Conteúdo

Capítulo 1

Introdução

A empresa "GameOn" pretende desenvolver um sistema para gerir jogos, jogadores e as partidas que estes efetuam. O objetivo deste projeto é implementar uma solução baseada em bases de dados dinâmicas, adequada aos requisitos. Nesto contexto, serão adquiridas habilidades em várias áreas de bancos de dados, incluindo modelagem de dados, implementação de soluções de banco de dados, controle transacional, níveis de isolamento, uso de vistas, procedimentos armazenados, gatilhos e funções. Para isto iremos utilizar DBeaver como um SQL software client, postgreSQL para gerir a BD e plpgsql para programar estruturalmente as funcionalidades necessárias.

Capítulo 2

Requisitos do Sistema

2.1 Jogadores

Os requisitos relacionados aos jogadores são os seguintes:

- Cada jogador deve ser identificado por um ID gerado pelo sistema.
- O email e o username são valores únicos e obrigatórios para cada jogador.
- Os jogadores tomam um dos estados 'Ativo', 'Inativo' ou 'Banido'.
- Cada jogador deve estar associado a uma região específica.

2.2 Regiões

Os requisitos relacionados às regiões são os seguintes:

- Cada região deve ser identificada por um nome único.
- Os jogadores devem ser associados às suas respetivas regiões, conforme mencionado na seção 2.1.
- Cada partida está associada a uma região e apenas os jogadores dessa região a podem jogar.

2.3 Jogos

Os requisitos relacionados aos jogos são os seguintes:

- Cada jogo deve ter como identificador uma referência alfanumérica de dimensão 10.
- O nome do jogo é obrigatório e único para cada jogo.
- Cada jogo deve ter um URL para uma página com os detalhes do jogo.
- É importante registar os jogadores que compraram determinado jogo, bem como a data e o preço associados à compra.

- Os jogos podem ter um conjunto de crachás que são atribuídos aos jogadores quando um limite de pontos nesse jogo é atingido. Portanto, é necessário registar o nome do crachá que é único para cada jogo, o limite de pontos e o url para a imagem do crachá.

2.4 Partidas

Os requisitos relacionados às partidas são os seguintes:

- Cada vez que o jogo é jogado, é criada uma partida com um número sequencial e único para cada jogo.
- As datas e horas de início e de fim de cada partida devem ser guardadas.
- As partidas pode ser normal de apenas um jogador, ou multi-jogador.
- As partidas normais devem ter informação sobre o grau de dificuldade (valor de 1 a 5) e estar associadas ao jogador que as joga e à pontuação por ele obtida.
- As partidas multi-jogador devem estar associadas aos jogadores que as jogam sendo necessário guardar as pontuações obtidas por cada jogador em cada partida.
- As partidas multi-jogador também devem conter informação sobre o estado em que se encontram, o qual pode tomar os valores 'Por iniciar', 'A aguardar jogadores', 'Em curso' e 'Terminada.
- Assume-se a existência de um sistema que atualiza a pontuação e estado durante a execução do jogo.
- Cada partida deve estar associada a uma região e apenas jogadores dessa região a podem jogar.

2.5 Crachás

Os requisitos relacionados aos crachás são os seguintes:

- Cada jogo pode ter um conjunto de crachás que são atribuídos aos jogadores quando um limite de pontos nesse jogo é atingido.
- Para cada crachá, é necessário registar o nome do crachá deve ser único para cada jogo, o limite de pontos e o url para a imagem do crachá.
- Devem ficar registados na base de dados os crachás que são atribuídos a cada jogador.

2.6 Estatísticas

Os requisitos relacionados às estatísticas são os seguintes:

- Para cada jogador, interessa registar o número de partidas que efetuou, o número de jogos diferentes que jogou e o total de pontos de todos os jogos e partidas efetuadas.
- Para cada jogo interessa registar o número de partidas, o número de jogadores e o total de pontos.
- Os dados das estatísticas devem ser atualizados continuamente à medida que os jogadores participam de partidas e interagem com a plataforma.

2.7 Relações de amizade entre jogadores

Os requisitos relacionados às relações de amizade entre jogadores são os seguintes:

- Os jogadores devem ser capazes de adicionar outros jogadores como amigos, criando uma relação de amizade.
- A base de dados deve armazenar as relações de amizade entre os jogadores.

