Escuela Superior
de Ingeniería y Tecnología
Universidad de La Laguna

INAZUMA ELEVEN DATABASE

Febe Fariña Aguirre
(alu0101430973@ull.edu.es)

Aday Chocho Aisa
(alu0101437538@ull.edu.es)

Pablo Pais Ficret
(alu0101311135@ull.edu.es)

1. Introducción y objetivos

Inazuma Eleven es una franquicia de medios creada por Akihiro Hino, presidente de Level-5, que comenzó con el lanzamiento del videojuego de mismo nombre en 2008. Es un videojuego de rol, RPG y fútbol cuya historia se centra en un chico llamado Mark Evans, capitán y portero del club de fútbol del Instituto Raimon. El objetivo del protagonista es participar en un torneo de fútbol llamado Fútbol Frontier y convertirse en los mejores del país.

El gran atractivo que ofreció el juego fue la combinación de géneros, dando a lugar a partidos de fútbol donde los jugadores pueden usar poderes (conocidos como supertécnicas) para marcar la diferencia, resultando en encuentros frenéticos y divertidos. Esta peculiaridad fue la base del éxito de la franquicia, que se fue expandiendo a tal punto que se lanzaron otros 5 juegos principales, 9 spin off, 7 temporadas de anime, 3 películas y grandes cantidades de merchandising, entre las que podemos destacar cartas tipo TCG, tazos, etc. Y no solo eso, si no que para 2024 se espera un juego más para la saga.

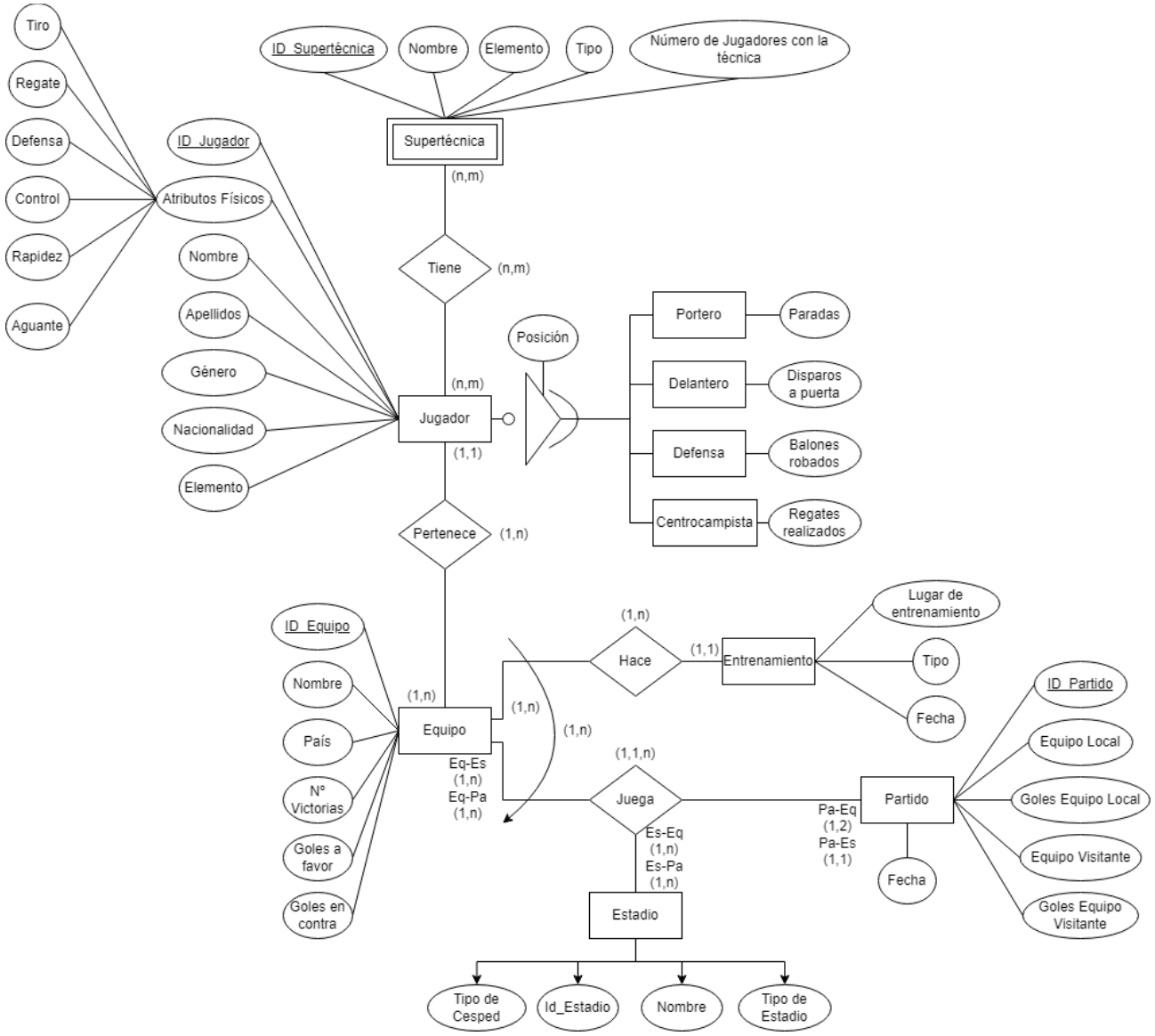
Debido a la gran cantidad de productos que se han lanzado al mercado, existe una gran cantidad de jugadores. Y esta cantidad se hace mayor si contamos las distintas versiones de cada jugador (por ejemplo, un jugador que juega para varios equipos o que en temporadas posteriores se vuelve adulto). Considerando lo anterior, se estima que existan en torno a 5000 jugadores en toda la franquicia. Como el próximo juego pretende incluir a casi todos esos jugadores, la construcción de equipos en un entorno competitivo puede volverse un reto al tener esa gran cantidad de jugadores.

Por ello, el objetivo de la base de datos es almacenar toda la información de esos jugadores, equipos, entrenamientos y partidos a lo largo de toda la franquicia para que los usuarios puedan acceder y emplear esos datos de manera sencilla, facilitando la construcción de equipos en un entorno competitivo.

2. Requisitos

- La base de datos debe poder almacenar a cualquier jugador del juego, con su información, tipo, posición, género, nacionalidad y atributos físicos.
- Según la posición se deben de poder almacenar los tiros, balones robados, regates o paradas.
- Se deben de poder almacenar las supertécnicas de los jugadores, incluyendo nombre, tipo y elemento. Además, se debe de contar el número de jugadores que la usan.
- Se debe de almacenar los equipos a los que pertenecen los jugadores, incluyendo nombre, nacionalidad de los equipos, las victorias y los goles (a favor y en contra).
- Se deben de almacenar los partidos que ocurren entre los equipos, guardando la fecha, equipos, goles y estadio en el que se juega.
- Se debe de almacenar la información de cada estadio, es decir, el nombre, el tipo de estadio y el tipo de césped.
- Debido a que ciertos torneos obligan a tener equipos entrenados, se deben de almacenar los entrenamientos hechos a distintos equipos, guardando el lugar, la fecha y el tipo. Se debe de impedir a un equipo jugar si no ha entrenado previamente al menos una vez.
- La duración de los partidos y/o entrenamientos se ha establecido en 2h. Un equipo no puede jugar o entrenar en las 2 horas anteriores o posteriores a un partido o entrenamiento. En un estadio o lugar tampoco se puede incluir un nuevo partido o entrenamiento si en las dos horas anteriores o posteriores está ocupado.

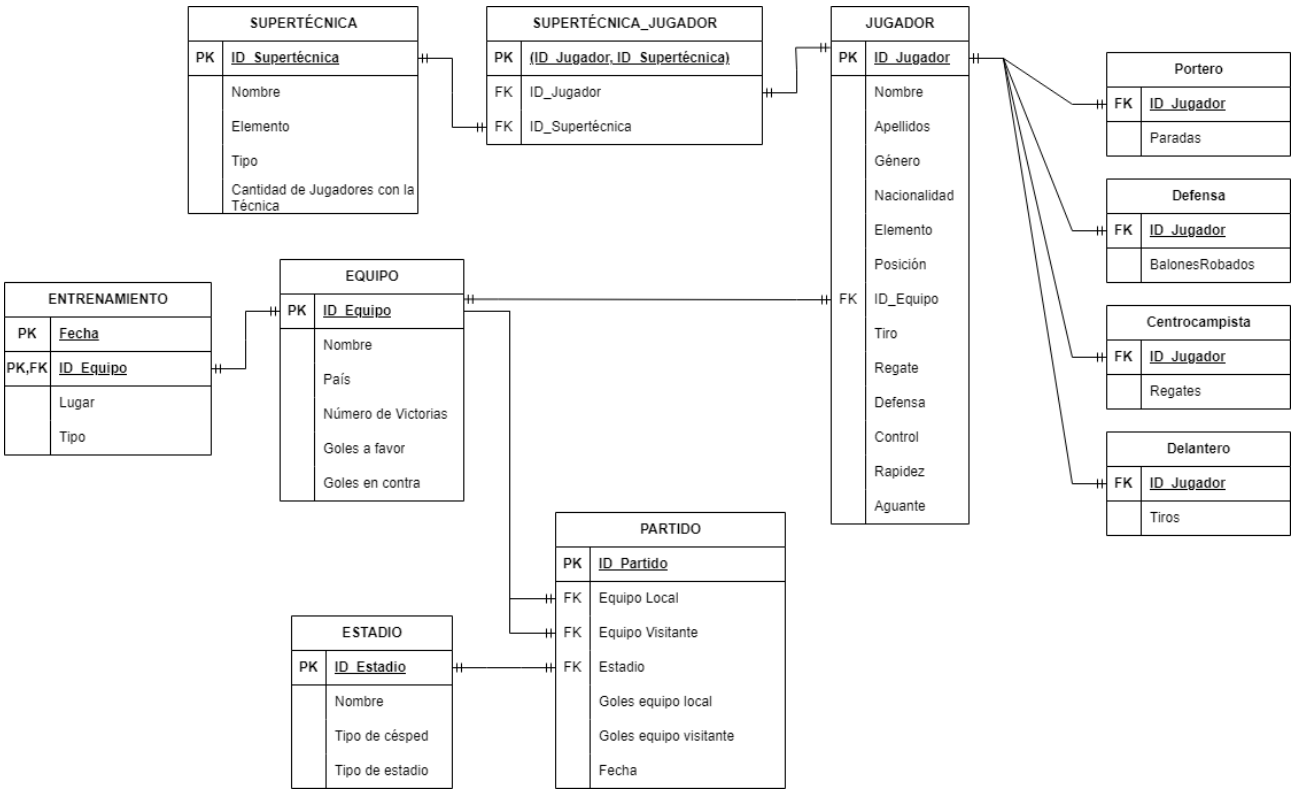
3. Diagrama E/R (Entidad Relación)



