



Universidade do Minho
Licenciatura em Engenharia Informática

Unidade Curricular de Bases de Dados

Ano Lectivo de 2024/2025

Hole in One

Grupo 31

**Duarte Escairo, Luís Soares, João Pedro Rodrigues,
Tiago Figueiredo**

Abril, 2025

Data de Receção	
Responsável	
Avaliação	
Observações	

Hole in One

Grupo 31

Duarte Escairo, Luís Soares, João Pedro Rodrigues,
Tiago Figueiredo

Abril, 2025

Resumo

No âmbito da Unidade Curricular de Bases de Dados, foi proposto este trabalho prático que visa iniciar e rotinar os alunos no planeamento e execução de projetos de Sistemas de Bases de Dados (SBD), com especial interesse na análise, planeamento, modelação, arquitetura e implementação. Ao longo deste relatório será apresentado todo o desenvolvimento do SBD implementado para uma empresa de alugueres de carros de golfe fictícia, criada pelo grupo de trabalho.

O Sr. Orlando Esbelto, fundador da Hole in One, contratou um grupo de estudantes de engenharia informática da Universidade do Minho para desenvolver uma Base de Dados (BD) para a sua empresa, de forma a que pudesse gerir as várias stands que possuía espalhadas pelo país, bem como o seu pessoal e clientes.

O grupo começou por definir o contexto onde a BD se iria assentar, discutindo a ordem pela qual os trabalhos seriam executados. Após estar completa esta definição, prosseguiram para o levantamento de requisitos juntamente com o pessoal da empresa, sendo esses depois devidamente organizados, para poderem começar a desenhar a BD. Seguiu-se a construção de um modelo conceptual da BD que seria implementada, sendo criado um diagrama de Entidade-Relacionamento (ER) que foi posteriormente convertido no seu respetivo modelo lógico.

Numa fase seguinte, o grupo dedicou-se à implementação física da BD que foi desenhada, codificando tudo que era necessário na linguagem SQL, seguindo-se o seu respetivo povoamento, através de um programa desenvolvido em *python*.

Área de Aplicação: Desenho e Arquitetura de Sistemas de Bases de Dados

Palavras-Chave: Bases de Dados Relacionais, Diagramas de ER, Modelo Conceptual, Modelo Lógico, Álgebra Relacional, MySQL

Índice Geral

1. Definição do Sistema	1
1.1 Contexto de aplicação	1
1.2 Motivação e Objetivos do Trabalho	1
1.3 Análise da Viabilidade do processo	2
1.4 Recursos e Equipa de Trabalho	3
1.5 Plano de Execução do Projeto	3
2. Levantamento e Análise de Requisitos	5
2.1 Método de Levantamento e de Análise de Requisitos Adotado	5
2.2 Organização dos Requisitos Levantados	6
2.3 Análise e Validação dos Requisitos	8
3. Modelação Conceptual	10
3.1 Apresentação da Abordagem de Modelação Realizada	10
3.2 Identificação e Caracterização das Entidades	10
3.3 Identificação e Caracterização dos Relacionamentos	12
3.4 Identificação e Caracterização dos Atributos	13
3.5 Apresentação e Explicação do Diagrama ER Produzido	16
4. Modelação Lógica	18
4.1 Construção e Validação do Modelo de Dados Lógico	18
4.2 Apresentação e Explicação do Modelo Lógico Produzido	19
4.3 Normalização de Dados	20
4.4 Validação do Modelo com Interrogações do Utilizador	21
5. Implementação Física	26
5.1 Apresentação e explicação da base de dados implementada	26
5.2 Criação de utilizadores da base de dados	29
5.3 Povoamento da base de dados	31
5.4 Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)	34
5.5 Definição e caracterização de vistas de utilização em SQL	39
5.6 Tradução das interrogações do utilizador para SQL	40
5.7 Indexação do Sistema de Dados	43
5.8 Implementação de procedimentos, funções e gatilhos	44
6. Implementação do sistema de migração dados	55
Alterações relativas à Parte I	60
Conclusões e Trabalho Futuro	61
Parte I	61
Parte II	62
Referências	63
Lista de Siglas e Acrónimos	64

Anexos	65
I. Diagramas de Gantt do plano de execução	66
II. Tabelas de requisitos	68
III. Diagramas Conceptuais	72
IV. <i>Input</i> para o <i>RelaX</i>	75
V. Código do programa de migração	78

Índice de Figuras

Figura 1 - Logotipo da Empresa	1
Figura 2 - Diagrama de Gantt do plano de execução da primeira fase	4
Figura 3 - Diagrama de Gantt do plano de execução da segunda fase	4
Figura 4 - Cabeçalho da Ata da reunião geral	5
Figura 5 - Questionário aplicado ao Sr. Esbelto	6
Figura 6 - Representação da entidade cliente dentro do <i>brModelo</i>	14
Figura 7 - Início da construção do diagrama conceptual	17
Figura 8 - Diagrama conceptual final	17
Figura 9 - Modelo lógico	19
Figura 10 - Esquema da expressão relativa a RM4	21
Figura 11 - Esquema da expressão aplicada para o ID Cliente 101	22
Figura 12 - Esquema da expressão aplicada para o ID Funcionário 2	23
Figura 13 - Esquema da expressão aplicada para o ID Função 3	25

Índice de Tabelas

Tabela 1 - Excerto dos requisitos gerais	7
Tabela 2 - Excerto dos requisitos de descrição	7
Tabela 3 - Excerto dos requisitos de manipulação	8
Tabela 4 - Excerto dos requisitos de controlo	8
Tabela 5 - Entidades identificadas	12
Tabela 6 - Relacionamentos identificados	13
Tabela 7 - Caracterização dos atributos do Cliente	14
Tabela 8 - Caracterização dos atributos do Aluguer	15
Tabela 9 - Caracterização dos atributos do Funcionário	15
Tabela 10 - Caracterização dos atributos do Carro	16
Tabela 11 - Caracterização dos atributos da Função	16
Tabela 12 - Dependências funcionais entre os elementos das tabelas	20
Tabela 13 - Resultado da aplicação da primeira expressão	22
Tabela 14 - Resultado da aplicação da segunda expressão para o ID Cliente 101	23
Tabela 15 - Resultado da aplicação da segunda expressão para o ID Funcionário 2	24
Tabela 16 - Resultado da aplicação da segunda expressão para o ID Função 325	
Tabela 17 - Tamanho em bytes de cada tipo de dados	34
Tabela 18 - Espaço ocupado pelos atributos de cada registo da relação Cliente	36
Tabela 19 - Espaço ocupado pelos atributos de cada registo da relação Telefones	36
Tabela 20 - Espaço ocupado pelos atributos de cada registo da relação Carro	37
Tabela 21 - Espaço ocupado pelos atributos de cada registo da relação Funcao	37
Tabela 22 - Espaço ocupado pelos atributos de cada registo da relação Funcionario	37
Tabela 23 - Espaço ocupado pelos atributos de cada registo da relação Aluguer	38
Tabela 24 - Tamanho total da BD	38
Tabela 25 - Taxa de crescimento anual da BD	38

1. Definição do Sistema

1.1 Contexto de aplicação

A Hole in One é uma empresa de alugueres de carros de golfe, fundada em 2004 pelo Sr. Orlando Esbelto, após um trágico acidente que o deixou bastante debilitado, ficando desde então com um problema nos joelhos. Visto que jogar golfe era uma das coisas que mais gostava de fazer, decidiu fundar a Hole in One e abrir a sua primeira stand para poder continuar a ter o golfe presente na sua vida.

A empresa está sediada em Vilamoura, no Algarve, zona onde O Sr. Esbelto vive, e onde abriu a primeira stand, no campo de golfe *Old Course*. Nestes últimos anos, à medida que a popularidade do desporto veio a aumentar, o Sr. Esbelto decidiu expandir a empresa, contando agora com três stands, uma delas na Senhora da Hora, no Porto, no campo de golfe City Golf, e outra em Braga no campo de golfe Clube Golfe Braga. Conta também com seis funcionários, três gerentes e três secretárias.



Figura 1 - Logotipo da Empresa

1.2 Motivação e Objetivos do Trabalho

Conforme o aumento da popularidade da modalidade e a respetiva expansão da empresa, foi necessária uma reestruturação da Hole in One, pois o velho sistema em papel utilizado já não dava conta de atender às necessidades da empresa e dos clientes, visto que a procura de registos antigos, ou mesmo alguma eventual atualização de dados era sempre um processo desgastante e demorado, quando na realidade deveria ser o mais simples possível.

Para além disto, devido ao facto de que o perfil dos clientes foi mudando, e sendo o Sr. Esbelto uma pessoa que sempre gostou de estar em sintonia com o mundo atual, houve uma necessidade de mudar a cara das suas stands, dando-lhes um ar mais profissional e moderno.

Com isto decidiu investir num sistema de base de dados para poder informatizar e automatizar todos os processos que anteriormente eram feitos em papel.

Após a primeira reunião entre os membros do grupo de estudantes e o Sr. Esbelto, ficaram decididos alguns objetivos que deveriam ser cumpridos com esta implementação da BD para a empresa, sendo eles:

1. Facilitar a gestão e registo da informação, quer seja relativa aos clientes, mas também relativa à própria empresa, tal como a informação dos seus funcionários, dos carros de golfe e também os registos dos alugueres;
2. Capacidade de armazenamento estruturada de modo a tornar mais fácil os trabalhos dos funcionários, tornando-os mais eficientes;
3. Obter uma ferramenta útil para a gestão eficiente da empresa, quer seja dos seus recursos humanos, mas também dos recursos financeiros;
4. Tornar mais simples a gestão e contacto dos clientes, para evitar confusões.

1.3 Análise da Viabilidade do processo

A empresa de alugueres, tendo o seu funcionamento baseado em livros de registos e papel, apresenta problemas a nível de eficiência e produtividade, motivos apontados pelo próprio fundador. Desta forma o investimento no SBD é essencial para o futuro da Hole in One. O grupo de estudantes chegou à conclusão que a implementação da BD conseguirá:

- Proporcionar uma maneira mais eficiente de organizar a informação, já que estará armazenada num só local, e não dependerá de papel e livros de registos
- Gerir de forma otimizada os recursos da empresa, visto que haverá uma perceção mais geral do estado da empresa, devido à informação estar mais condensada, o que não era possível com o antigo sistema
- Fornecer um serviço de alta qualidade aos clientes, aumentando a eficiência e rapidez dos registos dos alugueres em 40%

O Sr. Esbelto concluiu assim que o investimento no SBD era completamente viável e justificado. Para além disso, o grupo responsável pela contabilidade da empresa, informou o Sr. Esbelto que, com o aumento das previsões do turismo em Portugal, e com uma maior popularidade do golfe, está previsto só no próximo ano um aumento de 22% nos lucros da empresa, o que garante que se consegue pagar o investimento em cerca de três anos.

1.4 Recursos e Equipa de Trabalho

Para a implementação da BD são necessários recursos, tanto humanos, divididos em duas categorias, o pessoal interno (empresa) e externo (grupo de estudantes); bem como recursos materiais. Relativamente aos recursos humanos temos:

Membros do grupo de estudantes:

- Duarte Escairo
- João Pedro Rodrigues
- Luís Soares
- Tiago Figueiredo

Estes foram responsáveis pela implementação do SBD, bem como todos os trabalhos que dele derivam, como por exemplo, o levantamento de requisitos, as modelações conceptual e lógica etc. Apesar de todo o grupo acompanhar todo o trabalho desenvolvido, o mesmo foi dividido e atribuído a cada um dos membros, tendo as várias etapas sido desenvolvidas por um membro responsável, como se poderá ver no Diagrama de Gantt mais à frente. Quanto ao pessoal da empresa que se envolveu no processo de implementação da BD temos:

- Orlando Esbelto: Fundador da Hole in One
- José Calado: Gerente – Stande Vilamoura
- Rogério Samora: Gerente – Stand Porto
- Edgar Novo: Gerente – Stand Braga

O Sr. Esbelto acompanhou de perto todo o processo da implementação até estar concluído, participou em todas as reuniões que foram realizadas no âmbito do projeto, validou o trabalho feito pelo grupo de estudantes conforme ia sendo feito, e para além disso, também lhe foi feito um questionário durante o levantamento dos requisitos.

Os gerentes de cada uma das stands da empresa, participaram ativamente na reunião feita para o levantamento de requisitos, e validaram os mesmos.

Quanto aos recursos materiais, foi necessário *software* como o *MySQL Workbench* para a implementação física da BD, e também para o desenho do modelo lógico. O *brModelo* foi utilizado para o desenvolvimento do modelo conceptual. Foi também necessário *hardware*, pois cada membro do grupo de estudantes possui o seu computador.

1.5 Plano de Execução do Projeto

Depois da primeira reunião entre os membros do grupo de estudantes e o Sr. Esbelto, vários aspetos ficaram estabelecidos, sendo um deles o plano de execução do projeto, do qual resultaram dois Diagramas de Gantt, onde estão definidas as diversas tarefas para cada uma

das fases do trabalho. Ficou acordado que o projeto seria dividido em duas fases, a primeira com data limite de entrega a 21 de abril de 2025, e a segunda com data de entrega limite a 2 de junho de 2025.

A divisão das tarefas foi pensada de forma a que cada uma pudesse ser feita com tempo e de forma correta, tendo sido também considerado algum espaço de folga, caso houvesse qualquer atraso inesperado.

O grupo de estudantes decidiu dedicar ao projeto cerca de sete horas semanais (em média uma hora por dia), podendo aumentar ou diminuir este período conforme a sua disponibilidade, devido às atividades letivas. Os diagramas não consideram por exemplo momentos de avaliação (dias onde os estudantes não se dedicariam ao projeto), pelo simples facto dos membros do grupo, frequentarem anos letivos diferentes, ou seja, em determinados dias alguns membros estariam indisponíveis, enquanto que outros não.

No diagrama relativo à primeira fase foi incluído o tempo gasto na realização da definição do sistema. Os restantes pontos representam os prazos previstos e acordados com o Sr. Esbelto.

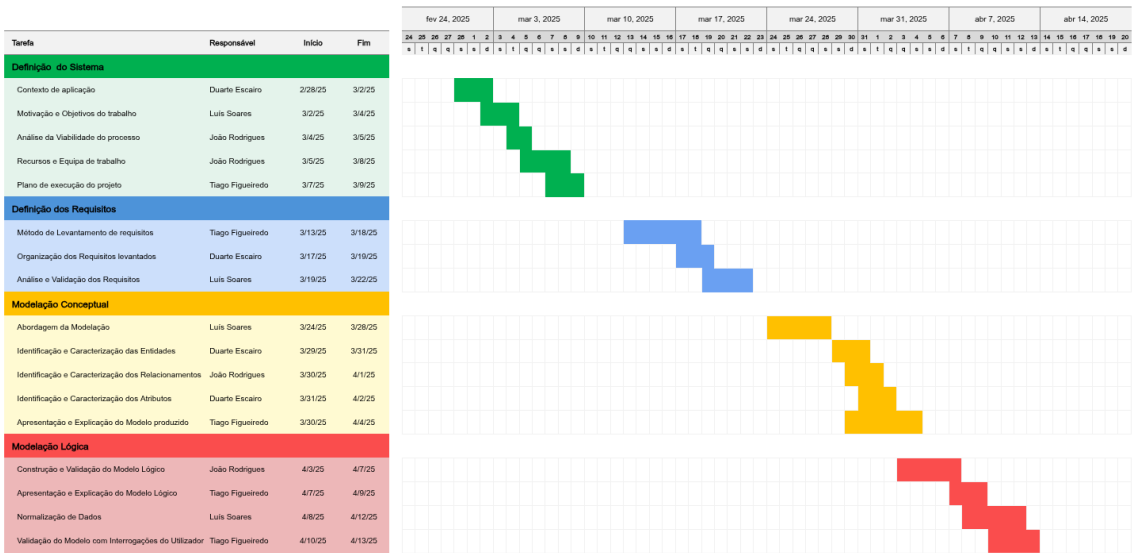


Figura 2 - Diagrama de Gantt do plano de execução da primeira fase

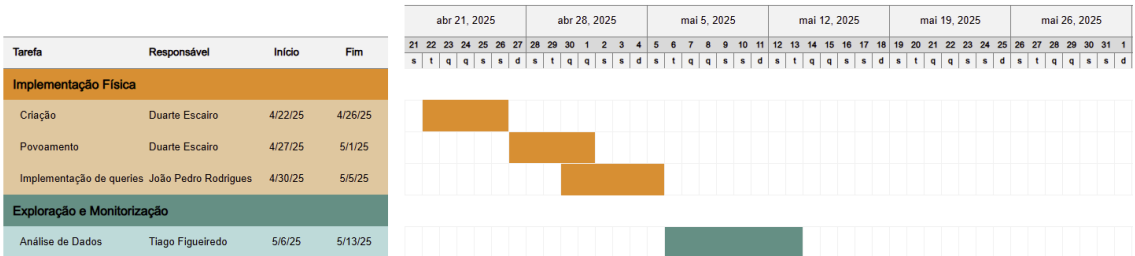


Figura 3 - Diagrama de Gantt do plano de execução da segunda fase

Os diagramas encontram-se disponíveis no [Anexo I](#).

2. Levantamento e Análise de Requisitos

2.1 Método de Levantamento e de Análise de Requisitos Adotado

Após a conclusão da primeira etapa do desenvolvimento do SBD, a definição do sistema, o grupo de estudantes prosseguiu com o trabalho, fazendo o levantamento de requisitos. Para esta nova etapa diversos métodos foram utilizados, foram feitas reuniões, por videoconferência para evitar gastos desnecessários a qualquer um dos envolvidos, uma geral, com todos os gerentes das stands da empresa e o seu fundador, e uma com a apenas a participação do Sr. Esbelto. Nesta última foi aplicado um questionário, ao próprio, para se perceber a forma como o fundador da empresa previa o impacto do investimento no dia a dia da empresa, bem como as expectativas relativas à BD. O grupo de estudantes teve também oportunidade de analisar diversos documentos da empresa disponibilizados pelo Sr. Esbelto.

Reunião geral da empresa:

Na reunião geral da empresa procurou-se perceber quais eram as tarefas levadas a cabo no dia a dia, que poderiam ser melhoradas com a implementação de uma BD. Todos os gerentes contribuíram na reunião, explicando todos os processos que eram realizados quando se tratava de um aluguer de um carro golfe.

Na reunião o grupo esteve atento também à hierarquia existente dentro da empresa para depois conseguir limitar os acessos à informação conforme fosse necessário.


ATA DE REUNIÃO – LEVANTAMENTO DE REQUISITOS		
RESPONSÁVEL: TIAGO FIGUEIREDO		
LOCAL: VIDEOCONFERÊNCIA ZOOM		
DATA E HORA: 14/3/2025, 10h30		
PARTICIPANTES:		
ORLANDO ESBELTO JOSÉ CALADO ROGÉRIO SAMORA EDGAR NOVO		


Figura 4 - Cabeçalho da Ata da reunião geral

Reunião com o Sr. Esbelto e Inquérito

Na reunião particular com o fundador da empresa o grupo de estudantes pretendeu perceber o funcionamento geral da empresa, bem como realçar alguma informação da reunião geral que o Sr. Esbelto considerasse particularmente relevante.

Quanto ao questionário aplicado, o mesmo pode ser visto abaixo.

QUESTIONÁRIO PARA LEVANTAMENTO DE REQUISITOS



1. Como prevê impacto no dia a dia, do investimento no Sistema de Base de Dados?

2. Explique, detalhadamente, as tarefas que executa no seu dia a dia de trabalho.

3. Quais são as áreas dentro da empresa que considera mais ineficientes?

4. Quais são as expectativas gerais da implementação da Base de Dados?

Figura 5 - Questionário aplicado ao Sr. Esbelto

Análise de Documentos

Os documentos analisados pelo grupo de estudantes tiveram um papel fundamental neste levantamento de requisitos pois foi com estes que se percebeu a informação que era necessária para se fazer um aluguer, bem como a quantidade de informação que a empresa possuía relativamente aos seus clientes.

Relativamente a outros métodos de levantamento de requisitos, como a análise do estado da arte, desde logo se percebeu que não seriam alternativas viáveis, devido a um acréscimo desnecessário ao investimento originalmente previsto.

