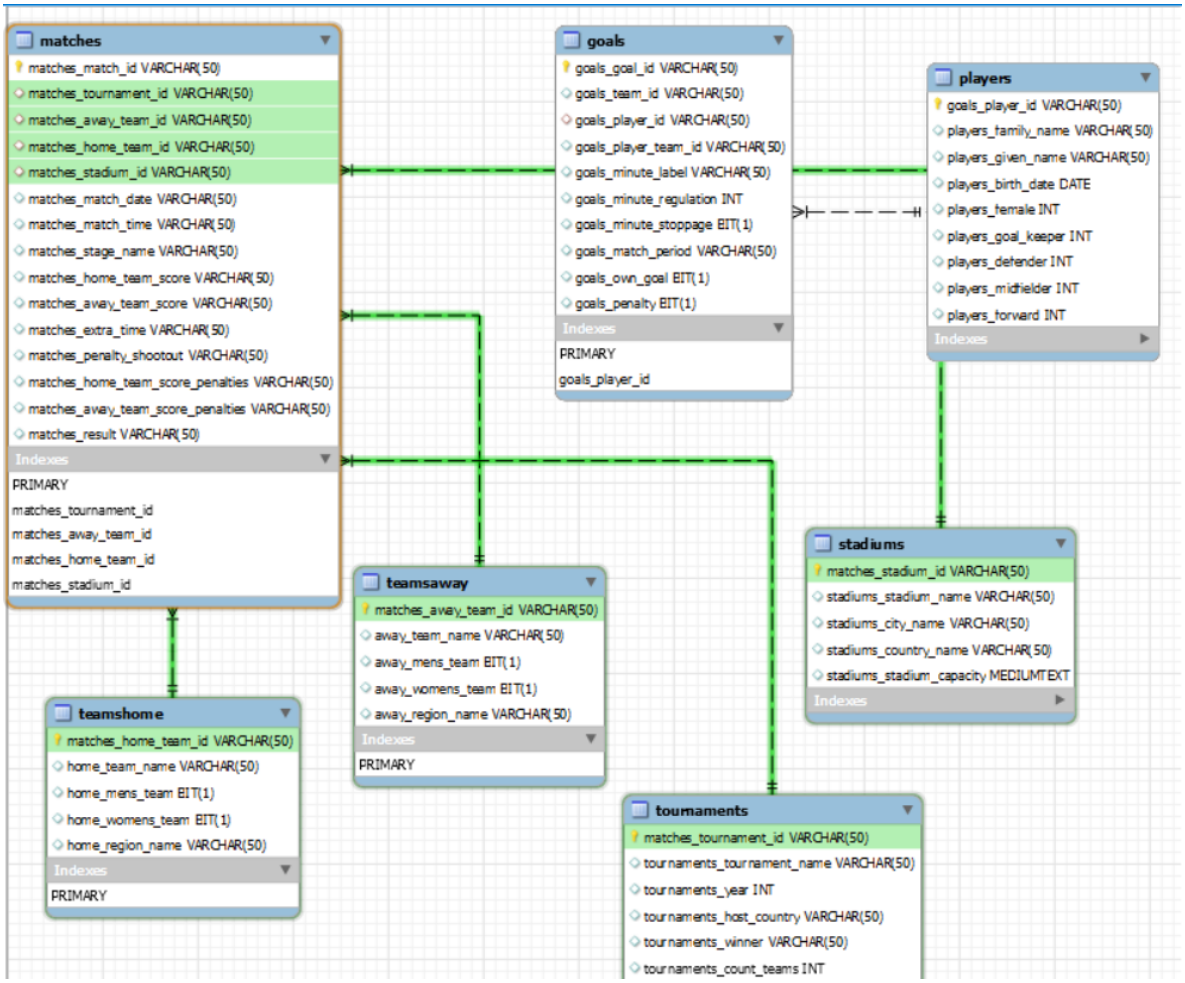
Avances Proyecto Integrador 3

Integrantes:

- Morales Luis
- Santiago Riofrio

- Modelo de base de datos (diseño físico).

Para obtener el modelo físico de base de datos se uso el entorno integrado MYSQL WorckBeanch



- Códigos de conexión y consultas.

El siguiente código fue implementado en Scala 3, el cual nos permite conectarnos con la base de datos y nos ayudara a hacer consultas dentro de ella

```scala
object DBDemo{
  @main def demo(): Unit =
    println("Demo")

    val xa = Transactor.fromDriverManager[IO](

      driver = "com.mysql.cj.jdbc.Driver",
      url = "jdbc:mysql://localhost:3306/sakila",
      user = "root",
      password = "#23@luis2002",
      logHandler = None

    )
```

- Códigos de consultas.

