



Universidade do Minho
Escola de Engenharia
Licenciatura em Engenharia Informática

Unidade Curricular de Bases de Dados

Ano Letivo de 2022/2023

El Refugio de Juegos de Quezada

David Teixeira [A100554]
João Pedro Pastore [A100543]
Luís Ferreira [A91672]
Tiago Rodrigues [A100827]

5 de junho de 2023

Data de Receção	
Responsável	
Avaliação	
Observações	

Resumo

O presente relatório descreve e acompanha o desenvolvimento de um Sistema de Gestão de Bases de Dados (SGBD). O objetivo deste SGBD é corrigir e otimizar as operações de uma pequena loja de aluguer de jogos, semelhante à Blockbuster. Ele visa melhorar a eficiência e manutenção dos dados da loja, utilizando operações e métodos considerados mais adequados pelo grupo, de acordo com os requisitos desta Unidade Curricular.

Ao longo deste relatório, serão apresentadas todas as fases do ciclo de vida de um SGBD, desde a Definição do Sistema até a Monitorização.

Primeiramente, abordamos o caso de estudo, identificando os principais problemas, objetivos e recursos necessários de forma clara e concisa.

Em seguida, realizamos uma análise extensa dos requisitos, garantindo que, no final do desenvolvimento do SGBD, todos os objetivos tenham sido devidamente cumpridos.

Com os requisitos bem elaborados, prosseguimos para o desenvolvimento de um modelo conceptual, que parte dos requisitos previamente analisados e aprovados. A partir desse modelo conceptual, elaboramos um modelo lógico, que foi automaticamente convertido em um modelo físico.

Com o modelo físico devidamente elaborado, procedemos ao preenchimento da base de dados com os dados necessários para o funcionamento da loja, e à implementação de operações de consulta e análise de dados. Em seguida, a base de dados é explorada e monitorizada para garantir o seu correto funcionamento, via demonstração.

Índice

1	Definição de Sistema	1
1.1	Contexto de aplicação e fundamentação do sistema	1
1.1.1	Contextualização	1
1.1.2	Fundamentação	2
1.2	Objetivos do Trabalho	2
1.3	Análise da viabilidade do processo	3
1.4	Recursos e Equipa de Trabalho	4
1.4.1	Recursos	4
1.4.2	Equipa de Trabalho	4
1.5	Revisão e Aprovação	5
1.6	Plano de Execução do Projeto	5
2	Levantamento e Análise de Requisitos	6
2.1	Método de levantamento e de análise de requisitos adotado	6
2.2	Organização dos requisitos levantados	6
2.2.1	Requisitos de descrição	6
2.2.2	Requisitos de Manipulação	8
2.2.3	Requisitos de controlo	9
2.3	Análise e validação geral dos requisitos	10

3	Modelação Conceptual	11
3.1	Apresentação da abordagem de modelação realizada	11
3.2	Identificação e caracterização das entidades	11
3.3	Identificação e caracterização dos relacionamentos	13
3.4	Identificação e caracterização da associação dos atributos com as entidades	16
3.5	Apresentação e explicação do diagrama ER produzido	19
4	Modelação Lógica	20
4.1	Construção e validação do modelo de dados lógico	20
4.1.1	Construção do Modelo de dados Lógico	20
4.1.2	Validação do Modelo de dados Lógico	25
4.2	Normalização de Dados	29
4.3	Conclusão	29
5	Implementação Física	30
5.1	Tradução do esquema lógico para o sistema de gestão de bases de dados escolhido em SQL	30
5.2	Definição e caracterização das vistas de utilização em SQL e tradução das interrogações do utilizador para SQL	31
5.2.1	Exemplos de Vistas	32
5.2.2	Exemplos de Interrogações	33
5.3	Cálculo do espaço da bases de dados (inicial e taxa de crescimento anual)	34
5.3.1	Tamanho Inicial da Base de Dados	34
5.3.2	Requisitos de Armazenamento	34
5.3.3	Taxa de Crescimento e Tamanho da Base De Dados Futura	36
6	Introdução de Dados no Sistema de Bases de Dados e Interface de utilização	38

6.1	Apresentação e modelo do sistema	38
6.2	Implementação do povoamento	42
6.3	Criação do Script em Python e a sua Implementação	43
6.3.1	Importação de Bibliotecas	43
6.3.2	Conexão com o Servidor MySQL	44
6.3.3	Função para Executar Consultas	44
6.3.4	Função para Executar Scripts SQL	45
6.3.5	Funções para Manipulação de Dados CSV	45
6.3.6	Limpeza da Base de Dados	46
6.3.7	Função execute_procedure	46
6.3.8	Função populate_database_from_csv	47
6.3.9	Função populate_database_from_sql	48
6.3.10	Menu Interativo	49
7	Implementação do Sistema de Painéis de Análise	51
7.1	Integração dos Dados no PowerBI e Definição e caracterização da vista de dados para análise	52
7.2	Apresentação e caracterização dos dashboards implementados	53
8	Conclusões e Trabalho Futuro	57
8.1	Anexos	59
8.1.1	Anexo 1 - Script de Inicialização da Base de Dados (CreateDB.sql)	59
8.1.2	Anexo 2 - Script de Povoamento Inicial	61
8.1.3	Anexo 2 - Script de Criação das Vistas (Views.sql)	63
8.1.4	Anexo 3 - Script de Criação das Procedures (Procedures.sql)	66
8.1.5	Anexo 4 - Script de Criação das Queries (Queries.sql)	67

Lista de Figuras

3.1	Relação: Aluguer cabeçalho - Cliente	13
3.2	Relação: Aluguer cabeçalho - Cliente	14
3.3	Relação: Cliente - Inquérito	14
3.4	Relação: Vendedor - Aluguer cabeçalho	15
3.5	Relação: Vendedor - Inquérito	16
3.6	Associação dos atributos com a entidade Vendedor	16
3.7	Associação dos atributos com a entidade Cliente	17
3.8	Associação dos atributos com a entidade Jogo	17
3.9	Associação dos atributos com a entidade Inquérito	17
3.10	Associação dos atributos com a entidade Aluguer Cabeçalho	18
3.11	Modelo conceptual com a relação Aluguer Linhas	18
3.12	Modelo conceptual	19
4.1	Tabela Jogo	20
4.2	Tabela AluguerLinhas	21
4.3	Tabela AluguerCab	22
4.4	Tabela Vendedor	23
4.5	Tabela Cliente	23
4.6	Tabela Inquérito	24

4.7	Modelo Lógico Obtido	25
4.8	Validação de [RM1]	26
4.9	Validação de [RM3]	26
4.10	Validação de [RM7]	27
4.11	Validação de [RM9]	28
5.1	Código da respetiva <i>Query</i>	32
5.2	Código da respetiva <i>Query</i>	32
5.3	Código da respetiva <i>Query</i>	33
5.4	Código da respetiva <i>Query</i>	33
6.1	Tabela de Jogos	38
6.2	Tabela de Vendedores	39
6.3	Tabela de Clientes	39
6.4	Tabela de Inqueritos	40
6.5	Tabela de Aluguer Cabeçalho	40
6.6	Tabela de Aluguer Linhas	41
6.7	Povoamento de Clientes : Exemplo de inserção de dados na tabela Cliente.	42
6.8	Povoamento de Vendedores : Exemplo de inserção de dados na tabela Vendedor.	42
6.9	Povoamento de Jogos : Exemplo de inserção de dados na tabela Jogo.	42
6.10	Povoamento de inquéritos : Exemplo de inserção de dados na tabela Inquérito.	42
6.11	Povoamento de Cabeçalho do aluguer : Exemplo de inserção de dados na tabela AluguerCab.	42
6.12	Povoamento de Linhas do aluguer : Exemplo de inserção de dados na tabela AluguerLinha.	43
6.13	Importação de Bibliotecas	43

6.14	Conexão com o Servidor MySQL	44
6.15	Função para Executar Consultas	44
6.16	Função para Executar Scripts SQL	45
6.17	Funções para Manipulação de Dados CSV	46
6.18	Limpeza da Base de Dados	46
6.19	Função <code>execute_procedure</code>	47
6.20	Funções <code>populate database from csv/populate database from sql</code>	49
6.21	Menu Interativo	50
7.1	<i>Get Data</i> no <i>PowerBI</i>	52
7.2	Conexão à Base de Dados, via <i>localhost</i>	53
7.3	Distribuição dos jogos por ano de lançamento	54
7.4	Jogos mais alugados desde sempre	54
7.5	Plataformas mais alugadas	54
7.6	Datas de lançamento de jogos em que houve mais alugueres	55
7.7	Jogos mais e menos apreciados pelos Clientes	55
7.8	Jogos que os Clientes mais querem jogar	56
7.9	Vendedores com maior número de alugueres	56

Lista de Tabelas

5.1	Requisitos de armazenamento	34
5.2	Tamanho de cada atributo da tabela Vendedor, e o respectivo tamanho total	34
5.3	Tamanho de cada atributo da tabela Inquerito, e o respectivo tamanho total	35
5.4	Tamanho de cada atributo da tabela AluguerCab, e o respectivo tamanho total	35
5.5	Tamanho de cada atributo da tabela AluguerLinhas, e o respectivo tamanho total	35
5.6	Tamanho de cada atributo da tabela Cliente, e o respectivo tamanho total	36
5.7	Tamanho de cada atributo da tabela Jogo, e o respectivo tamanho total .	36

Capítulo 1

Definição de Sistema

1.1 Contexto de aplicação e fundamentação do sistema

1.1.1 Contextualização

A loja de videojogos "El Refugio de Juegos de Quezada" localiza-se no bairro "São José das Pias", na cidade de "Ambriz", no distrito de Braga. Foi fundada em 1989 por Saul Ribeiro Quezada, um imigrante chileno que fugiu para Portugal em 1987 para escapar ao Regime de Augusto Pinochet durante a ditadura chilena. A loja pode ser comparada a um "Blockbuster de Videojogos", pois permite que os clientes aluguem temporariamente jogos, com prazo máximo de devolução de dois meses após o aluguer.

A loja possui tamanho médio e uma vitrine que exhibe os jogos mais recentes disponíveis. No interior, há diversas prateleiras e estantes com caixas de jogos em exposição, embora nenhuma delas contenha o jogo em si, sendo apenas decorativas. Cada caixa possui um autocolante na parte frontal com o preço do aluguer e um sensor na parte traseira para evitar possíveis roubos. O balcão de receção, onde o senhor Saul recebe os clientes, tem acesso a um grande armazém onde os jogos físicos são armazenados. Durante os anos em que o senhor Saul esteve à frente do negócio, os jogos eram organizados por ordem de lançamento, mas após o seu falecimento, a organização passou a ser feita com base no *Critics Score* dos jogos. Quando um jogo é alugado, os dados do cliente, como nome completo, idade, número de telemóvel e data de aluguer, são registados em um livro (ou em um arquivo de computador nos dias de hoje). Caso o cliente não devolva o jogo dentro de dois meses, ele é contactado.

Inicialmente, o senhor Saul era o único funcionário da loja. No entanto, no decorrer do tempo, o seu filho, Alberto Quezada, começou a ajudá-lo. Alguns meses depois,

um amigo de Alberto, José Cocas de Sousa, também começou a trabalhar na loja, sendo responsável pela limpeza e organização do armazém. Com o passar do tempo, mais pessoas foram contratadas para trabalhar na loja. Infelizmente, como mencionado anteriormente, o senhor Saul faleceu aos 65 anos em 2015, vítima de cancro pulmonar, deixando a loja sob os cuidados do seu filho Alberto e de José Cocas.

1.1.2 Fundamentação

Desde o falecimento do pai, o senhor Alberto tem enfrentado várias dificuldades em gerir o negócio. Ele tem adquirido uma grande quantidade de videojogos que acabam por não ser alugados pelos clientes, e recebe queixas de clientes que desejam alugar um determinado jogo, mas que não está mais disponível em stock. Um outro problema está relacionado com o preço dos videojogos, pois muitos clientes reclamam dos valores de aluguer de certos jogos, considerando-os elevados em relação ao conteúdo oferecido.

No entanto, quando tudo parecia perdido, um cliente da loja, o senhor engenheiro Paulo Maia Santos, após conversar com o senhor Alberto, percebeu que todos esses problemas poderiam estar relacionados com o método de organização das informações da loja. Por isso, ofereceu-se para desenvolver um sistema de base de dados capaz de armazenar informações importantes e relevantes sobre os jogos da loja. Dessa forma, o senhor Alberto poderá obter informações sobre quais jogos são mais apreciados pelos clientes (e os menos apreciados), quais jogos são mais solicitados (o que ajudará numa gestão de stock futura) e quais jogos precisam ter o seu stock aumentado ou reduzido. Além disso, o sistema permitirá consultar outros dados relevantes relacionados não apenas com jogos, mas também com funcionários e clientes, por exemplo.

1.2 Objetivos do Trabalho

Satisfeito por ter encontrado uma solução para o seu problema, o senhor Alberto elaborou uma lista de objetivos que pretende alcançar com a implementação do sistema de base de dados na sua loja. São eles:

- Gerir o inventário de videojogos disponíveis para aluguer. Isso inclui informações detalhadas sobre cada jogo, como o título, plataforma, género, data de lançamento e disponibilidade.
- Registrar os pedidos dos clientes, incluindo os jogos específicos que desejam alugar, as datas de aluguer e as informações de contacto.
- Armazenar informações históricas sobre o desempenho dos jogos em aluguer. Isso inclui dados como a frequência de aluguer, as preferências dos clientes e as receitas

geradas por cada jogo.

1.3 Análise da viabilidade do processo

Na sequência da definição dos objetivos, foi feita uma avaliação sobre a capacidade e os recursos que a loja tem para conseguir atingir os mesmos:

- Viabilidade técnica: A loja possui infraestruturas suficientemente capazes de suportar a produção de um SGBD. A equipa do engenheiro Paulo Santos, também foi analisada e considerada capaz e indicada para o trabalho.
- Viabilidade financeira: Apesar da loja estar a passar por momentos financeiros não muito favoráveis, o senhor Alberto decidiu arriscar em avançar para a produção do sistema, especialmente, após lhe ter sido prometido que o sistema poderia permitir recuperar o dinheiro perdido, em, pelo menos, 2 anos.
- Viabilidade operacional: As operações e funcionalidades do SGBD foram consideradas úteis para resolver os problemas da loja. Foi também verificado se existiam recursos suficientes para gerir e manter o sistema, como equipa dedicada, políticas de backup e recuperação, e planos de contingência para lidar com possíveis falhas ou interrupções.

1.4 Recursos e Equipa de Trabalho

1.4.1 Recursos

Humanos:

- Trabalhadores da loja, clientes e funcionários da empresa de desenvolvimento da base de dados.

Materiais:

- Inquéritos, com perguntas sobre o jogo que os clientes mais desejam jogar e o jogo que os clientes mais e menos gostaram de jogar.
- Hardware (1 servidor, 1 ponto de venda).
- Software (Sistema de Gestão de Bases de Dados e Aplicações de vendas e aprovisionamento).

1.4.2 Equipa de Trabalho

Pessoal Interno:

- Alberto Ribeiro Quezada, José Cocas de Sousa e outros vendedores da loja: Funcionamento da loja, atendimento aos clientes, entrega de questionários/inquéritos, validação dos alugueres realizados, definição de preços de aluguer.

Pessoal Externo:

- Engenheiro Paulo Maia Santos e respetiva equipa de trabalho: Levantamento dos requisitos, modelação do sistema, implementação do sistema, organização dos dados, avaliação dos dados dos inquéritos.

Outros:

- Clientes selecionados: Participação nos inquéritos de opinião e avaliação dos jogos.

1.5 Revisão e Aprovação

Depois de toda a definição do sistema, bem como a sua fundamentação, objetivos e viabilidade terem sido criados, tudo foi analisado pelos responsáveis e funcionários da loja. Em reunião e após algum tempo de ponderação, o grupo validou tudo o que foi definido. Sendo assim, o engenheiro Paulo e sua equipa foram contactados, e foi dada a aprovação para o início da criação da base de dados.

1.6 Plano de Execução do Projeto

Para planear o processo de desenvolvimento do sistema, foi realizado um plano concreto de trabalho, bem como um cronograma de execução. Neste plano, os vendedores continuariam as suas atividades comuns de alugar jogos, além de disponibilizar inquéritos para que os clientes pudessem preencher. Com os dados provenientes dos inquéritos armazenados, o próximo passo seria entregá-los à equipa responsável pela criação da base de dados. Além dos dados dos jogos e inquéritos, outras informações existentes incluem os dados dos clientes, dos vendedores e das faturas relacionadas aos alugueres feitos pelos clientes na loja. Com todos esses dados, a equipa irá construir a base de dados.

Capítulo 2

Levantamento e Análise de Requisitos

2.1 Método de levantamento e de análise de requisitos adotado

Para dar início ao processo de levantamento de requisitos, foi estabelecido que a equipa liderada pelo Engenheiro Paulo Maia Santos realizará reuniões periódicas com o Sr. Alberto e José Cocas de Sousa, com o objetivo de identificar os diversos processos ocorrentes na loja. Além disso, foram minuciosamente analisados os documentos contendo informações relevantes sobre o stock dos jogos.

2.2 Organização dos requisitos levantados

2.2.1 Requisitos de descrição

O objetivo dos requisitos de descrição é definir como os dados e informações devem ser representados, estruturados e organizados no sistema. Esses requisitos ajudam a garantir que o sistema possa lidar com diferentes tipos de dados e informações de maneira consistente e eficiente. Desta forma decidiu-se que os dados relativos aos jogos e clientes devem ser organizados das seguintes maneiras:

1. Cada “jogo” registado no sistema tem de ter associado os seguintes dados: id, nome, plataforma de lançamento, data de lançamento, valor de aluguer, sinopse, nota dos críticos, pedidos de aluguer e unidades em stock.

- A nota dos críticos deve ser representada num número de 0 a 100.
 - As plataformas de lançamento incluem várias consolas, e podem, também incluir o computador.
2. Cada “cliente” registado no sistema tem de ter associado os seguintes dados: NIF, número de telemóvel, nome e idade.
- O NIF deve ser único para cada cliente, funcionando como uma chave primária.
3. As faturas são divididas em duas tabelas, “Aluguer Cabeçalho” (que contém as informações gerais de um aluguer) e “Aluguer Linhas” (que contém as informações específicas de um aluguer).
4. Cada “aluguer cabeçalho” no sistema tem de ter associado os seguintes dados: id, preço total, período de aluguer, data do aluguer, NIF do cliente, ID do vendedor, método de pagamento.
- O NIF do cliente e o ID do vendedor são chaves estrangeiras que indicam quem fez o aluguer e quem alugou o jogo.
5. Cada “aluguer linhas” registado no sistema tem de ter associado os seguintes dados: id do aluguer cabeçalho, id do jogo, preço total, preço unitário e quantidade.
- O id do jogo e id do cabeçalho são chaves estrangeiras que juntas compõem a chave primária do “aluguer linhas”.
 - O preço unitário indica o preço de um jogo apenas. O preço total é obtido através da multiplicação entre o preço unitário e a quantidade.
6. Cada “vendedor” registado no sistema tem de ter associado os seguintes dados: id, nome.
- O id trata-se da sua chave primária.
7. Cada “inquérito” registado no sistema tem de ter associado os seguintes dados: id, jogo que mais gostou de jogar, jogo que menos gostou de jogar, jogo que quer jogar, id do vendedor e NIF do cliente.
- O id do vendedor e NIF do cliente indicam o vendedor que entregou e registou o inquérito e o cliente que respondeu ao inquérito.

