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

# Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

## Reserva de Viagens: Uma base de dados

**João Reis, João Gomes, Tiago Fraga**

Novembro 2016

BD

| | |
|---|---|
| Data de Recepção | |
| Responsável | |
| Avaliação | |
| Observações | |

# Reserva de Viagens: Uma base de dados

**João Reis, João Gomes, Tiago Fraga**

Novembro 2016

# Resumo

Neste trabalho proceder-se-á a uma base de dados baseada de uma dada temática.
Seguindo métodos bastante rigorosos, constrói-se faseadamente vários modelos até chegar ao produto final, que consiste numa base de dados funcional para um sistema de reserva de bilhetes de comboios.

**Área de Aplicação:** Desenho e arquitetura de Sistemas de Bases de Dados
**Palavras-Chave:** Bases de Dados Relacionais, Modelos Conceptuais, Modelos Lógicos, Modelos Físicos, MySQL

# Índice

# Índice de Figuras

# Índice de Tabelas

# 1. Introdução

## 1.1. Contextualização

Fundada de 1997, a empresa ferroviária **Train**sporte nasceu com o intuito de transportar estudantes de Vila Real às cidades de Braga, Porto e Lisboa devido à falta de meios e condições de transporte para estas grandes cidades de principal destino universitário. Devido à sua grande adesão por parte da comunidade estudantil e não só, o crescimento do negócio foi exponencial e rapidamente chegou a diferentes distritos de Portugal.

No entanto, apesar do imenso sucesso causado, os métodos de operação continuam os mesmos que primordialmente foram estabelecidos:

- Não existe registo de clientes;
- A reserva de bilhetes teria de ser feita obrigatoriamente nas bilheteiras;
- Não existe nenhuma base de dados de horários: são estabelecidos mensalmente na sede e enviados às diferentes estações, comunicando por meio telefónico entre estações os atrasos existentes.

A empresa não acompanhou a evolução tecnológica e pleno 2016 estava a afundar-se em papelada e contantes atrasos entre comboios. O crescimento de clientes estagnou, a comunicação entre estações era pobre e o CEO da empresa, Meireles, facilmente apercebeu-se que estavam a ser vitimas do seu próprio sucesso e que teriam de gastar parte do orçamento anual de modo a evoluir as infraestruturas existentes e lançar um serviço *online* de modo a festejar os 20 anos da **Train**sporte. O serviço *online* consistia não só numa *front-end* para clientes reservar bilhetes mas também para uma base de dados de horários, comboios e clientes.

O CEO contratou a famosa empresa portuguesa *TiJo^2, Lda.* de modo a desenvolver uma base de dados eficiente e organizada.

## 1.2. Apresentação do Caso de Estudo

De modo a utilizar o serviço Trainsporte, os utilizadores necessitam de estar registados no *website* criado especificamente para esse propósito, tendo portanto associar à sua conta dados pessoais relevantes para o funcionamento do site como um *username* único, palavra-chave, nome, NIF, e-mail, e morada.

Assim que autenticado na página *online* da empresa, poderá consultar as viagens disponíveis de acordo com as suas preferências. Uma vez introduzidos os locais de origem e destino desejados, bem como uma hora de partida e chegada conveniente, será disponibilizado ao cliente a lista lugares distinguidos pelo seu número, carruagem e classe (económica, normal ou especial) de um determinado comboio, que, para além duma capacidade máxima, tem também, para maior conveniência do utilizador, um nome característico.
Uma vez escolhida a viagem que liga expressamente a cidade em que se encontra até ao destino desejado, ser-lhe-á possível efetuar uma reserva onde lhe é escolher a quantidade de lugares que quer reservar, desde que não ultrapasse a capacidade máxima do comboio. De um modo geral, um cliente poderá reservar vários bilhetes correspondentes aos respetivos lugares de um comboio, que faz a viagem entre duas estações, obtendo no final, um preço final das suas reservas.

## 1.3. Motivação e Objetivos

Sendo o trabalho uma fundamental parte da avaliação da unidade curricular de Base de Dados (para desenvolver os conhecimentos dos alunos), elaboramos este projeto tendo conta não só essa obrigação, mas também com o prazer de adquirir novos conhecimentos duma parte importante da engenharia informática. Tendo então partido para a composição do mesmo, o nosso principal objetivo foi principalmente adquirir e aprofundar conhecimentos da realização dos diversos modelos, procedimentos e também a linguagem SQL.

## 1.4. Estrutura do Relatório

No presente relatório serão apresentados dois capítulos - *Validação do Projeto* e *Conclusões e Trabalho Futuro*.
No capítulo *Validação do Projeto* e *Conclusões* é conduzida a produção da base de dados através do caso de estudo apresentado, passado pelos vários procedimentos – modelo conceptual, lógico e físico – explicando de maneira bastante eximia, as várias escolhas e métodos escolhidos ao longo da elaboração.
Finalmente, na *Conclusões e Trabalho Futuro* fazendo uma observação ao que produzido, critica-se de maneira construtiva de modo a que melhor podemos melhorar, ao que estará bem feito, bem como dificuldades e adversidades encontradas ao longo da criação.

# 2. Validação do Projeto

De modo a proceder à validação do projeto, recorreu-se à metodologia presente nos capítulos 15, 16 e 17 do livro - *Connolly, T., Begg, C., Database Systems*, com as devidas adaptações à nossa situação.

## 2.1. Validação do modelo conceptual

Neste passo da validação do projeto, usou-se a ferramenta TerraER de modo a representar o modelo conceptual final, obtendo o seguinte resultado:

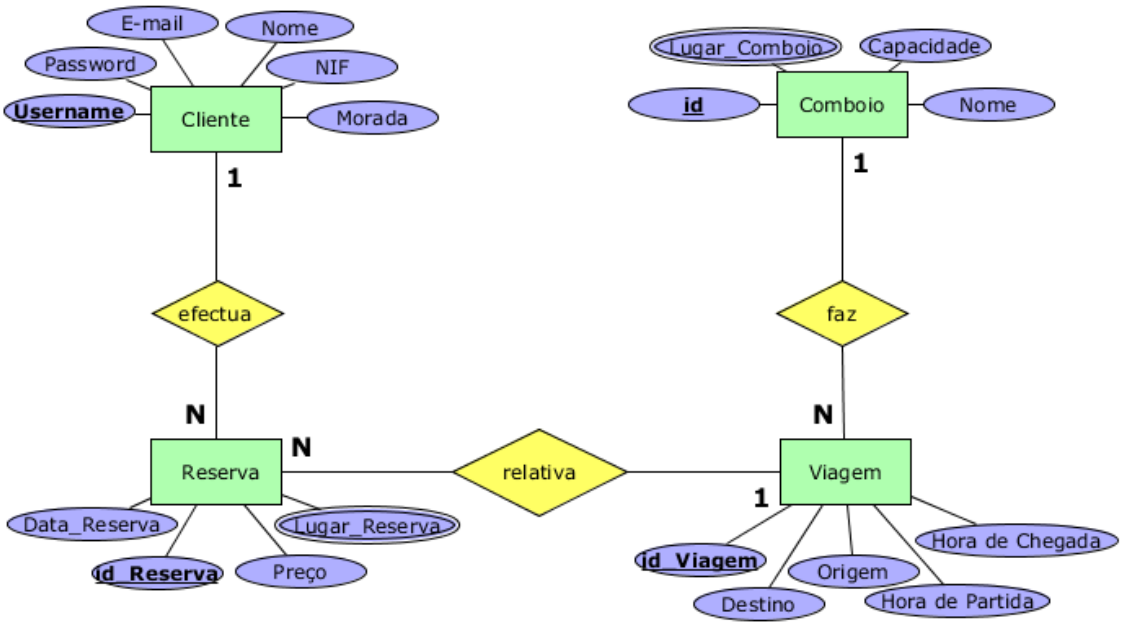



Figura 1 - Modelo Conceptual

### 2.1.1 Identificação das Entidades

Como entidades do modelo construído, enumeram-se as seguintes:

- Cliente
- Reserva
- Viagem

- Comboio

| Entidade | Descrição | Aliases | Ocorrência |
|---|---|---|---|
| **Cliente** | Termo usado para descrever utilizadores do serviço *online* | Utilizador | |
| **Reserva** | Termo usado para descrever o ato de reserva de um bilhete | | |
| **Comboio** | Termo que representa o meio onde serão transportados os Clientes | | |
| **Viagem** | Termo que representa a deslocação de um comboio de uma origem para um destino | | |

Tabela 1 - Entidades do Modelo Relacional

## 2.1.2 Identificação das Relacionamentos

No nosso modelo Conceptual temos três tipos de relacionamentos, sendo todos eles relacionamentos binários, isto é, um relacionamento entre dois tipos de entidades.

- **"Cliente" → "Reserva"**


O relacionamento entre "Cliente "e "Reserva", em termos de multiplicidade trata-se de uma relação de 1 para 'N', isto é, um cliente pode fazer 'N' reservas.

- **"Reserva" -> "Viagem"**


A multiplicidade neste relacionamento é de 'N' para 1, ou seja, existem 'N' reservas numa Viagem

- **"Comboio" -> "Reserva"**


Tal como no relacionamento anterior, a multiplicidade é de 1 para ‘N’, querendo dizer que um comboio realiza ‘N’ viagens.

De um modo sucinto,

| Entidade | Multiplicidade | Relacionamento | Multiplicidade | Entidade |
|---|---|---|---|---|
| Cliente | 1 | Efectua | N | Reserva |
| Comboio | 1 | Faz | N | Viagem |
| Reserva | N | Relativo a | 1 | Viagem |

Tabela 2 - Relacionamentos do Modelo Relacional

### 2.1.3 Identificar e associar atributos com entidades e tipos de relacionamentos

Neste ponto aborda-se atributos das várias entidades que estão presentes no modelo conceptual, visto que não existe nenhum atributo afeto aos relacionamentos entre as entidades.

**Na entidade ‘Cliente’ existem seis atributos.**
O atributo que representa a chave primária desta entidade é ‘Username’, uma vez que um cliente efetua um registo terá de ter o seu username distinto dos restantes já introduzidos no sistema. Além deste, temos alguns atributos simples, como a ‘Password’, o, o ‘E-mail’, o ‘Nome’, ‘NIF’ e ‘Morada’. Este ultimo é opcional uma vez que só necessário introduzir a morada se o cliente quiser receber a sua reserva por correio.

**Na segunda entidade denominada ‘Reserva’, existem quatro atributos.**
A chave primária desta entidade é o atributo ‘id_Reserva’ visto que todas as reservas vão ter uma identificação diferente, sendo possível, assim, distingui-las. Tem também como atributos simples a ‘Data’ em que é emitida e o seu ‘Preço’ total. Utilizamos por fim, um atributo multivariado, de seu nome ‘Lugar_Reserva’, isto porque, cada reserva vai poderá ter mais que um lugar reservado.

**Na entidade ‘Viagem’, existem cinco atributos.**
A chave primária desta entidade trata-se da ‘id_Viagem’. Os restantes atributos desta entidade são a ‘Hora_Chegada’, a ‘Hora_Partida’, a ‘Origem’, e o ‘Destino’.

**Finalmente, na entidade 'Comboio' existem quatro entidades.**

Os atributos 'id_Comboio', 'Capacidade' e 'Nome' são simples, sendo o primeiro a chave primária da entidade. Por fim, o atributo 'Lugar_Comboio' é multi-valorada porque num comboio existem vários lugares.

### 2.1.4 Identificação dos tipos dos atributos

Na tabela abaixo representada está exposta as características dos atributos das diferentes Entidades existentes.

| Entidade | Atributo | Característica |
|---|---|---|
| **Cliente** | username | Até 12 caracteres |
| | Password | De a 6 a 15 caracteres |
| | Nome | Até 50 caracteres |
| | E-mail | Até 50 caracteres |
| | Morada | Até 50 caracteres |
| | NIF | Exatamente 9 dígitos |
| **Reserva** | id_Reserva | Exatamente 9 caracteres |
| | Data_Reserva | Contém data e hora |
| | Preço | Contém um double positivo |
| | Lugar | Tuplo com 3 inteiros (Número, Carruagem, Classe) |
| **Viagem** | id_Viagem | Exatamente 9 caracteres |
| | Hora de Partida | Contém data e hora |
| | Hora de Chegada | Contém data e hora |
| | Destino | Até 50 caracteres |
| | Origem | Até 50 caracteres |
| **Comboio** | id_Comboio | Exatamente 9 caracteres |
| | Nome | Até 50 caracteres |
| | Capacidade | Inteiro |
| | Lugar | Tuplo com 3 inteiros (Número, Carruagem, Classe) |

Tabela 3 - Características dos atributos

### 2.1.5 Determinar chaves candidatas, primárias e alternativas

As chaves primárias das diferentes entidades presentes no modelo conceptual são as seguintes:

- Cliente → **Username**

  Usou-se o username como chave primária uma vez que todos utilizadores têm um que é distinto a todos os outros;

- Reserva → **id_Reserva**
  Uma vez que não existia nenhum atributo que possivelmente distinguisse uma reserva das restantes, usou-se um id para distinguir para essa mesma função.

- Viagem → **id_Reserva**
  Uma vez que não existia nenhum atributo que possivelmente distinguisse uma viagem das restantes, usou-se um id para distinguir para essa mesma função.

- Comboio → **id_Comboio**
  Apesar de existir um atributo que possibilitasse a distinção de um comboio dos restantes, o nome, este serve apenas como uma maneira informal de os apelidar para maior conveniência dos utilizadores, podendo até ter 50 caracteres, o que o torna pouco adequado como chave primária. Usou-se, portanto, um id.

### 2.1.6 Verificar redundância do modelo

Neste passo, examina-se o modelo conceptual de modo conceptual de modo a identificar a presença de qualquer tipo de redundância, e elimina-la caso exista. Para tal, seguimos três passos:

1. **Reexaminar relações um para um (1:1)**
   No nosso modelo não se encontra qualquer tipo de relação um para um, de modo a que não existe nada a alterar.

2. **Remover relações redundantes**
   Neste passo remover-se-ia ciclos, isto é, se uma informação é possível obter-se a partir de outras relações. Mais uma vez, no modelo conceptual elaborado não existe qualquer situação em que isto acontece, e portanto, não existe nada a alterar.

3. **Considerar dimensão do tempo**
   Finalmente, no último passo da verificação mais uma vez se o modelo conceptual elaborado continua inalterável após mais uma revisão. A condição temporal não tem qualquer efeito no que foi construído.

### 2.1.7 Validar modelo perante transações

1. **Apresentar a lista dos lugares reservados e das datas das reservas efetuadas por um determinado cliente.**
   Acedemos as reservas de um cliente e apresentamos uma lista com todos os lugares reservados por esse cliente bem como as datas dessas reservas.

