar` (
  `Comboio` INT NOT NULL,
  `Nr` INT NOT NULL,
  `Classe` VARCHAR(16) NOT NULL,
  `Carruagem` INT NOT NULL,
  PRIMARY KEY (`Comboio`, `Nr`),
  CONSTRAINT `fk_Lugar_Comboio1`
    FOREIGN KEY (`Comboio`)
    REFERENCES `EuroTrain`.`Comboio` (`ID_Comboio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 9 - Criação da tabela Lugar em SQL.

```sql
-- -----------------------------------------------------
-- Table `EuroTrain`.`Reserva`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Reserva` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Reserva` (
  `ID_Reserva` INT NOT NULL,
  `Data` DATE NOT NULL,
  `Comboio` INT NOT NULL,
  `Lugar` INT NOT NULL,
  `Código` VARCHAR(8) NOT NULL,
  `Preço` DECIMAL(5,2) NOT NULL,
  `Cliente` INT NOT NULL,
  `Viagem` INT NOT NULL,
  PRIMARY KEY (`ID_Reserva`),
  INDEX `fk_Reserva_Cliente_idx` (`Cliente` ASC),
  INDEX `fk_Reserva_Viagem1_idx` (`Viagem` ASC),
  INDEX `fk_Reserva_Lugar1_idx` (`Comboio` ASC, `Lugar` ASC),
  CONSTRAINT `fk_Reserva_Cliente`
    FOREIGN KEY (`Cliente`)
    REFERENCES `EuroTrain`.`Cliente` (`ID_Cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Reserva_Viagem1`
    FOREIGN KEY (`Viagem`)
    REFERENCES `EuroTrain`.`Viagem` (`ID_Viagem`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Reserva_Lugar1`
    FOREIGN KEY (`Comboio` , `Lugar`)
    REFERENCES `EuroTrain`.`Lugar` (`Comboio` , `Nr`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 10 - Criação da tabela Reserva em SQL.