2.8 Conversas e mensagens

Os requisitos relacionados às conversas e mensagens são os seguintes:

- É possível criar uma conversa entre vários jogadores com um identificador gerado pelo sistema e um nome.
- Associado à conversa, existem mensagens enviadas pelos jogadores.
- Cada mensagem deve ter um número de ordem único e sequencial para cada conversa que serve de identificador, a data e hora da mensagem, o texto e qual o jogador que enviou a mensagem.

Capítulo 3

Modelo de Dados

3.1 Modelo Conceptual

Foi realizado um modelo conceptual Entidade-Associação de forma a exprimir uma representação das entidades e relações entre estas, facilitando a compreensão do funcionamento da BD. Este modelo foi criado utilizando uma aplicação de desenho de diagramas nomeada "Dia"e comprimido para o ficheiro enviado na entrega do trabalho. Ao longo da criação deste modelo toma-mos algumas decisões que achamos importante realçar, tais como as entidades fracas e Entidades associativas escolhidas.

Entidades Fracas

Neste modelo representamos como entidade fraca as entidades relativas ás estatísticas de jogadores e de jogos. Não fazendo sentido a existência de estatísticas de um jogador sem a existência de jogador, bem como não faria sentido manter estatísticas de um jogador que seja removido da BD.

Entidade associativa (agregação)

Acontecimentos como o envio de uma mensagem ou o começo de uma partida por parte de um ou mais jogadores só acontecem se, respetivamente, o jogador estiver na conversa para onde vai enviar a mensagem e se os jogadores tiverem comprado o jogo do qual tentam iniciar uma partida. Disto isto utilizamos Entidades Associativas para garantir que os jogadores só jogam jogos que tenham comprado e jogadores só enviam mensagens em conversas a que pertençam.

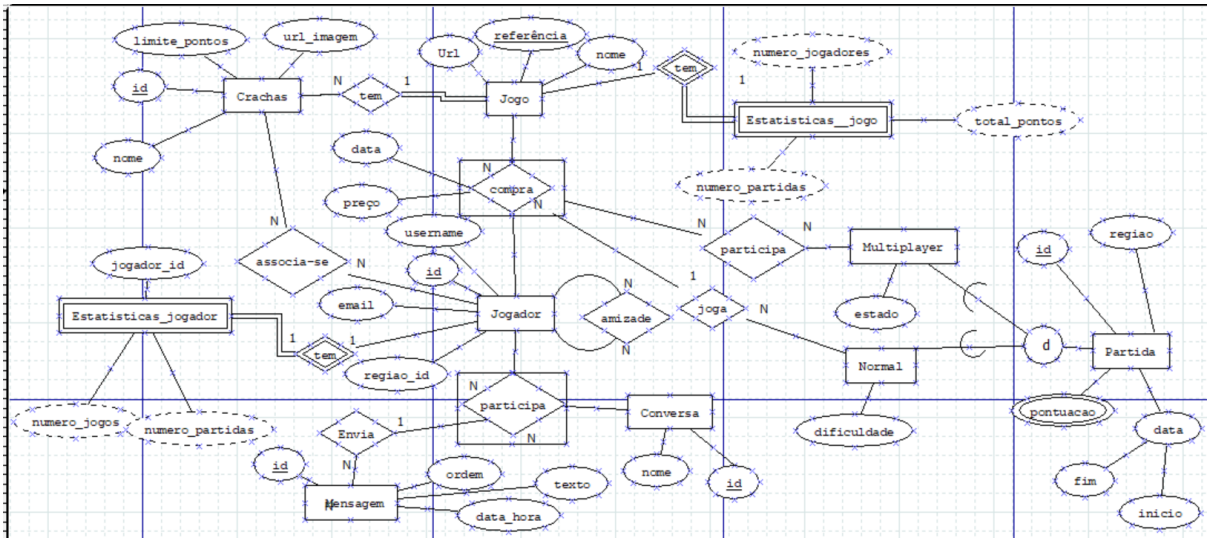

Figura 3.1: EA Model

3.2 Modelo Relacional

Tendo o modelo conceptual acabado, passamos para a criação de um modelo relacional deixando já explicitas as restrições de integridade e mantendo o modelo normalizado até a moda 3F, o modelo resultante foi o seguinte:

- **JOGADOR**(id, email, username, regiao_id)
- **JOGO**(referencia, nome, url)
- **PARTIDA_NORMAL**(id, jogo_referencia, data_inicio, data_fim, pontuacaom, dificuldade)

  FK: {jogo_referencia} de JOGO.referencia
- **PARTIDA_MULTIJOGADOR**(id, jogo_referencia, data_inicio, data_fim, pontuacao, estado)