Restricciones:

- Todos los valores numéricos deben de ser positivos.
- Un equipo no puede jugar o entrenar en las 2 horas anteriores o posteriores a un partido o entrenamiento.
- En un estadio o lugar de entrenamiento no se puede incluir un nuevo evento si en las dos horas anteriores o posteriores está ocupado.
- Un equipo no puede jugar en un partido contra sí mismo

4. Diagrama Relacional



5. Base de datos en SQL

5.1. Tablas y Tipos

5.1.1. Tabla Jugador

Tabla que contiene los jugadores de la base de datos.

```
CREATE TABLE JUGADOR(
id_jugador SERIAL NOT NULL,
nombre VARCHAR(10) NOT NULL,
apellidos VARCHAR(20),
genero GENERO NOT NULL,
nacionalidad VARCHAR(20) NOT NULL,
elemento ELEMENTO NOT NULL,
posicion POSICION NOT NULL,
id_equipo INTEGER NOT NULL,
tiro INTEGER NOT NULL,
regate INTEGER NOT NULL,
```

```
defensa INTEGER NOT NULL,
control INTEGER NOT NULL,
rapidez INTEGER NOT NULL,
aguante INTEGER NOT NULL,
PRIMARY KEY(id_jugador)
);
```

5.1.2. Tabla Equipo

Tabla que contiene los equipos del primer videojuego.

```
CREATE TABLE EQUIPO(
id_equipo SERIAL NOT NULL,
nombre VARCHAR(20) NOT NULL UNIQUE,
pais VARCHAR(20) NOT NULL,
victorias INTEGER DEFAULT 0 NOT NULL,
goles_a_favor INTEGER DEFAULT 0 NOT NULL,
goles_en_contra INTEGER DEFAULT 0 NOT NULL,
PRIMARY key(id_equipo)
);
```

5.1.3. Tabla Supertécnica

Tabla que contiene las supertécnicas (La supertécnica es un movimiento especial que hacen los jugadores).

```
CREATE TABLE SUPERTECNICA(
id_supertecnica SERIAL NOT NULL,
nombre VARCHAR(30) NOT NULL,
elemento ELEMENTO NOT NULL,
tipo TIPO_SUPERTECNICA NOT NULL,
cantidad_jugadores_con_supertecnica INTEGER DEFAULT 0 NOT NULL,
PRIMARY KEY(id_supertecnica)
);
```

5.1.4. Tabla Partido

```
CREATE TABLE EQUIPO(
id_equipo SERIAL NOT NULL,
nombre VARCHAR(20) NOT NULL UNIQUE,
pais VARCHAR(20) NOT NULL,
victorias INTEGER DEFAULT 0 NOT NULL,
```

```
goles_a_favor INTEGER DEFAULT 0 NOT NULL,
goles_en_contra INTEGER DEFAULT 0 NOT NULL,
PRIMARY key(id_equipo)
);
```

5.1.5. Tabla Estadio

Tabla que contiene los estadios en los que suceden los partidos.

```
CREATE TABLE ESTADIO(
id_estadio SERIAL NOT NULL,
nombre VARCHAR(50) NOT NULL,
cesped CESPED NOT NULL,
tipo TIPO_ESTADIO NOT NULL,
PRIMARY KEY(id_estadio)
);
```

5.1.6. Tabla Entrenamiento

Tabla con los entrenamientos que realizan los diferentes equipos de la base de datos.

```
CREATE TABLE ENTRENAMIENTO(
id_training SERIAL NOT NULL,
fecha TIMESTAMP NOT NULL,
id_equipo INTEGER NOT NULL,
lugar VARCHAR(20) NOT NULL,
tipo TIPO_ENTRENAMIENTO not NULL,
PRIMARY KEY(fecha, id_equipo)
);
```

5.1.7. Tabla Supertécnica Jugador

Tabla que contiene los id's de los jugadores y los id's de las supertécnicas asignadas a cada jugador.

```
CREATE TABLE SUPERTECNICA_JUGADOR(
id_jugador INTEGER NOT NULL,
id_supertecnica INTEGER NOT NULL,
PRIMARY KEY(id_jugador, id_supertecnica)
);
```

5.1.8. Tabla Portero

Tabla que contiene los id's de los jugadores que son porteros.

```
CREATE TABLE PORTERO(
id_jugador INTEGER NOT NULL,
paradas INTEGER DEFAULT 0 NOT NULL
);
```

5.1.9. Tabla Defensa

Tabla que contiene los id's de los jugadores que son defensa.

```
CREATE TABLE DEFENSA(
id_jugador INTEGER NOT NULL,
balones_robados INTEGER DEFAULT 0 NOT NULL
);
```

5.1.10. Tabla Centrocampista

Tabla que contiene los id's de los jugadores que son centrocampista.

```
CREATE TABLE CENTROCAMPISTA(
id_jugador INTEGER NOT NULL,
regates_realizados INTEGER DEFAULT 0 NOT NULL
);
```

5.1.11. Tabla Delantero

Tabla que contiene los id's de los jugadores que son delantero

```
CREATE TABLE DELANTERO(
id_jugador INTEGER NOT NULL,
disparos_a_puerta INTEGER DEFAULT 0 NOT NULL
);
```

5.1.12. Tipos de Datos

En nuestro esquema de base de datos se han creado los siguientes tipos de datos:

```
CREATE TYPE TIPO_ESTADIO AS ENUM('Cubierto', 'Exterior');
CREATE TYPE CESPED AS ENUM('Natural', 'Artificial', 'Sin cesped');
CREATE TYPE ELEMENTO AS ENUM('Bosque', 'Montaña', 'Aire', 'Fuego');
CREATE TYPE TIPO_ENTRENAMIENTO AS ENUM('Tiro a puerta', 'Vuelta al campo',
'Control de balon');
```

```
CREATE TYPE TIPO_SUPERTECNICA AS ENUM('Atajo', 'Tiro', 'Regate', 'Bloqueo');
CREATE TYPE GENERO AS ENUM ('Masculino', 'Femenino', 'desconocido');
CREATE TYPE POSICION AS ENUM('Portero', 'Defensa', 'Centrocampista', 'Delantero');
```

Estos tipos contienen enums, estos los usamos para asegurarnos de que algunos campos en concreto sigan dichos valores.

5.2. Carga de Datos

En este paso se va a enseñar cómo se insertan los datos en las diferentes tablas. Solo se mostrará una inserción de ejemplo, las demás inserciones se encuentran en *inazuma_db.sql*.

5.2.1. Tabla Jugador

```
INSERT INTO JUGADOR
(nombre,apellidos,genero,nacionalidad,elemento,posicion,id_equipo,
tiro,regate,defensa,control,rapidez,aguante)
VALUES ('Mark','Evans','Masculino','Japonés','Montaña','Portero',1,
72,72,77,70,79,68)
```

5.2.2. Tabla Equipo

```
INSERT INTO EQUIPO (nombre, pais, victorias, goles_a_favor, goles_en_contra)
VALUES ('Raimon', 'Japón', 0, 0, 0);
```

5.2.3. Tabla Supertécnica

```
INSERT INTO SUPERTECNICA (nombre, elemento, tipo,
cantidad_jugadores_con_supertecnica)
VALUES ('Triángulo Letal', 'Bosque', 'Tiro', 0)
```

5.2.4. Tabla Partido

```
INSERT INTO PARTIDO(id_equipo_local, id_equipo_visitante, id_estadio, goles_local,
goles_visitante, fecha)
VALUES (1, 2, 3, 1, 20, '2008-10-12 10:00:00')
```

5.2.5. Tabla Estadio

```
INSERT INTO ESTADIO (nombre, cesped, tipo)
VALUES ('Estadio Fútbol Frontier', 'Natural', 'Exterior')
```

5.2.6. Tabla Entrenamiento

```
INSERT INTO ENTRENAMIENTO(fecha, id_equipo, lugar, tipo)
VALUES ('2008-10-05 17:00:00', 1, 'Instituto Raimon', 'Vuelta al campo')
```

5.2.7. Tabla Supertécnica Jugador

```
INSERT INTO SUPERTECNICA_JUGADOR(id_jugador, id_supertecnica)
VALUES (23, 1)
```

5.3. Constraints

5.4.1. Claves ajenas

Establecen las claves ajenas de las tablas, creando las relaciones.

```
ALTER TABLE PARTIDO ADD CONSTRAINT fk_equipo_local FOREIGN KEY (id_equipo_local) REFERENCES EQUIPO(id_equipo) ON DELETE CASCADE;
ALTER TABLE PARTIDO ADD CONSTRAINT fk_equipo_visitante FOREIGN KEY (id_equipo_visitante) REFERENCES EQUIPO(id_equipo) ON DELETE CASCADE;
ALTER TABLE PARTIDO ADD CONSTRAINT fk_estadio FOREIGN KEY (id_estadio) REFERENCES ESTADIO(id_estadio) ON DELETE CASCADE;
ALTER TABLE ENTRENAMIENTO ADD CONSTRAINT fk_equipo FOREIGN KEY (id_equipo) REFERENCES EQUIPO(id_equipo) ON DELETE CASCADE;
ALTER TABLE JUGADOR ADD CONSTRAINT fk_equipo FOREIGN KEY (id_equipo) REFERENCES EQUIPO(id_equipo) ON DELETE CASCADE;
ALTER TABLE SUPERTECNICA_JUGADOR ADD CONSTRAINT fk_jugador FOREIGN KEY (id_jugador) REFERENCES JUGADOR(id_jugador) ON DELETE CASCADE;
ALTER TABLE SUPERTECNICA_JUGADOR ADD CONSTRAINT fk_supertecnica FOREIGN KEY (id_supertecnica) REFERENCES SUPERTECNICA(id_supertecnica) ON DELETE CASCADE;
ALTER TABLE PORTERO ADD CONSTRAINT fk_jugador FOREIGN KEY (id_jugador) REFERENCES JUGADOR(id_jugador) ON DELETE CASCADE;
ALTER TABLE DELANTERO ADD CONSTRAINT fk_jugador FOREIGN KEY (id_jugador) REFERENCES JUGADOR(id_jugador) ON DELETE CASCADE;
ALTER TABLE DEFENSA ADD CONSTRAINT fk_jugador FOREIGN KEY (id_jugador) REFERENCES JUGADOR(id_jugador) ON DELETE CASCADE;
ALTER TABLE CENTROCAMPISTA ADD CONSTRAINT fk_jugador FOREIGN KEY (id_jugador) REFERENCES JUGADOR(id_jugador) ON DELETE CASCADE;
```

