Trabalho Fase 1 – Projeto “GameOn”

Vasco Branco – 48259
João Pereira – 48264
Tiago Neves – 48292

Orientadores: Prof. Walter Vieira

Relatório final realizado no âmbito de Sistemas de Informação,
do curso de licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2022/2023

Maio de 2013

z

Resumo

Neste trabalho, o grupo recebeu um texto como enunciado e é desafiado a criar um modelo entidade-associação com base nesse texto. Esse modelo permite identificar as entidades presentes no texto e estabelecer as associações entre elas.

Em seguida, o grupo teve que utilizar o modelo entidade-associação criado para desenvolver um modelo relacional. Esse modelo relacional representa as entidades, seus atributos e os relacionamentos entre elas em um formato que pode ser implementado em um banco de dados relacional.

Além disso, o grupo precisa de utilizar a uma extensão para programação da linguagem SQL, denominada por PLPGSQL, para implementar diversas functions, procedures, views e triggers. Esses elementos adicionam funcionalidades ao banco de dados e permitem realizar operações complexas e automatizadas.

Durante o desenvolvimento desses componentes, o grupo aplicou os corretos níveis de isolamento, garantindo a integridade dos dados e evitando conflitos de acesso simultâneo. Além disso, foram aplicados os devidos tratamentos de erros para lidar com situações inesperadas e manter a consistência dos dados.

Em resumo, este trabalho envolve a criação de um modelo entidade-associação a partir de um texto, a tradução desse modelo para um modelo relacional, e a implementação de functions, procedures, views e triggers utilizando a extensão para programação da linguagem sql, com atenção aos níveis de isolamento e tratamento de erros.

Palavras-chave:

- **Extensão** PLPGSQL**;**
- **Function;**
- **Modelo entidade-associação;**
- **Modelo relacional;**
- **Níveis de isolamento;**
- **Procedures;**
- **Tratamento de erros;**
- **Triggers;**
- **Views:**

Introdução

O Capítulo 1 do relatório aborda a etapa de criação do banco de dados, que é essencial no processo de desenvolvimento de sistemas. Neste capítulo, serão apresentados os dois subcapítulos que abordam os principais modelos utilizados nesse processo: o Modelo Entidade-Associação e o Modelo Relacional. Esses modelos fornecem uma representação estruturada das entidades, atributos e relacionamentos presentes no sistema, permitindo uma melhor compreensão e organização dos dados.

O Capítulo 2 do relatório aborda o problema submetido para resolução na sua totalidade.

O Capítulo 3 do relatório aborda a etapa de desenvolvimento dos mecanismos e processos automáticos que solucionam todas as funcionalidades requeridas no enunciado.

O Capítulo 4 do relatório aborda a avaliação experimental das soluções implementadas.

O Capítulo 5 do relatório aborda a conclusão de todos os conceitos, processos e ferramentas utilizados no auxílio da construção do projeto.

1. Modelos

1.1. Modelo Entidade-Associação

O modelo entidade-associação é uma ferramenta utilizada para representar as entidades presentes em um sistema, bem como os relacionamentos entre elas. Neste subcapítulo, será abordada a criação do modelo entidade-associação com base no texto fornecido como enunciado do trabalho.

O processo de criação do modelo entidade-associação envolve a identificação das entidades presentes no texto e a definição dos relacionamentos entre elas. As entidades representam objetos ou conceitos do mundo real, enquanto os relacionamentos representam as associações entre essas entidades.

Após identificar as entidades, é importante estabelecer os relacionamentos entre elas. Os relacionamentos podem possuir diferentes cardinais, como "um para um", "um para muitos" ou "muitos para muitos" ou até conterem obrigatoriedade, sendo esta representada com duas linhas paralelas entre si. Esses relacionamentos são representados por meio de linhas que conectam as entidades no diagrama.

Além disso, é necessário definir os atributos das entidades, que são características ou propriedades que descrevem cada entidade.

O modelo entidade-associação fornece uma representação visual das entidades, seus atributos e os relacionamentos entre elas, auxiliando na compreensão e na estruturação do banco de dados.