  FK: {jogo_referencia} de JOGO.referencia
- **CRACHÁ**(id, jogo_referencia, nome, limite_pontos, url_imagem)

  FK: {jogo_referencia} de JOGO.referencia
- **CONVERSA**(id, nome)
- **MENSAGEM**(id, conversa_id, ordem, jogador_id, data_hora, texto)

  FK: {conversa_id ; jogador_id} de CONVERSA.id ; JOGADOR.id
- **ESTATÍSTICAS_JOGADOR**(jogador_id, numero_partidas, numero_jogos, total_pontos)

  FK: {jogador_id} de JOGADOR.id
- **ESTATÍSTICAS_JOGO**(jogo_referencia, numero_partidas, numero_jogadores, total_pontos)

  FK: {jogo_referencia} de JOGO.referencia

- **PARTICIPA_CONVERSA**(jogador_id, conversa_id)

  FK: {jogador_id ; conversa_id} de JOGADOR.id ; CONVERSA.id

- **CRACHA_JOGADOR**(jogador_id, cracha_id)

  FK: {jogador_id ; cracha_id} de JOGADOR.id ; CRACHA.id

- **COMPRA**(jogador_id, jogo_referencia, data_compra, preco)

  FK: {jogador_id ; jogo_referencia} de JOGADOR.id ; JOGO.referencia

- **PARTICIPAÇÃO_JOGOD**(jogador_id, partida_multi_id)

  FK: {jogador_id ; partida_multi_id} de JOGADOR.id ; PARTID.id

- **AMIZADE**(jogador_id1, jogador_id2) FK: {jogador_id1 ; jogador_id2} de JOGADOR.id; JOGADOR.id

Capítulo 4

Implementação da Base de Dados

4.1 Tabelas

De acordo com os modelos de dados apresentados anteriormente foram criados dois scripts SQL para a criação e remoção do modelo fisico.

- Criação: A criação das tabelas foi feita garantindo que as restrições dadas pelo cliente eram satisfeitas, mantendo também a coerência com os modelos de dados explicados na secção 2.
- Remoção: O script de remoção de tabelas consiste numa sequência de "drop tables".

Depois destes dois scripts foi feito um terceiro script para realizar um preenchimento inicial da base dados. Para isto utilizamos inserts, respeitando as restrições explicitas no script de criação de tabelas. Para confirmar o bom funcionamento destas restrições tentamos também a inserção de valores que não as respeitavam.

4.2 Funções

4.2.1 O que é?

Funções como as que vamos utilizar, recorrem ao PL/PgSQL para serem possíveis alguns tipo de instruções como IF's, for's entre outros. Estas funções retornam um valor e podem ou não ter parâmetros passados para a função que podem mudar o seu retorno.

4.2.2 Utilidade na BD

No trabalho usamos várias vezes funções para funcionalidades como obter os pontos de todos um jogadores para um certo jogo ou situação que necessitemos de triggers.

Criar, Desativar e Banir Jogadores

Para criar um jogador, primeiro é verificado se o email já existe, o username já existe e se a região não existe. Caso atenda a todos estes situações é inserido um jogador novo com o estado 'Ativo'. Para desativar ou banir um jogador é apenas verificado se o id existe, sendo a resposta positiva, atualizando-o com o respetivo estado, 'Inativo' ou 'Banido'.

Total de Pontos por Jogador

Nesta função é somado os pontos da partida normal e da partida multiplayer se o jogador existir. Caso o jogador não tenha pontos o retorno irá ser 0 em vez de null.

Total de Jogos por Jogador

Nesta função é necessário contar todos os jogos de um certo jogador. Para isto fazemos uma "união"das partidas normal e multiplayer realizadas pelo jogador, usando um distinct para os valores não serem repetidos.

Pontos do Jogo por Jogador

Para calcular os pontos de jogo por jogador, primeiro precisamos de uma referência de um jogo e que ela exista. Após isto é calculado quantas vezes o jogador aparece nas partidas normais e multiplayer usando o "union"e adicionando a pontuacao de cada partida ao retorno.

Banir de uma vista

Para banir um jogadores de uma vista "jogadorTotalInfo"que explicaremos noutro ponto do relatório, criamos uma função que chama o procedimento armazenado banir_jogador para todos os jogadores presentes em jogadoresTotalInfo.

Associar Crachas Final de Partida

Para a associação de crachás mal uma partida acabe, cria-se uma função que através de triggers (explicado em secções mais para a frente) no final de cada partida chamando o procedimento armazenado "AssociarCracha"já existente no projeto.

