

**Armazenamento e gestão de jogos**

|  | Miguel Marques 47204 |
| --- | --- |
|  | Paulo Rosa 44873 |

| Professor | Afonso Remédios |
| --- | --- |
|  |  |

Relatório da fase 1 realizado no âmbito de Sistemas de Informação,  
do curso de licenciatura em Engenharia Informática e de Computadores  
Semestre de Verão 2022/2023

Março de 2023

# Resumo

Neste projeto, é desenvolvido um modelo de dados que pretende organizar os dados de uma empresa fictícia de jogos.  
 É pedido uma série de requisitos, regras de negócio a criação de funções, triggers, views etc. Para a sua concretização é usada a linguagem PosgreSQL.

**Palavras-chave:** postgreSQL; Gestão; Jogos

# Abstract

In this project, a data model is developed that intends to organize the data of a fictitious game company.

It’s requested for us to do a series of requirements, business rules, the creation of functions, triggers, views, etc. For its implementation, the PosgreSQL language is used.

**Keywords:** postgreSQL; Games; Management

**Índice**

# 

[Resumo 1](#_30j0zll)

[Abstract 2](#_1fob9te)

[Lista de Figuras 4](#_tyjcwt)

[Lista de Tabelas 5](#_cx2f5lyhgmpk)

[1. Introdução 1](#_1t3h5sf)

[2. Desenvolvimento 2](#_1ksv4uv)

[3. Explicação da nossa resolução dos exercícios 10](#_kgxms9vhe0o)

[4. Conclusão 13](#)

[Webgrafia e recursos usados 14](#_1hmsyys)

# Lista de Figuras

[Diagrama do tipo Entidade-Associação da nossa solução: 8](#_mw2l8kkucof3)

[Diagrama da chaves estrangeiras 9](#_caz74012j9zz)

# 

# Lista de Tabelas

[Tabela: REGIAO 2](#_7rgbssz7ggcb)

[Tabela: AMIGOS 3](#_uwype812pxtx)

[Tabela: CHAT\_GROUP 3](#_iypwtq61h5ug)

[Tabela: CHAT\_GROUP\_PARTICIPANT 3](#_87v8smfaxtak)

[Tabela: CHATS 4](#_1nhdrjmpwlmh)

[Tabela: JOGO 4](#_v5vvm77p6u61)

[Tabela: COMPRA 4](#_kgeij9kdzb9x)

[Tabela: CRACHA 5](#_h09sboq4lku5)

[Tabela: CRACHAS\_ATRIBUIDOS 5](#_pqmxqw5gdz2b)

[Tabela: PARTIDA 5](#_ecoewkd2841o)

[Tabela: PARTIDA\_NORMAL 6](#_s2ytoffx106k)

[Tabela: PARTIDA\_MULTIJOGADOR 6](#_fvixuj8yjyo7)

[Tabela: PONTUACAO\_JOGADOR 6](#_4g7b3pxkacuc)

[Tabela: ESTATISTICA\_JOGADOR 7](#_54xh6tv5co4h)

[Tabela: ESTATISTICA\_JOGO 7](#_art76h3akvod)

# 

# Introdução

Para este trabalho é proposto a realização de um modelo de dados com as seguintes relações resumidas:

* Existem regiões identificados por uma string única
* Existem Jogadores com um “id” único
* Existe o registo dos amigos
* Existe o registo dos grupos de chat’s e as suas mensagens
* Existe o registo das estatísticas associadas a cada jogador
* Existe o registo dos jogos
* Existe o registo das compras destes jogos
* Existem estatísticas associadas ao jogos
* Existe o registo das partidas, que podem ser de partida normal (solo) ou multijogador
* Existe o registo da pontuação de cada partida e associado ao jogador envolvido
* Existem crachás com nomes únicos para cada jogo e há o registo de que jogadores tem esses crachás

# Desenvolvimento

Aqui listamos em mais detalhe as tabelas criadas. Indicamos as chaves primárias, estrangeiras, os tipos das colunas, etc. Nota: chaves primárias já são não null por default, por isso é trivial indicar na tabela também.

