LIGA BALONCESTO EN JAVA/MYSQL

Oliver Eduardo Lopez Perez

Campuslands

P1: SPUTNIK

Pedro Felipe Gomez Bonilla

2 de Agosto de 2024

Tabla de contenido

Introducción

El propósito de este proyecto es desarrollar un sistema para gestionar los partidos de una liga de baloncesto. Este sistema permitirá registrar partidos, registrar puntos para los equipos locales y visitantes, finalizar partidos, mostrar el ganador y obtener información básica de los partidos.

Caso de estudio

- Liga de baloncesto
La liga de baloncesto necesita un pequeño sistema para los partidos que se están llevando a cabo.

Los partidos contienen características en común como equipo local, equipo visitante, las cestas del equipo local, las cestas del equipo visitante, si el partido ha finalizado o no y la fecha del partido, además es necesario obtener la información básica del partido. Adicionalmente, existen dos clases de partidos, los partidos de liga, son partidos regulares que se desarrollan en un numero de jornada, por otra parte, los partidos playOffs , son partidos que se desarrollan en una ronda, como "octavos", "cuartos", "final".

Es obligatorio para todos los partidos obtener el resultado, registrar puntos para los locales, registrar puntos para los visitantes, obtener el ganador y dar por finalizado el encuentro. En los partidos playOffs si los equipos se encuentran empatados, no será posible finalizar el partido hasta que haya un ganador.

Finalmente, es necesario que el sistema cuente con un menú para poder registrar el partido según su clase y registrar los datos del partido, por defecto al registrar un partido su estado será en juego. También, la opción de poder finalizar el partido, mostrar el ganador y la información del partido.

Instalación general

Para ejecutar este proyecto, se requiere lo siguiente:

Java Development Kit (JDK) 8 o superior:

- Conjunto de herramientas para desarrollo con el lenguaje de programación “Java”.

Un entorno de desarrollo integrado (IDE) como IntelliJ o NetBeans:

- Aplicativo el cual facilita la creación de los diferentes archivos contenedores del código a ejecutar.

MySQL para la base de datos:

- Aplicativo el cual facilita la creación de tablas para almacenar información.

Clever Cloud para alojar la base de datos:

- Página web la cual permite crear y alojar bases de datos globales (Manejables desde cualquier parte).

En el repositorio de Github se encuentra una carpeta contenedora del proyecto con los archivos base correspondientes a un proyecto de Java por defecto.

De igual manera se encuentran en la carpeta “src/ligabaloncesto” las siguientes clases (Archivos.java):

Database.java (Conector del proyecto con la base de datos)
Ligabaloncesto.java (Clase principal para iniciar el sistema)
Partido.java (Clase abstracta que representa un partido de baloncesto.)
PartidoLiga.jaca (Clase que representa un partido de liga regular.)
PartidoPlayoff.java (Clase que representa un partido de play-offs.)
Sistema.java (Clase para gestionar la lógica del sistema y la interacción con el usuario.)

Ejecución

Construcción del Modelo Físico:

- Descripción:

El modelo físico del sistema se basa en la base de datos MySQL que almacena la información de los partidos.

- Implementación técnica:

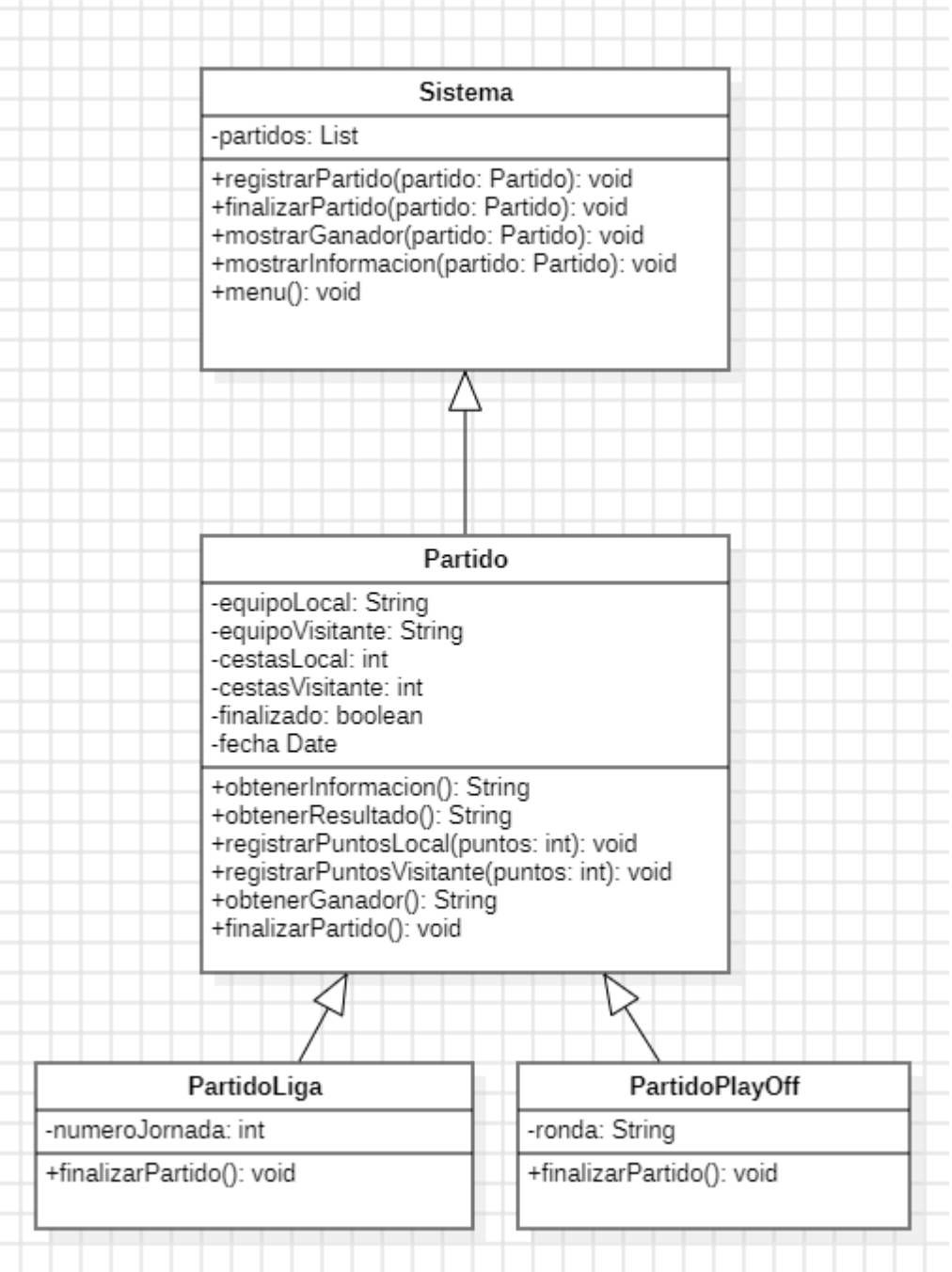
```sql
CREATE TABLE partidos (
    id INT AUTO_INCREMENT PRIMARY KEY,
    equipo_local VARCHAR(50) NOT NULL,
    equipo_visitante VARCHAR(50) NOT NULL,
    cestas_local INT DEFAULT 0,
    cestas_visitante INT DEFAULT 0,
    finalizado BOOLEAN DEFAULT FALSE,
    fecha DATE NOT NULL,
    tipo_partido VARCHAR(50) NOT NULL,
    numero_jornada INT NULL,
    ronda VARCHAR(50) NULL
);
```

Construcción del Modelo UML:

- Descripción:

El modelo UML describe las clases y las relaciones entre ellas en el sistema.

- Implementación técnica:



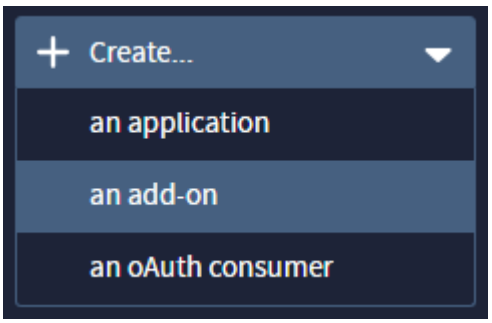
Base de Datos

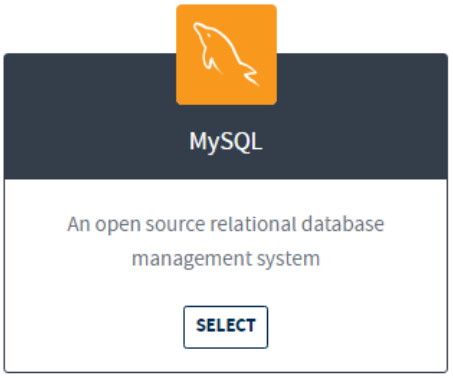
- Descripción

Alojamos la base de datos en Clever Cloud para facilitar el acceso y la gestión de la misma.

