



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Bases de Dados

Ano Letivo de 2022/2023

Rede Metropolitana

João Machado a89510 Lucas Oliveira a98695 Mike Pinto a89292 Rafael Gomes a96208

17 de fevereiro de 2025

BD

Data de Recepção	
Responsável	
Avaliação	
Observações	

Rede Metropolitana

João Machado a89510 Lucas Oliveira a98695 Mike Pinto a89292
Rafael Gomes a96208

17 de fevereiro de 2025

Resumo

Este relatório foi realizado no âmbito da Unidade Curricular de Bases de Dados, que tinha como objetivo principal desenvolver um sistema de base de dados.

O objetivo deste sistema é implementar uma base de dados para uma empresa de metro. Após uma primeira discussão foi decidido dividir o processo da implementação do sistema em fases.

Em primeiro, definimos os requisitos que a nossa base de dados deveria ser capaz de suportar. Após este processo foi feita a modelação conceptual. Em último, realizamos uma validação do modelo conceptual antes de avançarmos.

Com a validação, construímos o modelo lógico conceptual, realizamos a sua análise e validação através da normalização e das interrogações do utilizador.

Por fim, com base no modelo lógico foi gerado um modelo físico no **MySQL Workbench** cumprindo as modelações para a criação de uma base de dados segura e correta.

Dada a implementação física e, esta ser validada pelo cliente, deu-se o projeto de desenvolvimento da base de dados como terminado.

Área de Aplicação: Desenho, Arquitetura, Desenvolvimento e Implementação de Sistemas de Bases de Dados.

Palavras-Chave: Bases de Dados, Bases de Dados Relacionais, Análise de Requisitos, Entidades, Atributos, Relações, ,Modelo Conceptual, Modelo Lógico, Normalização, Vistas de Utilização, MySQL Workbench, SQL.

Índice

1	Definição do Sistema	1
1.1	Contexto de aplicação e Fundamentação do sistema	1
1.2	Motivação e Objetivos do Trabalho	1
1.3	Análise da viabilidade do processo	2
1.4	Recursos e Equipa de Trabalho	2
1.4.1	Recursos	2
1.4.2	Equipa de trabalho	3
1.5	Plano de Execução do Projeto	4
2	Levantamento e Análise de Requisitos	5
2.1	Método de levantamento e de análise de requisitos adotado	5
2.2	Organização dos requisitos levantados	6
2.2.1	Requisitos de Descrição	6
2.2.2	Requisitos de Manipulação	8
2.2.3	Requisitos de Controlo	9
2.3	Análise e validação geral dos requisitos	10
3	Modelação Conceptual	11
3.1	Apresentação e abordagem de modelação realizada	11
3.2	Identificação e caracterização das entidades	11
3.2.1	Entidade Viagem	11
3.2.2	Entidade Cliente	11
3.2.3	Entidade Funcionário	12
3.2.4	Entidade Bilhete	13
3.2.5	Entidade Estação	13
3.2.6	Entidade Veículo	14
3.3	Identificação e caracterização dos relacionamentos	14
3.3.1	Relacionamento Bilhete-Viagem	14
3.3.2	Relacionamento Veículo-Viagem	15
3.3.3	Relacionamento Cliente-Viagem	16
3.3.4	Relacionamento Funcionário-Bilhete	17
3.3.5	Relacionamento Funcionário-Estação	17
3.3.6	Relacionamento Funcionário-Veículo	18
3.3.7	Relacionamento Estação-Veículo	19
3.3.8	Relacionamento Cliente-Bilhete	20
3.3.9	Relacionamentos Estação-Viagem	21

3.4	Identificação e caracterização da associação dos atributos com as entidades e relacionamentos	22
3.5	Apresentação do diagrama ER produzido	23
4	Modelação Lógica	24
4.1	Construção e validação do modelo de dados lógico	24
4.2	Normalização de Dados	30
4.3	Apresentação e explicação do modelo lógico produzido	30
4.3.1	Conversão de relacionamentos: Um para Muitos	30
4.3.2	Conversão de relacionamentos: Muitos para Muitos	35
4.3.3	Modelo lógico completo	39
4.4	Validação do modelo com interrogações do utilizador	39
5	Implementação Física	40
5.1	Tradução do esquema lógico para o sistema de gestão de bases de dados escondido em SQL	40
5.1.1	Implementação da tabela: Bilhete	41
5.1.2	Implementação da tabela: Cliente	42
5.1.3	Implementação da tabela: ClienteContacto	42
5.1.4	Implementação da tabela: ClienteRealizaViagem	43
5.1.5	Implementação da tabela: Estação	44
5.1.6	Implementação da tabela: EstaçãoVeiculo	45
5.1.7	Implementação da tabela: Funcionario	46
5.1.8	Implementação da tabela: FuncionarioContacto	47
5.1.9	Implementação da tabela: FuncionarioVeiculo	48
5.1.10	Implementação da tabela: Veiculo	49
5.1.11	Implementação da tabela: Viagem	50
5.1.12	Implementação da tabela: ViagemVeiculo	51
5.2	Tradução das interrogações do utilizador para SQL (alguns exemplos)	52
5.2.1	Exemplos	52
5.3	Definição e caracterização das vistas de utilização em SQL (alguns exemplos)	53
5.4	Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)	55
5.5	Indexação do Sistema de Dados	59
5.6	Procedimentos Implementados(alguns exemplos)	60
5.7	Definição e caracterização de mecanismos de segurança e controlo	62
5.8	Plano de segurança e recuperação de dados	65
6	Implementação do Sistema de Recolha de Dados	68
6.1	Apresentação e modelo do sistema	68
6.2	Implementação do sistema de recolha	68
6.3	Funcionamento do sistema	73
7	Implementação do Sistema de Painéis de Análise	74
7.1	Definição e caracterização da vista de dados para análise	74
7.2	Povoamento das estruturas de dados para análise	74

7.3 Apresentação e caracterização dos dashboards implementados	75
8 Conclusões e trabalho futuro	77
Lista de Siglas e Acrónimos	79
Anexos	80
Anexo 1 - Script de criação da base de dados	80
Anexo 2 - Script parcial de povoamento	86
Anexo 3 - Queries em SQL	102
Anexo 4 - Funções usadas em SQL	116
Anexo 5 - Script de Backup	121
Anexo 6 - Script de criação do database	123

Lista de Figuras

1.1	Diagrama de Gantt	4
3.1	Relacionamento das entidades Bilhete-Viagem	14
3.2	Relacionamento das entidades Veículo-Viagem	15
3.3	Relacionamento das entidades Cliente-Viagem	16
3.4	Relacionamento das entidades Funcionário-Bilhete	17
3.5	Relacionamento das entidades Funcionário-Estação	17
3.6	Relacionamento das entidades Funcionário-Veículo	18
3.7	Relacionamento das entidades Estação-Veículo	19
3.8	Relacionamento das entidades Cliente-Bilhete	20
3.9	Relacionamento das entidades Estação-Viagem	21
3.10	Modelo Conceptual	23
4.1	Entidade lógica - Funcionário	24
4.2	Entidade lógica - Estação	26
4.3	Entidade lógica - Cliente	27
4.4	Entidade lógica - Bilhete	28
4.5	Entidade lógica - Viagem	29
4.6	Entidade lógica - Veículo	29
4.7	Relação Cliente - Bilhete	31
4.8	Relação Funcionário - Bilhete	31
4.9	Relação Viagem - Bilhete	32
4.10	Relação Funcionário - Estação	33
4.11	Relação Viagem - Estação	34
4.12	Relação Cliente - Viagem	35
4.13	Relação Funcionário - Veículo	36
4.14	Relação Viagem - Veículo	37
4.15	Relação Estação - Veículo	38
4.16	Modelo Lógico	39
5.1	Implementação da tabela: Bilhete	41
5.2	Implementação da tabela: Cliente	42
5.3	Implementação da tabela: ClienteContacto	42
5.4	Implementação da tabela: ClienteRealizaViagem	43
5.5	Implementação da tabela: Estação	44
5.6	Implementação da tabela: EstaçãoVeiculo	45
5.7	Implementação da tabela: Funcionario	46

5.8	Implementação da tabela: FuncionarioContacto	47
5.9	Implementação da tabela: FuncionarioVeiculo	48
5.10	Implementação da tabela: Veiculo	49
5.11	Implementação da tabela: Viagem	50
5.12	Implementação da tabela: ViagemVeiculo	51
5.13	Código da query 6	52
5.14	Código da query 7	53
5.15	Código da query 8	53
5.16	Vista <i>VerBilhetes</i> e <i>VerPassagemVeiculo</i>	54
5.17	Vista <i>VerCliente</i> e <i>VerClienteContacto</i>	54
5.18	Vista <i>VerEstações</i>	54
5.19	Vista <i>VerFuncionarios</i> e <i>VerFuncionariosContacto</i>	55
5.20	Vista <i>VerVeiculo</i> e <i>VerVeiculoManutencao</i>	55
5.21	Criação de Indexação do sistema	60
5.22	Procedimento de adição de um cliente no sistema.	60
5.23	Procedimento para verificar qual o horário que passam certos veiculos	61
5.24	Procedimento para visualizar a informação de uma Viagem específica.	61
5.25	Procedimento para registar informação de manutenção de veículos.	61
5.26	Criação de grupos de privilégios e privilégios de Administrador	62
5.27	Privilégios do grupo <i>PostoVenda</i>	63
5.28	Privilégios do grupo <i>RespEstacao</i>	64
5.29	Privilégios do grupo <i>TecnicoManutencao</i>	64
5.30	Privilégios do grupo <i>Cliente</i>	64
5.31	Criação dos utilizadores	65
5.32	Script em <i>python</i> do Backup	66
5.33	Script em <i>python</i> da execução do Backup	67
6.1	Ligação ao servidor da Base de Dados	68
6.2	Leitura do ficheiro "Estações.csv" e implementação da mesma na Base de Dados	69
6.3	Leitura do ficheiro "Clientes.csv" e implementação da mesma na Base de Dados	69
6.4	Leitura do ficheiro "ContactosClientes.csv" e implementação da mesma na Base de Dados	70
6.5	Leitura do ficheiro "Funcionarios.csv" e implementação da mesma na Base de Dados	70
6.6	Leitura do ficheiro "ContactosFuncionarios.csv" e implementação da mesma na Base de Dados	70
6.7	Leitura do ficheiro "Veiculos.csv" e implementação da mesma na Base de Dados	71
6.8	Leitura do ficheiro "Bilhetes.csv" e implementação da mesma na Base de Dados	71
6.9	Leitura do ficheiro "Viagens.csv" e implementação da mesma na Base de Dados	72
6.10	Leitura do ficheiro "ViagemVeiculo.csv" e implementação da mesma na Base de Dados	72
6.11	Leitura do ficheiro "FuncionariosVeiculo.csv" e implementação da mesma na Base de Dados	73
6.12	Leitura do ficheiro "EstaçõesVeiculo.csv" e implementação da mesma na Base de Dados	73

7.1	Total auferido ao longo dos anos	75
7.2	Contagem de veículos por estação	75
7.3	Estações - Mapa	76
7.4	Percentagem de salários	76

Lista de Tabelas

2.2	Requisitos de Manipulação	8
2.3	Requisitos de Controlo	9
3.1	Tabela de caracterização do modelo	22
5.1	Espaço ocupado no disco por cada tipo de dados[2]	56
5.2	Espaço ocupado em disco por Cliente	56
5.3	Espaço ocupado em disco por Funcionario	56
5.4	Espaço ocupado em disco por Bilhete	57
5.5	Espaço ocupado em disco por Viagem	57
5.6	Espaço ocupado em disco por Estação	57
5.7	Espaço ocupado em disco por Veículo	57
5.8	Espaço ocupado em disco por EstaçãoVeículo	57
5.9	Espaço ocupado em disco por FuncionarioVeiculo	58
5.10	Espaço ocupado em disco por ViagemVeiculo	58
5.11	Espaço ocupado em disco por ClienteRealizaViagem	58
5.12	Espaço ocupado em disco por ClienteContacto	58
5.13	Espaço ocupado em disco por FuncionarioContacto	58
5.14	Espaço ocupado em disco pela base de dados	59

1 Definição do Sistema

1.1 Contexto de aplicação e Fundamentação do sistema

Um elemento fulcral numa cidade é um sistema de transporte rápido e eficiente disponível para toda a população. O Senhor Abílio Inácio, explorador ávido da sua cidade, utilizava o sistema metropolitano frequentemente. Para ele o metro era a maneira mais rápida e conveniente de deslocar-se. No entanto, ele sempre teve dificuldades em orientar-se no metro devido a possuir um sistema antiquado, onde as informações sobre horários e rotas eram exibidas apenas em mapas e folhetos de papel. Ele achava difícil manusear esses mapas e muitas vezes demorava muito tempo a tentar descobrir a rota mais eficiente para o seu destino.

Para solucionar este problema, o Senhor Abílio decidiu usar todas as suas poupanças e comprar a companhia de metro denominada por "Belo Metro", visando melhorar a experiência dos utilizadores.

Após muitos meses de planeamento, Abílio Inácio, muito experiente em viagens de metro, decidiu que iria necessitar de uma base de dados, para dar resposta aos diversos problemas encontrados no sistema de metro, ao qual iria incluir informações atualizadas sobre estações, veículos, viagens e outras informações úteis para os passageiros.

1.2 Motivação e Objetivos do Trabalho

A principal motivação para a criação do sistema de base de dados foi o facto da companhia de metro possuir um sistema antiquado que levava os utilizadores a não optar por este meio de transporte. Tendo como consequência uma quebra de rendimentos e insatisfação dos clientes.

Os principais objetivos que o Senhor Abílio Inácio pretendia alcançar com a implementação do sistema de base de dados seria:

- Melhorar a gestão do vasto sistema de metro.
- Aumentar a satisfação por parte dos utilizadores.

- Facilitar o planeamento de viagens dos utilizadores de forma mais eficiente.
- Melhorar o acesso dos utilizadores às diversas informações do sistema de metro.

1.3 Análise da viabilidade do processo

Ao substituir o sistema antigo utilizado pela companhia de metro o Senhor Abílio Inácio acredita conseguir:

- Aumentar os rendimentos comparativamente ao antigo funcionamento do metro.
- Tomar conhecimento das linhas e rotas mais frequentadas pelos utilizadores, de modo a otimizar a gestão e manutenção de todo a rede.
- Obter uma locomoção mais rápida e organizada para os utilizadores.
- Facilitar a monitorização e tratamento de dados de todos os clientes, funcionários, bilhetes, entre outros...
- Aumentar a qualidade dos serviços prestados.
- Com a implementação de “queries” personalizadas toda a manutenção e gestão do sistema de metro seria facilitada.

1.4 Recursos e Equipa de Trabalho

Para além de todos os funcionários do metropolitano, o Senhor Abílio Inácio contratou uma empresa especializada no desenvolvimento de sistemas de bases de dados.

Informado pela empresa contratada, foi adquirido um conjunto de servidores para hospedarem todo o sistema e um conjunto de postos de vendas para substituir a atual de bilheteira.

1.4.1 Recursos

Os recursos disponíveis para a realização deste projeto podem ser divididos em duas categorias, recursos humanos e materiais, como:

- **Recursos humanos:**
 - Funcionários da empresa metropolitana
 - Principais clientes

- Um analista
- Funcionários destacados de uma empresa de desenvolvimento de base de dados.

- **Recursos Materiais:**

- Um conjunto de servidores instalados nas principais estações.
- Postos de vendas instalados em todas as estações.
- Veículos de metro equipados para condução autónoma.

1.4.2 Equipa de trabalho

Foi então criada uma equipa de trabalho para o desenvolvimento deste projeto:

- **Pessoal Interno:**

- Abílio Inácio.
- Miguel Teixeira, braço direito de Abílio Inácio e analista.
- Um funcionário de um posto de venda.
- Um responsável por uma estação do metro.
- Um técnico de manutenção.

Funções: Administração do metro, manutenção, atendimento ao cliente e validação de serviço.

- **Pessoal externo:**

- Engenheiros especializados na criação e manutenção de bases de dados destacados de uma empresa contratada.

Funções: Levantamento de requisitos, modelação e implementação do sistema de base de dados e manutenção do sistema.

- **Outros:**

- Principais utilizadores do sistema de metro.

Funções: Inquéritos de opinião, validação de serviços.

1.5 Plano de Execução do Projeto

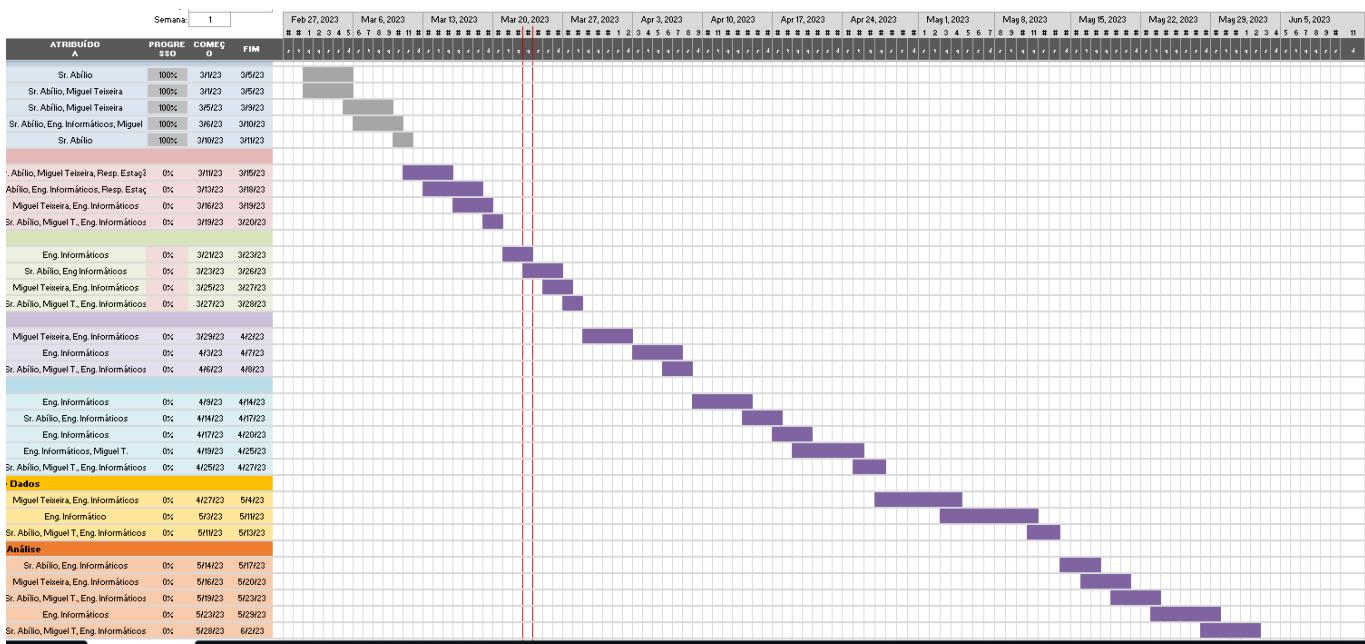


Figura 1.1: Diagrama de Gantt

2 Levantamento e Análise de Requisitos

2.1 Método de levantamento e de análise de requisitos adotado

A abordagem adotada nesta fase, de levantamento de requisitos, foi a recolha de informações de várias fontes credíveis, passando por:

- **Inquéritos:** Para além das frustrações de Abílio Inácio, foi realizado inquéritos aos utilizadores e funcionários da rede metropolitana com o intuito de desenvolver uma plataforma mais próxima das suas necessidades.
- **Análise da concorrência:** Durante vários dias, foram analisadas várias empresas de transportes concorrentes, observando as suas ofertas e serviços prestados, e quais as suas vantagens e desvantagens.
- **Clientes falsos:** Foram contratados clientes falsos para analisar a sua primeira interação com o sistema atual.
- **Observação:** Foi também observado como vários utilizadores interagiam com o sistema atual, antiquado, do sistema de metro, levantando as suas principais dificuldades e quais melhorias podiam ser realizadas.

2.2 Organização dos requisitos levantados

2.2.1 Requisitos de Descrição

Tabela 2.1 – continuação da página anterior

Nr. Requisito	Data	Descrição	Fonte	Revisor
RD1	13/03/2023	Os clientes devem ser registados no sistema no momento da compra do seu primeiro bilhete.	Funcionário Postos Venda.	Miguel Teixeira
RD2	13/03/2023	Um funcionário deve ser registado no sistema no momento em que é contratado, sendo necessário inserir o seu Contribuinte , usado como identificador do funcionário no sistema, nome completo, data de nascimento , qual o seu género , quais os seus contactos (pode conter mais que um contacto), o seu email , qual o seu cargo , qual o seu salário, IBAN e a sua morada (composto por rua, localidade, código-postal).	Abílio Inácio	Abílio Inácio
RD3	13/03/2023	No momento de registo de um cliente , o funcionário do posto de venda deve inserir o Contribuinte do cliente, usado para identificação de um cliente no sistema, o seu nome completo, data de nascimento , o seu género , quais os seus contactos (pode conter mais que um contacto), o seu email e a sua morada (composto por Rua, Localidade, código-postal).	Responsável de uma estação	Abílio Inácio
RD4	13/03/2023	No sistema o género é designado por um número: <ul style="list-style-type: none"> ▪ 0 - Masculino ▪ 1 - Feminino 	Responsável de uma estação	Miguel Teixeira
RD5	13/03/2023	No momento da compra de um bilhete deve ser inserido no sistema qual a forma de pagamento , qual o tipo de bilhete , qual a data de venda, data de validade e o preço do bilhete , sendo atribuído pelo sistema um número de venda único e sucessivo para identificar a venda no sistema.	Funcionário Posto de venda	Abílio Inácio

Continuação na próxima página

Tabela 2.1 – continuação da página anterior

Nr. Requisito	Data	Descrição	Fonte	Revisor
RD6	13/03/2023	<p>Podem ser vendidos vários tipos de bilhetes, designados no sistema pelo número:</p> <ul style="list-style-type: none"> ▪ 0 - Bilhete Diário; ▪ 1 - Bilhete Mensal; 	Miguel Teixeira	Miguel Teixeira
RD7	13/03/2023	<p>Todas as viagens realizadas por um cliente devem ser registadas no sistema com um identificador, único e sucessivo, atribuído automaticamente, a data da sua realização e quais a estações de origem e destino</p>	Miguel Teixeira	Abílio Inácio
RD8	13/03/2023	<p>Todas as estações devem ser registadas no sistema, possuindo um identificador único e sucessivo dado automaticamente pelo sistema, o seu nome, a sua localização composta por Latitude e Longitude, Pontos de interesse e descrição.</p>	Miguel Teixeira	Abílio Inácio
RD9	13/03/2023	<p>No momento de aquisição de um veículo deve ser registado o seu identificador, o número total de lugares e o número de carruagens</p>	Técnico de Manutenção	Miguel Teixeira
RD10	13/03/2023	<p>Numa estaçao trabalham vários funcionários, mas cada funcionario é designado para trabalhar, por norma, apenas numa estaçao.</p>	Abílio Inácio	Abílio Inácio
RD11	13/03/2023	<p>Existem vários cargos de funcionários designados no sistema por um número:</p> <ul style="list-style-type: none"> ▪ 0 - Funcionários de Postos de venda. ▪ 1 - Responsáveis de estações. ▪ 2 - Técnicos de manutenção. ▪ 3 - Administradores, como Miguel Teixeira e Abílio Inácio. 	Miguel Teixeira	Abílio Inácio
RD12	13/03/2023	<p>Um bilhete está associado apenas a um cliente, apesar de um cliente poder ter vários bilhetes.</p>	Responsável de uma estação	Abílio Inácio
RD13	13/03/2023	<p>Um cliente pode possuir um ou mais bilhetes, mas pode chegar a não realizar uma viagem.</p>	Responsável de uma estação	Miguel Teixeira

Continuação na próxima página

Tabela 2.1 – continuação da página anterior

Nr. Requisito	Data	Descrição	Fonte	Revisor
RD14	13/03/2023	Um técnico da manutenção pode realizar manutenção a um ou mais veículos e um veículo pode sofrer manutenção de vários técnicos de manutenção diferentes.	Técnico da Manutenção	Miguel Teixeira
RD15	13/03/2023	Todas as viagens precisam de possuir uma estaçao de origem e destino, assim como as horas da sua realização, apesar de uma estação poder ser origem e destino de várias viagens.	Responsável de uma estação	Miguel Teixeira
RD16	13/03/2023	Um bilhete está associado apenas a um funcionário , por norma ao funcionário de postos de venda, que o vendeu.	Miguel Teixeira	Abílio Inácio
RD17	13/03/2023	Uma viagem pode ser realizada por vários veículos , assim como um veículo pode realizar várias viagens .	Responsável de uma estação	Miguel Teixeira
RD18	13/03/2023	Um veículo tem de passar obrigatoriamente em uma ou mais estações , assim como numa estaçao passam um ou mais veículos , registando a hora da sua passagem.	Técnico da Manutenção	Abílio Inácio
RD19	13/03/2023	No sistema as formas de pagamento são representadas por um valor inteiro, designados por: <ul style="list-style-type: none">▪ 0 - Dinheiro▪ 1 - Multibanco	Funcionário Posto Venda	Abílio Inácio

2.2.2 Requisitos de Manipulação

Tabela 2.2: Requisitos de Manipulação

Nr. Requisito	Data	Descrição	Fonte	Revisor
RM1	13/03/2023	O sistema deverá contabilizar o número de bilhetes vendidos num dia.	Abílio Inácio	Abílio Inácio
RM2	13/03/2023	O sistema deverá poder aceder à informação do cliente ou funcionário através do seu Contribuinte.	Funcionário posto de vendas	Miguel Teixeira
RM3	13/03/2023	O sistema deverá identificar quais os clientes que realizaram mais viagens num dia.	Abílio Inácio	Abílio Inácio

Continuação na próxima página

Tabela 2.2 – continuação da página anterior

Nr. Requisito	Data	Descrição	Fonte	Revisor
RM4	13/03/2023	O sistema deve contabilizar a estação onde passam mais veículos.	Funcionário posto de vendas	Miguel Teixeira
RM5	13/03/2023	O sistema deverá calcular o total auferido num dia.	Abílio Inácio	Abílio Inácio
RM6	13/03/2023	O sistema deverá saber qual o tipo de bilhete que um cliente possui introduzindo o seu Contribuinte.	Responsável de uma estação	Miguel Teixeira
RM7	14/03/2023	O sistema deverá identificar quais os gastos totais de um cliente.	Miguel Teixeira	
RM8	14/03/2023	O sistema deverá identificar quem vendeu e para qual cliente um bilhete foi vendido através do número de venda do bilhete.	Responsável de uma estação	Miguel Teixeira
RM9	14/03/2023	O sistema deverá saber quais funcionários trabalham numa certa estação e quais os seus cargos.	Miguel Teixeira	Abílio Inácio
RM10	14/03/2023	O sistema deverá saber qual foi o cliente que realizou uma viagem e qual a origem e destino do cliente introduzindo o ID da viagem.	Responsável de uma estação	Miguel Teixeira
RM11	14/03/2023	O sistema deverá calcular a média do valor auferido entre datas.	Responsável de uma estação	Miguel Teixeira

2.2.3 Requisitos de Controlo

Tabela 2.3: Requisitos de Controlo

Nr. Requisito	Data	Descrição	Fonte	Revisor
RC1	15/03/2023	O Senhor Abílio Inácio e o Miguel Teixeira têm acesso total ao sistema, podendo realizar qualquer tipo de alteração/remoção/adição.	Abílio Inácio	Abílio Inácio

Continuação na próxima página

Tabela 2.3 – continuação da página anterior

Nr. Requisito	Data	Descrição	Fonte	Revisor
RC2	15/03/2023	Os funcionários de postos de venda só podem aceder aos registos de venda de bilhetes, registos de clientes e horários de passagem de veículos em estações, não podendo aceder a mais qualquer parte do sistema.	Responsável de uma estação	Miguel Teixeira
RC3	16/03/2023	Os funcionários de postos de venda apenas podem realizar operações de adição sobre venda de bilhetes e clientes, não podendo, no entanto, alterar ou remover informação.	Abílio Inácio	Abílio Inácio
RC4	16/03/2023	Os técnicos de manutenção podem aceder somente a informação relativas a veículos, e horário de passagem de veículos em estações, podendo remover e adicionar informação dos mesmos.	Miguel Teixeira	Miguel Teixeira
RC5	17/03/2023	Os funcionários responsáveis por cada estação podem realizar as mesmas operações de um funcionário de posto de venda e podem adicionar informação sobre estações, assim como fazer adição e remoções a informações sobre vendas de bilhetes e estações, não podendo aceder a mais qualquer parte do sistema.	Abílio Inácio	Abílio Inácio
RC6	17/03/2023	Os clientes podem apenas aceder a informação sobre estações e horários de passagem de veículos.	Miguel Teixeira	Abílio Inácio

2.3 Análise e validação geral dos requisitos

Após a fase de levantamento de requisitos procedemos à análise e validação dos mesmos. Foi marcada uma reunião com o cliente onde foi analisado e debatido todos os requisitos com o intuito de alcançar um consenso acerca das distintas particularidades e exigências no desenvolvimento da base de dados, em particular, qual o grau de exigência.

Durante esta etapa, o foco foi principalmente em harmonizar as discrepâncias observadas, de modo a prevenir possíveis complicações futuras relacionadas com interpretações divergentes dos requisitos do sistema.

Após a aprovação de todos os membros envolvidos foi iniciado a fase da modelação conceitual.

3 Modelação Conceptual

3.1 Apresentação e abordagem de modelação realizada

Após o levantamento, análise e validação dos requisitos deu-se inicio à fase de criação do modelação conceptual. Procedeu-se então à construção de um Diagrama Entidade-Relacionamento (ER) visando alcançar um sistema de base de dados que cumprisse todas as necessidades do cliente e do utilizador final. Inicialmente identificou-se quais as entidades participantes e os seus respectivos atributos. Por fim estabeleceu-se os seus relacionamentos dando origem a um modelo de fácil compreensão por qualquer elemento.

Esta é uma fase fulcral no desenvolvimento de um sistema de base de dados, devido à auxiliar na compreensão do domínio do sistema, quais as suas entidades e atributos, e de que forma o sistema projetado poderá cumprir todos os requisitos e necessidades do cliente.

3.2 Identificação e caracterização das entidades

Analizando os requisitos definidos foram identificadas as seguintes entidades:

3.2.1 Entidade Viagem

Viagem é uma representação de uma viagem que um cliente ou grupo de clientes pode realizar. Esta entidade possui os seguintes atributos:

- “**Id**”: é um identificador único para cada viagem, utilizado como chave primária.
- “**Data**”: é a data em que a viagem é realizada.

3.2.2 Entidade Cliente

Cliente é uma representação de uma pessoa que utiliza os serviços da empresa de Abílio Inácio. Esta entidade possui os seguintes atributos:

- “**Contribuinte**”: Número de Identificação Fiscal do cliente, utilizado como chave primária por ser de carácter único.
- “**Nome**”: é o nome completo do cliente.
- “**DataNascimento**”: é a data de nascimento do cliente.
- “**Género**”: representa o género do cliente.
- “**Email**”: email do cliente.
- “**Contacto**”: contacto telefónico de um cliente, atributo multivalorado devido a um cliente poder possuir um ou mais contactos.
- “**Morada**”: é um atributo composto, constituído por:
 - “**Rua**”: Rua no qual o cliente reside.
 - “**Localidade**”: localidade de residência do cliente.
 - “**CódigoPostal**”: código Postal da residência do cliente.

3.2.3 Entidade Funcionário

Funcionário é uma representação de uma pessoa que trabalha na empresa de Abílio Inácio. Esta entidade possui os seguintes atributos:

- “**Contribuinte**”: Número de Identificação Fiscal do funcionário, utilizado como chave primária por ser de carácter único.
- “**Nome**”: é o nome completo do funcionário.
- “**Data de Nascimento**”: é a data de nascimento do funcionário.
- “**Género**”: representa o género do funcionário.
- “**Cargo**”: representa a função que o funcionário desempenha na empresa.
- “**Salário**”: valor monetário que o funcionário recebe por mês por ser colaborador da rede metropolitana.
- “**IBAN**”: é o número da conta bancária onde o salário do funcionário é depositado.
- “**Email**”: email do funcionário.
- “**Contacto**”: contacto telefónico de um funcionário, atributo multivalorado devido a um Funcionário poder possuir um ou mais contactos.

- “**Morada**”: é um atributo composto, constituído por:
 - “**Rua**”: Rua no qual o funcionário reside.
 - “**Localidade**”: localidade de residência do funcionário.
 - “**CódigoPostal**”: código Postal da residência do funcionário.

3.2.4 Entidade Bilhete

Bilhete é uma representação de permissão para o uso dos serviços metropolitanos. Esta entidade possui os seguintes atributos:

- “**NrVenda**”: é um identificador único para cada bilhete, utilizado como chave primária.
- “**FormaPagamento**”: representa o método utilizado para efetuar o pagamento do bilhete.
- “**Tipo**”: representa o tipo de bilhete, podendo ser um bilhete diário ou mensal.
- “**Data**”: é um atributo composto por:
 - “**DataVenda**”: data em que o bilhete foi vendido.
 - “**DataValidade**”: data em que o bilhete perde a sua validade.
- “**Preço**”: é o preço cobrado pela aquisição do bilhete.

3.2.5 Entidade Estação

Estação é uma representação de uma estação de transportes metropolitanos. Esta entidade possui os seguintes atributos:

- “**Id**”: é um identificador único para cada estação, utilizado como chave primária.
- “**Nome**”: é o nome da estação.
- “**PontosInteresse**”: é uma lista de pontos de interesse localizados perto da estação, como restaurantes, hotéis, pontos turísticos, entre outros.
- “**Descrição**”: é uma breve descrição da estação, que pode conter informações sobre o seu estado, serviços oferecidos, entre outros.
- “**Localização**”: é um atributo composto por “**Latitude**” e “**Longitude**”, que especifica a localização geográfica da estação.

3.2.6 Entidade Veículo

Veículo é uma representação de um veículo de transportes metropolitanos. Esta entidade possui os seguintes atributos:

- “**Id**”: é um identificador único para cada veículo, utilizado como chave primária.
- “**NrLugares**”: é o número de assentos que o veículo possui.
- “**NrCarruagens**”: é o número de carruagens que o veículo possui.

3.3 Identificação e caracterização dos relacionamentos

3.3.1 Relacionamento Bilhete-Viagem

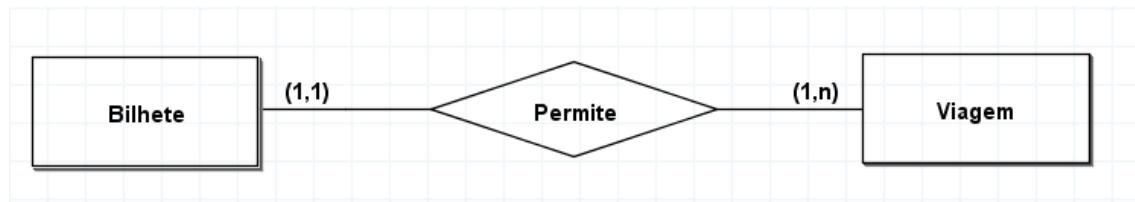


Figura 3.1: Relacionamento das entidades Bilhete-Viagem

Relacionamento: Bilhete permite Viagem

Descrição: foi estabelecida a relação entre os bilhete e viagem com o intuito de obter informação sobre quais bilhetes permitiram a realização de uma viagem.

Cardinalidade: Um para muitos (1:N).

Um bilhete permite uma ou mais viagens, mas uma viagem só pode ser permitida por um bilhete.

Atributos: Este relacionamento não tem atributos associados.

3.3.2 Relacionamento Veículo-Viagem

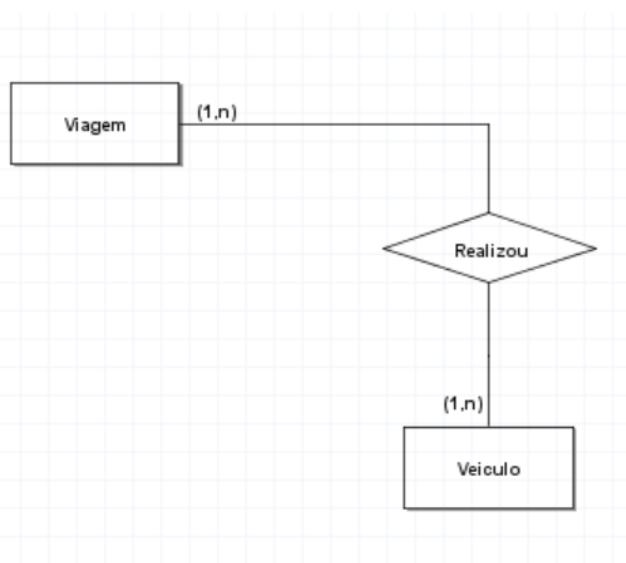


Figura 3.2: Relacionamento das entidades Veículo-Viagem

Relacionamento: Veículo realiza Viagem

Descrição: foi estabelecida a relação entre as viagens realizadas e os veículos existentes para obter a informação quais veículos estiveram envolvidos em uma viagem.

Cardinalidade: Muitos para Muitos (N:N)

Um veículo pode realizar várias viagens e uma viagem pode ser realizada por vários veículos.

Atributos: Este relacionamento não tem atributos associados.

3.3.3 Relacionamento Cliente-Viagem

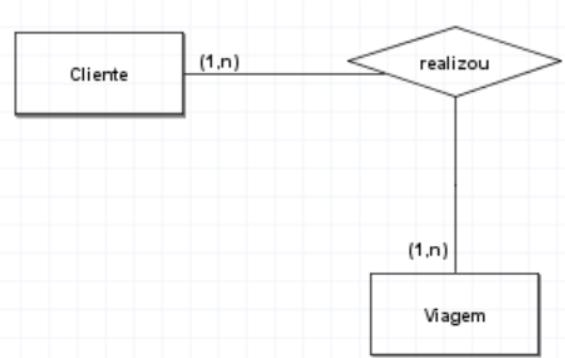


Figura 3.3: Relacionamento das entidades Cliente-Viagem

Relacionamento: Cliente realizou Viagem

Descrição: foi estabelecido a relação entre clientes e viagens com a intenção de se obter informação do histórico das viagens realizadas pelos clientes.

Cardinalidade: Muitos para muitos (N:N)

Um cliente pode fazer várias viagens, mas uma viagem pode ter 1 ou vários clientes.

Atributos: Este relacionamento não tem atributos associados.

3.3.4 Relacionamento Funcionário-Bilhete

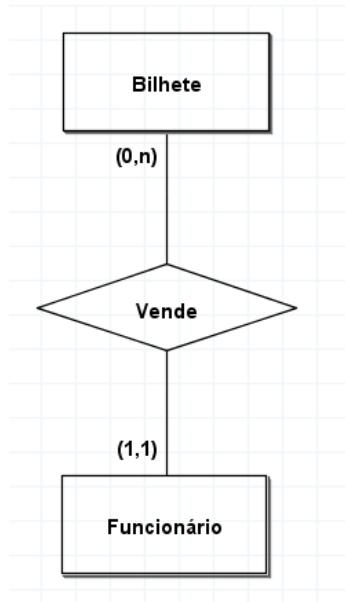


Figura 3.4: Relacionamento das entidades Funcionário-Bilhete

Relacionamento: Funcionário vende Bilhete

Descrição: foi estabelecido a relação entre funcionário e bilhete visando saber qual o funcionário realizou a venda de um bilhete.

Cardinalidade: Um para muitos (1:N)

Um funcionário pode ter vendido zero ou mais bilhetes, mas um bilhete só pode ser vendido por um único funcionário.

Atributos: Este relacionamento não tem atributos associados.

3.3.5 Relacionamento Funcionário-Estação

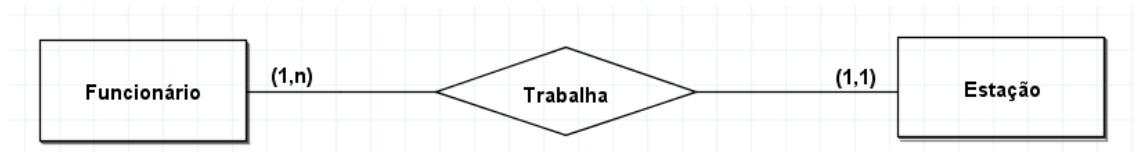


Figura 3.5: Relacionamento das entidades Funcionário-Estação

Relacionamento: Funcionário Trabalha Estação

Descrição: foi estabelecido a relação entre funcionário e estação para obtermos informação das estações onde os funcionários trabalham.

Cardinalidade: Um para muitos (1:N)

Um funcionário pode unicamente trabalhar numa estação, mas numa estação podem trabalhar vários funcionários.

Atributos: Este relacionamento não tem atributos associados.

3.3.6 Relacionamento Funcionário-Veículo

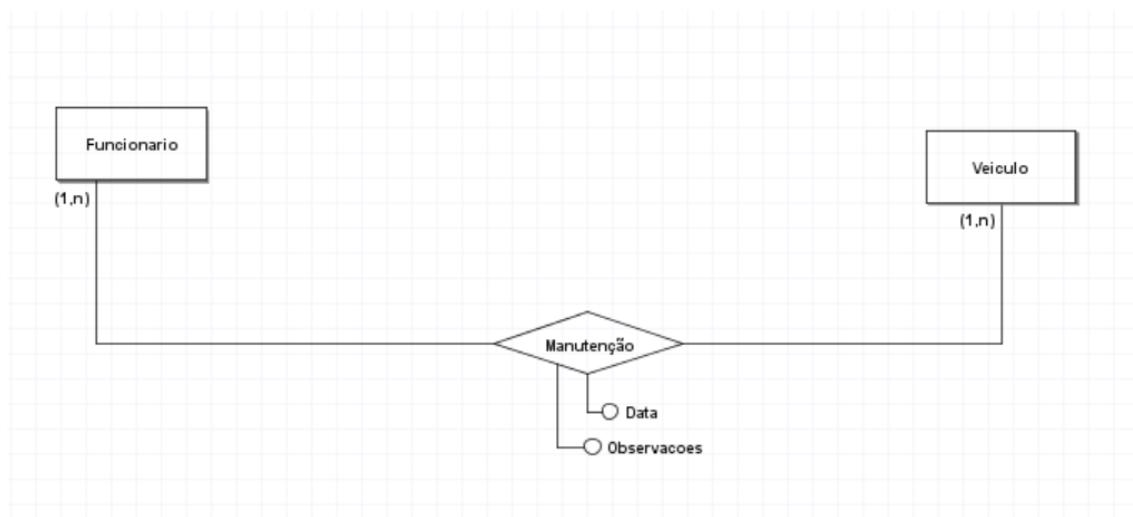


Figura 3.6: Relacionamento das entidades Funcionário-Veículo

Relacionamento: Funcionário Manutenção Veículo

Descrição: foi estabelecido a relação entre funcionário e veículo para obtermos a informação de quem executa a manutenção num veículo.

Cardinalidade Muitos para Muitos (N:N)

Um funcionário pode fazer manutenção a vários veículos, assim como um veículo pode sofrer manutenções por vários funcionários.

Atributos: Neste relacionamento está associado o atributo “Data” para sabermos a data da manutenção e o atributo “Observações” para descrever qual o objetivo e manutenção realizada.

3.3.7 Relacionamento Estação-Veículo

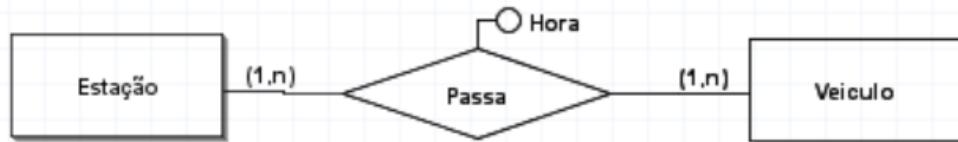


Figura 3.7: Relacionamento das entidades Estação-Veículo

Relacionamento: Veículo passa Estação

Descrição: foi estabelecido a relação entre estação e veículo para conseguirmos gerir e obter informação em que estações os veículos passam.

Cardinalidade: Muitos para Muitos (N:N)

Numa estação podem passar vários veículos, assim como um veículo pode passar em várias estações.

Atributos: Neste relacionamento está associado o atributo “Hora” para podermos informar o horário e dia em que um certo veículo passou numa estação.

3.3.8 Relacionamento Cliente-Bilhete

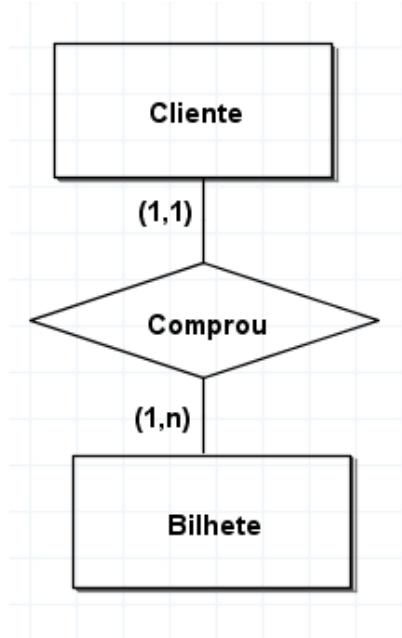


Figura 3.8: Relacionamento das entidades Cliente-Bilhete

Relacionamento: Cliente compra Bilhete

Descrição: foi estabelecido a relação entre cliente e bilhetes de forma, a qual cliente um certo bilhete pertence.

Cardinalidade: Um para Muitos (1:N)

Um cliente pode comprar vários bilhetes, mas um bilhete só pode ser comprado por um cliente.

Atributos: Este relacionamento não tem atributos associados.

3.3.9 Relacionamentos Estação-Viagem

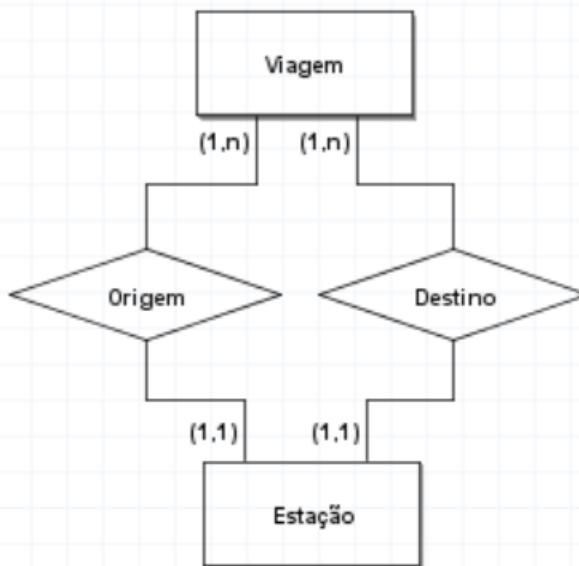


Figura 3.9: Relacionamento das entidades Estação-Viagem

Relacionamentos: Estação Origem Viagem e Estação Destino Viagem

Descrição: Foi estabelecida uma dupla relação entre estação e viagem com o intuito de saber qual a origem e destino da viagem de um cliente.

Cardinalidade: Um para muitos (1:N) em ambas as relações

Uma viagem tem somente uma estação de origem/destino, mas uma estação pode ser origem/destino de várias viagens.

Atributos: Este relacionamento não tem atributos associados.

3.4 Identificação e caracterização da associação dos atributos com as entidades e relacionamentos

Tabela 3.1: Tabela de caracterização do modelo

Entidade	Atributos	Tipo de Dados	Nulo	Composto	Multivalor	Derivado	Candidato
Cliente	Contribuinte	INT	Não	Não	Não	Não	Sim
	Nome	VARCHAR(75)	Não	Não	Não	Não	Não
	DataNascimento	DATE	Sim	Não	Não	Não	Não
	Genero	INT	Não	Não	Não	Não	Não
	Contacto	INT	Não	Não	Sim	Não	Não
	Email	VARCHAR(100)	Sim	Não	Não	Não	Não
	Rua	VARCHAR(100)	Não	Sim	Não	Não	Não
	Localidade	VARCHAR(30)	Não	Sim	Não	Não	Não
	Código-Postal	VARCHAR(8)	Não	Sim	Não	Não	Não
Funcionário	Contribuinte	INT	Não	Não	Não	Não	Sim
	Nome	VARCHAR(75)	Não	Não	Não	Não	Não
	Genero	INT	Não	Não	Não	Não	Não
	DataNascimento	DATE	Não	Não	Não	Não	Não
	Email	VARCHAR(100)	Não	Não	Não	Não	Não
	Contacto	INT	Não	Não	Sim	Não	Não
	Cargo	INT	Não	Não	Não	Não	Não
	Salario	DECIMAL(6,2)	Não	Não	Não	Não	Não
	IBAN	VARCHAR(50)	Não	Não	Não	Não	Não
	Rua	VARCHAR(100)	Não	Sim	Não	Não	Não
	Localidade	VARCHAR(30)	Não	Sim	Não	Não	Não
	Código-Postal	VARCHAR(8)	Não	Sim	Não	Não	Não
Bilhete	Nº Venda	INT	Não	Não	Não	Não	Sim
	FormaPagamento	INT	Não	Não	Não	Não	Não
	Tipo	INT	Não	Não	Não	Não	Não
	Venda	DATE	Não	Sim	Não	Não	Não
	Validade	DATE	Não	Sim	Não	Não	Não
	Preço	DECIMAL(5,2)	Não	Não	Não	Não	Não
Viagem	ID	INT	Não	Não	Não	Não	Sim
	Data	DATE	Não	Não	Não	Não	Não
Estação	ID	INT	Não	Não	Não	Não	Sim
	Nome	VARCHAR(50)	Não	Não	Não	Não	Não
	PontosInteresse	VARCHAR(150)	Sim	Não	Não	Não	Não
	Descrição	VARCHAR(300)	Não	Não	Não	Não	Não
	Latitude	DOUBLE	Não	Sim	Não	Não	Não
	Longitude	DOUBLE	Não	Sim	Não	Não	Não
Veículo	ID	VARCHAR(20)	Não	Não	Não	Não	Sim
	Nº Lugares	INT	Não	Não	Não	Não	Não
	Nº Carruagens	INT	Não	Não	Não	Não	Não

3.5 Apresentação do diagrama ER produzido

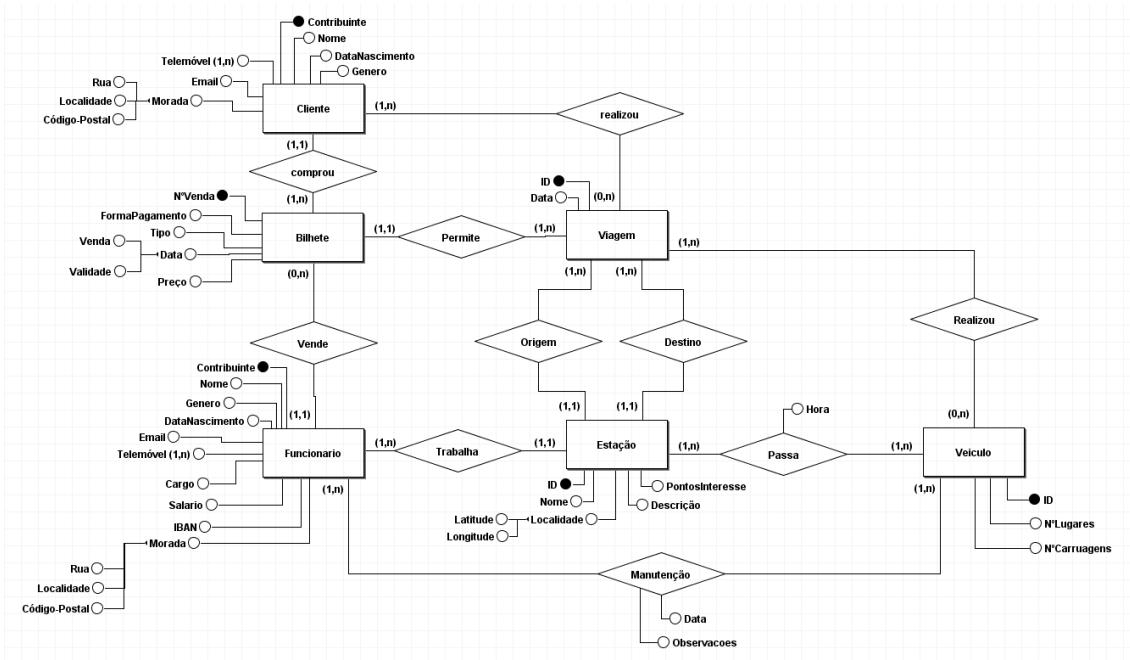


Figura 3.10: Modelo Conceptual

Após definidas todas as entidades, atributos e relacionamentos envolvidos foi desenvolvido o modelo conceptual da figura 3.10 conforme os requisitos e necessidades do cliente.

Foi apresentado o modelo criado ao cliente, no qual depois de algum debate, foi aprovado avançando então para a modelação lógica.

4 Modelação Lógica

4.1 Construção e validação do modelo de dados lógico

Na construção do modelo lógico foram seguidas as regras de conversão apartir do modelo conceptual.

Primeiramente todas as entidades do modelo conceptual são convertidas em tabelas lógicas, já para a conversão dos atributos por norma são associados diretamente à respectiva tabela lógica com excessão dos atributos multivalorados que pertencem a uma tabela lógica própria. Foi também definido para cada tabela qual a sua “chave primária”.

Posteriormente são analisadas as relações entre as entidades, onde os relacionamentos um para muitos são convertidos diretamente, atribuindo a tabela de cardinalidade N uma “chave estrangeira” (chave primária da tabela lógica de cardinalidade 1). Já para os relacionamentos muitos para muitos são decompostos em dois relacionamentos mais simples (um para muitos) criando assim uma tabela lógica intermediaria que irá possuir as “chaves estrangeiras” do relacionamento.

Após a conversão obtemos 8 tabelas referentes a entidades e atributos multivalorados:

- **Funcionário e FuncionárioContacto:**

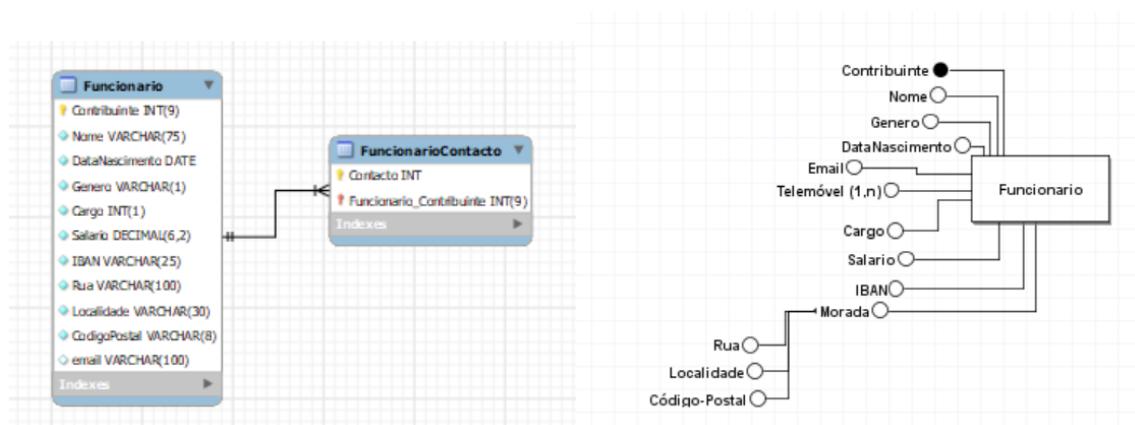


Figura 4.1: Entidade lógica - Funcionário

- **Tabela Lógica:** Funcionario

1. **Chave Primária:** Contribuinte como INT

2. **Atributos:**

- Nome: VARCHAR(75);
- DataNascimento: DATE;
- Genero: VARCHAR(1);
- Cargo: INT;
- Salario: DECIMAL(6,2);
- IBAN: VARCHAR(25);
- Rua: VARCHAR(100);
- Localidade: VARCHAR(30);
- CodigoPostal: VARCHAR(8);
- email: VARCHAR(100);

3. **Chave Estrangeira:** idEstação INT

- **Tabela Lógica:** FuncionarioContacto

1. **Chave Primária:** Contacto como INT

2. **Atributos:**

- Nome: Sem qualquer outro atributo associado;

3. **Chave Estrangeira:** Funcionario_Contribuinte como INT

- **Estação:**

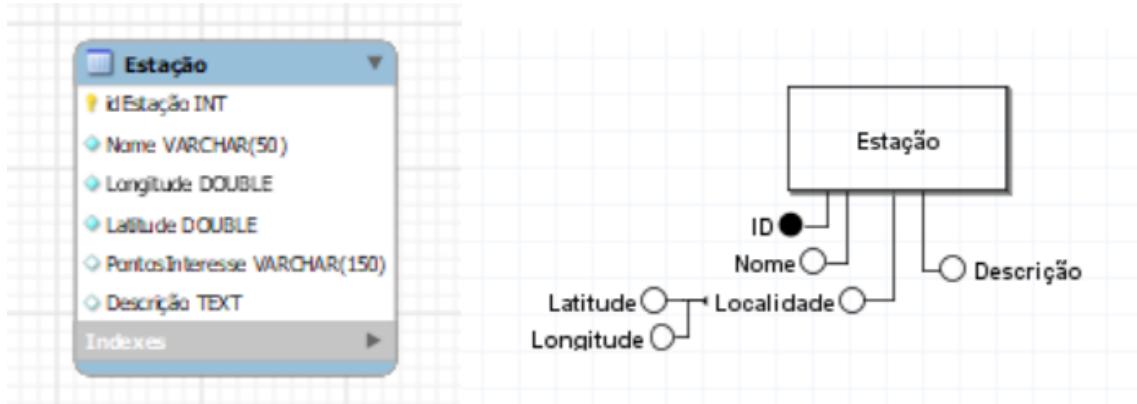


Figura 4.2: Entidade lógica - Estação

1. **Chave Primária:** idEstação como INT

2. **Atributos:**

- Nome: VARCHAR(50);
- Longitude: DOUBLE;
- Latitude: DOUBLE;
- PontosInteresse: VARCHAR(150);
- Descrição: TEXT;

3. **Chave estrangeira:** Não possui

▪ **Cliente:**

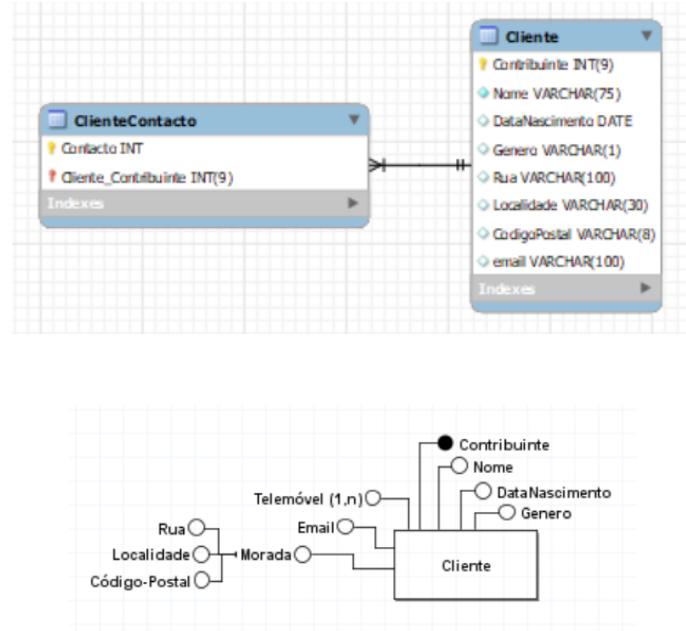


Figura 4.3: Entidade lógica - Cliente

1. **Chave Primária:** Contribuinte como INT(9)

2. **Atributos:**

- Nome: VARCHAR(75);
- DataNascimento: DATE;
- Género: VARCHAR(1);
- Rua: VARCHAR(100);
- Localidade: VARCHAR(30);
- CódigoPostal: VARCHAR(8);
- email: VARCHAR(100);

3. **Chave Estrangeira:** Não possui

▪ **Bilhete:**

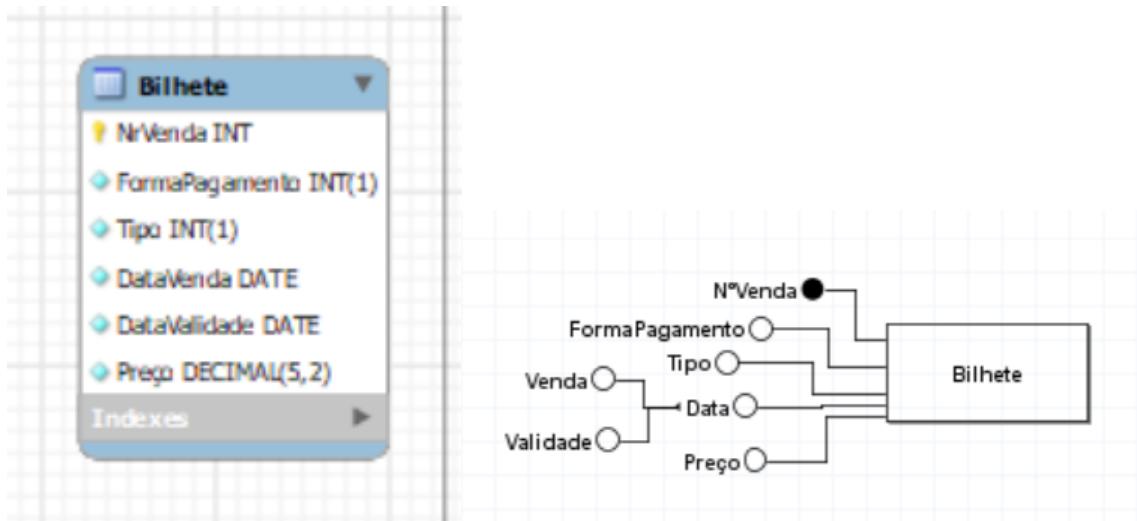


Figura 4.4: Entidade lógica - Bilhete

1. **Chave Primária:** NrVenda como INT
 2. **Atributos:**
 - FormaPagamento: INT(1);
 - Tipo: INT(1);
 - DataVenda: DATE;
 - DataValidade: DATE;
 - Preço: DECIMAL(5,2);
 3. **Chaves Estrangeiras:** Cliente: INT(9), Funcionário_Contribuinte: INT(9)
- **Viagem:**

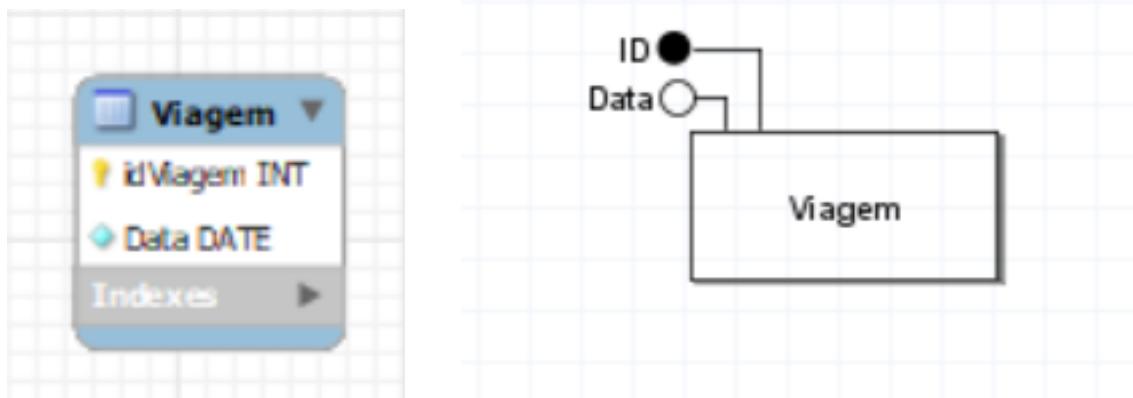


Figura 4.5: Entidade lógica - Viagem

1. **Chave Primária:** idViagem como INT
 2. **Atributos:**
 - Data: DATE
 3. **Chave Estrangeira:** Bilhete_NrVenda: INT
- **Veículo:**

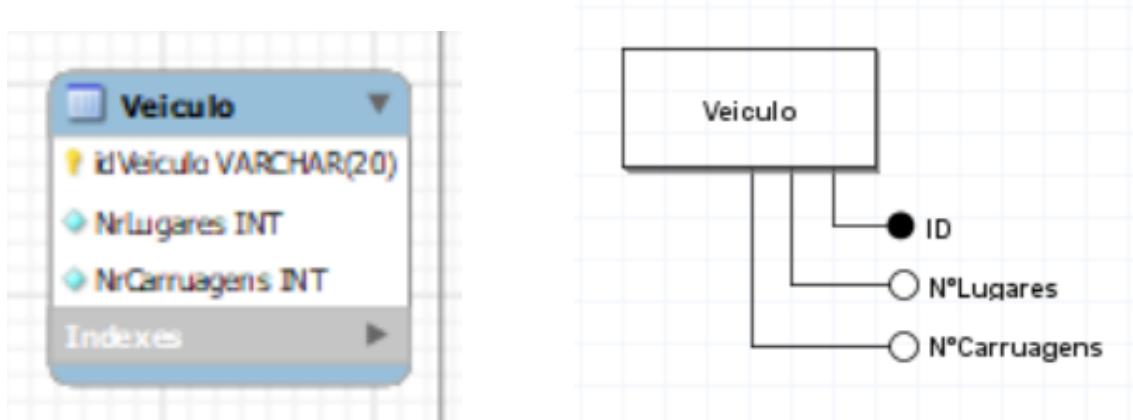


Figura 4.6: Entidade lógica - Veículo

1. **Chave Primária:** idVeiculo como VARCHAR(20)
2. **Atributos:** NrLugares: INT, NrCarruagens: INT
3. **Chave Estrangeira:** Não possui.

4.2 Normalização de Dados

A normalização visa alcançar o isolamento dos dados para que eventuais operações de inserção, modificação ou remoção realizadas sobre um dado atributo sejam realizadas numa única tabela e depois propagadas pelo resto da base de dados seguindo os relacionamentos estabelecidos.

Sendo assim, recorremos à *Primeira Forma Normal (1FN)*, à *Segunda Forma Normal (2FN)* e à *Terceira Forma Normal (3FN)* de modo a verificar a normalização do nosso modelo.

A *FN1* requer que cada atributo (coluna) de uma tabela contenha apenas valores atómicos, ou seja, não devem existir valores repetidos. Além disso, um atributo atómico não pode ser decomposto. Sendo assim, ao analisarmos o nosso modelo, verificamos a tabela *EstaçãoVeículo* possuía como chave primária uma chave composta pelas chaves estrangeiras de *Veículo* e *Estação*. Como a principal finalidade desta tabela seria acolher informação sobre a passagem de veículos em estações, a ocorrência de ambas as chaves poderia ser repetida. Para solucionar este problema atribuímos o atributo *Hora* como parte da chave primária, impedindo a repetição da mesma.

Para alcançar a *2FN* é necessário identificar atributos não chave que dependem parcialmente da chave primária e separá-los em tabelas independentes. Esses atributos são chamados de dependências parciais e podem ser tratados transferindo-os para uma nova tabela, com a parte relevante da chave primária. Sendo assim, ao analisarmos o nosso modelo, podemos concluir que todos os atributos existentes em todas as tabelas dependem inteiramente da sua chave primária, ou seja, cumpre os requisitos da *2FN*.

Por fim, a *3FN* pretende garantir que os atributos não chave dependam exclusivamente da chave primária e não tenham dependências transitivas entre si. Isso evita a redundância de dados e simplifica a estrutura da tabela. Sendo assim, o modelo cumpre os requisitos da *3FN*.

Assim, podemos concluir que o modelo está conforme as normas de normalização.

4.3 Apresentação e explicação do modelo lógico produzido

4.3.1 Conversão de relacionamentos: Um para Muitos

Neste relacionamento são necessárias apenas duas entidades lógicas para definir a relação. Sendo que a chave primária correspondente ao lado 1 é usada como chave estrangeira na entidade correspondente ao lado N.

Cliente - Bilhete

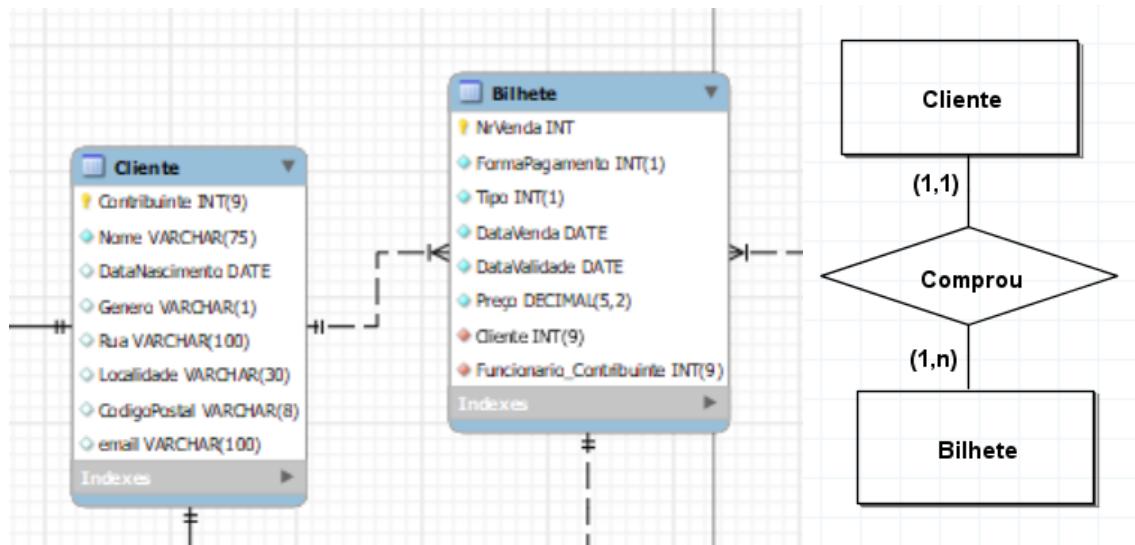


Figura 4.7: Relação Cliente - Bilhete

Funcionário - Bilhete

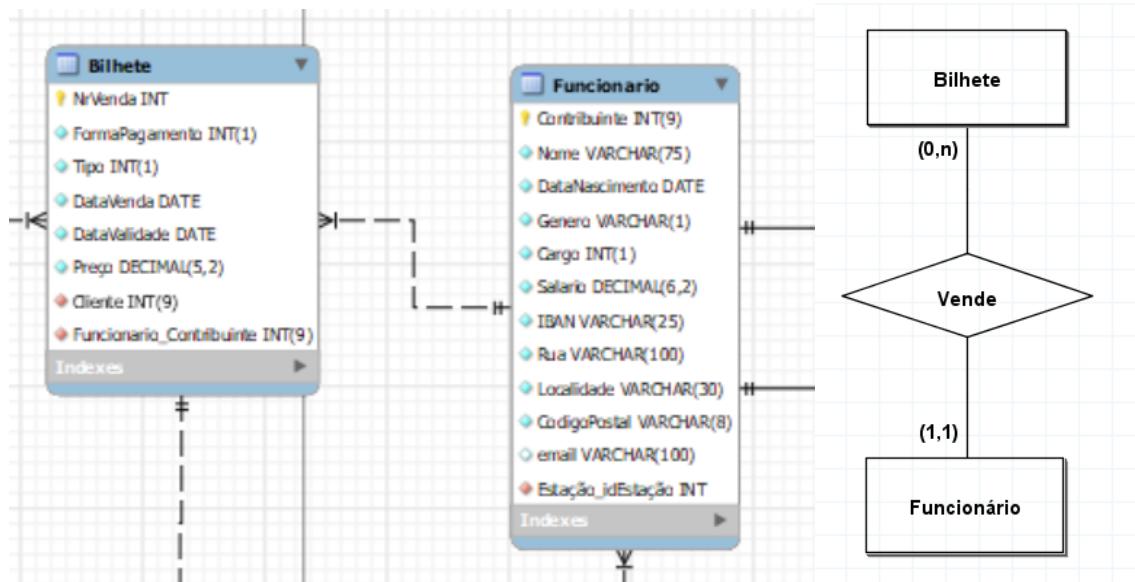


Figura 4.8: Relação Funcionário - Bilhete

Viagem - Bilhete

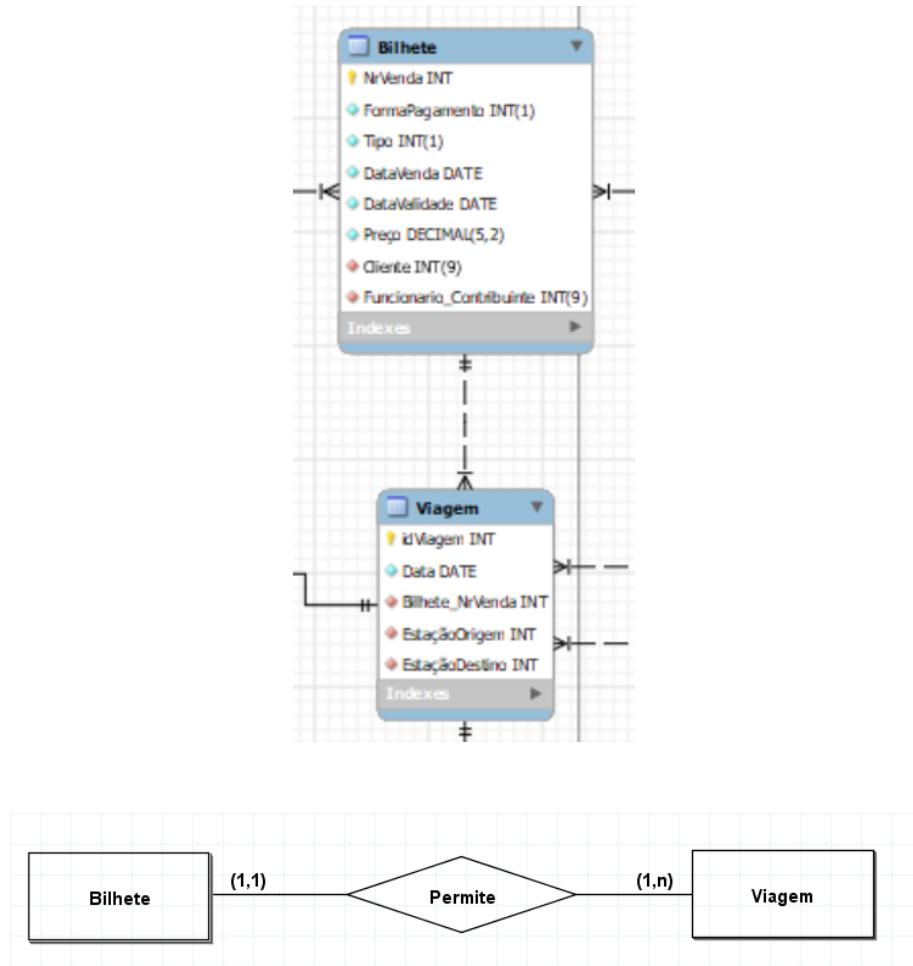


Figura 4.9: Relação Viagem - Bilhete

Funcionário - Estação

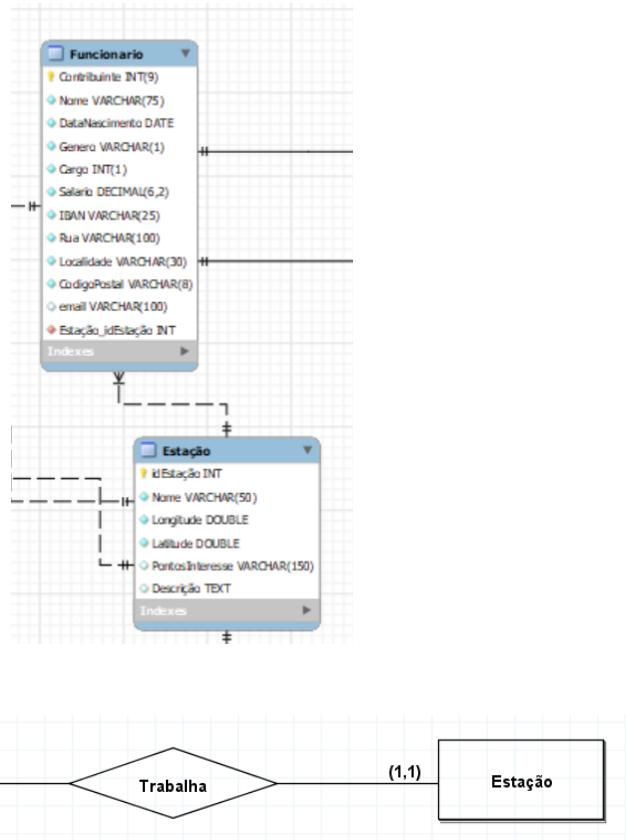


Figura 4.10: Relação Funcionário - Estação

Viagem - Estação

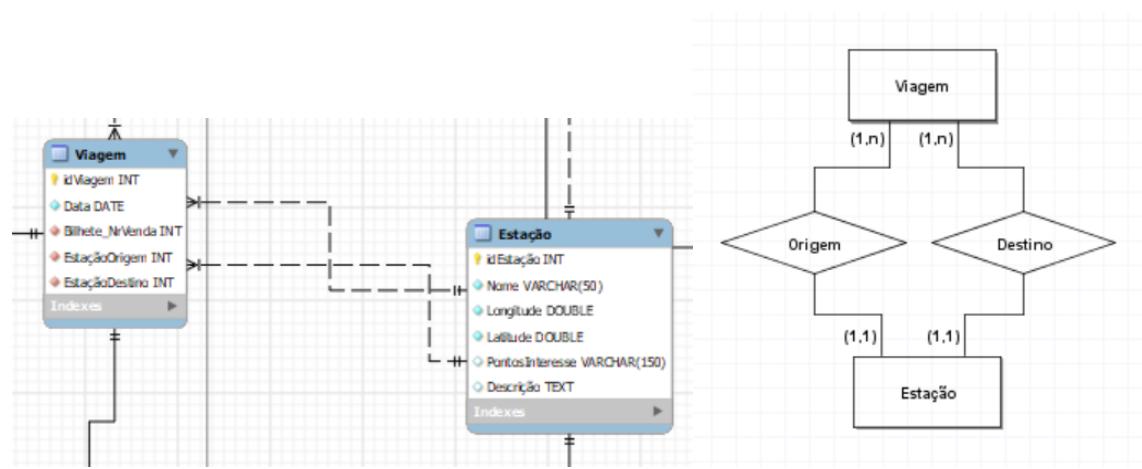


Figura 4.11: Relação Viagem - Estação

4.3.2 Conversão de relacionamentos: Muitos para Muitos

Neste relacionamento são necessárias três entidades lógicas para definir a relação, duas para as entidades e a terceira para o relacionamento. Sendo que as chaves primárias de cada entidade vão ser colocadas na entidade lógica resultante da relação e caso exista algum atributo na relação das entidades, este também será colocado na mesma tabela lógica.

Cliente - Viagem

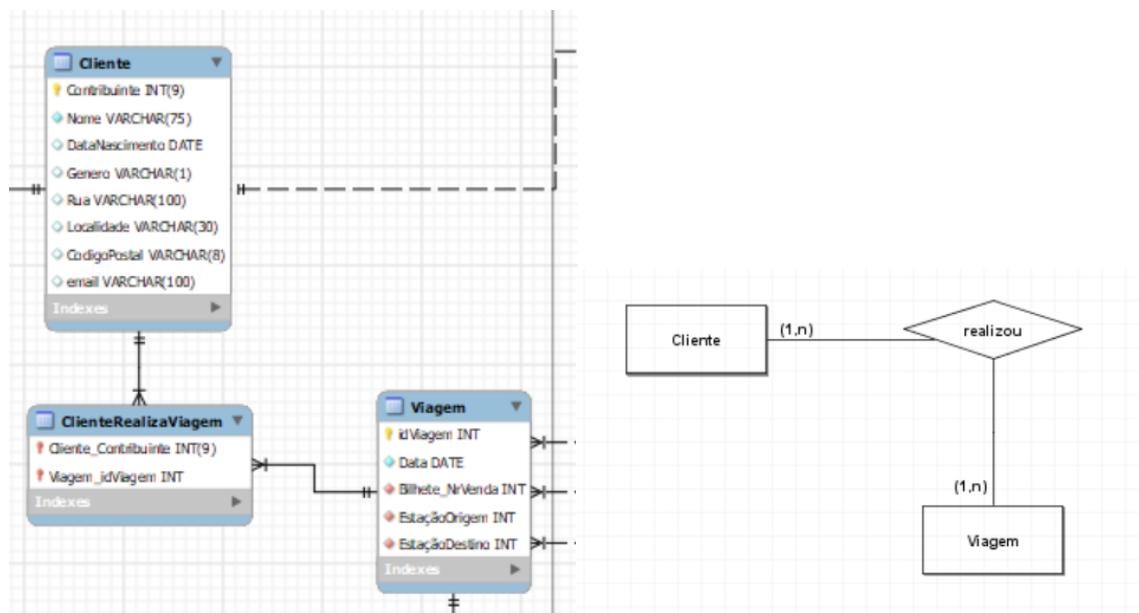


Figura 4.12: Relação Cliente - Viagem

Funcionário - Veículo

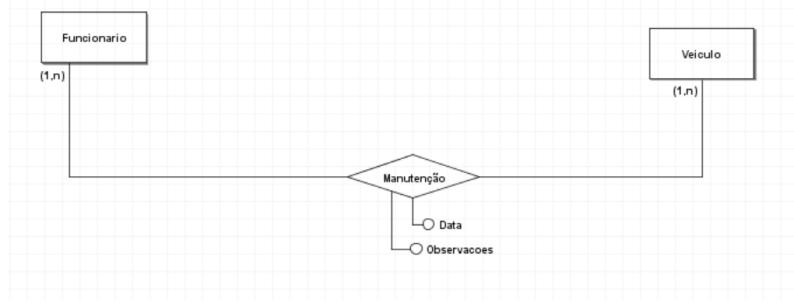
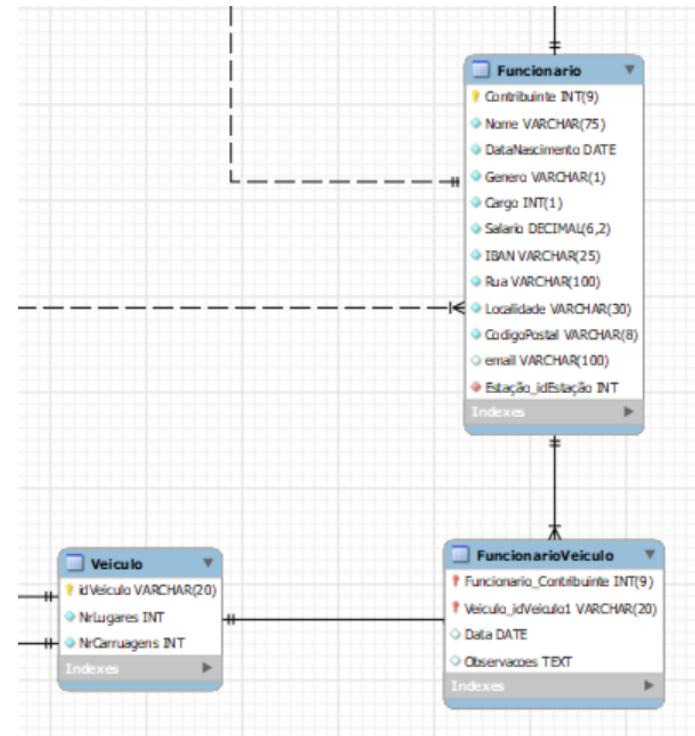