2.2 Organização dos Requisitos Levantados

Durante as reuniões feitas, o grupo de estudantes, recolheu informação relativa aos processos que eram realizados no dia a dia da empresa, questionando os gerentes e o fundador sobre os mesmos. Cada um explicou em detalhe o funcionamento da stand onde trabalhavam, e tal como era de esperar, o dia a dia de trabalho de cada um não era muito diferente, o que demonstrou uma consistência na empresa, e revelou ser um aspeto positivo, já que a implementação da BD seria feita de forma a corresponder às necessidades da empresa e não às necessidades específicas de cada stand.

Toda a informação recolhida ao longo do levantamento de requisitos foi organizada numa tabela de requisitos geral, como a que se pode ver abaixo.

Nr	Descrição	Data	Fonte	Analista
1	O cliente é caracterizado por ter um identificador único, nome, data de nascimento, telefone, email, morada.	17/03/2025	Reunião Geral	Luís Soares
2	O funcionário é caracterizado por ter um identificador único, nome, telefone, email da empresa, morada, local de trabalho, salário.	17/03/2025	Reunião Geral	Duarte Escairo
3	Um aluguer é caracterizado por ter um número de aluguer único, data de início e fim do aluguer.	17/03/2025	Reunião Geral	Luís Soares
4	O carro é caracterizado pelo número único do veículo, stand onde se encontra, ano de fabrico, número de lugares e custo diário do aluguer.	17/03/2025	Reunião Geral	João Pedro Rodrigues

Tabela 1 - Excerto dos requisitos gerais

Após esta recolha geral, era necessário dividir todos os requisitos em três categorias, requisitos de descrição que se referem a requisitos que providenciam informação a cerca de entidades, relacionamentos, atributos, restrições etc., requisitos de manipulação, que se referem a requisitos que definem as diversas operações que são possíveis dentro da base de dados implementada, e por fim requisitos de controlo, que se referem a requisitos que indicam as permissões e acessos dos vários utilizadores envolvidos, a determinados campos da BD. Essa divisão deu origem às três tabelas que se seguem.

Nr	Descrição	Data	Fonte	Analista
RD1	O cliente é caracterizado por ter um identificador único, nome, data de nascimento, telefone, email, morada.	17/03/2025	Reunião Geral	Luís Soares
RD2	O funcionário é caracterizado por ter um identificador único, nome, telefone, email da empresa, morada, local de trabalho, salário.	17/03/2025	Reunião Geral	Duarte Escairo
RD3	Um aluguer é caracterizado por ter um número de aluguer único, data de início e fim do aluguer.	17/03/2025	Reunião Geral	Luís Soares
RD4	O carro é caracterizado pelo número único do veículo, stand onde se encontra, ano de fabrico, número de lugares e custo diário do aluguer.	17/03/2025	Reunião Geral	João Pedro Rodrigues

Tabela 2 - Excerto dos requisitos de descrição

Nr	Descrição	Data	Fonte	Analista
RM1	Os dados de um cliente devem poder ser acedidos através do identificador do cliente.	18/03/2025	Reunião com o Sr. Esbelto	Luís Soares
RM2	Os dados de um funcionário devem poder ser acedidos através do identificador do funcionário.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RM3	Os dados de um aluguer devem poder ser acedidos através do número de aluguer.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RM4	É possível listar todos identificadores de cliente, e o seu respetivo nome.	17/03/2025	Reunião Geral	Duarte Escairo

Tabela 3 - Excerto dos requisitos de manipulação

Nr	Descrição	Data	Fonte	Analista
RC1	O fundador da empresa tem permissão para acrescentar, modificar, apagar e ler todos os registos do sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC2	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos clientes no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC3	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos alugueres no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC4	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos carros no sistema.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues

Tabela 4 - Excerto dos requisitos de controlo

As tabelas podem ser vistas na íntegra no [Anexo II](#).

Após toda a organização dos requisitos levantados, foi possível começar a perceber quais os pontos chave que seriam utilizados na construção dos modelos conceptual (MC) e lógico (ML), contudo era necessário primeiro validar toda a informação que havia sido recolhida.

2.3 Análise e Validação dos Requisitos

Após toda a recolha e organização dos requisitos levantados, o grupo de estudantes voltou a reunir com todos os gerentes e fundador da empresa, para confirmar toda a informação recolhida de modo a garantir que nenhum erro tinha sido cometido.

Dessa reunião, bastante participativa por parte de todos os intervenientes, foi possível corrigir alguns erros, tal como qual o atributo pelo qual se tem acesso aos dados de um cliente (inicialmente foi colocado o nome, quando na verdade deveria ser o identificador do cliente), devidos à má comunicação num primeiro momento (provavelmente devido a algumas falhas na rede durante a videoconferência), sendo também acrescentada informação em certos pontos

que os membros da empresa consideraram pertinentes, como por exemplo o atributo data de nascimento, nos clientes e o atributo número de lugares, nos carros de golfe.

Após todos os presentes validarem o trabalho que fora realizado, o grupo de estudantes tinha agora uma base sólida onde se apoiar para o restante desenvolvimento do processo.

3. Modelação Conceptual

3.1 Apresentação da Abordagem de Modelação Realizada

Depois de todas as reuniões feitas, e de todos os requisitos serem levantados e devidamente categorizados, o grupo de estudantes começou o desenvolvimento do modelo conceptual.

Primeiramente foram identificadas as entidades presentes nos requisitos de descrição, e foi definido para cada uma delas uma breve descrição, sinónimos (caso fosse possível), a sua ocorrência e o requisito que nos deu a informação sobre a mesma. Depois foram identificados os relacionamentos entre as entidades e foram definidas a cardinalidade e participação das mesmas nos relacionamentos. Identificamos novamente qual o requisito que nos deu a informação. Por fim caracterizamos os atributos de cada uma das entidades, e definimos para cada um uma breve descrição, o seu tipo, e se eram NULL, atributos multivalorados, e se eram a chave primária que identificava a entidade. Mais uma vez foram identificados os requisitos associados a informação.

Depois começamos o desenvolvimento do diagrama ER, e para isso utilizamos a ferramenta *brModelo* para a construção do diagrama conceptual, que utiliza a notação de *Chen* já conhecida pelo grupo. Na construção do diagrama baseamo-nos em toda informação previamente recolhida, principalmente nas entidades, relacionamentos e atributos identificados.

Importante salientar que dada a reduzida dimensão do projeto a desenvolver, o grupo de estudantes teve em consideração apenas uma vista geral de utilização, já que não seria necessário a divisão em várias vistas de utilização.

3.2 Identificação e Caracterização das Entidades

Para se poder desenhar os diagramas ER, no processo de desenvolvimento de um SBD, é crucial identificar e definir as entidades que os irão compor, dito isto, o grupo de estudantes, com todo o trabalho já previamente estabelecido, conseguiu identificar diversas entidades através dos requisitos de descrição levantados. Desta forma, as entidades utilizadas neste projeto foram as seguintes:

- Cliente – representa os indivíduos que permitem a continuidade das operações da empresa. Decidiu-se que estes seriam uma entidade, tal como descrito no RD1, devido ao seu papel fundamental para a existência da mesma. Assim, é possível associar uma pessoa que utiliza os serviços a uma entidade na base de dados.
- Aluguer – caracteriza o único tipo de serviço disponibilizado pela empresa. Já que o funcionamento dela exige a catalogação de todos os serviços realizados, foi necessário criar uma entidade que permitisse a consulta da atividade realizada, tal como é possível ver no RD3.
- Funcionário – corresponde à entidade que acolhe todas as características dos funcionários da empresa. Tendo em conta o seu papel fundamental para as operações de aluguer que a empresa disponibiliza, por uma questão de se conseguir melhorar a administração de pessoal e organização dos papéis que cada funcionário desempenha, foi necessário criar esta entidade com todas as suas particularidades descritas no RD2.
- Carro – devido ao facto de que o número de stands veio a aumentar, tornou-se essencial criar uma entidade que represente os veículos disponíveis para aluguer. Esta entidade permite gerir informações detalhadas sobre cada carro, como o número que o identifica, o seu número de lugares e custo diário de aluguer, garantindo assim um controlo eficiente da frota da empresa, conforme estabelecido no RD4.
- Função – cada funcionário desempenha um papel específico dentro da empresa, sendo necessário definir uma entidade que represente as diferentes funções existentes. Esta entidade permite associar cada funcionário à sua respetiva função, facilitando a gestão interna e a alocação de responsabilidades, conforme descrito no RD9.

Um resumo de toda a informação relativa às entidades identificadas encontra-se registada na tabela abaixo.

Entidade	Descrição	Sinónimos	Ocorrência	Requisito
Cliente	Pessoa que recorre aos serviços de aluguer prestados pela empresa.	...	Um cliente precisa de estar registado na empresa para poder fazer um aluguer de um carro.	RD1
Aluguer	Representa um registo feito, depois de um cliente requisitar os serviços da empresa.	...	Cada aluguer tem o número de identificação único associado no momento do registo.	RD3
Funcionário	Pessoa que presta diversos serviços dentro da empresa.	Empregado	Um funcionário exerce uma função mediante o seu estatuto dentro da empresa.	RD2
Carro	Produto usado nos serviços prestados pela empresa.	Veículo	Cada aluguer feito pela empresa apenas tem associado um carro.	RD4
Função	Representa um dos trabalhos realizados pelos funcionários.	Cargo	A cada funcionário é atribuída uma única função.	RD9

Tabela 5 - Entidades identificadas

3.3 Identificação e Caracterização dos Relacionamentos

Após estarem definidas as entidades a serem usadas no diagrama e na BD, prosseguiu-se para a identificação dos relacionamentos estabelecidos entre as mesmas. Mais uma vez a informação para esta identificação proveio dos requisitos de descrição anteriormente levantados. Assim sendo, os relacionamentos foram os seguintes:

- Cliente-Aluguer – A entidade Cliente está relacionada com a entidade Aluguer na medida em que: um cliente só é considerado um cliente quando realiza um aluguer na empresa e um aluguer só existe se houver um cliente que o faça. Para além disso, um cliente pode realizar diversos alugueres, contudo considera-se que cada cliente só pode fazer um aluguer de cada vez.
- Aluguer-Carro – A entidade Aluguer está relacionada com a entidade Carro, já que: quando um aluguer é realizado, este é obrigatoriamente associado a um único carro (ou veículo) mas este carro pode estar associado a outros alugueres. Por outro lado, um carro pode não se encontrar a ser alugado em um dado momento.
- Aluguer-Funcionário – A entidade Aluguer está relacionada com a entidade Funcionário, visto que: um funcionário pode ser responsável por tratar de 1 ou mais

alugueres, mas cada aluguer só pode ser tratado por um funcionário. Ainda assim, para um aluguer ser registado, o mesmo tem de ser, obrigatoriamente, tratado por um funcionário e um funcionário pode ou não estar a tratar de um aluguer.

- **Funcionário-Função** – A entidade Funcionário está relacionada com a entidade Função, visto que: um funcionário tem, obrigatoriamente, de exercer uma, e apenas uma, função. No entanto, é possível que uma certa função não esteja a ser exercida por nenhum funcionário. Além disso, uma função pode ser exercida por vários funcionários diferentes.

Finalizada toda esta análise e identificação, foi possível concluir quais os valores a serem atribuídos à cardinalidade e a participação das entidades, para cada um dos relacionamentos. Toda a informação respetiva dos mesmo encontra-se registada na tabela abaixo.

Entidade	Relacionamento	Cardinalidade	Participação	Entidade	Requisito
Cliente	Faz	1:N	T:T	Aluguer	RD5
Aluguer	De Um	N:1	T:P	Carro	RD6
Aluguer	Tratado	N:1	T:P	Funcionário	RD7
Funcionário	Exerce	N:1	T:P	Função	RD8

Tabela 6 - Relacionamentos identificados

3.4 Identificação e Caracterização dos Atributos

Uma vez realizada a devida identificação e caracterização das entidades e relacionamentos, foi momento de caracterizar todos os atributos associados aos mesmos (no nosso caso em particular apenas das entidades). A partir deste ponto coube ao grupo de estudantes definir para cada atributo o seu tipo (INT, DATE, VARCHAR, DECIMAL (7,2) etc.), se o atributo se pode encontrar com valor nulo (NULL), se se trata de um atributo multivalorado, e ainda se é chave primária. Com todos estes elementos definidos, foram construídas tabelas com toda a informação dos atributos relativos a cada entidade.

Novamente, através dos requisitos foi possível retirar a informação necessária para esta caracterização.

Vejamos por exemplo o requisito de descrição número 1 (RD1), através dele percebemos que um Cliente fica completamente definido se a ele lhe forem associados um

identificador único, um nome, uma data de nascimento, uma lista de telefones, um email e uma morada. (Através do RD9 fica claro que todas as moradas referidas nesta secção do relatório, implicam uma composição com uma Rua e uma Cidade).

Ficou claro para o grupo de estudantes que o identificador do Cliente, bem como a lista de telefones a ele associado, seriam representados como variáveis do tipo inteiro (INT), a data de nascimento como uma data (DATE) e os restantes atributos foram classificados como conjuntos de caracteres (VARCHAR ()), já que representavam nomes de uma forma geral. O único atributo que se poderá encontrar com o valor NULL é a data de nascimento, já que se considerou não ser um atributo essencial para a descrição da entidade.

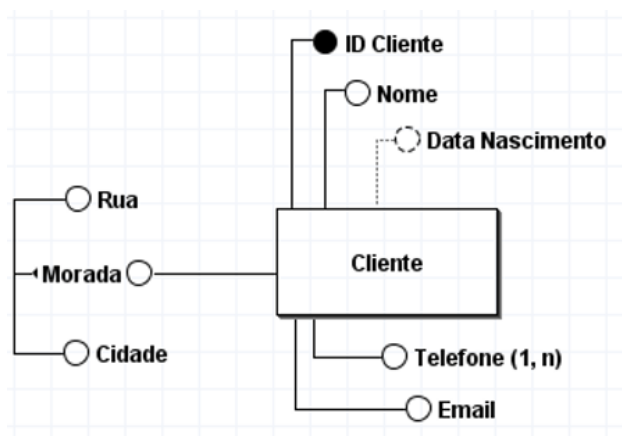


Figura 6 - Representação da entidade cliente dentro do *brModelo*

Entidade: Cliente

Atributos	Descrição	Tipo	NULL	Multivalorado	Chave Primária	Exemplo	Requisito
ID Cliente	Identificador único do cliente	INT	N	N	S	84	RD1
Nome	Nome do cliente	VARCHAR(100)	N	N	N	Tomás Pimenta	RD1
Data Nascimento	Data de nascimento do cliente	DATE	S	N	N	26/11/2005	RD1
Telefone	Lista de números de telefone	INT	N	S	N	[912345678, 926769123]	RD1
Email	Email do cliente	VARCHAR(50)	N	N	N	tomas.pimenta@yahoo.com	RD1
Morada (Rua, Cidade)	Morada do cliente	VARCHAR(100); VARCHAR(50)	N	N	N	Rua dos Pedreiros, Vila Verde	RD1

S/N - SIM/NÃO

Tabela 7 - Caracterização dos atributos do Cliente

Relativamente aos alugueres, percebeu-se através do requisito de descrição 3 (RD3), que estes ficam caracterizados, se tiverem um número de aluguer único associado, e as datas e início e fim do aluguer.

Neste caso, para representar o número do aluguer, escolheu-se uma variável do tipo inteiro (INT) e para as datas foi o tipo DATE. Considerou-se que nenhum dos atributos era

irrelevante para a caracterização da entidade, e por isso decidiu-se que nenhum poderia estar com valor nulo.

Entidade: Aluguer

Atributos	Descrição	Tipo	NULL	Multivalorado	Chave Primária	Exemplo	Requisito
Número de Aluguer	Número único que identifica o registo do aluguer	INT	N	N	S	347	RD3
Data de Início	Data de início do aluguer	DATE	N	N	N	18/10/2009	RD3
Data de Fim	Data de fim do aluguer	DATE	N	N	N	20/10/2009	RD3

S/N - SIM/NÃO

Tabela 8 - Caracterização dos atributos do Aluguer

Para a entidade Funcionário, a partir do requisito de descrição 2 (RD2), deduzimos que o mesmo fica bem definido se tiver um indentificador único, um nome, um telefone, um email da empresa, o local onde trabalha, a sua morada e o seu salário.

Mais uma vez, para o indentificador, considerou-se o seu tipo como um inteiro (INT), tal como para o telefone. O salário foi definido como um valor do tipo decimal. Para os restantes atributos, que eram na generalidade nomes, atribuiu-se o tipo VARCHAR (). Nenhum dos atributos do Funcionário pode estar com valor nulo pois considerou-se que os mesmos eram bastante importantes na definição da entidade.

Entidade: Funcionário

Atributos	Descrição	Tipo	NULL	Multivalorado	Chave Primária	Exemplo	Requisito
ID Funcionário	Identificador único do funcionário	INT	N	N	S	2	RD2
Nome	Nome do funcionário	VARCHAR(100)	N	N	N	Beatriz Pires	RD2
Telefone	Telefone do funcionário	INT	N	N	N	973561990	RD2
Email	Email da empresa do funcionário	VARCHAR(50)	N	N	N	beatriz2@hole.pt	RD2
Local Trabalho	Stand onde o funcionário trabalha	VARCHAR(20)	N	N	N	Vilamoura	RD2
Morada (Rua, Cidade)	Morada do funcionário	VARCHAR(100) VARCHAR(50)	N	N	N	Rua dos Bares, Albufeira	RD2
Salário	Salário do funcionário	DECIMAL(7,2)	N	N	N	1500,51	RD2

S/N - SIM/NÃO

Tabela 9 - Caracterização dos atributos do Funcionário

Para o Carro percebemos através de RD4, que o mesmo ficava completamente caracterizado se tivesse um número único que o identificasse, a stand à qual o carro pertencia, o ano em que foi produzido, o seu número de lugares e o custo diário do seu aluguer.

Com a exceção da stand onde pertence o carro que foi considerada como um VARCHAR () por se tratar de um nome, e do custo diário de aluguer que foi definido como um valor do tipo decimal, todos os atributos foram classificados como sendo variáveis do tipo

inteiro. Apenas se considerou o ano de produção para ter o valor nulo, dada a menor importância do atributo para a entidade.

Entidade: Carro

Atributos	Descrição	Tipo	NULL	Multivalorado	Chave Primária	Exemplo	Requisito
Número do Carro	Número único que identifica o carro	INT	N	N	S	14	RD4
Stand	Stand onde o carro pertence	VARCHAR(20)	N	N	N	Vilamoura	RD4
Ano	Ano de fabrico do carro	INT	S	N	N	2017	RD4
Número de Lugares	Número de lugares do carro	INT	N	N	N	4	RD4
Custo	Custo diário do aluguer do carro	DECIMAL(4,2)	N	N	N	30,25	RD4

S/N - SIM/NÃO

Tabela 10 - Caracterização dos atributos do Carro

Por fim, para a entidade Função, a partir do nono requisito de descrição (RD9), percebemos que esta ficava definida com um identificador da função e a sua designação.

Para o identificador utilizamos um inteiro (INT) para classificar o atributo, e um VARCHAR () para classificar a designação. Consideramos que nenhum dos dois se poderia encontrar com valor nulo.

Entidade: Função

Atributos	Descrição	Tipo	NULL	Multivalorado	Chave Primária	Exemplo	Requisito
ID Função	Identificador único da função	INT	N	N	S	1	RD9
Designação	Nome da função	VARCHAR(20)	N	N	N	Secretária	RD9

S/N - SIM/NÃO

Tabela 11 - Caracterização dos atributos da Função

3.5 Apresentação e Explicação do Diagrama ER

Produzido

Após a identificação e caracterização de todas as entidades, relacionamentos e atributos, o grupo de estudantes partiu para o desenvolvimento do diagrama ER. Para tal, foi utilizada a ferramenta de *software brModelo* que permite a construção de diagramas conceptuais utilizando a notação de *Chen*.