2. **Identificar o preço de uma reserva efetuada por um determinado cliente.**
   Neste caso selecionamos a reserva efetuada pelo cliente e apresentamos o valor do preço dessa reserva.

3. **Apresentar a lista dos destinos para uma origem.**
   Dada uma origem gerar uma lista com todos os destinos possíveis.

4. **Identificar o número total de lugares livres num comboio para uma determinada viagem.**
   Para saber o número total de lugares livres num comboio para uma viagem, temos de calcular a diferença entre o número de lugares no comboio e o número de lugares reservados para essa viagem.

5. **Fazer uma reserva para uma viagem, para uma determinada data para um lugar específico de uma classe do comboio.**
   Neste caso pesquisámos pelos lugares livres do comboio, em vez de se calcular os lugares como numa das alíneas anteriores, apresentamos todos os lugares livres de modo a selecionarmos o lugar ou os lugares que cumprem os requisitos pretendidos.

6. **Insere um cliente na base de dados.**
   Para inserir um cliente na base de dados, temos de inserir todos os dados que estão na entidade cliente.

## 2.2. Validação do modelo lógico

Na segunda parte da validação, seguindo os vários passos da metodologia presente no livro, chegou-se ao seguinte resultado:

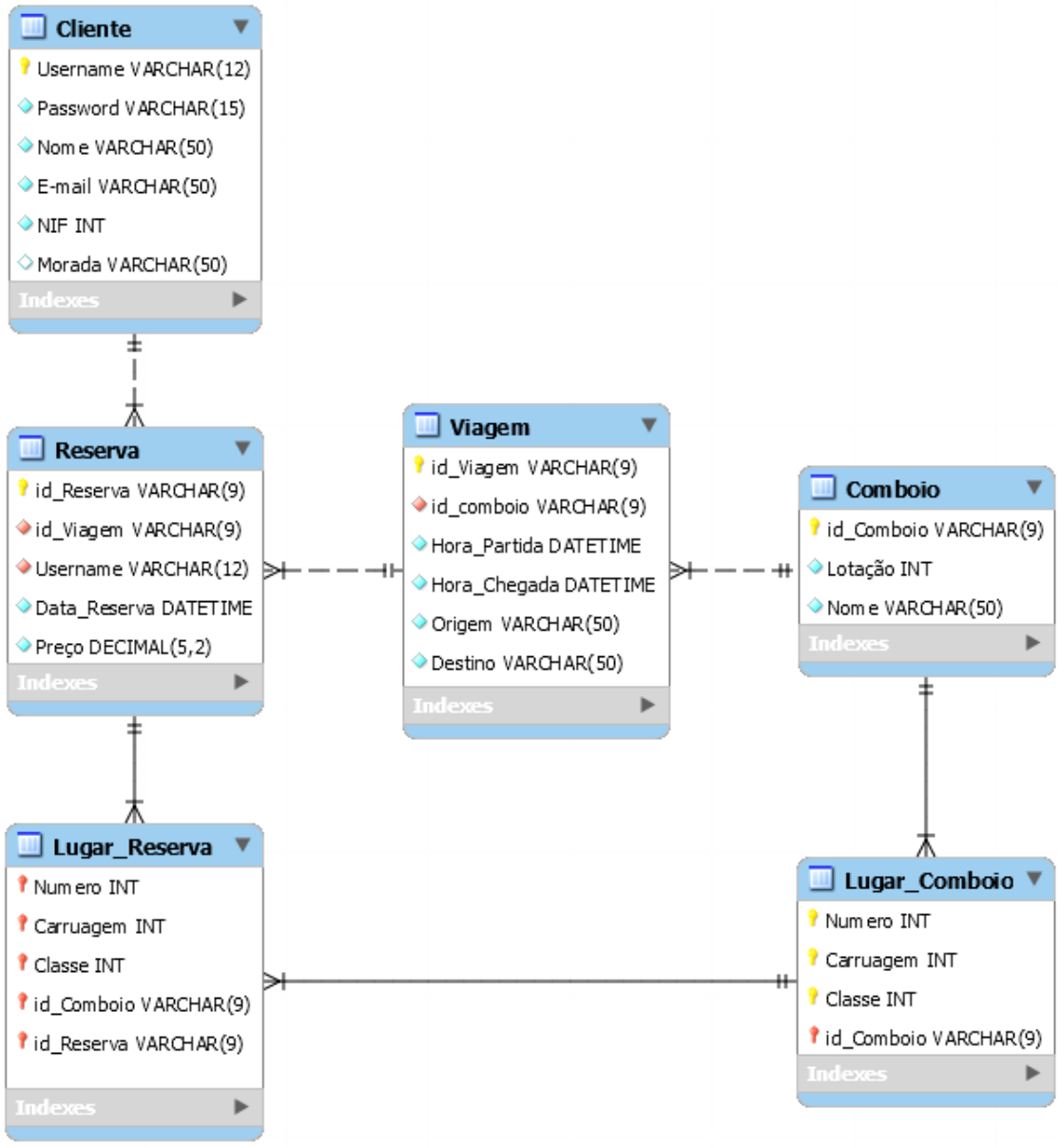



Figura 2 - Modelo Lógico

## 2.2.1 Derivar relações do modelo conceptual

Esta etapa tem como objetivo de criar relações para o modelo lógico representar entidades, relacionamentos e atributos que foram identificados. Descrever-se-á a composições de cada relação usando DBDL (*Database Definition Language*) para modelos relacionais.

1. **Entidades fortes**

   Sendo que uma entidade forte trata-se duma entidade em que a sua existência não depende de outra, todas as quatro entidades presentes no modelo conceptual – Cliente, Reserva, Bilhete e Comboio – estão de acordo com esta norma.

   Composição das entidades:

**Cliente** (Username, Password, Nome, E-mail, NIF, Morada)
**Chave Primária:** Username
**Chaves Alternativas:** E-mail, NCC

**Reserva** (id_Reserva, id_Viagem, Username, Data_Reserva, Preço, Lugar_Reserva)
**Chave Primária:** id_Reserva
**Chaves Estrangeiras:** Cliente, Viagem

**Viagem** (id_Viagem, Destino, Origem, Hora_de_Partida, Hora_de_Chegada, id_Comboio)
**Chave Primária:** id_Viagem
**Chaves Estrangeiras:** id_Comboio

**Comboio** (id_Comboio, Capacidade, Nome, Lugar_Comboio)
**Chave Primária:** id_Comboio
**Chave Alternativa:** Nome

2. **Entidades Fracas**

Tratando-se do oposto do ponto anterior, as entidades fracas referem—se a entidades que a sua existência depende de outras entidades. Assim sendo, e uma vez que todas as entidades são fortes, não existe qualquer fraca.

3. **Relações 1 para N**

Nesta etapa da derivação representa-se as relações 1 para N, isto é, 1 para vários que estão presentes no modelo conceptual.


