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

Unidade Curricular de Bases de Dados

Ano Lectivo de 2016/2017

Comboios Flavienses

Pedro Bragança(A70886), Daniel Vieira(a73974), Rafael Silva(A74264)

Novembro de 2016

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Comboios Flavienses

Pedro Bragança(A70886), Daniel Vieira(a73974),

Rafael Silva(A74264)

Novembro de 2016

Resumo

Este projeto insere-se na unidade curricular de Base de Dados, e consiste em levar a prática, usando para isso o MySql, as temáticas dadas na aulas teóricas e práticas, isto é, aprofundar os nossos conhecimentos no modelo conceptual, lógico e físico.
Para isso foi nos proposto um trabalho em que tivéssemos que fazer uma gestão de reservas de bilhete de comboios nacionais quer internacionais.
Para que este relatório seja completo vamos detalhar cada passo que façamos para a apresentação e resolução do problema citado no paragrafo anterior.
No final iremos apresentar uma analise crítica, objetiva, sucinta do nosso trabalho, evidenciando aquilo que tivemos mais ou menos dificuldades, isto é, os obstáculos que percorremos no desenrolar deste trabalho, bem como anexos e tabelas para demonstrar de uma forma mais clara o processo de elaboração deste problema que nos foi dado.

Área de Aplicação: Desenho e arquitectura de Sistemas de Bases de Dados.
Palavras-Chave: Bases de Dados Relacionais, Modelação, MySql, Entidades, Atributos, Chaves.

Índice

Índice de Figuras

Índice de Tabelas

1. Introdução

1.1 Contextualização

Este projeto que está inserido na unidade curricular de Base de Dados, unidade curricular esta que insere-se no Mestrado Integrado de Engenharia Informática teve, ao contrário dos anos anteriores, uma tema que nos foi dado pelo professor. Consiste na gestão de reservas de bilhetes de comboios nacionais e internacionais. A este tema foi nos dada liberdade para que façamos um enunciado que mostraremos nos capítulos seguintes.

1.2 Apresentação do Caso de Estudo

A tecnologia está ao nosso dispor como um bem para a sociedade,certo?
Logo para quê usar arquivos em papel, com grandes gastos em termos de horários de trabalhos bem como uma utilização de espaços reais tanto maior quanto aquilo que armazenamos, quando podemos usar tudo informaticamente, isto é, à base de uma plataforma eletrónica que podemos aceder a qualquer hora?
É aqui que Base de Dados entra, isto é, é uma forma de poupar dinheiro quer nas horas de trabalho de cada funcionário que se iriam precisar para arquivar ficheiros, bem como o espaço que lhe e concedido para guardar esses ficheiros arquivados. É tudo feito através de uma plataforma eletrónica, como já dissemos anteriormente, logo haverá evidentemente menos custos associados a horas de trabalho bem como ao espaço real.
Mas a principal vantagem é a analise de dados e para isso apresentamos queries, isto é, “programas” que façam perguntas em relação aos dados recolhidos para que depois possamos fazer observações, estudos sobre eles mesmos.

Logo para isso, representaremos um sistema de gestão de reservas de bilhetes de comboios de viagens nacionais e internacionais, como nos foi pedido, organizados consoante o levantamento de requisitos.

1.3 Motivação e Objectivos

Vivemos num mundo em constante evolução, logo não tardaria em que essa evolução chegasse a gestão e organização de dados. Em vez dos já velhos métodos de organização em papel alocados num grande espaço real, passamos a usar o a tecnologia para fazer a organização e gestão desses mesmos dados. E é no geral isto que se trata, uma boa gestão de dados.

No entanto isto trata-se de um sistema de reserva de bilhetes, logo evidentemente a grande vantagem disto é que podemos fazer uma reserva de bilhetes sem sair de casa, poupando tempo em deslocações a um espaço real de venda/reserva de bilhetes, tempo este que como se diz, custa dinheiro. Nos tempos de hoje em que vivemos uma crise financeira, qualquer coisa para poupar irá ser visto de bons olhos pelo consumidor e isto não foge à regra.

Evidentemente também há vantagens sem ser monetárias, como por exemplo, a comodidade de estar em casa a reservar, sem filas de espera, podendo fazer a reserva à hora que quiser, entre outras coisas.. Com este sistema de gestão de dados são só vantagens.

Logo neste projeto entra um sistema de reserva de bilhetes de comboios onde cada cliente poderá ver as viagens turísticas que poderá fazer, com informações precisas de cada viagem com o intuito de ser um serviço fácil.

1.4Estrutura do Relatório

Com este relatório iremos incidir evidentemente nas matérias dadas na aulas teóricas e práticas desde a apresentação de requisitos, modelo conceptual (identificação de cada entidade e atributos e os relacionamentos entre eles), modelos lógicos e fisicos com a ajuda de SQL para a construção de um Base de Dados propriamente dita bem como a criação de queries.
Tudo isto como tema que foi dado pelo professor: Reserva de bilhetes de viagens de comboio nacionais e internacionais.

1.5 Levantamento e análise de requisitos

- Na reserva há várias viagens no qual o cliente tem que escolher a que lhe mais convém. Em cada reserva só se reserve 1 viagem, logo se quiser 2 viagens tem que fazer uma reserva de novo.
- Na viagem há varias informações inportantes como por exemplo: Local da partida, Local de chegada, Hora de partida, Hora de chegada bem como o preço(toda esta informações consegue-se saber no momento em que se faz a reserva)

- Para fazer a reserva é preciso ser cliente e para ser cliente é preciso dar os seus dados pessoas como por exemplo, nome, nif, gênero, telemóvel, email, senha.
- Há 2 tipos de comboio,o nacional e o internacional.as letras correspondents são ‘N’ e ‘I’, respetivamente.
- Ao fazer a reserva, irá lhe ser dado um lugar que varia consoante o comboio, isto é, se é do ‘N’ ou ‘I’ consoante ser uma viagem Nacional ou Internacional, e o nº do lugar consoante a capacidade do comboio. Exemplo: I-31, comboio internacional, lugar 31.

2. Modelo conceptual

Para a 1º fase do trabalho, apresentamos o 1º modelo que aprendemos nas aulas práticas e teórica, o modelo conceptual.

Para isso, apresentaremos o modelo conceptual propriamente dito (elaboramos este modelo com o programa “Terra.er”), bem como uma análise de cada entidade, atributos, chaves e os relacionamentos que o compõem.

2.1-Identificação das Entidades

Entidade	Definição
Reserva	Identifica a reserva propriamente dita sendo que tem muitos atributos que a caraterizam
Cliente	Identifica o cliente que pretende reservar o bilhete com todos os seus dados pessoais
Comboio	Identificação do comboio no qual a viagem vai ser feita
Viagem	Identifica todas as informações precisas para que o cliente possa fazer essa mesmo viagem

Tabela 1 - Representação das Entidades

2.2-Relacionamento entre Entidades

Entidade	Multiplicidade	Relacionamento	Multiplicidade	Entidade
Cliente	1	efetua	N	*Reserva*
Reserva	N	assegura	1	*Viagem*
Viagem	N	cumpre	1	*Comboio*

Tabela 2 - Representação dos Relacionamentos

2.3- Identificação dos Atributos, Domínios e as relações com cada entidade

Entidade	Atributos	Definição de Atributos	Tipo e Tamanho	Chave Primária	Chave Estrangeira	Nulo
Cliente	Id_cliente	Identifica um cliente	Inteiro positivo	Sim	Não	Não
	nome	Nome do Cliente	String com comprimento de 50 carateres(máximo)	Não	Não	Não
	NIF	Número de contribuinte do Cliente	Inteiro positivo (9 algarismos)	Não	Não	Sim
	Gênero	Sexo do Cliente	Letra 'M' ou 'F'	Não	Não	Não
	telemóvel	Número de telemóvel do Cliente	Inteiro positivo com 9 algarismos	Não	Não	Não
	Senha	Senha do Cliente	String com comprimento de 7 caracteres no máximo	Não	Não	Não
	email	Email do Cliente	String com comprimento de 45 carateres(máximo)	Não	Não	Não
Reserva	Id_reserva	Identificação da reserva	Inteiro maior que 0	Sim	Não	Não

	data_reserva	Data da Viagem	Data	Não	Não	Não
	custo	Custo do bilhete	Decimal com 7 numeros	Não	Não	Não
	Cliente_id_Cliente	Identifcação do cliente que faz reserva	Int	Não	Sim	Não

Entidade	**Atributos**	**Definição de Atributos**	**Tipo e Tamanho**	**Chave Primária**	**Chave Estrangeira**	**Nulo**
Viagem	Id_viagem	Identifica a viagem	Inteiro maior que 0	Sim	Não	Não
	Partida	Identifica o local da partida	String com comprimento de 20 carateres(máximo)	Não	Não	Não
	Destino	Identifica o destino da viagem	String com comprimento de 20 carateres(máximo)	Não	Não	Não
	Data_Hora_partida	Hora da partida da viagem incluindo a data	Data	Não	Não	Não
	Data_Hora_chegada	Hora de chegada incluindo a data	Data	Não	Não	Não
	Preço_Primeira_Classe	Preço da viagem de	Float	Não	Não	Não

		Primeira Classe				
	Ocupação_Primeira_Classe	Identifica a ocupação no comboio	Inteiro positivo	Não	Não	Não
	Ocupação_Classe_Económica	Identifica a ocupação no comboio	Inteiro positivo	Não	Não	Não
	Preço_Economica	Preço da viagem economica	Float	Não	Não	Não
Comboio	Id_comboio	Identificação do comboio	Inteiro positivo	Sim	Não	Não
	Tipo_comboio	Identifica qual o comboio	'D' ou 'I' consoante o comboio	Não	Não	Não
	Capacidade	Capacidade do comboio	Inteiro positivo até 999	Não	Não	Não

Tabela 3 - Dicionário dos Dados

Devido a termos usado um atributos multivalorado na entidade Comboio, optamos por fazer uma tabela à parte mostrando as caraterística desse atributo multivalorado.