5.4.1. Checks de datos

Los checks de comprobación de datos se componen básicamente de un check que comprueba que en los partidos ambos equipos no sean el mismo y otros checks que comprueban que los valores sean positivos.

```
ALTER TABLE PARTIDO ADD CONSTRAINT not_equal_teams CHECK (id_equipo_local <> id_equipo_visitante);
ALTER TABLE JUGADOR ADD CONSTRAINT positive_stats CHECK (tiro > 0 AND regate > 0 AND defensa > 0 AND control > 0 AND rapidez > 0 AND aguante > 0);
ALTER TABLE PARTIDO ADD CONSTRAINT positive_goals CHECK (goles_local >= 0 AND goles_visitante >= 0);
ALTER TABLE EQUIPO ADD CONSTRAINT positive_wins_ties_loses CHECK (victorias >= 0);
ALTER TABLE EQUIPO ADD CONSTRAINT positive_goals CHECK (goles_a_favor >= 0 AND goles_en_contra >= 0);
ALTER TABLE PORTERO ADD CONSTRAINT positive_saves CHECK (paradas >= 0);
ALTER TABLE DELANTERO ADD CONSTRAINT positive_shots CHECK (disparos_a_puerta >= 0);
ALTER TABLE DEFENSA ADD CONSTRAINT positive_stolen_balls CHECK (balones_robados >= 0);
ALTER TABLE CENTROCAMPISTA ADD CONSTRAINT positive_dribbles CHECK (regates_realizados >= 0);
```

5.4. Triggers

5.4.1. Victorias

Se han realizado tres triggers que trabajan con el atributo victorias de la tabla equipo:

- El trigger **sumar_victorias**, que tras introducir un registro a la tabla *partidos* añade una victoria al equipo ganador, es decir, el que más goles haya marcado (en caso de empate no añade victorias).

```
-- Disparador que sume victorias a un equipo cuando gana un partido
CREATE OR REPLACE FUNCTION sumar_victorias() RETURNS TRIGGER AS $$
BEGIN
  IF NEW.goles_local > NEW.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias + 1 WHERE id_equipo = NEW.id_equipo_local;
  ELSIF NEW.goles_local < NEW.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias + 1 WHERE id_equipo = NEW.id_equipo_visitante;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER sumar_victorias AFTER INSERT ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE sumar_victorias();
```

- El trigger **restar_victorias**, que tras borrar un registro a la tabla *partidos* resta la victoria al equipo que ganó el partido (no hace nada en caso de empate).

```
CREATE TRIGGER actualizar_victorias AFTER UPDATE ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE actualizar_victorias();

-- Disparador que actualiza las victorias al borrar un registro en la tabla PARTIDO
CREATE OR REPLACE FUNCTION restar_victorias() RETURNS TRIGGER AS $$
BEGIN
  IF OLD.goles_local > OLD.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias - 1 WHERE id_equipo = OLD.id_equipo_local;
  ELSIF OLD.goles_local < OLD.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias - 1 WHERE id_equipo = OLD.id_equipo_visitante;
  END IF;

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restar_victorias BEFORE DELETE ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE restar_victorias();
```

- El trigger **actualizar_victorias**, que al actualizar un registro de la tabla *partidos* resta la victoria del equipo ganador antes de la modificación y se la suma al ganador después de la modificación.

```
-- Disparador que actualiza las victorias al modificar un registro en la tabla PARTIDO
CREATE OR REPLACE FUNCTION actualizar_victorias() RETURNS TRIGGER AS $$
BEGIN
  IF OLD.goles_local > OLD.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias - 1 WHERE id_equipo = OLD.id_equipo_local;
  ELSIF OLD.goles_local < OLD.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias - 1 WHERE id_equipo = OLD.id_equipo_visitante;
  END IF;

  IF NEW.goles_local > NEW.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias + 1 WHERE id_equipo = NEW.id_equipo_local;
  ELSIF NEW.goles_local < NEW.goles_visitante THEN
    UPDATE EQUIPO SET victorias = victorias + 1 WHERE id_equipo = NEW.id_equipo_visitante;
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

5.4.2. Goles a favor y en contra

Como en el caso anterior, se han realizado tres triggers que trabajan con los atributos de goles a favor y en contra de la tabla equipo:

- El trigger **sumar_goles**, que tras introducir un registro a la tabla *partidos* añade los goles a favor y en contra a cada equipo.

```
-- Disparador que sume goles a favor y en contra a los equipos
CREATE OR REPLACE FUNCTION sumar_goles() RETURNS TRIGGER AS $$
BEGIN
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor + NEW.goles_local WHERE id_equipo = NEW.id_equipo_local;
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor + NEW.goles_visitante WHERE id_equipo = NEW.id_equipo_visitante;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra + NEW.goles_visitante WHERE id_equipo = NEW.id_equipo_local;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra + NEW.goles_local WHERE id_equipo = NEW.id_equipo_visitante;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER sumar_goles AFTER INSERT ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE sumar_goles();
```

- El trigger **restar_goles**, que tras borrar un registro a la tabla *partidos* resta los respectivos goles a favor y en contra en cada equipo.

```
-- Disparador que actualiza los goles a favor y en contra a los equipos al borrar un registro en la tabla PARTIDO
CREATE OR REPLACE FUNCTION restar_goles() RETURNS TRIGGER AS $$
BEGIN
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor - OLD.goles_local WHERE id_equipo = OLD.id_equipo_local;
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor - OLD.goles_visitante WHERE id_equipo = OLD.id_equipo_visitante;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra - OLD.goles_visitante WHERE id_equipo = OLD.id_equipo_local;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra - OLD.goles_local WHERE id_equipo = OLD.id_equipo_visitante;

  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restar_goles BEFORE DELETE ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE restar_goles();
```

- El trigger **actualizar_goles**, que al actualizar un registro de la tabla *partidos* primero resta los goles a favor y en contra de cada equipo antes de la modificación y luego los vuelve a sumar después de la modificación.

```
-- Disparador que actualiza los goles a favor y en contra a los equipos
CREATE OR REPLACE FUNCTION actualizar_goles() RETURNS TRIGGER AS $$
BEGIN
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor - OLD.goles_local WHERE id_equipo = OLD.id_equipo_local;
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor - OLD.goles_visitante WHERE id_equipo = OLD.id_equipo_visitante;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra - OLD.goles_visitante WHERE id_equipo = OLD.id_equipo_local;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra - OLD.goles_local WHERE id_equipo = OLD.id_equipo_visitante;

  UPDATE EQUIPO SET goles_a_favor = goles_a_favor + NEW.goles_local WHERE id_equipo = NEW.id_equipo_local;
  UPDATE EQUIPO SET goles_a_favor = goles_a_favor + NEW.goles_visitante WHERE id_equipo = NEW.id_equipo_visitante;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra + NEW.goles_visitante WHERE id_equipo = NEW.id_equipo_local;
  UPDATE EQUIPO SET goles_en_contra = goles_en_contra + NEW.goles_local WHERE id_equipo = NEW.id_equipo_visitante;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER actualizar_goles AFTER UPDATE ON PARTIDO FOR EACH ROW EXECUTE PROCEDURE actualizar_goles();
```

5.4.3. Jugadores con supertécnica

Se han realizado tres triggers que trabajan con el atributo que recoge la cantidad de jugadores con la supertecnica de la tabla supertécnica:

- El trigger **jugadores_con_supertecnica**, que tras introducir un registro a la tabla supertecnica_*jugador*, suma una unidad a la cantidad de jugadores con esa supertécnica.

```
-- Disparador que sume usuarios de supertécnicas al añadir a la tabla jugador-supertécnicas
CREATE OR REPLACE FUNCTION jugadores_con_supertecnica() RETURNS TRIGGER AS $$
BEGIN
  UPDATE SUPERTECNICA SET cantidad_jugadores_con_supertecnica = cantidad_jugadores_con_supertecnica + 1
  WHERE id_supertecnica = NEW.id_supertecnica;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER jugadores_con_supertecnica AFTER INSERT ON SUPERTECNICA_JUGADOR
FOR EACH ROW EXECUTE PROCEDURE jugadores_con_supertecnica();
```

- El trigger **restar_jugadores_con_supertecnica**, que tras borrar un registro a la tabla supertecnica_*jugador*, resta una unidad a la cantidad de jugadores con esa supertécnica.

```sql
-- Disparador que actualiza los usuarios de supertécnicas al borrar un registro en la tabla PARTIDO
CREATE OR REPLACE FUNCTION restar_jugadores_con_supertecnica() RETURNS TRIGGER AS $$
BEGIN
  UPDATE SUPERTECNICA SET cantidad_jugadores_con_supertecnica = cantidad_jugadores_con_supertecnica - 1
  WHERE id_supertecnica = OLD.id_supertecnica;
  RETURN OLD;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER restar_jugadores_con_supertecnica BEFORE DELETE ON SUPERTECNICA_JUGADOR
FOR EACH ROW EXECUTE PROCEDURE restar_jugadores_con_supertecnica();
```

- El trigger **actualizar_jugadores_con_supertecnica**, que al actualizar un registro de la tabla supertecnica_*jugador*, resta una unidad a la cantidad de jugadores antes de la modificación y la suma después de la modificación.

```sql
-- Disparador que actualiza los usuarios de supertécnicas
CREATE OR REPLACE FUNCTION actualizar_jugadores_con_supertecnica() RETURNS TRIGGER AS $$
BEGIN
  UPDATE SUPERTECNICA SET cantidad_jugadores_con_supertecnica = cantidad_jugadores_con_supertecnica - 1
  WHERE id_supertecnica = OLD.id_supertecnica;
  UPDATE SUPERTECNICA SET cantidad_jugadores_con_supertecnica = cantidad_jugadores_con_supertecnica + 1
  WHERE id_supertecnica = NEW.id_supertecnica;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER actualizar_jugadores_con_supertecnica AFTER UPDATE ON SUPERTECNICA_JUGADOR
FOR EACH ROW EXECUTE PROCEDURE actualizar_jugadores_con_supertecnica();
```

5.4.4. Validaciones de fecha

Se han realizado 8 triggers empleando 4 funciones que validan los horarios de los partidos y entrenamientos:

- La función **validar_horario_partido** comprueba que a la hora de introducir un partido, no existan entrenamientos en las 2 horas anteriores y posteriores para ninguno de los equipos.

```sql
-- Disparador que valida la hora de inicio en un partido de un equipo
CREATE OR REPLACE FUNCTION validar_horario_partido()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE id_equipo = NEW.id_equipo_local
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) OR EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE id_equipo = NEW.id_equipo_visitante
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar un partido dentro de las 2 horas anteriores o posteriores a un entrenamiento.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_horario_partido_insert BEFORE INSERT ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_horario_partido();
CREATE TRIGGER validar_horario_partido_update BEFORE UPDATE ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_horario_partido();
```

- La función **validar_horario_entrenamiento**, similar a la función anterior pero aplicada a los entrenamientos.