```sql
-- -----------------------------------------------------
-- Table `EuroTrain`.`Viagem`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Viagem` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Viagem` (
  `ID_Viagem` INT NOT NULL,
  `Comboio` INT NOT NULL,
  `Cidade_Partida` INT NOT NULL,
  `Cidade_Chegada` INT NOT NULL,
  `Hora_Partida` TIME(0) NOT NULL,
  `Hora_Chegada` TIME(0) NOT NULL,
  `Duração` TIME(0) NOT NULL,
  PRIMARY KEY (`ID_Viagem`),
  INDEX `fk_Viagem_Comboio1_idx` (`Comboio` ASC),
  INDEX `fk_Viagem_Cidade1_idx` (`Cidade_Partida` ASC),
  INDEX `fk_Viagem_Cidade2_idx` (`Cidade_Chegada` ASC),
  CONSTRAINT `fk_Viagem_Comboio1`
    FOREIGN KEY (`Comboio`)
    REFERENCES `EuroTrain`.`Comboio` (`ID_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Cidade1`
    FOREIGN KEY (`Cidade_Partida`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Cidade2`
    FOREIGN KEY (`Cidade_Chegada`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;
```

Figura 11 - Criação da tabela Viagem em SQL.

6.3. Análise das Transações

De modo a tratar o Modelo Físico de forma eficiente, é necessário conhecer quais as transações ou queries que irão executar na nossa BD. Desta forma, é necessário realizar uma análise das transações para obter uma noção da carrega que as mesmas efetuam sobre a BD. Nesta análise, vamos nos focar na média, e nos instantes de tempo em que é atingido o pico de utilização da BD onde haverá um impacto significativo no desempenho.

Para uma melhor análise, vamos nos centrar em uma das transações que mais impacto tem no sistema, que é a da inserção de uma reserva no sistema. Este tipo de inserção é realizado diariamente, em grande volume e aumenta de forma exponencial com o número de viagens e clientes registados no sistema.

Transação - Inserção de uma Reserva no Sistema de Reservas da Euro-Train.
Volume de Transação:

- **Média:** 42 por Hora
- **Pico:** 64 por Hora {entre as 18h00 e as 20h00, todos os Dias}

INSERT INTO Reserva (ID_Reserva, Data, Comboio, Lugar, Código, Preço, Cliente, Viagem)
VALUES (3,'2016-12-26', 1, 3, '47k6twyj', 14.90, 3,1);

6.4. Controlo de concorrência

Quando o SGBD apenas suporta um utilizador de cada vez, o problema de concorrência é inexistente, no entanto, o desempenho não é nada favorável num ambiente deste tipo.

A implementação de ambientes de multi-utilizadores proporciona enormes vantagens quer no desempenho do sistema, quer no número de utilizadores a serem servidos pela BD, porém pode apresentar problemas de concorrência no futuro.

Para analisar o problema da Concorrência, vamos considerar as transações que operam sobre o SGBD. Uma transação pode corresponder a uma sequência de comandos que um utilizador autorizado ou um programa executa, com acesso à BD. Para preservar a integridade dos dados o SGBD tem de assegurar 4 características:

• *Atomicidade* – de uma transação, isto é, a transação é realizada na sua totalidade ou não é realizada.

• *Consistência* – a execução de uma transação sem nenhuma outra transação a executar concorrentemente, preserva a consistência da base de dados.

• *Isolamento* – quaisquer operações dentro de uma transação não são afetadas por operações de outras transações a decorrer concorrencialmente.

• *Durabilidade* – após uma transição terminar e confirmar os seus resultados na base de dados, o SGBD deve garantir que esses dados são persistentes.

Uma transação é sempre delimitada por ações terminais e inicia-se sempre com um comando *START TRANSACTION*, terminando com um comando *COMMIT* para confirmar as ações, ou *ROLLBACK* para interromper a transação e desfazer os efeitos das suas ações. Um exemplo de concorrência, é quando duas transações a decorrer em simultâneo, leem o valor do mesmo elemento e utilizam-no para executar uma determinada operação.

No caso da nossa BD, seria quando dois clientes querem efetuar uma reserva, e ambos selecionam o mesmo lugar (naquele exato momento estava disponível para ambos os clientes). Quando terminam, escrevem o novo valor, mas como não consideram o facto de estarem a operar sobre o mesmo valor, ou seja, poder existir interferência entre as operações de ambas as transações, o resultado final não é o que resultaria se uma transação fosse efetuada após a outra, sequencialmente.

No entanto, quando uma transação termina com sucesso, espera-se que os resultados na base de dados sejam corretos.

O controlo da concorrência é um dos componentes fundamentais de um SGBD, sem o qual não seria possível garantir tanto a integridade dos dados, como a utilização dos mesmos por vários utilizadores em simultâneo. Do ponto de vista do utilizador e da aplicação, o controlo de concorrência é responsável por lhes dar a ilusão de que têm a base de dados disponível apenas para si, sem terem de se preocupar com as ações de outros utilizadores ou aplicações. Como vimos, os acessos à base de dados são estruturados em transações. O objetivo do componente de controlo de concorrência é o de maximizar o número de transações que acedem concorrencialmente aos dados, garantido ao mesmo tempo que as transações não interferem de forma indesejável entre si.

Uma das funções importantes do controlo de concorrência é produzir um escalonamento correto das ações das diferentes transações que são iniciadas, função que é assumida pelo escalonador. O escalonador recebe as operações à medida que elas vão sendo disponibilizadas, e tem que decidir o que fazer, em tempo real, apenas com a informação que tem disponível.

No nosso caso, o problema de concorrência identificado anteriormente acontece, por exemplo, quando dois clientes tentam reservar o mesmo lugar, assim o SGBD tem de garantir que um determinado lugar só pode ser acedido para reserva por apenas um cliente. Para garantir isto, é necessário garantir que apenas uma transação, das várias existentes, seja processada, e que as restantes sejam bloqueadas até que esta termine. Quando a transação terminar (*COMMIT*), liberta-se as outras transações. Como é de esperar, como o lugar ficou reservado, todas as outras transações que tinham o mesmo objetivo foram interrompidas, indicando aos seus respetivos utilizadores que o lugar se encontra reservado.

6.5. Estimativa de espaço em disco necessário

Com o desenvolvimento de uma base de dados é necessário ter em conta o espaço em disco que está ou será utilizado para armazenar todos os dados que pretendemos. Cada registo de informação de uma determinada tabela ocupado espaço físico na memória, dependendo do tipo de dados a que este está associado.

De seguida será apresentada uma tabela contendo, para cada tipo de dados utilizado no nosso SGBD o tamanho que ocupa.

Data Type	Tamanho (*bytes*)
INTEGER	4
DECIMAL(5,2)	3
DATE	3
TIME	3
VARCHAR(N)	N+1*
TEXT(N)	N+2

**Todos os VARCHAR presentes no SGBD têm menos de 255 bytes, logo o tamanho é sempre N (número de bytes da string/caracteres) +1, se assim não fosse teríamos de considerar N+2.*

Tabela 4 – Tamanhos de cada um dos tipos de dados utilizados.

- **Tamanho dos registos nas tabelas da BD**

Através da tabela anterior podemos calcular o tamanho que cada registo ocupa em cada uma das tabelas da BD. Esta informação ajuda a estimar o tamanho das tabelas e como o crescimento da base de dados irá afetar o espaço utilizado em disco.

Tabela	Tamanho (*bytes*)
Cliente	208
Reserva	35
Viagem	25
Comboio	587
Lugar	29
Cidade	41
País	37

Tabela 5 – Tamanho que cada registo ocupa numa determinada tabela.

- **Tamanho inicial e escalabilidade do sistema**

Considerando um caso de estudo inicial de relativamente pequena dimensão e, fazendo consulta da tabela anteriormente apresentada, pode-se inferir facilmente um tamanho inicial para a base de dados em questão.

Assumindo um universo de cinco clientes com, em média, duas reservas feitas por cada, referentes a uma das seis viagens disponíveis. Viagens estas realizadas entre oito cidades da Europa que estão distribuídas por cinco diferentes países. Considerando ainda que estas viagens são realizadas por apenas dois comboios, nos quais cada apresenta uns meros doze lugares.

Tabela	Tamanho (*bytes*)
Cliente	208 x 5 = 1040
Reserva	35 x 10 = 350
Viagem	25 x 6 = 150
Comboio	587 x 2 = 1174
Lugar	29 x 24 = 696
Cidade	41 x 8 = 328
País	37 x 5 = 185
Total	3 923

Tabela 6 – Tamanho inicial da BD povoada.

Com estes pressupostos e sabendo o tamanho que cada registo ocupa no SGBD, conseguimos facilmente estimar o tamanho inicial da base de dados para o caso de estudo, que será, aproximadamente, *3923 Bytes (3.83 KBytes)*.

No entanto, o que é realmente importante é o impacto que estes insignificantes Bytes terão numa escala real, e de que maneira é que iriam crescer ao longo do tempo. Sendo assim, estimaremos a escalabilidade do nosso SGBD.

Tendo em conta que a Euro-Train é uma empresa de grande dimensão que atua em todas as cidades da Europa cuja a atração turística é elevada, acarretando desta maneira cerca de 50 cidades distribuídas por 20 países, algo que se pretende aumentar num futuro muito próximo.

Considerando ainda que a aplicação é relativamente recente, lançada há 3 meses, contando com uns meros 50 000 utilizadores que realizam um total de, em média, 1000 reservas por dia, isto é, 90 000 reservas desde o lançamento da aplicação.

E, por fim, tendo em conta os mais de 100 anos de existência da Euro-Train, sendo uma empresa bem desenvolvida a nível económico, pode contar com mais de 50 comboios com uma média de 200 lugares por comboio, capazes de realizar um leque de mais de 250 viagens entre os diferentes pontos turísticos de toda a Europa.

Tabela	Tamanho (*bytes*)
Cliente	208 x 50000 = 10 400 000
Reserva	35 x 90000 = 3 150 000
Viagem	25 x 250 = 6 250
Comboio	587 x 20 = 11 740
Lugar	29 x 10000 = 290 000
Cidade	41 x 50 = 2 050
País	37 x 20 = 740
Total	13 860 780

Tabela 7 – Escalonamento do tamanho da BD.

Desta maneira, nos três primeiros meses após o lançamento da aplicação o SGBD da Euro-Train contaria já com mais de *13 MBytes*. Este aumentaria consideravelmente se, em vez de três meses considerássemos um ano. Aí, os clientes iam com certeza ganhar ainda mais conhecimento da aplicação por isso consideraremos que estes quintuplicavam, quadruplicando assim as reservas, tendo em conta que o número de utilizadores inativos na aplicação iria também aumentar.

Efetuando alguns cálculos poderíamos então estimar um espaço ocupado de *65 MBytes*, correspondendo a cinco vezes mais Bytes ocupados do que primeiro trimestre do ano.

Como já sabemos a Euro-Train é uma empresa que pretende expandir-se, oferecendo assim mais viagens e entre mais cidades/países. Para isso será necessária a aquisição de ainda mais comboios, garantido um maior número de lugares disponíveis aos clientes.

Toda esta evolução proporcionará um aumento tanto ao nível de clientes como, consequentemente, ao nível de reservas feitas utilizando a aplicação da empresa.

Como podemos imaginar o aumento de espaço físico ocupado ao longo do tempo seria muito grande e acabaria por tomar dimensões que não eram esperadas alcançar através de uma SGBD com um povoamento inicial de uns insignificantes *3.83 KBytes.*

6.6. Definição das Vistas dos Utilizadores

Uma Vista de Utilizador é um mecanismo em SQL capaz de, a cada invocação, ter o seu resultado atualizado, conforme a informação correspondente à pesquisa é também atualizada.

As Vistas são utilizadas quando pretendemos um conjunto pré-definido de informação de uma ou mais tabelas presentes numa única tabela que, apesar de virtual e completamente independente da estrutura da base de dados, mostra a informação seletivamente tal como pretendemos.

De seguida serão descritos alguns exemplos de Vistas de Utilizadores do nosso SGBD.

- **Exemplos de Vistas de um Utilizador**

(a) Vista que mostra informação relativa às próximas viagens a realizarem-se no respetivo dia, assim como o tempo restante até às mesmas

