ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Sistema de Gestão de Jogos Fase 1

48268 – Marçorio Fortes
48315 – Rafael Costa
47539 – Bernardo Serra

Orientadores: Doutor Nuno Leite

Relatório para a Unidade Curricular de Sistemas de Informação da Licenciatura em Engenharia Informática e de Computadores
Semestre de Verão 2022/2023

8 de maio de 2023

Instituto Superior de Engenharia de Lisboa

Licenciatura em Engenharia Informática e de Computadores

Sistema de Gestão de Jogos Fase 1

48268 Marçorio Fortes

48315 Rafael Costa

47539 Bernardo Serra

Orientadores: Doutor Nuno Leite

Relatório para a Unidade Curricular de Sistemas de Informação da Licenciatura em Engenharia Informática e de Computadores

Semestre de Verão 2022/2023

8 de maio de 2023

Resumo

Este projeto está dividido em duas fases, sendo esta a primeira. Este relatório tem como objetivo a elaboração de uma base de dados para um sistema de gestão de jogos.

Com este trabalho pretendemos aplicar conceitos aprendidos anteriormente, na unidade curricular de Introdução Sistemas de Informação, nomeadamente a criação do modelo de dados conceptual e relacional, aplicando todas a as restrições de integridade e de modo que esteja normalizado até à 3NF.

Como foco principal temos objetivo de demostrar conhecimentos obtidos nesta unidade curricular ao nível do controlo transacional, níveis de isolamento, procedimentos armazenados através da resolução dos exercícios propostos.

Este relatório parte do pressuposto do acesso por parte do leitor ao código desenvolvido no âmbito do mesmo, não sendo assim necessário enunciá-lo em extensão, bastando apenas mencionar trechos do mesmo.

Abstract

This project is divided into two phases, this being the first one. The purpose of this report is to develop a database for a game management system.
With this work, we aim to apply concepts learned previously in the Introduction to Information Systems course, namely the creation of the conceptual and relational data model, applying all integrity constraints and ensuring that it is normalized up to the 3NF.
Our main focus is to demonstrate knowledge acquired in this course regarding transaction control, isolation levels, and stored procedures, through the resolution of proposed exercises.
This report assumes that the reader has access to the code developed within the scope of this project, so it is not necessary to describe it in detail, only to mention some excerpts.

Agradecimentos

Queremos agradecer ao Professor Doutor Nuno Leite pelo grande apoio que nos foi proporcionado em aulas e fora das aulas.

Índice

1. Introdução

Nesta parte do trabalho falaremos de aspetos teóricos que foram necessários para a execução do trabalho.

1.1 Controlo Transacional

O controlo transacional é um mecanismo utilizado em sistemas de gestão de bases de dados para garantir a consistência e integridade dos dados. Ele permite que as transações sejam executadas em blocos, onde todas as operações são tratadas como uma unidade atómica, ou seja, ou todas as operações são executadas com sucesso, ou nenhuma delas é executada. Isso evita que ocorram situações em que apenas parte de uma transação é concluída, o que pode levar a dados inconsistentes. O controlo transacional utiliza técnicas como **commit** e **rollback** para garantir a consistência dos dados e é amplamente utilizado em sistemas de gestão de bases de dados relacionais.

1.2 Níveis de Isolamento

Os níveis de isolamento são um conjunto de regras que definem como as transações devem ser gerenciadas e protegidas de interferências de outras transações que ocorrem simultaneamente em um banco de dados. Existem quatro níveis de isolamento: **READ UNCOMMITTED**, **READ COMMITTED**, **REPEATABLE READ** e **SERIALIZABLE**.

READ UNCOMMITTED: Neste nível de isolamento, uma transação pode ler dados que ainda não foram confirmados por outras transações, permitindo **DIRTY READ**. Esse nível de isolamento tem o menor grau de restrição, pois permite que as transações acessem dados em tempo real, mas pode causar problemas de consistência dos dados.

READ COMMITTED: Neste nível de isolamento, uma transação só pode ler dados que foram confirmados por outras transações, evitando **DIRTY READ**. As leituras neste nível de isolamento são mais seguras, mas ainda podem permitir problemas de inconsistência dos dados devido à possibilidade de outras transações atualizarem ou excluírem dados durante a leitura.