4.3 Procedimentos Armazenados

4.3.1 O que é?

Procedimentos armazenados são troços de códigos que depois de compilados são guardados prontos a executar posteriormente várias vezes. Este mesmo procedimento armazenado pode por sua vez ter parâmetros à espera de serem atribuídos, desta forma o comportamento e resultado do procedimento armazenado pode variar.

4.3.2 Utilidade na BD

Como resumido antes, procedimentos armazenados facilitam a reutilização rápida de código, dito isto, funcionalidades relativas a crachás, conversas e mudanças de estados de atividade de jogadores beneficiam de ser feitas através de procedimentos armazenados.

AssociarCracha

Para a associar aos crachás aos jogadores criou-se um procedimento armazenado onde se calcula o total de pontos do jogador no jogo, se este total for maior ou igual ao total de pontos necessários para obter o crachá este é atribuído ao jogador.

IniciarConversa

O inicio de uma conversa entre dois ou mais jogadores acontece através de um procedimento armazenado onde são passados como parâmetros o id do jogador, o id da conversa e o nome pretendido para a conversa, os id's são inseridos na tabela participacao_conversa e o nome da conversa por sua vez inserido na tabela conversa.

JuntarConversa

Para ser possível a um jogador juntar-se a uma conversa criou-se mais um procedimento armazenado que simplesmente faz a inserção do id do jogador e do id da conversa passados como parâmetros na tabela participacao_conversa.

EnviarMensagem

Enviar mensagens dentro das conversas é também possível devido a um procedimento armazenado que recebe como parâmetro um id de um jogador, da conversa e um texto. Primeiro verifica-se se algum id não existe ou o jogador não está na conversa. Caso passe estas condições, seleciona-se a ordem máxima do id da conversa e adiciona-se mais um pois tem de ser sequencial. Finalmente insere-se uma mensagem nova dando "rollback"caso alguma exceção tenha acontecido.

4.4 Vistas

4.4.1 O que é?

Vistas são uma forma mais "user friendly"de criar tabelas virtuais sem necessidade de usar memória adicional. Não só se torna uma opção otimizada como também promove a segurança da BD.

4.4.2 Utilidade na BD

Como resumido antes, vistas tornam possível a criação de tabelas virtuais com informação de várias tabelas já presentes na BD. Isto faz com que seja útil utilizar vistas em funcionalidades como na capacidade de ver a informação total sobre todos os jogadores não banidos presentes na BD.

JogadorTotalInfo

A informação dos jogadores não banidos obtem-se através de uma vista que seleciona toda a informação necessária para obter os resultados pretendidos e agrupa-a de acordo com o id de jogador.

4.5 Gatilhos

4.5.1 O que é?

Gatilhos ou triggers são um tipo especial de procedimento armazenado que executa na condição de um acontecimento.

4.5.2 Utilidade na BD

Para situações em que uma funcionalidade inclui uma condição para ser realizada utilizamos triggers. Situações estas como por exemplo a associação automática de crachas no fim de uma partida ou o banimento dos jogadores presentes numa vista depois de um DELETE.

Cracha Fim de Partida

Para associar o cracha a um jogador num fim de uma partida utilizamos um trigger que é executado quando a data de fim de partida deixa de ser null. Quando isto acontece selecionamos os jogadores presentes no jogo e seguidamente corremos todos os crachás chamando o procedimento armazenado "associarCracha"para verificar se algum jogador satisfaz as condições para obter o crachá.

Banir jogador em JogadorTotaInfo

Para banir os jogadores presentes na vista, utilizamos um trigger que executa quando acontece o DELETE da vista. Quando isto acontece simplesmente da-mos update a todas as rows da vista colocando o estado dos jogadores a "Banido".

4.6 Controlo Transacional e Níveis de Isolamento

Ao longo do trabalho fomos mantendo a atomicidade, consistência, o isolamento e a durabilidade das transações feitas. Para manter este controlo transacional, utilizamos o nivel de isolamento read commited, evitando assim dirty reads. Tendo garantido que não existe possibilidade de haver non-repeatable reads ou phantom reads não foi necessário nenhum nível de isolamento mais alto.

Capítulo 5

Testes

5.1 Script de Teste

Foi desenvolvido um script autônomo para executar todos os testes, que ao ser executado, lista para cada teste o seu nome e indicação se ele correu ou não com sucesso. Os testes foram realizados para cenários normais e de erro. Todos os testes estão no zip enviado com o relatório.

