
**Área Departamental de Engenharia de Electrónica e Telecomunicações e de Computadores**

# GameOn - F1

48253 : Carlos Guilherme Cordeiro Pereira (A48253@alunos.isel.pt)

48281 : Adolfo Morgado (A48281@alunos.isel.pt)

48335 : Rodrigo Henriques Correia (A48335@alunos.isel.pt)

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

Professor : Mestre Walter Jorge Mendes Vieira

**16 – 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 da empresa fictícia '*GameOn*'.

Através do trabalho realizado pretendemos demonstrar conhecimento sobre desenho da base de dados, e sua implementação bem como a criação de funcionalidades para a manipulação de dados da mesma 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 enunciar lo em extensão, bastando mencionar trechos do mesmo.

# Abstract

As part of the first phase of the course's work assignment, this report aims to create a database for the proposed problem of the fictitious company '*GameOn*'.

Through the work carried out, we intend to demonstrate knowledge about the design of the database, and its implementation, as well as the creation of functionalities for the manipulation of its data through the way in which we solve the proposed problems.

This report is based on the assumption that the reader has access to the code developed within the scope of the same, therefore it is not necessary to state it in length, just mentioning partly from it.

# Índice

# Lista de Figuras

# Lista de Listagens

# 1

# Modelo Entidade-Associação

## 1.1 Caso em Estudo

O caso de estudo proposto foi da empresa *GameOn* esta empresa pretende desenvolver um sistema para gerir jogos, jogadores e as partidas que estes efetuam, tento ainda uma funcionalidade amizade entre jogares e uma conversa entre os mesmos.

Os jogadores para alem da sua informação pessoal podem ainda ter a sua propria estatistica como por exemplo a pontuação total dos jogos que estes jogaram como tambem uma lista de amigos. Os jogadores podem comprar jogos e para isso tivemos de registar o preço e a data da compra. Os jogos têm como informacao geral um url para a descricao um id alfanumerico e o proprio nome permitindo os jogadores jogar partidas do mesmo. Os Jogos ainda teem a sua propria estatistica, como por exemplo o numero de jogadores que jogaram-no, o numero de vezes que este foi jogado e o total de pontos feitos no mesmo. Os jogadores só podem jogar em partidas da sua região, e as partidas podem ser normais ou multi-jogador, para as partidas normais interesa registar a dificuldade, já para as partidas multi-jogador interesa registar o estado ('Por iniciar', 'A aguardar jogadores', 'Em curso' ou 'Terminada'). Terminada a partida os jogadores têm a possiblidade de ganhar charchas unicos para cada jogo se atigirem uma certa pontuação, este cracha tem uma imagem, um nome e um limite de pontos minimo requerido. Quanto ao sistema de conversas, para cada conversa e necessario existir um nome e cada mensagem tem um numero de ordem, o texto, a data e o identificador do jogador que a enviou.

## 1.2 Restrições de Integridade

Após o estudo dos requerimentos do sistema o grupo chegou a conclusao das sequintes restrições de integridade:

- **RI1** - Valor unico e obrigatorio para o *username* e email do jogador bem como o nome do jogo;
- **RI2** - O jogador apenas pode tomar um dos estados ('Ativo', 'Inativo' ou 'Banido');
- **RI3** - O identificador do jogo tem de ser uma sequência alfanumérica de dimensão 10;
- **RI4** - A dificuldade da partida normal tem de ser um valor entre 1 a 5;
- **RI5** - O estado da partida multi-jogador toma valores como: 'Por iniciar', 'A aguardar jogadores', 'Em curso' ou 'Terminada';
- **RI6** - A data de inicio mais antiga que data de fim para detalhes da partida;
- **RI7** - Data fim tem de ser mais recente que data inicio e pode ser nulable, ou seja a partida pode ainda nao ter terminado;
- **RI8** - Jogador tem participar na conversa para enviar mensagens, sendo uma restrição obvia aos membros do grupo a discusao da mesma não o é, com isto queremos dizer que e dificil de perceber se esta e uma restricao de integridade do modelo EA, se é uma regra de negocio ou onde chegamos a um consenso ambas;
- **RI9** - Jogador tem de pertencer a região onde esta a ser jogada a partida, tal como a restrição anterior o grupo chegou a um consenso de colocar esta restrição tanto em modelo EA como implementar a mesma como regra de negocio.

## 1.3 Mudanças ao EA ao longo do tempo

Numa primeira fase o grupo tinha construído o modelo com as estatísticas tanto de jogo como de jogador como sendo fraca de ambas as suas classes mãe, no entanto apos discussão com o professor este fez nos aperceber que a solução não passava por esse caminho, mas sim por usar as Superclasses e Subclasses e assim o fizemos, pois

a alternativa que tínhamos era uma associação de 1:1 com obrigatoriedade do lado de estatística esta sendo fraca de sua classe mãe.

Inicialmente também tínhamos tanto crachá como partida como uma entidade simples, mas rapidamente nos percebemos que ambas seriam fracas de jogo pois sem este não existem nenhum dos dois.

Já numa fase posterior tínhamos também na associação entre partida e jogador o atributo pontuação, mas em grupo conseguimos nos aperceber que isso não seria uma solução viável uma vez que existem partidas de multijogador. Na mesma altura apercebemos nos que região teria de ter uma associação para partida, algo que não tinha até agora, para que pudéssemos posteriormente implementar a restrição de que os jogadores e partida seriam na mesma região.

Apenas no final e que nos apercebemos que tal como partida a conversa, mensagem e jogador teriam de ter a mesma estratégia.

Após todo o processo de pensamento no caso de estudo chegamos ao seguinte modelo Entidade-Associação:

