



**ISEL**  
INSTITUTO SUPERIOR DE  
ENGENHARIA DE LISBOA

# Nível Aplicacional da Base de Dados Para a Empresa GameOn

José Alves

Alexandre Severino

Diogo Carichas

Orientadores   Walter Vieira

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

Maio de 2023

# Resumo

TODO()

**Palavras-chave:** palavras;chave

# Abstract

TODO()

**Keywords:** key;words;

# Índice

<b>1</b>	<b>Introdução</b>	<b>1</b>
1.1	Objetivos . . . . .	1
<b>2</b>	<b>Modelo de dados</b>	<b>2</b>
2.1	Embeddables . . . . .	2
2.2	Relations . . . . .	2
2.3	Tables . . . . .	3
<b>3</b>	<b>Funcionalidade da Aplicação</b>	<b>4</b>
3.1	Acesso às Funcionalidades da Base de Dados . . . . .	4
3.2	Realização da Funcionalidade 2h . . . . .	5
3.2.1	Sem usar procedimentos e funções pgSql . . . . .	5
3.2.2	Usando procedimentos e funções pgSql . . . . .	5
3.3	Funcionalidade Adicional . . . . .	6
3.3.1	Implementação Com <i>Optimistic Locking</i> . . . . .	6
3.3.2	Implementação Com <i>Pessimistic Locking</i> . . . . .	6
3.3.3	Testes da Funcionalidade . . . . .	6
<b>4</b>	<b>Conclusão</b>	<b>7</b>
4.1	Aspetos a melhorar . . . . .	7
<b>A</b>	<b>Código das Classes do <i>Package</i> model.embeddables</b>	<b>9</b>
<b>B</b>	<b>Código das Classes do <i>Package</i> model.relations</b>	<b>15</b>
<b>C</b>	<b>Código das Classes do <i>Package</i> model.tables</b>	<b>19</b>

# Lista de Figuras

# Lista de Tabelas

# Capítulo 1

## Introdução

Uma base de dados por ela mesma tem o seu mérito, quando bem estruturada e implementada é uma mais valia para qualquer empresa que procure organizar a sua informação. No entanto esta não deve permanecer sozinha, se for acompanhada por uma aplicação capaz de comunicar numa outra linguagem de programação, a base de dados pode ser utilizada por aplicações e programas de computador com mais flexibilidade no seu uso.

Por forma a cumprir este objetivo foi desenvolvida uma aplicação *Java* que faz uso da biblioteca JPA, através da qual irá criar um modelo de dados e comunicar com a base de dados remota.

### 1.1 Objetivos

Em primeiro lugar é necessário implementar o modelo de dados ao nível aplicacional. Para este efeito a biblioteca JPA permite usar anotações do *Java* para determinar o tipo de cada campo de uma classe e como este deve interagir com os tipos de *PostgreSQL* [1].

Esta aplicação deverá ser capaz de permitir a um programador que a utilize de aceder a todas as funcionalidades presentes na fase 1, logo deverá implementar funções que façam uso das funções, procedimentos armazenados, vistas e gatilhos presentes no modelo de base de dados.

Para além de apresentar uma interface para acesso à base de dados esta aplicação deve também permitir ao programador escolher certas funcionalidades entre *optimistic locking* e *pessimistic locking*.

## Capítulo 2

# Modelo de dados

Antes de passar à implementação das funcionalidades da base de dados é necessário primeiro organizar as tabelas e relações em classes de *Java* correspondentes. Este modelo está dividido em três *packages* diferentes, todos dentro do *package model*. Dentro deste encontra-se **embeddables**, o qual contém classes de valores embebidos, ou seja, identificadores complexos que podem englobar mais que um atributo. Encontram-se também o *package relations*, que contém as tabelas que representam relações na base de dados e o *package tables*, o qual contém as tabelas principais.

Cada uma destas classes contém as anotações que sejam necessárias. Realça-se o facto de que as classes dentro do *package relations* têm a anotação **@Entity**. Esta característica deve-se ao facto de todas as tabelas da base de dados, incluindo as que representam relações, serem tratadas como entidades.

### 2.1 Embeddables

As classes dentro deste *package* servem como auxílio à implementação das classes mais complexas, pois estas permitem ter identificadores complexos que sejam representados por mais que um atributo. Todas estas classes incluem o uso da anotação **@Embeddable**.

A implementação destas classes encontra-se no anexo A.

### 2.2 Relations

Este *package* contém as classes que identificam relações do modelo que foram implementadas como tabelas na base de dados. Estas estão colocadas separadamente pois não representam uma entidade no modelo ER, mas sim uma relação. Todas contém as anotações **@Entity** e **@Table**, com o valor do campo **name** igual à tabela correspondente da base de dados PostgreSQL.

Como todas estas são representadas através de identificadores de outras entidades, todas têm um campo com a anotação **@EmbeddedId**, sendo este campo do tipo correspondente do *package embeddables*. A implementação destas classes encontra-se no anexo B.



## 2.3 Tables

Neste *package* encontram-se as tabelas da base de dados que representam entidades no modelo ER. Estas fazem uso das classes presentes em relations para ser mais fácil de aceder a certas informações, como, por exemplo, a lista de jogadores que compraram um jogo ou a lista de jogos que um jogador comprou. Estes estão presentes como campos das classes **Jogo** e **Jogador**, respetivamente.

Existem ainda funções que acompanham estas implementações pois todos estes campos estão declarados como **private**, sendo apenas possível aceder fazendo uso de um *getter* e alterando o seu valor com um *setter*.

A implementação destas classes encontra-se no anexo C.

## Capítulo 3

# Funcionalidade da Aplicação

Uma vez o modelo feito é agora possível proceder à implementação das funcionalidades desejadas do trabalho. Estas requerem primeiro acesso às funções e procedimentos armazenados criados na primeira parte, os quais vêm em formato das alíneas do primeiro enunciado 2d até 2l.

Com o acesso a estas funcionalidades obtido, procede-se à implementação da alínea 2h como na fase 1 mas sem usar qualquer procedimento armazenado ou função `pgSql`, ou seja, limitando as possibilidades para usar apenas interações entre JPA e PostgreSQL. Após esta implementação é realizada outra vez mas sem a limitação, o qual permite facilmente comparar as duas implementações e concluir qual será melhor.

Esta aplicação também permite aumentar em 20% o número de pontos associados a um crachá, o qual foi implementado com *optimistic locking* e *pessimistic locking*. A versão que utiliza *optimistic locking* vem acompanhada de testes que verificam que uma mensagem de erro é levantada quando existe uma alteração concorrente que inviabilize a operação.

### 3.1 Acesso às Funcionalidades da Base de Dados

O primeiro requisito é disponibilizar as funcionalidades presentes na base de dados, começando com o exercício 2d da primeira fase até ao exercício 2l. Estas são apresentadas na forma de funções da aplicação.

**Exercício 2d** Este exercício consiste em duas funções distintas, tal como na base de dados, pois estas efetuam operações diferentes. As duas funções são `createPlayer` e `setPlayerState`. Nas listagens 1 e 2 encontram-se as funções `createPlayer` e `setPlayerState`, respetivamente. Realça-se nestas implementações que a forma de chamar um função em PostgreSQL é como realizar uma *query* para a função, tal como a *string* criada na primeira linha de cada função.

---

```
1 public String createPlayer(String username, String email, String regiao) {  
2     String query = "SELECT createPlayer(?1, ?2, ?3)";
```

```

3     Query functionQuery = em.createNativeQuery(query)
4         .setParameter(1, username)
5         .setParameter(2, email)
6         .setParameter(3, regioao);
7     return (String) functionQuery.getSingleResult();
8 }

```

---

Listing 1: Código da função createPlayer

```

1 public String setPlayerState(int idJogador, String newState) {
2     String query = "SELECT setPlayerState(?1, ?2)";
3     Query functionQuery = em.createNativeQuery(query)
4         .setParameter(1, idJogador)
5         .setParameter(2, newState);
6     return (String) functionQuery.getSingleResult();
7 }

```

---

Listing 2: Código da função setPlayerState

**Exercício 2e** Este exercício apenas requer a implementação de uma função. Esta também tem apenas como objetivo chamar uma função da base de dados e não requer nível de isolamento acima do que está por definição em PostgreSQL. Tal como nas funções em 2d, esta apenas realiza uma *query* para a função na base de dados. A implementação encontra-se na listagem 3.

```

1 public Long totalPontosJogador(int idJogador) {
2     String query = "SELECT totalPontos from totalPontosJogador(?1)";
3     Query functionQuery = em.createNativeQuery(query);
4     functionQuery.setParameter(1, idJogador);
5     return (Long) functionQuery.getSingleResult();
6 }

