Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Sistema de Gestão de Jogos - F1

47206 : Tiago Alexandre Figueiredo Pardal (47206@alunos.isel.pt)

47202 : Manuel Maria Penha Gonçalves Cavaco Henriques (47202@alunos.isel.pt)

47198 : Ricardo Filipe Matos Rocha (47198@alunos.isel.pt)

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

Professor : Doutor Nuno Leite

15 – 04 – 2023

Resumo

No âmbito da primeira fase do trabalho prático da cadeira, este relatório tem como propósito a elaboração de uma base de dados para o problema proposto.

Através do trabalho realizado pretendemos demonstrar conhecimento sobre transações e níveis de transação, através da forma como resolvemos os problemas 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

Under the first phase of the assignment of this subject, this report has the purpose of building a database for the proposed problem.

Through this assignment we intend to demonstrate our knowledge about transactions and transaction levels, by presenting how the proposed problems were solved.

This report comes with the assumption that the reader has access to the developed code, being unnecessary to describe it to its full extent, as code snippets should suffice.

Índice

Lista de Figuras

1

Introdução

1.1 Níveis de Isolamento

De forma a manter a integridade transacional, assim como a consistência da DB, foi necessário atribuir a cada função e procedimento o nível de isolamento adequado.

Assim, consoante as operações que queríamos realizar, foi atribuído um dos seguintes níveis de isolamento:

- **READ COMMITED** - nível de isolamento mais baixo, uma vez que o PostgreSQL não implementa o nível de isolamento **READ UNCOMMITED**.

  Este nivel de isolamento garante que toda a informação que será lida, é informação que já se encontra committed, deste modo evitando a anomalia **DIRTY READ**;

- **REPEATABLE READ** - apenas é lida informação que já se encontra committed antes de qualquer leitura ou escrita realizada pela transação, isto é, após uma escrita ou leitura, toda a informação da tabela onde se realizou a escrita e / ou leitura será a informação committed antes desta, deste modo impedindo a ocorrência de **NON-REPEATABLE READS**;

- **SERIALIZABLE** - garante uma execução em serie das transações.

1.2 Vistas

Podem ser definidas de certa forma como uma camada de abstração sobre as relações presentes na base de dados, podendo ser consideradas de certa forma como interfaces obtidas através do result set de uma **query**. Os tuplos de uma vista são atualizados dinamicamente.

1.3 Funções

Funções permitem realizar operações sobre a base de dados diretamente, facilitando a dinâmica dos programas e evitando a realização de multiplas **queries** para um mesmo efeito.

1.4 Procedimentos

Procedimentos, por sua vez, são utilizados como transações, em que se pode realizar funções, dando uma adaptação em que permite o uso de sintaxe específica de transações com a chamada a funções incorporada.

1.5 Gatilhos

Os gatilhos permitem que caso numa relação, ou numa vista, seja realizada uma operação, uma função será executada em relação a essa operação. Os gatilhos têm três tipos e podem ser usados nas operações de **INSERT**, **UPDATE** ou **DELETE**. Um gatilho do tipo **INSTEAD OF** deverá repor a operação realizada numa vista por uma função, ou seja, no caso de ser aplicado numa operação de DELETE, esta operação não se realizará, enquanto que a função que está no gatilho vai se realizar. Um gatilho **BEFORE** executa uma função sobre uma relação antes que a operação possa ser realizada. Por fim, um gatilho **AFTER** realiza a sua função após a operação realizada na relação tenha concluído.

2

Modelo da base de dados

2.1 Caso em Estudo

No presente trabalho é nos proposto o desenvolvimento de um sistema de jogos e jogadores onde seja possível guardar os jogos adquiridos por cada jogador, as suas partidas individuais ou com outros jogadores sobre estes mesmos jogos e as conversas com outros jogadores, além de um mecanismo de amizade para ligar contas para mais fácil interação.

2.2 Modelo EA

Relativamente às tabelas de estatísticas, após grande discussão entre os elementos do grupo optámos por seguir o enunciado e usar tabelas, embora nos tivesse sido dada a possibilidade de usar vistas.