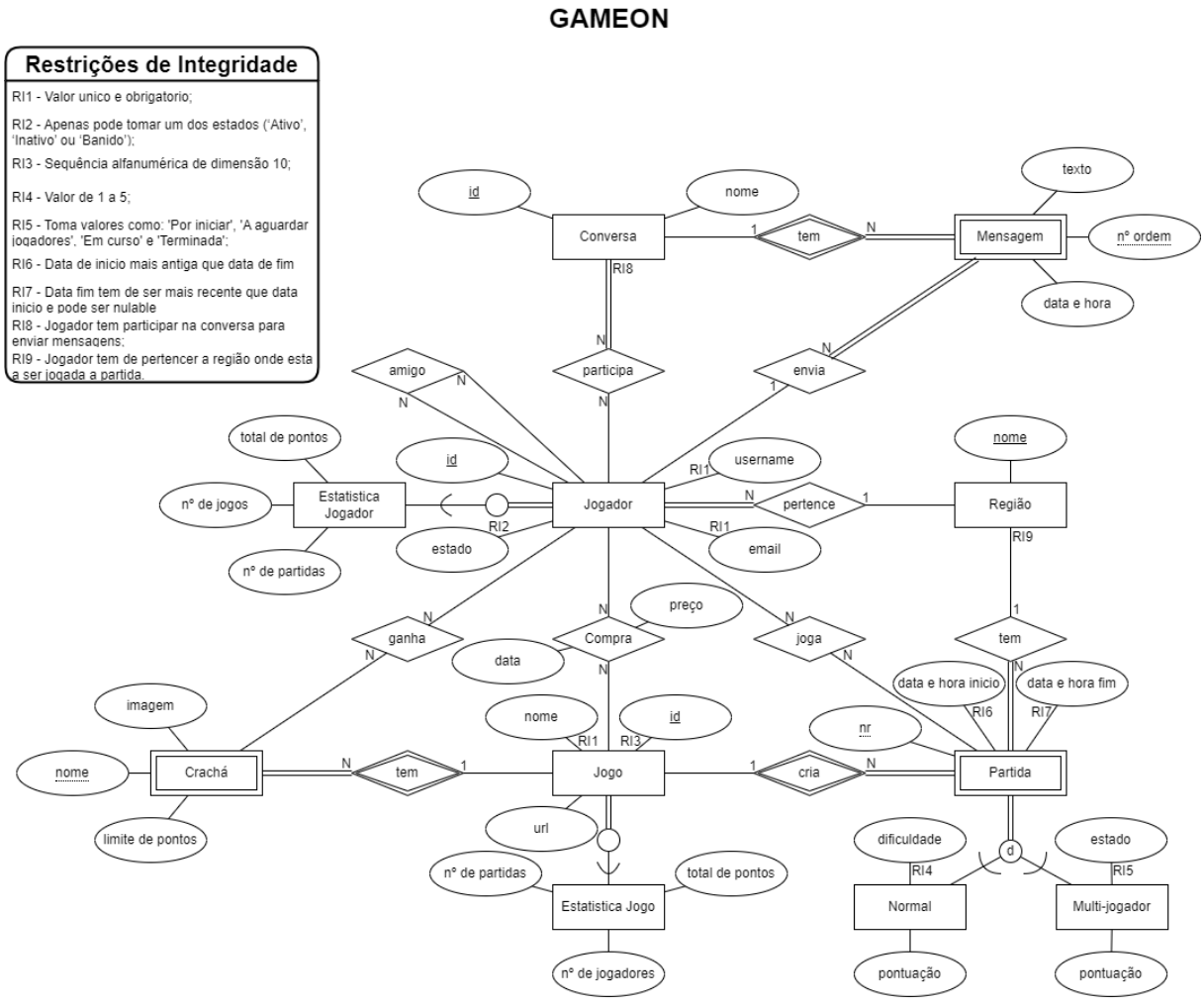


Figura 1.1: Diagrama EA

# 2

# Modelo Relacional

## 2.1 Entidades

Foram usadas as seguintes Entidades no modelo, mas para a passagem para o modelo relacional(passo intermedio entre modelo entidade-associação e modelo físico) seriam necessárias mais que demonstraremos seguidamente.

Regiao(nome)

Jogador(id, email, username, status)

Jogo(id, url, nome)

Conversa(id, nome)

Estática Jogador(nº de jogos, nº de partidas, total de pontos)

Estatística Jogo(nº de jogadores, nº de partidas, total de pontos)

Partida(nr, data e hora início, data e hora fim)

Normal(dificuldade, pontuação)

Multijogador(estado, pontuação)

Crachá(nome, imagem, limite de pontos)

Mensagem(id, texto, data e hora)

Compra(data, preço)

## 2.2 Passagem do modelo EA para Relacional

Na passagem do modelo EA para o modelo Relacional foi necessário criar o mapeamento das associações correspondessem as associações N:N tais como: Ganha (corresponde a associação entre crachá e jogador), Participa (corresponde a associação entre conversa e jogador) e Amigo (corresponde a associação entre jogador e jogador). Assim o mapeamento final das entidades ficou o seguinte:

Região(nome)

PK: Região(nome); FK:

Jogador(id, email, username, status, Região(nome))

PK: Jogador(id); FK: Região(nome)

Jogo(id, url, nome)

PK: Jogo(id); FK:

Conversa(id, nome)

PK: Conversa(id); FK:

Estatística Jogador(nº de jogos, nº de partidas, total de pontos, Jogador(id))

PK: ; FK: Jogador(id)

Estatística Jogo(nº de jogadores, nº de partidas, total de pontos, Jogo(id))

PK: ; FK: Jogo(id)

Partida(nr, data e hora início, data e hora fim, Jogo(id), Região(nome))

PK: Partida(nr); FK: Jogo(id), Região(nome)

Normal(dificuldade, pontuação, Partida(nr))

PK: Partida(nr); FK:

Multijogador(estado, pontuação, Partida(nr))

PK: Partida(nr); FK:

Crachá(nome, imagem, limite de pontos, Jogo(id))

PK: Crachá(nome); FK: Jogo(id)

Mensagem(nº de ordem, texto, data e hora, Conversa(id), Jogador(id))