Para todos os testes são usadas duas funcões auxiliares:

- **Função teste**: Tem como parâmetros o número do teste(integer), a sua descrição(varchar255) e o seu resultado(boolean). Esta função bastante simples pois apenas cria um "raise notice", um aviso, com os parâmetos expecificados. Caso o resultado seja true irá adicionar "Resultado OK"ao aviso, caso contrário, adiciona "Resultado FAIL"

  Ex: "Teste 1: Inserir jogador com dados bem passados: Resultado OK"

  Ou caso falhe: "Teste 1: Inserir Cliente com dados bem passados: Resultado FAIL"

- **Função simular erro insercao**: Retorna um trigger e não tem parâmetros. Esta função serve para criar triggers durante o script autonomo. Usa uma "raise exception"em vês de um "raise notice"pois neste caso queremos que o script pare. Sempre que testamos uma transação criamos um trigger novo com esta função e apagamos-o após o teste. O trigger é apagado para não influenciar os outros testes.

No final do script autónomo é feito um rollback, para a base de dados voltar ao estado antes dos testes.

5.2 Resultados dos Testes

A seguir, apresentamos os resultados dos testes realizados, organizados por categorias para facilitar a leitura.

5.2.1 Testes de criação de jogador

- **Teste 1**: Criar jogador com parâmetros possíveis

  Objetivo: Verificar se o jogador é criado corretamente com os parâmetros fornecidos.

- **Teste 2**: Erro ao criar jogador com email já existente

  Objetivo: Verificar se o sistema impede a criação de um jogador com um email que já existe na base de dados.

- **Teste 3**: Erro ao criar jogador com username já existente

  Objetivo: Verificar se o sistema impede a criação de um jogador com um username que já existe na base de dados.

- **Teste 4**: Erro ao criar jogador com região não existente

  Objetivo: Verificar se o sistema impede a criação de um jogador com uma região não existente.

- **Teste 5**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro na criação de um jogador.

5.2.2 Testes de desativação de jogador

- **Teste 6**: Desativar jogador existente

  Objetivo: Verificar se o jogador é desativado corretamente.

- **Teste 7**: Erro ao desativar jogador com id inexistente

  Objetivo: Verificar se o sistema impede a desativação de um jogador com um id inexistente.

- **Teste 8**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro ao desativar um jogador.

5.2.3 Testes de banimento de jogador

- **Teste 9**: Banir jogador existente

  Objetivo: Verificar se o jogador é banido corretamente.

- **Teste 10**: Erro ao banir jogador com id inexistente

  Objetivo: Verificar se o sistema impede o banimento de um jogador com um id inexistente.

- **Teste 11**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro ao banir um jogador.

5.2.4 Testes de cálculo de pontos e jogos

- **Teste 12**: Calcular total de pontos de jogador com pontos

  Objetivo: Verificar se o sistema calcula corretamente o total de pontos de um jogador que possui pontos.

- **Teste 13**: Calcular total de pontos de jogador sem pontos

  Objetivo: Verificar se o sistema calcula corretamente o total de pontos de um jogador que não possui pontos.

- **Teste 14**: Calcular total de pontos com id de jogador inválido

  Objetivo: Verificar se o sistema retorna um erro ao tentar calcular o total de pontos de um jogador com id inválido.

- **Teste 15**: Verificar total de jogos de jogador com jogos

  Objetivo: Verificar se o sistema calcula corretamente o total de jogos de um jogador que possui jogos.

- **Teste 16**: Verificar total de jogos de jogador sem jogos

  Objetivo: Verificar se o sistema calcula corretamente o total de jogos de um jogador que não possui jogos.

- **Teste 17**: Verificar total de jogos com id de jogador inválido

  Objetivo: Verificar se o sistema retorna um erro ao tentar calcular o total de jogos de um jogador com id inválido.

5.2.5 Testes de verificação de pontos e participantes em jogos

- **Teste 18**: Verificar total de pontos de jogo com jogadores

  Objetivo: Verificar se o sistema calcula corretamente o total de pontos em um jogo com jogadores.

- **Teste 19**: Verificar quantos jogadores jogaram o jogo

  Objetivo: Verificar se o sistema conta corretamente o número de jogadores que participaram em um jogo.

- **Teste 20**: Verificar nenhum jogador jogou o jogo

  Objetivo: Verificar se o sistema identifica corretamente quando nenhum jogador participou em um jogo.