Entidade: *Comboio*

2.4 Atributo Multi-valorado: *Lugar*

Atributo	Definição do atributo	Tipo e Tamanho	Chave Primária	Chave Estrangeira	Nulo
N_Lugar	Identifica o nº de lugar no comboio	Inteiro positivo	Sim	Não	Não
Classe	Identifica a classe do comboio	String com máximo de 20 carateres	Não	Não	Não
Comboio_id_Comboio	Identificação do comboio em relação aos restantes	Inteiro positivo	Sim	Sim	Não

Tabela 4 - Dicionário de Dados Atributo Multivalorado

2.5 -Explicação do porquê de cada chave primária

2.5.1 Cliente

- Nesta entidade, tivemos em mente em colocar o NIF como sendo chave primária, mas devido a observação de que não sendo obrigatório, pusemos essa questão de lado.
Pensamos em Nome mas descartamos logo essa porque podia haver pessoas diferentes com nomes iguais o que levaria a que houvesse erros nos registos
Logo a única ideia foi colocar um id_cliente que é auto incrementado e que leva a que haja singularidade de registos.

2.5.2 Reserva

- Nesta entidade foi muito rápido pensar em criar uma id_reserva que faz com que haja diferentes registos devido aos nossos atributos iniciais não poderem ser obrigados a ser diferentes uns dos outros.

2.5.3 Viagem

- No inicio ainda pensamos em usar o horário de partida do comboio mas visto que existe a possibilidade de haver dois comboios,2 viagens que têm a mesma hora de partida, descartamos essa ideia.
Logo, continuando com o mesmo pensamento, criamos uma id_viagem, logo não haverá multiplicidade de registos.

2.5.4 Comboio

- Visto que cada comboio pode fazer N viagens, criamos em consonância com o tipo de pensamento das entidades anteriores, uma id_comboio que vai registar cada comboio sendo um atributo auto-incrementado.

2.6 Verificação do modelo conceptual

Neste subcapítulo iremos analisar as redundâncias que poderá haver e eliminá-los para um melhor modelo.
Como não temos relacionamentos 1:1,não poderemos analisar evidentemente.
Em relação a possíveis redundâncias não encontramos nenhum caso em que possa haver vários caminhos entre as mesma entidades o que poderia levar, caso houvesse, a que o cliente tivesse a mesma informação variando somente no acesso a esta.

2.7 Verificação se existe consonância entre o modelo conceptual e o que se quer para o projeto

Num mero texto, iremos analisar o modelo conceptual de acordo com aquilo que se quer.
Um cliente ao querer fazer a reserva, vão lhe ser dadas varias viagens e de acordo com aquilo que deseja irá escolher uma delas.

3. Modelo lógico

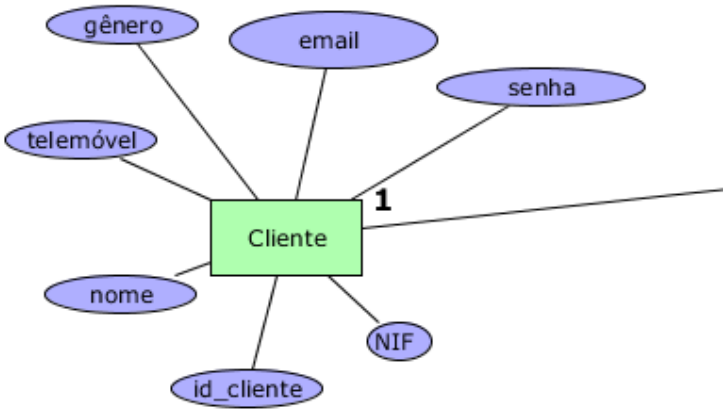
3.1 Ilustração do modelo lógico

Iremos agora apresentar o modelo lógico da nossa base de dados. Para isso recorremos À utilização da ferramenta do MYSQL Workbench.

Na conversão do modelo conceptual para o modelo lógico deixaremos de utilizar os termos entidades e relacionamentos passando então a retratar a mesma ideia através de tabelas.

Iremos agora explicar tudo de forma mais detalhada, incluíndo todas as decisões tomadas, desde a abordagem às entidades, os relacionamentos e a sua cardinalidade, atributos multivalorados, etc.

3.2 Descrição de cada relacionamento



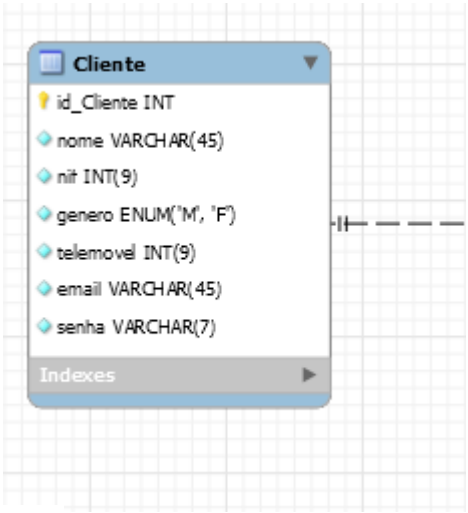
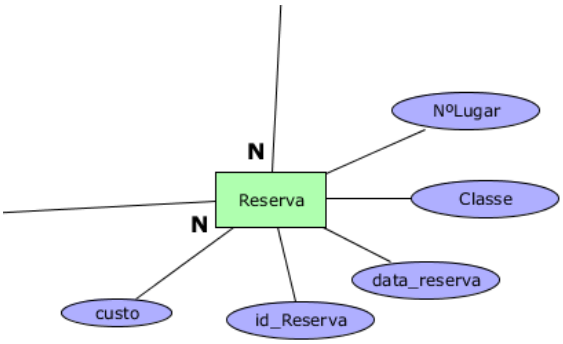