PK: Mensagem(nº de ordem); FK: Conversa(id), Jogador(id)

Joga (Jogador(id), Partida(nr))

PK: Jogador(id), Partida(nr); FK: Jogador(id), Partida(nr)

Compra(data, preço, Jogador(id), Jogo(id))

PK: Jogador(id), Jogo(id); FK: Jogador(id), Jogo(id)

Ganha(Crachá(nome), Jogador(id))

PK: Crachá(nome), Jogador(id); FK: Crachá(nome), Jogador(id)

Participa(Jogador(id), Conversa(id))

PK: Jogador(id), Conversa(id); FK: Jogador(id), Conversa(id)

Amigo(Jogador(id), Jogador(id))

PK: Jogador(id), Jogador(id); FK: Jogador(id), Jogador(id)

# 3

# Modelo Fisico

## 3.1 Domínios

Para a implementação da nossa solução achamos que os tipos existentes em PostgreSQL não traduziam a total realidade do nosso problema, devido a isso criamos domínios, estes que são apenas verificações de Regex para URL, EMAIL e como restrição de integridade ALFANUMERIC.

```
1 CREATE DOMAIN ALPHANUMERIC AS VARCHAR(10) CHECK (VALUE ~* '^[A-
    Z0-9]+$');
```

Listagem 3.1: Dominio *ALPHANUMERIC*

## 3.2 Tabelas

A tabela jogo embora não tenha muitas restrições (*Constrains*) implementa dois dos três tipos criados para a implementação da base de dados.

```
1 -- Jogo
2 CREATE TABLE IF NOT EXISTS jogo(
3     id          ALPHANUMERIC,
4     nome        VARCHAR(50) NOT NULL,
5     url         URL NOT NULL,
```

```
6
7       UNIQUE (nome),
8
9       CONSTRAINT pk_jogo PRIMARY KEY (id)
10  );
```

Listagem 3.2: Tabela jogo

## 3.3 Regras de Negócio

A rotina ***checkJogadorPartidaRegiao*** é uma das duas rotinas implementadas para cumprir com as regras de negócio que não conseguiram ser implementadas como restrições de integridade, esta verifica apenas de o jogador pode jogar uma partida, isto é apenas aceita o jogador na partida se este pertencer a mesma região que a partida está a ser jogada.

```
1  CREATE FUNCTION checkJogadorPartidaRegiao()
2      RETURNS TRIGGER LANGUAGE plpgsql
3  AS
4  $$
5      DECLARE
6          regiao_nome VARCHAR(50);
7      BEGIN
8          SELECT partida.nome_regiao INTO regiao_nome FROM partida
       WHERE partida.nr == NEW.nr_partida;
9          IF (regiao_nome != (SELECT jogador.nome_regiao FROM
      jogador WHERE jogador.id == NEW.id_jogador)) THEN
10             RAISE EXCEPTION 'O jogador nao pertence a regiao da
      partida';
11         END IF;
12     END;
13 $$;
14
15 CREATE TRIGGER checkJogadorPartidaRegiao BEFORE INSERT ON joga
16     FOR EACH ROW
17     EXECUTE FUNCTION checkJogadorPartidaRegiao();
```

Listagem 3.3: Rotina checkJogadorPartidaRegiao

## 3.4 Funcionalidades

Esta secção destina-se apenas as funcionalidades requeridas diretamente no projeto.

### 3.4.1 Funções

Uma das funções requeridas foi a função **PontosJogoPorJogador**, esta tem como objetivo retornar 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.

```
1  CREATE FUNCTION PontosJogoPorJogador(jogo_id ALPHANUMERIC)
2      RETURNS TABLE (jogador_id INT, total_pontos INT) LANGUAGE
       plpgsql
3  AS
4  $$
5      BEGIN
6          RETURN QUERY SELECT joga.id_jogador, totalPontosJogador(
       joga.id_jogador) FROM joga WHERE joga.id_jogador IN (
7          SELECT joga.id_jogador FROM joga WHERE joga.nr_partida
       IN (
8          SELECT partida.nr FROM partida WHERE partida.id_jogo ==
       jogo_id));
9      END;
10 $$;
```