Num primeiro momento foram construídas todas as entidades identificadas, bem como os seus respetivos atributos associados, tendo sido considerado se os atributos eram a chave primária de uma determinada entidade, se representavam um atributo composto ou multivalorado, ou se apenas se tratava de um atributo simples. Após este primeiro momento o diagrama construído apresentava este aspeto:

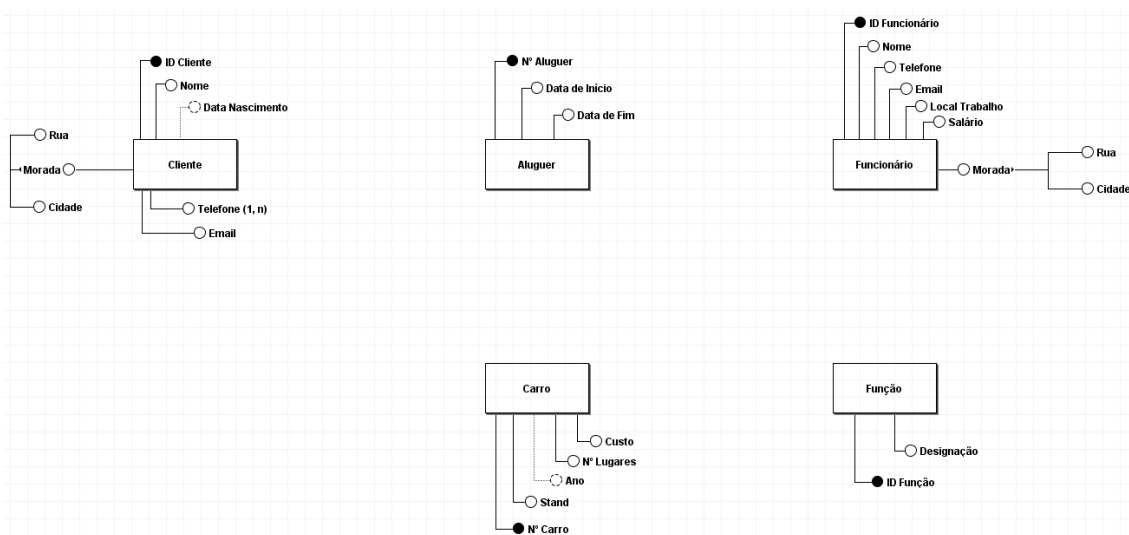


Figura 7 - Início da construção do diagrama conceitual

Num segundo momento restava apenas adicionar ao diagrama inicialmente produzido, os relacionamentos que se identificaram entre as várias entidades. Nesta parte da construção foram também definidos os valores da cardinalidade e participação nos relacionamentos. No final o diagrama construído foi este:

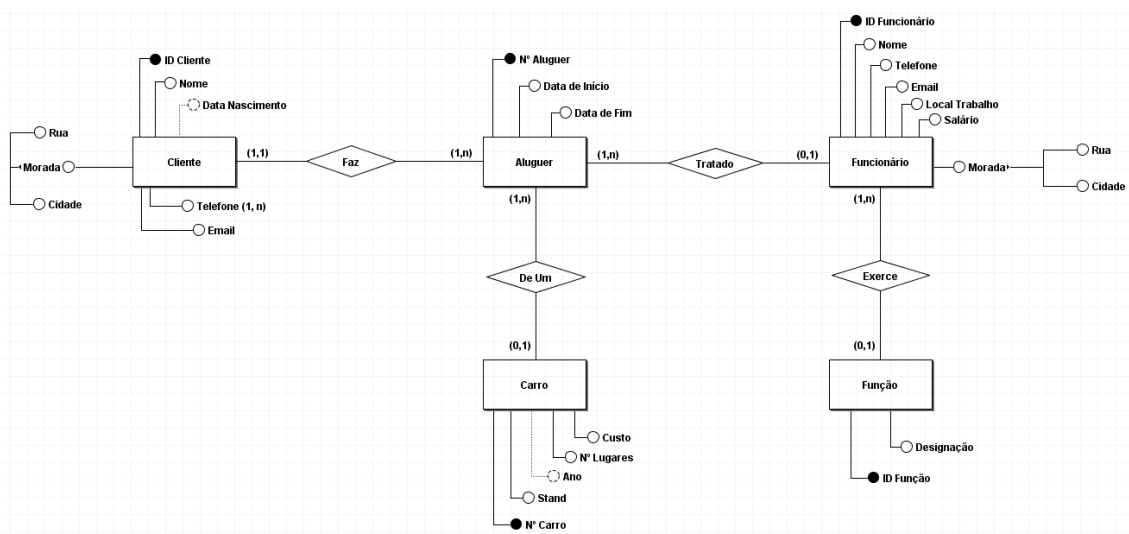


Figura 8 - Diagrama conceitual final

Os diagramas podem ambos ser vistos de uma forma mais clara no [Anexo III](#).

4. Modelação Lógica

4.1 Construção e Validação do Modelo de Dados Lógico

Após concluído todo o processo relativo à modelação conceptual, o grupo de estudantes partiu para o desenvolvimento da modelação lógica da BD a ser implementada. Começamos por derivar o modelo conceptual anteriormente feito, e dessa derivação resultaram as tabelas presentes no modelo lógico que será apresentado.

A tabela Cliente advém da entidade Cliente já previamente identificada, e por sua vez, os atributos nela presentes, também apresentam a mesma origem. A tabela Telefones representa o atributo multivalorado Telefone, associado à entidade cliente. Desta ligação existente entre o cliente e os seus telefones, resulta uma chave estrangeira na tabela Telefones que liga um determinado número a um determinado cliente.

A tabela Carro resulta da também de uma entidade já identificada, neste caso o Carro. Os atributos que a constituem também são os atributos já anteriormente descritos.

A tabela Funcionário provém mais uma vez das entidades descritas em pontos anteriores neste relatório, tal como os seus atributos associados. Esta tabela apresenta uma chave estrangeira que identifica a função que um determinado funcionário exerce, e é justificada através do relacionamento existente entre as entidades Funcionário e Função.

A tabela Função e os atributos nela presentes advêm novamente da entidade Função já previamente identificada.

Por fim, a tabela Aluguer resulta da entidade Aluguer, assim como os seus atributos. Esta tabela apresenta três chaves estrangeiras associadas, que existem devido a três relacionamentos onde esta entidade está incluída, sendo eles Cliente-Aluguer, Aluguer-Funcionário e Aluguer-Carro.

O modelo lógico foi construído com auxílio da ferramenta *MySQL Workbench*.

Para garantir uma boa qualidade do modelo, o grupo aplicou metodologias da normalização de dados (Formas Normais), tendo em vista uma BD íntegra, consistente e com redundância controlada. Apenas foram aplicadas as três primeiras formas normais.

4.2 Apresentação e Explicação do Modelo Lógico Produzido

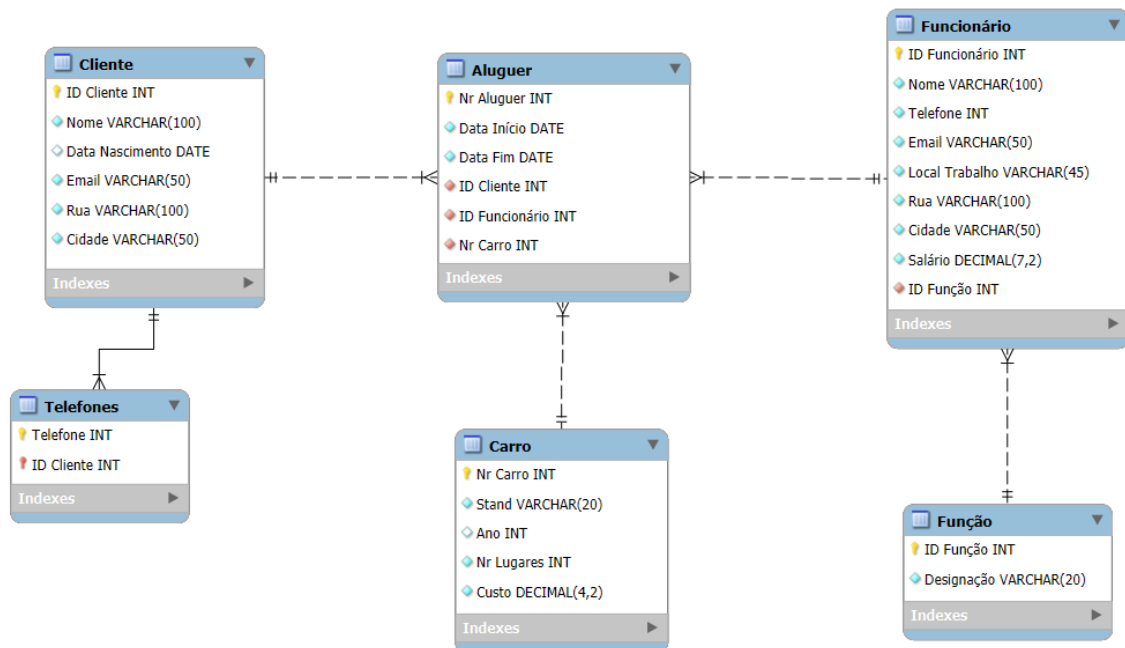


Figura 9 - Modelo lógico

O modelo lógico acima apresentado tem seis tabelas, cinco delas advêm das entidades anteriormente identificadas (Cliente, Aluguer, Funcionário, Carro, Função), a tabela Telefones foi construída para representar o atributo multivalorado Telefone associado a um cliente.

Através da informação organizada no momento da modelação conceptual, bem como das tabelas construídas, foi possível atribuir a cada uma das entidades presentes nas tabelas do modelo, os seus respetivos atributos.

Para a tabela Cliente, o atributo ID Cliente foi o escolhido para chave primário, já que se trata de um identificador único. Todos os atributos, à exceção da Data Nascimento não podem ter valor nulo, estando isso perceptível na figura através da cor do diamante que antecede os atributos (os diamantes a azul não podem ter valor nulo, os diamantes a branco podem ter valor nulo). A tabela Telefones apresenta uma chave primária composta, com os atributos Telefone e ID Cliente, que identificam o número de telefone em questão, e o cliente a que o número de telefone está associado, respetivamente. O ID Cliente é uma chave estrangeira já que referencia um cliente da tabela Cliente.

Na tabela Carro, o número do carro (Nr Carro) foi o atributo escolhido para chave primária, por se tratar de um número único. Todos os atributos à exceção do ano de fabrico (Ano) não podem ter valor nulo.

Para a tabela Função, o identificador da função (ID Função) foi escolhido para chave primária. Ambos os atributos da tabela (ID Função e Designação) não podem ter valor nulo.

Na tabela Funcionário, o atributo escolhido para chave primária foi a ID Funcionário. Nenhum dos atributos da tabela pode ser inicializado com o valor NULL. Como existe um

relacionamento de muitos para um, entre as entidades Funcionário e Função existe uma chave estrangeira na tabela Funcionário que referencia o ID Função, que retrata a função desempenhada pelo funcionário.

Por fim, para a tabela Aluguer, o atributo escolhido para chave primária foi o número de aluguer (Nr Aluguer), já que se trata de um número de identificação único. Nenhum dos atributos da tabela pode ter valor nulo. Devido a existência de três relacionamentos de um para muitos, sendo eles entre as entidades Cliente, Aluguer; Funcionário, Aluguer e Carro, Aluguer, a tabela apresenta três chaves estrangeiras, ID Cliente, ID Funcionário e Nr Carro que referenciam o cliente que faz o aluguer, o funcionário que trata do aluguer e o carro que é alugado, respetivamente.

4.3 Normalização de Dados

Para a verificação da integridade, consistência e redundância controlada do modelo construído, seguiu-se a aplicação dos métodos de normalização de dados, através das três primeiras Formas Normais.

Através da observação do diagrama do modelo lógico, conseguimos perceber que todas tabelas possuem uma chave primária, não existem grupos repetitivos e cada atributo é atómico (o atributo composto Morada, presente nas entidades Cliente e Funcionário, transformaram-se em Rua e Cidade). Desta forma concluiu-se que o modelo respeita a 1FN.

Cliente	ID Cliente -> Nome, Data Nascimento, Email, Rua, Cidade
Telefone	Telefone -> ID Cliente
Aluguer	Nr Aluguer -> Data Início, Data Fim, ID Cliente, ID Funcionário, Nr Carro
Carro	Nr Carro -> Stand, Ano, Nr Lugares, Custo
Funcionário	ID Funcionário -> Nome, Telefone, Email, Local Trabalho, Rua, Cidade, Salário, ID Função
Função	ID Função -> Designação

Tabela 12 - Dependências funcionais entre os elementos das tabelas

A partir da análise da tabela de dependências funcionais construída, percebe-se que não existe nenhuma dependência parcial, e adicionando o facto que o modelo se encontra na 1FN, concluímos que se encontra agora na 2FN.

Por fim para garantir que estava na terceira forma normal (3FN), bastou verificar na tabela se existiam dependências transitivas. Pela análise, verifica-se que não existem, logo, tendo isto em vista e acrescentando o facto do modelo já se encontrar na 2FN, concluímos que se encontra também na 3FN.

4.4 Validação do Modelo com Interrogações do Utilizador

Após toda a construção do modelo lógico e da normalização dos dados, o grupo de estudantes seguiu para uma validação do mesmo, implementando uma série de *queries* previamente enunciadas nos requisitos de manipulação. Estas foram traduzidas em expressões de Álgebra Relacional, como poderemos ver mais à frente. Com vista a testar e verificar a qualidade das mesmas, foi utilizada a ferramenta *RelaX* que permitiu a execução das expressões propostas, sendo também disponibilizados os esquemas de cada uma delas.

Nesta secção serão mostradas as expressões desenvolvidas, bem como os seus esquemas e as tabelas resultantes da sua aplicação. O *input* utilizado no *RelaX* encontra-se disponível no [Anexo IV](#).

O requisito de manipulação (RM4), indica que deve ser possível listar todos identificadores de cliente, e o seu respetivo nome.

Expressão de Álgebra Relacional:

$$\pi \text{ ID_Cliente, Nome (Cliente)}$$

Utilizamos a função de projeção (π) sobre a tabela Cliente, escolhemos os atributos que se pretendiam listar, neste caso o identificador do cliente e o seu nome.



Figura 10 - Esquema da expressão relativa a RM4

Cliente.ID_Cliente	Cliente.Nome
100	'Tiago'
101	'Duarte'
102	'Luis'
103	'Joao'
104	'Ana'
105	'Maria'
106	'Joaquim'
107	'Carlos'
108	'Jose'
109	'Mariana'
110	'Josefina'

Tabela 13 - Resultado da aplicação da primeira expressão

No requisito de manipulação (RM5), é dito que deve ser possível listar os números de alugueres que um determinado cliente fez, bem como o seu identificador e a data de início desses alugueres.

Expressão de Álgebra Relacional:

$$\pi \text{ Nr Aluguer, ID Cliente, Data Início } (\sigma \text{ ID Cliente} = \text{X} (\text{Aluguer}))$$

Utilizamos a função de seleção (σ), sobre a tabela Aluguer, para escolher as linhas que tinham o identificador do cliente pretendido, e aplicamos a função de projeção (π) a essa tabela já modificada, para listar os campos de interesse, neste caso o Nr de Aluguer, o ID do cliente e a data de início do aluguer.

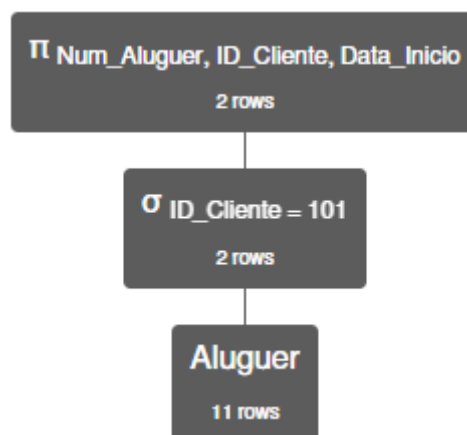


Figura 11 - Esquema da expressão aplicada para o ID Cliente 101

Aluguer.Num_Aluguer	Aluguer.ID_Cliente	Aluguer.Data_Inicio
71	101	2009-08-27
75	101	2009-09-25

Tabela 14 - Resultado da aplicação da segunda expressão para o ID Cliente 101

O requisito de manipulação (RM6), indica que deve ser possível listar os nomes de todos os clientes que foram atendidos por um determinado funcionário.

Expressão de Álgebra Relacional:

$\pi_{\text{Nome}} (\text{Cliente} \bowtie \text{Cliente.ID_Cliente} = \text{Aluguer.ID_Cliente} (\sigma_{\text{ID_Funcionario} = 2} (\text{Aluguer})))$

Utilizamos a função de seleção (σ), sobre a tabela Aluguer, para escolher as linhas que continham o identificador do funcionário pretendido. A aplicamos a função de junção (\bowtie), para juntar as tabelas Cliente e a tabela Aluguer (já pré-selecionada), mas apenas nas linhas onde os identificadores do cliente eram iguais. A partir dessa tabela maior criada, projetamos (π) apenas o atributo nome.

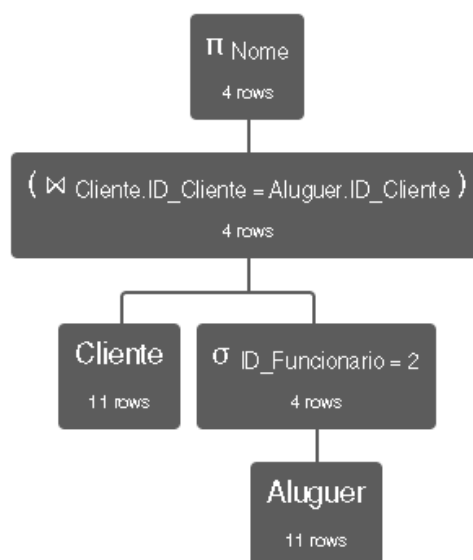


Figura 12 - Esquema da expressão aplicada para o ID Funcionário 2

Cliente.Nome
'Duarte'
'Joao'
'Joaquim'
'Jose'

Tabela 15 - Resultado da aplicação da segunda expressão para o ID Funcionário 2

No requisito de manipulação (RM7), é dito que deve ser possível listar todos os números dos carros que foram alugados por um funcionário que exerce uma determinada função.

Expressão de Álgebra Relacional:

π Carro. Nr Carro (Carro \bowtie Carro. Nr Carro=Aluguer. Nr Carro (Aluguer \bowtie Aluguer. ID

Funcionário = Funcionário. ID Funcionário (σ ID Função= X (Funcionário))))

Utilizamos a função de seleção (σ), sobre a tabela Funcionário, para escolher as linhas que continham o identificador da função pretendido. Para poder relacionar esta tabela, com a tabela Carro, encadeamos a função de junção (\bowtie) sobre as tabelas, Carro, Aluguer e Funcionário (já previamente selecionada) e aplicamos a função de projeção (π) para listar apenas o número do carro.

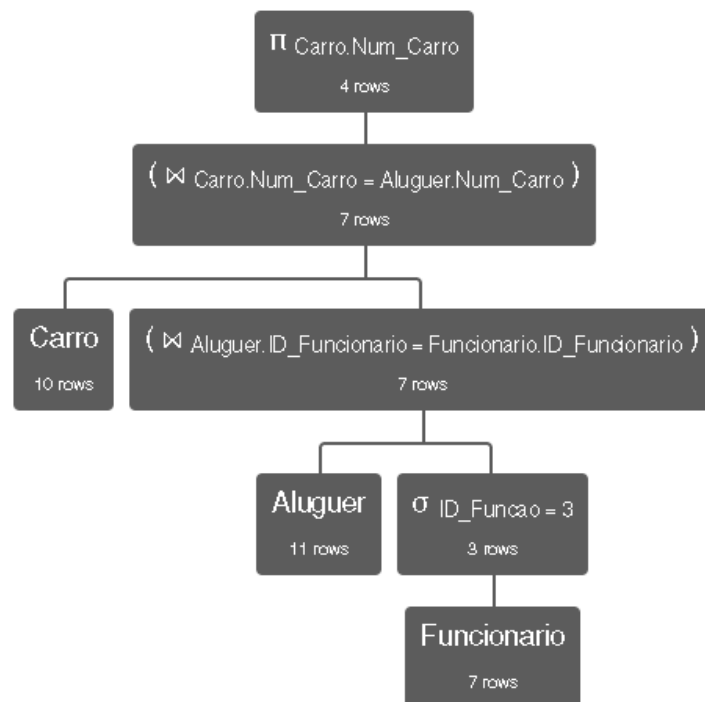


Figura 13 - Esquema da expressão aplicada para o ID Função 3

Carro.Num_Carro
1000
1004
1005
1007

Tabela 16 - Resultado da aplicação da segunda expressão para o ID Função 3

5. Implementação Física

5.1 Apresentação e explicação da base de dados implementada

Depois de criar toda a estrutura lógica para suportar a BD a ser implementada, o grupo prosseguiu para a implementação física da mesma. Para isso foram desenvolvidos vários *scripts* SQL, que definem todas as tabelas, *queries* e funcionalidades. Começamos por criar a BD em si através deste *script*:

```
CREATE DATABASE IF NOT EXISTS HoleInOne;  
-- DROP DATABASE HoleInOne;  
USE HoleInOne;
```

No primeiro comando é criada a BD com o nome “HoleInOne”. O uso das *keywords* `IF NOT EXISTS` serve de controlo, para criar a BD apenas se ela não existir. Depois, em comentário, encontra-se o comando responsável por destruir a BD. Este encontra-se em comentário para evitar destruir a BD quando ela está em uso, e por isso o comando só é verdadeiramente utilizado quando é realmente necessário. Por fim temos o comando responsável por avisar o *MySQL* que a BD que será utilizada é a “HoleInOne”, isto é, o comando `USE`.

