ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Trabalho prático

Sistemas de Informação

Fase 1

47224	André Graça
49149	Diogo Guerra
48459	Diogo Santos

Docente: Walter Vieira

Relatório de progresso realizado no âmbito do trabalho prático de Sistemas de informação, do curso de licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2022/2023

Maio de 2023

Resumo

Este relatório descreve o processo de desenvolvimento de um sistema de gerenciamento de jogos, jogadores e partidas para a empresa "GameOn". O objetivo do sistema é registrar e organizar informações relacionadas a jogadores, jogos, partidas, regiões, conversas e crachás.

O relatório começa descrevendo as etapas iniciais do projeto, que envolvem a identificação das entidades principais e seus atributos, bem como as relações entre elas. Para cada entidade, são definidos os atributos relevantes para o sistema. Em seguida, são apresentadas as restrições de integridade aplicadas ao sistema. Além disso, é discutido o grau das relações estabelecidas entre as entidades.

O relatório também aborda a modelagem do banco de dados, com a criação das tabelas correspondentes a cada entidade e a definição das chaves primárias e estrangeiras para estabelecer as relações entre as tabelas.

Uma parte importante do sistema são as funções, procedimentos e gatilhos implementados. São explicadas em detalhes as funções desenvolvidas e como foram escolhidos diferentes níveis de isolamento para cada procedimento, levando em consideração as necessidades específicas de cada operação. Foram selecionados níveis de isolamento adequados para cada procedimento, priorizando a consistência dos dados nos níveis mais altos e a performance nas operações simples nos níveis mais baixos. Essa abordagem equilibrada permitiu um controle eficiente da integridade dos dados no sistema, garantindo ao mesmo tempo uma boa performance.

Abstract

This report describes the development process of a game management system for the company "GameOn," which aims to record and organize information related to players, games, matches, regions, conversations, and badges.

The report begins by outlining the initial stages of the project, which involve identifying the main entities and their attributes, as well as the relationships between them. Relevant attributes for each entity are defined for the system. Additionally, integrity constraints applied to the system are presented, and the degree of relationships established between entities is discussed.

The report also covers the database modeling aspect, including the creation of tables corresponding to each entity and the definition of primary and foreign keys to establish relationships between tables.

An important aspect of the system is the implementation of functions, procedures, and triggers. The developed functions are explained in detail, and different levels of isolation were chosen for each procedure, considering the specific needs of each operation. Appropriate isolation levels were selected, prioritizing data consistency at higher levels and performance in simple operations at lower levels. This balanced approach allowed for efficient control of data integrity in the system while ensuring good performance.

Índice

Lista de Figuras

1. Introdução

O presente relatório tem como objetivo descrever o processo de desenvolvimento no nosso trabalho para construir uma base de dados, que visa gerir jogos, jogadores e as partidas realizadas por eles. Neste projeto, utilizamos as melhores práticas de modelagem de dados e implementação de um banco de dados relacional utilizando a linguagem SQL (Structured Query Language), especificamente o PostgreSQL.

Inicialmente, realizamos a análise dos requisitos e do enunciado do sistema, identificando as entidades envolvidas, seus atributos e as relações entre elas. Em seguida, construímos o modelo Entidade-Associação (EA), que nos proporcionou uma representação visual clara da estrutura dos dados e das interações entre as entidades.

A partir do modelo EA, avançamos para a fase de conversão, onde transformamos o modelo EA em um modelo relacional. Essa etapa envolveu a definição das tabelas, atributos, restrições de integridade e relacionamentos, levando em consideração as necessidades e regras de negócio do sistema.

Com o modelo relacional definido, implementamos as tabelas no banco de dados PostgreSQL, utilizando a linguagem SQL para criar as estruturas necessárias e estabelecer as relações entre as entidades. Cada tabela foi cuidadosamente projetada para armazenar os dados de forma organizada e eficiente, garantindo a integridade e consistência das informações.

Ao longo deste relatório, detalharemos cada etapa do processo de modelagem e implementação, apresentando o esquema do modelo Entidade-Associação, o código SQL das tabelas e suas respectivas descrições. Além disso, discutiremos as decisões tomadas durante o processo, explicando as razões por trás delas.

No final do projeto, esperamos fornecer um sistema robusto e eficiente para gerir jogos, jogadores e partidas, contribuindo para a melhoria de suas operações e oferecendo uma experiência aprimorada aos usuários.

2.Preparação da Construção da Base de Dados

Nesta seção, iremos explorar e explicar as ideias e questões fundamentais que tivémos ao construir esta base de dados que nos foi proposta. Descreveremos o contexto e os processos envolvidos, identificando as necessidades e limitações existentes.

2.1 Modelo Entidade-Associação

Primeiramente, construímos o modelo entidade-associação (EA) para representar visualmente as entidades, seus atributos e as relações entre elas. Esse modelo permitirá uma compreensão clara da estrutura dos dados e de como eles se relacionam. A partir do modelo EA, poderemos avançar para a construção do modelo relacional.

2.1.1 Formulação do Modelo Entidade-Associação

Como foi descrito, para construir o modelo Entidade-Associação, é necessário identificar, no texto que nos foi dado, os atributos das diferentes entidades mencionadas no enunciado, bem como as relações existentes entre elas. No fim, enumeramos algumas restrições de integridade para garantir a exatidão e a consistência dos dados.