**Enums criados:**

regiao\_enum(‘EU’, ‘NA’, ‘ASIA’),

estado\_enum(‘Ativo’, ‘Inativo’, ‘Banido’)

## Tabela: REGIAO

Chave(s) primária: id

Chave(s) estrangeiras:

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | regiao\_enum |  |

**Tabela: JOGADOR**

Chave(s) primária: id

Chave(s) estrangeiras: regiao ➜ id de REGIAO

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | INT |  |
| username | TEXT | Não null, único |
| email | TEXT | Não null, único |
| estado | estado\_enum | Não null |
| regiao | regiao\_enum | Não null |

## Tabela: AMIGOS

Chave(s) primária: id\_jogador\_pedinte e id\_jogador\_pedinte (junto)

Chave(s) estrangeiras:

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_jogador\_pedinte | INT |  |
| id\_jogador\_destino | INT |  |
| aceite | BOOLEAN | Não null, default false |

## Tabela: CHAT\_GROUP

Chave(s) primária: id

Chave(s) estrangeiras: id\_criador ➜ id de JOGADOR

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | INT |  |
| nome | TEXT | Não null |
| id\_criador | INT | Não null |
| participantes | INT ARRAY | Não null |

## Tabela: CHAT\_GROUP\_PARTICIPANT

Chave(s) primária: id\_chat\_group e id\_jogador

Chave(s) estrangeiras: id\_chat\_group ➜ id de CHAT\_GROUP, id\_jogador ➜ id de JOGADOR

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_chat\_group | INT |  |
| id\_jogador | INT | Não null |

## 

## Tabela: CHATS

Chave(s) primária: id

Chave(s) estrangeiras: id\_group ➜ id de CHAT\_GROUP

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | INT |  |
| id\_group | INT | Não null |
| dateAndTime | TIMESTAMP | Não null |
| mensagem | TEXT | Não null |

## Tabela: JOGO

Chave(s) primária: id

Chave(s) estrangeiras:

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | TEXT | Size == 10 |
| url | TEXT | Não null |
| nome | TEXT | Não null |

## Tabela: COMPRA

Chave(s) primária: id

Chave(s) estrangeiras: id\_jogador ➜ id de JOGADOR, id\_jogo ➜ id de JOGO

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | INT |  |
| dataa | TIMESTAMP | Não null |
| preco | DECIMAL(6, 2) | Não null |
| id\_jogador | INT | Não null |
| id\_jogo | CHAR(10) | Não null |

## Tabela: CRACHA

Chave(s) primária: id\_jogo e nome (junto)

Chave(s) estrangeiras: id\_jogo ➜ id de JOGO

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_jogo | CHAR(10) |  |
| nome | TEXT | Não null |
| pontosAssociados | INT | Não null |
| url | TEXT | Não null |

## Tabela: CRACHAS\_ATRIBUIDOS

Chave(s) primária: id\_jogador, id\_jogo e nome (junto)

Chave(s) estrangeiras: id\_jogador ➜ id de JOGADOR, (id\_jogo, nome) ➜ (id\_jogo, nome) de CRACHA

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_jogador | INT | Não null |
| id\_jogo | CHAR(10) | Não null |
| nome | TEXT | Não null |

## Tabela: PARTIDA

Chave(s) primária: id

Chave(s) estrangeiras: id\_jogo ➜ id de JOGO

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | INT |  |
| id\_jogo | CHAR(10) | Não null |
| data\_inicio | TIMESTAMP | Não null |
| data\_fim | TIMESTAMP | Não null |
| regiao | regiao\_enum | Não null |

## Tabela: PARTIDA\_NORMAL

Chave(s) primária: id\_partida

Chave(s) estrangeiras: id\_partida ➜ id de PARTIDA

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_partida | INT |  |
| dificuldade | INT | Não null, 1 <= dificuldade <= 5 |

## Tabela: PARTIDA\_MULTIJOGADOR

Chave(s) primária: id\_partida

Chave(s) estrangeiras: id\_partida ➜ id de PARTIDA

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_partida | INT |  |
| estado | TEXT | estado ==  'Por iniciar' | 'A aguardar jogadores' | 'Em curso’ | 'Terminada' |

## Tabela: PONTUACAO\_JOGADOR

Chave(s) primária: id\_partida e id\_jogador (junto)

Chave(s) estrangeiras: id\_partida ➜ id de PARTIDA, id\_jogador ➜ id de JOGADOR

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_partida | INT | Não null |
| id\_jogador | INT | Não null |
| pontuacao | INT | Não null |

## Tabela: ESTATISTICA\_JOGADOR

Chave(s) primária: id\_jogador

Chave(s) estrangeiras: id\_jogador ➜ id de JOGADOR

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id\_jogador | INT |  |
| numJogosQueComprou | INT | Não null |
| numPartidas | INT | Não null |
| totalPontos | INT | Não null |

## Tabela: ESTATISTICA\_JOGO

Chave(s) primária: id

Chave(s) estrangeiras: id ➜ id de JOGO

| Coluna | Tipo | Restrições |
| --- | --- | --- |
| id | CHAR(10) |  |
| totalPartidas | INT | Não null |
| totalPontos | INT | Não null |
| numJogadoresCompraram | INT | Não null |

### Diagrama do tipo Entidade-Associação da nossa solução:

Imagem 1: Diagrama Entidade-Associação

### Diagrama da chaves estrangeiras

Este diagrama, também chamado de diagrama da base da dados, pretende ilustrar de uma forma mais simples as relações entre as tabelas.