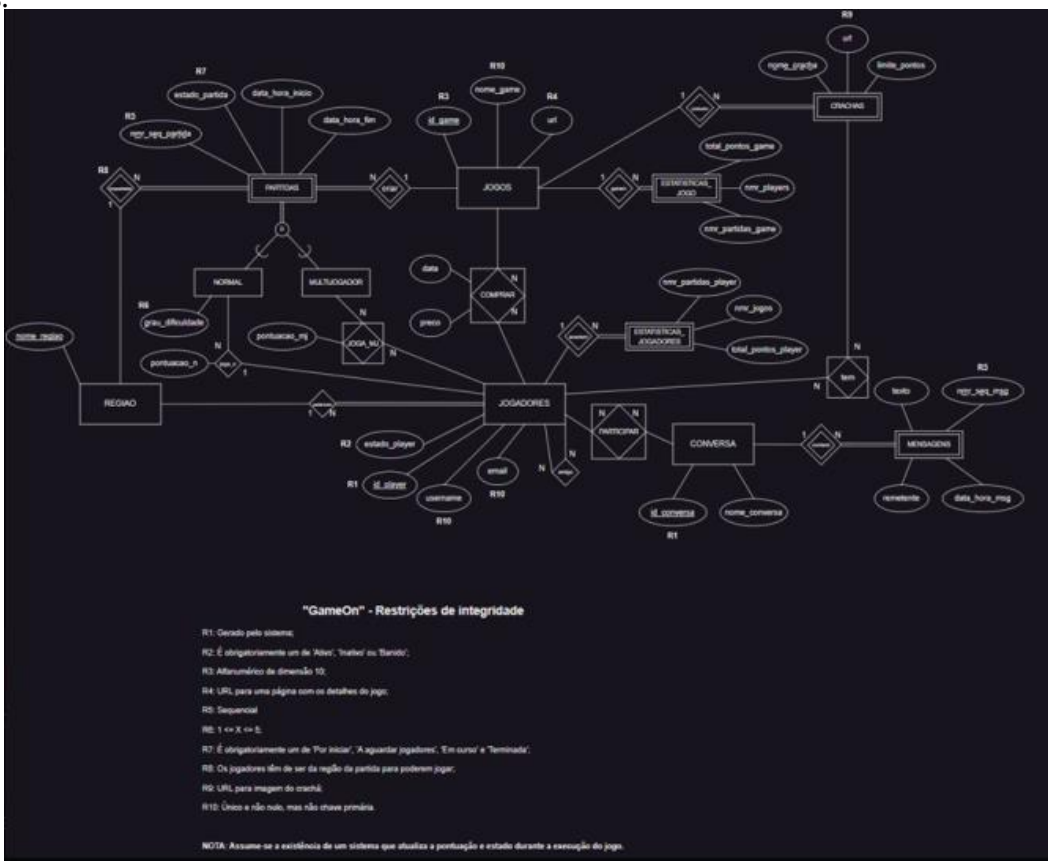

Figura 1 - Modelo EA

1.2. Modelo Racional

Modelo Relacional, que é uma representação estruturada das entidades, atributos e relacionamentos do sistema em formato de tabelas. Será explorado o processo de tradução do Modelo Entidade-Associação para o Modelo Relacional, onde as entidades se tornam tabelas, os atributos se tornam colunas e os relacionamentos são representados por meio de chaves primárias e estrangeiras. Assim, o modelo relacional foi construído com base nas seguintes regras:

- Passo 1: Entidade E 1. Entidade E → Relação T 2. Attrs(E) simples → Attrs(T) 3. Attrs(E) simples de 1 atributo composto → Attrs(T) 4. Atributo-chave de E → PK(T)
  Resumo: Criar as tabelas que não são fracas juntamente com os respetivos atributos.

- Passo 2: Entidade fraca W com entidade identificadora E 1. T relação que representa E 2. Entidade W → Relação U 3. Atributos simples/compostos de W → Attrs(U) (2 e 3 de Passo 1) 4. PK(T) → FK(U) 5. PK(T) + Atributos da chave parcial de W → PK(U)
  Resumo: Criar as tabelas fracas, juntamente com os seus atributos, colocando como foreign key e primary key a primary key da entidade não fraca.

