



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Base de Dados

Ano Letivo de 2024/2025

AgroAuto

Ana Beatriz Freitas(106853) João Paulo Azevedo(107367) Matilde Teixeira(106876)
Luís Miguel Coelho(106843)

2 de junho de 2025

BD

Data de Receção	
Responsável	
Avaliação	
Observações	

AgroAuto

Ana Beatriz Freitas(106853) João Paulo Azevedo(107367) Matilde Teixeira(106876)
Luís Miguel Coelho(106843)

2 de junho de 2025

Resumo

O desenvolvimento do sistema de bases de dados teve como objetivo principal a modelação e implementação de uma solução eficaz para a gestão de alugueres de tratores agrícolas. Este projeto dividiu-se em duas grandes fases: a conceção lógica e a implementação física do sistema.

A primeira etapa consistiu na recolha e análise de requisitos, que permitiu identificar os principais intervenientes e entidades do sistema, como clientes, tratores, funcionários, stands e alugueres. Com base nesta análise, foi desenvolvido um modelo conceptual utilizando diagramas entidade-relacionamento (*ER*), que representaram as entidades, os seus atributos e os relacionamentos entre elas.

Este modelo conceptual foi posteriormente transformado num modelo relacional, onde se definiram as tabelas, as chaves primárias e estrangeiras, assegurando a normalização e a integridade referencial dos dados. Para validar o modelo, foram formuladas interrogações relevantes, traduzidas em álgebra relacional, e acompanhadas pelas respectivas árvores de demonstração. Esta análise permitiu avaliar a adequação do modelo às necessidades funcionais do sistema.

Adicionalmente, foram identificadas oportunidades de otimização através da análise das interrogações mais frequentes, bem como a necessidade de índices e atributos derivados para melhorar o desempenho do sistema.

Com base no modelo relacional validado, procedeu-se à implementação física da base de dados no sistema de gestão *MySQL*. Nesta fase, foram criadas todas as tabelas definidas no esquema lógico, respeitando os tipos de dados e as restrições identificadas.

Foram também definidos utilizadores com diferentes perfis e permissões, representando cenários reais de acesso e gestão de dados. O povoamento inicial da base de dados foi realizado com registo exemplificativos para cada tabela.

O espaço ocupado pela base de dados foi estimado com base nos tipos de dados utilizados, e foi calculada uma taxa de crescimento anual, considerando a projeção realista de utilização do sistema ao longo do tempo.

Além disso, foram desenvolvidas várias *views SQL* para facilitar a consulta de dados relevantes pelos utilizadores, bem como um conjunto de *queries* representativas das funcionalidades esperadas. Procedeu-se ainda à criação de índices para optimizar consultas críticas e acelerar o desempenho do sistema.

A implementação física incluiu também procedimentos armazenados, funções e *triggers* para automatizar tarefas e reforçar a integridade e a eficiência da base de dados.

Uma componente fundamental deste projeto foi a integração de dados provenientes de fontes externas. Foi desenvolvido um sistema de migração que permitiu importar dados a partir de

uma base de dados *PostgreSQL*, de ficheiros CSV e ficheiros JSON, para os respetivos stands.

Cada tipo de ficheiro foi tratado com o devido cuidado, assegurando a correspondência entre os dados de origem e o modelo da base de dados central. O processo de migração foi implementado em *Python*, utilizando bibliotecas específicas para cada tipo de origem de dados, garantindo a inserção correta e eficiente dos registos.

Palavras-Chave: Alugueres, Gestão de alugueres, Tratores, Base de dados, Requisitos, Modelo Entidade-Relacionamento, *Queries*, *MySQL*, Povoamento, *Stored procedure*, *View*.

Índice

1 Definição do Sistema	1
1.1 Contexto de aplicação	1
1.2 Motivação e Objetivos	2
1.2.1 Motivação	2
1.2.2 Objetivos	3
1.3 Análise da Viabilidade do processo	3
1.4 Recursos e Equipa de Trabalho	4
1.4.1 Equipa de trabalho	4
1.4.2 Recursos Humanos	5
1.4.3 Recursos Materiais	5
1.5 Plano de execução do projeto	5
2 Levantamento e Análise de Requisitos	8
2.1 Método de Levantamento e de Análise dos Requisitos Adotados	8
2.2 Organização dos Requisitos Levantados	10
2.2.1 Requisitos de descrição	10
2.2.2 Requisitos de manipulação	11
2.2.3 Requisitos de controlo	13
2.3 Análise e Validação Geral dos Requisitos	16
3 Modelação Conceptual	17
3.1 Apresentação da Abordagem de Modelação Realizada	17
3.2 Identificação e Caracterização das Entidades	17
3.3 Identificação e Caracterização dos Relacionamentos	19
3.4 Identificação e Caracterização dos Atributos das Entidades e dos Relacionamentos.	21
3.5 Apresentação e Explicação do Diagrama ER Produzido	23
4 Modelação Lógica	25
4.1 Construção e Validação do Modelo de Dados Lógico	25
4.2 Apresentação e Explicação do Modelo Lógico Produzido	25
4.3 Normalização de Dados	26
4.4 Validação do Modelo com Interrogações do Utilizador	27
5 Revisões e Melhorias na Primeira Fase do Projeto	35

6 Implementação Física	38
6.1 Apresentação e explicação da base de dados implementada	38
6.2 Criação de utilizadores da base de dados	42
6.3 Povoamento da base de dados	45
6.4 Cálculo do espaço da base de dados (inicial e taxa de crescimento anual)	48
6.5 Definição e caracterização de vistas de utilização em SQL	52
6.5.1 MarcaMaisAlugada - Marca de tratores mais alugada no final de cada ano.	52
6.5.2 TratoresDisponíveis - lista de tratores disponíveis para alugar	53
6.6 Tradução das interrogações do utilizador para SQL	54
6.7 Indexação do Sistema de Dados	59
6.8 Implementação de procedimentos, funções e gatilhos	60
7 Implementação do sistema de migração dados	68
7.1 Migração de CSV	70
7.2 Migração de <i>PostgreSQL</i>	71
7.3 Migração de JSON	73
8 Conclusões e Trabalho Futuro	75
Lista de Siglas e Acrónimos	78
Anexos	79
Anexo 1	79
Anexo 2	80
Anexo 3	81
Anexo 4	82
Anexo 5	83
Anexo 6	84
Anexo 7	85

Lista de Figuras

1.1	Diagrama de GANTT - expectativa	6
1.2	Diagrama de GANTT - realidade	6
2.1	Ata da Reunião Nº04	14
3.1	Relacionamento binário Cliente - Aluguer	19
3.2	Relacionamento binário Funcionário - Aluguer	19
3.3	Relacionamento binário Funcionário - Stand	20
3.4	Relacionamento binário Aluguer - Trator	20
3.5	Relacionamento binário Trator - Stand	21
3.6	Diagrama ER do Sistema de Gestão de Alugueres de Tratores	24
4.1	Modelo Lógico do Sistema de Gestão de Alugueres de Tratores	26
4.2	Árvore de demonstração Q1	28
4.3	Árvore de demonstração Q2	29
4.4	Árvore de demonstração Q3	30
4.5	Árvore de demonstração Q4	31
4.6	Árvore de demonstração Q5	33
6.1	Tabela Cliente SQL	39
6.2	Tabela Stand SQL	40
6.3	Tabela Trator SQL	40
6.4	Tabela Funcionário SQL	41
6.5	Tabela Aluguer SQL	41
6.6	<i>Role</i> Funcionário	42
6.7	Criação de utilizadores e atribuição da <i>role</i> Funcionário a estes	43
6.8	Atribuição de permissões ao <i>role</i> Funcionário	43
6.9	<i>Role</i> administrador	44
6.10	Atribuição de permissões ao <i>role</i> administrador	44
6.11	Povoamento de Cliente	46
6.12	Povoamento de Stand	46
6.13	Povoamento de Funcionário	47
6.14	Povoamento de Trator	47
6.15	Povoamento de Aluguer	48
6.16	<i>View</i> MarcaMaisAlugada	53
6.17	<i>View</i> TratoresDisponiveis	53
6.18	Query 1 SQL	55

6.19	Query 2 SQL	56
6.20	Query 3 SQL	57
6.21	Query 4 SQL	58
6.22	Query 5 SQL	59
6.23	Definição de Índices no Sistema AgroAuto	60
6.24	Procedure ClientesMaisAtivos	61
6.25	Procedure RegistrosAlugueresFuncionario	61
6.26	Procedure LucroTotalAlugueres	62
6.27	Procedure HistoricoAlugueresCliente	63
6.28	Procedure registrarNovoAluguer	64
6.29	<i>Trigger</i> setTratorAlugado	65
6.30	<i>Trigger</i> setTratorLivre	66
6.31	Função calcularCustoAluguer	67
7.1	Programa de Migração de <i>Python</i>	69
7.2	Conexões MySQL de <i>PostgreSQL</i>	70
7.3	Migração de Funcionários de CSV	71
7.4	Migração de Tratores de <i>PostgreSQL</i>	72
7.5	Migração de Alugueres de JSON	73
8.1	Ata da Primeira Reunião	80
8.2	Formulário de Registo de um Cliente	81
8.3	Formulário de Registo de um Aluguer	82
8.4	Funções de migração de CSV	83
8.5	Funções de migração de PostgreSQL	84
8.6	Funções de migração de JSON	85

Lista de Tabelas

2.1	Documento dos Requisitos de Descrição	11
2.2	Dicionário dos Requisitos de Manipulação	13
2.3	Dicionário dos Requisitos de Controlo	15
3.1	Caracterização das entidades	18
3.2	Caracterização dos Relacionamentos	21
3.3	Caracterização dos atributos das entidades do sistema	22
6.1	Tamanho inicial da tabela Cliente	49
6.2	Tamanho inicial da tabela Aluguer	49
6.3	Tamanho inicial da tabela Trator	50
6.4	Tamanho inicial da tabela Funcionário	50
6.5	Tamanho inicial da tabela Stand	50
6.6	Tamanho estimado por tabela	51
6.7	Crescimento anual estimado da base de dados	51
8.1	Tabela de Discrepâncias Temporais	79

1 Definição do Sistema

1.1 Contexto de aplicação

O senhor Artur Fernandes Pires, de 48 anos, nasceu e cresceu em Olivença, uma região do Alentejo marcada pela forte presença da agricultura. Viveu parte da sua vida numa zona rural, com os pais, numa época onde predominava o trabalho árduo no campo. Foi neste contexto que Artur Pires sempre foi habituado a manusear o trator que o pai, Francisco Pires, tanto trabalhou para adquirir. Devido ao contacto próximo com este, Artur Pires aprendeu a valorizar o alívio de trabalho que a utilização do trator lhes permitia. Assim, motivado pelo contacto permanente e tão próximo com o campo, o Sr.Pires apercebeu-se, desde tenra idade, que gostava que a agricultura fosse a sua principal fonte de rendimento.

O seu pai, conhecido por todos da aldeia, era várias vezes solicitado por outros lavradores com o intuito de usufruírem do seu trator. Artur Pires, consciente da realidade que o rodeava e da escassez de maquinaria agrícola no contexto em que se inseria, percebeu que seria lucrativo tomar partido desta carência dos restantes habitantes da vila.

Com apenas 27 anos, o Sr.Pires fundou a AgroAuto, um estabelecimento dedicado ao aluguer de tratores e máquinas agrícolas, sediado no centro da vila de Olivença. O seu negócio local rapidamente apresentou grandes lucros e, motivado pela elevada procura de agricultores de vários distritos do país, percebeu a necessidade e vantagem de expansão. Após uma cuidadosa ponderação, decidiu abrir outro stand numa zona do Norte do país, elegendo Lamego, no Douro. Depois de 14 anos de negócio, o Sr.Pires alargou a sua empresa para uma terceira localização, marcando assim presença no sul, em Sagres.

Atualmente, a AgroAuto é uma empresa de referência no setor de aluguer de tratores e máquinas agrícolas, com sedes em Olivença, Lamego e Sagres. Desde a sua fundação, o Sr.Pires tem como principal missão facilitar a vida no campo, oferecendo soluções eficientes e personalizadas para cada consumidor. Com, atualmente, uma equipa de três funcionários distribuídos pelos diferentes stands, a AgroAuto garante um serviço de qualidade, controlando a manutenção dos equipamentos e promovendo a sua oferta nas respetivas regiões, cada um com o seu próprio método de registo e armazenamento de informações.

1.2 Motivação e Objetivos

1.2.1 Motivação

Desde a fundação da AgroAuto, o Sr. Artur Pires tem acompanhado de perto o crescimento da empresa, que se expandiu com a abertura de vários stands em diferentes regiões do país. Este crescimento trouxe consigo um aumento significativo no número de clientes e, consequentemente, da complexidade da gestão dos alugueres de tratores e equipamentos agrícolas.

A decisão de modernizar a gestão dos alugueres levou à necessidade de compreender e integrar os dados provenientes dos três stands da AgroAuto. Cada um destes stands desenvolveu, ao longo do tempo, métodos próprios para registar e organizar a sua informação, resultando em fontes de dados distintas e heterogéneas.

O stand de Olivença foi o primeiro a adotar uma solução digital rudimentar, tendo atualmente uma base de dados relacional local, que organiza a informação de forma estruturada e já parcialmente normalizada. O stand de Lamego, por sua vez, adotou uma abordagem mais simples, baseando-se na utilização de ficheiros CSV para registar os dados operacionais. Cada tipo de informação, como clientes, tratores, alugueres, é armazenado em ficheiros distintos, o que facilita a leitura, mas carece de ligação entre os dados e apresenta riscos de inconsistência ou redundância. O mais recente stand, localizado em Sagres, utiliza um sistema com registo de dados em ficheiros JSON. Esta estrutura permite maior flexibilidade e adaptação à digitalização, mas a sua forma hierárquica requer um processo de extração mais detalhado para integração numa base relacional.

Perante este cenário, tornou-se evidente a necessidade de uma solução que modernizasse e otimizasse os processos internos. Assim, o Sr.Pires decidiu implementar um sistema de base de dados central que permite a gestão de alugueres, com o objetivo de centralizar as operações dos diferentes stands, garantir um registo fiável das informações e automatizar tarefas repetitivas.

Serão exportados para a base de dados central todos os dados essenciais à gestão dos alugueres da AgroAuto, incluindo informações sobre clientes, adquiridas no registo dos mesmos, o histórico dos tratores alugados por estes, os dados dos funcionários de cada stand, os dados de cada filial e o registo dos alugueres efetuados.

A criação de uma base de dados e a digitalização dos processos permitirá à AgroAuto reduzir erros humanos, melhorar a eficiência no atendimento ao cliente e aliviar a sobrecarga administrativa. Esta modernização contribuirá para a sustentabilidade do negócio a longo prazo, permitindo que o Sr.Pires dedique mais tempo à definição de estratégias de crescimento, sem comprometer a qualidade dos serviços.

Além disso, o novo sistema reforçará a posição da AgroAuto no mercado, ao mesmo tempo que honra o legado de Francisco Pires, pai do Sr. Artur Pires, promovendo a mecanização agrícola e facilitando o trabalho dos agricultores. Ao alinhar tecnologia com tradição, a empresa prepara-se para enfrentar os desafios do setor agrícola com maior eficácia e inovação.

1.2.2 Objetivos

Após considerar as sugestões dos colaboradores dos diferentes stands e receber aconselhamento de outros empresários do setor, o Sr.Pires definiu uma série de objetivos a serem alcançados com a implementação do sistema de bases de dados, nomeadamente:

- Otimização da gestão do negócio, garantindo um acompanhamento mais eficiente dos alugueres em cada stand e melhorando a organização operacional;
- Melhoria no controlo financeiro da empresa, facilitando a gestão e controlo de pagamentos, reduzindo atrasos e garantindo maior transparência nas transações;
- Acompanhamento em tempo real da disponibilidade das diferentes máquinas, possibilitando uma resposta mais rápida e eficaz às solicitações dos clientes;
- Aprofundamento do conhecimento sobre os seus clientes possibilitando a implementação de estratégias personalizadas, como ofertas especiais para clientes frequentes ou incentivos para pagamentos a pronto;
- Automatização de processos administrativos, reduzindo a carga de trabalho e minimizando erros derivados da gestão manual;
- Melhoria da qualidade de serviço prestado, tornando a experiência do cliente mais satisfatória;

1.3 Análise da Viabilidade do processo

O Sr.Pires acredita que, ao implementar uma base de dados eficiente para informatizar e organizar os dados contabilísticos dos clientes e das suas máquinas, conseguirá obter informações mais precisas para análise e resolução de problemas do quotidiano das diferentes filiais. A adoção deste sistema permitirá melhorar diversos aspectos, tais como:

- **Plano de negócio:** Com a automatização dos processos administrativos, o Sr.Pires poderá dedicar mais tempo à estratégia empresarial, analisando oportunidades de crescimento, como a introdução de novos tipos de maquinaria agrícola ou a expansão para outras regiões do país. Este tempo adicional não permitirá apenas aumentar a oferta de equipamentos e serviços, mas também consolidar a reputação da AgroAuto como referência no setor agrícola.
- **Escalabilidade:** A expansão da AgroAuto e o aumento da complexidade das suas operações exigem uma gestão estruturada que acompanhe esse crescimento. Um sistema de bases de dados permitirá gerir eficientemente um volume crescente de informações, garantindo uma resposta rápida e organizada às necessidades do mercado agrícola. Com esta abordagem, a empresa poderá expandir-se sem comprometer a qualidade dos seus

serviços.

- **Viabilidade Tecnológica:** As soluções de *software* atuais oferecem sistemas de gestão robustos, modulares e adaptáveis às necessidades da AgroAuto garantindo uma transição eficiente. A integração com sistemas existentes e a possibilidade de personalização da plataforma garantirão um controlo rigoroso dos alugueres, da manutenção das máquinas e dos pagamentos dos clientes.
- **Redução de Custos Operacionais:** Embora a implementação do sistema represente um investimento inicial, os custos de manutenção serão reduzidos e amplamente compensados pela melhoria na gestão dos alugueres e pela redução de perdas financeiras. O Sr.Pires estima que, já no primeiro mês de funcionamento do novo sistema, será possível recuperar aproximadamente 20% das perdas acumuladas nas vendas do último ano, o que torna o investimento rapidamente rentável.

1.4 Recursos e Equipa de Trabalho

A implementação deste projeto requer uma equipa composta por profissionais de diferentes áreas, que trabalham em conjunto para garantir o sucesso da empresa. Cada funcionário contribui com a sua experiência, assegurando que todos os detalhes são cuidadosamente analisados para alcançar os melhores resultados possíveis.

1.4.1 Equipa de trabalho

O responsável pela gestão administrativa é o Sr.Pires, que assume um papel central na organização da empresa. As suas responsabilidades incluem a supervisão da parte financeira, a contratação de novos funcionários e a coordenação das operações entre os três stands, garantindo uma comunicação eficaz entre eles.

A restante equipa de funcionários interinos é composta pelo Diogo Costa, a Fátima Teixeira e o Fábio Vieira. Estes profissionais estão distribuídos pelos três stands e desempenham funções essenciais, como o atendimento ao consumidor, o registo de alugueres, a verificação da qualidade dos veículos e a promoção da empresa.

Além do pessoal interino, a empresa também conta com um profissional externo especializado.

O engenheiro de bases de dados, João Carmo, que pertence à empresa de desenvolvimento de *software*. Este estrutura e organiza as informações dos alugueres, dos clientes e das máquinas, assegurando que tudo é armazenado de forma segura e acessível.

1.4.2 Recursos Humanos

Os recursos humanos da empresa englobam todos os colaboradores que contribuem para o seu funcionamento, o engenheiro de base de dados, um representante da empresa de desenvolvimento de software, e ainda os próprios clientes, cuja participação ativa é fundamental para o aperfeiçoamento contínuo dos serviços, contribuindo para a identificação de melhorias e para o alinhamento dos serviços com as suas reais necessidades.

1.4.3 Recursos Materiais

Para garantir o sucesso da implementação do sistema de gestão de base de dados da Agro-Auto, é essencial contar com recursos adequados que sustentem o funcionamento eficiente do *software*. Estes recursos incluem componentes materiais, como *hardware* e *software*. A escolha cuidadosa destes recursos é fundamental para alcançar os objetivos propostos e assegurar que a empresa consiga beneficiar ao máximo das vantagens da sua implementação.

No que diz respeito ao *hardware*, serão necessários um servidor dedicado, que servirá de base para o armazenamento e processamento dos dados, e três postos de aluguer, destinados ao registo e gestão dos alugueres realizados pela empresa.

Quanto ao *software*, será utilizado um Sistema de Gestão de Base de Dados (SGBD), essencial para a organização e consulta eficiente das informações, bem como aplicações específicas para a gestão de alugueres, permitindo o registo e o acompanhamento das transações, dos clientes e dos tratores.

1.5 Plano de execução do projeto

O desenvolvimento do sistema de gestão de alugueres de tratores será realizado em várias fases, conforme acordado entre o Sr.Pires e a empresa de desenvolvimento de *software*. O objetivo é estruturar um plano claro e definido para a implementação do sistema de base de dados, garantindo que cada etapa seja cumprida dentro dos prazos estabelecidos e com a máxima eficácia.

Após a troca de ideias e a análise de diversos pontos, foi possível chegar a um consenso entre ambos. Desta forma, foi elaborado um Diagrama de GANTT para organizar as tarefas já concluídas e as que precisam de ser realizadas nos próximos meses, definindo os prazos de conclusão das diferentes tarefas necessárias.

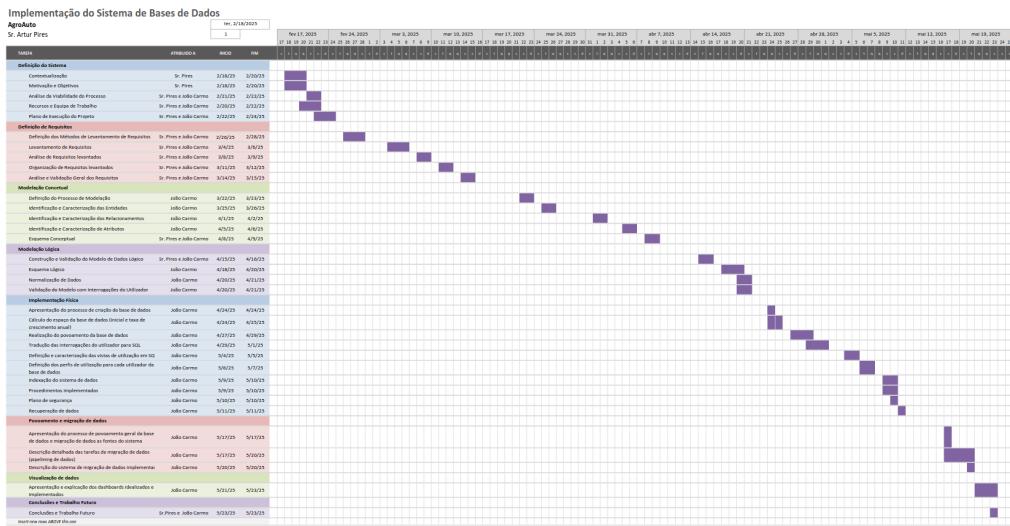


Figura 1.1: Diagrama de GANTT - expectativa

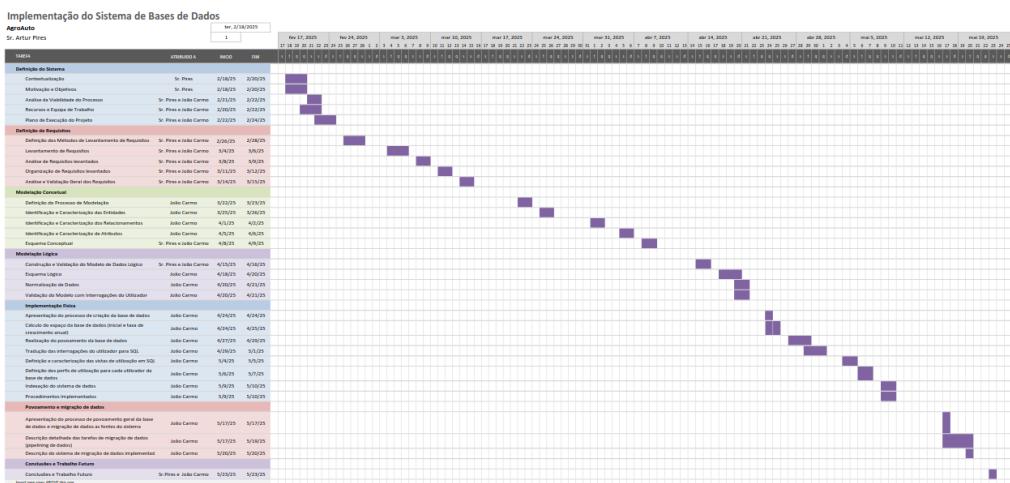


Figura 1.2: Diagrama de GANTT - realidade

O Diagrama de GANTT, que reflete a execução real do projeto, apresenta algumas diferenças em relação ao plano original. Apesar do planeamento cuidadoso e da definição clara de objetivos, durante a execução do projeto registaram-se algumas discrepâncias entre os intervalos definidos inicialmente e a realidade de execução. Estas diferenças devem-se, sobretudo, a fatores operacionais específicos da AgroAuto e à natureza dinâmica do desenvolvimento do sistema.