Figura 1 - conversão Entidade Cliente

Cliente = {id_cliente, NIF, genero, telemovel, senha, email, telemovel}

Chave(S) Primária(S) = id_cliente



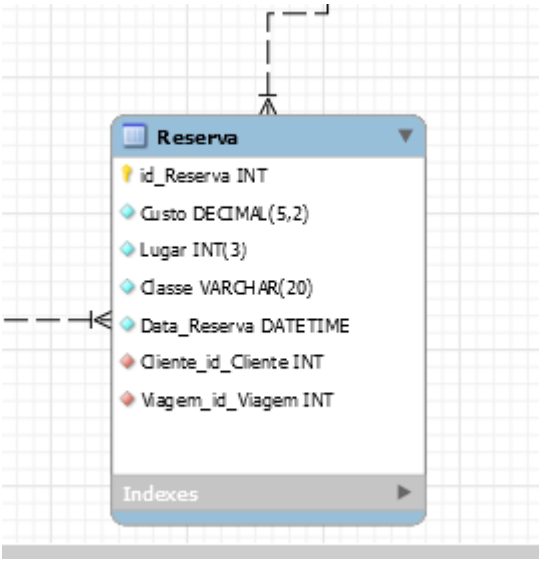


Figura 2 - Conversão da Entidade Reserva

Reserva = {id_reserva, custo, data_reserva, classe, NºLugar}

Chave(s) Primária(S) : id_reserva

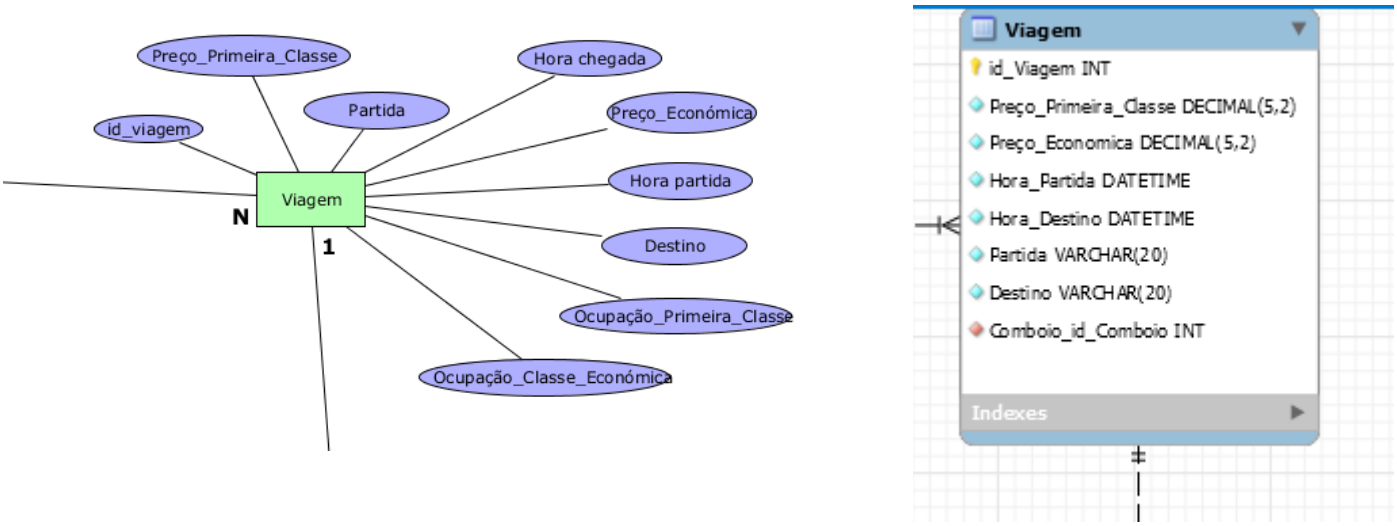


Figura 3 - Conversão da Entidade Viagem

Viagem = {id_viagem, Preço_Primeira_classe, Partida, Hora Chegada, Preço_economica, Hora Partida, Destino}

Chave(s) Primária(s) = id_viagem

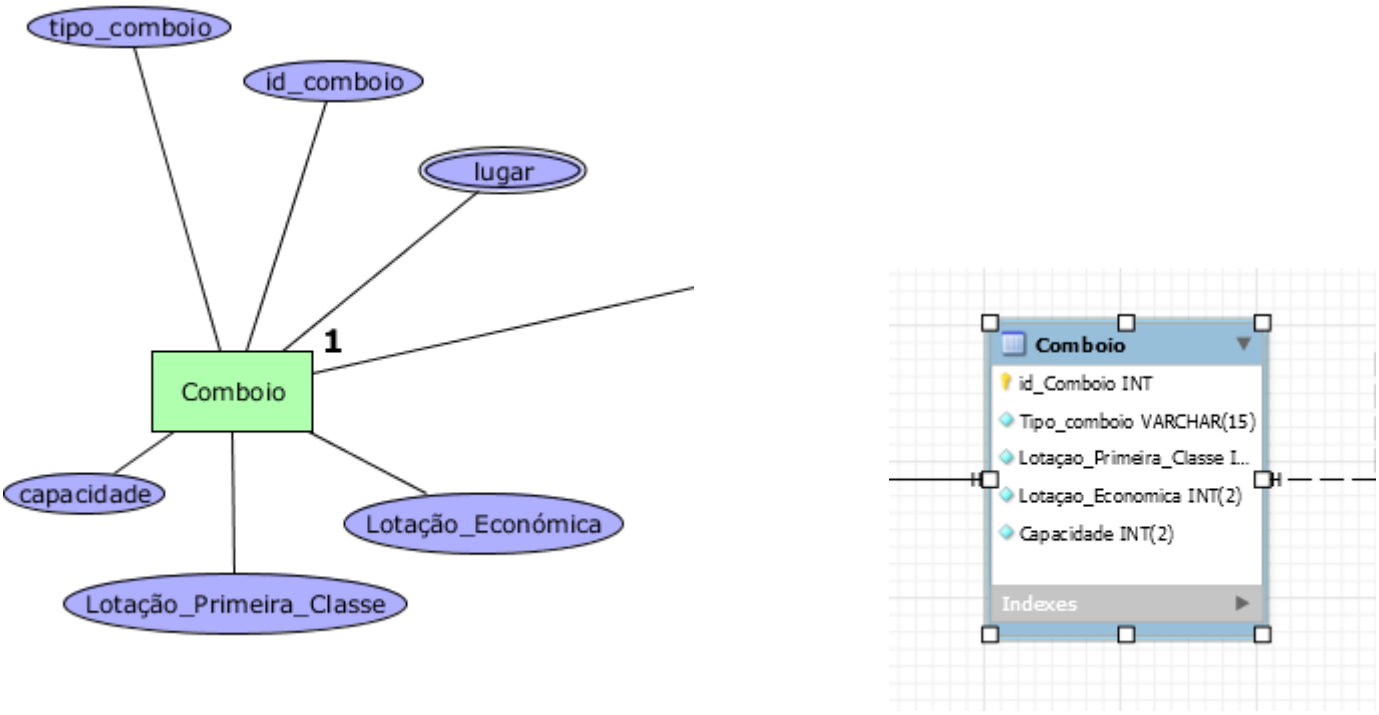


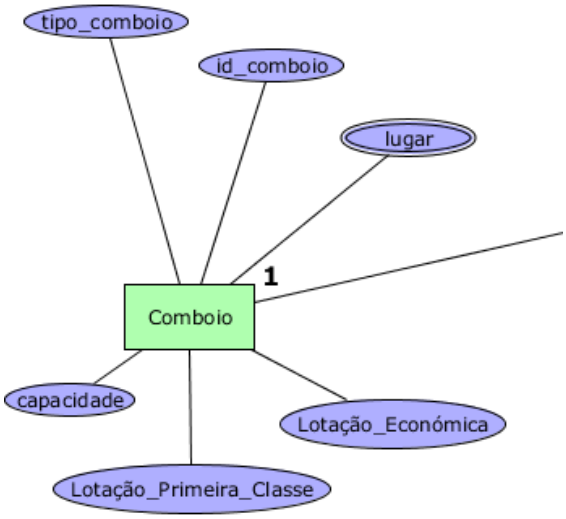
Figura 4 - Conversão da Entidade Comboio

Comboio = {id_comboio, tipo_comboio, Lotação_Economica, Lotação_Primeira_Classe, capacidade, lugares}
Chave(s) Primária(s) = id_comboio

3.3 Relacionamentos de 1 para 1 (1:1)

No nosso trabalho não se encontram presentes relacionamentos que apresentem uma definição de um para um.

3.4 Relacionamentos de 1 para muitos (1:n)



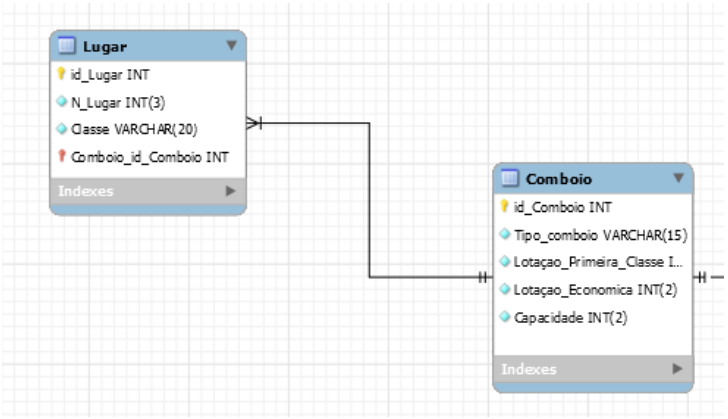
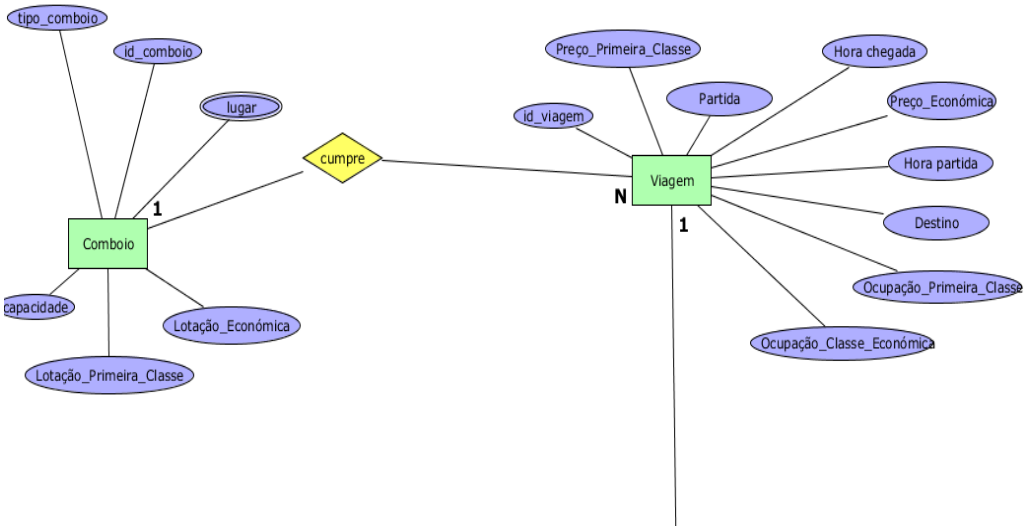