- Passo 3A: (chave estrangeira) Associação-tipo 1:1 de R(E1,E2) (Mais utilizada, com exceção de condições especiais, reduz o no. de tabelas) 1. T1 e T2 são as relações que representam E1 e E2 2. Escolhemos uma das relações, e.g. T2 3. Se uma das entidades (e.g. E2) tem participação total em R, por regra essa é a relação (e.g. T2) 4. Atributos simples/compostos de R → Attrs(E2) (2 e 3 de Passo 1) 5. PK(T1) → FK(T2)
  Resumo: Escolher uma associação com cardinalidade 1:1, escolher qualquer uma das entidades participantes e consecutivamente criar essa mesma tabela que foi escolhida juntamente com os seus atributos, colocando como foreign key a primary key da entidade que não foi escolhida, caso a relação tenha algum atributo esse mesmo atributo também deve ser incluído na criação da entidade escolhida.

- Passo 4A: (chave estrangeira) Associação-tipo 1:N de R(E1,E2) (Mais utilizada, com exceção de condições especiais, reduz o no. de tabelas) 1. T1 e T2 são as relações que representam E1 e E2 2. E1 tem cardinalidade N na associação → Relação T1 3. Atributos simples/compostos de R → Attrs(T1) (2 e 3 de Passo 1) 4. PK(T2) → FK(T1), ex: cada instância de E2 está associada no máximo a 1 instância de E1
  Resumo: Escolher uma associação com cardinalidade 1:N, escolher a entidade que possui o cardinal N nessa mesma relação, criar essa mesma tabela que foi escolhida juntamente com os seus atributos, colocando como foreign key a primary key da entidade com o cardinal 1.

- Passo 5: Associação-tipo N:N de R(E1,E2) 1. T1 e T2 são as relações que representam E1 e E2 2. R → Relação S 3. Atributos simples/compostos de R → Attrs(S) (2 e 3 de Passo 1) 4. PK(T1) e PK(T2) → FK(S) 5. PK(T1) + PK(T2) → PK(S)

  Resumo: Escolher uma associação com cardinalidade N:N, criar uma tabela para a relação em si, juntamente com os seus atributos, colocando como primary key e como foreign key as primary keys das entidades envolvidas nessa mesma relação.

2. Formulação do Problema

Neste capítulo explica-se o problema/desafio proposto para a primeira fase do projeto da disciplina de Sistemas de Informação.

Como primeira fase do projeto, foi no pedido que construíssemos um sistema para a gestão de jogos, utilizadores e partidas de uma empresa: "*A empresa "GameOn" pretende desenvolver um sistema para gerir jogos, jogadores e as partidas que estes efetuam.*".

2.1. Análise do problema

Para atingir tal objetivo tivemos de realizar diversos ficheiros (scripts) com a utilização de várias ferramentas (views, functions, procedures, etc.), ficheiros esses que serão o alvo de avaliação.

Também realizámos os devidos modelos de dados, modelo EA e modelo ER anteriormente explicados, de forma a auxiliar na organização da solução por nós apresentada.

3. Solução Proposta-Scripts

Nesta secção, é apresentado todos os mecanismos e processos que o grupo teve de efetuar para solucionar os problemas dispostos no enunciado, desde da alínea a) até à alínea o).

Estas soluções foram construídas à base de instruções "SQL" e da extensão para programar em sql denominada por "plpgsql".

Sendo mais específico, todas as soluções da modelagem do modelo físico consistem em instruções de criação, remoção e alteração de tabelas. Já a lógica de implementação dos mecanismos é baseada em procedures, functions, view, e triggers.

Deste modo, foi possível automatizar todas as soluções dispostas no trabalho efetuado pelo grupo.