**Cliente** usado em **Reserva:** relação **efectua**

**Cliente** (Username, Password, E-mail , Nome, NIF, Morada)
**Chave Primária:** Username
**Chaves Alternativas:** E-mail, NCC, NIF

**Reserva** (id_Reserva, Data_Reserva, Preço, Luga_Reserva, id_Cliente, Viagem)
**Chave Primária:** id_Reserva
**Chaves Estrangeiras:** Cliente, Viagem


**Viagem** usado em **Reserva:** relação **relativa**

**Reserva** (id_Reserva, Data_Reserva, Preço, Luga_Reserva, Cliente, Viagem)

**Chave Primária:** id_Reserva

**Chaves Estrangeiras:** Cliente, Viagem

**Viagem** (id_Viagem, Destino, Origem, Hora_de_Partida, Hora_de_Chegada, id_Comboio)

**Chave Primária:** id_Viagem

**Chaves Estrangeiras:** id_Comboio


**id_Comboio** usado em **Viagem:** relação **faz**

**Comboio** (id_Comboio, Capacidade, Nome, Lugar)

**Chave Primária:** id_Comboio

**Chave Alternativa:** Nome

**Viagem** (id_Viagem, Destino, Origem, Hora_de_Partida, Hora_de_Chegada, id_Comboio)

**Chave Primária:** id_Viagem

**Chaves Estrangeiras:** id_Comboio

4. **Relações 1 para 1**

   No modelo atual não existe qualquer relação 1 para 1.

5. **Relações recursivas 1 para 1**

   Não só não existem relações recursivas 1 para 1, como também não está presente qualquer recursividade.

6. **Subclasses e Superclasses**

No modelo apresentado não existe qualquer superclasse e/ou subclasse, uma vez que não está presente qualquer subgrupo ou até mesmo relação 1 para 1.

7. **Relações N para N**

   No modelo atual não está representado qualquer relação N para N.

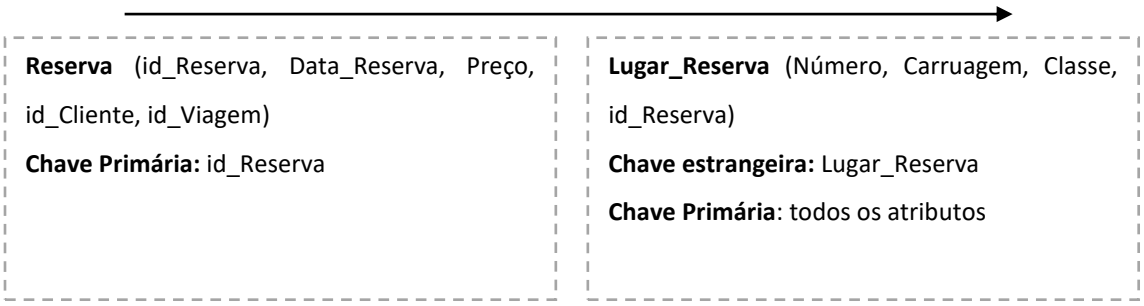
8. **Relações complexas**

   Todas as relações existentes no modelo atual tratam-se de relações e binárias, e portanto, não existe qualquer complexa.

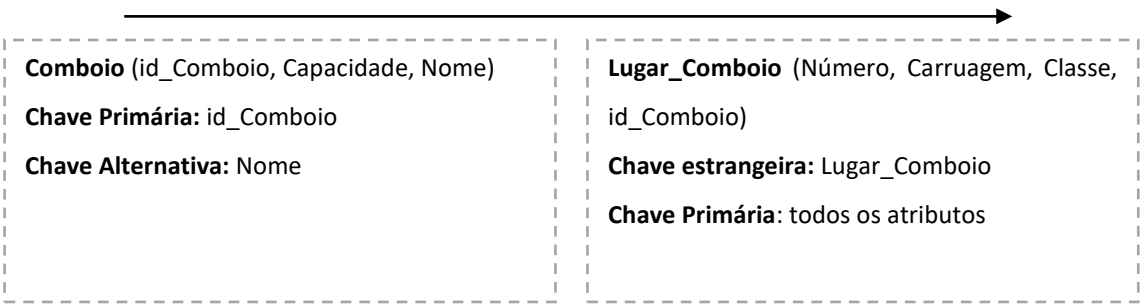
9. **Atributos Multivalorados**

   Existem dois atributos muti valorados, sendo eles Lugar_Reserva e Lugar_Comboio que fazem parte das entidades Reserva e Comboio, respetivamente.

**id_Reserva** vem de **Reserva**




**id_Comboio** vem de **Comboio**




## 2.2.2 Normalização

1. **1ª Forma Normal**

   De modo a uma tabela estar de acordo com a 1ª forma normal, é necessário que uma chave não contenha várias ocorrências do mesmo atributo.

   Ora, de acordo com uma análise ao modelo atualmente contruído é possível constatar que em todas as tabelas não existe a ocorrência de grupos repetidos, estando assim, de acordo com a primeira forma normal.

2. **2ª Forma Normal**

   Como anteriormente foi provado, o modelo encontra-se de acordo com 1ª forma normal, que é o primeiro requisito necessário para estar de cordo com a segunda.

   Outra condição necessária a segunda forma normal é que todas as chaves não primárias sejam dependentes da chave primária – algo que, através da observação das tabelas existentes, é também possível de confirmar. Existem, no entanto, chaves que poderiam por si mesmas identificar uma tabela, mas que no entanto foram anteriormente consideradas chaves candidatas, o que as torna também dependentes.

3. **3º Forma Normal**

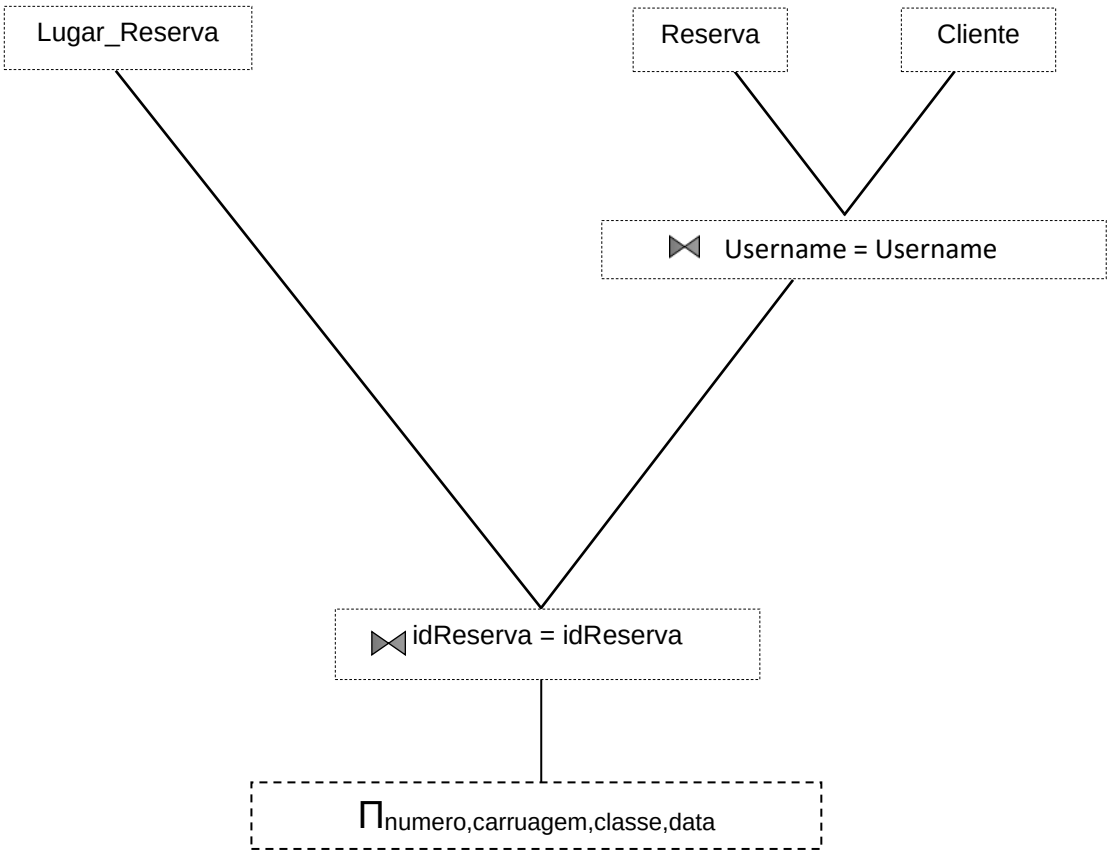
   Finalmente, para uma tabela encontrar-se de acordo com a terceira normal, é necessário que cumpra também a segunda.