Figura 5 – Conversão da Relação entre o atributo multivalorado Lugar e a entidade comboio

Lugar = { id_lugar, N_lugar, Classe, comboio_id_comboio}

Chave(s) Primária(s) : id_Lugar

Chave (s) Estrangeira(s) : comboio_id_comboio - referência a Comboio (id_comboio)



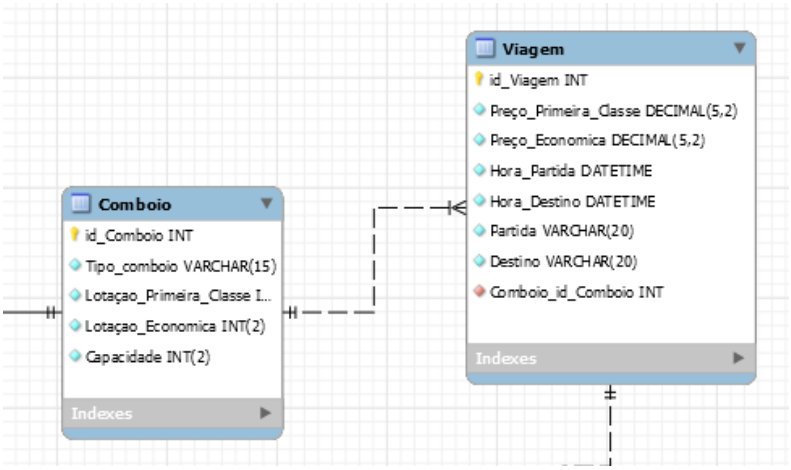


Figura 6 – conversão relação entre as Entidades
Comboio e Viagem

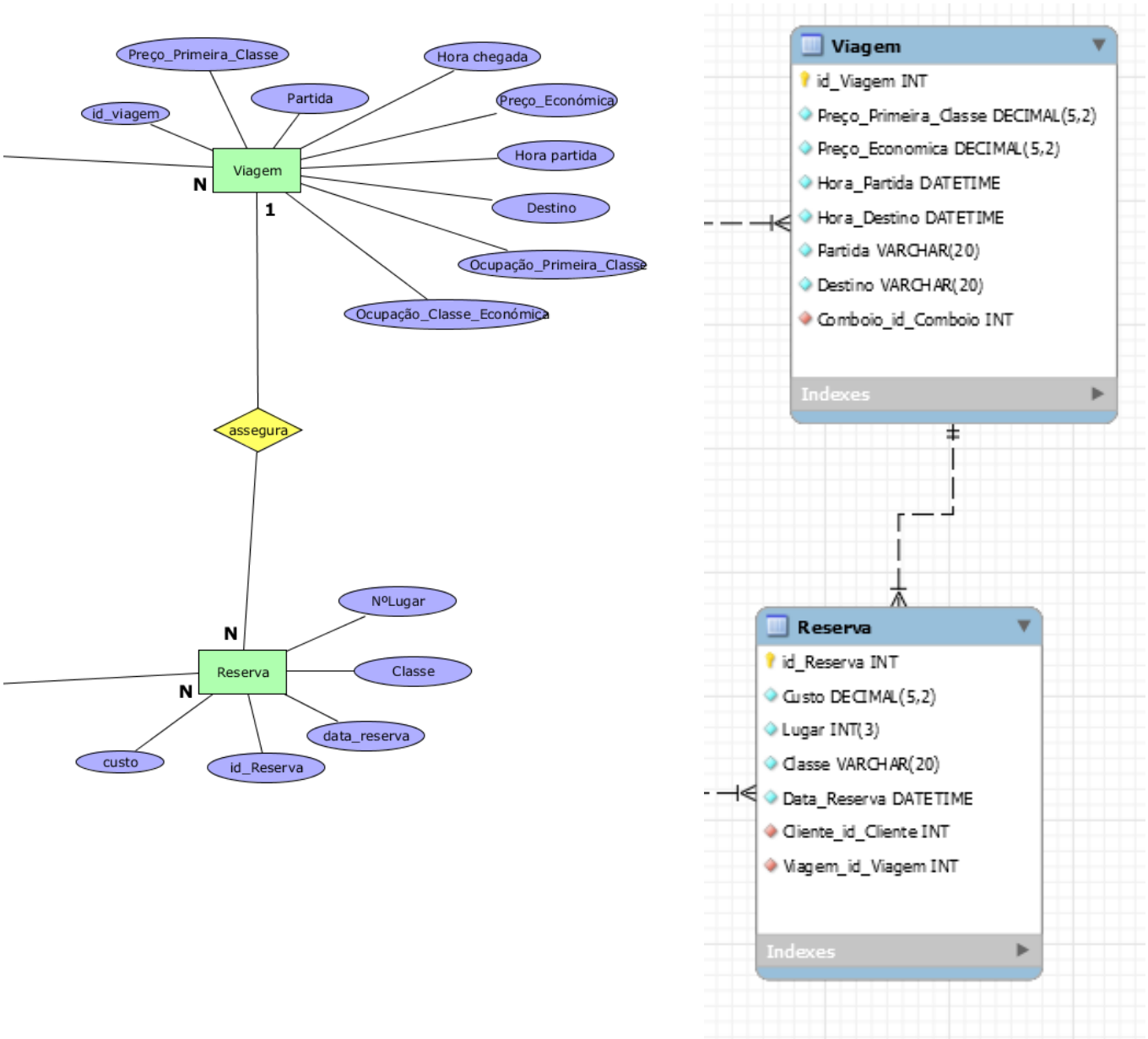
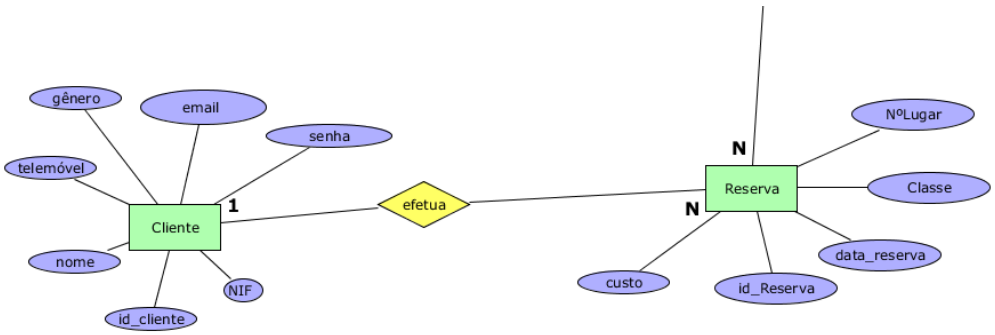


Figura 7 – Conversão da relação entre
as entidades Viagem e Reserva



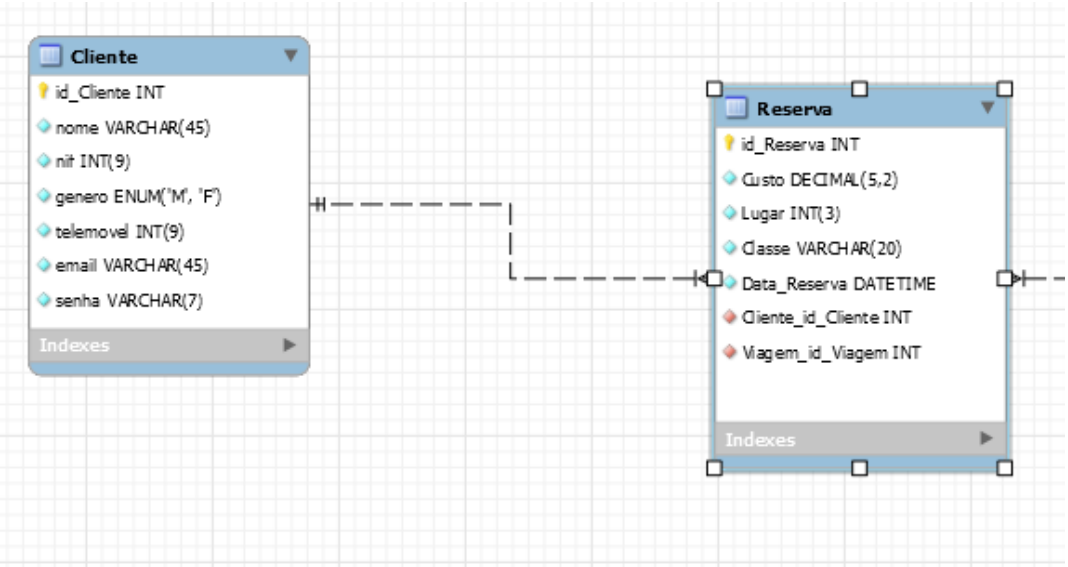


Figura 8 – Conversão entre as entidades Cliente e reserva

3.5 Validação das relações recorrendo à teoria da normalização

De modo a que fosse possível validar as relações impostas pelo nosso modelo lógico, foi necessário recorrer à normalização para que se garantisse que o nosso trabalho não apresente problemas futuramente.
A teoria da normalização tem como principal objetivo identificar as relações com base nas suas chaves e dependências entre os atributos e ainda identificar relações que possuam redundância de dados, impedindo que desse modo existam irregularidades na utilização e atualização da base de danos, garantindo ainda a integridade e flexibilidade dos dados.
Assim, iremos aplicar a normalização até à terceira forma normal (3FN) validando deste modo o modelo lógico idealizado.