3.1. Conjunto de Scripts

Observação: Para todas as funções e procedimentos realizamos um tratamento de dados, mais concretamente, dos dados fornecidos como parâmetros quando estes existem.

a) **Criar o modelo físico (1 script autónomo):**

Através da instrução “create table” é possível gerar uma tabela para uma base de dados relacional.
A sintaxe simples desta instrução é: create table table_name (parameter_name parameter_type);
É também importante de realçar que nesta instrução é possível definir as primary e foreign keys da mesma tabela , constraints e colocar flags nos valores, como por exemplo “not null”. Com esta instrução foi possível criar o modelo físico num único script autónomo.

b) Remover o modelo físico (1 script autónomo):

Através da instrução “drop” é possível remover uma tabela da base de dados relacional.
A sintaxe simples desta instrução é: drop table_name;
É também importante de realçar que nesta instrução é possível colocar “cascade” que permite remover não só a tabela em questão como também mas também todos os objetos dependentes dela, como restrições de chave estrangeira, visualizações ou triggers. Outra funcionalidade utilizada foi a cláusula “if exists”, caso a tabela exista o comando é executado. Com esta instrução foi possível remover o modelo físico num único script autónomo.

c) Preenchimento inicial da base de dados (1 script autónomo):

Através da instrução “insert into” é possível gerar uma tabela para uma base de dados relacional.
A sintaxe simples desta instrução é: insert into table_name values(parameter_name).
Com esta instrução foi possível "povoar“ o modelo físico num único script autónomo.

d) Criar os mecanismos que permitam criar o jogador, dados os seus email, região e username, desativar e banir o jogador:

Para criar um mecanismo que implementa-se a funcionalidade de criar um jogador, recorreu-se à criação de um procedure, através do comando “create or replace procedure criar_jogador(...)”, passando como parâmetro o email, username e nome_regiao, sendo todos os dados do tipo varchar. Dentro deste procedimento, verifica-se se o email ou o username já existem na tabela “JOGADORES” se esta condição for verdadeira então é executado a instrução “raise exception”. Caso Contrário o jogador é inserido na tabela com os respetivos dados passados como parâmetro através do comando “insert into(...) values(...)”.
Já para desativar e banir o jogador foi também criado procedures mas que só recebem um único parâmetro sendo este o id do jogador, sendo este um int. Ambos também realizam uma verificação, através doa instrução “if exists” para ver se o jogador existe, se o jogador não existir é executado o comando “raise exception”, caso contrário é feito um update na coluna “estado_player” no registo que corresponder ao id do jogador, isto na tabela de “JOGADORES”.
No caso de desativar o jogador é colocado o valor ‘Inativo’ e no caso de banir o jogador é colocado o valor ‘Banido’.

e) Criar, sem usar as tabelas de estatísticas, a função totalPontosJogador que recebe como parâmetro o identificador de um jogador e devolve o número total de pontos obtidos pelo jogador

Para criar uma função que implementa a funcionalidade de devolver o número total de pontos obtidos pelo jogador, recorreu-se à criação de uma function, através do comando "create or replace function totalPontosJogador(...), que recebe como parâmetro o id do jogador, sendo este um int. Dentro desta function, verifica-se se o id passado como parâmetro existe dentro na tabela de "JOGADORES" se não existir é executado o comando "raise exception", caso contrário é efetuada uma leitura da tabela "NORMAL" , para esse mesmo id de jogador passado como parâmetro, que é "agregada" com o comando "union all" à leitura da tabela "JOGA_MJ", sendo ainda feita uma leitura posterior que conta o número de id's distintos presentes na tabela e soma o número de pontos total desse mesmo jogador.

Depois é também efetuada uma verificação para confirmar se o id do player passado como parâmetro não existe na tabela de "ESTATISTICAS_JOGADORES", se a condição for verdadeira, então essa mesma tabela é povoada com os pontos do jogador.

A função retorna os pontos do jogador passado como parâmetro.

f) Criar, sem usar as tabelas de estatísticas, a função totalJogosJogador que recebe como parâmetro o identificador de um jogador e devolve o número total de jogos diferentes nos quais o jogador participou:

Para realizar esta alínea recorremos à criação de uma função, onde realizamos a associação de todos os jogos jogados por um jogador por meio do identificador desse mesmo jogador, retornando a contagem do total de jogos obtidos.

