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

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 casa 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 econtra-se `embeddables`, o qual contém classes de valores embebidos, ou seja, identificadores complexos que podem englobar mais que um atríbuto. 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 á 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 contêm 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.

Business Logic O *package* `businessLogic`, cujo código das suas classes está presente no anexo D, contém todas as funções representadas neste capítulo. Dentro deste *package* encontra-se também `BLServiceUtils`, presente no anexo E, o qual contém funções úteis para a classe `BLService`.

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.

3.1.1 Funções

Nesta secção encontram-se as funções chamadas tal como na base de dados PostgreSQL.

Exercício 2d Este exercício consiste em duas funções distintas, tal como na base de dados, pois estas eftuam 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.

```
public String createPlayer(String username, String email, String regiao) {
    String query = "SELECT createPlayer(?1, ?2, ?3)";
    Query functionQuery = em.createNativeQuery(query)
            .setParameter(1, username)
            .setParameter(2, email)
            .setParameter(3, regiao);
    return (String) functionQuery.getSingleResult();
}
```

Listing 1: Código da função createPlayer

```
public String setPlayerState(String playerName, String newState) {
    int idJogador = modelManager.getPlayerByEmail(playerName, em).getId();
    String query = "SELECT setPlayerState(?1, ?2)";
    Query functionQuery = em.createNativeQuery(query)
            .setParameter(1, idJogador)
            .setParameter(2, newState);
    return (String) functionQuery.getSingleResult();
}
```

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 isolamente 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.

```
public Long totalPontosJogador(String email) {
    int idJogador = modelManager.getPlayerByEmail(email, em).getId();
    String query = "SELECT totalPontos from totalPontosJogador(?1)";
    Query functionQuery = em.createNativeQuery(query);
    functionQuery.setParameter(1, idJogador);
    return (Long) functionQuery.getSingleResult();
}
```

Listing 3: Código da função totalPontosJogador

Exercício 2f Tal como as outras funções, esta é chamada através de uma *query*. Este exercício pretende permitir saber quantos pontos um jogador obteve no total das suas partidas em vários jogos.

```
public Long totalJogosJogador(String email) {
    int idJogador = modelManager.getPlayerByEmail(email, em).getId();
    String query = "SELECT totalJogos from totalJogosJogador(?1)";
    Query functionQuery = em.createNativeQuery(query);
    functionQuery.setParameter(1, idJogador);
    return (Long) functionQuery.getSingleResult();
}
```

Listing 4: Código da função totalJogosJogador

Exercício 2g Esta função é mais complexa que as outras devido ao facto de retornar uma tabela e não apenas um valor ou o resultado de uma operação e não há entidade para este resultado. Para resolver o problema é possível seguir várias opções: usar a função `getResultList` e colocar numa lista de arrays de objetos da classe `Object`, fazer duas queries, uma para os jogadores e outra para as pontuações e depois agrupá-las num objeto da classe `Map`, ou criar uma entidade apenas para os resultados desta função. A solução escolhida foi a primeira, tal como implementado na listagem 5

```
public Map<Integer, BigDecimal> PontosJogosPorJogador(String gameName) {
    String idJogo = modelManager.getGameByName(gameName, em).getId();
    String queryString = "SELECT jogadores, pontuacaoTotal from PontosJogosPorJogador(?1)";
    Query query = em.createNativeQuery(queryString);
    query.setParameter(1, idJogo);
    Map<Integer, BigDecimal> map = new java.util.HashMap<>(Map.of());
    List<Object[]> list  = query.getResultList();
    for (Object[] obj : list) {
        Integer idJogador = (Integer) obj[0];
        BigDecimal points = (BigDecimal) obj[1];
        map.put(idJogador,  points);
    }
    return map;
}
```

Listing 5: Código da função PontosJogosPorJogador

3.1.2 Procedimentos armazenados

Esta secção cobre a chamada aos procedimentos armazenados da base de dados. Estes requerem um tratamento diferente pois podem ter níveis de transação específicos.

Exercício 2h Este procedimento da base de dados permite associar um crachá a um jogador dados o identificador do jogador, o identificador do jogo e o nome do crachá. Esta transação necessita de um nível de isolamento `TRANSACTION_REPEATABLE_READ`, tal como se identifica na linha 6 da listagem 6. No final da transação é necessário fazer `transaction.commit()`, tal como na linha 12. Caso falhe, irá fazer `transaction.rollback()`, presente na linha 15.

```
1  public void associarCracha(int idJogador, String gameName, String nomeCracha) {
2      EntityTransaction transaction = transactionManager.startTransaction();
3      Connection cn = em.unwrap(Connection.class);
4      String idJogo = modelManager.getGameByName(gameName, em).getNome();
5      try {
6          transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_REPEATABLE_READ, transaction);
7          try (CallableStatement storedProcedure = cn.prepareCall("call associarCracha(?,?, ?)")) {
8              storedProcedure.setInt(1, idJogador);
9              storedProcedure.setString(2, idJogo);
10             storedProcedure.setString(3, nomeCracha);
11             storedProcedure.executeUpdate();
12             transaction.commit();
13         }
14     } catch(Exception e){
15         if(transaction.isActive()) transaction.rollback();
16     }
17 }
```

