Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores

Game On

Fase 1

Autores:	49419	Daniel Carvalho
	49465	Carolina Tavares
	49470	Ana Carolina Pereira

Grupo 2

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

Professor: Walter Vieira

8 de Maio de 2023

Índice

Lista de Figuras

Lista de Tabelas

1. Introdução

Este trabalho teve como objetivo o desenvolvimento de um modelo de dados adequado aos requisitos pretendidos e normalizado até à 3NF. Com isto, pretendeu-se também que fossem aplicados os tópicos lecionados nomeadamente controlo transacional, níveis de isolamento, vistas, procedimentos armazenados, gatilhos e funções, tendo como problema enunciado o seguinte:

A empresa "GameOn" pretende desenvolver um sistema para gerir jogos, jogadores e as partidas que estes efetuam. O sistema deve registar os dados dos jogadores que são identificados por um id gerado pelo sistema, tendo também o email e o username como valores únicos e obrigatórios. O jogador toma um dos estados 'Ativo', 'Inativo' ou 'Banido' e pertence a uma determinada região. Para cada região apenas se deve registar o seu nome. Os jogos têm como identificador uma referência alfanumérica de dimensão 10, um nome obrigatório e único e um url para uma página com os detalhes do jogo. Interessa registar os jogadores que compraram determinado jogo, a data e o preço associados à compra. Cada vez que o jogo é jogado, é criada uma partida com um número sequencial e único para cada jogo, devendo ser guardadas as datas e horas de início e de fim da partida. A partida 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. Devem ainda 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 está associada a uma região e apenas jogadores dessa região a podem jogar. De modo a recompensar os jogadores, cada jogo pode ter um conjunto de crachás que são atribuídos aos jogadores quando um limite de pontos nesse jogo é atingido. Para isso interessa registar o nome do crachá que é 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.

Deverão existir em tabelas próprias as estatísticas associadas aos jogadores e aos jogos. Interessa registar para cada jogador, 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 jogadores podem adicionar outros jogadores como amigos. Portanto interessa registar essa relação de amizade. É também possível criar uma conversa entre vários jogadores com um

identificador gerado pelo sistema e um nome. Associado à conversa existem mensagens com 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.

A partir do texto foi elaborado um Modelo Entidade-Associação (Figura 1) e um Modelo Relacional (Tabela 1).

2. Resolução das Alíneas

2.1 Alínea a) Criar o modelo físico

De forma a conceber o modelo físico dos requisitos do sistema, foi realizado um *modelo EA*, como apresentado na *figura 1*. A partir do modelo, as entidades, representadas por retângulos, correspondem a tabelas, assim como relações N:N, em que nessas tabelas apenas serão consideradas as chaves primárias de cada entidade da relação, no caso de relações N:1, a entidade do lado N irá adquirir a chave primária da entidade do lado 1. Neste modelo, existe também uma relação Disjoint, que irá gerar mais duas tabelas, neste caso. Cada tabela encontra-se acompanhada dos respectivos atributos, representados no modelo como elipses. Desses atributos, os que se apresentam sublinhados correspondem às chaves primárias de cada entidade. Assim, serão criadas 17 tabelas sendo elas:

- Jogador
- Jogos
- Compra
- Partida
- Jogar
- PartidaNormal
- PartidaMultijogador
- PontuacaoMultijogador
- Cracha
- Conversa
- Mensagem
- AssociarCracha
- CriarConversa
- JuntarConversa
- Amizade
- Jogador_Estat
- Jogo_Estat

2.2 Alínea b) Remover o modelo físico

Para realizar a remoção do modelo, foram utilizadas 2 instruções, sendo elas:

- **delete from ___nome_da_tabela___;**
- **drop table if exists ___nome_da_tabela___**

A primeira realiza apenas a eliminação dos dados na tabela, a segunda elimina a tabela e correspondentes dados.

2.3 Alínea c) Preenchimento inicial da base de dados

Após a criação das tabelas, as mesmas encontram-se vazias, não possuem quaisquer dados, daí a necessidade de serem criados dados iniciais para a base de dados, de forma a ser possível a utilização da mesma nas alíneas seguintes, para tal foi utilizada a instrução:

- **insert into ___nome_da_tabela___ (__atributos__) values**

Em que são colocados entre parênteses dados que respeitam o definido na criação de cada tabela, podem ser adicionadas várias linhas de dados.

