



**Universidade do Minho**  
Licenciatura em Engenharia Informática

# Unidade Curricular de Bases de Dados

Ano Lectivo de 2024/2025

**Bela Rent-a-Car**

**Carolina Martins, Diogo Ribeiro, Filipa Gonçalves,  
Lucas Robertson**

Junho, 2025,

# **BELA RENT-A-CAR**

**Grupo 13**

**Carolina Martins, Diogo Ribeiro, Filipa Gonçalves,  
Lucas Robertson**

**Junho, 2025**



## Resumo

Octávio Faísca herdou a empresa do falecido pai. No início, tudo corria bem, mas sucederam bastantes acontecimentos infelizes e perante a situação o mesmo considerou que a solução para os seus problemas seria um Sistema de Gestão de Base de Dados (SGBD) que encomendou a um grupo de alunos da Universidade do Minho. Este documento descreve o processo de desenvolvimento realizado pela equipa de alunos de Engenharia Informática. Primeiramente, o dono da empresa expôs a necessidade de criar uma base de dados (BD) e definiu o contexto em que esta seria desenvolvida. Reuniu-se com a equipa de desenvolvimento para discutir o plano de execução. Após definirem o método de levantamento de requisitos, estes foram recolhidos junto da empresa e organizados de forma estruturada.

Seguiu-se a criação e documentação de um diagrama Entidade-Relacionamento (ER), que foi posteriormente transformado num modelo lógico relacional, normalizado até à terceira forma normal (3FN). Para validar este modelo, foram implementadas interrogações utilizando álgebra relacional. Este modelo foi convertido para um modelo físico no Sistema de Gestão de Bases de Dados (SGBD) MySQL. Foram criados vários utilizadores da BD com diferentes privilégios, a BD foi povoada, e foram implementadas as interrogações e alguns procedimentos, funções e gatilhos essenciais.

**Área de Aplicação:** Desenho e arquitetura de Sistemas de Bases de Dados

**Palavras-Chave:** Bases de Dados, Diagramas ER, Modelo Relacional, Normalização de Bases de Dados Relacionais, Álgebra Relacional, SQL, MySQL

# Índice Geral

1. Definição do Sistema	1
1.1 Contextualização	1
1.2 Motivação	1
1.3 Objetivos	2
1.4 Viabilidade	3
1.5 Recursos e Equipa de trabalho	3
1.6 Plano de Execução	4
1.7 Revisão e Aprovação	5
2. Levantamento e Análise de Requisitos	6
2.1 Sugestões Gerais	6
2.2 Organização dos requisitos levantados	9
2.3 Analise e Validação Geral dos requisitos	10
3. Modelação Conceptual	11
3.1 Apresentação da Abordagem de Modelação realizada	11
3.2 Identificação e Caracterização das Entidades	11
3.3 Identificação e Caracterização dos Relacionamentos	12
3.4 Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos	14
3.5 Apresentação e Explicação do Diagrama ER Produzido	17
3.5 Validação do Modelo Conceptual	19
4. Modelação Lógica	20
4.1 Construção do Modelo de Dados Lógico	20
4.2 Aplicação e Apresentação do Modelo Lógico Produzido	20
4.3 Normalização dos Dados	24
4.4 Validação do Modelo com Interrogações do Utilizador	26
5. Implementação Física	35
5.1 Apresentação e Explicação da Base de Dados Implementada	35
5.2 Criação de utilizadores da base de dados	40
5.3 Povoamento da base de dados	42
5.4 Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)	46
5.5 Definição e caracterização de vistas de utilização em SQL	50
5.6 Tradução das interrogações do utilizador para SQL	51
5.7 Indexação do Sistema de Dados	55
5.8 Implementação de procedimentos, funções e gatilhos	56
5.9 Backup da Base de Dados	63
6. Implementação do sistema de migração dados	65
6.1 Descrição das fontes de dados utilizadas	65

6.2 Descrição do programa implementado	66
7. Conclusões e Trabalho Futuro	68
Referências	71
Lista de Siglas e Acrónimos	72
Anexos	73
I. Anexo 1 – Diagrama de Gantt	74
II. Anexo 1 – Tabela Geral dos requisitos	76
III. Anexo 3 – Requisitos categorizados	77
IV. Anexo 4 – Relações RelaX	81
V. Anexo 5 – Expressões de Algebra relacional	85
VI. Anexo 6 – Script de criação da BD BelaRentacar	87
VII. Anexo 7 – Povoamento da BD	90
VIII. Anexo 8 – Views	93
IX. Anexo 9 – Utilizadores e permissões	94
X. Anexo 10 – Utilizadores e permissões	97
XI. Anexo 11 – Script de migração	106

# Índice de Figuras

Figura 1 - Miniatura do Diagrama de Gantt	4
Figura 2 - Cabeçalho da ata da reunião de construção do cronograma para o desenvolvimento da BD.	5
Figura 3 - Cabeçalho da ata da reunião de revisão do projeto da BD.	5
Figura 4 - Cabeçalho da ata da reunião de levantamento de requisitos.	6
Figura 5 - Inquérito a Octávio Faísca	7
Figura 6 - Inquérito feito aos clientes de Bela Rent-A-Car	8
Figura 7 - Algumas respostas ao Inquérito feito aos clientes	8
Figura 8 - Primeiros requisitos da tabela global	9
Figura 9 - Primeiros requisitos da tabela de requisitos de manipulação	9
Figura 10 - Primeiros requisitos da tabela de requisitos de descrição	9
Figura 11 - Primeiros requisitos da tabela de requisitos de controlo	9
Figura 12 - Cabeçalho da ata da reunião de validação de requisitos.	10
Figura 13 - Entidades do modelo conceptual	18
Figura 14 - Versão final do modelo conceptual	18
Figura 15 - Modelo lógico realizado pela equipa de desenvolvimento	21
Figura 16 - Relação "Exerce".	25
Figura 17 - Relação "Funcao", "Filial" e "Cliente_Contacto".	25
Figura 18 - Relações "Cliente", "Funcionario" e "Automovel".	25
Figura 19 - Relação "Aluguer".	26
Figura 20 - Exemplo da relação "Automovel".	27
Figura 21 - Requisitos de manipulação que deram origem às interrogações	27
Figura 22 - Árvore de interrogação do RM8 aplicada aos dados de teste	28
Figura 23 - Árvore de interrogação do RM9 aplicada aos dados de teste	29
Figura 24 - Árvore de interrogação do RM10 aplicada aos dados de teste	29
Figura 25 - Árvore de interrogação do RM11 aplicada aos dados de teste	30
Figura 26 - Árvore de interrogação do RM12 aplicada aos dados de teste	31
Figura 27 - Árvore de interrogação do RM13 aplicada aos dados de teste	31
Figura 28 - Árvore de interrogação do RM14 aplicada aos dados de teste	32
Figura 29 - Árvore de interrogação do RM15 aplicada aos dados de teste	33
Figura 30 - Árvore de interrogação do RM16 aplicada aos dados de teste	33
Figura 31 - Árvore de interrogação do RM17 aplicada aos dados de teste	34
Figura 32 - Crontab Manager	64
Figura 33 - Miniatura do Gaunt final	69

# Índice de Tabelas

Tabela 1 - Entidades identificadas pela equipa de desenvolvimento	12
Tabela 2 - Relacionamentos identificados pela equipa de desenvolvimento	14
Tabela 3 - Atributos da entidade "Cliente"	15
Tabela 4 - Atributos da entidade "Automóvel"	15
Tabela 5 - Atributos da entidade "Aluguer"	16
Tabela 6 - Atributos da entidade "Funcionario"	16
Tabela 7 - Atributos da entidade "Funcao"	16
Tabela 8 - Atributos da entidade "Filial"	16
Tabela 9 – Dicionário de dados da entidade "Funcionario"	22
Tabela 10 - Dicionário de dados da entidade "Funcao"	22
Tabela 11 - Dicionário de dados da entidade "Exerce"	22
Tabela 12 - Dicionário de dados da entidade "Cliente"	23
Tabela 13 - Dicionário de dados da tabela "Cliente_Contactos"	23
Tabela 14 - Dicionário de dados da tabela "Automovel"	23
Tabela 15 - Dicionário de dados da tabela "Filial"	23
Tabela 16 - Dicionário de dados da tabela "Aluguer"	24
Tabela 17 - Bytes ocupados por dígito	46
Tabela 18 - Bytes ocupados pelo tipo DECIMAL(X,Y)	47
Tabela 19 - Bytes ocupados pelo tipo VARCHAR(X)	47
Tabela 20 - Bytes ocupados pela tabela Filial	47
Tabela 21 - Bytes ocupados pela tabela Funcionario	48
Tabela 22 - Bytes ocupados pela tabela Funcao	48
Tabela 23 - Bytes ocupados pela tabela Exerce	48
Tabela 24 - Bytes ocupados pela tabela Cliente	48
Tabela 25 - Bytes ocupados pela tabela Cliente_Contacto	49
Tabela 26 - Bytes ocupados pela tabela Automovel	49
Tabela 27 - Bytes ocupados pela tabela Aluguer	49
Tabela 28 - Tamanho ocupado pela BD BelaRentacar	50
Tabela 29 - Crescimento da BD BelaRentacar	50

# **1. Definição do Sistema**

## **1.1 Contextualização**

A empresa Bela Rent-A-Car Lda. estava inicialmente localizada numa pequena rua na cidade de Braga. Depois do falecimento do pai, Olavo Faísca, o seu filho mais velho, Octávio Faísca decidiu tomar conta do negócio de aluguer de automóveis e com o auxílio de um velho caderno que o pai lhe deixou. Inicialmente, as coisas corriam bem, contudo ambicioso como se caracterizava Octávio Faísca decidiu expandir. Saiu então do seu país, Portugal e abriu uma filial na Bélgica e outra no Mónaco. Escolheu o Mónaco pois abrir lá uma empresa era o sonho do seu pai e quando a oportunidade se apresentou, Octávio não pensou duas vezes antes de avançar. Escolheu a Bélgica, por motivos menos nobres. Octávio Faísca tinha uma grande paixão por chocolates belga e aliado a isso, e o grande motor desta decisão, a Bélgica é um excelente polo de negócio. Um dos grandes trunfos da Bela Rent-A-Car Lda. era precisamente permitir que os clientes levantassem um carro numa filial e o entregassem noutra, como por exemplo alugar em Portugal e devolver na Bélgica, algo que tornava a empresa ideal para férias longas e viagens internacionais. Octávio Faísca encarou então o desafio e trabalhou para garantir que toda a gente tem o carro que quer e precisa, onde quer e para tal procurou ajuda de um grupo de alunos de Engenharia Informática para desenvolver um Sistema de Base de Dados (SBD) para a sua empresa.

## **1.2 Motivação**

Inicialmente, o negócio fluía em todas as filiais, no entanto não demorou muito para Octávio Faísca, gerente, começar a ter problemas. Octávio tinha crescido na empresa, naquela pequena rua, estava habituado aos mesmos clientes recorrentemente, estes eram pessoas em que confiava então não tinham muitas preocupações, apenas anotava no caderno o nome e confiavam que a pessoa lhes retornaria o carro no dia combinado, à hora combinada. No entanto, tendo clientes novos, nem sempre isso acontecia e o registo manual mostrou-se ineficiente e também a causa de alguns erros. Como cada vez que aparecia um cliente, apenas anotava o seu nome, isto originou dados redundantes inexatos e uma grande dificuldade no acesso a dados antigos uma vez que à medida que a quantidade aumentava tornava-se mais complicado fazer a gestão dos mesmos.

O velho caderno com dicas e onde o seu pai tinha feito todos os seus negócios mostrava-se insuficiente, principalmente porque o caderno não conseguia estar em três localizações ao mesmo tempo. Então, foi decidido que cada localização passaria a usar uma forma de armazenar dados diferente, sendo elas: CSV, JSON, uma base de dados relacional em PostgreSQL. Devido à ineficiência e inexperiência de diversos funcionários com estes meios implementados aconteceram vários problemas, desde automóveis que estavam no país errado, clientes que pagaram a quantidade de tempo errada e não cumpriram o tempo de aluguer combinado. E o facto de haver três fontes diferentes de dados estava a criar ainda mais confusões e preocupações no serviço de alugar um carro numa filial e devolver noutra filial.

Octávio esforçava-se, mas o acumular de circunstâncias culminou em situações de furto de automóveis. Agora mais que nunca, o gerente principal estava desesperado por uma solução para trazer o seu negócio às preces de outro-horas e não ser uma desilusão para o seu falecido pai.

Assim, quando viu no *TikTok* um vídeo a falar sobre Base de Dados (BD), este julgou que um SGBD seria o necessário para ressuscitar e melhorar o negócio de forma que os dados já existentes pudessem ser migrados e os seus funcionários passassem a utilizar uma Base de Dados central que evitaria e resolveria diversos dos problemas que foram criados quando este tentou resolver as coisas com as diferentes fontes de dados.

Após recrutarem quatro alunos de Engenharia Informática na mesma universidade onde tinha estudado Gestão. O mesmo reuniu-se com os alunos e pediu que desenvolvessem um SBD que se adaptasse ao negócio em cada país, mas que estivesse a empresa interligada tendo como fonte os dados já existentes.

## 1.3 Objetivos

Inspirados pelos acontecimentos recentes, Octávio Faísca, gerente principal definiu um conjunto de objetivos que pretende alcançar com a implementação do futuro Sistema de Gestão de Base de Dados do seu negócio de aluguer de automóveis, nomeadamente:

- Organizar o modelo de negócio, bem como melhorar a sua capacidade e registo de automóveis alugados
- Saber a cada momento que carro é que está em cada loja, quais estão alugados e quais não estão.
- Obter uma ferramenta útil na gestão eficiente do negócio, tanto no que toca a recursos financeiros como humanos, com vista a acelerar o crescimento.
- Tornar mais simples e eficaz a gestão e o contacto com os clientes de forma a evitar informação duplicada ou informação incorreta, melhorando a integridade e confiança dos dados.
- Permitir melhorar a qualidade do serviço no geral de forma a manter os clientes satisfeitos.
- Otimizar a sincronização.

## 1.4 Viabilidade

A empresa de aluguer de carros e comerciais, com funcionamento num velho caderno apresenta várias falhas e problemas de produtividade e segurança que foram documentadas por Octávio Faísca através da sua experiência e da análise intensiva do caderno do seu falecido pai. Assim, Bela Rent-A-Car Lda. estima que ao ter um sistema mais eficiente de armazenamento e gestão de dados conseguirá:

- Recuperar, cerca de 10% do prejuízo do último semestre de forma a suportar o custo de desenvolvimento do sistema e investir em novos automóveis para melhorar o negócio.
- Recuperar os fluxos financeiros, evitar furtos e evitar erros humano, saber a cada momento o que cada cliente alugou, e o tempo de aluguer para retificar o valor cobrado de forma a não existirem erros.
- Melhorar a alocação de recursos ao conhecer os hábitos de aluguer dos clientes, através da análise dos automóveis alugados e da altura do ano, pretende-se saber as alturas em que são necessários mais automóveis de forma que no tempo restante os automóveis possam ser alocados para outra filial ou para outra função de forma a aumentar o lucro.
- Aumentar a segurança dos sistemas.

Assim, Octávio Faísca, diretor geral da empresa, concluiu que a implementação de uma SBD era o essencial para salvar o seu negócio. E como Bela Rent-A-Car Lda. não estava a lucrar assim tanto, decidiu que mão de obra barata de estudantes desesperados para ter projetos no *Github* era o ideal para a execução dessa tarefa.

## 1.5 Recursos e Equipa de trabalho

Os recursos humanos necessários para este projeto podem ser divididos em dois: o “pessoal interno” constituído pelos funcionários da empresa Bela Rent-A-Car Lda. E o “pessoal externo” constituído pela equipa de desenvolvimento da SBD.

O “pessoal interno” é composto pelas seguintes pessoas que representam a empresa no processo de desenvolvimento da SBD:

- Octávio Faísca – Diretor da empresa Bela Rent-A-Car Lda., gestor da filial de Portugal. Tem uma vasta experiência em gestão e conhece profundamente as necessidades da sua empresa. Apesar de viajar bastante entre as suas filiais passa a maior parte do tempo em Portugal e por isso tem contacto direto com a equipa de desenvolvimento.

- Leonidas Lindt – Gestora da Filial Belga, responsável pelos negócios realizados no país. Apresenta habilidades organizacionais e conhece as necessidades dos seus clientes.
- Alberto Senna – Gestor da Filial do Mónaco, responsável pelos negócios que acontecem no país. Tem uma vasta experiência em todo o tipo de automóveis e é responsável.

O “pessoal externo” é constituído pela equipa de alunos de Engenharia Informática escolhida por Octávio Faísca. Estes são responsáveis pelo desenvolvimento do SBD. Os nomes dos integrantes desta equipa são Carolina Martins, Diogo Ribeiro, Filipa Gonçalves e Lucas Robertson.

Já os recursos materiais podem dividir-se em dois tipos, *software* e *hardware*. Em termos de *software*: *brModelo*, Gestor de Sistemas de base de dados, *MySQL*, *Python*, Software necessário para realização de inquéritos.

Em termos de *hardware*, é fundamental existirem 4 computadores pessoais, um por cada elemento da equipa de desenvolvimento, com o propósito de servirem a elaboração do SBD. Estes serão providenciados pelos alunos em questão.

## 1.6 Plano de Execução

A equipa de desenvolvimento reuniu-se com Octávio Faísca, Dono, diretor geral e gerente da filial de Portugal de Bela Rent-A-Car para planear o desenvolvimento do SBD. A partir dessa reunião foi elaborado um diagrama de *Gantt* onde se definiu o período de execução e os responsáveis por cada tarefa. Isto foi efetuado tendo em conta os prazos de entrega solicitados por Octávio Faísca e os períodos em que a equipa não poderia trabalhar a tempo inteiro no projeto.

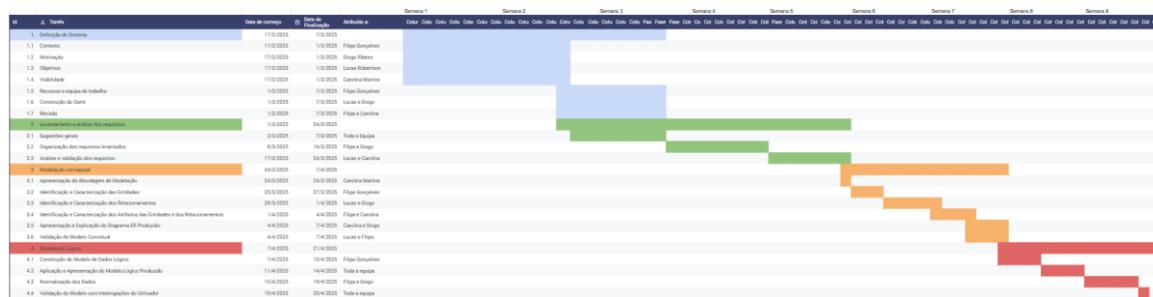


Figura 1 - Miniatura do Diagrama de Gantt



Figura 2 - Cabeçalho da ata da reunião de construção do cronograma para o desenvolvimento da BD.

## 1.7 Revisão e Aprovação

Após melhorar o plano de execução, a equipa de desenvolvimento formada por estudantes de engenharia informática da Universidade do Minho agendou uma reunião com os gerentes das filiais de Bela Rent-A-Car Lda. Estes confirmaram tudo o que foi definido e decidiram então fechar contrato com a equipa de desenvolvimento por um valor de 0€ para desenvolver o Sistema de Gestão de Base de Dados proposto.



Figura 3 - Cabeçalho da ata da reunião de revisão do projeto da BD.

## 2. Levantamento e Análise de Requisitos

### 2.1 Sugestões Gerais

Para o levantamento de requisitos, a equipa de desenvolvimento adotou diversas estratégias. Foi decidido que para além de serem realizadas diversas reuniões com os membros da Bela Rent-A-Car Lda. seria feito um inquérito a clientes dos diversos países para perceber o que se poderia melhorar em cada um deles. O velho caderno seria analisadometiculosamente de forma a entender como é que a empresa funcionava habitualmente, mas acabou também por se considerar necessário fazer um inquérito a Octávio Faísca, como gerente principal da empresa.

#### Reuniões com Bela Rent-A-Car Lda.

Foram marcadas 2 reuniões com a equipa principal de Bela Rent-A-Car composta pelo seu diretor principal e gerente da filial portuguesa, Octávio Faísca, a gerente da filial belga, Leonidas Lindt e o gerente da filial do Mónaco, Alberto Senna.

Na primeira reunião, o diretor principal da empresa explicou o que pretendiam com o SBD a ser desenvolvido num conceito geral. Já os gerentes das filiais explicaram como cada filial da empresa funcionava, que informações era necessário armazenar e consultar, como deveria ser controlado o acesso a dados sensíveis.

A equipa de desenvolvimento anotou a informação necessária para o futuro desenvolvimento da BD, bem como as suas fontes, guiando a equipa da empresa para fornecer todos os dados essenciais com correção e completude.

	<b>Recolha de requisitos</b>
	<b>DATA:</b> 02/03/2025
	<b>HORA:</b> 10:00H
<b>Participantes</b>	<b>Objetivo</b>
<ul style="list-style-type: none"><li>• Octávio Faísca (Diretor principal e Gerente Portugal)</li><li>• Leonidas Lindt (Gerente Monaco)</li><li>• Alberto Senna (Gerente Bélgica)</li><li>• Diogo Ribeiro (Equipa de desenvolvimento)</li></ul>	<ul style="list-style-type: none"><li>• Comunicação do que é pretendido pela Belos Carros Lda. com o Sistema de Gestão de Bela Rent-A-Car</li></ul>

Figura 4 - Cabeçalho da ata da reunião de levantamento de requisitos.

#### Inquérito a Octávio Faísca

De forma a retificar os requisitos levantados e como a equipa de desenvolvimento queria certificar-se que o dono da empresa ficava feliz com o seu trabalho especialmente tendo em conta que estamos a falar de

países diferentes com necessidades diferentes. Foi enviado um inquérito com perguntas relativas ao seu trabalho, que tipo de tarefas executavam, o que consideravam que seria necessário para o SBD entre outras.



**Inquérito para a Base de Dados**

**PARA:** Octávio Faísca

- Quais são as suas expectativas para a SBD?
- Descreva as tarefas que faz ao longo de um dia de trabalho.
- Quais das tarefas que referiu anteriormente acha que podem ser automatizadas?
- Sendo gestor de uma filial no país X, pensa que os outros gestores podem ter acesso à sua filial?
- Na sua filial, quem considera que deve ter acesso ao quê? Isto é, considera que as secretarias podem alugar carros, entre outros.

Figura 5 - Inquérito a Octávio Faísca

Com base nas respostas às perguntas, foram formulados e aprimorados vários requisitos.

#### **Inquérito a alguns clientes**

De forma a entender melhor as falhas do negócio e como este poderia melhorar com a implementação de uma SBD, vários clientes responderam a um inquérito que foi posto nas três lojas. Este inquérito permitiu entender diversos problemas que os clientes sentiam quando se dirigiam à Bela Rent-A-Car e a partir das respostas foram formulados diversos requisitos.

**Opinião sobre Bela Rent-A-Car**

Caro cliente, muito obrigado por realizar este inquérito.  
Esperamos que a sua experiência connosco tenha sido agradável e que os nossos serviços tenham estado de acordo com as expectativas.  
Estamos sempre à procura de melhorar, e queremos saber a sua opinião.

\* Indica uma pergunta obrigatória

Refere o país em que costuma aceder aos serviços de Fáscia Rent-A-Car Lda. \*

Portugal  
 Bélgica  
 Mónaco

Que automóvel alugou? \*

A sua resposta

Sentiu que o serviço esteve à altura do que era esperado? \*

1 2 3 4 5  
Mau      Excelente

Correu alguma coisa mal? Se sim, refira o quê. \*

A sua resposta

O que melhoraria no serviço. \*

A sua resposta

**Enviar** **Limpar formulário**

Figura 6 - Inquérito feito aos clientes de Bela Rent-A-Car

Refere o país em que costuma aceder aos serviços de Fáscia Rent-A-Car Lda.	Que automóvel alugou?	Sentiu que o serviço esteve à altura do que era esperado?	Correu alguma coisa mal? Se sim, refira o quê.	O que melhoraria no serviço.
Portugal	Audi A5	3	O carro não estava disponível às horas combinadas o que causou diversos constrangimentos	A organização
Portugal	Mercedes	4	Não correu nada mal, mas os funcionários pareciam confusos e só andavam á volta de um caderno.	Arranjar algum tipo de sistema eletrónico para os ajudar
Bélgica	Toyota	5	Tout s'est bien passé. Ils sont si mignons	Rien
Mónaco	Mercedes	4	Nothing was bad, but the employees seemed really confused.	They need a better manager and some organization
Bélgica	Audi	1	De auto die ik gehuurd had was niet beschikbaar, ik ga daar niet meer heen	Alle
Bélgica	Hyundai	3	Ze gaven me een andere auto van hetzelfde merk, maar niet de auto die ik wilde huren	verschillende dingen

Figura 7 - Algumas respostas ao Inquérito feito aos clientes

### Análise de documentação interna

Foi concedido, à equipa de desenvolvimento, acesso ao velho caderno. Ao analisar o objeto, a equipa extraiu informações referentes ao modo como a empresa operava internamente e como o SBD aumentava o lucro e a organização na Bela Rent-A-Car Lda. Em particular foram analisados os vários problemas que houve em relação a carros no país errado e concluiu-se que tal se devia à falta da correta identificação dos carros em diferentes países. Assim, uma má catalogação causou bastantes problemas no negócio. A documentação analisada não pode ser divulgada aos docentes uma vez que a equipa de desenvolvimento teve de assinar um acordo de não divulgação.

### Analise das fontes de dados já existentes

Para além da análise de documentação interna a equipa de desenvolvimento também teve acesso às diferentes formas que as filiais tentaram usar para gerir os seus dados. Estas foram analisadas intensivamente, tendo observado diversas lacunas e falta de coerência.

## 2.2 Organização dos requisitos levantados

À medida que o processo de levantamento de requisitos ia avançando, a equipa de desenvolvimento, tendo como base a tabela global que foi com o auxílio dos processos anteriormente falados, começou a dividir os requisitos.

Nº	Data	Hora	Tipo	Descrição	Vista de Utilização	Fonte	Analista
1	03/03/2025	15:15h	Descrição	Um cliente é caracterizado pelo seu nome, morada(rua, localidade, código-postal), NIF, local de trabalho e contactos.	Cientes	Octávio Faísca	Filipa Gonçalves
2	03/03/2025	15:17h	Descrição	Um cliente deve ser identificado por um número sequencial único.	Cientes	Leonidas Lindl	Filipa Gonçalves
3	03/03/2025	15:21h	Descrição	Os contactos dos clientes e dos funcionários podem ser diversos: números de telefone e endereços de email distintos entre clientes/funcionários. É obrigatório pelo menos um número de telefone.	Cientes/Gestão	Octávio Faísca	Carolina Martins
4	03/03/2025	15:22h	Descrição	Um Funcionário é caracterizado por um identificador sequencial único, o nome, NIF, o salário e os contactos.	Gestão	Octávio Faísca	Lucas Robertson
5	03/03/2025	15:23h	Descrição	Um automóvel é caracterizado por um identificador sequencial único, a marca, a kilometragem, o ano, o estado(Disponível, Ocupado), o tipo de consumo(Gasolina, Gasóleo, Diesel, ...) e o preço por dia do respetivo automóvel.	Automóveis	Alberto Senna	Diogo Ribeiro

Figura 8 - Primeiros requisitos da tabela global

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RM1	12	Registrar novos clientes e alterar os dados dos existentes	Gestão	Filipa Gonçalves
RM2	13	Registrar novos alugueres e fazer alterações aos já existentes	Descrição	Diogo Ribeiro
RM3	15	Adicionar novos funcionários, alterar dados já existentes e remover funcionários existentes	Gestão	Filipa Gonçalves
RM4	11, 17	Inserir novas funções de funcionários	Gestão	Carolina Martins
RM5	16	Verificar o estado de um determinado automóvel, isto é, ver se ele está ocupado ou disponível.	Automóveis/Gestão	Lucas Robertson

Figura 9 - Primeiros requisitos da tabela de requisitos de manipulação

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RD1	1	Um cliente é caracterizado pelo seu nome, morada(rua, localidade, código-postal), NIF, local de trabalho e contactos.	Cientes	Diogo Ribeiro
RD2	2	Um cliente deve ser identificado por um número sequencial único.	Cientes	Carolina Martins
RD3	7	Um cliente pode alugar mais do que um automóvel ao mesmo tempo, e um automóvel apenas pode ter um cliente associado	Cientes	Filipa Gonçalves
RD4	3	Os contactos dos clientes podem ser diversos: números de telefone e endereços de email distintos entre clientes. É obrigatório pelo menos um número de telefone.	Cientes	Lucas Robertson
RD5	5	Um automóvel é caracterizado por um identificador sequencial único, a marca, a kilometragem, o ano, o estado, o tipo de consumo e o preço por dia do respetivo automóvel.	Automóveis	Filipa Gonçalves

Figura 10 - Primeiros requisitos da tabela de requisitos de descrição

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RC1	32	Um aluguer apenas é referente a um único automóvel	Gestão	Filipa Gonçalves
RC2	19	Clientes que não estão registados não podem alugar automóveis.	Cientes/Gestão	Filipa Gonçalves
RC3	19	Automóveis que não estão registados não podem ser alugados	Automóveis/Gestão	Carolina Martins
RC4	20	Todos os funcionários podem registar alugueres uma vez que estão todos autorizados.	Gestão	Lucas Robertson
RC5	21	Todos os funcionários podem multar clientes.	Gestão	Diogo Ribeiro

Figura 11 - Primeiros requisitos da tabela de requisitos de controlo

Ao longo do processo de organização de requisitos foram estabelecidas quatro vistas de utilização distintas, que foram denominadas "Clientes", "Automóveis", "Alugueres" e "Gestão".

A vista “Cliente” concerne à interação com os clientes da empresa Bela Rent-A-Car. Os clientes não possuem acesso à BD, mas a interação de funcionários com os mesmos exige várias funcionalidades do SBD.

A vista “Automóveis” está relacionada ao produto que a empresa trabalha. Esta vista tem em consideração os aspetos dos automóveis.

A vista “Alugueres” está relacionada com a recolha e manipulação dos dados necessários para o aluguer de automóveis. A maioria das necessidades que compõe esta vista de utilização está relacionada com o trabalho dos funcionários da empresa.

A vista “Gestão” tem que ver com a gestão de funcionários, e os aspetos financeiros da empresa de forma a serem tomadas as melhores decisões possíveis para o negócio. O Diretor principal e os gestores de cada filial são os interessados nestas funcionalidades.

## 2.3 Analise e Validação Geral dos requisitos

Após uma minuciosa e cuidada revisão do documento de requisitos por parte da equipa informática, foi realizada uma reunião com os gestores de Bela Rent-A-Car Lda. Para validar todas as vertentes de utilização e cada requisito.



Figura 12 - Cabeçalho da ata da reunião de validação de requisitos.

## **3. Modelação Conceptual**

### **3.1 Apresentação da Abordagem de Modelação realizada**

Dando como concluída a recolha e análise dos requisitos, a equipa de desenvolvimento iniciou o planeamento da estrutura e design da BD. Começaram por construir um diagrama ER, de forma a terem uma visão das entidades, dos seus atributos e das relações que poderiam existir. Para obter esta representação foi usado o *brModelo*, que produz diagramas numa variante da notação *Chen*.

Assim, a equipa começou por identificar as entidades e os relacionamentos entre elas. Só depois é que caracterizou os atributos e descreveu os relacionamentos.

### **3.2 Identificação e Caracterização das Entidades**

A identificação de entidades é algo indispensável para o desenvolvimento de uma base de dados. Estas são a base de organização de uma BD e é necessário identificá-las antes de identificar relacionamentos e atributos. Uma entidade é refere-se a um conjunto de objetos com as mesmas propriedades, objetos esses com uma existência independente, seja esta física (tangível) ou apenas conceitual (Connolly & Begg, 2015, pp. 406-407). Esta definição vaga conduz a que possa haver diferentes conjuntos de entidades deduzidos a partir do mesmo documento de requisitos, pelo que a existência de cada entidade é devidamente justificada neste documento.

A primeira entidade identificada foi o “Cliente” e foi identificado em RD1, esta é uma entidade relativa aos indivíduos que procuram os serviços de aluguer de Bela Rent-A-Car Lda. Uma vez que existem vários clientes e que estes são independentes uns dos outros apesar de compartilharem várias propriedades. “Cliente” foi considerado uma entidade.

De seguida, foi identificado a entidade “Automóvel” em RD5 esta representa aquilo que os clientes procuram alugar quando se dirigem à Bela Rent-A-Car Lda. Foi considerada uma entidade pois existem vários automóveis de forma independente e que partilham as mesmas definições de características.

Aliado a estas duas entidades, “Aluguer” foi a entidade definida em RD15 que representa o único tipo de serviço prestado pela Bela Rent-A-Car Lda. Esta tem um papel de extrema importância na BD e em particular na vista de utilização com o mesmo nome. Foi considerada uma entidade dado que vários casos existem de forma independente, mas partilham entre si as mesmas características que estão presentes em RD15.

Aliada a estas, identifica-se a entidade “Funcionário” que é uma peça fundamental para o funcionamento da empresa de alugueres. Esta é uma peça vital para a gestão de recursos humanos (RH) na empresa (vista

“Gestão”). Tem uma existência tangível e independente, as suas características comuns a todos aqueles que são empregues pela empresa estão enumeradas em RD10.

A entidade “Função” descrita em RD12 é usada para caracterizar funções/cargos que cada “Funcionário” exerce (por exemplo, “Gestor”, “Secretária”, etc.). Não se justifica que “Função” seja um mero atributo de “Funcionário” pois a caracterização de “Funcionário” em RD10 não indica que este seja caracterizado por ela. Aliado a isto, ter uma entidade “Funcao” permite um maior controlo dos cargos que podem ser exercidos por um trabalhador e facilita a adição ou remoção de novas posições dentro da empresa.

Por último, a entidade “Filial” surge em RD8. Esta é uma entidade relativa às diversas localizações da empresa Bela Rent-A-Car Lda. Uma vez que já existem várias filiais e que existe a possibilidade de o número vir a aumentar, a equipa de desenvolvimento achou que filial não poderia ser apenas um simples atributo de “Automovel”, “Aluguer” ou “Funcionario”. Isto é algo que também permite evitar erros humanos.

Entidade	Definição	Sinónimos	Requisito
<b>Cliente</b>	Pessoa que solicita os serviços de Bela Rent-A-Car Lda.		RD1
<b>Automóvel</b>	Único produto disponível para aluguer na empresa Bela Rent-A-Car Lda.	Carro, Comercial	RD4
<b>Aluguer</b>	Único serviço prestado pela empresa Bela Rent-A-Car Lda.		RD15
<b>Funcionário</b>	Pessoa que trabalha na empresa Bela Rent-A-Car Lda.	Trabalhador	RD10
<b>Função</b>	Tipo de trabalho/cargo que um funcionário executa		RD12
<b>Filial</b>	Local onde a empresa exerce as suas atividades.		RD8

Tabela 1 - Entidades identificadas pela equipa de desenvolvimento

### 3.3 Identificação e Caracterização dos Relacionamentos

A identificação dos relacionamentos também é feita com base nos requisitos. Segue-se então uma fase de leitura não só dos requisitos de descrição, mas também dos restantes requisitos, de controlo e manipulação, pois estes podem conter informação para determinar características de um relacionamento.

Começamos por o requisito RD3 pois associa um “Cliente” a um “Aluguer”, é explícito que um cliente pode alugar vários automóveis, mas um automóvel apenas pode ser alugado por um cliente por vez. Assim, consideramos a cardinalidade deste relacionamento como muitos-para-um. Aliando a isto, não faz sentido a existência de um cliente que nunca alugou ou de um aluguer que não tenha sido feito. Portanto, este relacionamento binário tem participação total por parte das duas entidades.

Dando continuidade, vamos ter em conta o requisito RD14 que mostra a existência de um relacionamento entre as entidades “Funcionario” e “Aluguer” (“Funcionario” “Regista” “Aluguer”). Podemos também concluir que a relação de cardinalidade é um-para-muitos uma vez que cada

aluguer apenas pode ser registado por um único funcionário, apesar de um funcionário poder registar múltiplos alugueres. Já a participação, consideramos que esta seja total-parcial uma vez que podem existir funcionários sem existirem alugueres, mas o contrário não é verdade.

Ainda sobre a entidade “Aluguer”, podemos concluir que esta se relaciona com a entidade “Automovel”. A cardinalidade deste relacionamento é de muitos-para-um uma vez que apesar de se poder alugar o mesmo automóvel múltiplas vezes, um aluguer é apenas referente a um automóvel, isto é, não se pode alugar dois automóveis na mesma instância de aluguer como está referido no requisito RC1. Em relação à participação consideramos que esta é total-parcial uma vez que podem existir automóveis que não estão alugados ou que nunca foram alugados, mas não existe um aluguer sem um automóvel.

Mudando o foco para a entidade “Automovel” podemos concluir que este se encontra relacionado com a entidade “Filial” a partir do requisito RD9. A cardinalidade deste relacionamento é de muitos-para-um uma vez que existem vários automóveis, mas cada automóvel apenas pode estar localizado numa filial. Já em relação à participação, consideramos que esta seja total-total pois não faz sentido a existência de automóveis que não estejam associados a uma filial e não existe uma filial sem automóveis.

No registo RD13 temos um “Funcionario” exerce “Funcao”. Estas duas entidades encontram-se relacionadas com uma cardinalidade de muitos-para-muitos explícita no requisito referido. Em relação à participação, consideramos que seja total-total pois não faria sentido ter um funcionário que não exercesse uma função.

Em relação à entidade “Funcionario”, temos ainda “Funcionario” trabalha numa “Filial” como está referido em RD16. Estas duas entidades encontram-se relacionadas com uma cardinalidade muitos-para-um, uma vez que cada funcionário pode estar associado a apenas uma filial, mas uma filial pode ter muitos funcionários associados. No que toca à participação, consideramos que seja total-total pois não faria sentido ter um funcionário que não trabalhasse numa filial.

De forma a dar suporte à criação do diagrama ER, a equipa de desenvolvimento compilou a informação descrita na seguinte tabela. É de salientar que na coluna dos requisitos foram colocados mais requisitos para além dos requisitos de descrição, pois, como se referiu anteriormente, estes ajudam a aprimorar as conclusões em relação à cardinalidade e à participação

Entidade	Relacionamento	Cardinalidade	Participação	Entidade	Requisitos	Descrição
<b>Funcionario</b>	Regista	1:N	P:T	<b>Aluguer</b>	RD14	Funcionário regista um aluguer.
<b>Aluguer</b>	Referente	N:1	T:P	<b>Automovel</b>	RD18	O aluguer solicitado é referente a um automóvel
<b>Aluguer</b>	Feito por	N:1	T:T	<b>Cliente</b>	RD3	Cliente solicita à

						empresa o aluguer.
<b>Automovel</b>	localizado	N:1	T:T	<b>Filial</b>	RD9	Um automóvel encontra-se localizado numa filial
<b>Funcionario</b>	exerce	N:N	T:T	<b>Função</b>	RD13	Funcionário executas tarefas de uma função
<b>Funcionario</b>	Trabalha em	N:1	T:P	<b>Filial</b>	RD16, RD17	Funcionário trabalha numa Filial.
<b>Aluguer</b>	Iniciado numa	N:1	T:P	<b>Filial</b>	RD19	Um aluguer é iniciado numa filial
<b>Aluguer</b>	Finalizado numa	N:1	T:P	<b>Filial</b>	RD20	Um aluguer é finalizado numa filial.

Tabela 2 - Relacionamentos identificados pela equipa de desenvolvimento

### 3.4 Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos

Semelhante às secções anteriores, a documentação relativa aos atributos teve como base os requisitos recolhidos pela equipa de informática. Assim, o processo de identificação e caracterização dos atributos de cada entidade resulta de uma leitura cuidadosa dos requisitos associados as entidades previamente identificadas.

Tendo em conta o requisito RD1, é possível obter de forma bastante direta os atributos relativos à entidade “Cliente”, nomeadamente “Nome”, “NIF”, “Morada”, “LocalTrabalho”. Aliando a estes o requisito RD3, descreve o atributo “Contactos” elaborando que este pode ser constituído por números de telefone e endereços de e-mail, mas que tem de ter obrigatoriamente um número de telefone enquanto o endereço de email é opcional. Consideramos assim que “Contactos” é um atributo composto e este vai possuir dois atributos multivvalorados: “Telefone” e “Email”, sendo que o primeiro tem de conter pelo menos um valor e o último não tem essa exigência.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisit o
<b>Id</b>	Identificador sequencial do cliente	INT	N	1	RD2
<b>Nome</b>	Nome completo do Cliente	VARCHAR(75)	N	João Paulo	RD1
<b>Morada</b>	Nome da localidade, da rua e código-postal		N	-	RD1

<b>Rua</b>	Rua onde o cliente vive	VARCHAR(75)	N	Rua sei lá	RD1
<b>Localidade</b>	Localidade onde o cliente reside	VARCHAR(75)	N	Braga	RD1
<b>Codigo-Postal</b>	Código-postal do local onde o cliente reside	VARCHAR(10)	N	5225-032	RD1
<b>NIF</b>	Número de Identificação Fiscal	VARCHAR (9)	N	1111111222	RD1
<b>LocalTrabalho</b>	Designação do local onde o cliente trabalha	VARCHAR(100)	N	Fábrica de Gomas	RD1
<b>Contactos</b>	Contactos do cliente		N	911333444	RD1
<b>Telefone[1..n]</b>	Lista de números de telefone pessoais do cliente	VARCHAR(20)	N	963547890	RD4
<b>Emails [0..n]</b>	Lista de Endereços eletrónicos.	VARCHAR(200)	S	Jp123g@mail.com	RD4

Tabela 3 - Atributos da entidade "Cliente"

Tendo como base os requisitos RD5 para a entidade “Automóvel” podemos enumerar os atributos imediatos associados a esta entidade, nomeadamente “Id”, “Marca”, “Kilometros”, “Ano” e “TipodeConsumo”.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisito
<b>Id</b>	Identificador sequencial do Funcionário	INT	N	3	RD5
<b>Marca</b>	Marca e modelo do carro	VARCHAR(75)	N	BMW	RD5
<b>Kilometragem</b>	Kilometros que o carro já fez	DECIMAL(8,3)	N	200000km	RD5
<b>Ano</b>	Ano do carro	YEAR	N	2020	RD5
<b>Estado</b>	Estado atual do automóvel	VARCHAR(10)	N	Disponível	RD5; RD7
<b>TipodeConsumo</b>	Tipo de combustível utilizado	VARCHAR(10)	N	Elétrico	RD5; RD6
<b>PrecoDia</b>	Preço diário do aluguer do automóvel	DECIMAL(8,2)	N	56,00	RD5
<b>Restrições</b>					
<b>Estado IN {Disponível, Ocupado}</b>					
<b>TipodeConsumo IN {Gasolina, Gasóleo, Elétrico, Híbrido}</b>					

Tabela 4 - Atributos da entidade "Automóvel"

A entidade “Aluguer” é caracterizada com base nos requisitos RD15. Esta, possui os atributos, “Id”, “PrecoFinal”, “DataInicial”, “DataFinal” por fim possui um atributo opcional “Multa”.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisito
<b>Id</b>	Identificador sequencial do Aluguer	INT	N	2	RD15
<b>PrecoFinal</b>	Preço final do aluguer	DECIMAL(10,2)	N	700	RD15
<b>DataIncial</b>	Data do início do aluguer	DATETIME	N	12-03-2023	RD15
<b>DataFinal</b>	Data do fim do aluguer	DATETIME	N	12-04-2023	RD15

<b>Multa</b>	Preço da possível multa a ser paga	DECIMAL(5,2)	S	235.00	RD15
--------------	------------------------------------	--------------	---	--------	------

Tabela 5 - Atributos da entidade "Aluguer"

A entidade “Funcionario” apresenta os seguintes atributos que podem ser encontrados no requisito RD10. Assim, esta tem os seguintes atributos, “Id”, “NIF”, “Salario”, “Nome”, “Filial” e “Contactos”. Todos os atributos com a exceção deste último são considerados atributos simples não nulos. Já “Contactos” é um atributo composto e os seus subatributos serão “Telefone” que é referido como “obrigatório” e “Email” que é referido como “opcional. Temos assim a seguinte tabela sumário.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisito
<b>Id</b>	Identificador sequencial do Funcionário	INT	N	2	RD10
<b>NIF</b>	Número de Identificação Fiscal	VARCHAR(9)	N	114546786	RD10
<b>Salário</b>	Salário de um Funcionário	DECIMAL(8,2)	N	345,34	RD10
<b>Nome</b>	Nome completo do Funcionário	VARCHAR(75)	N	Maria João	RD10
<b>Contactos</b>	Contactos do Funcionário	-	N	964376855	RD10
<b>Telefone</b>	Lista de números de telefone pessoais do Funcionário	VARCHAR(20)	N	911234987	RD11
<b>Email</b>	Lista de Endereços eletrónicos	VARCHAR(100)	S	Joaomaria11@faiscarentacar.com	RD11

Tabela 6 - Atributos da entidade "Funcionario"

Ligada à entidade “Funcionario” temos a entidade “Função”. Esta é a entidade mais simples de caracterizar uma vez que apenas possui um identificador único, “Id”, uma designação, “Designacao” e um “SalarioBase”. Isto pode ser verificado no requisito RD12 onde vemos também que ambos os atributos são simples e não opcionais.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisito
<b>Id</b>	Identificador sequencial do cliente	INT	N	3	RD12
<b>Designacao</b>	Designação de uma função	VARCHAR(75)	N	Gestor	RD12
<b>SalarioBase</b>	Salário base da função	DECIMAL(8,2)	N	56,3	RD12

Tabela 7 - Atributos da entidade "Funcao"

Por fim, a única entidade em falta é “Filial” que é caracterizada no requisito RD8. Esta apresenta assim os atributos “Id”, “Local”.

Atributo	Descrição	Domínio	Nulo	Exemplo	Requisito
<b>Id</b>	Identificador sequencial da Filial	INT	N	1	RD8
<b>Local</b>	País em que se localiza a filial	VARCHAR(75)	N	Portugal	RD8

Tabela 8 - Atributos da entidade "Filial"

O leitor pode estar a perguntar-se qual a origem do domínio atribuído a cada atributo.

Para atributos em que serão guardados valores inteiros, atribui-se o tipo de dados INT (INTEGER).

Para guardar valores não inteiros, isto é, decimais foram escolhidos os tipos DECIMAL(10,2) e DECIMAL(5,2). O número dois representa as casas decimais onde serão armazenados cêntimos e os números dez e cinco representam o total de algarismos que constituem o número. A diferença entre os dois números representará as casas decimais onde serão armazenados os euros.. Um exemplo da utilização destes domínios está na entidade “Aluguer” nos atributos “PrecoFinal” e “Multa”.

Para armazenar datas, como é o caso dos atributos “DataIncial” e “DataFinal” da entidade “Aluguer” foi escolhido DATETIME uma vez que para além da informação referente à data guarda também informação horaria e a equipa de desenvolvimento considerou que perante o presente negocio seria uma informação necessária.

Para guardar texto, não se considerou que algum dos atributos armazenado necessitasse de muitos caracteres então utilizou-se o tipo VARCHAR pois este suportaria apenas o número de caracteres que a equipa de informática considerasse necessário. Este valor é sempre restringido, mas salientamos o cado de “NIF” porque independentemente da entidade é sempre constituído por 9 dígitos. Se houvesse dados maiores, a equipa escolheria usar o tipo TEXT.

### **3.5 Apresentação e Explicação do Diagrama ER Produzido**

Após a identificação e caracterização das entidades, dos seus atributos, a equipa de desenvolvimento procedeu à esquematização de um diagrama ER. Esta esquematização foi feita recorrendo ao software *brModelo* de forma a gerar de forma rápida e eficaz um diagrama conceptual em formato digital.

O processo de desenvolvimento foi maioritariamente dividido em duas etapas principais. Primeiro foram colocadas todas as entidades e os seus atributos no software mencionada. O resultado pode ser verificado na figura abaixo apresentada:

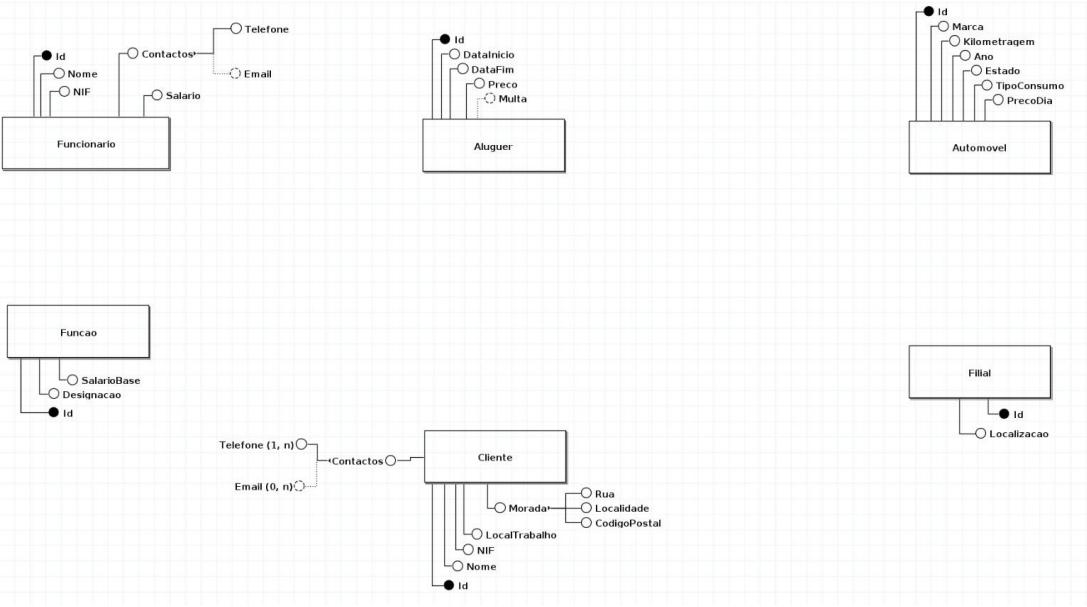


Figura 13 - Entidades do modelo conceptual

De seguida foram adicionados todos os relacionamentos, a cardinalidade e a participação. Para estes dois últimos é de notar que a notação do *brModelo* difere da notação *Chen*.

Por fim, apenas se ajustou o posicionamento dos elementos gráficos de forma a obter uma representação mais fácil de ler. Segue-se o diagrama ER produzido:

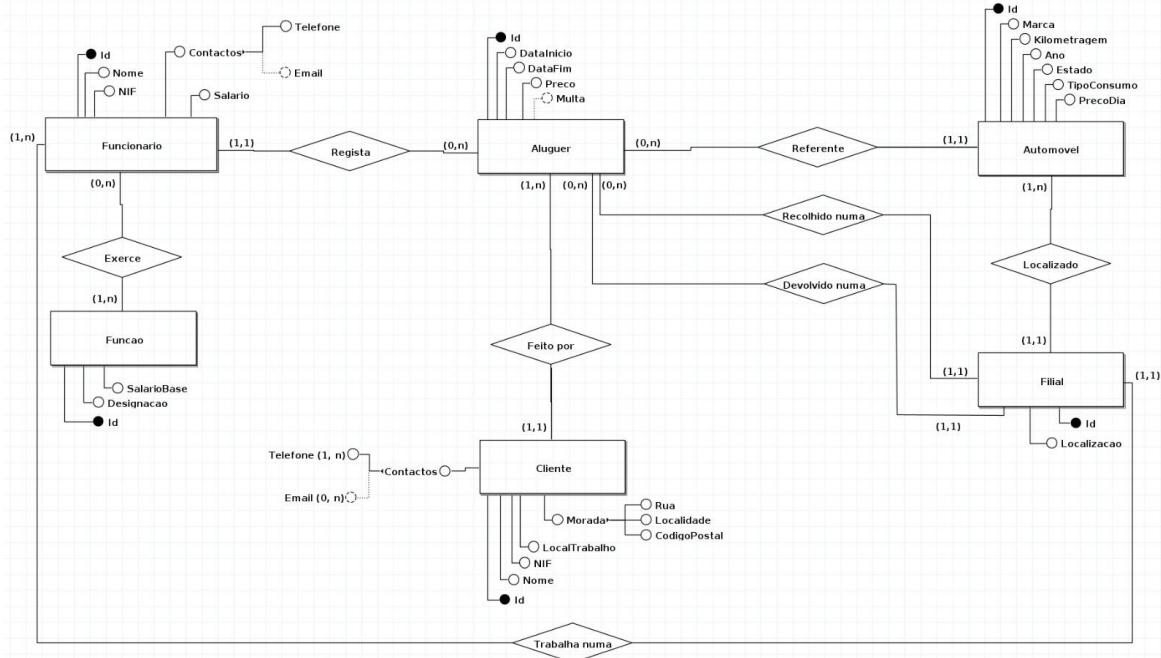


Figura 14 - Versão final do modelo conceptual

### **3.5 Validação do Modelo Conceitual**

Após terminar a elaboração do modelo conceptual e da sua documentação, a equipa de alunos apresentou o seu trabalho ao gerente principal da Bela Rent-A-Car Lda. Este requisitou mudanças principalmente nos atributos da entidade “Aluguer” e questionou também a existência da entidade “Filial”. Alterações foram feitas para satisfazer estes pedidos e foi também melhorada a documentação para não restarem dúvidas em relação à necessidade das entidades e aos seus atributos.

## 4. Modelação Lógica

### 4.1 Construção do Modelo de Dados Lógico

Para a construção do modelo de dados lógico foi utilizada uma abordagem de construção redundante que tinha como principal objetivo evitar erros. Assim, foi feita uma conversão manual pela equipa de desenvolvimento e foi utilizado o *brModelo* para fazer uma conversão automática. Estas conversões foram comparadas

### 4.2 Aplicação e Apresentação do Modelo Lógico Produzido

Para a construção do esquema lógico a equipa de desenvolvimento começou pela realização do processo de derivação de relações a partir do esquema conceptual produzido anteriormente. De forma a obter com uma maior precisão os atributos de cada tabela foram também considerados os atributos compostos em adição aos atributos simples da própria entidade.

De seguida passamos ao manuseamento dos atributos multivalorados. Cada atributo multivalorado dá origem a uma nova tabela com um relacionamento 1:N com a sua tabela de referência. Para além disso, as tabelas geradas contêm cada uma atributos chave estrangeira que identificam a relação que corresponde à entidade de origem do atributo multivalorado. Os restantes atributos correspondem aos valores dos atributos multivalorados.

Procedeu-se então ao mapeamento dos relacionamentos. Conforme a cardinalidade de cada relacionamento, pode ser gerado uma nova relação com duas chaves estrangeiras, ou pode ser adicionado um novo atributo à tabela da entidade com cardinalidade múltipla, a chave primária, para a relação um-para-muitos.

É de salientar que para além dos relacionamentos binários, temos também um relacionamento complexo, isto é que está definido entre três entidades ou mais entidades, no entanto, como todos os sub-relacionamentos são do tipo um-para-muitos, apenas se procedeu a acrescentar as devidas chaves estrangeiras à entidade em questão.

Agora com todas as tabelas preenchidas com os respetivos atributos, as diversas chaves candidatas foram determinadas. Este processo começou por analisar os atributos individualmente, identificando aqueles que eram capazes de distinguir univocamente um registo, passando assim a ser considerados chaves candidatas. Como nenhuma entidade apresentou mais do que duas chaves candidatas, o processo de decisão da chave primária foi relativamente simples e é explicado abaixo.

No final, obteve-se o conjunto completo de chaves candidatas. Destas, foram geralmente escolhidas como chaves primárias aquelas cujos valores eram controlados diretamente pela empresa Bela Rent-A-Car Lda.

Ficou assim pronta uma versão inicial do modelo lógico gerado manualmente pela equipa de desenvolvimento. Decidiu-se comparar esta versão gerada manualmente com uma versão gerada automaticamente pelo *brModelo*. Foram encontradas várias falhas e a equipa optou por não utilizar essa versão.

Segue-se assim a versão final do modelo lógico:

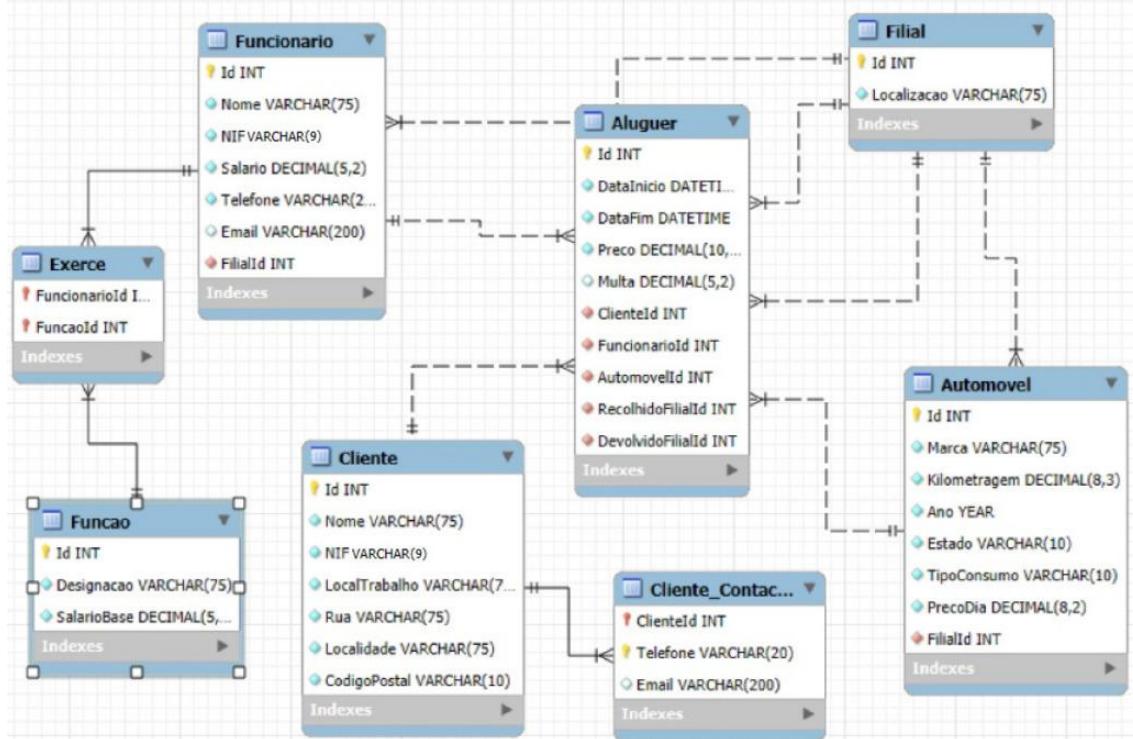


Figura 15 - Modelo lógico realizado pela equipa de desenvolvimento

Será agora descrito o processo de criação de cada relação.

Começando pela relação “Funcionario” que teve origem na entidade com o mesmo nome. Os seus atributos simples foram mapeados diretamente para atributos da relação: “Id”, “Nome”, “Salario” e “NIF”. O atributo composto “Contactos” deu origem aos atributos “Email” e “Telemovel”, dos quais “Email” pode assumir valores nulos pois tem uma natureza opcional que está específica no modelo conceitual. Foram selecionadas para chaves candidatas “Id” e “NIF” uma vez que ambas possuem a capacidade de identificar um funcionário, no entanto, acabou por ser escolhida para chave primária o “Id” pois o controlo do seu valor é feito pela empresa Bela Rent-A-Car Lda.

Atributo	Domínio	Nulo	Chave Estrangeira
<b>Id</b>	INT	N	N
<b>Nome</b>	VARCHAR(75)	N	N
<b>NIF</b>	VARCHAR(9)	N	N
<b>Salario</b>	DECIMAL(5,2)	N	N
<b>Telemovel</b>	VARCHAR(75)	N	N
<b>Email</b>	VARCHAR(75)	S	N
<b>Filial_Id</b>	INT	N	S

Tabela 9 – Dicionário de dados da entidade "Funcionario"

A tabela “Funcao” derivada da entidade do mesmo nome. Os seus dois atributos simples original dois atributos na tabela “Id” e “Designacao”, nenhum dos dois pode assumir valores nulos. Para chave primária foi escolhido o “Id” pois devido à simplicidade do domínio considerou-se que conduziria a um melhor desempenho do SBD.

Atributo	Domínio	Nulo	Chave Estrangeira
<b>Id</b>	INT	N	N
<b>Designacao</b>	VARCHAR(75)	N	N
<b>SalarioBase</b>	DECIMAL(8,2)	N	N

Tabela 10 - Dicionário de dados da entidade "Funcao"

Para a representação do relacionamento “Exerce”, devido à sua cardinalidade de muitos-para-muitos é necessário a criação de uma tabela com pelo menos dois atributos, uma chave estrangeira para “Funcionario” e outra para “Funcao”. Não existem quaisquer outros atributos para a relação uma vez que o relacionamento não apresenta atributos. A chave primária é consequentemente composta pelos únicos atributos da tabela uma vez que remover qualquer um deles levaria a que um funcionário exercesse várias funções, uma única função e vice-versa. Uma vez que ambos os atributos são chave primária, nenhum dos dois pode assumir valores nulos.

Atributo	Domínio	Nulo	Chave Estrangeira
<b>Funcionario_Id</b>	INT	N	S
<b>Funcao_Id</b>	INT	N	S

Tabela 11 - Dicionário de dados da entidade "Exerce"

A tabela “Cliente” originou a entidade com o mesmo nome. Os seus atributos simples “Id”, “Nome”, “Morada”, “NIF”, “LocaldeTrabalho”, deram origem a atributos na relação. O atributo composto “Morada” deu origem aos atributos simples “Rua”, “Localidade” e “codigopostal”. O único relacionamento que envolve a entidade “Cliente” é “Aluguer”, mas, uma vez que a cardinalidade múltipla se encontra do lado de “Aluguer”, este não dá origem a qualquer atributo na relação. Os atributos “Id” e “NIF” foram selecionados como chaves candidatas, sendo o primeiro escolhido para ser a chave primária uma vez que a empresa Bela Rent-A-Car Lda. Controla o seu valor.

Atributo	Domínio	Nulo	Chave Estrangeira
<b>Id</b>	INT	N	N
<b>Nome</b>	VARCHAR(75)	N	N
<b>NIF</b>	VARCHAR(9)	N	N
<b>LocalTrabalho</b>	VARCHAR(75)	N	N
<b>Rua</b>	VARCHAR(75)	N	N
<b>Localidade</b>	VARCHAR(75)	N	N

CodigoPostal	VARCHAR(75)	N	N
--------------	-------------	---	---

Tabela 12 - Dicionário de dados da entidade "Cliente"

A tabela “Cliente\_Contactos” foi derivada do atributo multivalorado com o mesmo nome da entidade “Cliente”. Cada registo nesta tabela deve conter um número de telefone e uma referência ao cliente ao qual este pertence.

Atributo	Domínio	Nulo	Chave Estrangeira
Cliente_Id	INT	N	S
Telefone	VARCHAR(20)	N	N
Email	VARCHAR(200)	S	N

Tabela 13 - Dicionário de dados da tabela "Cliente\_Contactos"

A entidade “Automovel” tem origem na entidade com o mesmo nome. Os seus atributos simples foram mapeados para a relação. Há um atributo que resulta do relacionamento de cardinalidade um-para-muitos com “Filial”. Este atributo tem o mesmo domínio que as chaves primárias e, é de notar que devida a participação total no relacionamento, “Filial”.

Atributo	Domínio	Nulo	Chave Estrangeira
Id	INT	N	N
Marca	VARCHAR(75)	N	N
Kilometragem	INT	N	N
Ano	INT(4)	N	N
Estado	VARCHAR(10)	N	N
TipoConsumo	VARCHAR(10)	N	N
PrecoDia	DECIMAL(5,2)	N	N
Filial_Id	INT	N	S

Tabela 14 - Dicionário de dados da tabela "Automovel"

A entidade “Filial” dá origem a uma tabela com o mesmo nome e os mesmos atributos simples da entidade: “Id” e “Localizacao”, nenhum dos quais pode ser nula. Não há relacionamentos que adicionem atributos à tabela. Ambos os seus dois atributos são chaves candidatas, mas escolheu-se “Id” uma vez que este tem um domínio mais simples.

Atributo	Domínio	Nulo	Chave Estrangeira
Id	INT	N	N
Localizacao	VARCHAR(75)	N	N

Tabela 15 - Dicionário de dados da tabela "Filial"

A tabela “Aluguer” tem origem na entidade do mesmo nome. Os seus atributos simples, “Id”, “Preco”, “DataInicio”, “DataFim”, “FilialDevolicao”, “FilialRecolha”, “multa” dos quais este ultimo pode ter valor

nulo. Há ainda dois atributos resultantes de relacionamentos de cardinalidade um-para-muito, resultantes de “Cliente” e “Funcionario”, a que demos os mesmos nomes. Estes atributos têm a natureza de chaves estrangeiras e como tem dominio INT para se referir a ambos. É de salientar que ambos os atributos têm uma participação total pelo que nenhum pode ter o valor nulo. Devido à sua unicidade, a única chave candidata e por consequência a chave primária é o “Id”.

Atributo	Domínio	Nulo	Chave Estrangeira
<b>Id</b>	INT	N	N
<b>DataInicio</b>	DATETIME	N	N
<b>DataFim</b>	DATETIME	N	N
<b>Preco</b>	DECIMAL(6,2)	N	N
<b>Multa</b>	DECIMAL(5,2)	S	N
<b>Funcionario_Id</b>	INT	N	S
<b>Cliente_Id</b>	INT	N	S
<b>Automovel_Id</b>	INT	N	S
<b>FilialRecolha_Id</b>	INT	N	S
<b>FilialDevolucao_Id</b>	INT	N	S

Tabela 16 - Dicionário de dados da tabela "Aluguer"

## 4.3 Normalização dos Dados

A normalização é uma técnica formal utilizada para analisar relações com base na sua chave primária (ou chaves candidatas) e nas dependências funcionais (Codd, 1972b). Esta técnica envolve um conjunto de regras que podem ser aplicadas para testar individualmente as relações, permitindo normalizar uma base de dados até ao grau desejado. (Connolly & Begg, 2015, p. 464)

Podemos começar então por concluir que todas as relações estão na 1FN uma vez que qualquer atributo escolhido é um atributo atómico e não existem atributos com múltiplos valores. Isto foi retificado no processo de escolha dos domínios.

Assim, todas as relações com chaves primárias simples encontram-se automaticamente na 2FN uma vez que como acabamos de concluir já se encontravam na 1FN e não existem chaves primárias constituídas por um grupo de atributos singular que admitam subconjuntos próprios.

Queremos então garantir que as restantes tabelas estão normalizadas. Começamos então por duas tabelas bastante parecidas: “Exerce”.

Na tabela “Exerce” não existe qualquer dependência, pois nem “Funcionario” é conhecido através de “Funcao”, nem “Funcao” é conhecida através de “Funcionario”. A tabela resulta de um relacionamento de muitos-para-muitos pelo que um funcionario pode ter várias funções.

Podemos assim concluir que esta tabela está na forma 3FN pois uma vez que não existem dependências funcionais, também não há dependências funcionais transitivas.

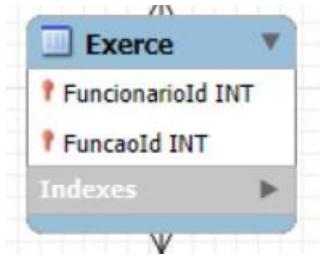


Figura 16 - Relação "Exerce".

As restantes tabelas com apenas dois ou três atributos estão automaticamente em 3FN pois a existência de dependências transitivas é impossível. Para as tabelas “Funcao”, “Cliente\_Contactos” e “Filial”, temos as respetivas dependências únicas  $Id \rightarrow Designacao$ ,  $SalarioBase$ ,  $Email$ ,  $Telefone \rightarrow Cliente$ ,  $Id \rightarrow Localizacao$ .

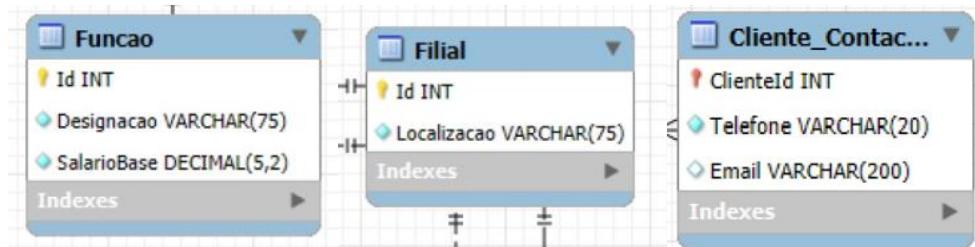


Figura 17 - Relação "Funcao", "Filial" e "Cliente\_Contacto".

Nas tabelas “Cliente”, “Funcionario” e “Automovel” o atributo “Id” está associado aos valores únicos dos restantes atributos da respetiva relação, isto é  $Id \rightarrow Nome$ , NIF, LocalTrabalho, Rua, Localidade, CodigoPostal,  $Id \rightarrow Nome$ , NIF, Salário, Email, Telefone e  $Id \rightarrow Marca$ , Kilometragem, Ano, Estado, TipoConsumo, PrecoDia, Filial, respetivamente. Assim verificamos que não existem dependências transitivas uma vez que todos os atributos que não “Id” dependem diretamente de “Id”.

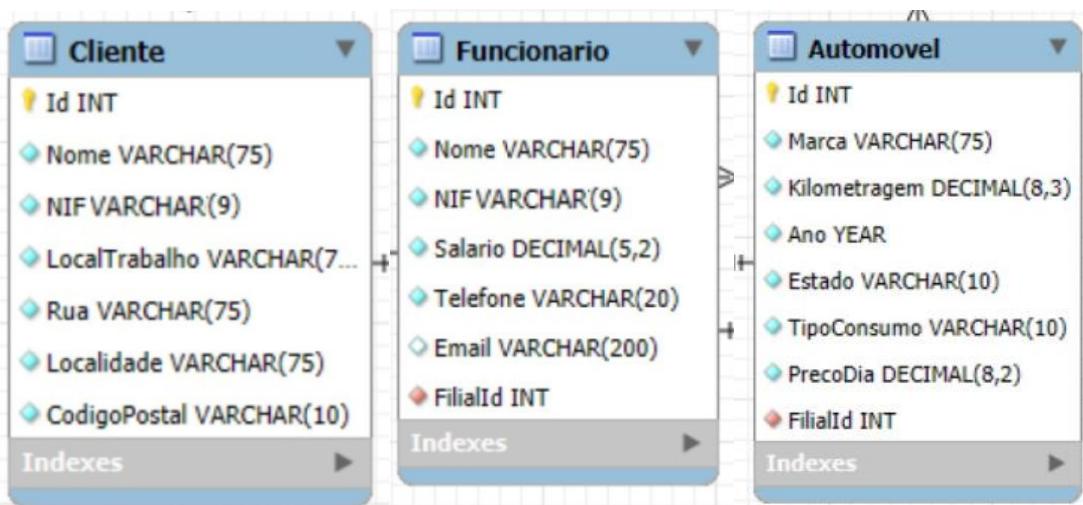


Figura 18 - Relações "Cliente", "Funcionario" e "Automovel".

Por fim, uma relação com maior grau é “Aluguer” onde “Id” está associado a um único valor de qualquer outro atributo:  $\text{Id} \rightarrow \text{Preco, DataInicio, DataFim, FilialDevolucao, FilialRecolha, Multa, Funcionario, Cliente}$ .

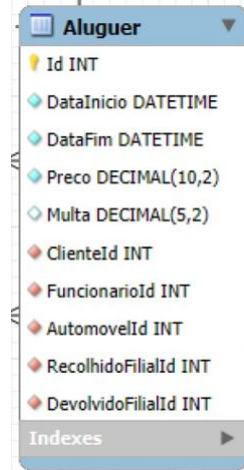


Figura 19 - Relação "Aluguer".

## 4.4 Validação do Modelo com Interrogações do Utilizador

A equipa de desenvolvimento validou o seu modelo através da implementação das interrogações pedidas pelos requisitos de manipulação. Para evitar erros no raciocínio e testar estas interrogações, foi utilizada a calculadora de álgebra relacional *RelaX*.

Antes de proceder a estes testes, foi necessário preparar a calculadora para tal. Na calculadora *RelaX*, as chaves estrangeiras e primárias não são especificadas, são consequências das interrogações introduzidas. Para implementar uma relação é apenas necessário enumerar todos os atributos e usar a sintaxe adequada. Uma vez que a calculadora não suportava todos os tipos de dados usados no modelo lógico: numéricos, textuais e datas. Foi feita uma escolha pela equipa de desenvolvimento de quais seriam os tipos adequados tendo em conta as interrogações e como estes seriam usados. Segue-se então a tabela de conversão dos tipos do *MySQL* para *RelaX*.

MySQL	RelaX
INT	number
DECIMAL(X,Y)	number
VARCHAR	string
DATETIME	date

Para povoar a BD de testes, foram adicionados manualmente regtos de dados aleatórios. Segue-se o exemplo da relação “Automovel”, o ficheiro completo com todas as tabelas pode ser encontrado em anexo.

```

Automovel = {
    Id:number, Marca:string, kilometragem:number, ano:number, Estado:string,
    PrecoDia:number, TipoConsumo:string, Filial:number
    1, 'Toyota Corolla', 45000, 2020, 'Disponível', 45, 'Gasolina', 1
    2, 'Volkswagen Golf', 30000, 2021, 'Ocupado', 50, 'Diesel', 2
    3, 'Peugeot 208', 15000, 2022, 'Disponível', 40, 'Gasolina', 3
    4, 'Renault Clio', 60000, 2019, 'Manutenção', 38, 'Gasolina', 1
    5, 'Ford Focus', 50000, 2020, 'Disponível', 42, 'Diesel', 2
    6, 'BMW Série 1', 25000, 2021, 'Ocupado', 65, 'Gasolina', 3
    7, 'Fiat 500', 12000, 2023, 'Disponível', 35, 'Elétrico', 1
    8, 'Audi A3', 40000, 2020, 'Disponível', 60, 'Gasolina', 2
    9, 'Mercedes A-Class', 28000, 2022, 'Disponível', 70, 'Gasolina', 3
    10, 'Nissan Leaf', 18000, 2021, 'Ocupado', 55, 'Elétrico', 1
    11, 'Honda Civic', 22000, 2022, 'Disponível', 48, 'Gasolina', 1
    12, 'Hyundai Ioniq 5', 8000, 2023, 'Disponível', 75, 'Elétrico', 2
    13, 'Opel Corsa', 35000, 2021, 'Manutenção', 37, 'Gasolina', 3
    14, 'Tesla Model 3', 15000, 2022, 'Ocupado', 85, 'Elétrico', 2
    15, 'Kia Ceed', 27000, 2020, 'Disponível', 43, 'Diesel', 1
}

```

Figura 20 - Exemplo da relação "Automovel".

De seguida procedeu-se à implementação das operações de manipulação de dados que correspondem aos requisitos que se encontram na figura abaixo:

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RM8	14	Listar todos os automoveis que já foram alugados e os respetivos clientes que os alugaram (Id, Nome, Id do automovel, marca, ano, estado, PrecoDia, TipoConsumo)	Automóveis/Gestão	Carolina Martins
RM9	15	Listar todo o tipo de função que existe associado a pelo menos um funcionario (id, nome, designcao, salariobase)	Gestão	Diogo Ribeiro
RM10	14	Listar todos os automoveis que nunca foram alugados.	Gestão	Lucas Robertson
RM11	14	Listas todos os clientes que não têm carros alugados no momento.	Gestão	Filipa Gonçalves
RM12	14	Listar todos os alugueres que estão a acontecer atualmente(id, automovel, cliente, preco, datainicio, datafim, filialinicio filalfim)	Gestão	Carolina Martins
RM13	34	Gerar uma lista do numero de alugueres das diferentes marcas de automóveis já alugados	Gestão	Filipa Gonçalves
RM14	35	Gerar uma lista ordenada por ordem decrescente de rendimento todas as filiais. (Id, Localização)	Gestão	Lucas Robertson
RM15	36	Listar todos os clientes assim como as filiais onde estes fizeram alugueres (Id do cliente, nome do cliente, id da filial e localização da filial)	Gestão/Cientes	Diogo Ribeiro
RM16	37	Verificar a Filial com um maior número de automóveis.	Gestão	Carolina Martins
RM17	38	Listar todos os alugueres associados a um cliente com base no seu identificador	Gestão	Filipa Gonçalves

Figura 21 - Requisitos de manipulação que deram origem às interrogações

A equipa começou então por implementar as interrogações pela ordem em que estas apareceram na tabela. Começou então pela RM8 onde se pretende listar todos os clientes juntamente com os carros que estes já alugaram. Para responder a esta interrogação utilizou-se a operação de junção natural. Na primeira junção, a cada aluguer foi associado o seu cliente respetivo e na segunda junção o aluguer foi associado ao respetivo automóvel para obter os dados do carro alugado. Por fim, apenas é aplicada uma projeção para extrair os dados relevantes.

```

$$\pi \text{ Cliente.Id, Cliente.Nome, Automovel.Id, Marca, ano, Estado, PrecoDia, } \\
\text{TipoConsumo} ( \\
(\text{Aluguer} \bowtie \text{Aluguer.cliente} = \text{Cliente.Id Cliente}) \\
\bowtie \text{Aluguer.automovel} = \text{Automovel.Id Automovel} \\
)$$

```

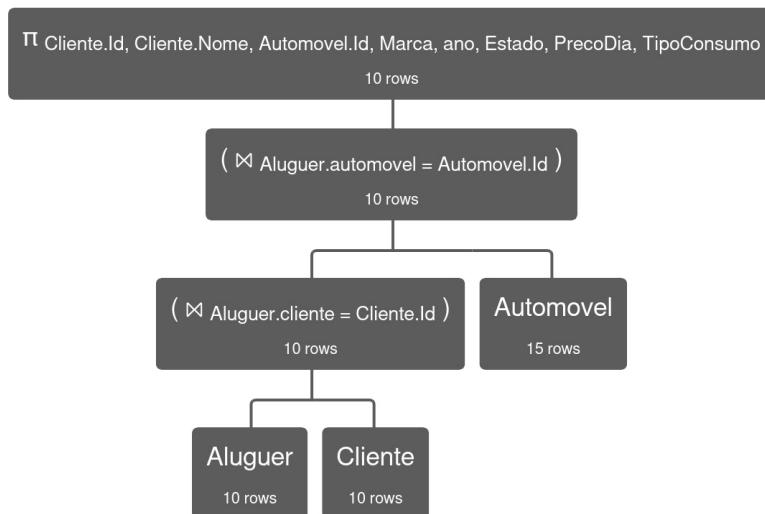


Figura 22 - Árvore de interrogação do RM8 aplicada aos dados de teste

Continuou-se assim para RM9 onde é pretendido mostrar todo o tipo de função que existe associado a pelo menos um funcionário. Para isso começamos por juntar Funcao e FuncaoFuncionario mas colocamos a condição de que Id = Funcao de forma a obter apenas as funções que estão associadas a algum funcionário. De seguida projetamos as informações consideradas pertinentes.

```

$$\pi \text{ Id, Designacao, SalarioBase } (\text{Funcao} \bowtie \text{Id=Funcao FuncaoFuncionario})$$

```

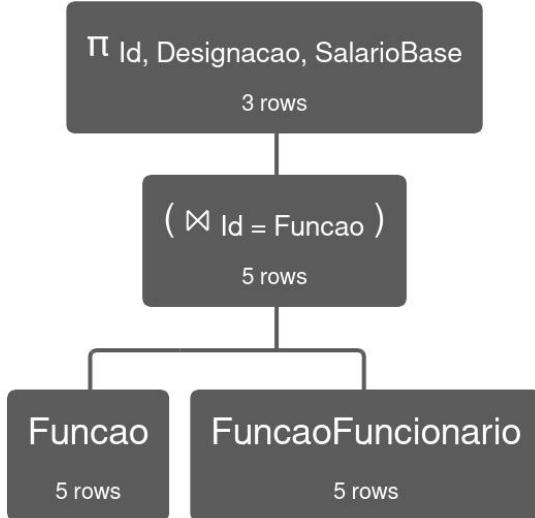


Figura 23 - Árvore de interrogação do RM9 aplicada aos dados de teste

Seguimos então para RM10 onde se pretende listar todos os automóveis que nunca foram alugados. Para isso fazemos a diferença entre o conjunto total dos automóveis e o conjunto dos automóveis que aparecem no registo de alugueres

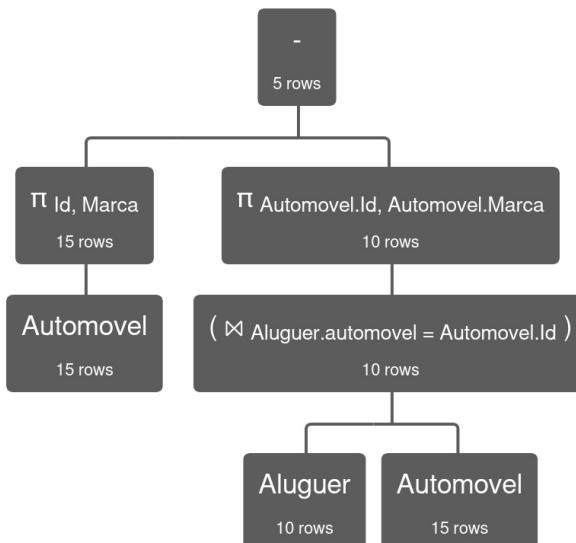
$$\begin{aligned} &\pi \text{Id, Marca} (\text{Automovel}) - \pi \text{Automovel.Id, Automovel.Marca} ( \\ &\quad \text{Aluguer} \bowtie \text{Aluguer.automovel} = \text{Automovel.Id} \text{ Automovel} \\ &\quad ) \end{aligned}$$


Figura 24 - Árvore de interrogação do RM10 aplicada aos dados de teste

Seguimos para RM11 onde se pretende listar todos os clientes que não têm carros alugados atualmente. Para isso filtrou-se os alugueres pelas datas, isto é, verificou-se quais eram os alugueres que começavam

antes do dia em que esta interrogação foi realizada uma vez que a calculadora relacional relax não suporta *keywords* como TODAY. De seguida fez se a junção entre os alugueres ativos e os clientes correspondentes. Por fim, subtraiu-se da lista de todos os alugueres aqueles que se encontravam ativos e projetou-se os dados considerados relevantes.

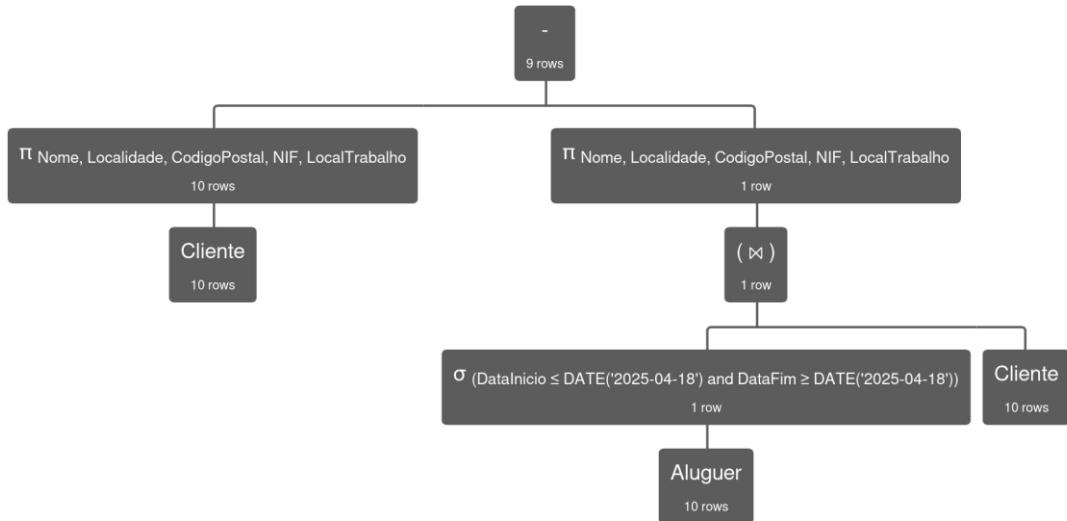
$$\begin{aligned} & \pi \text{ Nome, Localidade, CódigoPostal, NIF, LocalTrabalho } ( \\ & \quad \text{Cliente} \\ & ) - \pi \text{ Nome, Localidade, CódigoPostal, NIF, LocalTrabalho } ( \\ & \quad (\sigma \text{ (DataInicio} \leq \text{DATE('2025-04-18')} \wedge \text{DataFim} \geq \text{DATE('2025-04-18')}) \\ & \quad (\text{Aluguer})) \bowtie \text{Cliente} ) \end{aligned}$$


Figura 25 - Árvore de interrogação do RM11 aplicada aos dados de teste

Em contraste com o anterior em RM12 pretendemos listar todos os alugueres que estão a acontecer atualmente. Começamos por unir aluguer com cliente e de seguida realizar outra junção com Automovel. Aplicamos então uma seleção sobre o resultado da junção de forma a reter apenas os alugueres que estão ativos no momento. Por fim projetou-se os dados considerados importantes.

$$\begin{aligned} & \pi \text{ Nome, Marca, Preco, DataInicio, DataFim, filialRecolha, FilialDevolucao } ( \\ & \quad (\sigma \text{ (DataInicio} \leq \text{DATE('2025-04-18')} \wedge \text{DataFim} \geq \text{DATE('2025-04-18')}) ( \\ & \quad (\text{Aluguer} \bowtie \text{Cliente}) \bowtie \text{Automovel} \\ & \quad )) \end{aligned}$$

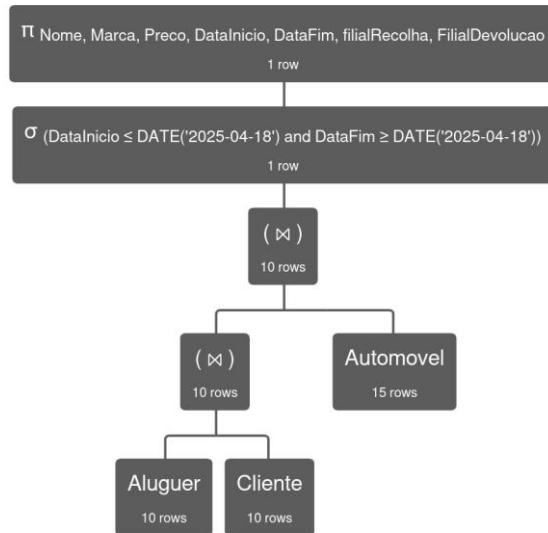


Figura 26 - Árvore de interrogação do RM12 aplicada aos dados de teste

Continuamos para RM13 onde se pretende obter uma lista do número de alugueres das diferentes marcas de automóveis que já foram alugadas. Para isso foi feita a junção entre aluguer e automóvel de forma a obter as informações deste último. De seguida, aplicou-se uma agregação sobre o atributo Marca usando a função de agravação count(\*) para obter o número total de alugueres por marca.

```

    γ Marca; count(*) → total (
        Aluguer ⊗ Aluguer.automovel = Automovel.Id Automovel
    )
    
```

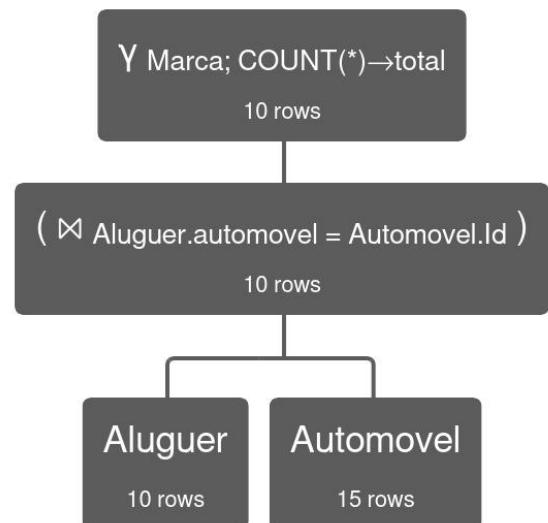


Figura 27 - Árvore de interrogação do RM13 aplicada aos dados de teste

Em RM14 pretendemos obter o rendimento total por Filial e ordenar os dados por ordem decrescente do rendimento (Receita). Começamos então por realizar uma junção entre aluguer e filial com base na

filialRecolha para obter a Filial em que o automóvel foi recolhido pois seria aí que o pagamento seria feito. Aplicamos então uma agregação sobre os sobre os atributos Filial.Id e Filial.Localizacao, com a função de agregação sum(Aluguer.Preco), de forma a calcular o total de receita gerada por cada filial. Por fim, os resultados foram ordenados por ordem decrescente.

```
 $\tau \text{ Receita DESC}(\gamma \text{ Filial.Id, Filial.Localizacao; sum(Aluguer.Preco)} \rightarrow \text{Receita})$ 
 $\quad \text{Aluguer} \bowtie \text{Aluguer.filialRecolha} = \text{Filial.Id Filial}$ 
 $\quad \quad \quad ))$ 
```

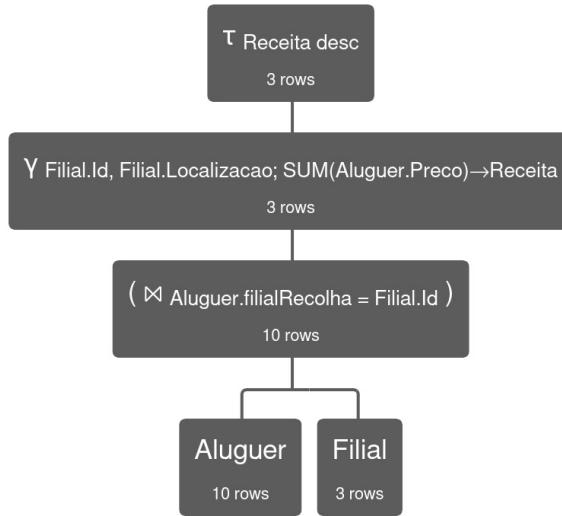


Figura 28 - Árvore de interrogação do RM14 aplicada aos dados de teste

Seguiu-se para RM15 onde se pretende obter uma lista dos clientes juntamente com as filiais onde estes efetuaram alugueres. Para tal foi efetuado uma junção entre as relações Aluguer e Cliente que permitiu associar a cada aluguer o respetivo Cliente. Fez-se também a junção entre aluguer e filial para associar a cada aluguer a Filial onde o automóvel foi recolhido. Por fim foram projetados os dados considerados relevantes.

```
 $\pi \text{ Cliente.Id, Cliente.Nome, Filial.Id, Filial.Localizacao} ($ 
 $\quad (\text{Aluguer} \bowtie \text{Aluguer.cliente} = \text{Cliente.Id Cliente})$ 
 $\quad \bowtie \text{Aluguer.filialRecolha} = \text{Filial.Id Filial}$ 
 $\quad \quad \quad ))$ 
```

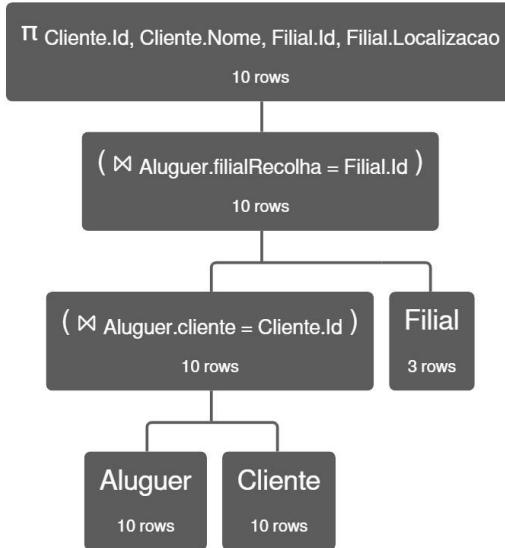


Figura 29 - Árvore de interrogação do RM15 aplicada aos dados de teste

Em RM16 pretende-se verificar qual é a filial com o maior número de automóveis. Com esse objetivo em vista foi feita uma agregação sobre a relação Aluguer, os dados foram agrupados por Filial. E para cada Filial é contado o número total de automóveis e esse valor é guardado no atributo total. O resultado da agregação é depois unido à relação Filial e por fim, projeta-se os atributos necessários.

```

    π Filial.Id, Filial.Localizacao, total (
      γ Filial; count(*) → total (Automovel)
        ⚡ Filial.Id = Filial Filial
    )
  
```

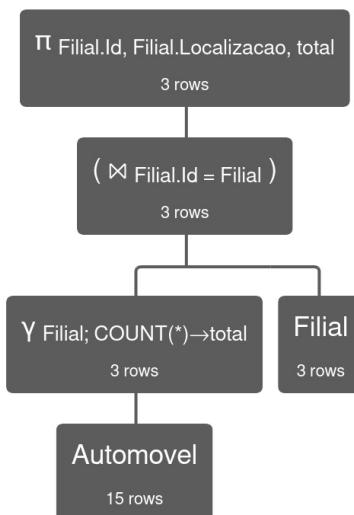


Figura 30 - Árvore de interrogação do RM16 aplicada aos dados de teste

Por último, em RM17 pretende-se listar todos os alugueres de um cliente com base no seu identificador. Para tal, começou por se aplicar uma seleção sobre a relação Aluguer e no caso foram apenas selecionados

alugueres em que o Id do Cliente fosse igual a 1. De seguida foram projetados todos os dados considerados necessários.

```
π Id, Preco, DataInicio, DataFim, filialRecolha, FilialDevolucao, multa,  
funcionario, automovel (  
    σ cliente = 1 (Aluguer)  
)
```

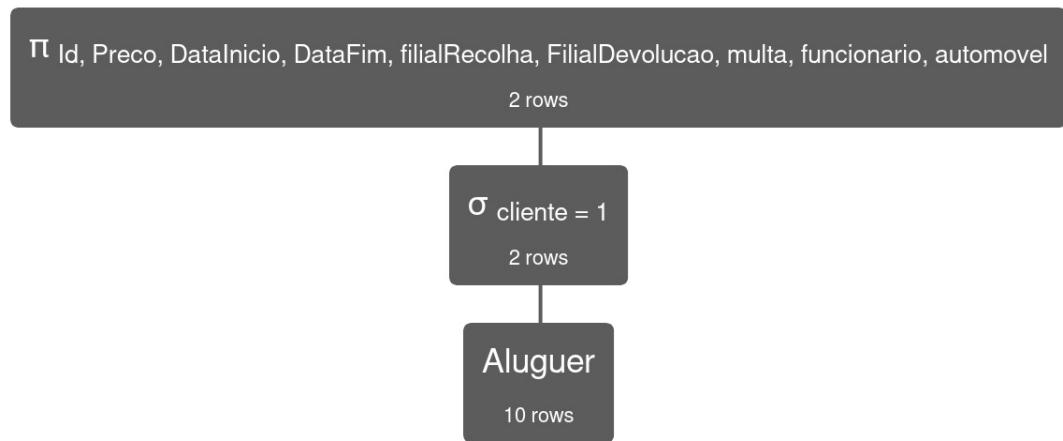


Figura 31 - Árvore de interrogação do RM17 aplicada aos dados de teste

## 5. Implementação Física

### 5.1 Apresentação e Explicação da Base de Dados Implementada

O modelo físico da BD é definido através de um script SQL (Structured Query Language) criado pela equipa de desenvolvimento. O mesmo é lido pelo SGBD MySQL que cria a Base de Dados para a empresa Bela Rent-a-Car assim como todas as tabelas que esta comporta.

O primeiro passo para a escrita do script foi feito através das instruções seguintes:

```
CREATE DATABASE IF NOT EXISTS BelaRentacar
    CHARACTER SET utf8mb4
    COLLATE utf8mb4_unicode_ci;

USE BelaRentacar;
```

Neste excerto podemos observar duas instruções separadas por ponto-e-vírgula. A primeira CREATE DATABASE IF NOT EXISTS BelaRentacar, cria a base de dados de nome “Belarentacar”. Através do uso de IF NOT EXISTS evitamos o erro de criar duas BDs com o mesmo nome. Na linha seguinte, a equipa optou por usar UTF-8 para armazenar os campos textuais uma vez que este permite que sejam guardados caracteres típicos portugueses e caracteres de línguas estrangeiras. Adicionou-se também COLLATE utf8mb4\_unicode\_ci; para as comparações não serem sensíveis a maiúsculas/minúsculas e acentos.

A segunda, USE BelaRentacar permite informar o MySQL que as próximas instruções serão aplicadas sobre a BD “BelaRentacar”.

Depois de criarmos a BD, definimos as suas tabelas, começando pela tabela Filial:

```
CREATE TABLE IF NOT EXISTS Filial (
    Id INT NOT NULL AUTO_INCREMENT,
    Localizacao VARCHAR(75) UNIQUE NOT NULL,
    PRIMARY KEY(Id)
) ENGINE = InnoDB;
```

A instrução CREATE TABLE IF NOT EXISTS Filial, cria a tabela Filial, se esta já não existir. Dentro dos parênteses definimos os atributos desta tabela (Id e Localizacao) e a sua chave primária (Id). O atributo Id é do tipo INT, não poderá ser nulo e será autoincrementado. O atributo

Localizacao é do tipo VARCHAR(75), isto é, pode conter até 75 caracteres, ele tem de ser único (UNIQUE), pois não há nenhuma Filial com a mesma localização, e não pode ser nulo (NOT NULL). Por fim, definimos o Id como chave primária desta entidade.

A segunda tabela a ser criada foi a Funcionario, através do seguinte comando:

```
CREATE TABLE IF NOT EXISTS Funcionario (
    Id INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(75) NOT NULL,
    NIF VARCHAR(9) NOT NULL UNIQUE CHECK (LENGTH(NIF) = 9),
    Salario DECIMAL(8,2) NOT NULL CHECK (Salario >= 0),
    Telefone VARCHAR(20) NOT NULL,
    Email VARCHAR(200) NULL,
    FilialId INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (FilialId) REFERENCES Filial(Id)
) ENGINE = InnoDB;
```

Como já observado anteriormente através de CREATE TABLE IF NOT EXISTS Funcionario, criamos a tabela Funcionario se esta não existir para evitar criar duas tabelas com o mesmo nome que podem causar inconsistências na nossa BD. Definimos os seguintes atributos: Id (Id INT NOT NULL AUTO\_INCREMENT) que semelhante ao anterior será auto incrementado de cada vez que uma nova instância de Funcionário é criada e também não pode ser nulo. O Nome (Nome VARCHAR(75) NOT NULL) pode ter até 75 caracteres e também não pode ser nulo. Para criar o NIF utilizamos NIF VARCHAR(9) NOT NULL e uma vez que todos os NIFs possuem sempre 9 caracteres. Para fortificar ainda mais este conceito adicionamos o UNIQUE CHECK (LENGTH(NIF) = 9) que verifica se o tamanho de NIF é 9 e apenas aceita casos em que isso aconteça e em que o conjunto de caracteres seja único. Para o Salario foi utilizado Salario DECIMAL(8,2) NOT NULL CHECK (Salario >= 0), pois este é um numero que decimal com 8 algarismos dos quais 2 representam a parte não inteira do mesmo, este também não pode ser nulo e tem de ser um número não negativo. Por fim temos Telefone VARCHAR(20) NOT NULL e Email VARCHAR(200) NULL, estes representam os contactos dos funcionários e este último não é obrigatório podendo então ser nulo.

De seguida, geramos a tabela Funcao:

```
CREATE TABLE IF NOT EXISTS Funcao (
```

```
    Id INT NOT NULL AUTO_INCREMENT,
    Designacao VARCHAR(75) UNIQUE NOT NULL,
```

```

SalarioBase DECIMAL(8,2) NOT NULL CHECK (SalarioBase >= 0),
PRIMARY KEY(Id)
) ENGINE = InnoDB;

```

Definimos os atributos Id, Designação e SalarioBase e identificamos o atributo Id como chave primária desta tabela. Definimos o atributo Id de forma semelhante aos anteriores utilizando NOT NULL AUTO\_INCREMENT. Para a Designação foi escolhida VARCHAR(75) UNIQUE NOT NULL, uma vez que esta não pode ser nula e a designação de cada função deve ser única. Já para SalarioBase definimos o mesmo como DECIMAL(8,2), sendo um número de 8 algoritmos que tem 2 reservados para a parte decimal, e verificamos se este é maior ou igual a zero (CHECK (SalarioBase >= 0)).

Depois de criarmos as tabelas Funcionario e Funcao, elaboramos a tabela Exerce (tabela que resulta do relacionamento N:M entre Funcionario e Função) :

```

CREATE TABLE IF NOT EXISTS Exerce (
    FuncionarioId INT NOT NULL,
    FuncaoId INT NOT NULL,
    PRIMARY KEY (FuncionarioId, FuncaoId),
    FOREIGN KEY (FuncionarioId) REFERENCES Funcionario(Id),
    FOREIGN KEY (FuncaoId) REFERENCES Funcao(Id)
) ENGINE = InnoDB;

```

Esta tabela contém os atributos FuncionarioId e FuncaoId, que compõem a chave primária (PRIMARY KEY (FuncionarioId, FuncaoId)), e são chaves estrangeiras pois FuncionarioId faz referência à tabela Funcionario através do Id (FOREIGN KEY (FuncionarioId) REFERENCES Funcionario(Id)) e FuncaoId, por sua vez, faz referência à tabela Funcao através do seu Id (FOREIGN KEY (FuncaoId) REFERENCES Funcao(Id)).

De seguida, criamos a tabela Cliente:

```

CREATE TABLE IF NOT EXISTS Cliente (
    Id INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(75) NOT NULL,
    NIF VARCHAR(9) NOT NULL UNIQUE CHECK (LENGTH(NIF) = 9),
    LocalTrabalho VARCHAR(75) NOT NULL,
    Rua VARCHAR(75) NOT NULL,
    Localidade VARCHAR(75) NOT NULL,

```

```

       CodigoPostal VARCHAR(10) NOT NULL,
        PRIMARY KEY(Id)
    ) ENGINE = InnoDB;

```

Esta tabela tem como chave primária o atributo Id, e contém os seguintes atributos: Id, Nome, NIF (é verificada a sua unicidade e se o seu tamanho é igual a 9 através de UNIQUE CHECK (LENGTH(NIF) = 9), tal como em Funcionario), LocalTrabalho, Rua, Localidade, CodigoPostal.

Associada à tabela Cliente, foi criada a tabela Cliente\_Contacto:

```

CREATE TABLE IF NOT EXISTS Cliente_Contacto (
    ClienteId INT NOT NULL,
    Telefone VARCHAR(20) NOT NULL,
    Email VARCHAR(200) NULL,
    PRIMARY KEY (ClienteId, Telefone),
    FOREIGN KEY (ClienteId) REFERENCES Cliente(Id)
) ENGINE = InnoDB;

```

A tabela em questão utiliza ClientId e Telefone como chave primária composta, sendo o atributo ClientId uma chave estrangeira que estabelece uma relação com a tabela Cliente através do seu Id.

Posteriormente, definimos a tabela Automovel:

```

CREATE TABLE IF NOT EXISTS Automovel (
    Id INT NOT NULL AUTO_INCREMENT,
    Marca VARCHAR(75) NOT NULL,
    Kilometragem DECIMAL(8,3) NOT NULL,
    Ano YEAR NOT NULL,
    Estado VARCHAR(10) NOT NULL CHECK (Estado IN ("Disponível", "Ocupado")),
    TipoConsumo VARCHAR(10) NOT NULL CHECK (TipoConsumo IN ("Gasolina", "Gasóleo",
    "Elétrico", "Híbrido")),
    PrecoDia DECIMAL(8,2) NOT NULL,
    FilialId INT NOT NULL,
    PRIMARY KEY (Id),
    FOREIGN KEY (FilialId) REFERENCES Filial(Id)
) ENGINE = InnoDB;

```

Esta tabela tem como chave primária o Id e contém uma chave estrangeira FilialId que faz referência à tabela Filial através de Id. Para evitar erros, garantimos que o Estado do automóvel

pode ser apenas “Disponível” ou “Ocupado” (CHECK (Estado IN (“Disponível”, “Ocupado”))), e que o seu tipo de consumo pode ser apenas “Gasolina”, “Gasóleo”, “Elétrico” ou “Híbrido” (CHECK ( TipoConsumo IN (“Gasolina”, “Gasóleo”, “Elétrico”, “Híbrido”))).

Por último, criamos a tabela Aluguer:

```
CREATE TABLE IF NOT EXISTS Aluguer (
    Id INT NOT NULL AUTO_INCREMENT,
    DataInicio DATETIME NOT NULL,
    DataFim DATETIME NOT NULL,
    CHECK (DataInicio < DataFim),
    Preco DECIMAL(10,2) NOT NULL,
    Multa DECIMAL(5,2) NULL,
    ClienteId INT NOT NULL,
    FuncionarioId INT NOT NULL,
    AutomovelId INT NOT NULL,
    RecolhidoFilialId INT NOT NULL,
    DevolvidoFilialId INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (ClienteId) REFERENCES Cliente(Id),
    FOREIGN KEY (FuncionarioId) REFERENCES Funcionario(Id),
    FOREIGN KEY (AutomovelId) REFERENCES Automovel(Id),
    FOREIGN KEY (RecolhidoFilialId) REFERENCES Filial(Id),
    FOREIGN KEY (DevolvidoFilialId) REFERENCES Filial(Id)
) ENGINE = InnoDB;
```

A presente tabela contém o atributo Id como chave primária. Esta também inclui cinco chaves estrangeiras: Clientel que remete à tabela Cliente (através do seu Id); Funcionariold que faz referência à tabela Funcionario via Id; Automovelld que estabelece uma relação à tabela Automóvel através do seu Id; RecolhidoFilialld e DevolvidoFilialld que faz referência à tabela Filial através do seu Id. Para evitar entradas inválidas, verificamos se DataInicio é antes de DataFim (CHECK (DataInicio < DataFim)).

No final, inserem-se as três filiais para garantir que elas estão disponíveis desde a criação da base de dados.

```
INSERT INTO Filial (Localizacao) VALUES
('Portugal'),
('Bélgica'),
('Mónaco');
```

Como também criamos as procedures, functions, triggers, Views e Users todos:  
Source ProceduresFunctionTrigger.sql;  
Source views.sql;  
Source Utilizadores.sql;

## 5.2 Criação de utilizadores da base de dados

Na gestão da base de dados da Bela Rent-A-Car, a criação de utilizadores é feita com uma clara organização e controlo de privilégios, refletindo os diferentes papéis dentro da empresa. Para tal, definem-se dois tipos principais de *roles* (funções): a *role* gestorFilial, destinada aos gestores das filiais, e a *role* Funcionario, destinada aos restantes funcionários.

```
DROP ROLE IF EXISTS 'gestorFilial';
CREATE ROLE 'gestorFilial';
DROP ROLE IF EXISTS 'Funcionario';
CREATE ROLE 'Funcionario';
```

Para a definição de privilégios, é necessário ter em mente não só os requisitos de controlo, mas também os de manipulação. A cada requisito estará associado um conjunto de instruções SQL que atribui as permissões estritamente necessárias para a execução das operações que define.

Dividimos os requisitos de controlo em dois com base nas duas funções principais que foram criadas. O utilizador gestorFilial possui permissões específicas que lhe permitem gerir os recursos humanos e materiais da empresa, bem como apoiar a gestão de clientes e alugueres.

Na tabela Funcao, o gestorFilial pode apenas inserir novos registo, permitindo adicionar novas funções à estrutura da empresa.

Na tabela Funcionario, tem permissões de atualização e eliminação, podendo editar ou remover dados de funcionários existentes.

Na tabela Automovel, possui total controlo com permissões de inserção, atualização e eliminação, podendo assim gerir integralmente a frota da filial.

Na tabela Cliente, tem permissões para eliminar e atualizar, o que permite corrigir ou remover registo de clientes conforme necessário.

Na tabela Cliente\_Contacto, pode inserir e eliminar, o que permite gerir os contactos associados a cada cliente.

Por fim, na tabela Aluguer, pode atualizar exclusivamente o campo DataFim, permitindo ao gestor registar o encerramento de um aluguer sem acesso a outros atributos sensíveis.

Para além disso também foram criados procedimentos que ajudassem a manter a integridade de base de dados intacta e que ajudassem a respeitar os requisitos RD9, RD14, RD16, RC2 e RC3

Assim, temos o procedimento armazenado AddFuncionarioComFuncao é responsável por inserir de forma automatizada um novo funcionário no sistema, garantindo que ele seja registado tanto na tabela de funcionários quanto na tabela de relação entre funcionário e função.

O procedimento novoCliente que ao adicionar um cliente, adiciona também o aluguer que este efetuará pois apenas podem existir clientes que já realizaram alugueres.

```

GRANT EXECUTE ON PROCEDURE BelaRentaCar.AddFuncionarioComFuncao TO
    'gestorFilial';
GRANT EXECUTE ON PROCEDURE BelaRentaCar.novoCliente TO 'gestorFilial';

```

Os utilizadores Octávio, Leonidas e Alberto, que representam o diretor e os gestores das filiais de Portugal, Bélgica e Mónaco, respetivamente, são criados com credenciais seguras e atribuídos tanto à role gestorFilial como à role Funcionario, permitindo-lhes executar tarefas administrativas e operacionais.

```

DROP USER IF EXISTS 'octavio'@'localhost';
CREATE USER 'octavio'@'localhost' IDENTIFIED BY 'Octavio#2025';
DROP USER IF EXISTS 'leonidas'@'localhost';
CREATE USER 'leonidas'@'localhost' IDENTIFIED BY 'Leonidas#2025';
DROP USER IF EXISTS 'alberto'@'localhost';
CREATE USER 'alberto'@'localhost' IDENTIFIED BY 'Alberto#2025';
GRANT 'gestorFilial' TO 'octavio'@'localhost';
GRANT 'gestorFilial' TO 'leonidas'@'localhost';
GRANT 'gestorFilial' TO 'alberto'@'localhost'

```

Além disso, todos os **gestores** recebem permissão de leitura (SELECT) sobre todas as tabelas da base de dados, garantindo acesso completo à informação:

```
GRANT SELECT ON BelaRentaCar.* TO 'gestorFilial';
```

Para além disso, apenas o Octávio, como Dono da Empresa pode ter propriedades sobre a entidade filial. Para além disso, como é necessário que uma filial tenha pelo menos um funcionário e um automóvel, achamos necessários criar o procedimento CriarFilialFuncionarioCarro.

```

GRANT DELETE, UPDATE ON BelaRentaCar.Filial to 'octavio'@'localhost';
GRANT EXECUTE ON PROCEDURE BelaRentaCar.CriarFilialFuncionarioCarro TO
    'octavio'@'localhost';

```

A role Funcionario, por sua vez, possui permissões mais restritas, permitindo apenas inserir novos alugueres através de uma procedure para poder haver handling de erros e atualizar a coluna Multa da tabela Aluguer. Esta role está alinhada com as funções dos funcionários José Martins e Eva Rocha, que também são criados com credenciais próprias e têm acesso limitado à base de dados.

-- RC4

```

GRANT EXECUTE ON PROCEDURE BelaRentaCar.novoAluguer TO 'Funcionario';
-- RC5

```

```

GRANT UPDATE (Multa) ON BelaRentaCar.Aluguer TO 'Funcionario';
DROP USER IF EXISTS 'josemartins'@'localhost';
CREATE USER 'josemartins'@'localhost' IDENTIFIED BY 'JoseMartins#2025';
DROP USER IF EXISTS 'evarocha'@'localhost';
CREATE USER 'evarocha'@'localhost' IDENTIFIED BY 'EvaRocha#2025';

```

Os funcionários têm acesso de leitura apenas às tabelas Aluguer, Automovel, Cliente e à view CarrosDisponiveis, de forma a limitar o acesso aos dados estritamente necessários às suas funções:

```

GRANT SELECT ON BelaRentaCar.Aluguer TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.Automovel TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.Cliente TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.CarrosDisponiveis TO 'Funcionario';

```

Adicionalmente, são definidas as roles padrão (SET DEFAULT ROLE) para cada utilizador, assegurando que, ao iniciar sessão, estes ativam automaticamente os privilégios correspondentes às suas funções.

```

SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'octavio'@'localhost';
SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'leonidas'@'localhost';
SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'alberto'@'localhost';
SET DEFAULT ROLE 'Funcionario' TO 'josemartins'@'localhost';
SET DEFAULT ROLE 'Funcionario' TO 'evarocha'@'localhost';

```

## 5.3 Povoamento da base de dados

O script de povoamento tem como objetivo inicializar a base de dados BelaRentaCar com um conjunto consistente e realista de dados para facilitar testes e validação do sistema.

Começa por inserir três filiais da empresa localizadas em Portugal, Bélgica e Mónaco. Em seguida, são adicionados dez clientes com os respetivos dados pessoais, como nome, morada, NIF e local de trabalho. Para complementar, é criada a tabela de contactos, que associa a cada cliente números de telefone e endereços de e-mail.

Posteriormente, são inseridos dez automóveis, com diversas características, tais como marca, ano, quilometragem, estado (disponível ou ocupado), tipo de consumo e a filial a que pertencem.

Depois, são adicionados cinco funcionários, com informações pessoais e salariais, associando cada um a uma filial específica.

Com os dados de clientes, funcionários e automóveis definidos, o script regista dez alugueres, indicando a duração, preço, multas possíveis, e associando cada aluguer ao cliente, funcionário, automóvel, assim como às filiais de recolha e devolução.

Por fim, são inseridas funções existentes na empresa — como “Gestor de Filial” ou “Assistente de Atendimento” — com os respetivos salários base, e faz-se a associação dinâmica entre os funcionários e as suas funções através da tabela de relacionamento Exerce.

Importa referir que este ficheiro de povoamento serve principalmente como ferramenta de teste e desenvolvimento. Por isso, utiliza variáveis dinâmicas (@ultimoFuncionarioid, @ultimoClienteid, @ultimoAutomovelld, @ultimoFuncaold) que capturam os maiores IDs atualmente presentes nas tabelas, garantindo que os dados inseridos não conflitam com os já existentes e que as relações entre as tabelas permanecem consistentes.

No caso comum de utilização — em que a base de dados começa vazia ou é povoada apenas uma vez estas variáveis não seriam necessárias, pois os IDs seriam atribuídos sequencialmente e sem riscos de conflito.

Desta forma, o script assegura um cenário realista e coerente para testes, respeitando a integridade referencial da base de dados, e podendo ser executado várias vezes sem corromper os dados existentes.

```
USE BelaRentaCar;
-- Inserir Filiais
INSERT INTO Filial (Localizacao) VALUES
('Portugal'),
('Bélgica'),
('Mónaco');

SET @ultimoFuncionarioId = (SELECT COALESCE(MAX(Id), 0) FROM Funcionario);
SET @ultimoClienteId = (SELECT COALESCE(MAX(Id), 0) FROM Cliente);
SET @ultimoAutomovelId = (SELECT COALESCE(MAX(Id), 0) FROM Automovel);
SET @ultimoFuncaoId = (SELECT COALESCE(MAX(Id), 0) FROM Funcao);

INSERT INTO Cliente (Nome, Rua, Localidade, CodigoPostal, NIF, LocalTrabalho)
VALUES
('Akio Toyoda', 'Rua Toyoda', 'Japao', '1234-234', 111111111, 'Toyota Group'),
('Oliver Blume', 'Rua Porsche', 'Alemanha', '5432-111', 222222222, 'Volkswagen
Group'),
('Carlos Tavares', 'Rua Europa', 'Portugal', '1500-000', 333333333,
'SteLLAntis'),
```

```

('Jim Farley', 'Ford Street', 'Estados Unidos', '9021-420', 444444444, 'Ford
Motor Company'),
('Luca de Meo', 'Via Milano', 'Itália', '2010-500', 555555555, 'Renault
Group'),
('Toshihiro Mibe', 'Rua Honda', 'Japao', '1245-678', 666666666, 'Honda Motor
Co.'),
('Makoto Uchida', 'Rua Yokohama', 'Japao', '1267-890', 777777777, 'Nissan Motor
Co.'),
('Mary Barra', 'Detroit Ave', 'Estados Unidos', '4822-300', 888888888, 'General
Motors'),
('Harald Krüger', 'München Straße', 'Alemanha', '8033-500', 999999999, 'BMW
Group'),
('Zhou Xiaoqing', 'Rua Geely', 'China', '1000-888', 101440101, 'Geely Holding
Group');

```

```

INSERT INTO Cliente_Contacto (ClienteId, Telefone, Email) VALUES
(1 + @ultimoClienteId, '910234567', 'akio.toyoda@toyota.jp'),
(2 + @ultimoClienteId, '920345678', 'oliver.blume@vwgroup.de'),
(3 + @ultimoClienteId, '930456789', 'carlos.tavares@stellantis.com'),
(4 + @ultimoClienteId, '940567890', 'jim.farley@ford.com'),
(5 + @ultimoClienteId, '950678901', 'luca.demeo@renault.it'),
(6 + @ultimoClienteId, '960789012', 'toshihiro.mibe@honda.jp'),
(7 + @ultimoClienteId, '970890123', 'makoto.uchida@nissan.jp'),
(8 + @ultimoClienteId, '980901234', 'mary.barra@gm.com'),
(9 + @ultimoClienteId, '990012345', 'harald.kruger@bmw.de'),
(10 + @ultimoClienteId, '900123456', 'zhou.xiaoqing@geely.cn');

```

```

INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado, PrecoDia, TipoConsumo,
FilialId) VALUES
('Toyota Corolla', 45000, 2020, 'Disponível', 45.00, 'Gasolina', 1),
('Volkswagen Golf', 30000, 2021, 'Ocupado', 50.00, 'Gasóleo', 2),
('Peugeot 208', 15000, 2022, 'Disponível', 40.00, 'Gasolina', 3),
('Renault Clio', 60000, 2019, 'Disponível', 38.00, 'Gasolina', 1),
('Ford Focus', 50000, 2020, 'Disponível', 42.00, 'Gasóleo', 2),
('BMW Série 1', 25000, 2021, 'Ocupado', 65.00, 'Gasolina', 3),
('Fiat 500', 12000, 2023, 'Disponível', 35.00, 'Elétrico', 1),
('Audi A3', 40000, 2020, 'Disponível', 60.00, 'Gasóleo', 2),
('Mercedes A-Class', 28000, 2022, 'Disponível', 70.00, 'Gasolina', 3),

```

```

('Nissan Leaf', 18000, 2021, 'Ocupado', 55.00, 'Elétrico', 1);

-- Inserir Funcionários
INSERT INTO Funcionario (Nome, NIF, Salario, Email, Telefone, FilialId) VALUES
('Cláudia Neves', '874512369', 3100.00, 'claudia.neves@belarentacar.com',
'963112233', 1),
('Rui Martins', '951753456', 2750.00, 'rui.martins@belarentacar.com',
'964223344', 2),
('Joana Ferreira', '782364159', 2900.00, 'joana.ferreira@belarentacar.com',
'965334455', 3),
('José Martins', '444556666', 2000.00, 'jose.martins@belarentacar.com',
'963456789', 1),
('Eva Rocha', '555667777', 2000.00, 'eva.rocha@belarentacar.com', '964567890',
2);

INSERT INTO Aluguer (DataInicio, DataFim, Preco, Multa, ClienteId,
FuncionarioId, AutomovelId, RecolhidoFilialId, DevolvidoFilialId) VALUES
('2025-04-01 00:00:00', '2025-04-05 00:00:00', 180.00, 0.00, 1 +
@ultimoClienteId, 2 + @ultimoFuncionarioId, 1 + @ultimoAutomovelId, 1, 1),
('2025-03-15 00:00:00', '2025-03-17 00:00:00', 100.00, 10.00, 2 +
@ultimoClienteId, 1 + @ultimoFuncionarioId, 2 + @ultimoAutomovelId, 2, 2),
('2025-04-10 00:00:00', '2025-04-13 00:00:00', 120.00, 0.00, 3 +
@ultimoClienteId, 3 + @ultimoFuncionarioId, 3 + @ultimoAutomovelId, 3, 3),
('2025-02-05 00:00:00', '2025-02-07 00:00:00', 76.00, 5.00, 4 +
@ultimoClienteId, 5 + @ultimoFuncionarioId, 4 + @ultimoAutomovelId, 1, 2),
('2025-01-20 00:00:00', '2025-01-24 00:00:00', 168.00, 0.00, 5 +
@ultimoClienteId, 4 + @ultimoFuncionarioId, 5 + @ultimoAutomovelId, 2, 2),
('2025-03-01 00:00:00', '2025-03-04 00:00:00', 195.00, 15.00, 1 +
@ultimoClienteId, 2 + @ultimoFuncionarioId, 6 + @ultimoAutomovelId, 3, 3),
('2025-04-05 00:00:00', '2025-04-07 00:00:00', 105.00, 0.00, 2 +
@ultimoClienteId, 1 + @ultimoFuncionarioId, 7 + @ultimoAutomovelId, 1, 1),
('2025-03-22 00:00:00', '2025-03-26 00:00:00', 240.00, 0.00, 3 +
@ultimoClienteId, 5 + @ultimoFuncionarioId, 8 + @ultimoAutomovelId, 2, 1),
('2025-02-18 00:00:00', '2025-02-22 00:00:00', 280.00, 20.00, 4 +
@ultimoClienteId, 3 + @ultimoFuncionarioId, 9 + @ultimoAutomovelId, 3, 3),
('2025-04-01 00:00:00', '2025-04-03 00:00:00', 110.00, 0.00, 5 +
@ultimoClienteId, 4 + @ultimoFuncionarioId, 1 + @ultimoAutomovelId, 1, 1);

```

```
-- Inserir Funções
INSERT INTO Funcao (Designacao, SalarioBase) VALUES
('Gestor de Filial', 2000),
('Assistente de Atendimento', 1100),
('Técnico de Manutenção', 1300),
('Operador de Reservas', 1200),
('Gestor Comercial', 1800);

-- Inserir associação Funcionario-Funcao (Exerce) com offset dinâmico
INSERT INTO Exerce (FuncionarioId, FuncaoId) VALUES
(@ultimoFuncionarioId + 1, @ultimoFuncaoId + 2),
(@ultimoFuncionarioId + 2, @ultimoFuncaoId + 2),
(@ultimoFuncionarioId + 3, @ultimoFuncaoId + 4),
(@ultimoFuncionarioId + 4, @ultimoFuncaoId + 1),
(@ultimoFuncionarioId + 5, @ultimoFuncaoId + 5);
```

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

Para a execução deste capítulo foi necessário entender a quantidade de memória que cada tipo de dados ocupa no MySQL.

A tabela apresentada a seguir resume o espaço em bytes ocupado por números decimais no MySQL, de acordo com a quantidade de dígitos que estes contêm. Esta informação é essencial para calcular o espaço total necessário para armazenar dados numéricos e estimar o crescimento anual da base de dados.

Dígitos	Bytes
1-2	1
3-4	2
5-6	3
7-9	4

Tabela 17 - Bytes ocupados por dígito

Além da tabela geral de bytes por número de dígitos, é importante entender como se calcula o espaço ocupado por valores decimais, considerando a parte inteira e a parte decimal. Na segunda tabela, apresentamos exemplos de definições de campos do tipo DECIMAL(p,s), onde p representa o número total de dígitos e s o número de dígitos decimais.

No caso de DECIMAL(5,2), por exemplo: a parte inteira ocupa 3 dígitos ( $5 - 2 = 3$ ), a parte decimal ocupa 2 dígitos. De acordo com a primeira tabela, a parte inteira de 3 dígitos ocupa 2 bytes e a parte decimal de 2 dígitos ocupa 1 byte. Assim, a ocupação total deste campo é 3.

Foi seguido o mesmo raciocínio para os restantes casos de DECIMAL na tabela.

DECIMAL(p,s)	Parte Inteira	Parte decimal	Ocupação total
DECIMAL(5, 2)	3	2	2 + 1 = 3 B
DECIMAL(10, 2)	8	2	4 + 1 = 5 B
DECIMAL(8, 3)	5	3	3 + 2 = 5 B
DECIMAL(8, 2)	6	2	3 + 1 = 4 B

Tabela 18 - Bytes ocupados pelo tipo DECIMAL(X,Y)

Seguindo para VARCHAR(x), de forma a estarmos preparados para todos as eventualidades consideramos o pior caso em todas as apariências de VARCHAR(x). Assim, como todos os VARCHAR(x) ocupam  $x + 1$  bytes (1 byte para armazenar o tamanho), temos:

VARCHAR(X)	Ocupação total
VARCHAR(9)	9 + 1 = 10 B
VARCHAR(10)	10 + 1 = 11 B
VARCHAR(20)	20 + 1 = 21 B
VARCHAR(75)	75 + 1 = 76 B
VARCHAR(200)	200 +1 = 201 B

Tabela 19 - Bytes ocupados pelo tipo VARCHAR(X)

Por fim, temos DATETIME que ocupa 8 bytes e Year que ocupa apenas 1 byte.

Iremos agora analisar tabela a tabela para calcular o tamanho da BD atualmente.

Começamos então pela tabela Filial:

Entidade: Filial		
Atributos	Dominio	Ocupação
Id	INT	4 B
Localizacao	VARCHAR(75)	76 B
<b>Total: 80 bytes</b>		

Tabela 20 - Bytes ocupados pela tabela Filial

Seguindo para a entidade Funcionario:

Entidade: Funcionario		
Atributos	Domínio	Ocupação
Id	INT	4 B
Nome	VARCHAR(75)	76 B
NIF	VARCHAR(9)	10 B
Salario	DECIMAL(5,2)	3 B
Telefone	VARCHAR(75)	76 B
Email	VARCHAR(75)	76 B

<b>Filial_Id</b>	INT	4 B
<b>Total: 249 B</b>		

Tabela 21 - Bytes ocupados pela tabela Funcionario

Seguimos para Funcao:

Entidade: Funcao		
Atributo	Domínio	Ocupação
<b>Id</b>	INT	4 B
<b>Designacao</b>	VARCHAR(75)	76 B
<b>SalarioBase</b>	DECIMAL(8,2)	4 B
<b>Total: 84 B</b>		

Tabela 22 - Bytes ocupados pela tabela Funcao

Relacionada com estas temos a tabela Exerce:

Entidade: Exerce		
Atributo	Domínio	Ocupação
<b>Funcionario_Id</b>	INT	4 B
<b>Funcao_Id</b>	INT	4 B
<b>Total: 8</b>		

Tabela 23 - Bytes ocupados pela tabela Exerce

Seguimos então para a entidade Cliente:

Entidade: Cliente		
Atributo	Domínio	Ocupação
<b>Id</b>	INT	4 B
<b>Nome</b>	VARCHAR(75)	76 B
<b>NIF</b>	VARCHAR(9)	10 B
<b>LocalTrabalho</b>	VARCHAR(75)	76 B
<b>Rua</b>	VARCHAR(75)	76 B
<b>Localidade</b>	VARCHAR(75)	76 B
<b>CodigoPostal</b>	VARCHAR(75)	76 B
<b>Total: 394 B</b>		

Tabela 24 - Bytes ocupados pela tabela Cliente

Ligado a esta temos a tabela Cliente\_contacto:

Entidade: Cliente_contacto		
Atributo	Domínio	Ocupação
<b>Cliente_Id</b>	INT	4 B
<b>Telefone</b>	VARCHAR(20)	21 B
<b>Email</b>	VARCHAR(200)	201 B

Total: 226 B
--------------

Tabela 25 - Bytes ocupados pela tabela Cliente\_Contacto

Temos ainda a tabela Automovel:

Entidade: Automovel		
Atributo	Domínio	Ocupação
<b>Id</b>	INT	4 B
<b>Marca</b>	VARCHAR(75)	76 B
<b>Kilometragem</b>	INT	4 B
<b>Ano</b>	YEAR	1 B
<b>Estado</b>	VARCHAR(10)	11 B
<b>TipoConsumo</b>	VARCHAR(10)	11 B
<b>PrecoDia</b>	DECIMAL(5,2)	3 B
<b>Filial_Id</b>	INT	4 B
<b>Total: 113</b>		

Tabela 26 - Bytes ocupados pela tabela Automovel

Por fim, temos a tabela Aluguer:

Entidade: Aluguer		
Atributo	Dominio	Ocupação
<b>Id</b>	INT	4 B
<b>DataInicio</b>	DATETIME	8 B
<b>DataFim</b>	DATETIME	8 B
<b>Preco</b>	DECIMAL(6,2)	3 B
<b>Multa</b>	DECIMAL(5,2)	3 B
<b>Funcionario_Id</b>	INT	4 B
<b>Cliente_Id</b>	INT	4 B
<b>Automovel_Id</b>	INT	4 B
<b>FilialRecolha_Id</b>	INT	4 B
<b>FilialDevolucao_Id</b>	INT	4 B
<b>Total: 46 B</b>		

Tabela 27 - Bytes ocupados pela tabela Aluguer

Tendo em conta os dados utilizados no povoamento da BD, tem-se a seguinte tabela abaixo. Note-se que este não é o tamanho da BD, mas o tamanho de uma BD com o mesmo número de registo em cada relação, registo estes das dimensões esperadas enunciadas anteriormente.