```

---

Listing 3: Código da função totalPontosJogador

## 3.2 Realização da Funcionalidade 2h

TODO(Descrever a funcionalidade)

### 3.2.1 Sem usar procedimentos e funções pgSql

TODO(Descrever a implementação sem pgSql)

### 3.2.2 Usando procedimentos e funções pgSql

TODO(Descrever a implementação com pgSql)

### **3.3 Funcionalidade Adicional**

TODO(Aumentar a 20% cena)

#### **3.3.1 Implementação Com *Optimistic Locking***

TODO(Descrever a implementação que usa optimistic locking)

#### **3.3.2 Implementação Com *Pessimistic Locking***

TODO(Descrever a implementação que usa pessimistic locking)

#### **3.3.3 Testes da Funcionalidade**

TODO(Descrever os testes e como se testou concorrência)

## Capítulo 4

# Conclusão

TODO()

### 4.1 Aspetos a melhorar

TODO()

# Referências

- [1] Thomas G. Lockhart Peter Eisentraut, Julien Rouhaud. Postgresql 15.2 documentation. [Online; Accessed 01-May-2023].

## Anexo A

# Código das Classes do *Package* `model.embeddables`

Neste anexo encontra-se a implementação das várias classes do *package* `Embeddables`

---

```
1 package model.embeddables;
2
3 import jakarta.persistence.Embeddable;
4 import java.io.Serializable;
5
6 // Composite key for table Compra
7 @Embeddable
8 public class CompraId implements Serializable {
9     private int jogador;
10    private String jogo;
11
12    public CompraId(){}
13    public void setJogoId(String jogo) {
14        this.jogo = jogo;
15    }
16
17    public String getJogoId() {
18        return jogo;
19    }
20
21    public void setJogadorId(int jogador) {
22        this.jogador = jogador;
23    }
24
25    public int getJogadorId() {
26        return jogador;
27    }
28 }
```

---

Listing 4: Código da classe `CompraId`

---

```
1 package model.embeddables;
2
3 import jakarta.persistence.Embeddable;
4
5 import java.io.Serializable;
6
7 // Composite jey for table cracha
8 @Embeddable
9 public class CrachaId implements Serializable {
10     private String nome;
11     private String jogo;
12
13     public CrachaId(){}
14     public String getNome() {
15         return nome;
16     }
17
18     public void setNome(String nome) {
19         this.nome = nome;
20     }
21
22     public String getJogo() {
23         return jogo;
24     }
25
26     public void setJogo(String jogo) {
27         this.jogo = jogo;
28     }
29
30 }
```

---

Listing 5: Código da classe CrachaId

---

```
1 package model.embeddables;
2
3 import jakarta.persistence.Embeddable;
4
5 import java.io.Serializable;
6
7 @Embeddable
8 public class CrachasAdquiridosId implements Serializable {
9     private int jogo;
10    private String jogador;
11    private String cracha;
12
13    public CrachasAdquiridosId(){}
14    public void setJogo(int jogo) {
15        this.jogo = jogo;
16    }
17 }
```



```

17
18     public int getJogo() {
19         return jogo;
20     }
21
22     public void setJogador(String jogador) {
23         this.jogador = jogador;
24     }
25
26     public String getJogador() {
27         return jogador;
28     }
29
30     public void setCracha(String cracha) {
31         this.cracha = cracha;
32     }
33
34     public String getCracha() {
35         return cracha;
36     }
37 }

```

---

Listing 6: Código da classe CrachasAdquiridosId

---

```

1  package model.embeddables;
2
3  import jakarta.persistence.Embeddable;
4  import jakarta.persistence.GeneratedValue;
5  import jakarta.persistence.GenerationType;
6
7  import java.io.Serializable;
8
9  //Composite key for table mensagem
10 @Embeddable
11 public class MensagemId implements Serializable {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private int id;
14     private int conversa;
15
16     public MensagemId(){ }
17     public void setid(int id) {
18         this.id = id;
19     }
20
21     public int getid() {
22         return id;
23     }
24
25     public int getConversa() {
26         return conversa;

```

```

27     }
28     public void setConversa(int conversa) {
29         this.conversa = conversa;
30     }
31
32 }