Realizamos também a verificação do identificador de jogador recebido como parâmetro da função, assegurando que o jogador existe na tabela "JOGADORES".

g) Criar a função PontosJogoPorJogador que recebe como parâmetro a referência de um jogo e devolve uma tabela com duas colunas (identificador de jogador, total de pontos) em que cada linha contém o identificador de um jogador e o total de pontos que esse jogador teve nesse jogo. Apenas devem ser devolvidos os jogadores que tenham jogado o jogo:

Em primeiro lugar, na função realizada, verificamos se a referência de jogo passada como parâmetro existe na tabela "JOGOS". Em seguida, procedemos à criação de uma tabela temporária e para preencher essa tabela realizamos uma união entre todos os jogadores presentes nas tabelas "NORMAL" e "JOGA_MJ" que tenham a referência de jogo associada a eles, inserindo nessa tabela o seu identificador de jogador e a soma de pontos de ambas as tabelas, ou seja, o total de pontos que obteve no jogo, tanto em jogo normal (single-player) como em jogo multijogador (multi-player).

Nesta função também realizamos o preenchimento da tabela “ESTATISTICAS_JOGO”.

h) Criar o procedimento armazenado associarCrachá que recebe como parâmetros o identificador de um jogador, a referência de um jogo e o nome de um crachá desse jogo e atribui o crachá a esse jogador se ele reunir as condições para o obter:

Para criar um procedimento armazenado que implemente a funcionalidade de atribuir um crachá ao jogador que joga um determinado jogo se o mesmo tiver os pontos suficientes, recorreu-se à criação de um procedure.

Primeiramente, verificamos se o id do jogador, passado como parâmetro (do tipo “int”), existe na tabela de “JOGADORES”, o id do jogo, passado como parâmetro (do tipo “int”), existe na tabela de “JOGOS” e por último se o nome do crachá, passado como parâmetro (do tipo “varchar”), existe na tabela de “CRACHAS”.

De seguida, são realizadas diversas leituras, lê-se da tabela “NORMAL” nos registos que apresentem um id igual ao do jogador em questão e “agrega-se” a tabela recolhida dessa leitura, através da clásula “union all”, com uma outra leitura da tabela “JOGA_MJ”, novamente onde os registos apresentem um id igual ao do jogador em questão.

Da tabela resultade da agregação das duas leituras previamente realizadas, retira-se o o id dos jogadores, os nomes das regiões e a soma de todos os seus pontos no jogo em questão, que são guardados numa variável com o nome “totalpontos”.

Por fim, verifica-se se os pontos do jogador são maiores ou iguais ao número de pontos necessários para adquirir o crachá, se esta condição verificar-se verdadeira, então o id do jogador juntamente com o nome do cracha, id do jogo e o nome da região serão inseridos na tabela “TEM”, que justamente representa a associação entre um jogador e os respetivos crachás.

Caso contrário, o comando “insert into” não será realizado.

No final deste procedure a tabela “nova_tabela” é removida, pois não terá mais nenhuma utilidade.

i) Criar o procedimento armazenado iniciarConversa iniciar uma conversa (chat) dados o identificador de um jogador e o nome da conversa. O jogador deve ficar automaticamente associado à conversa. O procedimento deve devolver num parâmetro de saída o identificador da conversa criada:

Criar o procedimento armazenado iniciarConversa iniciar uma conversa (chat) dados o identificador de um jogador e o nome da conversa. O jogador deve ficar automaticamente associado à conversa. O procedimento deve devolver num parâmetro de saída o identificador da conversa criada.

Para criar um mecanismo que implemente a funcionalidade de criar uma conversa com o id do jogador e o nome da conversa e posteriormente associar o jogador a essa mesma conversa, recorreu-se à criação de um procedure que recebe como parâmetro um “id_jogador” do tipo “int”, “nome_chat” do tipo “varchar”, e “out_conv” do tipo “int”.

Primeiramente, verifica-se seo id do jogador passado como parâmetro existe na tabela "JOGADORES".

De seguida, insere-se através do comando "insert into" na tabela "CONVERSAS" uma nova conversa com o nome passado como parâmetro.

Consequentemente, realiza-se uma leitura na tabela "CONVERSAS" para se obter o id da nova conversa posteriormente criada e o seu nome, também se efetua outra leitura na tabela "JOGADORES" para se obter os restantes dados do jogador passado como parâmetro.

Por fim, inserimos na tabela "CRIAR" o id do jogador, o nome da região e o id da conversa.

j) Criar o procedimento armazenado juntarConversa que recebe como parâmetros os identificadores de um jogador e de uma conversa e junta esse jogador a essa conversa:

Para criar um procedimento que implemente a funcionalidade de juntar o jogador à respetiva conversa, recorre-se à criação de um procedure que recebe como parâmetros um "id_jogador" do tipo "int" e um "id_conv" do tipo "int".

Primeiramente, verifica-se se o id da conversa está presente na tabela "CONVERSAS".

Consequentemente, é realizada uma leitura para recolher os dados do jogador.

Por fim, insere-se na tabela "CRIAR" o id do jogador, o nome da região e o id da conversa.

k) Criar o procedimento armazenado enviarMensagem que recebe como parâmetros os identificadores de um jogador e de uma conversa e o texto de uma mensagem e procede ao envio dessa mensagem para a conversa indicada, associando-a ao jogador também indicado.

Para criar um procedimento que implemente a funcionalidade de enviar uma mensagem para a conversa indicada, associando-a ao jogador também indicado, recorreu-se à criação de um "procedure" que recebe como parâmetros "id_jogador" do tipo "int", "id_conv" do tipo "int" e "msg" do tipo "varchar".

Primeiramente, verifica-se se a mensagem é igual ao valor "null", se o id do jogador existe na tabela "JOGADORES" e se o id da conversa existe na tabela "CONVERSAS".

Posteriormente, realiza-se uma leitura da tabela jogadores e guarda-se o valor na variável com o nome "v_nome_regiao".

Consequentemente verifica-se se existe o id do jogador na tabela "CRIAR", se não exisir é inserido nessa mesma tabela o id do jogador juntamente com o nome da região e o id da conversa.

Por fim, insere-se na tabela "MENSAGENS" o id da conversa, o número da mensagem, a mensagem em si, o id do jogador e o nome da região.

l) Criar a vista jogadorTotalInfo 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". Deve implementar na vista os cálculos sem aceder às tabelas de estatísticas.

Para criar uma vista que implemente a funcionalidade de aceder à informação sobre o 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", recorreu-se à "view".

É realizado uma leitura da tabela de jogadores, que é "agregada" à tabela "NORMAL" e também "agregada" à tabela "MULTIJOGADOR" onde o "estado_player" é diferente de "Banido".

A tabela resultante é agrupada pelo id dos jogadores.

m) Criar os mecanismos necessários para que, de forma automática, quando uma partida termina, se proceda à atribuição de crachás do jogo a que ela pertence.

Para criar mecanismos que implementem, de forma automática, quando uma partida termina, se proceda à atribuição de crachás do jogo a que ela pertence, recorreu-se à criação de triggers e de functions que os mesmos correm quando são activos, mais especificamente foram criados dois triggers, um para a tabela "MULTIJOGADOR" e outro para a tabela normal "NORMAL".

Primeiramente, é criado o trigger "associarCrachaMultijogador()", neste é especificado que é activo logo após ter sido executado o comando "update" na tabela "MULTIJOGADOR", isto é atingível a partir da cláusula "after update on multijogador", de seguida é também especificado que o trigger deve ser acionada para cada linha que foi afetada pelo evento que "disparou" o trigger, tal é possível através da clásula "for each row".

Por fim, é indicado que quando o trigger for acionadp irá executar a função desejada, neste caso é "associarCrachaMultijogador()", através da cláusula "execute function associarCrachaMultijogador".

A função tem com o objectivo associar o crachá respectivo ao jogador em questão.

Inicialmente, é feita uma verificação que acede às palavras-chave "old" e "new" (estas cláusulas que permitem o acesso aos antigos valores da coluna antes da atualização e também aos novos valores depois da atualização, respetivamente) e verifica se a antiga coluna "estado_partida" é diferente do novo valor dessa mesma coluna e se esse novo valor é equivalente a "Terminada", se tal condição se verificar verdadeira, então é realizado uma leitura da tabela "NORMAL" apenas nos registos que tenham sofrido uma alteração na coluna "estado_partida" para o valor "Terminado".

De seguida, recolhe-se o número de pontos que cada jogador pontuou num determinado jogo.

Por fim, é utilizado um ciclo "for" para obter cada crachá no jogo em questão e verificar se os pontos do jogador são iguais ou superiores aos pontos que o crachá requer, se esta condição

verificar-se verdadeira, então é inserido na tabela “TEM” os id do jogador, o nome do crachá, o id do jogo e o nome da região.