- **Teste 21**: Verificar pontos jogo por jogador com referência inválida

  Objetivo: Verificar se o sistema retorna um erro ao tentar verificar os pontos de um jogo por jogador com uma referência inválida.

5.2.6 Testes de atribuição de crachás

- **Teste 22**: Crachá atribuido corretamente

  Objetivo: Verificar se o crachá é atribuído corretamente ao jogador.

- **Teste 23**: Crachá não atribuido pois jogador não existe

  Objetivo: Verificar se o sistema impede a atribuição de um crachá a um jogador que não existe.

- **Teste 24**: Crachá não atribuido pois jogo não existe

  Objetivo: Verificar se o sistema impede a atribuição de um crachá a um jogador em um jogo que não existe.

- **Teste 25**: Crachá não existe

  Objetivo: Verificar se o sistema retorna um erro ao tentar atribuir um crachá que não existe.

- **Teste 26**: Crachá não atribuido por falta de pontos

  Objetivo: Verificar se o sistema impede a atribuição de um crachá a um jogador que não possui pontos suficientes.

- **Teste 27**: Crachá não existe no jogo

  Objetivo: Verificar se o sistema retorna um erro ao tentar atribuir um crachá que não existe no jogo.

- **Teste 28**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro na atribuição de um crachá.

5.2.7 Testes de criação e gerenciamento de conversas

- **Teste 29**: Conversa criada corretamente

  Objetivo: Verificar se a conversa é criada corretamente.

- **Teste 30**: Jogador associado à conversa corretamente

  Objetivo: Verificar se o jogador é adicionado corretamente à conversa.

- **Teste 31**: Conversa não criada por jogador não existir

  Objetivo: Verificar se o sistema impede a criação de uma conversa quando o jogador não existe.

- **Teste 32**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro na criação de uma conversa.

- **Teste 33**: Conversa juntada

  Objetivo: Verificar se o jogador é adicionado corretamente à conversa existente.

- **Teste 34**: Jogador não existe

  Objetivo: Verificar se o sistema retorna um erro ao tentar adicionar um jogador inexistente à conversa.

- **Teste 35**: Conversa não existe

  Objetivo: Verificar se o sistema retorna um erro ao tentar adicionar um jogador a uma conversa inexistente.

- **Teste 36**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro ao adicionar um jogador à conversa.

5.2.8 Testes de envio de mensagens

- **Teste 37**: Mensagem enviada

  Objetivo: Verificar se a mensagem é enviada corretamente.

- **Teste 38**: Jogador não existe

  Objetivo: Verificar se o sistema retorna um erro ao tentar enviar uma mensagem de um jogador inexistente.

- **Teste 39**: Conversa não existe

  Objetivo: Verificar se o sistema retorna um erro ao tentar enviar uma mensagem para uma conversa inexistente.

- **Teste 40**: Jogador não faz parte da conversa

  Objetivo: Verificar se o sistema retorna um erro ao tentar enviar uma mensagem de um jogador que não faz parte da conversa.

- **Teste 41**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro ao enviar uma mensagem.

5.2.9 Testes de mecanismos automáticos

- **Teste 42**: Foram adicionados todos os jogadores que não estão banidos

  Objetivo: Verificar se o sistema adiciona corretamente todos os jogadores que não estão banidos.

- **Teste 43**: Crachás adicionados ao jogador

  Objetivo: Verificar se todos os crachás são adicionados corretamente ao jogador.

- **Teste 44**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida corretamente quando ocorre um erro na adição de crachás ao jogador.

- **Teste 45**: Jogador banido

  Objetivo: Verificar se o jogador é banido corretamente.

- **Teste 46**: Transação revertida corretamente

  Objetivo: Verificar se a transação é revertida coretamente quando ocorre um erro ao banir um jogador.

5.3 Análise dos Resultados

Os testes realizados mostraram que todas as funcionalidades implementadas funcionam conforme esperado, tanto em cenários normais quanto em cenários de erro. As transações foram revertidas corretamente quando necessário, garantindo a consistência e integridade dos dados armazenados na base de dados.

Capítulo 6

Conclusão

Com este trabalho compreendemos melhor a importância da boa utilização dos níveis de isolamento e praticamos a utilização da linguagem de programação estrutural plpgsql. Tendo assim um maior leque de possibilidades para estruturar uma BD que tornou este trabalho num trabalho desafiante ao ponto de nos obrigar a realmente procurar e aprender sobre.