Listing 6: Código da função associarCracha

Exercício 2i Assim como no exercício 2h, este procedimento armazenado necessita de um nível de isolamento `TRANSACTION_REPEATABLE_READ`. Este procedimento cria uma nova conversa e coloca o jogador que a iniciou dentro dela. Este procedimento coloca o identificador da conversa no seu terceiro argumento, o qual é criado na linha 4 da listagem 7. Este valor é depois retornado. Caso a transação falhe, retorna explicitamente `null`, pois não se garante que `idConversa` permaneça com o valor `null` em caso de falha e consequente `rollback`.

```
1  public Integer iniciarConversa(int idJogador, String nomeConversa) {
2      EntityTransaction transaction = transactionManager.startTransaction();
3      Connection cn = em.unwrap(Connection.class);
4      Integer idConversa = null;
5      try {
6          transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_REPEATABLE_READ, transaction);
7          try (CallableStatement storedProcedure = cn.prepareCall("call iniciarConversa(?,?, ?)")) {
8              storedProcedure.setInt(1, idJogador);
9              storedProcedure.setString(2, nomeConversa);
10             storedProcedure.registerOutParameter(3, Types.INTEGER);
11             storedProcedure.executeUpdate();
12             idConversa = storedProcedure.getInt(3);
13             transaction.commit();
```

```
        }
    } catch(Exception e){
        if(transaction.isActive()) transaction.rollback();
        return null;
    }
    return idConversa;
}
```

Listing 7: Código da função iniciarConversa

Exercício 2j Este procedimento é dos mais simples, sendo que apenas necessita de um nível de isolamento `TRANSACTION_READ_COMMITTED`. Com o identificador de um jogador e o identificador de uma conversa, colocará esse par na tabela `jogador_conversa`, a qual indica que jogadores pertencem a cada conversa.

```
public void juntarConversa(int idJogador, int idConversa) {
    EntityTransaction transaction = transactionManager.startTransaction();
    Connection cn = em.unwrap(Connection.class);
    try {
        try (CallableStatement storedProcedure = cn.prepareCall("call juntarConversa(?,?)")) {
            storedProcedure.setInt(1, idJogador);
            storedProcedure.setInt(2, idConversa);
            transaction.commit();
        }
    } catch(Exception e){
        if(transaction.isActive()) transaction.rollback();
    }
}
```

Listing 8: Código da função juntarConversa

Exercício 2k Neste exercício o procedimento é, tal como em 2j, mais simples e requer apenas um nível de isolamento `TRANSACTION_READ_COMMITTED`. Usando o identificador de um jogador, o qual envia a mensagem, o identificador da conversa para onde a mensagem é enviada e o conteúdo da mensagem em questão. Esta função apenas cria uma transação e realiza `commit` em caso de sucesso e `rollback` em caso de falha.

```
public void enviarMensagem(int idJogador, int idConversa, String content) {
    EntityTransaction transaction = transactionManager.startTransaction();
    Connection cn = em.unwrap(Connection.class);
    try {
        transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_READ_COMMITTED, transaction);
        try (CallableStatement storedProcedure = cn.prepareCall("call enviarMensagem(?,?, ?)")) {
            cn.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
            storedProcedure.setInt(1, idJogador);
            storedProcedure.setInt(2, idConversa);
```

```
            storedProcedure.setString(3, content);
            storedProcedure.executeUpdate();
            transaction.commit();
        }
    } catch(Exception e){
        if(transaction.isActive()) transaction.rollback();
    }
}
```

Listing 9: Código da função enviarMensagem

3.1.3 Vistas

Esta última subsecção representa a implementação da vista presente na base de dados.

Exercício 2l Esta vista permite obter a informação de todos os jogadores com exceção dos que têm o estado `'banido'`. Para obter os valores desta tabela optou-se por criar uma entidade que representa estes resultados, pois uma vista funciona tal como uma tabela e, devido a esta característica, é efetuada uma *query* normal. Após obter os resultados esta função coloca-os no *standard output*.

```
public void jogadorTotalInfo(){
    String query = "Select * from jogadorTotalInfo";
    Query getAllInfo = em.createNativeQuery(query, JogadorTotalInfo.class);
    List<JogadorTotalInfo> allInfo =  getAllInfo.getResultList();
    for(JogadorTotalInfo jogador: allInfo){
        System.out.println(
                jogador.getEstado() + " " +
                jogador.getEmail() + " " +
                jogador.getUsername() + " " +
                jogador.getJogosParticipados() + " " +
                jogador.getPartidasParticipadas() + " " +
                jogador.getPontuacaoTotal() + " "
        );
    }
}
```

Listing 10: Código da função jogadorTotalInfo

3.2 Realização da Funcionalidade 2h

Originalmente este procedimento armazenado é chamado diretamente. No entanto implementou-se a função `associarCrachaModel`, a qual, com os mesmos parâmetros que a função `associarCracha`, efetua o mesmo procedimento mas fazendo uso de ferramentas da aplicação ao invés de usar funções pgSql.

Esta função começa por criar uma `TypedQuery`, na qual pretende determinar se o cracha pretendido existe, presnete na listagem 11.

```
String idJogo = modelManager.getGameByName(gameName, em).getId();
TypedQuery<String> crachaExistsQuery =
        em.createQuery(
                "select c.id.jogo from Cracha c where c.id.jogo = :idJogo and c.id.nome = :nomeCracha",
                String.class
        );

crachaExistsQuery.setParameter("idJogo", idJogo);
crachaExistsQuery.setParameter("nomeCracha", nomeCracha);
String foundIdJogo = crachaExistsQuery.getSingleResult();
```

Listing 11: Excerto da função associarCrachaModel para determinar se o cracha existe

O excerto da listagem 11 obtém um id de um jogo, o qual necessita de ser igual ao dado como parâmetro. Uma vez sabendo esta informação, é agora possível efetuar a *query* `getLimitPoints`, a qual permite saber qual o limite de pontos necessários para que um jogador obtenha esse crachá. O excerto da listagem 12 contém o código necessário para obter esta informação.

```
TypedQuery<Integer> getLimitPoints = em.createQuery(
        "select c.limite from Cracha c where c.id.nome = :nomeCracha",
        Integer.class
);
getLimitPoints.setParameter("nomeCracha", nomeCracha);
Integer limit = getLimitPoints.getSingleResult();
```

Listing 12: Excerto da função associarCrachaModel para determinar o limite de pontos do crachá

Após saber o limite do crachá, é preciso garantir que o jogador cumpre este limite, começando por verificar se este sequer comprou o jogo em questão. A classe `ModelManager` contém o método `ownsgGame`, o qual recebe como parâmetros o identificador de um jogador e o identificador de um jogo e retorna um valor booleano.

```
BigDecimal totalPoints = modelManager.getPlayerPoints(idJogo, idJogador);
```

Listing 13: Excerto da função associarCrachaModel para determinar se o jogador cumpre o limite de pontos

A próxima verificação determina se este jogador cumpre o limite, calculando o total de pontos nesse jogo com o método do `ModelManager`, `getPlayerPoints`, com o jogador como parâmetro.