- Implementación:

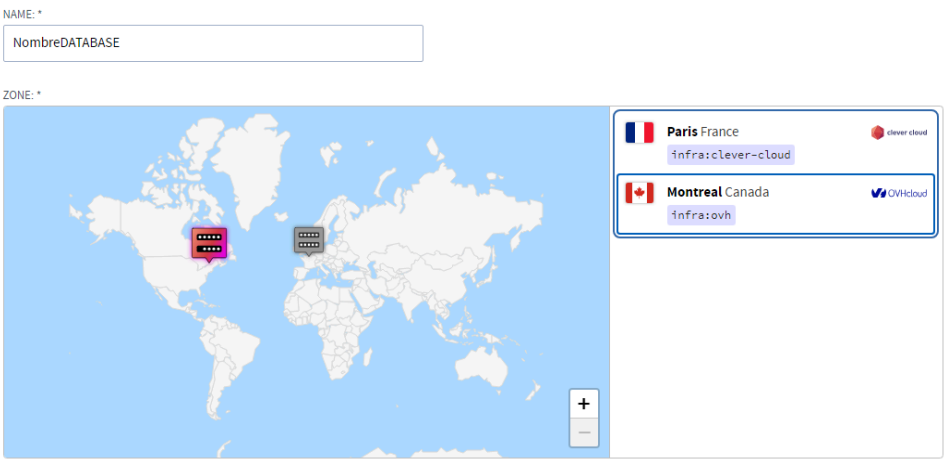
Crear Add-on de mySQL:





Seleccionar plan gratuito:

Nombrar base de datos y seleccionar servidor (fué usado el de canadá para este proyecto):

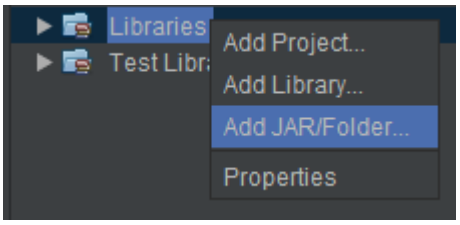


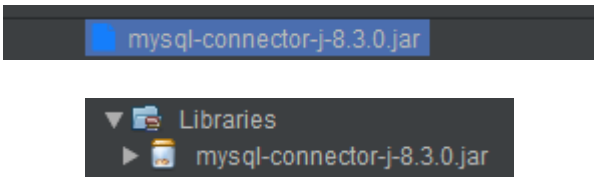
Importación de librerías

- Descripción:

Las librerías son herramientas externas a usar en entornos de desarrollo.
En este caso tenemos la librería “mysql-connector-j-8.3.0.jar” la cual permite conectar con una base de datos y ejecutar operaciones desde las clases de java.

- Implementación:
-





Conexión con MySQL usando JAVA (Database.java)

- Descripción

Configurar la conexión entre la base de datos MySQL y la aplicación Java en una clase.

- Implementación:

Importación de librerías (Con el .jar descargado previamente):

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;
```

Función (clase) para llamar la información de la base de datos (Nombre y contraseña) para la gestión de la info a contener aquí:

```
public class Database {
    private static final String URL = "jdbc:mysql://host:3306/user";
    private static final String USER = "user";
    private static final String PASSWORD = "password";

    public static Connection getConnection() throws SQLException {
        return DriverManager.getConnection(url:URL, user:USER, password:PASSWORD);
    }
}
```

Partido

- Descripción:

La clase Partido es una clase abstracta que representa un partido de baloncesto.

- Implementación:

Atributos: equipoLocal, equipoVisitante, cestasLocal, cestasVisitante, finalizado, fecha almacenan la información básica de un partido.

```
public abstract class Partido {
    protected String equipoLocal;
    protected String equipoVisitante;
    protected int cestasLocal;
    protected int cestasVisitante;
    protected boolean finalizado;
    protected Date fecha;

    public Partido(String equipoLocal, String equipoVisitante, Date fecha) {
        this.equipoLocal = equipoLocal;
        this.equipoVisitante = equipoVisitante;
        this.cestasLocal = 0;
        this.cestasVisitante = 0;
        this.finalizado = false;
        this.fecha = fecha;
    }
```

registrarPuntosLocal, registrarPuntosVisitante: Métodos para registrar puntos a los equipos:

```
public void registrarPuntosLocal(int puntos) {
    cestasLocal += puntos;
}

public void registrarPuntosVisitante(int puntos) {
    cestasVisitante += puntos;
}
```

guardarPartido: Guarda la información del partido en la base de datos:

```
public void guardarPartido(Connection conn) throws SQLException {
    String sql = "INSERT INTO partidos (equipo_local, equipo_visitante, cestas_local, cestas_visitante, finalizado, fecha) VALUES (?, ?, ?, ?, ?, ?)";
    try (PreparedStatement pstmt = conn.prepareStatement(string: sql)) {
        pstmt.setString(i: 1, string: equipoLocal);
        pstmt.setString(i: 2, string: equipoVisitante);
        pstmt.setInt(i: 3, i1: cestasLocal);
        pstmt.setInt(i: 4, i1: cestasVisitante);
        pstmt.setBoolean(i: 5, bln:finalizado);
        pstmt.setDate(i: 6, new java.sql.Date(date:fecha.getTime()));
        pstmt.executeUpdate();
    }
}
```

finalizarPartido, obtenerResultados, obtenerInformacion: Métodos abstractos que deben ser implementados por las subclases:

```
public abstract void finalizarPartido();
```

```
public String obtenerInformacion() {
    return "Partido entre " + equipoLocal + " y " + equipoVisitante + " el " + fecha.toString();
}

public String obtenerResultado() {
    return equipoLocal + " " + cestasLocal + " - " + cestasVisitante + " " + equipoVisitante;
}
```

PartidoLiga

- Descripción:

La clase PartidoLiga extiende Partido y representa un partido de liga regular.

- Implementación:

numeroJornada: Almacena el número de jornada en que se juega el partido:

```
public class PartidoLiga extends Partido {
    private int numeroJornada;

    public PartidoLiga(String equipoLocal, String equipoVisitante, Date fecha, int numeroJornada) {
        super(equipoLocal, equipoVisitante, fecha);
        this.numeroJornada = numeroJornada;
    }
```

finalizarPartido: Finaliza el partido marcándolo como finalizado:

```
public void finalizarPartido() {
    this.finalizado = true;
}
```

PartidoPlayOff

- Descripción:

La clase PartidoPlayOff extiende Partido y representa un partido de play-offs.

- Implementación:

ronda: Almacena la ronda en que se juega el partido:

```
public class PartidoPlayOff extends Partido {
    private String ronda;

    public PartidoPlayOff(String equipoLocal, String equipoVisitante, Date fecha, String ronda) {
        super(equipoLocal, equipoVisitante, fecha);
        this.ronda = ronda;
    }
```

finalizarPartido: Finaliza el partido solo si no hay empate:

```
public void finalizarPartido() {
    if (cestasLocal != cestasVisitante) {
        this.finalizado = true;
    } else {
        System.out.println(x: "El partido no puede finalizar en empate en PlayOffs.");
    }
}
```

Sistema

- Descripción:

La clase Sistema gestiona la lógica del sistema y la interacción con el usuario.

- Implementación:

partidos: Lista que almacena todos los partidos registrados:

```
public class Sistema {
    private List<Partido> partidos;

    public Sistema() {
        this.partidos = new ArrayList<>();
    }
```

registrarPartido: Añade un partido a la lista y lo guarda en la base de datos:

```
public void registrarPartido(Partido partido) {
    partidos.add(e: partido);
    try (Connection conn = Database.getConnection()) {
        partido.guardarPartido(conn);
    } catch (SQLException e) {
        System.out.println("Error al guardar el partido: " + e.getMessage());
    }
}
```

registrarPuntosLocal, registrarPuntosVisitante: Registran los puntos para los equipos locales y visitantes, respectivamente, y actualizan la base de datos:

```
public void registrarPuntosLocal(int indice, int puntos) {
    Partido partido = partidos.get(index: indice);
    partido.registrarPuntosLocal(puntos);
    actualizarPuntosEnBD(partido);
}

public void registrarPuntosVisitante(int indice, int puntos) {
    Partido partido = partidos.get(index: indice);
    partido.registrarPuntosVisitante(puntos);
    actualizarPuntosEnBD(partido);
}
```

actualizarPuntosEnBD: Actualiza los puntos de un partido en la base de datos:

```
private void actualizarPuntosEnBD(Partido partido) {
    String sql = "UPDATE partidos SET cestas_local = ?, cestas_visitante = ? WHERE equipo_local = ? AND equipo_visitante = ? AND fecha = ?";
    try (Connection conn = Database.getConnection();
         PreparedStatement pstmt = conn.prepareStatement(string: sql)) {
        pstmt.setInt(i: 1, i1: partido.cestasLocal);
        pstmt.setInt(i: 2, i1: partido.cestasVisitante);
        pstmt.setString(i: 3, string: partido.equipoLocal);
        pstmt.setString(i: 4, string: partido.equipoVisitante);
        pstmt.setDate(i: 5, new java.sql.Date(date:partido.fecha.getTime()));
        pstmt.executeUpdate();
    } catch (SQLException e) {
        System.out.println("Error al actualizar los puntos: " + e.getMessage());
    }
}
```

finalizarPartido: Finaliza un partido y actualiza su estado en la base de datos.

```
public void finalizarPartido(Partido partido) {
    partido.finalizarPartido();
    try (Connection conn = Database.getConnection()) {
        String sql = "UPDATE partidos SET finalizado = ? WHERE equipo_local = ? AND equipo_visitante = ? AND fecha = ?";
        try (PreparedStatement pstmt = conn.prepareStatement(string: sql)) {
            pstmt.setBoolean(i: 1, bln:partido.finalizado);
            pstmt.setString(i: 2, string: partido.equipoLocal);
            pstmt.setString(i: 3, string: partido.equipoVisitante);
            pstmt.setDate(i: 4, new java.sql.Date(date:partido.fecha.getTime()));
            pstmt.executeUpdate();
        }
    } catch (SQLException e) {
        System.out.println("Error al finalizar el partido: " + e.getMessage());
    }
}
```

mostrarGanador: Muestra el ganador de un partido:

```
public void mostrarGanador(Partido partido) {
    System.out.println("Ganador: " + partido.obtenerGanador());
}
```

mostrarInformacion: Muestra la información de un partido:

```
public void mostrarInformacion(Partido partido) {
    System.out.println(x: partido.obtenerInformacion());
}
```

menú: Método principal que presenta un menú para interactuar con el usuario y gestionar los partidos:

```
public void menu() {
    Scanner scanner = new Scanner(source: System.in);
    while (true) {
        System.out.println(x: "1. Registrar partido");
        System.out.println(x: "2. Registrar puntos para equipo local");
        System.out.println(x: "3. Registrar puntos para equipo visitante");
        System.out.println(x: "4. Finalizar partido");
        System.out.println(x: "5. Mostrar ganador");
        System.out.println(x: "6. Mostrar información del partido");
        System.out.println(x: "7. Salir");
        int opcion = scanner.nextInt();
        scanner.nextLine(); // consumir el salto de línea
```

Cada opción ejecuta las clases (funciones) anteriormente establecidas como por ejemplo, al registrar un partido:

```
switch (opcion) {
    case 1:
        System.out.println(x: "Tipo de partido (1. Liga, 2. PlayOff): ");
        int tipo = scanner.nextInt();
        scanner.nextLine(); // consumir el salto de línea

        System.out.println(x: "Equipo local: ");
        String equipoLocal = scanner.nextLine();
        System.out.println(x: "Equipo visitante: ");
        String equipoVisitante = scanner.nextLine();
        System.out.println(x: "Fecha (dd/MM/yyyy): ");
        String fechaStr = scanner.nextLine();
        Date fecha = new Date(s: fechaStr);

        if (tipo == 1) {
            System.out.println(x: "Número de jornada: ");
            int jornada = scanner.nextInt();
            Partido partidoLiga = new PartidoLiga(equipoLocal, equipoVisitante, fecha, numeroJornada: jornada);
            registrarPartido(partido:partidoLiga);
        } else {
            System.out.println(x: "Ronda (octavos, cuartos, final): ");
            String ronda = scanner.nextLine();
            Partido partidoPlayOff = new PartidoPlayOff(equipoLocal, equipoVisitante, fecha, ronda);
            registrarPartido(partido:partidoPlayOff);
        }
        break;
```

Este menú se navega usando la libreria de Scanner, la cual nos permite escribir desde la consola de ejecución del proyecto:.

```
run:
1. Registrar partido
2. Registrar puntos para equipo local
3. Registrar puntos para equipo visitante
4. Finalizar partido
5. Mostrar ganador
6. Mostrar información del partido
7. Salir
```

LigaBaloncesto

- Descripción:

Clase principal para ejecutar el sistema.

- Implementación:

LigaBaloncesto: Clase que contiene el método main para ejecutar el sistema:

```
public class LigaBaloncesto {
    public static void main(String[] args) {
        Sistema sistema = new Sistema();
        sistema.menu();
    }
}
```