2.2.2 Requisitos de Manipulação

Os requisitos de manipulação definem como as informações devem ser processadas, manipuladas e transformadas dentro do sistema. Eles ajudam a garantir que o sistema possa executar operações precisas e consistentes sobre os dados e informações, de acordo com as necessidades do utilizador e do negócio.

Eis os requisitos de manipulação:

1. Deve ser possível obter uma lista com os três jogos mais alugados pelos clientes.
2. Deve ser possível obter uma lista com os três jogos menos alugados pelos clientes.
3. Deve ser implementada a funcionalidade que permite verificar as plataformas que tiveram mais jogos alugados pelos clientes.
4. Deve ser implementada a funcionalidade que permite visualizar as datas de lançamento que tiveram maior volume de alugueres, isto é, o ano onde foram produzidos melhores jogos.
5. Deve ser possível identificar os clientes que mais investiram na loja.
6. Deve ser possível identificar os clientes que menos investiram na loja.
7. Deve ser implementada a funcionalidade que permite listar os compradores que alugaram o mesmo jogo mais do que uma vez.
8. Deve ser implementada a funcionalidade que permite consultar os jogos que os clientes mais gostaram de jogar.
9. Deve ser implementada a funcionalidade que permite consultar os jogos que os clientes menos gostaram de jogar.
10. Deve ser possível identificar os jogos que os clientes pretendem jogar.
11. Deve ser possível identificar os clientes que efetuaram mais alugueres na loja.
12. Deve ser possível identificar os clientes que efetuaram menos alugueres na loja.
13. Deve ser possível identificar os vendedores que distribuíram mais inquéritos na loja.
14. Deve ser possível identificar os vendedores que não distribuíram quaisquer inquéritos na loja.
15. Deve ser possível aceder a uma lista com todos os jogos que a loja possui (quer estejam em aluguer ou não).
16. Deve ser possível aceder a uma lista com informações acerca dos inquéritos.

17. Deve ser possível aceder a uma lista com informações acerca dos clientes.
18. Deve existir a opção de obter uma lista com todos os jogos de uma plataforma a desejar.
19. Deve ser possível obter uma lista organizada de jogos com base na nota dos críticos, em caso de empate, com base no preço e, em caso de novo empate, com base na data de lançamento do jogo.
20. Deve ser possível obter uma lista organizada de jogos com base no número de alugueres dos mesmos.
21. Deve ser possível inserir um novo registo de um jogo no sistema.
22. Deve ser possível inserir um novo registo de um cliente no sistema.

2.2.3 Requisitos de controlo

Os requisitos de controlo definem como o sistema deve gerir e monitorar as atividades dos utilizadores e do sistema em geral. Esses requisitos ajudam a garantir a segurança do sistema, evitando atividades maliciosas ou erros humanos que possam comprometer o desempenho ou a integridade dos dados.

Eis os requisitos de controlo:

1. Os dados dos jogos: nome, plataforma de lançamento, data de lançamento, sinopse e nota dos críticos, estão disponíveis para consulta por qualquer pessoa, porém não devem ser modificados.
2. Os dados dos jogos: pedidos de aluguer, só podem ser consultados e modificados pelos vendedores e pelo dono.
3. Os dados dos jogos: valor de aluguer, estão disponíveis para consulta por qualquer pessoa. No entanto, somente os vendedores e o dono têm permissão para modificá-los.
4. Os dados dos jogos: unidades em stock, estão restritos para consulta apenas para os vendedores e o dono. Além disso, somente o dono tem permissão para modificar esses dados.
5. Os dados dos clientes, inquéritos e vendedores estão disponíveis apenas para visualização para os vendedores e para o dono. Nenhum desses dados pode ser modificado.
6. Os vendedores e o dono têm permissão para visualizar os dados de todas as faturas, no entanto, os clientes só podem visualizar os dados das suas próprias faturas.

7. Somente os clientes têm permissão para fazer alugueres de jogos e responder a inquéritos.
8. O dono deve ter um perfil privado que, além de ter todos os privilégios mencionados anteriormente, inclui a capacidade de adicionar um novo jogo à base de dados.

2.3 Análise e validação geral dos requisitos

O processo de levantamento e análise de requisitos é um componente crucial no design e implementação de um sistema de gestão de bases de dados. As modelagens subsequentes são baseadas nos critérios definidos nessa fase, o que torna esse processo fundamental.

Durante o processo de levantamento, foram identificadas três fases: **requisitos de descrição, requisitos de manipulação e requisitos de controlo**.

Na fase de levantamento dos **requisitos de descrição**, a atenção foi voltada para a análise dos elementos essenciais para o funcionamento de uma loja de alugueres e a forma como esses elementos se relacionam entre si. Essa análise resultou nos requisitos apresentados.

Posteriormente, na elaboração dos **requisitos de manipulação**, foram avaliadas quais informações seriam relevantes para extrair da loja, levando em consideração os elementos e relacionamentos identificados na fase anterior.

Na fase final, durante o levantamento dos **requisitos de controlo**, foram definidas as restrições de administração da base de dados para garantir a coerência do sistema como um todo.

Ao longo desse processo, houve uma análise contínua dos requisitos levantados, envolvendo discussões para garantir a sua adaptação aos cenários reais e evitar situações pouco realistas no contexto da gestão do serviço proposto.

Depois de todos estes requisitos terem sido definidos e apresentados aos funcionários da loja e ao Senhor Alberto, tudo foi considerado credível. A equipa teve a autorização do Sr. Alberto para passar para a próxima fase do desenvolvimento do SGBD.

Capítulo 3

Modelação Conceptual

3.1 Apresentação da abordagem de modelação realizada

Com base nos requisitos previamente levantados e validados no capítulo anterior, é necessário iniciar o processo de planeamento do design da Base de Dados que será implementado.

A primeira etapa desse design é a criação de um Modelo Conceptual, que é um modelo independente do Sistema de Gestão de Bases de Dados a ser utilizado. O objetivo desse modelo é representar as diferentes entidades que irão compor o nosso sistema, bem como os atributos relacionados a cada uma delas e os respetivos relacionamentos.

Para isso, utilizaremos um Diagrama de Entidade-Relacionamento. A abordagem para criação desse diagrama será incremental: inicialmente, representaremos as entidades que foram identificadas como necessárias para modelar o sistema. Em seguida, representaremos os atributos de cada uma dessas entidades e identificaremos os relacionamentos, prestando atenção especial à adição de atributos nas entidades no caso de uma relação N para N. Por fim, concluiremos o processo representando as multiplicidades de cada relacionamento modelado.

3.2 Identificação e caracterização das entidades

Tendo em conta os requisitos levantados foi identificada a necessidade de criar as seguintes entidades:

- **Jogo** - A entidade jogo representa o artigo que é alugado pela loja. Este é caracterizado pelo seu nome, data de lançamento, plataforma de lançamento, sinopse, critics score, valor de aluguer, unidades em stock e pedidos de aluguer, sendo a nossa entidade mais complexa.
- **Aluguer cabeçalho** - Esta entidade representa uma pseudo fatura estando dividida em duas partes, aonde esta primeira tem o preço, período de aluguer, data e método de pagamento. Por fim esta entidade também tem na sua presença um identificador do aluguer linhas para indicar quais as presentes no cabeçalho.
- **Cliente** - Esta entidade representa o consumidor dos nossos serviços, tendo como características o seu nome, idade e como forma de identificação o seu NIF.
- **Vendedor** - A entidade vendedor representa o funcionário da loja que estará encarregue de realizar a manutenção e criação de inquéritos e criação de "Aluguer cabeçalho" para o aluguer de um produto. Este é caracterizado pelo seu nome e identificado pelo seu ID.
- **Inquérito** - Esta última entidade representa a nossa adição à loja de forma a tentar melhorar o seu funcionamento e trazer um maior desempenho comercial. É identificada pelo seu ID e definida pelo jogo que quer jogar, o jogo que mais gostou e o jogo que menos gostou.

3.3 Identificação e caracterização dos relacionamentos

▪ Relacionamento Aluguer Cabeçalho - Cliente

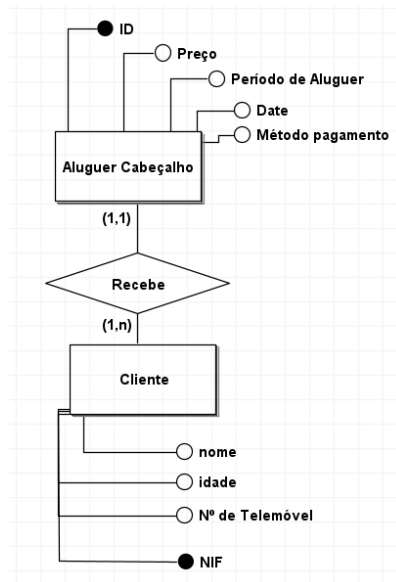


Figura 3.1: Relação: Aluguer cabeçalho - Cliente

Relacionamento: Cliente recebe Aluguer Cabeçalho

Descrição: Todos os Clientes na realização de um aluguer receberão um Aluguer Cabeçalho que terá informações do aluguer.

Cardinalidade: Cliente (1..*) - Aluguer Cabeçalho (1).

A cada cliente será associado um Aluguer Cabeçalho e vice-versa. Um cliente poderá ter várias faturas a si associadas de outros alugueres.

▪ Relacionamento Jogo - Aluguer Cabeçalho

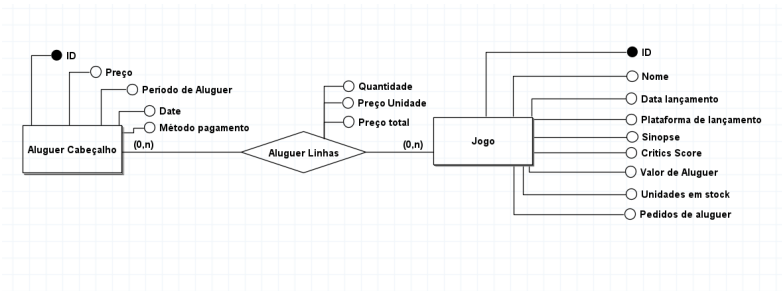


Figura 3.2: Relação: Aluguer cabeçalho - Cliente

Relacionamento: Jogo está contido numa Fatura (Aluguer Cabeçalho)

Descrição: O Jogo estará inserido numa fatura que será o aluguer cabeçalho, mas visto que este relacionamento tem uma cardinalidade de n para n é necessário dar atributos ao mesmo para identificar. Para isso foi criado o Aluguer Linhas que armazenará atributos para ser feita a associação entre as duas entidades, e este mesmo representará as várias linhas de uma fatura com essas informações.

Cardinalidade: Jogo (0..*) - Aluguer Cabeçalho (0..*).

Todos os Jogos poderão estar associados e presentes num qualquer Aluguer Cabeçalho via Aluguer Linhas.

Atributos: A quantidade irá indicar quantas cópias do jogo o Cliente pretende alugar. De seguida, teremos o Preço por Unidade que indicará o valor singular do jogo e Preço total que indicará o preço da soma das várias cópias alugadas.

▪ Relacionamento Cliente - Inquérito

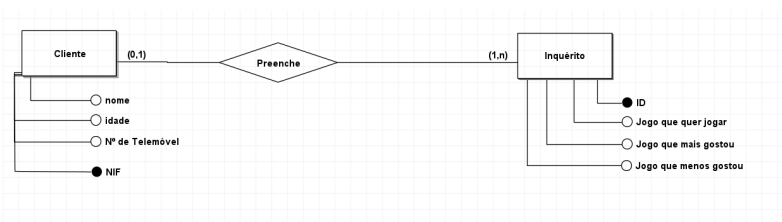


Figura 3.3: Relação: Cliente - Inquérito

Relacionamento: Cliente preenche Inquérito.

Descrição: Um cliente ao realizar um aluguer é pedido se pode preencher um inquérito para o melhoramento do funcionamento da loja.

Cardinalidade: Cliente (0..1) - Inquérito (1..*)

Cada cliente tem a opção de preencher o inquérito, podendo este não o fazer se assim o desejar. Caso queira, poderá preencher vários inquéritos correspondentes a vários alugueres.

- **Relacionamento Vendedor - Aluguer cabeçalho**

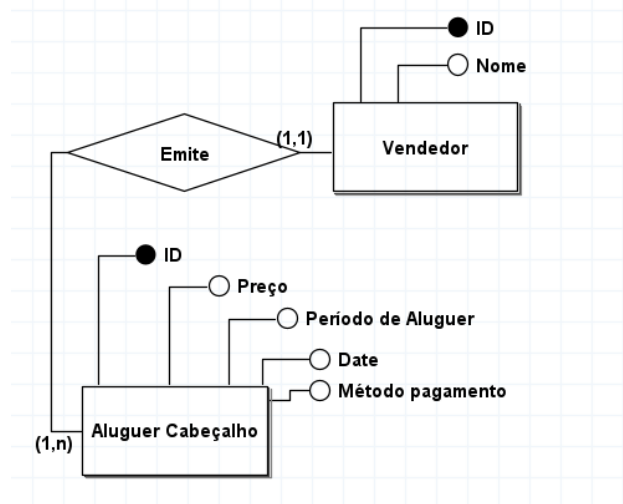


Figura 3.4: Relação: Vendedor - Aluguer cabeçalho

Relacionamento: Um vendedor cria o Aluguer Cabeçalho.

Descrição: Na realização de um aluguer, é emitida uma fatura pelo vendedor caracterizada como "Aluguer Cabeçalho" aonde estará o preço do aluguer, o seu período, data e o método de pagamento utilizado.

Cardinalidade: Vendedor (1) - Aluguer Cabeçalho (1..*)

Um vendedor terá a si associado várias faturas (Aluguer cabeçalho), mas cada fatura só terá presente um vendedor que será o que a gerou.

▪ Relacionamento Vendedor - Inquérito

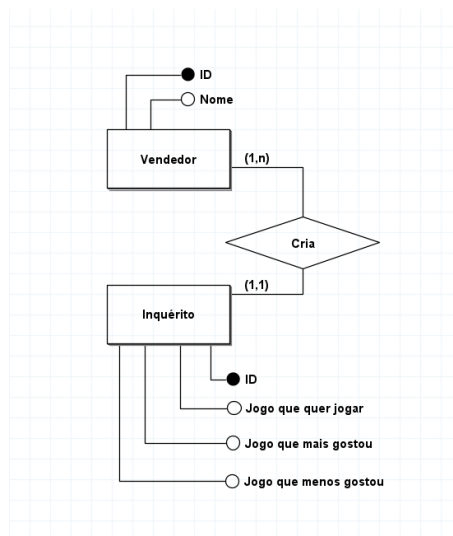


Figura 3.5: Relação: Vendedor - Inquérito

Relacionamento: Vendedor cria Inquérito

Descrição: Após ser realizado o aluguer, o vendedor gera um inquérito para que depois possa ser entregue ao cliente, caso este pretenda o preencher.

Cardinalidade: Vendedor (1..*) - Inquérito (1)

Cada vendedor terá associado a si vários inquéritos, mas um inquérito só poderá ter um vendedor associado a si que será o que o gerou.

3.4 Identificação e caracterização da associação dos atributos com as entidades

Entidade	Atributo	Descrição	Chave Primária	Tipo e Tamanho	Pode ser Nulo
Vendedor	ID	Identificador do Vendedor	Sim	INT	Não
	Nome	Nome do Vendedor	Não	VARCHAR (75)	Não

Figura 3.6: Associação dos atributos com a entidade Vendedor

Entidade	Atributo	Descrição	Chave Primária	Tipo e Tamanho	Pode ser Nulo
Cliente	Nome	Nome do cliente	Não	VARCHAR (75)	Não
	Idade	Idade do cliente	Não	INT	Não
	Número de telemóvel	Número de telemóvel do cliente	Não	INT	Não
	NIF	NIF do cliente	Sim	VARCHAR (75)	Não

Figura 3.7: Associação dos atributos com a entidade Cliente

Entidade	Atributo	Descrição	Chave Primária	Tipo e Tamanho	Pode ser Nulo
Jogo	ID	ID do jogo	Sim	INT	Não
	Valor de Aluguer	Preço do jogo	Não	DOUBLE	Não
	Critics Score	Nota dada pelos críticos ao jogo	Não	VARCHAR (75)	Não
	Pedidos de Aluguer	Pedidos de aluguer do jogo	Não	INT	Não
	Plataforma de Lançamento	Plataforma de lançamento do jogo	Não	VARCHAR (75)	Não
	Nome	Nome do jogo	Não	VARCHAR (75)	Não
	Data Lançamento	Data de lançamento do jogo	Não	DATE	Não
	Sinopse	Sinopse do jogo	Não	VARCHAR (512)	Não
	Unidades em Stock	Unidades em stock do jogo	Não	VARCHAR (75)	Não

Figura 3.8: Associação dos atributos com a entidade Jogo

Entidade	Atributo	Descrição	Chave Primária	Tipo e Tamanho	Pode ser Nulo
Inquérito	ID	Identificador do inquérito	Sim	INT	Não
	Jogo que quer jogar	ID do jogo que o cliente mais quer jogar	Não	INT	Não
	Jogo que mais jogou	ID do jogo que o cliente mais jogou	Não	INT	Não
	Jogo que menos jogou	ID do jogo que o cliente menos jogou	Não	INT	Não

Figura 3.9: Associação dos atributos com a entidade Inquérito

Entidade	Atributo	Descrição	Chave Primária	Tipo e Tamanho	Pode ser Nulo
Aluguer Cabeçalho	ID	Identificador do aluguer	Sim	INT	Não
	Preço	Preço total da compra	Não	DOUBLE	Não
	Período Aluguer	Tempo de aluguer do jogo dado pela loja	Não	DOUBLE	Não
	Date	Data do aluguer	Não	DATE	Não
	Método de pagamento	Método de pagamento da compra (dinheiro/cartão)	Não	VARCHAR (75)	Não

Figura 3.10: Associação dos atributos com a entidade Aluguer Cabeçalho

Relação	Atributo	Descrição	Tipo e Tamanho
Aluguer Linhas	Quantidade	Quantidade do jogo	INT
	Preço Unidade	Preço singular do item	DOUBLE
	Preço Total	Preço do item somando as cópias alugadas	DOUBLE

Figura 3.11: Modelo conceptual com a relação Aluguer Linhas

3.5 Apresentação e explicação do diagrama ER produzido

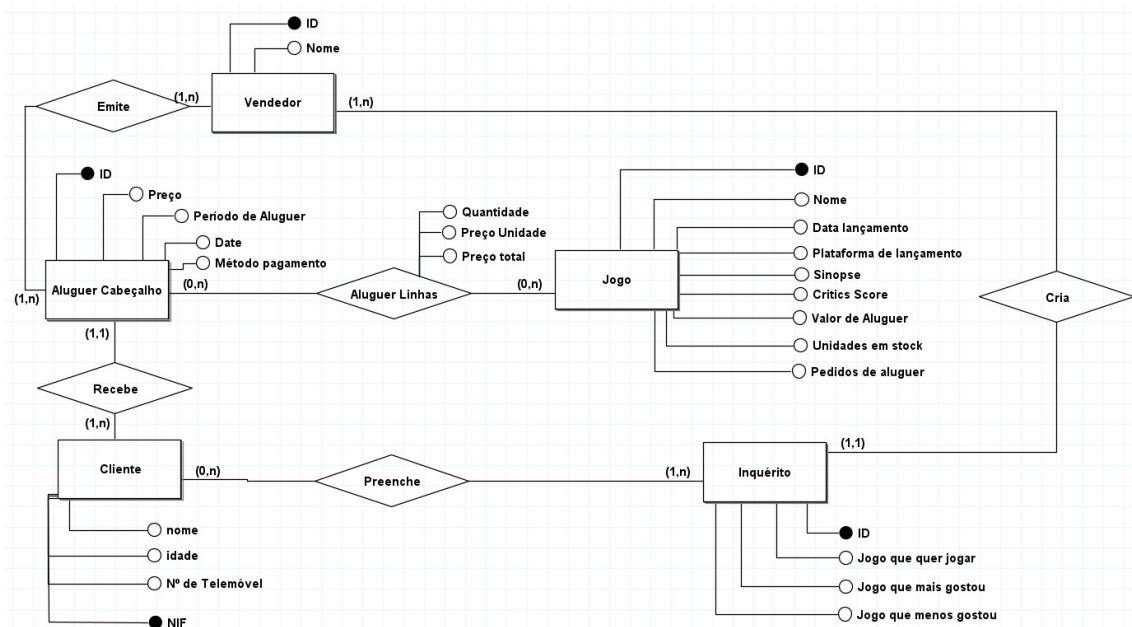


Figura 3.12: Modelo conceptual

Tendo em conta os relacionamentos presentes e as suas entidades, o diagrama pode ser interpretado da seguinte forma: Cada Cliente pode realizar um aluguer recebendo um Aluguer Cabeçalho que é emitido pelo Vendedor que servirá de fatura. Aquando do aluguer, o Vendedor também cria um Inquérito que será preenchido pelo Cliente se assim o entender. O Aluguer Cabeçalho terá na sua estrutura as Aluguer Linhas que irão conter os Jogos alugados e a sua quantidade para associar à fatura de aluguer.

