# Curso SQL

# Proyecto Final

## Temática: Base de datos del Futbol Argentino

### 1. Descripción de la temática de la base de datos

La base de datos presentada tiene como objetivo almacenar datos sobre la estructura de la Primera División del Futbol Argentino. A través de esta, podremos visualizar todo tipo de información sobre los equipos y consultar estadísticas de cada uno de ellos. Actualmente, la primera división cuenta con 28 equipos los cuales disputaran distintos partidos durante la temporada 2023. Los mismos jugaran como mínimo dos torneos, los cuales son: *"Futbol argentino primera división"* y *"Copa Argentina"*. Los mejores equipos de la temporada 2022 disputaran otros torneos tales como: *"Copa sudamericana", "Copa libertadores"* y *"Copa Argentina"*. Por otro lado, tendremos información sobre partidos no disputados como también los ya disputados con toda la información de interés. Además, también contara con información sobre árbitros y jugadores contando con la eficiencia en su desempeño. La finalidad de esta tabla es poder estimar probabilidades de los equipos de ganar sus próximos partidos de acuerdo al desempeño del equipo y sus jugadores, el adversario, arbitro, cancha, entre otros.

### 2. Listado de las tablas que comprenden la base de datos

Esta base de datos contara con cinco (5) tablas las cuales estarán vinculadas entre sí a partir de ciertas relaciones dependiendo de los datos. En la siguiente figura se muestra un diagrama de entidad-relación con las tablas generadas, los datos a ingresar y sus relaciones. Los recuadros azules corresponden a los nombres de las tablas, los óvalos rosas claros y oscuros corresponden a los datos que se ingresaran a cada uno de las tablas, los últimos corresponden a las *primary key* de cada tabla y los rombos amarillos corresponden a las relaciones entre las tablas.