Entidades	Nº de ocorrências	Ocupação
<b>Filial</b>	3	3 x 80 = 240
<b>Funcionario</b>	5	5 x 249 = 1245
<b>Funcao</b>	5	5 x 84 = 420

<b>Exerce</b>	5	$5 \times 8 = 40$
<b>Cliente</b>	10	$10 \times 394 = 3940$
<b>Cliente_contacto</b>	10	$10 \times 220 = 2200$
<b>Automovel</b>	10	$10 \times 113 = 1130$
<b>Aluguer</b>	10	$10 \times 46 = 460$
<b>Total: 9675 bytes</b>		

Tabela 28 - Tamanho ocupado pela BD BelaRentacar

Com o tamanho da BD calculado, 9675 bytes, pode analisar-se como esta aumentará ao longo do tempo, considerando diferentes taxas de crescimento anuais possíveis, como se vê na tabela abaixo.

Taxa de crescimento anual	Ano 0 (Bytes)	Ano 1 (Bytes)	Ano 2 (Bytes)	Ano 3 (Bytes)	Ano 4 (Bytes)	Ano 5 (Bytes)
10%	9675	10642.5	11706.75	12877.43	14165.17	15581.68
20%	9675	11610	13932	16718.4	20062.08	24074.5
25%	9675	12093.75	15117.1	18896.48	23620.61	29525.75
50%	9675	14512.5	21768.75	32653.125	48979.69	73468.53
75%	9675	16931.25	29629.69	51852	90741	158796

Tabela 29 - Crescimento da BD BelaRentacar

## 5.5 Definição e caracterização de vistas de utilização em SQL

O uso de *views* em SQL está frequentemente ligado ao controlo das permissões de acesso dos utilizadores. Uma *view* é uma tabela temporária cujos dados provêm de tabelas reais ou até de outras *views*.

A vista CarrosDisponiveis foi criada para listar exclusivamente os automóveis que estão no estado "Disponível", permitindo aos responsáveis pelo aluguer identificar rapidamente quais são os carros com disponibilidade para serem alugados, sem necessidade de filtrar manualmente toda a tabela de automóveis. Usamos o CREATE VIEW para a criação da vista que designamos por CarrosDisponiveis. A operação SELECT FROM WHERE é responsável pela seleção e pela projeção dos dados dos automóveis que respeitam a condição a.Estado = "Disponível".

```
CREATE VIEW CarrosDisponiveis AS
    SELECT * FROM Automovel a
    WHERE a.Estado = "Disponível";
```

A vista OrigemUsers agrupa os clientes por localidade, apresentando a contagem de utilizadores em cada região. Esta vista é importante para analisar a distribuição geográfica dos clientes,

apoioando decisões de marketing e planeamento da expansão dos serviços. Assim como na anterior utilizamos o CREATE VIEW para criar a vista que chamamos OrigemUsers. Utilizamos a operação SELECT que seleciona a localidade e calcula o número de utilizadores por localidade usando COUNT(\*). O FROM Cliente c especifica a tabela Cliente, que contém a informação dos utilizadores. E o GROUP BY c.Localidade agrupa os dados pela localidade para contar o número de utilizadores em cada uma.

```
CREATE VIEW OrigemUsers AS
SELECT c.Localidade, count(*) AS UsersPorOrigem FROM Cliente c
GROUP BY c.Localidade;
```

A vista MediaPrecosMaisAlugadas calcula a média do preço diário dos automóveis alugados, agrupando esta informação por cada filial onde os carros foram recolhidos. Com esta vista, é possível avaliar o desempenho financeiro de cada filial, comparando os preços médios praticados, o que pode ser fundamental para definir estratégias comerciais. Com esta vista, é possível identificar o tipo ou gama de carros mais apreciado em cada filial, permitindo compreender as preferências dos clientes em cada localização e, assim, ajustar o parque automóvel e a oferta de veículos de forma mais adequada às necessidades locais. Utilizamos o CREATE VIEW para criar a vista com o nome MediaPrecosMaisAlugadas. O SELECT AVG(au.PrecoDia) AS MediaPrecoDiario calcula a média do preço diário de aluguer dos automóveis. f.Localizacao: seleciona a localização da filial onde o automóvel foi recolhido e FROM Aluguer a: especifica a tabela de alugueres. INNER JOIN Automovel au ON a.AutomovelId = au.Id: junta os dados com a tabela de automóveis, para obter o preço diário. E o INNER JOIN Filial f ON a.RecolhidoFilialId = f.Id: junta os dados com a tabela de filiais, para obter a localização da filial onde o automóvel foi recolhido. Por fim, o GROUP BY f.Localizacao: agrupa os dados por localização de filial, para calcular a média do preço diário por filial.

```
CREATE VIEW MediaPrecosMaisAlugadas AS
SELECT AVG(au.PrecoDia) as MediaPrecoDiario, f.Localizacao FROM Aluguer
a
INNER JOIN Automovel au ON a.AutomovelId = au.Id
INNER JOIN Filial f ON a.RecolhidoFilialId = f.Id
GROUP BY f.Localizacao;
```

## 5.6 Tradução das interrogações do utilizador para SQL

Na tradução das interrogações de álgebra relacional para SQL seguiu-se a ordem em que foram feitas anteriormente, tendo ela sido também a mesma em que apareceram nos requisitos de manipulação.

Começando com o requisito RM8 que pretende listar todos os automóveis que já foram alugados assim como os respetivos clientes que os alugaram. Em SQL as consultas são feitas usando a instrução SELECT. Esta pode ser seguida por um conjunto de atributos que se pretende consultar, estes têm de ser separados por vírgulas para poder ser feita uma projeção. A seguir utilizamos o FROM Cliente AS CL que especifica as relações (tabelas) sobre as quais a consulta é efetuada. Quando é necessário combinar dados de várias tabelas, são utilizadas junções, no caso INNER JOIN, que permitem integrar informação de diferentes relações através de condições de igualdade entre chaves primárias e estrangeiras. Segue-se a instrução SQL que implementa esta funcionalidade:

```
SELECT CL.Id AS Cliente, CL.Nome, AU.Id AS Automovel, AU.Marca, AU.Ano,
       AU.Estado, AU.PrecoDia, AU.TipoConsumo
     FROM Cliente AS CL
   INNER JOIN Aluguer AL
      ON CL.Id = AL.ClienteId
   INNER JOIN Automovel AS AU
      ON AL.AutomovelId = AU.Id
```

De seguida, RM10 que pretende listar o Id e a Marca de todos os automóveis que nunca foram alugados. No SQL abaixo podemos ver que semelhante à instrução anterior utilizamos o SELECT para especificar as colunas que queremos projetar. E como queremos projetar dados dos automóveis que não foram alugados optamos por usar o WHERE NOT EXISTS que verifica se existe uma correspondência entre determinada chave estrangeira e primária. Se não existir sabemos que esse automóvel não foi alugado.

```
SELECT AU.Id AS Automovel, AU.Marca
      FROM Automovel AS AU
    WHERE NOT EXISTS (
        SELECT *
      FROM Aluguer AS AL
     WHERE AU.Id = AL.AutomovelId);
```

O requisito RM11 pretende listar todos os clientes que não têm um carro alugado de momento. Semelhante a ambas instruções anteriores utilizamos as instruções SELECT e WHERE NOT EXISTS e para fazer a verificação de um aluguer atualmente, verificamos se entre a data de início do aluguer e a data de fim do aluguer estava a data atual.

```

SELECT CL.Id AS Cliente, CL.Nome
      FROM Cliente AS CL
      WHERE NOT EXISTS(
SELECT AL.DataInicio, AL.DataFim
      FROM Aluguer AS AL
      WHERE AL.ClienteId = CL.Id
AND now() BETWEEN AL.DataInicio AND AL.DataFim);

```

O requisito RM12 pretende listar todos os alugueres que estão a acontecer atualmente. Semelhante aos anteriores foram usadas as instruções SELECT e WHERE para conseguir verificar se um aluguer se encontrava a decorrer no presente momento.

```

SELECT AL.Id, AL.AutomovelId, AL.Preco, AL.DataInicio, AL.DataFim,
       AL.RecolhidoFilialId, AL.DevolvidoFilialId
      FROM Aluguer AS AL
     WHERE now() BETWEEN AL.DataInicio AND AL.DataFim;

```

No requisito RM13 pretendemos obter uma lista do número de alugueres das diferentes marcas de automóveis disponíveis já alugados. Para isso utilizamos a instrução SELECT para selecionar a Marca do automóvel, e count(AL.Id) para calcular o número total de alugueres registados para cada marca de automóvel sem contar a linha da própria marca. Utilizamos o LEFT OUTER JOIN para assegurar que ao fazer a junção externa da tabela Automovel e da tabela Aluguer, mesmo que uma determinada marca não tenha sido alugada, ela apareça mas o seu count é 0. Por fim, o GROUP BY agrupa os resultados permitindo assim calcular o número de alugueres de cada marca individualmente.

```

SELECT AU.Marcas, count(AL.Id)
      FROM Automovel AS AU
      LEFT OUTER JOIN Aluguer AS AL
      ON AU.Id = AL.AutomovelId
      GROUP BY AU.Marcas;

```

No requisito RM14 queremos gerar uma lista por ordem decrescente de rendimento de todas as filiais.

```

SELECT FI.Id AS Filial, FI.Localizacao, sum(AL.Preco) AS Receita
      FROM Filial AS FI
      LEFT OUTER JOIN Aluguer AS AL
      ON FI.Id = AL.RecolhidoFilialId
      GROUP BY FI.Id
      ORDER BY (Rendimento) DESC;

```

No requisito RM15 queremos listar todos os clientes assim como as filiais onde os mesmos realizaram alugueres. Utilizamos assim, a instrução SELECT DISTINCT para selecionar clientes distintos, isto é não repetir o mesmo cliente mesmo que este tenha mais do que um aluguer. De seguida foi usado o INNER JOIN para juntar as tabelas de Cliente, Aluguer e Filial de forma a obter os dados necessários para resolver esta query.

```
SELECT DISTINCT CL.Id, CL.Nome, FI.Id, FI.Localizacao
    FROM Cliente AS CL
    INNER JOIN Aluguer AS AL
        ON CL.Id = AL.ClienteId
    INNER JOIN Filial AS FI
        ON AL.RecolhidoFilialId = FI.Id
```

No requisito RM16 pretendemos verificar qual é a Filial com um maior número de automóveis. Utilizamos novamente a instrução SELECT para selecionar as informações necessárias e COUNT(\*) para fazer a contagem dos carros em cada filial. Utilizamos o INNER JOIN para juntar as informações das tabelas Filial e Automovel e por fim agrupamos e ordenamos por ordem decrescente de NrCarros.

```
SELECT FI.Id AS Filial, FI.Localizacao, count(*) AS NrCarros
    FROM Automovel AS AU
    INNER JOIN Filial AS FI
        ON AU.FilialId = FI.Id
    GROUP BY (FI.Id)
    ORDER BY (NrCarros) DESC
    LIMIT 1;
```

Por fim, no requisito RM17 pretendemos listar todos os alugueres associados a um cliente com base no seu identificador único. Para isso optamos por fazer um PROCEDURE uma vez que precisamos que a query fosse capaz de receber o Id do Cliente. Assim, utilizamos o CREATE PROCEDURE para criar o procedimento e depois a instrução SELECT para selecionar todos os alugueres que pertenciam a um determinado cliente (WHERE AL.ClienteId = IdCliente;)

```
DELIMITER $$

CREATE PROCEDURE alugueresDoCliente
    (IN IdCliente INT)
    BEGIN
        SELECT AL./*
        FROM Aluguer AS AL
        WHERE AL.ClienteId = IdCliente;
        END $$
```

```
DELIMITER ;
```

## 5.7 Indexação do Sistema de Dados

Quando corretamente utilizados, os índices podem trazer grandes benefícios ao desempenho de uma base de dados. Um índice é uma estrutura de dados que organiza os valores de um ou mais atributos de uma tabela, evitando a necessidade de percorrer a tabela inteira para encontrar registos com base nesses atributos. Ao indexar atributos, é possível obter um desempenho superior nas pesquisas.

No entanto, é importante considerar que a utilização de índices pode tornar as operações de inserção e atualização mais lentas, pois o índice também precisa ser atualizado sempre que os dados mudam. Além disso, os índices ocupam espaço de armazenamento adicional. Por isso, é fundamental avaliar cuidadosamente a sua utilização, especialmente em bases de dados de grande dimensão, onde o tamanho dos índices pode ser significativo.

Devido ao reduzido número de registos atualmente existentes na base de dados BelaRentacar, a equipa de desenvolvimento optou por não implementar índices no produto entregue ao cliente, uma vez que todas as consultas apresentaram tempos de execução muito rápidos (em frações de segundo). No entanto, o código para a criação de diversos índices foi preparado, embora não tenha sido executado nesta fase. Este código poderá ser utilizado futuramente, caso alguma consulta apresente um desempenho insatisfatório à medida que a base de dados for crescendo. Assim, para cada interrogação da secção anterior, analisou-se a expressão SQL desenvolvida e procuraram-se instâncias onde ocorriam identificações de registos numa tabela com base nos valores de algum dos seus atributos. Temos, por exemplo, a identificação de registos de “Aluguer” com base em “Aluguer.Automovel”.

Em RM14 a consulta utiliza uma junção entre as tabelas Automovel e Aluguer, com base na chave estrangeira AL.AutomovelId, e realiza um group by por AU.Marcas. Para otimizar essa consulta, recomendou-se a criação de índices específicos que aceleram as operações de junção e group by. Foi criado um índice composto sobre as colunas Id e Marca:

```
CREATE INDEX idx_automovel_id_marca ON Automovel(Id, Marca);
```

Este índice ajuda a acelerar o processo de junção (JOIN) e, ao mesmo tempo, facilita o agrupamento (GROUP BY AU.Marcas), melhorando a eficiência geral da consulta.

Com este índice, espera-se uma redução no tempo de execução da consulta, especialmente em bases de dados com grande volume de dados, garantindo melhor desempenho no processo de análise e geração de relatórios.

## 5.8 Implementação de procedimentos, funções e gatilhos

O MySQL suporta a definição e execução de procedimentos, funções e triggers (ou gatilhos), que permitem, no próprio SGBD, o controlo de fluxo de execução de diversas consultas, sem ser necessário recorrer a software aplicacional para gerir operações condicionais e cíclicas. Os procedimentos e as funções são geralmente ferramentas úteis para a administração de uma base de dados, enquanto os triggers, que são executados quando ocorrem alterações nos conteúdos de uma tabela, são utilizados para assegurar a consistência do estado da base de dados, por exemplo, através da atualização de atributos derivados.

Iremos então começar por explicar as funções que sentimos necessidades de criar. Começando pela função calculaPreco que tem como objetivo calcular o preço de um aluguer automaticamente.

```
DELIMITER //
-- Função que calcula o custo do aluguer de um automóvel
DROP FUNCTION IF EXISTS calculaPreco;
CREATE FUNCTION calculaPreco(
    precoDia DECIMAL(8,2),
    inicio DATETIME,
    fim DATETIME
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE precoTotal DECIMAL(10,2);
    DECLARE nrDias INT;
    SET nrDias = DATEDIFF(fim, inicio) + 1;
    SET precoTotal = precoDia * nrDias;
    RETURN precoTotal;
END;
//
DELIMITER ;
```

Esta função recebe como parâmetros o preço por dia (precoDia) e duas datas (inicio e fim). Primeiramente, calcula o número total de dias de aluguer através da função DATEDIFF, adicionando 1 para incluir o dia de início. Em seguida, multiplica o número de dias pelo preço diário para obter o preço total. Por fim, retorna esse valor como um número decimal com duas casas decimais. A função é declarada como DETERMINISTIC, o que indica que, para os mesmos valores de entrada, ela retorna sempre o mesmo resultado. Esta função promove a padronização

e a consistência no cálculo do custo de aluguer, evitando erros de cálculo e simplificando a gestão financeira da aplicação.

De seguida, temos a função podeCriarAluguer que verifica se um aluguer pode ser criado, isto é, verifica se um automóvel se encontra disponível, se o cliente existe, se o automóvel se encontra na filial que o cliente o quer alugar e se não há nenhum outro aluguer referente a esse automóvel cujas datas colidam algures com as dadas.

```
DELIMITER //

DROP FUNCTION IF EXISTS podeCriarAluguer;
CREATE FUNCTION podeCriarAluguer
(Inicio DATETIME, Fim DATETIME, AutomovelId INT, FuncionarioId INT, FilOrigem
INT)
RETURNS BOOLEAN
NOT DETERMINISTIC
READS SQL DATA
SQL SECURITY DEFINER
BEGIN
...
END;

// 

DELIMITER ;
```

Esta função recebe como parâmetros a data de início e de fim do aluguer, o identificador do automóvel (AutomovelId), o identificador do funcionário (FuncionarioId) e o identificador da filial de origem (FilOrigem).

No seu corpo, a função executa as seguintes operações:

1. Valida a filial do funcionário — Verifica se o funcionário pertence à filial de origem.
2. Valida a filial e a disponibilidade do automóvel — Verifica se o automóvel está disponível e na filial correta.
3. Verifica a sobreposição de alugueres — Certifica-se de que não existem alugueres já registados para o mesmo automóvel que se sobreponham às datas fornecidas.

Caso todas as condições sejam satisfeitas, a função devolve TRUE (verdadeiro), permitindo a criação do aluguer. Caso contrário, devolve FALSE (falso).

A função é definida como NOT DETERMINISTIC, uma vez que o seu resultado pode variar em função do estado atual da base de dados (ex.: disponibilidade de veículos ou existência de

alugueres). Além disso, está definida como READS SQL DATA, pois consulta dados de tabelas, e tem nível de segurança SQL SECURITY DEFINER, garantindo que seja executada com os privilégios do utilizador que a criou.

Desta forma, a função garante integridade e consistência nos processos de gestão de alugueres, evitando conflitos e assegurando que os alugueres só são criados se todas as condições forem cumpridas.

Criamos o procedimento AddFuncionarioComFuncao, que insere as informações básicas do funcionário — como nome, NIF, salário, telefone, email e o identificador da filial — na tabela Funcionario. Em seguida, utiliza a função LAST\_INSERT\_ID() para obter o identificador único do funcionário recém-criado. Com esse identificador, realiza a inserção na tabela Exerce, associando o funcionário à função específica que ele irá exercer.

Este procedimento inclui uma transação que garante que ambas as inserções ocorram de forma atômica. Caso ocorra algum erro durante o processo, a transação é revertida automaticamente, assegurando a integridade dos dados. Além disso, o procedimento retorna uma mensagem indicando sucesso ou erro, facilitando o diagnóstico e controlo do fluxo de execução.

```
DELIMITER //  
DROP PROCEDURE IF EXISTS AddFuncionarioComFuncao;  
CREATE PROCEDURE AddFuncionarioComFuncao ( IN pNome VARCHAR(75), IN pNIF  
VARCHAR(9), IN pSalario DECIMAL(8,2), IN pTelefone VARCHAR(20), IN pEmail  
VARCHAR(200), IN pFilialId INT, IN pFuncaoId INT )  
BEGIN  
DECLARE EXIT HANDLER FOR SQLEXCEPTION  
  
BEGIN  
SELECT 'Erro no AddFuncionarioComFuncao' AS Mensagem;  
ROLLBACK;  
END;  
START TRANSACTION;  
  
INSERT INTO Funcionario (Nome, NIF, Salario, Telefone, Email, FilialId)  
VALUES (pNome, pNIF, pSalario, pTelefone, pEmail, pFilialId);  
  
INSERT INTO Exerce (FuncionarioId, FuncaoId)  
VALUES (LAST_INSERT_ID(), pFuncaoId);  
  
COMMIT;  
SELECT 'Sucesso na criação de Funcionário com Função' AS Mensagem;
```

```
END; //  
DELIMITER ;
```

Os próximos procedimentos estarão aqui explicados, no entanto o seu código encontrar-se-á em anexo.

De seguida, criamos o procedimento CriarFilialFuncionarioCarro, que automatiza o processo de criação de uma nova filial, adicionando-lhe um funcionário e um automóvel de forma integrada, segura e consistente.

Este procedimento recebe como parâmetros os dados da nova filial (localização), as informações do funcionário (nome, NIF, salário, telefone, email e ID da função) e os detalhes do automóvel (marca, quilometragem, ano, estado, tipo de consumo e preço por dia).

A execução do procedimento segue estas etapas:

- Início da transação — Todas as operações são realizadas no contexto de uma única transação, garantindo que ou tudo é executado com sucesso, ou nada é alterado em caso de erro.
- Inserção da filial — A nova filial é registada na tabela Filial com a respetiva localização. O seu identificador gerado automaticamente é armazenado em FilialId.
- Inserção do funcionário — É chamada a procedure AddFuncionarioComFuncao para inserir o novo funcionário já associado à filial recém-criada, bem como à função correspondente.
- Inserção do automóvel — É registrado um automóvel na tabela Automovel, associando-o à mesma filial e armazenando os dados fornecidos.

Caso ocorra qualquer erro em alguma destas operações, a transação é automaticamente revertida (ROLLBACK) e é apresentada uma mensagem de erro. Se tudo for executado com sucesso, a transação é confirmada (COMMIT) e é devolvida uma mensagem de sucesso.

Este procedimento garante a integridade referencial dos dados e facilita o processo de expansão do negócio, ao criar automaticamente todos os elementos necessários para iniciar a operação de uma nova filial.

O procedimento novoAluguer é responsável pela criação de um novo aluguer de um automóvel, assegurando a consistência e a integridade dos dados ao verificar previamente as condições necessárias para o aluguer.

Este procedimento recebe como parâmetros: as datas de início e fim do aluguer, o identificador do cliente, o identificador do funcionário que está a gerir o aluguer, o identificador do automóvel, e as identificações das filiais de origem e destino.

As principais etapas deste procedimento são:

1. Determinação do preço — Através de uma consulta à tabela Automovel, obtém o preço diário de aluguer. Em seguida, utiliza a função calculaPreco para calcular o custo total com base no preço diário e na duração do aluguer.
2. Validação de disponibilidade — Verifica, através da função podeCriarAluguer, se o aluguer é possível, garantindo que não há sobreposição de datas, que o automóvel está disponível e que o funcionário pertence à filial correta.
3. Criação do aluguer — Caso seja possível, inicia uma transação (START TRANSACTION), insere o novo aluguer na tabela Aluguer e confirma a transação (COMMIT). Em seguida, apresenta uma mensagem com o ID do aluguer criado.
4. Falha na criação — Se o aluguer não puder ser criado, retorna uma mensagem informando o motivo, incluindo a filial do funcionário e a filial do automóvel para ajudar na análise da situação.

Este procedimento garante que os alugueres só são criados quando todas as condições estão satisfeitas, prevenindo conflitos de agendamento e mantendo a integridade operacional do sistema de gestão de alugueres.

O procedimento armazenado novoCliente é responsável por registar um novo cliente no sistema e, caso todas as condições sejam cumpridas, proceder também à criação de um aluguer associado a esse cliente.

Este procedimento recebe como parâmetros as informações do cliente (nome, NIF, local de trabalho, morada completa e telefone), bem como os detalhes necessários para o aluguer: datas de início e fim, identificador do funcionário responsável, identificador do automóvel, e as filiais de origem e destino.

O procedimento realiza as seguintes operações:

1. Validação das filiais — Verifica se o funcionário pertence à filial de origem e se o automóvel está disponível e associado à mesma filial. Esta verificação garante que o aluguer será criado apenas se todos os requisitos logísticos estiverem reunidos.
2. Transação de criação — Se as validações forem bem-sucedidas, inicia uma transação para garantir a consistência dos dados.
3. Criação do cliente — Insere os dados do cliente na tabela Cliente e armazena o ID gerado.
4. Criação do contacto — Insere o telefone do cliente na tabela Cliente\_Contacto, já que cada cliente deve ter pelo menos um número de telefone.
5. Criação do aluguer — Chama o procedimento novoAluguer para criar o aluguer associado ao cliente recém-criado, incluindo todas as validações internas que este procedimento realiza.

6. Confirmação — Caso tudo seja bem-sucedido, confirma a transação com COMMIT e devolve uma mensagem indicando o sucesso da operação.
7. Falha na criação — Caso as validações iniciais não sejam satisfeitas, devolve uma mensagem explicando que o cliente não foi criado, incluindo as filiais do funcionário, do automóvel e de origem do aluguer para facilitar a análise do problema.

Este procedimento assegura que a criação de clientes e alugueres ocorre de forma atómica e segura, evitando erros de inserção parcial de dados e garantindo a integridade das operações no sistema.

O procedimento armazenado `deleteCliente` é utilizado para eliminar de forma controlada todos os registo relacionados a um cliente específico no sistema.

Este procedimento recebe como parâmetro o identificador do cliente (Id). Executa as seguintes operações:

1. Eliminação de contactos — Remove todos os contactos associados ao cliente na tabela `Cliente_Contacto`. Esta etapa é necessária para garantir a integridade referencial antes de eliminar outros dados relacionados.
2. Eliminação de alugueres — Remove todos os alugueres associados ao cliente na tabela `Aluguer`. Isto garante que não ficam registos de aluguer pendentes ou inconsistentes ligados ao cliente eliminado.

Este procedimento contribui para a manutenção da base de dados, assegurando que não existem referências órfãs ou dados inconsistentes relacionados a clientes eliminados.

Criámos um *trigger* chamada `tgOcupaAutomovel`, que é responsável por atualizar automaticamente o estado de um automóvel e a sua afiliação à filial correta assim que é criado um novo aluguer.

Esta *trigger* é executada após a inserção de um novo registo na tabela `Aluguer` (ou seja, logo após a criação de um aluguer). Para cada novo aluguer, o *trigger* realiza o seguinte:

- Atualiza o estado do automóvel — Se o momento atual (`NOW()`) estiver dentro do intervalo do aluguer (entre a data de início e a data de fim do aluguer), o *trigger* altera o estado do automóvel para “Ocupado”, indicando que o automóvel não está disponível para outros alugueres neste período.
- Atualiza a filial do automóvel — Define a filial do automóvel para a filial de devolução indicada no aluguer (`DevolvidoFilialId`), garantindo que o automóvel é corretamente transferido para a filial onde será devolvido após o aluguer.

Desta forma, o *trigger* assegura a integridade e a consistência dos dados na gestão da frota, facilitando a atualização automática do estado e localização dos automóveis sem necessidade de intervenções manuais.

```
DELIMITER //
```

```
-- Trigger que garante que após um automóvel ser alugado, passa a estar Ocupado
e a pertencer à filial de entrega
DROP TRIGGER IF EXISTS tgOcupaAutomovel;
CREATE TRIGGER tgOcupaAutomovel
AFTER INSERT ON Aluguer
FOR EACH ROW
BEGIN

IF now() BETWEEN NEW.DataInicio AND NEW.DataFim
    THEN
UPDATE Automovel AS A
SET A.Estado = 'Ocupado'
, FilialId = NEW.DevolvidoFilialId
WHERE A.Id = NEW.AutomovelId;
END IF;
END;
//
```

No intuito de atualizar diariamente o Estado dos automóveis de modo a os disponibilizar após os alugueres expirarem criou-se um evento que o faz às 23:00 de cada dia, verificando sempre se há algum aluguer que termine hoje para caso tal seja confirmado para um dado automóvel alterarmos o Estado dele de 'Ocupado' para 'Disponível'. Para implementar este evento necessitámos de utilizar CREATE EVENT para o criar, ON SCHEDULE EVERY 1 DAY para definir que o evento deve ser repetido todos os dias, e STARTS CURRENT\_DATE + INTERVAL 23 HOURS para o evento começar hoje às 23:00 e que estamos a somar 23 horas à data, visto que CURRENT\_DATE por si só tem o dia, mês e ano de hoje, mas com uma hora inicial 00:00:00.

Quando o evento é executado, é realizada uma operação UPDATE sobre a tabela Automovel, utilizando uma junção interna (INNER JOIN) com a tabela Aluguer. Esta junção é feita com base na condição de que o Id do automóvel (na tabela Automovel) seja igual ao AutomovelId presente na tabela Aluguer. Desta forma, associam-se a cada automóvel os alugueres que lhe dizem respeito.

Após esta associação, o evento atualiza o campo Estado de cada automóvel para 'Disponível', desde que exista um aluguer cuja data de fim (DataFim) coincida com a data atual (CURDATE()).

Assim, garante-se que os automóveis cujo período de aluguer termina no próprio dia fiquem automaticamente marcados como disponíveis para futuros alugueres.

## 5.9 Backup da Base de Dados

Foi criada *shell script* para realizar automaticamente uma cópia de segurança da base de dados BelaRentaCar. Este guarda o ficheiro de backup com a data atual numa subpasta chamada *backups*, localizada na mesma diretória onde o script se encontra. Ao ser executado, o *script* garante que essa pasta existe, gera o nome do ficheiro com base na data, e faz o backup da base de dados utilizando o *mysqldump*. No final, informa se o processo foi concluído com sucesso ou se ocorreu algum erro.

Para automatizar a execução deste *script* todos os dias às 23h, pode-se usar o *crontab*, um agendador de tarefas do sistema.

Com o *cron* num servidor, seria possível agendar os *backups* diários de forma automática.

```
#!/bin/bash
USER="root"
PASSWORD="root"
DATABASE="BelaRentaCar"
BACKUP_DIR="$(dirname "$(realpath "$0")")/backups"
mkdir -p "$BACKUP_DIR"
DATA=$(date +"%Y-%m-%d")
BACKUP_FILE="${BACKUP_DIR}/backup_${DATABASE}_${DATA}.sql"
mysqldump -u "$USER" -p"$PASSWORD" "$DATABASE" > "$BACKUP_FILE"
if [ $? -eq 0 ]; then
echo "Backup feito com sucesso: $BACKUP_FILE"
else
echo "Erro ao fazer backup"
fi
```

```
GNU nano 6.2                                         /tmp/crontab.EjPQU5/crontab
# Edit this file to introduce tasks to be run by cron.
#
# Each task to run has to be defined through a single line
# indicating with different fields when the task will be run
# and what command to run for the task
#
# To define the time you can provide concrete values for
# minute (m), hour (h), day of month (dom), month (mon),
# and day of week (dow) or use '*' in these fields (for 'any').
#
# Notice that tasks will be started based on the cron's system
# daemon's notion of time and timezones.
#
# Output of the crontab jobs (including errors) is sent through
# email to the user the crontab file belongs to (unless redirected).
#
# For example, you can run a backup of all your user accounts
# at 5 a.m every week with:
# 0 5 * * 1 tar -zcf /var/backups/home.tgz /home/
#
# For more information see the manual pages of crontab(5) and cron(8)
#
# m h dom mon dow   command
0 23 * * * /home/diogo/Desktop/BD/Bases-de-Datos/backup.sh >> /home/diogo/Desktop/BD/Bases-de-Datos/backups/ultimoBackup.log 2>&1
```

Figura 32 - Crontab Manager

# 6. Implementação do sistema de migração dados

## 6.1 Descrição das fontes de dados utilizadas

Assim, como foi pedido, foi criado um sistema de migração que permite povoar a base de dados BelaRentacar com três tipos de dados: *CSV (Comma-Separated Values)*, *JSON (JavaScript Object Notation)* e *PostgreSQL (base de dados relacional)*.