```

---

Listing 7: Código da classe MensagemId

---

```

1  package model.embeddables;
2
3  import jakarta.persistence.Embeddable;
4
5  import java.io.Serializable;
6
7  //Composite key for jogador_conversa relation participa
8  @Embeddable
9  public class ParticipaId implements Serializable {
10     private int jogador;
11     private int conversa;
12
13     public ParticipaId(){}
14
15     public void setJogadorId(int jogador) {
16         this.jogador = jogador;
17     }
18
19     public int getJogadorId() {
20         return jogador;
21     }
22
23     public void setConversaId(int conversa) {
24         this.conversa = conversa;
25     }
26
27     public int getConversaId() {
28         return conversa;
29     }
30 }

```

---

Listing 8: Código da classe ParticipaId

---

```

1  package model.embeddables;
2
3  import jakarta.persistence.Embeddable;
4  import jakarta.persistence.GeneratedValue;
5  import jakarta.persistence.GenerationType;
6
7  import java.io.Serializable;

```

```

8
9 // Composite Primary key for Partida
10 @Embeddable
11 public class PartidaId implements Serializable {
12     @GeneratedValue(strategy = GenerationType.IDENTITY)
13     private int partida;
14     private String jogo;
15
16     public PartidaId(){}
17     public void setPartidaId(int partida) {
18         this.partida = partida;
19     }
20
21     public int getPartidaId() {
22         return partida;
23     }
24
25     public void setJogoId(String jogo) {
26         this.jogo = jogo;
27     }
28
29     public String getJogoId() {
30         return jogo;
31     }
32 }

```

---

Listing 9: Código da classe PartidaId

---

```

1 package model.embeddables;
2
3 import jakarta.persistence.Embeddable;
4
5 import java.io.Serializable;
6
7 // Composite Primary key for Partida_Normal and Partida_MultiJogador
8 @Embeddable
9 public class PartidaNMId implements Serializable {
10     int partida;
11     String jogo;
12     int jogador;
13
14     public PartidaNMId(){}
15
16     public void setPartida(int partidaId) {
17         this.partida = partidaId;
18     }
19
20     public int getPartida() {
21         return partida;
22     }

```

```
23
24     public String getJogo() {
25         return jogo;
26     }
27
28     public int getJogador() {
29         return jogador;
30     }
31
32     public void setJogador(int jogadorId) {
33         this.jogador = jogadorId;
34     }
35
36     public void setJogo(String jogoId) {
37         this.jogo = jogoId;
38     }
39 }
```

---

Listing 10: Código da classe PartidaNMId

## Anexo B

# Código das Classes do *Package* model.relations

Estas listagens contêm código correspondente às classes presentes no *package relations*

---

```
1 package model.relations;
2
3 import jakarta.persistence.*;
4 import model.tables.Jogador;
5 import model.tables.Jogo;
6 import model.embeddables.CompraId;
7
8 import java.io.Serial;
9 import java.io.Serializable;
10 import java.util.Date;
11
12 @Entity
13 @Table(name="compra")
14 public class Compra implements Serializable {
15
16     @Serial
17     private static final long serialVersionUID = 1L;
18
19     public Compra() { }
20
21     @EmbeddedId
22     private CompraId id;
23     private Date data;
24     private Double preco;
25
26
27     // N:N relation Compra between jogo and jogador
28     @ManyToOne
29     @MapsId("jogo")
30     @JoinColumn(name = "jogo")
31     private Jogo jogo;
32
33     @ManyToOne
```

```

34     @MapsId("jogador")
35     @JoinColumn(name = "jogador")
36     private Jogador jogador;
37
38     //getter and setters
39     public Jogador getJogador() { return jogador; }
40
41     public void setJogador(Jogador jogador) { this.jogador = jogador; }
42
43     public Jogo getJogo() { return jogo; }
44
45     public void setJogo(Jogo jogo) { this.jogo = jogo; }
46
47     public CompraId getId() { return this.id; }
48
49     public Date getData() { return data; }
50
51     public void setData(Date data) { this.data = data; }
52
53     public void setPreco(Double preco) { this.preco = preco; }
54     public Double getPreco() { return preco; }
55
56 }

```

---

Listing 11: Código da classe Compra

---

```

1  package model.relations;
2
3  import jakarta.persistence.*;
4  import model.tables.Cracha;
5  import model.tables.Jogador;
6  import model.embeddables.CrchasAdquiridosId;
7
8  import java.io.Serializable;
9
10 @Entity
11 @Table(name="crachá_jogador")
12 public class CrchasAdquiridos implements Serializable {
13
14     @EmbeddedId
15     private CrchasAdquiridosId id;
16
17
18     // N:N realtion CrchasAdquiridos
19     @ManyToOne
20     @MapsId("jogador")
21     @JoinColumn(name="jogador", referencedColumnName = "jogador")
22     private Jogador jogador;
23
24