3.5.1 Primeira Forma Normal (1FN)

Esta regra é a primeira a ser aplicada e tem como objetivo a identificação de elementos informação repetidos nas tabelas, ou seja, verificar se todos os valores que correspondem a atributos são atómicos.
Esta estratégia consiste em pesquisar informação duplicada entre as tabelas e facilitar a identificação de uma chave primária por tabela.
Aplicando esta Norma no nosso modelo, podemos constatar que o mesmo já se encontra normalizado para a 1FN.

3.5.2 Segunda Forma Normal (2FN)

Esta segunda forma de normalização, após a aplicação da Primeira Forma Normal, teremos de confirmar se todos os atributos que não são chaves candidatas estão dependentes de qualquer uma chave candidata, ou seja, não é possível que existam dependências parciais.
Esta ideia consiste essencialmente em verificar as tabelas com uma chave primária composta, pois as tabelas com uma só chave primária já respeitam a Segunda Forma Normal.

Deste modo, através deste paradigma avançamos para a analise de cada tributo chave verificando a sua relevância na respetiva tabela, verificando ainda quais os atributos que não são dependentes da chave primária da tabela.
Aplicando esta regra ao nosso modelo, verificamos que o mesmo já cumpre todos os requisitos da 2FN.

3.5.3 Terceira Forma Normal (3FN)

Nesta última forma que iremos aplicar, e após a aplicação das duas anteriores, teremos também de analisar as dependências transitivas, ou seja, analisar todos os atributos que não são chaves candidatas serão alcançadas entre relações apenas de uma forma.
Esta Terceira Forma consiste em identificar os atributos que são dependentes de outros atributos que não são chaves. Estes não poderão ser aceites na tabela.
Verificando o nosso modelo verificamos que este também respeita a 3FN.
Podemos então concluir finalmente que o modelo lógico realizado respeita estas três Normas, encontrando-se então normalizado.

3.5.4 Verificar restrições de integridade

Aqui vamos tratar de mencionar o modo como foram tratadas as operações de remoção e/ou atualização dos dados existentes na nossa base dados e as limitações de cada atributo tabelado.

Cliente = {id_cliente, NIF, genero, telemovel, senha, email, telemovel}
Chave(s) Primária(s) = id_cliente (**NOT NULL, AUTO_INCREMENT**)

Reserva = {id_reserva, custo, data_reserva, classe, NºLugar}
Chave(s) Primária(s) : id_reserva(**UNSIGNED, NOT NULL, AUTO_INCREMENT**)

Viagem = {id_viagem, Preço_Primeira_classe, Partida, Hora Chegada, Preço_economica, Hora Partida, Destino}
Chave(s) Primária(s) = id_viagem(**UNSIGNED, NOT NULL, AUTO_INCREMENT**)

Comboio = { id_comboio, tipo_comboio, Lotação_Economica, Lotação_Primeira_Classe, capacidade, lugares}

Chave(s) Primária(s) = id_comboio(**UNSIGNED, NOT NULL, AUTO_INCREMENT**)

Lugar = { id_lugar, N_lugar, Classe, comboio_id_comboio}

Chave(s) Primária(s) : id_Lugar(**UNSIGNED, NOT NULL, AUTO_INCREMENT**)

Chave (s) Estrangeira(s) : comboio_id_comboio - referência a Comboio (id_comboio) (**UNSIGNED, NOT NULL, ON DELETE NO ACTION, ON UPDATE NO ACTION)**

Concluímos deste modo que:

1 – As restrições de integridade da entidade cumprem-se, visto que nenhuma das chaves primárias do modelo permitem valores nulos;

2 – As restrições de integridade referencial são cumpridas, pois não são permitidas chaves estrangeiras numa tabela sem que a respetiva chave primária exista noutra tabela, e estas mesmas também não poderão ser nulas;

3 – As restrições de integridade de domínio são cumpridas, pois de acordo com os domínios e tipos definidos nos requisitos, estes foram restringidos diretamentre no modelo Lógico;

4 – A multiplicidade manteve-se na conversão do modelo Conceptual para o Lógico.

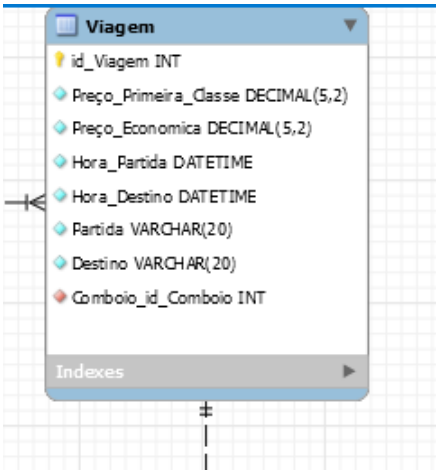
4. Modelo Físico

Chegamos à última etapa deste projeto, onde iremos recorrer a um SGBD, motor de bases de dados que nos permita transformar o projecto no produto final.
Utilizamos o MySQL Workbench novamente, visto este já ter sido utilizado para a realização do modelo lógico.
Tendo escolhido já a nossa ferramenta de trabalho iremos agora avançar na construção das tabelas apresentadas pelo modelo lógico, algumas queries e ainda transações úteis para um bom funcionamento da nossa espresa de comboios.

4.1 Implementação e conversão do Modelo Lógico para um SGBD

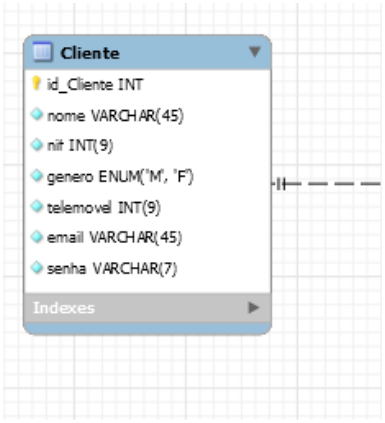
Como já referimos encima iremos recorrer à ferramenta do MySQL Workbench para fazer a conversão do Modelo Lógico para um SGBD.
Esta tarefa não foi muito complexa visto que existe um mecanismo neste ferramenta apelidada de Foward Enginner que, a partir da representação do nosso modelo Lógico, conseguiu gerar o respetivo código SQL que premite que se criem as respetivas tabelas respeitando estas os dados indicados e ainda todas as referências entre tabelas.

4.2 Criação de Relações



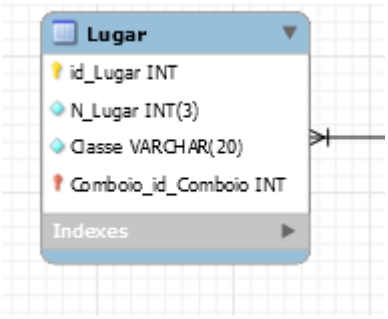

```
-- -----------------------------------------------------
-- Table `comboios_flavienses`.`viagem`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `comboios_flavienses`.`viagem` (
  `id_Viagem` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `Preço_Primeira_Classe` DECIMAL(5,2) NOT NULL,
  `Preço_Economica` DECIMAL(5,2) NOT NULL,
  `Hora_Partida` DATETIME NOT NULL,
  `Hora_Destino` DATETIME NOT NULL,
  `Partida` VARCHAR(20) NOT NULL,
  `Destino` VARCHAR(20) NOT NULL,
  `Comboio_id_Comboio` INT(10) UNSIGNED NOT NULL,
  PRIMARY KEY (`id_Viagem`),
  INDEX `Partido_UNIQUE` (`Partida` ASC),
  INDEX `Destino_UNIQUE` (`Destino` ASC),
  INDEX `fk_Viagem_Comboio1_idx` (`Comboio_id_Comboio` ASC),
  CONSTRAINT `fk_Viagem_Comboio1`
    FOREIGN KEY (`Comboio_id_Comboio`)
    REFERENCES `comboios_flavienses`.`comboio` (`id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 9 – Tabela Viagem



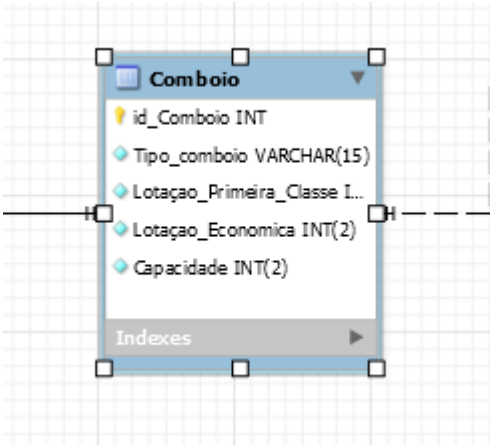