Finalmente, após todas as verificações necessárias, é criada uma nova entrada na tabela `CrachasAdquiridos`, através do modelo. Este código encontra-se na listagem 14.

```
CrachasAdquiridos crachaAdquirido = new CrachasAdquiridos();
CrachasAdquiridosId crachasAdquiridosId = modelManager.setCrachaAdquiridoId(idJogo, nomeCracha, idJogador);
crachaAdquirido.setId(crachasAdquiridosId);

Cracha cracha = em.find(Cracha.class, modelManager.setCrachaId(idJogo, nomeCracha));
crachaAdquirido.setCracha(cracha);

Jogador jogador = em.find(Jogador.class, idJogador); // Fetch the Jogador entity by ID
crachaAdquirido.setJogador(jogador);
```

Listing 14: Excerto da função associarCrachaModel que cria a nova entrada em CrachasAdquiridos

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`

```java
package model.embeddables;

import jakarta.persistence.Embeddable;
import java.io.Serializable;

// Composite key for table Compra
@Embeddable
public class CompraId implements Serializable {
    private int jogador;
    private String jogo;

    public CompraId(){}
    public void setJogoId(String jogo) {
        this.jogo = jogo;
    }

    public String getJogoId() {
        return jogo;
    }

    public void setJogadorId(int jogador) {
        this.jogador = jogador;
    }

    public int getJogadorId() {
        return jogador;
    }
}
```

Listing 15: Código da classe CompraId

```java
package model.embeddables;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

// Composite jey for table cracha
@Embeddable
public class CrachaId implements Serializable {
    private String nome;
    private String jogo;

    public CrachaId(){}
    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getJogo() {
        return jogo;
    }

    public void setJogo(String jogo) {
        this.jogo = jogo;
    }

}
```

Listing 16: Código da classe CrachaId

```java
package model.embeddables;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

@Embeddable
public class CrachasAdquiridosId implements Serializable {
    private int jogo;
    private String jogador;
    private String cracha;

    public CrachasAdquiridosId(){}
    public void setJogo(int jogo) {
        this.jogo = jogo;
    }
```

```
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 17: 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 18: 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 rewlation 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 19: 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;
```

```java

// Composite Primary key for Partida
@Embeddable
public class PartidaId implements Serializable {
    @GeneratedValue(strategy = GenerationType.IDENTITY)
    private int partida;
    private String jogo;

    public PartidaId(){}
    public void setPartidaId(int partida) {
        this.partida = partida;
    }

    public int getPartidaId() {
        return partida;
    }

    public void setJogoId(String jogo) {
        this.jogo = jogo;
    }

    public String getJogoId() {
        return jogo;
    }
}
```

Listing 20: Código da classe PartidaId

```java
package model.embeddables;

import jakarta.persistence.Embeddable;

import java.io.Serializable;

// Composite Primary key for Partida_Normal and Partida_MultiJogador
@Embeddable
public class PartidaNMId implements Serializable {
    int partida;
    String jogo;
    int jogador;

    public PartidaNMId(){}

    public void setPartida(int partidaId) {
        this.partida = partidaId;
    }

    public int getPartida() {
        return partida;
    }
```

```
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 21: Código da classe PartidaNMId

Anexo B

Código das Classes do *Package* `model.relations`

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

```java
package model.relations;

import jakarta.persistence.*;
import model.tables.Jogador;
import model.tables.Jogo;
import model.embeddables.CompraId;

import java.io.Serial;
import java.io.Serializable;
import java.util.Date;

@Entity
@Table(name="compra")
public class Compra implements Serializable {

    @Serial
    private static final long serialVersionUID = 1L;

    public Compra() { }

    @EmbeddedId
    private CompraId id;
    private Date data;
    private Double preco;


    // N:N relation Compra between jogo and jogador
    @ManyToOne
    @MapsId("jogo")
    @JoinColumn(name = "jogo")
    private Jogo jogo;

    @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 22: 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.CrachasAdquiridosId;
7
8   import java.io.Serializable;
9
10  @Entity
11  @Table(name="crachá_jogador")
12  public class CrachasAdquiridos implements Serializable {
13
14      @EmbeddedId
15      private CrachasAdquiridosId id;
16
17
18      // N:N realtion CrachasAdquiridos
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 23: 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 24: Código da classe Participa

Anexo C

Código das Classes do *Package* `model.tables`

```java
package model.tables;

import jakarta.persistence.*;
import model.relations.Compra;

import java.io.Serial;
import java.io.Serializable;
import java.util.ArrayList;
import java.util.List;

@Entity
@Table(name="jogo")
public class Jogo implements Serializable {

    @Serial
    private static final long serialVersionUID = 1L;

    @Id
    @GeneratedValue(strategy = GenerationType.UUID)
    private String id;
    private String nome;
    private String url;

    @OneToMany(mappedBy = "jogo")
    private List<Compra> compras = new ArrayList<>();

    public void addCompra(Compra compra){ this.compras.add(compra); }

    public List<Compra> getCompras() { return compras; }


    // 1:N pertence
    @OneToMany(mappedBy="jogo",cascade=CascadeType.PERSIST, orphanRemoval=true)
    private List<Cracha> crachas = new ArrayList<>();

```

```
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 25: Código da classe Jogo