1. Identificação das entidades e atributos:

   - Jogador: Identificado por um ID gerado pelo sistema, com atributos como email, username e estado.
   - Região: Representa as diferentes regiões dos jogadores, com o atributo nome.
   - Jogo: Possui um identificador alfanumérico, nome e URL para detalhes do jogo.
   - Partida: Representa as partidas jogadas, com informações como número sequencial(id), data e hora de início e fim. Assim sendo, definimos Partida como entidade fraca de Jogo
   - Crachá: São recompensas concedidas aos jogadores quando atingem um limite de pontos em um jogo, formados por nome, limite de pontos e URL para a sua imagem. Esta também é descrita por estar dependente de Jogo: “cada jogo pode ter um conjunto de crachás”.
   - Conversa: regista e organiza as interações entre os jogadores em uma conversa específica. Possui id e nome como atributos e é entidade fraca de Jogador pois uma conversa precisa

de jogadores para se interagirem-se um com os outros, logo a conversa é dependente da entidade Jogador.

- Mensagem: Cria um texto gerado pelo um jogador com meio de interagir com outro(s) jogadore(s) numa conversa. Os atributos desta entidade são: id, texto, data e hora. A mensagem é uma entidade fraca da Conversa porque ela depende da criação de uma conversa.
- Estatística: Mantém estatísticas relacionadas a cada jogador, como o número de partidas, número de jogos jogados e total de pontos.

2.1.2 Modelo EA

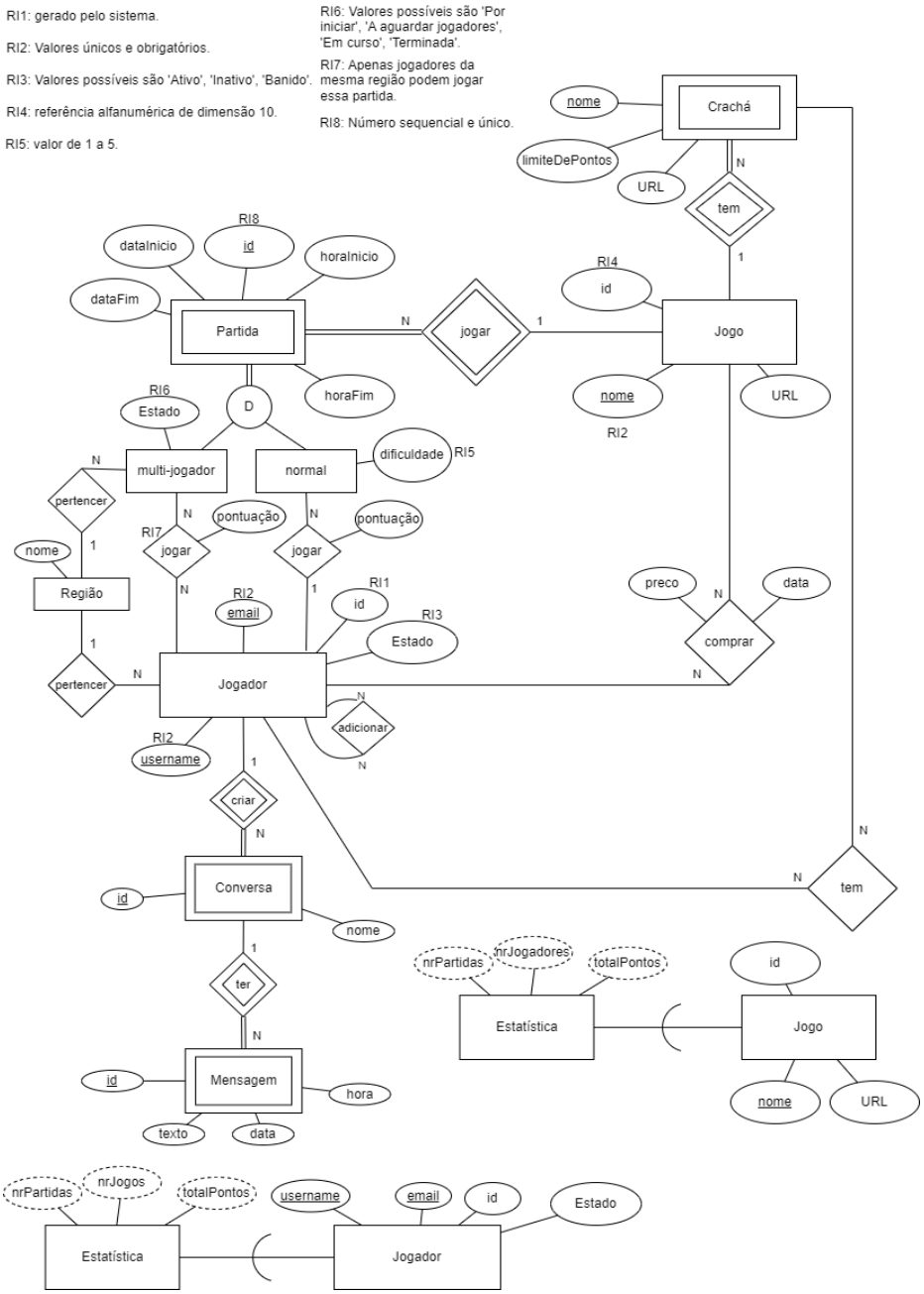

Figura 1- Modelo EA

O esquema representa a representação visual do modelo Entidade-Associação, que é uma representação gráfica das entidades, seus atributos e as relações entre elas. A figura apresenta a estrutura do esquema que servirá como base para a construção do modelo relacional e, posteriormente, para a implementação das tabelas no banco de dados.