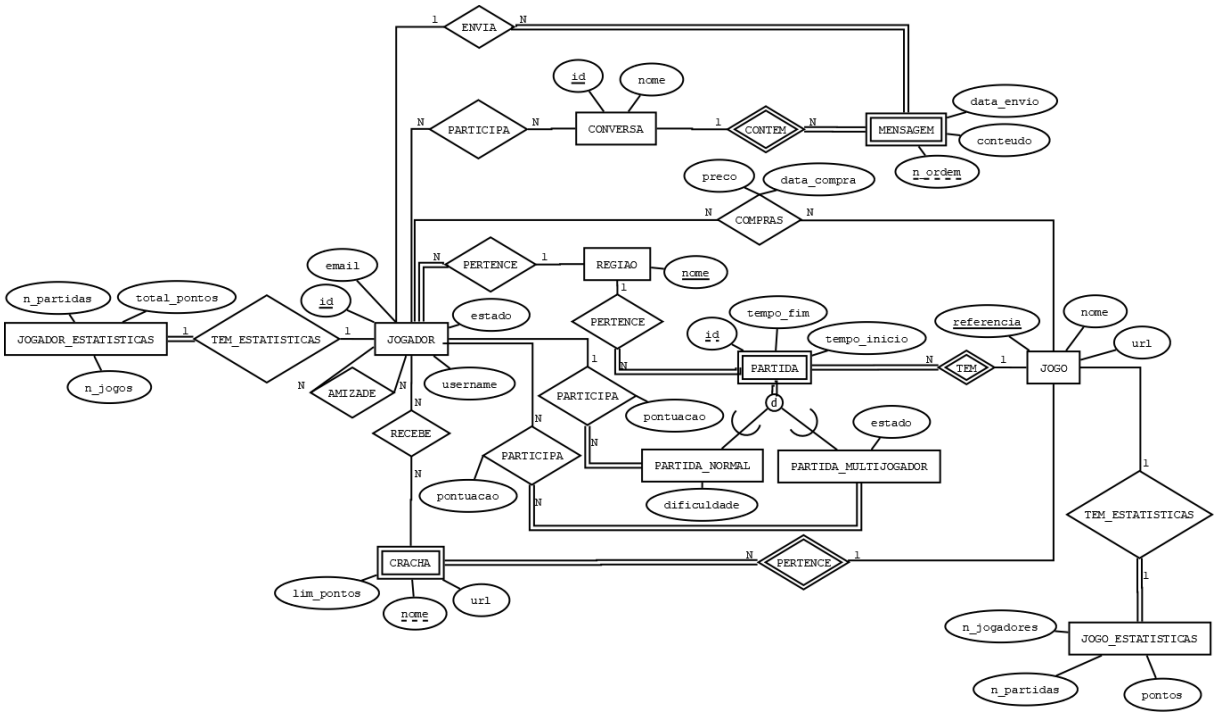

Figura 2.1: Diagrama EA

2.2.1 Restrições Integridade

Jogador

- id é gerado pelo sistema
- email é único e obrigatório
- username é único e obrigatório
- estado toma um dos seguintes valores: "Ativo", "Inativo", "Banido"

Jogo

- referência é uma sequência alfa numérica de dimensão 10
- nome é único e obrigatório

Partida

- chave é um número sequencial
- apenas jogadores associados à região da partida podem jogar

- tempo_inicio é anterior a tempo_fim

Partida Normal

- Grau de dificuldade é um valor inteiro entre 1 e 5

Partida Multijogador

- Estado toma um dos seguintes valores: "Por iniciar", "A aguardar jogadores", "Em curso", "Terminada"

2.3 Modelo Relacional

Na passagem do modelo EA para o modelo relacional tomamos as decisões mencionadas nesta secção.

Restrições de Integridade previstas e não implementadas no presente modelo serão mais tarde implementadas ao nível da aplicação java a ser desenvolvida na segunda fase do presente projeto.

2.3.1 Regiões

Tomamos a decisão de assumir as regiões que deveriam existir, restringindo-as apenas às a seguir mencionadas: 'EU', 'NA', 'SA', 'AS', 'AF', 'OC'.

Sendo estas regiões inseridas no script de inserção de dados e assumidas como sempre presentes na base de dados, partindo se assim do pressuposto da sua existência para efeitos de teste.

2.3.2 Partida Jogador e Multijogador

Optámos por manter em apenas uma relação a informação relativa a partidas multijogador. Visto acharmos que a separação da informação relativa aos jogadores e à sua pontuação demasiado custosa.

Assim optámos por não os separar, embora seguindo cegamente a normalização do modelo o devêssemos feito.

Seguindo a mesma lógica partida normal tem também presente o seu único jogador e a sua respetiva pontuação.

2.3.3 Partida

Relativamente à chave desta relação, formulada no enunciado como composta por, citando o mesmo: "... um número sequencial e único para cada jogo"

Logo sendo única para cada jogo e não apenas única, foi posta de parte a hipótese de se usar apenas um SERIAL para o id.

Sendo assim efetuamos uma função de nome **next_id** que obtém o id máximo para o presente jogo, e um gatilho de nome **partida_id_setter** que chama a função mencionada para todos os tuplos antes de um INSERT na tabela partida.

O tempo_fim pode ser null de forma a permitir partidas que ainda não terminaram e assim não têm ainda tempo de fim.

3

Detalhes de Implementação

3.1 Nota Prévia

Assim como é evidente nos outros capítulos deste relatório, uma vez mais, são aqui apenas referidos detalhes de implementação que achámos relevantes para a compreensão de como este trabalho foi efetuado e a razão das decisões tomadas no mesmo.

Não sendo assim mencionado aqui alíneas inteiras pedidas no relatório, por acharmos que a lógica das mesmas é trivial ou semelhante à de outras alíneas anteriores ou posteriores já explicadas.

3.2 Obter Total Pontos Jogador

Para obter o total de pontos de um jogador criamos uma função que recebe um id de jogador e soma todos os seus pontos obtidos em partidas normais com os obtidos em partidas multijogador.

Na soma destes valores rapidamente nos deparámos com um problema.

Se o jogador em causa não tiver participado em qualquer partida, ou qualquer partida do tipo normal ou multijogador o valor de pelo menos uma dessas somas será **NULL** e a soma de **NULL** com algo em postgres resulta também em **NULL**.

Sendo assim para solucionar este problema fizemos uso da função **COALESCE**, que retorna o primeiro valor não **NULL** de entre os parâmetros passados.

Assim bastou-nos, passar como primeiro parâmetro a soma das pontuações de uma das tabelas seguida de 0 como segundo parâmetro.

3.3 Validar Crachás

Sempre que uma partida é concluída, ou seja, é atribuído um valor de tempo_final, efetuamos a atribuição de crachás consoante os pontos obtidos, e os pontos requeridos para a obtenção dos mesmos.

Para isso é utilizado um gatilho que verifica caso exista uma atualização do tempo final de uma partida e caso a partida seja normal, são verificados os crachás a atribuir ao jogador cuja partida terminou. Caso a partida seja multijogador, é necessário verificar os vários jogadores que participaram na partida terminada para poder atribuir os respetivos crachás a cada um.

3.4 Iniciar Conversa

De forma a juntar um jogador a uma conversa, é relevante saber se este existe e se se encontra banido, pois caso contrário não poderá ser criada uma conversa, já que só fará sentido uma conversa com jogadores não banidos.

Após isto, o jogador será marcado como participante nessa conversa ao ser inserido na tabela apropriada, tabela **CONVERSA_PARTICIPANTES**.

No final, tal como pedido, é enviada uma mensagem a informar que o jogador se juntou à conversa, entendendo-se que esta mensagem deverá estar associada ao jogador em questão.

3.5 Juntar Conversa

Este procedimento tem um comportamento similar ao procedimento descrito anteriormente, recorrendo também ao mesmo princípio de que um jogador que se junta a uma conversa tem a mensagem informativa correspondente associada a si. É de realçar que é verificado o caso em que o jogador já está a participar na conversa, sendo desnecessário tentar a reinserção do mesmo.

3.6 Banir Jogadores

Foi implementado um mecanismo para a remoção de jogadores, que em vez de eliminar as suas informações da base de dados apenas os coloca como banidos.

É pedido que esta remoção se efetue sobre uma vista, mas visto que a mesma não ser mutável esta remoção é na verdade uma remoção sobre as relações que compõe a vista.

Assim, a remoção foi alterada através de um gatilho **BEFORE DELETE** que executa a função banFromView(), que por sua vez altera o estado de jogador para **Banido**.

3.7 Níveis Isolamento

Para definir os níveis de isolamento, utilizou-se princípios tais como a presença de duas leituras feitas à mesma informação ou até a possibilidade de duas transações poderem alterar a mesma informação, levando à inconsistência.

Assim, conseguiu-se definir para os vários procedimentos e funções os níveis de isolamento considerados como suficientes para manter uma boa consistência e desempenho da base de dados.

No entanto, para que cada procedimento e função tenha o seu nível de isolamento apropriado, foi necessário acrescentar procedimentos que definissem o nível de isolamento adequado. Efetuando se assim uma separação entre a definição do nível de isolamento e a lógica de acesso e / ou manipulação de dados.

3.7.1 Read Commited

O procedimento da alínea d (criar jogador) e as funções das alíneas e, f e g foram definidas com este nível de isolamento.

Visto que as funções apenas executam operações atómicas não deverão ocorrer situações de concorrência. Sendo que no caso da alínea d apenas se insere um jogador e nas alíneas e, f e g existe uma leitura apenas.

3.7.2 Repeatable Read

Quanto aos restantes procedimentos, entendeu-se que um nível de **REPEATABLE_READ** seria mais apropriado.

Verifica-se que é possível a existência de anomalias, como o **LOST UPDATE**, em alguns destes procedimentos, como os procedimentos restantes da alínea d, em que podem existir duas transações que procurem atualizar o estado de um mesmo jogador, ou então o caso da alínea h em que pode existir a tentativa de associar o mesmo crachá a um jogador.

No caso das conversas, existem várias leituras à mesma informação, sendo necessário que estas obtenham o mesmo resultado para a mesma transação, algo que o nível de isolamento de **READ_COMMITTED** não garante.

Para a alínea m, a função percorre várias tabelas fazendo uso de cursores, sendo necessário que a informação não se altere durante a sua execução.

Por fim, a alínea n efetua uma chamada a um procedimento anterior, **banirJogador()**, mantendo-se as motivos já mencionados para o uso do nível de isolamento **REPEATABLE_READ**.

3.8 Estatísticas

A realização das tabelas estatísticas referentes a Jogador e a Jogo.

Tendo sido já realizados as principais funções necessários para a realização das mesmas nas alíneas e, f e g.

Assim sendo a realização destas tabelas centra se no desenvolvimento de gatilhos que automatizem o uso destas funções reaproveitadas, e apenas mais algumas chamadas triviais.