```
CREATE VIEW ProximasViagens AS
SELECT TIMEDIFF(Hora_Partida,CURRENT_TIME()) AS 'Tempo até à viagem',
       CP.nome AS 'Cidade de Partida',
       Hora_Partida AS 'Hora de Partida',
       CC.nome AS 'Cidade de Chegada',
       Hora_Chegada AS 'Hora de Chegada'
FROM Viagem

INNER JOIN Cidade AS CP on CP.ID_Cidade = Cidade_Partida
INNER JOIN Cidade AS CC on CC.ID_Cidade = Cidade_Chegada
WHERE Hora_Partida > CURRENT_TIME()
ORDER BY TIMEDIFF(Hora_Partida,CURRENT_TIME());
```

Figura 12 - Criação da Vista (a) em SQL.

Tempo até à viagem	Cidade de Partida	Hora de Partida	Cidade de Chegada	Hora de Chegada
00:49:19	Braga	10:00:00	Porto	11:20:00
00:49:19	Braga	10:00:00	Madrid	14:00:00
02:19:19	Porto	11:30:00	Lisboa	15:00:00
05:19:19	Madrid	14:30:00	Braga	18:30:00
06:19:19	Lisboa	15:30:00	Porto	19:00:00
10:09:19	Porto	19:20:00	Braga	20:40:00

Figura 13 - Resultado da Vista (a).

(b) Vista que mostra o *top 5* das viagens mais baratas