No nosso entender, este sistema de migração refere-se à integração de três fontes de dados anteriormente utilizadas pelas diferentes filiais, cada uma com o seu próprio formato e sistema. Estas fontes tornaram-se agora obsoletas com a adoção da nova base de dados central em MySQL.

Este processo de migração foi concebido para ser executado apenas uma vez pela equipa técnica, com o objetivo de consolidar e centralizar toda a informação relevante na nova estrutura. Não se trata de uma funcionalidade contínua ou acessível aos funcionários comuns das filiais, que a partir de agora interagem exclusivamente com a base de dados central em MySQL.

Começamos por criar a fonte vinda da Filial da Bélgica (CSV) onde cada linha é uma entrada de uma certa entidade (o primeiro *token* da linha). Cada *token* é separado por ‘;’.

Decidimos não ter linhas de cabeçalho uma vez que estas não pareciam necessárias ou “convencionais” considerando que haveria vários tipos de dados num só ficheiro.

Tendo isso em consideração realizamos a divisão no próprio código *Python* usando o primeiro *token* da linha (Entidade).

Exemplo:

```
Cliente;Karel;Vermeulen;Stationsstraat;Bélgica;8500;090909090;Vermeulen BV  
Aluguer;2025-01-20      00:00:00;2025-01-24      00:00:00;170.00;0.00;5;4;5;2;2  
Funcao;Técnico de Manutenção;1350
```

De seguida criamos a fonte vinda da Filial de Portugal (JSON) aonde temos listas de cada entidade.

Exemplo:

```
"Cliente": [  
  { "Nome": "João Silva", "Rua": "Rua das Flores", "Localidade": "Lisboa",  
    "CodigoPostal": "1000-123", "NIF": "123456789", "LocalTrabalho": "EDP" },
```

```

{ "Nome": "Maria Fernandes", "Rua": "Avenida da Liberdade", "Localidade": "Lisboa", "CodigoPostal": "1050-456", "NIF": "987654321", "LocalTrabalho": "GALP" },
{ "Nome": "Pedro Santos", "Rua": "Rua do Almada", "Localidade": "Porto", "CodigoPostal": "4000-789", "NIF": "192837465", "LocalTrabalho": "Sonae" },
],
"Aluguer": [
{ "DataInicio": "2025-05-01 00:00:00", "DataFim": "2025-05-05 00:00:00", "Preco": 140.00, "Multas": 0.00, "FuncionarioId": 1, "ClienteId": 1, "AutomovelId": 1, "RecolhidoFilialId": 1, "DevolvidoFilialId": 1 },
{ "DataInicio": "2025-04-10 00:00:00", "DataFim": "2025-04-12 00:00:00", "Preco": 80.00, "Multas": 0.00, "FuncionarioId": 2, "ClienteId": 2, "AutomovelId": 2, "RecolhidoFilialId": 1, "DevolvidoFilialId": 2 },
]
,
```

Por último, implementámos a fonte proveniente do Mónaco (base de dados relacional) em PostgreSQL, que consiste numa base de dados exatamente igual à base de dados geral, com a exceção de não conter a tabela de filiais.

Criámos dois scripts semelhantes aos da base de dados principal: um para a criação da estrutura da base de dados e outro para o povoamento com dados.

Exemplo de povoação da BD do mónaco:

```

INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado, TipoConsumo, PrecoDia, FilialId) VALUES
('Ferrari Roma', 12000.500, 2022, 'Disponível', 'Gasolina', 850.00, 3),
('Bentley Continental', 9000.000, 2021, 'Disponível', 'Gasolina', 950.00, 3),
('Tesla Model S', 3000.250, 2023, 'Disponível', 'Elétrico', 600.00, 3),
('Porsche Taycan', 5000.000, 2022, 'Disponível', 'Elétrico', 700.00, 3);
```

## 6.2 Descrição do programa implementado

Para implementar o programa de migração de fontes de dados decidimos usar *Python* uma vez que era uma linguagem nova que ainda não tínhamos usado. Apesar disso, a equipa de desenvolvimento considerou que tinha adquirido bases suficientes em outras linguagens de programação para conseguir utilizar esta sem grandes problemas.

Para implementar a migração de dados propriamente dita, foram utilizadas as bibliotecas padrão da linguagem para manipulação de arquivos CSV e JSON, bem como os conectores específicos

para MySQL e PostgreSQL. Estas ferramentas permitiram extrair, transformar e carregar os dados de forma eficiente entre os diferentes sistemas de gestão de bases de dados.

```
import mysql.connector
from mysql.connector import IntegrityError
import psycopg2
from decimal import Decimal
import csv
import json
```

Implementámos uma função que lê o ficheiro CSV sequencialmente, identificando a entidade de cada registo (linha) através do primeiro token. Com base nessa informação, utilizamos o conector da base de dados para executar um `INSERT` com os dados restantes da linha. Como o CSV não suporta tipos de dados nativos, foi necessário realizar *casts manuais* para os tipos adequados definidos nas tabelas da base de dados.

Para garantir a robustez do processo, implementámos deteção e tratamento de exceções, nomeadamente para lidar com erros como entradas duplicadas causadas por restrições `UNIQUE`. Desta forma, mesmo que ocorram erros em algumas inserções, o programa continua a executar e a processar os restantes registos.

Adicionalmente, à semelhança do que foi feito com as outras fontes de dados, foi introduzida uma lógica de cálculo de offset com base no ID máximo existente por entidade na base de dados principal. Assim, evitam-se conflitos de integridade referencial, como duplicação de chaves primárias ou associações incoerentes entre entidades (ex.: um funcionário a apontar para uma filial inexistente ou de outro país).

A migração do ficheiro `JSON` foi realizada de forma distinta, já que os dados são lidos como listas completas por entidade em vez de sequencialmente. Como o ficheiro `JSON` já suporta tipos de dados nativamente, evitou-se a necessidade de realizar casts. Em semelhança à migração anterior, também aqui foi aplicada a lógica de offsets para IDs e chaves estrangeiras, bem como a deteção de exceções.

Por fim, na migração da base de dados do Mónaco, foi implementada uma função que realiza um `SELECT` de todas as entradas em cada tabela e insere os dados na base de dados geral. Não foi necessário ajustar os tipos de dados, já que o PostgreSQL é compatível com aqueles usados na base de dados de origem. Tal como nas outras fontes, foi aplicada a estratégia de offsets para evitar colisões de chaves e garantir a consistência das referências, com tratamento de exceções incluído.

## 7. Conclusões e Trabalho Futuro

A equipa de alunos, ao longo do último semestre, desenvolveu uma base de dados para a Empresa Bela Rent-a-Car, seguindo todo o ciclo de desenvolvimento de bases de dados, desde as fases iniciais de definição do sistema e levantamento de requisitos, passando pela modelação conceptual e lógica, até à implementação física da base de dados e sua exploração.

Neste processo, a equipa teve a oportunidade de aprofundar e consolidar conhecimentos em diversas áreas, como a gestão de projetos, a comunicação com o cliente, as diferentes formas de modelação de bases de dados, a interação com um SGBD e, sobretudo, a forma de documentar adequadamente cada fase do desenvolvimento.

Foi ainda possível explorar, de forma mais detalhada, temas como o funcionamento do motor de armazenamento de um SGBD e a forma como este impacta o espaço ocupado pela base de dados. Para tal, foi realizada uma experiência prática que permitiu à equipa adquirir um conhecimento mais profundo sobre esta temática.

Após a conclusão das duas fases, a equipa de desenvolvimento ponderou sobre os aspectos positivos e negativos de todo o processo de desenvolvimento. Pode então concluir que, na perspetiva dos mesmos, as tarefas propostas foram realizadas com correção apesar de reconhecerem que poderia haver diversas melhorias em diversos pontos nomeadamente em relação à documentação e em relação à implementação física.

A equipa de desenvolvimento considera que a sua inexperiência teve um papel na falta de documentação e na dificuldade da definição de requisitos concretos que pudessem servir como boa base para a criação do modelo conceptual e posteriormente do modelo lógico. Sendo a BD BelaRentacar a primeira que este grupo de alunos desenvolveu, várias etapas não eram conhecidas e algumas lacunas foram sentidas durante a fase de implementação de erros.

A equipa pode aprender com estes erros e já sabe com exatidão o que necessita de escrever e como necessita de escrever para obter algo conciso e bem fundamentado.

Apesar de não terem sido sentidas grandes dificuldades na aplicação dos conhecimentos adquiridos em aula pelos alunos, houve-as na gestão do número elevado de tarefas a realizar o que conduziu à fraca organização temporal.

Em primeiro lugar tendo em conta a primeira fase do trabalho foi concordado de forma unânime que a planificação de atividades original não foi totalmente respeitada. O levantamento de requisitos conduziu a um atraso na sua validação e nas fases de desenvolvimento subsequentes. Mesmo com esse atraso, o projeto esteve pronto atentamente pois a planificação original deixou alguma margem de atraso que permitia estes acontecimentos.

No que diz respeito à planificação da segunda fase do trabalho, não houveram atrasos significativos. Após a planificação da primeira fase a equipa de alunos decidiu ser mais realista em relação às suas possibilidades e capacidades, tendo em consideração semanas em que não teriam disponibilidade para trabalhos devido à elevada carga de avaliações.



Figura 33 - Miniatura do Gaunt final

Na implementação física percebeu-se que a consistência dos dados é um dos pilares fundamentais de qualquer sistema de gestão de bases de dados, mas a nossa equipa técnica compreendeu, ao longo do desenvolvimento, que garantir essa consistência é um desafio constante. Inserções incompletas, atualizações incorretas, falhas durante transações, entradas duplicadas, interações diretas e descontroladas com as tabelas são fatores que colocam em risco a integridade da base de dados.

Com base nessa experiência, identificámos que a forma mais eficaz de assegurar a integridade e a consistência da base de dados é através da disponibilização controlada de *stored procedures*. Estas *procedures* encapsulam toda a lógica de manipulação de dados, aplicam validações rigorosas, gerem transações com segurança e fornecem mensagens claras em caso de erro, o que nos permite manter a base de dados íntegra mesmo em cenários de falha parcial.

Ao restringirmos o acesso dos utilizadores apenas à execução destas *procedures* e não à base de dados diretamente garantimos que todas as operações seguem fluxos testados, validados e seguros. Mesmo que ocorra um erro por parte do utilizador ou por algum dado mal inserido, os mecanismos de deteção e tratamento de exceções presentes nas *procedures* impedem a propagação de inconsistências.

Para o futuro, temos como objetivo continuar a melhorar e otimizar as *procedures* existentes, bem como desenvolver novas, de forma a cobrir ainda mais operações do sistema. Com isso, pretendemos minimizar ao máximo o acesso direto à base de dados por parte dos utilizadores, promovendo um modelo totalmente controlado, seguro e confiável para a manipulação da informação.

Esta abordagem representa, para nós, o caminho certo para garantir a fiabilidade e sustentabilidade da base de dados no longo prazo.

A equipa de desenvolvimento deve agora continuar a prestar serviços à empresa Bela Rent-a-Car garantindo a operacionalidade da BD e procurando melhorar o seu funcionamento com base no comentário critico dos funcionários que a utilizam.

Para além disso, de forma a otimizar ainda mais a funcionalidade da empresa, a equipa de desenvolvimento e Octávio Faísca estão a discutir a possibilidade da implementação de uma aplicação para o telemóvel.

# Referências

- Cândido, C. (2020). *brModelo*. SIS4.com. <http://www.sis4.com/brModelo/>
- Connolly, T., & Begg, C. (2015). Database Systems: A Practical Approach to Design, Implementation, and Management (6th ed. – Global ed.). Pearson Education Limited.
- dbis-uibk. (2025a). RelaX – relational algebra calculator. <https://dbis-uibk.github.io/relax/landing>
- dbis-uibk. (2025b). RelaX – Help. <https://dbis-uibk.github.io/relax/help>
- Oracle. (2025a, May 30). MySQL 8.0 Reference Manual: Including MySQL NDB Cluster 8.0. MySQL. <https://downloads.mysql.com/docs/refman-8.0-en.a4.pdf>
- Python Software Foundation. (2025a). Python Developer 3.13.3. <https://docs.python.org/3/>
- PostgreSQL Global Development Group. (n.d.). PostgreSQL 16 documentation. PostgreSQL. <https://www.postgresql.org/docs/>
- Python Software Foundation. (n.d.). Python 3 documentation. <https://docs.python.org/3/>
- Python Software Foundation. (n.d.). json — JSON encoder and decoder. In Python 3 documentation. <https://docs.python.org/3/library/json.html>
- Python Software Foundation. (n.d.). csv — CSV File Reading and Writing. In Python 3 documentation. <https://docs.python.org/3/library/csv.html>

## **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.>>

<b>BD</b>	Base de Dados
<b>1FN</b>	Primeira Forma Normal
<b>2FN</b>	Segunda Forma Normal
<b>3FN</b>	Terceira Forma Normal
<b>RC</b>	<i>Requisito de Controlo</i>
<b>RD</b>	<i>Requisito de Descrição</i>
<b>RH</b>	<i>Recursos Humanos</i>
<b>RM</b>	<i>Requisito de Manipulação</i>
<b>SBD</b>	<i>Sistema de Bases de Dados</i>
<b>SGBD</b>	<i>Sistema de Gestão de Bases de Dados</i>
<b>SQL</b>	<i>Structured Query Language</i>
<b>CSV</b>	<i>Comma-Separated Values</i>
<b>JSON</b>	<i>JavaScript Object Notation</i>

## **Anexos**

# I. Anexo 1 – Diagrama de Gantt

Id	Tarefa	Data de começo	Data de finalização	Atribuído a:	Semana 1				Semana 2				Semana 3				
					Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Fase
1	Definição do Sistema	17/2/2025	7/3/2025														
1.1	Contexto	17/2/2025	1/3/2025	Filipa Gonçalves													
1.2	Motivação	17/2/2025	1/3/2025	Diogo Ribeiro													
1.3	Objetivos	17/2/2025	1/3/2025	Lucas Robertson													
1.4	Viabilidade	17/2/2025	1/3/2025	Carolina Martins													
1.5	Recursos e equipa de trabalho	1/3/2025	7/3/2025	Filipa Gonçalves													
1.6	Construção do Gantt	1/3/2025	7/3/2025	Lucas e Diogo													
1.7	Revisão	1/3/2025	7/3/2025	Filipa e Carolina													
2	Levantamento e análise dos requisitos	1/3/2025	24/3/2025														
2.1	Sugestões gerais	2/3/2025	7/3/2025	Toda a Equipa													
2.2	Organização dos requisitos levantados	8/3/2025	16/3/2025	Filipa e Diogo													
2.3	Análise e validação dos requisitos	17/3/2025	24/3/2025	Lucas e Carolina													
3	Modelação conceptual	24/3/2025	7/4/2025														
3.1	Apresentação da Abordagem de Modelação	24/3/2025	24/3/2025	Carolina Martins													
3.2	Identificação e Caracterização das Entidades	25/3/2025	27/3/2025	Filipa Gonçalves													
3.3	Identificação e Caracterização dos Relacionamentos	28/3/2025	1/4/2025	Lucas e Diogo													
3.4	Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos	1/4/2025	4/4/2025	Filipa e Carolina													
3.5	Apresentação e Explicação do Diagrama ER Produzido	4/4/2025	7/4/2025	Carolina e Diogo													
3.6	Validação do Modelo Conceitual	4/4/2025	7/4/2025	Lucas e Filipa													
4	Modelação Lógica	7/4/2025	21/4/2025														
4.1	Construção do Modelo de Dados Lógico	7/4/2025	10/4/2025	Filipa Gonçalves													
4.2	Aplicação e Apresentação do Modelo Lógico Produzido	11/4/2025	14/4/2025	Toda a equipa													
4.3	Normalização dos Dados	15/4/2025	19/4/2025	Filipa e Diogo													
4.4	Validação do Modelo com Interrogações do Utilizador	19/4/2025	20/4/2025	Toda a equipa													

g

Id	Tarefa	Data de começo	Data de finalização	Atribuído a:	Semana 4				Semana 5				Semana 6				
					Fase	Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12
1	Definição do Sistema	17/2/2025	7/3/2025														
1.1	Contexto	17/2/2025	1/3/2025	Filipa Gonçalves													
1.2	Motivação	17/2/2025	1/3/2025	Diogo Ribeiro													
1.3	Objetivos	17/2/2025	1/3/2025	Lucas Robertson													
1.4	Viabilidade	17/2/2025	1/3/2025	Carolina Martins													
1.5	Recursos e equipa de trabalho	1/3/2025	7/3/2025	Filipa Gonçalves													
1.6	Construção do Gantt	1/3/2025	7/3/2025	Lucas e Diogo													
1.7	Revisão	1/3/2025	7/3/2025	Filipa e Carolina													
2	Levantamento e análise dos requisitos	1/3/2025	24/3/2025														
2.1	Sugestões gerais	2/3/2025	7/3/2025	Toda a Equipa													
2.2	Organização dos requisitos levantados	8/3/2025	16/3/2025	Filipa e Diogo													
2.3	Análise e validação dos requisitos	17/3/2025	24/3/2025	Lucas e Carolina													
3	Modelação conceptual	24/3/2025	7/4/2025														
3.1	Apresentação da Abordagem de Modelação	24/3/2025	24/3/2025	Carolina Martins													
3.2	Identificação e Caracterização das Entidades	25/3/2025	27/3/2025	Filipa Gonçalves													
3.3	Identificação e Caracterização dos Relacionamentos	28/3/2025	1/4/2025	Lucas e Diogo													
3.4	Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos	1/4/2025	4/4/2025	Filipa e Carolina													
3.5	Apresentação e Explicação do Diagrama ER Produzido	4/4/2025	7/4/2025	Carolina e Diogo													
3.6	Validação do Modelo Conceitual	4/4/2025	7/4/2025	Lucas e Filipa													
4	Modelação Lógica	7/4/2025	21/4/2025														
4.1	Construção do Modelo de Dados Lógico	7/4/2025	10/4/2025	Filipa Gonçalves													
4.2	Aplicação e Apresentação do Modelo Lógico Produzido	11/4/2025	14/4/2025	Toda a equipa													
4.3	Normalização dos Dados	15/4/2025	19/4/2025	Filipa e Diogo													
4.4	Validação do Modelo com Interrogações do Utilizador	19/4/2025	20/4/2025	Toda a equipa													

Id	Tarefa	Data de começo	Data de finalização	Atribuído a:	Semana 7				Semana 8				Semana 9				
					Col1	Col2	Col3	Col4	Col5	Col6	Col7	Col8	Col9	Col10	Col11	Col12	Col13
1	Definição do Sistema	17/2/2025	7/3/2025														
1.1	Contexto	17/2/2025	1/3/2025	Filipa Gonçalves													
1.2	Motivação	17/2/2025	1/3/2025	Diogo Ribeiro													
1.3	Objetivos	17/2/2025	1/3/2025	Lucas Robertson													
1.4	Viabilidade	17/2/2025	1/3/2025	Carolina Martins													
1.5	Recursos e equipa de trabalho	1/3/2025	7/3/2025	Filipa Gonçalves													
1.6	Construção do Gantt	1/3/2025	7/3/2025	Lucas e Diogo													
1.7	Revisão	1/3/2025	7/3/2025	Filipa e Carolina													
2	Levantamento e análise dos requisitos	1/3/2025	24/3/2025														
2.1	Sugestões gerais	2/3/2025	7/3/2025	Toda a Equipa													
2.2	Organização dos requisitos levantados	8/3/2025	16/3/2025	Filipa e Diogo													
2.3	Análise e validação dos requisitos	17/3/2025	24/3/2025	Lucas e Carolina													
3	Modelação conceptual	24/3/2025	7/4/2025														
3.1	Apresentação da Abordagem de Modelação	24/3/2025	24/3/2025	Carolina Martins													
3.2	Identificação e Caracterização das Entidades	25/3/2025	27/3/2025	Filipa Gonçalves													
3.3	Identificação e Caracterização dos Relacionamentos	28/3/2025	1/4/2025	Lucas e Diogo													
3.4	Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos	1/4/2025	4/4/2025	Filipa e Carolina													
3.5	Apresentação e Explicação do Diagrama ER Produzido	4/4/2025	7/4/2025	Carolina e Diogo													
3.6	Validação do Modelo Conceitual	4/4/2025	7/4/2025	Lucas e Filipa													
4	Modelação Lógica	7/4/2025	21/4/2025														
4.1	Construção do Modelo de Dados Lógico	7/4/2025	10/4/2025	Filipa Gonçalves													
4.2	Aplicação e Apresentação do Modelo Lógico Produzido	11/4/2025	14/4/2025	Toda a equipa													
4.3	Normalização dos Dados	15/4/2025	19/4/2025	Filipa e Diogo													
4.4	Validação do Modelo com Interrogações do Utilizador	19/4/2025	20/4/2025	Toda a equipa													

Id	A_Tarefa	Data de começo	Data de finalização	Attribuído a:	Semana 10				Semana 11				Semana 12				Semana 13				Semana 14				Semana 15			
					Cola	Cola	Cola	Cola	Cola	Cola	Cola																	
1	Definição do Sistema	17/2/2005	27/2/2005	Filipa Gonçalves																								
1.1	Contexto	17/2/2005	17/2/2005	Filipa Gonçalves																								
1.2	Motivação	17/2/2005	17/2/2005	Diogo Ribeiro																								
1.3	Objetivos	17/2/2005	17/2/2005	Lucas Robertson																								
1.4	Viabilidade	17/2/2005	17/2/2005	Carolina Martins																								
1.5	Recursos e equipa de trabalho	17/2/2005	17/2/2005	Filipa Gonçalves																								
1.6	Construção do Gantt	17/2/2005	17/2/2005	Lucas e Diogo																								
1.7	Revisão	17/2/2005	17/2/2005	Filipa e Carolina																								
2	Leveramento e análise dos requisitos	17/2/2005	24/2/2005	Toda a Equipa																								
2.1	Sugestões gerais	17/2/2005	27/2/2005	Toda a Equipa																								
2.2	Organização dos requisitos levantados	17/2/2005	16/3/2005	Filipa e Diogo																								
2.3	Análise e validação dos requisitos	17/2/2005	24/3/2005	Lucas e Carolina																								
3	Modelação conceptual	17/2/2005	24/3/2005	Carolina e Diogo																								
3.1	Aprendizagem da Abstracção da Modelação	17/2/2005	24/3/2005	Carolina Martins																								
3.2	Identificação e Categorização das Entidades	17/2/2005	25/3/2005	Filipa Gonçalves																								
3.3	Identificação e Categorização das Relacionamentos	17/2/2005	26/3/2005	Lucas e Diogo																								
3.4	Identificação e Categorização das Atributos das Entidades e dos Relacionamentos	17/2/2005	1/4/2005	Carolina e Diogo																								
3.5	Aprendizagem e Explicação do Diagrama ER Produktivo	17/2/2005	4/4/2005	Carolina e Diogo																								
3.6	Validação do Modelo Conceitual	17/2/2005	7/4/2005	Lucas e Filipa																								
4	Modelação Lógica	17/2/2005	7/4/2005	21/4/2005																								
4.1	Criação do Modelo de Dados Lógico	17/2/2005	7/4/2005	Filipa Gonçalves																								
4.2	Aplicação e Aprendizagem do Modelo Lógico Produktivo	17/2/2005	11/4/2005	Toda a equipa																								
4.3	Normalização dos Dados	17/2/2005	15/4/2005	Filipa e Diogo																								
4.4	Validação do Modelo com Interrogações do Utilizador	17/2/2005	19/4/2005	Toda a equipa																								
5	Implementação Física	24/4/2005	24/4/2005	5/5/2005																								
5.1	Aprendizagem e explicação da base de dados implementada	24/4/2005	26/4/2005	Carolina Martins																								
5.2	Criação de utilizadores da base de dados	24/4/2005	26/4/2005	Diogo Ribeiro																								
5.3	Povoamento da base de dados	24/4/2005	26/4/2005	Lucas Robertson																								
5.4	Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)	24/4/2005	26/4/2005	Filipa Gonçalves																								
5.5	Definição e categorização de tipos de utilizador em SQL	27/4/2005	27/4/2005	Diogo Ribeiro																								
5.6	Tradução das interrogações do utilizador para SQL	27/4/2005	27/4/2005	Lucas Robertson																								
5.7	Indexação do Sistema de Dados	27/4/2005	27/4/2005	Filipa Gonçalves																								
5.8	Implementação de procedimentos, funções e gatilhos	27/4/2005	27/4/2005	Lucas Robertson																								
5.9	Backup da Base de Dados	27/4/2005	28/4/2005	Carolina Martins																								
6	Implementação do sistema de migração dados	28/4/2005	28/4/2005	Carolina Martins																								
6.1	Descrição das fontes de dados utilizadas	28/4/2005	28/4/2005	Carolina Martins																								
6.2	Descrição do programa implementado	28/4/2005	28/4/2005	Diogo Ribeiro																								
7	Conclusões e Trabalho Futuro	28/4/2005	28/4/2005	Filipa Gonçalves																								

## II. Anexo 1 – Tabela Geral dos requisitos

Nº	Data	Hora	Tipo	Descrição	Vista de Utilização	Fonte	Analista
1	03/03/2025	15:15h	Descrição	Um cliente é caracterizado pelo seu nome, morada(rua, localidade, código-postal), NIF, local de trabalho e contactos.	Clientes	Octávio Faísca	Filipa Gonçalves
2	03/03/2025	15:17h	Descrição	Um cliente deve ser identificado por um número sequencial único.	Clientes	Leonidas Lindt	Filipa Gonçalves
3	03/03/2025	15:21h	Descrição	Os contactos dos clientes e dos funcionários podem ser diversos: números de telefone e endereços de email distintos entre clientes/funcionários. É obrigatório pelo menos um número de telefone.	Clientes/Gestão	Octávio Faísca	Carolina Martins
4	03/03/2025	15:22h	Descrição	Um Funcionário é caracterizado por um identificador sequencial único, o nome, NIF, o salário e os contactos.	Gestão	Octávio Faísca	Lucas Robertson
5	03/03/2025	15:23h	Descrição	Um automóvel é caracterizado por um identificador sequencial único, a marca, a cilindragem, o ano, o estado(Disponível, Ocupado), o tipo de consumo(Gasolina, Gasóleo, Diesel, ...) e o preço por dia do respetivo automóvel.	Automóveis	Alberto Senna	Diogo Ribeiro
6	03/03/2025	15:25h	Descrição	Uma filial é caracterizada por um identificador único e a sua respetiva localização.	Gestão	Leonidas Lindt	Diogo Ribeiro
7	03/03/2025	15:26h	Descrição	Um cliente pode alugar mais do que um automóvel, mas um automóvel apenas pode ser alugado por um único cliente.	Clientes	Octávio Faísca	Filipa Gonçalves
8	04/03/2025	15:27h	Descrição	Um aluguer é definido por um identificador sequencial único, a data de inicio, a data de fim, a filial em que o automóvel é recolhido, a filial em que o automóvel é devolvido, o preço do aluguer e uma multa.	Gestão	Octávio Faísca	Filipa Gonçalves
9	04/03/2025	10:04h	Descrição	Um aluguer tem de ter um funcionário associado.	Gestão	Alberto Senna	Carolina Martins
10	04/03/2025	10:05h	Controlo	Apenas gestores podem: adicionar e remover carros; adicionar e remover funcionários; adicionar e remover clientes.	Gestão	Leonidas Lindt	Filipa Gonçalves
11	04/03/2025	10:07h	Controlo	Apenas os gestores podem alterar informações referentes a clientes, automóveis, funcionários	Gestão	Leonidas Lindt	Lucas Robertson
12	04/03/2025	10:08h	Manipulação	Registrar novos clientes e alterar os dados dos existentes	Gestão	Octávio Faísca	Diogo Ribeiro
13	04/03/2025	10:10h	Manipulação	Registrar novos alugueres e fazer alterações aos já existentes	Gestão	Octávio Faísca	Diogo Ribeiro
14	04/03/2025	11:50h	Manipulação	Deve ser possível listar todos os automóveis que já foram alugados e os respetivos clientes que os alugaram assim como aqueles que nunca foram alugados.	Automóveis/Gestão	Leonidas Lindt	Carolina Martins
15	05/03/2025	11:52h	Manipulação	Listar todo o tipo de função que existe associado a pelo menos um funcionário (id, nome, designação, salariobase)	Gestão	Alberto Senna	Filipa Gonçalves
16	05/03/2025	11:53h	Manipulação	Verificar o estado de um determinado automóvel, isto é, ver se ele está ocupado ou disponível.	Gestão	Octávio Faísca	Filipa Gonçalves
17	05/03/2025	11:54h	Controlo	Apenas os gestores podem inserir novas funções.	Gestão	Octávio Faísca	Filipa Gonçalves
18	05/03/2025	11:55h	Controlo	Um cliente apenas pode estar registado uma vez.	Gestão	Alberto Senna	Filipa Gonçalves
19	05/03/2025	11:56h	Controlo	Ambos clientes e automóveis precisam de estar registados.	Gestão	Octávio Faísca	Lucas Robertson
20	05/03/2025	12:48h	Controlo	Alugueres podem ser registrados por todos os tipos de funcionários uma vez que estão todos autorizados.	Gestão	Octávio Faísca	Filipa Gonçalves
21	05/03/2025	12:49h	Controlo	Muitas podem ser atribuídas por todos os funcionários	Gestão	Leonidas Lindt	Lucas Robertson
22	05/03/2025	12:50h	Controlo	Um automóvel apenas pode ser alugado por um cliente	Clientes/Automóveis/Gestão	Leonidas Lindt	Filipa Gonçalves
23	05/03/2025	12:54h	Controlo	Apenas um gestor pode "mover" um automóvel		Octávio Faísca	Filipa Gonçalves
24	05/03/2025	12:54h	Descrição	Um automóvel encontra-se localizado numa Filial	Automóveis/Gestão	Octávio Faísca	Lucas Robertson
25	05/03/2025	12:54h	Descrição	Um funcionário precisa de trabalhar numa e numa única Filial		Octávio Faísca	Diogo Ribeiro
26	05/03/2025	12:56h	Descrição	Uma Filial tem de ter pelo menos um funcionário	Gestão	Alberto Senna	Carolina Martins
27	05/03/2025	15:55h	Manipulação	Deve ser possível associar apenas um cliente e apenas um funcionário a alugueres	Gestão	Leonidas Lindt	Lucas Robertson
28	05/03/2025	15:55h	Manipulação	O preço de um aluguer deve ser calculado automaticamente	Gestão	Leonidas Lindt	Diogo Ribeiro
29	06/03/2025	10:05h	Descrição	Uma Função é designada por um identificador único e a respetiva designação.	Gestão	Octávio Faísca	Carolina Martins
30	06/03/2025	10:07h	Descrição	Um Funcionário pode exercer mais do que uma função	Gestão	Octávio Faísca	Lucas Robertson
31	06/03/2025	10:08h	Controlo	Um aluguer apenas pode ser registrado por um único funcionário e um funcionário pode registrar diversos alugueres.	Gestão	Octávio Faísca	Diogo Ribeiro
32	06/03/2025	10:10h	Controlo	Um aluguer apenas é referente a um único automóvel	Alugueres/Gestão/Automóvel	Octávio Faísca	Carolina Martins
33	06/03/2025	11:50h	Descrição/Controlo	Um aluguer tem de ser iniciado e terminado numa filial, não necessariamente a mesma		Octávio Faísca	Filipa Gonçalves
34	06/03/2025	15:23h	Manipulação	Gerar uma lista do numero de alugueres das diferentes marcas dos automóveis já alugados	Gestão	Alberto Senna	Lucas Robertson
35	06/03/2025	15:25h	Manipulação	Gerar uma lista ordenada por ordem decrescente de rendimento todas as filiais. (Id, Localização)	Gestão	Octávio Faísca	Diogo Ribeiro
36	06/03/2025	15:26h	Manipulação	Listar todos os clientes que já alugaram automóveis em mais do que uma filial	Gestão/Cientes	Leonidas Lindt	Carolina Martins
37	06/03/2025	15:27h	Manipulação	Verificar a Filial com um maior número de automóveis disponíveis.	Gestão	Octávio Faísca	Diogo Ribeiro
38	06/03/2025	15:30h	Manipulação	Listar todos os alugueres associados a um cliente com base no seu identificador	Gestão	Octávio Faísca	Filipa Gonçalves
39	06/03/2025	15:38h	Controlo	O sistema deve permitir importar dados de cada filial	Gestão	Octávio Faísca	Filipa Gonçalves
40	06/03/2025	15:45h	Controlo	Devem ser feitos backups diários para garantir a segurança de dados	Gestão	Octávio Faísca	Lucas Robertson
41	06/03/2025	16:30h	Controlo	Um automóvel apenas pode estar registado uma única vez	Gestão	Octávio Faísca	Diogo Ribeiro
42	06/03/2025	17:30h	Controlo	Um funcionário apenas pode estar registado uma única vez	Gestão	Octávio Faísca	Carolina Martins
43	06/03/2025	18:30h	Controlo	Apenas o Dono da empresa pode criar filiais, eliminar ou alterar as informações.	Gestão	Octávio Faísca	Filipa Gonçalves