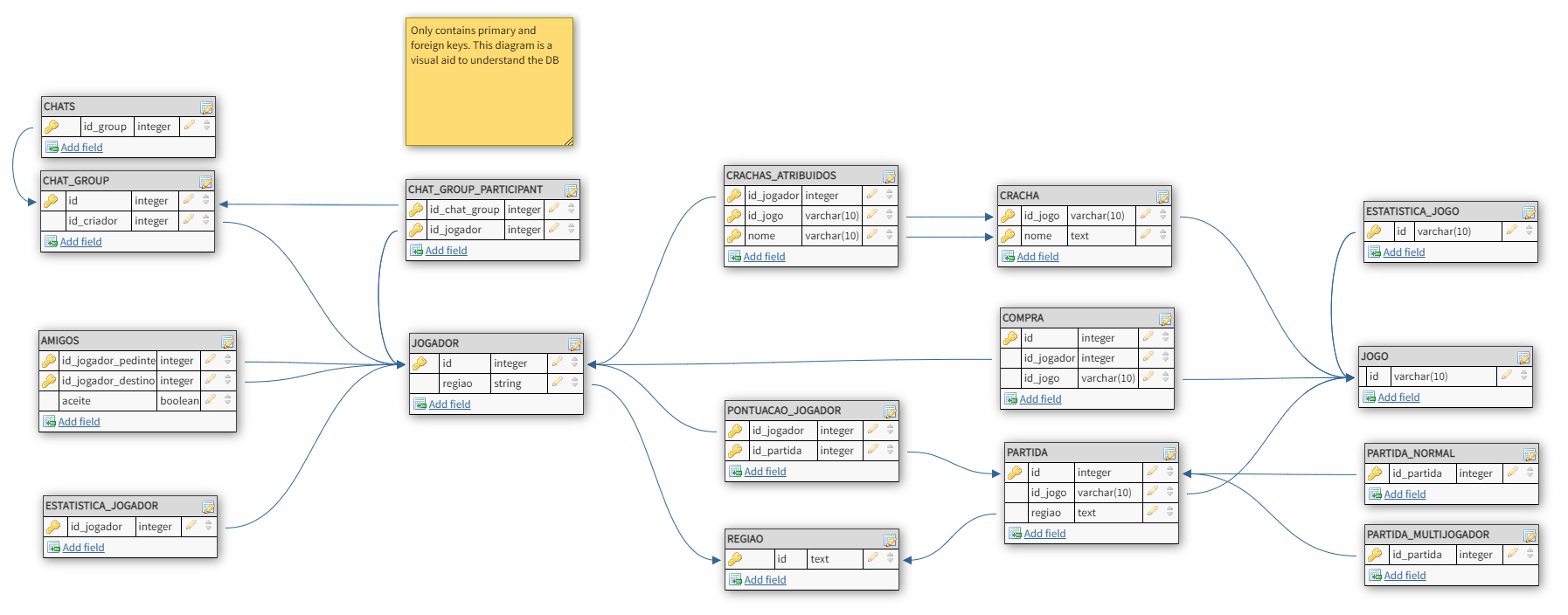


Imagem 2: Diagrama das chaves estrangeiras

# Explicação da nossa resolução dos exercícios

Nota: No PostgreSQL, por default, não há leitura de dados uncommited. Por default lê-se dados commited.

d) Nesta alínea criamos o procedimento armazenado “criarJogador” que insere um jogador na tabela JOGADOR dado o username, email e regiao. Visto que usámos uma chave primária que se auto-gera, criámos um loop que evita a duplicação de chaves, ao contrário de fazer *INSERT* e tento como o parâmetro da chave igual a *DEFAULT*.  
Usámos o nível de isolamento:

Também criámos o procedimento armazenado mudarEstadoJogador(usernamee *TEXT*, novoEstado estado\_enum).

Usámos o nível de isolamento:

e) Aqui criámos a função getTotalPointsOfPlayer(id INT) *RETURNS INT* que faz uso da operação *COALESCE* e *SUM* para fazer o somatório dos pontos.