```
CREATE VIEW Top5ViagensBaratas AS
SELECT CP.nome AS 'Cidade de Partida',
       Hora_Partida AS 'Hora de Partida',
       CC.nome AS 'Cidade de Chegada',
       Hora_Chegada AS 'Hora de Chegada',
       Reserva.Preço AS Preço
FROM Viagem
INNER JOIN Cidade AS CP on CP.ID_Cidade = Cidade_Partida
INNER JOIN Cidade AS CC on CC.ID_Cidade = Cidade_Chegada
INNER JOIN Reserva ON Reserva.Viagem = Viagem.ID_Viagem
ORDER BY Preço ASC
LIMIT 5;
```

Figura 14 - Criação da Vista (b) em SQL.

	Cidade de Partida	Hora de Partida	Cidade de Chegada	Hora de Chegada	Preço
▶	Braga	10:00:00	Porto	11:20:00	14.90
	Porto	19:20:00	Braga	20:40:00	14.90
	Porto	11:30:00	Lisboa	15:00:00	17.90
	Lisboa	15:30:00	Porto	19:00:00	17.90
	Porto	11:30:00	Lisboa	15:00:00	17.90

Figura 15 - Resultado da Vista (b).

- **Exemplos de Vistas de um Gestor**

(c) Vista que mostra a lista de clientes por ordem alfabética

```
CREATE VIEW Clientes AS
SELECT ID_Cliente, Nome FROM Cliente ORDER BY Nome;
```

Figura 16 - Criação da Vista (c) em SQL.

	ID_Cliente	Nome
▶	2	Ana
	4	Joana
	1	José
	5	Müller
	3	Patrick

Figura 17 - Resultado da Vista (c).

(d) Vista que mostra as últimas reservas realizadas pelos clientes, assim como alguma informação sobre a reserva

```
CREATE VIEW ProximasViagens AS
SELECT TIMEDIFF(Hora_Partida,CURRENT_TIME()) AS 'Tempo até à viagem',
       CP.nome AS 'Cidade de Partida',
       Hora_Partida AS 'Hora de Partida',
       CC.nome AS 'Cidade de Chegada',
       Hora_Chegada AS 'Hora de Chegada'
FROM Viagem

INNER JOIN Cidade AS CP on CP.ID_Cidade = Cidade_Partida
INNER JOIN Cidade AS CC on CC.ID_Cidade = Cidade_Chegada
WHERE Hora_Partida > CURRENT_TIME()
ORDER BY TIMEDIFF(Hora_Partida,CURRENT_TIME());
```

Figura 18 – Criação da vista (d) em SQL.

	ID_Reserva	Nome	Cidade de Partida	Cidade de Chegada
▶	10	Müller	Braga	Madrid
	9	Joana	Braga	Madrid
	8	Ana	Lisboa	Porto
	7	Müller	Madrid	Braga
	6	Patrick	Porto	Braga
	5	Ana	Porto	Lisboa
	4	José	Porto	Lisboa
	3	Patrick	Braga	Porto
	2	Ana	Braga	Porto
	1	José	Braga	Porto

Figura 19 - Resultado da Vista (d).

6.7. Definição das regras de acesso

No nosso projeto estão presentes três tipos de utilizadores: Administrador, Gestor e Cliente. Um Cliente é responsável por criar um registo seu na aplicação, inserir e atualizar reservas. Um Gestor está encarregue pela introdução e atualização de comboios e viagens. Sendo encarregue pela introdução de comboios, está também encarregue de introduzir os respetivos lugares. O Administrador tem autoridade para efetuar qualquer operação na base de dados.

Quanto às permissões para consultar dados, um Cliente tem acesso restrito às informações correspondentes às Reservas, Clientes e Comboios. Um Funcionário também se encontra limitado, tendo apenas acesso aos Comboios, Viagens, as Reservas, e parcialmente a informação dos Clientes. Em contrapartida, um Administrador tem acesso livre a qualquer informação da base de dados.

Desta forma, podemos definir as seguintes regras de acesso:

```
USE EuroTrain;

CREATE USER 'admin'@'localhost';
SET PASSWORD FOR 'admin'@'localhost' = PASSWORD('admin');

GRANT ALL ON EuroTrain.* TO 'admin'@'localhost';

CREATE USER 'gestor'@'localhost';
SET PASSWORD FOR 'gestor'@'localhost' = PASSWORD('gestor');

GRANT SELECT (ID_Cliente, Nome, Username, Email, Telemóvel, Cidade)
    ON EuroTrain.Cliente TO 'gestor'@'localhost';
GRANT SELECT ON EuroTrain.Reserva TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.Viagem TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.Comboio TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.Lugar TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.Cidade TO 'gestor'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.País TO 'gestor'@'localhost';


CREATE USER 'cliente'@'localhost';
SET PASSWORD FOR 'cliente'@'localhost' = PASSWORD('cliente');