```
-- -----------------------------------------------------
-- Table `comboios_flavienses`.`cliente`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `comboios_flavienses`.`cliente` (
  `id_Cliente` INT(11) NOT NULL AUTO_INCREMENT,
  `nome` VARCHAR(45) NOT NULL,
  `nif` INT(9) UNSIGNED NOT NULL,
  `genero` ENUM('M', 'F') NOT NULL,
  `telemovel` INT(9) NOT NULL,
  `email` VARCHAR(45) NOT NULL,
  `senha` VARCHAR(7) NOT NULL,
  PRIMARY KEY (`id_Cliente`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 10 – Tabela Cliente



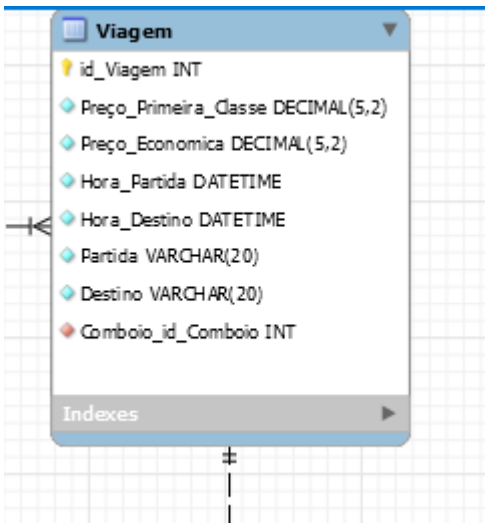

```
-- -----------------------------------------------------
-- Table `comboios_flavienses`.`lugar`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `comboios_flavienses`.`lugar` (
  `id_Lugar` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `N_Lugar` INT(3) UNSIGNED NOT NULL,
  `Classe` VARCHAR(20) NOT NULL,
  `Comboio_id_Comboio` INT(10) UNSIGNED NOT NULL,
  PRIMARY KEY (`id_Lugar`, `Comboio_id_Comboio`),
  INDEX `fk_Lugar_Comboio1_idx` (`Comboio_id_Comboio` ASC),
  CONSTRAINT `fk_Lugar_Comboio1`
    FOREIGN KEY (`Comboio_id_Comboio`)
    REFERENCES `comboios_flavienses`.`comboio` (`id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 11- Tabela Lugar




```
-- -----------------------------------------------------
-- Table `comboios_flavienses`.`comboio`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `comboios_flavienses`.`comboio` (
  `id_Comboio` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `Tipo_comboio` VARCHAR(15) NOT NULL,
  `Lotaçao_Primeira_Classe` INT(2) UNSIGNED NOT NULL,
  `Lotaçao_Economica` INT(2) UNSIGNED NOT NULL,
  `Capacidade` INT(2) UNSIGNED NOT NULL,
  PRIMARY KEY (`id_Comboio`))
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 12 – Tabela Comboio




```
-- -----------------------------------------------------
-- Table `comboios_flavienses`.`viagem`
-- -----------------------------------------------------
CREATE TABLE IF NOT EXISTS `comboios_flavienses`.`viagem` (
  `id_Viagem` INT(10) UNSIGNED NOT NULL AUTO_INCREMENT,
  `Preço_Primeira_Classe` DECIMAL(5,2) NOT NULL,
  `Preço_Economica` DECIMAL(5,2) NOT NULL,
  `Hora_Partida` DATETIME NOT NULL,
  `Hora_Destino` DATETIME NOT NULL,
  `Partida` VARCHAR(20) NOT NULL,
  `Destino` VARCHAR(20) NOT NULL,
  `Comboio_id_Comboio` INT(10) UNSIGNED NOT NULL,
  PRIMARY KEY (`id_Viagem`),
  INDEX `Partido_UNIQUE` (`Partida` ASC),
  INDEX `Destino_UNIQUE` (`Destino` ASC),
  INDEX `fk_Viagem_Comboio1_idx` (`Comboio_id_Comboio` ASC),
  CONSTRAINT `fk_Viagem_Comboio1`
    FOREIGN KEY (`Comboio_id_Comboio`)
    REFERENCES `comboios_flavienses`.`comboio` (`id_Comboio`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
ENGINE = InnoDB
DEFAULT CHARACTER SET = utf8;
```

Figura 13 – Tabela Viagem

5. Conclusões e Trabalho Futuro

Concluído o projeto é possível rever alguns pontos importantes na implementação desta BD como por exemplo, a escolha ou nao das entidades, e os seus derivados (atributos, chaves, relações).
Inserido nas temáticas da Unidade Curricular de Bases de Dados encontrámos algumas dificuldades na comparação de entidades.
O caso foi o bilhete no entanto depois de muito refletir compreendemos que o nosso bilhete foi para o “reserva”.
Outra parte importantíssima que provocou um atraso na realização deste projecto, foi o multivalor lugar.
No início não usariamos o lugar como multivalor mas sim uma entidade, no entanto depois de muito refletir e virmos que não daria certo o esquema lógico, optando então pela opção do multivalorado.
Outro problema mas que reconhecemos que melhoramos foi no uso de queries, procedure, no entanto a nivel de transações ainda podemos melhorar um bocado o uso delas.
Para uma melhor base de dados, concluimos que era melhor constuir um base com imensos dados,e foi isso que fizemos.
Visto que também acontece na vida real, decidimos criar 2 tipos de preço e também para dificultar o trabalho, visto que teriamos que relacionar o custo com os 2 tipos de preço, o que foi interessante para saber mais de Base de Dados.

Terminamos esta análise de forma positiva pois conseguimos melhorar e muito o nosso conhecimento em relação a base de dados, através do uso do MySQL, e foi algo que nos fez melhorar a programar, algo que vai ser a nossa vida no futuro.

