**UNIVERSIDAD TÉCNICA PARTICULAR DE LOJA**

Logotipo

Descripción generada automáticamente

**Materia:**

Practicum 1.1

**Tema:**

“ Proyecto Integrador

**Autores:**

• Morales Bautista Luis Miguel

• Riofrio Jaramillo Santiago Ismael

**Docente:**

• Ing. Ruiz Vivanco Omar Alexander

**Fecha:**

Octubre 2023 – Febrero 2023

**Introducción**

Este proyecto tuvo como objetivo brindar una experiencia teórica y práctica al unir las asignaturas de Fundamentos de Base de Datos y Programación Funcional y Reactiva, ofreciéndonos un reto en el cual ponemos en practica los conocimientos adquiridos a través del transcurso de las asignaturas.

**Apartado enfocado en el manejo de datos**

1. **Análisis de los Datos y Modelo Conceptual**

**Análisis de los Datos**

Como punto de partida inicial, se comenzó explorando los archivos CSV proporcionados para entender la naturaleza de los datos, identificando los elementos clave como entidades, atributos y posibles relaciones.

Para la exploración de los archivos se hizo uso del sistema de gestión de bases de datos (DBMS) denominado MySQL Workbench para importar los archivos CSV y a su vez se trabajó con en el entorno de desarrollo DataGrip, el cual nos permitirá ejecutar sentencias SQL para poder así analizar de mejor manera los datasets y sus datos correspondientes.

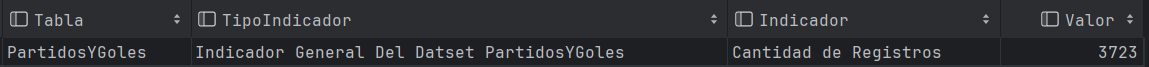
A continuación, se presentan las sentencias usadas con su respectiva salida generada, las cuales nos permitieron, de manera eficaz, obtener más información acerca de los datasets con los que se trabajo.

**Dataset: PartidosYGoles (dspartidosygoles)**

Sentencia SQL:

# Numero de Registros  
SELECT  
 'PartidosYGoles' AS Tabla,  
 'Indicador General Del Datset PartidosYGoles' AS TipoIndicador,  
 'Cantidad de Registros' AS Indicador,  
 *COUNT*(\*) AS Valor  
FROM  
 dspartidosygoles;

Salida:



Sentencia SQL:

# Exploracion de Nulls

# Minute Regulation  
SELECT DISTINCT  
 goals\_minute\_regulation AS "Minute Regulation"  
FROM  
 dspartidosygoles  
ORDER BY  
 1;

# Playe Id

SELECT DISTINCT  
 goals\_player\_id AS "Player Id"  
FROM  
 dspartidosygoles  
ORDER BY  
 1;

Salidas:

Tabla

Descripción generada automáticamente con confianza baja Interfaz de usuario gráfica, Aplicación

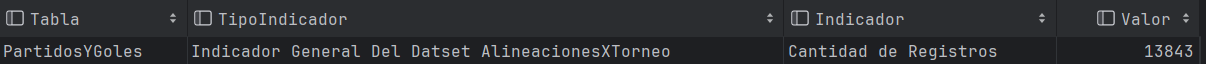
Descripción generada automáticamente

**Dataset: AlineacionesXTorneo (alineacionesxtorneo-2)**

Sentencias:

# Numero de Registros  
SELECT  
 'PartidosYGoles' AS Tabla,  
 'Indicador General Del Datset AlineacionesXTorneo' AS TipoIndicador,  
 'Cantidad de Registros' AS Indicador,  
 *COUNT*(\*) AS Valor  
FROM  
 `dsalineacionesxtorneo-2` ;

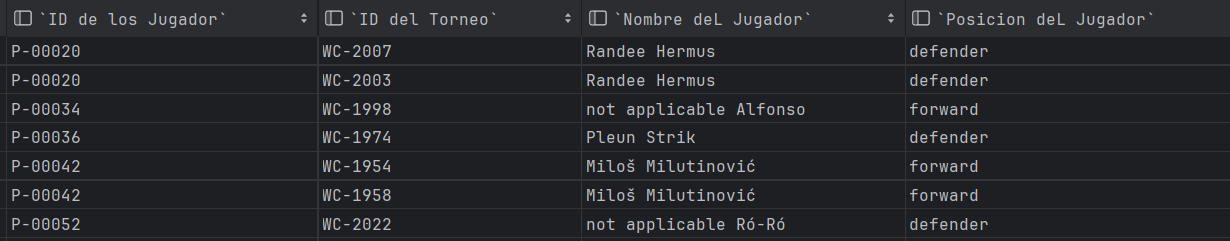
Salida:



Sentencias:

# Identifiacion de Relaciones entre Tablas  
  
SELECT  
  
 squads\_player\_id AS 'ID de los Jugador',  
 squads\_tournament\_id AS 'ID del Torneo',  
 *CONCAT*(players\_given\_name, ' ', players\_family\_name) AS 'Nombre deL Jugador',  
 squads\_position\_name'Posicion deL Jugador'  
  
FROM  
 `dsalineacionesxtorneo-2`  
  
ORDER BY  
 squads\_player\_id;