GRANT INSERT ON EuroTrain.Cliente TO 'cliente'@'localhost';
GRANT SELECT, INSERT, UPDATE ON EuroTrain.Reserva TO 'cliente'@'localhost';
GRANT SELECT ON EuroTrain.Viagem TO 'cliente'@'localhost';
GRANT SELECT ON EuroTrain.Comboio TO 'cliente'@'localhost';
GRANT SELECT ON EuroTrain.Lugar TO 'cliente'@'localhost';
GRANT SELECT ON EuroTrain.Cidade TO 'cliente'@'localhost';
GRANT SELECT ON EuroTrain.País TO 'cliente'@'localhost';
```

Figura 20 - Criação dos diferentes utilizadores e regras de acesso.

7. Conclusões e Trabalho Futuro

A implementação da base de dados proposta neste relatório resulta de uma abordagem que nos pareceu adequada à resolução do problema, onde, na nossa opinião, os principais objetivos foram alcançados. Desta forma, ao analisarmos cuidadosamente a nossa implementação da base de dados, é-nos possível apontar tanto pontos fortes como obviamente alguns pontos fracos.

Relativamente aos pontos fracos ou caraterísticas que poderiam ser melhoradas podemos mencionar algumas como, por exemplo, a restrição do limite máximo de um lugar por cada reserva de viagem. Isto é, caso um cliente queira reservar mais do que um lugar, serão criadas na base de dados tantas reservas quantos esses mesmos lugares, o que por um lado pode sobrecarregar um pouco a base de dados sem necessidade de tal, caso não houvesse esta mesma restrição. Outra das caraterísticas que pode ser vista como um ponto fraco, é o facto de uma determinada viagem ter de ocorrer todos os dias da semana e ser sempre efetuada pelo mesmo comboio, não havendo assim viagens diferentes para dias diferentes, o que implica que aos fins-de-semana haja exatamente as mesmas viagens que nos restantes dias da semana, o que na realidade nunca acontece.

Por outro lado, é possível apontar aqueles que nós achamos que são os pontos fortes da nossa implementação da base de dados. Primeiramente, aquilo que mais salta á vista é a simplicidade com que esta se apresenta estruturada, o que a torna bastante intuitiva e de fácil alteração. Apesar da sua simplicidade, a consistência e a reduzida redundância de dados são igualmente um dos seus fortes. Estas foram conseguidas através da modelação da base de dados que achamos mais correta e assertiva.

A realização deste trabalho permitiu-nos acompanhar todos os estados de desenvolvimento de uma base de dados, desde a análise de requisitos até, por fim, à criação de um modelo físico, capaz de suportar as transações de informação pretendidas. O processo de levantamento de requisitos permitiu desenvolver a capacidade de abstração do grupo, de maneira a que através de uma contextualização do problema foram conseguidas diversas soluções para o problema. O desenvolvimento do modelo conceptual permitiu-nos chegar a uma solução viável para o problema, capaz de corresponder a todos os requisitos anteriormente apresentados, através da definição das entidades e correspondentes atributos e os relacionamentos presentes. De seguida, o desenvolvimento do modelo lógico aprofundou todos os conceitos do modelo anterior, aproximando-os de um modelo de programação orientada a SGBD. Esta parte do processo permitiu-nos aplicar regras de normalização de

tabelas e integridade referencial entre as mesmas. Por último, a conversão para modelo físico possibilitou-nos aprofundar os conhecimentos de SQL com a criação do esquema físico da base de dados, povoamento de tabelas e consulta de informação sob a forma de queries.

Apesar de a implementação da base de dados apresentada neste trabalho estar limitada ao âmbito da Reserva de viagens em comboios nacionais e internacionais, é possível imaginar novas opções, dentro do contexto em estudo, que seriam facilmente implementáveis e tornariam a base de dados ainda mais útil. Como já foi falado atrás, a possibilidade de adicionar à base de dados os funcionários da empresa, registando, desta forma, os horários de serviço que estes efetuam, era uma maneira eficaz de gerir toda a companhia ferroviária e não apenas as reservas dos seus clientes.

Por último, num ponto de vista mais abrangente, podemos concluir que as bases de dados são fundamentais para a gestão e tratamento da informação e constituem uma tecnologia sem a qual seria muito difícil o crescimento do mundo digital, principalmente nos dias de hoje com a quantidade de informação que é necessária gerar e armazenar a cada segundo.

Referências

1. Thomas Connolly, Carolyn Begg 2005. DATABASE SYSTEMS A Practical Approach to Design, Implementation, and Management, Harlow, Addison-Wesley.
2. Feliz Gouveia 2014. Fundamentos de Base de Dados, FCA.

Lista de Siglas e Acrónimos

BD Base de Dados

SQL Structured Query Language

SGBD Sistema de Gestão de Base de Dados

SBDR Sistema de Base de Dados Relacional

SBD Sistema de Base de Dados

Anexos

1. Script completo da criação do esquema física.
2. Script completo de povoamento.
3. Exemplos de queries em SQL.

I. Anexo 1 – Script completo da criação do esquema físico

```
-- Universidade do Minho
-- Mestrado Integrado em Engenharia Informática
-- Unidade Curricular de Bases de Dados
-- 2016/2017
--
-- Caso de Estudo: "EuroTrain"
-- Criação da base de dados utilizando a script gerada pelo MySQL Workbench.
--

SET @OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='TRADITIONAL,ALLOW_INVALID_DATES';

-- -----------------------------------------------------
-- Schema EuroTrain
-- -----------------------------------------------------
DROP SCHEMA IF EXISTS `EuroTrain` ;

-- -----------------------------------------------------
-- Schema EuroTrain
-- -----------------------------------------------------
CREATE SCHEMA IF NOT EXISTS `EuroTrain` DEFAULT CHARACTER SET utf8 ;
USE `EuroTrain` ;

-- -----------------------------------------------------
-- Table `EuroTrain`.`País`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`País` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`País` (
  `ID_País` INT NOT NULL,
  `Nome` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`ID_País`))
ENGINE = InnoDB;
```