2.2 Modelo ER

Este modelo é obtido a partir da aplicação das regras de passagem para o Modelo Relacional ao Modelo EA. Ao longo das subsecções irá se explicado como foram aplicadas essas regras para cada caso.

2.2.1 Entidades

A regra de passagem para Entidades é aplicada da seguinte forma, todos os atributos simples presentes na entidade no Modelo EA passam a fazer parte da entidade no Modelo ER e que de todas as chaves presentes no Modelo EA escolhemos 1 para ser a chave primária no Modelo ER, as restantes passam a ser chaves candidatas.

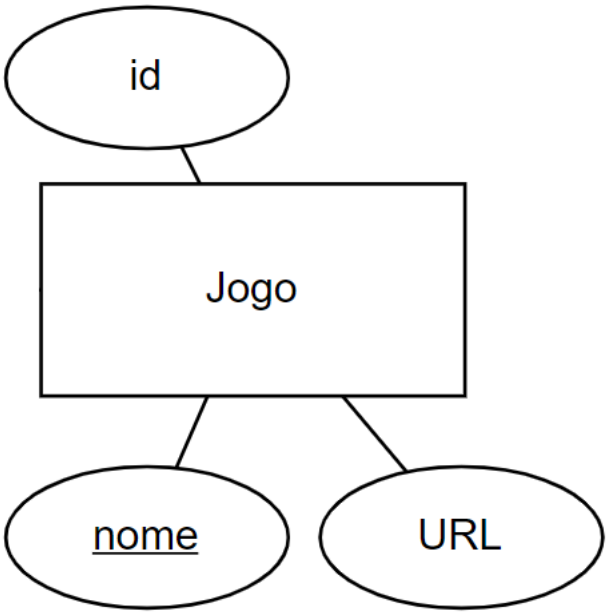

Figura 2 - Entidade

Na figura acima está presente a entidade *Jogo* que possui 3 atributos simples, sendo um deles uma possível chave porque é único, pois este encontra-se sublinhado. Aplicando a regra descrita em cima:

Jogo (nome, id, URL)

PK: nome

AK: id

Optámos por escolher o atributo nome como chave primária, e o atributo id como chave candidata, os restantes atributos são colocados na entidade.

2.2.2 Entidades Fracas

A regra de passagem para Entidades Fracas é aplicada de forma semelhante á regra de passagem de Entidades, com a diferença de a chave primária tem de ser composta por uma chave da entidade fraca e pela chave primária da entidade da qual ela depende.

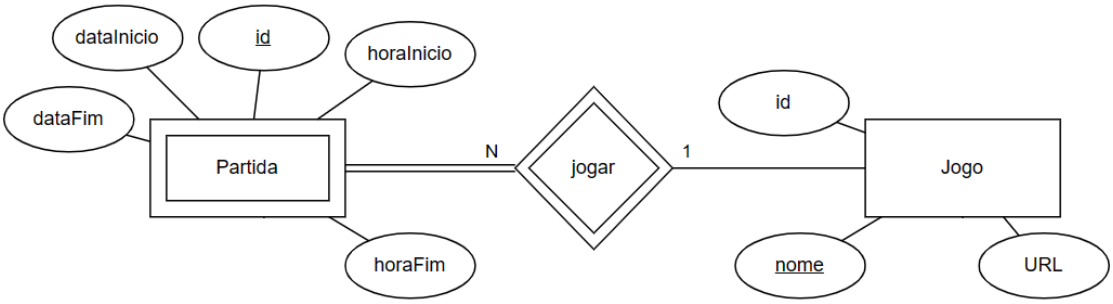

Figura 3 - Entidade Fraca

Na figura acima está representado a entidade *Partida* que é entidade fraca de *Jogo*, como tínhamos observado na secção anterior *Jogo* tem como chave primária o atributo nome, logo a chave primária de entidade será composta por id e nomeJogo que é uma referência para o atributo nome em *Jogo*.

Partida (id, nomeJogo, dataInicio, dataFim)

PK: id e nomeJogo

FK: {nomeJogo} de Jogo.nome

2.2.3 Associações

A regra de passagem para Associações é aplicada transformando a associação numa relação e a lógica por de trás da escolha da sua chave primária depende do seu grau de associação, no nosso Modelo EA, não está presente nenhuma associação de grau 1:1, pelo que não será abordada nas seguintes subsecções.

2.2.3.1 Grau 1:N

A regra de passagem para Associações de grau 1:N é aplicada da seguinte forma, a entidade do lado do "N" terá uma referência para entidade do lado do "1", ou seja, a entidade do lado "N" irá ter uma chave estrangeira que referência a chave primária da entidade do lado do "1".

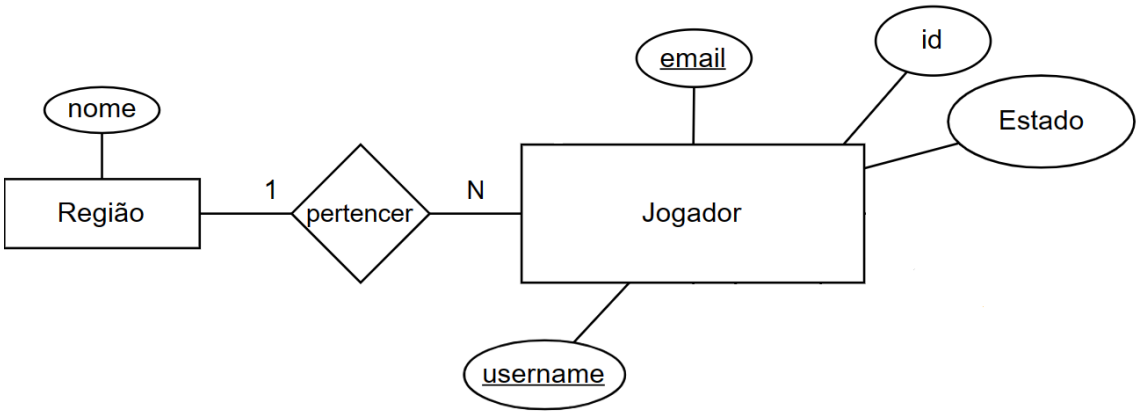

Figura 4 - Associação grau 1:N

Na figura acima está representada a entidade *Jogador* que, como dito em cima, terá uma referência para a chave primária de *Regiao*.

Jogador (id, estado, userName, email, nomeRegiao)

PK: id

AK: estado e userName

FK: {nomeRegiao} de Regiao.nome

2.2.3.2 Grau N:N

A regra de passagem para Associações de grau N:N é aplicada da seguinte forma, é criada uma entidade no modelo ER, que representa a associação, esta terá uma chave primária composta pela chave primária de ambas as entidades que associa.

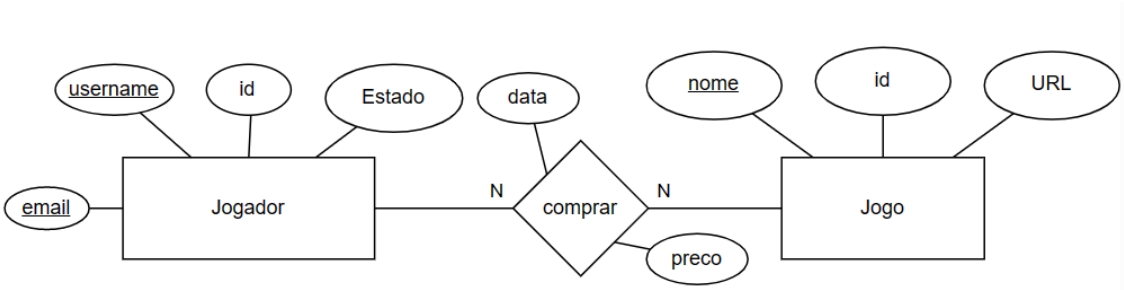

Figura 5 - Associação de grau N:N

Na figura acima está representada a associação *comprar* que quando passada para o modelo ER, ficará com uma chave primária composta por nome (chave primária de Jogo) e id (chave primária de Jogador).

Comprar (idJogador, nomeJogo, preco, data)

PK: idJogador e nomeJogo

FK: {idJogador} de Jogador.id e {nomeJogo} de Jogo.nome

2.2.4 Generalizações

A regra de passagem de generalizações depende do tipo de generalização, no nosso caso apenas iremos abordar o tipo de generalização disjunto.

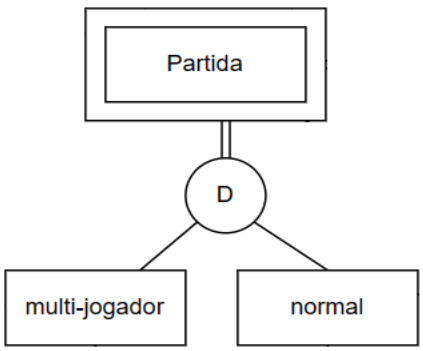

Figura 6 - Generalização Disjunta

Na figura acima está representada a entidade *partida* e os seus dois possíveis tipos, entidade *multi-jogador* e entidade *normal*, neste caso ambas entidades terão como chave primária a chave primária de partida e os atributos simples que cada contém.

MultiJogador (idPartida, nomeJogo, estado, nomeRegiao)

PK: idPartida e nomeJogo

FK: {idPartida} de Partida.id , {nomeJogo} de Jogo.nome e {nomeRegiao} de Regiao.nome

Um outro tipo de generalizações presente no nosso trabalho é a associação direta, onde ambas as entidades partilharam a chave primária entre si.