Listagem 3.4: Função PontosJogoPorJogador

### 3.4.2 Procedimentos

O procedimento seguinte tem como objetivo mudar o estado do jogador, no entanto para criar algum tipo de controlo de verificação achamos que seria interessante avisar o utilizador de que se o id que este fornecer não for um id de um utilizador conhecido a função notifica do acontecimento. Também acrescentamos a verificação de ver se o estado atual e o mesmo que o estado futuro, caso isso se confirme não alteramos e notificamos o utilizador do sucedido.

```
1  CREATE PROCEDURE updateEstadoJogador(id_jogador INT, new_estado
       VARCHAR(10))
```

```
2     LANGUAGE plpgsql
3 AS
4 $$
5     BEGIN
6         -- Checks
7         IF (id_jogador NOT IN (SELECT jogador.id FROM jogador))
    THEN
8             RAISE NOTICE 'jogador not found';
9         END IF ;
10        IF ((SELECT jogador.estado FROM jogador WHERE jogador.id
     == id_jogador) == new_estado) THEN
11            RAISE NOTICE 'jogador already has this estado';
12        END IF ;
13        -- expected
14        UPDATE jogador SET estado = new_estado WHERE jogador.id
    = id_jogador;
15    END ;
16 $$;
```

Listagem 3.5: Procedimento updateEstadoJogador

### 3.4.3 Gatilhos

Este gatilho foi criado á necessidade da instrução *DELETE* sobre a vista jogadorTotalInfo permita colocar os jogadores envolvidos no estado "Banido".

```
1 CREATE FUNCTION banirJogador()
2     RETURNS trigger LANGUAGE plpgsql
3 AS
4 $$
5     DECLARE
6         jogador_id INT;
7     BEGIN
8         SELECT jogador.id INTO jogador_id FROM jogador WHERE
    jogador.username == OLD.username;
9         UPDATE jogador SET estado = 'Banido' WHERE jogador.id ==
     jogador_id;
10    END;
11 $$;
```

```
12
13 CREATE TRIGGER banirJogador INSTEAD OF DELETE ON
       jogadorTotalInfo
14      FOR EACH ROW
15      EXECUTE FUNCTION banirJogador();
```

Listagem 3.6: Gatilho banirJogador

### 3.4.4 Vistas

Para finalizar esta vista foi criada para permitir 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". Não foi usada a tabela de estatísticas pois foi explicitamente proibido o mesmo para esta funcionalidade no projeto.

```
1 CREATE VIEW jogadorTotalInfo AS
2      SELECT jogador.id, jogador.estado, jogador.email, jogador.
      username, totalJogosJogador(jogador.id) AS total_jogos,
3      totalPartidasJogador(jogador.id) AS total_partidas,
      totalPontosJogador(jogador.id) AS total_pontos
4      FROM jogador WHERE jogador.estado != 'Banido';
```

Listagem 3.7: Vista jogadorTotalInfo

## 3.5 Detalhes

# 4

# Validação

## 4.1 Controlo transacional

## 4.2 Testagem

# 5

# Conclusão

Na execução deste trabalho reforçamos as nossas capacidades de desenvolvimento de uma base de dados relacional, desde a criação de um modelo físico ao desenvolvimento de funcionalidades capazes de ser executáveis numa aplicação.

Com este trabalho os membros do grupo também conseguiram consolidar os conteúdos programáticos lecionados em aula, nomeadamente: tabelas, ligações entre as mesmas, funções, procedimentos armazenados, vistas, *triggers* entre outros.

Contudo ainda não foi possível, devido à capacidade do grupo, definir níveis de isolamento para as funcionalidades acima ilustradas.

Em suma, apesar de algumas adversidades e imperfeições o grupo conseguiu desenvolver os modelos de uma base dados e implementá-la com sucesso.