```
-- -----------------------------------------------------
-- Table `EuroTrain`.`Cidade`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Cidade` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Cidade` (
  `ID_Cidade` INT NOT NULL,
  `País` INT NOT NULL,
  `Nome` VARCHAR(32) NOT NULL,
  PRIMARY KEY (`ID_Cidade`),
  INDEX `fk_Cidade_País1_idx` (`País` ASC),
  CONSTRAINT `fk_Cidade_País1`
    FOREIGN KEY (`País`)
    REFERENCES `EuroTrain`.`País` (`ID_País`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `EuroTrain`.`Cliente`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Cliente` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Cliente` (
  `ID_Cliente` INT NOT NULL,
  `Nome` VARCHAR(64) NOT NULL,
  `Username` VARCHAR(32) NOT NULL,
  `Password` VARCHAR(32) NOT NULL,
  `Cidade` INT NOT NULL,
  `Email` VARCHAR(64) NOT NULL,
  `Telemóvel` INT NOT NULL,
  PRIMARY KEY (`ID_Cliente`),
  INDEX `fk_Cliente_Cidade1_idx` (`Cidade` ASC),
  CONSTRAINT `fk_Cliente_Cidade1`
    FOREIGN KEY (`Cidade`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `EuroTrain`.`Comboio`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Comboio` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Comboio` (
  `ID_Comboio` INT NOT NULL,
  `Tipo` VARCHAR(64) NOT NULL,
```

```
  `Descrição` TEXT(512) NOT NULL,
  `Capacidade` INT NOT NULL,
  PRIMARY KEY (`ID_Comboio`))
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `EuroTrain`.`Viagem`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Viagem` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Viagem` (
  `ID_Viagem` INT NOT NULL,
  `Comboio` INT NOT NULL,
  `Cidade_Partida` INT NOT NULL,
  `Cidade_Chegada` INT NOT NULL,
  `Hora_Partida` TIME(0) NOT NULL,
  `Hora_Chegada` TIME(0) NOT NULL,
  `Duração` TIME(0) NOT NULL,
  PRIMARY KEY (`ID_Viagem`),
  INDEX `fk_Viagem_Comboio1_idx` (`Comboio` ASC),
  INDEX `fk_Viagem_Cidade1_idx` (`Cidade_Partida` ASC),
  INDEX `fk_Viagem_Cidade2_idx` (`Cidade_Chegada` ASC),
  CONSTRAINT `fk_Viagem_Comboio1`
    FOREIGN KEY (`Comboio`)
    REFERENCES `EuroTrain`.`Comboio` (`ID_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Cidade1`
    FOREIGN KEY (`Cidade_Partida`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Viagem_Cidade2`
    FOREIGN KEY (`Cidade_Chegada`)
    REFERENCES `EuroTrain`.`Cidade` (`ID_Cidade`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `EuroTrain`.`Lugar`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Lugar` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Lugar` (
  `Comboio` INT NOT NULL,
  `Nr` INT NOT NULL,
  `Classe` VARCHAR(16) NOT NULL,
  `Carruagem` INT NOT NULL,
```

```
  PRIMARY KEY (`Comboio`, `Nr`),
  CONSTRAINT `fk_Lugar_Comboio1`
    FOREIGN KEY (`Comboio`)
    REFERENCES `EuroTrain`.`Comboio` (`ID_Comboio`)
    ON DELETE CASCADE
    ON UPDATE CASCADE)
ENGINE = InnoDB;


-- -----------------------------------------------------
-- Table `EuroTrain`.`Reserva`
-- -----------------------------------------------------
DROP TABLE IF EXISTS `EuroTrain`.`Reserva` ;

CREATE TABLE IF NOT EXISTS `EuroTrain`.`Reserva` (
  `ID_Reserva` INT NOT NULL,
  `Data` DATE NOT NULL,
  `Comboio` INT NOT NULL,
  `Lugar` INT NOT NULL,
  `Código` VARCHAR(8) NOT NULL,
  `Preço` DECIMAL(5,2) NOT NULL,
  `Cliente` INT NOT NULL,
  `Viagem` INT NOT NULL,
  PRIMARY KEY (`ID_Reserva`),
  INDEX `fk_Reserva_Cliente_idx` (`Cliente` ASC),
  INDEX `fk_Reserva_Viagem1_idx` (`Viagem` ASC),
  INDEX `fk_Reserva_Lugar1_idx` (`Comboio` ASC, `Lugar` ASC),
  CONSTRAINT `fk_Reserva_Cliente`
    FOREIGN KEY (`Cliente`)
    REFERENCES `EuroTrain`.`Cliente` (`ID_Cliente`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Reserva_Viagem1`
    FOREIGN KEY (`Viagem`)
    REFERENCES `EuroTrain`.`Viagem` (`ID_Viagem`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE,
  CONSTRAINT `fk_Reserva_Lugar1`
    FOREIGN KEY (`Comboio` , `Lugar`)
    REFERENCES `EuroTrain`.`Lugar` (`Comboio` , `Nr`)
    ON DELETE NO ACTION
    ON UPDATE CASCADE)
ENGINE = InnoDB;


SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;
```

II. Anexo 2 – Script completo de povoamento

```
-- Universidade do Minho
-- Mestrado Integrado em Engenharia Informática
-- Unidade Curricular de Bases de Dados
-- 2016/2017
--
-- Caso de Estudo: "Euro-Train"
-- Povoamento parcial da base de dados.