```
1   package model.tables;
2
3   import jakarta.persistence.*;
4   import model.relations.Compra;
5   import model.relations.CrachasAdquiridos;
6   import model.relations.Participa;
7
8   import java.io.Serial;
9   import java.io.Serializable;
10  import java.util.ArrayList;
11  import java.util.List;
12
13  @Entity
14  @Table(name="jogador")
15  public class Jogador implements Serializable {
16      @Serial
17      private static final long serialVersionUID = 1L;
18
19      public Jogador(){ }
20
21
```

```
22      @Id
23      @GeneratedValue(strategy = GenerationType.IDENTITY)
24      private int id;
25      private String email;
26      private String username;
27      private String estado;
28      private String regiao;
29      @OneToMany(mappedBy = "jogador")
30      private List<Compra> compras = new ArrayList<>();
31      public void addCompra(Compra compra){ this.compras.add(compra); }
32      public List<Compra> getCompras() { return compras; }
33
34
35      //N:N Crachas
36
37      @OneToMany(mappedBy = "jogador")
38      private List<CrachasAdquiridos> crachasAdquiridosList = new ArrayList<>();
39
40      public List<CrachasAdquiridos> getCrachasList() {
41          return crachasAdquiridosList;
42      }
43
44      public void adquirirCracha(CrachasAdquiridos crachasAdquiridos) {
45          this.crachasAdquiridosList.add(crachasAdquiridos);
46      }
47
48
49      //N:N Participa Relation
50      @OneToMany(mappedBy = "jogador")
51      private List<Participa> participaList = new ArrayList<>();
52      public List<Participa> getParticipaList() { return this.participaList; }
53      public void addParticipacao(Participa participa){this.participaList.add(participa);}
54
55      @OneToMany(mappedBy = "jogador", cascade = CascadeType.PERSIST)
56      private List<Partida_Normal> partidasNormais = new ArrayList<>();
57
58      public List<Partida_Normal> getPartidasNormais() {
59          return partidasNormais;
60      }
61
62      public void addPartida_Normal(Partida_Normal partida_normal){
63          this.partidasNormais.add(partida_normal);
64      }
65
66      @OneToMany(mappedBy = "jogador", cascade = CascadeType.PERSIST)
67      private List<Partida_MultiJogador> partidasMultiJogador = new ArrayList<>();
68
69      public void addPartidaMultiJogador(Partida_MultiJogador partida){
70          this.partidasMultiJogador.add(partida);
71      }
72      public List<Partida_MultiJogador> getPartidasMultiJogador() {
```

```
73          return partidasMultiJogador;
74      }
75
76
77
78      //getter and setters
79      public int getId(){
80          return this.id;
81      }
82
83      public String getEmail(){
84          return this.email;
85      }
86
87      public void setEmail(String email){
88          this.email = email;
89      }
90
91      public String getUsername(){
92          return this.username;
93      }
94
95      public void setUsername(String username){
96          this.username = username;
97      }
98
99      public String getEstado(){
100         return this.estado;
101     }
102
103     public void setEstado(String estado){
104         this.estado = estado;
105     }
106
107     public String getRegiao(){
108         return this.regiao;
109     }
110
111     public void setRegiao(String regiao){
112         this.regiao = regiao;
113     }
114
115 }
```

Listing 26: Código da classe Jogador

```
1  package model.tables;
2
3  import jakarta.persistence.*;
4  import model.embeddables.PartidaId;
```

```
5
6  import java.io.Serial;
7  import java.io.Serializable;
8  import java.util.Date;
9
10 @Entity
11 @Table(name="partida")
12 public class Partida implements Serializable {
13
14     @Serial
15     private static final long serialVersionUID = 1L;
16
17     public Partida() { }
18
19     @EmbeddedId
20     private PartidaId id;
21
22     String estado;
23     String nome_regiao;
24
25     Date dtinicio;
26
27     Date dtfim;
28
29     // 1:N
30     @ManyToOne
31     @MapsId
32     @JoinColumn(name = "id")
33     private Jogo jogo;
34
35     public void setJogo(Jogo jogo) {
36         this.jogo = jogo;
37     }
38
39     public Jogo getJogo() {
40         return jogo;
41     }
42
43     public PartidaId getId() {
44         return id;
45     }
46
47     @OneToOne(mappedBy = "partida")
48     private Partida_MultiJogador partida_multiJogador;
49
50     public Partida_MultiJogador getPartida_multiJogador() {
51         return partida_multiJogador;
52     }
53
54     public void setPartida_multiJogador(Partida_MultiJogador partida_multiJogador) {
55         this.partida_multiJogador = partida_multiJogador;
```

```
56      }
57
58      @OneToOne(mappedBy = "partida")
59      private Partida_Normal partida_normal;
60
61      public Partida_Normal getPartida_normal() {
62          return partida_normal;
63      }
64
65      public void setPartida_normal(Partida_Normal partida_normal) {
66          this.partida_normal = partida_normal;
67      }
68
69      public void setId(PartidaId id) {
70          this.id = id;
71      }
72
73      public String getEstado() {
74          return estado;
75      }
76
77      public void setEstado(String estado) {
78          this.estado = estado;
79      }
80
81      public String getNome_regiao() {
82          return nome_regiao;
83      }
84
85      public void setNome_regiao(String nome_regiao) {
86          this.nome_regiao = nome_regiao;
87      }
88
89      public Date getDtinicio() {
90          return dtinicio;
91      }
92
93      public void setDtinicio(Date dtinicio) {
94          this.dtinicio = dtinicio;
95      }
96
97      public void setDtfim(Date dtfim) {
98          this.dtfim = dtfim;
99      }
100
101     public Date getDtfim() {
102         return dtfim;
103     }
104 }
```

Listing 27: Código da classe Partida

```java
package model.tables;

import jakarta.persistence.*;
import model.embeddables.PartidaMId;
import model.embeddables.PartidaNId;

import java.io.Serial;
import java.io.Serializable;

@Entity
@Table(name="partida_multijogador")
public class Partida_MultiJogador implements Serializable {
    @Serial
    private static final long serialVersionUID = 1L;

    public Partida_MultiJogador() { }

    @EmbeddedId
    private PartidaMId partida_multiId;

    int pontuacao;

    @OneToOne
    @JoinColumns({
            @JoinColumn(name = "partida", referencedColumnName = "partida", insertable = false, updatable = fa
            @JoinColumn(name = "jogo", referencedColumnName = "jogo", insertable = false, updatable = false)
    })
    private Partida partida;


    @ManyToOne
    @MapsId
    @JoinColumn(name = "jogador", referencedColumnName = "jogador")
    private Jogador jogador;

    public Jogador getJogador() {
        return jogador;
    }

    public void setJogador(Jogador jogador) {
        this.jogador = jogador;
    }

    public Partida getPartida() {
        return partida;
    }

    public void setPontuacao(int pontuacao) {
        this.pontuacao = pontuacao;
    }
```

```
51
52      public void setPartida_multiId(PartidaMId partida) {
53          this.partida_multiId = partida;
54      }
55
56      public PartidaMId getPartida_multiId() {
57          return partida_multiId;
58      }
59
60      public int getPontuacao() {
61          return pontuacao;
62      }
63
64  }
```

Listing 28: Código da classe Partida_MultiJogador

```
1   package model.tables;
2
3   import jakarta.persistence.*;
4   import model.embeddables.PartidaNId;
5
6   import java.io.Serial;
7   import java.io.Serializable;
8
9   @Entity
10  @Table(name="partida_normal")
11  public class Partida_Normal extends Partida implements Serializable {
12
13      @Serial
14      private static final long serialVersionUID = 1L;
15
16      public Partida_Normal() { }
17
18      @EmbeddedId
19      private PartidaNId partida_normalId;
20
21      private int pontuacao;
22
23      @OneToOne
24      @JoinColumns({
25              @JoinColumn(name = "partida", referencedColumnName = "partida", insertable = false, updatable = fa
26              @JoinColumn(name = "jogo", referencedColumnName = "jogo", insertable = false, updatable = false)
27      })
28      private Partida partida;
29
30      public Partida getPartida() {
31          return partida;
32      }
33
```

```
34      @ManyToOne
35      @MapsId
36      @JoinColumn(name = "jogador", referencedColumnName = "jogador")
37      private Jogador jogador;
38
39      public Jogador getJogador() {
40          return jogador;
41      }
42
43      public void setJogador(Jogador jogador) {
44          this.jogador = jogador;
45      }
46
47      public PartidaNId getPartida_normalId() {
48          return partida_normalId;
49      }
50
51      public void setPartida_normalId(PartidaNId partida) {
52          this.partida_normalId = partida;
53      }
54
55      public int getPontuacao() {
56          return pontuacao;
57      }
58
59      public void setPontuacao(int pontuacao) {
60          this.pontuacao = pontuacao;
61      }
62
63
64  }
```

Listing 29: Código da classe Partida_Normal

```
1   package model.tables;
2
3
4   import jakarta.persistence.*;
5   import model.embeddables.CrachaId;
6   import model.relations.CrachasAdquiridos;
7
8   import java.io.Serial;
9   import java.io.Serializable;
10  import java.util.ArrayList;
11  import java.util.List;
12
13  @Table(name="cracha")
14  @Entity
15  public class Cracha implements Serializable {
16
```

```
17      @Serial
18      private static final long serialVersionUID = 1L;
19
20      public static long getSerialVersionUID() {
21          return serialVersionUID;
22      }
23
24      @EmbeddedId
25      private CrachaId id;
26
27      int limite;
28
29      String url;
30
31      @ManyToOne
32      @MapsId
33      @JoinColumn(name="jogo")
34      private Jogo jogo;
35
36      public Jogo getJogo() {
37          return jogo;
38      }
39
40      public void setJogo(Jogo jogo) {
41          this.jogo = jogo;
42      }
43
44      @OneToMany(mappedBy = "cracha")
45      private List<CrachasAdquiridos> crachasAdquiridosList = new ArrayList<>();
46
47      public List<CrachasAdquiridos> getCrachasList() {
48          return crachasAdquiridosList;
49      }
50
51      public void addCrachas(CrachasAdquiridos crachasAdquiridos) {
52          this.crachasAdquiridosList.add(crachasAdquiridos);
53      }
54
55
56      public CrachaId getId() {
57          return this.id;
58      }
59
60      public void setId(CrachaId id) {
61          this.id = id;
62      }
63
64      public int getLimite() {
65          return limite;
66      }
67
```

```
68      public void setLimite(int limit) {
69          this.limite = limit;
70      }
71
72      public String getUrl() {
73          return url;
74      }
75
76      public void setUrl(String url) {
77          this.url = url;
78      }
79
80  }
```

Listing 30: Código da classe Cracha

```
1   package model.tables;
2
3   import jakarta.persistence.*;
4   import model.relations.Participa;
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="conversa")
13  public class Conversa implements Serializable {
14
15      @Serial
16      private static final long serialVersionUID = 1L;
17
18      public Conversa() { }
19
20      @Id
21      @GeneratedValue(strategy = GenerationType.IDENTITY)
22      private int id;
23      private String nome;
24
25      // N:N relation participa
26      @OneToMany(mappedBy = "conversa")
27      private List<Participa> participantes = new ArrayList<>();
28
29      public List<Participa> getParticipantes(){ return this.participantes; }
30
31      public void addJogador(Participa participante){
32          this.participantes.add(participante);
33      }
34
```

```
35      // 1:N relation contem
36      @OneToMany(mappedBy = "conversa", cascade = CascadeType.PERSIST, orphanRemoval = true)
37      private List<Mensagem> mensagens = new ArrayList<>();
38
39      public List<Mensagem> getMensagens() {
40          return mensagens;
41      }
42
43      public void addMensagem(Mensagem mensagem) {
44          this.mensagens.add(mensagem);
45      }
46
47      //getter and setters
48      public int getId() { return id; }
49      public void setNome(String nome){ this.nome = nome; }
50      public String getNome(){ return this.nome; }
51  }
```

Listing 31: Código da classe Conversa

```
1   package model.tables;
2
3   import jakarta.persistence.*;
4   import model.embeddables.MensagemId;
5
6   import java.io.Serial;
7   import java.io.Serializable;
8   import java.sql.Date;
9
10  @Entity
11  @Table(name = "mensagem")
12  public class Mensagem implements Serializable {
13
14      @Serial
15      private static final long serialVersionUID = 1L;
16      public Mensagem() { }
17
18      @EmbeddedId
19      private MensagemId mensagemId;
20
21      String conteudo;
22
23      Date data;
24
25
26      @ManyToOne
27      @MapsId
28      @JoinColumn(name="conversa")
29      private Conversa conversa;
30
```

```
31      public MensagemId getMensagemId() {
32          return mensagemId;
33      }
34
35      public void setMensagemId(MensagemId mensagemId) {
36          this.mensagemId = mensagemId;
37      }
38
39      public Date getData() {
40          return data;
41      }
42
43      public void setData(Date data) {
44          this.data = data;
45      }
46
47      public String getConteudo() {
48          return conteudo;
49      }
50
51      public void setConteudo(String conteudo) {
52          this.conteudo = conteudo;
53      }
54  }
```

Listing 32: Código da classe Mensagem

Anexo D

Código das Classes do *Package* businessLogic

```java
package businessLogic;

import java.math.BigDecimal;
import java.sql.CallableStatement;
import java.sql.Connection;
import java.sql.Types;
import java.util.List;
import java.util.Map;


import businessLogic.BLserviceUtils.ModelManager;
import businessLogic.BLserviceUtils.TransactionManager;
import jakarta.persistence.*;
import model.embeddables.CrachasAdquiridosId;
import model.relations.CrachasAdquiridos;
import model.tables.Cracha;
import model.tables.Jogador;
import model.views.JogadorTotalInfo;

/**
 * Hello world!
 *
 */
public class BLService
{
    EntityManagerFactory emf = Persistence.createEntityManagerFactory("JPAex");
    EntityManager em = emf.createEntityManager();

    private final ModelManager modelManager = new ModelManager(em);
    private final TransactionManager transactionManager = new TransactionManager(em);

    /**
     * 1. (a)
     * Access to the funcionalities 2d to 2l
     */
```

```
36      // These 2 functions constitute the exercise 2d, these do not require a transaction level above
37      // read uncommitted since they only preform an update to the respective tables once
38      public String createPlayer(String username, String email, String regiao) {
39          String query = "SELECT createPlayer(?1, ?2, ?3)";
40          Query functionQuery = em.createNativeQuery(query)
41                  .setParameter(1, username)
42                  .setParameter(2, email)
43                  .setParameter(3, regiao);
44          return (String) functionQuery.getSingleResult();
45
46      }
47
48      public String setPlayerState(String playerName, String newState) {
49          int idJogador = modelManager.getPlayerByEmail(playerName, em).getId();
50          String query = "SELECT setPlayerState(?1, ?2)";
51          Query functionQuery = em.createNativeQuery(query)
52                  .setParameter(1, idJogador)
53                  .setParameter(2, newState);
54          return (String) functionQuery.getSingleResult();
55      }
56
57      // Exercise 2e
58      public Long totalPontosJogador(String email) {
59          int idJogador = modelManager.getPlayerByEmail(email, em).getId();
60          String query = "SELECT totalPontos from totalPontosJogador(?1)";
61          Query functionQuery = em.createNativeQuery(query);
62          functionQuery.setParameter(1, idJogador);
63          return (Long) functionQuery.getSingleResult();
64      }
65
66      // Exercise 2f
67      public Long totalJogosJogador(String email) {
68          int idJogador = modelManager.getPlayerByEmail(email, em).getId();
69          String query = "SELECT totalJogos from totalJogosJogador(?1)";
70          Query functionQuery = em.createNativeQuery(query);
71          functionQuery.setParameter(1, idJogador);
72          return (Long) functionQuery.getSingleResult();
73      }
74
75      // Exercise 2g (Temporary implementation, not optimized)
76
77      public Map<Integer, BigDecimal> PontosJogosPorJogador(String gameName) {
78          String idJogo = modelManager.getGameByName(gameName, em).getId();
79          String queryString = "SELECT jogadores, pontuacaoTotal from PontosJogosPorJogador(?1)";
80          Query query = em.createNativeQuery(queryString);
81          query.setParameter(1, idJogo);
82          Map<Integer, BigDecimal> map = new java.util.HashMap<>(Map.of());
83          List<Object[]> list  = query.getResultList();
84          for (Object[] obj : list) {
85              Integer idJogador = (Integer) obj[0];
86              BigDecimal points = (BigDecimal) obj[1];
```

```
87              map.put(idJogador,  points);
88          }
89          return map;
90      }
91
92
93      // Exercise 2h
94      public void associarCracha(int idJogador, String gameName, String nomeCracha) {
95          EntityTransaction transaction = transactionManager.startTransaction();
96          Connection cn = em.unwrap(Connection.class);
97          String idJogo = modelManager.getGameByName(gameName, em).getNome();
98          try {
99              transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_REPEATABLE_READ, transaction);
100             try (CallableStatement storedProcedure = cn.prepareCall("call associarCracha(?,?, ?)")) {
101                 storedProcedure.setInt(1, idJogador);
102                 storedProcedure.setString(2, idJogo);
103                 storedProcedure.setString(3, nomeCracha);
104                 storedProcedure.executeUpdate();
105                 transaction.commit();
106             }
107         } catch(Exception e){
108             if(transaction.isActive()) transaction.rollback();
109         }
110     }
111
112     // Exercise 2i
113     public Integer iniciarConversa(int idJogador, String nomeConversa) {
114         EntityTransaction transaction = transactionManager.startTransaction();
115         Connection cn = em.unwrap(Connection.class);
116         Integer idConversa = null;
117         try {
118             transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_REPEATABLE_READ, transaction);
119             try (CallableStatement storedProcedure = cn.prepareCall("call iniciarConversa(?,?, ?)")) {
120                 storedProcedure.setInt(1, idJogador);
121                 storedProcedure.setString(2, nomeConversa);
122                 storedProcedure.registerOutParameter(3, Types.INTEGER);
123                 storedProcedure.executeUpdate();
124                 idConversa = storedProcedure.getInt(3);
125                 transaction.commit();
126             }
127         } catch(Exception e){
128             if(transaction.isActive()) transaction.rollback();
129             return null;
130         }
131         return idConversa;
132     }
133
134     // Exercise 2j
135     public void juntarConversa(int idJogador, int idConversa) {
136         EntityTransaction transaction = transactionManager.startTransaction();
137         Connection cn = em.unwrap(Connection.class);
```

```
138         try {
139             try (CallableStatement storedProcedure = cn.prepareCall("call juntarConversa(?,?)")) {
140                 storedProcedure.setInt(1, idJogador);
141                 storedProcedure.setInt(2, idConversa);
142                 transaction.commit();
143             }
144         } catch(Exception e){
145             if(transaction.isActive()) transaction.rollback();
146         }
147     }
148 
149     // Exercise 2k
150     public void enviarMensagem(int idJogador, int idConversa, String content) {
151 
152         EntityTransaction transaction = transactionManager.startTransaction();
153         Connection cn = em.unwrap(Connection.class);
154         try {
155             transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_READ_COMMITTED, transaction);
156             try (CallableStatement storedProcedure = cn.prepareCall("call enviarMensagem(?,?, ?)")) {
157                 cn.setTransactionIsolation(Connection.TRANSACTION_READ_COMMITTED);
158                 storedProcedure.setInt(1, idJogador);
159                 storedProcedure.setInt(2, idConversa);
160                 storedProcedure.setString(3, content);
161                 storedProcedure.executeUpdate();
162                 transaction.commit();
163             }
164         } catch(Exception e){
165             if(transaction.isActive()) transaction.rollback();
166         }
167     }
168 
169     public void jogadorTotalInfo(){
170         String query = "Select * from jogadorTotalInfo";
171         Query getAllInfo = em.createNativeQuery(query, JogadorTotalInfo.class);
172         List<JogadorTotalInfo> allInfo =  getAllInfo.getResultList();
173         for(JogadorTotalInfo jogador: allInfo){
174             System.out.println(
175                     jogador.getEstado() + " " +
176                     jogador.getEmail() + " " +
177                     jogador.getUsername() + " " +
178                     jogador.getJogosParticipados() + " " +
179                     jogador.getPartidasParticipadas() + " " +
180                     jogador.getPontuacaoTotal() + " "
181             );
182         }
183     }
184 
185 
186     public void associarCrachaModel(int idJogador, String gameName, String nomeCracha){
187         String idJogo = modelManager.getGameByName(gameName, em).getId();
188         TypedQuery<String> crachaExistsQuery =
```

```
189                 em.createQuery(
190                         "select c.id.jogo from Cracha c where c.id.jogo = :idJogo and c.id.nome = :nomeCracha"
191                         String.class
192                 );
193 
194         crachaExistsQuery.setParameter("idJogo", idJogo);
195         crachaExistsQuery.setParameter("nomeCracha", nomeCracha);
196         String foundIdJogo = crachaExistsQuery.getSingleResult();
197 
198         if(foundIdJogo.equals(idJogo)){
199             TypedQuery<Integer> getLimitPoints = em.createQuery(
200                     "select c.limite from Cracha c where c.id.nome = :nomeCracha",
201                     Integer.class
202             );
203             getLimitPoints.setParameter("nomeCracha", nomeCracha);
204             Integer limit = getLimitPoints.getSingleResult();
205 
206             if(modelManager.ownsGame(idJogador, idJogo)){
207                 BigDecimal totalPoints = modelManager.getPlayerPoints(idJogo, idJogador);
208                 if(limit <= totalPoints.intValue() ){
209 
210                     if(!modelManager.ownsBadge(idJogador)){
211                         CrachasAdquiridos crachaAdquirido = new CrachasAdquiridos();
212                         CrachasAdquiridosId crachasAdquiridosId = modelManager.setCrachaAdquiridoId(idJogo, no
213                         crachaAdquirido.setId(crachasAdquiridosId);
214 
215                         Cracha cracha = em.find(Cracha.class, modelManager.setCrachaId(idJogo, nomeCracha));
216                         crachaAdquirido.setCracha(cracha);
217 
218                         Jogador jogador = em.find(Jogador.class, idJogador); // Fetch the Jogador entity by ID
219                         crachaAdquirido.setJogador(jogador);
220 
221 
222                     }
223 
224                 }
225 
226             }
227 
228         }
229     }
230 
231     private void aumentarPontosCracha20(String nomeCracha, String idJogo, LockModeType lockType) {
232         EntityTransaction transaction = transactionManager.startTransaction();
233         Connection cn = em.unwrap(Connection.class);
234         try {
235             transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_READ_COMMITTED, transaction);
236             String selectQuery = "SELECT c FROM Cracha c WHERE c.id.nome = ?1 AND c.id.jogo = ?2";
237             TypedQuery<Cracha> selectTypedQuery = em.createQuery(selectQuery, Cracha.class);
238             selectTypedQuery.setParameter(1, nomeCracha);
239             selectTypedQuery.setParameter(2, idJogo);
```