   De seguida verifica-se se alguma das colunas existentes nessa tabela é dependente de outra(s), algo que não acontece em nenhuma das tabelas existentes no modelo lógico construído.

### 2.2.3 Validar relações perante transações

De modo a validar as relações presentes no modelo lógico, usou-se a álgebra relacionar para as seguintes transações:

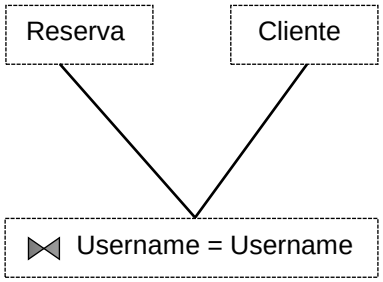
1. **Apresentar a lista dos lugares reservados e das datas das reservas efetuadas por um determinado cliente.**




$\pi_{numero,\ carruagem,\ classe,data}$ (Lugar_Reserva ⋈ (Reserva ⋈ Cliente)).

---

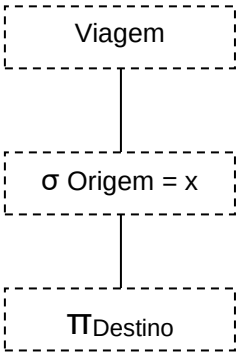
**2. Identificar o preço de uma reserva efetuada por um determinado cliente.**




$\pi_{preço}$ (Reserva ⋈ Cliente)

---

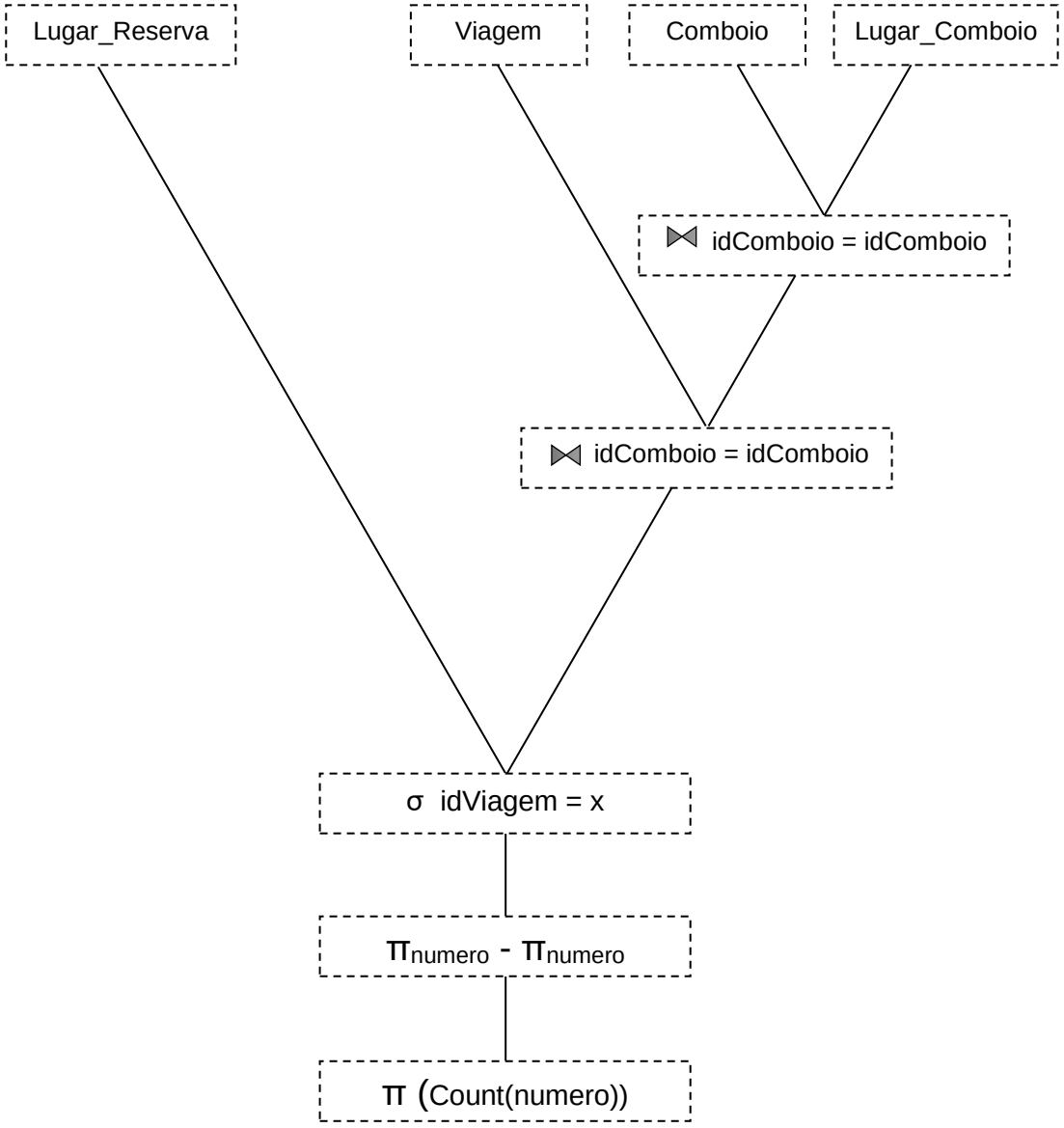
**3. Apresentar a lista dos destinos para uma origem.**




**$\Pi_{Destino}(\sigma$ origem = x (Viagem))**

---

4. **Identificar o número total de lugares livres num comboio para uma determinada viagem.**



$\Pi$(Count(Numero))(($Pl_{Lugar\_Reserva.Numero}$ – $Pl_{Lugar\_Comboio.Numero}$ ) ($\sigma$ idViagem=x (Viagem ⋈ (Comboio ⋈ Lugar_Comboio))

---

5. **Fazer uma reserva para uma viagem, para uma determinada data para um lugar específico de uma classe do comboio.**

6. **Insere um cliente na base de dados.**

Nestes dois pontos finais não se aplica a álgebra relacional, visto envolver inserções de elementos. A álgebra relacional é aplicada a operações de consulta, isto é, que não alterem o estado da base de dados.

## 2.2.4 Restrições de Integridade

- **Informação obrigatória**
  Em quase todas as entradas das tabelas presentes no modelo lógico são não-nulas, assim como nele indicado. No entanto, na tabela Cliente existe uma coluna que não necessita obrigatoriamente de preenchimento, sento ela a Morada.