6. de Siglas e Acrónimos

BD – Bases de Dados

SGBD – Sistema de Gestão de Base de Dados

SQL – Structured Query Language

7. Anexos

I -Modelo Conceptual

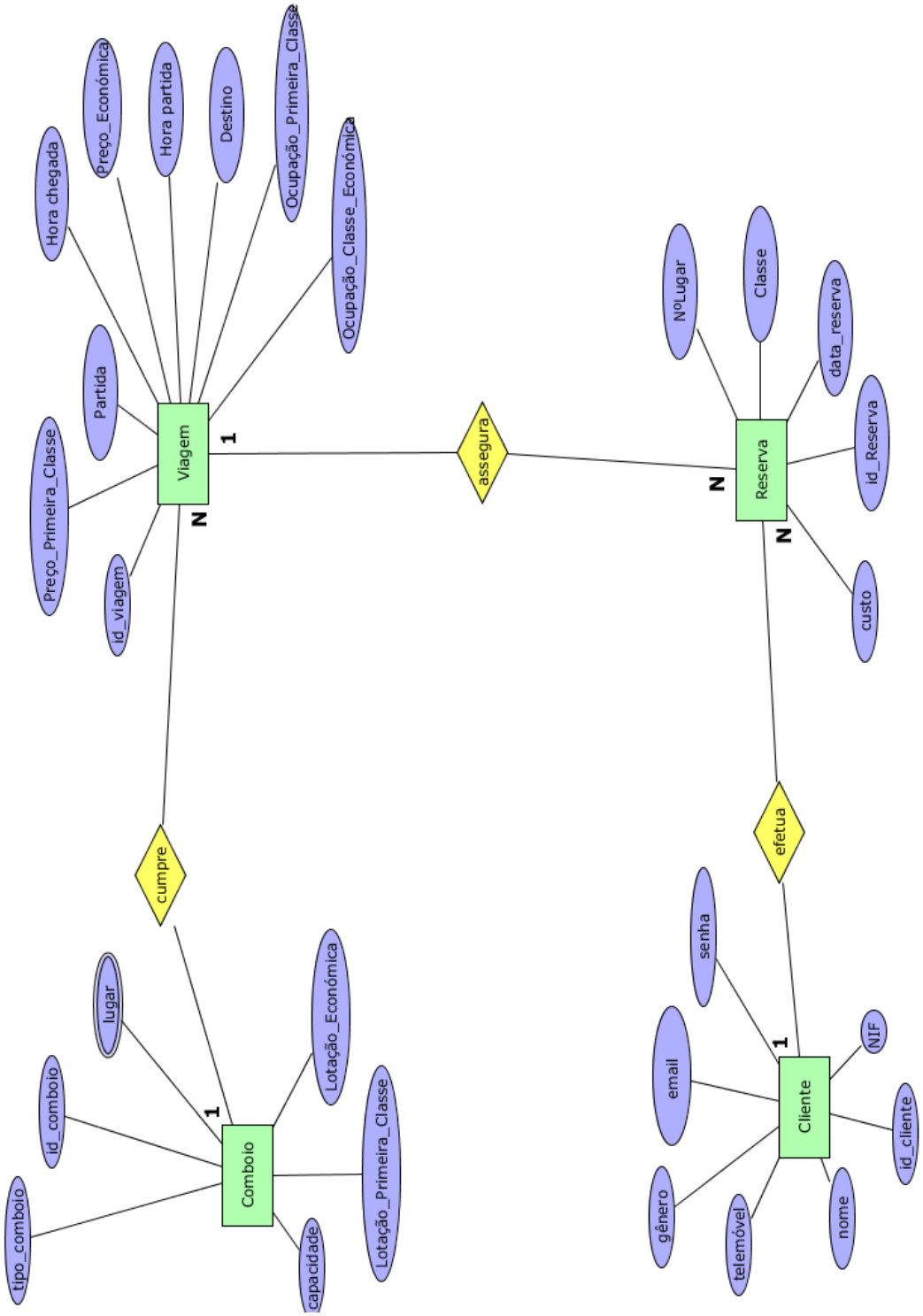


Figura 14 - Modelo Conceptual

II – Modelo Lógico

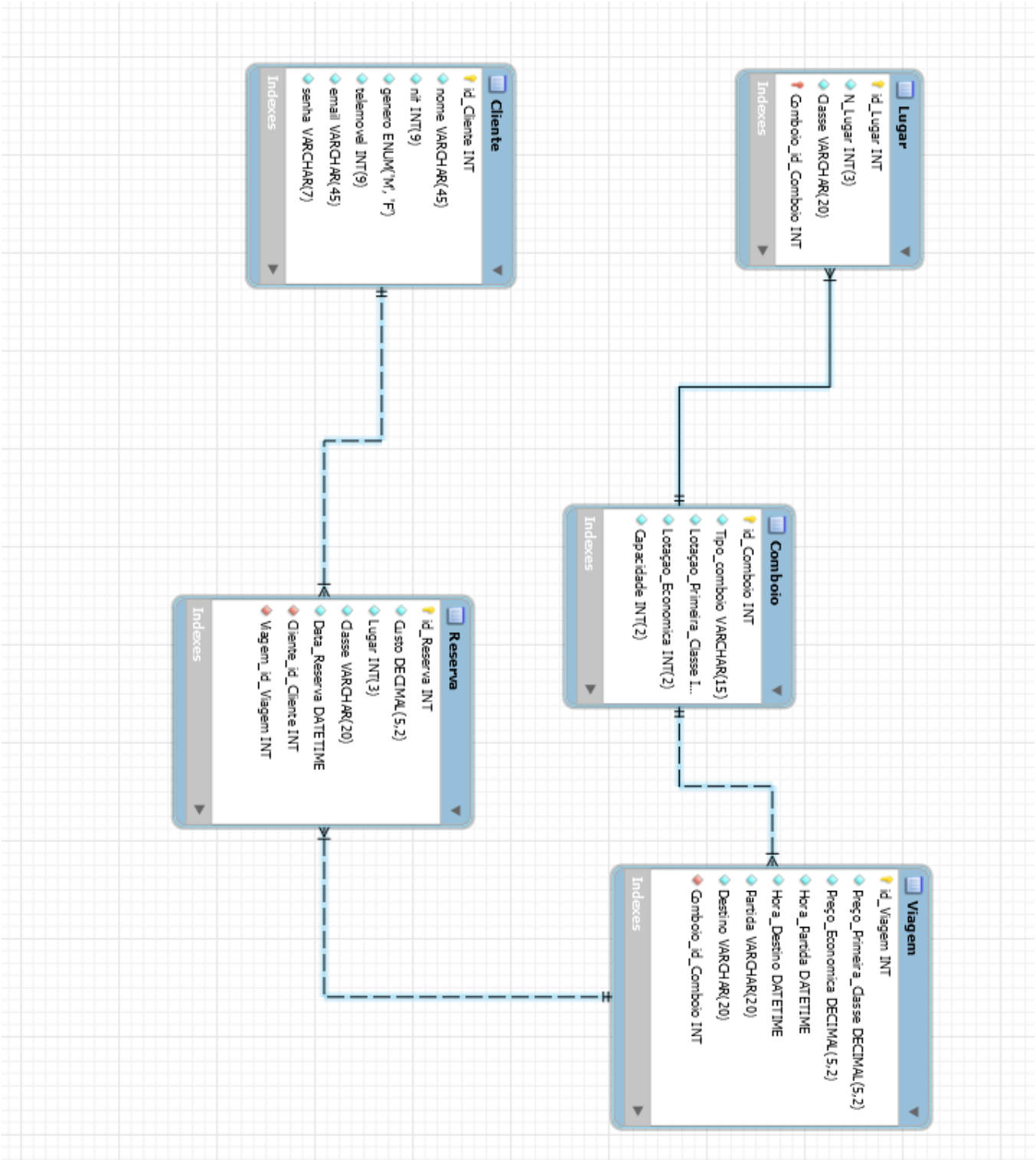


Figura 15 - Modelo Lógico

III – Procedimentos SQL

```sql
-- Query 1 (Quantidade de viagens por cada cliente)

drop view if exists viagens_totais;
create view Viagens_Totais as
    select c.nome, c.nif,  count(*) as Total_Viagens
    from cliente as c
        inner join reserva as r
        on r.Cliente_id_Cliente=c.id_Cliente
    group by c.email;

-- Query 2 (Total gasto por cada cliente)

drop view if exists total_gasto;
create view Total_Gasto as
    select c.nome, c.nif  , Sum(Custo) as Gasto
    from cliente as c
        inner join reserva as r
        on r.Cliente_id_Cliente=c.id_Cliente
    group by c.email;

-- Query 3 (Duraçao de cada viagem)

drop view if exists duração_viagem;
create view Duração_viagem as
    select Partida, Destino, timediff(Hora_Destino,Hora_partida) as Duração
    from viagem
    group by id_Viagem;
```

Figura 16 – Quantidade de viagens por cliente || Total gasto por cada Cliente || Duração de cada Viagem

```sql
-- Query 4 (Facturaçao por cada viagem)

drop view if exists faturaçao_viagem;
create view Faturaçao_Viagem as
    select  Partida, Destino, Sum(Custo) as Faturação
    from viagem as v
        inner join reserva as r
        on v.id_Viagem=r.Viagem_id_Viagem
    group by id_Viagem;

-- Query 5 (Radiografia dos clientes-Homem/Mulher)

drop view if exists clientes_radiografia;
create view Clientes_radiografia as
    select c.genero,  count(*) as Homem_Mulher
    from cliente as c

    group by c.genero;

-- Query 6(Quantas viagens por classe)

drop view if exists cliente_classe;
create view Cliente_Classe as
    select r.Classe,  count(*) as Cliente_Classe1
    from reserva as r
    group by r.Classe;
```

Figura 17 – Receita de cada viagem || Totalidade dos clientes de cada gênero || Quantas Viagens em cada classe

```sql
-- Query 7(Quantas viagens internacionais/nacionais temos)

drop view if exists viagens_n_i;
create view viagens_N_I as
    select Tipo_comboio, count(*) as Total
    from comboio as c
        inner join viagem as v
        where c.id_Comboio=v.Comboio_id_Comboio
    group by Tipo_comboio;

-- Query 8(Quantas viagens de partida por cada estacao)

drop view if exists Cidade;
create view Cidade as
    select Partida  , Count(*) as Total_Estação
    from viagem as v
    group by Partida;

 -- Query 9(Quantas viagem economicas/1ºparte de cada cliente)

drop view if exists quantidade_classes;
create view quantidade_classes as

select c.nome, c.nif, (select count(*) from reserva as r where c.id_Cliente=r.Cliente_id_Cliente and Classe='1ªClasse') as Primeira_classe,
    (select count(*) from reserva as r where c.id_Cliente=r.Cliente_id_Cliente and Classe='Economica') as Economica
from
cliente as c
    group by c.id_Cliente;
```