Na primeira fase do projeto, uma das principais razões para os desvios verificados esteve relacionada com a disponibilidade limitada do Sr.Pires para validação de requisitos e esclarecimento de processos internos, o que levou a pequenos atrasos na fase de levantamento e análise de requisitos. Também a revisão de alguns pressupostos iniciais do modelo de dados e a deteção

de ambiguidades nos requisitos iniciais exigiram novas reuniões com o Sr.Pires, o que culminou no atraso de diversas tarefas. Adicionalmente, a falta de documentação detalhada sobre os procedimentos atuais da empresa exigiu um maior esforço na recolha e organização da informação necessária, impactando ligeiramente o tempo previsto para a modelação conceptual.

Durante a segunda fase do projeto, a migração de dados enfrentou desafios devido a diferenças entre *PostgreSQL*, MySQL e JSON. Foi necessário ajustar consultas SQL e validar os dados antes da inserção, causando atrasos no cronograma. Apesar disso, conseguimos otimizar o processo e garantir a integração correta das informações.

Além disso, a necessidade de corrigir dados inconsistentes prolongou algumas etapas. Foram encontrados problemas de formatação de datas e tipos numéricos, exigindo ajustes antes da importação. Apesar do esforço extra, essas correções foram essenciais para garantir a precisão e estabilidade do sistema.

Estas discrepâncias, expostas na Tabela de Discrepâncias Temporais, embora pontuais, revelam a importância de manter uma abordagem flexível no planeamento e o impacto dos desvios em relação ao planeamento inicial. No geral, o cronograma manteve-se dentro dos limites aceitáveis, permitindo a conclusão do projeto com sucesso, respeitando os objetivos definidos e mantendo a qualidade esperada.

Posteriormente, foi realizada uma reunião entre o Sr.Pires, a equipa de trabalho e o representante da empresa de desenvolvimento, com o objetivo de validar a estrutura da base de dados proposta. Esta reunião permitiu esclarecer dúvidas pendentes e ajustar alguns detalhes referidos. A discussão incluiu a análise do plano de implementação apresentado no diagrama de GANTT. Após esta validação conjunta, o Sr.Pires aprovou formalmente o projeto.

2 Levantamento e Análise de Requisitos

A definição de requisitos é um processo essencial na gestão de projetos, que serve como base para a execução bem-sucedida do projeto. Este processo envolve a identificação, documentação e gestão das necessidades e expectativas das partes envolvidas, garantindo alinhamento ao longo de todo o ciclo de vida do projeto.

Além de funcionar como um modelo para a condução do projeto, a definição de requisitos assegura que todos os envolvidos compreendam o que deve ser entregue, sob quais condições e dentro de qual prazo. Também é fundamental considerar aspectos relacionados à operacionalidade do sistema, especialmente no caso de uma base de dados, analisando e documentando os dados envolvidos, desde o armazenamento até à exploração ao longo do tempo.

Uma definição de requisitos precisa e detalhada minimiza riscos como desvios dos objetivos, falhas de comunicação ou revisões dispendiosas. Desta forma, contribui para que o projeto cumpra as suas metas e atenda plenamente às expectativas e objetivos delineados pelo Sr.Pires.

2.1 Método de Levantamento e de Análise dos Requisitos Adotados

Para garantir um levantamento de requisitos eficiente e a projeção de uma base de dados eficaz, segura e escalável, torna-se crucial fazer uma análise detalhada acerca do funcionamento da empresa AgroAuto, identificando e clarificando os dados e processos implementados.

Posto isto, com o objetivo de obter uma visão mais completa e precisa das necessidades do sistema, será fundamental combinar técnicas que permitam entender as necessidades dos utilizadores e capturar todas as informações necessárias. A seguir, são apresentados os principais métodos utilizados:

Reuniões regulares com o Sr.Pires e os funcionários: Estas reuniões, registadas em atas, como Ata da Reunião Nº01, serão essenciais para compreender os desafios operacionais e identificar os requisitos do sistema. Um maior entendimento acerca das principais necessidades da empresa irá proporcionar um levantamento de requisitos específicos para a base de dados e uma análise das expectativas dos funcionários em relação à otimização dos processos.

Acompanhamento de operações no dia a dia do trabalho: O acompanhamento do quotidiano da empresa será fundamental para observar de perto o funcionamento dos diferentes stands e identificar objetivos práticos para a base de dados. Através da análise das rotinas de trabalho, será possível compreender melhor os processos de atendimento, gestão de veículos e registo de alugueres, garantindo que o sistema a ser desenvolvido refletira com precisão as necessidades operacionais da empresa.

Averiguação da performance de outros stands: A avaliação do desempenho de outras empresas permitirá identificar boas práticas e possíveis melhorias a serem implementadas. Ao estudar métodos de administração, aplicação tecnológica e desempenho operacional, será possível comparar esses parâmetros com a realidade do negócio e otimizar o sistema a desenvolver.

Análise da documentação da empresa: Através da exploração de contratos, análise do Formulário de Registo do Cliente, Formulário de Registo de Alugueres e outros documentos, será possível identificar requisitos fundamentais para o desenvolvimento de um sistema que atenda às necessidades do negócio de forma precisa e eficiente.

Inquéritos aos clientes: A realização de inquéritos permite recolher *feedback* sobre a experiência dos clientes, abrangendo o atendimento, a qualidade dos veículos e a organização e profissionalismo dos colaboradores. A análise das respostas ajudará a definir requisitos para um sistema mais intuitivo e ajustado às necessidades dos utilizadores.

2.2 Organização dos Requisitos Levantados

Após uma análise cuidadosa de toda a informação recolhida, reuniram-se as informações mais relevantes de modo a procedermos ao desenho da base de dados. Neste momento, definimos as nossas áreas de interesse e agrupámo-las de modo a obter a visão total da base de dados.

2.2.1 Requisitos de descrição

O processo de definição dos requisitos teve origem em falhas concretas identificadas pelo Sr.Pires na gestão das filiais da AgroAuto.

Durante a observação de tarefas rotineiras no stand de Lamego, verificou-se uma situação em que dois clientes reservaram o mesmo trator para o mesmo período, devido à ausência de um registo fiável do seu estado. Este episódio evidenciou a necessidade de garantir que cada trator tivesse um estado constantemente atualizado no sistema e que cada aluguer fosse associado, de forma inequívoca, a um único trator, prevenindo assim conflitos de agendamento. Estas necessidades estão refletidas nos requisitos RD20 e RD21.

Durante a análise das respostas dos clientes aos inquéritos propostos, o Sr.Pires apontou que clientes frequentes eram obrigados a fornecer os seus dados pessoais sempre que realizavam um novo aluguer. Esta repetição aumentava o tempo de atendimento, originava erros no preenchimento e duplicação de dados. Para colmatar esta lacuna, foi definido que cada cliente deve ter um identificador único e um registo completo com os seus dados pessoais. Estas necessidades surgem explicitadas nos requisitos RD02 a RD06. Daqui também surgiu a urgência da definição de identificadores únicos para todos os clientes, explícito no requisito RD01.

Durante o encerramento mensal de contas e a consequente análise da documentação relativa à gestão de pagamentos, o Sr.Pires detetou incoerências nos valores totais dos alugueres e nos registos das formas de pagamento anotadas. Esta constatação evidenciou a urgência de padronizar o registo do método, tipo e estado do pagamento em cada aluguer, de forma a garantir um controlo financeiro mais rigoroso e transparente. Estas necessidades estão contempladas nos requisitos RD08 a RD12.

Os restantes requisitos de descrição foram identificados através dos métodos de levantamento previamente descritos e resultam de uma reflexão detalhada sobre as reais necessidades operacionais da empresa.

Com base nesta análise e nos processos internos da AgroAuto, foi possível estabelecer um conjunto de requisitos de descrição que orientam de forma sólida e coerente a estruturação da base de dados central.

Nr	Vista	Data e Hora	Descrição	Fonte	Analista
RD01	Aluguer	10/03/2025 09:00	Todos os identificadores atribuídos são únicos e gerados sequencialmente.	Sr.Pires	João Carmo
RD02	Aluguer	10/03/2025 10:10	No momento do registo de todos os clientes é atribuído um identificador único e guardado o nome completo, a data de nascimento, informação sobre o documento de identificação e carta de condução, a morada, o número de telemóvel e o email do cliente.	Sr.Pires	João Carmo
RD03	Aluguer	10/03/2025 11:40	O documento de identificação inclui número, data de validade do documento e NIF.	Sr.Pires	João Carmo
RD04	Aluguer	10/03/2025 13:50	A carta de condução inclui data de validade da carta e habilitação.	Sr.Pires	João Carmo
RD05	Aluguer	10/03/2025 15:00	A morada deve conter código postal, localidade e rua.	Sr.Pires	João Carmo
RD06	Aluguer	10/03/2025 15:40	Se o método de pagamento do aluguer de um trator selecionado for o cartão de crédito, o registo do cliente deve guardar o seu número, CVV e data de validade do cartão.	Sr.Pires	João Carmo
RD07	Aluguer	10/03/2025 16:30	Ao registar um aluguer, o sistema deve associar um único cliente ao seu registo.	Sr.Pires	João Carmo
RD08	Aluguer	10/03/2025 17:11	Um aluguer é registado guardando o seu identificador único, data de início e fim e o preço total do aluguer.	Sr.Pires	João Carmo
RD09	Aluguer	10/03/2025 17:57	No momento do registo do aluguer é selecionado o método, o estado e o tipo de pagamento.	Sr.Pires	João Carmo
RD10	Aluguer	10/03/2025 18:45	O método de pagamento para o aluguer pode ser cartão de crédito ou dinheiro.	Sr.Pires	João Carmo
RD11	Aluguer	11/03/2025 09:30	O estado do pagamento é distinguido entre em atraso e concluído.	Sr.Pires	João Carmo
RD12	Aluguer	11/03/2025 10:10	O tipo do pagamento pode ser classificado como a pronto ou a prestações.	Sr.Pires	João Carmo
RD13	Aluguer	11/03/2025 10:52	Ao registar um aluguer, o sistema deve associar um único funcionário ao seu registo.	Sr.Pires	João Carmo
RD14	Aluguer	11/03/2025 11:42	O funcionário é caracterizado pelo seu identificador único, nome completo, telemóvel, e IBAN.	Sr.Pires	João Carmo
RD15	Aluguer	11/03/2025 14:00	O sistema deve associar todos os funcionários ao único stand em que trabalham.	Sr.Pires	João Carmo
RD16	Aluguer	11/03/2025 14:30	Cada stand é caracterizado pelo seu identificador único, morada, número de telefone e email.	Sr.Pires	João Carmo
RD17	Aluguer	11/03/2025 15:01	A morada inclui localidade, código postal e rua.	Sr.Pires	João Carmo
RD18	Aluguer	11/03/2025 15:43	O sistema deve associar cada trator ao stand a que pertence.	Sr.Pires	João Carmo
RD19	Aluguer	11/03/2025 16:19	Ao trator é associado um identificador único, a marca, o modelo, preço diário e o estado do trator.	Sr.Pires	João Carmo
RD20	Aluguer	11/03/2025 16:59	O estado do trator pode ser classificado como livre ou alugado.	Sr.Pires	João Carmo
RD21	Aluguer	11/03/2025 17:30	O registo de um aluguer refere-se a um único trator.	Sr.Pires	João Carmo
RD22	Aluguer	11/03/2025 17:56	Um stand tem de ter pelo menos um trator alocado a este	Sr.Pires	João Carmo
RD23	Aluguer	11/03/2025 18:24	Um stand tem pelo menos um funcionário alocado a este	Sr.Pires	João Carmo

Tabela 2.1: Documento dos Requisitos de Descrição

2.2.2 Requisitos de manipulação

A segunda vertente identificada no desenvolvimento do sistema da AgroAuto corresponde à componente funcional, de manipulação do sistema, relacionada com o uso prático da base de dados. Os requisitos definidos nesta fase resultam da análise das necessidades operacionais da empresa e visam assegurar que o sistema permite responder de forma eficaz às ações esperadas

no contexto real de utilização.

Durante uma das reuniões de planeamento com o Sr.Pires, este realçou a importância de aceder rapidamente a informações fundamentais sobre os clientes. Por isso, definiu-se o requisito RM01, que prevê a possibilidade de listar todos os clientes, ordenados pelo seu identificador. Esta funcionalidade visa facilitar a consulta e gestão eficiente do histórico dos clientes.

Com o objetivo de melhorar o controlo diário da frota, o Sr.Pires expressou a necessidade de conhecer, no início de cada dia, tanto os veículos que estão alugados como os disponíveis para aluguer, exposto nos requisitos RM02 e RM03. Esta distinção facilita a alocação de recursos e a previsão de disponibilidade, permitindo uma gestão proativa.

Durante a observação do funcionamento das diferentes filiais, verificou-se que os funcionários tinham dificuldade em encontrar tratores com características específicas. Assim, foi definido o requisito RM06, que permite filtrar os tratores disponíveis por marca, modelo, estado ou localização, agilizando o processo de seleção para possível aluguer.

O Sr.Pires também sublinhou a utilidade de visualizar o histórico de alugueres de um cliente, especialmente em contexto de atendimento ou análise de comportamento. Para isso, foi criado o requisito RM07, que apresenta os alugueres de forma ordenada por data, destacando os mais recentes.

Do ponto de vista estratégico, o Sr.Pires expressou interesse em obter métricas de desempenho comercial e de utilização da frota. Assim surgiram os requisitos RM09, RM10, RM11 e RM12, que permitem extrair dados estatísticos relevantes para a tomada de decisão e planeamento.

Os restantes requisitos de manipulação foram identificados com base nos métodos de levantamento previamente descritos e resultam de uma análise cuidadosa das rotinas diárias e exigências da gestão operacional da AgroAuto.

Através desta análise, foi possível definir um conjunto de requisitos de manipulação que asseguram não só a fiabilidade na consulta e atualização dos dados, mas também a extração de informação relevante para a tomada de decisão, promovendo uma utilização eficaz e estratégica do sistema de base de dados.

Nr	Vista	Data e hora	Descrição	Fonte	Analista
RM01	Aluguer	17/03/2025 08:54	A qualquer altura deverá ser possível obter uma lista com os registos de todos os clientes. A mesma é ordenada de forma crescente pelo idCliente.	Sr.Pires	João Carmo
RM02	Aluguer	17/03/2025 10:00	A cada dia, o sistema deve apresentar uma lista agrupada por stand, com o identificador único e o modelo de todos os tratores alugados, ordenados alfabeticamente pela marca do mesmo.	Sr.Pires	João Carmo
RM03	Aluguer	17/03/2025 11:00	No início de cada dia o sistema deve ser capaz de apresentar a lista de tratores disponíveis para aluguer;	Sr.Pires	João Carmo
RM04	Aluguer	17/03/2025 11:30	A qualquer momento, o sistema deve verificar se um cliente está apto a fazer um aluguer, verificando se a sua carta de condução é válida. É verificada se a data de validade desta é superior à data de término do aluguer e se a habilitação é do tipo T.	Sr.Pires	João Carmo
RM05	Aluguer	17/03/2025 13:00	Ao registrar um novo aluguer, o sistema deve remover esse pedido caso o cliente tenha débito em falta de alugueres anteriores ou caso a carta de condução não seja válida.	Sr.Pires	João Carmo
RM06	Aluguer	17/03/2025 14:30	A consulta de tratores disponíveis para aluguer deve permitir a filtragem por marca, modelo, estado ou stand.	Sr.Pires	João Carmo
RM07	Aluguer	17/03/2025 16:00	Os alugueres realizados por um cliente devem ser apresentados ordenados pela data de início, com os alugueres mais recentes primeiro.	Sr.Pires	João Carmo
RM08	Aluguer	17/03/2025 16:40	O estado de um aluguer é atualizado para 'concluído' apenas se o trator já tiver sido devolvido e inspecionado.	Sr.Pires	João Carmo
RM09	Aluguer	17/03/2025 17:25	No fim de cada ano, o sistema deve apresentar a marca de tratores mais alugada.	Sr.Pires	João Carmo
RM10	Aluguer	18/03/2025 09:00	O sistema deve conseguir determinar qual foi o funcionário com mais vendas num mês específico.	Sr.Pires	João Carmo
RM11	Aluguer	18/03/2025 10:00	O sistema deve conseguir determinar qual o número de alugueres na empresa no último trimestre.	Sr.Pires	João Carmo
RM12	Aluguer	18/03/2025 12:00	O sistema deve conseguir determinar o valor total faturado em cada stand no final do mês.	Sr.Pires	João Carmo
RM13	Aluguer	18/03/2025 15:00	O sistema permite a importação de informações provenientes de ficheiros CSV, JSON e de bases de dados relacionais, nomeadamente PostgreSQL.	Sr.Pires	João Carmo

Tabela 2.2: Dicionário dos Requisitos de Manipulação

2.2.3 Requisitos de controlo

Por fim, a última vertente é a de controlo, e está relacionada com a gestão de acessos e permissões no ambiente da base de dados. Com base nas funções desempenhadas por diferentes perfis de utilizadores e nas necessidades identificadas durante o levantamento de requisitos, foi necessário estabelecer regras claras quanto ao nível de acesso permitido a cada utilizador, assegurando uma utilização segura e adequada da informação.

ATA DA REUNIÃO Nº 04 – DISCUSSÃO DE NÍVEIS DE ACESSO

No dia 27 de março de 2025, às 14h30, realizou-se uma reunião na Sala de Reuniões da sede de Olivença da empresa AgroAuto de forma a discutir a gestão de acessos e permissões no ambiente da base de dados.

A reunião foi conduzida pelo Sr. Pires, administrador do negócio e contou com a participação de um engenheiro de base de dados. Não houve ausências.

Durante a sessão, com base em queixas previamente reportadas pelos funcionários ao Sr. Pires, foram analisadas situações reais do dia a dia laboral, nas quais se identificaram dificuldades práticas na execução de determinadas tarefas, sobretudo quando o administrador não se encontra disponível.

Entre os pontos discutidos, destacou-se a necessidade de permitir a consulta logo pela manhã, da lista de tratores disponíveis para aluguer, uma funcionalidade considerada essencial para garantir agilidade no atendimento e eficiência na operação diária.

Foi também referido que, com frequência, os funcionários necessitam de atualizar regtos de clientes e alugueres, como ajustar datas, corrigir dados pessoais ou inserir observações nos contratos.

Adicionalmente, durante a reunião, foi sublinhada a importância de garantir a segurança da informação e a confidencialidade de determinados dados estratégicos da empresa.

Com base nestas observações, o engenheiro de base de dados comprometeu-se a propor soluções técnicas para colmatar as dificuldades identificadas, ficando acordado que será agendada uma nova reunião para validação das mesmas junto dos funcionários.

Presentes

- Sr. Artur Pires (Administrador);
- João Carmo (Engenheiro de base de dados);

Figura 2.1: Ata da Reunião Nº04

Em consequência da reunião realizada com o Sr.Pires foi necessária a criação de permissões específicas para os dois tipos de utilizadores, o administrador e o funcionário, seguindo um modelo de segurança baseado em privilégios mínimos. Por padrão, nenhum utilizador possui acesso a qualquer funcionalidade, sendo as permissões concedidas apenas conforme necessário para o desempenho das suas funções.

A observação levantada da "necessidade de permitir a consulta logo pela manhã, da lista de tratores disponíveis para aluguer "levou à implementação do requisito RC11 que permite ao administrador e a todos os funcionários consultar os tratores disponíveis para aluguer diaria-

mente.

Do mesmo modo, identificou-se que "os funcionários necessitam de atualizar registo de clientes e alugueres". Esta necessidade operacional justificou a criação dos requisitos RC12 e RC13, que conferem aos funcionários a capacidade de manter atualizados tanto os dados dos clientes quanto as informações relativas aos contratos de aluguer.

Durante a reunião, foi também sublinhada a "importância de garantir a segurança da informação e a confidencialidade de determinados dados estratégicos da empresa", o que levou à criação dos requisitos RC02 a RC08, que estabelecem permissões exclusivas para o administrador sobre informações consideradas sensíveis ou estratégicas.

Os restantes requisitos foram definidos com base nos problemas previamente descritos e observados, visando uma estrutura de permissões que equilibra as necessidades operacionais diárias com a proteção de dados sensíveis, em conformidade com as diretrizes estabelecidas pelo Sr.Pires durante a reunião de 27 de março.

Nr	Vista	Data e hora	Descrição	Fonte	Analista
RC01	Aluguer	01/04/2025 07:22	O sistema apenas deve operar entre as 07:00 e 19:00.	Sr.Pires	João Carmo
RC02	Aluguer	01/04/2025 08:00	Apenas o administrador pode consultar qual o trimestre com maior número de tratores alugados na empresa.	Sr.Pires	João Carmo
RC03	Aluguer	01/04/2025 09:10	Apenas o administrador pode conhecer quais os clientes que mais tratores alugaram a partir de determinada data.	Sr.Pires	João Carmo
RC04	Aluguer	01/04/2025 09:40	Apenas o administrador pode consultar a marca de tratores mais alugada no final de cada ano.	Sr.Pires	João Carmo
RC05	Aluguer	01/04/2025 10:11	Apenas o administrador pode consultar todos os registo de alugueres realizados entre determinadas datas por parte de qualquer funcionário.	Sr.Pires	João Carmo
RC06	Aluguer	01/04/2025 10:55	Apenas o administrador pode consultar a lista de todos os funcionários.	Sr.Pires	João Carmo
RC07	Aluguer	01/04/2025 12:01	Apenas o administrador pode determinar qual o funcionário com mais alugueres num dado mês.	Sr.Pires	João Carmo
RC08	Aluguer	01/04/2025 14:20	Apenas o administrador pode consultar o lucro total dos alugueres efetuados num dado período de tempo.	Sr.Pires	João Carmo
RC09	Aluguer	01/04/2025 15:07	O administrador e todos os funcionários podem consultar a lista de todos os clientes.	Sr.Pires	João Carmo
RC10	Aluguer	01/04/2025 15:43	O administrador e todos os funcionários podem visualizar a lista de tratores atualmente alugados.	Sr.Pires	João Carmo
RC11	Aluguer	01/04/2025 16:26	O administrador e todos os funcionários podem, a cada dia, consultar a lista de tratores disponíveis para alugar.	Sr.Pires	João Carmo
RC12	Aluguer	01/04/2025 17:04	O administrador e todos os funcionários podem alterar informação presente na ficha de registo de qualquer cliente.	Sr.Pires	João Carmo
RC13	Aluguer	01/04/2025 17:40	O administrador e todos os funcionários podem alterar informação presente na ficha de registo de qualquer aluguer.	Sr.Pires	João Carmo
RC14	Aluguer	01/04/2025 18:33	O administrador e todos os funcionários podem consultar o histórico de alugueres de qualquer cliente.	Sr.Pires	João Carmo
RC15	Aluguer	01/04/2025 19:53	Apenas o administrador pode consultar o valor total faturado por cada stand ao final de um mês	Sr.Pires	João Carmo

Tabela 2.3: Dicionário dos Requisitos de Controlo

2.3 Análise e Validação Geral dos Requisitos

Após a fase de levantamento e definição dos requisitos, procedeu-se à sua validação. Este processo é fundamental durante o desenvolvimento do sistema, pois permite compreender quais informações precisam de ser armazenadas na base de dados.

Posteriormente, realizou-se outra reunião com o Sr.Pires para discutir a viabilidade do projeto desenvolvido até ao momento. Durante essa reunião, foram apresentadas as especificações levantadas, e algumas discordâncias foram identificadas. Destacou-se a preocupação expressa pelo cliente relativamente ao acesso, partilha e consulta dos dados pessoais dos funcionários por parte de outros colaboradores, solicitando restrições mais rigorosas para evitar acessos indevidos.

Além disso, foram levantadas interrogações sobre a necessidade de armazenar determinadas informações, como os dados completos do cartão de crédito dos clientes, apelando a questões de segurança e conformidade com as normas de proteção de dados. Também houve dúvidas sobre o método de cálculo do preço total do aluguer, principalmente em relação a possíveis taxas adicionais ou descontos aplicáveis em determinadas condições.

Após os requisitos estarem validados e ajustados conforme o *feedback* do Sr.Pires, decidiu-se avançar para a próxima etapa do projeto, a modelação conceptual.

3 Modelação Conceptual

3.1 Apresentação da Abordagem de Modelação Realizada

A modelação conceptual do sistema de aluguer de tratores foi desenvolvida com o objetivo de representar, de forma abstrata e rigorosa, os principais elementos informativos e estruturais do domínio da aplicação. Esta fase permitiu captar os requisitos essenciais do sistema, organizando-os numa representação formal que sustenta as fases subsequentes de modelação lógica e física da base de dados.

A abordagem seguida baseou-se na utilização do modelo Entidade-Relacionamento (ER), segundo a notação clássica proposta por Peter Chen, complementada pelas boas práticas descritas por Connolly e Begg. Esta notação permitiu representar graficamente entidades, atributos, relacionamentos, cardinalidades e restrições de participação, contribuindo para uma compreensão clara e precisa da estrutura de dados do sistema.

O processo de modelação incluiu os seguintes passos fundamentais, nomeadamente, a identificação das entidades relevantes com base nos requisitos fornecidos e a determinação dos atributos necessários para caracterizar cada entidade. Posteriormente, realizou-se a definição e documentação dos relacionamentos entre entidades, com base nos requisitos do sistema, o estabelecimento das cardinalidades mínimas e máximas, garantindo a correta interpretação das regras de negócio e a caracterização das participações, totais ou parciais, das entidades nos relacionamentos.

Para a construção do diagrama ER foi utilizada a ferramenta *BRModelo*, com símbolos padronizados para entidades, relacionamentos, atributos simples e compostos, chaves primárias e cardinalidades.