```
240             selectTypedQuery.setLockMode(lockType);
241             Cracha cracha = selectTypedQuery.getSingleResult();
242             transactionManager.setIsolationLevel(cn, Connection.TRANSACTION_REPEATABLE_READ, transaction);
243             String query =
244                     "UPDATE crachá c SET c.limit = c.limit * 1.2, c.version = c.version + 1 " +
245                             "WHERE c.id = :crachaId AND c.version = :crachaVersion";
246             Query updateQuery = em.createNativeQuery(query);
247             updateQuery.setParameter("crachaId", cracha.getId());
248             updateQuery.setParameter("crachaVersion", Cracha.getSerialVersionUID());
249             int updatedCount = updateQuery.executeUpdate();
250             if (updatedCount == 0) {
251                 switch (lockType) {
252                     case OPTIMISTIC -> throw new OptimisticLockException("Concurrent update detected for Crach
253                     case PESSIMISTIC_READ -> throw new PessimisticLockException("Concurrent update detected fo
254                     default -> throw new Exception("Concurrent update detected for Cracha");
255                 }
256             }
257             transaction.commit();
258         } catch (Exception e) {
259             if (transaction.isActive()) transaction.rollback();
260             System.out.println("Error Message: " + e.getMessage());
261         }
262     }
263
264     /**
265      * 2.(a), 2.(b), 2.(c)
266      */
267
268     // Exercise 2 (a)
269     public void aumentarPontosOptimistic(String nomeCracha, String idJogo) {
270         aumentarPontosCracha20(nomeCracha, idJogo, LockModeType.OPTIMISTIC);
271     }
272
273     // Exercise 2 (b) -> Inside Test Folder
274
275     // Exercise 2 (c)
276     public void aumentarPontosPessimistic(String nomeCracha, String idJogo, LockModeType lockType) {
277         aumentarPontosCracha20(nomeCracha, idJogo, LockModeType.PESSIMISTIC_READ);
278     }
279 }
```

Listing 33: Código da classe BLService

Anexo E

Código das Classes do *Package* businessLogic.BLServiceUtils

```java
package businessLogic.BLserviceUtils;

import jakarta.persistence.EntityManager;
import jakarta.persistence.EntityTransaction;
import jakarta.persistence.Query;
import jakarta.persistence.TypedQuery;
import model.embeddables.CrachaId;
import model.embeddables.CrachasAdquiridosId;
import model.relations.Compra;
import model.tables.Jogador;
import model.tables.Jogo;

import java.math.BigDecimal;
import java.sql.Connection;
import java.sql.SQLException;

public class ModelManager {
    public ModelManager(EntityManager em){
        this.em = em;

    }
    private final EntityManager em;


    public Jogo getGameByName(String gameName, EntityManager em){
        TypedQuery<Jogo> query = em.createQuery("SELECT j FROM Jogo j WHERE j.nome = ?1", Jogo.class);
        query.setParameter(1, gameName);
        return query.getSingleResult();
    }

    public Jogador getPlayerByEmail(String email, EntityManager em){
        TypedQuery<Jogador> query = em.createQuery("SELECT j FROM Jogador j WHERE j.email = ?1", Jogador.class
        query.setParameter(1, email);
        return query.getSingleResult();
    }
```

```
36
37      public CrachasAdquiridosId setCrachaAdquiridoId(String idJogo, String nomeCracha, int idJogador){
38          CrachasAdquiridosId crachasAdquiridosId = new CrachasAdquiridosId();
39          crachasAdquiridosId.setJogo(idJogo);
40          crachasAdquiridosId.setCracha(nomeCracha);
41          crachasAdquiridosId.setJogador(idJogador);
42          return  crachasAdquiridosId;
43      }
44
45      public CrachaId setCrachaId(String idJogo, String nomeCracha){
46          CrachaId crachaId = new CrachaId();
47          crachaId.setJogo(idJogo);
48          crachaId.setNome(nomeCracha);
49          return crachaId;
50      }
51
52      public BigDecimal getPlayerPoints(String idJogo, int idJogador){
53          String jogadoresQuery = "SELECT pontuacaoTotal from PontosJogosPorJogador(?1) WHERE jogadores = ?2";
54          Query pontuacaoQuery = em.createNativeQuery(jogadoresQuery);
55          pontuacaoQuery.setParameter(1, idJogo);
56          pontuacaoQuery.setParameter(2, idJogador);
57          return (BigDecimal) pontuacaoQuery.getSingleResult();
58      }
59
60      public Boolean ownsGame(int idJogador, String idJogo){
61          TypedQuery<Compra> getCompra = em.createQuery(
62                  "select c from Compra c "  +
63                          "where c.jogador.id = :idJogador and c.jogo.id = :idJogo",
64                  Compra.class
65          );
66          getCompra.setParameter("idJogador", idJogador);
67          getCompra.setParameter("idJogo", idJogo);
68          return getCompra.getResultList().isEmpty();
69      }
70
71      public Boolean ownsBadge(int idJogador){
72          Query hasCracha = em.createQuery("SELECT ca.id.jogo from CrachasAdquiridos ca WHERE ca.id.jogador = ?1
73          hasCracha.setParameter(1, idJogador);
74          return !hasCracha.getResultList().isEmpty();
75      }
76  }
```

Listing 34: Código da classe ModelManager

```
1   package businessLogic.BLserviceUtils;
2
3   import jakarta.persistence.EntityManager;
4   import jakarta.persistence.EntityTransaction;
5
6   import java.sql.Connection;
```

```
7  import java.sql.SQLException;
8
9  public class TransactionManager {
10
11     public TransactionManager(EntityManager em){
12         this.em = em;
13     }
14
15     private EntityManager em;
16     public EntityTransaction startTransaction(){
17         EntityTransaction transaction = em.getTransaction();
18         transaction.begin();
19         return transaction;
20     }
21
22     public void setIsolationLevel(Connection cn, Integer isolationLevel, EntityTransaction transaction) throws
23         transaction.rollback();
24         cn.setTransactionIsolation(isolationLevel);
25         transaction.begin();
26     }
27 }
```

Listing 35: Código da classe TransactionManager