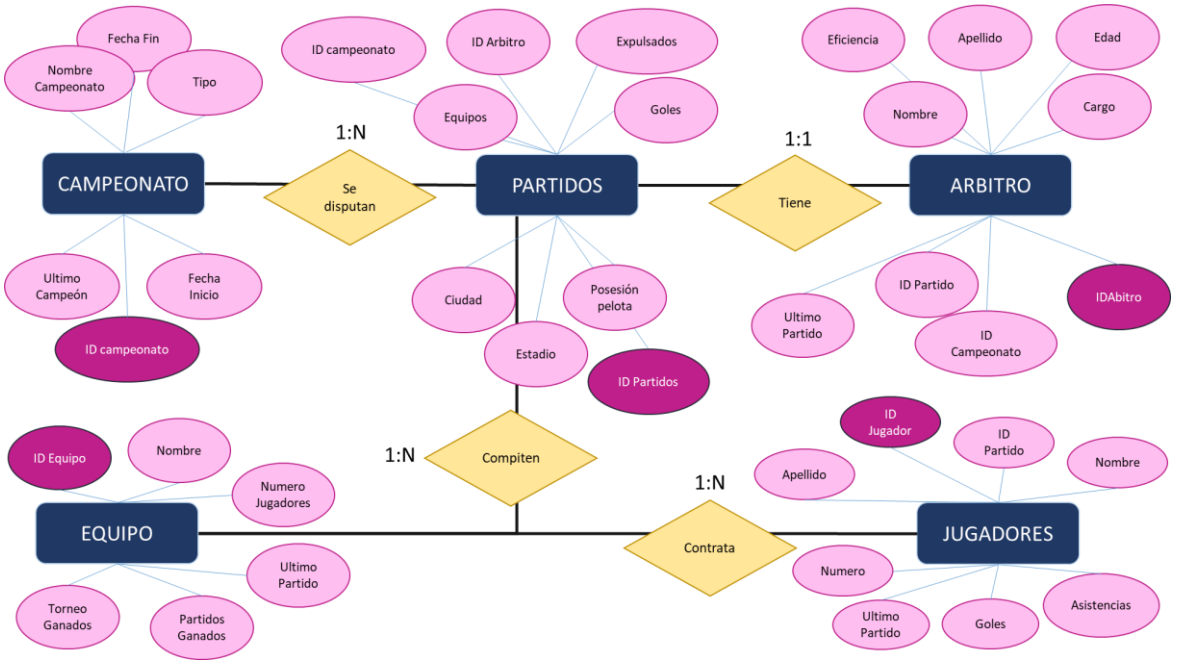

Fig.1 Diagrama Entidad- Relación

3. **Descripción de cada tabla, listado de campos, abreviaturas de nombres de campos, nombres completos de campos, tipos de datos, tipo de clave (foránea, primaria, índice(s))**
   a) *Campeonatos:* o internacional. Los posibles torneos son: Copa sudamericana, Copa libertadores, Copa Argentina, Futbol argentino primera división o Copa Argentina. esta tabla muestra los torneos que disputaran los distintos equipos. Estos pueden ser del tipo nacional. Esta tabla se vincula con la tabla de *partidos* a través de una relación de 1 a muchos, ya que dentro de un campeonato puede haber múltiples partidos. Esta tabla cuenta con una *primary key* llamada IDCampeonato la cual es un entero que se completara automáticamente (AUTO_INCREMENT)
   b) *Partidos:* en esta se visualizan las características de cada uno de los partidos a disputados y los próximos a disputarse. Los partidos ya disputados contaran con todas las estadísticas de ambos equipos mientras que aquellos que todavía no se jugaron solo completaran con datos de los equipos (local y visitantes), arbitro y sitio donde se jugaran. Por otro lado, es importante mencionar que la *pimary key* se llama *IDPartidos* la cual es un entero que se completara automáticamente (AUTO_INCREMENT). A su vez, cuenta con dos *foreign key* ya que se relaciona con la tabla equipos(*IDEquipos*) y árbitros (IDArbitros).

   c) *Árbitros*: en esta tabla se muestran los datos personales de los árbitros que participaran en cada uno de los partidos, así como también las estadísticas de su desempeño. Esta tabla se relaciona con partidos. Además, en cuanto a los campos que se relacionan con otras tablas podemos nombrar a la *pimary key* se cual se denomina *IDArbitros* la cual es un entero que se completara automáticamente (AUTO_INCREMENT). A su vez, cuenta con dos *foreign key* ya que se relaciona con la tabla Partidos(*IDPartidos*) y Campeonatos (IDCampeonatos).

   d) *Jugadores:* en esta se observan las características de cada uno de los equipos que disputaran los torneos. Entre la información presentada se muestra en que categoría se encuentra el equipo y las estadísticas del torneo que se encuentran disputando actualmente. Esta tabla se relaciona con equipo. En cuanto a la *pimary key* la nombramos como *IDJugadores* la cual es un entero que se completara automáticamente (AUTO_INCREMENT). Con respecto a la *Foreign key* podemos nombrar solo a la relacionada con la tabla *Equipos(IDEquipos).*

   e) *Equipos:* provee los datos de los jugadores de cada equipo, así como también sus estadísticas de juego. Esta tabla se relaciona con partidos. Por último, la *pimary key* se llama de esta tabla se llama *IDEquipos* la cual es un entero que se completara automáticamente (AUTO_INCREMENT).

En la Fig. 2 se muestran las tablas, y los tipos de datos que se ingresaran a cada una de estas