O mesmo processo é utilizado num outro trigger e numa outra função para a tabela “NORMAL” , pois não é possível colocar na cláusula de um só trigger que deve activar quando ocorre um update na tabela “A” ou na tabela “B”, assim o grupo optou por criar dois triggers diferentes e separar a lógica de ambas as tabelas.

n) Criar os mecanismos necessários para que a execução da instrução DELETE sobre a vista jogadorTotalInfo permita colocar os jogadores envolvidos no estado “Banido”.

Para criar mecanismos que implementem, de forma automática, quando ocorre o comando “DELETE” na view “jogadorTotalInfo”, que permitam colocar os jogadores envolvidos no estado “Banido”, recorreu-se à criação de triggers e de functions que os mesmos correm quando são activos.

Primeiramente, é criado o trigger “excluirVista r()”, neste é especificado que é activo quando é executado o comando “delete” na view “jogadorTotalInfo” (o comando delete não chega a ser executado efectivamente) , isto é atingível a partir da cláusula “instead of delete on jogadorTotalInfo”, de seguida é também especificado que o trigger deve ser acionada para cada linha que foi afetada pelo evento que “disparou” o trigger, tal é possível através da clásula “for each row”.

Por fim, é indicado que quando o trigger for acionadp irá executar a função desejada, neste caso é “excluirVista()”, através da cláusula “execute function excluirVista()”.

A função tem com o objectivo colocar os estado dos jogadores em questão equivalente a “Terminado”.

É executado o comando “update” na tabela de “JOGADORES” colocando a coluna “estado_player” igual ao valor “Terminado” para os id dos jogadores que se encontrem no retorno da view.

o) Criar um script autónomo com os testes das funcionalidades de 2d a 2n para cenários normais e de erro. Este script, ao ser executado, deve listar, para cada teste, o seu nome e indicação se ele correu ou não com sucesso;

Para criar os testes das alíneas 2d a 2n foi implementado um único script, indicando para cada alínea se correu com sucesso ou se falhou.

3.2. Niveis de Isolamento

Em SQL, os níveis de isolamento são mecanismos que controlam como as transações interagem entre si em um banco de dados relacional. Eles garantem a consistência e a integridade dos dados durante a execução de transações simultâneas.

Existem vários níveis de isolamento que podem ser configurados num banco de dados, sendo os mais comuns:

- Read Uncommitted: Neste nível, uma transação pode ler dados não confirmados de outras transações. Isso significa que uma transação pode ver alterações feitas por outras transações que ainda não foram confirmadas (ou revertidas). Esse nível oferece o menor grau de isolamento e não é recomendado para ambientes de produção, pois pode resultar em leituras inconsistentes.

- Read Committed: Neste nível, uma transação só pode ler dados que foram confirmados por outras transações. Isso evita leituras de dados não confirmados, mas ainda pode resultar em problemas de inconsistência, pois diferentes leituras dentro da mesma transação podem retornar resultados diferentes, caso outra transação tenha modificado os dados entre as leituras.

- Repeatable Read: Neste nível, uma transação garante que todas as leituras dentro da transação retornem os mesmos resultados, mesmo que outras transações estejam modificando os dados ao mesmo tempo. Isso é alcançado por meio do bloqueio de leitura exclusiva em todas as linhas e tabelas que estão sendo lidas pela transação. No entanto, esse nível ainda permite a ocorrência de anomalias conhecidas como "phantom read", onde uma transação vê novas linhas que foram inseridas por outras transações após o início da transação atual.

- Serializable: Esse é o nível mais alto de isolamento e garante que todas as transações ocorram como se fossem executadas sequencialmente, uma após a outra. Ele evita todas as anomalias de leitura, incluindo "phantom read". Isso é alcançado por meio de bloqueios de leitura e gravação exclusivos em todas as linhas e tabelas que são acessadas pela transação. No entanto, o nível de isolamento serializável pode resultar em bloqueios mais longos e pode afetar o desempenho em sistemas com grande concorrência.