3.2 Identificação e Caracterização das Entidades

A identificação das entidades do sistema da AgroAuto foi realizada com base nos requisitos previamente definidos. A partir das descrições dos processos de negócio, identificamos cinco entidades fundamentais: Cliente, Trator, Stand, Funcionário e Aluguer.

A entidade Cliente foi estabelecida para representar os utilizadores do serviço de aluguer, permitindo o registo e a identificação única de cada pessoa que contrata os serviços da empresa. O requisito RD02 determina que, no momento do registo do cliente, seja atribuído um identificador único, implicando a sua modelação como entidade independente. Os requisitos RD03, RD04 e RD05 reforçam esta necessidade ao especificarem a gestão de dados legais e de morada, associados a cada cliente.

A entidade Aluguer foi identificada com base nos requisitos RD08, RD09, RD10, RD11 e RD12 que descrevem com detalhe os dados que caracterizam cada instância de aluguer, incluindo prazos, valores e modalidades de pagamento.

A existência da entidade Funcionário é justificada pelo requisito RD14, que define a obrigatoriedade de registar informação individualizada de cada colaborador, incluindo a sua identificação e dados bancários.

A entidade Stand foi identificada com base nos requisitos RD16 e RD17, que definem um conjunto de atributos próprios que caracterizam a unidade física onde se realizam os alugueres.

Por fim, o Trator é o recurso físico central do sistema, cuja modelação como entidade decorre dos requisitos RD19 e RD20, onde se especifica que cada trator possui um identificador, características técnicas e um estado, livre ou alugado.

Estes dados são necessários para o controlo de disponibilidade e cálculo de custos.

Entidade	Descrição	Ocorrência	Sinónimos
Cliente	Representa os utilizadores que contratam serviços de aluguer de tratores. Esta entidade guarda informação pessoal e legal necessária para a celebração do contrato de aluguer.	Cada cliente possui um registo único obrigatório, necessário para efetuar qualquer aluguer. O cliente é obrigado a fornecer os seus dados pessoais e de pagamento.	Consumidor
Aluguer	Regista a operação de aluguer de um trator, incluindo dados temporais, financeiros e contratuais associados.	A cada aluguer é atribuído um número único, no momento do registo.	-
Funcionário	Representa os colaboradores responsáveis por registrar e gerir os alugueres, atendendo os clientes e administrando os processos relacionados aos tratores disponíveis no stand.	Os funcionários estão registados de forma única na base de dados.	Colaborador, Trabalhador
Stand	Representa os locais físicos onde os tratores se encontram disponíveis para aluguer, bem como onde trabalham os funcionários.	Os stands têm um número único que os identifica na base de dados.	Filial
Trator	Entidade que constitui o recurso principal disponibilizado. Os tratores são a entidade mais importante da empresa, sendo disponibilizados aos clientes mediante pagamento.	Os tratores estão registados de forma única na base de dados.	Veículo, Maquinaria Agrícola, máquina, equipamento

Tabela 3.1: Caracterização das entidades

3.3 Identificação e Caracterização dos Relacionamentos

Na identificação dos relacionamentos entre as entidades do sistema de aluguer de tratores, foram novamente considerados os princípios fundamentais definidos por Connolly e Begg, assim como as orientações gerais de modelação Entidade-Relacionamento (ER). Para cada relacionamento identificado, procedeu-se à determinação da cardinalidade, definição das restrições de participação, documentação dos tipos de relacionamento e representação através de diagramas ER.

Relacionamento binário *requisita* (Cliente – Aluguer)

Este relacionamento binário representa a associação de um cliente a cada aluguer efetuado. De acordo com o requisito RD07, ao registar um aluguer, o sistema deve associar um único cliente ao seu registo. Assim, a cardinalidade do relacionamento é de 1:N, onde um cliente pode estar associado a vários alugueres, mas cada aluguer está obrigatoriamente associado a apenas um cliente. A participação é total na entidade Aluguer, pois o aluguer não existe sem um cliente associado, e parcial na entidade Cliente, dado que nem todos os clientes precisam necessariamente de ter efetuado alugueres.

O relacionamento é exposto no seguinte diagrama ER:



Figura 3.1: Relacionamento binário Cliente - Aluguer

Relacionamento binário *regista* (Funcionário – Aluguer)

Este relacionamento binário reflete a responsabilidade dos funcionários no registo dos alugueres, conforme especificado no requisito RD13. A cardinalidade é igualmente de 1:N, uma vez que um funcionário pode registar vários alugueres, mas cada aluguer é registado por um único funcionário. A participação é total na entidade Aluguer, dado que todos os alugueres devem ser registados por um funcionário, e parcial na entidade Funcionário, pois um funcionário pode não ter ainda registado nenhum aluguer.

A representação gráfica do relacionamento é apresentada no diagrama ER seguinte:



Figura 3.2: Relacionamento binário Funcionário - Aluguer

Relacionamento binário *trabalha_num* (Funcionário – Stand)

Este relacionamento binário expressa a associação de cada funcionário ao stand onde exerce funções, conforme indicado no requisito RD15 e RD23. A cardinalidade é de N:1, considerando que cada funcionário trabalha num único stand, e que cada stand pode ter vários funcionários. A participação é total em ambas as entidades, uma vez que todos os funcionários estão associados a um stand e todos os stands possuem pelo menos um funcionário.

O seguinte diagrama ER representa o relacionamento apresentado:



Figura 3.3: Relacionamento binário Funcionário - Stand

Relacionamento binário *refere-se* (Aluguer – Trator)

Este relacionamento binário representa a associação de cada aluguer a um único trator, suportado pelo requisito RD21. A cardinalidade é de N:1, já que cada aluguer envolve apenas um trator por transação, e um trator pode estar envolvido em vários aluguers, ao longo do tempo. A participação é total na entidade Aluguer, pois este não pode ser registado sem especificar o trator, e parcial na entidade Trator, dado que um trator pode ainda não ter sido alugado.

O relacionamento é exposto no seguinte diagrama ER:



Figura 3.4: Relacionamento binário Aluguer - Trator

Relacionamento binário *pertence* (Trator – Stand)

Por fim, este relacionamento binário resulta do requisito RD18, que determina que cada trator deve estar associado ao stand a que pertence. A cardinalidade é de N:1, uma vez que um stand pode ter vários tratores, mas cada trator pertence a um único stand. A participação é total em ambas as entidades, dado que todos os tratores têm obrigatoriamente de estar associados a um stand, e todos os stands devem ter pelo menos um trator.

A representação gráfica do relacionamento é apresentada no diagrama ER seguinte:



Figura 3.5: Relacionamento binário Trator - Stand

Entidade A	Relacionamento	Cardinalidade	Participação	Entidade B
Cliente	requisita	1:N	P:T	Aluguer
Funcionário	regista	1:N	P:T	Aluguer
Funcionário	trabalha_num	N:1	T:T	Stand
Aluguer	refere-se	N:1	T:P	Trator
Trator	pertence	N:1	T:T	Stand

Tabela 3.2: Caracterização dos Relacionamentos

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

A definição dos atributos das entidades e dos relacionamentos foi realizada a partir da análise detalhada dos requisitos do sistema, procurando garantir a completude sem redundância de informação. Os atributos foram organizados por entidade, com justificação da sua inclusão com base nos requisitos de descrição.

Os atributos da entidade Cliente foram identificados com o objetivo de garantir a integridade dos dados pessoais, fiscais e de contacto. O atributo idCliente é a chave primária, sendo este um identificador único, baseado no RD01. Outros atributos como nomeCompleto e dataNascimento foram incluídos para identificar o cliente e validar a sua elegibilidade e os contactos númeroTelemovel e email são essenciais para comunicação direta, atributos expostos no requisito RD02. O atributo composto cartãoCidadão, integrado pelos atributos simples númeroDocumento, dataValidadeDocumento e NIF permite a identificação oficial do cliente, conforme RD03. Adicionalmente, o atributo composto e nulo cartãoCrédito, subdividido em númeroCartão, dataValidadeCartão e CVV, incluído apenas quando o cliente opta por pagamento com cartão de crédito, RD06, sendo estes campos opcionais e relacionados com a forma de pagamento escolhida. Desta forma, são atributos identificados como potencialmente nulos.

Foram definidos atributos que caracterizam a entidade Aluguer e o seu estado financeiro, sendo esta identificada univocamente por idAluguer, conforme o RD01. Esta entidade armazena dados temporais como dataInício e dataTérmino para registar o intervalo de aluguer, baseado no RD08, bem como preçoTotal, que indica o valor monetário total da operação. Este atributo é derivado, sendo calculado a partir da diferença entre dataTérmino e dataInício, multiplicada

pelo atributo preçoDiário do trator alugado. Os atributos métodoPagamento, estadoPagamento e tipoPagamento foram definidos segundo os requisitos RD09 a RD12.

Os atributos referentes às entidades Funcionário, Stand e Trator foram definidos através da metodologia exposta anteriormente, visando expor com maior veracidade as necessidades do Sr.Pires.

Todos os relacionamentos identificados não possuem atributos próprios.

Entidade	Atributo	Descrição	Tipo e Tamanho de Dados	Chave Primária	Nulo	Composto	Multivalorado	Derivado
Cliente	idCliente	Identificador único do Cliente	INT	Sim	Não	Não	Não	Não
	nomeCompleto	Nome Completo	VARCHAR(100)	Não	Não	Não	Não	Não
	dataNascimento	Data de Nascimento	DATE	Não	Não	Não	Não	Não
	cartãoCidadão	Documento de identificação	-	Não	Não	Sim	Não	Não
	NIF	Número de Identificação Fiscal	VARCHAR(9)	Não	Não	Não	Não	Não
	númeroDocumento	Número do Documento de Identificação	VARCHAR(8)	Não	Não	Não	Não	Não
	dataValidadeDocumento	Data de Validade do Documento de Identificação	DATE	Não	Não	Não	Não	Não
	morada	Morada do Cliente	-	Não	Não	Sim	Não	Não
	rua	Nome da Rua	VARCHAR(100)	Não	Não	Não	Não	Não
	localidade	Nome da Localidade	VARCHAR(75)	Não	Não	Não	Não	Não
	códigoPostal	Código Postal	VARCHAR(10)	Não	Não	Não	Não	Não
	númeroTelemóvel	Números de telemóvel	VARCHAR(9)	Não	Não	Não	Não	Não
	email	Endereço de email	VARCHAR(100)	Não	Não	Não	Não	Não
	cartaCondução	Carta de Condução	-	Não	Não	Sim	Não	Não
	dataValidadeCarta	Data de Validade da Carta de Condução	DATE	Não	Não	Não	Não	Não
	habilitação	Tipo de Habilitação da Carta de Condução	VARCHAR(75)	Não	Não	Não	Não	Não
	cartãoCrédito	Cartão de Crédito	-	Não	Sim	Sim	Não	Não
	dataValidadeCartão	Data de Validade do Cartão de Crédito	DATE	Não	Sim	Não	Não	Não
	númeroCartão	Número do Cartão de Crédito	VARCHAR(19)	Não	Sim	Não	Não	Não
	CVV	Valor de Verificação do Cartão	VARCHAR(3)	Não	Sim	Não	Não	Não
Aluguer	idAluguer	Identificador único do Aluguer	INT	Sim	Não	Não	Não	Não
	dataInício	Data do Início do Aluguer	DATE	Não	Não	Não	Não	Não
	dataTermino	Data de cessão do Aluguer	DATE	Não	Não	Não	Não	Não
	preçoTotal	Valor Total do Preço do Aluguer	DECIMAL(8,2)	Não	Não	Não	Não	Sim
	métodoPagamento	Método de Pagamento (Cartão de Crédito ou Dinheiro)	ENUM('CartaoCredito','Dinheiro')	Não	Não	Não	Não	Não
Funcionário	idFuncionário	Identificador único do Funcionário	INT	Sim	Não	Não	Não	Não
	nomeCompleto	Nome Completo	VARCHAR(75)	Não	Não	Não	Não	Não
Trator	númeroTelemóvel	Números de telemóvel	VARCHAR(9)	Não	Não	Não	Não	Não
	idTrator	Identificador único do Trator	INT	Sim	Não	Não	Não	Não
	modelo	Modelo do Trator	VARCHAR(75)	Não	Não	Não	Não	Não
	marca	Marca do Trator	VARCHAR(75)	Não	Não	Não	Não	Não
	preçoDiário	Preço predefinido diariamente para aluguer do trator	DECIMAL(8,2)	Não	Não	Não	Não	Não
Stand	estado	Estado do trator (Livre ou Alugado)	ENUM('Livre','Alugado')	Não	Não	Não	Não	Não
	idStand	Identificador único do Stand	INT	Sim	Não	Não	Não	Não
	morada	Morada do Stand	-	Não	Não	Sim	Não	Não
	rua	Nome da Rua	VARCHAR(100)	Não	Não	Não	Não	Não
	localidade	Nome da Localidade	VARCHAR(75)	Não	Não	Não	Não	Não
	códigoPostal	Código Postal	VARCHAR(75)	Não	Não	Não	Não	Não
	númeroTelefone	Número de Telefone do Stand	VARCHAR(9)	Não	Não	Não	Não	Não
email	Endereço de Email do Stand	VARCHAR(75)	Não	Não	Não	Não	Não	Não

Tabela 3.3: Caracterização dos atributos das entidades do sistema

3.5 Apresentação e Explicação do Diagrama ER Produzido

O diagrama entidade-relacionamento produzido resulta da aplicação da abordagem de modelação conceptual orientada aos requisitos do sistema de gestão de alugueres de tratores definidos anteriormente. A construção do modelo foi realizada utilizando a ferramenta *BRModelo*, adotando a notação clássica de Chen, adequada à representação explícita de entidades, relacionamentos e atributos, bem como das respetivas cardinalidades e restrições de participação.

A construção do diagrama iniciou-se pela criação das entidades identificadas anteriormente, cada uma representando um conjunto de objetos ou pessoas com características comuns no sistema. Cada entidade foi representada com os seus atributos mais relevantes, garantindo que toda a informação necessária para o funcionamento do sistema estivesse contemplada. Os atributos das entidades foram representados conforme a sua natureza e classificação: simples, compostos, derivados ou nulos, embora esta análise tenha sido abordada detalhadamente anteriormente.

Posteriormente, foram definidos os relacionamentos entre as entidades com base nas ligações descritas nos mesmos requisitos. As cardinalidades foram cuidadosamente analisadas para refletir as restrições de negócio implícitas, garantindo integridade semântica ao modelo. A participação das entidades em cada relacionamento foi devidamente caracterizada como total ou parcial, conforme a obrigatoriedade da sua associação.

Com esta representação conceptual, o diagrama ER assegura uma base sólida para fases subsequentes como a modelação lógica e a implementação física da base de dados.

Para validar a adequação do modelo às necessidades reais do negócio, foi realizada uma breve reunião com o Sr.Pires. Durante esta reunião, cada entidade, atributo e relacionamento foi cuidadosamente examinado e discutido, permitindo confirmar que o modelo capturava corretamente os processos de negócio e as regras operacionais da empresa.

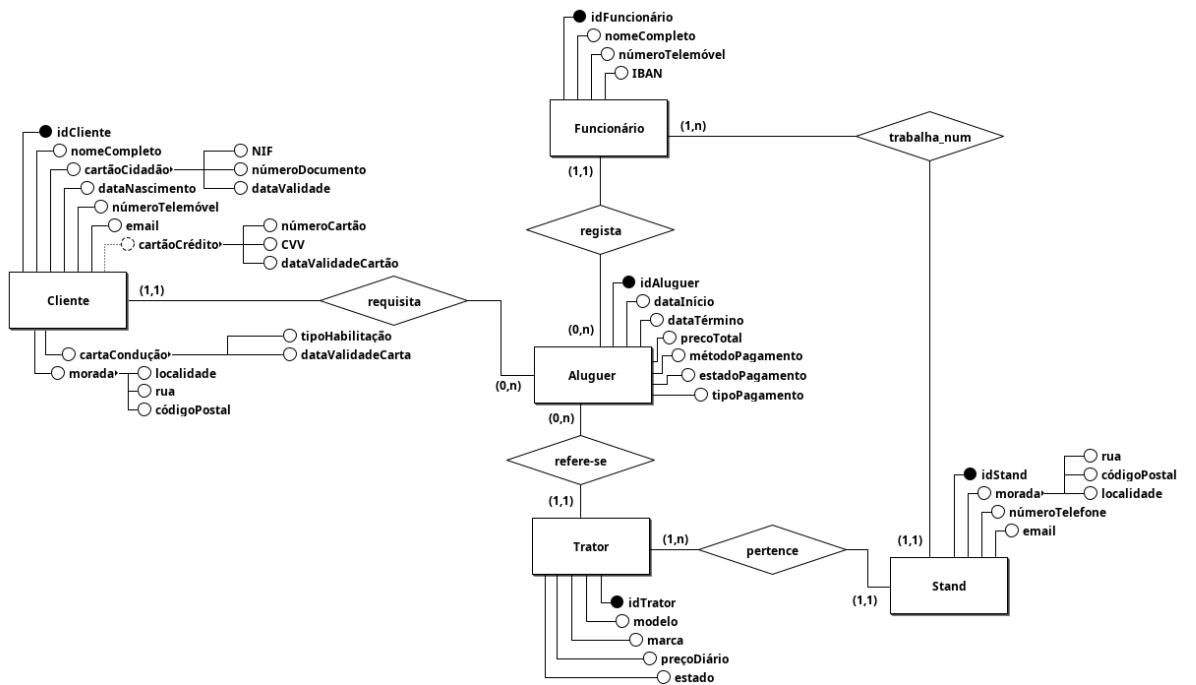


Figura 3.6: Diagrama ER do Sistema de Gestão de Alugueres de Tratores

4 Modelação Lógica

4.1 Construção e Validação do Modelo de Dados Lógico

A construção do modelo de dados lógico baseou-se no modelo relacional proposto por E.F. Codd, no qual os dados são representados através de relações e tabelas, permitindo uma organização estruturada e consistente.

Este modelo foi elaborado com base no esquema conceptual previamente definido, convertendo as entidades em relações, os atributos em colunas e os relacionamentos em chaves estrangeiras. O processo seguiu os princípios fundamentais da teoria relacional, nomeadamente a identificação de chaves primárias, a definição de dependências funcionais e a eliminação de redundâncias, garantindo a conformidade com as formas normais.

A modelação lógica foi realizada com o apoio da ferramenta *MySQL Workbench*, que permitiu a representação visual do modelo relacional, a verificação de integridade referencial e a preparação para a posterior implementação física da base de dados.

4.2 Apresentação e Explicação do Modelo Lógico Produzido

O modelo lógico apresentado resulta da conversão direta das entidades e relacionamentos definidos na fase de modelação conceptual.

As cinco entidades identificadas no modelo conceptual foram convertidas em tabelas relacionais, com os respetivos atributos e chaves primárias unívocas bem definidas. No caso do Cliente, todos os seus atributos simples, como NIF, email, dataNascimento, foram convertidos como colunas da tabela. Atributos compostos como morada, formada por rua, localidade e códigoPostal, foram representados pelos seus atributos constituintes.

Por outro lado, os relacionamentos N:1 e 1:N do modelo conceptual foram tratados através da inclusão de chaves estrangeiras nas tabelas envolvidas, seguindo as regras clássicas da modelação relacional. As chaves estrangeiras garantem a integridade referencial entre as tabelas, sendo suportadas pelas relações de dependência lógica do sistema. Desta forma, o relaci-

onamento binário “realiza” entre Cliente e Aluguer é refletido através da chave estrangeira idCliente em Aluguer, bem como, o relacionamento “processa” entre Funcionário e Aluguer originou a chave estrangeira idFuncionário em Aluguer.

Para facilitar a interpretação do modelo lógico, foi incluída uma legenda que identifica claramente os relacionamentos, do modelo conceptual, através das chaves estrangeiras. Esta legenda ajuda a visualizar de forma imediata como as entidades se conectam entre si no sistema.

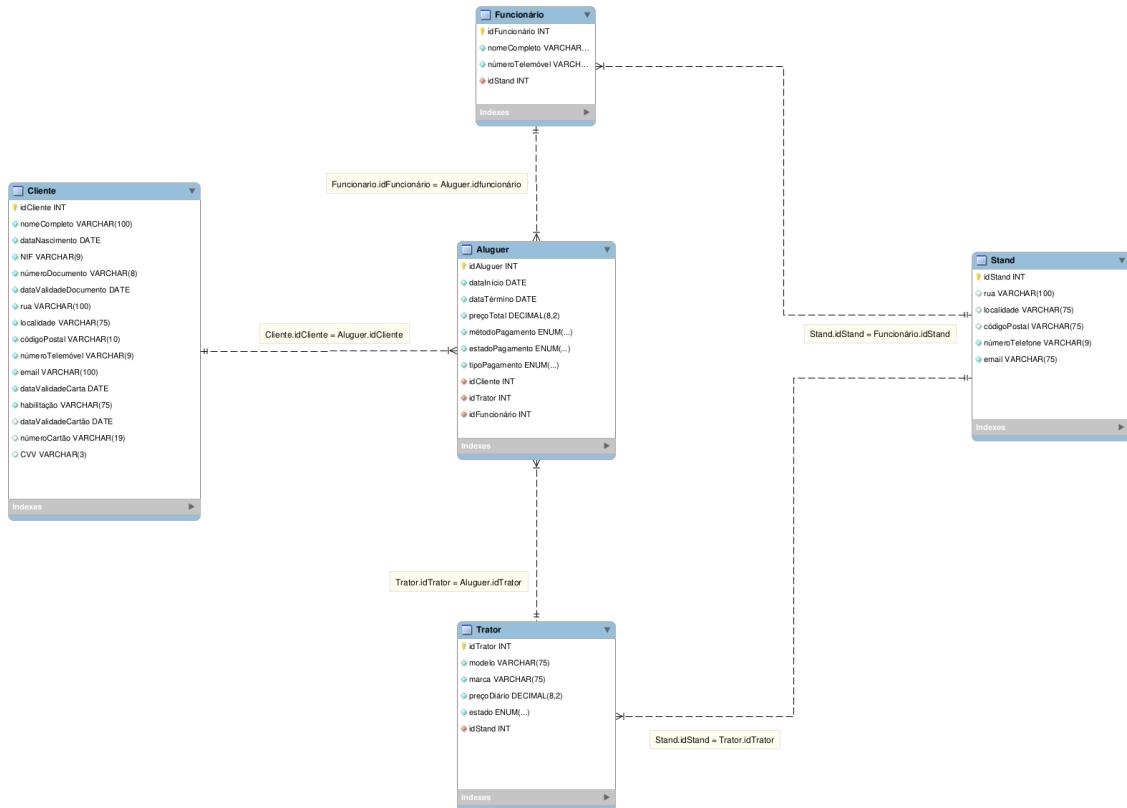


Figura 4.1: Modelo Lógico do Sistema de Gestão de Alugueres de Tratores

4.3 Normalização de Dados

O modelo lógico encontra-se normalizado até à Terceira Forma Normal (3FN). Esta normalização foi aplicada com o objetivo de garantir a redução de redundâncias, evitar anomalias de inserção, atualização ou eliminação e assegurar a consistência dos dados ao longo do tempo.

Todas as tabelas apresentam atributos atómicos, ou seja, cada campo armazena um único valor indivisível. Por exemplo, o atributo composto morada foi devidamente decomposto nos seus componentes: rua, localidade e códigoPostal, satisfazendo assim a Primeira Forma Normal

(1FN).

Como todas as tabelas têm chaves primárias simples, e todos os atributos não-chave dependem totalmente da chave primária, concluímos que a 2FN é satisfeita. Exemplificando, na tabela Aluguer, as colunas como dataInício, preçoTotal ou métodoPagamento dependem exclusivamente do idAluguer.

Além disso, não existem dependências transitivas entre atributos não-chave. Todos os atributos não-chave dependem apenas da chave primária e não de outros atributos não-chave, o que nos permite concluir que a Terceira Forma Normal (3FN) é respeitada. Por exemplo, na tabela Cliente, o email ou o númeroCartão não dependem de outro atributo não-chave como nomeCompleto, mas apenas da chave primária idCliente.

4.4 Validação do Modelo com Interrogações do Utilizador

Com base nos requisitos previamente definidos, procedeu-se à formulação de um conjunto de interrogações representativas das funcionalidades do sistema. Estas *queries* foram utilizadas para validar o modelo relacional, assegurando que este é capaz de responder corretamente às necessidades de informação identificadas.

Query 1

A *query* 1 tem como objetivo apresentar, diariamente, uma lista de todos os tratores que se encontram alugados, agrupando essa informação por stand, ordenados alfabeticamente pela marca do mesmo. Esta *query* é formulada a partir do requisito **RM02**.

Para atingir este resultado, a expressão em álgebra relacional é construída por etapas lógicas. Primeiramente, é aplicada uma operação de seleção sobre a tabela Trator para filtrar apenas os tratores cujo estado seja 'Alugado'. Esta operação assegura que apenas são considerados os tratores que estão atualmente alugados.

Os tratores selecionados são depois associados à tabela Stand através de uma operação de junção natural baseada na coluna idStand de ambas as tabelas. Esta junção permite relacionar cada trator com o stand a que pertence.

De seguida é feita uma projeção das colunas relevantes para a expressão final.

Por fim, os dados resultantes são ordenados pela coluna idStand, para agrupar os tratores por stand e, referente a cada stand, pela marca do trator, alfabeticamente, conforme solicitado no requisito.

A seguir, apresenta-se a árvore de demonstração da expressão algébrica, detalhando as operações realizadas passo a passo até à obtenção do resultado final da consulta.

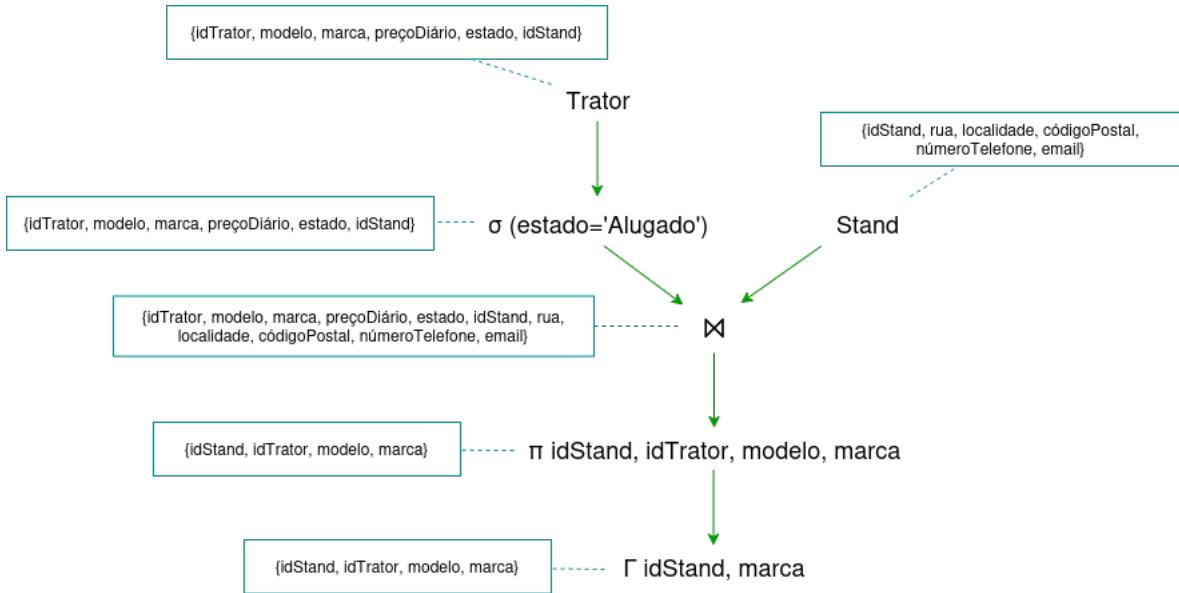


Figura 4.2: Árvore de demonstração Q1

A expressão em álgebra relacional que traduz esta *query* e reflete a sequência lógica das operações realizadas é:

$$Q1 \leftarrow \tau_{\text{idStand}, \text{marca}} \left(\pi_{\text{idStand}, \text{idTrator}, \text{modelo}, \text{marca}} \left(\sigma_{\text{estado}=\text{'Alugado'}}(\text{Trator}) \bowtie \text{Stand} \right) \right)$$

Query 2

A *query* 2, baseada no requisito **RM04**, tem como objetivo verificar se um cliente, ainda sem nenhum aluguer associado, está apto a realizar um aluguer. Para isso, o sistema avalia se a carta de condução do cliente continua válida até à data de término pretendida do aluguer e se a habilitação do cliente é do tipo 'T', necessária para conduzir tratores. O resultado apresenta o identificador e o nome completo do cliente, caso este reúna as condições necessárias para efetuar um aluguer.

Para realizar esta verificação, a *query* utiliza a variável `idClientePretendido` para identificar o cliente que pretende realizar o aluguer e a variável `dataTérminoPretendida` que indica a data prevista para o final do aluguer pretendido.

Desta forma, a expressão seleciona na tabela **Cliente** os diferentes clientes que satisfazem três condições em simultâneo: o cliente que está a ser verificado tem de ter o mesmo identificador que o `idClientePretendido`, a carta de condução tem de ter validade até, ou para além, da

data em que terminaria o aluguer pretendido e a habilitação do cliente tem de ser do tipo 'T', correspondente ao tipo necessário para conduzir tratores.

Depois é feita uma projeção das colunas que realmente interessam para a expressão final, apresentando assim a informação do cliente apenas se cumprir todos os critérios.

A seguinte árvore de demonstração espelha os passos que o sistema segue para alcançar a expressão final.

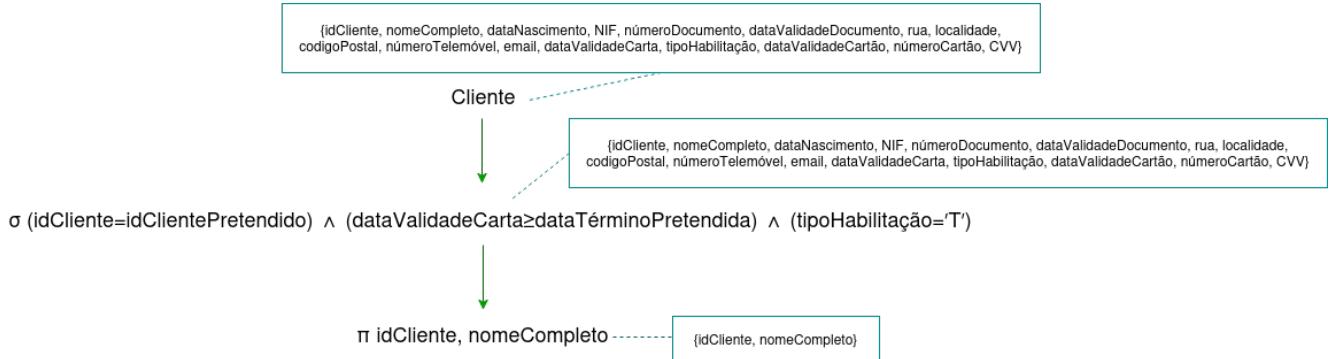


Figura 4.3: Árvore de demonstração Q2

Segue-se a expressão em álgebra relacional que traduz a *query* enunciada:

$$Q2 \leftarrow \pi_{idCliente, nomeCompleto} \left(\sigma_{(idCliente=idClientePretendido) \wedge (dataValidadeCarta \geq dataTérminoPretendida) \wedge (tipoHabilitação='T')} (\text{Cliente}) \right)$$

Query 3

A *query* 3, baseada no requisito **RM10**, tem como objetivo determinar o funcionário que realizou o maior número de alugueres num determinado mês.

Para ser possível adaptar o intervalo de análise mensal, a expressão utiliza variáveis que representam os limites temporais do mês em questão, dataInícioMes e dataFimMes.

A expressão começa por aplicar uma operação de seleção sobre a tabela Aluguer para filtrar apenas os regtos cujas datas de início se encontrem dentro do intervalo temporal definido.

Em seguida, efetua-se uma junção natural entre os alugueres filtrados e a tabela Funcionário, permitindo associar cada aluguer ao funcionário responsável pela sua realização.

Após a junção, é criada uma relação chamada TotaisPorFuncionário que aplica uma operação de agregação pelos identificadores dos funcionários, calculando, para cada um, o total de

alugueres realizados naquele período. O resultado deste cálculo é armazenado na coluna totalAlugueres.

Em seguida, é criada uma segunda relação denominada ValorMáximo, que calcula o valor máximo entre todos os totais de vendas dos diferentes funcionários, atribuindo-lhe o nome maxAlugueres.

Por fim, é realizada uma operação de produto cartesiano entre TotaisPorFuncionário e ValorMáximo, seguida de uma seleção para isolar o funcionário cujo totalAlugueres é igual a maxAlugueres, e uma projeção para obter apenas o identificador desse funcionário encontrando, desta forma, o funcionário que realizou o maior número de alugueres no período especificado.

A seguinte árvore de demonstração espelha o processo de criação da expressão em álgebra relacional.

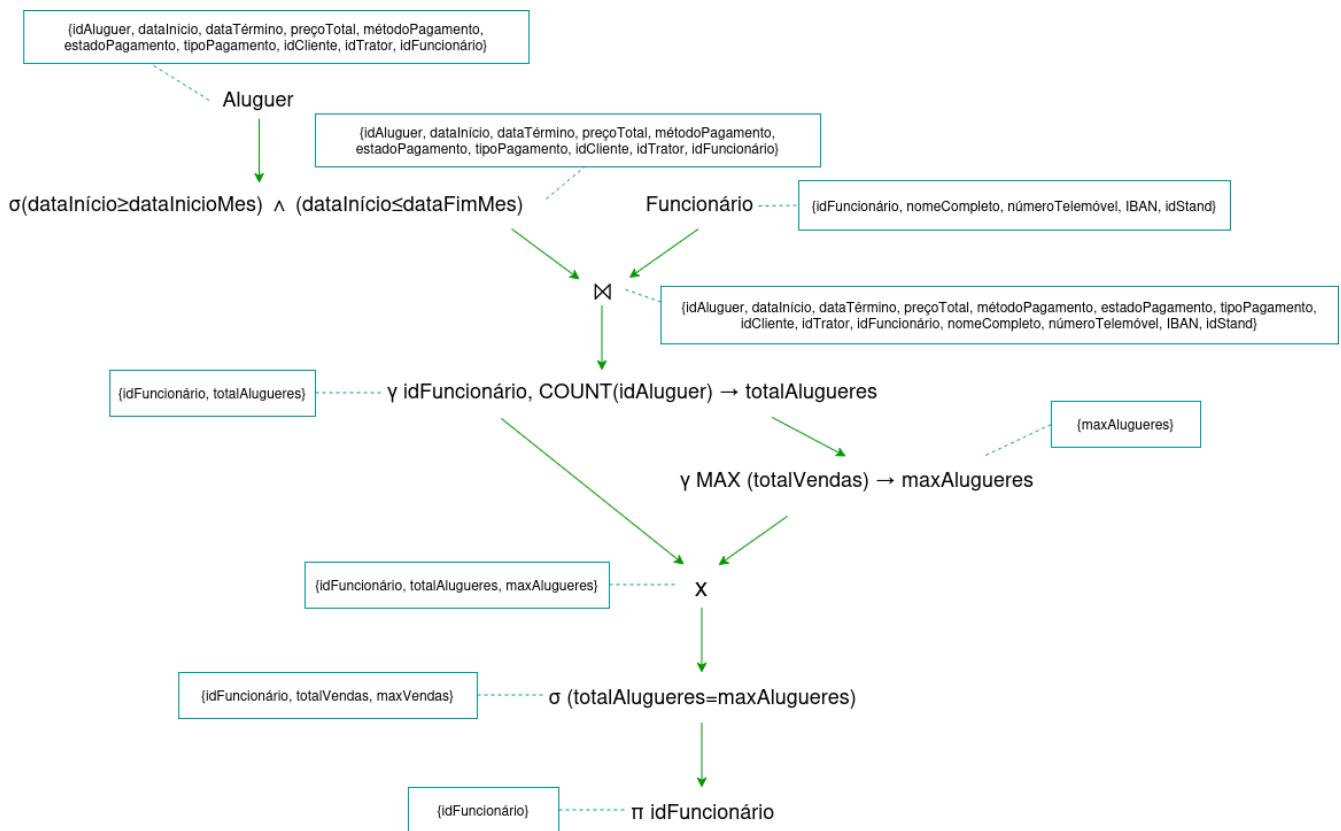


Figura 4.4: Árvore de demonstração Q3

A expressão em álgebra relacional que traduz esta *query* e reflete a sequência lógica das operações realizadas é:

$$\text{TotaisPorFuncionário} \leftarrow \gamma_{\text{idFuncionário}, \text{COUNT}(\text{idAluguer}) \rightarrow \text{totalAlugueres}} \left((\sigma_{(\text{dataInício} \geq \text{dataInícioMes}) \wedge (\text{dataInício} \leq \text{dataFimMes})}(\text{Aluguer})) \bowtie \text{Funcionário} \right)$$

$$\text{ValorMáximo} \leftarrow \gamma_{\text{MAX}(\text{totalAlugueres}) \rightarrow \text{maxAlugueres}}(\text{TotaisPorFuncionário}) Q3 \leftarrow \pi_{\text{idFuncionário}} \left(\sigma_{\text{totalAlugueres} = \text{máximo}} \right)$$

Query 4

A query 4 tem como objetivo determinar o número total de alugueres realizados pela empresa durante o último trimestre. Esta query é formulada a partir do requisito **RM11**.

Para definir o intervalo correspondente ao trimestre em análise, são utilizadas duas variáveis. A variável `dataInícioTrimestre` representa a data de início do trimestre e a variável `dataFimTrimestre` representa a data de fim do trimestre. Estas variáveis permitem a adaptação da query para qualquer trimestre do ano.

Primeiramente, é aplicada uma operação de seleção sobre a tabela `Aluguer` para considerar apenas os registo cuja data de início esteja dentro do intervalo temporal definido pelas variáveis temporais,

Em seguida, é aplicada uma operação de agregação que conta o número de ocorrências `idAluguer`, permitindo assim calcular o total de alugueres efetuados no período em questão. O resultado desta contagem é renomeado como `totalAlugueres`.

A seguinte árvore de demonstração reflete a sequência lógica das operações realizadas:

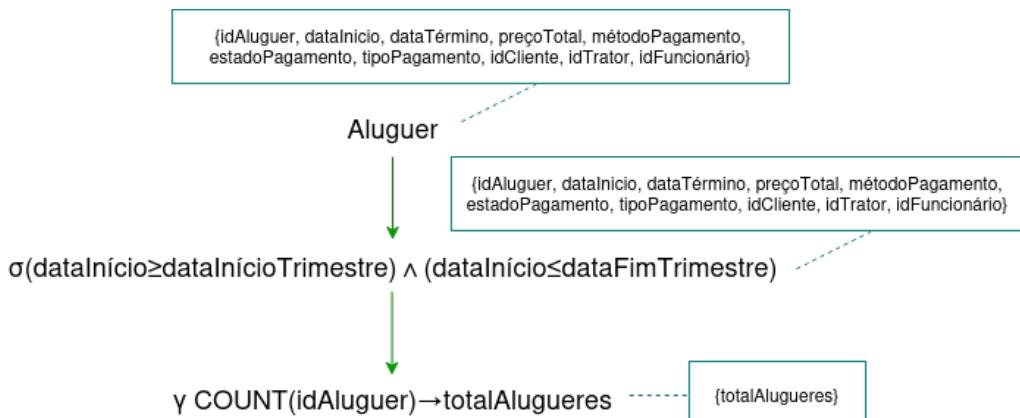


Figura 4.5: Árvore de demonstração Q4

A seguinte expressão permite atingir o objetivo determinado pela *query* enunciada.

$$Q4 \leftarrow \gamma_{\text{COUNT}(\text{idAluguer}) \rightarrow \text{totalAlugueres}} \left(\sigma_{(\text{dataInício} \geq \text{dataInícioTrimestre}) \wedge (\text{dataInício} \leq \text{dataFimTrimestre})} (\text{Aluguer}) \right)$$

Query 5

A *query* 5, baseada no requisito **RM12**, tem como objetivo calcular o valor total faturado em cada stand no final de um determinado mês.

Para ser possível adaptar o intervalo de análise mensal, a expressão utiliza variáveis que representam os limites temporais do mês em questão, *dataInícioMes* e *dataFimMes*.

Primeiramente, é aplicada uma operação de seleção sobre a tabela *Aluguer* para filtrar apenas os registos cuja data de início se encontre dentro do intervalo temporal do mês em análise.

De seguida, os alugueres filtrados são associados à tabela *Trator* através de uma operação de junção natural. Esta junção permite associar cada aluguer ao respetivo trator e, por extensão, ao stand ao qual o trator pertence, através do atributo *idStand*.

Após a junção, é aplicada uma projeção de forma a obter apenas os atributos relevantes para o cálculo: o identificador do stand, *idStand*, e o valor total de cada aluguer, *preçoTotal*.

Por fim, é efetuada uma agregação sobre os dados, calculando para cada stand o total faturado durante o mês. Este total é obtido através da soma dos valores individuais de cada aluguer, *preçoTotal*, e é armazenado no atributo *totalFaturado*.

A seguir, apresenta-se a árvore de demonstração da expressão algébrica, detalhando as operações realizadas passo a passo até à obtenção do resultado final da consulta.

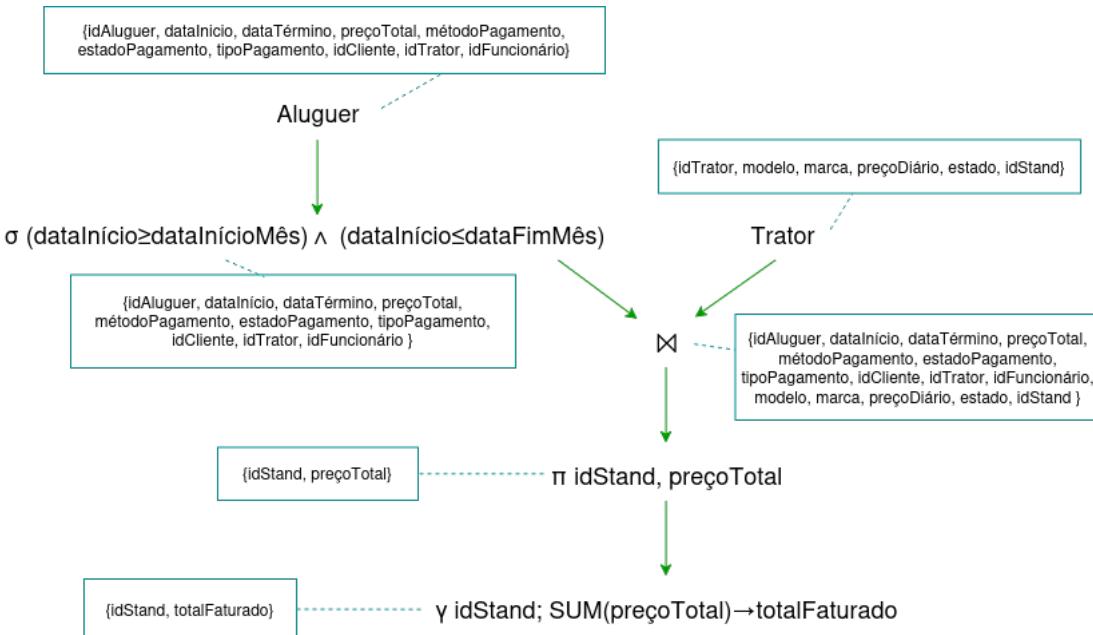


Figura 4.6: Árvore de demonstração Q5

A seguinte expressão traduz o requisito identificado anteriormente em álgebra relacional:

$$Q5 \leftarrow \gamma_{idStand, SUM(preçoTotal) \rightarrow totalFaturado} \left(\pi_{idStand, preçoTotal} \left(\sigma_{(dataInício \geq dataInícioMes) \wedge (dataInício \leq dataFimMes)} (Aluguer) \bowtie Trator \right) \right)$$

Com o objetivo de garantir que o modelo relacional definido é capaz de suportar eficazmente todas as ações de exploração previstas, procedeu-se à elaboração de um mapa de interrogações, no qual foram analisadas cinco *queries* representativas das funcionalidades do sistema. Esta análise permitiu verificar que os dados necessários para responder às diferentes interrogações se encontram devidamente representados e relacionados no esquema, assegurando a integridade, a consistência e a eficiência das operações de consulta.

Simultaneamente, tornou-se possível identificar quais as tabelas mais frequentemente envolvidas nas interrogações, avaliando assim o seu impacto sobre o desempenho global do sistema. Verificou-se que as tabelas Cliente, Aluguer e Trator são recorrentemente acedidas, estando envolvidas em diversas operações de junção natural, projeção e seleção.

Durante a análise, foram ainda identificados potenciais pontos de estrangulamento no sistema, particularmente em *queries* que implicam grandes volumes de dados. Por exemplo, a query 1 que apresenta, diariamente, os tratores alugados agrupados por stand e ordenados por marca pode ser exigente em termos de desempenho, especialmente se houver muitos registo ou se não existirem índices adequados nas colunas usadas nas operações de junção natural, ordenação e agregação.

Foi igualmente possível identificar oportunidades para otimização do modelo relacional através da criação de atributos derivados. Um exemplo disso é a necessidade frequente de verificar se um cliente já realizou algum aluguer, operação que implica contar registo na tabela Aluguer de forma repetida. Para evitar esse custo computacional, poderá ser vantajoso criar um atributo derivado atualizado automaticamente a cada operação de aluguer.

5 Revisões e Melhorias na Primeira Fase do Projeto

Durante uma reunião com o Sr. Pires, no contexto de uma análise geral das necessidades operacionais e de segurança da AgroAuto, foram assinaladas algumas preocupações relativas ao controlo de acessos e à integridade da informação. Estas observações, embora não técnicas, serviram de ponto de partida para uma revisão mais aprofundada, realizada pelo engenheiro da base de dados, de forma a assegurar a conformidade com as melhores práticas de modelação e segurança da informação.

Inclusão de um novo requisito de controlo:

Durante a análise detalhada dos requisitos de controlo inicialmente definidos, RC01 a RC14, e das *queries* planeadas para implementação no sistema, verificámos que, embora a maioria das operações de leitura e escrita sobre os dados estivessem devidamente enquadradas pelos requisitos existentes, havia uma lacuna no que respeita ao acesso à tabela Stand.

Mais concretamente, a *query* 5, cuja função é calcular o valor total faturado por cada stand no final de um determinado mês, revelou a necessidade de aceder diretamente a dados agregados por stand. Esta funcionalidade não estava contemplada em nenhum dos requisitos de controlo definidos até então, sendo que nenhum deles previa explicitamente o acesso administrativo à tabela Stand, nem a qualquer métrica de faturação por stand.

Dado que a faturação representa informação sensível, de natureza estratégica para a gestão da empresa, considerou-se adequado que apenas o administrador tivesse acesso a essa informação, o que está de acordo com a lógica de segurança já aplicada nos requisitos RC02 a RC08, onde apenas o administrador pode consultar métricas agregadas ou sensíveis como lucros, históricos de alugueres ou desempenho de funcionários.

Assim, foi acrescentado o novo requisito de controlo: “Apenas o administrador pode consultar o valor total faturado por cada stand ao final de um mês.”

Alteração do tipo de dados dos atributos dataValidadeCarta e dataValidadeCartão:

Durante a fase de desenvolvimento e preparação das *queries* da base de dados, foi identificado que determinadas colunas originalmente definidas com o tipo de dados VARCHAR, nomeadamente dataValidadeCarta e dataValidadeCartão, estavam a dificultar a implementação de operações relacionadas com a comparação e cálculo de datas.

Estas colunas representam datas de validade associadas à carta de condução dos clientes e aos seus cartões de identificação, respetivamente, sendo essenciais para a execução correta de funcionalidades como a verificação da aptidão de um cliente para alugar um trator, tal como definido na *query* 2.

Manter estas datas em formato VARCHAR implicaria conversões constantes para o tipo DATE em cada *query*, aumentando não só a complexidade do código SQL, mas também o risco de erros e inconsistências, especialmente em cenários com diferentes formatos de data. Além disso, tal abordagem poderia comprometer a eficiência das consultas, uma vez que as funções de conversão impedem a utilização eficaz de índices sobre essas colunas.

Nesse sentido, optou-se por alterar o tipo de dados de ambas as colunas para DATE, permitindo uma gestão mais robusta e segura da informação temporal.

Alteração do tipo de dados dos atributos `métodoPagamento`, `estadoPagamento`, `tipoPagamento` e `estadoTrator`:

Foi decidido alterar o tipo de dados dos atributos `métodoPagamento`, `estadoPagamento`, `tipoPagamento` e `estadoTrator` de VARCHAR para ENUM, com o objetivo de reforçar a consistência e integridade dos dados. Cada um destes campos admite apenas um conjunto restrito de valores válidos e bem definidos, por exemplo, o `métodoPagamento` pode ser "Cartão de Crédito" ou "Dinheiro", enquanto o `estadoPagamento` pode assumir os valores "Em atraso" ou "Concluído". Ao utilizar o tipo ENUM, restringimos os valores possíveis apenas às opções previamente definidas, o que previne erros de inserção, como valores inválidos ou incorretamente escritos. Além disso, esta abordagem facilita a validação dos dados e contribui para uma maior legibilidade e manutenção do sistema.

Alteração do tipo de dados de INT para VARCHAR:

No contexto da base de dados AgroAuto, foi necessário realizar alterações aos tipos de dados de certos atributos inicialmente definidos como inteiros, INT, nomeadamente `numeroCartao`, `CVV`, `NIF`, `numeroDocumento` e `numeroTelemovel`, substituindo-os por campos do tipo VARCHAR. Esta decisão prende-se com o facto destes dados representarem identificadores e não valores numéricos sobre os quais se realizem operações matemáticas. Por exemplo, os números de cartão de crédito podem ter até 19 dígitos, o que excede a capacidade segura de tipos como INT e pode originar perda de precisão ou truncamentos. Além disso, estes valores podem conter zeros à esquerda, informação que é automaticamente descartada em campos numéricos, mas preservada em VARCHAR. Por estas razões, a alteração para VARCHAR garante maior fiabilidade, flexibilidade e conformidade com as práticas recomendadas de modelação de dados para este tipo de informação sensível e identificativa.