```

```

25     @ManyToOne
26     @JoinColumns({
27         @JoinColumn(name = "cracha_nome", referencedColumnName = "nome"),
28         @JoinColumn(name = "cracha_jogador", referencedColumnName = "jogador")
29     })
30     private Cracha cracha;
31
32
33     public CrachasAdquiridosId getId() {
34         return id;
35     }
36
37     public void setId(CrachasAdquiridosId id) {
38         this.id = id;
39     }
40
41     public Cracha getCracha() {
42         return cracha;
43     }
44
45     public void setCracha(Cracha cracha) {
46         this.cracha = cracha;
47     }
48
49     public Jogador getJogador() {
50         return jogador;
51     }
52
53     public void setJogador(Jogador jogador) {
54         this.jogador = jogador;
55     }
56 }

```

---

Listing 12: Código da classe CrachasAdquiridos

---

```

1  package model.relations;
2
3  import jakarta.persistence.*;
4  import model.tables.Conversa;
5  import model.tables.Jogador;
6  import model.embeddables.ParticipaId;
7
8  import java.io.Serial;
9  import java.io.Serializable;
10
11  @Entity
12  @Table(name = "jogador_conversa")
13  public class Participa implements Serializable {
14
15      @Serial

```

```

16     private static final long serialVersionUID = 1L;
17
18     public Participa(){ }
19     @EmbeddedId
20     private ParticipaId id;
21
22     @ManyToOne
23     @MapsId("jogador")
24     @JoinColumn(name="jogador")
25     private Jogador jogador;
26
27     @ManyToOne
28     @MapsId("conversa")
29     @JoinColumn(name="conversa")
30     private Conversa conversa;
31
32
33     //getters and setters
34     public ParticipaId getId() {return id;}
35
36     public void setId(ParticipaId id) { this.id = id; }
37
38     public Jogador getJogador() { return jogador; }
39
40     public void setJogador(Jogador jogador) { this.jogador = jogador; }
41
42     public Conversa getConversa() { return conversa; }
43
44     public void setConversa(Conversa conversa) { this.conversa = conversa; }
45 }

```

---

Listing 13: Código da classe Participa



## Anexo C

# Código das Classes do *Package* model.tables

Neste anexo encontra-se o código do ficheiro deleteSchema.sql

---

```
1 package model.tables;
2
3 import jakarta.persistence.*;
4 import model.relations.Compra;
5
6 import java.io.Serial;
7 import java.io.Serializable;
8 import java.util.ArrayList;
9 import java.util.List;
10
11 @Entity
12 @Table(name="jogo")
13 public class Jogo implements Serializable {
14
15     @Serial
16     private static final long serialVersionUID = 1L;
17
18     @Id
19     @GeneratedValue(strategy = GenerationType.UUID)
20     private String id;
21     private String nome;
22     private String url;
23
24     @OneToMany(mappedBy = "jogo")
25     private List<Compra> compras = new ArrayList<>();
26
27     public void addCompra(Compra compra){ this.compras.add(compra); }
28
29     public List<Compra> getCompras() { return compras; }
30
31
32     // 1:N pertence
33     @OneToMany(mappedBy="jogo",cascade=CascadeType.PERSIST, orphanRemoval=true)
```

```

34     private List<Cracha> crachas = new ArrayList<>();
35
36     public List<Cracha> getCrachas() {
37         return crachas;
38     }
39
40     public void addCracha(Cracha cracha) {
41         this.crachas.add(cracha);
42     }
43     @OneToMany(mappedBy = "jogo", cascade = CascadeType.PERSIST, orphanRemoval=true)
44     private List<Partida> partidas = new ArrayList<>();
45
46     public List<Partida> getPartidas() {
47         return partidas;
48     }
49
50     public void addPartida(Partida partida){
51         partidas.add(partida);
52     }
53
54     public String getId(){ return this.id; }
55     public String getNome(){ return this.nome; }
56     public void setNome(String nome) { this.nome = nome; }
57
58     public String getUrl(){ return this.url; }
59     public void setUrl(String url) { this.url = url; }
60
61 }

```

---

Listing 14: Código da classe Jogo