Capítulo 4

Modelação Lógica

4.1 Construção e validação do modelo de dados lógico

4.1.1 Construção do Modelo de dados Lógico

O modelo lógico foi construído a partir do modelo conceptual, no qual as entidades foram transformadas em tabelas e os identificadores de uma entidade passaram a ser chamados de *Primary Key* ou *Foreign Key* (este último, caso haja alguma informação importante de um dos requisitos em outra tabela). Após a conversão, foram obtidas as seguintes tabelas.

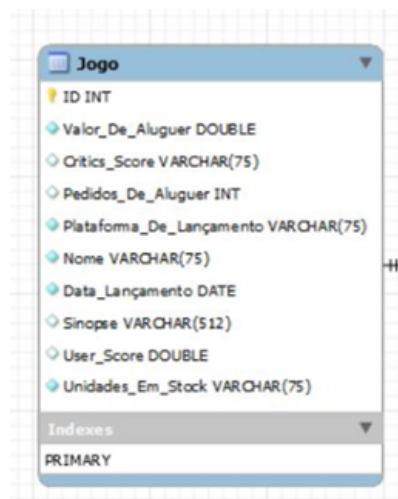


Figura 4.1: Tabela Jogo

Chave Primária: ID (INT)

Atributos:

- Valor de Aluguer (DOUBLE)
- Critics Score (VARCHAR(75))
- Pedidos De Aluguer (INT)
- Plataforma de Lançamento (VARCHAR(75))
- Nome (VARCHAR(75))
- Data de Lançamento (DATE)
- Sinopse (VARCHAR(512))
- User Score (DOUBLE)
- Unidades Em Stock (VARCHAR(75))

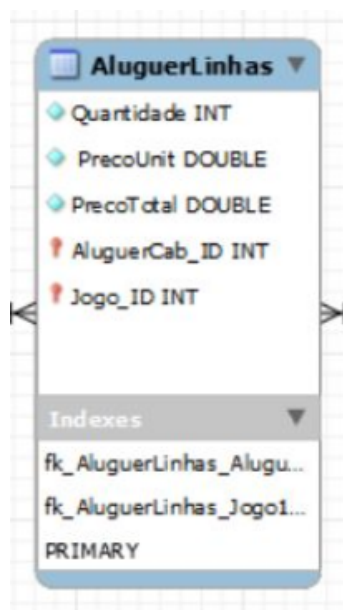


Figura 4.2: Tabela AluguerLinhas

Chave Primária Composta: JogoID (INT) e AluguerCabID (INT)

Atributos:

- Quantidade (INT)

- PreçoUnit (DOUBLE)
- PreçoTotal (DOUBLE)

Chaves Estrangeiras: JogolID (INT) e AluguerCabID (INT)

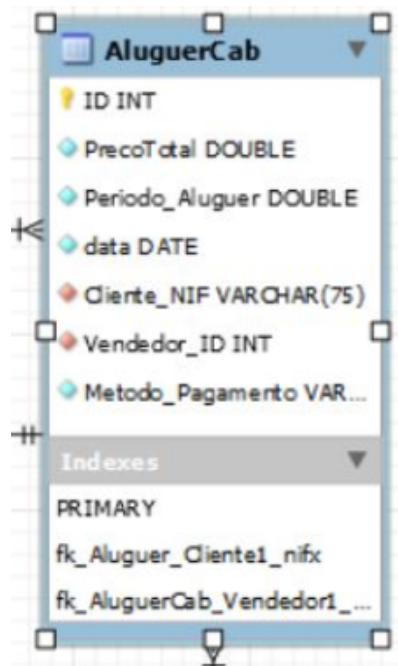


Figura 4.3: Tabela AluguerCab

Chave Primária: ID (INT)

Atributos:

- MetodoPagamento (VARCHAR(75))
- data (DATE)
- PeriodoAluguer (DOUBLE)
- PreçoTotal (DOUBLE)

Chaves Estrangeiras:

- ClienteNIF (VARCHAR(75))
- VendedorID (INT)

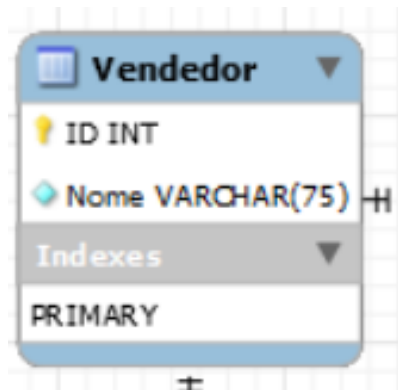


Figura 4.4: Tabela Vendedor

Chave Primária: ID (INT)

Atributos:

- Nome (VARCHAR(75))

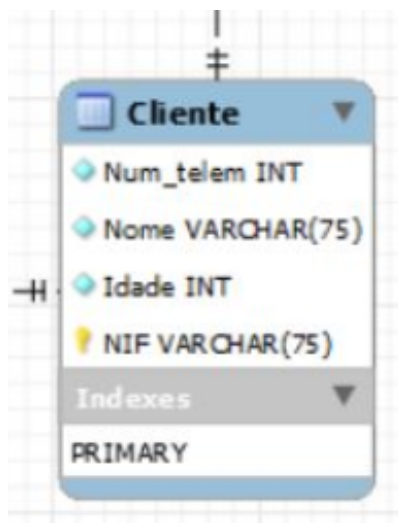


Figura 4.5: Tabela Cliente

Chave Primária: NIF (VARCHAR(75))

Atributos: Nome (VARCHAR(75)), NumTelem (INT) e Idade (INT)

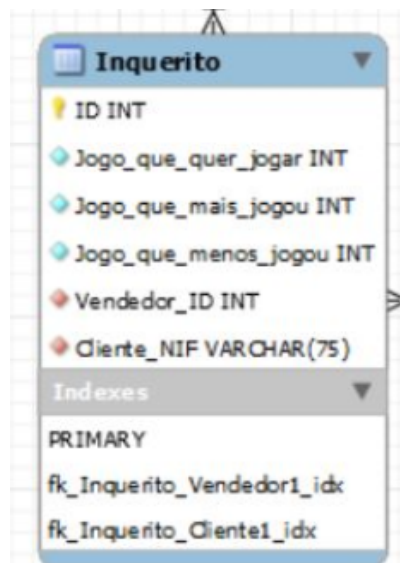


Figura 4.6: Tabela Inquérito

Chave Primária: ID (INT)

Atributos: jogoquequerjogar (INT), jogoquemaisjogou (INT), jogoquemenosjogou (INT)

Chave Estrangeira: VendedorID (INT), ClienteNIF (VARCHAR(75))

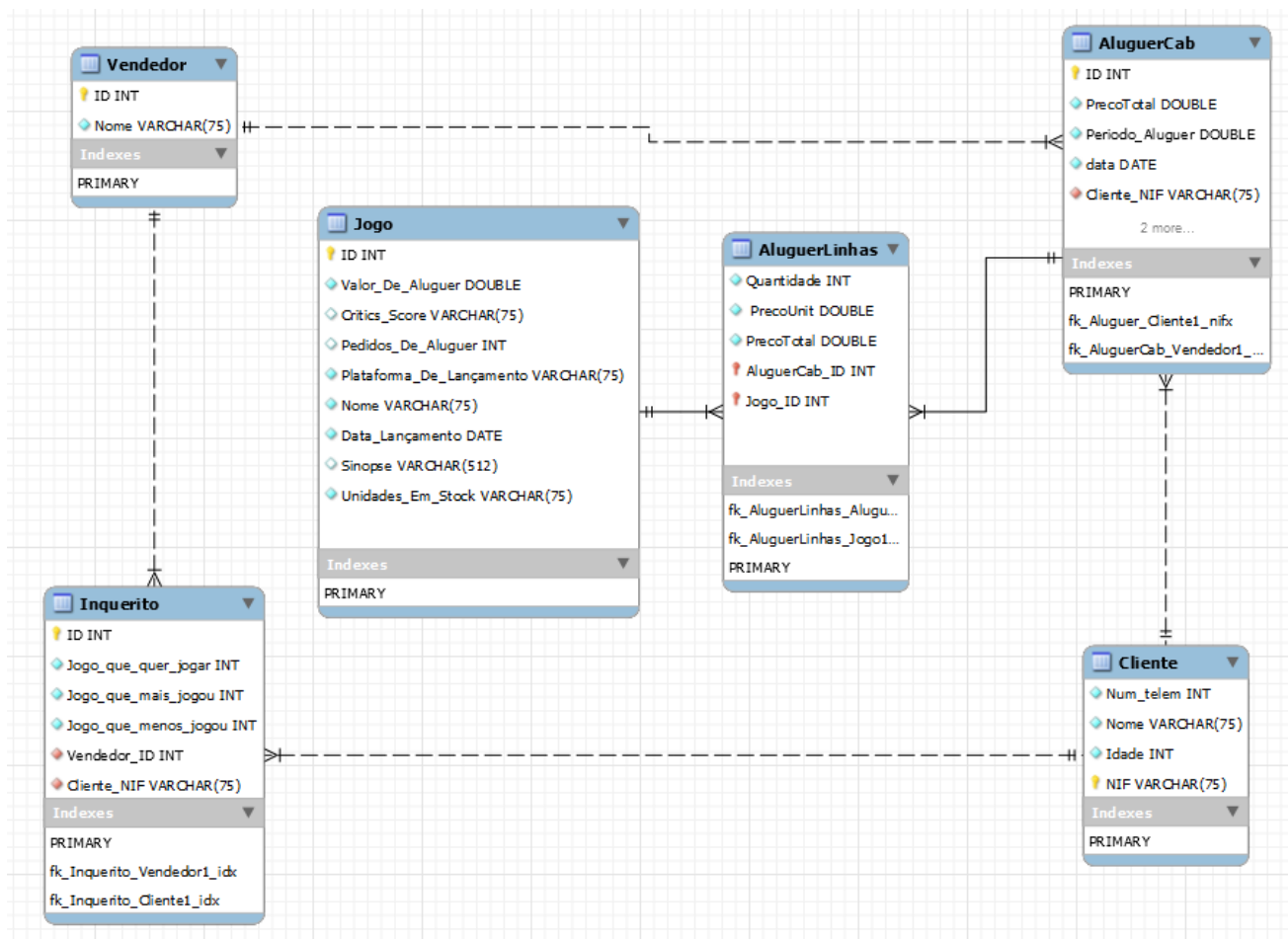


Figura 4.7: Modelo Lógico Obtido

4.1.2 Validação do Modelo de dados Lógico

Para avaliar a viabilidade do modelo lógico, foi determinado que seriam realizadas uma série de interrogações lógicas, com o intuito de investigar a capacidade de manipulação necessária conforme os requisitos estabelecidos previamente.

[RM1] – O utilizador deve ter a capacidade de selecionar a plataforma de lançamento desejada para consultar os jogos.

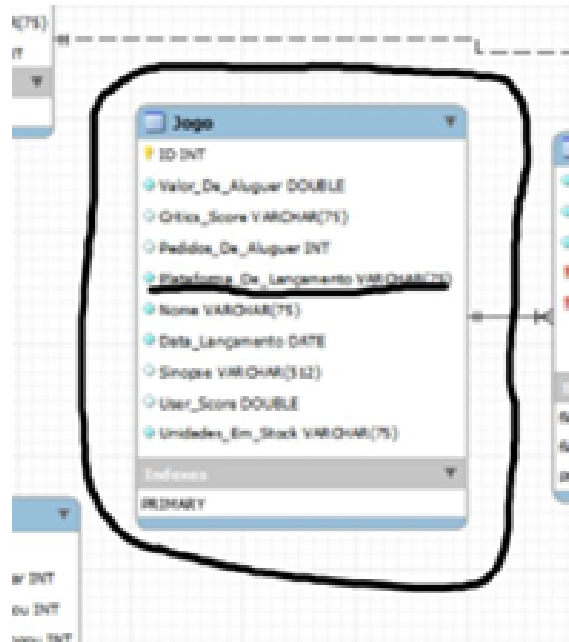


Figura 4.8: Validação de [RM1]

Na busca por acesso aos jogos de uma plataforma específica, é necessário consultar a tabela de jogos e filtrar os registros que correspondam à plataforma desejada. Essa operação permitirá visualizar os jogos disponíveis exclusivamente para essa plataforma em particular.

[RM3] – Deve ser implementada a funcionalidade que permita mostrar os jogos que foram mais e menos alugados pelos clientes.

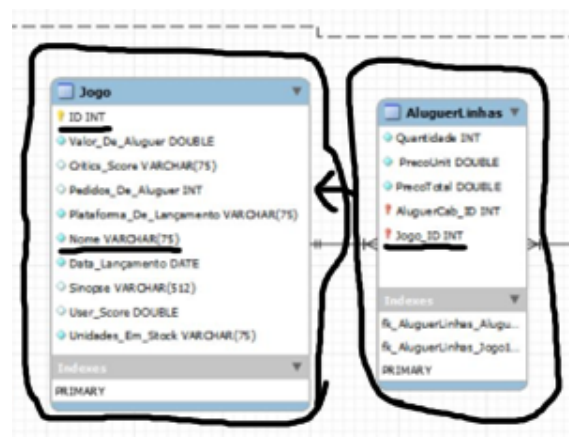


Figura 4.9: Validação de [RM3]

Para atender ao terceiro requisito, é necessário aceder a tabela "AluguerLinhas" a fim de obter informações sobre os jogos que foram alugados. Isso pode ser feito utilizando

o atributo "Jogo__ID", que serve como referência para a tabela "Jogo". Ao analisar o ID presente na tabela "Jogo", é possível obter o nome do jogo associado ao aluguer em questão.

Além disso, examinando todos os alugueres realizados, é possível determinar quais jogos foram alugados e quantas vezes cada um deles foi alugado, fornecendo assim um panorama completo das informações necessárias. Essa análise envolve a utilização da tabela "AluguerLinhas" em conjunto com a tabela "Jogo" para obter todos os dados relevantes.

[RM7] – Deve ser implementada a funcionalidade que permite visualizar os clientes que mais e menos faturaram na loja.

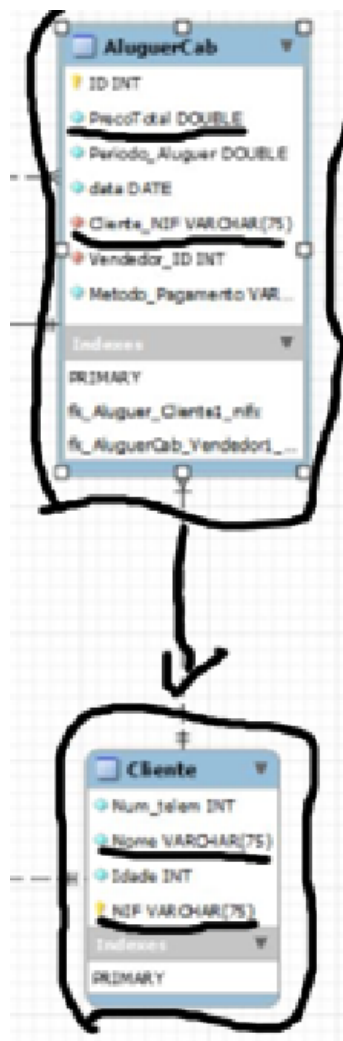


Figura 4.10: Validação de **[RM7]**

Para identificar os clientes que mais e menos faturaram, é necessário consultar a

tabela "AluguerCab". A partir dessa tabela, é possível verificar o ID do cliente associado a cada aluguer, assim como o valor total gasto no aluguer.

Para calcular o faturamento total de cada cliente, percorre-se a tabela "AluguerCab" e, para cada aluguer analisado, adiciona-se o valor gasto pelo cliente correspondente ao valor já calculado. Dessa forma, é possível obter o valor total gasto por cada cliente ao longo de todos os alugueres registados.

Essa lógica de percorrer a tabela "AluguerCab" e agregar os valores gastos por cada cliente, permite determinar tanto o cliente que mais faturou, identificado pelo maior valor total gasto, quanto o cliente que menos faturou, identificado pelo menor valor total gasto.

[RM9] – Deve ser implementada a funcionalidade que permite consultar quais os clientes que alugaram um mesmo jogo mais de uma vez.

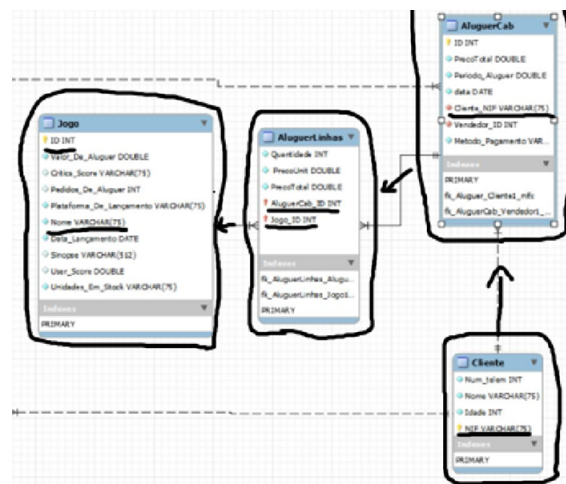


Figura 4.11: Validação de **[RM9]**

O requisito 9 envolve a obtenção de informações específicas, onde é necessário aceder a tabela "AluguerCab" utilizando o NIF (Número de Identificação Fiscal) do cliente como referência. Em seguida, é necessário aceder as linhas associadas a esse cliente e, a partir dessas linhas, obter informações sobre os jogos alugados.

O procedimento descrito acima deve ser repetido para todos os clientes, permitindo analisar a quantidade de vezes que um determinado jogo é registado nas faturas de um mesmo cliente. Dessa forma, é possível identificar os jogos que foram alugados mais de uma vez pelo mesmo cliente.