Inclusão de um novo requisito de manipulação:

A inclusão de um requisito de manipulação tornou-se necessária para assegurar que o processo de importação de dados, a partir de ficheiros CSV, JSON e de bases de dados relacionais como o PostgreSQL neste caso, seja realizado de forma fiável e controlada. Estes dados

externos podem apresentar variações na estrutura, tipos de dados inconsistentes, campos em falta, valores inválidos ou duplicados, o que exige um tratamento adequado antes da sua integração no sistema. O requisito de manipulação garante que o sistema é capaz de mapear corretamente os dados externos para o modelo interno, aplicar transformações e validações conforme as regras de negócio, e gerir eventuais erros durante o processo. Sem este requisito, a importação poderia comprometer a integridade da base de dados e afetar o funcionamento do sistema.

Assim, foi acrescentado um novo requisito de manipulação: “O sistema permite a importação de informações provenientes de ficheiros CSV, JSON e de bases de dados relacionais, nomeadamente PostgreSQL”

Atualização do Diagrama de GANTT:

Inicialmente, o diagrama foi elaborado com base na primeira fase do projeto, mas, com a implementação da segunda fase, tornou-se necessário atualizá-lo para refletir as novas tarefas, prazos e dependências introduzidas. Esta atualização garantiu que o planeamento se mantivesse alinhado com o progresso real do projeto, permitindo uma melhor organização e controlo do trabalho realizado.

6 Implementação Física

6.1 Apresentação e explicação da base de dados implementada

A implementação física da base de dados foi realizada no sistema de gestão *MySQL*, utilizando o *MySQL Workbench* como ferramenta de modelação e execução das definições estruturais. Esta plataforma permitiu a tradução direta do modelo lógico previamente desenvolvido e validado, garantindo que a estrutura final mantivesse coerência com o esquema lógico já estabelecido e validado.

O processo de construção do esquema físico foi conduzido de forma metódica, envolvendo a criação das tabelas e a definição precisa de chaves primárias e estrangeiras. Esta etapa revelou-se fundamental para transformar o desenho lógico numa solução funcional, estruturada e coerente, proporcionando uma base sólida para suportar as operações de gestão de alugueres de tratores no contexto da empresa AgroAuto.

Cada tabela foi concebida com foco na integridade dos dados, garantindo que todas as relações entre entidades foram corretamente modeladas. As chaves primárias foram definidas como auto-incrementadas, assegurando a unicidade dos registos de forma automática. As ligações entre tabelas foram estabelecidas através de chaves estrangeiras, garantindo a integridade referencial entre as entidades do modelo de dados. Cada chave estrangeira foi definida para reforçar a consistência das relações e assegurar que todas as associações no sistema respeitam as regras de integridade impostas pelo esquema lógico.

Segue-se, abaixo, a descrição do processo de criação de cada uma das tabelas da base de dados do sistema da AgroAuto.

Tabela Cliente

A tabela Cliente foi criada para armazenar os dados pessoais e administrativos dos clientes da AgroAuto. Contempla atributos como `idCliente`, definido como chave primária e auto-incrementado, `nomeCompleto`, `dataNascimento`, `NIF`, `contactos` e informações relacionadas com os documentos de identificação e pagamento, como `numeroDocumento`, `dataValidadeDocumento`, `numeroCartao` e `CVV`. Os campos essenciais foram definidos com a restrição NOT NULL, assegurando a consistência dos dados inseridos.

Os únicos campos que permitem valores NULL são a `dataValidadeCartão`, o `numeroCartao` e o `CVV`, por se tratarem de informações opcionais, apenas necessárias quando o cliente opta por pagar com cartão de crédito. Esta decisão foi tomada para garantir flexibilidade no armazenamento dos dados, já que nem todos os clientes utilizarão este método de pagamento.

```
CREATE TABLE IF NOT EXISTS `AgroAuto`.`Cliente` (
  `idCliente` INT NOT NULL AUTO_INCREMENT,
  `nomeCompleto` VARCHAR(100) NOT NULL,
  `dataNascimento` DATE NOT NULL,
  `NIF` VARCHAR(9) NOT NULL,
  `numeroDocumento` VARCHAR(8) NOT NULL,
  `dataValidadeDocumento` DATE NOT NULL,
  `rua` VARCHAR(100) NOT NULL,
  `localidade` VARCHAR(75) NOT NULL,
  `codigoPostal` VARCHAR(10) NOT NULL,
  `numeroTelemovel` VARCHAR(9) NOT NULL,
  `email` VARCHAR(100) NOT NULL,
  `dataValidadeCarta` DATE NOT NULL,
  `habilitacao` VARCHAR(75) NOT NULL,
  `dataValidadeCartao` DATE NULL,
  `numeroCartao` VARCHAR(19) NULL,
  `CVV` VARCHAR(3) NULL,
  PRIMARY KEY (`idCliente`));
```

Figura 6.1: Tabela Cliente SQL

Tabela Stand

Em seguida, a tabela Stand foi estruturada para representar as diferentes filiais da empresa AgroAuto. Cada stand corresponde a um ponto de atendimento, possuindo atributos como `idStand` (a chave primária), `rua`, `localidade`, `codigoPostal`, `numeroTelefone` e `email`.

O identificador do stand desempenha um papel fundamental na integridade do sistema, sendo utilizado como chave estrangeira noutras tabelas, como `Trator`, assegurando que cada trator pertence a um stand específico, e `Aluguer`, onde a relação entre os clientes e as filiais é devidamente controlada. Esta ligação garante que todas as operações de aluguer e gestão de frotas estejam corretamente associadas às unidades físicas da empresa, permitindo consultas eficientes e uma organização precisa da informação. O respetivo código pode ser consultado abaixo.

```

CREATE TABLE IF NOT EXISTS `AgroAuto`.`Stand` (
    `idStand` INT NOT NULL AUTO_INCREMENT,
    `rua` VARCHAR(100) NULL,
    `localidade` VARCHAR(75) NULL,
    `codigoPostal` VARCHAR(75) NULL,
    `numeroTelefone` VARCHAR(9) NOT NULL,
    `email` VARCHAR(75) NOT NULL,
    PRIMARY KEY (`idStand`));

```

Figura 6.2: Tabela Stand SQL

Tabela Trator

A tabela Trator foi desenvolvida para armazenar os dados dos tratores disponíveis na AgroAuto, garantindo um registo preciso da frota da empresa. Inclui atributos essenciais, como **idTrator** (chave primária), **modelo**, **marca**, **preço diário** e **estado**, além da chave estrangeira **idStand**, que define o stand onde cada unidade está alocada.

Para assegurar a consistência dos dados, o campo **estado** foi implementado com o tipo **ENUM**, permitindo apenas os valores '**Livre**' e '**Alugado**'. Esta abordagem evita inserções inválidas e facilita a classificação do status dos tratores no sistema.

A seguir, apresenta-se a estrutura detalhada da tabela e a sua implementação.

```

CREATE TABLE IF NOT EXISTS `AgroAuto`.`Trator` (
    `idTrator` INT NOT NULL AUTO_INCREMENT,
    `modelo` VARCHAR(75) NOT NULL,
    `marca` VARCHAR(75) NOT NULL,
    `precoDiario` DECIMAL(8,2) NOT NULL,
    `estado` ENUM('Livre', 'Alugado') NOT NULL,
    `idStand` INT NOT NULL,
    PRIMARY KEY (`idTrator`),
    FOREIGN KEY (`idStand`) REFERENCES `AgroAuto`.`Stand` (`idStand`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);

```

Figura 6.3: Tabela Trator SQL

Tabela Funcionário

A tabela Funcionario foi criada para armazenar os dados dos colaboradores da AgroAuto, garantindo um registo preciso da equipa disponível em cada stand. Inclui os atributos **idFuncionario** (chave primária), **nomeCompleto**, **numeroTelemovel** e a chave estrangeira **idStand**, que associa cada funcionário à sua unidade de trabalho.

Esta relação entre funcionário e filial é essencial para identificar quais colaboradores estão

alocados a cada localização. A seguir, apresenta-se o código correspondente à sua implementação.

```
CREATE TABLE IF NOT EXISTS `AgroAuto`.`Funcionario` (
    `idFuncionario` INT NOT NULL AUTO_INCREMENT,
    `nomeCompleto` VARCHAR(75) NOT NULL,
    `numeroTelemovel` VARCHAR(9) NOT NULL,
    `idStand` INT NOT NULL,
    PRIMARY KEY (`idFuncionario`),
    FOREIGN KEY (`idStand`) REFERENCES `AgroAuto`.`Stand` (`idStand`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);
```

Figura 6.4: Tabela Funcionário SQL

Tabela Aluguer

A tabela Aluguer é responsável por registar todas as operações de aluguer realizadas na plataforma. Para garantir um acompanhamento detalhado de cada transação, esta tabela inclui campos como idAluguer, dataInicio, dataTermino, precoTotal, que permitem rastrear o período de aluguer e o valor pago pelo cliente. A tabela Aluguer também estabelece relações essenciais com as entidades Cliente, Trator e Funcionario através de chaves estrangeiras, garantindo a integridade referencial no sistema. Destaca-se a aplicação da restrição ON DELETE SET NULL na referência ao funcionário, permitindo que um aluguer continue válido mesmo que o colaborador responsável pela operação seja removido da base de dados. Esta abordagem preserva o histórico de transações e evita inconsistências nos registo.

```
CREATE TABLE IF NOT EXISTS `AgroAuto`.`Aluguer` (
    `idAluguer` INT NOT NULL AUTO_INCREMENT,
    `dataInicio` DATE NOT NULL,
    `dataTermino` DATE NOT NULL,
    `precoTotal` DECIMAL(8,2) NOT NULL,
    `metodoPagamento` ENUM('CartaoCredito', 'Dinheiro') NOT NULL,
    `estadoPagamento` ENUM('EmAtraso', 'Concluido') NOT NULL,
    `tipoPagamento` ENUM('APronto', 'EmPrestacoes') NOT NULL,
    `idCliente` INT NOT NULL,
    `idTrator` INT NOT NULL,
    `idFuncionario` INT NOT NULL,
    PRIMARY KEY (`idAluguer`),
    FOREIGN KEY (`idCliente`) REFERENCES `AgroAuto`.`Cliente` (`idCliente`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION,
    FOREIGN KEY (`idTrator`) REFERENCES `AgroAuto`.`Trator` (`idTrator`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION,
    FOREIGN KEY (`idFuncionario`) REFERENCES `AgroAuto`.`Funcionario` (`idFuncionario`)
    ON DELETE RESTRICT
    ON UPDATE NO ACTION);
```

Figura 6.5: Tabela Aluguer SQL

6.2 Criação de utilizadores da base de dados

No desenvolvimento da base de dados, foi essencial definir diferentes utilizadores com níveis de acesso distintos, de forma a garantir a segurança, a organização e o controlo sobre os dados armazenados.

No contexto da AgroAuto, foram definidos dois perfis diferentes de utilização, refletindo os requisitos de controlo identificados e estabelecidos anteriormente. A adoção de um modelo baseado em *roles* oferece uma abordagem escalável e eficiente à gestão de permissões. Ao centralizar os privilégios em perfis lógicos de utilização, torna-se possível atribuir, modificar ou anular permissões a um grupo de utilizadores de forma uniforme, sem necessidade de intervenção individual sobre cada conta.

A criação destes utilizadores foi orientada tendo em conta que, inicialmente, nenhum utilizador possuía acesso a qualquer recurso da base de dados. Atribuíram-se, assim, apenas as permissões estritamente necessárias para o desempenho das funções de cada perfil, em conformidade com os requisitos de controlo definidos.

Funcionário

No sistema AgroAuto, a *role* funcionário foi criada para representar o perfil de um colaborador da empresa, com responsabilidades centradas na gestão de clientes e na operação dos processos de aluguer de tratores.

```
-- DROP ROLE IF EXISTS 'funcionario';
CREATE ROLE 'funcionario';
```

Figura 6.6: *Role* Funcionário

Esta função foi atribuída aos três trabalhadores atuais da AgroAuto: Diogo Costa, Fatima Teixeira e Fábio Vieira, *dcosta*, *fteixeira* e *fvieira*, respetivamente. Estes utilizadores desempenham as mesmas tarefas no sistema, beneficiando, assim, de um conjunto uniforme de permissões adequadas às suas funções. Todos os utilizadores foram criados localmente, com palavra-passe segura, e configurados para assumirem por defeito a *role* funcionário.

```

-- DROP USER IF EXISTS 'dcosta'@'localhost';
CREATE USER 'dcosta'@'localhost' IDENTIFIED BY 'senhaDiogo123!';
-- DROP USER IF EXISTS 'fteixeira'@'localhost';
CREATE USER 'fteixeira'@'localhost' IDENTIFIED BY 'senhaFatima123!';
-- DROP USER IF EXISTS 'fvieira'@'localhost';
CREATE USER 'fvieira'@'localhost' IDENTIFIED BY 'senhaFabio123';

GRANT 'funcionario' TO 'dcosta'@'localhost';
SET DEFAULT ROLE 'funcionario' TO 'dcosta'@'localhost';

GRANT 'funcionario' TO 'fteixeira'@'localhost';
SET DEFAULT ROLE 'funcionario' TO 'fteixeira'@'localhost';

GRANT 'funcionario' TO 'fvieira'@'localhost';
SET DEFAULT ROLE 'funcionario' TO 'fvieira'@'localhost';

```

Figura 6.7: Criação de utilizadores e atribuição da *role* funcionário a estes

Entre os requisitos identificados para o perfil de funcionário, destaca-se a necessidade de consultar, de forma eficiente, a disponibilidade de tratores. Para tal, foi desenvolvida a *view* TratoresDisponiveis, que permite aceder rapidamente a essa informação, referente ao RC11. Adicionalmente, para possibilitar a análise detalhada do histórico de alugueres de um determinado cliente, foi implementada a *stored procedure* HistoricoAlugueresCliente, baseada no RC14. Para dar suporte à visualização diária da frota atualmente alugada por stand, foi criada a *view* TratoresAlugadosPorStandDiario, mencionada no RC10.

O perfil de funcionário tem ainda privilégios de consulta e atualização sobre clientes, RC09 e RC12, permitindo a gestão eficiente dos seus dados. Além disso, pode visualizar os alugueres existentes, com possibilidade de atualização quando necessário, baseado no RC13.

```

GRANT SELECT, UPDATE ON AgroAuto.Cliente TO 'funcionario';
GRANT SELECT, UPDATE ON AgroAuto.Aluguer TO 'funcionario';
GRANT SELECT ON AgroAuto.TratoresDisponiveis TO 'funcionario';
GRANT SELECT ON AgroAuto.TratoresAlugadosPorStandDiario TO 'funcionario';

GRANT EXECUTE ON PROCEDURE AgroAuto.HistoricoAlugueresCliente TO 'funcionario';

```

Figura 6.8: Atribuição de permissões ao *role* Funcionário

Administrador

Já a *role* administrador foi configurada para herdar os privilégios da *role* funcionário, através do mecanismo de herança do sistema de gestão de bases de dados. Esta herança garante que o Sr. Pires tem acesso a todas as operações básicas atribuídas aos funcionários, ao mesmo tempo que dispõe de permissões adicionais específicas.

```

-- DROP ROLE IF EXISTS 'administrador';
CREATE ROLE 'administrador';

-- DROP USER IF EXISTS 'admin'@'localhost';
CREATE USER 'admin'@'localhost' IDENTIFIED BY 'Admin123!';

GRANT 'funcionario' TO 'administrador';

GRANT 'administrador' TO 'admin'@'localhost';
SET DEFAULT ROLE 'administrador' TO 'admin'@'localhost';

```

Figura 6.9: *Role administrador*

Este utilizador possui permissões de leitura sobre diversas *views* como MarcaMaisAlugada e TotalFaturadoPorStandMensal, que correspondem aos requisitos de controlo RC04 e RC15, respetivamente. Além disso, tem permissões de execução sobre várias *stored procedures* operacionais, tais como ClientesMaisAtivos, LucroTotalAlugueres, FuncionariosComMaisAlugueresMes, requisitos RC03, RC08 e RC07, respetivamente, entre outras, permitindo consultas estratégicas e apoio à gestão da AgroAuto.

Para complementar as funcionalidades associadas aos requisitos de controlo previamente mencionados, foram também implementadas outras *views* e *procedures* que dão suporte a necessidades específicas do sistema, tais como consultas detalhadas, agregação de dados e automatização de processos frequentes.

```

GRANT SELECT ON AgroAuto.Funcionario TO 'administrador';
GRANT SELECT ON AgroAuto.MarcamaisAlugada TO 'administrador';
GRANT SELECT ON AgroAuto.TotalFaturadoPorStandMensal TO 'administrador';

GRANT EXECUTE ON PROCEDURE AgroAuto.ClientesMaisAtivos TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.TotalAlugueresPorTrimestre TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.FuncionariosComMaisAlugueresMes TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.LucroTotalAlugueres TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.RegistrosAlugueresFuncionario TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.VerificaClienteAptoAluguer TO 'administrador';
GRANT EXECUTE ON PROCEDURE AgroAuto.registarNovoAluguer TO 'administrador';

```

Figura 6.10: Atribuição de permissões ao *role administrador*

Todo o código SQL associado à implementação das *views* e *stored procedures* mencionadas neste tópico será detalhado em tópicos seguintes. Nos próximos tópicos, serão apresentadas as estruturas completas das *views* e *procedures*, acompanhadas de uma explicação sobre a sua funcionalidade e integração no modelo lógico da AgroAuto.

6.3 Povoamento da base de dados

O processo de povoamento da base de dados da AgroAuto foi realizado com o intuito de validar a estrutura relacional do modelo lógico e garantir que todas as tabelas estavam aptas a armazenar e a relacionar corretamente os dados.

Para garantir um povoamento coerente e estruturado, foram inseridos manualmente aproximadamente seis registos por tabela, utilizando dados fictícios, mas realistas e alinhados com a lógica do negócio estabelecida em requisitos anteriormente mencionados. Algumas tabelas, como Funcionário e Stand, receberam um número de entradas ajustado aos requisitos recolhidos na fase de análise, refletindo a dimensão operacional esperada.

O povoamento foi realizado por meio de instruções SQL do tipo `INSERT INTO`, executadas diretamente após a criação e configuração das tabelas. As inserções seguiram uma ordem sequencial, respeitando a hierarquia das dependências de integridade referencial entre as entidades. Desta forma, as entradas da tabela Cliente foram inseridas antes dos alugueres, garantindo que todos os registos de Aluguer possuíam um cliente válido, os tratores foram registados previamente, de modo que os alugueres pudessem referenciá-los corretamente e a tabela Funcionario foi povoada antes de ser associada aos alugueres, assegurando que todos os alugueres tivessem um funcionário responsável atribuído.

Tabela Cliente

Nesta tabela, foram inseridos seis registos distintos de clientes, cada um com dados únicos de identificação, morada e contacto, refletindo cenários realistas dentro do contexto da AgroAuto. Além das informações básicas, como nome, data de nascimento e número de identificação fiscal (NIF), foram incluídos atributos específicos para suportar o processo de aluguer.

A estrutura do povoamento foi planeada para assegurar integridade referencial, garantindo que os clientes inseridos possam ser referenciados em futuras operações de aluguer.

```

INSERT INTO `AgroAuto`.`Cliente`
    (nomeCompleto, dataNascimento, NIF, numeroDocumento, dataValidadeDocumento, rua, localidade, CódigoPostal, numeroTelemovel, email,
    dataValidadeCarta, habilitacao, dataValidadeCartao, numeroCartao, CVN)
VALUES
('João Azevedo', '2005-06-15', '123456789', '98765432', '2030-12-31', 'Rua das Flores', 'Braga', '4700-123', '912345678', 'joao.azevedo@gmail.com',
'2035-06-15', 'T', '2027-08-01', '1234123412341234', '123'),
('Maria Silva', '1990-02-20', '987654321', '12345678', '2028-10-10', 'Av. Central', 'Guimarães', '4800-456', '911234567', 'maria.silva@gmail.com',
'2032-02-20', 'T', '2026-04-01', '4321432143214321', '456'),
('Pedro Santos', '1978-11-03', '246813579', '11223344', '2027-07-30', 'Rua dos Pescadores', 'Viana do Castelo', '4900-321', '913456789', 'pedro.santos@hotmail.com',
'2028-11-03', 'B', '2025-12-31', '1111222233334444', '789'),
('Ana Freitas', '1995-04-18', '135792468', '55443322', '2029-03-15', 'Rua Nova', 'Porto', '4000-222', '914567890', 'ana.freitas@gmail.com',
'2030-04-18', 'A', '2027-10-10', '5555666677778888', '321'),
('Miguel Moreira', '1982-09-12', '112358132', '99887766', '2031-05-05', 'Av. das Indústrias', 'Lisboa', '1000-000', '915678901', 'miguel.moreira5@gmail.com',
'2033-09-12', 'T', '2028-09-01', '6666777788889999', '654'),
('Matilde Teixeira', '1998-12-25', '223344556', '66778899', '2032-11-11', 'Rua do Sol', 'Coimbra', '3000-111', '916789012', 'matilde2005.teixeira@gmail.com',
'2035-12-25', 'B', '2029-06-30', '7777888899990000', '987');

```

Figura 6.11: Povoamento de Cliente

Tabela Stand

Nesta tabela, foram inseridos três registos, representando localizações estratégicas da Agro-Auto: Olivença, Lamego e Sagres. Cada entrada corresponde a um stand físico da empresa, permitindo a correta distribuição dos tratores e gestão dos alugueres conforme a filial responsável.

```

INSERT INTO `AgroAuto`.`Stand` (rua, localidade, codigoPostal, numeroTelefone, email)
VALUES
('Rua São Martinho da Bouça', 'Olivença', '4700-001', '253123123', 'olivença@agroauto.pt'),
('Rua Da Malha', 'Lamego', '8000-002', '289456456', 'lamego@agroauto.pt'),
('Rua Do Porto Seguro', 'Sagres', '7500-002', '247567456', 'sagres@agroauto.pt');

```

Figura 6.12: Povoamento de Stand

Tabela Funcionário

Foram inseridos três funcionários, cada um associado a um stand específico, conforme definido nos requisitos anteriormente. Estes colaboradores desempenham funções essenciais na gestão dos alugueres e no atendimento aos clientes, garantindo o funcionamento eficiente de cada filial.

```

INSERT INTO `AgroAuto`.`Funcionario` (nomeCompleto, numeroTelemovel, idStand)
VALUES
('Diogo Costa',      '910000001', 1),
('Fátima Teixeira',  '910000002', 2),
('Fábio Vieira',     '910000003', 3);

```

Figura 6.13: Povoamento de Funcionário

Tabela Trator

Foram registados seis tratores, com modelos e marcas distintas e devidamente associado a um stand específico. Estes foram distribuídos por diferentes stands, garantindo que cada filial possui pelo menos um trator alocado, conforme definido nos requisitos.

```

INSERT INTO `AgroAuto`.`Trator` (modelo, marca, precoDiario, estado, idStand)
VALUES
('T8.410',      'New Holland',    250.00, 'Livre', 1),
('6145R',       'John Deere',    300.00, 'Alugado', 2),
('MF 85.265',   'Massey Ferguson', 280.00, 'Alugado', 3),
('Arion 660',    'CLAAS',        260.00, 'Livre', 1),
('Maxxum 150',   'Case IH',       270.00, 'Alugado', 2),
('T7.290',       'New Holland',    310.00, 'Alugado', 3);

```

Figura 6.14: Povoamento de Trator

Tabela Aluguer

Por fim foram registados seis alugueres distintos. Cada aluguer está corretamente associado a um cliente diferente, que realiza a alocação a um trator e a um funcionário, responsável pela gestão da operação. Para assegurar coerência e integridade referencial, os tratores e funcionários escolhidos pertencem de cada aluguer estão associados ao mesmo stand, garantindo que todas as transações seguem a estrutura organizacional da AgroAuto.

```

INSERT INTO `AgroAuto`.`Aluguer`
  (dataInicio, dataTermino, precoTotal, metodoPagamento, estadoPagamento, tipoPagamento, idCliente, idTrator, idFuncionario)
VALUES
('2025-05-20', '2025-05-25', 1500.00, 'CartaoCredito', 'Concluido', 'APronto',      1, 1, 1),
('2025-05-20', '2025-05-22', 900.00, 'Dinheiro',       'Concluido', 'APronto',      2, 2, 2),
('2025-05-25', '2025-06-18', 7000.00, 'CartaoCredito', 'Concluido', 'EmPrestacoes', 3, 3, 3),
('2025-05-20', '2025-06-23', 10500.00, 'Dinheiro',       'Concluido', 'APronto',      5, 2, 2),
('2025-05-25', '2025-05-27', 930.00, 'CartaoCredito', 'EmAtraso',   'EmPrestacoes', 4, 6, 3),
('2025-05-28', '2025-06-11', 4050.00, 'Dinheiro',       'Concluido', 'APronto',      6, 5, 2);

```

Figura 6.15: Povoamento de Aluguer

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

Para estimar a dimensão inicial da base de dados da AgroAuto, foi calculado o espaço ocupado por um registo em cada tabela, considerando os tipos de dados utilizados e o espaço reservado pelo MySQL para armazená-los.

Cada cálculo baseou-se na soma dos tamanhos individuais dos atributos, respeitando as definições de armazenamento do MySQL e a estrutura lógica previamente definida

A tabela Cliente, por conter o maior número de atributos, apresenta um volume de 625 *bytes* por registo, enquanto outras tabelas possuem tamanhos mais reduzidos, conforme detalhado na análise seguinte.

Atributo	Tipo	Bytes
idCliente	INT	4
nomeCompleto	VARCHAR(100)	100
dataNascimento	DATE	3
NIF	VARCHAR(9)	9
númeroDocumento	VARCHAR(8)	8
dataValidadeDocumento	DATE	3
rua	VARCHAR(100)	100
localidade	VARCHAR(75)	75
códigoPostal	VARCHAR(10)	10
númeroTelemóvel	VARCHAR(9)	9
email	VARCHAR(100)	100
dataValidadeCarta	DATE	3
habilitação	VARCHAR(75)	75
dataValidadeCartão	DATE	3
númeroCartão	VARCHAR(19)	19
CVV	VARCHAR(3)	3
Total		625

Tabela 6.1: Tamanho inicial da tabela Cliente

A tabela Aluguer apresenta um tamanho reduzido por registo, refletindo a sua função de armazenamento essencial de transações sem informações excessivas. O cálculo final resulta em 30 bytes por registo, demonstrando uma estrutura otimizada e adequada para um elevado volume de transações sem comprometer a eficiência do sistema.

Atributo	Tipo	Bytes
idAluguer	INT	4
dataInício	DATE	3
dataTérmino	DATE	3
preçoTotal	DECIMAL(8,2)	5
métodoPagamento	ENUM('CartaoCredito','Dinheiro')	1
estadoPagamento	ENUM('EmAtraso','Concluido')	1
tipoPagamento	ENUM('APronto','EmPrestacoes')	1
idCliente	INT	4
idTrator	INT	4
idFuncionário	INT	4
Total		30

Tabela 6.2: Tamanho inicial da tabela Aluguer

O tamanho total de 164 bytes por registo da tabela Trator reflete a necessidade de armazenar

atributos detalhados, particularmente os campos modelo e marca, que ocupam 75 bytes cada, podendo ser considerados relativamente extensos.

Atributo	Tipo	Bytes
idTrator	INT	4
modelo	VARCHAR(75)	75
marca	VARCHAR(75)	75
preçoDiário	DECIMAL(8,2)	5
estado	ENUM('Livre', 'Alugado')	1
idStand	INT	4
Total		164

Tabela 6.3: Tamanho inicial da tabela Trator

A tabela Funcionário apresenta um tamanho compacto, totalizando 92 bytes por registo, refletindo a necessidade de armazenar apenas informações essenciais sobre cada colaborador.

Atributo	Tipo	Bytes
idFuncionário	INT	4
nomeCompleto	VARCHAR(75)	75
númeroTelemóvel	VARCHAR(9)	9
idStand	INT	4
Total		92

Tabela 6.4: Tamanho inicial da tabela Funcionário

A tabela Stand apresenta um consumo de 363 bytes por registo, refletindo a necessidade de armazenar informações detalhadas sobre cada filial da empresa.

Atributo	Tipo	Bytes
idStand	INT	4
rua	VARCHAR(100)	100
localidade	VARCHAR(75)	75
codigoPostal	VARCHAR(75)	75
númeroTelefone	VARCHAR(9)	9
email	VARCHAR(100)	100
Total		363

Tabela 6.5: Tamanho inicial da tabela Stand

Cada cálculo baseou-se na soma dos tamanhos individuais dos atributos, respeitando as definições de armazenamento do MySQL e a estrutura lógica previamente definida. A tabela

seguinte apresenta o tamanho estimado para cada entidade:

Tabela	Total de bytes por registo
Cliente	625
Aluguer	30
Trator	164
Funcionário	92
Stand	363
Total	1274

Tabela 6.6: Tamanho estimado por tabela

A soma total dos espaços ocupados por um registo de cada tabela resulta em 1274 *bytes*, definindo a dimensão inicial do sistema.

Para estimar a evolução do armazenamento ao longo do tempo, foi considerado um cenário de expansão controlada, alinhado com o crescimento sustentável da AgroAuto. As estimativas foram definidas com base na inserção gradual de novos registos por ano, mantendo um equilíbrio entre capacidade operacional e necessidade de armazenamento.

Estima-se o registo de cerca de 150 novos clientes por ano, acompanhando o crescimento da procura pelos serviços da empresa. Em relação aos alugueres, prevê-se a realização de aproximadamente 500 novas operações anuais, refletindo a dinâmica comercial da plataforma.

Quanto aos recursos físicos, prevê-se a introdução de 20 novos tratores por ano, suportando a renovação da frota e a adaptação às necessidades do mercado. A equipa de funcionários deverá crescer pontualmente, com uma contratação anual, enquanto a estrutura física da empresa será expandida gradualmente, com a abertura de um novo stand a cada cinco anos, permitindo a sua consolidação no setor.

Com base nestas premissas, foi calculado o espaço ocupado por cada registo e estimado o crescimento anual esperado da base de dados, facilitando o planeamento da sua evolução e garantindo uma gestão eficiente do armazenamento.

Tabela	Novos registos por ano	Bytes por registo	Total anual(KB)
Cliente	150	625	91,6
Aluguer	500	30	14,6
Trator	20	164	3,2
Funcionário	1	92	0,1
Stand	0,2	363	0,1
Total			109,6 KB ≈ 0,11 MB

Tabela 6.7: Crescimento anual estimado da base de dados

Com um tamanho inicial estimado de 1,27 KB, a base de dados deverá crescer cerca de 0,11 MB por ano. Ao fim de cinco anos, o volume total armazenado deverá atingir aproximadamente 0,8 MB, exigindo um planeamento estratégico para garantir que o desempenho do sistema se mantém eficiente à medida que o volume de dados aumenta.

6.5 Definição e caracterização de vistas de utilização em SQL

No sistema de gestão AgroAuto, a criação de *views* desempenha um papel fundamental ao facilitar o acesso a informações relevantes, eliminando a necessidade de repetir consultas SQL complexas. Apresentam-se de seguida duas vistas implementadas especificamente para responder aos requisitos de controlo do sistema, uma relacionada com a popularidade das marcas de tratores alugados e outra focada na disponibilidade atual da frota. Desta forma, assegura-se não só a eficiência no acesso à informação, mas também o cumprimento das regras de segurança e controlo definidas para o sistema.

6.5.1 MarcaMaisAlugada - Marca de tratores mais alugada no final de cada ano.

Esta vista tem como objetivo identificar, no final de cada ano, qual a marca de trator mais alugada. O seu funcionamento baseia-se numa análise dos registo de aluguer realizados durante o ano em curso. A criação desta vista justifica-se pela necessidade de fornecer dados estatísticos essenciais para a tomada de decisões estratégicas da empresa, o que permite identificar tendências de mercado e orientar futuras aquisições de equipamento com base no comportamento real dos clientes.

A estrutura da consulta inicia-se pela seleção da marca do trator e do número total de alugueres dessa marca, através da correspondência entre os campos idTrator das tabelas Aluguer e Trator. De seguida, realiza uma junção das tabelas Aluguer e Trator com base no idTrator, para conseguir saber a marca associada a cada aluguer. Aplica-se um filtro temporal WHERE YEAR(dataInicio)= YEAR(CURDATE()) para considerar apenas os alugueres iniciados no ano atual, garantindo que a análise seja sempre relevante para o período corrente.

A agregação é realizada por marca, utilizando COUNT(*) para contabilizar o número total de alugueres associados a cada uma. A utilização da cláusula HAVING em conjunto com uma *subquery* permite isolar exclusivamente a marca com o valor mais elevado de alugueres. Esta *subquery* interna calcula o número máximo de alugueres por marca, assegurando que, em caso de empate, todas as marcas com esse valor máximo sejam incluídas no resultado final.

O resultado da vista é uma lista que contém a marca mais requisitada durante o ano, acompanhada do respetivo número total de alugueres.

```

CREATE VIEW MarcaMaisAlugada AS
SELECT
    Trator.marca,
    COUNT(*) AS totalAlugueres
FROM AgroAuto.Aluguer
JOIN AgroAuto.Trator ON Aluguer.idTrator = Trator.idTrator
WHERE YEAR(dataInicio) = YEAR(CURDATE()) -- Apenas ano atual
GROUP BY Trator.marca
HAVING totalAlugueres = (
    SELECT MAX(totalAlugueres) FROM (
        SELECT COUNT(*) AS totalAlugueres
        FROM AgroAuto.Aluguer
        JOIN AgroAuto.Trator ON Aluguer.idTrator = Trator.idTrator
        WHERE YEAR(dataInicio) = YEAR(CURDATE())
        GROUP BY Trator.marca
    ) AS subquery
);

```

Figura 6.16: View MarcaMaisAlugada

6.5.2 TratoresDisponiveis - lista de tratores disponíveis para alugar

A vista TratoresDisponiveis fornece uma lista dos tratores que se encontram disponíveis para aluguer. Esta vista foi desenvolvida para otimizar as operações quotidianas da empresa, eliminando a necessidade de consultas repetitivas à base de dados e proporcionando aos funcionários uma ferramenta rápida e eficaz para verificar instantaneamente a disponibilidade da frota durante o atendimento aos clientes.

A consulta percorre exclusivamente a tabela Trator e seleciona os campos idTrator, modelo e marca. Aplica-se uma filtragem através da condição WHERE estado = 'Livre', que permite obter apenas os registo de tratores cujo estado atual está disponível para aluguer. Como não existe necessidade de relacionar dados com outras tabelas, a vista apresenta um desempenho otimizado e fornece exatamente a informação pretendida de forma direta e eficiente.

```

CREATE VIEW TratoresDisponiveis AS
SELECT
    idTrator,
    modelo,
    marca
FROM AgroAuto.Trator
WHERE estado = 'Livre';

```

Figura 6.17: View TratoresDisponiveis

6.6 Tradução das interrogações do utilizador para SQL

A tradução das interrogações previamente formuladas em álgebra relacional para SQL permite testar a estrutura lógica da base de dados e validar a correta implementação das relações entre entidades. Através desta conversão, garantimos que as consultas definidas a nível teórico podem ser aplicadas diretamente no sistema, assegurando a recuperação eficiente dos dados e a resposta às necessidades dos utilizadores. De seguida, serão apresentadas as *queries* correspondentes às interrogações desenvolvidas na fase de modelação.

Query 1 – Lista de tratores alugados por stand

A *Query 1*, correspondente ao requisito RM02, tem como objetivo fornecer, diariamente, uma listagem dos tratores atualmente alugados, organizados por stand e ordenados alfabeticamente pela marca.

Para isso, foi criada a *view* TratoresAlugadosPorStandDiario, garantindo uma consulta eficiente e estruturada dos dados. A utilização de uma *view* neste contexto é particularmente valiosa, pois permite que a listagem dos tratores alugados seja atualizada automaticamente a cada consulta, sem necessidade de intervenções manuais. O uso do campo CURRENT_DATE assegura que a informação apresentada corresponde sempre à realidade do dia da execução, permitindo um acompanhamento em tempo real.

Desta forma, em vez de executar repetidamente a mesma lógica de consulta sobre os dados da tabela Trator, a implementação da *view* centraliza essa lógica, tornando a recuperação dos registo mais rápida e eficiente.

O uso de DISTINCT assegura que cada trator seja apresentado apenas uma vez, eliminando possíveis duplicações na visualização.

A junção entre as tabelas Trator e Stand estabelece a relação entre cada trator e o stand onde este está alocado, facilitando a identificação das unidades alugadas em cada localização.

O filtro WHERE t.estado = 'Alugado' assegura que a consulta retorna exclusivamente os tratores atualmente em utilização, eliminando dados irrelevantes e garantindo que apenas os registo pertinentes sejam apresentados. A tabela resultante exibe os dados agrupados por stand e ordenados alfabeticamente pela marca.

```

-- DROP VIEW IF EXISTS TratoresAlugadosPorStandDiario;
CREATE VIEW TratoresAlugadosPorStandDiario AS
SELECT DISTINCT
    CURRENT_DATE AS dataConsulta,
    s.idStand,
    t.idTrator,
    t.modelo,
    t.marca
FROM
    AgroAuto.Trator t
    JOIN AgroAuto.Stand s ON t.idStand = s.idStand
WHERE
    t.estado = 'Alugado' ;

SELECT *
FROM TratoresAlugadosPorStandDiario
ORDER BY idStand, marca;

```

Figura 6.18: Query 1 SQL

Query 2 – Verificação de aptidão de Cliente

A *Query 2*, baseada no requisito RM04, foi desenvolvida para verificar se um cliente está apto a realizar um aluguer até uma data específica. Para esta validação, foi implementada a *stored procedure* VerificaClienteAptoAluguer, permitindo uma abordagem estruturada, reutilizável e eficiente, ao invés de depender de consultas individuais.

Neste contexto, a utilização de um *procedure* permitiu a centralização da lógica de verificação, eliminando a necessidade de múltiplas consultas dispersas e melhorou a eficiência, permitindo que a verificação seja executada rapidamente sempre que necessário.

O *procedure* recebe como parâmetros o identificador do cliente e a data final pretendida, permitindo uma avaliação precisa da sua aptidão para o aluguer.

A lógica interna segue três etapas fundamentais. Inicialmente é feita a identificação do cliente, assegurando que a análise está vinculada a um utilizador válido. De seguida, realiza-se a verificação das condições necessárias para aptidão, avaliando se a data de validade da carta de condução é igual ou superior à data final do aluguer e se a habilitação corresponde ao tipo 'T', garantindo que o cliente está legalmente autorizado a utilizar o trator. Por fim, o resultado é explicitado na coluna estadoAptidao indicando se o cliente está "Apto" ou "Não Apto", conforme a validação das condições anteriores.

```

-- DROP PROCEDURE IF EXISTS VerificaClienteAptoAluguer;
DELIMITER $$

CREATE PROCEDURE VerificaClienteAptoAluguer (
    IN pidClientePretendido INT,
    IN pdataTerminoPretendida DATE
)
BEGIN
    SELECT
        idCliente,
        nomeCompleto,
        IF(dataValidadeCarta >= pdataTerminoPretendida AND habilitacao = 'T',
            'Apto', 'Não Apto') AS estadoAptidao
    FROM AgroAuto.Cliente
    WHERE idCliente = pidClientePretendido;
END $$

DELIMITER ;

```

Figura 6.19: Query 2 SQL

Query 3 – Funcionário com Mais Alugueres num Mês

A *Query 3*, baseada no requisito RM10, foi desenvolvida para identificar o funcionário que realizou o maior número de alugueres num determinado mês. Para garantir a eficiência e reutilização da lógica, foi implementada o *procedure* FuncionariosComMaisAlugueresMes, que recebe como parâmetros a data de início e a data de fim do intervalo mensal a analisar.

A utilização de um *procedure* nesta consulta oferece vantagens significativas, como a centralização do processamento, permitindo que a verificação seja feita de forma estruturada e padronizada sempre que necessário. Além disso, melhora o desempenho da execução ao evitar duplicação de lógica em consultas dispersas e facilita futuras modificações sem necessidade de reescrever múltiplas *queries*.

O *procedure* realiza uma junção entre as tabelas Funcionario e Aluguer, filtrando os registo com base nas datas fornecidas e agrupando os alugueres por funcionário para contar quantos foram realizados por cada um.

Na cláusula HAVING, é implementada uma *subquery* essencial para identificar corretamente o funcionário com mais alugueres no período especificado. Esta *subquery* agrupa os alugueres por idFuncionario, conta os alugueres de cada colaborador e determina o maior desses totais com a função MAX. Esse valor máximo é então comparado na *query* principal, garantindo que apenas os funcionários cujo número de alugueres é igual ao maior valor registrado sejam retornados.

Esta abordagem permite que o resultado final inclua todos os funcionários que compartilhem o primeiro lugar, tratando corretamente situações de empate. Sem a *subquery*, não seria possível isolar diretamente os melhores desempenhos, pois a *query* principal não teria acesso ao valor máximo previamente calculado sobre o mesmo conjunto de dados. Assim, a *subquery* atua como referência para garantir que apenas os funcionários com o melhor desempenho mensal sejam selecionados.

```

-- DROP PROCEDURE IF EXISTS FuncionariosComMaisAlugueresMes;
DELIMITER $$

CREATE PROCEDURE FuncionariosComMaisAlugueresMes (
    IN pDataInicioMes DATE,
    IN pDataFimMes DATE
)
BEGIN
    SELECT
        F.idFuncionario,
        F.nomeCompleto,
        COUNT(A.idAluguer) AS totalAlugueres
    FROM AgroAuto.Funcionario F
    JOIN AgroAuto.Aluguer A ON F.idFuncionario = A.idFuncionario
    WHERE A.dataInicio BETWEEN pDataInicioMes AND pDataFimMes
    GROUP BY F.idFuncionario, F.nomeCompleto
    HAVING totalAlugueres = (
        SELECT
            MAX(totalAlugueres)
        FROM (
            SELECT
                idFuncionario,
                COUNT(idAluguer) AS totalAlugueres
            FROM AgroAuto.Aluguer
            WHERE dataInicio BETWEEN pDataInicioMes AND pDataFimMes
            GROUP BY idFuncionario
        ) AS subquery
    );
END $$
DELIMITER ;

```

Figura 6.20: Query 3 SQL

Query 4 – Total de Alugueres por Trimestre

A *Query 4*, correspondente ao requisito RM11, foi desenvolvida para calcular o número total de alugueres realizados pela empresa dentro de um trimestre específico. Para isso, foi implementada o *procedure* TotalAlugueresPorTrimestre, que recebe dois parâmetros: a data de início e a data de fim do período a analisar.

O *procedure* executa uma `SELECT COUNT(*)`, responsável por calcular o total de registos na tabela `Aluguer`, filtrando os dados pelo intervalo estabelecido como o trimestre. Isto garante que apenas os alugueres realizados dentro do intervalo especificado sejam contabilizados, evitando erros de contagem fora do período de interesse.

Esta abordagem permite que o total de alugueres seja obtido de maneira rápida e precisa, bastando fornecer as datas limite como argumentos.

```

-- DROP PROCEDURE IF EXISTS TotalAlugueresPorTrimestre;
DELIMITER $$

CREATE PROCEDURE TotalAlugueresPorTrimestre(
    IN pdataInicioTrimestre DATE,
    IN pdataFimTrimestre DATE
)
BEGIN
    SELECT COUNT(*) AS totalAlugueres
    FROM AgroAuto.Aluguer
    WHERE dataInicio BETWEEN pdataInicioTrimestre AND pdataFimTrimestre;
END $$

DELIMITER ;

```

Figura 6.21: Query 4 SQL

Query 5 – Total Faturado por Stand por Mês

A *Query 5*, correspondente ao requisito RM12, tem como objetivo calcular o montante total faturado por cada stand num determinado mês. Para organizar previamente os dados necessários para esta análise, foi criada a *view* TotalFaturadoPorStandMensal, garantindo uma estrutura acessível e otimizada para consultas periódicas.

A escolha de implementação de uma *view*, em vez de um *procedure*, neste contexto, justifica-se pela necessidade de acesso imediato aos dados agregados sem exigir processamento adicional em cada execução. Como a faturação por stand segue uma lógica fixa, a pré-estruturação dos dados permite que qualquer consulta a esta *view* seja realizada de forma direta e eficiente.

Desta forma, a *query* obtém informações da tabela Aluguer, juntando-as à tabela Trator para associar cada aluguer ao respetivo stand. A extração do ano e mês da data de início permite consolidar os dados mensalmente, enquanto a soma dos valores do campo precoTotal determina o total faturado. A agregação pelos campos stand, ano e mês assegura a correta organização das informações.

Com a *view* disponível, qualquer análise mensal pode ser efetuada com um simples SELECT, filtrando os resultados pelo ano e mês pretendidos. Esta abordagem reduz a repetição da lógica de agregação, tornando o acesso aos dados mais intuitivo e eficiente, sem necessidade de cálculos redundantes a cada consulta.

```

CREATE VIEW TotalFaturadoPorStandMensal AS
SELECT
    t.idStand,
    YEAR(a.dataInicio) AS ano,
    MONTH(a.dataInicio) AS mes,
    SUM(a.precoTotal) AS totalFaturado
FROM
    AgroAuto.Aluguer a
    JOIN AgroAuto.Trator t ON a.idTrator = t.idTrator
GROUP BY
    t.idStand, ano, mes;
SELECT *
FROM TotalFaturadoPorStandMensal
WHERE ano = 2025 AND mes = 5;

```

Figura 6.22: Query 5 SQL

6.7 Indexação do Sistema de Dados

A criação de índices na base de dados AgroAuto teve como principal objetivo melhorar a performance das consultas mais frequentes, otimizando o tempo de resposta em operações de leitura, filtragem e junção entre tabelas.

Os índices foram aplicados de forma estratégica sobre campos que são habitualmente utilizados em filtros, ordenações ou ligações entre tabelas. Para operações que dependem de filtros sobre atributos, foram criados índices como `idx_trator_estado`, permitindo a recuperação rápida de tratores disponíveis para aluguer. A indexação em atributos como `idx_trator_precoDiario` e `idx_trator_marca` melhora o desempenho de consultas que classificam tratores por preço e marca, respectivamente, tornando a visualização dos dados mais ágil.

Além disso, todos os campos que representam chaves estrangeiras foram também indexados, o que permite melhorar o desempenho de operações que envolvem junções entre tabelas, como a obtenção de aluguéis associados a um determinado cliente ou funcionário.

Embora os índices sejam essenciais para a otimização da leitura, o seu uso excessivo pode impactar negativamente operações de escrita, como inserções e atualizações, uma vez que cada modificação nos dados exige também a atualização dos índices associados. Por isso, foi adotada uma abordagem equilibrada, garantindo benefícios na recuperação de dados sem comprometer a eficiência da manutenção do sistema.

```

-- Índices para junções na tabela Aluguer
CREATE INDEX idx_aluguer_idCliente ON Aluguer(idCliente);
CREATE INDEX idx_aluguer_idFuncionario ON Aluguer(idFuncionario);
CREATE INDEX idx_aluguer_idTrator ON Aluguer(idTrator);

-- Índice para consultar tratores por preço
CREATE INDEX idx_trator_precioDiario ON Trator(precioDiario);
-- Índice para consultar tratores por marca
CREATE INDEX idx_trator_marca ON Trator(marca);
-- Índice para verificar tratores livres para aluguer
CREATE INDEX idx_trator_estado ON Trator(estado);
-- Índice para junção na tabela Trator
CREATE INDEX idx_trator_idStand ON Trator(idStand);

-- Índice para junção na tabela Funcionário
CREATE INDEX idx_funcionario_idStand ON Funcionario(idStand);

```

Figura 6.23: Definição de Índices no Sistema AgroAuto

6.8 Implementação de procedimentos, funções e gatilhos

Procedimentos

A implementação de *procedures* na base de dados AgroAuto teve como objetivo automatizar operações frequentes e garantir controlo sobre o acesso aos dados, seguindo os requisitos de gestão e permissões estabelecidos.

ClientesMaisAtivos - Clientes que mais tratores alugaram depois de certa data

Este procedimento permite obter os clientes que mais tratores alugaram após uma data específica. Este procedimento visa identificar padrões de utilização do serviço, ajudando na tomada de decisões estratégicas e no acompanhamento de clientes frequentes.

Desta forma, os alugueres são filtrados com base na data indicada através do parâmetro pDataInicio. Em seguida, é contabilizado o número de alugueres realizados por cada cliente e, por fim, os resultados são ordenados por volume de atividade, retornando os 10 clientes com maior número de alugueres nesse período.

```

DELIMITER $$

CREATE PROCEDURE ClientesMaisAtivos (
    IN pDataInicio DATE)
BEGIN
    SELECT
        Cliente.idCliente,
        nomeCompleto,
        COUNT(*) AS totalAlugueres
    FROM AgroAuto.Aluguer
    JOIN AgroAuto.Cliente ON Aluguer.idCliente = Cliente.idCliente
    WHERE dataInicio >= pDataInicio
    GROUP BY Cliente.idCliente, nomeCompleto
    ORDER BY totalAlugueres DESC
    LIMIT 10;
END $$

DELIMITER ;

```

Figura 6.24: Procedure ClientesMaisAtivos

RegistrosAlugueresFuncionario – Alugueres realizados entre determinadas datas por parte de qualquer funcionário

Este procedimento calcula o número de alugueres realizados por cada funcionário dentro de um intervalo de datas definido pelos parâmetros pDataInício e pDataFim.

Ao filtrar os registos nesse período e agrupar os dados por funcionário, permite obter a contagem total de alugueres efetuados por cada um.

Esta funcionalidade é útil para monitorizar a atividade individual de cada funcionário, possibilitando análises de desempenho e apoio à gestão da eficiência operacional.

```

DELIMITER $$

CREATE PROCEDURE RegistrosAlugueresFuncionario (
    IN pDataInicio DATE,
    IN pDataFim DATE
)
BEGIN
    SELECT idFuncionario, COUNT(*) AS totalAlugueres
    FROM AgroAuto.Aluguer
    WHERE dataInicio BETWEEN pDataInicio AND pDataFim
    GROUP BY idFuncionario;
END $$

DELIMITER ;

```

Figura 6.25: Procedure RegistrosAlugueresFuncionario

LucroTotalAlugueres – Cálculo do lucro total dos alugueres num período

Este procedimento calcula o lucro total gerado pelos alugueres dentro de um intervalo de tempo estabelecido. A sua implementação visa fornecer uma visão financeira clara sobre a receita gerada, auxiliando na gestão e tomada de decisões estratégicas.