### III. Anexo 3 – Requisitos categorizados

#### Requisitos de descrição

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RD1	1	Um cliente é caracterizado pelo seu nome, morada(rua, localidade, código-postal), NIF, local de trabalho e contactos.	Clientes	Diogo Ribeiro
RD2	2	Um cliente deve ser identificado por um número sequencial único.	Clientes	Carolina Martins
RD3	7	Um cliente pode alugar mais do que um automóvel ao mesmo tempo, e um automóvel apenas pode ter um cliente associado	Clientes	Filipa Gonçalves
RD4	3	Os contactos dos clientes podem ser diversos: números de telefone e endereços de email distintos entre clientes. É obrigatório pelo menos um número de telefone.	Clientes	Lucas Robertson
RD5	5	Um automóvel é caracterizado por um identificador sequencial único, a marca, a kilometragem, o ano, o estado, o tipo de consumo e o preço por dia do respetivo automóvel.	Automóveis	Filipa Gonçalves
RD6	5	O tipo de consumo de uma automóvel apenas pode estar entre os seguintes: Gasolina, Gasóleo, Elétrico, Híbrido	Automóveis	Filipa Gonçalves
RD7	5	O estado do automóvel apenas pode estar entre os seguintes: Disponível, Ocupado.	Automóveis	Lucas Robertson
RD8	6	Uma filial é caracterizada por um identificador único e a sua respetiva localização.	Gestão	Diogo Ribeiro
RD9	24	Um automóvel encontra-se localizado numa Filial	Gestão	Diogo Ribeiro
RD10	4	Um Funcionário é caracterizado por um identificador sequencial único, o nome, NIF, o salário e os contactos.	Gestão	Carolina Martins
RD11	3	Os contactos dos funcionários podem ser diversos: números de telefone e endereços de email distintos entre funcionários. É obrigatório pelo menos um número de telefone e um endereço de email.	Gestão	Filipa Gonçalves
RD12	29	Uma Função é designada por um identificador único e a respetiva designação e o salario base.	Gestão	Carolina Martins
RD13	30	Um funcionário pode executar mais do que uma função	Gestão	Lucas Robertson
RD14	9	Um aluguer tem de ter um funcionário associado.	Alugueres	Filipa Gonçalves
RD15	8	Um aluguer é definido por um identificador sequencial único, a data de inicio, a data de fim, a filial em que o automóvel é recolhido, a filial em que o automóvel é devolvido, o preço do aluguer e uma multa.	Alugueres	Diogo Ribeiro
RD16	25	Um funcionário tem obrigatoriamente de trabalhar numa Filial	Gestão	Carolina Martins
RD16	25	Um funcionário não pode trabalhar em mais que uma Filial ao mesmo tempo.	Gestão	Filipa Gonçalves
RD17	26	Uma Filial pode ter mais do que um funcionário, mas tem de ter pelo menos um.	Gestão	Lucas Robertson
RD18	32	Um Aluguer apenas pode ter um automóvel associado. Podem ser feitos múltiplos alugueres, no entanto cada aluguer é apenas referente a um único automóvel.	Alugueres	Diogo Ribeiro
RD19	33	Um aluguer tem de ser iniciado numa Filial.	Alugueres	Carolina Martins
RD20	33	Um aluguer tem de ser finalizado numa Filial	Alugueres	Carolina Martins

## Requisitos de controlo

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RC1	32	Um aluguer apenas é referente a um único automóvel	Gestão	Filipa Gonçalves
RC2	19	Clientes que não estão registados não podem alugar automóveis.	Clientes/Gestão	Filipa Gonçalves
RC3	19	Automóveis que não estão registados não podem ser alugados	Automoveis/Gestão	Carolina Martins
RC4	20	Todos os funcionários podem registrar alugueres uma vez que estão todos autorizados.	Gestão	Lucas Robertson
RC5	21	Todos os funcionários podem multar clientes.	Gestão	Diogo Ribeiro
RC6	23	Um carro apenas pode ser "transportado" de uma filial para outra pelo respectivos gerente de cada filial.	Gestão	Filipa Gonçalves
RC7	18	Um cliente apenas tem uma única ficha de cliente mesmo que alugue carros nos diversos polos.	Clientes/Gestão	Filipa Gonçalves
RC8	22	Um carro não pode ser alugado por mais do que um cliente ao mesmo tempo.	Automoveis/Gestão	Carolina Martins
RC9	10	Apenas os gestores podem inserir novas funções de funcionários	Gestão	Filipa Gonçalves
RC10	10	Apenas os gestores podem adicionar novos funcionários e modificar ou remover funcionários já existentes.	Gestão	Lucas Robertson
RC11	10	Apenas os gestores podem adicionar novos automoveis e remover ou modificar automóveis já existentes.	Gestão	Diogo Ribeiro
RC12	31	Um aluguer é registrado por um único funcionário	Alugueres	Diogo Ribeiro
RC13	10	Apenas os gestores podem adicionar novos clientes e modificar ou remover clientes já existentes	Gestão	Diogo Ribeiro
RC14	43	Apenas o Dono da empresa pode adicionar novas filiais e modificar ou remover filiais já existentes	Gestão	Filipa Gonçalves

## Requisitos de Manipulação

Nº	Requisito Original	Descrição	Vista de Uso	Revisor
RM1	12	Registar novos clientes e alterar os dados dos existentes	Gestão	Filipa Gonçalves
RM2	13	Registrar novos alugueres e fazer alterações aos já existentes	Descrição	Diogo Ribeiro
RM3	15	Adicionar novos funcionários, alterar dados já existentes e remover funcionários existentes	Gestão	Filipa Gonçalves
RM4	11, 17	Inserir novas funções de funcionários	Gestão	Carolina Martins
RM5	16	Verificar o estado de um determinado automóvel, isto é, ver se ele está ocupado ou disponível.	Automoveis/Gestão	Lucas Robertson
RM6	27	Associar apenas um cliente e apenas um funcionário a um ou mais alugueres.	Gestão	Filipa Gonçalves
RM7	28	O preço de aluguer deve ser automaticamente calculado ao colocar as informações de um aluguer	Gestão	Filipa Gonçalves
RM8	14	Listar todos os automóveis que já foram alugados e os respetivos clientes que os alugaram (Id, Nome, Id do automóvel, marca, ano, estado, PrecoDia, TipoConsumo)	Automóveis/Gestão	Carolina Martins
RM9	15	Listar todo o tipo de função que existe associado a pelo menos um funcionário (id, nome, designação, salariobase)	Gestão	Diogo Ribeiro
RM10	14	Listar todos os automóveis que nunca foram alugados.	Gestão	Lucas Robertson
RM11	14	Listas todos os clientes que não têm carros alugados no momento.	Gestão	Filipa Gonçalves
RM12	14	Listar todos os alugueres que estão a acontecer atualmente(id, automovel, cliente, preco, datainicio, datafim, filialinicio filialfim)	Gestão	Carolina Martins
RM13	34	Gerar uma lista do numero de alugueres das diferentes marcas de automóveis já alugados	Gestão	Filipa Gonçalves
RM14	35	Gerar uma lista ordenada por ordem decrescente de rendimento todas as filiais. (Id, Localização)	Gestão	Lucas Robertson
RM15	36	Listar todos os clientes assim como as filiais onde estes fizeram alugueres (id do cliente, nome do cliente, id da filial e localização da filial)	Gestão/Cientes	Diogo Ribeiro
RM16	37	Verificar a Filial com um maior número de automóveis.	Gestão	Carolina Martins
RM17	38	Listar todos os alugueres associados a um cliente com base no seu identificador	Gestão	Filipa Gonçalves



## IV. Anexo 4 – Relações RelaX

group: Bela RentACar

```
Cliente = {
    Id:number,        Nome:string,        Rua:string,        Localidade:string,
    CodigoPostal:string, NIF:number, LocalTrabalho:string
    1, 'Akio Toyoda', 'Rua Toyoda', 'Japao', '1234-234', 111111111, 'Toyota
Group'
    2, 'Oliver Blume', 'Rua Porsche', 'Alemanha', '5432-111', 222222222,
'Volkswagen Group'
    3, 'Carlos Tavares', 'Rua Europa', 'Portugal', '1500-000', 333333333,
'Ste llantis'
    4, 'Jim Farley', 'Ford Street', 'Estados Unidos', '9021-420', 444444444,
'Ford Motor Company'
    5, 'Luca de Meo', 'Via Milano', 'Itália', '2010-500', 555555555, 'Renault
Group'
    6, 'Toshihiro Mibe', 'Rua Honda', 'Japao', '1245-678', 666666666, 'Honda
Motor Co.'
    7, 'Makoto Uchida', 'Rua Yokohama', 'Japao', '1267-890', 777777777,
'Nissan Motor Co.'
    8, 'Mary Barra', 'Detroit Ave', 'Estados Unidos', '4822-300', 888888888,
'General Motors'
    9, 'Harald Krüger', 'München Straße', 'Alemanha', '8033-500', 999999999,
'BMW Group'
    10, 'Zhou Xiaoqing', 'Rua Geely', 'China', '1000-888', 101010101, 'Geely
Holding Group'
}
```

```
Emails = {
    Cliente:number, Email:string
    1, 'akio.toyoda@toyota.jp'
    2, 'oliver.blume@vwgroup.de'
    3, 'carlos.tavares@stellantis.com'
    4, 'jim.farley@ford.com'
    5, 'luca.demeo@renault.it'
    6, 'toshihiro.mibe@honda.jp'
    7, 'makoto.uchida@nissan.jp'
    8, 'mary.barra@gm.com'
```

```

9, 'harald.kruger@bmw.de'
10, 'zhou.xiaoqing@geely.cn'
}

Telefones = {
    Cliente:number, Telefone:string
    1, '910234567'
    2, '920345678'
    3, '930456789'
    4, '940567890'
    5, '950678901'
    6, '960789012'
    7, '970890123'
    8, '980901234'
    9, '990012345'
    10, '900123456'
}

Funcionario = {
    Id:number, Nome:string, NIF:number, Email:string, Telefone:string,
    Filial:number
    1, 'Octavio Faísca', 111223333, 'octfaisca@belarentacar.com',
    '960123456', 1
    2, 'Leonidas Lindt', 222334444, 'lindt@belarentacar.com', '961234567',
    2
    3, 'Afonso Senna', 333445555, 'senna@belarentacar.com', '962345678', 3
    4, 'José Martins', 444556666, 'jose.martins@belarentacar.com',
    '963456789', 1
    5, 'Eva Rocha', 555667777, 'eva.rocha@belarentacar.com', '964567890', 2
}

Funcao = {
    Id:number, Designacao:string, SalarioBase:number
    1, 'Gestor de Filial', 2000
    2, 'Assistente de Atendimento', 1100
    3, 'Técnico de Manutenção', 1300
    4, 'Operador de Reservas', 1200
    5, 'Gestor Comercial', 1800
}

```

```

FuncaoFuncionario = {
    Funcionario:number, Funcao:number
    1, 1
    2, 1
    3, 1
    4, 2
    5, 5
}

Filial = {
    Id:number, Localizacao:string
    1, 'Portugal'
    2, 'Mónaco'
    3, 'Bélgica'
}

Automovel = {
    Id:number, Marca:string, kilometragem:number, ano:number,
Estado:string, PrecoDia:number, TipoConsumo:string, Filial:number
    1, 'Toyota Corolla', 45000, 2020, 'Disponível', 45, 'Gasolina', 1
    2, 'Volkswagen Golf', 30000, 2021, 'Ocupado', 50, 'Diesel', 2
    3, 'Peugeot 208', 15000, 2022, 'Disponível', 40, 'Gasolina', 3
    4, 'Renault Clio', 60000, 2019, 'Manutenção', 38, 'Gasolina', 1
    5, 'Ford Focus', 50000, 2020, 'Disponível', 42, 'Diesel', 2
    6, 'BMW Série 1', 25000, 2021, 'Ocupado', 65, 'Gasolina', 3
    7, 'Fiat 500', 12000, 2023, 'Disponível', 35, 'Elétrico', 1
    8, 'Audi A3', 40000, 2020, 'Disponível', 60, 'Gasolina', 2
    9, 'Mercedes A-Class', 28000, 2022, 'Disponível', 70, 'Gasolina', 3
    10, 'Nissan Leaf', 18000, 2021, 'Ocupado', 55, 'Elétrico', 1
    11, 'Honda Civic', 22000, 2022, 'Disponível', 48, 'Gasolina', 1
    12, 'Hyundai Ioniq 5', 8000, 2023, 'Disponível', 75, 'Elétrico', 2
    13, 'Opel Corsa', 35000, 2021, 'Manutenção', 37, 'Gasolina', 3
    14, 'Tesla Model 3', 15000, 2022, 'Ocupado', 85, 'Elétrico', 2
    15, 'Kia Ceed', 27000, 2020, 'Disponível', 43, 'Diesel', 1
}

Aluguer = {

```

```
        Id:number,      Preco:number,      DataInicio:date,      DataFim:date,
filialRecolha:number,      FilialDevolucao:number,      multa:number,
funcionario:number, cliente:number, automovel:number

1, 180, 2025-04-01, 2025-04-05, 1, 1, 0, 2, 1, 1
2, 100, 2025-03-15, 2025-03-17, 2, 2, 10, 1, 2, 2
3, 120, 2025-04-10, 2025-04-13, 3, 3, 0, 3, 3, 3
4, 76, 2025-02-05, 2025-02-07, 1, 2, 5, 5, 4, 4
5, 168, 2025-01-20, 2025-01-24, 2, 2, 0, 4, 5, 5
6, 195, 2025-03-01, 2025-03-04, 3, 3, 15, 2, 1, 6
7, 105, 2025-04-05, 2025-04-07, 1, 1, 0, 1, 2, 7
8, 240, 2025-03-22, 2025-03-26, 2, 1, 0, 5, 3, 8
9, 280, 2025-02-18, 2025-02-22, 3, 3, 20, 3, 4, 9
10, 110, 2025-04-01, 2025-04-03, 1, 1, 0, 4, 5, 10
}
```

## V. Anexo 5 – Expressões de Algebra relacional

RM8

$$\begin{aligned} \pi & \text{ Cliente.Id, Cliente.Nome, Automovel.Id, Marca, ano, Estado, PrecoDia,} \\ & \quad \text{TipoConsumo (} \\ & \quad (\text{Aluguer } \bowtie \text{ Aluguer.cliente} = \text{Cliente.Id Cliente}) \\ & \quad \bowtie \text{ Aluguer.automovel} = \text{Automovel.Id Automovel} \\ & \quad ) \end{aligned}$$

RM9

$$\pi \text{ Id, Designacao, SalarioBase } (\text{Funcao } \bowtie \text{ Id=Funcao FuncaoFuncionario})$$

RM10

$$\begin{aligned} \pi & \text{ Id, Marca (Automovel) - } \pi \text{ Automovel.Id, Automovel.Marca (} \\ & \quad \text{Aluguer } \bowtie \text{ Aluguer.automovel} = \text{Automovel.Id Automovel} \\ & \quad ) \end{aligned}$$

RM11

$$\begin{aligned} \pi & \text{ Nome, Localidade, CodigoPostal, NIF, LocalTrabalho (} \\ & \quad \text{Cliente} \\ & \quad ) - \pi \text{ Nome, Localidade, CodigoPostal, NIF, LocalTrabalho (} \\ & \quad (\sigma \text{ (DataInicio} \leq \text{DATE('2025-04-18')} \wedge \text{DataFim} \geq \text{DATE('2025-04-18')}) (} \\ & \quad \text{Aluguer}) \bowtie \text{ Cliente}) \end{aligned}$$

RM12

$$\begin{aligned} \pi & \text{ Nome, Marca, Preco, DataInicio, DataFim, filialRecolha, FilialDevolucao (} \\ & \quad (\sigma \text{ (DataInicio} \leq \text{DATE('2025-04-18')} \wedge \text{DataFim} \geq \text{DATE('2025-04-18')) (} \\ & \quad (\text{Aluguer } \bowtie \text{ Cliente}) \bowtie \text{ Automovel} \\ & \quad )) \end{aligned}$$

RM13

$$\begin{aligned} \gamma & \text{ Marca; count(*) } \rightarrow \text{total (} \\ & \quad \text{Aluguer } \bowtie \text{ Aluguer.automovel} = \text{Automovel.Id Automovel} \\ & \quad ) \end{aligned}$$

RM14

$$\begin{aligned} \tau & \text{ Receita DESC}(\gamma \text{ Filial.Id, Filial.Localizacao; sum(Aluguer.Preco) } \rightarrow \text{Receita(} \\ & \quad \text{Aluguer } \bowtie \text{ Aluguer.filialRecolha} = \text{Filial.Id Filial} \\ & \quad )) \end{aligned}$$

RM15

```
π Cliente.Id, Cliente.Nome, Filial.Id, Filial.Localizacao (
    (Aluguer ⋙ Aluguer.cliente = Cliente.Id Cliente)
    ⋙ Aluguer.filialRecolha = Filial.Id Filial
)
```

RM16

```
π Filial.Id, Filial.Localizacao, total (
    (γ Filial; count(*) → total (Automovel))
    ⋙ Filial.Id = Filial Filial
)
```

RM17

```
π Id, Preco, DataInicio, DataFim, filialRecolha, FilialDevolucao, multa,
    funcionario, automovel (
        σ cliente = 1 (Aluguer)
)
```

## VI. Anexo 6 – Script de criação da BD BelaRentacar

```
-- Criar a base de dados
DROP DATABASE IF EXISTS BelaRentaCar;
CREATE DATABASE IF NOT EXISTS BelaRentaCar
CHARACTER SET utf8mb4
COLLATE utf8mb4_unicode_ci;

USE BelaRentaCar;

-- Tabela Filial
CREATE TABLE IF NOT EXISTS Filial (
    Id INT NOT NULL AUTO_INCREMENT,
    Localizacao VARCHAR(75) UNIQUE NOT NULL,
    PRIMARY KEY(Id)
)ENGINE = InnoDB;

-- Tabela Funcionario
CREATE TABLE IF NOT EXISTS Funcionario (
    Id INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(75) NOT NULL,
    NIF VARCHAR(9) NOT NULL UNIQUE CHECK (LENGTH(NIF) = 9),
    Salario DECIMAL(8,2) NOT NULL CHECK (Salario >= 0),
    Telefone VARCHAR(20) NOT NULL,
    Email VARCHAR(200) NULL,
    FilialId INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (FilialId) REFERENCES Filial(Id)
)ENGINE = InnoDB;

-- Tabela Funcao
CREATE TABLE IF NOT EXISTS Funcao (
    Id INT NOT NULL AUTO_INCREMENT,
    Designacao VARCHAR(75) UNIQUE NOT NULL,
    SalarioBase DECIMAL(8,2) NOT NULL CHECK (SalarioBase >= 0),
    PRIMARY KEY(Id)
)ENGINE = InnoDB;

-- Tabela de associação Exerce (N:N entre Funcionario e Funcao)
```

```

CREATE TABLE IF NOT EXISTS Exerce (
    FuncionarioId INT NOT NULL,
    FuncaoId INT NOT NULL,
    PRIMARY KEY (FuncionarioId, FuncaoId),
    FOREIGN KEY (FuncionarioId) REFERENCES Funcionario(Id),
    FOREIGN KEY (FuncaoId) REFERENCES Funcao(Id)
)ENGINE = InnoDB;

```

-- Tabela Cliente

```

CREATE TABLE IF NOT EXISTS Cliente (
    Id INT NOT NULL AUTO_INCREMENT,
    Nome VARCHAR(75) NOT NULL,
    NIF VARCHAR(9) NOT NULL UNIQUE CHECK (LENGTH(NIF) = 9),
    LocalTrabalho VARCHAR(75) NOT NULL,
    Rua VARCHAR(75) NOT NULL,
    Localidade VARCHAR(75) NOT NULL,
    CodigoPostal VARCHAR(10) NOT NULL,
    PRIMARY KEY(Id)
)ENGINE = InnoDB;

```

-- Tabela de Contactos do Cliente (atributo multivalorado)

```

CREATE TABLE IF NOT EXISTS Cliente_Contacto (
    ClienteId INT NOT NULL,
    Telefone VARCHAR(20) NOT NULL,
    Email VARCHAR(200) NULL,
    PRIMARY KEY (ClienteId, Telefone),
    FOREIGN KEY (ClienteId) REFERENCES Cliente(Id)
)ENGINE = InnoDB;

```

-- Tabela Automovel

```

CREATE TABLE IF NOT EXISTS Automovel (
    Id INT NOT NULL AUTO_INCREMENT,
    Marca VARCHAR(75) NOT NULL,
    Kilometragem DECIMAL(8,3) NOT NULL,
    Ano YEAR NOT NULL,
    Estado VARCHAR(10) NOT NULL CHECK ( Estado IN ( "Disponível", "Ocupado") ),
    TipoConsumo VARCHAR(10) NOT NULL CHECK ( TipoConsumo IN ("Gasolina",
    "Gasóleo", "Elétrico", "Híbrido")),
    PrecoDia DECIMAL(8,2) NOT NULL,
    FilialId INT NOT NULL,

```

```

        PRIMARY KEY (Id),
        FOREIGN KEY (FilialId) REFERENCES Filial(Id)
    )ENGINE = InnoDB;

-- Tabela Aluguer
CREATE TABLE IF NOT EXISTS Aluguer (
    Id INT NOT NULL AUTO_INCREMENT,
    DataInicio DATETIME NOT NULL,
    DataFim DATETIME NOT NULL,
    CHECK (DataInicio < DataFim),
    Preco DECIMAL(10,2) NOT NULL,
    Multa DECIMAL(5,2) NULL,
    ClienteId INT NOT NULL,
    FuncionarioId INT NOT NULL,
    AutomovelId INT NOT NULL,
    RecolhidoFilialId INT NOT NULL,
    DevolvidoFilialId INT NOT NULL,
    PRIMARY KEY(Id),
    FOREIGN KEY (ClienteId) REFERENCES Cliente(Id),
    FOREIGN KEY (FuncionarioId) REFERENCES Funcionario(Id),
    FOREIGN KEY (AutomovelId) REFERENCES Automovel(Id),
    FOREIGN KEY (RecolhidoFilialId) REFERENCES Filial(Id),
    FOREIGN KEY (DevolvidoFilialId) REFERENCES Filial(Id)
)ENGINE = InnoDB;

```

```

Source ProceduresFunctionTrigger.sql;
Source views.sql;
Source Utilizadores.sql;

```

```

-- Inserir Filiais
INSERT INTO Filial (Localizacao) VALUES
('Portugal'),
('Bélgica'),
('Mónaco');

```

## VII. Anexo 7 – Povoamento da BD

```
USE BelaRentaCar;
-- Inserir Filiais
INSERT INTO Filial (Localizacao) VALUES
('Portugal'),
('Bélgica'),
('Mónaco');

SET @ultimoFuncionarioId = (SELECT COALESCE(MAX(Id), 0) FROM Funcionario);
SET @ultimoClienteId = (SELECT COALESCE(MAX(Id), 0) FROM Cliente);
SET @ultimoAutomovelId = (SELECT COALESCE(MAX(Id), 0) FROM Automovel);
SET @ultimoFuncaoId = (SELECT COALESCE(MAX(Id), 0) FROM Funcao);

INSERT INTO Cliente (Nome, Rua, Localidade, CodigoPostal, NIF, LocalTrabalho)
VALUES
('Akio Toyoda', 'Rua Toyoda', 'Japao', '1234-234', 111111111, 'Toyota Group'),
('Oliver Blume', 'Rua Porsche', 'Alemanha', '5432-111', 222222222, 'Volkswagen
Group'),
('Carlos Tavares', 'Rua Europa', 'Portugal', '1500-000', 333333333,
'Ste llantis'),
('Jim Farley', 'Ford Street', 'Estados Unidos', '9021-420', 444444444, 'Ford
Motor Company'),
('Luca de Meo', 'Via Milano', 'Itália', '2010-500', 555555555, 'Renault
Group'),
('Toshihiro Mibe', 'Rua Honda', 'Japao', '1245-678', 666666666, 'Honda Motor
Co.'),
('Makoto Uchida', 'Rua Yokohama', 'Japao', '1267-890', 777777777, 'Nissan Motor
Co.'),
('Mary Barra', 'Detroit Ave', 'Estados Unidos', '4822-300', 888888888, 'General
Motors'),
('Harald Krüger', 'München Straße', 'Alemanha', '8033-500', 999999999, 'BMW
Group'),
('Zhou Xiaoqing', 'Rua Geely', 'China', '1000-888', 101440101, 'Geely Holding
Group');

INSERT INTO Cliente_Contacto (ClienteId, Telefone, Email) VALUES
```

```

(1 + @ultimoClienteId, '910234567', 'akio.toyoda@toyota.jp'),
(2 + @ultimoClienteId, '920345678', 'oliver.blume@vwggroup.de'),
(3 + @ultimoClienteId, '930456789', 'carlos.tavares@stellantis.com'),
(4 + @ultimoClienteId, '940567890', 'jim.farley@ford.com'),
(5 + @ultimoClienteId, '950678901', 'luca.demeo@renault.it'),
(6 + @ultimoClienteId, '960789012', 'toshihiro.mibe@honda.jp'),
(7 + @ultimoClienteId, '970890123', 'makoto.uchida@nissan.jp'),
(8 + @ultimoClienteId, '980901234', 'mary.barra@gm.com'),
(9 + @ultimoClienteId, '990012345', 'harald.kruger@bmw.de'),
(1 + @ultimoClienteId, '900123456', 'zhou.xiaqing@geely.cn');

```

```

INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado, PrecoDia, TipoConsumo,
FilialId) VALUES
('Toyota Corolla', 45000, 2020, 'Disponível', 45.00, 'Gasolina', 1),
('Volkswagen Golf', 30000, 2021, 'Ocupado', 50.00, 'Gasóleo', 2),
('Peugeot 208', 15000, 2022, 'Disponível', 40.00, 'Gasolina', 3),
('Renault Clio', 60000, 2019, 'Disponível', 38.00, 'Gasolina', 1),
('Ford Focus', 50000, 2020, 'Disponível', 42.00, 'Gasóleo', 2),
('BMW Série 1', 25000, 2021, 'Ocupado', 65.00, 'Gasolina', 3),
('Fiat 500', 12000, 2023, 'Disponível', 35.00, 'Elétrico', 1),
('Audi A3', 40000, 2020, 'Disponível', 60.00, 'Gasóleo', 2),
('Mercedes A-Class', 28000, 2022, 'Disponível', 70.00, 'Gasolina', 3),
('Nissan Leaf', 18000, 2021, 'Ocupado', 55.00, 'Elétrico', 1);

```

-- Inserir Funcionários

```

INSERT INTO Funcionario (Nome, NIF, Salario, Email, Telefone, FilialId) VALUES
('Cláudia Neves', '874512369', 3100.00, 'claudia.neves@belarentacar.com',
'963112233', 1),
('Rui Martins', '951753456', 2750.00, 'rui.martins@belarentacar.com',
'964223344', 2),
('Joana Ferreira', '782364159', 2900.00, 'joana.ferreira@belarentacar.com',
'965334455', 3),
('José Martins', '444556666', 2000.00, 'jose.martins@belarentacar.com',
'963456789', 1),
('Eva Rocha', '555667777', 2000.00, 'eva.rocha@belarentacar.com', '964567890',
2);

```

```

INSERT INTO Aluguer (DataInicio, DataFim, Preco, Multa, ClienteId,
FuncionarioId, AutomovelId, RecolhidoFilialId, DevolvidoFilialId) VALUES
('2025-04-01 00:00:00', '2025-04-05 00:00:00', 180.00, 0.00, 1 +
@ultimoClienteId, 2 + @ultimoFuncionarioId, 1 + @ultimoAutomovelId, 1, 1),
('2025-03-15 00:00:00', '2025-03-17 00:00:00', 100.00, 10.00, 2 +
@ultimoClienteId, 1 + @ultimoFuncionarioId, 2 + @ultimoAutomovelId, 2, 2),
('2025-04-10 00:00:00', '2025-04-13 00:00:00', 120.00, 0.00, 3 +
@ultimoClienteId, 3 + @ultimoFuncionarioId, 3 + @ultimoAutomovelId, 3, 3),
('2025-02-05 00:00:00', '2025-02-07 00:00:00', 76.00, 5.00, 4 +
@ultimoClienteId, 5 + @ultimoFuncionarioId, 4 + @ultimoAutomovelId, 1, 2),
('2025-01-20 00:00:00', '2025-01-24 00:00:00', 168.00, 0.00, 5 +
@ultimoClienteId, 4 + @ultimoFuncionarioId, 5 + @ultimoAutomovelId, 2, 2),
('2025-03-01 00:00:00', '2025-03-04 00:00:00', 195.00, 15.00, 1 +
@ultimoClienteId, 2 + @ultimoFuncionarioId, 6 + @ultimoAutomovelId, 3, 3),
('2025-04-05 00:00:00', '2025-04-07 00:00:00', 105.00, 0.00, 2 +
@ultimoClienteId, 1 + @ultimoFuncionarioId, 7 + @ultimoAutomovelId, 1, 1),
('2025-03-22 00:00:00', '2025-03-26 00:00:00', 240.00, 0.00, 3 +
@ultimoClienteId, 5 + @ultimoFuncionarioId, 8 + @ultimoAutomovelId, 2, 1),
('2025-02-18 00:00:00', '2025-02-22 00:00:00', 280.00, 20.00, 4 +
@ultimoClienteId, 3 + @ultimoFuncionarioId, 9 + @ultimoAutomovelId, 3, 3),
('2025-04-01 00:00:00', '2025-04-03 00:00:00', 110.00, 0.00, 5 +
@ultimoClienteId, 4 + @ultimoFuncionarioId, 1 + @ultimoAutomovelId, 1, 1);

```

-- Inserir Funções

```

INSERT INTO Funcao (Designacao, SalarioBase) VALUES
('Gestor de Filial', 2000),
('Assistente de Atendimento', 1100),
('Técnico de Manutenção', 1300),
('Operador de Reservas', 1200),
('Gestor Comercial', 1800);