```sql
-- Disparador que valida la hora de inicio en un entrenamiento de un equipo
CREATE OR REPLACE FUNCTION validar_horario_entrenamiento()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE NEW.id_equipo = id_equipo_local
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) OR EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE NEW.id_equipo = id_equipo_visitante
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar un entrenamiento dentro de las 2 horas anteriores o posteriores a un partido.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_horario_entrenamiento_insert BEFORE INSERT ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_horario_entrenamiento();
CREATE TRIGGER validar_horario_entrenamiento_update BEFORE UPDATE ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_horario_entrenamiento();
```

- La función **validar_partidos_mismo_equipo**, que comprueba que cada equipo de el partido no tenga otro partido en las 2 horas anteriores y posteriores.

```sql
-- Disparador que valida la hora de inicio entre partidos del mismo equipo
CREATE OR REPLACE FUNCTION validar_partidos_mismo_equipo()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE id_equipo_local = NEW.id_equipo_local AND id_partido != NEW.id_partido
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) OR EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE id_equipo_local = NEW.id_equipo_visitante AND id_partido != NEW.id_partido
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) or EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE id_equipo_visitante = NEW.id_equipo_local AND id_partido != NEW.id_partido
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) OR EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE id_equipo_visitante = NEW.id_equipo_visitante AND id_partido != NEW.id_partido
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar un partido dentro de las 2 horas anteriores o posteriores a otro partido del mismo equipo.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;
```

```sql
CREATE TRIGGER validar_partidos_mismo_equipo_insert BEFORE INSERT ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_partidos_mismo_equipo();
CREATE TRIGGER validar_partidos_mismo_equipo_update BEFORE UPDATE ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_partidos_mismo_equipo();
```

- La función **validar_entrenamientos_mismo_equipo**, similar a la función anterior pero comprobando que el equipo no tenga otro entrenamiento en las 2 horas anteriores y posteriores..

```sql
-- Disparador que valida la hora de inicio en un entrenamiento
CREATE OR REPLACE FUNCTION validar_entrenamientos_mismo_equipo()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE NEW.id_equipo = id_equipo AND id_training != NEW.id_training
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar un entrenamiento dentro de las 2 horas anteriores o posteriores a un entrenamiento del mismo equipo.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_entrenamientos_mismo_equipo_insert BEFORE INSERT ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_entrenamientos_mismo_equipo();
CREATE TRIGGER validar_entrenamientos_mismo_equipo_update BEFORE UPDATE ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_entrenamientos_mismo_equipo();
```

5.4.5. Validaciones de lugar

Se han realizado 4 triggers empleando 2 funciones que validan los lugares de los partidos y entrenamientos:

- La función **validar_lugar_partidos**, que comprueba que cada estadio no tenga otro partido en las 2 horas anteriores y posteriores.

```sql
-- Disparador que valida el lugar en un partido
CREATE OR REPLACE FUNCTION validar_lugar_partido()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM PARTIDO
    WHERE id_estadio = NEW.id_estadio
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar el partido, el estadio está ocupado las 2 horas anteriores o posteriores.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_lugar_partido_insert BEFORE INSERT ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_lugar_partido();
CREATE TRIGGER validar_lugar_partido_update BEFORE UPDATE ON PARTIDO FOR EACH ROW EXECUTE FUNCTION validar_lugar_partido();
```

- La función **validar_lugar_entrenamientos**, similar a la función anterior pero comprobando que el lugar de entrenamiento no tenga otro entrenamiento en las 2 horas anteriores y posteriores..

```sql
-- Disparador que valida el lugar en un entrenamiento de un equipo
CREATE OR REPLACE FUNCTION validar_lugar_entrenamiento()
RETURNS TRIGGER AS $$
BEGIN
  IF EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE NEW.lugar = lugar
      AND ((fecha >= NEW.fecha AND fecha <= NEW.fecha + INTERVAL '2 hours') OR (fecha <= NEW.fecha AND fecha >= NEW.fecha - INTERVAL '2 hours'))
  ) THEN
    RAISE EXCEPTION 'No se puede programar un entrenamiento, el lugar está ocupado las 2 horas anteriores o posteriores.';
  END IF;

  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER validar_lugar_entrenamiento_insert BEFORE INSERT ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_lugar_entrenamiento();
CREATE TRIGGER validar_lugar_entrenamiento_update BEFORE UPDATE ON ENTRENAMIENTO FOR EACH ROW EXECUTE FUNCTION validar_lugar_entrenamiento();
```

5.4.6. Herencia de jugadores

Se han realizado tres trigger que al introducir un jugador, este crea un registro en otra tabla (según su posición) inicializando a 0 el atributo de dicha tabla.

- El trigger **insertar_jugador**, que crea el registro en otra tabla según la posición del jugador.

```
-- Disparador que añade jugadores a las respectivas tablas de las posiciones
CREATE OR REPLACE FUNCTION insertar_jugador()
RETURNS TRIGGER AS $$
BEGIN
  IF NEW.posicion = 'Portero' THEN
    INSERT INTO PORTERO(id_jugador, paradas) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Defensa' THEN
    INSERT INTO DEFENSA(id_jugador, balones_robados) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Centrocampista' THEN
    INSERT INTO CENTROCAMPISTA(id_jugador, regates_realizados) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Delantero' THEN
    INSERT INTO DELANTERO(id_jugador, disparos_a_puerta) VALUES (NEW.id_jugador, 0);
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER insertar_jugador AFTER INSERT ON JUGADOR FOR EACH ROW EXECUTE FUNCTION insertar_jugador();
```

- El trigger **eliminar_jugador**, que elimina el registro en la correspondiente tabla según la posición del jugador.

```
-- Disparador que elimina jugadores a las respectivas tablas de las posiciones
CREATE OR REPLACE FUNCTION eliminar_jugador()
RETURNS TRIGGER AS $$
BEGIN
  IF OLD.posicion = 'Portero' THEN
    DELETE FROM PORTERO WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Defensa' THEN
    DELETE FROM DEFENSA WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Centrocampista' THEN
    DELETE FROM CENTROCAMPISTA WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Delantero' THEN
    DELETE FROM DELANTERO WHERE id_jugador = OLD.id_jugador;
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER eliminar_jugador AFTER DELETE ON JUGADOR FOR EACH ROW EXECUTE FUNCTION eliminar_jugador();
```

- El trigger **eliminar_jugador**, que elimina el registro en la correspondiente tabla según la posición del jugador.

```sql
-- Disparador que actualiza jugadores a las respectivas tablas de las posiciones
CREATE OR REPLACE FUNCTION actualizar_jugador()
RETURNS TRIGGER AS $$
BEGIN
  IF OLD.posicion = 'Portero' THEN
    DELETE FROM PORTERO WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Defensa' THEN
    DELETE FROM DEFENSA WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Centrocampista' THEN
    DELETE FROM CENTROCAMPISTA WHERE id_jugador = OLD.id_jugador;
  ELSIF OLD.posicion = 'Delantero' THEN
    DELETE FROM DELANTERO WHERE id_jugador = OLD.id_jugador;
  END IF;

  IF NEW.posicion = 'Portero' THEN
    INSERT INTO PORTERO(id_jugador, paradas) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Defensa' THEN
    INSERT INTO DEFENSA(id_jugador, balones_robados) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Centrocampista' THEN
    INSERT INTO CENTROCAMPISTA(id_jugador, regates_realizados) VALUES (NEW.id_jugador, 0);
  ELSIF NEW.posicion = 'Delantero' THEN
    INSERT INTO DELANTERO(id_jugador, disparos_a_puerta) VALUES (NEW.id_jugador, 0);
  END IF;
  RETURN NEW;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER actualizar_jugador AFTER DELETE ON JUGADOR FOR EACH ROW EXECUTE FUNCTION actualizar_jugador();
```

5.4.7. Inclusividad

Para la relación de inclusividad se ha realizado un trigger que comprueba que exista entrenamientos para ambos equipos de un partido. Si alguno de los equipos no tiene un entrenamiento anterior, **inclusividad_equipo** lanzará una excepción indicando cuál.

```sql
-- Disparador que comprueba la inclusividad (es necesario un entrenamiento para jugar un partido)
CREATE OR REPLACE FUNCTION inclusividad_equipo()
RETURNS TRIGGER AS $$
BEGIN
  IF NOT EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE NEW.id_equipo_local = id_equipo AND (fecha < NEW.fecha)
  )  THEN
    RAISE EXCEPTION 'No se puede programar un partido mientras el equipo local no haya entrenado entrenamientos.';
  END IF;

  IF NOT EXISTS (
    SELECT 1
    FROM ENTRENAMIENTO
    WHERE NEW.id_equipo_VISITANTE = id_equipo AND (fecha < NEW.fecha)
  ) THEN
    RAISE EXCEPTION 'No se puede programar un partido mientras el equipo visitante no haya entrenado entrenamientos.';
  END IF;
END;
$$ LANGUAGE plpgsql;

CREATE TRIGGER inclusividad_equipo BEFORE INSERT ON PARTIDO FOR EACH ROW EXECUTE FUNCTION inclusividad_equipo();
```

6. Ejemplos de consultas

Para probar la correcta carga de los datos y el funcionamiento de las distintas reglas que hemos establecido, vamos a realizar distintas consultas de ejemplo.

6.1. SELECT sobre la tabla JUGADORES

En este caso, vamos a intentar obtener el nombre y apellidos de todos los jugadores que jueguen en el Raimon que sean del elemento Fuego.

Query | Query History

```
1  SELECT J.NOMBRE, J.APELLIDOS FROM PUBLIC.JUGADOR J
2  INNER JOIN PUBLIC.EQUIPO E ON J.ID_EQUIPO = E.ID_EQUIPO
3  WHERE J.ELEMENTO = 'Fuego' AND E.NOMBRE = 'Raimon';
```

Data Output | Messages | Notifications

	nombre character varying (10)	apellidos character varying (20)
1	Tod	Ironside
2	Sam	Kincaid
3	Axel	Blaze

6.2. UPDATE sobre la tabla PARTIDO

A continuación vamos a cambiar los goles de un partido, haciendo que un equipo pase a ser el ganador y, por consecuencia, cambie su atributo de goles a favor y en contra además de sus victorias.

Query | Query History

```
1 SELECT * FROM PUBLIC.EQUIPO
2 WHERE ID_EQUIPO = 1 OR ID_EQUIPO = 2
3 ORDER BY ID_EQUIPO ASC;
```

Data Output | Messages | Notifications

	id_equipo [PK] integer	nombre character varying (20)	pais character varying (20)	victorias integer	goles_a_favor integer	goles_en_contra integer
1	1	Raimon	Japón	9	26	36
2	2	Royal Academy	Japón	1	21	3

Para este ejemplo, vamos a cambiar uno de los partidos entre el Raimon y la Royal Academy. Vamos a cambiar el resultado de 1-2 a 2-1, haciendo que el Raimon reste una victoria y la Royal Academy sume una. Además, el número de goles cambiará de igual forma a las victorias.

Query | Query History

```
2 SET GOLES_LOCAL = 2,
3 GOLES_VISITANTE = 1
4 WHERE ID_PARTIDO = 6;
```

Data Output | Messages | Notifications

```
UPDATE 1

Query returned successfully in 70 msec.
```

Query | Query History

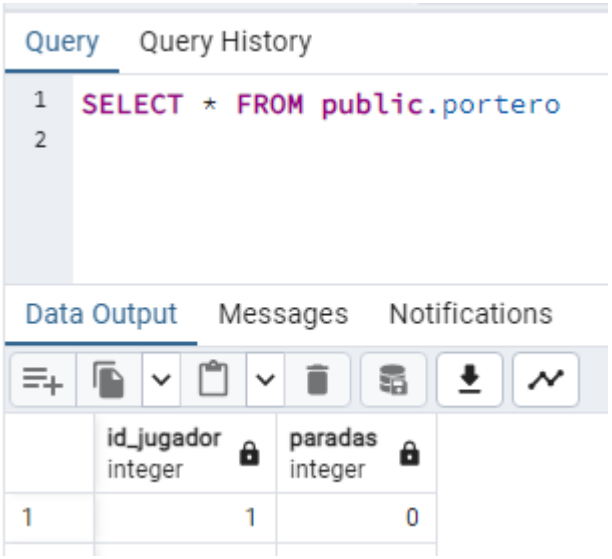
```
1 SELECT * FROM PUBLIC.EQUIPO
2 WHERE ID_EQUIPO = 1 OR ID_EQUIPO = 2;
```

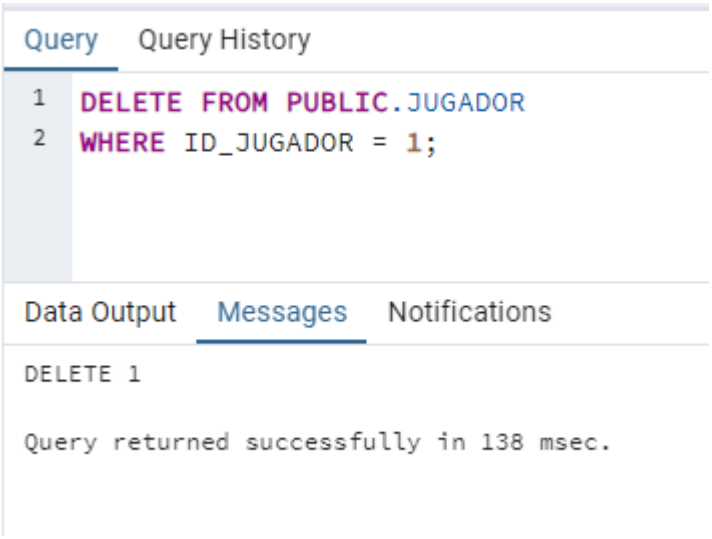
Data Output | Messages | Notifications

	id_equipo [PK] integer	nombre character varying (20)	pais character varying (20)	victorias integer	goles_a_favor integer	goles_en_contra integer
1	1	Raimon	Japón	8	25	37
2	2	Royal Academy	Japón	2	22	2

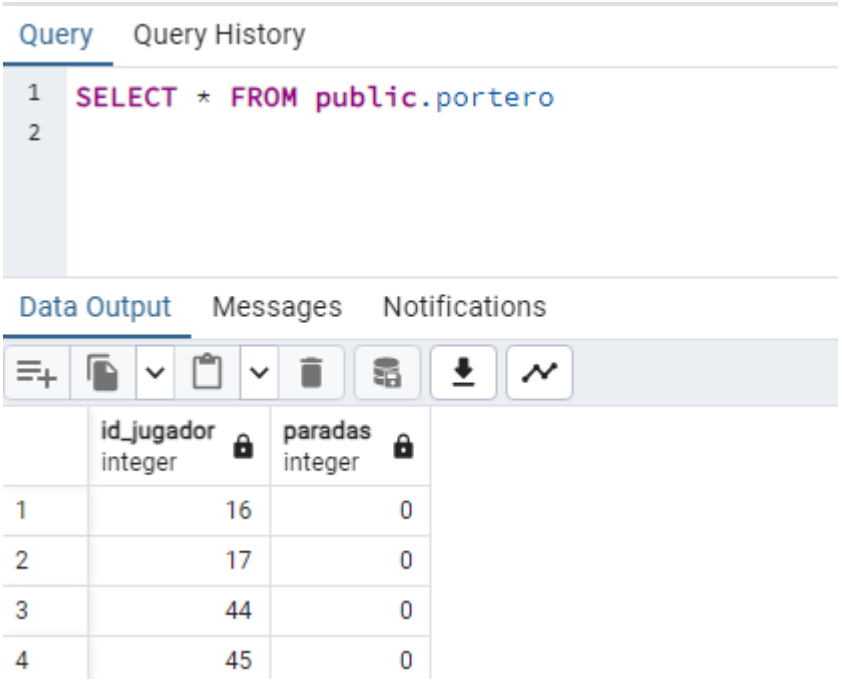
6.3. DELETE sobre la tabla JUGADOR

Ya que, por jerarquía, cada jugador está representado por su posición en una tabla dedicada a esta, vamos a probar eliminar un jugador de su tabla principal para ver cómo se elimina de la tabla correspondiente a su posición. En específico, vamos a borrar a Mark Evans, el jugador con identificador 1, un portero.



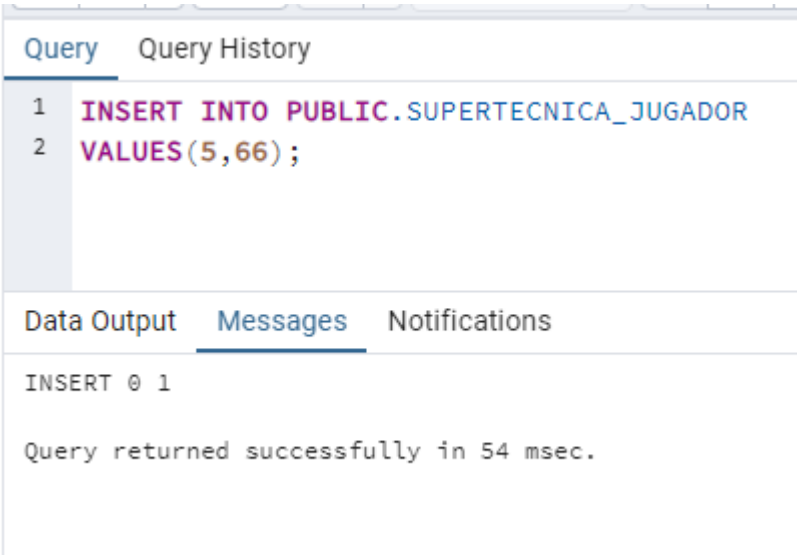


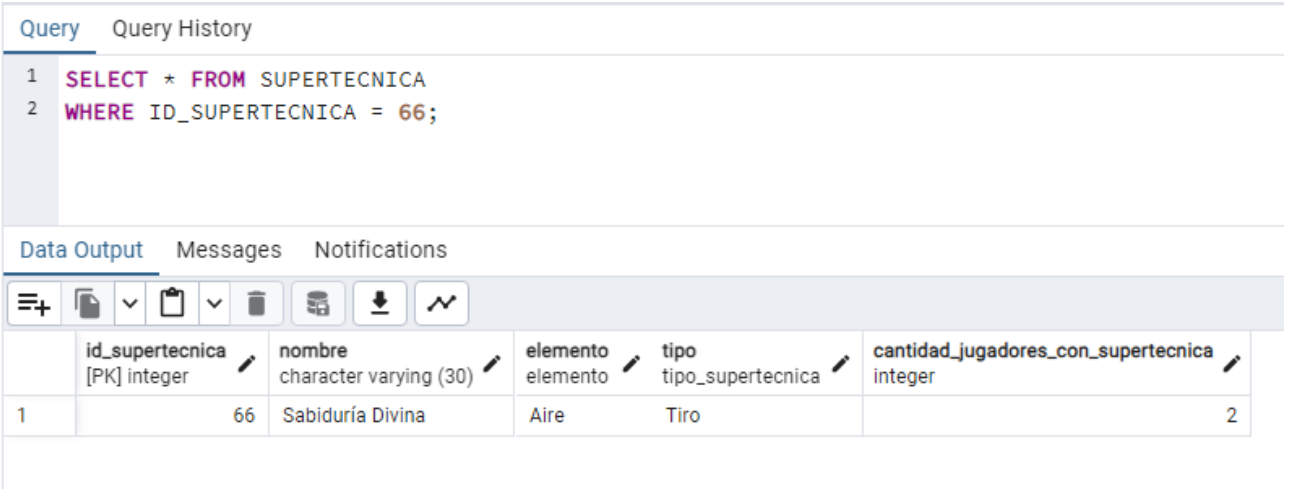
Una vez borrado, vamos a comprobar que se ha eliminado la entrada en la tabla portero que indicaba que el jugador 1 era de este tipo.



6.4. INSERT sobre la tabla SUPERTECNICA_JUGADOR

Finalmente, vamos a insertar una nueva relación entre jugador y supertécnica. De esta forma podemos comprobar cómo la columna de jugadores con dicha supertécnica aumenta en 1. Por ejemplo, vamos a darle al jugador Tod Ironside la supertécnica Sabiduría Divina., supertécnica que sólo posee una persona.



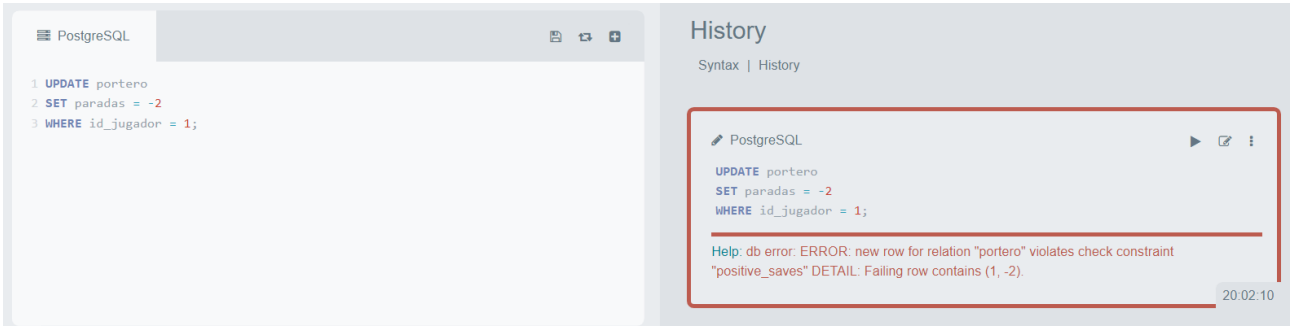


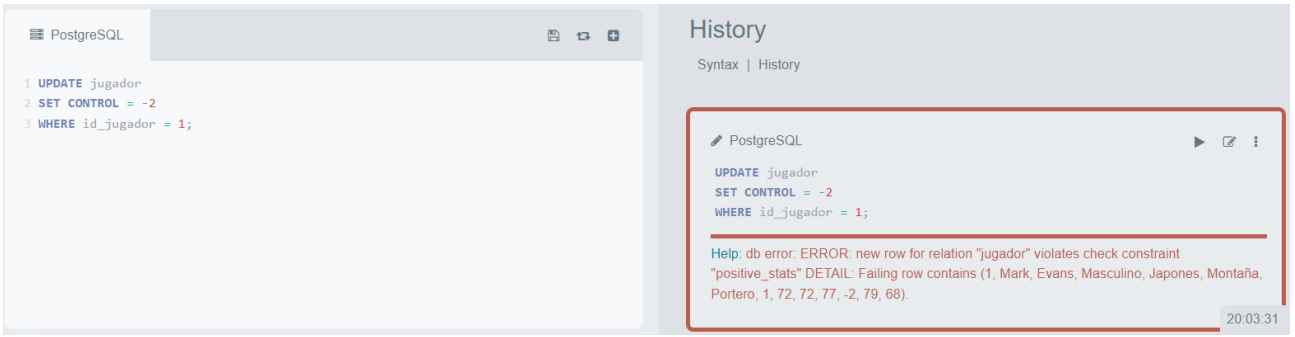
Podemos ver como la cantidad de jugadores con esta supertécnica ha subido a 2.

6.5. Comprobación de checks

6.5.1. Partido entre mismos equipos

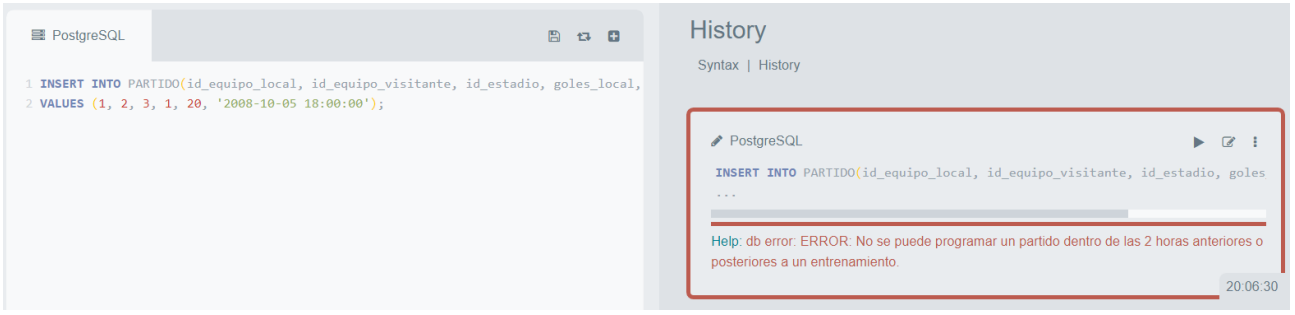
6.5.2. Valores Positivos



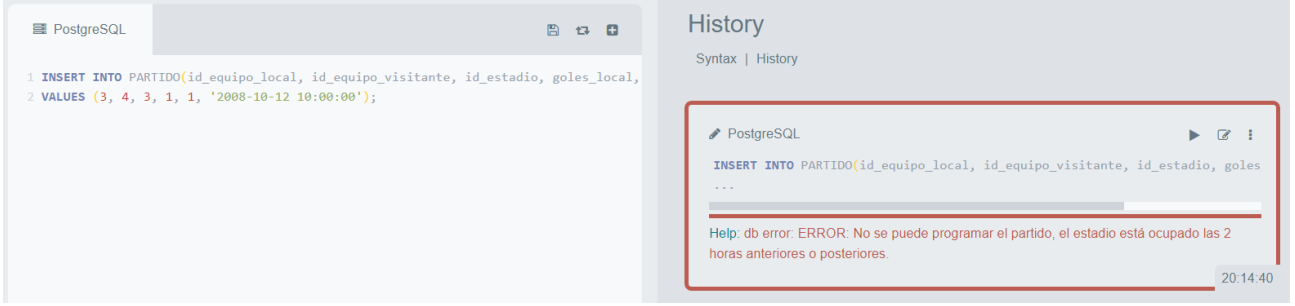


6.6. Comprobación de triggers.

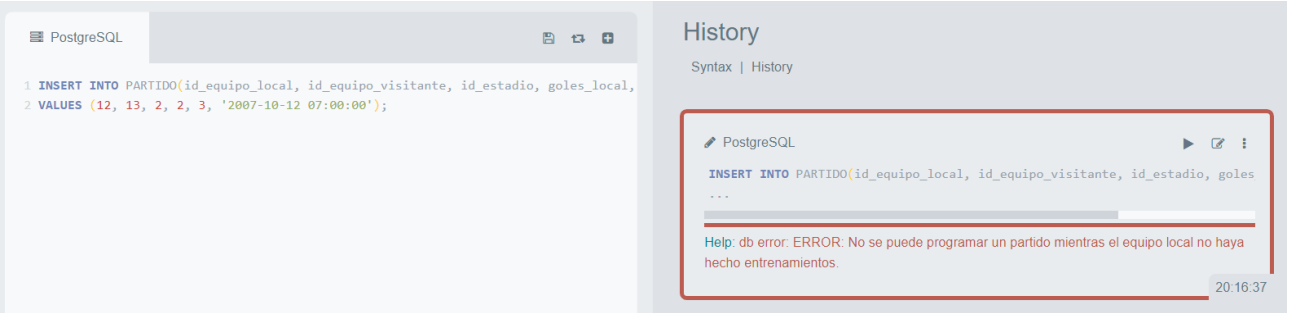
6.6.1. Partido cerca de entrenamiento de un equipo



6.6.2. Partidos cerca en el mismo estadio



6.6.3. Partidos sin realizar entrenamientos



7. Implementación API REST

7.1. Operaciones CRUD

En nuestra API se aplican operaciones CRUD a las tablas Jugador, Equipo, Estadio, Supertécnica, Partido, Entrenamiento y Supertécnica Jugador. Mientras que en las tablas de Portero, Delantero, Centrocampista y Defensa solo se permiten la operación de actualización y de obtención de datos.

Para poder consumir la API lo primero que se debe hacer es ejecutar la API como tal, usando el siguiente comando se ejecuta:

flask --app app run --host 0.0.0.0 --port 8080

Posteriormente, a través de cualquier navegador o usando extensiones de visual studio code como Thunder Client. Podemos usar el link que nos da el API y se muestra la siguiente:

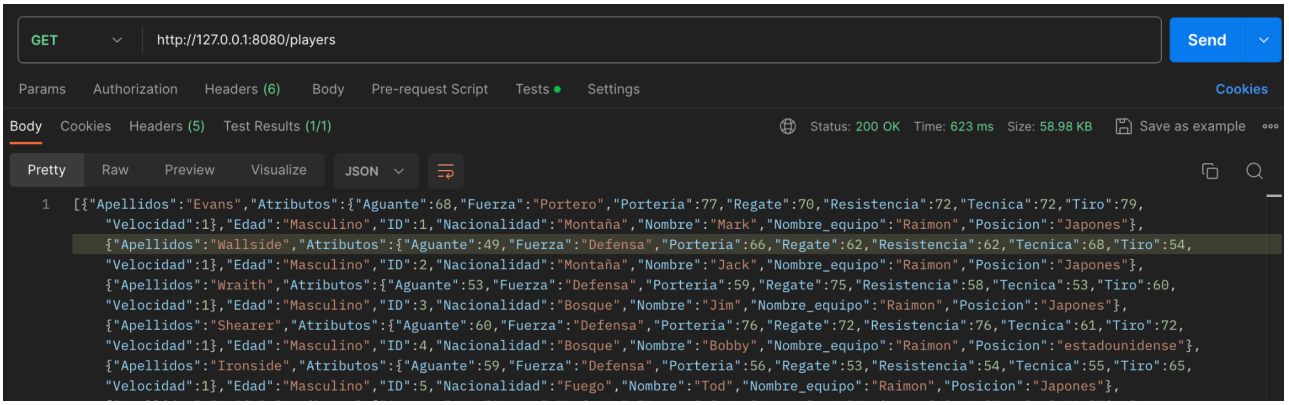
En cada uno de los apartados de header si hacemos click sobre alguno de ellos nos muestra las operaciones get sobre algunas de las tablas de la base de datos en formato JSON.

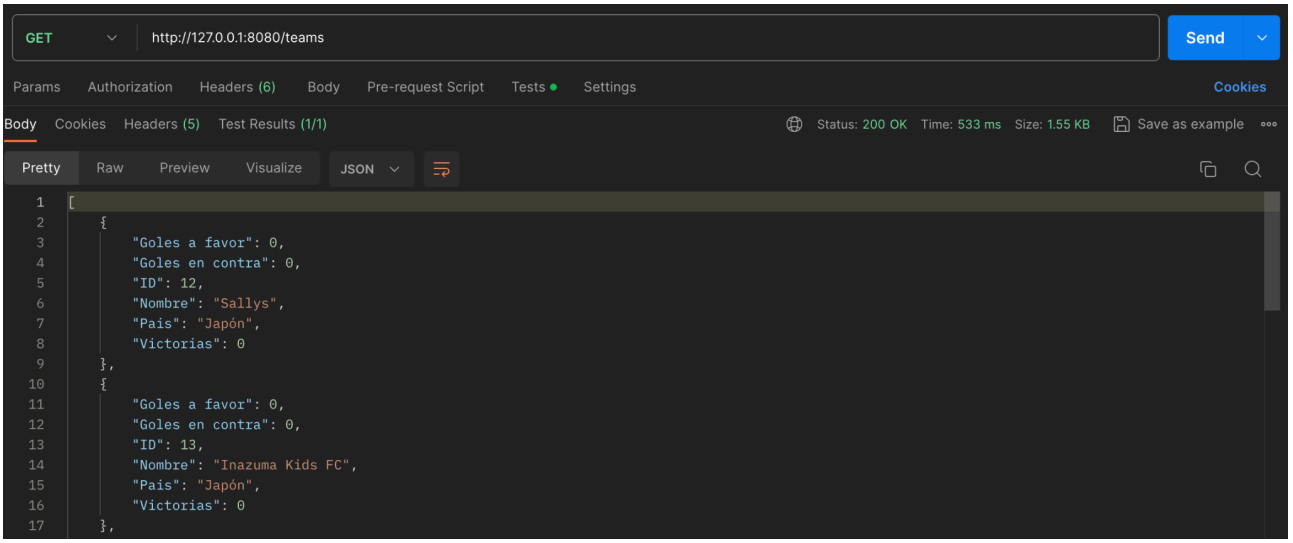
Ahora voy a mostrar los endpoints de nuestra API, sobre qué tablas se ejecutan dichos endpoints y qué operaciones podemos realizar:

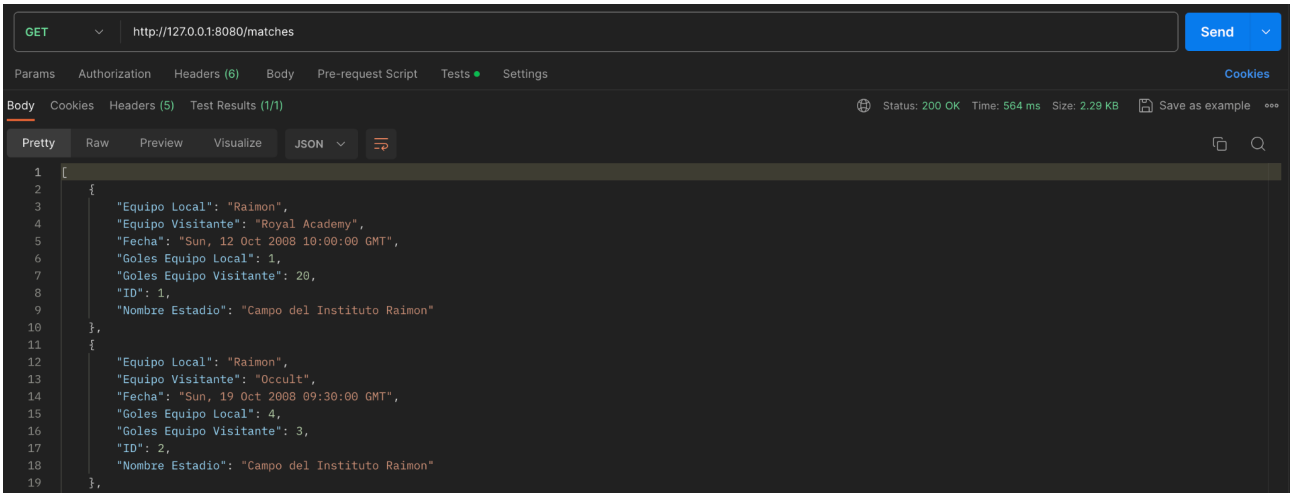
Endpoint	Tablas	Operaciones
/players/	JUGADOR	GET, POST, PUT, DELETE
/teams/	EQUIPO	GET, POST, PUT, DELETE
/stadiums/	ESTADIO	GET, POST, PUT, DELETE
/special_moves/	SUPERTECNICA	GET, POST, PUT, DELETE
/matches/	PARTIDO	GET, POST, PUT, DELETE
/trainings/	ENTRENAMIENTO	GET, POST, PUT, DELETE
/special_move_for_player/	SUPERTECNICA_JUGADOR	GET, POST, PUT, DELETE
/goalkeeper/	PORTERO	GET, PUT
/defense/	DEFENSA	GET, PUT
/midfielder/	CENTROCAMPISTA	GET, PUT
/forward/	DELANTERO	GET, PUT

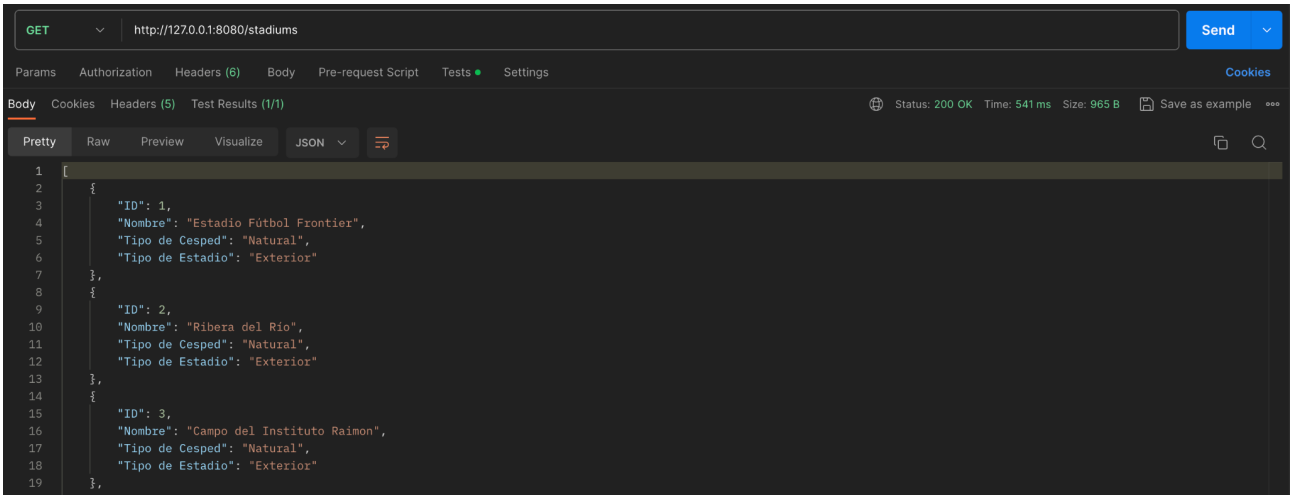
7.2. Pruebas de la API

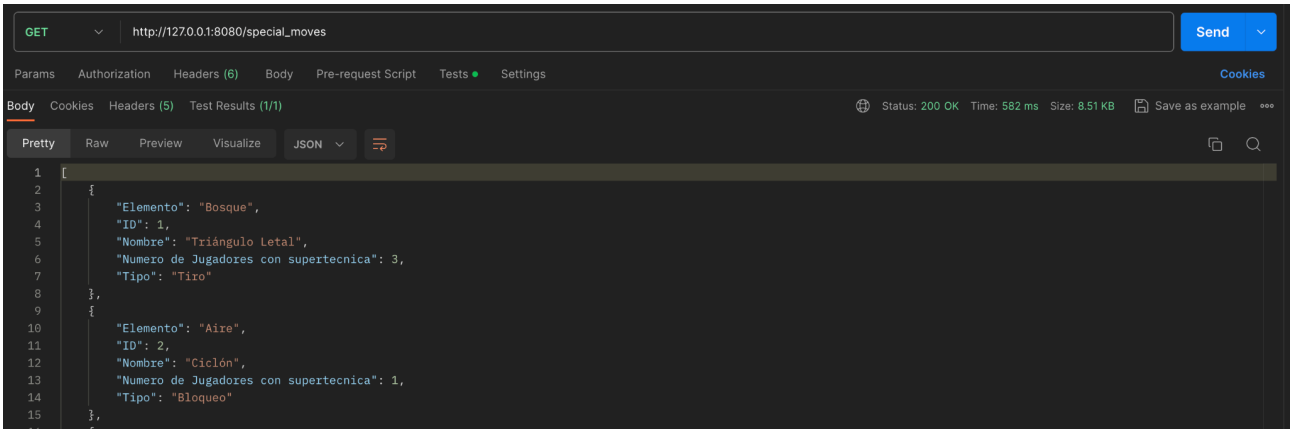
7.2.1. GET

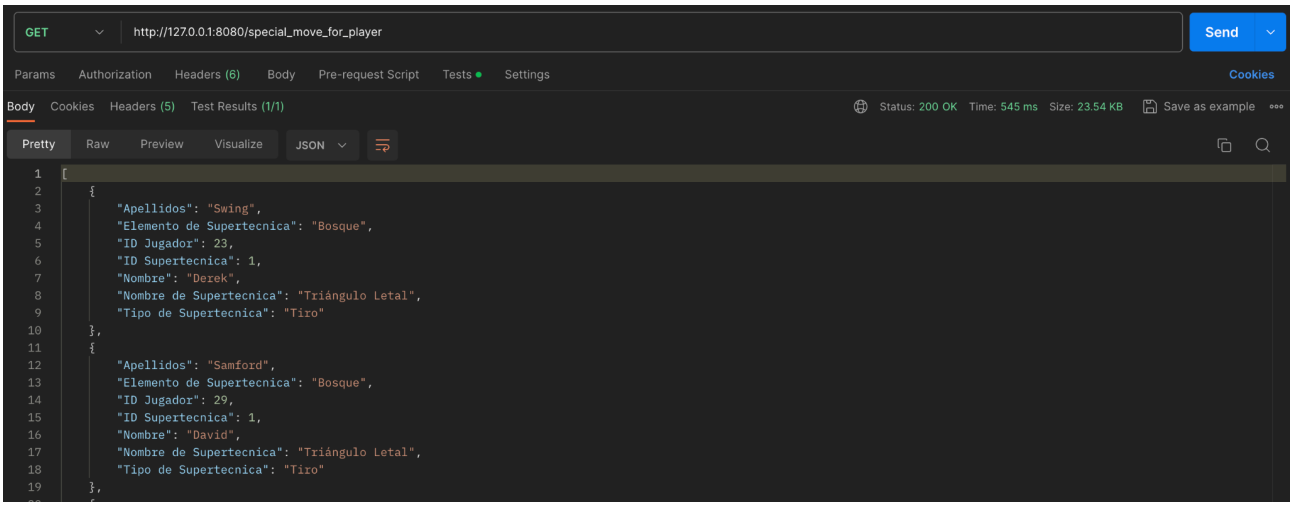








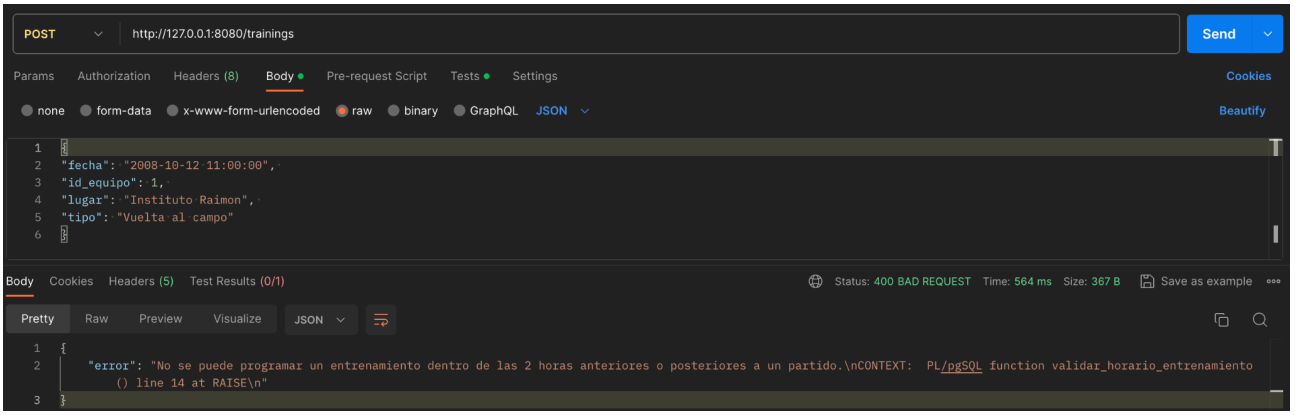


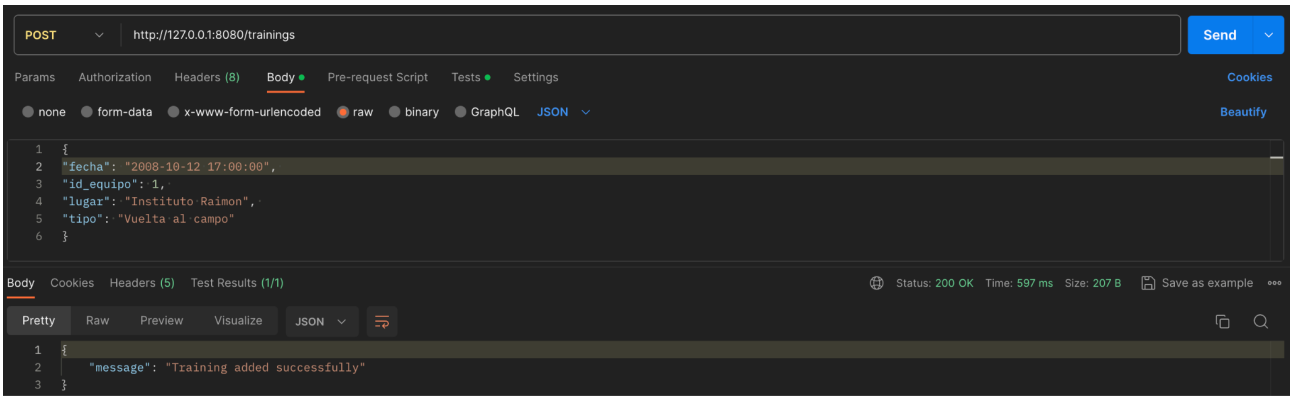


En los últimos cuatro endpoints nos encontramos con un resultado similar al endpoint de “/players/”, pero se muestran los jugadores que sean porteros, defensa, mediocampista o delanteros según el endpoint elegido.

7.2.2. POST

En este apartado se van a realizar algunos POST de ejemplo enseñando algunos errores de disparadores y haciendo la misma inserción sin que el disparador falle. En esta primera prueba se va a probar si el disparador que no permite que un entrenamiento se programa dentro de las 2 horas anteriores o posteriores a un partido funciona.

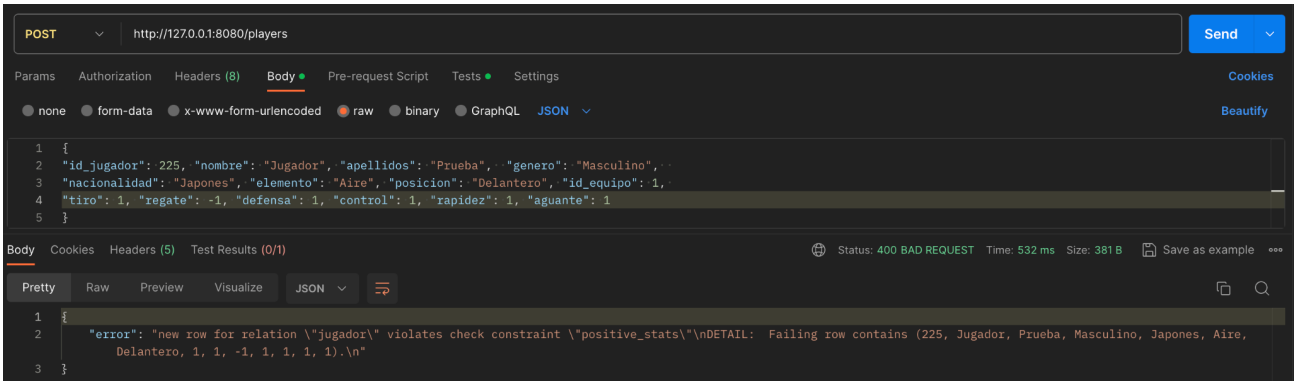




Como se puede observar en la prueba anterior a las 11 no podemos añadir un entrenamiento, mientras que a las 17:00 si podemos.

Existen 3 disparadores que hacen lo mismo, pero se encargan de comprobar si hay un entrenamiento cercano a otro entrenamiento, partido cercano a otro partido y partido cercano a un entrenamiento.

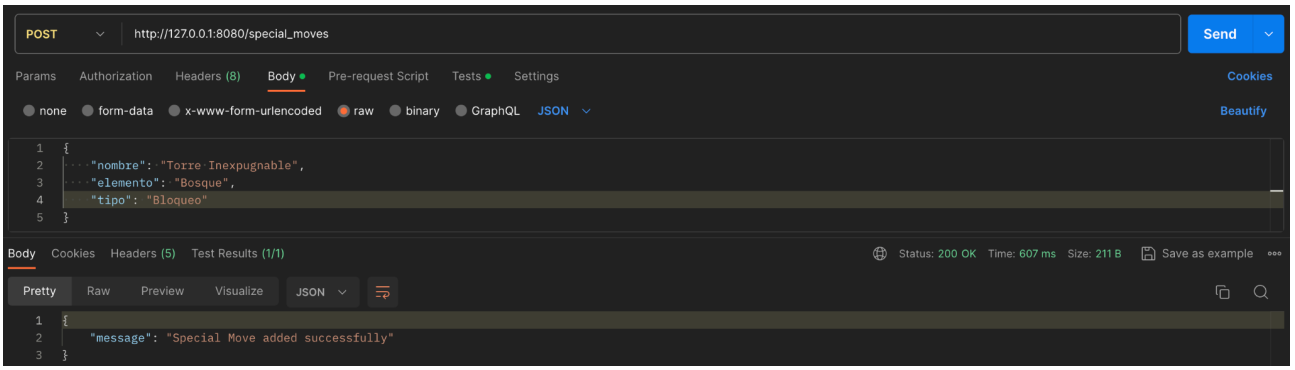
Si se intenta añadir un jugador o un equipo y se añade algún valor negativo no se permite insertar ese jugador o equipo a su respectiva tabla.



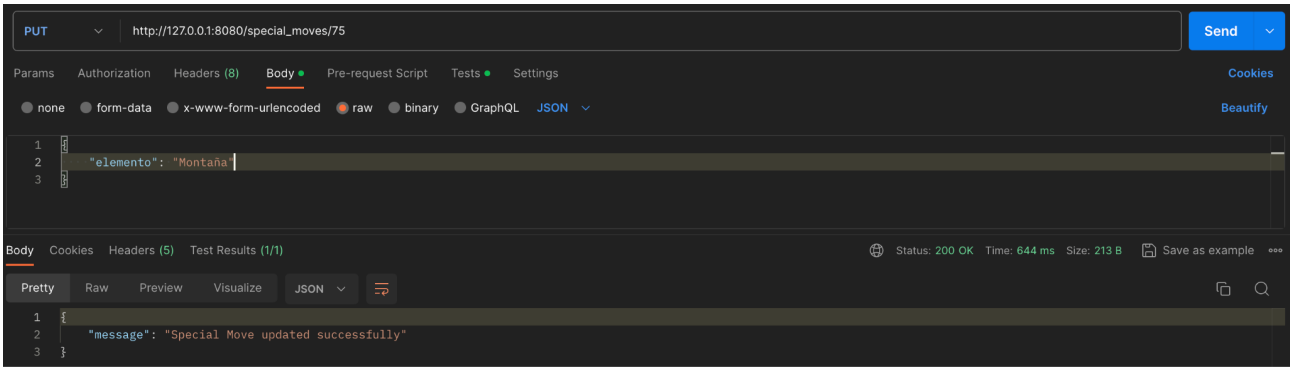
7.2.3. PUT

En todas las tablas podemos realizar la operación PUT con todos los atributos que queramos menos el identificador.

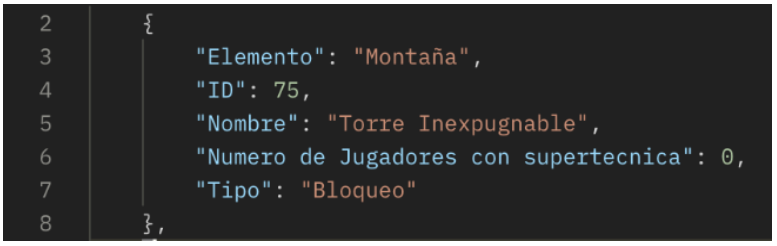
Como ejemplo voy a añadir la siguiente supertecnica.



Después se modifica el campo elemento.



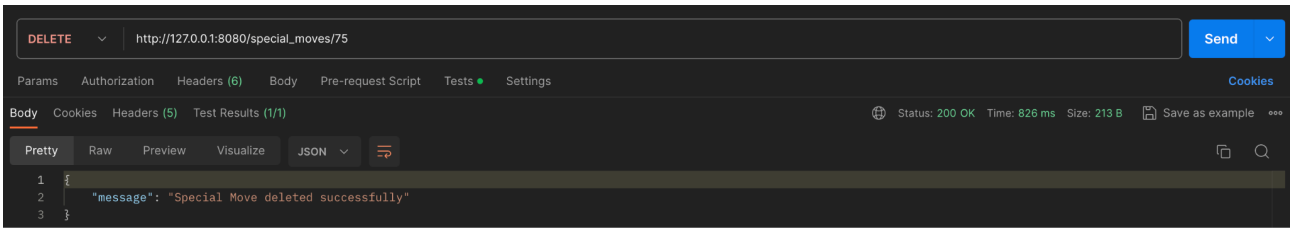
Si realizamos un get podemos ver el campo que se ha modificado.



7.2.4. DELETE

La operación DELETE la podemos realizar en todas las tablas menos en las tablas de Portero, Delantero, Defensa y Centrocampista.

En nuestro ejemplo se va a eliminar la entrada que se añadió a la tabla Supertecnica en el ejemplo anterior.



8. Conclusiones

Para concluir nos gustaría decir que nos ha parecido un proyecto muy interesante ya que nos permite ver como es el desarrollo de una API en PostgreSQL desde cero. Empezando por diagramas entidad-relación hasta conseguir una API funciona usando Flask.

Este proyecto nos ha permitido ver que con un diseño basado en un videojuego se puede conseguir un resultado bastante bueno y original, aunque no sea nada innovador nos permite conocer las tecnologías que se pueden utilizar para este tipo de proyectos con una base de datos relativamente sencilla.

9. Referencias

1. Directorio de GitHub