REPEATABLE READ: Neste nível de isolamento, uma transação garante que as leituras sejam repetíveis, ou seja, ao executar a mesma consulta novamente, os resultados serão os mesmos. Isso ocorre porque as leituras são protegidas contra atualizações e exclusões de dados por outras

transações. No entanto, novas inserções podem ser realizadas por outras transações, o que ainda pode causar problemas de inconsistência dos dados.

SERIALIZABLE: Neste nível de isolamento, as transações são executadas sequencialmente, garantindo que as leituras e gravações sejam isoladas de outras transações. Isso evita completamente problemas de inconsistência dos dados, mas pode resultar em um desempenho mais baixo devido à necessidade de bloquear completamente as transações em vez de permitir que elas executem paralelamente.

1.3 Vistas

Usando a terminologia SQL, uma Vista (**VIEW**) consiste numa única tabela constituída a partir de outras tabelas ou outras Vistas anteriormente definidas. As Vistas não têm dados próprios, os dados de uma Vista são manipulados nas tabelas base que servem de suporte a essa vista e uma vista é armazenada como um comando **SELECT**. Não tem existência física embora apareça ao utilizador como se o tivesse, pelo que: origina algumas limitações às operações de **UPTADE**.

1.4 Procedimentos Armazenados

Em geral, não se devem usar os procedimentos armazenados para a implementação da lógica aplicacional (isto é, de processos de negócio). A sua utilização deve ser restringida a aspetos que tenham a ver com a natureza intrínseca dos dados, como, por exemplo, validação de regras de negócio associados a restrições de integridade dos dados, não suportadas diretamente pelo SGBD.

Bom desempenho – Permitem a diminuição do tráfego na rede.

Bom encapsulamento – A utilização de procedimentos evita que se acedam diretamente aos dados, contribuindo para uma maior consistência dos mesmos.

Maiores níveis de reutilização - Procedimentos podem ser partilhados por várias aplicações

Maior segurança - Utilizadores podem ter permissão para executar um procedimento e não para aceder diretamente aos dados; lógica ocultada para os clientes.

1.5 Gatilhos

Os gatilhos (ou triggers, em inglês) são objetos de banco de dados que executam automaticamente um conjunto de ações sempre que ocorre uma determinada operação em uma tabela, como **INSERT**, **UPDATE** ou **DELETE**. Eles podem ser utilizados para garantir a integridade referencial dos dados e para automatizar tarefas. Existem dois tipos principais de gatilhos: os de linha, que são executados para cada linha (**ROW**) afetada pela operação, e os de instrução (**STATEMENT**), que são executados uma vez para cada operação.

1.6 Funções

Permitem a realização de operações sobre a base de dados diretamente, facilitando a dinâmica dos programas e evitando a realização de múltiplas **queries** para um mesmo efeito.

2. Formulação do Problema

A empresa “GameOn” quer desenvolver um sistema para gerir jogos, jogadores e partidas. Os jogadores pertencem. Os jogos têm um conjunto de crachás para recompensar jogadores. As partidas podem ser normais ou multiplayer. Jogadores podem adicionar amigos e conversar entre si. Deve haver tabelas para estatísticas de jogadores e jogos.

2.1 Modelo EA

Com o objetivo de cumprir todos os requisitos possíveis de mostrar no modelo EA, foi criado o Diagrama da EA presente na Figura 1. Tivemos alguns problemas na implementação onde tivemos de recorrer ao professor até conseguirmos obter a melhor solução possível.

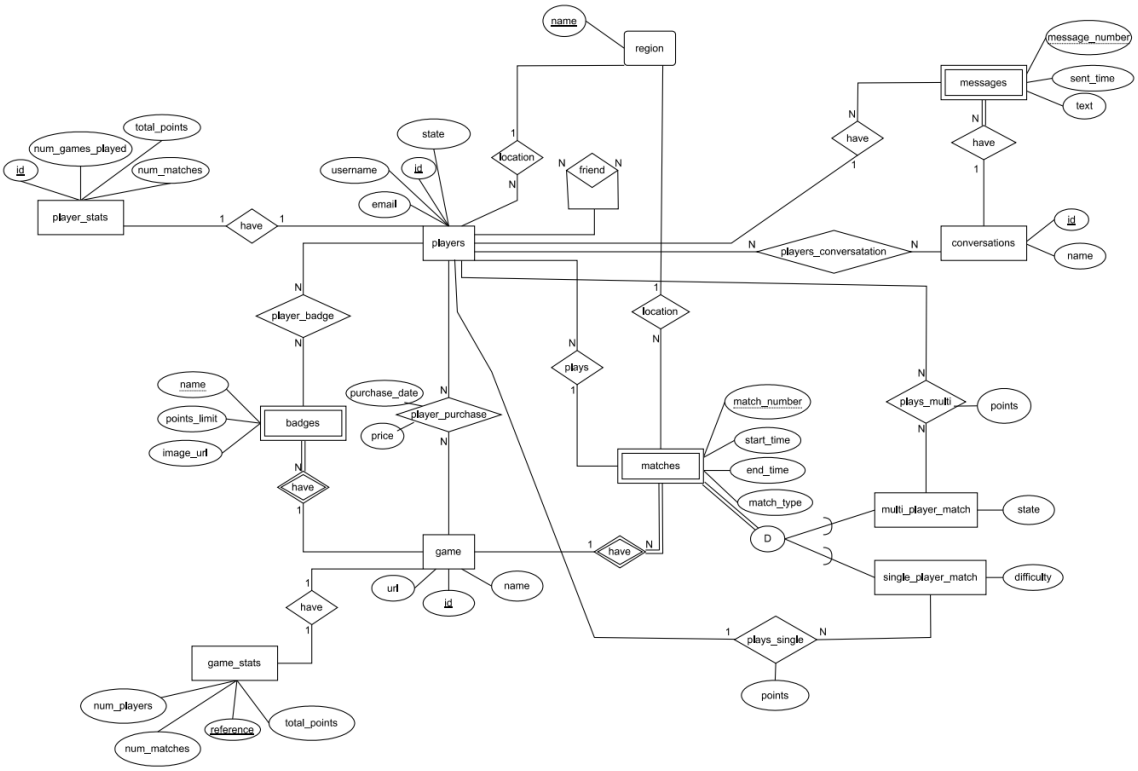

Figura 1 - Diagrama EA

2.2 Restrições de Integridade

Players (Jogadores)

- email é único e não pode ser nulo.
- username é único e não pode ser nulo.
- state é não nulo e tomando um dos seguintes valores: ‘Atvio’, ‘Inativo’ ou ‘Banido’.

Game (Jogo)

- reference é uma referência alfanumérica de dimensão 10 (varchar(10)).
- Name é único e não pode ser nulo.

Regions (região)

- name não pode ser nulo e tomando um dos seguintes valores: ‘Europe’,’Asia’, ‘SouthAmerica’,’Africa’, ‘Australia’ ou ‘North America’.

Matches (Partidas)

- start_time (hora de início) é sempre anterior a end_time (hora de fim)
- end_time inicialmente encontra-se a nulo até o final da partida.
- As partidas só podem ter jogadores da mesma região

Single_player_match (partida dum jogador)

- difficulty não pode ser nulo e toma valores entre 1 a 5, quanto maior o numero maior a dificuldade.

Multi_player_match (partida de vários jogadores)

- state não pode se nulo e toma um dos seguintes valores: ‘Por iniciar’, ‘A aguardar jogadores’, ‘Em curso’, ‘Terminada’.

2.3 Modelo Relacional

Não sendo este o foco do relatório, decidimos não representar o modelo relacional pelo facto de este estar implícito no modelo físico.

3. Detalhes de Implementação

3.1 Nota Prévia

Uma vez mais, são apenas referidos detalhes da implementação que achámos relevantes para compreensão de como o trabalho foi efetuado e a razão das decisões tomadas no mesmo.
Não sendo mencionado aqui alíneas inteiras pedidas no relatório, por considerarmos que a logica das mesmas é trivial ou semelhante à de outras alíneas anteriores ou posteriores já explicadas.

3.2 Modelo físico e preenchimento inicial da BD.

Muitas das informações que não puderam ser mostradas no modelo ER pelas suas limitações foram implementadas nesta parte do trabalho. No modelo físico, por ser mais especifico que o modelo ER é projetado para otimizar o desempenho, a segurança e a integridade dos dados no sistema de gerenciamento de jogos em estudo.
Preenchemos as tabelas com informação que achamos relevantes para a realização dos problemas propostos.

3.3 Criação de jogador e manipulação do seu estado (state).

Foi criado os mecanismos para a realização de criação dum novo jogador através de um procedimento. Onde recebe como parâmetro, o email, região e username e o estado inicial 'Ativo'.
Foi utilizado a instrução **INSERT INTO** players com os parâmetros referidos e sem necessidade de passar o id devido ao facto de este ser do tipo **SERIAL INTEGER**, ou seja, o novo id é gerado automaticamente.
Para alterar os estados do jogador para 'Banido' e 'Inativo' foi necessário apenas fazer um **UPTADE** do estado para respetivamente 'Banido' ou 'Inativo'.

3.4 Função totalPontosJogador

De modo a ter a melhor implementação do modelo ER que se traduziu para o modelo físico, temos duas tabelas que registam as pontuações, uma tabela para as partidas multi_player e uma tabela para as partidas single_player.
De modo a implentar a função, começamos por confirmamos a existência do id passado como parâmetro, caso não exista lançamos uma **RAISE EXCEPTION.** De seguida, caso o jogador exista, fazemos a soma da pontuação de todas as partidas single_player e multi_player, retornando a soma de ambas as somas.

3.5 Função totalJogosJogador

Esta função é muito parecida com a função da secção **3.4**. Sendo a logica de implementação a mesma só que neste caso contamos o número de jogos diferentes jogados.

3.6 Função PontosJogoPorJogador

A função pontosJogoPorJogador recebe como parâmetro a referência de um jogo e retorna uma query tabela com duas colunas (player_id, total_points).

Primeiramente começamos por confirmar se o jogo existe, caso não exista lançamos um **RAISE EXCEPTION.** Caso exista fazemos a união das tabelas single_player_match e play_multi usando o comando **UNION ALL** e de seguida realizamos um **LEFT JOIN** com a tabela multi_player_match, fazemos um **GRUOP BY** player_id e somamos os pontos totais de cada jogador.

3.7 Procedimento armazenado associarCrachá

Este procedimento 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 se o jogador tiver atingido o limite de pontos para o crachá passado como parâmetro.

Para fazer este procedimento começamos por confirmar se o jogador já tem o crachá ou não, caso já tenho crachá lançamos um **RAISE EXCEPTION**, caso contrario chamamos a função PontosJogoPorJogador() e guardamos o total de pontos que o jogador tem no jogo. De seguida, vamos confirmar os pontos necessários para obter o crachá, caso tenha os pontos necessários o crachá é atribuído ao jogador, caso contrário é lançado um **RAISE EXCEPTION.**

3.8 Procedimento armazenado iniciarConversa

Este procedimento é responsável por iniciar uma conversa, recebendo o id do jogador e o nome da conversa.

Começamos por confirmar se o jogador está ativo, lançando um **RAISE EXCEPTION** caso não esteja. De seguida fazemos uma inserção em conversation, passando apenas o nome o nome como parâmetro do insert, foi gerado automaticamente um novo id, como foi referido na secção **3.4**, na criação do jogador. Por dependência de chaves tivemos de fazer uma inserção na tabela players_conversation também.

3.9 Procedimento armazenado juntarConversa

Este procedimento foi muito simples de implementar. Tivemos que confirmar se o jogador já estava na conversa ou se encontrava-se ‘Inativo’, nesses dois casos lançamos um R**AISE EXEPTION.** Não estando na conversa ou ‘Inativo’, foi realizado um **INSERT INTO** players_conversation com os parâmetros recebidos (player_id, conversation_id).

3.10 Procedimento armazenado enviarMensagem

De modo similar aos procedimentos anteriores, neste também fizemos confirmações inicias para ver o estado do jogador e se este esta incluído na conversa. Caso esteja 'Ativo' e esteja na conversa iremos obter o numero da ultima mensagem associado a conversa, fazendo um inserção na tabela messages com o ultimo numero +1, o sent_time vai ser o retorn da chamada a função now(). oOs restantes valores serão os parâmetros passados ao procedimento (player_id, conversation_id e o text).

3.11 Criação da Vista jogadorTotalInfo

Para a criação da vista de modo a não aceder as tabelas de estatística utilizamos duas funções implementadas, que faziam parte dos exercícios propostos, que são respetivamente a função totalJogosJogador() da **secção 3.5** e totalPontosJogador () da **secção 3.4.** Foi criada outra função chamada partidas_jogador () que recebe um player_id e retorna a contagem de todas as partidas que o jogador participou. A utilização destas funções facilitou a criação da vista onde apresentamos apenas os jogadores com estado diferente de 'Banido'.

3.12 Atribuir Crachá no Final

Neste exercício usamos o mecanismo de Gatilhos (TRIGGER), depois duma atualização na tabela matches (**AFTER UPTADE**) o gatilho executa a função atribuir_crachas_trigger().

A função executada vai fazer confirmações a cerca do end_time (hora de fim), de modo a ver se continua a execução da função ou se lança um **RAISE EXCEPTION.** Caso seja para executar a função, realizamos um **FOREACH** a cada crachá do jogo para atribuir o crachá ao jogador no caso de ser singleplayer (caso tenha atingido os pontos necessários), no caso de ser multiplayer fizemos um **FOREACH** dentro de outro **FOREACH** de modo a percorrer o id de todos os jogadores e o nome de todos os crachás (do jogo que terminou).

3.13 Gatilho Banir Jogador

Este gatilho (**TRIGGER**) juntamente com a função executada pela mesma foi criado de modo, que em vez de eliminar as informações do jogador da base de dados se coloque o jogador no estado 'Banido'. A atualização na tabela jogador é repercutida sobre a Vista (**VIEW**) jogadorTotalInfo.

4. Conclusões

Neste trabalho tratou-se o problema da gestão do sistema de gerenciamento de jogos onde tivemos de aplicar conhecimentos anteriormente adquiridos e conhecimentos adquiridos ao longo do semestre na unidade curricular de Sistemas de Informação. Enfrentamos alguns problemas na criação do modelo ER (EA), porém com ajuda do professor foi-nos possível ultrapassar esse impasse, criando um bom modelo. Durante o desenvolvimento e a passagem para o modelo físico tivemos de voltar a atrás e rever o modelo EA, onde tínhamos cometido um pequeno erro, que foi corrigido. Não enfrentamos dificuldades na criação das primeiras funções, porém houve uma certa dificuldade em perceber os níveis de isolamento das transações, essas dificuldades foram colmatadas posteriormente, muito deveu-se a ajuda extraordinária do professor.

Acreditamos que todos os objetivos foram atingidos, realizamos testes para cada procedimento e função de modo a confirmar o funcionamento do mesmo.

Pensamos que poderíamos ter feito um modelo EA que traduzida para o modelo físico tivéssemos só uma tabela responsável pelo tratamento dos pontos, diferente da solução utilizado no trabalho, onde os jogos multijogadores e normal têm respetivamente cada um a sua própria tabela de pontos. No entanto, acreditamos que a solução inicial também era uma excelente solução. Sendo assim decidimos ficar como tínhamos feito inicialmente, para evitar grandes mudanças no trabalho.

Com esta primeira fase do trabalho conseguimos obter conhecimentos muito importantes para o nosso curso, iremos aplicar parte deste conhecimento em outras unidades curriculares que estão decorrer em simultâneo com SI, nomeadamente na unidade curricular Laboratório de Software (LS). Acreditamos também, que com a próxima fase que iremos realizar seremos aptos para executar um projeto final de curso muito promissor.

A solução obtida atingiu resultados satisfatórios.

Referências

[1] Material de estudo fornecido pelo Prof. Walter Vieira (slides com a matéria).

[2]