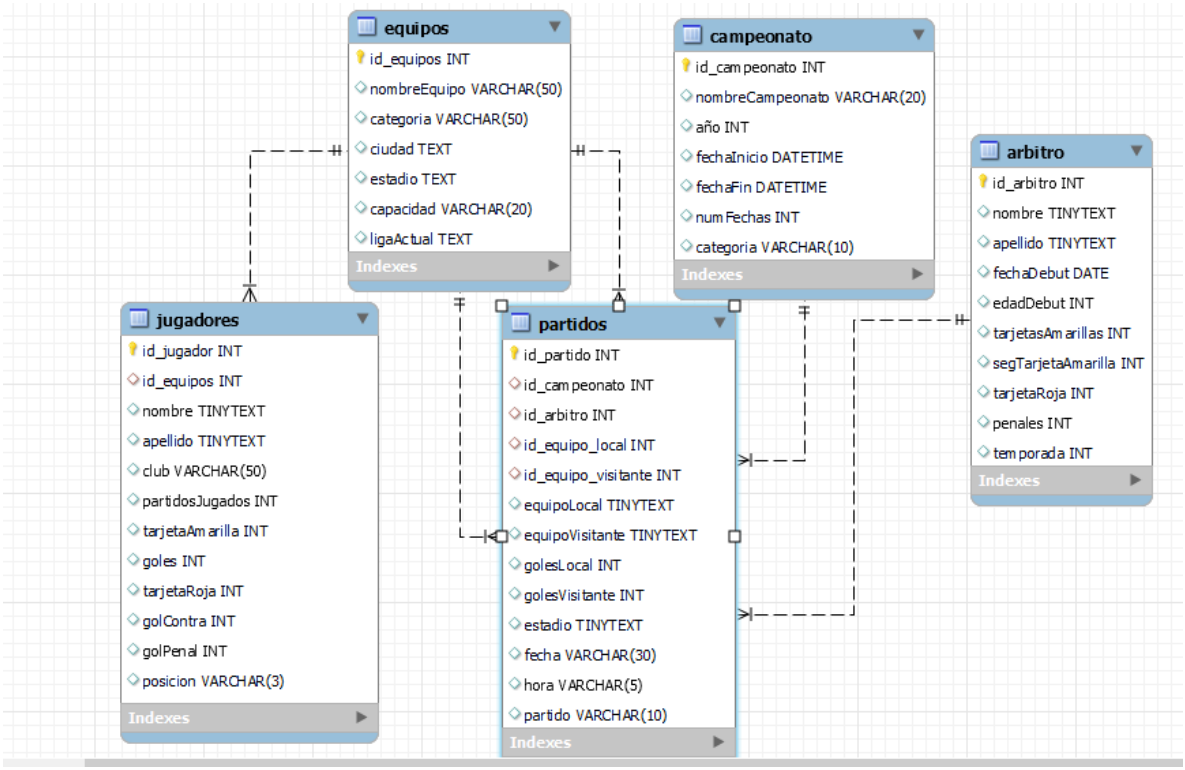


Fig.2 Diagrama Entidad- Relación

## 4. Creación de tablas en .sql

A continuación, se muestra el código realizado en MYSQL sobre la temática nombrada. En primer lugar, se procedió a crear la base de datos (schema) llamada: *futbol_argentino* luego se crearon cada una de las tablas nombradas anteriormente detallando los tipos de datos y cuales son las variables que se corresponden a las *primary keys* y *foreign keys*

```
-- Creacion del SCHEMA
CREATE SCHEMA IF NOT EXISTS futbol_argentino;

-- Usamos el schema de interes
use futbol_argentino;

-- 01 Creacion tabla CAMPEONATO
CREATE TABLE campeonato(
   id_campeonato INT NOT NULL AUTO_INCREMENT,
   nombreCampeonato VARCHAR(20),
   año INT,
   fechaInicio DATETIME,
   fechaFin DATETIME,
   ultimoCampeon TEXT(10),
   tipo TEXT(10),
   numFechas INT,
   categoria VARCHAR(10),
   PRIMARY KEY(id_campeonato));
```

```
-- 02 Creacion tabla ARBITRO
CREATE table arbitro (
   id_arbitro INT NOT NULL AUTO_INCREMENT,
   -- id_campeonato INT NOT NULL,
   nombre TEXT (20),
   apellido TEXT(20),
   posicion TEXT(10),
   edad INT,
   ultimoPartido DATETIME,
   porcEficiencia FLOAT,
   PRIMARY KEY (id_arbitro) );

-- 03 Creacion tabla EQUIPOS
CREATE TABLE equipos(
id_equipos INT NOT NULL AUTO_INCREMENT,
id_campeonato INT,
categoria VARCHAR (10),
numJugador INT,
ultimoPartido DATETIME,
partidoGanados INT,
partidoPerdido INT,
partidoEmpatados INT,
campeonatosGanados INT,
estadio TEXT(20),
ciudad TEXT(10),
PRIMARY KEY (id_equipos),
FOREIGN KEY (id_campeonato) REFERENCES campeonato(id_campeonato) ON DELETE CASCADE);

-- 04 Creacion tabla PARTIDOS
CREATE TABLE partidos (
id_partido INT NOT NULL AUTO_INCREMENT,
id_campeonato INT,
id_arbitro INT,
equipoLocal TEXT(10),
equipoVisitante TEXT(10),
golesLocal INT,
golesVisitante INT,
numExpulsados INT,
posLocal FLOAT,
posVisitante FLOAT,
PRIMARY KEY (id_partido),
FOREIGN KEY (id_campeonato) REFERENCES campeonato (id_campeonato)  ON DELETE CASCADE,
FOREIGN KEY (id_arbitro) REFERENCES arbitro(id_arbitro) ON DELETE CASCADE);

-- 05 Creación tabla JUGADORES
CREATE TABLE jugadores(
id_jugador INT NOT NULL AUTO_INCREMENT,
id_partido INT,
```

```
nombre TEXT (20),
apellido TEXT(20),
ultimoPartido DATETIME,
goles INT,
asistencias INT,
numCamiseta INT,
PRIMARY KEY (id_jugador),
FOREIGN KEY (id_partido) REFERENCES partidos(id_partido)  ON DELETE CASCADE);

-- Visualización de las columas
SELECT * FROM arbitro;
SELECT * FROM campeonato;
SELECT * FROM equipos;
SELECT * FROM jugadores;
SELECT * FROM partidos;
-- Información sobtre el tipo de dato en cada columnas
DESCRIBE arbitro;
DESCRIBE campeonato;
DESCRIBE equipos;
DESCRIBE jugadores;
DESCRIBE partidos;
```

5. **Objetos**

Para esta base de datos se crearon una serie de objetos con el fin de manipular y procesar la información que se encuentra en la base de datos. A continuación, se explican cada uno de ellos.