Usámos o nível de isolamento:  
  
f) Criámos a função getNumGames\_aPlayerPlayed(idJogador INT) *RETURNS INT*. A nossa solução consistiu em fazer um *INNER JOIN* entre a tabela PARTIDA e PONTUACAO\_JOGADOR sobre a chave da partida e também filtrar pelos jogadores cujo “id” seja igual ao do parâmetro. No *SELECT* é feito o *COUNT* com um *DISTINCT* para contar as partidas encontradas com esta condição.

g) Ao se fazer um *INNER JOIN* entre as tabelas PONTUACAO\_JOGADOR e PARTIDA, associando os ids de partida de ambas as tabelas, a tabela resultante irá possuir tuplos que contêm o id de JOGADOR, o id de JOGO e os pontos de cada partida, mas como se filtra os resultados para sobrar apenas tuplos que referem o id de JOGO escolhido, apenas são apresentados os resultados de pontos por partida desse JOGO. Por fim ao se fazer *GROUP BY* de id de JOGADOR, as tabelas com o mesmo id são *‘merged’*, e como entre elas a única diferença são os pontos, ao se aplicar o *SUM* na coluna pontos, obter-se-á a soma desses pontos para cada jogo.

Esse processo é todo associado ao *RETURN QUERY* que retorna essa *query* pela função.

h) Primeiro começa-se por procurar o CRACHA fornecido e guardar os pontos associados a ele para uso futuro, aproveitando para verificar se esse mesmo CRACHA existe. Depois utiliza-se a função do exercício **g)**, ‘PontosJogoPorjogador’ e filtra-se a partir do id de JOGADORfornecido, obtendo o total de pontos do JOGADOR escolhido no JOGO proposto, e guarda-se esse resultado.  
Por fim, comparam-se os pontos do CRACHA com os pontos do JOGADOR, e caso o JOGADOR tenha pontos suficientes insere-se na tabela CRACHAS\_ATRIBUIDOS o tuplo que associa um CRACHA de um JOGO a um JOGADOR.

O nível de isolamento escolhido foi o default (read committed) pois não houve necessidade de nenhuma outra proteção neste caso.

i) Antes de se inserir um tuplo que representa o CHAT\_GROUP realiza-se uma verificação com o intuito de saber se o id de JOGADOR fornecido é válido.  
Depois da inserção o *trigger* chatGroupCreatorIsParticipantTrigger é acionado e corre a função associada para cada tuplo com o intuito de inserir na tabela CHAT\_GROUP\_PARTICIPANT o proprio JOGADOR que criou o CHAT\_GROUP.  
Foi decidido que a tabela CHAT\_GROUP iria ter uma coluna ‘id\_criador’ que representa o JOGADOR que criou o CHAT\_GROUP, e que a tabela CHAT\_GROUP PARTICIPANT que consiste numa tabela com tuplos associados a CHAT\_GROUP e JOGADOR para identificar cada JOGADOR que participa no CHAT\_GROUP iria também ter o JOGADOR que é o criador pois essa decisão é mais consistente e beneficial em futuras *queries*.

O nível de isolamento escolhido foi o *default*  (*read committed*) pois não houve necessidade de nenhuma outra proteção neste caso.

k) Apenas será criado o tuplo na tabela CHATS que representa uma mensagem associada a um CHAT\_GROUP, ao JOGADOR que a enviou e as suas restantes características. Isto, logo depois de se fazer a validação de todos os parâmetros fornecidos.

O nível de isolamento escolhido foi o *default*  (*read committed*) pois não houve necessidade de nenhuma outra proteção neste caso.

l) Para realizar a vista os dados vão ser extraídos em 3 partes; Primeiramente a Tabela JOGADOR para apresentar na view as colunas id, estado, email e username. Depois é realizado um *FULL OUTER JOIN* com o resultado de uma query à tabela PONTUAÇAO\_JOGADOR que como no exercício g) retornando o total de pontos. E por fim, a terceira parte consiste no FULL OUTER JOIN numa query à tabela PARTIDAS mas fazendo um COUNT DISTINCT para se saber o número total de partidas associadas a esse jogador.

m) É feito um ciclo, para cada jogador envolvido na partida, é calculado o nome do crachá concursante os pontos que o jogador teve na partida, e estes são inseridos na tabela CRACHA\_ATRIBUIDOS, caso haja duplicação no caso de o jogador ja ter crachá, é feito um raise notice.

n) Cada vez que há uma tentativa de eliminar um tuplo da vista ‘jogadorTotalInfo’ o trigger irá acionar e substituir esse delete pela sua própria lógica, que consiste apenas em substituir a coluna de estado do JOGADOR por ‘Banido’. Consequentemente excluindo o tuplo que se tentou eliminar da vista.

# Conclusão

Neste trabalho conseguimos encontrar uma solução viável para o problema proposto na forma de um modelo de bases de dados relacional. Nos exercícios foi posto em teste a funcionalidade do modelo de dados. E por fim, foram estudados novos conhecimentos relativamente ao DBMS PostgreSQL.

# Webgrafia e recursos usados

* <https://www.postgresql.org/docs/current/>
* <https://www.postgresqltutorial.com/>
* <https://app.diagrams.net/>