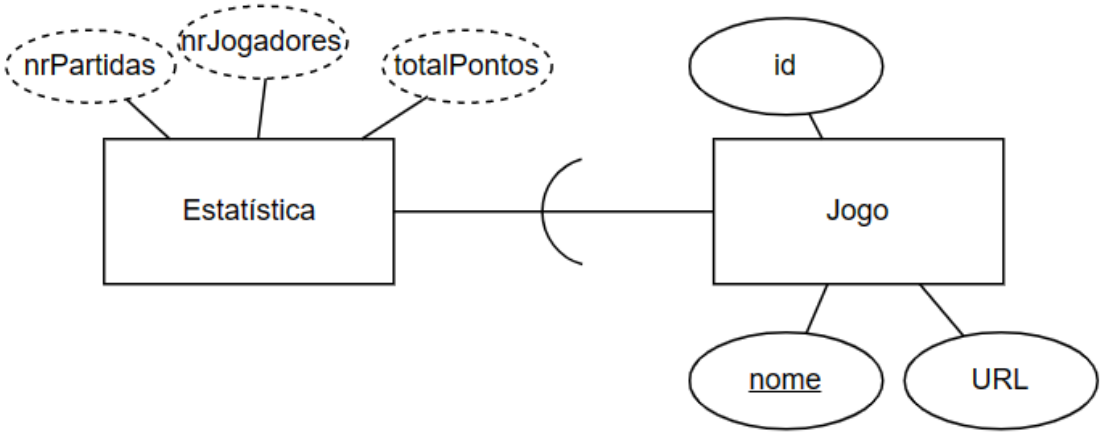

Figura 7 - Associação Direta

Na figura acima está representada a associação direta entre *Jogo* e *Estatística*, o efeito que esta associação tem no modelo ER, é que *Estatística* terá como chave primária a chave primária de *Jogo*, isto é o nome do jogo.

EstatísticaJogador (idJogador, nrPartidas, nrJogos, totalPontosJogos)

PK: idJogador

FK: {idJogador} de Jogador.id

2.2.5 Modelo ER

Jogador (id, estado, userName, email, nomeRegiao)
PK: id
AK: email e userName
FK: {nomeRegiao} de Regiao.nome

Jogo (nome, id, URL)
PK: nome
AK: id

Regiao (nome)
PK: nome

Conversa (id, idJogador, nome)
PK: id e idJogador
FK: {idJogador} de Jogador.id

Mensagem (id, idConversa, idJogador, texto, data, hora)
PK: id, idConversa e idJogador
FK: {idConversa} de Conversa.id e {idJogador} de Conversa.idJogador

Cracha (nome, nomeJogo, limiteDePontos, URL)
PK: nome e nomeJogo
FK: {nomeJogo} de Jogo.nome

Partida (id, nomeJogo, dataInicio, dataFim)
PK: id e nomeJogo
FK: {nomeJogo} de Jogo.nome

MultiJogador (idPartida, nomeJogo, Estado, nomeRegiao)
PK: idPartida e nomeJogo;
FK: {idPartida} de Partida.id , {nomeRegiao} de Regiao.nome e {nomeJogo} de Partida.nomeJogo

Normal (idPartida, nomeJogo, dificuldade, idJogador, Pontuacao)
PK: idPartida, nomeJogo
FK: {idPartida} de Partida.id , {idJogador} de Jogador.id e {nomeJogo} de Partida.nomeJogo

Jogar (idPartida, nomeJogo, Pontuacao)
PK: idPartida e nomeJogo
FK: {idPartida} de Partida.id e {idJogador} de Partida.nomeJogo

Comprar (idJogador, nomeJogo, preco, data)
PK: idJogador e nomeJogo
FK: {idJogador} de Jogador.id e {nomeJogo} de Jogo.nome

Tem (idJogador, nomeCracha, nomeJogo)
PK: idJogador, nomeCracha e nomeJogo
FK: {idJogador} de Jogador.id e {nomeCracha} de Cracha.nome e {nomeJogo} de Cracha.nomeJogo

Adicionar (idJogador, idJogadorAmigo)
PK: idJogador e idJogadorAmigo
FK: {idJogador} de Jogador.id e {idJogadorAmigo} de Jogador.id

EstatisticaJogador (idJogador, nrPartidas, nrJogos, totalPontosJogos)
PK: idJogador
FK: {idJogador} de Jogador.id

EstatisticaJogo (idJogador, nrPartidas, nrJogadores, totalPontos)
PK: idJogador
FK: {idJogador} de Jogador.id

Figura 8 - Modelo ER

3. Implementação das Funções, Procedures e Triggers

Neste capítulo, apresentaremos em detalhes a criação das funções, procedures e triggers desenvolvidos como parte deste projeto. Esses elementos desempenham um papel fundamental no sistema, permitindo a execução de tarefas específicas, o processamento de dados e a automação de determinadas ações.

Ao longo deste capítulo, explicaremos a lógica por trás de cada função, procedure e trigger criados. Também discutiremos as decisões tomadas durante o processo de desenvolvimento, destacando as melhores práticas utilizadas e os benefícios proporcionados por cada elemento implementado.

Durante o desenvolvimento do sistema, realizamos extensos testes para garantir a qualidade e a confiabilidade de todas as operações implementadas. Os testes foram

cuidadosamente planejados e executados, abrangendo uma variedade de cenários para verificar o correto funcionamento das funções, procedimentos e gatilhos.

Além disso, ao implementar as procedures, demos uma atenção especial ao nível de isolamento, visando garantir a consistência e a integridade dos dados durante as operações. Estudamos e aplicamos as melhores práticas de isolamento, considerando os requisitos específicos do sistema e as necessidades de transações concorrentes.

3.1 Funções

A criação das funções foi realizada com base no que nos foi pedido no enunciado. Exploraremos os passos envolvidos na criação de cada função, incluindo a definição dos parâmetros, o processamento dos dados e a lógica implementada.

3.1.1 totalPontosJogador

A função "totalPontosJogador" é uma função implementada no sistema para calcular os pontos totais que o jogador obteve em todas as partidas. A função recebe como parâmetro o identificar do jogador e retorna o número correspondente á soma de todos os pontos que o jogador fez nas suas partidas.

Esta função começa por verificar se o jogo com o identificar fornecido existe na tabela "Jogador". Caso não exista é lançado uma exceção que indica que o jogador não existe. Caso exista a função faz uma consulta á tabela "Jogar", que é onde estão presentes as partidas multi-jogador com a sua pontuação, para obter a pontuação que obteve nas partidas multi-jogador que participou. Caso o jogador não tenha participado em nenhuma partida multi-jogador é usado a função "coalesce", que tem como parâmetros o valor obtido da tabela e o valor caso o valor da tabela retorne null, ou seja, caso o jogador não tenha participado em partidas multi-jogador é retornado '0', pois o valor obtido da tabela "Jogar" é null. Depois é feito uma consulta á tabela "Normal", que é onde estão presentes as partidas normais com a sua pontuação, para obter a pontuação das partidas normais em que o jogador participou, é usado os mesmos mecanismos de verificação, isto é, também é usada a função "coalesce" nesta pesquisa caso o jogador não tenha realizado nenhuma partida normal. O resultado desta função será a soma dos valores obtidos das duas pesquisas que é o total de pontos que o jogador obteve.

4.1.2 totalJogosJogador

Função “totalJogosJogador” serve para calcular no sistema, o número total de diferentes jogos jogado pelo o jogador selecionado. A função recebe como parâmetro o ID do jogador e retorna um inteiro que contém o número total de jogos diferentes que o jogador participou.

Função “totalJogosJogador” declara um inteiro “totalJogos” que é usado para armazenar o número total de jogos diferentes que o jogador jogou e o mesmo é o retorno da função. No início da função é verificado se o parâmetro ID do jogador existe na tabela Jogador, se não existe lança uma exceção com a informação que o ID inserido não existe. A seguir, é feita uma consulta para contar o número de jogos diferentes que o jogador com o ID referido participou a partir das tabelas “Jogar” e “Normal”, para isso é feito um COUNT no atributo “nomeJogo” mas com valores diferentes, ao seja, é usada a declaração DISTINCT. A contagem referida é atribuída para o inteiro “totalJogos”. Para consultar o atributo “nomeJogo” das tabelas “Jogar” e “Normal” que tenham o ID do jogador coicidente, usa-se a declaração INNER JOIN. Por fim é retornado o número total de jogos diferentes que o jogador selecionado participou.

Para testar a função, pode-se utilizar o seguinte comando:

Select totalJogosJogador(3);

Isso retornará o número total de jogos diferentes que o jogador de ID 3 jogou.

4.1.3 PontosJogoPorJogador

A função "PontosJogoPorJogador" é uma função implementada no sistema para calcular os pontos totais dos vários jogadores em um determinado jogo. A função recebe como parâmetro o nome do jogo e retorna uma tabela contendo o ID de cada jogador e o total de pontos obtidos por ele nesse jogo.

Esta função inicia verificando se o jogo com o nome fornecido existe na tabela "Jogo". Caso não exista, a função gera uma exceção informando que o jogo não existe. Em seguida, a função realiza uma consulta para obter os jogadores e suas pontuações da tabela "Normal" para o jogo especificado. Em seguida, utiliza a cláusula "UNION ALL" para unir os resultados com outra consulta que obtém os jogadores e suas pontuações da tabela "Jogar" para as partidas associadas ao jogo, isto assegura que obtenho os pontos tanto dos jogos Normais como os de MultiJogador pois a tabela “Jogar” contém a pontuação destes. O resultado final é retornado como uma tabela contendo o ID do jogador e o total de pontos.

Para testar a função, pode-se utilizar o seguinte comando:

select * from PontosJogoPorJogador('SpaceInv');

Isso retornará os jogadores e seus respectivos totais de pontos no jogo "SpaceInv".

4.2 Procedures

Discutiremos a criação dos procedures, que são blocos de código SQL que podem ser executados em conjunto para realizar uma determinada operação. As procedures fornecem uma forma estruturada de realizar ações complexas, facilitando o desenvolvimento e a manutenção do sistema. Descreveremos as etapas para criação de cada procedure, destacando sua finalidade e os resultados esperados.

4.2.1 criarJogador

O procedimento de armezenamento "criarJogador" cria o mecanismo de criar novos jogadores na tabela "Jogadores" dados os seus email, região e username.

O procedimento recebe como parâmetros o nome do jogador: "nomeJogador", email do jogador: "emailJogador" e a região do jogador: "regiaoJogador".

O início do procedimento faz três verificações. Primeiro verifica se o nome do Jogador já existe, se sim lança execeção, segundo verifica se o email do jogador já existe, se sim lança exceção e por último verifica se a região existe, se não existe é criada uma nova região na tabela Regiao.

Depois de todas as verificações forem válidas é feita uma inserção na tabela Jogador um novo jogador com os valores dos atributos: "nomeJogador", "emailJogador" e "regiaoJogador".

Para testar o procedimento, pode-se usar a seguinte instrução:

CALL criarJogador('Albertina', 'albertinajosefina27@gmail.com', 'Madeira');

4.2.2 desativarJogador

O procedimento de armezenamento "desativarJogador" cria o mecanismo para modificar o estado do jogador para 'inativo'.

O procedimento recebe como parâmetros o ID do jogador: "jogadorId".

No início do procedimento define o nível de isolamento de transação como "repeatable read".
A seguir faz duas verificações que são: verificar se o parâmetro "jogadorId" que refere ao ID do jogador existe ou não, se não existir lança exceção e verifica se o jogador referido já está no estado 'inativo' ou não, se sim lança exceção.