O procedimento filtra os alugueres entre as datas fornecidas pelos parâmetros pDataInicio e pDataFim e soma os valores de precoTotal para determinar o lucro total.

```
DELIMITER $$

CREATE PROCEDURE LucroTotalAlugueres (
    IN pDataInicio DATE,
    IN pDataFim DATE
)
BEGIN
    SELECT
        SUM(precoTotal) AS lucroTotal
    FROM AgroAuto.Aluguer
    WHERE dataInicio BETWEEN pDataInicio AND pDataFim;
END $$

DELIMITER ;
```

Figura 6.26: Procedure LucroTotalAlugueres

HistoricoAlugueresCliente – Consulta do histórico de alugueres de um cliente

Este procedimento retorna o histórico completo de alugueres de um cliente específico. Desta forma, possibilita um acompanhamento detalhado da relação do cliente com a empresa, útil para análises internas e atendimento personalizado.

Desta forma, o procedimento filtra os alugueres pertencentes ao cliente fornecido pelo parâmetro pidCliente e exibe detalhes como datas do aluguer, identificador do aluguer e nome do cliente. Os resultados são ordenados de forma decrescente, permitindo visualizar os alugueres mais recentes primeiro.

```

DELIMITER $$

CREATE PROCEDURE HistoricoAlugueresCliente(IN pidCliente INT)
BEGIN
    SELECT
        Aluguer.idCliente,
        Cliente.nomeCompleto,
        Aluguer.idAluguer,
        Aluguer.dataInicio,
        Aluguer.dataTermino
    FROM AgroAuto.Aluguer
    JOIN AgroAuto.Cliente ON Aluguer.idCliente = Cliente.idCliente
    WHERE Aluguer.idCliente = pidCliente
    ORDER BY Aluguer.dataInicio DESC;
END $$

DELIMITER ;

```

Figura 6.27: Procedure HistoricoAlugueresCliente

regarNovoAluguer - Registo de Um Novo Aluguer com Validações

O procedimento `regarNovoAluguer` foi desenvolvido para encapsular a lógica de inserção de um novo aluguer, garantindo que todas as operações relacionadas sejam executadas de forma segura e consistente. Para assegurar a integridade dos dados, este processo é estruturado como uma transação, permitindo que qualquer falha durante a execução resulte em um *rollback*, evitando regtos incompletos ou inconsistentes na base de dados.

Antes de registar o aluguer, o procedimento valida se o trator está livre para utilização. Caso contrário, a operação é cancelada imediatamente.

O aluguer é inserido com um preço inicial de 0 e o estado do pagamento é deduzido com base no tipo de pagamento. Assim, se o tipo de pagamento for "A Pronto", o estado de pagamento é definido como "Concluído", se o tipo de pagamento for "Em Prestações", o estado de pagamento é definido como "Em Atraso".

Após a inserção, o preço total do aluguer é calculado utilizando a função `calcularCustoAluguer`, que determina o custo com base no preço diário do trator e no número de dias de aluguer. Esta abordagem centraliza o cálculo, evitando duplicações e garantindo a robustez da transação. Este campo é atualizado no regsto do aluguer.

A lógica de modificação do estado do trator é delegada a um *trigger* previamente definido, garantindo que alterações no aluguer impactam corretamente o estado do veículo sem necessidade de manipulação direta dentro da *procedure*.

Caso alguma etapa falhe, a transação é revertida, garantindo a consistência dos dados.

```

DELIMITER $$

CREATE PROCEDURE registrarNovoAluguer (
    IN pdataInicio DATE,
    IN pdataTermino DATE,
    IN pmetodoPagamento ENUM('CartaoCredito', 'Dinheiro'),
    IN ptipoPagamento ENUM('APronto', 'EmPrestacoes'),
    IN pidCliente INT,
    IN pidTrator INT,
    IN pidFuncionario INT
)
BEGIN
    -- Manipulação de erros
    DECLARE EXIT HANDLER FOR SQLEXCEPTION
    BEGIN
        ROLLBACK;
    END;

    START TRANSACTION;

    -- Verifica disponibilidade do trator
    IF NOT EXISTS (SELECT 1 FROM Trator WHERE idTrator = pidTrator AND estado = 'Livre') THEN
        ROLLBACK;
        END IF;

    -- Inserir aluguer sem calcular preço
    INSERT INTO Aluguer (
        dataInicio, dataTermino, precoTotal, metodoPagamento,
        estadoPagamento, tipoPagamento, idCliente, idTrator, idFuncionario
    )
    VALUES (
        pdataInicio, pdataTermino, 0, pmetodoPagamento, IF(ptipoPagamento = 'APronto', 'Concluido', 'EmAtraso'),
        ptipoPagamento, pidCliente, pidTrator, pidFuncionario
    );

    -- Atualiza preço usando a função
    SET @novoId = LAST_INSERT_ID();
    UPDATE Aluguer
    SET precoTotal = calcularCustoAluguer(@novoId)
    WHERE idAluguer = @novoId;

    COMMIT;
END $$

DELIMITER ;

```

Figura 6.28: Procedure registrarNovoAluguer

Triggers

O sistema AgroAuto implementa dois *triggers* essenciais para automatizar a gestão do estado dos tratores durante o processo de aluguer, garantindo a consistência dos dados e eliminando a necessidade de intervenção manual para atualizar a disponibilidade dos equipamentos.

setTratorAlugado - Atualização do Estado do Trator no Início do Aluguer

Este *trigger* é acionado automaticamente sempre que um novo registo de aluguer é inserido na base de dados, assegurando que o trator correspondente tem o campo estado como "Alugado".

A sua lógica verifica se o trator associado ao novo aluguer ainda está disponível, e, caso se confirme, atualiza o seu estado imediatamente, garantindo que a informação refletida na base de dados esteja sempre correta.

Este mecanismo reduz o risco de erros manuais, assegura a consistência dos dados relativamente à disponibilidade dos equipamentos e evita problemas de sobreposição de reservas.

```
DELIMITER $$

CREATE TRIGGER setTratorAlugado
AFTER INSERT ON Aluguer
FOR EACH ROW
BEGIN
    IF (SELECT estado FROM Trator WHERE idTrator = NEW.idTrator) <> 'Alugado' THEN
        UPDATE Trator
        SET estado = 'Alugado'
        WHERE idTrator = NEW.idTrator;
    END IF;
END$$

DELIMITER ;
```

Figura 6.29: Trigger setTratorAlugado

setTratorLivre - Atualização do Estado do Trator Após Aluguer

O *trigger* setTratorLivre complementa o funcionamento do setTratorAlugado, sendo responsável por libertar automaticamente os tratores assim que os alugueres terminam.

Este mecanismo é executado após qualquer atualização na tabela Aluguer, verificando, para cada registo modificado, se duas condições essenciais foram cumpridas: o pagamento do aluguer foi concluído, *estadoPagamento* = 'Concluido', e a data de término do aluguer já ocorreu *dataTermino* < CURRENT_DATE.

Apenas quando ambas as condições são verdadeiras, o sistema atualiza automaticamente o estado do trator para "Livre", tornando-o novamente disponível para aluguer.

Esta dupla verificação é essencial para garantir que os equipamentos não sejam libertados prematuramente, evitando possíveis conflitos na gestão de frota e assegurando que cada aluguer seja encerrado após o cumprimento das obrigações contratuais.

```

DELIMITER $$

CREATE TRIGGER setTratorLivre
AFTER UPDATE ON Aluguer
FOR EACH ROW
BEGIN
    -- Apenas atualiza para 'Livre' se já passou o prazo E o pagamento está concluído
    IF NEW.estadoPagamento = 'Concluido' AND NEW.dataTermino < CURRENT_DATE THEN
        UPDATE Trator
        SET estado = 'Livre'
        WHERE idTrator = NEW.idTrator;
    END IF;
END$$

DELIMITER ;

```

Figura 6.30: *Trigger setTratorLivre*

Função

No sistema AgroAuto, as funções armazenadas foram criadas para automatizar cálculos e garantir precisão nas operações do sistema. Ao encapsular a lógica dentro de uma função, evita-se repetição de código e melhora-se a eficiência das consultas.

Função calcularCustoAluguer – Automação do Cálculo de Preço

A função calcularCustoAluguer foi desenvolvida para encapsular a lógica de cálculo do custo total de um aluguer, garantindo precisão e eliminando redundâncias no processamento de dados.

Esta função evita duplicações na lógica, garantindo que todos os cálculos sejam realizados de maneira uniforme em diferentes partes do sistema. Este também permite que o custo seja calculado de forma dinâmica sempre que necessário, sem necessidade de cálculos manuais. Por outro lado. Ao calcular o preço total com base na duração do aluguer e no preço diário do trator, assegura-se que não há inconsistências nos valores armazenados.

Esta função recebe como parâmetro o identificador do aluguer, pldAluguer, determinando para qual registo o cálculo será aplicado. De seguida, calcula a duração do aluguer, utilizando DATEDIFF(dataTermino, dataInicio) + 1 para contabilizar todos os dias do período. Multiplica a duração obtida pelo preço diário do trator, obtido através de uma junção entre as tabelas Aluguer e Trator, garantindo que o valor correto seja utilizado no cálculo. Por fim, retorna o preço total do aluguer, permitindo a utilização deste em diversas operações, como inserções ou atualizações.

```
DELIMITER $$

CREATE FUNCTION calcularCustoAluguer(pIdAluguer INT) RETURNS DECIMAL(8,2)
DETERMINISTIC
BEGIN
    DECLARE precoTotal DECIMAL(8,2);

    SELECT (DATEDIFF(dataTermino, dataInicio) + 1) * precoDiario
    INTO precoTotal
    FROM Aluguer
    JOIN Trator ON Aluguer.idTrator = Trator.idTrator
    WHERE Aluguer.idAluguer = pIdAluguer;

    RETURN precoTotal;
END$$

DELIMITER ;
```

Figura 6.31: Função calcularCustoAluguer

7 Implementação do sistema de migração dados

Com o objetivo de integrar três fontes de dados heterogéneas, uma baseada num modelo relacional, outra em formato CSV e uma terceira em JSON, desenvolveu-se um sistema de migração de dados para a base de dados central da empresa AgroAuto. Como referido anteriormente, armazenava as suas informações em formatos distintos, tornando necessária a unificação desses dados em uma estrutura relacional única.

Esta migração teve como finalidade consolidar a informação operacional, nomeadamente sobre clientes, tratores, alugueres e funcionários, numa única estrutura relacional, garantindo maior eficiência na gestão e criando uma base sólida para futuras análises e funcionalidades avançadas. Com esse sistema, a empresa passa a dispor de um repositório centralizado, facilitando consultas, relatórios e a otimização dos processos internos.

Programa de Migração de *Python*

Para agilizar e automatizar o processo de migração de dados, foi desenvolvido um programa em *Python* com capacidade para lidar com três formatos distintos de origem de dados: uma base de dados relacional em PostgreSQL, ficheiros CSV e ficheiros JSON.

O programa inicia importando os módulos necessários, garantindo a correta interação com as diferentes fontes de dados: `psycopg2` para conectar à base de dados *PostgreSQL* utilizada pelo stand de Olivença, `mysql.connector` para comunicar com a base de dados central *MySQL* onde os dados serão integrados, `csv` para leitura e tratamento de ficheiros CSV e `json` para interpretação de ficheiros JSON.

Após a preparação inicial, o programa processa as três fontes de dados, lendo e interpretando o conteúdo de cada uma. Com base numa lógica predefinida, seleciona a fonte mais adequada para a migração, garantindo a consistência dos dados. Os dados da fonte selecionada são inseridos na base de dados central em MySQL, utilizando as ligações e comandos apropriados.

Além disso, como cada stand trabalha com um sistema próprio, foi necessário associar um identificador único (`idStand`) a cada fonte de dados, garantindo que os registos migrados preservam corretamente a sua origem. Esta estratégia permite que cada conjunto de dados seja integrado corretamente na base de dados central, respeitando a estrutura organizacional

da AgroAuto.

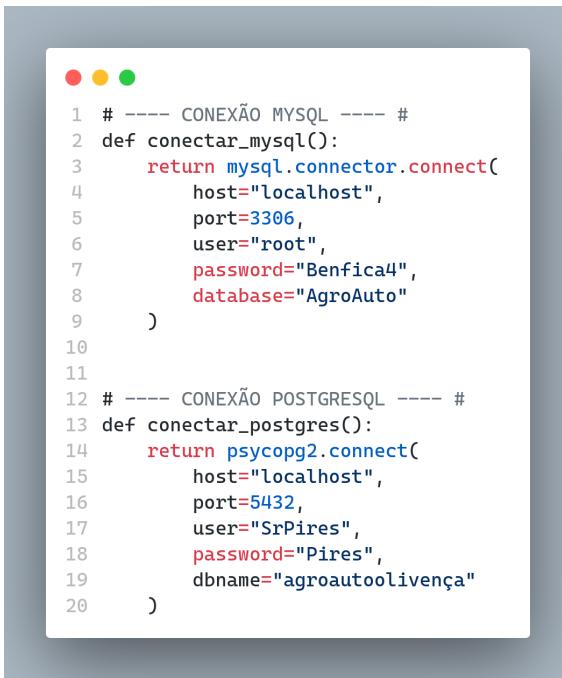


```
1 # ---- ASSOCIAÇÃO STAND FONTE DE DADOS ---- #
2 stands_migracao = {
3     1: "bd",
4     2: "csv",
5     3: "json"
6 }
7
8
9 # ---- CHAMADA DAS MIGRAÇÕES ---- #
10 if __name__ == "__main__":
11     for id_stand, tipo in stands_migracao.items():
12         print(f"\n>>> A migrar dados para stand {id_stand} usando {tipo.upper()}")
13
14         if tipo == "csv":
15             migrar_clientes_csv(f"csv/clientes.csv", id_stand)
16             migrar_tratores_csv(f"csv/tratores.csv", id_stand)
17             migrar_funcionarios_csv(f"csv/funcionarios.csv", id_stand)
18             migrar_alugueres_csv(f"csv/alugueres.csv", id_stand)
19
20         elif tipo == "json":
21             migrar_clientes_json(f"json/clientes.json", id_stand)
22             migrar_tratores_json(f"json/tratores.json", id_stand)
23             migrar_funcionarios_json(f"json/funcionarios.json", id_stand)
24             migrar_alugueres_json(f"json/alugueres.json", id_stand)
25
26         elif tipo == "bd":
27             migrar_de_postgres_clientes(id_stand)
28             migrar_de_postgres_tratores(id_stand)
29             migrar_de_postgres_funcionarios(id_stand)
30             migrar_de_postgres_alugueres(id_stand)
31
32     print("\n>>> Todas as migrações foram concluídas com sucesso.")
```

Figura 7.1: Programa de Migração de *Python*

Inicialmente, são definidas duas funções auxiliares para gerir a conexão com as bases de dados. A função `conectar_mysql` estabelece a ligação à base de dados central MySQL, utilizando as credenciais locais, como o *host*, utilizador, *password* e nome da base de dados. Da mesma forma, a função `conectar_postgres` é responsável por conectar à base de dados do stand de Olivença, permitindo posteriormente a migração dos seus dados.

Após a conexão, o programa executa a extração e interpretação dos ficheiros (*parse*), dependendo do seu formato de origem, CSV e JSON ou da base de dados em *PostgreSQL*. Para os ficheiros CSV e JSON, os dados são lidos e organizados, enquanto na base de dados *PostgreSQL*, são executadas consultas para obter os registos estruturados. Com estas informações processadas, os dados são inseridos na base de dados central, garantindo que são armazenados de forma consistente e integrada.



```

1 # ----- CONEXÃO MYSQL ----- #
2 def conectar_mysql():
3     return mysql.connector.connect(
4         host="localhost",
5         port=3306,
6         user="root",
7         password="Benfica4",
8         database="AgroAuto"
9     )
10
11
12 # ----- CONEXÃO POSTGRESQL ----- #
13 def conectar_postgres():
14     return psycopg2.connect(
15         host="localhost",
16         port=5432,
17         user="SrPires",
18         password="Pires",
19         dbname="agroautoolivença"
20     )

```

Figura 7.2: Conexões MySQL de PostgreSQL

7.1 Migração de CSV

Os ficheiros CSV utilizados pertencem ao stand de Lamego e contêm os dados relativos aos alugueres, clientes, funcionários e tratores. Para realizar a importação estruturada destes dados na base de dados MySQL, foram desenvolvidas funções como `migrar_funcionarios_csv` que associa corretamente cada funcionário ao seu respetivo stand, identificado por `id_stand`.

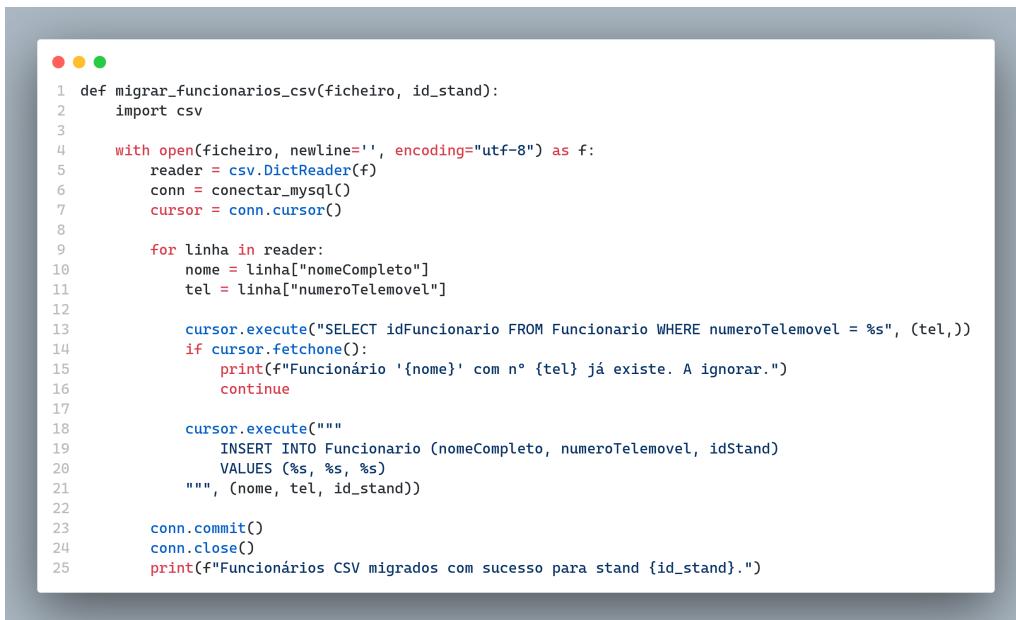
O processo inicia-se com a leitura do ficheiro CSV, que é processado linha a linha. Em seguida, é estabelecida uma ligação à base de dados MySQL através da função `conectar_mysql()`, permitindo a execução de comandos SQL via um cursor.

Para cada funcionário registado no CSV, são extraídos o nome completo e o número de telemóvel. Antes de proceder à inserção na tabela Funcionario, a função verifica se já existe um registo com o mesmo número de telemóvel, evitando duplicações. Como definido anteriormente, pelo menos um funcionário deve estar associado a cada stand, garantindo a integridade dos dados migrados. Caso já exista um funcionário com esse número, a linha é ignorada e o processo continua com o próximo registo.

Caso o funcionário não exista na base de dados, a função executa um comando INSERT para adicionar os seus dados à tabela Funcionario, incluindo o campo `idStand`, que especifica o stand ao qual pertence.

Após o processamento de todas as entradas, a função finaliza a operação com um `commit()`,

confirmando as alterações na base de dados, e encerra a conexão com `close()`, garantindo que os recursos são corretamente liberados.



```
● ● ●
1 def migrar_funcionarios_csv(ficheiro, id_stand):
2     import csv
3
4     with open(ficheiro, newline='', encoding="utf-8") as f:
5         reader = csv.DictReader(f)
6         conn = conectar_mysql()
7         cursor = conn.cursor()
8
9         for linha in reader:
10             nome = linha["nomeCompleto"]
11             tel = linha["numeroTelemovel"]
12
13             cursor.execute("SELECT idFuncionario FROM Funcionario WHERE numeroTelemovel = %s", (tel,))
14             if cursor.fetchone():
15                 print(f"Funcionário '{nome}' com nº {tel} já existe. A ignorar.")
16                 continue
17
18             cursor.execute("""
19                 INSERT INTO Funcionario (nomeCompleto, numeroTelemovel, idStand)
20                 VALUES (%s, %s, %s)
21             """, (nome, tel, id_stand))
22
23             conn.commit()
24             conn.close()
25             print(f"Funcionários CSV migrados com sucesso para stand {id_stand}.")
```

Figura 7.3: Migração de Funcionários de CSV

Além da migração dos funcionários, as restantes tabelas, como Clientes, Alugueres e Tratores, foram integradas na base de dados central utilizando funções similares. Cada função foi adaptada para processar corretamente os dados específicos de cada entidade, garantindo a extração, validação e inserção dos regístos sem duplicações ou inconsistências. O mesmo princípio de verificação prévia foi aplicado, assegurando que apenas dados corretos e relevantes fossem transferidos para a base de dados AgroAuto. A implementação destas funções é visível nos Anexos. A implementação destas funções é visível nos anexos.

7.2 Migração de PostgreSQL

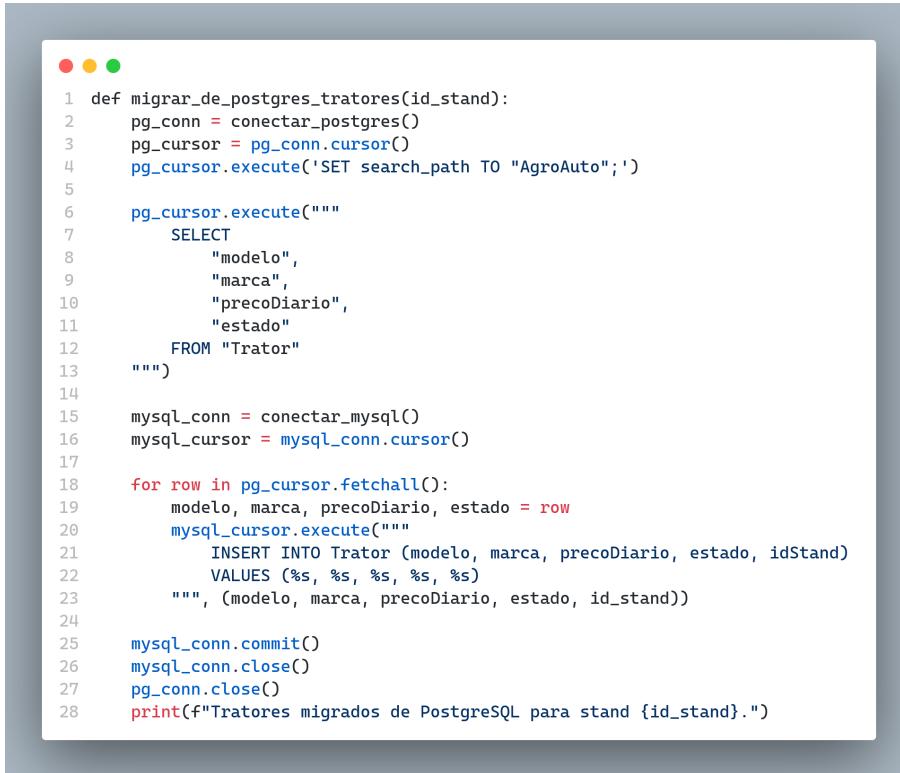
A função `migrar_de_postgres_tratores` foi desenvolvida para transferir os tratores registados na base de dados *PostgreSQL* para a base de dados MySQL, associando cada registo ao seu respetivo stand, identificado pelo parâmetro `id_stand`.

O processo inicia-se com a conexão à base de dados *PostgreSQL*, utilizando a função `conectar_postgres()`. Uma vez estabelecida a ligação, é executado o comando `SET search_path TO "AgroAuto";` para garantir que todas as operações sejam realizadas no esquema correto. Em seguida, uma consulta SQL extrai os campos essenciais da tabela Trator, incluindo modelo, marca, preço diário e estado.

Após a extração dos dados, o programa conecta-se à base de dados MySQL através da função `conectar_mysql()`, obtendo um cursor para a inserção estruturada dos registos.

Cada trator recuperado da base de dados *PostgreSQL* é processado individualmente, e uma instrução INSERT adiciona o registo à tabela Trator da base de dados MySQL, incluindo o campo adicional idStand, que preserva a associação ao stand de origem.

Por fim, um `commit()` é executado para confirmar as alterações, garantindo que todos os registos foram corretamente migrados. As conexões com as bases de dados são então encerradas, e uma mensagem de sucesso é exibida, certificando que os tratores foram integrados na base de dados central.



```
 1 def migrar_de_postgres_tratores(id_stand):
 2     pg_conn = conectar_postgres()
 3     pg_cursor = pg_conn.cursor()
 4     pg_cursor.execute('SET search_path TO "AgroAuto";')
 5
 6     pg_cursor.execute("""
 7         SELECT
 8             "modelo",
 9             "marca",
10             "precoDiario",
11             "estado"
12         FROM "Trator"
13     """)
14
15     mysql_conn = conectar_mysql()
16     mysql_cursor = mysql_conn.cursor()
17
18     for row in pg_cursor.fetchall():
19         modelo, marca, precoDiario, estado = row
20         mysql_cursor.execute("""
21             INSERT INTO Trator (modelo, marca, precoDiario, estado, idStand)
22             VALUES (%s, %s, %s, %s, %s)
23             """ , (modelo, marca, precoDiario, estado, id_stand))
24
25     mysql_conn.commit()
26     mysql_conn.close()
27     pg_conn.close()
28     print(f"Tratores migrados de PostgreSQL para stand {id_stand}.")
```

Figura 7.4: Migração de Tratores de *PostgreSQL*

Além da migração dos tratores, as restantes tabelas armazenadas na base de dados *PostgreSQL*, como Clientes, Alugueres e Funcionários, foram igualmente transferidas para a base de dados central MySQL utilizando funções similares. Cada processo foi adaptado para extrair, validar e inserir os dados corretamente, garantindo que todas as informações fossem preservadas e integradas sem duplicações ou inconsistências. A implementação das restantes funções de migração pode ser consultada nos anexos.