Figura 4.13: Relação Funcionário - Veículo

Viagem - Veículo

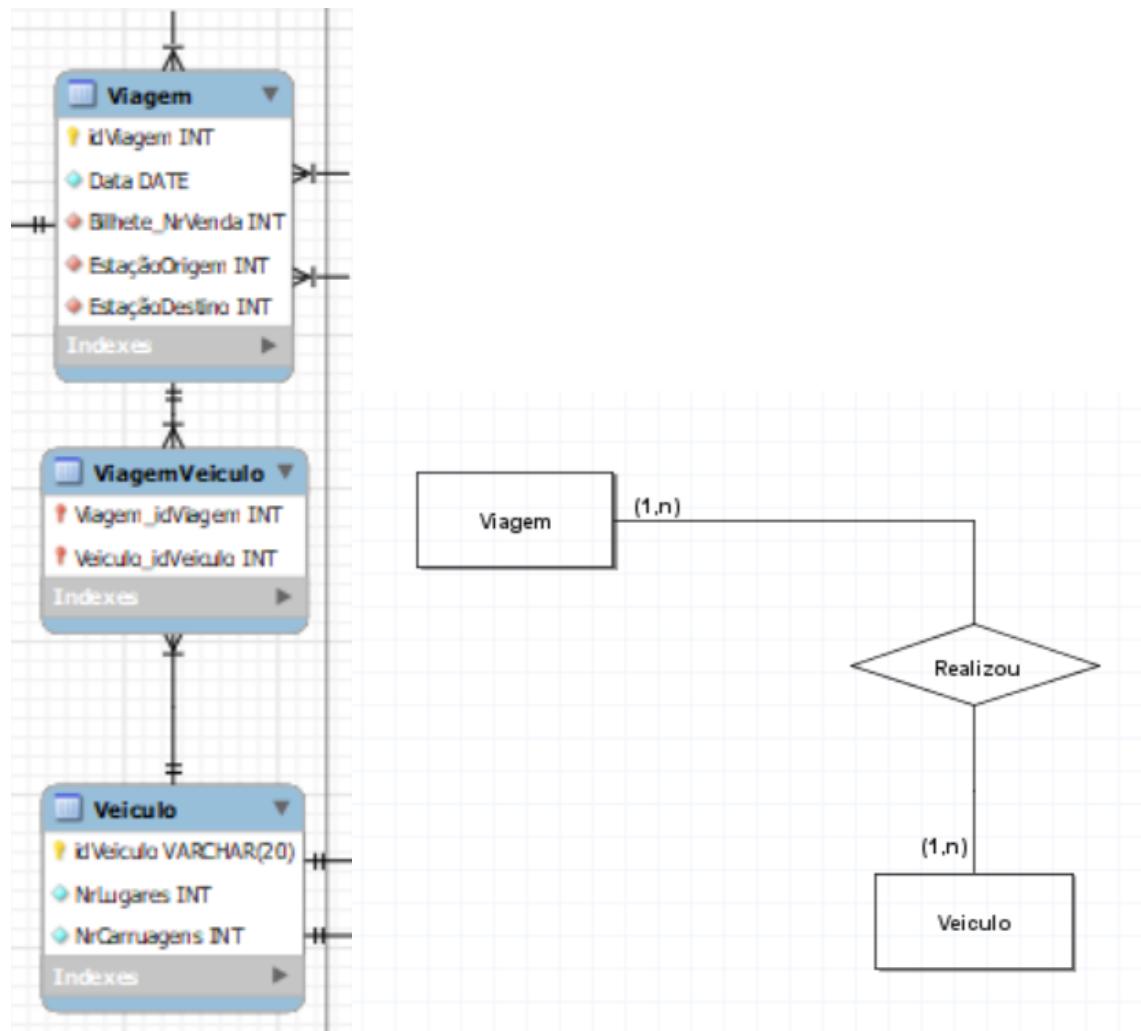


Figura 4.14: Relação Viagem - Veículo

Estação - Veículo

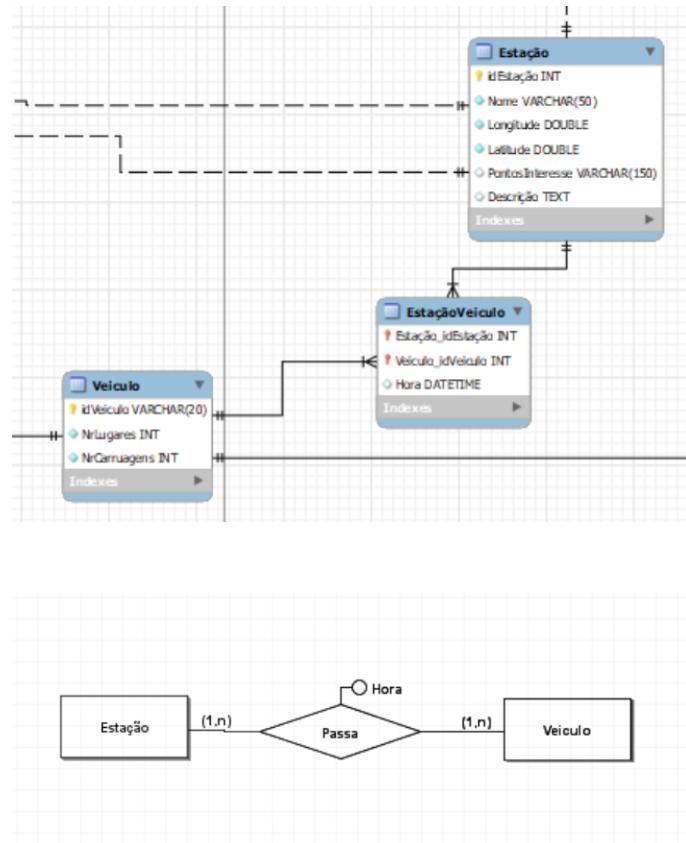


Figura 4.15: Relação Estação - Veículo

4.3.3 Modelo lógico completo

Após convertidas todas as entidades e relacionamentos envolvidos, podemos observar na figura 4.16 o modelo lógico completo.

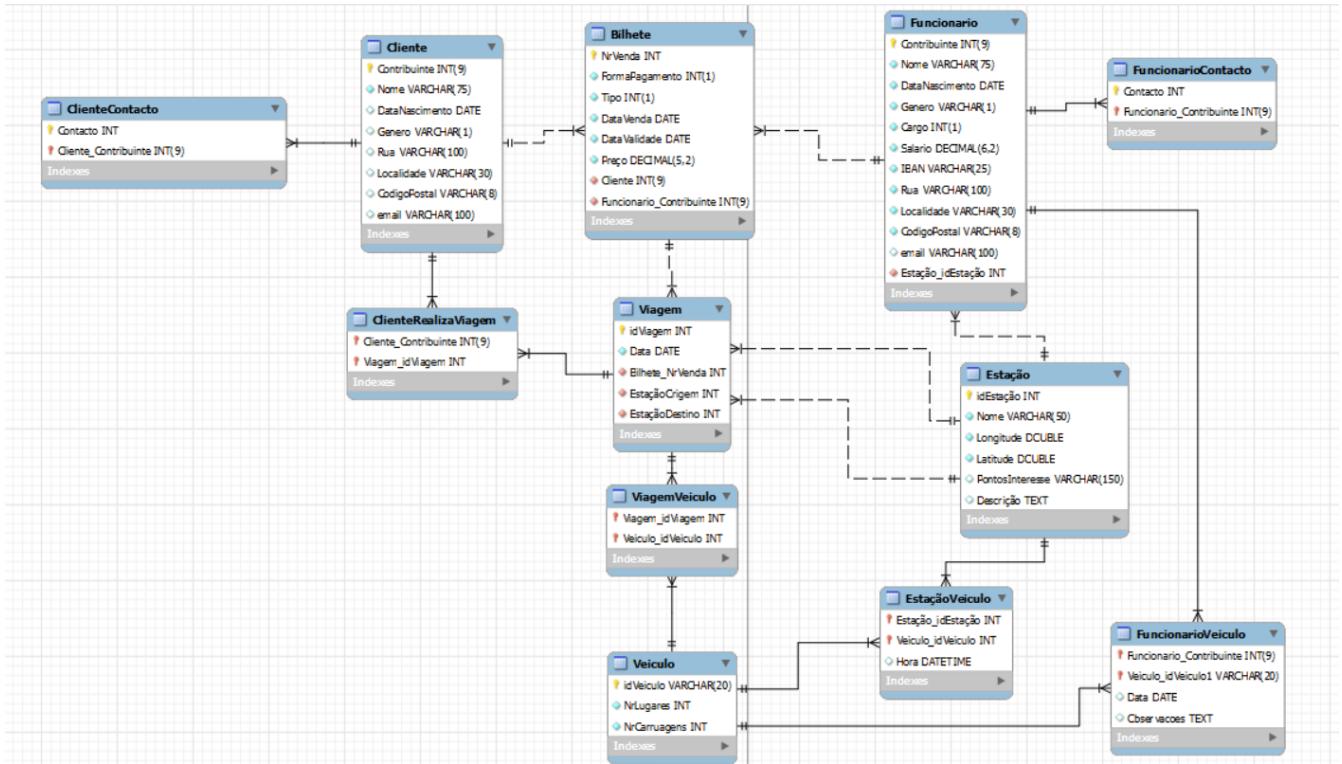


Figura 4.16: Modelo Lógico

4.4 Validação do modelo com interrogações do utilizador

Após a construção do modelo lógico procedeu-se à análise e validação do mesmo. Contactou-se o cliente onde lhe foi apresentado o modelo criado e debatido de que forma este poderia alcançar as suas necessidades e requisitos. Como podemos observar pelo esquema lógico da figura 4.16, este está conforme os requisitos levantados ao cliente como, por exemplo, o requisito *RD1* da tabela 2.2, para um bilhete ser inserido no sistema deverá existir no sistema um registo do cliente, sendo sempre necessário registar o cliente previamente ao registo do bilhete. Isto é garantido através da chave estrangeira do Cliente na tabela bilhete.

Tendo a aprovação do modelo por unanimidade das partes envolvidas procedeu-se para a próxima fase, a implementação física.

5 Implementação Física

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

De modo a aproveitar ao máximo os recursos oferecidos pelo *MySQL*, decidimos utilizar o *MySQL Workbench* como ferramenta de desenho, desenvolvimento e administração de base de dados ao longo de todo processo. Essa escolha permitiu-nos explorar de forma mais eficiente os mecanismos disponíveis.

Procedemos então à conversão das tabelas do nosso modelo lógico, resultando nas seguintes implementações:

5.1.1 Implementação da tabela: Bilhete

```
-- -----
-- Tabela Bilhete
-----

-- Drop table `Bilhete`


Create Table if not exists `BeloMetroDB`.`Bilhete` (
    `NrVenda` Int unsigned unique auto_increment NOT NULL,
    `FormaPagamento` int NOT NULL check (`FormaPagamento` between 0 and 1),
    `Tipo` int NOT NULL check (`Tipo` between 0 and 1),
    `DataVenda` DATE NOT NULL,
    `DataValidade` DATE NOT NULL,
    `Preço` DECIMAL(5,2) NOT NULL,
    `Funcionario` INT unsigned NOT NULL,
    `Cliente` INT unsigned NOT NULL,
    primary key(`NrVenda`),
    constraint `fk_Bilhete_Funcionario`
        foreign key(`Funcionario`)
        references `BeloMetroDB`.`Funcionario` (`Contribuinte`)
        On delete no action
        ON update no action,
    constraint `fk_Bilhete_Cliente`
        foreign key(`Cliente`)
        references `BeloMetroDB`.`Cliente` (`Contribuinte`)
        ON delete no action
        on update no action)
Engine = InnoDB;
```

Figura 5.1: Implementação da tabela: Bilhete

5.1.2 Implementação da tabela: Cliente

```
-- -----
-- Tabela Clientes
-- -----


-- Drop Table `Cliente`
CREATE TABLE if not exists `BeloMetroDB`.`Cliente` (
    `Contribuinte` INT unsigned NOT NULL check (`Contribuinte` between 100000000 and 999999999),
    `Nome` varchar(75) NOT NULL,
    `DataNascimento` DATE,
    `Genero` char,
    `Rua` varchar(100) NOT NULL,
    `Localidade` varchar(30) NOT NULL,
    `CodigoPostal` varchar(8) NOT NULL,
    `email` varchar(100),
    primary key(`Contribuinte`)
)
engine = InnoDB;
```

Figura 5.2: Implementação da tabela: Cliente

5.1.3 Implementação da tabela: ClienteContacto

```
-- -----
-- Tabela ClienteContactos
-- -----


Create table if not exists `BeloMetroDB`.`ClienteContactos` (
    `Cliente` INT unsigned NOT NULL,
    `Contacto` Int unsigned unique NOT NULL check (`Contacto` between 100000000 and 999999999),
    Primary key(`Cliente`,`Contacto`),
    Index `fk_ClienteTelefone_Cliente_idx` (`Cliente` ASC),
    Constraint `fk_ClienteTelefone_Cliente`
        Foreign key (`Cliente`)
        References `BeloMetroDB`.`Cliente` (`Contribuinte`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
Engine = InnoDB;
```

Figura 5.3: Implementação da tabela: ClienteContacto

5.1.4 Implementação da tabela: ClienteRealizaViagem

```
-- -----
-- Tabela ClienteRealizaViagem
-- -----



-- Drop table `ClienteRealizaViagem`



Create Table If Not Exists `BeloMetroDB`.`ClienteRealizaViagem` (
    `Cliente` Int unsigned NOT NULL,
    `Viagem` Int unsigned unique NOT NULL,
    Primary Key(`Cliente`,`Viagem`),
    constraint `fk_ClienteRealizaViagem_Cliente`
        foreign key (`Cliente`)
        references `BeloMetroDB`.`Cliente` (`Contribuinte`)
        On Delete No Action
        On Update No Action,
    Constraint `fk_ClienteRealizaViagem_Viagem`
        foreign key (`Viagem`)
        References `BeloMetroDB`.`Viagem` (`IdViagem`)
        On Delete No Action
        On Update No Action)
Engine = InnoDB;
```

Figura 5.4: Implementação da tabela: ClienteRealizaViagem

5.1.5 Implementação da tabela: Estação

```
-- -----
-- Tabela Estação
-- -----
-- Drop table `Estação`  
  
Create Table If Not Exists `BeloMetroDB`.`Estação` (
    `IdEstação` Int unsigned auto_increment unique NOT NULL,
    `Nome` varchar(50) NOT NULL,
    `Longitude` Double NOT NULL,
    `Latitude` Double NOT NULL,
    `PontosInteresse` Varchar(150) NOT NULL,
    `Descrição` TEXT NOT NULL,
    Primary Key(`IdEstação`))
Engine = InnoDB;
```

Figura 5.5: Implementação da tabela: Estação

5.1.6 Implementação da tabela: EstaçãoVeiculo

```
-- -----
-- Tabela EstaçãoVeiculo
-- -----


-- Drop Table `EstaçãoVeiculo`


Create Table If Not Exists `BeloMetroDB`.`EstaçãoVeiculo`(
    `Estação` Int unsigned NOT NULL,
    `Veiculo` varchar(20) NOT NULL,
    `Hora` DATETIME NOT NULL,
    primary key (`Estação`, `Veiculo`, `Hora`),
    constraint `fk_EstaçãoVeiculo_Estação`
        foreign key (`Estação`)
        References `BeloMetroDB`.`Estação` (`IdEstação`)
        On Delete No action
        On Update No action,
    constraint `fk_EstaçãoVeiculo_Veiculo`
        foreign key(`Veiculo`)
        References `BeloMetroDB`.`Veiculo` (`IdVeiculo`)
        On Delete No Action
        On Update No Action)
Engine = InnoDB;
```

Figura 5.6: Implementação da tabela: EstaçãoVeiculo

5.1.7 Implementação da tabela: Funcionario

```
-- Tabela Funcionario
-- -----
-- Drop Table `Funcionario` 

Create Table `BeloMetroDB`.`Funcionario` (
  `Contribuinte` INT unsigned NOT NULL check (`Contribuinte` between 100000000 and 999999999),
  `Nome` varchar(75) NOT NULL,
  `DataNascimento` DATE,
  `Genero` char,
  `Rua` varchar(100) NOT NULL,
  `Localidade` varchar(30) NOT NULL,
  `CodigoPostal` varchar(8) NOT NULL,
  `email` varchar(100),
  `cargo` INT NOT NULL check (`cargo` between 0 and 3),
  `Salario` DECIMAL(6,2) NOT NULL,
  `IBAN` varchar(25) NOT NULL,
  `Estação` Int unsigned Not NULL,
  primary key(`Contribuinte`),
  Constraint `fk_Funcionario_Estação`
    Foreign Key(`Estação`)
      references `BeloMetroDB`.`Estação` (`IdEstação`)
      On Delete No Action
      On Update No Action
);
```

Figura 5.7: Implementação da tabela: Funcionario

5.1.8 Implementação da tabela: FuncionarioContacto

```
-- -----
-- Tabela FuncionarioContactos
-- -----



-- Drop table `FuncionarioContactos`



Create table if not exists `BeloMetroDB`.`FuncionarioContactos` (
  `Funcionario` INT unsigned NOT NULL,
  `Contacto` Int unsigned NOT NULL check (`Contacto` between 100000000 and 999999999),
  Primary key(`Funcionario`,`Contacto`),
  Index `fk_FuncionarioContacto_Funcionario_idx` (`Funcionario` ASC),
  Constraint `fk_FuncionarioContacto_Funcionario`
    Foreign key (`Funcionario`)
    References `BeloMetroDB`.`Funcionario` (`Contribuinte`)
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
Engine = InnoDB;
```

Figura 5.8: Implementação da tabela: FuncionarioContacto

5.1.9 Implementação da tabela: FuncionarioVeiculo

```
-- -----
-- Tabela FuncionarioVeiculo
-- -----



-- Drop Table `FuncionarioVeiculo`



Create Table If Not Exists `BeloMetroDB`.`FuncionarioVeiculo`(
  `Funcionario` Int unsigned NOT NULL,
  `Veiculo` Varchar(20) NOT NULL,
  `Data` Date NOT NULL,
  `Observações` TEXT NOT NULL,
  primary key (`Funcionario`,`Veiculo`,`Data`),
  constraint `fk_FuncionarioVeiculo_Funcionario`
    foreign key (`Funcionario`)
    References `BeloMetroDB`.`Funcionario`(`Contribuinte`)
    On Delete No action
    On Update No action,
  constraint `fk_FuncionarioVeiculo_Veiculo`
    foreign key(`Veiculo`)
    References `BeloMetroDB`.`Veiculo`(`IdVeiculo`)
    On Delete No Action
    On Update No Action)
Engine = InnoDB;
```

Figura 5.9: Implementação da tabela: FuncionarioVeiculo

5.1.10 Implementação da tabela: Veiculo

```
-- -----
-- Tabela Veiculo
-- -----
-- Drop table `Veiculo`  
  
Create Table If Not Exists `BeloMetroDB`.`Veiculo` (
  `IdVeiculo` varchar(20) Unique Not Null,
  `NrCarruagens` Int Unsigned Not Null,
  `NrLugares` Int Unsigned Not Null,
  Primary Key (`IdVeiculo`))
Engine = InnoDB;
```

Figura 5.10: Implementação da tabela: Veiculo

5.1.11 Implementação da tabela: Viagem

```
-- -----
-- Tabela Viagem
-- -----
-- Drop table `Viagem`
Create Table If not Exists `BeloMetroDB`.`Viagem` (
    `IdViagem` Int unsigned auto_increment unique NOT NULL,
    `Bilhete` Int unsigned NOT NULL,
    `Origem` Int unsigned NOT NULL,
    `Destino` Int unsigned NOT NULL,
    `Data` DATE NOT NULL,
    primary key(`IdViagem`),
    Constraint `fk_Viagem_Bilhete`
        foreign Key(`Bilhete`)
        References `BeloMetroDB`.`Bilhete` (`NrVenda`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    Constraint `fk_ViagemEstação_Origem`
        Foreign Key(`Origem`)
        References `BeloMetroDB`.`Estação` (`IdEstação`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    Constraint `fk_ViagemEstação_Destino`
        Foreign Key(`Destino`)
        References `BeloMetroDB`.`Estação` (`IdEstação`)
        ON DELETE NO ACTION
        ON UPDATE NO ACTION
)
engine = InnoDB;
```

Figura 5.11: Implementação da tabela: Viagem

5.1.12 Implementação da tabela: ViagemVeiculo

```
-- -----
-- Tabela ViagemVeiculo
-- -----



-- Drop Table `ViagemVeiculo`



Create Table If Not Exists `BeloMetroDB`.`ViagemVeiculo`(
    `Viagem` Int unsigned NOT NULL,
    `Veiculo` varchar(20) NOT NULL,
    primary key (`Viagem`, `Veiculo`),
    constraint `fk_ViagemVeiculo_Viagem`
        foreign key (`Viagem`)
        References `BeloMetroDB`.`Viagem` (`IdViagem`)
        On Delete No action
        On Update No action,
    constraint `fk_ViagemVeiculo_Veiculo`
        foreign key(`Veiculo`)
        References `BeloMetroDB`.`Veiculo` (`IdVeiculo`)
        On Delete No Action
        On Update No Action)
Engine = InnoDB;
```

Figura 5.12: Implementação da tabela: ViagemVeiculo

5.2 Tradução das interrogações do utilizador para SQL (alguns exemplos)

A tradução das interrogações do utilizador para SQL refere-se à transformação de perguntas ou solicitações feitas pelos utilizadores em que possam ser executados numa base de dados, ou seja, isso envolve interpretar a intenção do utilizador e expressá-la de maneira a recuperar, manipular ou atualizar dados na base de dados.

5.2.1 Exemplos

RM6 - O sistema deverá saber qual o tipo de bilhete que um cliente possui introduzindo o seu Contribuinte

```
delimiter $$  
Create Procedure tipoBilheteCliente(In Nif_Cliente INT)  
begin  
    Select C.Contribuinte As 'Contribuinte do Cliente', C.Nome As 'Nome do Cliente', B.NrVenda As 'Numero de Bilhete',  
        Case B.Tipo  
            When 0 Then 'Bilhete Diario'  
            When 1 Then 'Bilhete Mensal'  
        End AS 'Tipo de Bilhete', B.DataVenda as 'Data de Venda', B.DataValidade as 'Data de Validade'  
    From Bilhete as B Right outer Join Cliente as C  
        On B.Cliente = C.Contribuinte  
        Where B.NrVenda Is Not Null and C.Contribuinte = Nif_Cliente  
        Order by B.DataVenda DESC;  
end $$  
delimiter ;
```

Figura 5.13: Código da query 6

RM7 - O sistema deverá identificar o número e o nome das estações por onde passa um certo veículo numa certa Data

```

delimiter $$

Create Procedure REFuncPorEstacao(In Id_Estacao INT)
Begin
    Select F.Contribuinte As 'Contribuinte Funcionario', F.Nome As 'Nome Funcionario',
    Case F.Cargo
        When 0 Then 'Funcionario Posto Venda'
        When 1 Then 'Responsavel Estação'
        When 2 Then 'Tecnico de Manutenção'
        When 3 then 'Administrador'
    END AS 'Cargo', E.IdEstação As 'Id Estação' , E.Nome As 'Nome da Estação'
    From Funcionario as F Right outer Join Estação as E
        On F.Estação = E.IdEstação
        where E.IdEstação = Id_Estacao
        Order by F.Cargo desc, F.Nome ASC;

End $$

delimiter ;

```

Figura 5.14: Código da query 7

RM8 - O sistema deverá identificar quem vendeu e para qual cliente um bilhete foi vendido através do número de venda do bilhete

```

delimiter $$

create Procedure FPInfoBilhete(In NrVenda_Bilhete INT)
Begin
    Select B.NrVenda AS 'Numero Venda Bilhete', C.Contribuinte AS 'Contribuinte Cliente' ,
        C.Nome AS 'Nome Cliente', F.Contribuinte AS 'Contribuinte Funcionario' ,F.Nome AS 'Nome Funcionario'
    From Bilhete As B Right outer Join Cliente as C
        On B.Cliente = C.Contribuinte
        Right Outer Join Funcionario as F
            On B.Funcionario = F.Contribuinte
            where B.Funcionario IS NOT NULL AND B.Cliente IS NOT NULL AND B.NrVenda = NrVenda_Bilhete
            order by C.Nome ASC, F.Nome ASC;

end $$

delimiter ;

```

Figura 5.15: Código da query 8

5.3 Definição e caracterização das vistas de utilização em SQL (alguns exemplos)

Para poder garantir um certo controlo na visualização dos dados do sistema por parte dos utilizadores, foram definidas as seguintes vistas de utilização:

Vista para a visualização de bilhetes do sistema bem como horário de passagem de veículos em estações:

```
-- drop view VerBilhetes
Create View VerBilhetes as
  Select B.NrVenda As 'Nr. Venda',
  Case B.FormaPagamento
    When 0 Then 'Dinheiro'
    When 1 Then 'Multibanco'
  End As 'Forma Pagamento',
  Case B.Tipo
    When 0 Then 'Diario'
    When 1 Then 'Mensal'
  End As 'Tipo Bilhete', B.DataVenda As 'Data Venda', B.DataValidade As 'Data Validade', B.Preço as 'Preço',
  C.Contribuinte As 'NIF Cliente', C.Nome As 'Nome Cliente', F.Contribuinte As 'Nif Funcionario', F.Nome as 'Nome Funcionario'
  From Bilhete As B Inner Join Cliente As C
    On C.Contribuinte = B.Cliente
    Inner Join Funcionario As F
      On F.Contribuinte = B.Funcionario
  Order by B.DataVenda DESC;

Create View VerPassagemVeiculo As
  select E.Nome As 'Nome Estação', V.IdVeiculo As 'Veiculo', F.Hora As 'Hora passagem'
  From EstaçãoVeiculo as F
    Inner Join Estação as E
      On E.IdEstação = F.Estação
      Inner Join Veiculo V
        On V.IdVeiculo = F.Veiculo
  Order by F.Hora DESC;
```

Figura 5.16: Vista *VerBilhetes* e *VerPassagemVeiculo*

Vista para a visualização de clientes e dos seus respetivos contactos.

```
-- drop view Verclientes
Create view Verclientes as
  Select C.Contribuinte, C.Nome, C.DataNascimento, C.Genero, concat(C.Rua, ' ', C.Localidade, ' ', C.CodigoPostal) as Morada, C.Email
  from cliente As C
  order by C.Nome Asc;

-- drop view VerClienteContacto
Create view VerClienteContacto as
  Select C.Contribuinte, C.Nome, F.Contacto
  From Cliente as C
    Inner Join clientecontactos as F
      On C.Contribuinte = F.Cliente
  order by C.Nome Asc;
```

Figura 5.17: Vista *VerCliente* e *VerClienteContacto*

Vista para a visualização de informação de estações.

```
-- drop view VerEstações
Create View VerEstações as
  Select Nome As 'Estação', concat(Longitude, ' ', Latitude) as 'Localização', PontosInteresse as 'Pontos de Interesse', Descrição as 'Descrição'
  From Estação
  Order by Nome Asc;
```

Figura 5.18: Vista *VerEstações*

Vista para a visualização de funcionários e dos seus respetivos contactos.

```
-- drop view VerFuncionarios
create view VerFuncionarios as
  Select F.Contribuinte, F.Nome, F.DataNascimento, F.Genero, concat(F.Rua, ' ', F.Localidade, ' ', F.CodigoPostal) as Morada, F.email,
  Case F.Cargo
    When 0 then 'Funcionario Posto Venda'
    When 1 then 'Responsavel Estação'
    When 2 then 'Técnico de manutenção'
    When 3 then 'Administrador'
  End As 'Cargo' , E.Nome As 'Estação Desnignada'
  From Funcionario As F
  Inner Join Estação As E
  On F.Estação = E.IdEstação
  Order by F.Nome Asc;

-- drop view VerFuncionarioContacto
create view VerFuncionarioContacto as
  Select C.Contribuinte, C.Nome, F.Contacto
  From Funcionario as C
  Inner Join Funcionariocontactos as F
  On C.Contribuinte = F.Funcionario
  order by C.Nome Asc;
```

Figura 5.19: Vista *VerFuncionarios* e *VerFuncionarioContacto*

Vista para a visualização de informações relativas a veículos e das suas respetivas manutenções.

```
-- drop view VerVeiculos
create view VerVeiculos as
  Select V.IdVeiculo as 'Identificador', V.NrCarruagens as 'Nr. Carruagens', V.NrLugares as 'Nr. Lugares'
  From Veiculo as V
  Order by V.NrLugares;

-- drop view VerVeiculosManutencao
create view VerVeiculosManutencao as
  Select F.Nome as 'Tecnico', V.Veiculo as 'Identificador Veiculo', V.Data as 'Data Realização', V.Observações
  From Funcionarioveiculo as V
  Inner Join Funcionario as F
  On V.Funcionario = F.Contribuinte
  order by V.Data Desc;
```

Figura 5.20: Vista *VerVeiculo* e *VerVeiculoManutencao*

5.4 Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)

Uma componente importante na construção de uma base de dados é a alocação e gestão de espaço físico o *hardware*. Esta componente é bastante fulcral pois impacta diretamente os custos associados à manutenção da base de dados. Tendo tudo isto em consideração, foi

realizada uma análise do modelo desenvolvido com o objetivo de determinar o espaço a alocar para um funcionamento eficiente da base de dados.[1, 2]

Tabela 5.1: Espaço ocupado no disco por cada tipo de dados[2]

Tipo de Dados	Tamanho (Bytes)
INT	4
VARCHAR(N)	N+1
DATE	3
DATETIME	8
DOUBLE	8
TEXT	$L+2$, onde $L < L^{2^{16}} = Length$
DECIMAL	4

Tabela 5.2: Espaço ocupado em disco por Cliente

	Atributos	Tipos de dados	Espaço no disco
Cliente	Contribuinte	INT	4
	Nome	VARCHAR(75)	76
	DataNascimento	DATE	3
	Genero	INT	4
	Contacto	INT	4
	Email	VARCHAR(100)	101
	Rua	VARCHAR(100)	101
	Localidade	VARCHAR(30)	31
	Código-Postal	VARCHAR(8)	9
Total			333

Tabela 5.3: Espaço ocupado em disco por Funcionario

	Atributos	Tipos de dados	Espaço no disco
Funcionário	Contribuinte	INT	4
	Nome	VARCHAR(75)	76
	Genero	INT	4
	DataNascimento	DATE	3
	Email	VARCHAR(100)	101
	Contacto	INT	4
	Cargo	INT	4
	Salario	DECIMAL	4
	IBAN	VARCHAR(50)	51
Total			392

Tabela 5.4: Espaço ocupado em disco por Bilhete

	Atributos	Tipos de dados	Espaço no disco
Bilhete	Nº Venda	INT	4
	FormaPagamento	INT	4
	Tipo	INT	4
	Venda	DATE	3
	Validade	DATE	3
	Preço	DECIMAL	4
Total			22

Tabela 5.5: Espaço ocupado em disco por Viagem

	Atributos	Tipos de dados	Espaço no disco
Viagem	ID	INT	4
	Data	DATE	3
Total			
	7		

Tabela 5.6: Espaço ocupado em disco por Estação

	Atributos	Tipos de dados	Espaço no disco
Estação	ID	INT	4
	Nome	VARCHAR(50)	51
	PontosInteresse	VARCHAR(150)	151
	Descrição	VARCHAR(300)	301
	Latitude	DOUBLE	8
	Longitude	DOUBLE	8
Total			523

Tabela 5.7: Espaço ocupado em disco por Veículo

	Atributos	Tipos de dados	Espaço no disco
Veículo	ID	VARCHAR(20)	21
	Nº Lugares	INT	4
	Nº Carruagens	INT	4
Total			
	29		

Tabela 5.8: Espaço ocupado em disco por EstaçãoVeículo

	Atributos	Tipos de dados	Espaço no disco
EstaçãoVeículo	Estação_idEstação	INT	4
	Veiculo_idVeiculo	VARCHAR(20)	21
	Hora	DATETIME	8
Total			
	35		

Tabela 5.9: Espaço ocupado em disco por FuncionarioVeiculo

	Atributos	Tipos de dados	Espaço no disco
FuncionarioVeiculo	Funcionario_Contribuinte	INT	4
	Veiculo_idVeiculo	VARCHAR(20)	21
	Data	DATE	3
	Observacoes	TEXT	L+2
Total			30+L

Tabela 5.10: Espaço ocupado em disco por ViagemVeiculo

	Atributos	Tipos de dados	Espaço no disco
ViagemVeiculo	Viagem_idViagem	INT	4
	Veiculo_idVeiculo	VARCHAR(20)	21
Total			25

Tabela 5.11: Espaço ocupado em disco por ClienteRealizaViagem

	Atributos	Tipos de dados	Espaço no disco
ClienteRealizaViagem	Cliente_Contribuinte	INT	4
	Viagem_idViagem	INT	4
Total			8

Tabela 5.12: Espaço ocupado em disco por ClienteContacto

	Atributos	Tipos de dados	Espaço no disco
ClienteContacto	Contacto	INT	4
	Cliente_Contribuinte	INT	4
Total			8

Tabela 5.13: Espaço ocupado em disco por FuncionarioContacto

	Atributos	Tipos de dados	Espaço no disco
FuncionarioContacto	Contacto	INT	4
	Funcionario_Contribuinte	INT	4
Total			8

Tabela 5.14: Espaço ocupado em disco pela base de dados

Tabela	Espaço no Disco
Cliente	$12*333 = 3996$
Funcionário	$29*392 = 11368$
Bilhete	$32*22 = 704$
Viagem	$65*7 = 455$
Estação	$9*523 = 4707$
Veículo	$12*29 = 348$
EstaçãoVeículo	$130*35 = 4550$
FuncionarioVeiculo	$26 * (30 + L) \leq 338 + 26 * L$
ViagemVeiculo	$65*25 = 1625$
ClienteRealizaViagem	$65*8 = 520$
ClienteContacto	$15*8 = 120$
FuncionarioContacto	$27*8 = 216$
Total	$\leq 28947 + 26L$

Considerando o povoamento da base de dados e que este veio dos dados de utilização do metro durante uma semana temos que uma estimativa de quanto espaço o nosso modelo ocupa no disco de: $28947 + 26L$ Bytes. Com estes dados conseguimos extrapolar que ao final de um ano teríamos um espaço ocupado em disco de: $1505244 + 1352L$ Bytes.

Segundo a empresa Metro do Porto houve um aumento de **51.6%**[3] de validações face ao ano anterior por isso esperamos um crescimento de volume de dados similar sem considerar os aumentos de infraestruturas e funcionários inerentes com este aumento de volume de clientes por isso esta percentagem é para considerar uma projeção estimada.

5.5 Indexação do Sistema de Dados

De modo a melhorar algumas funções de procura e visualização, foram criados alguns índices nas tabelas *Funcionario*, *Bilhete* e *Viagem* de acordo com a figura seguinte:

```

-- Criação de indices em funcionario
Create Index index_Func On Funcionario (Cargo);

-- Criação de indices em Bilhete
Create Index index_Bilhete On Bilhete (tipo);

-- Criação de indices em Viagens
Create Index index_viagem On Viagem (Data);

```

Figura 5.21: Criação de Indexação do sistema

O objetivo destas indexações é otimizar os procedimentos que envolvem os atributos *Cargo* de um *Funcionario*, *tipo* de um *Bilhete* e *Data* de uma *Viagem*.

5.6 Procedimentos Implementados(alguns exemplos)

Para garantir todos os requisitos do cliente foram criados procedimentos, tanto para inserção de dados no sistema como para consulta. Alguns dos procedimentos implementados:

Procedimento para adicionar um cliente ao sistema, usado pelos utilizadores *PostoVenda* e *RespEstação*.

```

-- Adiciona um utilizador à base de dados.
-- Drop Procedure addCliente
delimiter $$

Create Procedure addCliente(In Nif_Cliente Int, In Nome_Cliente Varchar(75),
                           In Nasc_Cliente Date, In Genero_Cliente char(1),
                           In Rua_Cliente Varchar(100), In Localidade_Cliente Varchar(30),
                           In CP_Cliente Varchar(8), In Email_Cliente Varchar(100),
                           In Contacto_Cliente int)

Begin
Start Transaction;
Set autocommit = 0;

-- Criar o cliente
Insert Into Cliente(Contribuinte, Nome, DataNascimento, Genero, Rua, Localidade, CodigoPostal, Email)
values (Nif_Cliente, Nome_Cliente, Nasc_Cliente, Genero_Cliente, Rua_Cliente, Localidade_Cliente, CP_Cliente, Email_Cliente);
-- Adicionar Contacto
Insert Into ClienteContactos(Cliente, Contacto)
Values (Nif_Cliente, Contacto_Cliente);

Commit;
end $$
delimiter ;

```

Figura 5.22: Procedimento de adição de um cliente no sistema.

Procedimento para verificar qual o horario que um certo veículo passou em uma estação.

```
-- Ver quais horas passaram veiculos numa Estação
delimiter $$

Create Procedure FPVerHoraVeiculo(In Id_Estacao int)
Begin
    Select E.IDEstação AS 'Id Estação', E.NOME AS 'Nome Estação', F.HORA AS 'Hora', F.Veiculo AS 'Veiculo'
    From Estação AS E Right outer Join EstaçãoVeiculo AS F
        On E.IDEstação = F.Estação
        Where F.Estação = Id_Estacao AND F.HORA IS NOT NULL
        order by F.HORA DESC;
end $$

delimiter ;
```

Figura 5.23: Procedimento para verificar qual o horário que passam certos veiculos

Procedimento para visualizar a informação de uma Viagem com o nome da respetiva estação de origem e destino.

```
-- saber qual foi o cliente que realizou uma viagem e qual a origem e destino do cliente introduzindo o ID da viagem.
delimiter $$

Create Procedure infoViagem(In Id_Viagem Int)
Begin
    Select V.IdViagem as 'Id Viagem', C.Contribuinte AS 'Contribuinte Cliente', C.Nome AS 'Nome Cliente',
    Origem.Nome AS 'Origem', Destino.Nome AS 'Destino', V.Data As 'Data Realização'
    From Viagem V
    Join Estação Origem On V.Origem = Origem.IdEstação
    Join Estação Destino On V.Destino = Destino.IdEstação
        Inner Join clienteRealizaViagem On clienteRealizaViagem.Viagem = V.IdViagem
        Inner Join cliente As C On C.Contribuinte = clienterealizaviagem.Cliente
    Where V.IdViagem = Id_Viagem;
End $$

delimiter ;
```

Figura 5.24: Procedimento para visualizar a informação de uma Viagem específica.

Procedimento para registar informação de manutenção em veículos, usados pelos técnicos da manutenção.

```
-- Regista uma manutenção a um veiculo
-- Drop Procedure registaManutencaoVeiculo
delimiter $$

Create Procedure registaManutencaoVeiculo(In Id_Veiculo Varchar(20), In Data_Manutencao DATE, In Id_Func Int, In Observ TEXT)
Begin
    Start Transaction;
    Set autocommit = 0;
    Insert Into funcionarioveiculo(Funcionario, Veiculo, Data, Observações)
        Values (Id_Func, Id_Veiculo, Data_Manutencao, Observ);
    commit;
end $$

delimiter ;
```

Figura 5.25: Procedimento para registar informação de manutenção de veículos.

Os restantes procedimentos encontram-se em *Anexo*.

5.7 Definição e caracterização de mecanismos de segurança e controlo

Para garantir a segurança e integridade da base de dados, foram estabelecidos a criação de alguns utilizadores como *Cliente*, *Posto Venda*, *Responsável Estação*, *Técnico de manutenção* e *Administrador*. Para cada um foi-lhes a atribuir os respetivos privilégios de manipulação do sistema conforme os requisitos de controlo levantados, como podemos observar nas seguintes figuras:

```
USE belometrodb;

-- Criação de grupo de privilégios

-- Drop Role `Administrador`;
CREATE ROLE IF NOT EXISTS Administrador;

-- Drop Role `PostoVenda`;
CREATE ROLE IF NOT EXISTS PostoVenda;

-- Drop Role RespEstacao;
CREATE ROLE IF NOT EXISTS RespEstacao;

-- Drop Role TecnicoManutencao;
CREATE ROLE IF NOT EXISTS TecnicoManutencao;

-- Drop Role Cliente;
CREATE ROLE IF NOT EXISTS Cliente;

-- Privilegios do grupo de Administradores

-- Show grants for Administrador;
GRANT ALL PRIVILEGES
ON `belometrodb`.* 
TO Administrador
WITH GRANT OPTION;
```

Figura 5.26: Criação de grupos de privilégios e privilégios de Administrador

```

-- Show grants for PostoVenda;
Grant Select
  on `belometrodb`.Verclientes
  to PostoVenda;

Grant Select
  on `belometrodb`.verclientecontacto
  to PostoVenda;

Grant Select
  on `belometrodb`.verbilhetes
  to PostoVenda;

Grant Select
  on `belometrodb`.verpassagemveiculo
  to PostoVenda;

grant execute on procedure `belometrodb`.FPInfoBilhete to PostoVenda;
grant execute on procedure `belometrodb`.FPVerEstacao to PostoVenda;
grant execute on procedure `belometrodb`.FPVerHoraVeiculo to PostoVenda;
grant execute on procedure `belometrodb`.infoViagem to PostoVenda;
grant execute on procedure `belometrodb`.PVClientePorNif to PostoVenda;
grant execute on procedure `belometrodb`.PVContactoCliente to PostoVenda;
grant execute on procedure `belometrodb`.tipoBilheteCliente to PostoVenda;
grant execute on procedure `belometrodb`.addCliente to PostoVenda;
grant execute on procedure `belometrodb`.addClienteContacto to PostoVenda;
grant execute on procedure `belometrodb`.registaVenda to PostoVenda;
grant Select on belometrodb.verEstações to PostoVenda;

```

Figura 5.27: Privilégios do grupo *PostoVenda*

```
-- Privilegios do grupo RespEstacao
-- Show grants for RespEstacao;

Grant execute on procedure `belometrodb`.REFuncPorEstacao to RespEstacao;
Grant execute on procedure `belometrodb`.REContactoFuncionario to RespEstacao;
Grant execute on procedure `belometrodb`.REFuncionarioPorNif to RespEstacao;
Grant execute on procedure `belometrodb`.addEstacao to RespEstacao;
Grant execute on procedure `belometrodb`.cancelaVenda to RespEstacao;
Grant execute on procedure `belometrodb`.removeCliente to RespEstacao;
Grant Select on `belometrodb`.verfuncionarios to RespEstacao;
Grant Select on `belometrodb`.verfuncionariocontacto to RespEstacao;
grant Select on belometrodb.verEstacoes to RespEstacao;
```

Figura 5.28: Privilégios do grupo *RespEstacao*

```
-- Privilegios do grupo TecnicoManutencao
-- Show grants for TecnicoManutencao

grant execute on procedure `belometrodb`.`registaManutencaoVeiculo` to TecnicoManutencao;
grant execute on procedure `belometrodb`.addVeiculo to TecnicoManutencao;
grant execute on procedure `belometrodb`.removeVeiculo to TecnicoManutencao;
grant execute on procedure `belometrodb`.registaPassagemEstacao to TecnicoManutencao;
```

Figura 5.29: Privilégios do grupo *TecnicoManutencao*

```
-- Privilegios do grupo Cliente
-- Show grants for Cliente

grant Select on belometrodb.verpassagemveiculo to Cliente;
grant Select on belometrodb.verEstacoes to Cliente;
```

Figura 5.30: Privilégios do grupo *Cliente*

```

-- Criação de Utilizadores

-- Show grants for 'administrador1'@'localhost' USING Administrador;
-- Drop user 'administrador1'@'localhost';
Create user if not exists 'administrador1'@'localhost'
    identified by '1234' default role Administrador;

-- Show grants for 'postovenda'@'localhost' USING PostoVenda;
-- Drop user 'postovenda'@'localhost';
Create user if not exists 'postovenda'@'localhost'
    identified by '1234' default role PostoVenda;

-- Show grants for 'respEstacao'@'localhost' USING RespEstacao, PostoVenda;
-- Drop user 'respEstacao'@'localhost';
Create user if not exists 'respEstacao'@'localhost'
    identified by '1234' default role RespEstacao, PostoVenda;

-- Show grants for 'tecnicomantencao'@'localhost' Using TecnicoManutencao;
-- Drop user 'tecnicomantencao'@'localhost';
Create user if not exists 'tecnicomantencao'@'localhost'
    identified by '1234' default role TecnicoManutencao;

-- Show grants for 'cliente'@'localhost' Using Cliente;
-- Drop user 'cliente'@'localhost';
Create user if not exists 'cliente'@'localhost'
    identified by '1234' default role Cliente;

```

Figura 5.31: Criação dos utilizadores

5.8 Plano de segurança e recuperação de dados

Para o plano de segurança, Abílio Inácio achou que a melhor forma seria ter um backup da sua Base de Dados com o intuito de salvaguarda, caso algum erro pudesse acontecer e pôr em causa o seu projeto.

```

import subprocess

host = 'localhost'
usuario = 'root'
senha = '12345'
nome_base_dados = 'belometrodb'
arquivo_backup = 'belometroBackup.sql'

def faz_backup(host, usuario, senha, nome_base_dados, arquivo_backup):
    comandomysqldump = f'mysqldump --routines -h {host} -u {usuario} -p{senha} {nome_base_dados} > {arquivo_backup}'
    subprocess.run(comandomysqldump, shell=True)
    print(f'O backup da base de dados {nome_base_dados} foi criado com sucesso em {arquivo_backup}.')
#def drop_base_dados(host, usuario, senha, nome_base_dados):
#    comando_drop = f'DROP DATABASE {nome_base_dados}'
#    subprocess.run(['mysql', '-h', host, '-u', usuario, '-p' + senha, '-e', comando_drop])
#    print(f'A base de dados {nome_base_dados} foi removida com sucesso.')
faz_backup(host, usuario, senha, nome_base_dados, arquivo_backup)
#drop_base_dados(host, usuario, senha, nome_base_dados)

```

Figura 5.32: Script em *python* do Backup

No script utilizamos o comando *mysqldump* que guarda as informações das tabelas e as suas rotinas, tais como, *Procedure*, *Functions* em um ficheiro *.sql*.

```

import subprocess
import mysql.connector

config = {
    'host': 'localhost',
    'user': 'root',
    'password': '12345'
}

host = 'localhost'
usuario = 'root'
senha = '12345'
nome_base_dados = 'belometrodb'
arquivo_backup = 'belometroBackup.sql'

def execute_backup(host, usuario, senha, nome_base_dados, arquivo_backup):
    comando_mysql = f'mysql -h {host} -u {usuario} -p{senha} {nome_base_dados} < {arquivo_backup}'

    subprocess.run(comando_mysql, shell=True)

    print(f'O backup da base de dados {nome_base_dados} foi executado com sucesso em {arquivo_backup}.')
    print(f'A Base de Dados '{nome_base_dados}' foi criado com sucesso!']

    execute_backup(host, usuario, senha, nome_base_dados, arquivo_backup)

```

Figura 5.33: Script em *python* da execução do Backup

Para executar o Backup que foi guardado utilizamos outro script que primeiramente conecta-se ao servidor em que queremos guardar as informações da Base de Dados, assegurar criamos a própria Base de Dados antes de executar o ficheiro Backup, e por fim utilizamos o comando *mysql* para executar o Backup onde estão todas as tabelas guardadas.

6 Implementação do Sistema de Recolha de Dados

6.1 Apresentação e modelo do sistema

Para realizarmos a coleta de dados para a nossa base de dados, achou-se que a melhor forma seria utilizar os ficheiros *csv* para guardar as informações de cada entidade, e com esses ficheiros utilizarmos *scripts* em *python*, para os ler e enviar as informações para as suas respetivas tabelas.

6.2 Implementação do sistema de recolha

```
import mysql.connector
import csv
import codecs

config = {
    'host': 'localhost',
    'user': 'root',
    'password': '12345',
    'database': 'belometrodb',
    'charset': 'utf8mb4'
}

conn = mysql.connector.connect(**config)
```

Figura 6.1: Ligação ao servidor da Base de Dados

Primeiramente, estabelecemos ligação ao nosso servidor do *Mysql Workbench*, utilizando as credenciais respetivas da sua base de dados.

```

conn = mysql.connector.connect(**config)

mycursor = conn.cursor()

with codecs.open('Estações.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3], row[4])
        all_values.append(value)

query = "insert into estação (`Nome`, `Longitude`, `Latitude`, `PontosInteresse`, `Descrição`) values (%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

```

Figura 6.2: Leitura do ficheiro "Estações.csv" e implementação da mesma na Base de Dados

De seguida, executamos a leitura do ficheiro em relação às estações em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Estação e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('Clientes.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7])
        all_values.append(value)

query = "insert into cliente ('Contribuinte', 'Nome', 'DataNascimento', 'Genero', 'Rua', 'Localidade', 'CodigoPostal', 'email') values (%s,%s,%s,%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

```

Figura 6.3: Leitura do ficheiro "Clientes.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos clientes em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Cliente e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('ContactosClientes.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1])
        all_values.append(value)

    query = "insert into clientecontactos (`Cliente`, `Contacto`) values (%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()

```

Figura 6.4: Leitura do ficheiro "ContactosClientes.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos contactos dos clientes em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade ClientesContactos e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('Funcionarios.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3], row[4], row[5], row[6], row[7], row[8], row[9], row[10], row[11])
        all_values.append(value)

    query = "insert into funcionario ('Contribuinte', 'Nome', 'DataNascimento', 'Genero', 'Cargo', 'Salario', 'IBAN', 'Rua', 'Localidade', 'CodigoPostal', 'email', 'Estação') values (%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()

```

Figura 6.5: Leitura do ficheiro "Funcionarios.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos funcionários em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Funcionário e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('ContactosFuncionarios.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1])
        all_values.append(value)

    query = "insert into funcionariocontactos (`Funcionario`, `Contacto`) values (%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()

```

Figura 6.6: Leitura do ficheiro "ContactosFuncionarios.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos contactos dos funcionários em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade ContactosFuncionarios e no final fazemos *commit* para a Base de Dados.

```
with codecs.open('Veiculos.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2])
        all_values.append(value)

    query = "insert into veiculo (`IdVeiculo`, `NrCarruagens`, `NrLugares`) values (%s,%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()
```

Figura 6.7: Leitura do ficheiro "Veiculos.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos veículos em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Veiculo e no final fazemos *commit* para a Base de Dados.

```
with codecs.open('Bilhetes.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3], row[4], row[5], row[6])
        all_values.append(value)

    query = "insert into bilhete (`FormaPagamento`, `Tipo`, `DataVenda`, `DataValidade`, `Preço`, `Funcionario`, `Cliente`) values (%s,%s,%s,%s,%s,%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()
```

Figura 6.8: Leitura do ficheiro "Bilhetes.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos bilhetes em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Bilhete e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('Viagens.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3])
        all_values.append(value)

    query = "insert into viagem (`Data`, `Bilhete`, `Origem`, `Destino`) values (%s,%s,%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()

```

Figura 6.9: Leitura do ficheiro "Viagens.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação às viagens em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade Viagem e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('ViagemVeiculo.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1])
        all_values.append(value)

    query = "insert into viagemveiculo (`Viagem`, `Veiculo`) values (%s,%s)"
    mycursor.executemany(query, all_values)
    conn.commit()

```

Figura 6.10: Leitura do ficheiro "ViagemVeiculo.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação às veículos que realizam as viagens em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade ViagemVeiculo e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('FuncionariosVeiculos.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3])
        all_values.append(value)

query = "insert into funcionarioveiculo (`Funcionario`, `Veiculo`, `Data`, `Observações`) values (%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

```

Figura 6.11: Leitura do ficheiro "FuncionariosVeiculo.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos funcionários que trabalham em respetivas estações em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade FuncionarioVeiculo e no final fazemos *commit* para a Base de Dados.

```

with codecs.open('EstaçõesVeículos.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2])
        all_values.append(value)

query = "insert into estaçãoveiculo (`Hora`, `Veiculo`, `Estação`) values (%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()
conn.close()

```

Figura 6.12: Leitura do ficheiro "EstaçõesVeiculo.csv" e implementação da mesma na Base de Dados

Executamos a leitura do ficheiro em relação aos veículos que passam em certas estações a uma certa hora em que lemos as suas respetivas colunas e implementamos nos respetivos atributos da entidade EstaçãoVeiculo e no final fazemos *commit* para a Base de Dados.

6.3 Funcionamento do sistema

Executamos o script numa bash em que automaticamente executa o código anteriormente mencionado e envia para a respetiva Base de Dados em que fizemos a conexão.

7 Implementação do Sistema de Painéis de Análise

7.1 Definição e caracterização da vista de dados para análise

O Power BI consiste no processo de idealizar e configurar o aspeto visual de um painel. É a etapa que definimos a visualização, gráficos e tabelas que serão usadas para comunicar as informações de maneira mais clara e eficaz aos utilizadores finais. Algumas das características e vantagens do Power BI são:

- Escolha dos aspetos visuais mais adequados
- Organização
- Personalização e formatação
- Interatividade e filtros

7.2 Povoamento das estruturas de dados para análise

O povoamento das estruturas de dados para análise refere-se ao processo de carregar os dados relevantes numa base de dados para dentro da plataforma do Power BI. E para tal, seguimos as seguintes etapas:

- **Conexão dos dados:** conectar a base de dados à plataforma Power BI.
- **Transformação dos dados:** a possibilidade de limpar, filtrar e manipular os dados antes de carregá-los.
- **Modelagem dos dados:** consiste na definição de estruturas no Power BI, ou seja, a criação de tabelas e relacionamentos entre tabelas, de modo a organizarmos os dados conforme as nossas necessidades.
- **Carregamento de dados:** após feitos os passos anteriores, finalmente podemos criar

os painéis interativos, onde podemos carregar os dados necessários de modo a conseguirmos visualizar o que pretendemos.

7.3 Apresentação e caracterização dos dashboards implementados

Através do Power BI implementamos os seguintes dashboards:

Total auferido ao longo dos anos

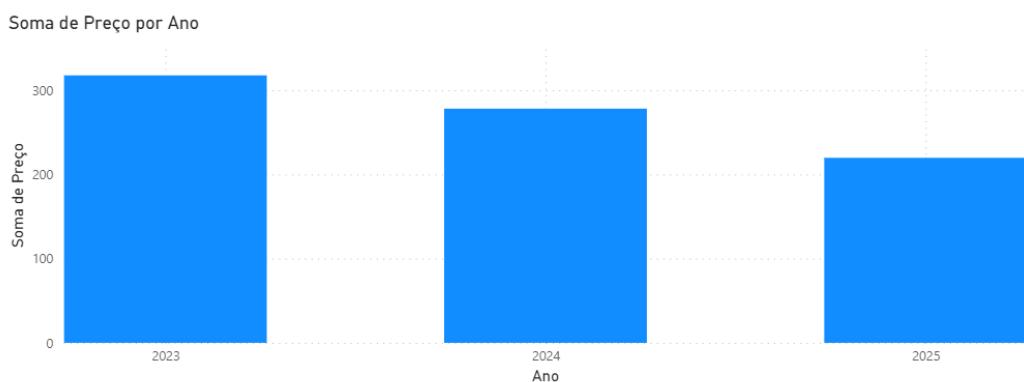


Figura 7.1: Total auferido ao longo dos anos

Contagem de veículos por estação

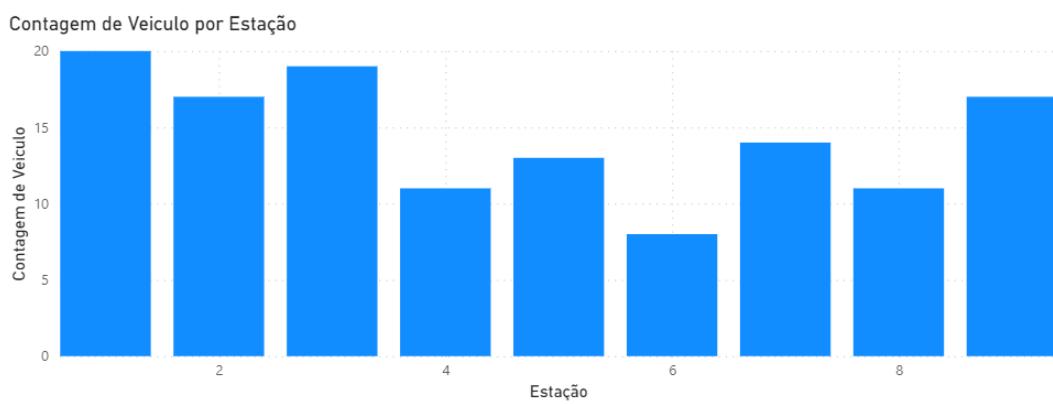


Figura 7.2: Contagem de veículos por estação

Estações - Mapa

Primeiro PontosInteresse por Nome, Latitude e Longitude

Nome ● Estação de C... ● Estação de E... ● Estação de S... ● Estação de ... ● Estação d... ● Estação d... ● Estação d... ● Estação d... ● Estação d...

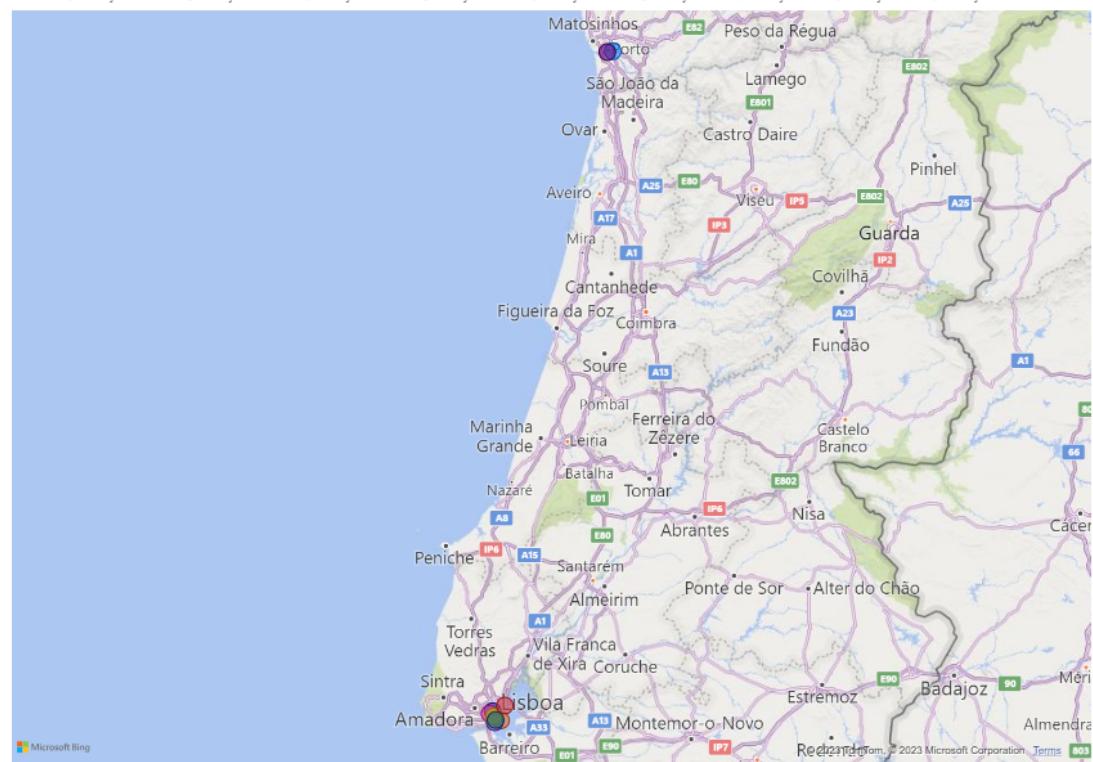


Figura 7.3: Estações - Mapa

Percentagem de salários

Mínimo de Salario e Máximo de Salario por cargo

● Mínimo de Salario ● Máximo de Salario

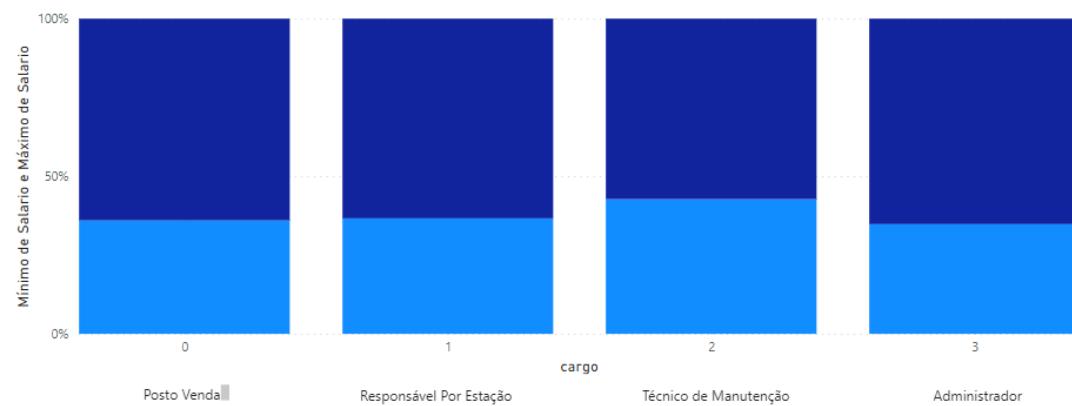


Figura 7.4: Percentagem de salários

8 Conclusões e trabalho futuro

Este projeto foi desenvolvido a pedido do senhor Abílio Inácio que comprou a empresa de metro da sua terra, a "Belo Metro".

Para iniciar o desenvolvimento do projeto da BD, foi elaborada uma sessão junto do cliente de modo a compreender o funcionamento do sistema e os requisitos que integravam o mesmo. Uma vez terminada a reunião, a equipa de desenvolvimento elaborou uma listagem dos requisitos da BD, tendo estes sido revistos, posteriormente, com o cliente.

De seguida, procedemos com a construção do modelo conceptual e lógico, tendo estes sido apresentados e validados pelo cliente em sessões de acompanhamento para a equipa demonstrar o desenvolvimento do projeto ao cliente. Com estes modelos a equipa obteve uma compreensão mais nítida de como a base de dados seria desenvolvida de forma a responder com eficiência às demandas do sistema de metro.

Por último, realizou-se a construção física da base de dados, sendo escolhido o *software mySQL* como sistema de gestão de base de dados.

Alguns pontos a melhorar no projeto que a equipa sugere são uma melhoria nos painéis de análise do *Power BI*, reestruturação da tabela EstaçãoVeículo em vez de registar a hora de quando um veículo passa na estação ter um horário das horas previstas de passagem de todos os veículos que irão passar naquela estação e por fim algumas otimizações das sistemas em *SQL*.

No futuro, a equipa sugere a verificação periódica da BD, de forma a compreender possíveis alterações necessárias a serem feitas de forma a acompanhar o crescimento do sistema de metro. Em adição, também é sugerida a manutenção e acompanhamento dos sistemas físicos que suportam a base de dados, para garantir que funcionam corretamente e serão suficientes para garantir o correto funcionamento da base de dados.

Referências

- [1] dev.mysql.com, 2018, *MySQL 8.0 Reference Manual*,
<https://dev.mysql.com/doc/refman/8.0/en/>
- [2] w3schools.com, 2018, *SQL Data Types for MySQL, SQL Server, and MS Access*,
https://www.w3schools.com/sql/sql_datatypes.asp
- [3] Porto.pt, 2023, *Metro do Porto regista em 2022 uma média de 178 mil validações por dia*, <https://www.porto.pt/pt/noticia/metro-do-porto-regista-em-2022-uma-media-de-178-mil-validacoes-por-dia>

Lista de Siglas e Acrónimos

«Apresentar uma lista com todas as siglas e acrónimos utilizados durante a realização do trabalho. O formato base para esta lista deverá ser da forma como abaixo se apresenta.»

BD Base de Dados

DW Data Warehouse

OLTP On-Line Analytical Processing

... ...

Anexos

1. Script de criação da base de dados
2. Script parcial de povoamento
3. Queries em SQL
4. Funções usadas em SQL
5. Script de Backup
6. Script de criação do dataset

Anexo 1 - Script de criação da base de dados

```
— Schema BeloMetroDB
```

```
— Drop Schema BeloMetroDB;
```

```
Create Schema if not exists 'BeloMetroDB'  
    default charset=utf8mb4;  
Use BeloMetroDB;
```

```
— Tabela Clientes
```

```
— Drop Table 'Cliente'  
CREATE TABLE if not exists 'BeloMetroDB'.'Cliente' (  
    'Contribuinte' INT unsigned NOT NULL check ('  
        Contribuinte' between 100000000 and 999999999),  
    'Nome' varchar(75) NOT NULL,  
    'DataNascimento' DATE,  
    'Genero' char,  
    'Rua' varchar(100) NOT NULL,  
    'Localidade' varchar(30) NOT NULL,
```

```
    'CodigoPostal' varchar(8) NOT NULL,
    'email' varchar(100),
        primary key('Contribuinte')
    )
engine = InnoDB;
```

— Tabela ClienteContactos

```
Create table if not exists 'BeloMetroDB'.'ClienteContactos' (
    'Cliente' INT unsigned NOT NULL,
    'Contacto' INT unsigned unique NOT NULL check ('Contacto' between 100000000 and 999999999),
    Primary key('Cliente', 'Contacto'),
    Index 'fk_ClienteTelefone_Cliente_idx' ('Cliente' ASC),
    Constraint 'fk_ClienteTelefone_Cliente'
        Foreign key ('Cliente')
        References 'BeloMetroDB'.'Cliente' (
            'Contribuinte')
    ON DELETE NO ACTION
    ON UPDATE NO ACTION)
Engine = InnoDB;
```

— Tabela Estação

— Drop table 'Estação'

```
Create Table If Not Exists 'BeloMetroDB'.'Estação' (
    'IdEstação' INT unsigned auto_increment unique NOT NULL,
    'Nome' varchar(50) NOT NULL,
    'Longitude' Double NOT NULL,
    'Latitude' Double NOT NULL,
    'PontosInteresse' Varchar(150) NOT NULL,
    'Descrição' TEXT NOT NULL,
    Primary Key('IdEstação'))
Engine = InnoDB;
```

— Tabela Funcionario

— Drop Table 'Funcionario'

```
Create Table 'BeloMetroDB'.'Funcionario' (
```

```

    'Contribuinte' INT unsigned NOT NULL check ('Contribuinte' between 100000000 and 99999999),
    'Nome' varchar(75) NOT NULL,
    'DataNascimento' DATE,
    'Genero' char,
    'Rua' varchar(100) NOT NULL,
    'Localidade' varchar(30) NOT NULL,
    'CodigoPostal' varchar(8) NOT NULL,
    'email' varchar(100),
    'cargo' INT NOT NULL check ('cargo' between 0 and 3),
    'Salario' DECIMAL(6,2) NOT NULL,
    'IBAN' varchar(25) NOT NULL,
    'Estação' Int unsigned Not NULL,
        primary key('Contribuinte'),
    Constraint 'fk_Funcionario_Estação'
        Foreign Key('Estação')
        references 'BeloMetroDB'.'Estação' ('IdEstação')
        On Delete No Action
        On Update No Action
);

```

— Tabela ClienteContactos

— Drop table 'FuncionarioContactos'

```

Create table if not exists 'BeloMetroDB'.'FuncionarioContactos'
(
    'Funcionario' INT unsigned NOT NULL,
    'Contacto' Int unsigned NOT NULL check ('Contacto' between
        100000000 and 99999999),
    Primary key('Funcionario', 'Contacto'),
    Index 'fk_FuncionarioContacto_Funcionario_idx' ('Funcionario
        ' ASC),
    Constraint 'fk_FuncionarioContacto_Funcionario'
        Foreign key ('Funcionario')
        References 'BeloMetroDB'.'Funcionario' (
            'Contribuinte')
        ON DELETE NO ACTION
        ON UPDATE NO ACTION)
Engine = InnoDB;

```

— Tabela Bilhete

```

— Drop table 'Bilhete'

Create Table if not exists 'BeloMetroDB'.'Bilhete' (
    'NrVenda' Int unsigned unique auto_increment NOT NULL,
    'FormaPagamento' int NOT NULL check ('FormaPagamento'
        between 0 and 1),
    'Tipo' int NOT NULL check ('Tipo' between 0 and 1),
    'DataVenda' DATE NOT NULL,
    'DataValidade' DATE NOT NULL,
    'Preço' DECIMAL(5,2) NOT NULL,
    'Funcionario' INT unsigned NOT NULL,
    'Cliente' INT unsigned NOT NULL,
    primary key('NrVenda'),
    constraint 'fk_Bilhete_Funcionario'
        foreign key('Funcionario')
        references 'BeloMetroDB'.'Funcionario' ('Contribuinte')
        On delete no action
        ON update no action,
    constraint 'fk_Bilhete_Cliente'
        foreign key('Cliente')
        references 'BeloMetroDB'.'Cliente' ('Contribuinte')
        ON delete no action
        on update no action)
Engine = InnoDB;

```

— Tabela Viagem

— Drop table 'Viagem'

```

Create Table If not Exists 'BeloMetroDB'.'Viagem' (
    'IdViagem' Int unsigned auto_increment unique NOT NULL,
    'Bilhete' Int unsigned NOT NULL,
    'Origem' Int unsigned NOT NULL,
    'Destino' Int unsigned NOT NULL,
    'Data' DATE NOT NULL,
    primary key('IdViagem'),
    Constraint 'fk_Viagem_Bilhete'
        foreign Key('Bilhete')
        References 'BeloMetroDB'.'Bilhete' ('NrVenda')
        ON DELETE NO ACTION
        ON UPDATE NO ACTION,
    Constraint 'fk_ViagemEstação_Origem'
        Foreign Key('Origem')
        References 'BeloMetroDB'.'Estação' ('IdEstação')
        ON DELETE NO ACTION

```

```

        ON UPDATE NO ACTION,
        Constraint 'fk_ViagemEstação_Destino'
            Foreign Key('Destino')
            References 'BeloMetroDB'.'Estação' ('IdEstação')
            ON DELETE NO ACTION
        ON UPDATE NO ACTION
    )
engine = InnoDB;

```

— Tabela ClienteRealizaViagem

```

— Drop table 'ClienteRealizaViagem'

Create Table If Not Exists 'BeloMetroDB'.'ClienteRealizaViagem'
(
    'Cliente' Int unsigned NOT NULL,
    'Viagem' Int unsigned unique NOT NULL,
    Primary Key('Cliente', 'Viagem'),
    constraint 'fk_ClienteRealizaViagem_Cliente'
        foreign key ('Cliente')
        references 'BeloMetroDB'.'Cliente' ('Contribuinte')
    On Delete No Action
    On Update No Action,
    Constraint 'fk_ClienteRealizaViagem_Viagem'
        foreign key ('Viagem')
        References 'BeloMetroDB'.'Viagem' ('IdViagem')
    On Delete No Action
    On Update No Action)
Engine = InnoDB;

```

— Tabela Veiculo

```

— Drop table 'Veiculo'

Create Table If Not Exists 'BeloMetroDB'.'Veiculo' (
    'IdVeiculo' varchar(20) Unique Not Null,
    'NrCarruagens' Int Unsigned Not Null,
    'NrLugares' Int Unsigned Not Null,
    Primary Key ('IdVeiculo'))
Engine = InnoDB;

```

— Tabela ViagemVeiculo

```
— Drop Table ‘ViagemVeiculo’

Create Table If Not Exists ‘BeloMetroDB’.’ViagemVeiculo’(
    ‘Viagem’ Int unsigned NOT NULL,
    ‘Veiculo’ varchar(20) NOT NULL,
    primary key (‘Viagem’, ‘Veiculo’),
    constraint ‘fk_ViagemVeiculo_Viagem’
        foreign key (‘Viagem’)
            References ‘BeloMetroDB’.’Viagem’ (‘IdViagem’)
            On Delete No action
            On Update No action,
    constraint ‘fk_ViagemVeiculo_Veiculo’
        foreign key(‘Veiculo’)
            References ‘BeloMetroDB’.’Veiculo’ (‘IdVeiculo’)
            On Delete No Action
            On Update No Action)
Engine = InnoDB;
```

```
— Tabela EstaçãoVeiculo
```

```
— Drop Table ‘EstaçãoVeiculo’

Create Table If Not Exists ‘BeloMetroDB’.’EstaçãoVeiculo’(
    ‘Estação’ Int unsigned NOT NULL,
    ‘Veiculo’ varchar(20) NOT NULL,
    ‘Hora’ DATETIME NOT NULL,
    primary key (‘Estação’, ‘Veiculo’, ‘Hora’),
    constraint ‘fk_EstaçãoVeiculo_Estação’
        foreign key (‘Estação’)
            References ‘BeloMetroDB’.’Estação’ (‘IdEstação’)
            On Delete No action
            On Update No action,
    constraint ‘fk_EstaçãoVeiculo_Veiculo’
        foreign key(‘Veiculo’)
            References ‘BeloMetroDB’.’Veiculo’ (‘IdVeiculo’)
            On Delete No Action
            On Update No Action)
Engine = InnoDB;
```

```
— Tabela FuncionarioVeiculo
```

```

— Drop Table 'FuncionarioVeiculo'

Create Table If Not Exists 'BeloMetroDB'.'FuncionarioVeiculo'(
    'Funcionario' Int unsigned NOT NULL,
    'Veiculo' Varchar(20) NOT NULL,
    'Data' Date NOT NULL,
    'Observações' TEXT NOT NULL,
    primary key ('Funcionario', 'Veiculo', 'Data'),
    constraint 'fk_FuncionarioVeiculo_Funcionario'
        foreign key ('Funcionario')
        References 'BeloMetroDB'.'Funcionario' ('Contribuinte')
        On Delete No action
        On Update No action,
    constraint 'fk_FuncionarioVeiculo_Veiculo'
        foreign key('Veiculo')
        References 'BeloMetroDB'.'Veiculo' ('IdVeiculo')
        On Delete No Action
        On Update No Action)
Engine = InnoDB;

```

Anexo 2 - Script parcial de povoamento

```

— Esquema: "BeloMetroDB"
USE 'BeloMetroDB';

— Permissão para fazer operações de remoção de dados.
SET SQL_SAFE_UPDATES = 0;

— Povoamento da Tabela 'Estação'

— Select * FROM Estação
— Delete from Estação

INSERT INTO Estação
    (Nome, Longitude, Latitude, PontosInteresse, Descrição)
Values
    ("Estação do Rossio", "-9.141814", "38.714330",
     "Arquitetura, História", "A Estação do Rossio
     , também conhecida como Estação Ferroviária
     de Lisboa, é uma das estações mais
     emblemáticas da cidade. Sua arquitetura
     deslumbrante e história rica atraem
     visitantes de todo o mundo."),
    ("Estação de São Bento", -8.611997, 41.145862,
     "Azulejos, Patrimônio", "A Estação de São
     Bento, também conhecida como Estação
     Ferroviária de São Bento, é uma das
     estações mais emblemáticas da cidade.
     Sua arquitetura deslumbrante e história
     rica atraem visitantes de todo o mundo.")

```

- Bento é famosa por seus magníficos painéis de azulejos, que retratam cenas históricas de Portugal. É considerada uma das mais belas estações de trem do mundo."),
- ("Estação de Campanhã", -8.585926, 41.148643, "Modernidade, Conexões", "A Estação de Campanhã é uma estação ferroviária moderna e movimentada, que serve como um importante hub de transporte na região do Porto. Possui várias conexões para outras cidades e países."),
- ("Estação do Cais do Sodré", -9.145870, 38.705398, "Vista para o rio, Transporte marítimo", "Localizada perto do Rio Tejo, a Estação do Cais do Sodré oferece uma bela vista para o rio. Além dos trens, também é um importante ponto de partida para viagens de barco."),
- ("Estação do Oriente", -9.098002, 38.767738, "Arquitetura futurista, Comércio", "Projetada pelo renomado arquiteto Santiago Calatrava, a Estação do Oriente é um exemplo impressionante de arquitetura moderna. É parte de um complexo que inclui um grande centro comercial."),
- ("Estação de Santa Apolónia", -9.116125, 38.712793, "História, Terminal de cruzeiros", "A Estação de Santa Apolónia é uma das estações de trem mais antigas de Lisboa, com uma história que remonta ao século XIX. Também serve como terminal de cruzeiros para navios que visitam a cidade."),
- ("Estação de Entrecampos", -9.155891, 38.746857, "Conexões, Próximo ao campus universitário", "A Estação de Entrecampos é uma importante estação de trem em Lisboa, com várias conexões para outras partes da cidade e do país. Também está localizada perto do campus universitário."),
- ("Estação de Sete Rios", -9.170247, 38.737826, "Parque Zoológico, Terminal de autocarros", "A Estação de Sete Rios é conhecida por abrigar o Parque Zoológico de Lisboa. Além dos trens, também serve como um terminal de ônibus para rotas intermunicipais e internacionais."),
- ("Estação do Marquês de Pombal", -9.149510, 38.727659, "Praça do Marquês, Comércio", "Localizada perto da Praça do Marquês de Pombal, essa estação é um importante ponto de acesso ao centro de Lisboa. Também está rodeada de lojas e comércio!")
- ;

```

— Povoamento da Tabela 'Cliente'

— Select * FROM Cliente
— Delete from Cliente

INSERT INTO Cliente
( Contribuinte , Nome, DataNascimento , Genero , Rua, Localidade
, CodigoPostal , Email)
VALUES
(987654321, "Joao Silva","1985-05-12", "M", "Rua das
Flores", "Lisboa", "1000-001", "joao.silva@example.
com"),
(373954712, "Ana Santos", "1992-09-23", "F", "Rua do
Carmo", "Porto", "4000-002", "ana.santos@example.com
"),
(837264726, "Pedro Almeida", "1978-11-05", "M", "Avenida
da Liberdade", "Braga", "4700-003", "pedro.
almeida@example.com"),
(138274928, "Sofia Pereira", "1989-06-19", "F", "Rua da
Boavista", "Coimbra", "3000-004", "sofia.
pereira@example.com"),
(938372948, "Ricardo Costa", "1995-03-30", "M", "Rua do
Comércio", "Faro", "8000-005", "ricardo.costa@example.
com"),
(482719283, "Beatriz Ferreira", "1983-08-14", "F", "
Avenida da República", "Setúbal", "2900-006", "
beatriz.ferreira@example.com"),
(749384625, "André Santos", "1976-12-02", "M", "Rua de
Santa Catarina", "Évora", "7000-007", "andre.
santos@example.com"),
(583927846, "Mariana Oliveira", "1990-07-25", "F", "Rua
das Figueiras", "Viseu", "3500-008", "mariana.
oliveira@example.com"),
(383748739, "Miguel Pereira", "1982-04-10", "M", "
Avenida dos Aliados", "Guimarães", "4800-009", "
miguel.pereira@example.com"),
(498303948, "Carolina Silva", "1998-01-08", "F", "Rua
do Sol", "Portimão", "8500-010", "carolina.
silva@example.com"),
(948393848, "Tiago Rodrigues", "1987-09-17", "M", "Rua
dos Cedros", "Aveiro", "3800-011", "tiago.
rodrigues@example.com"),
(859393847, "Inês Costa", "1993-06-21", "F", "Avenida da
Boavista", "Bragança", "5300-012", "ines.
costa@example.com")
;

```

```

— Povoamento da Tabela ‘ClienteContactos’
— Select * FROM ‘ClienteContactos’
— Delete FROM ‘ClienteContactos’

INSERT INTO clientecontactos
  (Cliente, Contacto)
Values
  (987654321, '912345678') ,
  (987654321, '917733488') ,
  (373954712, '931234567') ,
  (837264726, '961234567') ,
  (138274928, '922345678') ,
  (938372948, '925678901') ,
  (938372948, '253856829') ,
  (938372948, '210495824') ,
  (482719283, '934567890') ,
  (749384625, '935678901') ,
  (583927846, '961098765') ,
  (383748739, '918765432') ,
  (498303948, '919876543') ,
  (948393848, '936543210') ,
  (859393847, '966543210')
;
— Povoamento Tabela Funcionario

— Select * FROM ‘Funcionario’
— Delete FROM ‘Funcionario’

INSERT INTO Funcionario
  (Contribuinte, Nome, DataNascimento, Genero, Cargo,
   Salario, IBAN, Rua, Localidade, CodigoPostal, Email,
   Estação)
VALUES
  (849398594, "Abilio Inacio", "1966-03-14", "M",
   3, 8500.00, "PT50000033994959030495849", "
   Praça dos Transportes", "Vila Bela",
   "5000-285", "abilio.inacio@example.com", 1),
  (748309385, "Miguel Teixeira", "1983-02-27", "M", 3,
   4540.50, "PT5000330405929495040309", "Praceta da
   Imaginação", "Braga", "4715-288", "miTeixeira@example
   .com", 1),
  (536512321, "António Pereira", "1980-08-10", "M",
   1, 2500.00, "PT50002700000001234567893", "
   Rua da Liberdade", "Porto", "4000-001", "
   antonio.pereira@example.com", 1),
  (325876932, "Maria Silva", "1992-03-25", "F", 0,

```

1800.50, "PT50002700000009876543210", "Rua dos Lírios", "Lisboa", "1000-002", "maria.silva@example.com", 1),
(867854211, "Manuel Santos", "1985-11-12", "M", 2, 3000.75, "PT50002700000024681357901", "Avenida Central", "Coimbra", "3000-003", "manuel.santos@example.com", 1),
(436876565, "Sofia Ferreira", "1990-06-05", "F", 1, 3500.25, "PT50002700000013579246802", "Praça da República", "Braga", "4700-004", "sofia.ferreira@example.com", 2),
(349712867, "Carlos Oliveira", "1988-09-28", "M", 0, 2200.00, "PT50002700000056789012345", "Rua do Comércio", "Faro", "8000-005", "carlos.oliveira@example.com", 3),
(457682430, "Ana Martins", "1995-02-15", "F", 0, 1950.75, "PT50002700000090123456789", "Avenida da Liberdade", "Évora", "7000-006", "ana.martins@example.com", 2),
(657045312, "João Sousa", "1983-07-20", "M", 2, 3200.50, "PT50002700000031415926536", "Rua das Flores", "Guimarães", "4800-007", "joao.sousa@example.com", 2),
(768906656, "Teresa Rodrigues", "1991-04-08", "F", 2, 3750.25, "PT50002700000027182818273", "Avenida dos Aliados", "Viseu", "3500-008", "teresa.rodrigues@example.com", 3),
(456679111, "Miguel Almeida", "1987-09-18", "M", 1, 2350.00, "PT5000270000007777777712", "Rua do Carmo", "Setúbal", "2900-009", "miguel.almeida@example.com", 3),
(256732490, "Inês Costa", "1993-12-06", "F", 0, 2050.75, "PT5000270000008888888829", "Avenida da Boavista", "Aveiro", "3800-010", "ines.costa@example.com", 4),
(876462100, "André Pereira", "1982-05-22", "M", 2, 3150.50, "PT50002700000099999999948", "Rua dos Cedros", "Portimão", "8500-011", "andre.pereira@example.com", 4),
(454798008, "Carolina Fernandes", "1999-01-12", "F", 1, 3800.25, "PT50002700000055555555567", "Rua Auria", "Faro", "2678-289", "carol@example.com", 4),
(839274947, "Pedro Santos", "1984-05-02", "M", 1, 2800.00, "PT50002700000012345678911", "Rua das Oliveiras", "Vila Nova de Gaia", "4400-003", "pedro.santos@example.com", 5),

(294847192, "Sofia Costa", "1991-09-15", "F", 2,
3200.50, "PT50002700000098765432120", "
Avenida dos Girassóis", "Matosinhos",
"4450-004", "sofia.costa@example.com", 5),
(246813579, "Miguel Ferreira", "1986-12-08", "M",
0, 2400.75, "PT50002700000024681357939", "
Rua da Fonte", "Braga", "4700-005", "miguel.
ferreira@example.com", 5),
(135792468, "Ana Rodrigues", "1993-06-25", "F",
1, 3800.25, "PT50002700000013579246848", "
Avenida Central", "Coimbra", "3000-006", "ana
.rodrigues@example.com", 6),
(567890123, "Diogo Oliveira", "1989-03-19", "M",
1, 2200.00, "PT50002700000056789012357", "
Rua das Flores", "Porto", "4000-007", "diogo.
.oliveira@example.com", 7),
(638263747, "Mariana Silva", "1996-08-10", "F",
2, 3100.75, "PT50002700000090123456766", "
Avenida da Liberdade", "Lisboa", "1000-008",
"mariana.silva@example.com", 6),
(234567890, "Ana Santos", "1990-03-14", "F", 0, 2500.00,
"PT50002700000023456789044", "Rua das Azáleas", "
Sintra", "2710-009", "ana.santos@example.com", 6),
(987654321, "Carlos Silva", "1985-08-27", "M",
0, 3200.50, "PT50002700000098765432111", "
Avenida dos Lírios", "Cascais", "2750-002", "
carlos.silva@example.com", 7),
(456789012, "Sofia Pereira", "1992-05-02", "F",
2, 2900.75, "PT50002700000045678901233", "Rua
do Parque", "Porto", "4000-005", "sofia.
.pereira@example.com", 7),
(329585474, "Pedro Almeida", "1988-12-19", "M",
0, 2200.25, "PT50002700000012345678922", "
Avenida Central", "Braga", "4700-004", "pedro
.almeida@example.com", 8),
(789012345, "Marta Fernandes", "1995-06-05", "F",
2, 2800.00, "PT50002700000078901234566", "
Rua das Flores", "Coimbra", "3000-007", "
marta.fernandes@example.com", 8),
(345678901, "Ricardo Santos", "1991-09-18", "M",
1, 2600.75, "PT50002700000034567890155", "
Avenida da Liberdade", "Lisboa", "1000-003",
"ricardo.santos@example.com", 8),
(901234567, "Inês Carvalho", "1987-02-25", "F",
1, 3500.50, "PT50002700000090123456799", "Rua
dos Girassóis", "Faro", "8000-006", "ines.
.carvalho@example.com", 9),