-- Base de dados de trabalho
USE EuroTrain;

-- Povoamento da tabela "País"
INSERT INTO País (ID_País, Nome)
	VALUES
	(1, 'Portugal'),
	(2, 'Espanha'),
	(3, 'França'),
	(4, 'Alemanha'),
	(5, 'Luxemburgo');
-- SELECT * FROM País;

-- Povoamento da tabela "Cidade"
INSERT INTO Cidade (ID_Cidade, País , Nome)
	VALUES
	(1,1, 'Braga'),
	(2,1, 'Porto'),
	(3,1, 'Lisboa'),
	(4,2, 'Madrid'),
	(5,2, 'Barcelona'),
	(6,3, 'Paris'),
	(7,4, 'Munique'),
	(8,5, 'Luxemburgo');
-- SELECT * FROM Cidade

-- Povoamento da tabela "Cliente"
INSERT INTO Cliente (ID_Cliente, Nome, Username, Password, Email, Telemóvel, Cidade)
	VALUES
	(1, 'José', 'jose33', 'RPd234','ri@mail.com', 943213968, 1),
	(2, 'Ana', 'ana22', 'dfssw21', 'ana@mail.com', 973267943, 3),
```

```
    (3, 'Patrick', 'patrick11', 'wqer31', 'pat@mail.com', 953138527 , 8),
    (4, 'Joana', 'ju213', 'dfs123', 'ju@mail.com', 973351234, 1),
    (5, 'Müller', 'muller23', 'gfa456', 'muller@mail.com', 958643231 , 7);
-- SELECT * FROM Cliente

-- Povoamento da tabela "Comboio"
INSERT INTO Comboio (ID_Comboio, Tipo, Descrição, Capacidade)
    VALUES
        (1, 'Alfa', 'O Alfa permite-lhe viajar com todo o conforto e segurança, de norte a sul do país, e oferece-lhe o ambiente ideal para trabalhar ou descontrair durante a viagem. Para além de serviços específicos para as classes Conforto e Turística, pode desfrutar ainda de cafetaria, minibar, refeições, Wi-Fi gratuito e WC.',0),
        (2, 'Internacional', 'O Internacional permite-lhe viajar com todo o conforto e segurança, para um destino da europa, e oferece-lhe o ambiente ideal para trabalhar ou descontrair durante a viagem. Para além de serviços específicos para as classes Conforto e Turística, pode desfrutar ainda de cafetaria, minibar, refeições, Wi-Fi gratuito e WC. As cadeiras estão adaptadas a longas viagens.',0);
-- SELECT * FROM Comboio

DELIMITER $$
CREATE TRIGGER ActCapacidadeInc
    AFTER INSERT ON Lugar
    FOR EACH ROW
BEGIN
    UPDATE Comboio
        SET Capacidade = Capacidade + 1
        WHERE ID_Comboio = NEW.Comboio;
END $$

DELIMITER $$
CREATE TRIGGER ActCapacidadeDec
    AFTER DELETE ON Lugar
    FOR EACH ROW
BEGIN
    UPDATE Comboio
        SET Capacidade = Capacidade - 1
        WHERE ID_Comboio = OLD.Comboio;
END $$

-- Povoamento da tabela "Lugar"
INSERT INTO Lugar (Comboio, Nr, Classe, Carruagem)
    VALUES
    (1,1, 'Turística',1),
    (1,2, 'Turística',1),
    (1,3, 'Turística',1),
    (1,4, 'Turística',1),
    (1,5, 'Turística',1),
    (1,6, 'Turística',2),
    (1,7, 'Turística',2),
    (1,8, 'Turística',2),
    (1,9, 'Turística',2),
    (1,10, 'Conforto',3),
```

```
        (1,11, 'Conforto',3),
        (1,12, 'Conforto',3),
        (2,1, 'Turística',1),
        (2,2, 'Turística',1),
        (2,3, 'Turística',1),
        (2,4, 'Turística',1),
        (2,5, 'Turística',1),
        (2,6, 'Turística',2),
        (2,7, 'Turística',2),
        (2,8, 'Turística',2),
        (2,9, 'Turística',2),
        (2,10, 'Conforto',3),
        (2,11, 'Conforto',3),
        (2,12, 'Conforto',3);
-- SELECT * FROM Lugar

-- Povoamento da tabela "Viagem"
INSERT INTO Viagem (ID_Viagem, Comboio, Cidade_Partida, Cidade_Chegada, Hora_Partida, Hora_Chegada, Duração)
        VALUES
        (1,1, 1, 2 , '10:00:00', '11:20:00','1:20:00'),
        (2,1, 2, 3 , '11:30:00', '15:00:00','3:30:00'),
        (3,1, 3, 2 , '15:30:00', '19:00:00','3:30:00'),
        (4,1, 2, 1 , '19:20:00', '20:40:00','1:20:00'),
        (5,2, 1, 4 , '10:00:00', '14:00:00','4:00:00'),
        (6,2, 4, 1 , '14:30:00', '18:30:00','4:00:00');
-- SELECT * FROM Viagem

-- Povoamento da tabela "Reserva"
INSERT INTO Reserva (ID_Reserva, Data, Comboio, Lugar, Código, Preço, Cliente, Viagem)
        VALUES
        (1,'2016-12-26', 1, 10,'74ytrefg', 18.90, 1,1),
        (2,'2016-12-26', 1, 11, 'hw567i4w', 18.90, 2,1),
        (3,'2016-12-26', 1, 3, '47k6twyj', 14.90, 3,1),
        (4,'2016-12-26', 1, 7, '74ytfeww', 17.90, 1,2),
        (5,'2016-12-26', 1, 8, '423eei4w', 17.90, 2,2),
        (6,'2016-12-27', 1, 7, 'grvr4s21', 14.90, 3,4),
        (7,'2016-12-27', 2, 7, 'fvfqgwef', 33.90, 5,6),
        (8,'2016-12-28', 1, 3, 'fwfqqwrj', 17.90, 2,3),
        (9,'2016-12-30', 2, 10, 'it8745ur', 39.90, 4,5),
        (10,'2016-12-30', 2, 12, 'vbfq3341', 39.90, 5,5);

-- SELECT * FROM Reserva
```

III. Anexo 3 – Exemplos de queries em SQL

Neste anexo são apresentados exemplos de resoluções de algumas queries em SQL, assim como os seus resultados.

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

```
USE EuroTrain;

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

Figura 21 - Resolução SQL da query 1.