- **Restrições do domínio dos atributos**
  No ponto 2.1.4 construiu-se uma tabela de modo a estabelecer tamanhos ou limites para um determinado atributo, bem como o seu tipo. O modelo lógico respeita todos estes tamanhos, bem como o seu tipo.

| Entidade | Atributo | Domínio |
|---|---|---|
| **Cliente** | username | VARCHAR(12) |
| | Password | VARCHAR(15) |
| | Nome | VARCHAR(50) |
| | E-mail | VARCHAR(50) |
| | Morada | VARCHAR(50) |
| | NIF | INT |
| **Reserva** | id | VARCHAR(9) |
| | Data_Reserva | DATETIME |
| | Preço | DECIMAL(5,2) |

| | | |
|---|---|---|
| **Viagem** | id | VARCHAR(9) |
| | Hora de Partida | DATETIME |
| | Hora de Chegada | DATETIME |
| | Destino | VARCHAR(50) |
| | Origem | VARCHAR(50) |
| **Comboio** | id | VARCHAR(9) |
| | Nome | VARCHAR(50) |
| | Capacidade | INT |
| **Lugar_Reserva** | Número | INT |
| | Carruagem | INT |
| | Classe | INT |
| | id_Reserva | VARCHAR(9) |
| | id_Comboio | VARCHAR(9) |
| **Lugar_Comboio** | Número | INT |
| | Carruagem | INT |
| | Classe | INT |
| | id_Comboio | VARCHAR(9) |

Tabela 4 - Domínio dos Atributos

- **Multiplicidade**

  Tendo que conta que o modelo lógico foi construído tendo em conta o modelo conceptual, todas as multiplicidades são respeitadas.

- **Integridade de Entidade**

  Todas as chaves primárias das diferentes tabelas são não nulas.

- **Integridade referencial**

  **Reserva** (id_Reserva, id_Viagem, Username, Data_Reserva, Preço)

  **Chaves Estrangeiras:**

  Username **referencia** Cliente (Username) ON DELETE NO ACTION ON UPDATE NO ACTION

  id_Viagem **referencia** Viagem (id_Viagem) ON DELETE NO ACTION ON UPDATE NO ACTION

  **Viagem** (id_Viagem, Destino, Origem, Hora_de_Partida, Hora_de_Chegada, id_Comboio)

  **Chaves Estrangeiras:**

id_Comboio **referencia** Comobio (id_Comboio) ON DELETE NO ACTION ON UPDATE NO ACTION

**Lugar_Reserva** (Numero, Carruagem, Classe, id_Comboio, id_Reserva)
**Chaves Estrangeiras:**
(Numero, Carruagem, Classe, id_Comboio) **referencia** Lugar_Comboio(Numero, Carruagem, Classe, id_Comboio) ON DELETE NO ACTION ON UPDATE NO ACTION
Id_Reserva **referencia** Reserva (id_Reserva) ON DELETE CASCADE ON UPDATE NO ACTION

**Lugar_Reserva** (Numero, Carruagem, Classe, id_Comboio, id_Reserva)
**Chaves Estrangeiras:**
id_Comboio **referencia** Comboio (id_Comboio) ON DELETE NO ACTION ON UPDATE NO ACTION

- **Restrições Gerais**

  Quanto à restrições gerais, existem as seguintes:

  i. Um cliente nunca pode fazer um número de reservas superior ao número de lugares disponíveis;
  ii. O número de lugares reservados para uma dada viagem tem de ser inferior ao número de lugares disponíveis de um comboio.

## 2.2.5 Resultado Final

Após seguir os vários passos para a validação, obteu-se o seguinte:

**Cliente** (Username, Password, Nome, E-mail, NIF, Morada)
**Chave Primária:** Username
**Chaves Alternativas:** E-mail, NCC

**Reserva** (id_Reserva, id_Viagem, Username, Data_Reserva, Preço)
**Chave Primária:** id_Reserva
**Chaves Estrangeiras:** Cliente, Viagem

**Lugar_Reserva** (Numero, Carruagem, Classe, id_Comboio, id_Reserva)
**Chave Primária:** Numero, Carruagem, Classe, id_Comboio, id_Reserva
**Chave Estrangeira:** Numero, Carruagem, Classe, id_Comboio, id_Reserva

**Viagem** (id_Viagem, Destino, Origem, Hora_de_Partida, Hora_de_Chegada, id_Comboio)
**Chave Primária:** id_Viagem
**Chaves Estrangeiras:** id_Comboio

**Comboio** (id_Comboio, Capacidade, Nome, Lugar_Comboio)
**Chave Primária:** id_Comboio
**Chave Alternativa:** Nome

**Lugar_Comboio** (Numero, Carruagem, Classe, id_Comboio)
**Chave Primária:** Numero, Carruagem, Classe, id_Comboio
**Chave Estrangeira:** id_Comboio

## 2.3. Validação do modelo físico

### 2.3.1 Tradução do modelo lógico para físico

Após concluída a validação do modelo lógico, partiu-se para a criação do modelo físico. Tomando como correto o modelo lógico anteriormente criado, usou-se a ferramenta "Forward Engineer.." do software onde foi criado – *MySQL Workbench* – de modo a elaborar autenticamente o código SQL.
O código gerado, respetivo a cada tabela, é o seguinte:

**TABELA CLIENTE:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Cliente` (
  `Username` VARCHAR(12) NOT NULL,
  `Password` VARCHAR(15) NOT NULL,
  `Nome` VARCHAR(50) NOT NULL,
  `E-mail` VARCHAR(50) NOT NULL,
  `NIF` INT UNSIGNED NOT NULL,
  `Morada` VARCHAR(50) NULL,
  PRIMARY KEY (`Username`))
ENGINE = InnoDB;
```

**TABELA COMBOIO:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Comboio` (
  `id_Comboio` VARCHAR(9) NOT NULL,
  `Lotação` INT UNSIGNED NOT NULL,
  `Nome` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id_Comboio`))
ENGINE = InnoDB;
```

**TABELA VIAGEM:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Viagem` (
  `id_Viagem` VARCHAR(9) NOT NULL,
  `id_comboio` VARCHAR(9) NOT NULL,
  `Hora_Partida` DATETIME NOT NULL,
  `Hora_Chegada` DATETIME NOT NULL,
  `Origem` VARCHAR(50) NOT NULL,
  `Destino` VARCHAR(50) NOT NULL,
  PRIMARY KEY (`id_Viagem`),
  INDEX `Comboio_idx` (`id_comboio` ASC),
  CONSTRAINT `Comboio`
    FOREIGN KEY (`id_comboio`)
    REFERENCES `reserva_bilhetes`.`Comboio` (`id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

**TABELA RESERVA:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Reserva` (
  `id_Reserva` VARCHAR(9) NOT NULL,
  `id_Viagem` VARCHAR(9) NOT NULL,
  `Username` VARCHAR(12) NOT NULL,
  `Data_Reserva` DATETIME NOT NULL,
  `Preço` DECIMAL(5,2) UNSIGNED NOT NULL,
  PRIMARY KEY (`id_Reserva`),
  INDEX `Viagem_idx` (`id_Viagem` ASC),
  INDEX `Cliente_idx` (`Username` ASC),
  CONSTRAINT `Viagem`
    FOREIGN KEY (`id_Viagem`)
```

```
    REFERENCES `reserva_bilhetes`.`Viagem` (`id_Viagem`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
  CONSTRAINT `Cliente`
    FOREIGN KEY (`Username`)
    REFERENCES `reserva_bilhetes`.`Cliente` (`Username`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