```

(678901234, "André Ferreira", "1994-07-08", "M",
 0, 2000.25, "PT5000270000067890123488", "
  Avenida Central", "Guimarães", "4800-001", "
  andre.ferreira@example.com", 9),
(432109876, "Beatriz Oliveira", "1989-11-15", "F",
 2, 2900.00, "PT5000270000043210987677", "
  Rua da Fonte", "Setúbal", "2900-004", "
  beatriz.oliveira@example.com", 9)
;

— Povoamento da Tabela 'FuncionarioContactos'
— Select * FROM 'FuncionarioContactos'
— Delete FROM 'FuncionarioContactos'

INSERT INTO FuncionarioContactos
(Funcionario, Contacto)
VALUES
(536512321, '927654321'),
(325876932, '968765432'),
(867854211, '937654321'),
(436876565, '966789012'),
(349712867, '938901234'),
(457682430, '929012345'),
(657045312, '917654321'),
(768906656, '969012345'),
(456679111, '938765432'),
(256732490, '966789012'),
(876462100, '937654321'),
(454798008, '925678901'),
(987654321, '913456789'),
(246813579, '926543210'),
(135792468, '927654321'),
(567890123, '915678901'),
(901234567, '938901234'),
(234567890, '926789012'),
(987654321, '914567890'),
(456789012, '919876543'),
(789012345, '936543210'),
(345678901, '935678901'),
(901234567, '939012345'),
(678901234, '927890123'),
(432109876, '935678901'),
(839274947, '931234567'),
(329585474, '936543210')
;

— Povoamento da Tabela 'Veiculo'

```

```

— Select * FROM 'Veiculo'
— Delete FROM 'Veiculo'

INSERT INTO Veiculo
    (IdVeiculo, NrCarruagens, NrLugares)
Values
    ("A-001", 120, 6),
    ("B-002", 80, 4),
    ("C-003", 200, 8),
    ("D-004", 150, 6),
    ("E-005", 100, 5),
    ("F-006", 180, 7),
    ("G-007", 90, 4),
    ("H-008", 130, 6),
    ("I-009", 160, 8),
    ("J-010", 110, 5),
    ("K-011", 140, 7),
    ("L-012", 170, 6)
;

— Povoamento da Tabela 'Bilhete'

— Select * FROM 'Bilhete' veiculo
— Delete FROM 'Bilhete'

Insert Into Bilhete
    (FormaPagamento, Tipo, DataVenda, DataValidade, Preço,
     Funcionario, Cliente)
values
    (1, 0, "2023-01-15", "2023-01-16", 25.50,
     325876932, 383748739),
    (0, 1, "2023-02-10", "2023-03-10", 30.80,
     325876932, 987654321),
    (1, 1, "2023-03-05", "2023-04-05", 12.75,
     325876932, 859393847),
    (1, 0, "2023-04-20", "2023-04-21", 18.90,
     457682430, 749384625),
    (0, 0, "2023-05-12", "2023-05-13", 9.99,
     457682430, 498303948),
    (1, 0, "2023-06-30", "2023-07-01", 42.30,
     457682430, 482719283),
    (1, 1, "2023-07-17", "2023-08-17", 35.25,
     349712867, 837264726),
    (0, 1, "2023-08-22", "2023-09-22", 27.60,
     349712867, 383748739),
    (1, 0, "2023-09-08", "2023-09-09", 19.95,

```

349712867, 948393848),
(0, 0, "2023-10-05", "2023-10-06", 15.60,
256732490, 948393848),
(1, 1, "2023-11-18", "2023-12-18", 50.20,
256732490, 383748739),
(1, 0, "2023-12-25", "2023-12-26", 28.70,
256732490, 948393848),
(0, 1, "2024-01-04", "2024-02-04", 32.90,
246813579, 837264726),
(1, 1, "2024-02-19", "2024-03-19", 14.35,
246813579, 373954712),
(1, 0, "2024-03-10", "2024-03-11", 21.80,
246813579, 498303948),
(0, 0, "2024-04-14", "2024-04-15", 10.50,
234567890, 859393847),
(1, 0, "2024-05-27", "2024-05-28", 16.75,
234567890, 383748739),
(1, 1, "2024-06-02", "2024-07-02", 38.90,
234567890, 859393847),
(0, 1, "2024-07-07", "2024-08-07", 29.20,
987654321, 583927846),
(1, 0, "2024-08-14", "2024-08-15", 24.99,
987654321, 948393848),
(0, 0, "2024-09-09", "2024-09-10", 11.80,
987654321, 837264726),
(1, 1, "2024-10-22", "2024-11-22", 45.60,
329585474, 373954712),
(1, 0, "2024-11-30", "2024-12-01", 31.25,
329585474, 482719283),
(0, 1, "2025-01-08", "2025-02-08", 34.70,
329585474, 383748739),
(1, 1, "2025-02-12", "2025-03-12", 17.85,
678901234, 482719283),
(0, 0, "2025-03-25", "2025-03-26", 13.60,
678901234, 583927846),
(1, 0, "2025-04-18", "2025-04-19", 20.45,
678901234, 383748739),
(0, 0, "2025-05-11", "2025-05-12", 18.80,
234567890, 383748739),
(1, 1, "2025-06-24", "2025-07-24", 40.25,
349712867, 373954712),
(1, 0, "2025-07-01", "2025-07-02", 26.90,
987654321, 859393847),
(0, 1, "2025-08-05", "2025-09-05", 31.40,
246813579, 948393848),
(1, 1, "2025-09-10", "2025-10-10", 15.75,
457682430, 948393848)

```

;

— Povoamento da Tabela ‘Viagem’

— Select * FROM ‘Viagem’
— Delete FROM ‘Viagem’

Insert Into Viagem
    (Data, Bilhete, Origem, Destino)
Values
    ("2023-01-15", 1, 2, 4),
    ("2023-01-16", 1, 7, 9),
    ("2023-02-10", 2, 3, 4),
    ("2023-02-11", 2, 5, 9),
    ("2023-02-12", 2, 1, 3),
    ("2023-02-13", 2, 6, 3),
    ("2023-02-14", 2, 1, 5),
    ("2023-02-15", 2, 4, 9),
    ("2023-03-05", 3, 2, 1),
    ("2023-03-06", 3, 7, 8),
    ("2023-04-20", 4, 9, 8),
    ("2023-04-21", 4, 3, 2),
    ("2023-05-12", 5, 1, 2),
    ("2023-05-13", 5, 7, 8),
    ("2023-06-30", 6, 3, 1),
    ("2023-07-01", 6, 2, 3),
    ("2023-07-17", 7, 5, 3),
    ("2023-07-18", 7, 9, 1),
    ("2023-08-22", 8, 1, 2),
    ("2023-08-23", 8, 9, 1),
    ("2023-09-08", 9, 3, 1),
    ("2023-09-09", 9, 6, 5),
    ("2023-10-05", 10, 4, 3),
    ("2023-10-06", 10, 2, 1),
    ("2023-11-18", 11, 3, 5),
    ("2023-11-19", 11, 7, 9),
    ("2023-12-25", 12, 2, 5),
    ("2023-12-26", 12, 1, 4),
    ("2024-01-04", 13, 7, 8),
    ("2024-01-05", 13, 9, 1),
    ("2024-02-19", 14, 5, 2),
    ("2024-02-20", 14, 4, 7),
    ("2024-03-10", 15, 7, 9),
    ("2024-03-11", 15, 1, 9),
    ("2024-04-14", 16, 8, 2),
    ("2024-04-15", 16, 7, 3),
    ("2024-05-27", 17, 6, 4),

```

```
("2024-05-28", 17, 5, 3),  
("2024-06-02", 18, 1, 2),  
("2024-06-03", 18, 3, 4),  
("2024-07-07", 19, 5, 6),  
("2024-07-08", 19, 7, 8),  
("2024-08-14", 20, 9, 1),  
("2024-08-15", 20, 9, 2),  
("2024-09-09", 21, 8, 3),  
("2024-09-10", 21, 5, 4),  
("2024-10-22", 22, 6, 2),  
("2024-10-23", 22, 1, 3),  
("2024-11-30", 23, 5, 6),  
("2024-12-01", 23, 8, 2),  
("2025-01-08", 24, 9, 1),  
("2025-01-09", 24, 7, 6),  
("2025-02-12", 25, 3, 5),  
("2025-02-13", 25, 4, 7),  
("2025-03-25", 26, 3, 7),  
("2025-03-26", 26, 4, 9),  
("2025-04-18", 27, 3, 9),  
("2025-04-19", 27, 7, 2),  
("2025-05-11", 28, 8, 1),  
("2025-05-12", 28, 9, 3),  
("2025-06-24", 29, 7, 5),  
("2025-06-25", 29, 6, 8),  
("2025-07-01", 30, 8, 2),  
("2025-07-02", 30, 1, 2),  
("2025-08-05", 31, 1, 9)
```

```
;
```

```
— Povoamento da Tabela ‘EstaçãoVeiculo’
```

```
— Select * FROM ‘EstaçãoVeiculo’
```

```
— Delete FROM ‘EstaçãoVeiculo’
```

```
Insert Into EstaçãoVeiculo  
    (Hora, Veiculo, Estação)  
Values
```

```
(2023 01 15 22:12:13,A 001,2),  
(2023 01 16 00:48:39,A 001,4),  
(2023 01 16 10:54:46,E 005,7),  
(2023 01 16 14:12:43,E 005,9),  
(2023 02 10 21:54:49,J 010,3),  
(2023 02 11 02:27:45,J 010,4),  
(2023 02 11 02:15:01,G 007,5),  
(2023 02 11 06:44:34,G 007,9),  
(2023 02 12 23:20:43,B 002,1),
```

(2023 02 13 01:14:54 ,B 002 ,3) ,
(2023 02 13 13:54:07 ,D 004 ,6) ,
(2023 02 13 17:59:31 ,D 004 ,3) ,
(2023 02 14 23:37:05 ,I 009 ,1) ,
(2023 02 15 00:51:16 ,I 009 ,5) ,
(2023 02 15 05:11:41 ,C 003 ,4) ,
(2023 02 15 09:41:13 ,C 003 ,9) ,
(2023 03 05 03:27:23 ,H 008 ,2) ,
(2023 03 05 07:38:27 ,H 008 ,1) ,
(2023 03 06 09:37:47 ,F 006 ,7) ,
(2023 03 06 10:44:14 ,F 006 ,8) ,
(2023 04 20 14:22:13 ,L 012 ,9) ,
(2023 04 20 19:07:19 ,L 012 ,8) ,
(2023 04 21 00:19:31 ,K 011 ,3) ,
(2023 04 21 04:29:52 ,K 011 ,2) ,
(2023 05 12 23:14:56 ,A 001 ,1) ,
(2023 05 13 01:40:26 ,A 001 ,2) ,
(2023 05 13 09:23:54 ,I 009 ,7) ,
(2023 05 13 13:24:56 ,I 009 ,8) ,
(2023 06 30 04:50:33 ,E 005 ,3) ,
(2023 06 30 06:58:18 ,E 005 ,1) ,
(2023 07 01 13:38:29 ,C 003 ,2) ,
(2023 07 01 17:01:55 ,C 003 ,3) ,
(2023 07 17 15:34:08 ,G 007 ,5) ,
(2023 07 17 19:33:38 ,G 007 ,3) ,
(2023 07 18 23:17:11 ,B 002 ,9) ,
(2023 07 19 02:06:53 ,B 002 ,1) ,
(2023 08 22 20:36:21 ,J 010 ,1) ,
(2023 08 23 00:01:58 ,J 010 ,2) ,
(2023 08 23 03:27:38 ,D 004 ,9) ,
(2023 08 23 04:49:41 ,D 004 ,1) ,
(2023 09 08 11:00:19 ,K 011 ,3) ,
(2023 09 08 14:08:48 ,K 011 ,1) ,
(2023 09 09 05:58:37 ,H 008 ,6) ,
(2023 09 09 07:56:52 ,H 008 ,5) ,
(2023 10 05 17:40:46 ,F 006 ,4) ,
(2023 10 05 22:39:41 ,F 006 ,3) ,
(2023 10 06 12:42:29 ,L 012 ,2) ,
(2023 10 06 17:13:27 ,L 012 ,1) ,
(2023 11 18 05:16:09 ,A 001 ,3) ,
(2023 11 18 09:06:30 ,A 001 ,5) ,
(2023 11 19 14:13:28 ,G 007 ,7) ,
(2023 11 19 17:18:25 ,G 007 ,9) ,
(2023 12 25 21:10:36 ,I 009 ,2) ,
(2023 12 26 01:47:19 ,I 009 ,5) ,
(2023 12 26 11:29:30 ,E 005 ,1) ,
(2023 12 26 16:04:31 ,E 005 ,4) ,

(2024 01 04 18:22:59 ,J 010,7) ,
(2024 01 04 23:09:23 ,J 010,8) ,
(2024 01 05 17:31:51 ,B 002,9) ,
(2024 01 05 20:55:55 ,B 002,1) ,
(2024 02 19 22:29:59 ,D 004,5) ,
(2024 02 20 01:13:28 ,D 004,2) ,
(2024 02 20 23:51:30 ,C 003,4) ,
(2024 02 21 02:11:56 ,C 003,7) ,
(2024 03 10 05:58:24 ,F 006,7) ,
(2024 03 10 09:37:08 ,F 006,9) ,
(2024 03 11 05:08:23 ,H 008,1) ,
(2024 03 11 08:12:53 ,H 008,9) ,
(2024 04 14 18:08:32 ,L 012,8) ,
(2024 04 14 21:30:44 ,L 012,2) ,
(2024 04 15 19:32:32 ,K 011,7) ,
(2024 04 15 21:54:31 ,K 011,3) ,
(2024 05 27 06:34:14 ,A 001,6) ,
(2024 05 27 11:08:36 ,A 001,4) ,
(2024 05 28 01:49:41 ,C 003,5) ,
(2024 05 28 02:50:11 ,C 003,3) ,
(2024 06 02 02:08:38 ,B 002,1) ,
(2024 06 02 05:43:44 ,B 002,2) ,
(2024 06 03 10:29:05 ,G 007,3) ,
(2024 06 03 13:39:14 ,G 007,4) ,
(2024 07 07 16:18:35 ,I 009,5) ,
(2024 07 07 19:12:18 ,I 009,6) ,
(2024 07 08 15:11:32 ,E 005,7) ,
(2024 07 08 18:46:31 ,E 005,8) ,
(2024 08 14 23:44:08 ,J 010,9) ,
(2024 08 15 02:56:49 ,J 010,1) ,
(2024 08 15 20:58:58 ,H 008,9) ,
(2024 08 15 23:44:08 ,H 008,2) ,
(2024 09 09 01:41:20 ,D 004,8) ,
(2024 09 09 04:13:12 ,D 004,3) ,
(2024 09 10 02:26:48 ,L 012,5) ,
(2024 09 10 07:04:53 ,L 012,4) ,
(2024 10 22 11:22:38 ,F 006,6) ,
(2024 10 22 15:15:24 ,F 006,2) ,
(2024 10 23 21:56:53 ,K 011,1) ,
(2024 10 23 23:17:37 ,K 011,3) ,
(2024 11 30 20:56:20 ,A 001,5) ,
(2024 11 30 22:04:18 ,A 001,6) ,
(2024 12 01 15:48:05 ,J 010,8) ,
(2024 12 01 20:35:22 ,J 010,2) ,
(2025 01 08 12:57:11 ,G 007,9) ,
(2025 01 08 14:57:20 ,G 007,1) ,
(2025 01 09 02:55:49 ,H 008,7) ,

```

(2025 01 09 05:58:30 ,H 008,6) ,
(2025 02 12 19:47:16 ,B 002,3) ,
(2025 02 12 21:13:51 ,B 002,5) ,
(2025 02 13 10:35:25 ,D 004,4) ,
(2025 02 13 15:31:05 ,D 004,7) ,
(2025 03 25 21:01:23 ,F 006,3) ,
(2025 03 26 00:40:28 ,F 006,7) ,
(2025 03 26 04:18:49 ,C 003,4) ,
(2025 03 26 05:56:03 ,C 003,9) ,
(2025 04 18 19:49:07 ,K 011,3) ,
(2025 04 18 21:12:56 ,K 011,9) ,
(2025 04 19 19:41:20 ,I 009,7) ,
(2025 04 20 00:27:08 ,I 009,2) ,
(2025 05 11 14:41:04 ,E 005,8) ,
(2025 05 11 17:18:14 ,E 005,1) ,
(2025 05 12 11:32:56 ,L 012,9) ,
(2025 05 12 15:53:16 ,L 012,3) ,
(2025 06 24 19:26:18 ,A 001,7) ,
(2025 06 24 23:47:03 ,A 001,5) ,
(2025 06 25 00:21:25 ,D 004,6) ,
(2025 06 25 01:24:36 ,D 004,8) ,
(2025 07 01 07:07:44 ,J 010,8) ,
(2025 07 01 10:01:21 ,J 010,2) ,
(2025 07 02 06:56:37 ,G 007,1) ,
(2025 07 02 11:04:13 ,G 007,2) ,
(2025 08 05 05:40:56 ,B 002,1) ,
(2025 08 05 08:04:07 ,B 002,9) ,
;

— Povoamento da Tabela ‘FuncionarioVeiculo’

— Select * FROM ‘FuncionarioVeiculo’
— Delete FROM ‘FuncionarioVeiculo’

Insert Into FuncionarioVeiculo
  (Funcionario , Veiculo , data , observações)
Values
  (867854211,C-003,2023-01-10,Verificar os freios
   traseiros) ,
  (657045312,J-010,2023-02-05,Trocar o óleo do
   motor) ,
  (768906656,F-006,2023-03-15,Revisar sistema
   elétrico) ,
  (876462100,B-002,2023-04-20,Substituir pneus
   dianteiros) ,
  (294847192,G-007,2023-05-25,Verificar os níveis
   de fluidos) ,

```

(638263747,L-012,2023-06-10,Limpar os filtros de ar),
(456789012,K-011,2023-07-15,Reparar a luz de emergência),
(789012345,D-004,2023-08-20,Verificar a suspensão),
(432109876,H-008,2023-09-25,Realizar manutenção do sistema de ar condicionado),
(867854211,A-001,2023-10-10,Substituir as palhetas dos limpadores de para-brisa),
(657045312,C-003,2023-11-05,Verificar os freios dianteiros),
(768906656,J-010,2023-12-15,Trocar o filtro de combustível),
(876462100,F-006,2024-01-20,Revisar sistema de tração),
(294847192,B-002,2024-02-25,Verificar a iluminação interna),
(638263747,G-007,2024-03-10,Reparar o sistema de som),
(456789012,L-012,2024-04-15,Realizar manutenção dos assentos),
(789012345,K-011,2024-05-20,Verificar o sistema de comunicação),
(432109876,D-004,2024-06-25,Reparar os espelhos retrovisores),
(867854211,H-008,2024-07-10,Verificar o sistema de frenagem),
(657045312,A-001,2024-08-05,Realizar manutenção das portas),
(768906656,C-003,2024-09-15,Trocar as pastilhas de freio),
(876462100,J-010,2024-10-20,Verificar o sistema de arrefecimento),
(294847192,F-006,2024-11-25,Reparar o sistema de iluminação externa),
(638263747,B-002,2024-12-10,Realizar manutenção dos sistemas de segurança),
(456789012,G-007,2025-01-15,Verificar o sistema de direção),
(789012345,L-012,2025-02-20,Trocar o filtro de ar-condicionado)
;

— Povoamento da Tabela ‘ViagemVeiculo’

— Select * FROM ‘ViagemVeiculo’

```
— Delete FROM ‘ViagemVeiculo’
```

```
Insert Into ViagemVeiculo
(Viagem, Veiculo)
Values
(1,A-001),
(2,E-005),
(3,J-010),
(4,G-007),
(5,B-002),
(6,D-004),
(7,I-009),
(8,C-003),
(9,H-008),
(10,F-006),
(11,L-012),
(12,K-011),
(13,A-001),
(14,I-009),
(15,E-005),
(16,C-003),
(17,G-007),
(18,B-002),
(19,J-010),
(20,D-004),
(21,K-011),
(22,H-008),
(23,F-006),
(24,L-012),
(25,A-001),
(26,G-007),
(27,I-009),
(28,E-005),
(29,J-010),
(30,B-002),
(31,D-004),
(32,C-003),
(33,F-006),
(34,H-008),
(35,L-012),
(36,K-011),
(37,A-001),
(38,C-003),
(39,B-002),
(40,G-007),
(41,I-009),
(42,E-005),
```

```

(43,J-010),
(44,H-008),
(45,D-004),
(46,L-012),
(47,F-006),
(48,K-011),
(49,A-001),
(50,J-010),
(51,G-007),
(52,H-008),
(53,B-002),
(54,D-004),
(55,F-006),
(56,C-003),
(57,K-011),
(58,I-009),
(59,E-005),
(60,L-012),
(61,A-001),
(62,D-004),
(63,J-010),
(64,G-007),
(65,B-002)
;

```

Anexo 3 - Queries em SQL

```

— Queries
— Caso de Estudo: BeloMetroDB

USE 'BeloMetroDB';

— Numero de Bilhete Vendidos em um dia

delimiter $$

Create Function NrBilhetesVendidosDia(Data_Dia DATE) returns int
    deterministic
begin
    declare num int;
    set num = (Select COUNT(*) As 'Total de Bilhetes Vendidos'
                FROM Bilhete
                Where DataVenda = Data_Dia);
    return num;
end $$

delimiter ;

```

```

— DROP FUNCTION ValorAuferidoDia

— VALOR AUFERIDO NA VENDA DE BILHETE EM UM DIA
delimiter $$

Create Function ValorAuferidoDia(Data_Dia DATE) returns double
    deterministic
begin
    declare num double;
    set num = (Select SUM(Preço) AS 'Total Auferido'
                FROM Bilhete
                Where DataVenda = Data_Dia);
    return num;
end $$

— DROP FUNCTION GastosCliente
— Total gasto por um Cliente em bilhetes
delimiter $$

Create Function GastosCliente(NIF int) returns double
    deterministic
begin
    declare num double;
    set num = (select SUM(Preço) AS 'Gastos'
                From bilhete
                where Cliente = NIF);
    return num;
end $$

— DROP FUNCTION MediaGanhosEntreDatas
— Media de ganhos do belo metro entre datas
DELIMITER $$

CREATE FUNCTION MediaGanhosEntreDatas(data_inicio DATE, data_fim
    DATE) RETURNS DOUBLE DETERMINISTIC
BEGIN
    DECLARE num double;
    DECLARE dias INT;

    SET num = (SELECT SUM(Preço) AS 'Ganhos'
                FROM bilhete
                WHERE DataVenda BETWEEN data_inicio AND data_fim)
                ;
    SET dias = DATEDIFF(data_fim, data_inicio) + 1;

```

```

        RETURN num / dias;
END $$

DELIMITER ;

DELIMITER $$
create function MaisViagensPorDia(data_escolhida date) returns
    varchar(255) deterministic
BEGIN
    Declare nome varchar(255);

    set nome = (SELECT c.Nome
    From cliente as c
        Inner Join clienterealizaviagem as r
    On c.Contribuinte = r.Cliente
        Inner Join Viagem as V
    on V.IdViagem = r.viagem
    where V.Data = data_escolhida
    group by c.Nome
    order by COUNT(V.IdViagem) DESC
    limit 1);

    Return nome;

END $$

DELIMITER ;

select e.Nome AS estacao , COUNT(*) AS 'TotalVeiculos'
    From estacao veiculo v
        Inner Join estacao On v.Estação = e.IdEstação
    group by v.Estação
    order by TotalVeiculos DESC
    limit 1;

— DROP FUNCTION VeiculoPassaEstacao
DELIMITER $$

Create Function VeiculoPassaEstacao() returns varchar(255)
    deterministic
BEGIN
    declare estacao_nome varchar(255);

    Select e.Nome into estacao_nome
    from estacao veiculo ev
        Inner join estacao e On ev.Estação = e.IdEstação
    group by ev.Estação

```

```

        order by COUNT(ev.Veiculo) DESC
        limit 1;

        return estacao_nome;

END $$
DELIMITER ;

Select VeiculoPassaEstacao()

Use belometrodb;

— Adiciona um utilizador à base de dados.
— Drop Procedure addCliente
delimiter $$
Create Procedure addCliente(In Nif_Cliente Int , In Nome_Cliente
Varchar(75) ,
In Nasc_Cliente
Date
, in Genero_Cliente
char
(1) ,
In Rua_Cliente Varchar(100) , In Localidade_Cliente Varchar(30) ,
In CP_Cliente Varchar(8) , In Email_Cliente Varchar(100) ,
In Contacto_Cliente int)

Begin
Start Transaction;
Set autocommit = 0;

— Criar o cliente
Insert Into Cliente(Contribuinte , Nome, DataNascimento ,
Genero , Rua, Localidade , CodigoPostal , Email)
values (Nif_Cliente , Nome_Cliente , Nasc_Cliente ,
Genero_Cliente , Rua_Cliente ,
Localidade_Cliente , CP_Cliente , Email_Cliente
);
— Adicionar Contacto
Insert Into ClienteContactos(Cliente , Contacto)
Values (Nif_Cliente , Contacto_Cliente);

Commit;
end $$
delimiter ;

```

```

— Remove um Cliente do sistema
— Drop procedure removeCliente
delimiter $$ 
Create Procedure removeCliente(ln Nif_Cliente Int)
Begin
Start Transaction;
    Set autocommit = 0;
    Delete from clientecontactos
        where Cliente = Nif_Cliente;
    Delete from Cliente
        Where Contribuinte = Nif_Cliente;
    commit;
end $$ 
delimiter ;

— Adiciona um contacto a um utilizador
— Drop Procedure addClienteContacto
delimiter $$ 
Create Procedure addClienteContacto(ln Nif_Cliente int , ln
    Contacto_Cliente int)
Begin
    Start Transaction;
    Set autocommit = 0;
        Insert Into clientecontactos(Cliente , Contacto)
            Values (Nif_Cliente , Contacto_Cliente);
    Commit;
End $$ 
delimiter ;

— Regista uma manutenção a um veiculo
— Drop Procedure registaManutencaoVeiculo
delimiter $$ 
Create Procedure registaManutencaoVeiculo(ln Id_Veiculo Varchar
    (20) , ln Data_Manutencao DATE, ln Id_Func Int , ln Observ TEXT
    )
Begin
    Start Transaction;
    Set autocommit = 0;
        Insert Into funcionarioveiculo(Funcionario ,
            Veiculo , Data , Observações)
            Values (Id_Func , Id_Veiculo ,
                Data_Manutencao , Observ);
    commit;
end $$ 
delimiter ;

— Adiciona uma estação

```

```

— Drop Procedure addEstação;
delimiter $$

Create Procedure addEstação( In Nome_Estacao VARCHAR(50) , in
    Longitude_Estacao double , in Latitude_Estacao double , in
    Pontos_Estacao Varchar(150) , in Descricao_Estacao Text )
Begin
    Start Transaction;
    set autocommit = 0;
    Insert into Estação( Nome, Longitude , Latitude , PontosInteresse ,
        Descrição )
        Values( Nome_Estacao , Longitude_Estacao , Latitude_Estacao
            , Pontos_Estacao , Descricao_Estacao );
    commit;
End $$

— Remove uma estação
— Drop Procedure removeEstação
delimiter $$

Create Procedure removeEstação( In Id_Estação int )
Begin
    Start Transaction;
    set autocommit = 0;
    Delete from Estação Where IdEstação = Id_Estação;
    commit;
end $$

— Regista a passagem de um veiculo numa estação
— Drop Procedure registaPassagemEstação;
delimiter $$

Create Procedure registaPassagemEstação( In Id_Estacao Int , in
    Id_Veiculo Varchar(20) , in Hora_Passagem DateTime )
Begin
    Start Transaction;
    set autocommit = 0;
        Insert into estaçãooveiculo( Estação , Veiculo , Hora )
            values( Id_Estacao , Id_Veiculo , Hora_Passagem );
    commit;
end $$

— Adiciona uma venda
— Drop procedure registaVenda
delimiter $$

Create Procedure registaVenda( In tipo_bilhete INT , In
    Pagamento_Bilhete int , In Preço_Bilhete decimal(5,2) , In

```

```

Func_Bilhete Int, In Cliente_Bilhete Int, In
DataVenda_Bilhete Date)
Begin
    Declare data_validade DATE;
Start Transaction;
Set autocommit = 0;

if(tipo_bilhete = 0) then
    set data_validade = addDate(DataVenda_bilhete, 1);
else set data_validade = addDate(DataVenda_bilhete, 30);
end if;

Insert into Bilhete(FormaPagamento, Tipo, DataVenda,
DataValidade, Preço, Funcionario, Cliente)
values(Pagamento_Bilhete, tipo_bilhete,
DataVenda_Bilhete, data_validade, Preço_Bilhete,
Func_Bilhete, Cliente_Bilhete);

commit;
end $$

delimter ;

— Cancela a venda de um bilhete, se este não possuir nenhuma
viagem
— Drop Procedure cancelaVenda;
delimter $$

Create Procedure cancelaVenda(In Id_Bilhete Int)
begin
Declare existe_Viagem Int;
Start Transaction;
Set autocommit = 0;
Set existe_Viagem = (Select count(*) From Viagem Where Bilhete =
Id_Bilhete);
If(existe_Viagem = 0) then
    Delete From Bilhete
        Where NrVenda = Id_Bilhete;
end if;
commit;
end $$

delimter ;

— Adiciona uma viagem ao sistema
— Drop Procedure addViagem;
delimter $$

Create Procedure addViagem(In Bilhete_Viagem Int, In
Origem_Viagem Int, In Destino_Viagem Int, In Data_Viagem Date
)

```

```

Begin
Start Transaction;
set autocommit = 0;
Insert into Viagem( Bilhete , Origem , Destino , Data)
values( Bilhete_Viagem , Origem_Viagem ,
Destino_Viagem , Data_Viagem );
commit;
end $$

— Adiciona Funcionario
— Drop Procedure addFuncionario;
delimiter $$

Create Procedure addFuncionario( In Nif_Func int , In Nome_Func
Varchar(75) , In Nasc_Func Date , In Genero_Func char(1) , In
Rua_Func Varchar(100) ,
In Local_Func
Varchar(30) ,
In CP_Func
Varchar(8) ,
In Email_Func
Varchar(100) ,
In cargo_Func
Int ,

```

```

In Salario_Func decimal (6,2),
,
In IBAN_Func Varchar(25), In Estacao_Func int, In contacto_Func INT)

Begin
Start Transaction;
Set autocommit = 0;
Insert into Funcionario(Contribuinte, Nome, DataNascimento, Genero, Rua, Localidade, CodigoPostal, email, cargo, Salario, IBAN, Estação)
value(Nif_Func, Nome_Func, Nasc_Func, Genero_Func, Rua_Func, Local_Func, CP_Func, Email_Func, cargo_Func, Salario_Func, IBAN_Func, Estacao_Func);
Insert into funcionariocontactos(Funcionario, Contacto)
value(Nif_Func, contacto_Func);

commit;
End $$

delimiter ;

— Remove um Funcionario do sistema
— Drop Procedure removeFuncionario
delimiter $$

Create Procedure removeFuncionario(In Nif_Func Int)
Begin
Start Transaction;
Delete From FuncionarioContactos
Where Funcionario = Nif_Func;
Delete From Funcionario
Where Contribuinte = Nif_Func;
commit;
end $$

delimiter ;

— Adiciona um contacto a um funcionario
— Drop Procedure addFuncionarioContacto
delimiter $$
```

```

Create Procedure addFuncionarioContacto( In Nif_Func int , In
Contacto_Func int )
Begin
    Start Transaction;
    Set autocommit = 0;
    Insert Into funcionariocontactos( Funcionario , Contacto )
        Values ( Nif_Func , Contacto_Func );
    Commit;
End $$

— Adiciona um veiculo a uma viagem
— Drop Procedure addVeiculoViagem
delimiter $$

create Procedure addVeiculoViagem( In Id_Viagem int , In
Id_Veiculo varchar(20) )
begin
    Start Transaction;
    Insert into viagemveiculo(Viagem , Veiculo )
        value( Id_Viagem , Id_Veiculo );
end $$

— Adiciona um veiculo ao sistema .
— Drop Procedure addVeiculo
delimiter $$

Create Procedure addVeiculo( In Id_Veiculo Varchar(20) , In
Carruagens_Veiculo Int , In Lugares_Veiculo Int )
Begin
    Start transaction;
    Set autocommit = 0;
    Insert into Veiculo( IdVeiculo , NrCarruagens , NrLugares )
        Value( Id_Veiculo , Carruagens_Veiculo , Lugares_Veiculo );
    commit;
end $$

— Remove um veiculo do sistema .
— Drop Procedure removeVeiculo
delimiter $$

Create Procedure removeVeiculo( In Id_Veiculo Varchar(20) )
Begin
    Start transaction;
    set autocommit = 0;
        Delete From Veiculo Where IdVeiculo = Id_Veiculo ;
    commit;
End $$
```

```

delimiter ;
Use belometrodb

— Informação de um cliente através do seu Contribuinte
delimiter $$

Create Procedure PVClientePorNif(In Nif_Cliente INT)
BEGIN
    select C.Contribuinte , C.Nome, C.DataNascimento As 'Data
        Nascimento',
    Case C.Genero
        When 0 Then 'Masculino'
        When 1 Then 'Feminino'
    End As 'Genero', concat(C.Rua, ' ', C.Localidade, ' ',
        C.CodigoPostal) As Morada, C.Email
    FROM Cliente AS C
        Where C.Contribuinte = Nif_Cliente;
end $$

delimiter ;

— Ver Contactos de um Cliente
delimiter $$

Create Procedure PVContactoCliente(In Nif_Cliente INT)
BEGIN
    Select C.Contribuinte AS 'Contribuinte', C.Nome AS 'Nome
        ', F.Contacto AS 'Contacto'
    FROM clientecontactos AS F Right outer join Cliente AS C
        ON C.Contribuinte = F.Cliente
    Where F.Cliente = Nif_Cliente
        order by C.Nome ASC;
end $$

delimiter ;

— Identificar quem vendeu e para qual cliente um bilhete foi
    vendido através do número de venda do bilhete.
delimiter $$

create Procedure FPIInfoBilhete(In NrVenda_Bilhete INT)
Begin
    Select B.NrVenda AS 'Numero Venda Bilhete',
        Case B.Tipo
            When 0 Then 'Bilhete Diario'
            When 1 Then 'Bilhete Mensal'
        End As 'Tipo Bilhete',
        Case B.FormaPagamento
            When 0 Then 'Dinheiro'
            When 1 Then 'Multibanco'
        End As 'Forma de Pagamento', C.Contribuinte AS 'Contribuinte Cliente',

```

```

C.Nome AS 'Nome Cliente', F.Contribuinte AS 'Contribuinte Funcionario', F.Nome AS 'Nome Funcionario'
From Bilhete As B Right outer Join Cliente as C
    On B.Cliente = C.Contribuinte
    Right Outer Join Funcionario as F
        On B.Funcionario = F.
            Contribuinte
        where B.Funcionario IS NOT NULL
            AND B.Cliente IS NOT NULL AND
            B.NrVenda = NrVenda_Bilhete
        order by C.Nome ASC, F.Nome ASC;
end $$

— Ver quais horas passaram veiculos numa Estação
delimiter $$

Create Procedure FPVerHoraVeiculo(In Id_Estacao int)
Begin
    Select E.IDEstação AS 'Id Estação', E.NOME AS 'Nome Estação', F.HORA AS 'Hora', F.Veiculo AS 'Veiculo'
        From Estação AS E Right outer Join
            EstaçãoVeiculo As F
                On E.IDEstação = F.Estação
    Where F.Estação = Id_Estacao AND F.HORA IS NOT NULL
        order by F.HORA DESC;
end $$

— Ver Informacao de uma Estação
delimiter $$

Create Procedure FPVerEstacao(In Id_Estacao int)
Begin
    select * FROM estação
        where estação.IdEstação = Id_Estacao;
end $$

— Saber qual o tipo de bilhete que um cliente possui
    introduzindo o seu NIF.

delimiter $$

Create Procedure tipoBilheteCliente(In Nif_Cliente INT)
begin
    Select C.Contribuinte As 'Contribuinte do Cliente', C.
        Nome As 'Nome do Cliente', B.NrVenda As 'Numero de
        Venda Bilhete',

```

```

Case B.Tipo
    When 0 Then 'Bilhete Diário'
    When 1 Then 'Bilhete Mensal'
End AS 'Tipo de Bilhete', B.DataVenda as 'Data
de Venda', B.DataValidade as 'Data de
Validade'
From Bilhete as B Right outer Join Cliente as C
    On B.Cliente = C.Contribuinte
    Where B.NrVenda Is Not Null and C.
        Contribuinte = 749384625
    Order by B.DataVenda DESC;
end $$

— saber qual foi o cliente que realizou uma viagem e qual a
origem e destino do cliente introduzindo o ID da viagem.
delimiter $$

Create Procedure infoViagem( In Id_Viagem Int )
Begin
    Select V.IdViagem as 'Id Viagem', C.Contribuinte AS 'Contribuinte Cliente', C.Nome AS 'Nome Cliente',
        Origem.Nome AS 'Origem', Destino.Nome AS 'Destino', V
        .Data AS 'Data Realização'
        From Viagem V
        Join Estação Origem On V.Origem = Origem.
            IdEstação
        Join Estação Destino On V.Destino =
            Destino.IdEstação
        Inner Join clienteRealizaViagem
            On clienteRealizaViagem.
                Viagem = V.IdViagem
        Inner Join cliente As C
            On C.Contribuinte =
                clienterealizaviagem.
                    Cliente
    Where V.IdViagem = Id_Viagem;
End $$

Use Belometrodb

— Informação de um funcionario através do seu Contribuinte

delimiter $$

create Procedure REFuncionarioPorNif( In Nif_Funcionario INT )
Begin
    Select * FROM funcionario
        Where Funcionario.Contribuinte = Nif_Funcionario

```

```

;

end $$

delimiter ;

— Ver Contactos de um funcionario através do seu Contribuinte

delimiter $$
create procedure REContactoFuncionario(In Nif_Funcionario INT)
Begin
    Select F.Nome, Case F.Cargo
        When 0 Then 'Funcionario Posto Venda'
        When 1 Then 'Responsavel Estação'
        When 2 Then 'Tecnico de Manutenção'
        When 3 then 'Administrador'
    END AS 'Cargo', E.Nome, C.Contacto
    From Funcionario as F inner Join FuncionarioContactos AS C
    On F.Contribuinte = C.Funcionario
        inner Join Estação as E
    On E.IdEstação = F.Estação
    where F.Contribuinte = Nif_Funcionario;
End $$

delimiter ;

— Ver os funcionarios de uma estação através do ID da estação
delimiter $$
Create Procedure REFuncPorEstacao(In Id_Estacao INT)
Begin
    Select F.Contribuinte As 'Contribuinte Funcionario', F.
        Nome As 'Nome Funcionario',
        Case F.Cargo
            When 0 Then 'Funcionario Posto Venda'
            When 1 Then 'Responsavel Estação'
            When 2 Then 'Tecnico de Manutenção'
            When 3 then 'Administrador'
        END AS 'Cargo', E.IdEstação As 'Id Estação', E.
        Nome As 'Nome da Estação'
    From Funcionario as F Right outer Join Estação
        as E
        On F.Estação = E.IdEstação
    where E.IdEstação = Id_Estacao
    Order by F.Cargo desc, F.Nome ASC;
End $$

delimiter ;

Use belometroDB;

delimiter $$
```

```

Create Procedure TMinfoVeiculo( In  Id_Veiculo Varchar(20))
Begin
    Select V.IdVeiculo As 'Identificador', V.NrCarruagens As
        'Nr. Carruagens', V.NrLugares As 'Nr. Lugares'
        From Veiculo as V
    Where V.IdVeiculo = Id_Veiculo ;
end $$

delimiter ;

```

Anexo 4 - Funções usadas em SQL

```

USE belometrodb;

— Criação de grupo de piligeios

— Drop Role 'Administrador';
CREATE ROLE if not exists Administrador;

— Drop Role 'PostoVenda';
CREATE ROLE if not exists PostoVenda;

— Drop Role RespEstacao;
Create Role if not exists RespEstacao;

— Drop Role TecnicoManutencao;
Create Role if not exists TecnicoManutencao;

— Drop Role Cliente;
Create Role if not exists Cliente;

— Privilegios do grupo de Administradores

— Show grants for Administrador;
Grant all privileges
    on 'belometrodb'.*
    to Administrador
    with grant option;

— Privilegios do grupo de PostoVenda

— Show grants for PostoVenda;
Grant Select
    on 'belometrodb'.VerClientes
    to PostoVenda;

```

```

Grant Select
    on 'belometrodb '.verclientecontacto
    to PostoVenda;

Grant Select
    on 'belometrodb '.verbilhetes
    to PostoVenda;

Grant Select
    On 'belometrodb '.verpassagemveiculo
    To PostoVenda;

grant execute on procedure 'belometrodb '.FPInfoBilhete to
    PostoVenda;
grant execute on procedure 'belometrodb '.FPVerEstacao to
    PostoVenda;
grant execute on procedure 'belometrodb '.FPVerHoraVeiculo to
    PostoVenda;
grant execute on procedure 'belometrodb '.infoViagem to
    PostoVenda;
grant execute on procedure 'belometrodb '.PVClientePorNif to
    PostoVenda;
grant execute on procedure 'belometrodb '.PVContactoCliente to
    PostoVenda;
grant execute on procedure 'belometrodb '.tipobilheteCliente to
    PostoVenda;
grant execute on procedure 'belometrodb '.addCliente to
    PostoVenda;
grant execute on procedure 'belometrodb '.addClienteContacto to
    PostoVenda;
grant execute on procedure 'belometrodb '.registaVenda to
    PostoVenda;
grant Select on belometrodb.verEstações to PostoVenda;

— Privilegios do grupo RespEstacao
— Show grants for RespEstacao;

Grant execute on procedure 'belometrodb '.REFuncPorEstacao to
    RespEstacao;
Grant execute on procedure 'belometrodb '.REContactoFuncionario
    to RespEstacao;
Grant execute on procedure 'belometrodb '.REFuncionarioPorNif to
    RespEstacao;
Grant execute on procedure 'belometrodb '.addEstação to
    RespEstacao;
Grant execute on procedure 'belometrodb '.cancelaVenda to
    RespEstacao;

```

```

Grant execute on procedure 'belometrodb'.removeCliente to
  RespEstacao;
Grant Select on 'belometrodb'.verfuncionarios to RespEstacao;
Grant Select on 'belometrodb'.verfuncionariocontacto to
  RespEstacao;
grant Select on belometrodb.verEstações to RespEstacao;

— Privilegios do grupo TecnicoManutencao
— Show grants for TecnicoManutencao

grant execute on procedure 'belometrodb'.''
  registaManutencaoVeiculo' to TecnicoManutencao;
grant execute on procedure 'belometrodb'.addVeiculo to
  TecnicoManutencao;
grant execute on procedure 'belometrodb'.removeVeiculo to
  TecnicoManutencao;
grant execute on procedure 'belometrodb'.registaPassagemEstação
  to TecnicoManutencao;

— Privilegios do grupo Cliente
— Show grants for Cliente

grant Select on belometrodb.verpassagemveiculo to Cliente;
grant Select on belometrodb.verEstações to Cliente;

— Criação de Utilizadores

— Show grants for 'administrador1'@'localhost' USING
  Administrador;
— Drop user 'administrador1'@'localhost';
Create user if not exists 'administrador1'@'localhost'
  identified by '1234'
  default role Administrador;

— Show grants for 'postovenda'@'localhost' USING PostoVenda;
— Drop user 'postovenda'@'localhost';
Create user if not exists 'postovenda'@'localhost'
  identified by '1234'
  default role PostoVenda;

— Show grants for 'respEstacao'@'localhost' USING RespEstacao,
  PostoVenda;
— Drop user 'respEstacao'@'localhost';
Create user if not exists 'respEstacao'@'localhost'
  identified by '1234'
  default role RespEstacao, PostoVenda;

```

```

— Show grants for 'tecnicomamanutencao'@'localhost' Using
  TecnicoManutencao;
— Drop user 'tecnicomamanutencao'@'localhost';
Create user if not exists 'tecnicomamanutencao'@'localhost'
  identified by '1234'
  default role TecnicoManutencao;

— Show grants for 'cliente'@'localhost' Using Cliente;
— Drop user 'cliente'@'localhost';
Create user if not exists 'cliente'@'localhost'
  identified by '1234'
  default role Cliente;

flush privileges;

— Remover grupo de privilegio a utilizadores

— Revoke 'Administrador' from 'administrador1'@'localhost'
— Revoke 'PostoVenda' from 'postovenda'@'localhost'
— Revoke 'RespEstacao' from 'respEstacao'@'localhost'
— Revoke 'PostoVenda' from 'respEstacao'@'localhost'
— Revoke 'Cliente' from 'cliente'@'localhost'

  Use belometroDB;

— Criação de indices em funcionario
Create Index index_Func On Funcionario (Contribuinte);

— Criação de indices em Clientes
Create Index index_Clientes On Cliente (Contribuinte);

— Criação de indices em Viagens
Create Index index_viagem On Viagem (IdViagem);

  Use belometrodb;

— drop view Verclientes
create view Verclientes as
  Select C.Contribuinte, C.Nome, C.DataNascimento, C.
    Genero, concat(C.Rua, ' ', ' ', C.Localidade, ' ', ' ', C.
      CodigoPostal) as Morada, C.Email
  from Cliente As C
  order by C.Nome Asc;

— drop view VerClienteContacto
create view VerClienteContacto as
  Select C.Contribuinte, C.Nome, F.Contacto
  From Cliente as C

```

```

        Inner Join clientecontactos as F
On C.Contribuinte = F.Cliente
order by C.Nome Asc;

— drop view VerFuncionarios
create view VerFuncionarios as
Select F.Contribuinte, F.Nome, F.DataNascimento, F.
Genero, concat(F.Rua, ' ', F.Localidade, ' ', F.
CodigoPostal) as Morada, F.email,
Case F.Cargo
    When 0 then 'Funcionario Posto Venda'
    When 1 then 'Responsavel Estação'
    When 2 then 'Técnico de manutenção'
    When 3 then 'Administrador'
End As 'Cargo', E.Nome As 'Estação Desnignada'
From Funcionario As F
        Inner Join Estação As E
On F.Estação = E.IdEstação
Order by F.Nome Asc;

— drop view VerFuncionarioContacto
create view VerFuncionarioContacto as
Select C.Contribuinte, C.Nome, F.Contacto
From Funcionario as C
        Inner Join funcionariocontactos as F
On C.Contribuinte = F.Funcionario
order by C.Nome Asc;

— drop view VerVeiculos
create view VerVeiculos as
Select V.IdVeiculo as 'Identificador', V.NrCarruagens as
'Nr. Carruagens', V.NrLugares as 'Nr. Lugares'
        From Veiculo as V
Order by V.NrLugares;

— drop view VerVeiculosManutencao
create view VerVeiculosManutencao as
Select F.Nome as 'Tecnico', V.Veiculo as 'Identificador
Veiculo', V.Data as 'Data Realização', V.Observações
From funcionarioveiculo as V
        Inner Join Funcionario as F
On V.Funcionario = F.Contribuinte
order by V.Data Desc;

— drop view VerBilhetes
Create View VerBilhetes as
Select B.NrVenda As 'Nr. Venda',

```

```

Case B.FormaPagamento
    When 0 Then 'Dinheiro'
    When 1 Then 'Multibanco'
    End As 'Forma Pagamento',
Case B.Tipo
    When 0 Then 'Diario'
    When 1 Then 'Mensal'
    End As 'Tipo Bilhete', B.DataVenda As 'Data Venda', B.
        DataValidade As 'Data Validade', B.Preço as 'Preço',
    C.Contribuinte As 'NIF Cliente', C.Nome As 'Nome Cliente',
    F.Contribuinte As 'Nif Funcionario', F.Nome as 'Nome
        Funcionario'
    From Bilhete As B Inner Join Cliente As C
        On C.Contribuinte = B.Cliente
    Inner Join Funcionario As F
    On F.Contribuinte = B.Funcionario
    Order by B.DataVenda DESC;

Create View VerPassagemVeiculo As
    select E.Nome As 'Nome Estação', V.IdVeiculo As 'Veiculo
        ', F.Hora As 'Hora passagem'
    From EstaçãoVeiculo as F
        Inner Join Estação as E
    On E.IdEstação = F.Estação
        Inner Join Veiculo V
    On V.IdVeiculo = F.Veiculo
    Order by F.Hora DESC;

— drop view VerEstações
Create View VerEstações as
    Select Nome As 'Estação', concat(Longitude, ', ', ,
        Latitude) as 'Localização', PontosInteresse as 'Pontos de Interesse', Descrição as 'Descrição'
        From Estação
    Order by Nome Asc;

```

Anexo 5 - Script de Backup

```

import subprocess

host = 'localhost'
user = 'root'
password = '1234'
database_name = 'belometrodb'
arquivo_backup = 'belometroBackup.sql'

```

```

def faz_backup(host, user, password, database_name,
arquivo_backup):

    comando mysqldump = f'mysqldump -h {host} -u {user} -p{password} {database_name} > {arquivo_backup}'

    subprocess.run(comando mysqldump, shell=True)

    print(f'O backup da base de dados {database_name} foi criado com sucesso em {arquivo_backup}.')

```

```

#def drop_base_dados(host, user, password, database_name):
#
#    comando_drop = f'DROP DATABASE {database_name}'
#
#    subprocess.run(['mysql', '-h', host, '-u', user, '-p' +
#password, '-e', comando_drop])
#
#    print(f'A base de dados {database_name} foi removida com sucesso.')

```

```

faz_backup(host, user, password, database_name, arquivo_backup)
#drop_base_dados(host, user, password, database_name)

import subprocess
import mysql.connector

config = {
    'host': 'localhost',
    'user': 'root',
    'password': '1234'
}

host = 'localhost'
usuario = 'root'
senha = '1234'
nome_base_dados = 'belometrodb'
arquivo_backup = 'belometroBackup.sql'

```

```

def execute_backup(host, usuario, senha, nome_base_dados,
arquivo_backup):

    comando_mysql = f'mysql -h {host} -u {usuario} -p{senha} {
```

```

        nome_base_dados} < {arquivo_backup}'

subprocess.run(comando_mysql, shell=True)

print(f'O backup da base de dados {nome_base_dados} foi
executado com sucesso em {arquivo_backup}.')

```

```

con = mysql.connector.connect(**config)

cursor = con.cursor()

cursor.execute(f"CREATE DATABASE {nome_base_dados}")

con.commit()

cursor.close()

con.close()

print(f"A Base de Dados '{nome_base_dados}' foi criado com
sucesso!")

execute_backup(host, usuario, senha, nome_base_dados,
arquivo_backup)

```

Anexo 6 - Script de criação do database

```

import mysql.connector
import csv
import codecs

config = {
    'host': 'localhost',
    'user': 'root',
    'password': '1234',
    'database': 'belometrodb',
    'charset': 'utf8mb4'
}

conn = mysql.connector.connect(**config)

mycursor = conn.cursor()

with codecs.open('Estações.csv', 'r', encoding='utf-8') as

```

```

csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3], row[4])
        all_values.append(value)

query = "insert into estação ('Nome', 'Longitude', 'Latitude', 'PontosInteresse', 'Descrição') values (%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('Clientes.csv', 'r', encoding='utf-8') as
    csv_file:
        csvfile = csv.reader(csv_file, delimiter=',')
        next(csvfile)
        all_values = []
        for row in csvfile:
            value = (row[0], row[1], row[2], row[3], row[4], row[5],
                      row[6], row[7])
            all_values.append(value)

query = "insert into cliente ('Contribuinte', 'Nome', 'DataNascimento', 'Genero', 'Rua', 'Localidade', 'CodigoPostal', 'email') values (%s,%s,%s,%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('ContactosClientes.csv', 'r', encoding='utf-8') as
    csv_file:
        csvfile = csv.reader(csv_file, delimiter=',')
        next(csvfile)
        all_values = []
        for row in csvfile:
            value = (row[0], row[1])
            all_values.append(value)

query = "insert into clientecontactos ('Cliente', 'Contacto') values (%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('Funcionarios.csv', 'r', encoding='utf-8') as
    csv_file:
        csvfile = csv.reader(csv_file, delimiter=',')

```

```

next(csvfile)
all_values = []
for row in csvfile:
    value = (row[0], row[1], row[2], row[3], row[4], row[5],
              row[6], row[7], row[8], row[9], row[10], row[11])
    all_values.append(value)

query = "insert into funcionario ('Contribuinte', 'Nome',
    DataNascimento', 'Genero', 'Cargo', 'Salario', 'IBAN',
    'Rua', 'Localidade', 'CodigoPostal', 'email', 'Estação') values (%s,
    %s,%s,%s,%s,%s,%s,%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('ContactosFuncionarios.csv', 'r', encoding='utf
-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1])
        all_values.append(value)

query = "insert into funcionariocontactos ('Funcionario',
    'Contacto') values (%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('Veiculos.csv', 'r', encoding='utf-8') as
    csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2])
        all_values.append(value)

query = "insert into veiculo ('IdVeiculo', 'NrCarruagens',
    'NrLugares') values (%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('Bilhetes.csv', 'r', encoding='utf-8') as
    csv_file:

```

```

csvfile = csv.reader(csv_file, delimiter=',')
next(csvfile)
all_values = []
for row in csvfile:
    value = (row[0], row[1], row[2], row[3], row[4], row[5],
              row[6])
    all_values.append(value)

query = "insert into bilhete ('FormaPagamento', 'Tipo', 'DataVenda', 'DataValidade', 'Preço', 'Funcionario', 'Cliente') values (%s,%s,%s,%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('Viagens.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1], row[2], row[3])
        all_values.append(value)

query = "insert into viagem ('Data', 'Bilhete', 'Origem', 'Destino') values (%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('ViagemVeiculo.csv', 'r', encoding='utf-8') as csv_file:
    csvfile = csv.reader(csv_file, delimiter=',')
    next(csvfile)
    all_values = []
    for row in csvfile:
        value = (row[0], row[1])
        all_values.append(value)

query = "insert into viagemveiculo ('Viagem', 'Veiculo') values (%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('FuncionariosVeiculos.csv', 'r', encoding='utf-8') as csv_file:

```

```

csvfile = csv.reader(csv_file, delimiter=',')
next(csvfile)
all_values = []
for row in csvfile:
    value = (row[0], row[1], row[2], row[3])
    all_values.append(value)

query = "insert into funcionarioveiculo ('Funcionario', 'Veiculo'
    , 'Data', 'Observações') values (%s,%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()

with codecs.open('EstaçõesVeículos.csv', 'r', encoding='utf-8')
    as csv_file:
        csvfile = csv.reader(csv_file, delimiter=',')
        next(csvfile)
        all_values = []
        for row in csvfile:
            value = (row[0], row[1], row[2])
            all_values.append(value)

query = "insert into estaçãoveiculo ('Hora', 'Veiculo', 'Estação'
    ) values (%s,%s,%s)"
mycursor.executemany(query, all_values)
conn.commit()
conn.close()

```