2.4 Alínea d) Criar jogador e respectivos dados, e definir o estado do jogador

Para a realização desta alínea, foram criados dois *procedures* distintos para cada funcionalidade. O *procedure* **novoJogador** que recebe como parâmetro o *username, email* e *região* do novo jogador que se pretende criar, realizando um insert na tabela *Jogador* que por '*default*' coloca o estado do jogador como '*Ativo*'. O segundo *procedure*, denominado de **InativoOuBanido**, recebe como parâmetro um *booleano* e o *username* do jogador que se pretende desativar ou banir, que em caso de ser passado o valor '*true*' realiza um update da tabela *Jogador*, colocando o seu estado a '*Banido*', caso seja passado o valor '*false*', realiza-se um update da mesma tabela, no entanto, colocando o estado do jogador a '*Inativo*'.

2.5 Alínea e) Função totalPontosJogador

A função **totalPontosJogador** recebe como parâmetro o *id* de um jogador (*int*) e retorna o *número de pontos* que o jogador obteve (*int*). Para implementar esta função, realiza-se a união entre as tabelas *PartidaNormal* e *PontuacaoMultijogador*, de forma a obter todas as partidas jogadas, de seguida seleciona-se as linhas onde se encontra o *id* do jogador dado como parâmetro, a partir desta filtragem realiza-se um select da soma das pontuações do jogador sobre o qual é feito um cast para *int*.

2.6 Alínea f) Função totalJogosJogador

A função **totalJogosJogador** tem como parâmetro o *id* de um jogador e retorna o *número total de jogos diferentes* nos quais o jogador participou. Na implementação desta função, é passado *idjogador* (*int*) e é retornado uma t*abela com apenas uma coluna*, *totalJogos* (*int*), para tal é utilizada uma *query*, dentro da qual se realiza a união da tabela *PartidaNormal* e *PontuacaoMultijogador*, uma vez que são as tabelas que contêm como atributos tanto *id* de jogadores como os *números das partidas*, sendo assim possível ver as partidas realizadas pelo jogador dado como parâmetro, realiza-se de seguida a junção do resultado da união com a tabela *Jogar* onde o número da partida é igual, desse resultado realiza-se a seleção das linhas da tabela do jogador pretendido e os resultados são agrupados por esse mesmo id. Por fim, é feito um select da contagem de jogos nos quais o jogador participou acompanhado de um cast para o tipo *int*.

2.7 Alínea g) Função PontosJogoPorJogador

A função **PontosJogoPorJogador** tem como parâmetro a *referência de um jogo* (*varchar(10)*) e retorna uma *tabela com duas colunas*, sendo elas *identificador de jogador* (*int*) e o *total de pontos* (*int*). Na implementação da função é usada uma *query*, na qual sobre a tabela *Jogar* é feita uma junção com a união entre *PartidaNormal* e *PontuacaoMultijogador*, dentro da união as pontuações são selecionadas e somadas, e os valores são agrupadas segundo o *id* do jogador e *número da partida* (este último é apenas feito para ser possível a seleção do número da partida) obtendo assim a pontuação total de cada jogador nos diferentes modos de jogo. De seguida, é realizada a seleção das linhas com o id do jogo igual ao dado como parâmetro e agrupado por *id* dos diferentes jogadores. Por fim, realiza-se um select de cada *id* do jogador e a soma das pontuações nos diferentes modos de jogo, para ambos é realizado um cast para o tipo *int*.

2.8 Alínea h) Stored Procedure associarCrachá

O *stored procedure* **associarCracha** recebe três parâmetros de entrada - *id* (*do tipo int*), *id*t (*do tipo varchar(10)*) e um *nome* (*do tipo varchar*). Desta forma, o objetivo deste *procedure* consiste em verificar se um jogador atingiu o limite de pontos necessário para obter um crachá e caso a condição seja verdadeira, então associar um crachá a esse jogador.

Assim, optou-se por realizar uma condição *if* que consistia em fazer duas consultas à base de dados. A primeira, retorna a pontuação máxima do jogador em todos os jogos em que ele participou, enquanto que a segunda retorna o limite de pontos necessários para obter o crachá. Caso a pontuação máxima do jogador for menor que o limite de pontos necessário para obter o crachá, a função retorna a mensagem "O jogador não atingiu o limite de pontos para obter este crachá.". Caso contrário, criou-se previamente uma tabela *associarCrachá* de forma a que os valores dados como entrada do *procedure* fossem assim adicionados.