**TABELA LUGAR_COMBOIO:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Lugar_Comboio` (
  `Numero` INT UNSIGNED NOT NULL,
  `Carruagem` INT UNSIGNED NOT NULL,
  `Classe` INT UNSIGNED NOT NULL,
  `id_Comboio` VARCHAR(9) NOT NULL,
  PRIMARY KEY (`Numero`, `Carruagem`, `Classe`, `id_Comboio`),
  INDEX `Reserva_idx` (`id_Comboio` ASC),
  CONSTRAINT `ComboioL`
    FOREIGN KEY (`id_Comboio`)
    REFERENCES `reserva_bilhetes`.`Comboio` (`id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

**TABELA LUGAR_RESERVA:**

```
CREATE TABLE IF NOT EXISTS `reserva_bilhetes`.`Lugar_Reserva` (
  `Numero` INT UNSIGNED NOT NULL,
  `Carruagem` INT UNSIGNED NOT NULL,
  `Classe` INT UNSIGNED NOT NULL,
  `id_Comboio` VARCHAR(9) NOT NULL,
  `id_Reserva` VARCHAR(9) NOT NULL,
  PRIMARY KEY (`Numero`, `Carruagem`, `Classe`, `id_Comboio`,
`id_Reserva`),
  INDEX `ReservaL_idx` (`id_Reserva` ASC),
  INDEX `Lugar_idx` (`Numero` ASC, `Carruagem` ASC, `Classe`
ASC, `id_Comboio` ASC),
  CONSTRAINT `Reserva`
```

```
    FOREIGN KEY (`id_Reserva`)
    REFERENCES `reserva_bilhetes`.`Reserva` (`id_Reserva`)
    ON DELETE CASCADE
    ON UPDATE NO ACTION,
  CONSTRAINT `Lugar`
    FOREIGN KEY (`Numero` , `Carruagem` , `Classe` ,
`id_Comboio`)
    REFERENCES `reserva_bilhetes`.`Lugar_Comboio` (`Numero` ,
`Carruagem` , `Classe` , `id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB;
```

### 2.3.2 Análise de transações/relações

Neste ponto proceder-se-á análise de transações/relações do modelo físico de forma a entender de maneira mais exata a atividade nas diferentes tabelas.

De modo a obter resultados, testar-se-á relativamente às *queries* anteriormente listadas:

1. **Apresentar a lista dos lugares reservados e das datas das reservas efetuadas por um determinado cliente.**
2. **Identificar o preço de uma reserva efetuada por um determinado cliente.**
3. **Apresentar a lista dos destinos para uma origem.**
4. **Identificar o número total de lugares livres num comboio para uma determinada viagem.**
5. **Fazer uma reserva para uma viagem, para uma determinada data para um lugar específico de uma classe do comboio.**
6. **Insere um cliente na base de dados.**

Tendo em conta que a base de dados foi toda criada pensando na vista do utilizador, foram só criadas *queries* que dissessem respeito a essa mesma vista.

Elaborando então uma tabela sobre a atividade destas, e tendo em conta que existem apenas leituras e inserções:

| Transação/ Relação | 1. | | 2. | | 3. | | 4. | | 5. | | 6. | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | L | I | L | I | I | I | L | I | L | I | L | I |
| **Cliente** | X | | | | | | | | | | | X |

| | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **Reserva** | **X** | | **X** | | | | **X** | | **X** | **X** | | |
| **Lugar_Reserva** | | | | | | | **X** | | | **X** | | |
| **Viagem** | **X** | | | | **X** | | **X** | | **X** | | | |
| **Comboio** | | | | | | | **X** | | | | | |
| **Lugar_Comboio** | | | | | | | | | | | | |

Tabela 5 - Análise de Transações

**L – Leitura; I – Inserção**

Como é possível retirar da tabela, as tabelas com mais atividade são a Reserva e a Viagem.

### 2.3.3 Projeção do espaço ocupado

**TABELA CLIENTE**

| Atributo | Tipo | Espaço |
|---|---|---|
| Username | VARCHAR(12) | 13 Bytes |
| Password | VARCHAR(15) | 16 Bytes |
| Nome | VARCHAR(50) | 51 Bytes |
| E-mail | VARCHAR(50) | 51 Bytes |
| NIF | INT | 4 Bytes |
| Morada | VARCHAR(50) | 51 Bytes |

Tabela 6 - Tamanho duma entrada na tabela Cliente

Cada entrada na tabela **Cliente** ocupará 186 Bytes.

**TABELA RESERVA**

| Atributo | Tipo | Espaço |
|---|---|---|
| id_Reserva | VARCHAR(9) | 10 Bytes |
| id_Viagem | VARCHAR(9) | 10 Bytes |
| Username | VARCHAR(12) | 13 Bytes |
| Data_Reserva | DATETIME | 8 Bytes |
| Preço | DECIMAL(5,2) | 3 Bytes |

Tabela 7 - Tamanho duma entrada na tabela Reserva

Cada entrada na tabela **Reserva** ocupará 44 Bytes.

**TABELA LUGAR_RESERVA**

| Atributo | Tipo | Espaço |
|---|---|---|
| Numero | INT | 4 Bytes |
| Carruagem | INT | 4 Bytes |
| Classe | INT | 4 Bytes |
| id_Comboio | VARCHAR(9) | 10 Bytes |
| id_Reserva | VARCHAR(9) | 10 Bytes |

Tabela 8 - Tamanho duma entrada na tabela Lugar_Reserva

Cada entrada na tabela **Lugar_Reserva** ocupará 32 Bytes.

**TABELA VIAGEM**

| Atributo | Tipo | Espaço |
|---|---|---|
| id_Viagem | VARCHAR(9) | 10 Bytes |
| Id_Comboio | VARCHAR(9) | 10 Bytes |
| Hora_Partida | DATETIME | 8 Bytes |
| Hora_Chegad | DATETIME | 8 Bytes |
| Origem | VARCHAR(50) | 51 Bytes |
| Destino | VARCHAR(50) | 51 Bytes |

Tabela 9 - Tamanho duma entrada na tabela Viagem

Cada entrada na tabela **Viagem** ocupará 138 Bytes.

**TABELA COMBOIO**

| Atributo | Tipo | Espaço |
|---|---|---|
| id_Comboio | VARCHAR(9) | 10 Bytes |
| Lotação | INT | 4 Bytes |
| Nome | VARCHAR(50) | 51 Bytes |

Tabela 10 - Tamanho duma entrada na tabela Comboio

Cada entrada na tabela **Comboio** ocupará 65 Bytes.

**TABELA LUGAR_COMBOIO**

| Atributo | Tipo | Espaço |
|---|---|---|
| Numero | INT | 4 Bytes |
| Carruagem | INT | 4 Bytes |
| Classe | INT | 4 Bytes |
| id_Comboio | VARCHAR(9) | 10 Bytes |

Tabela 11 - Tamanho duma entrada na tabela Lugar_Comboio

Cada entrada na tabela **Lugar_Comboio** ocupará 22 Bytes.

Após contactar a empresa cliente de modo a obter projeções sobre o uso da plataforma *online*, foram adiantadas as seguintes aproximações diárias:

- **Clientes:** São esperados um registo de 75 clientes por dia;
- **Reservas:** São esperadas cerca de 4000 reservas por dia;
- **Viagens:** São esperadas cerca de 20 viagens por dia;
- **Lugar_Comboio:** Cada comboio tem em média 200 lugares;
- **Lugar_Reserva:** Espera-se que todos os lugares sejam reservados para todas as viagens;

Calcula-se ainda que empresa tenha disponíveis sempre 15 comboios todos os dias.