a) **VISTAS:**

La primera **v**ista llamada ***vw_partidos_instituto*** creada tiene como objetivo conocer los partidos a disputar por un equipo, en este caso pusimos como ejemplo al club Instituto Atlético Central Córdoba. Sin embargo, funciona con cualquier otro club. Esta vista se lleva a cabo a partir de la tabla partidos, la cual posee información de interés sobre los distintos encuentros disputados y/o a disputarse en los distintos torneos. Particularmente se hace un filtro/selección a partir de las columnas equipoLocal y equipoVisitante.

```
204    ###############################################################################
205    -- DESAFIO COMPLEMENTARIO CREACION DE VISTAS
206 •  select* from partidos;
207    -- Creamos una vista. Lo ideal es ponerle vw adelante para decir que es una vista
208    -- 01. Vista de los partidos a disputar por Instituto
209 •  CREATE VIEW vw_partidos_instituto as
210        SELECT *
211        from partidos
212        WHERE equipoLocal = "Instituto" OR equipoVisitante = "Instituto";
213
```

La segunda vista se denomina ***vw_jugadores_goles*** la cual tiene como objetivo generar una tabla con los jugadores con mas goles y sin tarjetas rojas. Para esta se considera la tabla jugadores y se toman las columnas goles y tarjetas rojas para hacer las selecciones.

```sql
-- 02. Buscamos a los jugadores con mas goles y sin tarjetas rojas
CREATE VIEW vw_jugadores_goles as
SELECT *
from jugadores
WHERE goles > 0 and tarjetaRoja = 0;
```

La tercer tabla se llama ***vw_arbitros_2023*** *y* tiene como objetivo generar una tabla solo con los árbitros de la temporada 2023. Esta es creada a partir de la tabla árbitros la cual tiene, los nombres y apellidos de árbitros así como otras características pero que han dirigido partidos desde el 2015. Para este solo seleccionamos los nombres y apellidos de quienes disputaran el campeonato actual.

```sql
-- 03. Buscamos a los arbitros de la temporada 2023
SELECT * FROM arbitro;
CREATE VIEW vw_arbitros_2023 as
SELECT nombre, apellido
from arbitro
WHERE temporada = 2023;
```

La próxima denominada ***vw_jugadores_partidos*** tiene como objetivo buscar solo a los volantes de todos los equipos . Para este solo utilizamos la tabla jugadores y solo devuelve las columnas nombre, apellido, club.

```sql
-- 05. Buscamos a los jugadores de una posicion
CREATE VIEW vw_jugadores_partidos as
SELECT nombre, apellido, club
from jugadores WHERE posicion = 'Vol';
```

**b) FUNCIONES**

Para la creación se las funciones se utilizo la propia herramienta de MYSQL WORKBENCH. Sin embargo se copio y pego lo detallado por esta herramienta. En este caso se desarrollaron dos funciones.

En el primer caso se llama func_equipo_ganador, la cual devuelve el equipo ganador de un encuentro. Esta es del tipo *deterministica* y devuelve un varchar que sera el nombre del equipo ganador.

En esta función se deben ingresar dos parámetros de entrada (equipo_local y equipo_visitante) con las cuales se hace una selección de la tabla ***partidos*** que coincida con estos dos equipos. Una vez encontrado el partido, se lleva a cabo una diferencia entre los goles de ambos equipos utilizando las columnas ***golesLocal*** y ***golesVisitante.*** Seguidamente, el código ingresa en una expresión condicional la cual tiene como objetivo ver si la diferencia es positiva, negativa o es cero. En el caso que la diferencia de goles sea positiva sera el equipo local el ganador, en el caso que sea negativa la diferencia lo contrario y si es cero el resultado del partido fue un empate. A partir de esta expresión condicional se devuelve el nombre del equipo ganador dependiendo del resultado de la diferencia nombrada. Por otro lado, también se detallo otra condición, ya que en la tabla existen partidos que aun no fueron disputados (próximas fechas)

```
#### --- 01. func_equipo_ganador
-- La funcion equipo_ganador permite que a traves del paso de 2 parametros, equipo local
-- y el equipo visitante devuelva el nombre del equipo ganador. Esto se realiza gracias a la diferencia
-- de goles entre ambos equipos
USE `futbol_argentino`;
DROP function IF EXISTS `func_equipo_ganador`;
USE `futbol_argentino`;
DROP function IF EXISTS `futbol_argentino`.`func_equipo_ganador`;
;
DELIMITER $$
USE `futbol_argentino`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `func_equipo_ganador`(equipo_local VARCHAR(60),equipo_visitante VARCHAR(60))
RETURNS varchar(60) CHARSET utf8mb4
DETERMINISTIC
BEGIN
    DECLARE visitante_goles INT;
    DECLARE local_goles INT;
    DECLARE dif_goles INT;
    SET visitante_goles = (SELECT golesVisitante  FROM partidos WHERE equipoLocal = equipo_local AND equipoVisitante = equipo_visitante
    SET local_goles = (SELECT golesLocal  FROM partidos WHERE equipoLocal = equipo_local AND equipoVisitante = equipo_visitante);
    SET dif_goles = local_goles - visitante_goles;
    IF visitante_goles = NULL OR local_goles = NULL THEN
        RETURN 'Partido pendiente';
        END IF;
    IF dif_goles < 0 THEN
        RETURN equipo_visitante;
        ELSEIF dif_goles > 0 THEN
        RETURN equipo_local;
        ELSE
        RETURN 'EMPATE' ;
    END IF;
END$$
DELIMITER ;
;
```

La siguiente función se denomina ***func_proximos_partidos*** la cual tiene como objetivo mostrar los próximos partidos de un equipo así como también los partidos ya disputados. Para este se consideran tanto los partidos de local como de visitante, esto se debe a que se lleva a cabo la selección de la tabla partidos tanto de la columna ***equipoLocal*** como ***equipoVIsitante.*** Para correr esta función se deben ingresar dos parámetros de entrada: el numero de la fecha de interés y el equipo de interés. La salida de esta función es un varchar donde se concatena ciertas cadenas de texto según una expresión condicional. Si el equipo juega de visitante se realiza una expresión como : "*el equipo xxx juega de visitante contra 'yyy por la 'echa zzz.".* Mientras que si el equipo juega de local se mostrara la siguientes expresión: "*el equipo xxx juega de local contra 'yyy por la 'echa zzz.".* Esto dependerá de la selección realizada anteriormente si es local o es visitante en la fecha ingresada

```
#### ---- Funcion 02 proximos_partidos
-- La funcion proximos_partidos permite que a traves del paso del 2 parametros, la fecha y el equipo de interes
-- devolvera el equipo contra el que jugara en la fecha interes.
--  Tambien menciona si juega de visitante o de local
USE `futbol_argentino`;
DROP function IF EXISTS `proximos_partidos`;

USE `futbol_argentino`;
DROP function IF EXISTS `futbol_argentino`.`proximos_partidos`;
;

DELIMITER $$
USE `futbol_argentino`$$
CREATE DEFINER=`root`@`localhost` FUNCTION `func_proximos_partidos`(num_fecha VARCHAR(60),equipo VARCHAR(60)) RETURNS varchar(60) CHARS
    DETERMINISTIC
BEGIN
    DECLARE partidoLocal_select VARCHAR (60);
    DECLARE partidoVisitante_select VARCHAR (60);
    SET partidoLocal_select = (SELECT equipoLocal FROM partidos WHERE partido = num_fecha AND (equipoLocal = equipo OR equipoVisitante
    SET partidoVisitante_select = (SELECT equipoVisitante FROM partidos WHERE partido = num_fecha AND (equipoLocal = equipo OR equipoV
    IF partidoLocal_select = equipo  THEN
    RETURN CONCAT(equipo,' juega de local contra ',partidoVisitante_select, ' por la ', num_fecha);
    ELSE
    RETURN CONCAT(equipo, ' juega de visitante contra ',partidoLocal_select, ' por la ', num_fecha);
    END IF;
END$$

DELIMITER ;
;
```

**c) STORED PROCEDURES**

Para la creación se los procedimientos se utilizó la propia herramienta de  MYSQL WORKBENCH. Sin embargo se copio y pego lo detallado por esta herramienta.

El primer procedimiento se denomina ***sp_pos_jugadores*** el cual permite ordenar los paramentros ingresados. Particularmente, tiene dos paramentros de entrada, por un lado se debe indicar el campo de ordenamiento de una tabla y mediante un segundo parámetro, si el orden es descendente o ascendente. Para este ejemplo se utilizó la tabla jugadores y se ordeno la tabla según la columna goles de forma ascendente, mostrando a los goleadores del campeonato.

```
-- El ST sp_pos_jugadores permite ordenar los parametros ingresados
-- indicar a través de un parámetro el campo de ordenamiento de una tabla y
-- mediante un segundo parámetro, si el orden es descendente o ascendente.
-- En este caso se utiliza la tabla jugadores
USE `futbol_argentino`;
-- DROP procedure IF EXISTS `sp_pos_jugadores`;


DELIMITER $$
USE `futbol_argentino`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_pos_jugadores`(IN col_jugador VARCHAR(50), IN tipo_orden VARCHAR(5))
BEGIN
    SET @campo = col_jugador;
    SET @orden = tipo_orden;
    SET @clausula = CONCAT('SELECT * FROM jugadores ORDER BY ',@campo, ' ' , @orden);
    PREPARE runSQL FROM @clausula;
    EXECUTE runSQL;
    DEALLOCATE PREPARE runSQL;
END$$

DELIMITER ;
;

-- Ejemplo para ejecutar del stored procedure
CALL sp_pos_jugadores('goles','ASC');
```

El segundo procedimiento se denomina ***sp_delete*** el cual permite eliminar aquellos equipos que descenderán la próxima temporada y no serán tenidos en cuenta. En este procedimiento se deben ingesar dos parametros por un lado la tabla de interes que para nuestro ejemplo es la tala ***equipos*** y por otro lado la columna de donde se eliminara el dato, para nuestro ejemplo el equipo de interés.

```
#####
-- El ST sp_delete permite eliminar aquellos equipos que descenderan la proxima temporada
-- y no seran tenidos en cuenta. Se deben ingesar 2 parametros por un lado la tabla y por otro lado
-- el equipo de interes
USE `futbol_argentino`;
DROP procedure IF EXISTS `sp_delete`;

USE `futbol_argentino`;
DROP procedure IF EXISTS `futbol_argentino`.`sp_delete`;
;
DELIMITER $$
USE `futbol_argentino`$$
CREATE DEFINER=`root`@`localhost` PROCEDURE `sp_delete`(in pNombreTabla VARCHAR(50),in equipo_descender VARCHAR(50))
BEGIN
    SET SQL_SAFE_UPDATES = 0;
    DELETE FROM equipos WHERE nombreEquipo = equipo_descender;
     SET SQL_SAFE_UPDATES = 1;
END$$

DELIMITER ;
;

-- Ejemplo de ejecucion y prueba del registro eliminado
CALL sp_delete('equipos','Quilmes');
SELECT * FROM equipos WHERE nombreEquipo = 'Quilmes';
```

**d) TRIGGERS**

Para esta base de datos se crearon dos triggers, por un lado relacionada a la tabla partidos y por otro lado a los equipos. Como primer caso para ambos triggers se crearon las tablas donde se ubicaran los datos que se fueron actualizando de la primera tabla.
El primer trigger se denomina ***tr_add_new_partido*** tiene como objetivo obtener los datos cada vez que se insertan los partidos de una nueva fecha de un torneo en la tabla ***partidos***. Para ello en un paso previo se creo la tabla ***new_partido,*** en la cual se irán detallando estas actualizaciones, concretamente se guardaran los datos de:

- id partido
- Nombre del equipo local
- Nombre del equipo visitantes
- Fecha de actualización se utiliza la función current_date()
- Fecha de actualización se utiliza la función current_date()
- Tipo de cambio, en este caso siempre sera la palabra *INSERT*

Este trigger es del tipo ***AFTER INSERT***, es decir que luego de insertar un registro (datos de la nueva fecha del torneo) se ira completando y guardando en la tabla new_partido estas actualizaciones realizadas.

```sql
-- 01. Este trigger tiene como objetivo insertar los partidos de una nueva fecha de un torneo
-- Previo a la creacion del trigger creamos la tabla donde se muestran las actualizaciones
CREATE TABLE IF NOT EXISTS new_partido (
id_partido INT PRIMARY KEY,
equipoLocal VARCHAR (30),
equipoVisitante varchar(30),
fechaActualizacion date,
usuarioActualizacion varchar(30),
tipoOperacion VARCHAR(15)
);
-- Creamos el trigger de tipo AFTER e INSERT
CREATE TRIGGER `tr_add_new_partido`
    AFTER INSERT on `partidos`
    FOR EACH ROW
    INSERT INTO `new_partido` (id_partido,equipoLocal, equipoVisitante, fechaActualizacion,usuarioActualizacion, tipoOperacion )
    VALUES (NEW.id_partido,NEW.equipoLocal, NEW.equipoVisitante, current_date(), session_user(), 'INSERT');

-- Insertamos datos en la tabla partidos para probar si funciona el trigger
INSERT INTO partidos (id_partido, id_campeonato, id_arbitro, id_equipo_local, id_equipo_visitante, equipoLocal,equipoVisitante, golesLocal, golesVisitante, estadio, fecha, hora, partido)
    VALUES (225, 5, 63, 22, 14, 'Rosario Central', 'Defensa y Justicia', 0 , 2, 'Gigante de Arroyito', '30 de mayo', '21:45', 'Fecha 17');

-- Corroboramos la tabla partido donde inseramos el registro
SELECT * FROM partidos
    WHERE id_partido = 225;
-- Corroboramos viendo la tabla donde se nos guardan las actualizaciones
SELECT * FROM new_partido;
```

El segundo trigger se denomina ***tr_actualizacion_jugadores t***iene como objetivo obtener los datos cada vez que se actualizan los datos de jugadores en la tabla ***jugagores.*** Concretamente se tiene como finalidad guardar los datos cada vez que se actualiza el equipo de un jugador por ejemplo cuando es vendido a otro. Para ello en un paso previo se creo la tabla ***actualizacion_jugador*** en la cual se irán detallando estas actualizaciones, concretamente se guardaran los datos de:

- id jugador
- Nombre  y apellido
- Club actual: para este se coloca NEW.club
- Club anterior: para este se coloca OLD.club
- Fecha de actualización se utiliza la función current_date()
- Usuario de la base de datos que actualizo la información. Se utiliza la función, session_user()
- Tipo de cambio, en este caso siempre sera la palabra *UPDATE*

Este trigger es del tipo ***BEFORE UPDATE***, es decir que previo a la actualización de un registro (cambio del club de un jugador) se ira completando y guardando en la tabla ***actualizacion_jugador*** estas actualizaciones realizadas.

6.

```sql
-- 02. Este trigger tiene como objetivo actualizar el equipo de un jugador por ejemplo cuando es vendido
# a otro equipo
-- Previo a la creacion del trigger creamos la tabla donde se muestran las actualizaciones
SELECT * FROM jugadores;

CREATE TABLE IF NOT EXISTS actualizacion_jugador (
id_jugador INT PRIMARY KEY,
nombre VARCHAR (50),
apellido VARCHAR (50),
clubActual VARCHAR (50),
clubAnterior VARCHAR (50),
fechaActualizacion date,
usuarioActualizacion varchar(30),
tipoOperacion VARCHAR(15)
);

-- Creamos el trigger de tipo BEFORE  y UPDATE
CREATE TRIGGER `tr_actualizacion_jugadores`
BEFORE UPDATE on `jugadores`
FOR EACH ROW
INSERT INTO `actualizacion_jugador` (id_jugador,nombre, apellido,clubActual,clubAnterior,fechaActualizacion,usuarioActualizacion, tipoOperacion )
VALUES (NEW.id_jugador,NEW.nombre, NEW.apellido,NEW.club,OLD.club, current_date(), session_user(), 'UPDATE');
-- Se tuvo que poner eliminar el modo seguro porque saltaba error
SET SQL_SAFE_UPDATES = 0;
-- Actualizamos datos en la tabla partidos para probar si funciona el trigger
UPDATE jugadores
SET club = 'Rosario Central'
WHERE nombre = 'Lanzillotta' AND apellido = ' Federico';

-- Corroboramos la tabla partido donde inseramos el registro
SELECT * FROM jugadores WHERE nombre = 'Lanzillotta' AND apellido = ' Federico';
-- Corroboramos viendo la tabla donde se nos guardan las actualizaciones
SELECT * FROM actualizacion_jugador;
```

## 7. USUARIOS

Se realizó la creación de dos usuarios para la utilización de la base de datos creadas. En cada uno de los casos se estableció una contraseña predeterminada. Sin embargo, aun estos usuarios no tienen ningún permiso establecido en nuestra base de datos.

```sql
# Creacion de dos usuarios con una contraseña dentro del localhost
drop USER 'usuario01'@'localhost';
CREATE USER 'usuario01'@'localhost' IDENTIFIED BY 'usuario01';
CREATE USER 'usuario02'@'localhost' IDENTIFIED BY 'usuario02';
```

Posteriormente se establecieron permisos diferentes a cada uno de los usuarios. En el caso del *usuario01* le asignaron permiso de solo lectura para todas las tablas, es decir donde solo pueda visualizar la información a través de la consulta *SELECT.* Mientras que para el *usuario02,* además de los permisos de lectura, también se le brindo el permiso de inserción y modificación de datos con los comandos *UPDATE* e *INSERT.*

```sql
# Asignacion de permisos
#Usuario 01 se le asgnan permisos de solo lectura a cada una de las tablas (Solo select)
GRANT SELECT ON futbol_argentino.arbitro TO 'usuario01'@'localhost';
GRANT SELECT ON futbol_argentino.campeonato TO 'usuario01'@'localhost';
GRANT SELECT ON futbol_argentino.equipos TO 'usuario01'@'localhost';
GRANT SELECT ON futbol_argentino.jugadores TO 'usuario01'@'localhost';
GRANT SELECT ON futbol_argentino.partidos TO 'usuario01'@'localhost';
```

```
# Usuario 02 permisos de lectura, insercion y modificacion de datos.
GRANT SELECT, UPDATE, INSERT ON futbol_argentino.arbitro TO 'usuario02'@'localhost';
GRANT SELECT, UPDATE, INSERT ON futbol_argentino.campeonato TO 'usuario02'@'localhost';
GRANT SELECT, UPDATE, INSERT ON futbol_argentino.equipos TO 'usuario02'@'localhost';
GRANT SELECT, UPDATE, INSERT ON futbol_argentino.jugadores TO 'usuario02'@'localhost';
GRANT SELECT, UPDATE, INSERT ON futbol_argentino.partidos TO 'usuario02'@'localhost';
# Vemos los permisos para el usuario 02
SHOW GRANTS FOR  'usuario02'@'localhost';
```