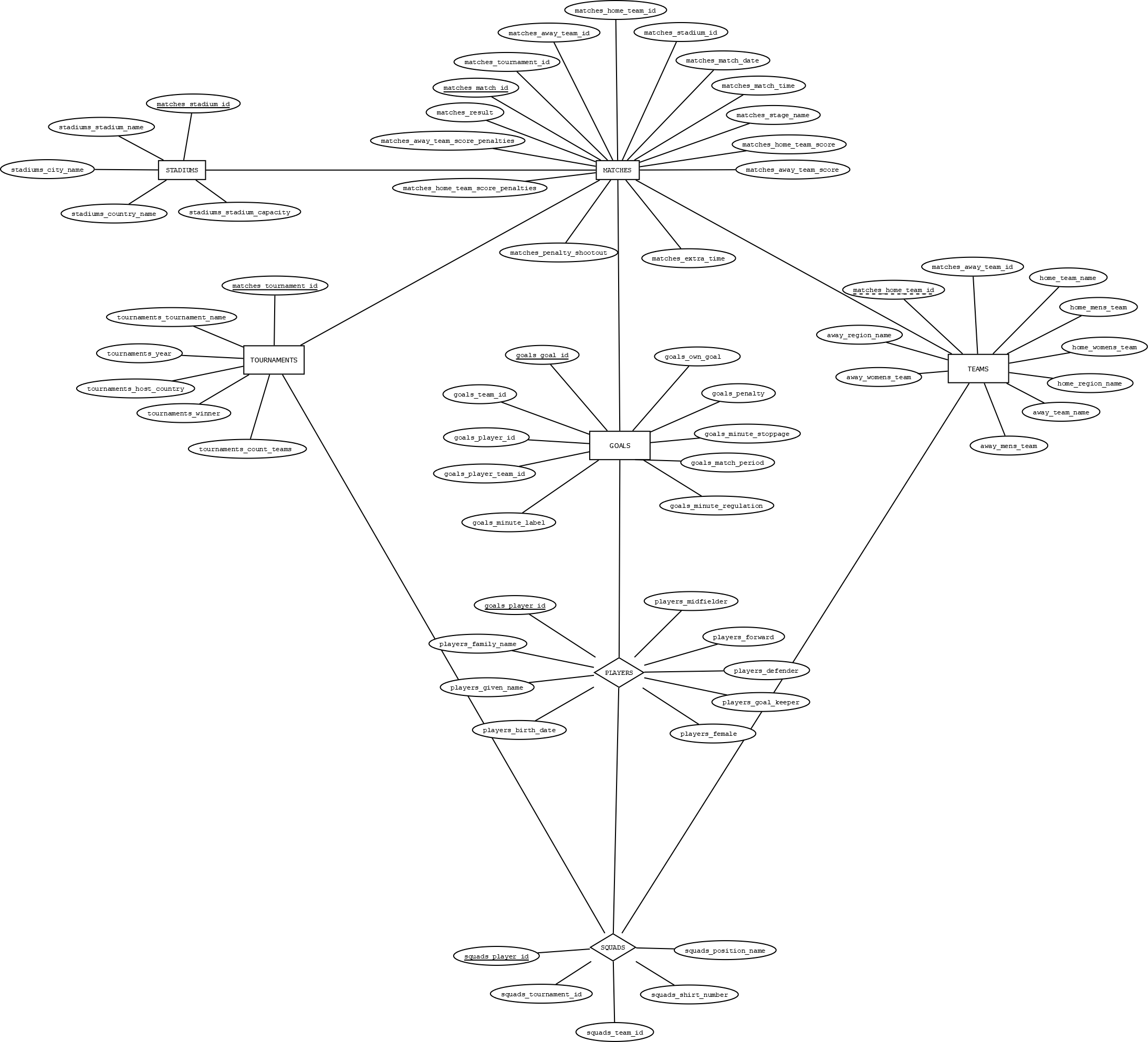
Salida:



**Modelo Conceptual Entidad-Relación (E/R)**

Para la diagramación del modelo (E/R) se optó por utilizar una herramienta que nos permita representar las entidades identificadas, sus atributos, y las relaciones entre ellas. En este caso se utilizó la herramienta DIA para capturar la estructura lógica de los datos sin preocuparnos por detalles de implementación.

A continuación, se presenta el diagrama (E/R) hecho en la herramienta mencionada.