Após a criação concreta da BD, partimos para a criação das tabelas, contudo para criação destas, foi necessário ter em atenção a sua ordem de criação, já que o facto de algumas terem chaves estrangeiras a referenciar outras, poderia gerar erros. Dito isto, depois de alguma análise por parte do grupo, percebemos que uma das formas possíveis de criar estas tabelas seria a seguinte:

Cliente → Telefones → Carro → Funcao → Funcionario → Aluguer

De acordo com a ordem acima apresentada, começamos por criar a tabela *Cliente* através da instrução `CREATE TABLE`. A tabela tem exatamente a mesma estrutura e informação que a tabela *Cliente* presente no modelo lógico, ou seja, os atributos nela contidos, *ID Cliente*, *Nome*, *Data Nascimento*, *Email*, *Rua*, *Cidade* estão agora definidos na tabela física

da BD. Os tipos já anteriormente definidos para os diversos atributos também foram mantidos (INT, VARCHAR (), DATE), assim como a verificação do atributo poder ou não ter valor nulo. Estes dois aspetos foram implementados na tabela física através do uso dos tipos de dados nativos do MySQL, e das *keywords* NOT NULL, respetivamente. A chave primária desta tabela, tal como já visto no modelo lógico é o ID Cliente, e esta foi explicitamente selecionada na tabela física. Toda esta informação pode ser confirmada e verificada no código SQL correspondente desenvolvido:

```
CREATE TABLE Cliente(  
    ID_Cliente INT NOT NULL,  
    Nome VARCHAR(100) NOT NULL,  
    Data_Nascimento DATE,  
    Email VARCHAR(50),  
    Rua VARCHAR(100) NOT NULL,  
    Cidade VARCHAR(50) NOT NULL,  
    PRIMARY KEY(ID_Cliente)  
);
```

As restantes tabelas construídas seguiram a mesma estratégia e critérios utilizados para a criação da tabela Cliente, ou seja, foram mantidos os mesmos atributos, tipos de dados e informações relativas ao valor nulo (ou não nulo) dos atributos. Porém existem dois casos na criação das tabelas que são importantes referir, sendo que ambos se encontram já na segunda tabela a ser criada, a tabela Telefones. Esta foi desenvolvida no modelo lógico e derivou de um atributo multivalorado pertencente à entidade Cliente. Assim sendo, esta tabela apresenta uma chave primária composta por dois atributos, o ID_Cliente e Telefone, sendo o ID_Cliente uma chave estrangeira e primária, em simultâneo. Para a identificação e caracterização da chave estrangeira foram utilizadas as *keywords* FOREIGN KEY e REFERENCES.

```
CREATE TABLE Telefones(  
    Telefone INT NOT NULL,  
    ID_Cliente INT NOT NULL,  
    PRIMARY KEY(Telefone, ID_Cliente),  
    FOREIGN KEY(ID_Cliente) REFERENCES Cliente(ID_Cliente)  
);
```

As tabelas seguintes seguem as regras já referidas no parágrafo anterior. Relativamente às chaves primárias compostas, não existe mais nenhuma tabela onde tal aconteça, porém, o uso de chaves estrangeiras será encontrado em mais duas tabelas (Funcionario e Aluguer). Em ambas, os procedimentos utilizados para a descrição dessas chaves estrangeiras foi o mesmo descrito na tabela Telefones. Assim sendo, com estas explicações, já não é necessário justificar mais nenhuma informação referente às próximas

tabelas e por isso, o grupo não achou relevante mencionar mais nenhum aspeto em relação à criação das seguintes tabelas:

```
CREATE TABLE Carro(  
    Nr_Carro INT NOT NULL,  
    Stand VARCHAR(20) NOT NULL,  
    Ano INT,  
    Nr_Lugares INT NOT NULL,  
    Custo DECIMAL(4,2) NOT NULL,  
    PRIMARY KEY(Nr_Carro)  
);
```

```
CREATE TABLE Funcao(  
    ID_Funcao INT NOT NULL,  
    Designacao VARCHAR(20) NOT NULL,  
    PRIMARY KEY(ID_Funcao)  
);
```

```
CREATE TABLE Funcionario(  
    ID_Funcionario INT NOT NULL,  
    Nome VARCHAR(100) NOT NULL,  
    Telefone INT NOT NULL,  
    Email VARCHAR(50),  
    Local_Trabalho VARCHAR(45),  
    Rua VARCHAR(100) NOT NULL,  
    Cidade VARCHAR(50) NOT NULL,  
    Salario DECIMAL(7,2) NOT NULL,  
    ID_Funcao INT NOT NULL,  
    PRIMARY KEY(ID_Funcionario),  
    FOREIGN KEY(ID_Funcao) REFERENCES Funcao(ID_Funcao)  
);
```

```

CREATE TABLE Aluguer(
    Nr_Aluguer INT NOT NULL,
    Data_Inicio DATE NOT NULL,
    Data_Fim DATE NOT NULL,
    ID_Cliente INT NOT NULL,
    ID_Funcionario INT NOT NULL,
    Nr_Carro INT NOT NULL,
    PRIMARY KEY(Nr_Aluguer),
    FOREIGN KEY(ID_Cliente) REFERENCES Cliente(ID_Cliente),
    FOREIGN KEY(ID_Funcionario) REFERENCES Funcionario(ID_Funcionario),
    FOREIGN KEY(Nr_Carro) REFERENCES Carro(Nr_Carro)
);

```

5.2 Criação de utilizadores da base de dados

A BD criada vai ser utilizada por vários utilizadores, cada um com diferentes acessos, permissões, e por isso foi preciso criá-los de acordo com os requisitos de controlo já levantados. Nesses requisitos estão especificadas as funções que cada tipo de funcionário pode ou não executar. Para generalizar as funções desempenhadas, o grupo optou por desenvolver ROLES, ou seja, um conjunto de privilégios que estão associados a cada um dos utilizadores.

Através da análise dos requisitos de controlo foi possível identificar a existência de três funções distintas, que os funcionários da empresa podem exercer, sendo elas dono/fundador (que se refere ao Sr. Orlando Esbelto), gerente e secretário. Assim sendo o código SQL referente à criação desses papéis (ROLES) é o seguinte:

```

CREATE ROLE Dono;
CREATE ROLE Gerente;
CREATE ROLE Secretario;

```

Depois da sua criação foi necessário atribuir os acessos e permissões às respetivas funções, para isso, o grupo teve em consideração os requisitos de controlo já apresentados neste relatório.

O RC1 e RC7 deram origem ao código SQL que descreve os acessos e permissões que o dono/fundador tem.

```

GRANT SELECT, INSERT, UPDATE, DELETE
ON HoleInOne.* TO Dono;

```

A partir dele conseguimos perceber que o dono/fundador pode realizar todas as operações sobre (SELECT, INSERT, UPDATE, DELETE) sobre qualquer uma das tabelas da BD (*), sendo o único que pode aceder à tabela Funcionarios.

Depois os requisitos de controlo RC2, RC3, RC4 permitiram descrever os privilégios que os funcionários que exercem a função de gerente têm. O código SQL referente a estes é o seguinte:

```
GRANT SELECT, INSERT, UPDATE, DELETE  
ON HoleInOne.Cliente TO Gerente;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE  
ON HoleInOne.Telefones TO Gerente;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE  
ON HoleInOne.Aluguer TO Gerente;
```

```
GRANT SELECT, INSERT, UPDATE, DELETE  
ON HoleInOne.Carro TO Gerente;
```

Como é possível observar, este tipo de funcionários apenas tem acesso às tabelas Cliente, Telefones, Aluguer e Carro, e em cada uma delas pode realizar qualquer uma das operações (SELECT, INSERT, UPDATE, DELETE).

Por fim, os requisitos RC5 e RC6 descrevem os acessos que os funcionários do tipo Secretário podem ter, e essas permissões são traduzidas no seguinte código:

```
GRANT INSERT, SELECT ON HoleInOne.Cliente TO Secretario;
```

```
GRANT INSERT, SELECT ON HoleInOne.Telefones TO Secretario;
```

```
GRANT INSERT, SELECT ON HoleInOne.Aluguer TO Secretario;
```

Neste caso os secretários apenas podem ter acesso às tabelas Cliente, Telefones, e Aluguer, e nessas tabelas apenas podem realizar as operações de INSERT e SELECT.

Evidentemente estes papéis têm de ser associados com utilizadores concretos, e por isso o grupo criou os utilizadores dentro do sistema, correspondentes aos funcionários que a empresa emprega atualmente.

```
CREATE USER 'Orlando'@'localhost' IDENTIFIED BY 'bdevida';
```

```
CREATE USER 'EdgarNovo'@'localhost' IDENTIFIED BY 'reidaspikenas';
```

```
CREATE USER 'Beatriz'@'localhost' IDENTIFIED BY 'princesadasarabias';
```

Estes utilizadores acima criados representam um exemplo de cada uma das funções existentes dentro da BD. O utilizador Orlando é o dono/fundador, o utilizador EdgarNovo é um gerente da empresa e por fim a Beatriz é uma secretária. Ao criar cada um destes utilizadores, foi associada a cada um, uma palavra-passe, que vem a seguir às *keywords* IDENTIFIED BY. Após a criação destes, foi então associada o respetivo *ROLE* que representa as suas funções dentro da empresa.

```
GRANT Dono TO 'Orlando'@'localhost';

GRANT Gerente TO 'EdgarNovo'@'localhost';

GRANT Secretario TO 'Beatriz'@'localhost';
```

5.3 Povoamento da base de dados

Existem várias formas de povoar uma BD, e neste caso o grupo decidiu implementar três delas. É importante notar que, independentemente da forma e método escolhido para povoar a BD é necessário respeitar a ordem pela qual as tabelas são povoadas, já que por exemplo, não é possível inserir registos na tabela Telefones antes de inserir registos na tabela Cliente, isto porque a tabela Telefones referencia a tabela Clientes através de uma chave estrangeira. Já nos tínhamos deparado com este problema na criação das tabelas, e por isso, para resolver este problema podemos usar uma ordem idêntica aquela já estabelecida em 5.1.

Como temos seis tabelas para povoar, o grupo decidiu que cada método iria povoar duas tabelas.

O primeiro método escolhido consiste no uso de ficheiros CSV. Esta estratégia foi aplicada para as tabelas Cliente e Telefones, e se forem inseridas por esta ordem não haverá qualquer conflito. Cada ficheiro CSV contém os dados referentes a essas tabelas, e esses dados são inseridos na BD central através de um programa de migração de dados escrito em *python*. Os ficheiros CSV contém os seguintes dados.

```
ID_Cliente, Nome, Data_Nascimento, Email, Rua, Cidade
100, Tiago, 2005-12-07, tiago@gmail.com, Rua_Tiago, Barcelos
101, Duarte, 2005-12-27, duarte@gmail.com, Rua_Duarte, Barcelos
102, Luis, 2005-10-14, luis@gmail.com, Rua_Luis, Vila_Verde
103, Joao, 2004-02-10, joao@gmail.com, Rua_Joao, Braga
104, Ana, 2000-01-01, ana@gmail.com, Rua_Ana, Albufeira
105, Maria, 1999-04-07, maria@gmail.com, Rua_Maria, Lisboa
106, Joaquim, 1951-04-14, joaquim@gmail.com, Rua_Joaquim, Porto
107, Carlos, 1990-06-06, carlos@gmail.com, Rua_Carlos, Ponte_Lima
108, Jose, 2005-11-05, jose@gmail.com, Rua_Jose, Lisboa
```

```
109,Mariana,2004-10-15,mariana@gmail.com,Rua_Mariana,Braga
110,Josefina,1970-07-20,josefina@gmail.com,Rua_Josefina,Braga
```

Estes são referentes à tabela Cliente, tal como se pode ver pela primeira linha, os próximos dizem respeito à tabela Telefones.

```
Telefone,ID_Cliente
912561490,100
945189011,102
938900133,103
956566570,104
978154351,105
914467100,106
945674411,107
900981233,108
967674412,109
911155567,110
922278874,100
```

O segundo método escolhido foi o uso de ficheiros JSON. Neste caso, os ficheiros foram responsáveis por povoar as tabelas Funcao e Funcionario, que ao serem inseridas por esta ordem respeitam o problema anteriormente referido. Mais uma vez, os dados contidos nestes ficheiros são colocados na BD central através do programa de migração já referido. O conteúdo do primeiro ficheiro JSON que irá ser mostrada a seguir, refere-se à tabela Funcao, e o segundo à tabela Funcionario.

```
[
  {"ID_Funcao": 1, "Designacao": "Dono"},
  {"ID_Funcao": 2, "Designacao": "Gerente"},
  {"ID_Funcao": 3, "Designacao": "Secretaria"}
]
```

```
[
  {"ID_Funcionario": 1, "Nome": "Orlando", "Telefone": 911177464,
  "Email": "orlando@hole.pt", "Local_Trabalho": "Vilamoura", "Rua":
  "Rua_Orlando", "Cidade": "Vilamoura", "Salario": 2000.0, "ID_Funcao":
  1},
```

```
  {"ID_Funcionario": 2, "Nome": "Beatriz", "Telefone": 956894531,
  "Email": "beatriz@hole.pt", "Local_Trabalho": "Vilamoura", "Rua":
```



```

"Rua_Beatriz", "Cidade": "Albufeira", "Salario": 1000.0, "ID_Funcao":
3},
{"ID_Funcionario": 3, "Nome": "Jose", "Telefone": 912145743,
"Email": "jose@hole.pt", "Local_Trabalho": "Vilamoura", "Rua":
"Rua_Jose", "Cidade": "Quarteira", "Salario": 1500.0, "ID_Funcao": 2},

{"ID_Funcionario": 4, "Nome": "Iara", "Telefone": 976494522,
"Email": "iara@hole.pt", "Local_Trabalho": "Porto", "Rua": "Rua_Iara",
"Cidade": "Maia", "Salario": 1000.0, "ID_Funcao": 3},
{"ID_Funcionario": 5, "Nome": "Rogerio", "Telefone": 900748741,
"Email": "rogerio@hole.pt", "Local_Trabalho": "Porto", "Rua":
"Rua_Rogerio", "Cidade": "Porto", "Salario": 1500.0, "ID_Funcao": 2},

{"ID_Funcionario": 6, "Nome": "Ines", "Telefone": 912345678,
"Email": "ines@hole.pt", "Local_Trabalho": "Braga", "Rua": "Rua_Ines",
"Cidade": "Barcelos", "Salario": 1000.0, "ID_Funcao": 3},

{"ID_Funcionario": 7, "Nome": "Edgar", "Telefone": 925444828,
"Email": "edgar@hole.pt", "Local_Trabalho": "Braga", "Rua":
"Rua_Edgar", "Cidade": "Carvalhal", "Salario": 1500.0, "ID_Funcao": 2}
]

```

Por fim, o último método utilizado foi o modelo de dados relacional do SQL, através do uso da instrução `INSERT`. Este método foi responsável por povoar as tabelas restantes, ou seja, a tabela Carro e Aluguer. Mais uma vez esta ordem respeita as dependências entre as várias tabelas, e o uso destas instruções de inserção foi feita pelo programa de migração. A seguir será mostrado o código SQL que insere os registos das tabelas Carro e Aluguer respetivamente.

```

INSERT INTO Carro
(Nr_Carro, Stand, Ano, Nr_Lugares, Custo)
VALUES (1000, 'Vilamoura', 2017, 4, 25.0),
(1001, 'Vilamoura', 2020, 4, 25.0),
(1002, 'Vilamoura', 2016, 2, 15.0),
(1003, 'Braga', 2013, 2, 15.0),
(1004, 'Braga', 2015, 2, 15.0),
(1005, 'Braga', 2021, 4, 25.0),
(1006, 'Porto', 2017, 4, 25.0),
(1007, 'Porto', 2013, 2, 15.0),
(1008, 'Porto', 2016, 4, 25.0),
(1009, 'Porto', 2009, 2, 15.0);

```

```

INSERT INTO Aluguer
(Nr_Aluguer, Data_Inicio, Data_Fim, ID_Cliente, ID_Funcionario, Nr_Carro)
VALUES (69, '2009-08-18', '2009-08-20', 106, 2, 1000),
(70, '2009-08-23', '2009-08-26', 108, 2, 1005),
(71, '2009-08-27', '2009-09-01', 101, 4, 1007),
(72, '2009-09-07', '2009-09-14', 104, 3, 1005),
(73, '2009-09-13', '2009-09-15', 107, 4, 1000),
(74, '2009-09-20', '2009-09-29', 105, 5, 1003),
(75, '2009-09-25', '2009-10-02', 101, 2, 1004),
(76, '2009-10-01', '2009-10-04', 109, 1, 1001),
(77, '2009-10-09', '2009-10-16', 100, 6, 1000),
(78, '2009-10-14', '2009-10-20', 102, 1, 1002),
(79, '2009-10-25', '2009-11-20', 103, 2, 1007);

```

Tal como já foi referido, todos estes métodos utilizaram o programa de migração de dados escrito em *python* para povoar a BD central, esse programa irá ser descrito e analisado ao pormenor mais adiante na secção 6. deste relatório.

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

Todas as BD necessitam de dispositivos onde possam operar e, por isso mesmo, para efetuar as suas operações, é necessário, principalmente, memória e poder de processamento. Tendo em consideração o aumento do tamanho da BD implementada e, para além disso, para que se consiga saber os requisitos de *hardware* onde esta irá operar, foi necessário calcular a sua dimensão base e a sua dimensão em caso de expansão das operações da empresa. Para isso, foi necessário consultar o tamanho, na documentação do MySQL, os tipos de dados que foram usados nos atributos de cada uma das relações criadas no projeto. De seguida, segue-se uma tabela com os tipos de dados e, para cada um, o seu respetivo tamanho.

Tipo de Dados	Tamanho (Bytes)
INT	4
DATE	3
VARCHAR(N), N ≤ 64	N+1
VARCHAR(N), N > 64	N+2
DECIMAL(4,2)	2
DECIMAL(7,2)	4

Tabela 17 - Tamanho em bytes de cada tipo de dados

Estes tamanhos foram calculados com base na documentação consultada pelo grupo e segue-se a justificação dos tamanhos calculados:

- DATE – tipo com tamanho fixo e estabelecido pelo MySQL, ocupando exatamente três bytes, independentemente do valor armazenado.
- INT – tipo numérico inteiro com tamanho fixo de 4 bytes, definido pelo MySQL. Representa inteiros entre -2.147.483.648 e 2.147.483.647.
- VARCHAR(N) – este tipo de dados tem um tamanho variável que depende de dois fatores: o conjunto de caracteres (*charset*) e o número máximo de caracteres (N) definidos. Por padrão, o MySQL utiliza a codificação utf8mb4, o que significa que cada caractere pode ocupar até 4 bytes (no pior caso). Para além disso, no MySQL, quando um campo textual ocupa menos do que 255 bytes, tem um prefixo de 1 byte, mas quando ocupa mais do que 255 bytes, já são necessários 2 bytes para o prefixo. Assim, no pior cenário possível (ou seja, se todos os caracteres ocuparem 4 bytes), o espaço ocupado por um campo VARCHAR(N) pode ser calculado pelas fórmulas:

$$\text{Tamanho_em_bytes} = N + 1, N \leq 64$$

$$\text{Tamanho_em_bytes} = N + 2, N > 64$$

- DECIMAL (p, s) – este tipo representa números decimais com precisão fixa. O parâmetro p representa o número total de dígitos (parte inteira + parte decimal), e s representa quantos desses dígitos estão depois da vírgula (parte decimal). O armazenamento interno em bytes é calculado com base na seguinte lógica definida pelo MySQL:
 - Cada grupo de até 9 dígitos ocupa 4 bytes.
 - Os dígitos são separados em parte inteira e parte decimal.
 - Cada parte é convertida em blocos de 9 dígitos, e cada bloco ocupa:
 - 1–2 dígitos → 1 byte
 - 3–4 dígitos → 2 bytes
 - 5–6 dígitos → 3 bytes
 - 7–9 dígitos → 4 bytes

Cálculo para os tipos usados:

DECIMAL (7,2) → parte inteira: 5 dígitos, parte decimal: 2 dígitos

Parte inteira (5 dígitos): ocupa 3 bytes

Parte decimal (2 dígitos): ocupa 1 byte

Total = 3 + 1 = 4 bytes

DECIMAL (4,2) → parte inteira: 2 dígitos, parte decimal: 2 dígitos

Parte inteira (2 dígitos): ocupa 1 byte

Parte decimal (2 dígitos): ocupa 1 byte

Total = 1 + 1 = 2 bytes

Tendo isto em conta, é possível calcular o tamanho que 1 registo ocupa em cada uma das tabelas. É de salientar que no caso dos atributos do tipo VARCHAR foi sempre considerado o pior caso, isto é, assumimos que os bytes alocados em memória são todos usados e, assim, depois de calculado o tamanho total da BD, sabemos que nunca poderá ser maior do que o valor calculado. Com isto, é possível escolher um *hardware* cuja capacidade nunca seja excedida pela demanda da BD e, assim, garantir a estabilidade e o desempenho do sistema ao longo do tempo. Além disso, este cálculo fornece uma base sólida para estimar o crescimento futuro da base de dados, facilitando decisões relacionadas à escalabilidade e à manutenção preventiva. Ao adotar esta abordagem conservadora, reduz-se o risco de perda de dados, de degradação do desempenho e de paragens inesperadas por falta de recursos, assegurando que a infraestrutura tecnológica esteja sempre preparada para suportar as exigências operacionais da aplicação.

A Tabela 18 apresenta a estimativa do espaço ocupado por cada registo da relação Cliente, considerando o pior caso para os atributos do tipo VARCHAR. O cálculo do seu tamanho é feito através da soma dos bytes usados por cada um dos seus respetivos atributos.

Cliente		
Atributo	Tipo de Dados	Tamanho (bytes)
ID_Cliente	INT	4
Nome	VARCHAR(100)	102
Data_Nascimento	DATE	3
Email	VARCHAR(50)	51
Rua	VARCHAR(100)	102
Cidade	VARCHAR(50)	51
Tamanho total: 313		

Tabela 18 - Espaço ocupado pelos atributos de cada registo da relação Cliente

A Tabela 19 mostra o espaço ocupado por cada registo da relação Telefones, onde se destacam os atributos Telefone e ID_Cliente, ambos do tipo INT. O cálculo segue a mesma lógica de considerar o espaço total ocupado por cada registo, para se estimar a dimensão desta tabela na base de dados.

Telefones		
Atributo	Tipo de Dados	Tamanho (bytes)
Telefone	INT	4
ID_Cliente	INT	4
Tamanho total: 8		

Tabela 19 - Espaço ocupado pelos atributos de cada registo da relação Telefones

A Tabela 20 apresenta o espaço ocupado por cada registo da relação Carro, considerando os tamanhos máximos possíveis dos atributos, nomeadamente do tipo VARCHAR. O cálculo segue a mesma lógica de considerar o espaço total ocupado por cada registo e somá-los.

Carro		
Atributo	Tipo de Dados	Tamanho (bytes)
Nr_Carro	INT	4
Stand	VARCHAR(20)	21
Ano	INT	4
Nr_Lugares	INT	4
Custo	DECIMAL(4,2)	2
Tamanho total: 35		

Tabela 20 - Espaço ocupado pelos atributos de cada registo da relação Carro

A Tabela 21 identifica o espaço necessário para armazenar cada registo da relação Função, assumindo os valores máximos possíveis para cada atributo. Tal como nas restantes tabelas, esta estimativa permite avaliar o impacto da relação na dimensão global da BD.

Funcao		
Atributo	Tipo de Dados	Tamanho (bytes)
ID_Funcao	INT	4
Designacao	VARCHAR(20)	21
Tamanho total: 25		

Tabela 21 - Espaço ocupado pelos atributos de cada registo da relação Funcao

A Tabela 22 descreve o espaço total ocupado por um registo da relação Funcionário, considerando os limites superiores de armazenamento dos atributos, especialmente os do tipo VARCHAR. Esta estimativa é útil para prever o volume de dados associado a esta entidade.

Funcionário		
Atributo	Tipo de Dados	Tamanho (bytes)
ID_Funcionario	INT	4
Nome	VARCHAR(100)	102
Telefone	INT	4
Email	VARCHAR(50)	51
Local_Trabalho	VARCHAR(45)	46
Rua	VARCHAR(100)	102
Cidade	VARCHAR(50)	51
Salario	DECIMAL(7,2)	4
ID_Funcao	INT	4
Tamanho total: 368		

Tabela 22 - Espaço ocupado pelos atributos de cada registo da relação Funcionario

A Tabela 23 apresenta a estimativa do espaço ocupado por cada registo da relação Aluguer, com base no tamanho máximo dos dados possíveis para cada campo. Este cálculo contribui para aferir o impacto da relação no armazenamento total da BD.

Aluguer		
Atributo	Tipo de Dados	Tamanho (bytes)
Nr	INT	4
Data_Inicio	DATE	3
Data_Fim	DATE	3
ID_Cliente	INT	4
ID_Funcionario	INT	4
Nr_Carro	INT	4
Tamanho total: 22		

Tabela 23 - Espaço ocupado pelos atributos de cada registo da relação Aluguer

Tendo em conta as tabelas acima, foi possível estimar o tamanho total ocupado pela BD.

Base de Dados			
Relação	Número de Registos	Tamanho p/ Registo (bytes)	Tamanho (bytes)
Cliente	11	313	3443
Telefones	11	8	88
Carro	10	35	350
Funcao	3	25	75
Funcionario	7	368	2576
Aluguer	11	22	242
Tamanho total: 6774 bytes ≈ 6.8kB			

Tabela 24 - Tamanho total da BD

Feito o cálculo do espaço total ocupado pela BD, foi possível analisar como este aumentará ao longo do tempo, quando consideradas várias taxas de crescimento. O resultado dessa análise encontra-se na tabela abaixo.

Base de Dados					
Taxa de Crescimento Anual	1º Ano	2º Ano	3º Ano	5º Ano	10º Ano
5%	7,14 kB	7,48 kB	7,82 kB	8,5 kB	10,2 kB
10%	7,48 kB	8,16 kB	8,84 kB	10,2 kB	13,6 kB
20%	8,16 kB	9,52 kB	10,88 kB	13,6 kB	20,4 kB
30%	8,84 kB	10,88 kB	12,92 kB	17 kB	27,2 kB
50%	10,2 kB	13,6 kB	17 kB	23,8 kB	40,8 kB

Tabela 25 - Taxa de crescimento anual da BD

5.5 Definição e caracterização de vistas de utilização em SQL

No contexto de BD, a utilização de vistas revela-se essencial tanto para facilitar o acesso controlado e organizado à informação, tanto como para responder de forma simples e eficiente às interrogações mais frequentes dos diferentes utilizadores do sistema. Estas são usadas também para controlar as permissões atribuídas a cada utilizador. Uma vista é uma tabela temporária, que é criada ou através de dados de tabelas já existentes ou através de outras vistas, e permitem abstrair a estrutura da BD e controlar o acesso a atributos específicos ou a subconjuntos de registos.

A vista `NumerosCarrosAlugadosporFuncionáriosTipo2` foi criada com o objetivo de representar um subconjunto de informação da BD, que inclui apenas os carros que foram alugados por funcionários cuja função corresponde a “Gerente”, com `ID_Funcao = 2`. A instrução `CREATE VIEW` permite criar esta tabela temporária, e o nome da vista foi definida de forma descritiva, facilitando a sua identificação. As operações `SELECT`, `FROM` e `INNER JOIN` garantem a seleção correta dos dados, combinando a informação das tabelas `Funcionario`, `Aluguer` e `Carro`. De seguida, apresenta-se a vista em código SQL:

```
CREATE VIEW NumerosCarrosAlugadosporFuncionárioTipo2 AS
SELECT Ca.Nr_Carro
FROM Carro AS Ca
INNER JOIN( Aluguer AS A
INNER JOIN(
SELECT * FROM Funcionario
WHERE ID_Funcao = 2) AS F
ON A.ID_Funcionario = F.ID_Funcionario)
ON Ca.Nr_Carro = A.Nr_Carro;
```

Nesta definição foi aplicada uma seleção sobre a tabela `Funcionario`, filtrando apenas os funcionários cuja função é 2 (ou seja, função “Gerente”). De seguida, foi feita uma junção `JOIN` com a tabela `Aluguer`, para obter os alugueres efetuados por esses funcionários, e, por fim, a junção com a tabela `Carro` permitiu identificar os números dos carros efetivamente alugados pelos gerentes. Desta forma, a projeção limita a vista ao atributo `Nr_Carro`, ocultando, assim, o resto dos dados. Também seria possível criar uma vista de forma a que o utilizador não tivesse permissões totais, mas que, sem criar conflitos, conseguisse consultar apenas o resultado específico de uma vista. Por exemplo, no caso da vista apresentada em baixo, é concedido o privilégio de leitura (`SELECT`) apenas sobre esta vista, ao papel “Gerente”.

```
GRANT SELECT ON NumerosCarrosAlugadosporFuncionarioTipo2 TO Gerente;
```

Desta forma, mesmo que o utilizador com o papel “Gerente” não tenha permissões sobre as tabelas originais, consegue aceder de forma controlada à informação relevante, filtrada e projetada por esta vista. Esta abordagem permite manter a segurança dos dados, o que evita a exposição de dados privados ou irrelevantes para cada tipo de utilizador.

5.6 Tradução das interrogações do utilizador para SQL

Durante a análise dos requisitos e validação do modelo lógico, foram identificadas diversas interrogações (através dos requisitos de manipulação) que os utilizadores do SBD poderão realizar no seu dia a dia. Estas interrogações mostram necessidades operacionais e de gestão, tais como:

- Consultar dados de clientes;
- Verificar informação sobre alugueres;
- Obter listagens associadas a funções específicas dos funcionários.

A linguagem SQL foi utilizada para desenvolver essas interrogações, permitindo aceder, filtrar e cruzar dados das diferentes tabelas. Primeiramente, temos o RM1. Este requisito consiste em consultar os dados de um cliente específico, e, esta interrogação permite aceder a todos os dados de um cliente específico, neste caso com o “ID_Cliente = 100”. Usamos a instrução `SELECT` para fazer uma consulta, o `FROM` para indicar de que tabela estamos a ir buscar os dados e o `WHERE` é usado de forma a apenas projetar um certo atributo com um certo valor (100 nesta interrogação).

```
SELECT *  
FROM Cliente  
WHERE ID_Cliente = 100;
```

Passando agora para o RM2, este requisito existe de forma a ser possível consultar os dados de um funcionário específico. Similarmente à consulta anterior, esta *query* permite verificar toda a informação associada a um determinado funcionário. Da mesma forma, usamos o `SELECT` para fazer a consulta (seguido de atributos), o comando `FROM` de modo a ir buscar os dados da tabela (neste caso “Funcionario”) e por fim, novamente, o comando `WHERE` para filtrar os dados do atributo escolhido também neste comando (como na interrogação abaixo, ID_Funcionario = 1).

```
SELECT *  
FROM Funcionario  
WHERE ID_Funcionario = 1;
```


De seguida, temos o RM3. Este permite consultar os dados de um certo aluguer, e permite aceder ao registo completo desse mesmo aluguer. Neste caso, trata-se de um aluguer com o número 69, útil para verificar datas, cliente e veículo associados. O requisito RM3 funciona da mesma forma que as outras duas interrogações anteriormente dadas, ou seja, da mesma forma, o `SELECT` serve para fazer a consulta, o `FROM` para escolher a tabela da qual vamos buscar os dados e o `WHERE` para filtrar novamente os valores dos atributos (`Nr_Aluguer = 69;`).

```
SELECT *  
FROM Aluguer  
WHERE Nr_Aluguer = 69;
```

Seguidamente, temos a implementação da instrução SQL do RM4. Este requisito faz uma listagem simples com os nomes e identificadores de todos os clientes registados no sistema. Desta vez, apesar de termos certas instruções como nas de cima, não temos um `WHERE`, apenas um `SELECT` que faz a listagem do `ID_Cliente` e do `Nome`, e o comando `FROM` que, novamente, indica qual tabela de onde os dados são retirados.

```
SELECT ID_Cliente, Nome  
FROM Cliente;
```

Passamos agora ao RM5, que serve para listar os alugueres realizados por um cliente específico. Este apresenta os alugueres realizados por um certo cliente (`ID_Cliente = 100`), incluindo o número do aluguer e a data de início. Para fazer uma listagem usamos o comando `SELECT`, seguido dos atributos da tabela que se quer consultar. De seguida, usamos o comando `FROM` para escolher a tabela da qual retiramos os atributos, e, por fim, usamos o comando `WHERE` para filtrar o cliente cujo o ID é 100 (`ID_Cliente = 100`).

```
SELECT Nr_Aluguer, ID_Cliente, Data_Inicio  
FROM Aluguer  
WHERE ID_Cliente = 100;
```

Continuando com as interrogações, temos agora o RM6, o qual existe para listar todos os clientes atendidos por um determinado funcionário. Este é capaz de identificar isto cruzando os dados entre as tabelas “Cliente” e “Aluguer”. Começamos por utilizar o comando `SELECT`, para obter apenas o nome do cliente. De seguida, usamos o comando `FROM` para escolher a tabela que vai ser usada (Cliente), seguido do *alias* `AS C` para simplificar a sua referência ao longo da *query*. A seguir aplicamos um `INNER JOIN` para fazer uma junção interna entre a tabela Cliente e uma *subquery* da tabela Aluguer (`SELECT * FROM Aluguer WHERE ID_Funcionario = 1`), que apenas seleciona os alugueres em que o `ID_Funcionario = 1`. Por fim, utilizamos a

cláusula **ON**, que é uma condição de junção. Esta, associa cada cliente ao aluguer, apenas se o seu **ID_Cliente** for igual ao **ID_Cliente** do aluguer.

```
SELECT C.Nome
FROM Cliente AS C
INNER JOIN (SELECT * FROM Aluguer WHERE ID_Funcionario = 1) AS A
ON C.ID_Cliente = A.ID_Cliente;
```

Por fim, temos o RM7, que serve para listar os carros alugados por funcionários com uma certa função. Esta consulta identifica os carros alugados por funcionários que exercem uma determinada função, neste caso com **ID_Funcao = 2**. Nesta interrogação, iniciamos da mesma forma como as outras todas, começamos com o **SELECT** pois queremos o número do carro (**Nr_Carro**), e de seguida usamos **FROM** e o **AS**, para consultar a tabela Carro e para nos referirmos a Carro como “Ca”, respetivamente. Da mesma forma que em RM6, aqui usamos também um **INNER JOIN**, de forma a fazer a junção interna entre a tabela Carro e uma *subquery* da tabela Funcionario. Esta *subquery*, da mesma forma que em RM6, também só seleciona os funcionários que exercem a função com ID igual a 2 (**ID_Funcao = 2**). Usamos também **AS** de novo para nos referirmos a Aluguer como “A”. De seguida, executamos a cláusula **ON**, ainda dentro do **INNER JOIN**, que liga a tabela Aluguer com a *subquery* de Funcionario, usando a correspondência entre os seus ID's. Por fim, já fora do **INNER JOIN**, fazemos uso de outra cláusula **ON**, de forma a fazer a junção entre a tabela Carro e a junção do comando **ON** dentro do **INNER JOIN**, através do número do carro.

```
SELECT Ca.Nr_Carro
FROM Carro AS Ca
INNER JOIN Aluguer AS A
ON Ca.Nr_Carro = A.Nr_Carro
INNER JOIN (
    SELECT ID_Funcionario
    FROM Funcionario
    WHERE ID_Funcao = 2
) AS F
ON A.ID_Funcionario = F.ID_Funcionario;
```

5.7 Indexação do Sistema de Dados

Os índices são estruturas de dados que aceleram a busca e acesso a dados numa tabela. Apesar de aumentar o tempo das inserções, atualizações e exclusões, já que os índices também precisam de ser atualizados, facilitam o tempo de procura devido ao uso de estruturas de dados desenhadas para acelerar este tipo de processo. Na realização deste trabalho prático decidimos que os índices poderiam ser utilizados em algumas das relações que criamos no contexto da nossa BD, apesar de que, devido à dimensão do projeto, a diferença do tempo computacional da realização das operações que necessitamos foi, antes e depois da criação dos índices, praticamente, nula. Contudo, caso a dimensão da BD aumentasse (tal como foi previsto na Tabela 25) a criação dos índices seria uma grande vantagem, já que, do ponto de vista do uso da BD no contexto das operações diárias da empresa, a produtividade seria maior, havendo um maior rendimento no processamento das consultas mais frequentes. Além disso, os índices contribuiriam para uma melhor escalabilidade do sistema, permitindo que a BD continuasse a apresentar tempos de resposta aceitáveis mesmo com o aumento significativo do volume de dados. Assim, embora neste estágio inicial os ganhos de desempenho não tenham sido expressivos, a adoção de índices representa uma boa prática de desenvolvimento e preparação para o crescimento futuro da base de dados. Dito isto, o grupo achou necessária a implementação dos seguintes índices:

- `idx_Datas_Aluguer` – este índice otimiza consultas que filtram ou ordenam registros com base no intervalo de datas de aluguer (`Data_Inicio` e `Data_Fim`). Aachamos necessária a sua implementação, já que auxilia na verificação de sobreposição de datas, por exemplo, para verificar se um carro já está alugado num determinado período. É vantajoso quando se usa: o `TRIGGER VerificarDisponibilidadeCarro` e a `FUNCTION CarroAlugadoAgora`.

```
CREATE INDEX idx_Datas_Aluguer ON Aluguer(Data_Inicio, Data_Fim);
```

- `idx_aluguer_Nr_Carro` – acelera consultas que procuram por alugueres de um carro específico. Foi criado já que o grupo achou que era essencial para se poder filtrar rapidamente todos os alugueres de um carro, pois, no contexto das operações diárias da empresa, é necessário que se façam várias consultas sobre a disponibilidade dos carros que estão disponíveis nas frotas dos diferentes *stands*. É vantajoso quando se usa: o `TRIGGER VerificarDisponibilidadeCarro` e a `FUNCTION CarroAlugadoAgora`.

```
CREATE INDEX idx_aluguer_Nr_Carro ON Aluguer(Nr_Carro);
```

- `idx_aluguer_ID_Cliente` - melhora consultas que procuram por alugueres feitos por um determinado cliente (`ID_Cliente`). Isto permite calcular mais rapidamente quantos alugueres um cliente fez ou recuperar o seu histórico de alugueres, o que é bastante importante no decorrer das atividades da empresa, já que, acelera o processo de obtenção das informações de um dado cliente, sendo possível, para os funcionários da empresa, que tratem dos registos dos alugueres mais rapidamente. É utilizado na `FUNCTION TotalAlugueresCliente`.

```
CREATE INDEX idx_aluguer_ID_Cliente ON Aluguer(ID_Cliente);
```

- `idx_aluguer_ID_Funcionario` - este índice otimiza consultas que procuram por alugueres feitos por um determinado funcionário (`ID_Funcionario`). Assim, é possível diminuir o tempo de processamento computacional quando se procura pela chave estrangeira (`ID_Funcionario`) na tabela `Aluguer`, o que permite, ao realizar todo o tipo de comandos e instruções seja em funções, queries ou procedimentos, que se encontrem os casos correspondentes ao que se pretende, mais rapidamente. É utilizado, por exemplo, no `PROCEDURE MostrarAlugueresPorFuncionario`.

```
CREATE INDEX idx_aluguer_ID_Funcionario ON Aluguer(ID_Funcionario);
```

5.8 Implementação de procedimentos, funções e gatilhos

A implementação de procedimentos, funções e gatilhos (*triggers*) num SGBD revela-se bastante importante por diversos motivos, tanto a nível funcional como estrutural. Estes elementos permitem realizar as operações diretamente no servidor da BD, promovendo maior organização, segurança, desempenho e integridade dos dados. Ao colocar estes diretamente no SGBD, evita-se a duplicação de código e, assim, qualquer alteração à lógica é feita num único ponto, facilitando a manutenção, o que reduz o risco de inconsistências. Para além disto, funções e procedimentos podem ser reutilizados por diferentes utilizadores. Além do mais, é possível restringir o acesso direto às tabelas e expor apenas os procedimentos e funções necessários. Desta forma, os utilizadores interagem com a BD de forma controlada, prevenindo acessos indevidos ou alterações incorretas.