2.9 Alínea i) Stored Procedure inciarConversa

O *stored procedure* **iniciarConversa** recebe dois parâmetros de entrada - *id_jogador* (*do tipo int*) e um *nome* (*do tipo varchar*). Assim, o objetivo deste *procedure* é iniciar uma nova conversa entre jogadores e adicionar a conversa à tabela conversas. Desta forma, é inserido, na tabela conversas, um novo registo com o nome da conversa (nome).

2.10 Alínea j) Stored Procedure juntarConversa

O *stored procedure* **juntarConversa** recebe dois parâmetros de entrada - jogador_id (*do tipo int*) e *conversa_id* (*do tipo int*). Desta forma, o objetivo deste *procedure* é juntar um jogador a uma conversa já existente inserindo na tabela *juntarConversa* (criada previamente).

Assim, foi necessário fazer uma verificação que consistia em verificar se a conversa com o *id* passado como entrada, existia na tabela conversa. Se a *query* retornar pelo menos 1 linha, é adicionado um novo registo na tabela *juntarConversa* com os valores passados como entrada do *procedure*. Caso contrário, a função retorna a mensagem "A conversa não existe."

2.11 Alínea k) Stored Procedure enviarMensagem

O *stored procedure* **enviarMensagem** recebe três parâmetros de entrada - *id_jogador* (*do tipo int*), *id_conversa* (*do tipo int*) e *mensagem* (*do tipo text*). Este *procedure* tem como objetivo permitir que um jogador envie uma mensagem para uma conversa já existente registando essa mensagem na tabela mensagens. Numa primeira fase, é inserida uma nova linha na tabela mensagens com a data e hora atuais, a mensagem escrita pelo jogador e o id da conversa. Posto isto é enviada a mensagem "Mensagem enviada com sucesso para a conversa % pelo jogador %".

NOTA:

É importante referir que para os stored procedures das alíneas d), i) e k) foi utilizado o nível de isolamento *read commited* uma vez que este especifica que as instruções não podem ler dados que foram modificados, mas não confirmados por outras transações. Os dados podem ser alterados por outras transações entre declarações individuais dentro da transação atual, resultando em leituras não repetíveis.

Enquanto que para as alíneas h) e j) foi utilizado o nível de isolamento *repeatable read*, uma vez que este, garante que qualquer dado lido não pode ser alterado, isto é, se a transação ler os mesmos dados novamente então vai encontrar os dados lidos anteriormente, inalterados e disponíveis para leitura.

2.12 Alínea l) Vista jogadorTotalInfo

Esta vista permite 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*'. Para implementação desta vista, partindo da tabela *Jogador* é utilizado um select para o *id*, *username*, *email* e *estado* do jogador, dentro desta instrução são chamadas as funções *totalJogosJogador* e *totalPontosJogador* em que são passados como parâmetro cada *id* dos diferentes jogados. Finalmente, realiza a seleção das linhas em que o estado do jogador tem de ser diferente de '*Banido*'.

2.13 Alínea m) Mecanismos automáticos de atribuição de crachás

Para esta alínea, foram criados dois *triggers* e uma função, em que o *trigger* **atribuir_chacha_pn**, apenas é executado após um *insert* na tabela *PartidaNormal*, referenciando a tabela como '*novo*', e para cada linha da mesma tabela a função será executada. O segundo *trigger*,

atribuir_cracha_pm, é executado após um *update* na tabela *Partida*, referência a tabela como '*novo*', e para cada linha da tabela, quando o valor de *dtfim* e *hfim* (data de fim, hora de fim) são diferentes de *null* a função é executada. Como referido anteriormente, ao fim de cada *trigger* uma função é executada, em ambos os casos, sendo ela a função **atribuir_cracha** que não tem parâmetro e retorna um *trigger*, nesta função é criado um *if* em que caso a pontuação seja superior ao limite de pontos associado a algum crachá presente na tabela, haverá um insert na tabela Atribuir como o número da partida e o id do jogador associado à tabela referenciada como '*novo*'. O retorno desta função é *null*, uma vez que o valor retornado não é utilizado.

2.14 Alínea n) Mecanismos automáticos de execução DELETE sobre a vista jogadorTotalInfo

O objetivo destes mecanismos é a execução da instrução *DELETE* sobre a *vista* **jogadorTotalInfo** colocando os jogadores envolvidos na vista, no estado '*Banido*'. Nesta implementação foi criado um *trigger*, **banDeletePlayerInfo**, que realiza um '*instead of*', ou seja, em vez de detetar um *delete* sobre a *vista* **jogadorTotalInfo**, para cada linha presente na vista executa a função **trataLinha**. Por sua vez, a função **trataLinha** não recebe qualquer parâmetro e retorna um *trigger*. Dentro da função, realiza-se uma condição *if* que em caso de TG (*trigger operation*) seja qualquer outra que não '*DELETE*', será retornada uma *excepção* que envia a mensagem '*gatilho inválido*'. Após a verificação que TG é '*DELETE*', então será realizado um *update* sobre a tabela *Jogador*, onde o estado será alterado para '*Banido*' onde o *id* do jogador na tabela *Jogador* iguala a *vista* **jogadorTotalInfo**. Esta função retorna um valor '*new*' que corresponde a um valor que não será utilizado.

3. Conclusão

Durante a realização do trabalho, foi possível rever vários conteúdos lecionados na cadeira de precedência à presente (ISI), nomeadamente, na elaboração do Modelo Entidade-Associação (Modelo EA - Figura 1) e no Modelo Relacional (Tabela 1), obtido através do EA.

Estes modelos foram alterados durante o processo de realização das alíneas, uma vez que segundo o contexto de diferentes alíneas, conferiu a realização de ajustar o modelo de forma a alcançar a solução final do Modelo EA e consequente Modelo Relacional.

Nesta fase do projeto, foi conseguida a consolidação do uso do controlo transacional, níveis de isolamento e criação de procedimentos armazenados, gatilhos e funções.

Referências

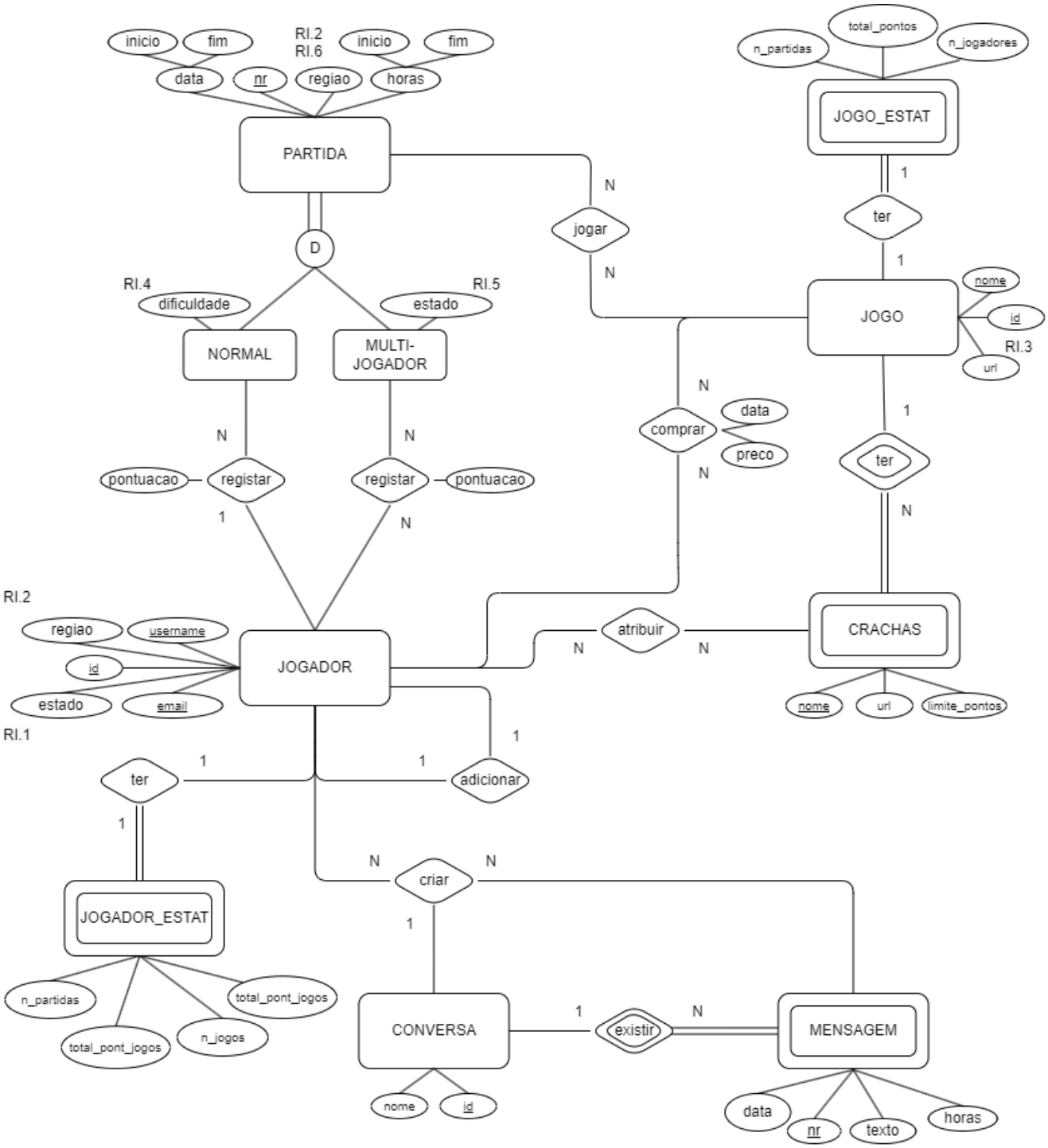
[1] https://2223moodle.isel.pt/pluginfile.php/1204845/mod_resource/content/3/SisInf_M2_SP_Trig_Func%28v3%29.pdf.

[2] https://2223moodle.isel.pt/pluginfile.php/1201190/mod_resource/content/9/SisInf_M1_Transacc%CC%A7o%CC%83es%28v4%29.pdf

[3] https://www.postgresqltutorial.com/

[4] Ramez Elmasri and Shamkant B. Navathe, “Fundamentals of Database Systems Seventh Edition”, 2015.

A.1 Diagramas da Aplicação



RI :
1. Estado do jogador varia entre "ativo", "inativo" e "banido";
2. Para cada região apenas se deve registar o seu nome;
3. Os jogos têm como identificador uma referência alfanumérica de dimensão 10;
4. O grau de dificuldade varia entre os valores 1 e 5;
5. O estado da partida multijogador, pode tomar os valores 'Por iniciar', 'A aguardar jogadores', 'Em curso' e 'Terminada';
6. Cada partida está associada a uma região e apenas jogadores dessa região a podem jogar.

Figura 1 – Modelo Entidade-Associação.

A.2 Modelo Relacional

JOGADOR (id, username, email, estado, regiao)

JOGOS (idt, nome, url)

COMPRA (idjogador, idjogo, dt, preco)
FK: {idjogador} de JOGADOR.id
{idjogo} de JOGOS.idt

PARTIDA (nr, dtinicio, dtfim, hinicio, hfim, regiao)

JOGAR (idjogo, nrpartida)
FK: {idjogo} de JOGOS.idt
{nrpartida} de PARTIDA.nr

PARTIDANORMAL (nr, id, dificuldade, pontuacao)
FK: {nr} de PARTIDA.nr
{id} de JOGADOR.id

PARTIDAMULTIJOGADOR (nr, estado)
FK: {nr} de PARTIDA.nr

PONTUACAOMULTIJOGADOR (id, nr, pontuacao)
FK: {id} de JOGADOR.id
{nr} de PARTIDA.nr

CRACHA (idt, nome, url_img, lim_pontos)
FK: {idt} de de JOGOS.idt

CONVERSA (idnt, nome)

MENSAGENS (nr, id_jogador, dt, hora, texto, idnt)
FK: {idnt} de CONVERSA.idnt
{id_jogador} de JOGADOR.id

ASSOCIARCRACHA (id, idt, nome)
FK: {idt, nome} de CRACHA.id/CRACHA.nome

CRIARCONVERSA (jogador_id, conversa_id)
FK: {jogador_id} de JOGADOR.id
{conversa_id} de CONVERSA.idnt

JUNTARCONVERSA (jogador_id, conversa_id)
FK: {jogador_id} de JOGADOR.id
{conversa_id} de CONVERSA.idnt

AMIZADE (idjogador1, idjogador2)
FK: {idjogador1} de JOGADOR.id
{idjogador2} de JOGADOR.id

JOGADOR_ESTAT (idjogador, n_partidas, n_jogos, total_pontos_jogos_partidas)
FK: {idjogador} de JOGADOR.id

JOGO_ESTAT (idjogo, n_partidas, total_pontos, n_jogadores)
FK: {idjogo} de JOGOS.idt