Depois de todas as verificações forem válidas é feito um update da tabela Jogador em que modifica o atributo estado para um ‘varchar’ ‘inativo’ só para a linha que contém o ID do Jogador “jogadorId”.

Para testar o procedimento, pode-se usar a seguinte instrução:

CALL desativarJogador(1);

4.2.3 banirJogador

O procedimento de armezenamento “banirJogador” cria o mecanismo para modificar o estado do jogador para ‘banido’.

O procedimento recebe como parâmetros o ID do jogador: “jogadorId”.

No início do procedimento define o nível de isolamento de transação como "repeatable read".
A seguir faz duas verificações que são: verificar se o parâmetro “jogadorId” que refere ao ID do jogador existe ou não, se não existir lança exceção e verifica se o jogador referido já está no estado ‘banido’ ou não, se sim lança exceção.

Depois de todas as verificações forem válidas é feito um update da tabela Jogador em que modifica o atributo estado para um ‘varchar’ ‘banido’ só para a linha que contém o ID do Jogador “jogadorId”.

Para testar o procedimento, pode-se usar a seguinte instrução:

CALL banirJogador(2);

4.2.4 associarCracha

O procedimento "associarCracha" é responsável por associar um crachá a um jogador num determinado jogo. O procedimento recebe como parâmetros o ID do jogador, o ID do jogo e o nome do crachá.

O procedimento inicia definindo o nível de isolamento da transação como "repeatable read". Em seguida, verifica se o jogador com o ID fornecido existe na tabela "Jogador" e se o crachá com o nome fornecido existe na tabela "Cracha".

Além disso, verifica se o jogo com o ID fornecido existe na tabela "Jogo". A partir daí, o procedimento realiza uma consulta para obter o limite de pontos do crachá correspondente ao nome fornecido. Em seguida, obtém o nome do jogo correspondente ao ID fornecido para o poder passar à função “PontosJogoPorJogador”.

O procedimento verifica se o total de pontos do jogador no jogo, obtido através da função "PontosJogoPorJogador", é menor que o limite de pontos do crachá. Se for menor, gera uma exceção informando que o jogador não tem pontos suficientes para obter o crachá.

Caso contrário, o procedimento realiza a inserção na tabela "Tem" para associar o crachá ao jogador no jogo correspondente. Um aviso é emitido indicando que o crachá foi atribuído. Se ocorrer alguma exceção durante o processo de inserção, ela é tratada e o procedimento é finalizado.

O nível de isolamento “repeatable read” foi escolhido para garantir que o tuplo da tabela “Jogo” que fosse verificado existir na primeira verificação não fosse eliminado enquanto a transação estivesse a decorrer e mais tarde não existisse quando quisessemos ir ler o nome dele.

Para utilizar o procedimento, pode-se utilizar o seguinte comando:

CALL associarCracha(1, '7', 'Cracha’);

4.2.5 iniciarConversa

O procedimento “iniciarConversa” é responsável por iniciar uma conversa associando a essa conversa o jogador que a criou. O procedimento recebe como parâmetros o identificador do jogador e o nome da conversa e tem um parâmetro de saída que é o identificador gerado para essa conversa.

O procedimento foi implementado através de outros dois procedimentos distintos, “iniciarConversaLogic” e “iniciarConversaTrans”. O procedimento principal “iniciarConversa” chama a procedimento “iniciarConversaTrans”, que é o procedimento transacional que chama a e que gere os erros gerados pela lógica, “iniciarConversaLogic”. O procedimento “iniciarConversaLogic” é o procedimento onde está presente a logica do procedimento, este começa por verificar se o jogador com o identificar fornecido existe na tabela “Jogador”. Caso não exista, o procedimento gera uma exceção informando que o jogador não existe. Caso exista, o procedimento faz uma inserção na tabela “Conversa” e coloca no parâmetro de retorno o identificador gera para a conversa.

O nível de isolamento….

4.2.6 juntarConversa

O procedimento "juntarConversa" é responsável por juntar um jogador a uma conversa que já existe. O procedimento recebe como parâmetros o identificar do jogador que quero adicionar á conversa e o identificador da conversa.

O procedimento foi implementado através de outros dois procedimentos distintos, "juntarConversaLogic" e "juntarConversaTrans". O procedimento principal "juntarConversa" chama a procedimento "juntarConversaTrans", que é o procedimento transacional que chama a e que gere os erros gerados pela lógica, "juntarConversaLogic". O procedimento "juntarConversaLogic" é o procedimento onde está presente a logica do procedimento, este começa por verificar se o identificador fornecido pertence a um jogador existente na tabela "Jogador". Caso não pertença, o procedimento gera uma exceção indicando que o jogador não existe. Caso pertença, é feito uma pesquisa na tabela "Conversa" para verificar se existe alguma conversa com o identificador dado. Caso não exista, é lançado uma exceção que informa o utilizador que a conversa não existe. Caso exista, faz uma inserção na tabela conversa de forma a associar o jogador á conversa.

O nível de isolamento "repeatable read" foi escolhido para garantir que o tuplo nome da tabela "Conversa" fosse verificado antes de fazer a inserção na tabela pois até ao momento de inserção a conversa pode ser eliminada, e nesse caso não adicionamos o jogador á conversa.

4.2.7 enviarMensagem

O procedimento de armezenamento "enviarMensagem" cria o mecanismo para enviar uma mensagem escrita de um jogador a uma conversa indicada.

O procedimento recebe como parâmetros os identificadores da mensagem: "conversaId", do jogador: "jogadorId" e o texto da mensagem: "textoMensagem".