Na realização deste projeto foram implementados vários procedimentos, funções e gatilhos. Primeiramente serão explicados os procedimentos, de seguida, as funções e, para terminar, os gatilhos. Tendo isto em conta, os procedimentos são:

- `NumerosCarrosAlugadosporFuncionarioTipo` - este procedimento é responsável por, tal como é descrito pelo seu nome, calcular o número de carros alugados por cada tipo de função dos funcionários. Ele permite filtrar os alugueres com base no cargo do funcionário responsável, o que facilita a análise de operações por função, o que poderia ser interessante para a empresa, caso fosse necessário analisar a rentabilidade dos seus diferentes empregados, por tipo de função. Para implementar este, é:

1. realizada uma subconsulta à tabela `Funcionario` para obter apenas os funcionários cujo `ID_Funcao` corresponde ao valor do parâmetro `Tipo` passado na chamada do procedimento;

```
SELECT * FROM Funcionario WHERE ID_Funcao = Tipo
```

2. de seguida, essa subconsulta é associada à tabela `Aluguer` com base no identificador do funcionário (`ID_Funcionario`), de forma a obter os alugueres efetuados apenas por esses funcionários;

```
Aluguer AS A
  INNER JOIN (
    SELECT * FROM Funcionario WHERE ID_Funcao = Tipo
  ) AS F
  ON A.ID_Funcionario = F.ID_Funcionario
```

3. por fim, os alugueres obtidos são associados à tabela `Carro` através da coluna `Nr_Carro`, e o procedimento retorna os números dos carros alugados.

```

CREATE PROCEDURE NumerosCarrosAlugadosporFuncionarioTipo(IN Tipo INT)
BEGIN

    SELECT Ca.Nr_Carro
    FROM Carro AS Ca
    INNER JOIN (
        Aluguer AS A
        INNER JOIN (
            SELECT * FROM Funcionario WHERE ID_Funcao = Tipo
        ) AS F
        ON A.ID_Funcionario = F.ID_Funcionario
    )
    ON Ca.Nr_Carro = A.Nr_Carro;

END $$

```

- Adiciona_User_e_Telefone – através da passagem de vários argumentos usados no procedimento, este, tal como o seu nome indica, insere um novo cliente na BD e, simultaneamente, regista o respetivo número de telefone, garantindo que ambas as operações são realizadas de forma segura e consistente através de uma transação (TRANSACTION). Assim, no contexto das operações de registo necessárias no dia a dia da empresa, é possível usar este procedimento para anexar um novo utilizador dos serviços da empresa. Para isto, foi criado o código SQL com a seguinte lógica:

1. para garantir que todas as instruções seguintes são executadas como uma única unidade de trabalho, é usada a TRANSACTION;

START TRANSACTION;

2. de seguida, são inseridas as informações passadas como argumento nas tabelas Cliente e Telefones;

```

INSERT INTO Cliente
(ID_Cliente, Nome, Data_Nascimento, Email, Rua, Cidade)
VALUES (Id, Nome, Nasc, Email, Rua, Cidade);

INSERT INTO Telefones
(Telefone, ID_Cliente)
VALUES (Telefone, Id);

```

3. caso ambas as inserções tenham sido bem-sucedidas, os dados são efetivamente gravados na BD usando a instrução COMMIT.

```

CREATE PROCEDURE Adiciona_User_e_Telefone(IN Id INT,
                                           IN Nome VARCHAR(100),
                                           IN Nasc DATE,
                                           IN Email VARCHAR(50),
                                           IN Rua VARCHAR(100),
                                           IN Cidade VARCHAR(50),
                                           IN Telefone INT)

BEGIN

START TRANSACTION;

INSERT INTO Cliente
(ID_Cliente, Nome, Data_Nascimento, Email, Rua, Cidade)
VALUES (Id, Nome, Nasc, Email, Rua, Cidade);

INSERT INTO Telefones
(Telefone, ID_Cliente)
VALUES (Telefone, Id);

COMMIT;

END $$

```

- **MostrarCarrosAlugadosStand** – ao passar o nome do *stand* da qual se precisa de informações, este procedimento devolve o número total de alugueres realizados por cada Carro, dessa *stand*. Por uma questão de manutenção ou até substituição da frota, este procedimento pode ser chamado para que possa obter a informação necessária. Com esse fim, foi desenvolvido o seguinte código SQL:

1. juntam-se as tabelas Aluguer e Carro através da coluna comum Nr_Carro. Assim, o sistema sabe qual carro foi usado no aluguer, e pode aceder também a qual das *stands* pertence o carro;

```

FROM Aluguer AS A
INNER JOIN Carro AS C ON A.Nr_Carro = C.Nr_Carro

```

2. de seguida, filtram-se os registos de carros para incluir apenas os pertencentes ao stand indicado no parâmetro aux;

```

WHERE C.Stand = aux

```

3. para terminar, são agrupados os registos por número de carro, assim, usa-se COUNT (*) para contar quantas vezes cada carro do stand foi alugado.

DELIMITER \$\$

```
CREATE PROCEDURE MostrarCarrosAlugadosStand(IN aux VARCHAR(20))
BEGIN
```

```
SELECT A.Nr_Carro, COUNT(*) AS Numero_de_Alugueres
FROM Aluguer AS A INNER JOIN Carro AS C
ON A.Nr_Carro = C.Nr_Carro
WHERE C.Stand = aux
GROUP BY Nr_Carro;
```

```
END $$
```

- `ListarFuncionarios_e_Funcao` – este procedimento é responsável por listar todos os nomes dos funcionários registados na BD, juntamente com a sua Designacao da tabela Função. Tem importância porque, assim, o utilizador tem um procedimento próprio para tal, o que facilita o acesso a estes dados. Com essa finalidade, foi desenvolvido o seu código o SQL:

1. a consulta começa por aceder à tabela Funcionario, que recebe o *alias* "F";

```
FROM Funcionario AS F
```

2. Através de um `INNER JOIN`, o SGBD faz a correspondência entre: o `F.ID_Funcao`: e o `FC.ID_Funcao`. Assim, associa-se a cada funcionário a sua respetiva função textual (como por exemplo, "Gerente"). Por ser `INNER JOIN`, só entram no resultado os funcionários que tenham uma função válida (ou seja, com correspondência exata no campo `ID_Funcao` da tabela Funcao);

```
FROM Funcionario AS F
INNER JOIN Funcao AS FC
ON F.ID_Funcao = FC.ID_Funcao;
```

3. para terminar, seleciona-se as colunas pretendidas na tabela resultante, isto é, as colunas Nome e Designacao (provenientes das tabelas Funcionario e Funcao, respetivamente).


```
DELIMITER $$
```

```
CREATE PROCEDURE ListarFuncionarios_e_Funcao()  
BEGIN
```

```
SELECT F.Nome, FC.Designacao  
FROM Funcionario AS F  
INNER JOIN Funcao AS FC  
ON F.ID_Funcao = FC.ID_Funcao;
```

```
END $$
```

- **MostrarAlugueresPorFuncionario** – este procedimento, tal como o seu nome indica, mostra todos os alugueres realizados por um dado funcionário. Por uma questão de avaliação da produtividade da empresa, este procedimento foi criado e, por isso, é, provavelmente, um dos procedimentos mais importantes no que toca à obtenção de estatísticas internas à empresa. Com esse intuito, foi escrito o seguinte código o SQL:

1. primeiramente, juntam-se as tabelas **Funcionario** e **Aluguer** usando um **INNER JOIN** que faz ‘match’ entre os campos **ID_Funcionario** de cada uma das tabelas. Isso permite associar cada aluguer ao funcionário que o realizou;

```
FROM Funcionario AS F INNER JOIN Aluguer AS A  
ON F.ID_Funcionario = A.ID_Funcionario
```

2. após isto, todos os alugueres feitos por cada funcionário são agrupados e, assim, **COUNT (*)** pode contar quantos registos de aluguer existem por funcionário;

```
GROUP BY F.ID_Funcionario;
```

3. Com isto, temos a lógica do procedimento concluída.

```
DELIMITER $$
```

```
CREATE PROCEDURE MostrarAlugueresPorFuncionario()  
BEGIN
```

```
SELECT F.ID_Funcionario, COUNT(*) AS 'Nº Alugueres'  
FROM Funcionario AS F INNER JOIN Aluguer AS A  
ON F.ID_Funcionario = A.ID_Funcionario  
GROUP BY F.ID_Funcionario;
```

```
END $$
```

- `ApagarAlugueresAntigos` – este procedimento é, talvez, dos mais importantes de implementar. A sua chamada pode ser feita passando como argumento a data (`DATE`) até à qual não se querem alugueres. Assim, é possível diminuir o número de registos guardados na BD, o que permite melhorar a performance, caso esta tenha a mesma deteriorada. Para essa finalidade, foi escrito o seguinte código o SQL:

1. a tabela `Aluguer` (com o *alias* “A”) é selecionada e verifica-se a condição que elimina todos os registos em que a data de fim do aluguer seja anterior à data recebida como parâmetro.

```
DELETE FROM Aluguer AS A
WHERE A.Data_Fim < aux;
```

2. com isto, fica-se com o procedimento:

```
DELIMITER $$

CREATE PROCEDURE ApagarAlugueresAntigos(IN aux DATE)
BEGIN

DELETE FROM Aluguer AS A
WHERE A.Data_Fim < aux;

END $$
```

A principal diferença entre procedimentos e funções em SQL é que as funções são obrigadas a retornar um valor, enquanto que os procedimentos podem simplesmente executar instruções sem retorno. Devido a essa característica, as funções são especialmente úteis quando é necessário obter um resultado direto e reutilizável em *queries* ou em validações lógicas. Tendo isso em consideração, foram implementadas duas funções com objetivos distintos, mas práticos no contexto da BD:

- `TotalAlugueresCliente` – esta função tem como objetivo retornar o número total de alugueres realizados por um determinado cliente, com base no seu ID. Para executar esta função, é necessário chamá-la com um `cliente_id` `INT`, que é o ID do cliente da qual queremos informações. Em termos de lógica de execução, é bastante simples:
1. é declarada uma variável `total` do tipo `INT` que conta quantos registos existem na tabela `Aluguer` com `ID_Cliente = cliente_id` e armazena o resultado em `total`;

```

BEGIN
    DECLARE total INT;
    SELECT COUNT(*) INTO total
FROM Aluguer
    WHERE ID_Cliente = cliente_id;

```

2. depois disto, apenas é necessário retornar esse valor.

```
CREATE FUNCTION TotalAlugueresCliente(cliente_id INT) RETURNS INT
```

```
DETERMINISTIC
```

```

BEGIN
    DECLARE total INT;
    SELECT COUNT(*) INTO total
FROM Aluguer
    WHERE ID_Cliente = cliente_id;
    RETURN total;
END $$

```

```
DELIMITER $$
```

- CarroAlugadoAgora – para determinar se um carro passado como argumento na função (carro_id INT) está, nesse momento, alugado, é retornada uma variável de tipo INT com valor 1 caso seja verdade, ou, 0 caso seja falso. Para isto, o grupo desenvolveu o seguinte código em SQL:

1. declara-se uma variável emUso do tipo BOOLEAN;

```
DECLARE emUso BOOLEAN;
```

2. verifica-se se existe um registo na tabela Aluguer com: Nr_Carro = carro_id e cuja data atual (CURDATE()) esteja entre a data de início e de fim do aluguer;

```

SELECT *
FROM Aluguer
WHERE Nr_Carro = carro_id
AND CURDATE() BETWEEN Data_Inicio AND Data_Fim

```

3. usa-se a função `EXISTS(...)` para verificar se essa condição é satisfeita e retorna-se o resultado lógico (`TRUE` ou `FALSE`).

```
CREATE FUNCTION CarroAlugadoAgora(carro_id INT) RETURNS BOOLEAN

DETERMINISTIC

BEGIN
    DECLARE emUso BOOLEAN;

    SELECT EXISTS(
        SELECT *
    FROM Aluguer
    WHERE Nr_Carro = carro_id
    AND CURDATE() BETWEEN Data_Inicio AND Data_Fim
    ) INTO emUso;
    RETURN emUso;
END $$
```

Para terminar, tal como foi dito no início desta secção, o grupo também decidiu implementar gatilhos (`TRIGGER`). Os gatilhos permitem aplicar regras automáticas que asseguram a validade dos dados inseridos ou modificados (como por exemplo, impedir o aluguer de um carro já reservado). Devido ao facto destes executarem automaticamente quando se faz uma dada operação em uma qualquer tabela (isto é, executa automaticamente quando se faz uma determinada instrução na/nas tabela/tabelas para o qual foi codificado), foi considerada importante a implementação do seguinte `TRIGGER`:

- `VerificarDisponibilidadeCarro` - este gatilho é responsável por, antes de uma inserção na tabela `Aluguer`, verificar se a data inserida na nova instância de `Aluguer` pertence a um intervalo de tempo ao qual o carro já esteja alugado. Assim, é garantido que o sistema não regista casos de aluguer em que hajam conflitos de datas. Caso a data seja inválida, é retornada uma mensagem de erro indicativa. Para implementar este gatilho, foi escrito o seguinte código SQL:

1. primeiramente, é definido o `TRIGGER` com o nome `VerificarDisponibilidadeCarro` que é ativado antes (`BEFORE`) de se inserir uma nova linha na tabela `Aluguer`.

```
CREATE TRIGGER VerificarDisponibilidadeCarro

BEFORE INSERT ON Aluguer

FOR EACH ROW
```

2. depois disto, é declarada a variável que vai armazenar o número de conflitos detetados, ou seja, sobreposições de datas;

DECLARE alugado INT;

3. de seguida, é necessário verificar, para todos os alugueres existentes com o carro pretendido, se existe alguma sobreposição de datas entre o novo aluguer (NEW.Data_Inicio, NEW.Data_Fim) e os alugueres já registados;

```
SELECT COUNT(*) INTO alugado
FROM Aluguer
WHERE Nr_Carro = NEW.Nr_Carro
AND (
    NEW.Data_Inicio BETWEEN Data_Inicio AND Data_Fim
    OR
    NEW.Data_Fim BETWEEN Data_Inicio AND Data_Fim
    OR
    Data_Inicio BETWEEN NEW.Data_Inicio AND NEW.Data_Fim
    OR
    Data_Fim BETWEEN NEW.Data_Inicio AND NEW.Data_Fim
);
```

4. para terminar, verifica-se, depois de iterar sobre a tabela, se existe algum conflito de datas. Caso exista, é usado o comando `SIGNAL` para lançar um erro personalizado, impedindo que o novo aluguer seja inserido.

```

DELIMITER $$
CREATE TRIGGER VerificarDisponibilidadeCarro
BEFORE INSERT ON Aluguer
FOR EACH ROW
BEGIN
    DECLARE alugado INT;

    SELECT COUNT(*) INTO alugado
    FROM Aluguer
    WHERE Nr_Carro = NEW.Nr_Carro
    AND (
        NEW.Data_Inicio BETWEEN Data_Inicio AND Data_Fim
        OR
        NEW.Data_Fim BETWEEN Data_Inicio AND Data_Fim
        OR
        Data_Inicio BETWEEN NEW.Data_Inicio AND NEW.Data_Fim
        OR
        Data_Fim BETWEEN NEW.Data_Inicio AND NEW.Data_Fim
    );
    IF alugado > 0 THEN
        SIGNAL SQLSTATE '45000'
        SET MESSAGE_TEXT = 'Erro: O carro já está alugado nesse período.';
    END IF;
END $$

```

6. Implementação do sistema de migração dados

Nesta secção é apresentada a implementação de um programa para realizar a migração de dados de três fontes distintas – com modelos de dados relacional, CSV e JSON e fazer a sua integração na BD projetada (HoleInOne). Além disso, também se pretende descrever as fontes de dados utilizadas, caracterizando-as e descrevendo-as e justificar as ferramentas e tecnologia utilizadas.

A migração dos dados provenientes das fontes escolhidas é feita de forma direta através de um programa desenvolvido em Python, que faz uso da biblioteca `mysql.connector` de forma a que se possa comunicar com o SGBD do MySQL.

- Fonte Relacional (SQL):

A migração dos modelos de dados relacionais consiste em instruções SQL do tipo `INSERT INTO` diretamente incorporadas no código *python*, referentes às tabelas Carro e Aluguer, ou seja, os dados são inseridos diretamente através de instruções SQL estáticas enviadas para o servidor MySQL.

- Fonte CSV:

Para realizar a migração de dados a partir de uma fonte CSV, foram utilizados dois ficheiros CSV, mais especificamente `Cliente.csv` e `Telefones.csv`, ou seja, estes ficheiros contêm os dados dos Clientes e dos seus números telefónicos a inserir nas tabelas `Cliente` e `Telefones`. Cada linha dos ficheiros CSV representa um registo a inserir, onde os valores separados por vírgulas correspondem a cada coluna da tabela.

- Fonte JSON:

Já na realização da migração de dados a partir da fonte JSON, utilizámos também dois ficheiros, relativos a duas tabelas, sendo elas a tabela `Funcao` e `Funcionario`, nomeadamente os ficheiros `Funcao.json` e `Funcionario.json`. Estes ficheiros contêm listas de objetos em formato JSON, sendo cada um destes objetos um registo a adicionar à BD.

O programa é constituído por três funções principais, cada uma relativa a uma das fontes utilizadas, além, também, da função *main* e de uma função auxiliar usada na fonte JSON.

- `inserir_dados_relacionais(conn):`

Função que recebe como argumento a variável *conn* que representa a ligação à BD MySQL e executa comandos SQL previamente definidos para inserir registos diretamente nas tabelas Carro e Aluguer.

```
def inserir_dados_relacionais(conn):
    cursor = conn.cursor()
    try:
        # Inserir dados na tabela Carro
        cursor.execute("""
            INSERT INTO Carro (Nr_Carro, Stand, Ano, Nr_Lugares,
Custo) VALUES
                (1000, 'Vilamoura', 2017, 4, 25.0),
                (1001, 'Vilamoura', 2020, 4, 25.0),
                (1002, 'Vilamoura', 2016, 2, 15.0),
                (1003, 'Braga', 2013, 2, 15.0),
                (1004, 'Braga', 2015, 2, 15.0),
                (1005, 'Braga', 2021, 4, 25.0),
                (1006, 'Porto', 2017, 4, 25.0),
                (1007, 'Porto', 2013, 2, 15.0),
                (1008, 'Porto', 2016, 4, 25.0),
                (1009, 'Porto', 2009, 2, 15.0);
        """)

        # Inserir dados na tabela Aluguer
        cursor.execute("""
            INSERT INTO Aluguer (Nr_Aluguer, Data_Inicio, Data_Fim,
ID_Cliente, ID_Funcionario, Nr_Carro) VALUES
                (69, '2009-08-18', '2009-08-20', 106, 2, 1000),
                (70, '2009-08-23', '2009-08-26', 108, 2, 1005),
                (71, '2009-08-27', '2009-09-01', 101, 4, 1007),
                (72, '2009-09-07', '2009-09-14', 104, 3, 1005),
                (73, '2009-09-13', '2009-09-15', 107, 4, 1000),
                (74, '2009-09-20', '2009-09-29', 105, 5, 1003),
                (75, '2009-09-25', '2009-10-02', 101, 2, 1004),
                (76, '2009-10-01', '2009-10-04', 109, 1, 1001),
                (77, '2009-10-09', '2009-10-16', 100, 6, 1000),
                (78, '2009-10-14', '2009-10-20', 102, 1, 1002),
                (79, '2009-10-25', '2009-11-20', 103, 2, 1007);
        """)
```



```

        conn.commit()
        print("Dados relacionais (Carro e Aluguer) inseridos com
sucesso.")
    except Error as e:
        print(f"Erro ao inserir dados relacionais: {e}")
        conn.rollback()
    finally:
        cursor.close()

```

• `inserir_dados_csv(tabela, ficheiro_csv, colunas, conn):`

Função que recebe como argumento o nome da tabela na qual vai inserir os dados, o nome do ficheiro CSV onde se encontram os dados e uma lista de objetos, onde cada objeto irá servir para armazenar um determinado dado relativo a uma linha do ficheiro CSV e, mais uma vez, a variável que representa a ligação à BD. A função abre o ficheiro CSV e lê o mesmo linha a linha, onde, de cada vez que lê uma linha, armazena os seus valores na lista de objetos referida anteriormente e, por fim, usando o nome da tabela e os dados armazenados, envia a instrução SQL `INSERT INTO` para o servidor.

```

def inserir_dados_csv(nome_tabela, caminho_csv, colunas, conn):
    try:
        #conn = mysql.connector.connect(**config)
        cursor = conn.cursor()

        with open(caminho_csv, newline='', encoding='utf-8') as
csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                valores = [row[col] for col in colunas]
                placeholders = ', '.join(['%s'] * len(colunas))
                query = f"INSERT INTO {nome_tabela} ({',
'.join(colunas)}) VALUES ({placeholders})"
                cursor.execute(query, valores)

        conn.commit()
        print(f"Dados inseridos com sucesso na tabela {nome_tabela}")
    except Exception as e:
        print(f"Erro ao inserir dados na tabela {nome_tabela}: {e}")
    finally:
        try:

```

```

        if conn.is_connected():
            cursor.close()
    except:
        pass

```

• `inserir_dados_json(tabela, dados, conn):`

Função que recebe como argumento o nome da tabela onde se pretende inserir os dados, uma lista de dicionários *python* obtidos ao abrir o ficheiro JSON relativo à tabela e a variável que simboliza a ligação à base de dados. Através do nome da tabela e da lista de dicionários python obtidos do ficheiro JSON, tal e qual é feito na função `inserir_dados_CSV`, é criada uma instrução SQL do tipo `INSERT INTO`, onde a mesma é também enviada para o servidor para ser executada.

```

def inserir_dados_json(tabela, dados, conn):
    cursor = conn.cursor()

    if not dados:
        print(f"Nenhum dado para inserir na tabela {tabela}.")
        return

    # Preparar colunas e placeholders baseando-se nas keys do primeiro
dict
    colunas = dados[0].keys()
    placeholders = ", ".join(["%s"] * len(colunas))
    colunas_str = ", ".join(colunas)

    sql = f"INSERT INTO {tabela} ({colunas_str}) VALUES
({placeholders})"

    try:
        for linha in dados:
            valores = tuple(linha[col] for col in colunas)
            cursor.execute(sql, valores)

        conn.commit()
        print(f"Dados inseridos com sucesso na tabela {tabela}")
    except Error as e:
        print(f"Erro ao inserir dados na tabela {tabela}: {e}")
        conn.rollback()
    finally:
        cursor.close()

```

- `carregar_json(ficheiro):`

Recebe como argumento o nome do ficheiro JSON e devolve uma lista de dicionários python que será posteriormente usada na função `inserir_dados_json`.

- **Main:**

A função *main* é responsável por definir as configurações da ligação MySQL, os ficheiros CSV e JSON a usar, o formato da tabela em questão para as tabelas relativas à Fonte CSV e os nomes das tabelas a usar nas instruções SQL.

Por fim, chama a função `inserir_dados_CSV` para as tabelas Cliente e Telefones, repetindo o mesmo processo para a função `inserir_dados_json` nas tabelas Funcao e Funcionario e termina chamando a função `inserir_dados_relacionais` para inserir os dados nas tabelas restantes (Aluguer, Carro)

Justificação das tecnologias usadas:

- **python:** Linguagem de programação versátil e flexível, que permite fácil manipulação de ficheiros de dados, além de nos permitir conectar à BD projetada com muita facilidade.
- **mysql.connector:** Biblioteca oficial da Oracle para ligação ao MySQL, que permite ligar aplicações Python a base de dados MySQL, assim como executar *queries* ou outros tipos de instruções que permitam interagir com a BD.
- **JSON:** É um formato de dados representado por pares chave-valor, o que torna mais fácil a sua leitura e interpretação, além também de ser muito usado em serviços web e como forma de armazenar dados.
- **CSV:** Formato de ficheiro de texto simples, onde os dados são separados por vírgulas ou outro delimitador, o que permite representar tabelas de forma simples, o que acaba por ser útil para guardar dados referentes aos atributos das tabelas da BD projetada.

O código do programa de migração está na sua íntegra no [Anexo V](#).

Alterações relativas à Parte I

Durante o desenvolvimento da segunda fase do trabalho prático, o grupo sentiu necessidade de alterar duas coisas feitas na primeira fase.

A primeira delas foi a mudança dos tipos de dois atributos, em duas tabelas distintas (na modelação lógica e por sua vez na implementação física), o Salário na tabela Funcionario, e o Custo na tabela Carro. Ambos os atributos eram do tipo inteiro (INT), contudo, depois de alguma discussão, o grupo decidiu alterar esses atributos para o tipo decimal (DECIMAL (7,2) e DECIMAL (4,2) respetivamente). Esta mudança fez sentido para o grupo pois, como ambos os atributos representavam valores monetários, por uma questão de coerência com a realidade, estes valores deveriam ser decimais. Assim sendo, o suporte textual e as tabelas associadas às entidades que continham esses atributos foram devidamente atualizadas.

A segunda e última mudança foi a substituição completa dos requisitos de controlo. Quando o grupo chegou ao ponto **5.2**, onde era necessário codificar as permissões dos diferentes utilizadores, verificou que os requisitos de controlo não eram muito claros e descritivos, relativamente aos privilégios que cada tipo de utilizador deveria ter, por isso decidiu reformular todos os requisitos de controlo de modo a clarifica-los, e também para dar alguma variedade aos acessos dos vários utilizadores. Com esta alteração foi necessário atualizar as tabelas com os requisitos de controlo e tabela geral com todos os requisitos.

Conclusões e Trabalho Futuro

Parte I

Nesta primeira fase do desenvolvimento do trabalho proposto, foi preparada toda uma base sólida do Sistema de Base de Dados a ser implementado, tendo sido feita uma recolha de informação relativa à empresa para a qual será feita essa implementação, bem como alguns objetivos a serem alcançados (Definição do Sistema). Após esta contextualização do problema a ser resolvido, foram levantados os requisitos que explicam o funcionamento das atividades da empresa (Levantamento e Análise de Requisitos). Com toda esta informação, foi possível construir o diagrama Entidade-Relacionamento, com todas as entidades, relacionamentos e atributos identificados (Modelação Conceptual), a partir deste foi possível derivar para o modelo lógico do problema, aproximando-nos cada vez mais da implementação física em concreto. Tivemos também oportunidade de garantir a qualidade do modelo produzido através da normalização dos dados, bem como da criação de algumas *queries* que visavam corresponder a alguns dos requisitos de manipulação previamente levantados (Modelação Lógica).

Ao longo desta fase do trabalho tivemos oportunidade de colocar em prática o conhecimento adquirido ao longo das aulas teóricas e práticas, contudo a aplicação desse conhecimento trouxe alguns desafios, como por exemplo na questão do levantamento de requisitos, que foram refeitos algumas vezes, de forma a que se adequassem às decisões tomadas posteriormente. Apesar disto não ter representado nenhum constrangimento para o desenvolvimento do projeto, o grupo de trabalho compreendeu que esta não foi uma decisão muito prudente de ser tomada, contudo foi necessária. De uma forma geral, o trabalho fluiu bem, sendo as tarefas bem distribuídas e executadas com o tempo necessário. Ao longo das sessões de validação tivemos oportunidade de esclarecer alguns pontos do trabalho, para poder melhorá-los, e também esclarecemos algumas dúvidas referentes aos mesmos.

Numa próxima fase iremos abordar a parte física da Base de Dados que está a ser desenvolvida, utilizando as ferramentas do *MySQL*, implementando todo o código necessário para a criação das tabelas, bem como das *queries* que visam responder a algumas questões já definidas.

Contamos que o trabalho até agora desenvolvido, nos sirva como uma boa referência para esta próxima segunda fase.

Parte II

Nesta segunda e última fase do trabalho prático, foi desenvolvida a parte física do SGDB conceptualmente preparado na fase anterior. Foram codificadas todas as tabelas, *queries*, procedimentos, funções, *triggers*, etc. (Implementação Física), pedidas no enunciado do trabalho, bem como um programa em *python* responsável por migrar dados de três fontes distintas (modelo de dados relacional, JSON e CSV), e povoar a BD central desenvolvida (Implementação do sistema de migração de dados).

Mais uma vez, foi possível colocar em prática o conhecimento adquirido nas aulas teóricas e práticas, principalmente no que toca à linguagem SQL. Para que todos os *scripts* estivessem em sintonia, e a BD funcionasse corretamente, foi necessário bastante trabalho por parte do grupo.

Enfrentamos algumas adversidades no processo de desenvolvimento do programa de migração de dados, mas felizmente, depois de algumas tentativas e erros tudo ficou funcional.

De uma forma geral, esta segunda fase correu bem e dentro das previsões do grupo e conseguimos organizar o trabalho de forma a que pudéssemos realizar cada um dos pontos atempadamente.

Em suma, tendo em conta todo o trabalho feito nestas duas fases, achamos que conseguimos aplicar de forma correta e coerente todo o conhecimento que aprendemos ao longo do semestre, e por isso estamos contentes com o trabalho realizado.

Referências

- brModelo (<https://www.brmodeloweb.com/lang/pt-br/index.html>)
- RelaX (<https://dbis-uibk.github.io/relax/calc/local/uibk/local/0>)
- MySQL Workbench (<https://www.mysql.com/products/workbench/>)
- Connolly, T., Begg, C., **Database Systems: A Practical Approach to Design, Implementation, and Management**, 6th edition, Pearson, January, 2014.
- MySQL Documentation (<https://dev.mysql.com/doc/refman/8.4/en/data-types.html>)

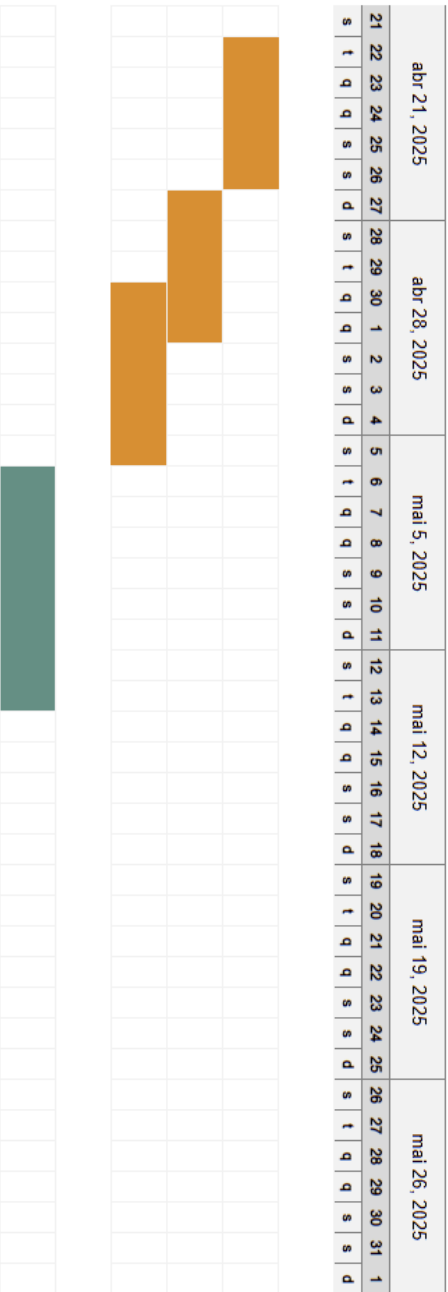
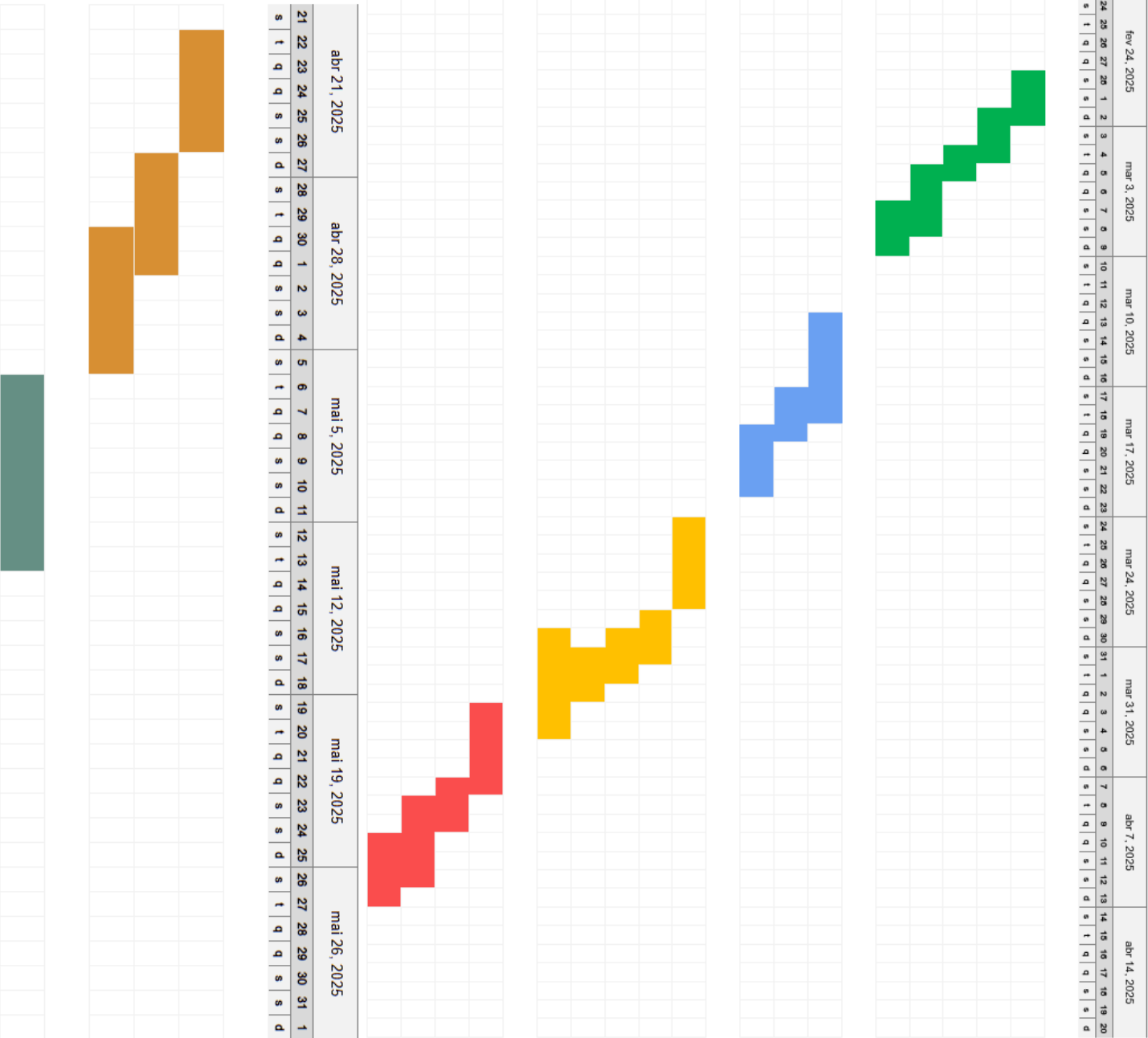
Lista de Siglas e Acrónimos

SBD	Sistema de Base de Dados
BD	Base de Dados
ER	Entidade – Relacionamento
MC	Modelo Conceptual
ML	Modelo Lógico
1FN	Primeira Forma Normal
2FN	Segunda Forma Normal
3FN	Terceira Forma Normal
SQL	<i>Structured Query Language</i>

Anexos

I. Diagramas de Gantt do plano de execução

Tarefa	Responsável	Início	Fim
Definição do Sistema			
Contexto de aplicação	Duarte Escalro	2/28/25	3/2/25
Motivação e Objetivos do trabalho	Luís Soares	3/2/25	3/4/25
Análise da Viabilidade do processo	João Rodrigues	3/4/25	3/5/25
Recursos e Equipe de trabalho	João Rodrigues	3/5/25	3/8/25
Plano de execução do projeto	Tiago Figueiredo	3/7/25	3/9/25
Definição dos Requisitos			
Método de Levantamento de requisitos	Tiago Figueiredo	3/13/25	3/18/25
Organização dos Requisitos levantados	Duarte Escalro	3/17/25	3/19/25
Análise e Validação dos Requisitos	Luís Soares	3/19/25	3/22/25
Modelação Conceptual			
Abordagem da Modelação	Luís Soares	3/24/25	3/28/25
Identificação e Caracterização das Entidades	Duarte Escalro	3/29/25	3/31/25
Identificação e Caracterização dos Relacionamentos	João Rodrigues	3/30/25	4/1/25
Identificação e Caracterização dos Atributos	Duarte Escalro	3/31/25	4/2/25
Apresentação e Explicação do Modelo produzido	Tiago Figueiredo	3/30/25	4/4/25
Modelação Lógica			
Construção e Validação do Modelo Lógico	João Rodrigues	4/3/25	4/7/25
Apresentação e Explicação do Modelo Lógico	Tiago Figueiredo	4/7/25	4/9/25
Normalização de Dados	Luís Soares	4/8/25	4/12/25
Validação do Modelo com Interrogações do Utilizador	Tiago Figueiredo	4/10/25	4/13/25



II. Tabelas de requisitos

Hole in One

Processo de desenvolvimento do Sistema de Base de Dados

Levantamento de Requisitos

Recolha geral dos requisitos

Nr	Descrição	Data	Fonte	Analista
1	O cliente é caracterizado por ter um identificador único, nome, data de nascimento, telefone, email, morada.	17/03/2025	Reunião Geral	Luís Soares
2	O funcionário é caracterizado por ter um identificador único, nome, telefone, email da empresa, morada, local de trabalho, salário.	17/03/2025	Reunião Geral	Duarte Escairo
3	Um aluguer é caracterizado por ter um número de aluguer único, data de início e fim do aluguer.	17/03/2025	Reunião Geral	Luís Soares
4	O carro é caracterizado pelo número único do veículo, stand onde se encontra, ano de fabrico, número de lugares e custo diário do aluguer.	17/03/2025	Reunião Geral	João Pedro Rodrigues
5	O fundador da empresa tem permissão para acrescentar, modificar, apagar e ler todos os registos do sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
6	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos clientes no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
7	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos alugueres no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
8	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos carros no sistema.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
9	Os dados de um cliente devem poder ser acedidos através do identificador do cliente.	18/03/2025	Reunião com o Sr. Esbelto	Luís Soares
10	Os dados de um funcionário devem poder ser acedidos através do identificador do funcionário.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
11	Os dados de um aluguer devem poder ser acedidos através do número de aluguer.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
12	Um cliente faz um registo de aluguer de cada vez.	17/03/2025	Reunião Geral	Duarte Escairo
13	O cliente só pode alugar um carro.	17/03/2025	Reunião Geral	Duarte Escairo
14	O funcionário trata do registo do aluguer.	17/03/2025	Reunião Geral	Duarte Escairo
15	É possível listar todos identificadores de cliente, e o seu respetivo nome.	17/03/2025	Reunião Geral	Duarte Escairo
16	Um funcionário exerce uma função.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
17	Todos os secretários tem permissão para acrescentar e ler registos de clientes no sistema.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
18	Todos os secretários tem permissão para acrescentar e ler registos de alugueres no sistema.	18/03/2025	Reunião com o Sr. Esbelto	Duarte Escairo
19	É possível listar os números de alugueres que um determinado cliente fez, bem como o seu identificador e a data de início desses alugueres.	17/03/2025	Reunião Geral	Tiago Figueiredo
20	É possível listar os nomes de todos os clientes que foram atendidos por um determinado funcionário.	17/03/2025	Reunião Geral	Tiago Figueiredo
21	É possível possível listar todos os números dos carros que foram alugados por um funcionário que exerce uma determinada função.	17/03/2025	Reunião Geral	Luís Soares
22	É possível registar novos clientes.	17/03/2025	Reunião Geral	João Pedro Rodrigues
23	É possível adicionar novos funcionários.	17/03/2025	Reunião Geral	João Pedro Rodrigues
24	A função é caracterizada por um identificador único e pela designação.	17/03/2025	Reunião Geral	Luís Soares
25	Um cliente pode ter vários telefones associados.	17/03/2025	Reunião Geral	Tiago Figueiredo
26	Uma morada é composta por Rua e Cidade.	17/03/2025	Reunião Geral	Duarte Escairo
27	Apenas o fundador tem permissão para interagir com os registos dos funcionários.	18/03/2025	Reunião com o Sr. Esbelto	Duarte Escairo

Hole in One

Processo de desenvolvimento do Sistema de Base de Dados

Levantamento de Requisitos

Requisitos de Descrição



Nr	Descrição	Data	Fonte	Analista
RD1	O cliente é caracterizado por ter um identificador único, nome, data de nascimento, telefone, email, morada.	17/03/2025	Reunião Geral	Luís Soares
RD2	O funcionário é caracterizado por ter um identificador único, nome, telefone, email da empresa, morada, local de trabalho, salário.	17/03/2025	Reunião Geral	Duarte Escairo
RD3	Um aluguer é caracterizado por ter um número de aluguer único, data de início e fim do aluguer.	17/03/2025	Reunião Geral	Luís Soares
RD4	O carro é caracterizado pelo número único do veículo, stand onde se encontra, ano de fabrico, número de lugares e custo diário do aluguer.	17/03/2025	Reunião Geral	João Pedro Rodrigues
RD5	Um cliente faz um registo de aluguer de cada vez.	17/03/2025	Reunião Geral	Duarte Escairo
RD6	O cliente só pode alugar um carro.	17/03/2025	Reunião Geral	Duarte Escairo
RD7	O funcionário trata do registo do aluguer.	17/03/2025	Reunião Geral	Duarte Escairo
RD8	Um funcionário exerce uma função.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RD9	A função é caracterizada por um identificador único e pela designação.	17/03/2025	Reunião Geral	Luís Soares
RD10	Um cliente pode ter vários telefones associados.	17/03/2025	Reunião Geral	Tiago Figueiredo
RD11	Uma morada é composta por Rua e Cidade.	17/03/2025	Reunião Geral	Duarte Escairo

Hole in One

Processo de desenvolvimento do Sistema de Base de Dados

Levantamento de Requisitos

Requisitos de Manipulação



Nr	Descrição	Data	Fonte	Analista
RM1	Os dados de um cliente devem poder ser acedidos através do identificador do cliente.	18/03/2025	Reunião com o Sr. Esbelto	Luís Soares
RM2	Os dados de um funcionário devem poder ser acedidos através do identificador do funcionário.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RM3	Os dados de um aluguer devem poder ser acedidos através do número de aluguer.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RM4	É possível listar todos identificadores de cliente, e o seu respetivo nome.	17/03/2025	Reunião Geral	Duarte Escairo
RM5	É possível listar os números de alugueres que um determinado cliente fez, bem como o seu identificador e a data de início desses alugueres.	17/03/2025	Reunião Geral	Tiago Figueiredo
RM6	É possível listar os nomes de todos os clientes que foram atendidos por um determinado funcionário.	17/03/2025	Reunião Geral	Tiago Figueiredo
RM7	É possível listar todos os números dos carros que foram alugados por um funcionário que exerce uma determinada função.	17/03/2025	Reunião Geral	Luís Soares
RM8	É possível registar novos clientes.	17/03/2025	Reunião Geral	João Pedro Rodrigues
RM9	É possível adicionar novos funcionários.	17/03/2025	Reunião Geral	João Pedro Rodrigues

Hole in One

Processo de desenvolvimento do Sistema de Base de Dados

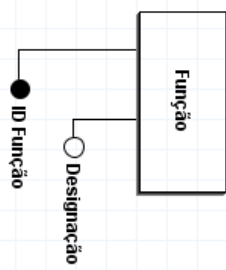
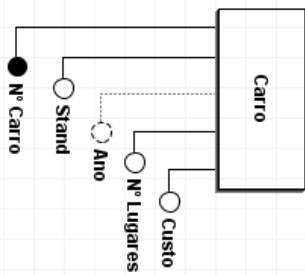
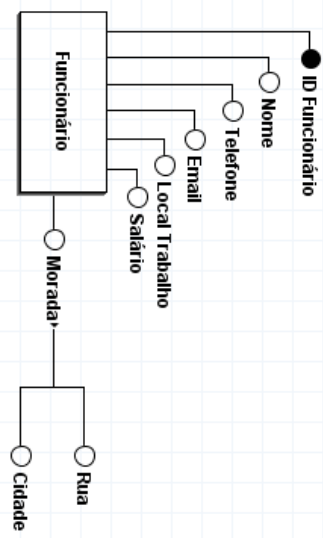
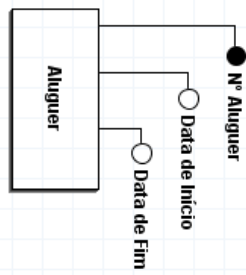
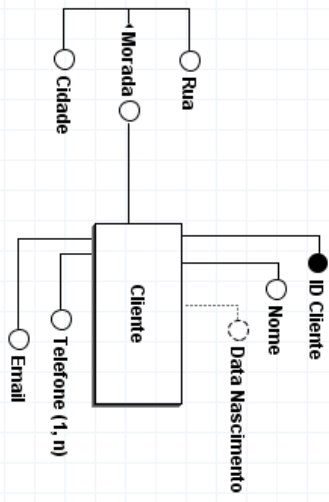
Levantamento de Requisitos

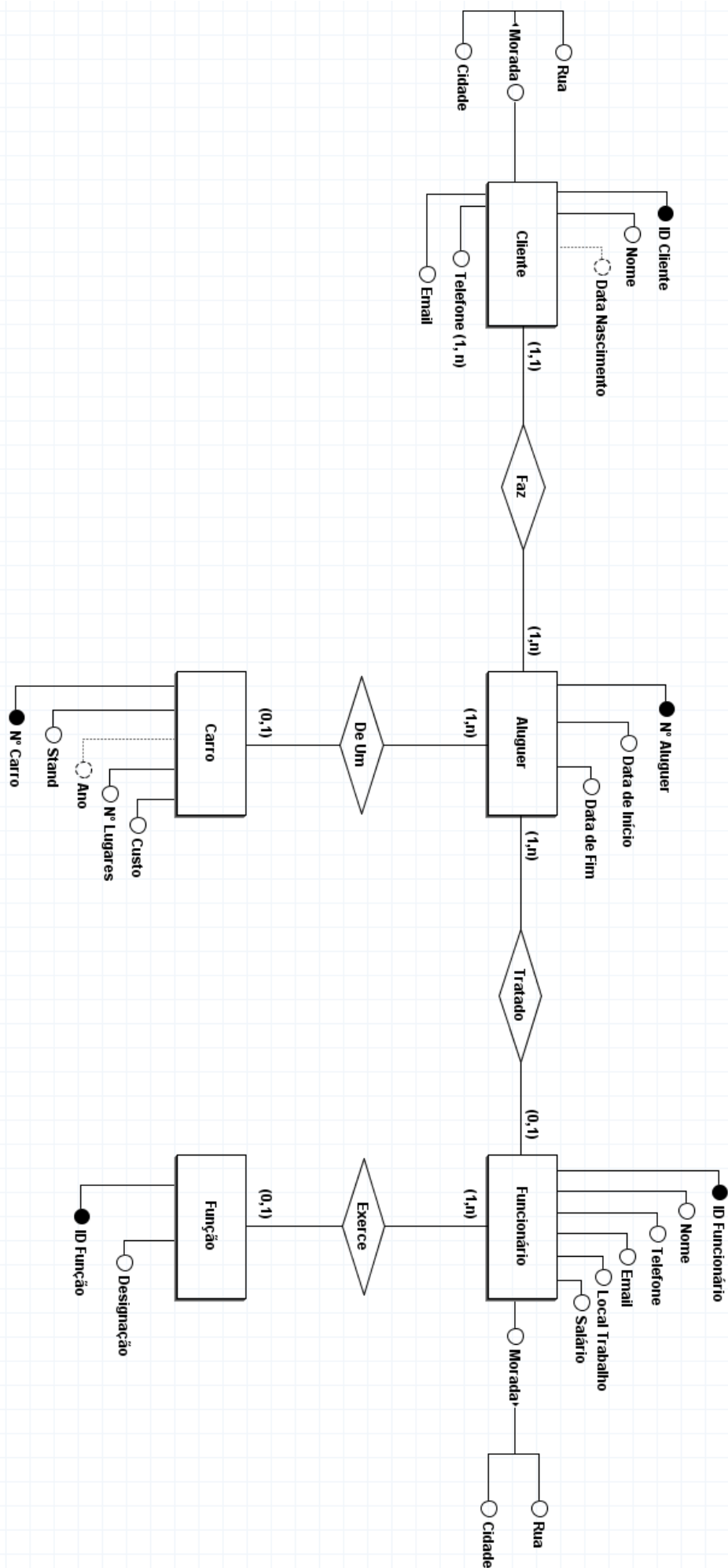
Requisitos de Controlo



Nr	Descrição	Data	Fonte	Analista
RC1	O fundador da empresa tem permissão para acrescentar, modificar, apagar e ler todos os registos do sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC2	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos clientes no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC3	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos alugueres no sistema.	17/03/2025	Reunião Geral	Tiago Figueiredo
RC4	Todos os gerente devem ter permissão para acrescentar, modificar, apagar e ler os registos dos carros no sistema.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RC5	Todos os secretários tem permissão para acrescentar e ler registos de clientes no sistema.	18/03/2025	Reunião com o Sr. Esbelto	João Pedro Rodrigues
RC6	Todos os secretários tem permissão para acrescentar e ler registos de alugueres no sistema.	18/03/2025	Reunião com o Sr. Esbelto	Duarte Escairo
RC7	Apenas o fundador tem permissão para interagir com os registos dos funcionários.	18/03/2025	Reunião com o Sr. Esbelto	Duarte Escairo

III. Diagramas Conceptuais





IV. *Input para o RelaX*

group:hole_in_one

```
Cliente = {ID_Cliente, Nome, Data_Nascimento, Email, Rua, Cidade
  100, Tiago, 2005-12-07, 'tiago@gmail.com', Rua_Tiago, Barcelos
  101, Duarte, 2005-12-27, 'duarte@gmail.com', Rua_Duarte, Barcelos
  102, Luis, 2005-10-14, 'luis@gmail.com', Rua_Luis, Vila_Verde
  103, Joao, 2004-02-10, 'joao@gmail.com', Rua_Joao, Braga
  104, Ana, 2000-01-01, 'ana@gmail.com', Rua_Ana, Albufeira
  105, Maria, 1999-04-07, 'maria@gmail.com', Rua_Maria, Lisboa
  106, Joaquim, 1951-04-14, 'joaquim@gmail.com', Rua_Joaquim, Porto
  107, Carlos, 1990-06-06, 'carlos@gmail.com', Rua_Carlos, Ponte_Lima
  108, Jose, 2005-11-05, 'jose@gmail.com', Rua_Jose, Lisboa
  109, Mariana, 2004-10-15, 'mariana@gmail.com', Rua_Mariana, Braga
  110, Josefina, 1970-07-20, 'josefina@gmail.com', Rua_Josefina,
Braga
}
```

```
Telefones = {Telefone, ID_Cliente
  912561490, 100
  945189011, 102
  938900133, 103
  956566570, 104
  978154351, 105
  914467100, 106
  945674411, 107
  900981233, 108
  967674412, 109
  911155567, 110
  922278874, 100
}
```

```
Aluguer = {Num_Aluguer, Data_Inicio, Data_Fim, ID_Cliente,
ID_Funcionario, Num_Carro
```

```

69, 2009-08-18, 2009-08-20, 106, 2, 1000
70, 2009-08-23, 2009-08-26, 108, 2, 1005
71, 2009-08-27, 2009-09-01, 101, 4, 1007
72, 2009-09-07, 2009-09-14, 104, 3, 1005
73, 2009-09-13, 2009-09-15, 107, 4, 1000
74, 2009-09-20, 2009-09-29, 105, 5, 1003
75, 2009-09-25, 2009-10-02, 101, 2, 1004
76, 2009-10-01, 2009-10-04, 109, 1, 1001
77, 2009-10-09, 2009-10-16, 100, 6, 1000
78, 2009-10-14, 2009-10-20, 102, 1, 1002
79, 2009-10-25, 2009-11-20, 103, 2, 1007
}

```

```

Funcionario = {ID_Funcionario, Nome, Telefone, Email, Local_Trabalho,
Rua, Cidade, Salario, ID_Funcao
    1, Orlando,911177464,'orlando@hole.pt', Vilamoura, Rua_Orlando,
Vilamoura, 2000, 1
    2, Beatriz,956894531,'beatriz@hole.pt', Vilamoura, Rua_Beatriz,
Albufeira, 1000, 3
    3, Jose,912145743,'jose@hole.pt', Vilamoura, Rua_Jose,
Quarteira, 1500, 2
    4, Iara,976494522,'iara@hole.pt',Porto, Rua_Iara, Maia, 1000, 3
    5, Rogerio,900748741,'rogerio@hole.pt', Porto, Rua_Rogerio,
Porto, 1500, 2
    6, Ines,912345678,'ines@hole.pt',Braga, Rua_Ines, Barcelos,
1000, 3
    7, Edgar,9254448282,'edgar@hole.pt', Braga, Rua_Edgar,
Carvalhal, 1500, 2
}

```

```

Carro = {Num_Carro, Stand, Ano, Num_Lugares, Custo
    1000, Vilamoura, 2017, 4, 25
    1001, Vilamoura, 2020, 4, 25
    1002, Vilamoura, 2016, 2, 15
    1003, Braga, 2013, 2, 15
    1004, Braga, 2015, 2, 15
    1005, Braga, 2021, 4, 25
    1006, Porto, 2017, 4, 25
    1007, Porto, 2013, 2, 15
    1008, Porto, 2016, 4, 25
    1009, Porto, 2009, 2, 15
}

```

```
}
```

```
Funcao = {ID_Funcao, Designacao
```

```
1, Dono
```

```
2, Gerente
```

```
3, Secretaria
```

```
}
```

V. Código do programa de migração

```
import json
import mysql.connector
import csv
import os
from mysql.connector import Error

def carregar_json(nome_ficheiro):
    with open(nome_ficheiro, 'r', encoding='utf-8') as jsonfile:
        return json.load(jsonfile)

#Função para inserir dados de um Json numa tabela
def inserir_dados_json(tabela, dados, conn):
    cursor = conn.cursor()

    if not dados:
        print(f"Nenhum dado para inserir na tabela {tabela}.")
        return

    # Preparar colunas e placeholders baseando-se nas keys do primeiro
    dict
    colunas = dados[0].keys()
    placeholders = ", ".join(["%s"] * len(colunas))
    colunas_str = ", ".join(colunas)

    sql = f"INSERT INTO {tabela} ({colunas_str}) VALUES
({placeholders})"

    try:
        for linha in dados:
            valores = tuple(linha[col] for col in colunas)
            cursor.execute(sql, valores)
        conn.commit()
        print(f"Dados inseridos com sucesso na tabela {tabela}")
```

```

except Error as e:
    print(f"Erro ao inserir dados na tabela {tabela}: {e}")
    conn.rollback()
finally:
    cursor.close()

#Função para inserir dados de um CSV numa tabela
def inserir_dados_csv(nome_tabela, caminho_csv, colunas, conn):
    try:
        #conn = mysql.connector.connect(**config)
        cursor = conn.cursor()

        with open(caminho_csv, newline='', encoding='utf-8') as
csvfile:
            reader = csv.DictReader(csvfile)
            for row in reader:
                valores = [row[col] for col in colunas]
                placeholders = ', '.join(['%s'] * len(colunas))
                query = f"INSERT INTO {nome_tabela} ({',
'.join(colunas)}) VALUES ({placeholders})"
                cursor.execute(query, valores)

            conn.commit()
            print(f"Dados inseridos com sucesso na tabela {nome_tabela}")
    except Exception as e:
        print(f"Erro ao inserir dados na tabela {nome_tabela}: {e}")
        conn.rollback()
    finally:
        try:
            if conn.is_connected():
                cursor.close()
        except:
            pass

# Função para inserir dados relacionais (Carro e Aluguer)
def inserir_dados_relacionais(conn):
    cursor = conn.cursor()
    try:
        # Inserir dados na tabela Carro
        cursor.execute("""

```

```

        INSERT INTO Carro (Nr_Carro, Stand, Ano, Nr_Lugares,
Custo) VALUES
        (1000, 'Vilamoura', 2017, 4, 25.0),
        (1001, 'Vilamoura', 2020, 4, 25.0),
        (1002, 'Vilamoura', 2016, 2, 15.0),
        (1003, 'Braga', 2013, 2, 15.0),
        (1004, 'Braga', 2015, 2, 15.0),
        (1005, 'Braga', 2021, 4, 25.0),
        (1006, 'Porto', 2017, 4, 25.0),
        (1007, 'Porto', 2013, 2, 15.0),
        (1008, 'Porto', 2016, 4, 25.0),
        (1009, 'Porto', 2009, 2, 15.0);
    """

    # Inserir dados na tabela Aluguer
    cursor.execute("""
        INSERT INTO Aluguer (Nr_Aluguer, Data_Inicio, Data_Fim,
ID_Cliente, ID_Funcionario, Nr_Carro) VALUES
        (69, '2009-08-18', '2009-08-20', 106, 2, 1000),
        (70, '2009-08-23', '2009-08-26', 108, 2, 1005),
        (71, '2009-08-27', '2009-09-01', 101, 4, 1007),
        (72, '2009-09-07', '2009-09-14', 104, 3, 1005),
        (73, '2009-09-13', '2009-09-15', 107, 4, 1000),
        (74, '2009-09-20', '2009-09-29', 105, 5, 1003),
        (75, '2009-09-25', '2009-10-02', 101, 2, 1004),
        (76, '2009-10-01', '2009-10-04', 109, 1, 1001),
        (77, '2009-10-09', '2009-10-16', 100, 6, 1000),
        (78, '2009-10-14', '2009-10-20', 102, 1, 1002),
        (79, '2009-10-25', '2009-11-20', 103, 2, 1007);
    """)

    conn.commit()

    print("Dados relacionais (Carro e Aluguer) inseridos com
sucesso.")

    except Error as e:
        print(f"Erro ao inserir dados relacionais: {e}")
        conn.rollback()

    finally:
        cursor.close()

```



```

def main():

    # Configurações da ligação MySQL
    config = {
        'host': '*****',
        'user': '*****',
        'password': '*****',
        'database': 'HoleInOne',
    }

    # Tabelas para inserir dados CSV
    tabelas_CSV = {
        "Cliente": {
            "csv": "Cliente.csv",
            "colunas": ["ID_Cliente", "Nome", "Data_Nascimento",
"Email", "Rua", "Cidade"]
        },
        "Telefones": {
            "csv": "Telefones.csv",
            "colunas": ["Telefone", "ID_Cliente"]
        }
    }

    # Tabelas para inserir dados JSON
    tabelas_JSON = {
        "Funcao": "Funcao.json",
        "Funcionario": "Funcionario.json",
    }

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

    # Inserir dados CSV
    for tabela, dados in tabelas_CSV.items():
        if os.path.exists(dados["csv"]):
            inserir_dados_csv(tabela, dados["csv"], dados["colunas"],
conn)
        else:
            print(f"Ficheiro {dados['csv']} não encontrado.")

    # Inserir dados JSON
    try:

```

```

conn = mysql.connector.connect(**config)
if conn.is_connected():

    for tabela, arquivo in tabelas_JSON.items():
        dados = carregar_json(arquivo)
        inserir_dados_json(tabela, dados, conn)
except Error as e:
    print(f"Erro na ligação ao MySQL: {e}")
finally:
    if conn.is_connected():
        # Inserir dados relacionais (Carro e Aluguer)
        inserir_dados_relacionais(conn)
        conn.close()

main()

```