Em suma, os níveis de isolamento permitem aos programadores controlar o equilíbrio entre consistência e desempenho, de uma base de dados relacional. Ao selecionar o nível adequado, obtem-se a segurança necessária para prevenir as eventuais anomalias de concorrência, prejudicando o mínimo possível a “performance” das transações.

O processo utilizado para a atribuição dos níveis de isolamento aos procedures, views e functions foi: Deteção dos possíveis erros de concorrência nos mecanismos em questão e verificar, com base, nos níveis de deteção previamente explicados, qual o nível mais adequado perante as anomalias que poderiam ocorrer.

d) Todos os procedimentos presentes nesta alínea possuem apenas uma ação de escrita, logo podemos concluir que a única anomalia possível é o "Dirty write".
   Para proteger os procedures desta anomalia é apenas necessário o nível "read uncommitted".

e) O procedimento presente nesta alínea possui leituras e escritas, uma das leituras é feita nos mesmos registos que a escrita, logo podemos concluir que as anomalias possíveis são "Dirty write" e "Dirty read".
   Para proteger o procedure desta anomalia é necessário o nível "read committed".

f) O procedimento presente nesta alínea apenas possui leituras, logo podemos concluir que não é possível existirem anomalias de concorrência.
   Para proteger o procedure é apenas necessário o nível "read uncommitted".

g) O procedimento presente nesta alínea apenas possui leituras, logo podemos concluir que não é possível existirem anomalias de concorrência.
   Para proteger o procedure é apenas necessário o nível "read uncommitted".

h) O procedimento presente nesta alínea possui leituras e uma escrita, no entanto nenhuma leitura é realizada no mesmo local que a escrita, logo podemos concluir que a única anomalia possível é "Dirty write".
   Para proteger o procedure é apenas necessário o nível "read uncommitted".

i) O procedimento presente nesta alínea possui leituras e escritas, uma das leituras é feita nos mesmos registos que a escrita, logo podemos concluir que as anomalias possíveis são "Dirty write" e "Dirty read".
   Para proteger o procedure desta anomalia é necessário o nível "read committed".

j) O procedimento presente nesta alínea possui uma leitura e uma escrita, no entanto nenhuma leitura é realizada no mesmo local que a escrita, logo podemos concluir que não é possível existirem anomalias de concorrência.
   Para proteger o procedure é apenas ncessário o nível "read uncommitted".

k) O procedimento presente nesta alínea possui leituras e escritas, uma das leituras é feita nos mesmos registos que a escrita, logo podemos concluir que as anomalias possíveis são "Dirty write" e "Dirty read".
   Para proteger o procedure desta anomalia é necessário o nível "read committed".

l) presente nesta alínea apenas possui uma leitura, logo podemos concluir que não é possível existirem anomalias de concorrência.
   Para proteger o procedure é apenas necessário o nível “read uncommitted”.

m) O procedimento presente nesta alínea possui leituras e escritas, no entanto nenhuma leitura é realizada no mesmo local que a escrita, logo podemos concluir que a anomalia possível é “Dirty write”.
   Para proteger o procedure é apenas ncessário o nível “read uncommitted”.

n) Os procedimentos presentes nesta alínea possuem apenas uma ação de escrita, logo podemos concluir que a única anomalia possível é o “Dirty write”.
   Para proteger os procedure desta anomalia é apenas necessário o nível “read uncommitted”.

4. Avaliação Experimental

Com a resolução das diferentes alíneas propostas, o grupo deparou-se com diversos erros de sintaxe e falhas de encadeamento lógico de comandos tanto na linguagem SQL como na extensão para programação da mesma plpgsql.
Com isto, foi possível adquirir experiência e conhecimento em relação aos tópicos discutidos em aula e postos à prova no trabalho prático.

5. Conclusões

A partir da resolução deste trabalho, o grupo conseguiu aprender os conteúdos lecionados nas aulas teóricas e práticas.
Desde a conceção dos modelos EA e Relacional até ao desenvolvimento dos scripts que implementam as funcionalidades descritas no enunciado do trabalho prático.

Em suma, acreditamos que as soluções propostas atingiram os resultados pretendidos e que os elementos do grupo desenvolveram melhores capacidades de entender, estruturar e desenvolver todos os conteúdos que dizem respeito ao trabalho em questão.