	Nome	Preço
▶	Müller	39.90
	Joana	39.90
	Müller	33.90
	José	18.90
	Ana	18.90
	Ana	17.90
	José	17.90
	Ana	17.90
	Patrick	14.90
	Patrick	14.90

Figura 22 - Resultados da query 1.

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

```
USE EuroTrain;

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

Figura 23 - Resolução SQL da query 2.

	Comboio	Partida	Chegada	Duração
▶	1	Braga	Porto	01:20:00
	1	Porto	Lisboa	03:30:00
	1	Lisboa	Porto	03:30:00
	1	Porto	Braga	01:20:00
	2	Braga	Madrid	04:00:00
	2	Madrid	Braga	04:00:00

Figura 24 - Resultados da query 2.

3. **Query que apresenta a lista de lugares disponíveis de uma dada viagem.**

```
USE EuroTrain;

DELIMITER $$
CREATE PROCEDURE procedureLugaresLivres (IN Data DATE,IN Hora TIME(0),IN cidPartida VARCHAR(32), IN cidChegada VARCHAR(32))
BEGIN
    SELECT L.Nr AS Lugar, L.Classe AS Classe, L.Carruagem AS Carruagem  FROM Viagem
        INNER JOIN Cidade AS CP
        ON CP.ID_Cidade = Viagem.Cidade_Partida
            INNER JOIN Cidade AS CC
            ON CC.ID_Cidade = Viagem.Cidade_Chegada
                INNER JOIN Lugar AS L
                ON L.Comboio = Viagem.Comboio
                WHERE Viagem.Hora_Partida = Hora AND CP.Nome = cidPartida AND CC.Nome = cidChegada
                    AND L.Nr NOT IN (SELECT Lugar.Nr FROM Lugar
                                INNER JOIN Reserva ON Lugar.Nr = Reserva.Lugar
                                INNER JOIN Viagem ON Reserva.Viagem = Viagem.ID_Viagem
                                INNER JOIN Cidade AS CPP ON CPP.ID_Cidade = Viagem.Cidade_Partida
                                INNER JOIN Cidade AS CCC ON CCC.ID_Cidade = Viagem.Cidade_Chegada
                                WHERE Reserva.Data = Data AND Viagem.Hora_Partida = Hora AND Reserva.Comboio = Lugar.Comboio
                                        AND CPP.Nome = cidPartida AND CCC.Nome = cidChegada );
END $$

SET @data = '2016-12-26';
SET @hora = '10:00:00';
SET @CidPartida = 'Braga';
SET @CidChegada = 'Porto';
CALL procedureLugaresLivres(@data,@hora,@CidPartida,@CidChegada);

DROP PROCEDURE procedureLugaresLivres;
```

Figura 25 - Resolução SQL da query 3.

	Lugar	Classe	Carruagem
▶	1	Turística	1
	2	Turística	1
	4	Turística	1
	5	Turística	1
	6	Turística	2
	7	Turística	2
	8	Turística	2
	9	Turística	2
	12	Conforto	3

Figura 26 - Resultados da query 3.

4. **Query que apresenta a lista das reservas de uma viagem dada, ordenada pelo nome do Cliente.**

```
USE EuroTrain;

DELIMITER $$
CREATE PROCEDURE procedureReservasViagem (IN Data DATE,IN Hora TIME(0),IN CidadePartida VARCHAR(32), IN CidadeChegada VARCHAR(32))
BEGIN
    SELECT Cliente.Nome, Reserva.Lugar, Reserva.Preço FROM Viagem
    INNER JOIN Cidade AS CP
    ON CP.ID_Cidade = Viagem.Cidade_Partida
        INNER JOIN Cidade AS CC
        ON CC.ID_Cidade = Viagem.Cidade_Chegada
            INNER JOIN Reserva
            ON Reserva.Viagem = Viagem.ID_Viagem
                INNER JOIN Cliente
                ON CLiente.ID_Cliente = Reserva.Cliente
    WHERE CP.Nome = CidadePartida AND CC.Nome = CidadeChegada AND Reserva.Data = Data AND Viagem.Hora_Partida = Hora
    ORDER BY Cliente.Nome ASC;
END $$

SET @data = '2016-12-26';
SET @hora = '11:30:00';
SET @cidadePartida = 'Porto';
SET @cidadeChegada = 'Lisboa';
CALL procedureReservasViagem(@data, @hora, @cidadePartida, @cidadeChegada);

DROP PROCEDURE procedureReservasViagem;
```

Figura 27 - Resolução SQL da query 4.

	Nome	Lugar	Preço
▶	Ana	8	17.90
	José	7	17.90

Figura 28 - Resultados da query 4.

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

```
USE EuroTrain;

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

Figura 29 - Resolução SQL da query 5.

	Username	Número de Reservas
▶	ana22	3
	muller23	2
	patrick11	2
	jose33	2
	ju213	1

Figura 30 - Resultados da query 5.