15 * 65 = 965

A partir dos dados obtidos, pode-se chegar a uma estimativa de quanto aumentará o espaço a cada dia:

75 * 186 = 13950
4000 * 44 = 176000
20 * 138 = 2760
200 * 32 = 6400
200 * 22 = 4400
**Total =** 203510 = 1.55Mb

Alem dos 965 bits adicionais, acresce 1.55Mb a cada dia que passa.

### 2.3.1 Construção das Queries em SQL

Uma vez que todas as queries construídas precisam de argumentos, todo o código realizado trata-se de *procedures*, não existindo qualquer view.

**QUERY 1**

Apresentar a lista dos lugares reservados por um determinado cliente.

```
USE reserva_bilhetes;

DELIMITER $$
CREATE PROCEDURE lugaresReservados(IN cliente VARCHAR(9))
BEGIN
      SELECT  R.Username,  Nome,  id_Reserva,  Origem,  Destino,
Hora_Partida
            FROM reserva AS R INNER JOIN cliente AS C
                  ON C.Username = R.Username
            INNER JOIN Viagem AS V
                        ON V.id_Viagem = R.id_Viagem

    WHERE R.Username=cliente;
END $$
```

**QUERY 2**

Identificar o preço de uma reserva efetuada por um determinado cliente.

```
DELIMITER $$
CREATE  PROCEDURE  precoReserva(IN  clt  VARCHAR(9),  IN  rsv
VARCHAR(9))
BEGIN
SELECT Preço
      FROM reserva
            WHERE Username = clt AND id_Reserva = rsv;
END $$
```

**QUERY 3**

Apresentar a lista dos destinos para uma origem.

```
DELIMITER $$
CREATE PROCEDURE listaDestinos(IN o VARCHAR(50))
BEGIN
      SELECT Origem, Destino
            FROM viagem
    WHERE origem = o;
```

**QUERY 4**

Identificar o numero total de lugares livres num comboio para uma viagem.

```
USE reserva_bilhetes;

DELIMITER $$
CREATE PROCEDURE lugaresLivres(IN viagem VARCHAR(9))
BEGIN
      SELECT Lotação -
            (SELECT COUNT(VI.id_Viagem)
                  FROM lugar_reserva AS LR INNER JOIN reserva AS
R
              ON R.id_Reserva = LR.id_Reserva
              INNER JOIN viagem AS VI
                  ON VI.id_Viagem = R.id_Viagem
            WHERE VI.id_Viagem = viagem)

             FROM viagem AS V INNER JOIN comboio AS C
                  ON V.id_Comboio = C.id_Comboio
       WHERE V.id_Viagem = viagem;
END

CALL lugaresLivres('viagem1');
```

**QUERY 5**

Fazer uma reserva para uma viagem, para uma determinada data para um lugar específico de uma classe do comboio.

```
USE reserva_bilhetes;

DELIMITER $$
CREATE PROCEDURE efectuaReserva (IN idReserva VARCHAR(9), IN
idViagem VARCHAR(9), IN Userx VARCHAR(12), IN DataRes DATETIME,
IN preco DECIMAL(5,2), IN num INT, IN carr INT, IN class INT)
BEGIN
  DECLARE ErroTransacao BOOL DEFAULT 0;
  DECLARE idComboio VARCHAR(9);
  DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET ErroTransacao =
1;

  START TRANSACTION;
  SET idComboio = (SELECT id_Comboio FROM viagem AS V
```

```
                    WHERE V.id_Viagem = idViagem);

  IF NOT EXISTS
    (SELECT * FROM reserva
      WHERE reserva.id_Reserva = idReserva)
  THEN
    INSERT INTO
reserva(id_Reserva,id_Viagem,Username,Data_Reserva,Preço)
           VALUES (idReserva,idViagem,Userx,DataRes,preco);
    INSERT INTO
lugar_reserva(Numero,Carruagem,Classe,id_Comboio,id_Reserva)
           VALUES (num,carr,class,idComboio,idReserva);
  ELSE
    INSERT INTO
lugar_reserva(Numero,Carruagem,Classe,id_Comboio,id_Reserva)
           VALUES (num,carr,class,idComboio,idReserva);
  END IF;

  IF ErroTransacao THEN
    ROLLBACK;
  ELSE
    COMMIT;
  END IF;

END$$

CALL          efectuaReserva('res9','viagem1','usr4','2016-01-02
16:34:27',30,5,1,1);
```

**QUERY 6**

Insere um cliente na base de dados.

```
USE reserva_bilhetes;

--Criação de um cliente

DELIMITER $$
CREATE PROCEDURE insere_cliente
    (IN USER VARCHAR(12), Pass VARCHAR(15), Name VARCHAR(50),
Email VARCHAR(50), Nfiscal INT, Adress VARCHAR(50))
```

```
BEGIN
    -- Declaraçãoo de um handler para tratamento de erros.
    DECLARE ErroTransacao BOOL DEFAULT 0;
    DECLARE CONTINUE HANDLER FOR SQLEXCEPTION SET ErroTransacao
= 1;
    -- Início da transação
    START TRANSACTION;
    -- 1ª Operação - INSERT
    INSERT INTO cliente
        (Username, Password, Nome, E-mail, NIF, Morada)
        VALUES(USER,Pass,Name,Email,Nfiscal,Adress);
    IF ErroTransacao THEN
        -- Desfazer as operações realizadas.
        ROLLBACK;
    ELSE
        -- Confirmar as operações realizadas.
        COMMIT;
    END IF;
END $$
```

# 3. Conclusões e Trabalho Futuro

Terminada a elaboração do modelo físico e respetivas queries, é momento de olhar para o criado e criticar de maneira construtiva o que foi produzido ao longo dos últimos meses durante longas horas.

Relativamente à construção do modelo lógico, embora sendo uma tarefa relativamente simples e fácil de fazer, tivemos alguma dificuldade no que toca a definir um lugar. No final, concordamos que seria um atributo multivalorado do comboio e não uma entidade por si mesma, como inicialmente consideramos.
No que fiz respeito ao modelo lógico, foi onde encontramos mais dificuldades – na atribuição das chaves estrangeiras e tipo de ligações. Nas tabelas multi-valoradas foi difícil escolher qual o tipo de ligações entre elas, chegando no entanto a resultado que nos pareceu correcto – uma ligação entre as tabelas Lugar_Reserva e Lugar_Comboio.
Finalmente, relativo ao último modelo, este foi muito simples de elaborar uma vez que recorremos ao *Forward Engineering* do 'MySQL Workbench'. As queries, após estudo de SQL foram algo simples de criar.

Em modo geral, consideramos que a base de dados têm dois aspetos que poderiam sem dúvida ser evoluídos: funciona apenas para viagens em expresso, isto é, que conectem duas cidades sem qualquer paragem, o que é raro de acontecer na maioria das viagens que acontecem no quotidiano. Outro aspeto que deveria ser melhorado é o aumento das vistas existentes – no momento só existe assim como só possível a vista do utilizador, visto a base de dados assim o restringe.

Em forma de conclusão, vista a situação proposta bem como as suas restrições, embora tenha alguns problemas, consideramos que obtemos um resultado final positivo e que realmente contribuiu para o nosso desenvolvimento nesta unidade curricular.

# Referências


- Connolly, T., Begg, C., Database Systems, A Practical Approach to Design, Implementation, and Management, Addison-Wesley, 4ª Edição, 2004.
- *Data Type Storage Requirements.* [online] Disponível em: https://dev.mysql.com/doc/refman/5.7/en/storage-requirements.html [Acedido a 27 de outubro]
- Damas, Luís, SQL – Structured Query Language, FCA – Editora de Informática, 6ª Edição