Essa mesma lógica foi aplicada aos demais requisitos de manipulação, garantindo a confiabilidade e consistência do modelo lógico adotado.

4.2 Normalização de Dados

Após analisar o modelo em questão, é possível concluir que todas as três formas normais (1FN, 2FN e 3FN) estão a ser cumpridas, tornando o modelo válido em termos de normalização.

1ª Forma Normal (1FN): Os atributos são atômicos, ou seja, não há valores repetidos e nem valores multivalorados dentro das tabelas. Isso significa que as tabelas estão devidamente estruturadas, *i.e.*, com cada atributo a representar uma única informação.

2ª Forma Normal (2FN): Os atributos normais dependem unicamente da chave primária da tabela. Essa condição só pode ser cumprida se a 1ª Forma Normal também for obedecida. Portanto, é garantido que os atributos estão devidamente relacionados à chave primária e não apresentam dependências funcionais parciais.

3ª Forma Normal (3FN): Não há atributos dependentes, ou seja, não existem atributos que possam ser gerados a partir de outros atributos. Essa condição só pode ser cumprida se as duas formas normais anteriores também estiverem a ser cumpridas. Portanto, é assegurado que não há redundância de dados e que as informações estão devidamente distribuídas em entidades relacionadas.

Com base na análise realizada, é possível afirmar que o modelo segue os princípios de normalização e está estruturado de forma a evitar redundância de dados, garantindo um bom desempenho e integridade das informações.

4.3 Conclusão

Em resumo, o modelo lógico mostrou-se eficaz e adequado para a produção da base de dados. Os testes lógicos e a normalização foram fundamentais para avaliar sua eficiência e identificar eventuais problemas. Além disso, essas etapas permitiram otimizar os modelos iniciais, revelando erros e áreas que poderiam ser aprimoradas, visando alcançar a melhor versão possível.

Agora, com o modelo na sua forma ideal, o próximo passo é a implementação física da base de dados. Nesta fase, iremos de facto transformar as ideias em algo real.

Capítulo 5

Implementação Física

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

Para facilitar a criação de um sistema de gestão de bases de dados, foram utilizadas ferramentas como o MySQL Workbench em conjunto com o MySQL.

O *Workbench* possui uma funcionalidade chamada *Forward Engineering*, que permite gerar o código necessário para criar o sistema com base no modelo lógico definido. Foi utilizada esta funcionalidade, por ser bastante intuitiva e simples. O código gerado via *forward engineering* foi colocado em um ficheiro .sql chamado *CreateDB.sql*. Este ficheiro é sempre o primeiro a ser executado, sendo todos os outros evidentemente dependentes deste.

No processo de povoamento, é importante também ter dados realistas para preencher a base de dados. O preenchimento pode ser feito de duas maneiras :

- Através de um ficheiro chamado *PopulateScript.sql*, que contém código SQL responsável por inserir registos nas tabelas (Jogo, Cliente, Vendedor, Inquérito, etc...).
- Através de um *script* em *python* que automatiza a inserção dos dados através da leitura de seis .csv distintos, que possuem os dados de cada uma das seis tabelas.

Esses registos são feitos de modo a se assemelharem a dados que poderiam existir na vida real.

Por exemplo, na tabela "cliente", são utilizados números de telefone que seguem um formato válido e que poderiam ser encontrados em situações reais. Todos os jogos possuem atributos cujos valores vão de acordo com os *standards* do mundo real.

Esse tipo de abordagem ajuda a tornar o sistema mais próximo da realidade.

5.2 Definição e caracterização das vistas de utilização em SQL e tradução das interrogações do utilizador para SQL

Para a definição das interrogações e caracterização das vistas de utilização em SQL, foram utilizadas *stored procedures* e *views* (vistas).

As *stored procedures* funcionam de forma semelhante a funções, ou seja, recebem entradas (*input*) e retornam um resultado (*output*).

Optamos por utilizá-las devido à sua simplicidade. Para qualquer tipo de consulta que necessite de valores fornecidos pelo utilizador, basta passar esses valores como argumentos para a *procedure* correspondente. Além disso, essas funções são definidas quando o script responsável pela criação destas (*Procedures.sql*) é carregado. Isso permite que as interrogações sejam executadas de forma mais rápida.


Além das *stored procedures*, também utilizamos as *views*, que foram criadas apenas para etiquetar cada consulta que não seja do tipo *procedure* (o que efetivamente foi crucial para o correto funcionamento do script de automatização do SGBD).

As *stored procedures* foram agrupadas no arquivo *Procedures.sql*, enquanto as vistas foram definidas no arquivo *Views.sql*. Essa separação foi feita principalmente para fins de *debug*, uma vez que encontramos alguns problemas ao executar *Procedures*, devido aos delimitadores que afetavam o parser dentro do script em *python*. Para fins de demonstração manual, utilizamos o arquivo *Queries.sql*, que contém todas as queries e que por sua vez, não necessitaria de manter as vistas como sendo vistas, já que não estaríamos a tentar automatizar o processo.

5.2.1 Exemplos de Vistas

:

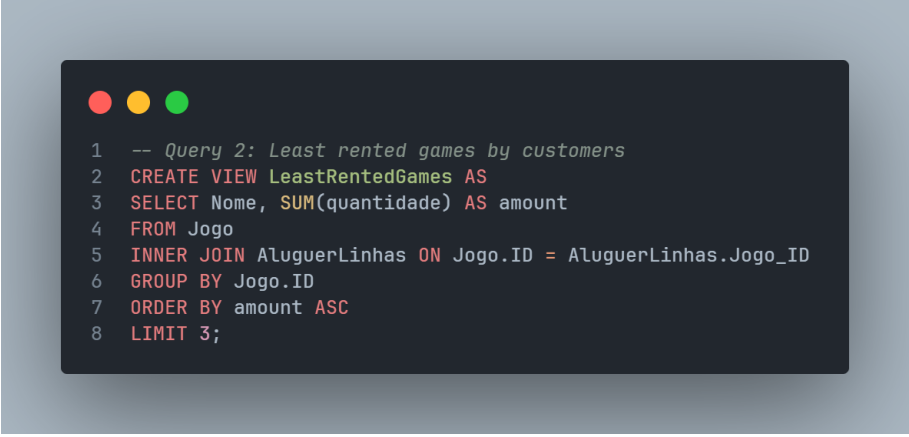
- Consulta da Lista de Plataformas com mais Alugueres



```
1  -- Query 3: Most rented platforms by customers
2  CREATE VIEW MostRentedPlatforms AS
3  SELECT plataforma_de_lançamento, SUM(quantidade) AS amount
4  FROM Jogo
5  INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
6  GROUP BY jogo.plataforma_de_lançamento
7  ORDER BY amount DESC
8  LIMIT 3;
9
```

Figura 5.1: Código da respetiva Query

- Consulta da Lista de jogos com menos Alugueres



```
1  -- Query 2: Least rented games by customers
2  CREATE VIEW LeastRentedGames AS
3  SELECT Nome, SUM(quantidade) AS amount
4  FROM Jogo
5  INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
6  GROUP BY Jogo.ID
7  ORDER BY amount ASC
8  LIMIT 3;
```

Figura 5.2: Código da respetiva Query

5.2.2 Exemplos de Interrogações

:

- Consulta de Jogos por plataforma.

```
1  -- Query 18: Get games based on the desired launch platform
2  CREATE PROCEDURE GetGamesByPlatform(IN platform VARCHAR(50))
3  SELECT *
4  FROM jogo
5  WHERE Plataforma_De_Lançamento = platform;
6
```

Figura 5.3: Código da respetiva Query

- Inserção de um novo cliente na Base de Dados.

```
1  -- Query 22: Insert a new client into Cliente table
2  CREATE PROCEDURE AddCliente(
3      IN p_Num_telem INT,
4      IN p_Nome VARCHAR(75),
5      IN p_Idade INT,
6      IN p_NIF INT
7  )
8  INSERT INTO `GameBlockbuster`.`Cliente` (
9      `Num_telem`,
10     `Nome`,
11     `Idade`,
12     `NIF`
13 ) VALUES (
14     p_Num_telem,
15     p_Nome,
16     p_Idade,
17     p_NIF
18 );
19
```

Figura 5.4: Código da respetiva Query

5.3 Cálculo do espaço da bases de dados (inicial e taxa de crescimento anual)

5.3.1 Tamanho Inicial da Base de Dados

No que toca ao tamanho inicial da base de dados, o software *MySQLWorkbench* revela que esta possui 192.0 *KiB*. Tal tamanho, deve-se à quantidade reduzida de entradas nas tabelas introduzidas pelo script de povoamento (cerca de 16 entradas por tabela), assim como ao tipo de dados que são utilizados em cada entidade.

5.3.2 Requisitos de Armazenamento

Com base no *MySQL 8.0 Reference Manual* e nos tipos de dados que a nossa base de dados utiliza, obtemos a seguinte lista de requisitos para guardar tipos de dados:

Data Type	Storage Required
INT	4 bytes
DOUBLE	8 bytes
DATE	3 bytes
VARCHAR(M)	$L + 1$ bytes if values require 0 – 255 bytes, $L + 2$ bytes if more

Tabela 5.1: Requisitos de armazenamento

Com base nestes requisitos, podemos calcular o tamanho máximo que cada entrada em cada uma das nossas tabelas ocupará.

Assim:

- Vendedor

Atributos	Tipo de Dados	Tamanho (bytes)
ID	INT	4
Nome	VARCHAR(75)	$75 \times 2 + 1 = 151$
Tamanho Total		155

Tabela 5.2: Tamanho de cada atributo da tabela Vendedor, e o respetivo tamanho total

- Inquerito

Atributos	Tipo de Dados	Tamanho (bytes)
ID	INT	4
Jogo__que__quer__jogar	INT	4
Jogo__que__mais__jogou	INT	4
Jogo__que__menos__jogou	INT	4
Vendedor__ID	INT	4
Cliente__NIF	VARCHAR(75)	151
Tamanho Total		171

Tabela 5.3: Tamanho de cada atributo da tabela Inquerito, e o respetivo tamanho total

- AluguerCab

Atributos	Tipo de Dados	Tamanho (bytes)
ID	INT	4
PrecoTotal	DOUBLE	8
Periodo__Aluguer data	DOUBLE DATE	8 3
Vendedor__ID	INT	4
Cliente__NIF	VARCHAR(75)	151
Metodo__Pagamento	VARCHAR(75)	151
Tamanho Total		329

Tabela 5.4: Tamanho de cada atributo da tabela AluguerCab, e o respetivo tamanho total

- AluguerLinhas

Atributos	Tipo de Dados	Tamanho (bytes)
Quantidade	INT	4
PrecoUnit	DOUBLE	8
PrecoTotal	DOUBLE	8
AluguerCab__ID	INT	4
Jogo__ID	INT	4
Tamanho Total		28

Tabela 5.5: Tamanho de cada atributo da tabela AluguerLinhas, e o respetivo tamanho total

- Cliente

Atributos	Tipo de Dados	Tamanho (bytes)
Num__telem	INT	4
Nome	VARCHAR(75)	151
Idade	INT	4
NIF	VARCHAR(75)	151
Tamanho Total		310

Tabela 5.6: Tamanho de cada atributo da tabela Cliente, e o respetivo tamanho total

▪ Jogo

Atributos	Tipo de Dados	Tamanho (bytes)
ID	INT	4
Valor__De__Aluguer	DOUBLE	8
Critics__Score	VARCHAR(75)	151
Pedidos__De__Aluguer	INT	4
Plataforma__De__Lançamento	VARCHAR(75)	151
Nome	VARCHAR(75)	151
Data__Lançamento	DATE	3
Sinopse	VARCHAR(512)	$2 \times 512 + 2 = 1026$
Unidades__Em__Stock	VARCHAR(75)	151
Tamanho Total		1649

Tabela 5.7: Tamanho de cada atributo da tabela Jogo, e o respetivo tamanho total

5.3.3 Taxa de Crescimento e Tamanho da Base De Dados Futura

Consideremos que a loja tenha, por ano, a sua biblioteca de jogos (i.e., o número de jogos diferentes) aumentada em 120%.

Sabendo que a loja atualmente possui 16 jogos diferentes, podemos concluir que no ano seguinte, a loja teria cerca de $16 + (1.2 \times 16) \approx 35$ jogos. Esta adição de jogos, obrigaria a uma inserção na tabela Jogo de 19 registos.

Cada jogo ocupará, no máximo, 1649 bytes. Portanto, ao inserir 19 jogos na tabela, o tamanho da base de dados aumentaria para $(196,608 + 19 \times 1649)$ bytes, que é aproximadamente 222.6 KiB.

Podemos representar a expansão do negócio e o aumento das necessidades de vendedores e faturas da seguinte forma:

Seja x o número de novos vendedores necessários. Cada vendedor ocupará 155 bytes

na base de dados. Portanto, o tamanho adicional necessário para os vendedores será $155 \times x$ bytes.

Seja y o número de linhas associadas ao aluguer em uma fatura. Cada cabeçalho do aluguer ocupará 329 bytes, e cada linha associada ocupará 28 bytes. Portanto, o tamanho adicional necessário para as faturas será $(329 + 28 \times y)$ bytes.

Além disso, a chegada de α novos clientes exigirá adicionar $(\alpha \times 310)$ bytes à base de dados.

Dessa forma, podemos resumir as necessidades de expansão do negócio em termos matemáticos da seguinte maneira:

Tamanho adicional necessário para os vendedores: $155x$ bytes. Tamanho adicional necessário para as faturas: $(329 + 28y)$ bytes. Tamanho adicional necessário para os novos clientes: $(\alpha \times 310)$ bytes.

Estes cálculos não valorizam a tabela inquérito, por ser algo completamente imprevisível.

Tomemos como valores x , y , β e α o número de novos vendedores necessários, o número de linhas associadas ao aluguer em uma fatura, o número de faturas e o número de novos clientes, respectivamente.

Ao definirmos $x = 5$, $y = 2$, $\beta = 100$ e $\alpha = 120$, temos:

Tamanho da Base de Dados Após Um Ano: $\text{Tamanho Atual} + (155 \cdot 5 + (329 \cdot 100 + 28 \cdot 2) + 120 \cdot 310) + 19 \cdot 1649$

Tamanho Final: $196608 + 102262 = 298870$ bytes ≈ 292 KiB, que iria corresponder a um crescimento de cerca de 52% (com a ausência da tabela inquéritos).

Capítulo 6

Introdução de Dados no Sistema de Bases de Dados e Interface de utilização

6.1 Apresentação e modelo do sistema

Para a recolha de dados para o povoamento da base de dados foi feita a conversão de um ficheiro excel (.xlsx) que o dono da loja tinha para um script em SQL (caso se pretenda povoar a base de dados se forma "manual") e para seis ficheiros .csv (caso seja desejado automatizar o povoamento da BD), aonde são realizado os inserts para as diversas tabelas com a informação dos jogos para alugar, os vendedores que trabalham na loja, os clientes que a frequentam e dados de faturas.

As tabelas em excel utilizadas foram as seguintes:

ID	Valor_De_Alugar	Critics_Score	Pedidos_De_Alugar	plataforma_De_Lançamento	Nome	Data_Lançamento	Sinopse	Unidades_Em_Stock
1	4.99	90	50	PlayStation 4	The Last of Us Part II	19/06/2020	Na sequência do primeiro jogo, Ellie e Joel continuam a lutar pela sobrevivência num mundo pós-apocalíptico infestado por zombies.	20
2	3.49	88	30	Xbox One	Gears 5	10/09/2019	No papel de Kait Diaz, embarca numa jornada épica para descobrir a origem dos inimigos e revela a verdadeira ameaça para Sera.	15
3	2.99	92	45	Nintendo Switch	The Legend of Zelda: Breath of the Wild	03/03/2017	Explora um vasto mundo aberto cheio de mistérios e aventuras enquanto tentas salvar a Princesa Zelda e derrotar Ganon.	10
4	5.99	87	40	PlayStation 5	Marvel's Spider-Man: Miles Morales	12/11/2020	Assume o papel de Miles Morales e protege a cidade de Nova Iorque como o novo Spider-Man.	5
5	1.99	85	20	PC	The Witcher 3: Wild Hunt	19/05/2015	Entra no mundo de Geralt of Rivia e embarca numa aventura épica de caça a monstros e descoberta de segredos.	25
6	4.49	89	35	Xbox Series X	Halo Infinite	08/12/2021	Retorna ao universo de Halo e enfrenta um novo desafio como o Master Chief contra os forças do Covenant.	12
7	3.99	93	55	PlayStation 5	Ratchet & Clank: Rift Apart	11/06/2021	Viaja por diferentes dimensões com Ratchet e Clank, explorando mundos incríveis e enfrentando inimigos perigosos.	18
8	2.49	86	25	Nintendo Switch	Super Mario Odyssey	27/10/2017	Acompanha Mario numa aventura por vários reinos para salvar a Princesa Peach das garras do maligno Bowser.	7
9	4.99	90	40	PlayStation 4	God of War	20/04/2018	Segue a jornada de Kratos, o deus da guerra, num mundo nórdico brutal enquanto enfrenta criaturas mitológicas e descobre segredos familiares.	10
10	3.29	88	30	Xbox One	Red Dead Redemption 2	26/10/2018	Vive a vida de um fora da lei no Velho Oeste e embarca numa história épica de caça e recompensas e sobrevivência.	20
11	2.99	91	35	PC	Cyberpunk 2077	10/12/2020	Explora a cidade futurista de Night City neste RPG de ação, onde escolhas e consequências moldam o teu destino.	15
12	4.99	87	30	Nintendo Switch	Animal Crossing: New Horizons	20/03/2020	Cria a tua própria ilha paradisíaca, interage com os teus vizinhos animais e desfruta da tranquilidade desta experiência única.	8
13	1.99	84	15	PlayStation 4	Uncharted 4: A Thief's End	10/05/2016	Acompanha Nathan Drake na sua última aventura, repleta de tesouros perdidos, tirotes emocionantes e segredos sombrios.	5
14	5.99	93	50	Xbox Series X	Fable	31/10/2022	Volta a Albion nesta reimaginação do clássico jogo de RPG, onde as tuas escolhas moldam o mundo e a tua jornada heróica.	10
15	2.99	89	25	PC	Fallout 4	10/11/2015	Explora a devastada Wasteland de Boston neste RPG de ação pós-apocalíptico, onde cada decisão tem consequências duradouras.	10
16	6.99	70	1	Nintendo GameCube	Sonic Heroes	30/12/2003	Rail Canyon Sucks	100

Figura 6.1: Tabela de Jogos

ID	Nome
1	João Silva
2	Ana Santos
3	Pedro Costa
4	Sara Ferreira
5	Miguel Oliveira
6	Mariana Rodrigues
7	Tiago Pereira
8	Carolina Almeida
9	Ricardo Santos
10	Inês Costa
11	André Fernandes
12	Marta Sousa
13	António Silva
14	Beatriz Cardoso
15	Hugo Ribeiro
16	Anti-Inquéritos

Figura 6.2: Tabela de **Vendedores**

Num_telem	Nome	Idade	NIF
912345678	Marta Costa	28	123456789
933456789	João Silva	33	123789456
966789012	Mariana Ferreira	30	321456789
911345678	Beatriz Cardoso	23	456321987
966666666	Ana Rodrigues	22	456789123
911234567	André Santos	29	456987123
934567890	Hugo Martins	32	654123987
922222222	Sofia Pereira	27	654321987
934678901	Cláudia Silva	31	654987123
911111111	Pedro Almeida	31	789123456
922345678	Inês Costa	24	789456321
922456789	Diogo Gonçalves	27	789654321
933214567	Rui Oliveira	36	987123456
966123456	Carolina Sousa	26	987321654
933333333	Ricardo Santos	35	987654321

Figura 6.3: Tabela de **Clientes**

ID	Jogo que quer jogar	Jogo que mais jogou	Jogo que menos jogou	Vendedor ID	Cliente NIF
1	2	5	3	1	123456789
2	1	4	2	3	987654321
3	3	2	1	2	456789123
4	5	1	4	4	789123456
5	4	3	5	5	654321987
6	7	6	8	6	789123456
7	10	12	9	7	654321987
8	12	10	11	8	123789456
9	9	13	14	9	987321654
10	14	8	15	10	456987123
11	11	15	13	11	789456321
12	6	11	12	12	321456789
13	3	14	10	13	456321987
14	5	1	7	14	789654321
15	8	3	6	15	654987123
16	1	5	3	4	789123456

Figura 6.4: Tabela de **Inqueritos**

ID	PrecoTotal	Periodo_Aluguer	data	Cliente_NIF	Vendedor_ID	Metodo_Pagamento
1	14.99	3	5/1/2023	123456789	1	Cartão de crédito
2	10.49	2	5/2/2023	987654321	2	Dinheiro
3	8.99	2	5/3/2023	456789123	3	Cartão de débito
4	19.99	5	5/4/2023	789123456	4	Transferência bancária
5	6.99	1	5/5/2023	654321987	5	Cartão de crédito
6	12.99	3	5/6/2023	789123456	6	Dinheiro
7	9.99	2	5/7/2023	654321987	7	Cartão de débito
8	16.99	4	5/8/2023	123789456	8	Transferência bancária
9	13.99	3	5/9/2023	987321654	9	Cartão de crédito
10	8.49	2	5/10/2023	456987123	10	Dinheiro
11	11.99	3	5/11/2023	789456321	11	Cartão de débito
12	18.99	4	5/12/2023	321456789	12	Transferência bancária
13	9.99	2	5/13/2023	456321987	13	Cartão de crédito
14	14.99	3	5/14/2023	789654321	14	Dinheiro
15	6.58	1	5/15/2023	987321654	15	Transferência bancária
16	12.99	3	7/3/2023	987321654	6	Transferência bancária

Figura 6.5: Tabela de **Aluguer Cabeçalho**

Quantidade ▼	PrecoUnit ▼	PrecoTotal ▼	AluguerCab_ID ▼	Jogo_ID ▼
2	4.99	9.98	1	1
1	3.49	3.49	1	2
3	2.99	8.97	2	3
2	5.99	11.98	3	4
1	1.99	1.99	4	5
3	4.49	13.47	5	6
2	3.99	7.98	6	7
1	2.49	2.49	7	8
3	4.99	14.97	8	9
2	3.29	6.58	9	10
1	2.99	2.99	10	11
3	4.99	14.97	11	12
2	1.99	3.98	12	13
1	5.99	5.99	13	14
3	2.99	8.97	14	15
2	3.29	6.58	15	10

Figura 6.6: Tabela de **Aluguer Linhas**

6.2 Implementação do povoamento

Para o povoamento, como anteriormente referido, são disponibilizadas duas diferentes maneiras de povoar a BD:

- Através de um script de povoamento .sql
- Através de um script realizado na linguagem *python*.

Aqui temos alguns *samples* com o código do script de povoamento .sql :

```
-- Populate table 'Cliente'
INSERT INTO `GameBlockbuster`.`Cliente` (`Num_telef`, `Nome`, `Idade`, `NIF`) VALUES
(912345678, 'Marta Costa', 28, '123456789'),
(933333333, 'Ricardo Santos', 35, '987654321'),
(966666666, 'Ana Rodrigues', 22, '456789123'),
```

Figura 6.7: Povoamento de **Clientes**: Exemplo de inserção de dados na tabela Cliente.

```
-- Populate table 'Vendedor'
INSERT INTO `GameBlockbuster`.`Vendedor` (`ID`, `Nome`, `Inquerito_ID`) VALUES
(1, 'João Silva', NULL),
(2, 'Ana Santos', NULL),
(3, 'Pedro Costa', NULL),
```

Figura 6.8: Povoamento de **Vendedores**: Exemplo de inserção de dados na tabela Vendedor.

```
-- Populate table 'Jogo'
INSERT INTO `GameBlockbuster`.`Jogo` (`ID`, `Valor_De_Aluguer`, `Critics_Score`, `Pedidos_De_Aluguer`, `Plataforma_De_Lançamento`, `Nome`, `Data_Lançamento`, `Sinopse`, `Unidades_Fe_Stock`) VALUES
(1, 4.99, '90', 50, 'PlayStation 4', 'The Last of Us Part II', '2020-06-19', 'Na sequência do primeiro jogo, Ellie e Joel continuam a lutar pela sobrevivência num mundo pós-apocalíptico infestado por zombies.', '20'),
(2, 3.49, '88', 30, 'Xbox One', 'Gears 5', '2019-09-10', 'No papel de Kait Diaz, embarca numa jornada épica para descobrir a origem dos inimigos e revela a verdadeira ameaça para Sera.', '15'),
(3, 2.99, '92', 45, 'Nintendo Switch', 'The Legend of Zelda: Breath of the Wild', '2017-03-03', 'Explora um vasto mundo aberto cheio de mistérios e aventuras enquanto tentas salvar a Princesa Zelda e derrotar Ganon.', '10'),
```

Figura 6.9: Povoamento de **Jogos**: Exemplo de inserção de dados na tabela Jogo.

```
-- Populate table 'Inquerito'
INSERT INTO `GameBlockbuster`.`Inquerito` (`ID`, `Jogo_que_quer_jogar`, `Jogo_que_mais_jogou`, `Jogo_que_menos_jogou`, `Vendedor_ID`, `Cliente_NIF`) VALUES
(1, 2, 5, 3, 1, '123456789'),
(2, 1, 4, 2, 3, '987654321'),
(3, 3, 2, 1, 2, '456789123'),
```

Figura 6.10: Povoamento de **inquéritos**: Exemplo de inserção de dados na tabela Inquerito.

```
-- Populate table 'AluguerCab'
INSERT INTO `GameBlockbuster`.`AluguerCab` (ID, PrecoTotal, Período_Aluguer, data, Cliente_NIF, Vendedor_ID, Metodo_Pagamento) VALUES
(1, 14.99, 3, '2023-05-01', '123456789', 1, 'Cartão de crédito'),
(2, 10.49, 2, '2023-05-02', '987654321', 2, 'Dinheiro'),
(3, 8.99, 2, '2023-05-03', '456789123', 3, 'Cartão de débito'),
```

Figura 6.11: Povoamento de **Cabeçalho do aluguer**: Exemplo de inserção de dados na tabela AluguerCab.

```
-- Populate table 'AluguerLinhas'
INSERT INTO `GameBlockbuster`.`AluguerLinhas` (`Quantidade`, `PrecoUnit`, `PrecoTotal`, `AluguerCab_ID`, `Jogo_ID`) VALUES
(2, 4.99, 9.98, 1, 1),
(1, 3.49, 3.49, 1, 2),
(3, 2.99, 8.97, 2, 3),
```

Figura 6.12: Povoamento de **Linhas do aluguer**: Exemplo de inserção de dados na tabela AluguerLinha.

Foram adicionadas por volta de 16 entradas em cada tabela, de modo a ser possível ter um grupo de amostras de teste, para garantir o correto funcionamento do SGBD.

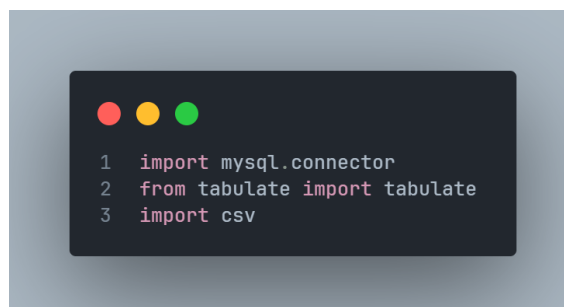
A outra metodologia adotada em relação ao povoamento da Base de Dados será abordada no tópico seguinte.

6.3 Criação do Script em Python e a sua Implementação

No âmbito de automatizar a criação da Base de Dados, o seu povoamento e a integração de *queries*, foi utilizado um pequeno script realizado na linguagem de programação *Python*, com o auxílio de quatro bibliotecas.

6.3.1 Importação de Bibliotecas

- `mysql.connector`: Biblioteca que fornece uma interface Python para conexão com o servidor MySQL.
- `tabulate`: Biblioteca utilizada para formatar as respostas das consultas em formato de tabela.
- `csv`: Biblioteca utilizada para manipulação de arquivos CSV, necessária para o povoamento da base de dados.



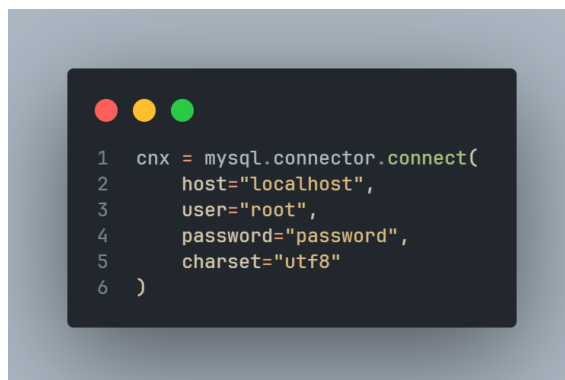
```
1 import mysql.connector
2 from tabulate import tabulate
3 import csv
```

Figura 6.13: Importação de Bibliotecas

6.3.2 Conexão com o Servidor MySQL

O script estabelece uma conexão com o servidor MySQL usando as seguintes credenciais:

- **Host:** endereço do servidor MySQL.
- **Utilizador:** nome de utilizador para autenticação no servidor MySQL.
- **Senha:** senha para autenticação no servidor MySQL.



```
1 cnx = mysql.connector.connect(  
2     host="localhost",  
3     user="root",  
4     password="password",  
5     charset="utf8"  
6 )
```

Figura 6.14: Conexão com o Servidor MySQL

6.3.3 Função para Executar Consultas

A função `execute_query(query)` é responsável por executar uma consulta SQL no servidor MySQL. Ela recebe como parâmetro a consulta a ser executada e utiliza o cursor para realizar a operação. Os resultados da consulta são recuperados usando o método `fetchall()`, e caso haja resultados, são formatados sob a forma de uma tabela usando a função `tabulate()`. A função também trata o caso de não haver resultados encontrados.



```
1 def execute_query(query):  
2     cursor.execute(query)  
3     results = cursor.fetchall()  
4     if len(results) > 0:  
5         headers = [desc[0] for desc in cursor.description]  
6         print(tabulate(results, headers, tablefmt="fancy_grid"))  
7     else:  
8         print("No results found.")
```

Figura 6.15: Função para Executar Consultas

6.3.4 Função para Executar Scripts SQL

A função `executeQuery(file_path)` é responsável por executar scripts SQL armazenados em arquivos. Ela recebe como parâmetro o caminho do arquivo SQL a ser executado. O conteúdo do arquivo é lido e as instruções SQL são separadas usando o caractere ";" como delimitador. Cada instrução é executada individualmente, ignorando instruções em branco. Após a execução das instruções, as alterações são confirmadas no servidor MySQL.



```
1 def executeQuery(file_path):
2     # Set the encoding when opening the SQL file
3     with open(file_path, 'r', encoding='utf8') as sql_file:
4         sql_script = sql_file.read()
5         statements = sql_script.split(';')
6         for statement in statements:
7             if statement.strip():
8                 cursor.execute(statement)
9                 cnx.commit()
10            print(f"Script {file_path} executed successfully.")
```

Figura 6.16: Função para Executar Scripts SQL

6.3.5 Funções para Manipulação de Dados CSV

As funções `insert_data_to_csv(csv_file, table_name, column_names, new_entries)` e `insert_data_from_csv(csv_file, table_name, column_names)` são responsáveis por inserir dados em tabelas a partir de arquivos CSV.

A função `insert_data_to_csv` abre o arquivo CSV em modo de escrita, cria um escritor CSV e insere as novas entradas no arquivo.

A função `insert_data_from_csv` abre o arquivo CSV em modo de leitura, cria um leitor CSV e itera sobre as linhas do arquivo. Para cada linha, os valores das colunas especificadas são extraídos e uma *query* SQL de inserção é executada no servidor MySQL.


```

1 def insert_data_to_csv(csv_file, table_name, column_names, new_entries):
2     with open(csv_file, 'a', newline='') as file:
3         writer = csv.writer(file, delimiter=";")
4         for entry in new_entries:
5             writer.writerow(entry)
6     cnx.commit()
7
8
9 def insert_data_from_csv(csv_file, table_name, column_names):
10    with open(csv_file, newline='') as file:
11        reader = csv.DictReader(file, delimiter=";")
12
13        for row in reader:
14            values = [row[column] for column in column_names]
15            placeholders = ', '.join(['%s'] * len(column_names))
16            sql = f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({placeholders})"
17            cursor.execute(sql, values)
18    cnx.commit()

```

Figura 6.17: Funções para Manipulação de Dados CSV

6.3.6 Limpeza da Base de Dados

A função `cleanDB()` é responsável por limpar a base de dados, dando DROP na base de dados GameBlockbuster, caso exista. Ela executa uma *query* SQL para remover a base de dados e confirma as alterações no servidor MySQL.

```

1 def cleanDB():
2     cursor.execute("DROP DATABASE IF EXISTS GameBlockbuster")

```

Figura 6.18: Limpeza da Base de Dados

6.3.7 Função `execute_procedure`

A função `execute_procedure` é responsável por chamar uma determinada *procedure* armazenada na base de dados, utilizando o módulo `cursor`. Ela aceita dois parâmetros: `procedure_name` e `params` (opcional).

Se o parâmetro `params` for fornecido, a função chamará o procedimento armazenado `procedure_name` passando os parâmetros. Caso contrário, ela chamará o procedimento sem nenhum parâmetro.

Em seguida, a função itera sobre os resultados armazenados no cursor através do método `stored_results()`. Se houver resultados, eles são recuperados utilizando `result.fetchall()`. Os resultados são então exibidos na forma de uma tabela utilizando a função `tabulate` do módulo `tabulate`, com os cabeçalhos das colunas sendo obtidos a partir de `result.description`. Se nenhum resultado for encontrado, a mensagem "No results found." é exibida.

Após a exibição dos resultados, a função realiza um `commit` na conexão com a base de dados através de `cnx.commit()` para confirmar as alterações. Finalmente, a mensagem "Procedure executed successfully." é exibida para indicar que o procedimento foi executado com sucesso.



```
1 def execute_procedure(procedure_name, params=None):
2     if params:
3         cursor.callproc(procedure_name, params)
4     else:
5         cursor.callproc(procedure_name)
6
7     for result in cursor.stored_results():
8         results = result.fetchall()
9         if len(results) > 0:
10             headers = [desc[0] for desc in result.description]
11             print(tabulate(results, headers, tablefmt="fancy_grid"))
12         else:
13             print("No results found.")
14
15     cnx.commit()
16     print("Procedure executed successfully.")
```

Figura 6.19: Função `execute_procedure`

6.3.8 Função `populate_database_from_csv`

A função `populate_database_from_csv` é responsável por povoar a base de dados a partir de arquivos CSV. Ela segue as seguintes etapas:

1. Executa o script "CreateDB.sql" utilizando a função `executeQuery()` para criar a

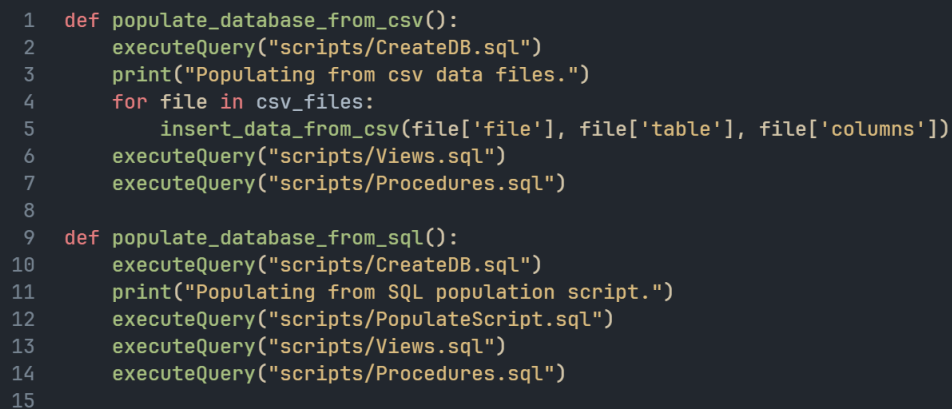
base de dados.

2. Exibe a mensagem "Populating from csv data files." para indicar o início do povoamento.
3. Itera sobre cada arquivo CSV especificado em `csv_files`.
4. Para cada ficheiro, chama a função `insert_data_from_csv()` passando o caminho do ficheiro, a tabela correspondente e as colunas a serem inseridas.
5. Executa o script "Views.sql" utilizando a função `executeQuery()` para criar as *views* na base de dados.
6. Executa o script "Procedures.sql" utilizando a função `executeQuery()` para criar as *procedures* armazenadas na base de dados.

6.3.9 Função `populate_database_from_sql`

A função `populate_database_from_sql` é responsável por povoar o banco de dados a partir de um script SQL de povoamento. Ela segue as seguintes etapas:

1. Executa o script "CreateDB.sql" utilizando a função `executeQuery()` para criar a base de dados.
2. Exibe a mensagem "Populating from SQL population script." para indicar o início do povoamento.
3. Executa o script "PopulateScript.sql" utilizando a função `executeQuery()` para realizar o povoamento a partir do script SQL.
4. Executa o script "Views.sql" utilizando a função `executeQuery()` para criar as *views* na base de dados.
5. Executa o script "Procedures.sql" utilizando a função `executeQuery()` para criar as *procedures* armazenadas na base de dados.



```

1  def populate_database_from_csv():
2      executeQuery("scripts/CreateDB.sql")
3      print("Populating from csv data files.")
4      for file in csv_files:
5          insert_data_from_csv(file['file'], file['table'], file['columns'])
6      executeQuery("scripts/Views.sql")
7      executeQuery("scripts/Procedures.sql")
8
9  def populate_database_from_sql():
10     executeQuery("scripts/CreateDB.sql")
11     print("Populating from SQL population script.")
12     executeQuery("scripts/PopulateScript.sql")
13     executeQuery("scripts/Views.sql")
14     executeQuery("scripts/Procedures.sql")
15

```

Figura 6.20: Funções populate database from csv/populate database from sql

6.3.10 Menu Interativo

Finalmente o *script* possui um menu interativo, onde são descritas as diversas queries existentes.

O menu possui um loop onde é exibido, utilizando a variável `menu` e solicita ao utilizador que faça uma escolha. Com base na escolha deste, ele executa a *query* correspondente, chamando a função `execute__query()`.

As opções do menu de "1" a "22" são tratadas individualmente e cada uma chama uma função ou executa uma consulta específica. A opção "0" interrompe o loop e finaliza o programa. A opção "99" chama a função `cleanDB()` para limpar a base de dados antes de sair.