```

-- Inserir associação Funcionario-Funcao (Exerce) com offset dinâmico

```

INSERT INTO Exerce (FuncionarioId, FuncaoId) VALUES
(@ultimoFuncionarioId + 1, @ultimoFuncaoId + 2),
(@ultimoFuncionarioId + 2, @ultimoFuncaoId + 2),
(@ultimoFuncionarioId + 3, @ultimoFuncaoId + 4),
(@ultimoFuncionarioId + 4, @ultimoFuncaoId + 1),
(@ultimoFuncionarioId + 5, @ultimoFuncaoId + 5);

```

## VIII. Anexo 8 – Views

```
CREATE VIEW CarrosDisponiveis AS  
SELECT * FROM Automovel a  
WHERE a.Estado = "Disponível";
```

```
CREATE VIEW OrigemUsers AS  
SELECT c.Localidade, count(*) AS UsersPorOrigem FROM Cliente c  
GROUP BY c.Localidade;
```

```
CREATE VIEW MediaPrecosMaisAlugadas AS  
SELECT AVG(au.PrecoDia) as MediaPrecoDiario, f.Localizacao FROM Aluguer a  
INNER JOIN Automovel au ON a.AutomovelId = au.Id  
INNER JOIN Filial f ON a.RecolhidoFilialId = f.Id  
GROUP BY f.Localizacao;
```

## IX. Anexo 9 – Utilizadores e permissões

```
-- =====
-- GESTORES
-- =====

DROP ROLE IF EXISTS 'gestorFilial';
CREATE ROLE 'gestorFilial';

-- RC9
GRANT INSERT ON BelaRentaCar.Funcao TO 'gestorFilial';

-- RC10
GRANT DELETE, UPDATE ON BelaRentaCar.Funcionario TO 'gestorFilial';
GRANT EXECUTE ON PROCEDURE BelaRentaCar.AddFuncionarioComFuncao TO
'gestorFilial';

-- RC11
GRANT INSERT, DELETE, UPDATE ON BelaRentaCar.Automovel TO 'gestorFilial';

-- RC13
GRANT DELETE, UPDATE ON BelaRentaCar.Cliente TO 'gestorFilial';
GRANT EXECUTE ON PROCEDURE BelaRentaCar.novoCliente TO 'gestorFilial';

GRANT INSERT, DELETE ON BelaRentaCar.Cliente_Contacto TO 'gestorFilial';

GRANT UPDATE (DataFim) ON BelaRentaCar.Aluguer TO 'gestorFilial';

DROP USER IF EXISTS 'octavio'@'localhost';
CREATE USER 'octavio'@'localhost' IDENTIFIED BY 'Octavio#2025';
GRANT DELETE, UPDATE ON BelaRentaCar.Filial to 'octavio'@'localhost';

DROP USER IF EXISTS 'leonidas'@'localhost';
CREATE USER 'leonidas'@'localhost' IDENTIFIED BY 'Leonidas#2025';

DROP USER IF EXISTS 'alberto'@'localhost';
CREATE USER 'alberto'@'localhost' IDENTIFIED BY 'Alberto#2025';
```

```

GRANT 'gestorFilial' TO 'octavio'@'localhost';
GRANT 'gestorFilial' TO 'leonidas'@'localhost';
GRANT 'gestorFilial' TO 'alberto'@'localhost';

-- =====
-- FUNCIONÁRIOS
-- =====

DROP ROLE IF EXISTS 'Funcionario';
CREATE ROLE 'Funcionario';
-- RC4
GRANT EXECUTE ON PROCEDURE BelaRentaCar.novoAluguer TO 'Funcionario';
-- RC5
GRANT UPDATE (Multa) ON BelaRentaCar.Aluguer TO 'Funcionario';
DROP USER IF EXISTS 'josemartins'@'localhost';
CREATE USER 'josemartins'@'localhost' IDENTIFIED BY 'JoseMartins#2025';

DROP USER IF EXISTS 'evarocha'@'localhost';
CREATE USER 'evarocha'@'localhost' IDENTIFIED BY 'EvaRocha#2025';

GRANT SELECT ON BelaRentaCar.* TO 'gestorFilial';
GRANT SELECT ON BelaRentaCar.Aluguer TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.Automovel TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.Cliente TO 'Funcionario';
GRANT SELECT ON BelaRentaCar.CarrosDisponiveis TO 'Funcionario';

GRANT 'Funcionario' TO 'josemartins'@'localhost';
GRANT 'Funcionario' TO 'evarocha'@'localhost';

GRANT 'Funcionario' TO 'octavio'@'localhost';
GRANT 'Funcionario' TO 'leonidas'@'localhost';
GRANT 'Funcionario' TO 'alberto'@'localhost';

SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'octavio'@'localhost';
GRANT EXECUTE ON PROCEDURE BelaRentaCar.CriarFilialFuncionarioCarro TO
'octavio'@'localhost';
SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'leonidas'@'localhost';
SET DEFAULT ROLE 'gestorFilial', 'Funcionario' TO 'alberto'@'localhost';

```

```
SET DEFAULT ROLE 'Funcionario' TO 'josemartins'@'localhost';
SET DEFAULT ROLE 'Funcionario' TO 'evarocha'@'localhost';

GRANT USAGE ON BelaRentaCar.* TO 'octavio'@'localhost';
GRANT USAGE ON BelaRentaCar.* TO 'leonidas'@'localhost';
GRANT USAGE ON BelaRentaCar.* TO 'alberto'@'localhost';

GRANT USAGE ON BelaRentaCar.* TO 'josemartins'@'localhost';
GRANT USAGE ON BelaRentaCar.* TO 'evarocha'@'localhost';
```

## X. Anexo 10 – Utilizadores e permissões

```
USE BelaRentaCar;

DELIMITER //

-- Função que calcula o custo do aluguer de um automóvel
DROP FUNCTION IF EXISTS calculaPreco;
CREATE FUNCTION calculaPreco(
    precoDia DECIMAL(8,2),
    inicio DATETIME,
    fim DATETIME
)
RETURNS DECIMAL(10,2)
DETERMINISTIC
BEGIN
    DECLARE precoTotal DECIMAL(10,2);
    DECLARE nrDias INT;

    SET nrDias = DATEDIFF(fim, inicio) + 1;
    SET precoTotal = precoDia * nrDias;

    RETURN precoTotal;
END;
// 

DELIMITER ;

DELIMITER //
-- Função que calcula se um aluguer pode ser criado
DROP FUNCTION IF EXISTS podeCriarAluguer;
CREATE FUNCTION podeCriarAluguer
    (Inicio DATETIME
    ,Fim DATETIME
    ,AutomovelId INT
    ,FuncionarioId INT
    ,FilOrigem INT)
RETURNS BOOLEAN
NOT DETERMINISTIC
```

```

READS SQL DATA
SQL SECURITY DEFINER
BEGIN
    DECLARE filialFuncionario INT; -- Id da filial de um funcionário
    DECLARE filialAutomovel INT; -- Id da filial de um carro
    DECLARE validade BOOLEAN;

    SET validade = FALSE;

    SELECT F.FilialId INTO filialFuncionario
        FROM Funcionario AS F
        WHERE F.Id = FuncionarioId;

    SELECT A.FilialId INTO filialAutomovel
        FROM Automovel AS A
        WHERE A.Id = AutomovelId;
        AND A.Estoque = 'Disponível';

    IF filialFuncionario = FilOrigem -- Garante que o funcionario trabalha na
    filial correta
        AND filialAutomovel = FilOrigem -- Garante que o automovel está
    na filial correta e está disponível
        AND NOT EXISTS( -- Garante que não há Alugueres em conflito com este
            SELECT 'conflito'
            FROM Aluguer AS A
            WHERE A.AutomovelId = AutomovelId
            AND A.DataInicio <= Fim
            AND Inicio <= A.DataFim
        )
    THEN
        SET validade = TRUE;
    END IF;
    RETURN validade;
END;
//


DELIMITER ;

DELIMITER //
DROP PROCEDURE IF EXISTS AddFuncionarioComFuncao;

```

```

CREATE PROCEDURE AddFuncionarioComFuncao(
    IN pNome VARCHAR(75),
    IN pNIF VARCHAR(9),
    IN pSalario DECIMAL(8,2),
    IN pTelefone VARCHAR(20),
    IN pEmail VARCHAR(200),
    IN pFilialId INT,
    IN pFuncaoId INT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT 'Erro no AddFuncionarioComFuncao' AS Mensagem;
        ROLLBACK;
    END;

    START TRANSACTION;

    INSERT INTO Funcionario (Nome, NIF, Salario, Telefone, Email, FilialId)
    VALUES (pNome, pNIF, pSalario, pTelefone, pEmail, pFilialId);

    INSERT INTO Exerce (FuncionarioId, FuncaoId)
    VALUES (LAST_INSERT_ID(), pFuncaoId);

    COMMIT;
    SELECT 'Sucesso na criação de Funcionário com Função' AS Mensagem;

END;
//
DELIMITER ;

DELIMITER //
```

DROP PROCEDURE IF EXISTS CriarFilialFuncionarioCarro;

```

CREATE PROCEDURE CriarFilialFuncionarioCarro(
    IN pLocalizacao VARCHAR(75),
    IN pNomeFuncionario VARCHAR(75),
    IN pNIFFuncionario VARCHAR(9),
    IN pSalarioFuncionario DECIMAL(8,2),
    IN pFilialId INT,
    IN pCarroId INT
)
BEGIN
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT 'Erro no CriarFilialFuncionarioCarro' AS Mensagem;
        ROLLBACK;
    END;

    START TRANSACTION;

    INSERT INTO FilialFuncionario (FilialId, FuncionarioId, CarroId)
    VALUES (pFilialId, LAST_INSERT_ID(), pCarroId);

    COMMIT;
    SELECT 'Sucesso na criação de Funcionário com Carro' AS Mensagem;

END;
//
DELIMITER ;
```

```

    IN pTelefoneFuncionario VARCHAR(20),
    IN pEmailFuncionario VARCHAR(200),
    IN pFuncaoId INT,
    IN pMarca VARCHAR(75),
    IN pKilometragem DECIMAL(8,3),
    IN pAno YEAR,
    IN pEstado VARCHAR(10),
    IN pTipoConsumo VARCHAR(10),
    IN pPrecoDia DECIMAL(8,2)
)
BEGIN
    DECLARE vFilialId INT;

    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        SELECT 'Erro no CriarFilialFuncionarioCarro' AS Mensagem;
        ROLLBACK;
    END;

    START TRANSACTION;

    INSERT INTO Filial (Localizacao) VALUES (pLocalizacao);
    SET vFilialId = LAST_INSERT_ID();

    CALL AddFuncionarioComFuncao(
        pNomeFuncionario,
        pNIFFuncionario,
        pSalarioFuncionario,
        pTelefoneFuncionario,
        pEmailFuncionario,
        vFilialId,
        pFuncaoId
    );

    INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado, TipoConsumo,
    PrecoDia, FilialId)
    VALUES (pMarca, pKilometragem, pAno, pEstado, pTipoConsumo, pPrecoDia,
    vFilialId);

```

```

        COMMIT;

        SELECT 'Sucesso na criação de uma Filial com Funcionário e Carro' AS
Mensagem;

END;
//

DELIMITER ;

DELIMITER //
-- Procedimento para criar um aluguer
DROP PROCEDURE IF EXISTS novoAluguer;
CREATE PROCEDURE novoAluguer
    (IN Inicio DATETIME, IN Fim DATETIME, IN ClienteId INT
     , IN FuncionarioId INT, IN AutomovelId INT, IN FilOrigem INT, IN
FilDestino INT)
BEGIN
    -- Cálculo do preço do aluguer
    DECLARE precoDia DECIMAL (8,2);
    DECLARE precoTotal DECIMAL (10,2);

    -- SELECT precoDia, Inicio, Fim;

    SELECT A.precoDia INTO precoDia
        FROM Automovel AS A
        WHERE A.Id = AutomovelId;

    SELECT calculaPreco (precoDia, Inicio, Fim) INTO precoTotal;

    IF podeCriarAluguer(Inicio, Fim, AutomovelId, FuncionarioId, FilOrigem)
    THEN
        START TRANSACTION;
        INSERT INTO Aluguer
            (DataInicio, DataFim, Preco, ClienteId, FuncionarioId,
AutomovelId, RecolhidoFilialId, DevolvidoFilialId)
            VALUES (Inicio, Fim, precoTotal, ClienteId, FuncionarioId,
AutomovelId, FilOrigem, FilDestino);
    END IF;
END;
//
```

```

        SELECT CONCAT('Aluguer criado com id ', LAST_INSERT_ID()) AS Mensagem;

        COMMIT;

        ELSE
            SELECT 'Aluguer não criado' AS Mensagem,
                   (SELECT FilialId FROM Funcionario WHERE Id = FuncionarioId)
            AS 'Filial do Funcionário',
                   (SELECT FilialId FROM Automovel WHERE Id = AutomovelId) AS
            'Filial do Automóvel',
            FilOrigem AS 'Filial de Origem';
        END IF;
    END;
    //

DELIMITER ;

DELIMITER //
-- Procedimento para criar um cliente
DROP PROCEDURE IF EXISTS novoCliente;
CREATE PROCEDURE novoCliente
    (IN Nome VARCHAR(75),
     IN NIF VARCHAR(9),
     IN LocalTrabalho VARCHAR(75),
     IN Rua VARCHAR(75),
     IN Localidade VARCHAR(75),
     IN CodigoPostal VARCHAR(10),
     IN Telefone VARCHAR(20),
     IN Inicio DATETIME, IN Fim DATETIME, IN FuncionarioId INT,
     IN AutomovelId INT, IN FilOrigem INT, IN FilDestino INT)
BEGIN
    DECLARE id INT;
    DECLARE nrAlugueres INT;

    DECLARE filialFuncionario INT; -- Id da filial de um funcionário
    DECLARE filialAutomovel INT; -- Id da filial de um carro

    SELECT F.FilialId INTO filialFuncionario
        FROM Funcionario AS F
        WHERE F.Id = FuncionarioId;

```

```

SELECT A.FilialId INTO filialAutomovel
    FROM Automovel AS A
    WHERE A.Id = AutomovelId
        AND A.Estado = 'Disponível';

IF filialFuncionario = FilOrigem AND filialAutomovel = FilOrigem
THEN
    START TRANSACTION;

    -- É necessário criar 1º a entrada do Cliente antes do Aluguer
    INSERT INTO Cliente
        (Nome, NIF, LocalTrabalho, Rua, Localidade, CodigoPostal)
    VALUES (Nome, NIF, LocalTrabalho, Rua, Localidade, CodigoPostal);

    SET id = LAST_INSERT_ID(); -- id do último cliente inserido (por
causa do auto_increment)

    INSERT INTO Cliente_Contacto -- Um cliente tem de ter
obrigatoriamente um número de telefone
        (ClienteId, Telefone)
    VALUES (id, Telefone);

    CALL novoAluguer(Inicio, Fim, id, FuncionarioId, AutomovelId,
FilOrigem, FilDestino);

    SELECT CONCAT('Cliente criado com id ', id) AS Mensagem;

    COMMIT;
ELSE
    SELECT 'Cliente não criado pois o aluguer é inválido' AS Mensagem,
filialFuncionario AS 'Filial do Funcionário',
filialAutomovel AS 'Filial do Automóvel';
END IF;

END;
//

DELIMITER ;

```

```
DELIMITER //
```

```

-- Procedure que garante que tudo o que depende de um Cliente também é apagado
com ele.
DROP PROCEDURE IF EXISTS deleteCliente;
CREATE PROCEDURE deleteCliente
    (IN idDado INT)
BEGIN
    DELETE FROM Cliente_Contacto
        WHERE ClienteId = idDado;
    DELETE FROM Aluguer
        WHERE ClienteId = idDado;
    DELETE FROM Cliente
        WHERE Id = idDado;
END;
//


DELIMITER ;

DELIMITER //
-- Trigger que garante que após um automóvel ser alugado, passa a estar Ocupado
e a pertencer à filial de entrega
DROP TRIGGER IF EXISTS tgOcupaAutomovel;
CREATE TRIGGER tgOcupaAutomovel
    AFTER INSERT ON Aluguer
    FOR EACH ROW
BEGIN
    IF now() BETWEEN NEW.DataInicio AND NEW.DataFim
    THEN
        UPDATE Automovel AS A
            SET A.Estado = 'Ocupado'
            , FilialId = NEW.DevolvidoFilialId
            WHERE A.Id = NEW.AutomovelId;
    END IF;
END;
//


DELIMITER ;

DELIMITER //
-- Evento que atualiza o estado dos automóveis
DROP EVENT IF EXISTS atualizaEstadoDiario;

```

```
CREATE EVENT atualizaEstadoDiario
    ON SCHEDULE EVERY 1 DAY
        STARTS CURRENT_DATE + INTERVAL 23 HOUR -- Alterável se quiserem
DO
    UPDATE Automovel AS AU
        INNER JOIN Aluguer AS AL
            ON AU.Id = AL.AutomovelId
        SET AU.Estoado = 'Disponível'
        WHERE CURDATE() = DATE(AL.DataFim);
//  
  
DELIMITER ;
```

## XI. Anexo 11 – Script de migração

```
import mysql.connector
from mysql.connector import IntegrityError
import psycopg2
from decimal import Decimal
import csv
import json

def insereCliente(cursor, clientes):
    for cliente in clientes:
        try:
            cursor.execute("""
                INSERT INTO Cliente (Nome, Rua, Localidade, CodigoPostal, NIF,
LocalTrabalho)
                VALUES (%s, %s, %s, %s, %s, %s)
                """, (
                    cliente['Nome'],
                    cliente['Rua'],
                    cliente['Localidade'],
                    cliente['CodigoPostal'],
                    cliente['NIF'],
                    cliente['LocalTrabalho']
                ))
        except IntegrityError as e:
            if e errno == 1062:
                print(f"Entrada duplicada Cliente ignorada: {e}")
            else:
                print(f"Erro de integridade Cliente: {e}")
        except Exception as e:
            print(f"Erro inesperado Cliente: {e}")

def insereClienteContacto(cursor, contactos, offset):
    for contacto in contactos:
        try:
            cursor.execute("""
                INSERT INTO Cliente_Contacto (ClienteId, Telefone, Email)
                VALUES (%s, %s, %s)
                """, (

```

```

        contacto['ClienteId'] + offset,
        contacto['Telefone'],
        contacto['Email']
    ))
except IntegrityError as e:
    if e.errno == 1062:
        print(f"Entrada duplicada Cliente_Contacto ignorada: {e}")
    else:
        print(f"Erro de integridade Cliente_Contacto: {e}")
except Exception as e:
    print(f"Erro inesperado Cliente_Contacto: {e}")

def insereAutomovel(cursor, automoveis):
    for auto in automoveis:
        try:
            cursor.execute("""
                INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado,
                TipoConsumo, PrecoDia, FilialId)
                VALUES (%s, %s, %s, %s, %s, %s)
                """,
                (
                    auto['Marca'],
                    auto['Kilometragem'],
                    auto['Ano'],
                    auto['Estado'],
                    auto['TipoConsumo'],
                    auto['PrecoDia'],
                    auto['FilialId']
                ))
        except IntegrityError as e:
            if e.errno == 1062:
                print(f"Entrada duplicada Automovel ignorada: {e}")
            else:
                print(f"Erro de integridade Automovel: {e}")
        except Exception as e:
            print(f"Erro inesperado Automovel: {e}")

def insereFuncionario(cursor, funcionarios):
    for func in funcionarios:
        try:
            cursor.execute("""

```

```

        INSERT INTO Funcionario (Nome, NIF, Salario, Email, Telefone,
FilialId)
            VALUES (%s, %s, %s, %s, %s, %s)
        """", (
            func['Nome'],
            func['NIF'],
            func['Salario'],
            func['Email'],
            func['Telefone'],
            func['FilialId']
        ))
    except IntegrityError as e:
        if e errno == 1062:
            print(f"Entrada duplicada Funcionario ignorada: {e}")
        else:
            print(f"Erro de integridade Funcionario: {e}")
    except Exception as e:
        print(f"Erro inesperado Funcionario: {e}")

def inserirAluguer(cursor,
alugueres, offsetCliente, offsetAutomovel, offsetFuncionario):
    for aluguer in alugueres:
        try:
            cursor.execute("""
                INSERT INTO Aluguer (DataInicio, DataFim, Preco, Multa,
FuncionarioId, ClienteId, AutomovelId, RecolhidoFilialId, DevolvidoFilialId)
                    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
            """", (
                aluguer['DataInicio'],
                aluguer['DataFim'],
                aluguer['Preco'],
                aluguer['Multas'],
                aluguer['FuncionarioId'] + offsetFuncionario,
                aluguer['ClienteId'] + offsetCliente,
                aluguer['AutomovelId'] + offsetAutomovel,
                aluguer['RecolhidoFilialId'],
                aluguer['DevolvidoFilialId']
            )))
        except IntegrityError as e:
            if e errno == 1062:

```

```

        print(f"Entrada duplicada Aluguer ignorada: {e}")
    else:
        print(f"Erro de integridade Aluguer: {e}")
except Exception as e:
    print(f"Erro inesperado Aluguer: {e}")

def insereFuncao(cursor, funcoes):
    for funcao in funcoes:
        try:
            cursor.execute("""
                INSERT INTO Funcao (Designacao, SalarioBase)
                VALUES (%s, %s)
            """, (
                funcao['Designacao'],
                funcao['SalarioBase']
            ))
        except IntegrityError as e:
            if e errno == 1062:
                print(f"Entrada duplicada Funcao ignorada: {e}")
            else:
                print(f"Erro de integridade Funcao: {e}")
        except Exception as e:
            print(f"Erro inesperado Funcao: {e}")

def insereExerce(cursor, exerces, offsetFuncao, offsetFuncionario):
    for exerce in exerces:
        try:
            cursor.execute("""
                INSERT INTO Exerce (FuncionarioId, FuncaoId)
                VALUES (%s, %s)
            """, (
                exerce['FuncionarioId'] + offsetFuncionario,
                exerce['FuncaoId'] + offsetFuncao
            ))
        except IntegrityError as e:
            if e errno == 1062:
                print(f"Entrada duplicada Exerce ignorada: {e}")
            else:
                print(f"Erro de integridade Exerce: {e}")
        except Exception as e:
            print(f"Erro inesperado Exerce: {e}")

```

```

        print(f"Erro inesperado Exerce: {e}")

def get_offset(tabela,cursor):
    cursor.execute(f"SELECT MAX(Id) FROM {tabela};")
    result = cursor.fetchone()
    return result[0] if result[0] else 0

def migraJson(cursor):
    with open('Dados/PortugalBD.json', 'r', encoding='utf-8') as f:
        dados = json.load(f)

    offset_funcao = get_offset("Funcao",cursor)
    offset_funcionario = get_offset("Funcionario",cursor)
    offset_cliente = get_offset("Cliente",cursor)
    offset_automovel = get_offset("Automovel",cursor)

    insereCliente(cursor, dados.get('Cliente', []))
    insereClienteContacto(cursor, dados.get('Cliente_Contacto',
    []),offset_cliente)
    insereAutomovel(cursor, dados.get('Automovel', []))
    insereFuncionario(cursor, dados.get('Funcionario', []))
    insereAluguer(cursor, dados.get('Aluguer',
    []),offset_cliente,offset_automovel,offset_funcionario)
    insereFuncao(cursor, dados.get('Funcao', []))
    insereExerce(cursor, dados.get('Exerce',
    []),offset_funcao,offset_funcionario)

def migraCsv(cursor):
    with open('Dados/BelgaBD.csv', newline='', encoding='utf-8') as ficheiro:
        leitor = csv.reader(ficheiro, delimiter=';')
        offset_funcao = get_offset("Funcao",cursor)
        offset_funcionario = get_offset("Funcionario",cursor)
        offset_cliente = get_offset("Cliente",cursor)
        offset_automovel = get_offset("Automovel",cursor)
        for linha in leitor:
            print(linha)

```

```

print(offset_funcao,offset_cliente,offset_funcionario,offset_automovel)

    try:
        tipo = linha[0]

        if tipo == 'Cliente':
            nome,     rua,     localidade,     codigoPostal,     nif,
localTrabalho = linha[1:]
            cursor.execute(
                "INSERT INTO Cliente (Nome, Rua, Localidade,
CodigoPostal, NIF, LocalTrabalho) VALUES (%s, %s, %s, %s, %s);",
                (nome,     rua,     localidade,     codigoPostal,     nif,
localTrabalho)
            )

        elif tipo == 'Cliente_Contacto':

            clienteId = int(linha[1])
            telefone = linha[2]
            email = linha[3] if linha[3] else None
            cursor.execute(
                "INSERT INTO Cliente_Contacto (ClienteId,
Telefone, Email) VALUES (%s, %s, %s);",
                (clienteId + offset_cliente, telefone, email)
            )

        elif tipo == 'Automovel':

            marca = linha[1]
            kilometragem = float(linha[2])
            ano = int(linha[3])
            estado = linha[4]
            tipoConsumo = linha[5]
            precoDia = Decimal(linha[6])
            filialId = int(linha[7])
            cursor.execute(
                "INSERT INTO Automovel (Marca, Kilometragem, Ano,
Estado, TipoConsumo, PrecoDia, FilialId) VALUES (%s, %s, %s, %s, %s, %s);",
                (marca,     kilometragem,     ano,     estado,     tipoConsumo,
precoDia,     filialId)
            )

```

```

        )

    elif tipo == 'Funcionario':

        nome = linha[1]
        nif = linha[2]
        salario = Decimal(linha[3])
        email = linha[4] if linha[4] else None
        telefone = linha[5]
        filialId = int(linha[6])
        cursor.execute(
            "INSERT INTO Funcionario (Nome, NIF, Salario,
Email, Telefone, FilialId) VALUES (%s, %s, %s, %s, %s, %s);",
            (nome, nif, salario, email, telefone, filialId)
        )

    elif tipo == 'Aluguer':

        dataInicio = linha[1] # assumindo formato correto
        dataFim = linha[2]
        preco = Decimal(linha[3])
        multa = Decimal(linha[4]) if linha[4] else None
        clienteId = int(linha[5])
        funcionarioId = int(linha[6])
        automovelId = int(linha[7])
        recolhidoFilialId = int(linha[8])
        devolvidoFilialId = int(linha[9])
        cursor.execute(
            "INSERT INTO Aluguer (DataInicio, DataFim, Preco,
Multas, ClienteId, FuncionarioId, AutomovelId, RecolhidoFilialId,
DevolvidoFilialId) VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s);",
            (dataInicio, dataFim, preco, multa, clienteId +
            offset_cliente, funcionarioId + offset_funcionario, automovelId +
            offset_automovel, recolhidoFilialId, devolvidoFilialId)
        )

    elif tipo == 'Funcao':


        designacao = linha[1]
        salarioBase = Decimal(linha[2])

```

```

        cursor.execute(
            "INSERT INTO Funcao (Designacao, SalarioBase)
VALUES (%s, %s);",
            (designacao, salarioBase)
        )

    elif tipo == 'Exerce':

        funcionarioId = int(linha[1])
        funcaoId = int(linha[2])
        cursor.execute(
            "INSERT INTO Exerce (FuncionarioId, FuncaoId)
VALUES (%s, %s);",
            (funcionarioId + offset_funcionario, funcaoId +
offset_funcao)
        )
    except IntegrityError as e:
        # Erro de entrada duplicada (código 1062 no MySQL)
        if e.errno == 1062:
            print(f"Entrada duplicada ignorada: {e}")
        else:
            print(f"Erro de integridade: {e}")
    except Exception as e:
        # Para outros erros
        print(f"Erro inesperado: {e}")

def migraPostgres(pgCursor, mysqlCursor):
    def safe_insert(query, data):
        try:
            mysqlCursor.execute(query, data)
        except mysql.connector.IntegrityError as e:
            if e.errno == mysql.connector.errorcode.ER_DUP_ENTRY:
                print(f"Entrada duplicada ignorada: {data}")
            else:
                raise e

```

offset\_funcao = get\_offset("Funcao",mysqlCursor)

```

offset_funcionario = get_offset("Funcionario",mysqlCursor)
offset_cliente = get_offset("Cliente",mysqlCursor)
offset_automovel = get_offset("Automovel",mysqlCursor)

# Funcao
insert_funcao = """
    INSERT INTO Funcao (Designacao, SalarioBase) VALUES (%s, %s)
"""

pgCursor.execute("SELECT Designacao, SalarioBase FROM Funcao;")
for row in pgCursor.fetchall():
    safe_insert(insert_funcao, row)

# Funcionario
insert_funcionario = """
    INSERT INTO Funcionario (Nome, NIF, Salario, Telefone, Email, FilialId)
        VALUES (%s, %s, %s, %s, %s, %s)
"""

pgCursor.execute("SELECT Nome, NIF, Salario, Telefone, Email, FilialId FROM
Funcionario;")
for row in pgCursor.fetchall():
    safe_insert(insert_funcionario, row)

# Exerce
insert_exerce = "INSERT INTO Exerce (FuncionarioId, FuncaoId) VALUES (%s,
%s)"
pgCursor.execute("SELECT FuncionarioId, FuncaoId FROM Exerce;")
for row in pgCursor.fetchall():
    row = list(row)
    row[0] += offset_funcionario
    row[1] += offset_funcao
    safe_insert(insert_exerce, row)

# Cliente
insert_cliente = """
    INSERT INTO Cliente (Nome, NIF, LocalTrabalho, Rua, Localidade,
CodigoPostal)
        VALUES (%s, %s, %s, %s, %s, %s)
"""

pgCursor.execute("SELECT Nome, NIF, LocalTrabalho, Rua, Localidade,
CodigoPostal FROM Cliente;")

```

```

for row in pgCursor.fetchall():
    safe_insert(insert_cliente, row)

# Cliente_Contacto
insert_cliente_contacto = """
    INSERT INTO Cliente_Contacto (ClienteId, Telefone, Email)
    VALUES (%s, %s, %s)
"""

pgCursor.execute("SELECT      ClienteId,      Telefone,      Email      FROM
Cliente_Contacto;")
for row in pgCursor.fetchall():
    row = list(row)
    row[0] += offset_cliente
    safe_insert(insert_cliente_contacto, row)

# Automovel
insert_automovel = """
    INSERT INTO Automovel (Marca, Kilometragem, Ano, Estado, TipoConsumo,
PrecoDia, FilialId)
    VALUES (%s, %s, %s, %s, %s, %s, %s)
"""

pgCursor.execute("SELECT Marca, Kilometragem, Ano, Estado, TipoConsumo,
PrecoDia, FilialId FROM Automovel;")
for row in pgCursor.fetchall():
    safe_insert(insert_automovel, row)

# Aluguer
insert_aluguer = """
    INSERT INTO Aluguer (DataInicio, DataFim, Preco, Multa, ClienteId,
FuncionarioId, AutomovelId, RecolhidoFilialId, DevolvidoFilialId)
    VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
"""

pgCursor.execute("""
    SELECT DataInicio, DataFim, Preco, Multa, ClienteId, FuncionarioId,
AutomovelId, RecolhidoFilialId, DevolvidoFilialId FROM Aluguer;
""")

for row in pgCursor.fetchall():
    row = list(row)

```

```

        row[4] += offset_cliente
        row[5] += offset_funcionario
        row[6] += offset_automovel
        safe_insert(insert_aluguer, row)

    print("Migração concluída com sucesso.")

def main():
    conn = mysql.connector.connect(
        host="localhost",
        user="root",
        password="root",
        database="BelaRentaCar"
    )

    conn_params = {
        'host': 'localhost',
        'port': 5432,
        'database': 'belarentacarmonaco',
        'user': 'root',
        'password': 'root'
    }

    pgconn = psycopg2.connect(**conn_params)

    pgCursor = pgconn.cursor()

    cursor = conn.cursor()

    migraCsv(cursor)
    migraJson(cursor)
    migraPostgres(pgCursor, cursor)
    conn.commit()

    pgCursor.execute("SELECT * FROM Automovel")
    resultados = pgCursor.fetchall()

```

```
for row in resultados:  
    print(row)  
  
cursor.close()  
conn.close()  
pgCursor.close()  
pgconn.close()  
  
  
if __name__ == "__main__":  
    main()
```