Figura 18 – quantas viagens Internacionais / Nacionais existem || Quantas Partidas cada estação tem || Quantas viagens económicas / primeira classe de cada cliente

```sql
-- Query 10 (os 5 melhores clientes da empresa)

drop view if exists five_best_clients;
create view five_best_clients as

select c.nome, c.nif  , Sum(Custo) as Money
    from cliente as c
        inner join reserva as r
        on r.Cliente_id_Cliente=c.id_Cliente
    group by c.nif order by Money desc
    limit 5;

-- Query 11 (Quantidade d ereservas por dia)

drop view if exists hora_Reserva;
create view hora_Reserva as
    select date(r.Data_Reserva), count(*) as Numero_Reservas from reserva as r
            group by date(r.Data_Reserva);

-- Query 12 (Quantidade de viagens feitas por comboio)

drop view if exists Comboio_Viagens;
create view Comboio_Viagens as
    select c.id_Comboio, count(*) Numero_Viagens from comboio as c
        inner join viagem as v on c.id_Comboio = v.Comboio_id_Comboio
        group by c.id_Comboio;
```

Figura 19 – os 5 clientes mais gastadores || quantidade de reservas por dia || quantidade de viagens feitas por cada comboio

```
-- Procedure 1 (Todos dos clientes)

DELIMITER $$
drop procedure if exists GetA_n_Clientes $$
create procedure Get_n_Clientes(in quantidade int)
  begin
       select * from cliente
       limit quantidade;
  end $$
DELIMITER ;

-- Procedure 2 ( Dado uma partida, mostras os destinos)

DELIMITER $$
drop procedure if exists dest_Part $$
create procedure dest_Part(in partida varchar(25))
  begin
       if (not exists(select v.Partida from viagem as v where (v.Partida=partida)))
            then select ' Partida nao existe' as msg1;
       else
           select v.id_Viagem, v.Hora_Partida, v.Partida, v.Destino from viagem as v
           where v.Partida=partida
           group by v.id_Viagem;
       end if;
  end $$
DELIMITER ;
```

Figura 20 – devolve uma lista de todos os clients || Mostra os destinos que dado local de partida alcança

```
DELIMITER $$
 drop procedure if exists date_Viag $$
 create procedure date_Viag(in date1 date)
   begin
        if (not exists(select v.Hora_Partida from viagem as v where (date(v.Hora_Partida)=date1)))
             then select 'Nao ha viagens nesse dia' as msg1;
        else
            select date1, v.Partida, v.Destino from viagem as v
            where (date(v.Hora_Partida)=date1)
            group by v.id_Viagem;
        end if;
   end $$
 DELIMITER ;

-- Procedure 4 (Viagem por cliente)

DELIMITER $$
drop procedure if exists Viagem_por_Cliente $$
create procedure Viagem_por_Cliente(in cliente varchar(45))
    begin
            if (not exists(select c.nif, c.email from cliente as c where (c.nome=cliente)))
                then select ' Cliente nao existe' as msg1;
            else
                select c.nome, c.email, v.Partida, v.Destino from cliente as c
                inner join reserva as r on c.id_Cliente=r.Cliente_id_Cliente and cliente=c.nome
                inner join viagem as v on v.id_Viagem = r.Viagem_id_Viagem
                group by r.id_Reserva;
            end if;
    end $$
DELIMITER ;
```

Figura 21 - Mostra as viagens de dado dia || mostra as viagens por cliente

```
-- Procedure 5 (Calcular o valor gasto de um certo cliente)

DELIMITER $$
drop procedure if exists total_gasto_Cliente $$
create procedure total_gasto_Cliente(in cliente varchar(45))
    begin
        if (not exists(select c.nif, c.email from cliente as c where (c.nome=cliente)))
            then select ' Cliente nao existe' as msg1;
        else
            select c.nome, c.nif  , Sum(Custo) as Gasto
            from cliente as c
            inner join reserva as r
            on r.Cliente_id_Cliente=c.id_Cliente and cliente=c.nome
            group by c.email;
        end if;
    end $$
DELIMITER ;

-- Procedure 6 (Quantas reservas foram feitas num certo dia)

DELIMITER $$
 drop procedure if exists date_Reserv $$
 create procedure date_Reserv(in date1 date)
   begin
        if (not exists(select r.Data_reserva from reserva as r where (date(r.Data_reserva)=date1)))
                then select 'Nao ha reservas nesse dia' as msg1;
        else
            select date1, count(*) as Numero_Reservas from reserva as r
            where (date(r.Data_Reserva)=date1);
        end if;
   end $$
 DELIMITER ;
```

Figura 23 – valor gasto por um certo cliente || quantas reservas foram feitas num dado dia

```
-- Procedure 7 (Dado uma viagem e um lugar, diz-me que se senta nesse lugar)

DELIMITER $$
 drop procedure if exists lugar_Cliente $$
 create procedure lugar_Cliente(in idViagem int, in assento int)
   begin
        if (not exists(select r.Viagem_id_Viagem from reserva as r where (r.Viagem_id_Viagem=idViagem)))
                then select 'Viagem incorreta' as msg1;
        else
            select c.nome, c.email, r.Cliente_id_Cliente as ID_CLIENTE, r.lugar from reserva as r
                inner join cliente as c on r.Cliente_id_Cliente=c.id_Cliente
            where idViagem=r.Viagem_id_Viagem and assento=lugar;
        end if;
   end $$
 DELIMITER ;

 -- Procedure 8 (Factiraço num determinado mes)

 DELIMITER $$
 drop procedure if exists mes_Fact $$
 create procedure mes_Fact(in date1 date)
   begin
        if (not exists(select r.Data_Reserva from reserva as r where (date(r.Data_Reserva)=date1)))
                then select 'Nao houve faturaçao nesse dia' as msg1;
        else
            select date1 as Data_Pedida, Sum(Custo) from reserva as Facturaçao
            where (date(Data_reserva)=date1);
        end if;
   end $$
 DELIMITER ;
```

Figura 24 – quem ocupa dado lugar || faturação do mês

```
-- Transaçao 1 (Insere Cliente)
DELIMITER $$
drop procedure if exists insere_Cliente $$
create procedure insere_Cliente(in nome1 varchar(45), in nif1 int(9), in genero1 enum('M','F'), in telemovel1 int(9),
                in email1 varchar(45), in senha1 varchar(7))

    begin

    declare Erro bool default 0;
    declare continue handler for sqlexception set Erro = 1;

        start transaction;

        select @id_Cliente := max(id_Cliente)
        from cliente;

        set @id_Cliente = @id_CSiente  + 1;

        insert into cliente(id_Cliente, nome, nif, genero, telemovel, email, senha)
        values(@id_Cliente, nome1, nif1, genero1, telemovel1, email1, senha1);


        if Erro then
            select 'Erro no registo' as Msg;
            rollback;
        else
            select 'Registo efetuado' as Msg;
            commit;
        end if;

    end $$
DELIMITER ;
```

Figura 22 – insere um cliente

```
-- Transaçao 2 (Insere Comboio)
DELIMITER $$
drop procedure if exists insere_Comboio $$
create procedure insere_Comboio(in Tipo_Comboio1 varchar(20), in Lotaçao_Primeira_Classe1 int, in Lotaçao_Economica1 int, in Capacidade1 int)
    begin

        declare Erro bool default 0;
        declare continue handler for sqlexception set Erro = 1;

        start transaction;

        select @id_Comboio := max(id_Comboio)
        from comboio;

        set @id_Comboio = @id_Comboio  + 1;

        insert into comboio
        (id_Comboio, Tipo_comboio, Lotaçao_Primeira_Classe, Lotaçao_Economica , Capacidade)
        values
        (@id_Comboio, Tipo_comboio1, Lotaçao_Primeira_Classe1, Lotaçao_Economica1, Capacidade1);
        select 'Registo efetuado' as Msg;

        if Erro then
            select 'Erro no resgisto' as Msg;
            rollback;
        else
            select 'Registo efetuado' as Msg;
            commit;
        end if;

    end $$
DELIMITER ;
```

Figura 23 – insere um comboio

```
-- Transaçao 3(Update Cliente)

DELIMITER $$
drop procedure if exists update_Cliente $$
create procedure update_Cliente (in id_Cliente1 int, in telemovel1 int(9))
    begin

        declare Erro bool default 0;
        declare continue handler for sqlexception set Erro = 1;

        start transaction;

        update cliente
            set telemovel = telemovel1
            where id_Cliente = id_Cliente1;
            select 'Upate do numero efetuado com sucesso' as Msg;

        if Erro then
            select 'Erro no resgisto' as Msg;
            rollback;
        else
            select 'Update efetuado' as Msg;
            commit;
        end if;

    end $$
DELIMITER ;
```

Figura 24 – update de um cliente

```
-- Transaçao 4(Update Comboio)
DELIMITER $$
drop procedure if exists update_Comboio $$
create procedure update_Comboio (in id_Comboio1 int, in Tipo_Comboio1 varchar(20))
        begin

            declare Erro bool default 0;
            declare continue handler for sqlexception set Erro = 1;

            start transaction;

                update comboio
                set  Tipo_Comboio = Tipo_Comboio1
                where id_Comboio = id_Comboio1;

        if Erro then
            select 'Erro no resgisto' as Msg;
            rollback;
        else
            select 'Upate do comboio efetuado com sucesso' as Msg;
            commit;
        end if;

        end $$
```

Figura 25 – update de um comboio