No início do procedimento faz três verificações que são: verificar se o parâmetro "jogadorId" que refere ao ID do jogador existe ou não, se não existir lança exceção, verifica se o parâmetro "conversaId" que refere ao ID da conversa existe ou não, se não existir lança exceção e verifica se a conversa referida contém o jogador referido, se contém, então é feita a inserção na Tabela Conversa uma nova conversa que contém os valores dos atributos: "conversaId", "jogadorId" e "textoMensagem", se não contém é lançada uma exceção.

Para testar o procedimento, pode-se usar a seguinte instrução:

CALL enviarMensagem(2, 5, 'Ola');

4.3 Triggers

Aqui abordaremos a implementação dos triggers, que são acionadores automáticos que respondem a eventos específicos no banco de dados. Os triggers permitem a execução de ações predefinidas quando determinadas operações são realizadas nas tabelas, como inserção, atualização ou exclusão de registros.

4.3.1 atribuicaoCracha

O trigger “atribuicaoCracha” é responsável por acionar o procedimento “associarCracha” quando é feita uma alteração no tuplo estado para o valor ‘Terminada’ ou quando é feita uma inserção na tabela “Normal”. Este trigger é acionado após as alterações das tabelas, usando a cláusula AFTER UPDATE e AFTER INSERT.

A função “atribuicaoCracha” é uma função que retorna um trigger, esta começa por verificar se a operação que a acionou é um UPDATE ou um INSERT. Se não for, é lançada uma exceção indicando que o gatilho é inválido. Em seguida é feita uma pesquisa na tabela “Jogo” para obter o identificador do jogo a partir do nome NEW.NOMEJOGO, pois este é um dos parâmetros do procedimento “associarCracha”, isto é possível por causa da cláusula FOR EACH ROW que nos dá acesso á palavra-chave NEW que indica qual é o valor da linha que acionou o gatilho.

4.3.2 banirJogadores

O trigger "banirJogadores" é responsável por acionar o procedimento "banirJogador" quando uma linha é deletada da view "jogadorTotalInfo". Esse trigger é acionada em vez da operação de exclusão default da view (INSTEAD OF Trigger).

A função "banirJogadores" é uma função do tipo trigger que recebe a ação do trigger. Ela verifica se a operação que acionou o trigger é uma operação de exclusão (DELETE). Se não for, é lançada uma exceção informando que o gatilho é inválido. Em seguida, o procedimento "banirJogador" é chamado passando o ID do jogador que está sendo excluído, usando “old” keyword. Esse procedimento é responsável por realizar as ações necessárias para banir o jogador.

Após a chamada do procedimento, é emitido um aviso indicando que o gatilho foi acionado. Por fim, a função retorna null.

A trigger "banirJogadores" é criada utilizando o comando "CREATE TRIGGER". Ela é definida como "INSTEAD OF DELETE", ou seja, será acionada em vez da operação de exclusão default na view "jogadorTotalInfo". A cada linha deletada, a trigger executa o procedimento "banirJogadores".

4.4 View

Nesta secção, abordaremos a implementação da view, que é uma tabela virtual que permite referenciar informação que é manipulada frequentemente ou para fornecer perspetivas de dados a utilizadores. Os tuplos de uma view não existem fisicamente na Base de Dados e não armazena dados como as tabelas normais fazem.

4.4.1 jogadorTotalInfo

A view "jogadorTotalInfo" cria uma tabela virtual que permita aceder à informação sobre identificador, estado, email, username, número total de jogos em que participou, número total de partidas em que participou e número total de pontos que já obteve de todos os jogadores cujo estado seja diferente de "Banido". Os cálculos número total de jogos em que o jogador participou é feita a partir da chamada da função totalJogosJogador, número total de partidas em que o jogador participou é feita ao chamar a função totalPartidasJogador e número total de pontos que já obteve de todos os jogadores é chamada a função totalPontosJogador. Para obtenção de todos os dados referidos o estado do jogador não pode ser 'Banido'.

5. Conclusões

Em conclusão, este relatório representa o resultado de um esforço dedicado à implementação de um sistema de gestão de jogos. Através da análise cuidadosa dos requisitos, modelagem eficiente de entidades e relações, criação de funções e procedimentos robustos e adoção de níveis adequados de isolamento, conseguimos desenvolver uma solução que integra dados de jogadores, partidas, jogos e estatísticas de forma coesa e consistente.

Ao longo desse processo, enfrentamos desafios e tomamos decisões fundamentais que impactaram diretamente na eficiência e segurança do sistema. O cuidado na escolha dos níveis de isolamento, considerando as características de cada operação e garantindo a consistência dos dados, foi um dos aspectos cruciais para o bom funcionamento do sistema.

Além disso, a implementação de funções e procedimentos bem estruturados proporcionou uma melhor organização do código e facilitou a realização de operações complexas de forma simplificada. Através dos testes realizados, verificamos a eficácia e integridade das funcionalidades desenvolvidas.

Este relatório representa não apenas um registro documentado de todo o trabalho realizado, mas também uma demonstração do comprometimento em buscar soluções eficientes e seguras para o gerenciamento de jogos. Em última análise, este projeto reflete a importância da análise cuidadosa dos requisitos, o planejamento adequado da estrutura do banco de dados, a implementação de funcionalidades robustas e a preocupação constante com a qualidade e integridade dos dados.