7.3 Migração de JSON

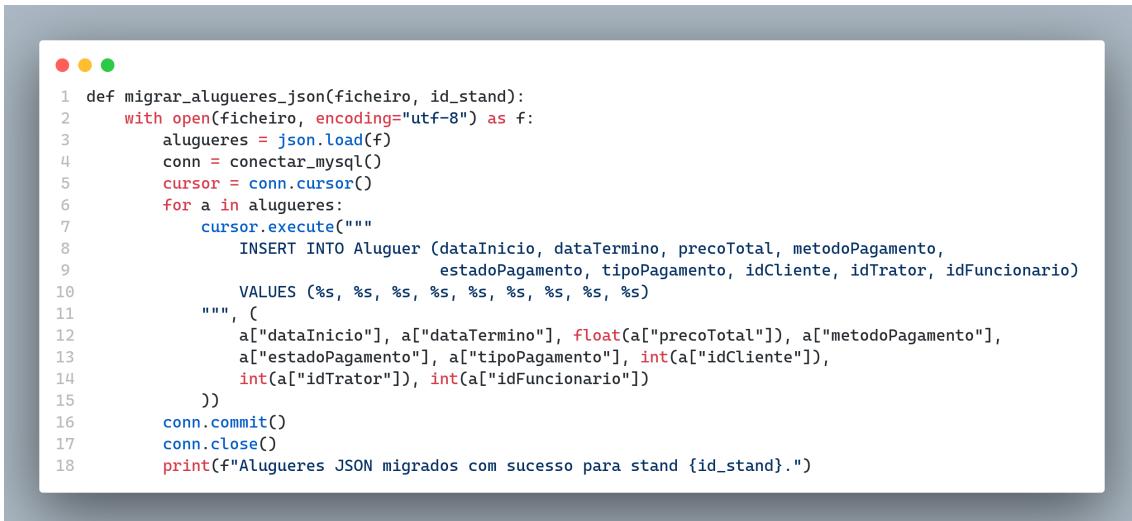
A função `migrar_alugueres_json` foi desenvolvida para importar regtos de alugueres a partir de um ficheiro JSON e inseri-los na base de dados MySQL, garantindo a correta associação dos alugueres ao respetivo stand, identificado pelo parâmetro `id_stand`.

O ficheiro JSON é aberto com codificação UTF-8, e os dados são lidos através da função `json.load()`, resultando numa lista de dicionários, onde cada dicionário representa um aluguer.

Em seguida, o sistema estabelece uma ligação à base de dados MySQL com a função `conectar_mysql()`, obtendo um cursor para inserção dos regtos. Para cada aluguer presente na lista, é executada uma instrução `INSERT INTO` na tabela `Aluguer`, garantindo que os dados sejam armazenados corretamente.

Os valores inseridos são extraídos diretamente dos dicionários do ficheiro JSON, e os campos numéricos, como `precoTotal`, `idCliente`, `idTrator` e `idFuncionario`, são convertidos explicitamente para `float` ou `int`, conforme necessário, assegurando que os tipos de dados estejam corretos.

Após a inserção de todos os regtos, a função executa um `commit()` para confirmar as alterações e encerra a ligação à base de dados, garantindo a integridade dos dados. Finalmente, exibe-se uma mensagem indicando o sucesso da operação para o stand especificado.



```
 1 def migrar_alugueres_json(ficheiro, id_stand):
 2     with open(ficheiro, encoding="utf-8") as f:
 3         alugueres = json.load(f)
 4         conn = conectar_mysql()
 5         cursor = conn.cursor()
 6         for a in alugueres:
 7             cursor.execute("""
 8                 INSERT INTO Aluguer (dataInicio, dataTermino, precoTotal, metodoPagamento,
 9                         estadoPagamento, tipoPagamento, idCliente, idTrator, idFuncionario)
10                     VALUES (%s, %s, %s, %s, %s, %s, %s, %s, %s)
11             """, (
12                 a["dataInicio"], a["dataTermino"], float(a["precoTotal"]), a["metodoPagamento"],
13                 a["estadoPagamento"], a["tipoPagamento"], int(a["idCliente"]),
14                 int(a["idTrator"]), int(a["idFuncionario"])
15             ))
16         conn.commit()
17         conn.close()
18         print(f"Alugueres JSON migrados com sucesso para stand {id_stand}.")
```

Figura 7.5: Migração de Alugueres de JSON

Além da migração dos alugueres a partir de JSON, as restantes entidades armazenadas nesse formato, como Clientes, Funcionários e Tratores, foram igualmente integradas na base de dados central MySQL utilizando funções similares.

Com essa abordagem, a gestão da AgroAuto passa a contar com um repositório centralizado, permitindo consultas eficientes, relatórios detalhados e análises operacionais de forma estruturada e otimizada. A implementação destas funções é visível nos anexos.

8 Conclusões e Trabalho Futuro

A primeira fase do projeto centrou-se na conceção de um sistema de base de dados relacional para suporte da gestão de alugueres de tratores da empresa AgroAuto. As atividades desenvolvidas abrangiam todas as etapas iniciais de desenvolvimento de um sistema de bases de dados, incluindo a definição do sistema e objetivos, levantamento e análise de requisitos, modelação conceptual e modelação lógica.

Entre os aspectos positivos mais relevantes, destaca-se a clareza na identificação das entidades, relacionamentos e atributos essenciais ao domínio de negócio, permitindo construir um modelo conceptual robusto e representativo. Além disso, a aplicação sistemática da álgebra relacional revelou-se uma mais-valia para validar a lógica do modelo e antecipar necessidades reais de informação por parte dos utilizadores.

Foram também alcançados bons resultados na organização do modelo lógico, respeitando as regras de normalização até à 3.^a forma normal, o que assegura a integridade e eficiência dos dados. A planificação e distribuição de tarefas, suportada por um diagrama GANTT, contribuiu para uma gestão eficaz do tempo e dos recursos da equipa.

No entanto, o processo não esteve isento de dificuldades. A complexidade do domínio exigiu um esforço considerável na validação dos requisitos, e nem sempre foi possível obter consenso imediato entre os diferentes intervenientes. Verificou-se também alguma complexidade na formulação de interrogações através de expressões em álgebra relacional, nomeadamente nas que envolviam agregações encadeadas.

Complementando o trabalho desenvolvido na fase de modelação, a segunda parte do projeto focou-se na implementação física da base de dados, recorrendo à criação de procedures, views e consultas SQL para satisfazer os requisitos operacionais previamente definidos. A utilização de procedures permitiu incorporar lógica condicional e operações dinâmicas, enquanto as views possibilitaram a agregação e apresentação de dados de forma clara e reutilizável. Além disso, foi realizada a migração de dados a partir de ficheiros CSV, SQL e JSON, garantindo a integração coerente de dados provenientes de diferentes fontes num sistema relacional unificado. Esta fase reforçou a ligação entre a estrutura conceptual e a prática de implementação, resultando num sistema funcional, escalável e alinhado com as necessidades da AgroAuto.

No entanto, alguns desafios surgiram, como a gestão da integridade dos dados durante a migração, exigindo validações adicionais para evitar inconsistências e corrigir problemas como valores nulos, formatos incompatíveis e dados duplicados. Além disso, algumas operações exigiram uma sequência específica de inserções para evitar problemas de integridade referencial,

especialmente na relação entre Clientes, Alugueres e Tratores. Apesar destas dificuldades, a base de dados resultante é funcional, escalável e alinhada com as necessidades da AgroAuto, proporcionando uma estrutura robusta para futuras expansões.

Com o sistema de base de dados estruturado e operacional, é possível identificar novas oportunidades de evolução que acompanhem o crescimento e a complexidade real da operação da empresa. Entre os desenvolvimentos futuros mais relevantes, destaca-se a possibilidade de incluir o registo detalhado das manutenções dos tratores, a gestão de faturas associadas aos alugueres e a integração de funcionalidades logísticas, como a coordenação de entregas e recolhas. Estas melhorias tornariam o sistema mais abrangente e completo, contribuindo para uma gestão cada vez mais eficiente, integrada e adaptada à realidade da AgroAuto.

Referências

- [1] T. Connolly and C. Begg, "Database Systems: A Practical Approach to Design, Implementation, and Management," Addison-Wesley, Global Edition, Sep. 26, 2014. ISBN: 978-1292061184.
- [2] O. Belo, "Bases de Dados Relacionais: Implementação com MySQL," FCA – Editora de Informática, 376 p., Sep. 2021. ISBN: 978-972-722-921-5.
- [3] J. Reis and M. Housley, "Fundamentals of Data Engineering: Plan and Build Robust Data Systems," 1st ed., O'Reilly Media, Jul. 2022.
- [4]

Lista de Siglas e Acrónimos

BD Base de Dados

SGBD Sistema de Gestão de Base de Dados

CSV Comma-Separated Values

JSON JavaScript Object Notation

RD Requisito de Descrição

RM Requisito de Manipulação

RC Requisito de Controlo

ER Entidade-Relacionamento

FN Forma Normal

SQL Structured Query Language

Anexos

Nesta secção apresentam-se documentos que complementam e sustentam o conteúdo desenvolvido ao longo do relatório. Estes anexos incluem tabelas e materiais de apoio relevantes à compreensão do sistema implementado.

Anexo 1

Tarefa	Duração Prevista	Duração Real	Discrepância	Justificação
Levantamento de Requisitos	3 dias	5 dias	2 dias	Necessidade de mais reuniões com o Sr.Pires para clarificação de funcionalidades essenciais.
Análise e Validação General dos Requisitos	2 dias	5 dias	3 dias	O Sr.Pires evidenciou a necessidade de alterações com impacto em entidades e relacionamentos e também sugeriu adaptações ao modelo com base em novos casos de uso identificados.
Modelação Conceptual	2 dias	4 dias	2 dias	Reestruturação da relação entre entidades e revisão de pressupostos iniciais do modelo.
Construção e Validação do Modelo de Dados Lógico	2 dias	3 dias	1 dia	O modelo teve de ser ajustado para garantir conformidade com os requisitos de normalização e integridade referencial.
Validação do Modelo com Interrogações do Utilizador	2 dias	5 dias	3 dias	Algumas queries exigiram reformulação para corresponder mais fielmente aos requisitos reais do negócio.
Povoamento da Base de Dados	3 dias	5 dias	2 dias	Dados inconsistentes ou erros na formatação dos ficheiros CSV, JSON e SQL, exigindo ajustes na transformação e validação antes da inserção.
Migração de Dados	3 dia	5 dias	2 dias	Problemas de compatibilidade entre os sistemas PostgreSQL, MySQL e JSON, exigindo reformulação das queries de extração ou conversão dos dados antes da importação.

Tabela 8.1: Tabela de Discrepâncias Temporais

Anexo 2

ATA DA REUNIÃO Nº 01 – LEVANTAMENTO DE REQUISITOS

No dia 05 de março de 2025, às 10h00, realizou-se uma reunião na Sala de Reuniões da sede de Olivença da empresa AgroAuto para o levantamento de requisitos do novo sistema de gestão de aluguer de veículos. A reunião foi conduzida pelo Sr. Pires, administrador do negócio, contou com a participação dos funcionários dos três stands, com os funcionários de Lamego e Sagres presentes por videochamada, além de um engenheiro de base de dados. Não houve ausências.

A reunião teve como principais objetivos identificar as necessidades do sistema, analisar as operações diárias do negócio, avaliar a documentação existente e levantar sugestões e preocupações das partes envolvidas. Durante a discussão, foram destacadas algumas funcionalidades essenciais, como a gestão de alugueres em tempo real, um módulo para acompanhamento da manutenção dos veículos e um histórico detalhado dos clientes. Além disso, identificou-se a necessidade de integrar o sistema com um meio de pagamento online.

Na análise documental, foram examinados contratos de aluguer e registo financeiros, sendo apontada a necessidade de padronização para facilitar a integração digital. Entre as sugestões apresentadas, os funcionários solicitaram um sistema intuitivo e acessível a dispositivos móveis, sempre mantendo o compromisso com os clientes de um processo de aluguer ágil, transparente e descomplicado.

Como próximos passos, ficou definido que serão recolhidos mais dados através de inquéritos aos clientes até o dia 09 de março, sob responsabilidade dos representantes de cada stand presentes na reunião. A próxima reunião para validação dos requisitos foi sugerida para o dia 13 de março.

A reunião foi encerrada às 11h30, com o compromisso de acompanhamento contínuo do levantamento de requisitos e a realização de novos encontros conforme necessário.

Presentes

- Sr. Artur Pires (Administrador);
- D.Fátima Teixeira (Funcionária de Olivença);
- Diogo Costa (Funcionário de Lamego);
- Fábio Vieira (Funcionário de Sagres);
- João Carmo (Engenheiro de base de dados);

Figura 8.1: Ata da Primeira Reunião

Anexo 3

FORMULÁRIO DE FICHA DE CLIENTE

1. DADOS PESSOAIS

Nome Completo: António Maria Santos Teixeira
Data de Nascimento: 09/12/1966
NIF: 254 678 639
Documento de Identificação: Cartão de Cidadão Passaporte Outro: _____
Número do Documento: 651079037 4 ZZ0
Validade do Documento: 15/08/2030
Rua: Rua Alexandre Herculano 487
Código Postal: 5100-107
Localidade: Lamego
Telemóvel: 96908675
E-mail: antoniomariaat@gmail.com

2. DADOS DA CARTA DE CONDUÇÃO

Validade: 10/06/2030
Habilitação Específica (Alguns tratores exigem categorias de carta de condução diferentes)
Caso aplicável, indicar a categoria: Categoria T

3. TERMOS E CONDIÇÕES

O cliente declara que todas as informações fornecidas são verdadeiras e comprehende que está sujeito às condições estabelecidas nos contratos de aluguer posteriores.

Assinatura do Cliente: 
Data: 15/03/2025

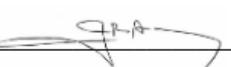
Assinatura do Funcionário Responsável: 
Data: 16/03/2025

Figura 8.2: Formulário de Registo de um Cliente

Anexo 4

FORMULÁRIO DE REGISTO DE ALUGUER

1. IDENTIFICAÇÃO DO CLIENTE

idCliente: 109

Nome Completo: António Maria Antunes Teixeira

NIF: 254 678 639

Telefone de Contacto: 96908675

2. INFORMAÇÕES PARA ALUGUER

Método de Pagamento: Cartão de Crédito Dinheiro

Número do Cartão de Crédito (se aplicável): 4342 1563 9034 5307

Validade: 12/28

CVV: 123

Tipo de Veículo Pretendido: Trator Agrícola

IdTrator: 065

Data de Início do Aluguer: 01/04/2025

Data de Término do Aluguer: 07/04/2025

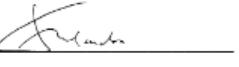
Preço total: 1.225€

Tipo de Pagamento: A pronto A prestações

Estado de Pagamento: Em atraso Concluído

3. TERMOS E CONDIÇÕES DO ALUGUER

O cliente declara que todas as informações fornecidas são verdadeiras e comprehende que o aluguer do veículo está sujeito às condições estabelecidas no contrato de aluguer anexo.

Assinatura do Cliente: 
Data: 15/03/2025

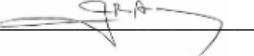
Assinatura do Funcionário Responsável: 
idFuncionario: 009
Data: 16/03/2025

Figura 8.3: Formulário de Registo de um Aluguer

Anexo 5

```
● ● ●
1 # ----- MIGRAÇÃO CSV ----- #
2
3 def migrar_clientes_csv(ficheiro, id_stand):
4     with open(ficheiro, newline='', encoding="utf-8") as f:
5         reader = csv.DictReader(f)
6         conn = conectar_mysql()
7         cursor = conn.cursor()
8         for linha in reader:
9             cursor.execute("""
10                 INSERT INTO Cliente (nomeCompleto, dataNascimento, NIF, numeroDocumento, dataValidadeDocumento, rua, localidade, codigoPostal,
11                                     numeroTelemovel, email, dataValidadeCarta, habilitacao, dataValidadeCartao, numeroCartao, CVV)
12                         VALUES (%s, %s, %s)
13             """,
14                 linha["nomeCompleto"], linha["dataNascimento"], linha["NIF"], linha["numeroDocumento"],
15                 linha["dataValidadeDocumento"], linha["rua"], linha["localidade"], linha["codigoPostal"],
16                 linha["numeroTelemovel"], linha["email"], linha["dataValidadeCarta"], linha["habilitacao"],
17                 linha["dataValidadeCartao"], linha["numeroCartao"], linha["CVV"]
18             )
19         conn.commit()
20         conn.close()
21         print(f"Clientes CSV migrados com sucesso para stand {id_stand}.")
22
23
24 def migrar_tratores_csv(ficheiro, id_stand):
25     with open(ficheiro, newline='', encoding="utf-8") as f:
26         reader = csv.DictReader(f)
27         conn = conectar_mysql()
28         cursor = conn.cursor()
29         for linha in reader:
30             cursor.execute("""
31                 INSERT INTO Trator (modelo, marca, precoDiario, estado, idStand)
32                         VALUES (%s, %s, %s, %s)
33             """,
34                 (linha["modelo"], linha["marca"], float(linha["precoDiario"]), linha["estado"], id_stand))
35         conn.commit()
36         conn.close()
37         print(f"Tratores CSV migrados com sucesso para stand {id_stand}.")
38
39 def migrar_alugueres_csv(ficheiro, id_stand):
40     with open(ficheiro, newline='', encoding="utf-8") as f:
41         reader = csv.DictReader(f)
42         conn = conectar_mysql()
43         cursor = conn.cursor()
44         for linha in reader:
45             cursor.execute("""
46                 INSERT INTO Aluguer (dataInicio, dataTermino, precoTotal, metodoPagamento,
47                                     estadoPagamento, tipoPagamento, idCliente, idTrator, idFuncionario)
48                         VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
49             """,
50                 linha["dataInicio"], linha["dataTermino"], float(linha["precoTotal"]),
51                 linha["metodoPagamento"], linha["estadoPagamento"], linha["tipoPagamento"],
52                 linha["idCliente"], linha["idTrator"], linha["idFuncionario"]
53             )
54         conn.commit()
55         conn.close()
56         print(f"Alugueres CSV migrados com sucesso para stand {id_stand}.")
```

Figura 8.4: Funções de migração de CSV

Anexo 6

```
● ● ●
1 # ---- MIGRAÇÃO DE POSTGRESQL ---- #
2
3 def migrar_de_postgres_clientes(id_stand):
4     pg_conn = conectar_postgres()
5     pg_cursor = pg_conn.cursor()
6     pg_cursor.execute('SET search_path TO "AgroAuto";')
7     pg_cursor.execute("""
8         SELECT
9             "nomeCompleto",
10            "dataNascimento",
11            "NIF",
12            "numeroDocumento",
13            "dataValidadeDocumento",
14            "rua",
15            "localidade",
16            "codigoPostal",
17            "numeroTelemovel",
18            "email",
19            "dataValidadeCarta",
20            "habilitacao",
21            "dataValidadeCartao",
22            "numeroCartao",
23            "CVV"
24        FROM "Cliente"
25        """
26    )
27
28    mysql_conn = conectar_mysql()
29    mysql_cursor = mysql_conn.cursor()
30
31    for row in pg_cursor.fetchall():
32        mysql_cursor.execute("""
33            INSERT INTO Cliente (nomeCompleto, dataNascimento, NIF, numeroDocumento, dataValidadeDocumento, rua, localidade, codigoPostal,
34                                numeroTelemovel, email, dataValidadeCarta, habilitacao, dataValidadeCartao, numeroCartao, CVV)
35            VALUES (%s, %s, %s)
36        """, row)
37
38    mysql_conn.commit()
39    mysql_conn.close()
40    pg_conn.close()
41    print(f"Clientes migrados de PostgreSQL para stand {id_stand}.")
42
43 def migrar_de_postgres_funcionarios(id_stand):
44     pg_conn = conectar_postgres()
45     pg_cursor = pg_conn.cursor()
46     pg_cursor.execute('SET search_path TO "AgroAuto";')
47     pg_cursor.execute('SELECT "nomeCompleto", "numeroTelemovel" FROM "Funcionario"')
48
49
50    mysql_conn = conectar_mysql()
51    mysql_cursor = mysql_conn.cursor()
52
53    for row in pg_cursor.fetchall():
54        nome, tel = row
55        mysql_cursor.execute("SELECT idFuncionario FROM Funcionario WHERE numeroTelemovel = %s", (tel,))
56        if mysql_cursor.fetchone():
57            print(f"Funcionario '{nome}' com n° {tel} já existe. A ignorar.")
58            continue
59
60        mysql_cursor.execute("""
61            INSERT INTO Funcionario (nomeCompleto, numeroTelemovel, idStand)
62            VALUES (%s, %s, %s)
63        """, (nome, tel, id_stand))
64
65    mysql_conn.commit()
66    mysql_conn.close()
67    pg_conn.close()
68    print(f"Funcionários migrados de PostgreSQL para stand {id_stand}.")
69
70 def migrar_de_postgres_alugueres(id_stand):
71     pg_conn = conectar_postgres()
72     pg_cursor = pg_conn.cursor()
73     pg_cursor.execute('SET search_path TO "AgroAuto";')
74     pg_cursor.execute("""
75         SELECT
76             "dataInicio",
77             "dataTermino",
78             "precoTotal",
79             "metodoPagamento",
80             "estadoPagamento",
81             "tipoPagamento",
82             "idCliente",
83             "idTrator",
84             "idFuncionario"
85         FROM "Aluguer"
86        """
87    )
88
89
90    mysql_conn = conectar_mysql()
91    mysql_cursor = mysql_conn.cursor()
92
93    for row in pg_cursor.fetchall():
94        mysql_cursor.execute("""
95            INSERT INTO Aluguer (dataInicio, dataTermino, precoTotal, metodoPagamento,
96                                estadoPagamento, tipoPagamento, idCliente, idTrator, idFuncionario)
97            VALUES (%s, %s, %s, %s, %s, %s, %s, %s)
98        """, row)
99
100   mysql_conn.commit()
101  mysql_conn.close()
102  pg_conn.close()
103  print(f"Alugueres migrados de PostgreSQL para stand {id_stand}.")
```

Figura 8.5: Funções de migração de PostgreSQL

Anexo 7

```
● ● ●
1 # ---- MIGRAÇÃO JSON ---- #
2
3 def migrar_tratores_json(ficheiro, id_stand):
4     with open(ficheiro, encoding="utf-8") as f:
5         tratores = json.load(f)
6         conn = conectar_mysql()
7         cursor = conn.cursor()
8         for t in tratores:
9             cursor.execute("""
10                 INSERT INTO Trator (modelo, marca, precoDiario, estado, idStand)
11                 VALUES (%s, %s, %s, %s, %s)
12                 """ , (t["modelo"], t["marca"], float(t["precoDiario"]), t["estado"], id_stand))
13         conn.commit()
14         conn.close()
15         print(f"Tratores JSON migrados com sucesso para stand {id_stand}.")
16
17
18 def migrar_clientes_json(ficheiro, id_stand):
19     with open(ficheiro, encoding="utf-8") as f:
20         clientes = json.load(f)
21         conn = conectar_mysql()
22         cursor = conn.cursor()
23         for c in clientes:
24             cursor.execute("""
25                 INSERT INTO Cliente (
26                     nomeCompleto, dataNascimento, NIF, numeroDocumento,
27                     dataValidadeDocumento, rua, localidade, CodigoPostal,
28                     numeroTelemovel, email, dataValidadeCarta, habilitacao,
29                     dataValidadeCartao, numeroCartao, CVV
30                 )
31                 VALUES (%s, %s, %s)
32                 """
33                 , (
34                     c["nomeCompleto"], c["dataNascimento"], c["NIF"], c["numeroDocumento"],
35                     c["dataValidadeDocumento"], c["rua"], c["localidade"], c["codigoPostal"],
36                     c["numeroTelemovel"], c["email"], c["dataValidadeCarta"], c["habilitacao"],
37                     c["dataValidadeCartao"], c["numeroCartao"], c["CVV"]
38                 ))
39         conn.commit()
40         conn.close()
41         print(f"Clientes JSON migrados com sucesso para stand {id_stand}.")
42
43 def migrar_funcionarios_json(ficheiro, id_stand):
44     import json
45
46     with open(ficheiro, encoding="utf-8") as f:
47         funcionarios = json.load(f)
48         conn = conectar_mysql()
49         cursor = conn.cursor()
50
51         for f in funcionarios:
52             nome = f["nomeCompleto"]
53             tel = f["numeroTelemovel"]
54
55             cursor.execute("SELECT idFuncionario FROM Funcionario WHERE numeroTelemovel = %s", (tel,))
56             if cursor.fetchone():
57                 print(f"Funcionário '{nome}' com nº {tel} já existe. A ignorar.")
58                 continue
59
60             cursor.execute("""
61                 INSERT INTO Funcionario (nomeCompleto, numeroTelemovel, idStand)
62                 VALUES (%s, %s, %s)
63                 """
64                 , (nome, tel, id_stand))
65
66         conn.commit()
67         conn.close()
68         print(f"Funcionários JSON migrados com sucesso para stand {id_stand}.")
```

Figura 8.6: Funções de migração de JSON