8. TRANSACCIONES

## Sentencias TCL

## Se eligen 2 tablas: partidos y jugadores

# TABLA 01

Se realizaron una serie de modificaciones en los registros controladas por transacciones. Para esto se eligieron 2 tablas: *partidos* y *jugadores*. En la primera tabla, se eliminó un registro donde previamente se comenzó con el script *START TRANSACTION.* Luego se realizó la prueba para comprobar que el registro realmente haya sido borrado.

```
# --------  01. Ejemplo de la tabla partidos   --------
-- Comenzamos con la transaccion
START TRANSACTION;
-- Eliminamos un registro segun una condicion
DELETE FROM
partidos
WHERE
equipoLocal = 'San Lorenzo';

-- Probamos si se borro o no el registro. En este caso deberian aparecer los valores como null
SELECT * FROM partidos
WHERE
equipoLocal = 'San Lorenzo';
```

Posteriormente, corrimos el comando *ROLLBACK* con el fin de volver hacia atrás y evitar el borrado del registro. Luego comprobamos que esta acción se haya hecho correctamente.

```
-- Volvemos para atras, previo al delete
ROLLBACK;
-- O lo podemos dejar de forma permanente
-- COMMMIT;
-- Revisamos que se haya generado el commit o el rollback
SELECT * FROM partidos
WHERE
equipoLocal = 'San Lorenzo';
```

Mientras que, en la segunda tabla, *equipo,* se llevó a cabo una inserción de 8 nuevos registros iniciando también una transacción. Se agregó un *savepoint* a posteriori de la inserción del registro número 4 y otro *savepoint* a posteriori del registro número 8 con el fin de realizar un *ROLLBACK.*

```sql
-- Comenzamos transaccion
START TRANSACTION;
-- Insertamos 4 registros
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
 VALUES (66, 'Comunicaciones',  'B Metropolitana',  'Buenos Aires', 'Alfredo Ramos',    3500,   'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
VALUES (67, 'San Miguel','B Metropolitana','Los Polvorines', 'Estadio Malvinas Argentinas', 7176, 'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
VALUES (68, 'Los Andes', 'B Metropolitana', 'Lomas de Zamora',  'Eduardo Gallardón', 38000, 'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, ciudad,categoria , estadio, capacidad, ligaActual)
 VALUES (69,'Villa San Carlos','B Metropolitana', 'Berisso', 'Genacio Sálice', 4000,'B Metropolitana');
-- Generamos el primer savepoint
savepoint lote_1;
-- Insertamos 4 registros mas
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
VALUES (70, 'Dock Sud'  ,'B Metropolitana'  ,'Dock Sud', 'Estadio de los Inmigrantes' , 9500,'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
 VALUES (71, 'Sacachispas'  ,'B Metropolitana', 'Villa Soldati' ,'Estadio Beto Larrosa', 10000, 'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
VALUES (72, 'Argentino de Merlo', 'B Metropolitana', 'Merlo'    ,'Juan Carlos Brieva', 11000, 'B Metropolitana');
INSERT INTO equipos (id_equipos, nombreEquipo, categoria ,ciudad, estadio, capacidad, ligaActual)
 VALUES (73, 'Acassuso',    'B Metropolitana','Boulogne','La Quema',    800 ,'B Metropolitana');
-- Generamos el segundo savepoint
savepoint lote_2;
```

Posteriormente se realizaron las comprobaciones tanto de forma previa como posterior al *ROLLBACK* con el fin de confirmar que los *savepoints* se hayan realizado correctamente.

```sql
-- validamos algun dato del savepoint 1
SELECT * FROM equipos WHERE id_equipos = 66;

-- validamos algun dato del savepoint
SELECT * FROM equipos WHERE id_equipos = 73;

-- Vuelvo al primer segmento
ROLLBACK TO lote_1;

-- validamos algun dato del savepoint 1 y 2
SELECT * FROM equipos WHERE id_equipos = 66;
SELECT * FROM equipos WHERE id_equipos = 73;
```

9. BACKUP BASE DE DATOS

Por último, se llevó a cabo un backup del proyecto de la base de datos creada, incluyendo en éste solamente las tablas. En este backup solo incluyen sólo los datos, dejando de lado su estructura. Se realizo el back up a través de *administracion => data export* donde se exportaron solo los datos (*dump data only*) en un solo archivo con la opcion "*export to self-contained file*". Además, también se incluyó el esquema. Las tablas que se exportaron son: jugadores, equipos, partidos, campeonatos y árbitros.

Posteriormente para probar que se haya hecho correctamente este backup. En primer lugar, se eliminó el esquema para luego volver a crearlo como así también las tablas ya que solo se

exportaron los datos y no la estructura. Por último, se importaron los datos a través de las herramientas de *mysql.*