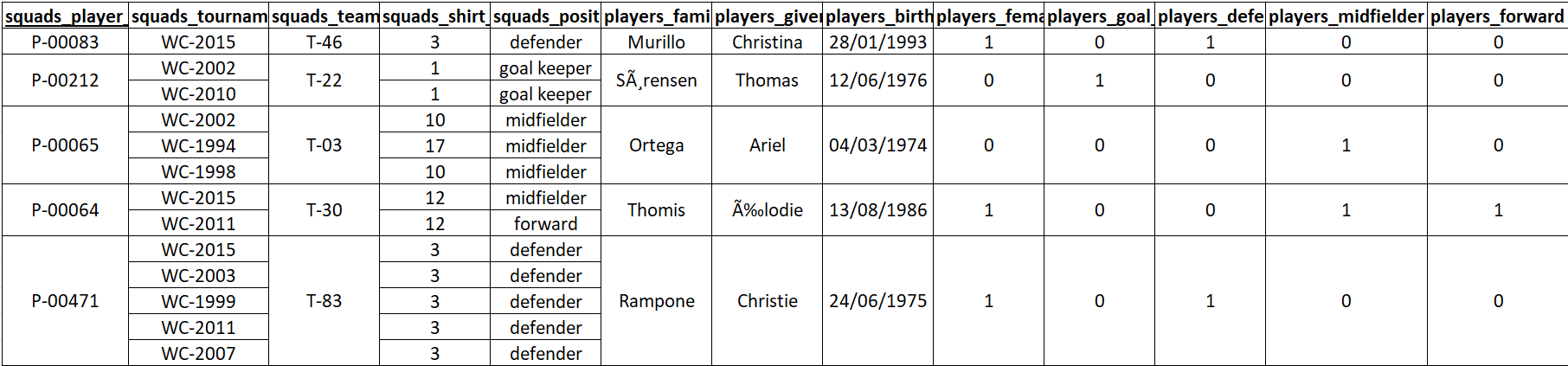
1. **Normalización**

Se analizaron los datos para identificar dependencias funcionales y transitivas entre atributos, lo que ayudó a entender las relaciones de unicidad y dependencia. Se aplicaron las reglas de normalización (hasta la tercera forma normal) para asegurar que la base de datos estuviera libre de redundancias innecesarias y anomalías de inserción, actualización o eliminación.

**Primer Forma Normal (Datos Atómicos / Identificación Datos Repetitivos)**

En esta primera forma de normalización se busca que los datos de cada atributo del dataset sean atómicos, además de analizarse los datos repetitivos; para esto usaremos la herramienta Excel que nos ayudara a representar de mejor manera dichos datos.

**Dataset: AlineacionesXTorneo (alineacionesxtorneo-2)**

****

**Dataset: PartidosYGoles (dspartidosygoles)**

**Imagen que contiene edificio, biombo

Descripción generada automáticamente**

****

**Imagen que contiene Tabla

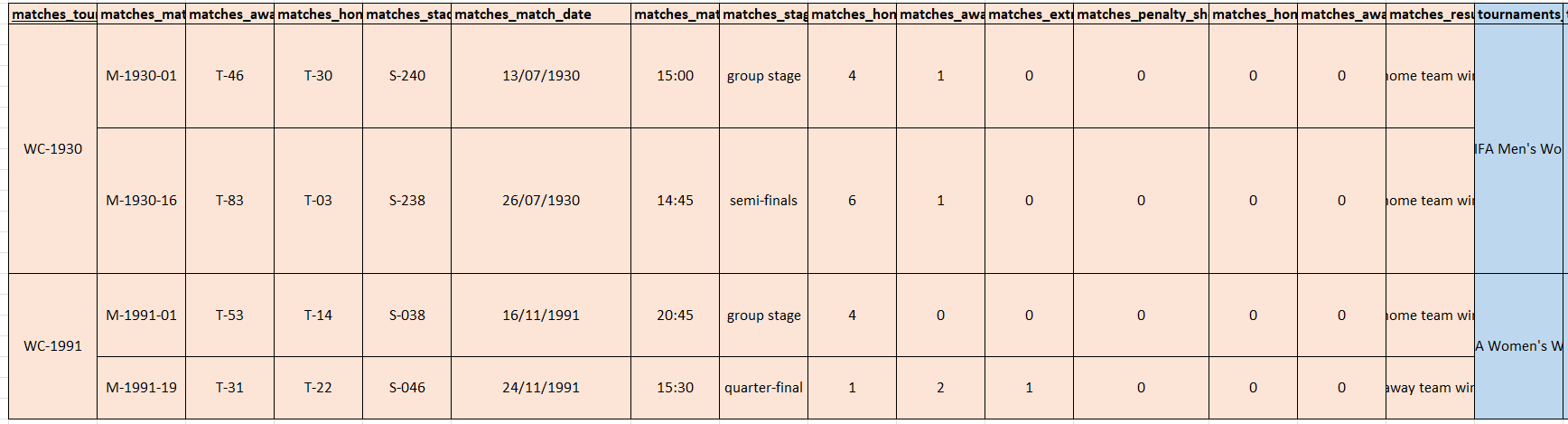
Descripción generada automáticamente**

**Segunda Forma Normal (Dependencias Funcionales)**

En esta segunda forma de normalización se busca ubicar las dependencias funcionales que existen en cada tabla, así de esta forma poder identificar las posibles relaciones que existen entre los datasets.

**Tabla

Descripción generada automáticamenteDataset: AlineacionesXTorneo (alineacionesxtorneo-2)**

**Dataset: PartidosYGoles (dspartidosygoles)**

****

****

**Tercera Forma Normal (Dependencias Transitivas)**

En esta tercera forma de normalización se busca emplear las dependencias transitivas que existen en cada tabla, así de esta forma poder ejemplificar de mejor manera las tablas así también como las relaciones existen dentro de los datasets.

**Tabla

Descripción generada automáticamenteDataset: AlineacionesXTorneo (alineacionesxtorneo-2)**

****

**Dataset: PartidosYGoles (dspartidosygoles)**

**Gráfico, Tabla

Descripción generada automáticamente**

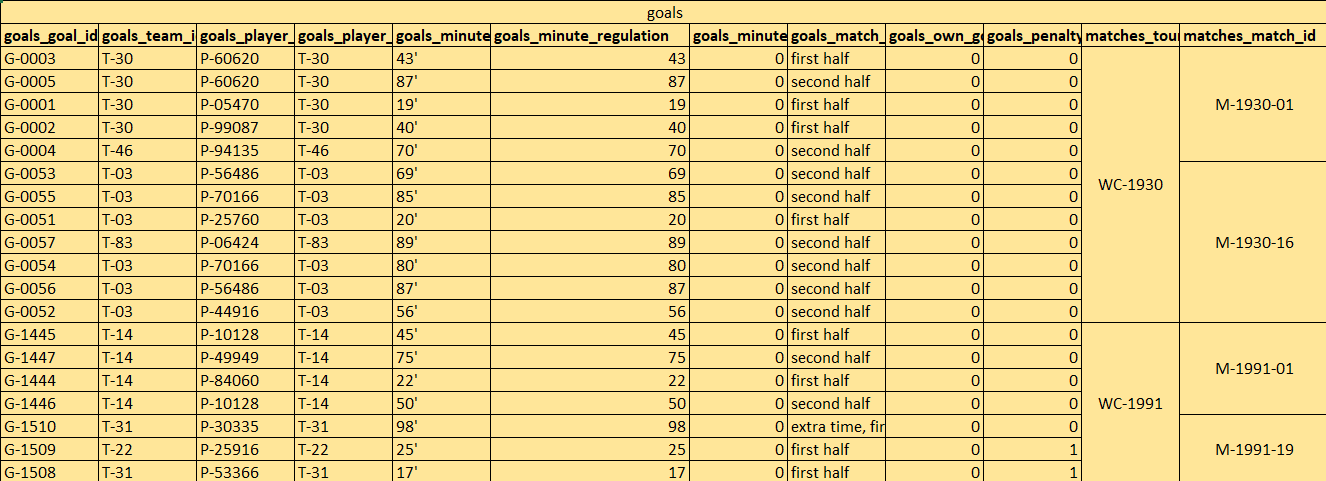
**Calendario

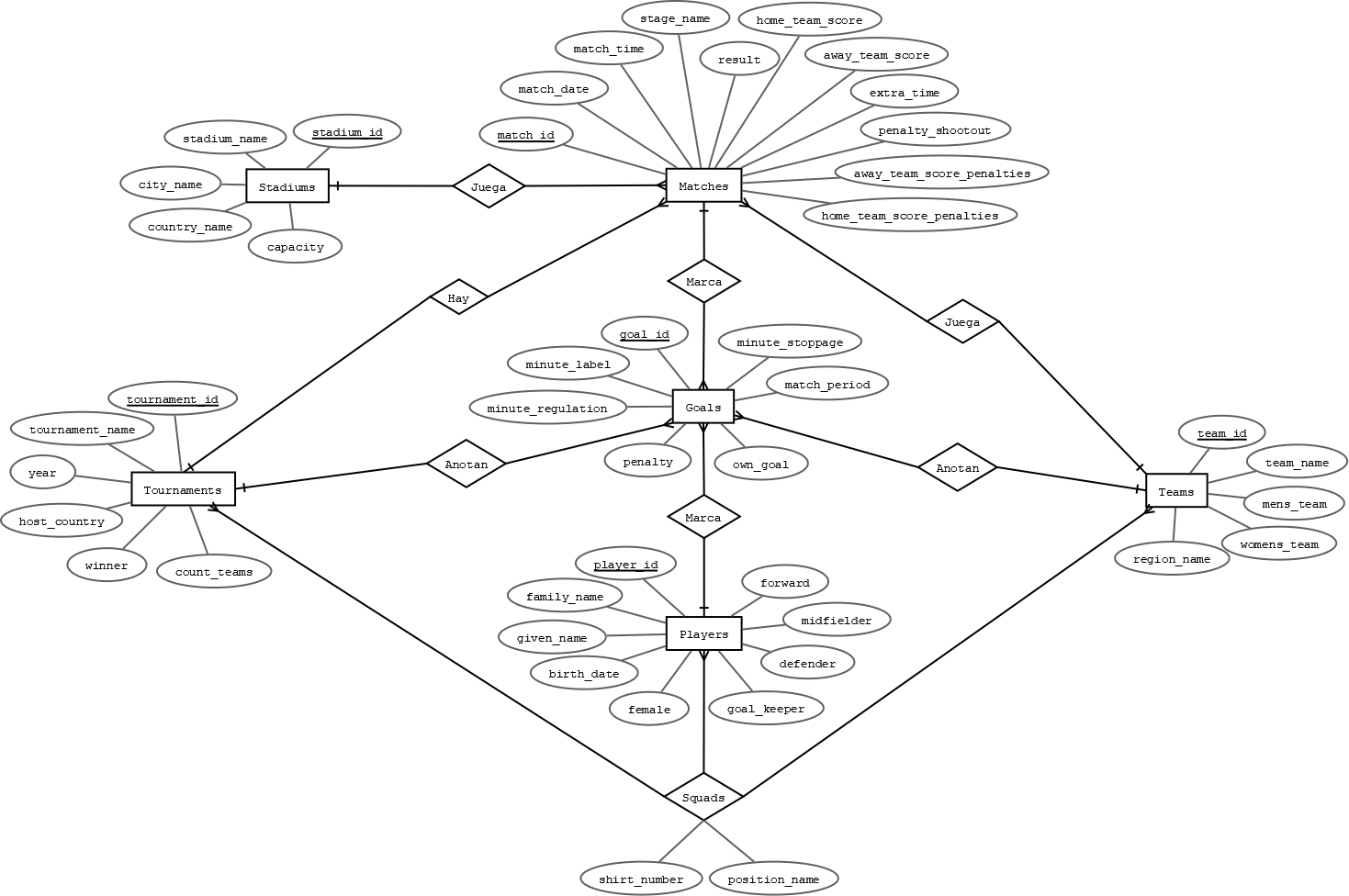
Descripción generada automáticamente**

****

**Tabla

Descripción generada automáticamente**

****

**Modelo Entidad Relación Normalizado**

**Justificación Modelo Conceptual (Normalizado)**

Después de haber aplicado las técnicas de normalización se opto por mejorar el modelo (E/R) optando por descartar atributos que resultaban redundantes dentro del modelo inicial, así también como cambiar los nombres de los atributos, esto debido a que sus etiquetas anteriores resultaban confusas y podrían haber errores tanto en el modelo de datos como al momento de construir un modelo físico.

1. **Modelo Relacional**

Para el siguiente punto se convirtió el modelo conceptual (E/R) en un modelo relacional.

Esto implicó definir tablas, relaciones para cada entidad y relación, estableciendo claves primarias y foráneas para mantener la integridad referencial.

Para cada tabla identificada, se definió un esquema detallado que incluyó nombres de columnas, tipos de datos y restricciones; como claves primarias y foráneas, condiciones, restricciones de unicidad, etc.

Al igual que el modelo anterior, la herramienta que se usó para la esquematizar dicho modelo es el programa de DIA, ya que también nos ofrece las herramientas apropiadas para construir este modelo.

**Diagrama

Descripción generada automáticamente**Modelo Relacional

1. **Implementación del Modelo Físico**

Para la implementación del modelo físico, se creó la base de datos utilizando un sistema de gestión de bases de datos DBMS (MySQL Workbench), según el modelo relacional diseñado. Para un trabajo más eficiente se utilizó el entorno de desarrollo DataGrip en el cual se desarrollaron scripts y se utilizaron herramientas para importar los datos de los archivos CSV a las tablas correspondientes de la base de datos, asegurando que los datos se ajustaran a las restricciones definidas en el esquema.

A continuación, se presentan las sentencias DDL utilizadas para la creación del modelo físico

**Script SQL:**

# Creacion de la Tabla Stadiums  
DROP TABLE IF EXISTS Stadiums;  
CREATE TABLE  
 Stadiums  
SELECT DISTINCT  
 matches\_stadium\_id AS stadium\_id,  
 stadiums\_stadium\_name AS stadium\_name,  
 stadiums\_city\_name AS city\_name,  
 stadiums\_country\_name AS country\_name,  
 stadiums\_stadium\_capacity AS stadium\_capacity  
FROM  
 dspartidosygoles;  
  
ALTER TABLE Stadiums  
ADD PRIMARY KEY (stadium\_id);  
  
  
 # Creacion de la Tabla Teams  
DROP TABLE IF EXISTS Teams;  
CREATE TABLE  
 Teams  
SELECT DISTINCT  
 matches\_away\_team\_id AS team\_id,  
 away\_team\_name AS team\_name,  
 away\_mens\_team AS mens\_team,  
 away\_womens\_team AS womens\_team,  
 away\_region\_name AS region\_name  
  
FROM  
 dspartidosygoles;  
  
ALTER TABLE Teams  
ADD PRIMARY KEY (team\_id);

# Creacion de la Tabla Tournaments  
DROP TABLE IF EXISTS Tournaments;  
CREATE TABLE  
 Tournaments  
SELECT DISTINCT  
 matches\_tournament\_id AS tournament\_id,  
 tournaments\_tournament\_name AS tournament\_name,  
 tournaments\_year,  
 tournaments\_host\_country AS host\_country,  
 tournaments\_winner AS winner,  
 tournaments\_count\_teams AS count\_teams  
FROM  
 dspartidosygoles;  
  
ALTER TABLE Tournaments  
ADD PRIMARY KEY (tournament\_id);

# Creacion de la Tabla Players  
DROP TABLE IF EXISTS Players;  
CREATE TABLE  
 Players  
SELECT DISTINCT  
 squads\_player\_id AS player\_id,  
 players\_family\_name AS family\_name,  
 players\_given\_name AS given\_name,  
 players\_birth\_date AS birth\_date,  
 players\_female AS female,  
 players\_goal\_keeper AS goal\_keeper,  
 players\_defender AS defender,  
 players\_midfielder AS midfielder,  
 players\_forward AS forward  
FROM  
 `dsalineacionesxtorneo-2`;  
  
ALTER TABLE Players  
ADD PRIMARY KEY (player\_id);

# Creacion de la Tabla Goals  
DROP TABLE IF EXISTS Goals;  
CREATE TABLE  
 Goals  
SELECT DISTINCT  
 goals\_goal\_id AS goal\_id,  
 goals\_team\_id AS team\_id,  
 goals\_player\_id AS player\_id,  
 goals\_player\_team\_id AS player\_team\_id,  
 matches\_tournament\_id AS tournament\_id,  
 goals\_minute\_label AS minute\_label,  
 goals\_minute\_regulation AS minute\_regulation,  
 goals\_minute\_stoppage AS minute\_stoppage,  
 goals\_match\_period AS match\_period,  
 goals\_own\_goal AS own\_goal,  
 goals\_penalty AS penalty  
FROM  
 dspartidosygoles  
  
WHERE dspartidosygoles.goals\_minute\_regulation IS NOT NULL;  
  
ALTER TABLE Goals  
ADD PRIMARY KEY (goal\_id),  
ADD CONSTRAINT team\_id\_1 FOREIGN KEY (team\_id) REFERENCES Teams(team\_id),  
ADD FOREIGN KEY (player\_id) REFERENCES Players(player\_id),  
ADD CONSTRAINT team\_id\_2 FOREIGN KEY (player\_team\_id) REFERENCES Teams(team\_id),  
ADD FOREIGN KEY (tournament\_id) REFERENCES Tournaments(tournament\_id);

# Creacion de la Tabla Matches  
DROP TABLE IF EXISTS Matches;  
CREATE TABLE  
 Matches  
SELECT DISTINCT  
 matches\_match\_id AS match\_id,  
 matches\_tournament\_id AS tournament\_id,  
 matches\_away\_team\_id AS away\_team\_id,  
 matches\_home\_team\_id AS home\_team\_id,  
 matches\_stadium\_id AS stadium\_id,  
 matches\_match\_date AS match\_date,  
 matches\_match\_time AS match\_time,  
 matches\_stage\_name AS stage\_name,  
 matches\_home\_team\_score AS home\_team\_score,  
 matches\_away\_team\_score AS away\_team\_score,  
 matches\_extra\_time AS extra\_time,  
 matches\_penalty\_shootout AS penalty\_shootout,  
 matches\_home\_team\_score\_penalties AS home\_team\_score\_penalties,  
 matches\_away\_team\_score\_penalties AS away\_team\_score\_penalties,  
 matches\_result AS result  
FROM  
 dspartidosygoles;  
  
ALTER TABLE Matches  
ADD PRIMARY KEY (match\_id),  
ADD FOREIGN KEY (tournament\_id) REFERENCES Tournaments(tournament\_id),  
ADD CONSTRAINT teamid\_1 FOREIGN KEY (away\_team\_id) REFERENCES Teams(team\_id),  
ADD CONSTRAINT teamid\_2 FOREIGN KEY (home\_team\_id) REFERENCES Teams(team\_id),  
ADD FOREIGN KEY (stadium\_id) REFERENCES Stadiums(stadium\_id);

# Creacion de la Tabla SQUADS  
DROP TABLE IF EXISTS Squads;  
CREATE TABLE  
 Squads  
SELECT DISTINCT  
 squads\_player\_id AS player\_id,  
 squads\_tournament\_id AS tournament\_id,  
 squads\_team\_id AS team\_id,  
 squads\_shirt\_number AS shirt\_number,  
 squads\_position\_name AS position\_name  
FROM  
 `dsalineacionesxtorneo-2`;  
  
ALTER TABLE Squads  
ADD FOREIGN KEY (player\_id) REFERENCES Players(player\_id),  
ADD FOREIGN KEY (tournament\_id) REFERENCES Tournaments(tournament\_id),  
ADD FOREIGN KEY (team\_id) REFERENCES Teams(team\_id);

**Modelo Fisico**

Como punto final del modelo físico y después de haber aplicado cada uno de los puntos anteriores a nuestra base de datos, se mostrará el diseño obtenido a través de la herramienta DataGrip la cual nos proporciona una funcionalidad para modelar nuestro diseño y así comprobar cualquier falencia en alguna relación entre tablas que se hayan podido ocasionar debido a errores de sintaxis, errores lógicos o de ejecución al haber creado nuestras sentencias.

**Diseño Físico**

Imagen que contiene Diagrama

Descripción generada automáticamente

**Apartado del manejo de programación**

**Estadísticas del Dataset**

En este proyecto, previo a la construcción del programa, previamente se analiza los datasets proporcionados para conocer de mejor manera los datos con los que se va a trabajar.

A continuación, se proporciona el proceso mediante el cual se obtuvieron las estadísticas generales extraídas de cada uno de los datasets.

**Proceso:**

Para la obtención de las estadísticas generales de cada uno de los datasets, se empleó un programa el cual nos permitirá la lectura de los datasets (con extensión .csv) de tal manera que podamos manipular los datos mediante sentencias de código y así poder las estadísticas requeridas. Para este programa se hizo uso del lenguaje de programación Scala en la versión 3, con ayuda del entorno virtual IntelliJ IDEA ULTIMATE

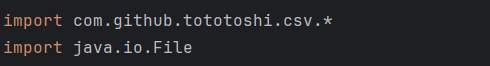
**Dependencias Implementadas:**

Previo a la implementación del código, se utilizaron algunas dependencias, como son:



**Librerías Implementadas:**

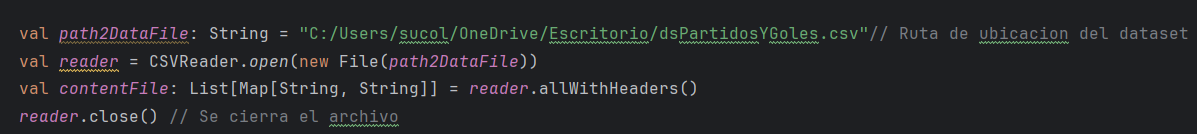
Que nos permitirá hacer uso de la librería la cual nos ayudará a realizar la lectura de los datasets.



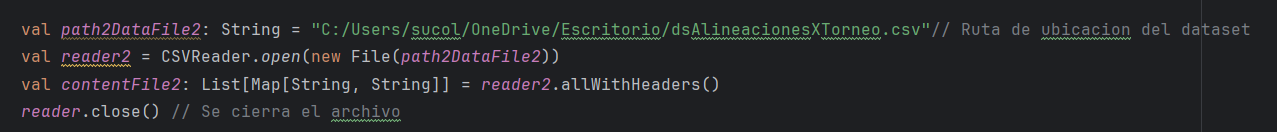
**Código Lectura de Archivos:**

A continuación, se muestra la implementación del código para la lectura de los dataset.

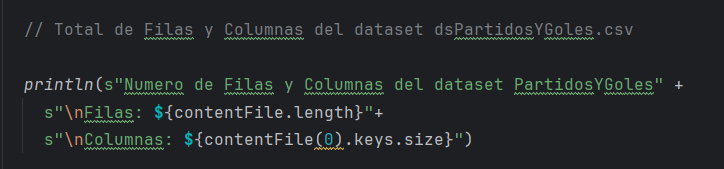
**PartidoYGoles:**

****

**AlineacionesXTorneo:**

****

**Estadísticas del Dataset**

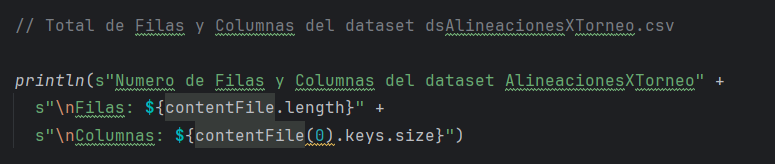
****

**Estadísticas del dataset PartidoYGoles:**

**Texto

Descripción generada automáticamente**

**Codigo para las Estadísticas del dataset AlineacionesXTorneo:**

****

**Estadísticas del dataset AlineacionesXTorneo:**

**Texto

Descripción generada automáticamente**

**Carga de Datos**

Se implementaron dos técnicas para la carga de datos en la base de datos:

**Generación de scripts:**

Se crearon scripts SQL para insertar los datos en la base de datos. Estos scripts se generaron a partir de los archivos CSV y se ejecutaron en el DBMS para cargar los datos.

**Código para generar carga de datos en la tabla goals**

generateDataGoals(*contentFile*)  
  
def generateDataGoals(data: List[Map[String, String]]) =  
 val sqlInsert = s"INSERT INTO goals(goal\_id, team\_id, player\_id, player\_team\_id, tournament\_id, minute\_label, minute\_regulation, minute\_stoppage, match\_period, own\_goal, penalty) VALUES('%s', '%s', '%s', '%s', '%s', '%s', %d, %d, '%s', %d, %d);"  
 val goalsTuple = data  
 .map(  
 row => (row("goals\_goal\_id").trim,  
 defaultValueString(row("goals\_team\_id")),  
 defaultValueString(row("goals\_player\_id")),  
 defaultValueString(row("goals\_player\_team\_id")),  
 defaultValueString(row("matches\_tournament\_id")),  
 defaultValueString(row("goals\_minute\_label")),  
 defaultValueInt(row("goals\_minute\_regulation")).toInt,  
 defaultValueInt(row("goals\_minute\_stoppage")).toInt,  
 defaultValueString(row("goals\_match\_period")),  
 defaultValueInt(row("goals\_own\_goal")).toInt,  
 defaultValueInt(row("goals\_penalty")).toInt  
 )  
 ).distinct  
 .map(t11 => sqlInsert.formatLocal(java.util.Locale.*US*, t11.\_1, t11.\_2, t11.\_3, t11.\_4, t11.\_5, t11.\_6, t11.\_7, t11.\_8, t11.\_9, t11.\_10, t11.\_11))  
 goalsTuple.foreach(*println*)

**Código para generar carga de datos en la tabla matches**

generateDataMatches(*contentFile*)  
def generateDataMatches(data: List[Map[String, String]]) =  
 val sqlInsert = s"INSERT INTO matches(match\_id, tournament\_id, away\_team\_id, home\_team\_id, stadium\_id, match\_date, match\_time, stage\_name, home\_team\_score, away\_team\_score, extra\_time, penalty\_shootout, home\_team\_score\_penalties, away\_team\_score\_penalties, result) VALUES('%s', '%s', '%s', '%s', '%s', '%s', '%s', '%s', %d, %d, %d, %d, %d, %d, '%s');"  
 val matchesTuple = data  
 .map(  
 row => (row("matches\_match\_id").trim,  
 defaultValueString(row("matches\_tournament\_id")),  
 defaultValueString(row("matches\_away\_team\_id")),  
 defaultValueString(row("matches\_home\_team\_id")),  
 defaultValueString(row("matches\_stadium\_id")),  
 defaultValueString(row("matches\_match\_date")),  
 defaultValueString(row("matches\_match\_time")),  
 defaultValueString(row("matches\_stage\_name")),  
 defaultValueInt(row("matches\_home\_team\_score")).toInt,  
 defaultValueInt(row("matches\_away\_team\_score")).toInt,  
 defaultValueInt(row("matches\_extra\_time")).toInt,  
 defaultValueInt(row("matches\_penalty\_shootout")).toInt,  
 defaultValueInt(row("matches\_home\_team\_score\_penalties")).toInt,  
 defaultValueInt(row("matches\_away\_team\_score\_penalties")).toInt,  
 defaultValueString(row("matches\_result"))  
 )  
 ).distinct  
 .map(t15 => sqlInsert.formatLocal(java.util.Locale.*US*, t15.\_1, t15.\_2, t15.\_3, t15.\_4, t15.\_5, t15.\_6, t15.\_7, t15.\_8, t15.\_9, t15.\_10, t15.\_11, t15.\_12, t15.\_13, t15.\_14, t15.\_15))  
 matchesTuple.foreach(*println*)

**Manipulación directa de la base de datos**

Se utilizó la interfaz del DBMS para insertar datos directamente en la base de datos. Este método permitió una mayor flexibilidad y control sobre los datos insertados.

**Código para generar carga de datos directamentente en la tabla matches**

generateDataManipulateTournaments(contentFile).foreach(insert => insert.run.transact(xa).unsafeRunSync())  
  
def generateDataManipulateStadium(data: List[Map[String, String]]) =  
 val stadiumTuple = data  
 .map(  
 row => (row("matches\_stadium\_id").trim,  
 defaultValueString(row("stadiums\_stadium\_name")),  
 defaultValueString(row("stadiums\_city\_name")),  
 defaultValueString(row("stadiums\_country\_name")),  
 defaultValueInt(row("stadiums\_stadium\_capacity")).toInt  
 )  
 ).distinct  
 .map(t5 => sql"INSERT INTO stadiums(stadium\_id, stadium\_name, city\_name, country\_name, stadium\_capacity) VALUES(**$**{t5.\_1}, **$**{t5.\_2}, **$**{t5.\_3}, **$**{t5.\_4}, **$**{t5.\_5})".update)  
  
 stadiumTuple

**Código para generar carga de datos directamentente en la tabla teams**

def generateDataManipulateTeams(data: List[Map[String, String]]) =  
 val teamsTuple = data  
 .map(  
 row => (row("matches\_away\_team\_id").trim,  
 defaultValueString(row("away\_team\_name")),  
 defaultValueInt(row("away\_mens\_team")).toInt,  
 defaultValueInt(row("away\_womens\_team")).toInt,  
 defaultValueString(row("away\_region\_name"))  
 )  
 ).distinct  
 .map(t5 => sql"INSERT INTO teams(team\_id, team\_name, mens\_team, womens\_team, region\_name) VALUES(**$**{t5.\_1}, **$**{t5.\_2}, **$**{t5.\_3}, **$**{t5.\_4}, **$**{t5.\_5})".update)  
  
 teamsTuple

**Código para generar carga de datos directamentente en la tabla tournaments**

def generateDataManipulateTournaments(data: List[Map[String, String]]) =  
 val tournamentsTuple = data  
 .map(  
 row => (row("matches\_tournament\_id").trim,  
 defaultValueString(row("tournaments\_tournament\_name")),  
 defaultValueInt(row("tournaments\_year")).toInt,  
 defaultValueString(row("tournaments\_host\_country")),  
 defaultValueString(row("tournaments\_winner")),  
 defaultValueInt(row("tournaments\_count\_teams")).toInt  
 )  
 ).distinct  
 .map(t6 => sql"INSERT INTO tournaments(tournament\_id, tournament\_name, tournaments\_year, host\_country, winner, count\_teams) VALUES(**$**{t6.\_1}, **$**{t6.\_2}, **$**{t6.\_3}, **$**{t6.\_4}, **$**{t6.\_5}, **$**{t6.\_6})".update)  
  
 tournamentsTuple

**Consultas SQL**

Se plantearon y respondieron varias preguntas utilizando sentencias SQL. Estas consultas incluyeron operaciones de inserción, actualización, eliminación y selección de datos. Algunas de las consultas también involucraron operaciones más avanzadas, como join, subconsultas y funciones de agregación.

**Consultas para obtener el promedio de capacidad de los estadios**

def promedioCapacidadEstadios() = {  
 sql"SELECT stadium\_name, AVG(stadium\_capacity) FROM stadiums group by 1"  
 .query[(String, Double)]  
 .to[List]  
 .transact(*xa*)  
 .unsafeRunSync()

**Consultas para obtener el número de estadios que tiene un país.**

def numEstadioPais() = {  
 sql"""select distinct city\_name, count(stadium\_name) from stadiums group by 1"""  
 .query[(String, Double)]  
 .to[List]  
 .transact(*xa*)  
 .unsafeRunSync()  
}

**Interfaz Grafica**

Se construyo una interfaz gráfica de usuario simple para que obtener la data de una mejor manera.

import tkinter as tk  
  
# Función para manejar el clic del botón  
def button\_click():  
label.config(text="Hola " + entry.get())  
  
# Crear una ventana  
root = tk.Tk()  
root.title("Ejemplo de Interfaz de Usuario")  
  
# Crear un widget de etiqueta  
label = tk.Label(root, text="Ingrese su nombre:")  
label.pack()  
  
# Crear un widget de entrada de texto  
entry = tk.Entry(root)  
entry.pack()  
  
# Crear un widget de botón  
button = tk.Button(root, text="Saludar", command=*button\_click*)  
button.pack()  
  
# Ejecutar el bucle de eventos  
root.mainloop()