Este script também realiza *backup* dos dados, i.e, sempre que é dado um DROP na base de dados, os dados existentes são guardados num `.csv`, incluindo os possíveis dados resultantes de *queries* de inserção de novos dados nas tabelas.

```

1 while True:
2     print(menu)
3     choice = input("Enter your choice: ")
4
5     if choice == "1":
6         execute_query("SELECT * FROM MostRentedGames")
7     elif choice == "2":
8         execute_query("SELECT * FROM LeastRentedGames")
9     elif choice == "3":
10        execute_query("SELECT * FROM MostRentedPlatforms")
11    elif choice == "4":
12        execute_query("SELECT * FROM ReleaseDatesWithMostRentals")
13    elif choice == "5":
14        execute_query("SELECT * FROM TopSpendingCustomers")
15    elif choice == "6":
16        execute_query("SELECT * FROM LowSpendingCustomers")
17    elif choice == "7":
18        execute_query("SELECT * FROM CustomersWithMultipleGameRentals")
19    elif choice == "8":
20        execute_query("SELECT * FROM MostLikedGames")
21    elif choice == "9":
22        execute_query("SELECT * FROM LeastLikedGames")
23    elif choice == "10":
24        execute_query("SELECT * FROM DesiredGames")
25    elif choice == "11":
26        execute_query("SELECT * FROM TopRentingSellers")
27    elif choice == "12":
28        execute_query("SELECT * FROM LowRentingSellers")
29    elif choice == "13":
30        execute_query("SELECT * FROM TopSurveyIssuingSellers")
31    elif choice == "14":
32        execute_query("SELECT * FROM SellersWithoutSurveys")
33    elif choice == "15":
34        execute_query("SELECT * FROM FullJogo")
35    elif choice == "16":
36        execute_query("SELECT * FROM FullInquerito")
37    elif choice == "17":
38        execute_query("SELECT * FROM FullCliente")
39    elif choice == "18":
40        platform = input("Enter the desired launch platform: ")
41        execute_procedure("GetGamesByPlatform", (platform,))
42    elif choice == "19":
43        execute_procedure("SortGamesByCriteria")
44    elif choice == "20":
45        execute_procedure("SortGamesByRentalCount")
46    elif choice == "21":
47        game_id = input("Enter the ID of the game: ")
48        rental_value = float(input("Enter the rental value of the game: "))
49        critics_score = input("Enter the critic's score of the game: ")
50        rental_requests = input(
51            "Enter the number of rental requests for the game: ")
52        launch_platform = input("Enter the launch platform of the game: ")
53        game_name = input("Enter the name of the game: ")
54        release_date = input(
55            "Enter the release date of the game (YYYY-MM-DD): ")
56        synopsis = input("Enter the synopsis of the game: ")
57        stock_units = input("Enter the number of stock units for the game: ")
58        params = (game_id, rental_value, critics_score, rental_requests,
59                  launch_platform, game_name, release_date, synopsis, stock_units)
60        execute_procedure("AddJogo", params)
61        print("Game inserted successfully.")
62        new_entry = [str(val) for val in params]
63        insert_data_to_csv('data/jogo.csv', 'Jogo', ['ID', 'Valor_De_Aluguer', 'Critics_Score', 'Pedidos_De_Aluguer', 'Plataforma_De_Lançamento', 'Nome', 'Data_Lançamento', 'Sinopse', 'Unidades_Em_Stock'], [new_entry])
64    elif choice == "22":
65        client_phone = input("Enter the client's phone number: ")
66        client_name = input("Enter the name of the client: ")
67        client_age = input("Enter the age of the client: ")
68        client_nif = input(
69            "Enter the NIF (tax identification number) of the client: ")
70        params = (client_phone, client_name, client_age, client_nif)
71        execute_procedure("AddCliente", params)
72        print("Client inserted successfully.")
73        new_entry = [str(val) for val in params]
74        insert_data_to_csv('data/cliente.csv', 'Cliente', ['Num_telef', 'Nome', 'Idade', 'NIF'], [new_entry])
75    elif choice == "99":
76        break
77    elif choice == "99":
78        cleanDB()
79        break
80    else:
81        print("Invalid choice. Please try again.")

```

Figura 6.21: Menu Interativo

Capítulo 7

Implementação do Sistema de Painéis de Análise

A implementação de um sistema de painéis de análise corresponde à fase de exploração ou monitorização do ciclo de vida uma base de dados.

De modo a realmente "ver os dados", será utilizada para esta tarefa o *software PowerBI* da *Microsoft*.

7.1 Integração dos Dados no PowerBI e Definição e caracterização da vista de dados para análise

1. Para conectar o *PowerBI* à nossa base de dados, é necessário utilizar o campo *get data*, e posteriormente seleccionar a opção *MySQL Database*.

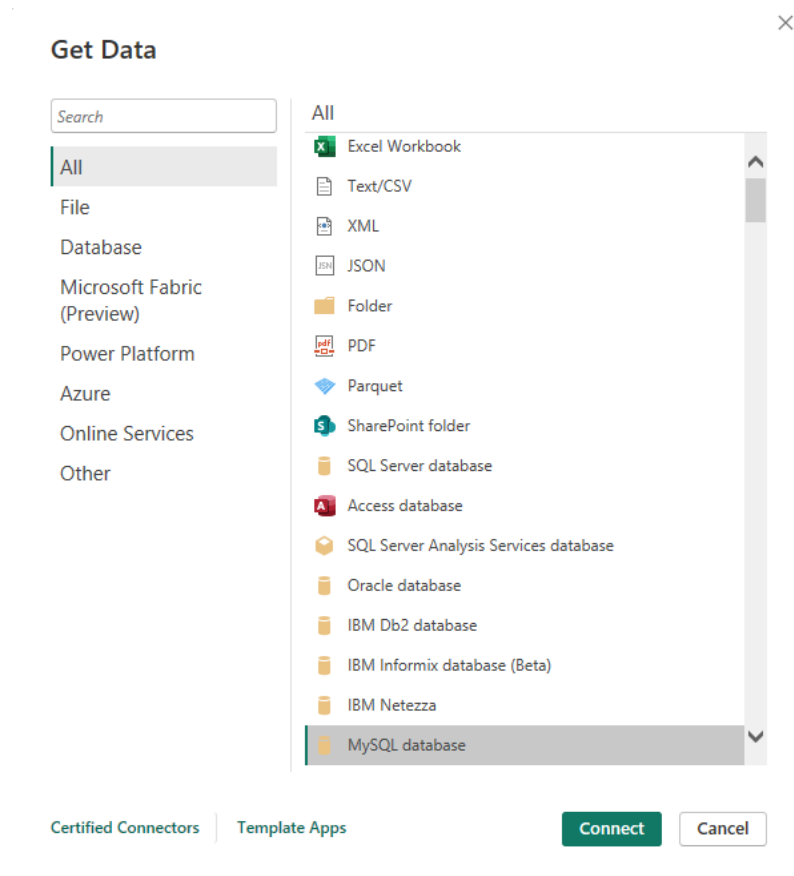


Figura 7.1: *Get Data* no *PowerBI*

2. Após pressionar *connect*, deve introduzir o nome do servidor (no caso *localhost*, porque a nossa máquina acomoda o nosso próprio servidor) e o nome da base de dados (no nosso caso *gameblockbuster*).



Figura 7.2: Conexão à Base de Dados, via *localhost*

3. Agora em *Navigator*, temos a opção de selecionar as tabelas que pretendemos utilizar para análise. Após selecionar as tabelas, pressionamos *load*.

A partir daqui importamos as tabelas e queries necessárias para a conversão de dados para um modelo visual.

7.2 Apresentação e caracterização dos dashboards implementados

Com os dados integrados na ferramenta, podemos obter diversos modelos estatísticos que se baseiam nas tabelas utilizadas.

Para servir como exemplo prático, eis alguns modelos estatísticos que o grupo considerou relevantes:

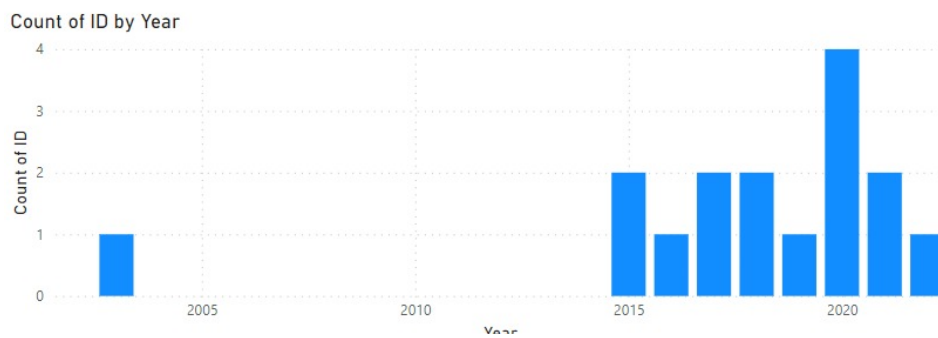


Figura 7.3: Distribuição dos jogos por ano de lançamento

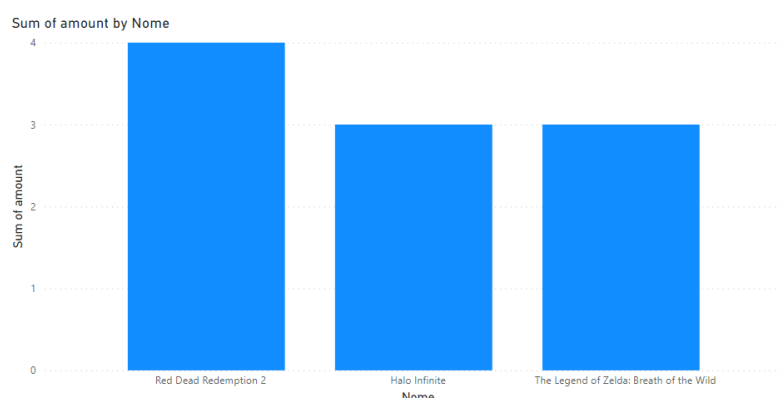


Figura 7.4: Jogos mais alugados desde sempre

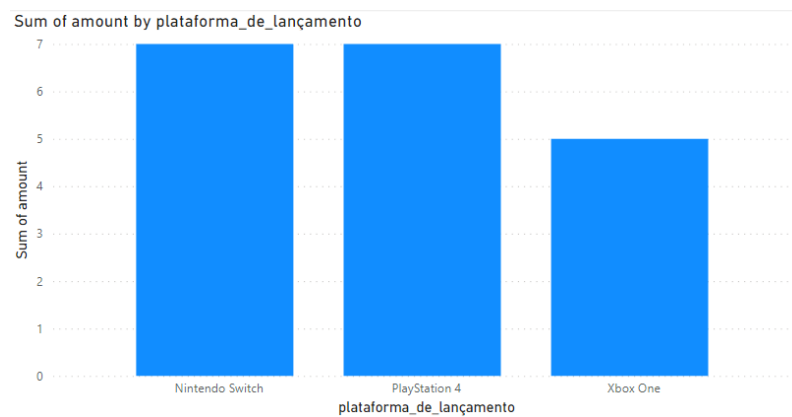


Figura 7.5: Plataformas mais alugadas

Sum of amount by release_year

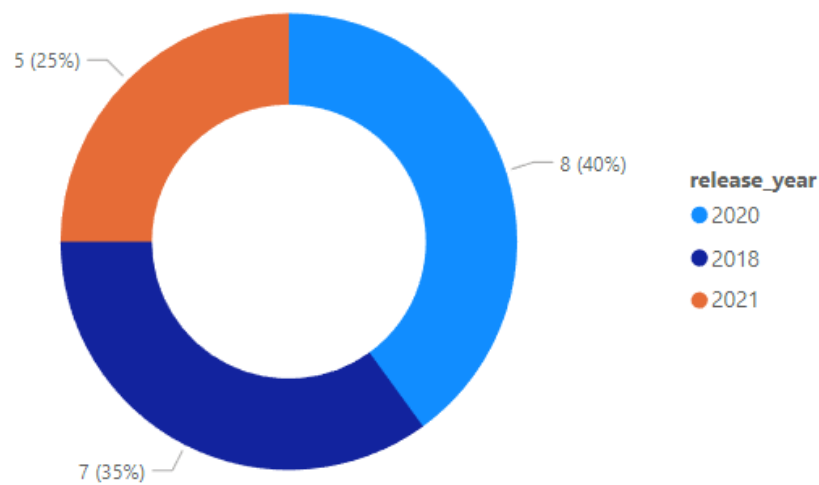


Figura 7.6: Datas de lançamento de jogos em que houve mais alugueres

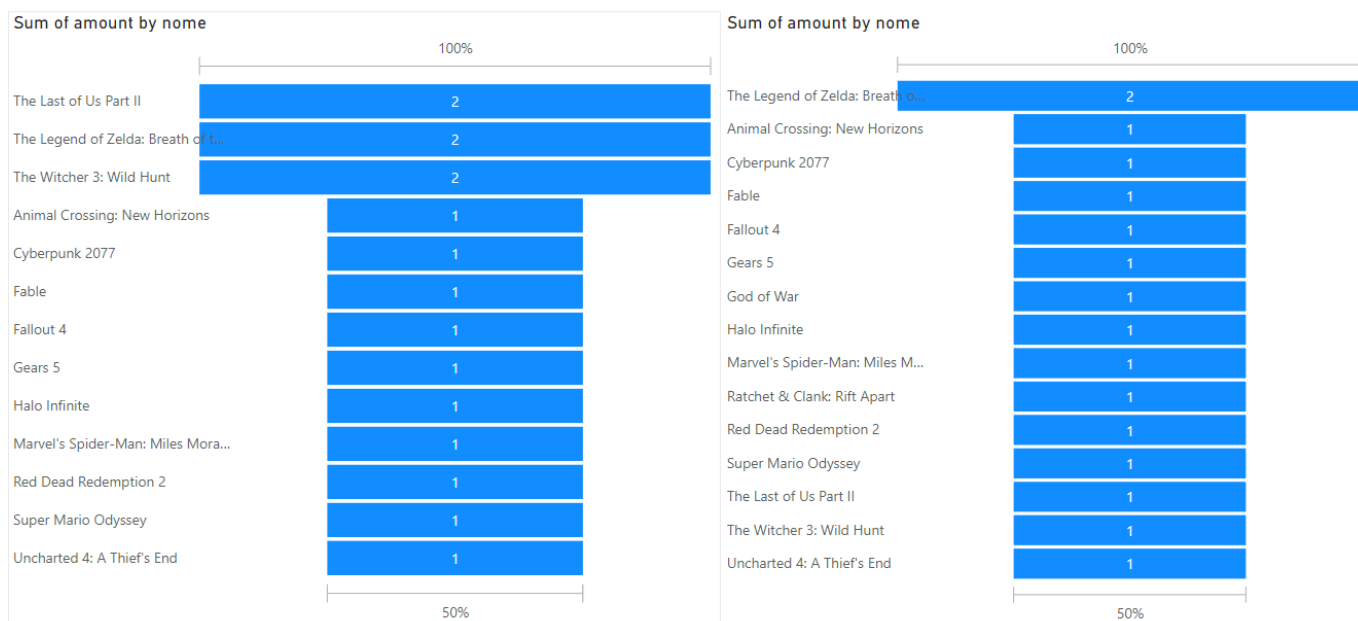


Figura 7.7: Jogos mais e menos apreciados pelos Clientes

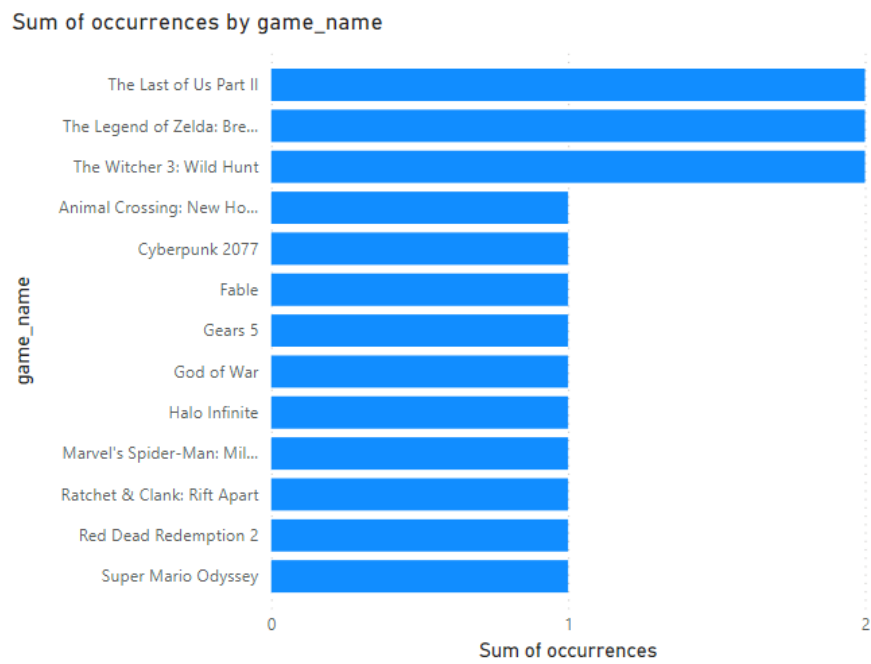


Figura 7.8: Jogos que os Clientes mais querem jogar

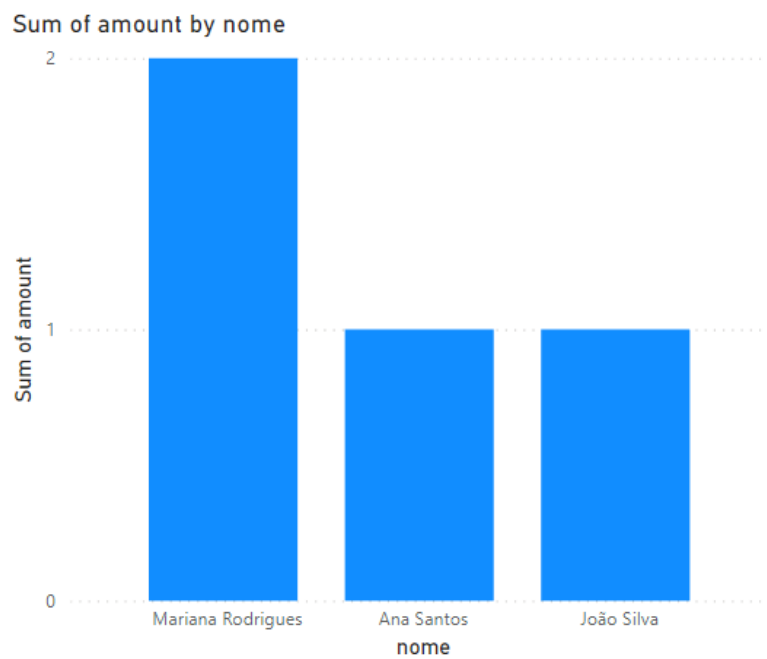


Figura 7.9: Vendedores com maior número de alugueres

Capítulo 8

Conclusões e Trabalho Futuro

Com a implementação deste Sistema de Gestão de Bases de Dados (SGBD), a loja do Sr. Alberto terá a oportunidade de expandir o seu negócio de forma mais organizada e eficiente. As funcionalidades e recursos oferecidos pelo SGBD, como inquéritos e operações de consulta eficientes, ajudarão a empresa a abandonar o seu estado de prejuízo e começar a obter lucro.

Este SGBD garante uma gestão empresarial mais organizada, ágil e informativa, com o objetivo de aumentar o número de funcionários e o stock da empresa.

Durante o desenvolvimento deste sistema, foram adotadas metodologias e operações específicas, com as razões para essas escolhas tendo explicadas de forma clara e concisa. No entanto, também podemos identificar possíveis melhorias que podem ser implementadas em momentos futuros.

Percebe-se que a implementação do SGBD proporcionará uma série de benefícios à loja do Sr. Alberto, tais como:

Melhor organização: O SGBD permite armazenar e gerir os dados da loja de maneira estruturada, facilitando a localização e recuperação das informações necessárias. Isso resulta em processos mais eficientes e redução de erros.

Agilidade nas operações: Com a automatização de tarefas repetitivas e a utilização de consultas otimizadas, as operações diárias da loja serão realizadas de forma mais rápida e eficiente. Isso economiza tempo e recursos, permitindo que a equipa se concentre em atividades mais estratégicas.

Informações relevantes: O SGBD oferece recursos de análise de stock (por exemplo), fornecendo ao Sr. Alberto *insights* valiosos sobre o desempenho da sua loja. Ele poderá tomar decisões com base em dados concretos, identificar tendências de mercado e ajustar a sua estratégia para maximizar os lucros.

Expansão do negócio: Com uma gestão mais eficaz dos recursos, como funcionários e stock, a loja do Sr. Alberto estará preparada para enfrentar um crescimento sustentável. O SGBD permitirá dimensionar as operações de maneira adequada, evitando problemas com sobrecarga ou desperdício de recursos.

Como mencionado anteriormente, ainda existem oportunidades de melhoria que podem ser consideradas no futuro, como:

- Inserção de auto-incrementadores nas tabelas do SGBD, para evitar a inserção de novos registos com ID's não sucessivos.
- Realizar a troca de Critics__Score de VARCHAR(75) para INT, na tabela Jogo.
- Criação de novas *queries* que sejam úteis para os administradores do SGBD, como por exemplo, queries que removem clientes da base de dados, sem afetar as facturas.

Referências

<https://dev.mysql.com/doc/refman/8.0/en/storage-requirements.html>

8.1 Anexos

8.1.1 Anexo 1 - Script de Inicialização da Base de Dados (CreateDB.sql)

```
SET @@OLD_UNIQUE_CHECKS=@@UNIQUE_CHECKS, UNIQUE_CHECKS=0;
SET @@OLD_FOREIGN_KEY_CHECKS=@@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0;
SET @@OLD_SQL_MODE=@@SQL_MODE, SQL_MODE='ONLY_FULL_GROUP_BY,STRICT_TRANS_TABLES,NO_ZERO_IN_DATE,NO_ZERO_DATE,
ERROR_FOR_DIVISION_BY_ZERO,NO_ENGINE_SUBSTITUTION';

CREATE SCHEMA IF NOT EXISTS 'GameBlockbuster' DEFAULT CHARACTER SET utf8;
USE 'GameBlockbuster';

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'Vendedor' (
  'ID' INT NOT NULL,
  'Nome' VARCHAR(75) NOT NULL,
  PRIMARY KEY ('ID')
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'Jogo' (
  'ID' INT NOT NULL,
  'Valor_De_Aluguer' DOUBLE NOT NULL,
  'Critics_Score' VARCHAR(75) NULL,
  'Pedidos_De_Aluguer' INT NULL,
  'Plataforma_De_Lançamento' VARCHAR(75) NOT NULL,
  'Nome' VARCHAR(75) NOT NULL,
  'Data_Lançamento' DATE NOT NULL,
  'Sinopse' VARCHAR(512) NULL,
  'Unidades_Em_Stock' VARCHAR(75) NOT NULL,
  PRIMARY KEY ('ID')
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'Cliente' (
  'Num_telem' INT NOT NULL,
```

```

'Nome' VARCHAR(75) NOT NULL,
'Idade' INT NOT NULL,
'NIF' VARCHAR(75) NOT NULL,
PRIMARY KEY ('NIF')
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'Inquerito' (
  'ID' INT NOT NULL,
  'Jogo_que_quer_jogar' INT NOT NULL,
  'Jogo_que_mais_jogou' INT NOT NULL,
  'Jogo_que_menos_jogou' INT NOT NULL,
  'Vendedor_ID' INT NOT NULL,
  'Cliente_NIF' VARCHAR(75) NOT NULL,
  PRIMARY KEY ('ID'),
  INDEX 'fk_Inquerito_Vendedor1_idx' ('Vendedor_ID' ASC) VISIBLE,
  INDEX 'fk_Inquerito_Cliente1_idx' ('Cliente_NIF' ASC) VISIBLE,
  CONSTRAINT 'fk_Inquerito_Vendedor1'
    FOREIGN KEY ('Vendedor_ID')
      REFERENCES 'GameBlockbuster'.'Vendedor' ('ID')
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT 'fk_Inquerito_Cliente1'
    FOREIGN KEY ('Cliente_NIF')
      REFERENCES 'GameBlockbuster'.'Cliente' ('NIF')
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'AluguerCab' (
  'ID' INT NOT NULL,
  'PrecoTotal' DOUBLE NOT NULL,
  'Periodo_Aluguer' DOUBLE NOT NULL,
  'data' DATE NOT NULL,
  'Cliente_NIF' VARCHAR(75) NOT NULL,
  'Vendedor_ID' INT NOT NULL,
  'Metodo_Pagamento' VARCHAR(75) NOT NULL,
  PRIMARY KEY ('ID'),
  INDEX 'fk_Aluguer_Cliente1_nifx' ('Cliente_NIF' ASC) VISIBLE,
  INDEX 'fk_AluguerCab_Vendedor1_idx' ('Vendedor_ID' ASC) VISIBLE,
  CONSTRAINT 'fk_Aluguer_Cliente1'
    FOREIGN KEY ('Cliente_NIF')
      REFERENCES 'GameBlockbuster'.'Cliente' ('NIF')
      ON DELETE NO ACTION
      ON UPDATE NO ACTION,
  CONSTRAINT 'fk_AluguerCab_Vendedor1'
    FOREIGN KEY ('Vendedor_ID')
      REFERENCES 'GameBlockbuster'.'Vendedor' ('ID')
      ON DELETE NO ACTION
      ON UPDATE NO ACTION
) ENGINE = InnoDB;

CREATE TABLE IF NOT EXISTS 'GameBlockbuster'.'AluguerLinhas' (
  'Quantidade' INT NOT NULL,

```

```

'PrecoUnit' DOUBLE NOT NULL,
'PrecoTotal' DOUBLE NOT NULL,
'AluguerCab_ID' INT NOT NULL,
'Jogo_ID' INT NOT NULL,
INDEX 'fk_AluguerLinhas_AluguerCab1_idx' ('AluguerCab_ID' ASC) VISIBLE,
INDEX 'fk_AluguerLinhas_Jogo1_idx' ('Jogo_ID' ASC) VISIBLE,
PRIMARY KEY ('AluguerCab_ID', 'Jogo_ID'),
CONSTRAINT 'fk_AluguerLinhas_AluguerCab1'
    FOREIGN KEY ('AluguerCab_ID')
    REFERENCES 'GameBlockbuster'.'AluguerCab' ('ID')
    ON DELETE NO ACTION
    ON UPDATE NO ACTION,
CONSTRAINT 'fk_AluguerLinhas_Jogo1'
    FOREIGN KEY ('Jogo_ID')
    REFERENCES 'GameBlockbuster'.'Jogo' ('ID')
    ON DELETE NO ACTION
    ON UPDATE NO ACTION
) ENGINE = InnoDB;

SET SQL_MODE=@OLD_SQL_MODE;
SET FOREIGN_KEY_CHECKS=@OLD_FOREIGN_KEY_CHECKS;
SET UNIQUE_CHECKS=@OLD_UNIQUE_CHECKS;

```

8.1.2 Anexo 2 - Script de Povoamento Inicial

```

INSERT INTO 'GameBlockbuster'.'Vendedor' ('ID', 'Nome') VALUES
(1, 'João Silva'),
(2, 'Ana Santos'),
(3, 'Pedro Costa'),
(4, 'Sara Ferreira'),
(5, 'Miguel Oliveira'),
(6, 'Mariana Rodrigues'),
(7, 'Tiago Pereira'),
(8, 'Carolina Almeida'),
(9, 'Ricardo Santos'),
(10, 'Inês Costa'),
(11, 'André Fernandes'),
(12, 'Marta Sousa'),
(13, 'António Silva'),
(14, 'Beatriz Cardoso'),
(15, 'Hugo Ribeiro'),
(16, 'Anti-Inquéritos');

INSERT INTO 'GameBlockbuster'.'Jogo' ('ID', 'Valor_De_Aluguer', 'Critics_Score', 'Pedidos_De_Aluguer',
'Plataforma_De_Lançamento', 'Nome', 'Data_Lançamento', 'Sinopse', 'Unidades_Em_Stock') VALUES
(1, 4.99, '90', 50, 'PlayStation 4', 'The Last of Us Part II', '2020-06-19', 'Ellie e Joel lutam zombies.', '20'),
(2, 3.49, '88', 30, 'Xbox One', 'Gears 5', '2019-09-10', 'desc.', '15'),
(3, 2.99, '92', 45, 'Nintendo Switch', 'The Legend of Zelda: Breath of the Wild', '2017-03-03', 'desc.', '10'),
(4, 5.99, '87', 40, 'PlayStation 5', 'Marvel''s Spider-Man: Miles Morales', '2020-11-12', 'desc', '5'),
(5, 1.99, '85', 20, 'PC', 'desc.', '25'),
(6, 4.49, '89', 35, 'Xbox Series X', 'Halo Infinite', '2021-12-08', 'desc', '12'),
(7, 3.99, '93', 55, 'PlayStation 5', 'Ratchet & Clank: Rift Apart', '2021-06-11', 'desc', '18'),

```



```
(8, 2.49, '86', 25, 'Nintendo Switch', 'Super Mario Odyssey', '2017-10-27', 'desc', '7'),
(9, 4.99, '90', 40, 'PlayStation 4', 'God of War', '2018-04-20', 'desc', '10'),
(10, 3.29, '88', 30, 'Xbox One', 'Red Dead Redemption 2', '2018-10-26', 'desc', '20'),
(11, 2.99, '91', 35, 'PC', 'Cyberpunk 2077', '2020-12-10', 'desc', '15'),
(12, 4.99, '87', 30, 'Nintendo Switch', 'Animal Crossing: New Horizons', '2020-03-20', 'desc', '8'),
(13, 1.99, '84', 15, 'PlayStation 4', 'Uncharted 4: A Thief's End', '2016-05-10', 'desc', '5'),
(14, 5.99, '93', 50, 'Xbox Series X', 'Fable', '2022-10-31', 'desc', '10'),
(15, 2.99, '89', 25, 'PC', 'Fallout 4', '2015-11-10', 'desc', '10'),
(16, 6.99, '70', 1, 'Nintendo GameCube', 'Sonic Heroes', '2003-12-30', 'Rail Canyon Sucks', 100);
```

```
INSERT INTO 'GameBlockbuster'. 'Cliente' ('Num_telef', 'Nome', 'Idade', 'NIF') VALUES
```

```
(912345678, 'Marta Costa', 28, '123456789'),
(933333333, 'Ricardo Santos', 35, '987654321'),
(966666666, 'Ana Rodrigues', 22, '456789123'),
(911111111, 'Pedro Almeida', 31, '789123456'),
(922222222, 'Sofia Pereira', 27, '654321987'),
(933456789, 'João Silva', 33, '123789456'),
(966123456, 'Carolina Sousa', 26, '987321654'),
(911234567, 'André Santos', 29, '456987123'),
(922345678, 'Inês Costa', 24, '789456321'),
(934567890, 'Hugo Martins', 32, '654123987'),
(966789012, 'Mariana Ferreira', 30, '321456789'),
(933214567, 'Rui Oliveira', 36, '987123456'),
(911345678, 'Beatriz Cardoso', 23, '456321987'),
(922456789, 'Diogo Gonçalves', 27, '789654321'),
(934678901, 'Cláudia Silva', 31, '654987123');
```

```
INSERT INTO 'GameBlockbuster'. 'Inquerito' ('ID', 'Jogo_que_quer_jogar', 'Jogo_que_mais_jogou',
'Jogo_que_menos_jogou', 'Vendedor_ID', 'Cliente_NIF') VALUES
```

```
(1, 2, 5, 3, 1, '123456789'),
(2, 1, 4, 2, 3, '987654321'),
(3, 3, 2, 1, 2, '456789123'),
(4, 5, 1, 4, 4, '789123456'),
(5, 4, 3, 5, 5, '654321987'),
(6, 7, 6, 8, 6, '789123456'),
(7, 10, 12, 9, 7, '654321987'),
(8, 12, 10, 11, 8, '123789456'),
(9, 9, 13, 14, 9, '987321654'),
(10, 14, 8, 15, 10, '456987123'),
(11, 11, 15, 13, 11, '789456321'),
(12, 6, 11, 12, 12, '321456789'),
(13, 3, 14, 10, 13, '456321987'),
(14, 5, 1, 7, 14, '789654321'),
(15, 8, 3, 6, 15, '654987123'),
(16, 1, 5, 3, 4, '789123456');
```

```
INSERT INTO GameBlockbuster.AluguerCab (ID, PrecoTotal, Periodo_Aluguer, data, Cliente_NIF, Vendedor_ID,
Metodo_Pagamento) VALUES
```

```
(1, 14.99, 3, '2023-05-01', '123456789', 1, 'Cartão de crédito'),
(2, 10.49, 2, '2023-05-02', '987654321', 2, 'Dinheiro'),
(3, 8.99, 2, '2023-05-03', '456789123', 3, 'Cartão de débito'),
(4, 19.99, 5, '2023-05-04', '789123456', 4, 'Transferência bancária'),
(5, 6.99, 1, '2023-05-05', '654321987', 5, 'Cartão de crédito'),
```

```
(6, 12.99, 3, '2023-05-06', '789123456', 6, 'Dinheiro'),
(7, 9.99, 2, '2023-05-07', '654321987', 7, 'Cartão de débito'),
(8, 16.99, 4, '2023-05-08', '123789456', 8, 'Transferência bancária'),
(9, 13.99, 3, '2023-05-09', '987321654', 9, 'Cartão de crédito'),
(10, 8.49, 2, '2023-05-10', '456987123', 10, 'Dinheiro'),
(11, 11.99, 3, '2023-05-11', '789456321', 11, 'Cartão de débito'),
(12, 18.99, 4, '2023-05-12', '321456789', 12, 'Transferência bancária'),
(13, 9.99, 2, '2023-05-13', '456321987', 13, 'Cartão de crédito'),
(14, 14.99, 3, '2023-05-14', '789654321', 14, 'Dinheiro'),
(15, 6.58, 1, '2023-05-15', '987321654', 15, 'Transferência bancária'),
(16, 12.99, 3, '2023-07-03', '987321654', 6, 'Transferência bancária');
```

```
INSERT INTO 'GameBlockbuster'.'AluguerLinhas' ('Quantidade', 'PrecoUnit', 'PrecoTotal', 'AluguerCab_ID',
'Jogo_ID') VALUES
(2, 4.99, 9.98, 1, 1),
(1, 3.49, 3.49, 1, 2),
(3, 2.99, 8.97, 2, 3),
(2, 5.99, 11.98, 3, 4),
(1, 1.99, 1.99, 4, 5),
(3, 4.49, 13.47, 5, 6),
(2, 3.99, 7.98, 6, 7),
(1, 2.49, 2.49, 7, 8),
(3, 4.99, 14.97, 8, 9),
(2, 3.29, 6.58, 9, 10),
(1, 2.99, 2.99, 10, 11),
(3, 4.99, 14.97, 11, 12),
(2, 1.99, 3.98, 12, 13),
(1, 5.99, 5.99, 13, 14),
(3, 2.99, 8.97, 14, 15),
(2,3.29,6.58,15,10);
```

8.1.3 Anexo 2 - Script de Criação das Vistas (Views.sql)

```
-- Views --

-- Query 1: Most rented games by customers
CREATE VIEW MostRentedGames AS
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID
ORDER BY amount DESC
LIMIT 3;

-- Query 2: Least rented games by customers
CREATE VIEW LeastRentedGames AS
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID
ORDER BY amount ASC
LIMIT 3;
```

```

-- Query 3: Most rented platforms by customers
CREATE VIEW MostRentedPlatforms AS
SELECT plataforma_de_lançamento, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY jogo.plataforma_de_lançamento
ORDER BY amount DESC
LIMIT 3;

-- Query 4: Release dates with the highest volume of rentals
CREATE VIEW ReleaseDatesWithMostRentals AS
SELECT YEAR(data_lançamento) AS release_year, SUM(quantidade) AS amount
FROM jogo
INNER JOIN aluguerlinhas ON jogo.id = aluguerlinhas.Jogo_ID
GROUP BY YEAR(data_lançamento)
ORDER BY amount DESC
LIMIT 3;

-- Query 5: Top spending customers in the store
CREATE VIEW TopSpendingCustomers AS
SELECT nome, SUM(precototal) AS Gasto_Total
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
GROUP BY nif
ORDER BY Gasto_Total DESC
LIMIT 3;

-- Query 6: Low spending customers in the store
CREATE VIEW LowSpendingCustomers AS
SELECT nome, SUM(precototal) AS Gasto_Total
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
GROUP BY nif
ORDER BY Gasto_Total ASC
LIMIT 3;

-- Query 7: Customers who have rented the same game more than once
CREATE VIEW CustomersWithMultipleGameRentals AS
SELECT nome
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
INNER JOIN aluguerlinhas ON aluguercab.id = aluguerlinhas.aluguercab_id
GROUP BY cliente.nif, aluguerlinhas.jogo_id
HAVING COUNT(*) > 1;

-- Query 8: Games that customers have enjoyed playing the most
CREATE VIEW MostLikedGames AS
SELECT jogo.nome, COUNT(*) AS amount
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_mais_jogou
GROUP BY jogo.nome
ORDER BY amount DESC;

```

```

-- Query 9: Games that customers have enjoyed playing the least
CREATE VIEW LeastLikedGames AS
SELECT jogo.nome, COUNT(*) AS amount
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_menos_jogou
GROUP BY jogo.nome
ORDER BY amount DESC;

-- Query 10: Games that customers want to play
CREATE VIEW DesiredGames AS
SELECT jogo.nome AS game_name, COUNT(*) AS occurrences
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_quer_jogar
GROUP BY jogo.nome
ORDER BY occurrences DESC;

-- Query 11: Sellers who have made the most rentals in the store
CREATE VIEW TopRentingSellers AS
SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN aluguercab ON vendedor.id = aluguercab.vendedor_id
GROUP BY vendedor.id
ORDER BY amount DESC
LIMIT 3;

-- Query 12: Sellers who have made the least rentals in the store
CREATE VIEW LowRentingSellers AS
SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN aluguercab ON vendedor.id = aluguercab.vendedor_id
GROUP BY vendedor.id
ORDER BY amount ASC
LIMIT 3;

-- Query 13: Sellers who have issued the most surveys in the store
CREATE VIEW TopSurveyIssuingSellers AS
SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN inquerito ON vendedor.id = inquerito.vendedor_id
GROUP BY vendedor.id
ORDER BY amount DESC
LIMIT 3;

-- Query 14: Sellers who have not issued any surveys
CREATE VIEW SellersWithoutSurveys AS
SELECT nome
FROM vendedor
WHERE vendedor.id NOT IN (SELECT vendedor_id FROM inquerito);

-- Query 15: Full content of the Jogo table
CREATE VIEW FullJogo AS

```

```

SELECT ID, Valor_De_Aluguer,Critics_Score,Pedidos_De_Aluguer, Plataforma_De_Lançamento,
Nome, Data_Lançamento,Unidades_Em_Stock
FROM Jogo;

```

```

-- Query 16: Full content of the Inquerito table
CREATE VIEW FullInquerito AS
SELECT *
FROM Inquerito;

```

```

-- Query 17: Full content of the Cliente table
CREATE VIEW FullCliente AS
SELECT *
FROM Cliente;

```

8.1.4 Anexo 3 - Script de Criação das Procedures (Procedures.sql)

```

-- Procedures --

```

```

-- Query 18: Get games based on the desired launch platform
CREATE PROCEDURE GetGamesByPlatform(IN platform VARCHAR(50))
SELECT *
FROM FullJogo
WHERE Plataforma_De_Lançamento = platform;

```

```

-- Query 19: Sort games by critics' score, price, and release date
CREATE PROCEDURE SortGamesByCriteria()
SELECT *
FROM FullJogo
ORDER BY critics_score DESC, valor_de_aluguer ASC, data_lançamento DESC;

```

```

-- Query 20: Sort games by the number of rentals
CREATE PROCEDURE SortGamesByRentalCount()
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID

```

```

UNION

```

```

SELECT Nome, 0 AS amount
FROM Jogo
WHERE ID NOT IN (SELECT Jogo_ID FROM AluguerLinhas)
ORDER BY amount DESC;

```

```

-- Query 21: Insert a new game into Jogo table
CREATE PROCEDURE AddJogo(
    IN p_ID INT,
    IN p_Valor_De_Aluguer DOUBLE,
    IN p_Critics_Score VARCHAR(75),
    IN p_Pedidos_De_Aluguer INT,

```

```

    IN p_Plataforma_De_lançamento VARCHAR(75),
    IN p_Nome VARCHAR(75),
    IN p_Data_lançamento DATE,
    IN p_Sinopse VARCHAR(512),
    IN p_Unidades_Em_Stock VARCHAR(75)
)
INSERT INTO 'GameBlockbuster'..'Jogo' (
    'ID',
    'Valor_De_Aluguer',
    'Critics_Score',
    'Pedidos_De_Aluguer',
    'Plataforma_De_lançamento',
    'Nome',
    'Data_lançamento',
    'Sinopse',
    'Unidades_Em_Stock'
) VALUES (
    p_ID,
    p_Valor_De_Aluguer,
    p_Critics_Score,
    p_Pedidos_De_Aluguer,
    p_Plataforma_De_lançamento,
    p_Nome,
    p_Data_lançamento,
    p_Sinopse,
    p_Unidades_Em_Stock
);

-- Query 22: Insert a new client into Cliente table
CREATE PROCEDURE AddCliente(
    IN p_Num_telem INT,
    IN p_Nome VARCHAR(75),
    IN p_Idade INT,
    IN p_NIF INT
)
INSERT INTO 'GameBlockbuster'..'Cliente' (
    'Num_telem',
    'Nome',
    'Idade',
    'NIF'
) VALUES (
    p_Num_telem,
    p_Nome,
    p_Idade,
    p_NIF
);

```

8.1.5 Anexo 4 - Script de Criação das Queries (Queries.sql)

```

-- Views --

-- Query 1: Most rented games by customers

```

```

CREATE VIEW MostRentedGames AS
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID
ORDER BY amount DESC
LIMIT 3;

-- Query 2: Least rented games by customers
CREATE VIEW LeastRentedGames AS
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID
ORDER BY amount ASC
LIMIT 3;

-- Query 3: Most rented platforms by customers
CREATE VIEW MostRentedPlatforms AS
SELECT plataforma_de_lançamento, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY jogo.plataforma_de_lançamento
ORDER BY amount DESC
LIMIT 3;

-- Query 4: Release dates with the highest volume of rentals
CREATE VIEW ReleaseDatesWithMostRentals AS
SELECT YEAR(data_lançamento) AS release_year, SUM(quantidade) AS amount
FROM jogo
INNER JOIN aluguerlinhas ON jogo.id = aluguerlinhas.Jogo_ID
GROUP BY YEAR(data_lançamento)
ORDER BY amount DESC
LIMIT 3;

-- Query 5: Top spending customers in the store
CREATE VIEW TopSpendingCustomers AS
SELECT nome, SUM(precototal) AS Gasto_Total
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
GROUP BY nif
ORDER BY Gasto_Total DESC
LIMIT 3;

-- Query 6: Low spending customers in the store
CREATE VIEW LowSpendingCustomers AS
SELECT nome, SUM(precototal) AS Gasto_Total
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
GROUP BY nif
ORDER BY Gasto_Total ASC
LIMIT 3;

```

```

-- Query 7: Customers who have rented the same game more than once
CREATE VIEW CustomersWithMultipleGameRentals AS
SELECT nome
FROM cliente
INNER JOIN aluguercab ON cliente.nif = aluguercab.cliente_nif
INNER JOIN aluguerlinhas ON aluguercab.id = aluguerlinhas.aluguercab_id
GROUP BY cliente.nif, aluguerlinhas.jogo_id
HAVING COUNT(*) > 1;

-- Query 8: Games that customers have enjoyed playing the most
CREATE VIEW MostLikedGames AS
SELECT jogo.nome, COUNT(*) AS amount
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_mais_jogou
GROUP BY jogo.nome
ORDER BY amount DESC;

-- Query 9: Games that customers have enjoyed playing the least
CREATE VIEW LeastLikedGames AS
SELECT jogo.nome, COUNT(*) AS amount
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_menos_jogou
GROUP BY jogo.nome
ORDER BY amount DESC;

-- Query 10: Games that customers want to play
CREATE VIEW DesiredGames AS
SELECT jogo.nome AS game_name, COUNT(*) AS occurrences
FROM jogo
INNER JOIN inquerito ON jogo.id = inquerito.jogo_que_quer_jogar
GROUP BY jogo.nome
ORDER BY occurrences DESC;

-- Query 11: Sellers who have made the most rentals in the store
CREATE VIEW TopRentingSellers AS
SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN aluguercab ON vendedor.id = aluguercab.vendedor_id
GROUP BY vendedor.id
ORDER BY amount DESC
LIMIT 3;

-- Query 12: Sellers who have made the least rentals in the store
CREATE VIEW LowRentingSellers AS
SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN aluguercab ON vendedor.id = aluguercab.vendedor_id
GROUP BY vendedor.id
ORDER BY amount ASC
LIMIT 3;

-- Query 13: Sellers who have issued the most surveys in the store
CREATE VIEW TopSurveyIssuingSellers AS

```



```

SELECT nome, COUNT(*) AS amount
FROM vendedor
INNER JOIN inquerito ON vendedor.id = inquerito.vendedor_id
GROUP BY vendedor.id
ORDER BY amount DESC
LIMIT 3;

-- Query 14: Sellers who have not issued any surveys
CREATE VIEW SellersWithoutSurveys AS
SELECT nome
FROM vendedor
WHERE vendedor.id NOT IN (SELECT vendedor_id FROM inquerito);

-- Query 15: Full content of the Jogo table
CREATE VIEW FullJogo AS
SELECT ID, Valor_De_Aluguer,Critics_Score,Pedidos_De_Aluguer,
Plataforma_De_Lançamento, Nome, Data_Lançamento,Unidades_Em_Stock
FROM Jogo;

-- Query 16: Full content of the Inquerito table
CREATE VIEW FullInquerito AS
SELECT *
FROM Inquerito;

-- Query 17: Full content of the Cliente table
CREATE VIEW FullCliente AS
SELECT *
FROM Cliente;

-- Procedures --

-- Query 18: Get games based on the desired launch platform
CREATE PROCEDURE GetGamesByPlatform(IN platform VARCHAR(50))
SELECT *
FROM FullJogo
WHERE Plataforma_De_Lançamento = platform;

-- Query 19: Sort games by critics' score, price, and release date
CREATE PROCEDURE SortGamesByCriteria()
SELECT *
FROM FullJogo
ORDER BY critics_score DESC, valor_de_aluguer ASC, data_lançamento DESC;

-- Query 20: Sort games by the number of rentals
CREATE PROCEDURE SortGamesByRentalCount()
SELECT Nome, SUM(quantidade) AS amount
FROM Jogo
INNER JOIN AluguerLinhas ON Jogo.ID = AluguerLinhas.Jogo_ID
GROUP BY Jogo.ID

UNION

```

```

SELECT Nome, 0 AS amount
FROM Jogo
WHERE ID NOT IN (SELECT Jogo_ID FROM AluguerLinhas)
ORDER BY amount DESC;

```

-- Query 21: Insert a new game into Jogo table

```

CREATE PROCEDURE AddJogo(
    IN p_ID INT,
    IN p_Valor_De_Aluguer DOUBLE,
    IN p_Critics_Score VARCHAR(75),
    IN p_Pedidos_De_Aluguer INT,
    IN p_Plataforma_De_lançamento VARCHAR(75),
    IN p_Nome VARCHAR(75),
    IN p_Data_lançamento DATE,
    IN p_Sinopse VARCHAR(512),
    IN p_Unidades_Em_Stock VARCHAR(75)
)
INSERT INTO 'GameBlockbuster'.'Jogo' (
    'ID',
    'Valor_De_Aluguer',
    'Critics_Score',
    'Pedidos_De_Aluguer',
    'Plataforma_De_lançamento',
    'Nome',
    'Data_lançamento',
    'Sinopse',
    'Unidades_Em_Stock'
) VALUES (
    p_ID,
    p_Valor_De_Aluguer,
    p_Critics_Score,
    p_Pedidos_De_Aluguer,
    p_Plataforma_De_lançamento,
    p_Nome,
    p_Data_lançamento,
    p_Sinopse,
    p_Unidades_Em_Stock
);

```

-- Query 22: Insert a new client into Cliente table

```

CREATE PROCEDURE AddCliente(
    IN p_Num_telem INT,
    IN p_Nome VARCHAR(75),
    IN p_Idade INT,
    IN p_NIF INT
)
INSERT INTO 'GameBlockbuster'.'Cliente' (
    'Num_telem',
    'Nome',
    'Idade',
    'NIF'
) VALUES (
    p_Num_telem,

```

```

    p_Nome,
    p_Idade,
    p_NIF
);

```

8.1.6 Anexo 5 - Script em Python para Automatização da Base de Dados

```

import os
import mysql.connector
from tabulate import tabulate
import csv

cnx = mysql.connector.connect(
    host="localhost",
    user="root",
    password="password",
    charset="utf8"
)
cursor = cnx.cursor()

def execute_query(query):
    cursor.execute(query)
    results = cursor.fetchall()
    if len(results) > 0:
        headers = [desc[0] for desc in cursor.description]
        print(tabulate(results, headers, tablefmt="fancy_grid"))
    else:
        print("No results found.")

def executeQuery(file_path):
    # Set the encoding when opening the SQL file
    with open(file_path, 'r', encoding='utf8') as sql_file:
        sql_script = sql_file.read()
        statements = sql_script.split(';')
        for statement in statements:
            if statement.strip():
                cursor.execute(statement)
        cnx.commit()
        print(f"Script {file_path} executed successfully.")

def insert_data_to_csv(csv_file, table_name, column_names, new_entries):
    with open(csv_file, 'a', newline="") as file:
        writer = csv.writer(file, delimiter=";")
        for entry in new_entries:
            writer.writerow(entry)
    cnx.commit()

```

```

def insert_data_from_csv(csv_file, table_name, column_names):
    with open(csv_file, newline="") as file:
        reader = csv.DictReader(file, delimiter=";")

        for row in reader:
            values = [row[column] for column in column_names]
            placeholders = ', '.join(['%s'] * len(column_names))
            sql = f"INSERT INTO {table_name} ({', '.join(column_names)}) VALUES ({placeholders})"
            cursor.execute(sql, values)
            cnx.commit()

def cleanDB():
    cursor.execute("DROP DATABASE IF EXISTS GameBlockbuster")

def execute_procedure(procedure_name, params=None):
    if params:
        cursor.callproc(procedure_name, params)
    else:
        cursor.callproc(procedure_name)

    for result in cursor.stored_results():
        results = result.fetchall()
        if len(results) > 0:
            headers = [desc[0] for desc in result.description]
            print(tabulate(results, headers, tablefmt="fancy_grid"))
        else:
            print("No results found.")

    cnx.commit()
    print("Procedure executed successfully.")

def populate_database_from_csv():
    executeQuery("scripts/CreateDB.sql")
    print("Populating from csv data files.")
    for file in csv_files:
        insert_data_from_csv(file['file'], file['table'], file['columns'])
    executeQuery("scripts/Views.sql")
    executeQuery("scripts/Procedures.sql")

def populate_database_from_sql():
    executeQuery("scripts/CreateDB.sql")
    print("Populating from SQL population script.")
    executeQuery("scripts/PopulateScript.sql")
    executeQuery("scripts/Views.sql")
    executeQuery("scripts/Procedures.sql")

# variables -----
csv_files = [
    {
        'file': 'data/vendedor.csv',
        'table': 'Vendedor',

```

```

        'columns': ['ID', 'Nome']
    },
    {
        'file': 'data/jogo.csv',
        'table': 'Jogo',
        'columns': ['ID', 'Valor_De_Aluguer', 'Critics_Score', 'Pedidos_De_Aluguer', 'Plataforma_De_Lançamento', 'Nome'],
    },
    {
        'file': 'data/cliente.csv',
        'table': 'Cliente',
        'columns': ['Num_telem', 'Nome', 'Idade', 'NIF']
    },
    {
        'file': 'data/inquerito.csv',
        'table': 'Inquerito',
        'columns': ['ID', 'Jogo_que_quer_jogar', 'Jogo_que_mais_jogou', 'Jogo_que_menos_jogou', 'Vendedor_ID', 'Cliente_ID'],
    },
    {
        'file': 'data/AluguerCab.csv',
        'table': 'AluguerCab',
        'columns': ['ID', 'PrecoTotal', 'Periodo_Aluguer', 'data', 'Cliente_NIF', 'Vendedor_ID', 'Metodo_Pagamento']
    },
    {
        'file': 'data/AluguerLinhas.csv',
        'table': 'AluguerLinhas',
        'columns': ['Quantidade', 'PrecoUnit', 'PrecoTotal', 'AluguerCab_ID', 'Jogo_ID']
    }
]

```

```
menu = ""
```

```
Menu:
```

1. Most rented games by customers
2. Least rented games by customers
3. Most rented platforms by customers
4. Release dates with the highest volume of rentals
5. Top spending customers in the store
6. Low spending customers in the store
7. Customers who have rented the same game more than once
8. Games that customers have enjoyed playing the most
9. Games that customers have enjoyed playing the least
10. Games that customers want to play
11. Sellers who have made the most rentals in the store
12. Sellers who have made the least rentals in the store
13. Sellers who have issued the most surveys in the store
14. Sellers who have not issued any surveys
15. Full content of the Jogo table
16. Full content of the Inquerito table
17. Full content of the Cliente table
18. Get games based on the desired launch platform
19. Sort games by critics' score, price, and release date
20. Sort games by the number of rentals
21. Insert a new game into Jogo table
22. Insert a new client into Cliente table

```

0. Exit
99. Clean Database
"""

#script-----

# Execute SQL
cursor.execute("CREATE DATABASE IF NOT EXISTS GameBlockbuster")
cursor.execute("USE GameBlockbuster")

# Check if the table exists
# Execute the query to retrieve table names
cursor.execute("SHOW TABLES")

# Fetch all the rows returned by the query
tables = cursor.fetchall()

# Check if any tables are returned
while True:

    if len(tables) > 0:
        print("There are tables in the database.")
        break
    else:
        print("Menu:")
        print("(1) Populate database from CSV files?")
        print("(2) Populate database from SQL script?")
        choice = input("Enter your choice: ")

        if choice == "1":
            populate_database_from_csv()
            break
        elif choice == "2":
            populate_database_from_sql()
            break
        else:
            print("Invalid choice.")

while True:
    print(menu)
    choice = input("Enter your choice: ")

    if choice == "1":
        execute_query("SELECT * FROM MostRentedGames")
    elif choice == "2":
        execute_query("SELECT * FROM LeastRentedGames")
    elif choice == "3":
        execute_query("SELECT * FROM MostRentedPlatforms")
    elif choice == "4":
        execute_query("SELECT * FROM ReleaseDatesWithMostRentals")

```

```

elif choice == "5":
    execute_query("SELECT * FROM TopSpendingCustomers")
elif choice == "6":
    execute_query("SELECT * FROM LowSpendingCustomers")
elif choice == "7":
    execute_query("SELECT * FROM CustomersWithMultipleGameRentals")
elif choice == "8":
    execute_query("SELECT * FROM MostLikedGames")
elif choice == "9":
    execute_query("SELECT * FROM LeastLikedGames")
elif choice == "10":
    execute_query("SELECT * FROM DesiredGames")
elif choice == "11":
    execute_query("SELECT * FROM TopRentingSellers")
elif choice == "12":
    execute_query("SELECT * FROM LowRentingSellers")
elif choice == "13":
    execute_query("SELECT * FROM TopSurveyIssuingSellers")
elif choice == "14":
    execute_query("SELECT * FROM SellersWithoutSurveys")
elif choice == "15":
    execute_query("SELECT * FROM FullJogo")
elif choice == "16":
    execute_query("SELECT * FROM FullInquerito")
elif choice == "17":
    execute_query("SELECT * FROM FullCliente")
elif choice == "18":
    platform = input("Enter the desired launch platform: ")
    execute_procedure("GetGamesByPlatform", (platform,))
elif choice == "19":
    execute_procedure("SortGamesByCriteria")
elif choice == "20":
    execute_procedure("SortGamesByRentalCount")
elif choice == "21":
    game_id = input("Enter the ID of the game: ")
    rental_value = float(input("Enter the rental value of the game: "))
    critics_score = input("Enter the critic's score of the game: ")
    rental_requests = input(
        "Enter the number of rental requests for the game: ")
    launch_platform = input("Enter the launch platform of the game: ")
    game_name = input("Enter the name of the game: ")
    release_date = input(
        "Enter the release date of the game (YYYY-MM-DD): ")
    synopsis = input("Enter the synopsis of the game: ")
    stock_units = input("Enter the number of stock units for the game: ")
    params = (game_id, rental_value, critics_score, rental_requests,
              launch_platform, game_name, release_date, synopsis, stock_units)
    execute_procedure("AddJogo", params)
    print("Game inserted successfully.")
    new_entry = [str(val) for val in params]
    insert_data_to_csv('data/jogo.csv', 'Jogo', ['ID', 'Valor_De_Aluguer', 'Critics_Score', 'Pedidos_De_Aluguer',
        'Plataforma_De_Lançamento', 'Nome', 'Data_Lançamento', 'Sinopse', 'Unidades_Em_Stock'], [new_entry])
elif choice == "22":

```

```

client_phone = input("Enter the client's phone number: ")
client_name = input("Enter the name of the client: ")
client_age = input("Enter the age of the client: ")
client_nif = input(
    "Enter the NIF (tax identification number) of the client: ")
params = (client_phone, client_name, client_age, client_nif)
execute_procedure("AddCliente", params)
print("Client inserted successfully.")
new_entry = [str(val) for val in params]
insert_data_to_csv('data/cliente.csv', 'Cliente', ['Num_telem', 'Nome', 'Idade', 'NIF'], [new_entry])
elif choice == "0":
    break
elif choice == "99":
    cleanDB()
    break
else:
    print("Invalid choice. Please try again.")

cursor.close()
cnx.close()

```