Funciones que nos permitirán realizar las consultas por medio de código a tra vez de funciones

```scala
def find(id: Int): ConnectionIO[Option[Actor]] =

  sql"SELECT a.actor_id, a.first_name, a.last_name FROM actor a where a.actor_id = $id"
    .query[Actor]
    .option

def listAllActors(): ConnectionIO[List[Actor]] =

  sql"SELECT a.actor_id, a.first_name, a.last_name FROM actor a"
    .query[Actor]
    .to[List]

def listOfFilms(): ConnectionIO[List[Film]] =

  sql"""
        SELECT f.film_id, f.title, f.release_year, group_concat(CONCAT(a.first_name, ' ', a.last_name)) as actor_list
        FROM film f, film_actor fa, actor a
        WHERE f.film_id = fa.film_id
        AND fa.actor_id = a.actor_id
        GROUP BY f.film_id, f.title
        """
    .query[Film]
    .to[List]
def tittleCategory(): ConnectionIO[List[CategoryFilm]] =

  sql"""
        SELECT f.title, c.name AS category_name
        FROM film f, category c, film_category fc
        WHERE f.film_id = fc.film_id
        AND c.category_id = fc.category_id
        ORDER BY f.title
listOfFilms()
```

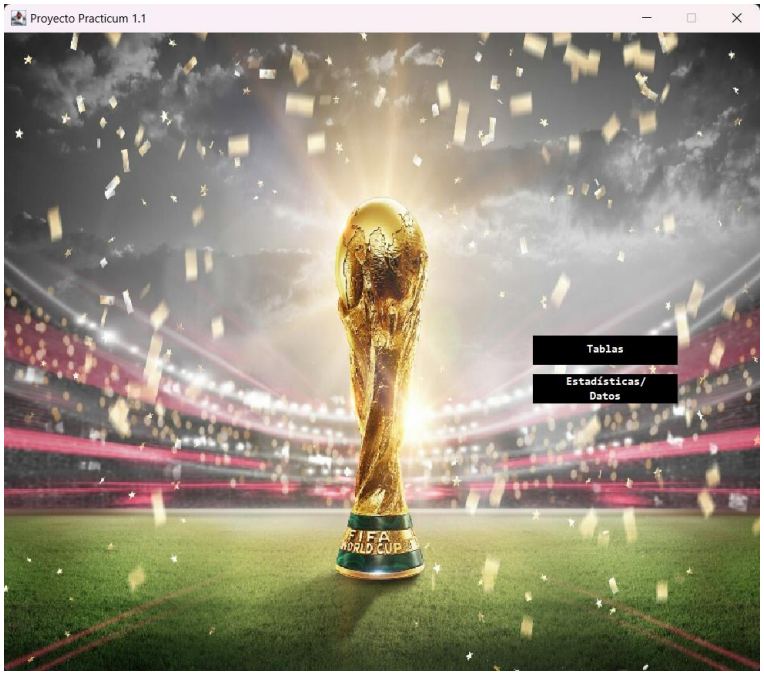
Avance 2 (Avance de Interfaz Completo)

Para el desarrollo de la interfaz gráfica de nuestro proyecto realizamos un boceto como referencia para ir creando poco a poco nuestra interfaz. Este boceto se realizado en canva, y la implementación, bien no está terminada totalmente, si se han podido implementar la página principal, y la segunda página.

La segunda página se muestra al presionar un botón, y en esta se mostrarán las tablas de nuestra base de datos mediante un botón; dependiendo del botón que presione el usuario, se mostrará una tabla u otra. En nuestra implementación, de momento no muestra los datos, ya que es una versión temprana, pero si muestra una tabla de ejemplo a la hora de presionar el botón.

A continuación, se muestran unas capturas de la implementación del código y el resultado de la implementación de la interfaz gráfica, faltan añadir botones y acciones para los botones; pero como ya se mencionó, la página principal y una segunda página ya funcionan correctamente, aunque no estén finalizadas del todo.

Página Principal:



Código Página Principal:

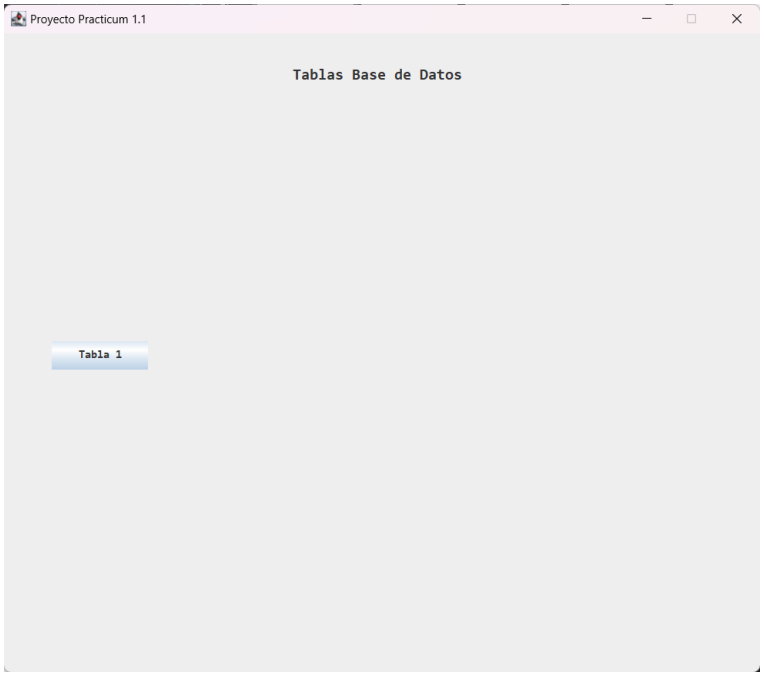
```scala
def inicio =
  val ventana = new JFrame("Proyecto Practicum 1.1")
  ventana.setSize(800, 700)
  ventana.setResizable(false)
  ventana.setDefaultCloseOperation(WindowConstants.EXIT_ON_CLOSE)
  ventana.setLocationRelativeTo(null)

  val paginas = new CardLayout
  val panel = new JPanel(paginas)
  val contenedor: Container = ventana.getContentPane

  val pagPrincipal = new JLabel
  val image = new ImageIcon("src/main/scala/bimestre2/imagenes/imagen1.png")
  pagPrincipal.setIcon(new ImageIcon(image.getImage.getScaledInstance(
    ventana.getWidth, ventana.getHeight, Image.SCALE_REPLICATE)))
  pagPrincipal.setHorizontalAlignment(SwingConstants.CENTER)
  pagPrincipal.setVerticalTextPosition(SwingConstants.CENTER)

  val button1 = new JButton("Tablas")
  button1.setBounds(550, 315, 150, 30) // Ajusta la posición y el tamaño del botón
  button1.setBackground(Color.BLACK) // Establece el color de fondo del botón
  button1.setForeground(Color.WHITE) // Establece el color del texto del botón
  button1.setOpaque(true) // Hace que el botón sea opaco
  button1.setBorderPainted(false) // Quita el borde pintado del botón
  button1.setFocusPainted(false) // Quita el borde de enfoque del botón
  button1.setFont(new Font("Consolas", Font.BOLD, 12))
  pagPrincipal.add(button1)

  button1.addActionListener(new ActionListener() {
    def actionPerformed(e: ActionEvent): Unit = {
      pagina1(panel, paginas)
    }
  })
```

El JFrame es la venta en como tal, está no cambia en ningún momento y se mantiene con sus características iniciales. Después hay un CardLayaout, que nos permitirá ir cambiando de página dependiendo del botón que se presione. Las páginas se crean dentro de un JPanel que nos permite almacenar los JLabel que muestran el contenido en si de la página, estos se añaden al JPanel junto con una etiqueta que nos permitirá después ir cambiando de página gracias al CardLayout.

También para los botones hay que añadir una acción, esto se logra gracias a las librería de java, awt.event, la cual nos permite implementar una función donde definimos la acción que realizará el botón, en este caso es la siguiente:

```
def pagina1(p: JPanel, c: CardLayout) =
  val pag1 = new JLabel

  val button = new JButton("Tabla 1")
  button.setBounds(50, 320, 100, 30)
  button.setBorderPainted(false) // Quita el borde pintado del botón
  button.setFocusPainted(false) // Quita el borde de enfoque del botón
  button.setFont(new Font("Consolas", Font.BOLD, 12))
  pag1.add(button)

  val tablas = new CardLayout
  val panelTabla = new JPanel(tablas)
  panelTabla.setBounds(200, 90, 550, 560)
  pag1.add(panelTabla)

  button.addActionListener(new ActionListener() {
    def actionPerformed(e: ActionEvent): Unit = {
      tabla1(panelTabla, tablas)
    }
  })

  val texto = new JLabel("Tablas Base de Datos")
  texto.setBounds(300, 30, 200, 30)
  texto.setFont(new Font("Consolas", Font.BOLD, 16))
  pag1.add(texto)

  p.add(pag1, "1")
  c.show(p, "1")
```

Resultado:



La cual muestra el contenido de otra página totalmente diferente a la principal. Dentro de esta página realizamos el mismo proceso, pero esta vez solo para un espacio dentro de la página, para que en ese espacio se muestren las tablas dependiendo del botón que se presione.

En este caso solo hay un botón para mostrar una tabla vacía a modo de ejemplo.

```
100   def tabla1(p: JPanel, c: CardLayout) =
101     val espacioTabla = new JLabel
102
103     val nombreTabla = new JLabel("Tabla 1")
104     nombreTabla.setBounds(250, 5, 100, 30)
105     nombreTabla.setFont(new Font("Consolas", Font.BOLD, 12))
106     espacioTabla.add(nombreTabla)
107
108     val table = new JTable(50, 10)
109     val scroll = new JScrollPane(table)
110     scroll.setBounds(0, 40, 550, 540)
111     espacioTabla.add(scroll)
112
113     p.add(espacioTabla, "1")
114     c.show(p, "1")
```

Resultado: