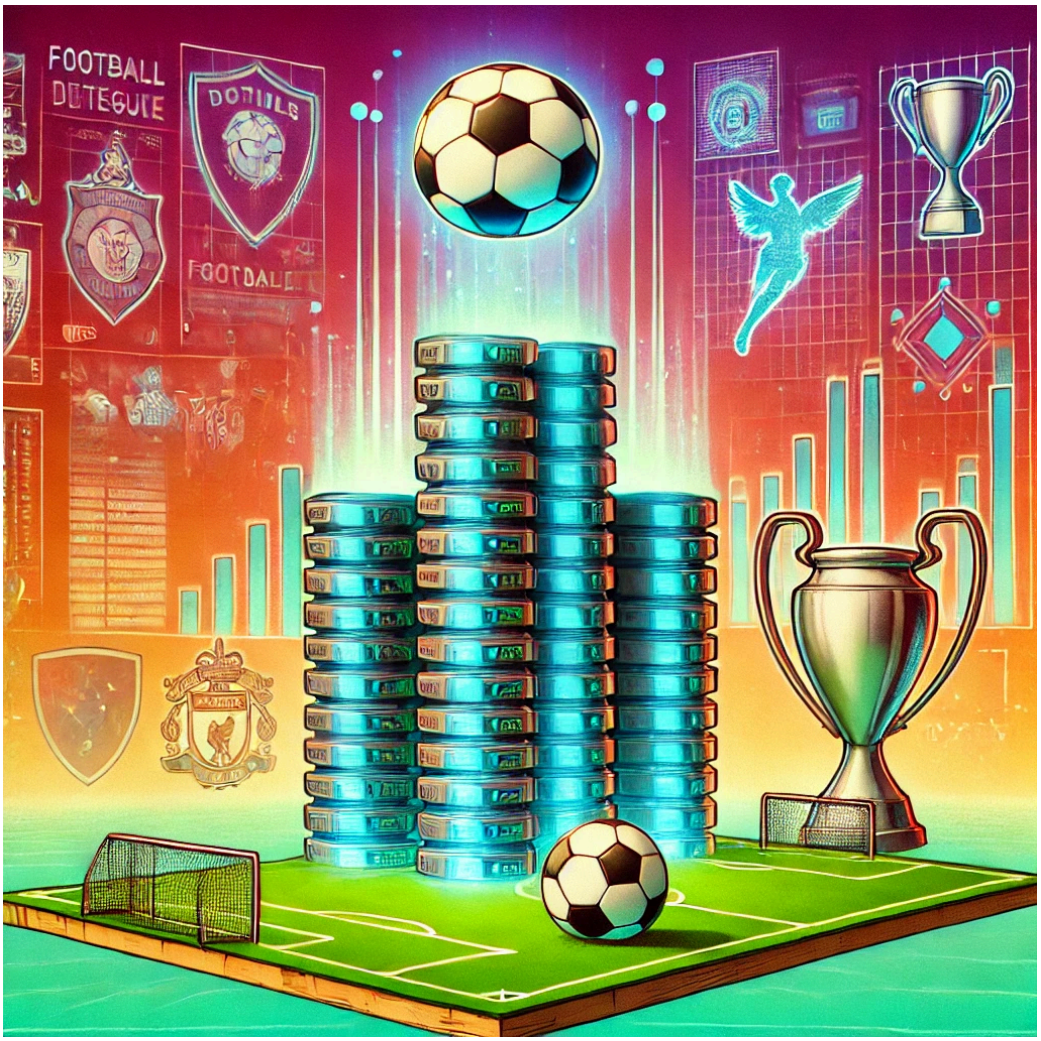
BASES DE DATOS

CURSO 2024-2025

Memoria Primera Base de Datos: Liga de fútbol



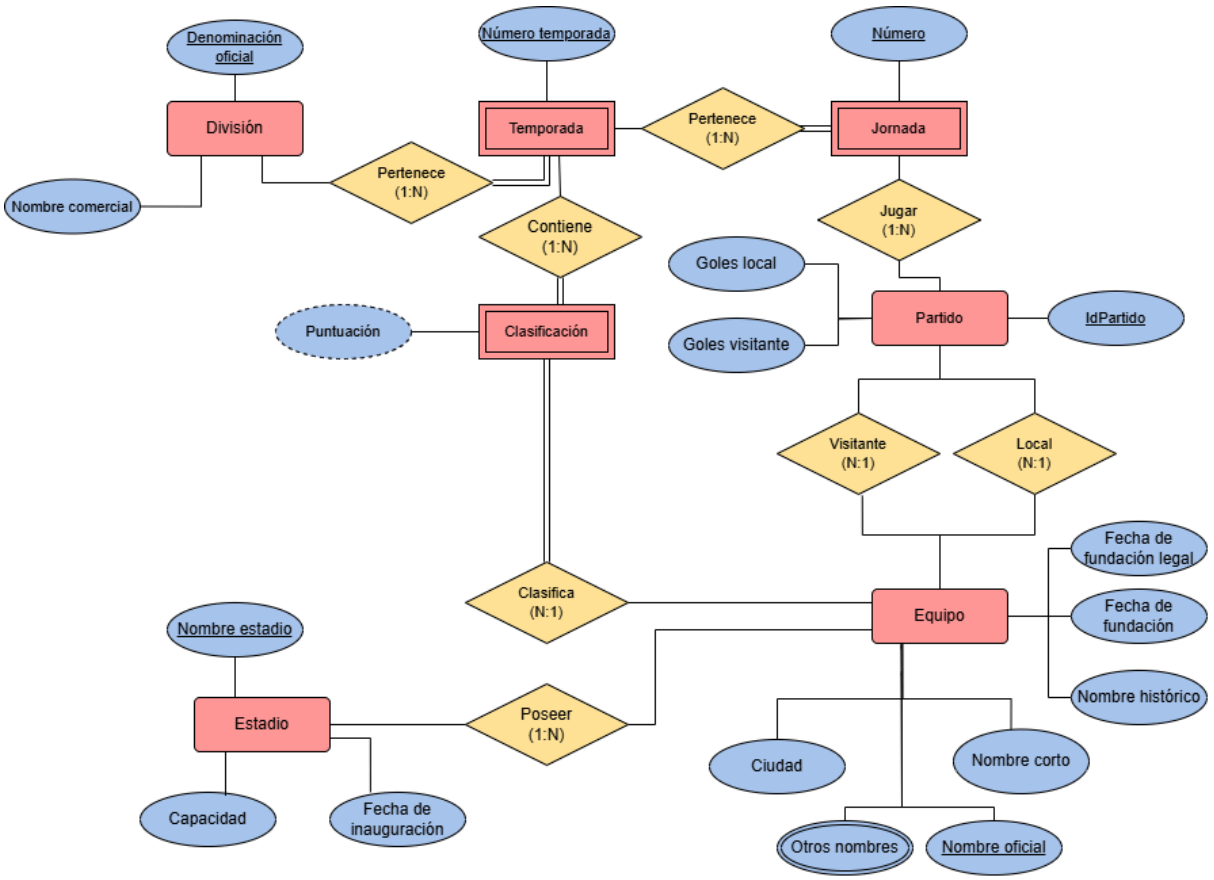
Grupo: J2.10
Fecha de entrega: 14/03/2025
Autores:

- Fernando Pastor Peralta (897113) .....897113@unizar.es
- Juan José Muñoz Lahoz (902677) .....902677@unizar.es
- Marcos San Julián Fuertes (899849) .....899849@unizar.es

ÍNDICE:

Parte 1: Creación de una base de datos

1.1. Esquema E/R global y listado de las restricciones

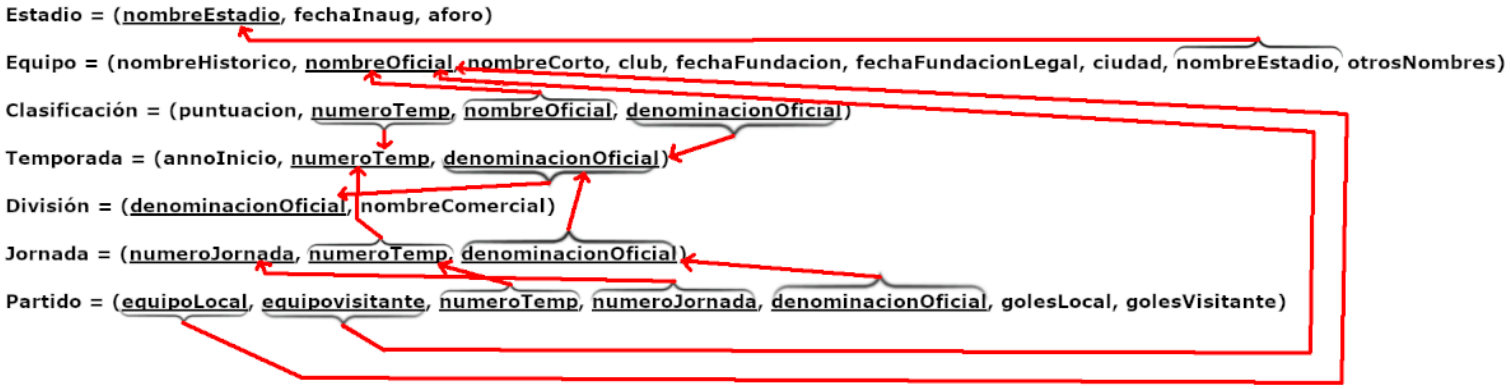


Restricciones:
- No se puede enfrentar un equipo contra sí mismo.
- No se podrá insertar un partido que se juegue en una temporada cuyo año inicial no haya llegado en el momento de inserción.
- La puntuación se calculará como (partidos ganados*3 + partidos empatados).
- Los puestos se podrán consultar mediante una consulta SQL.
- Un equipo solo puede tener un estadio.
- No se pueden meter temporadas con un año menor a 1972.
+ restricciones en el apartado 3.2.

Otras opciones alternativas para la creación del esquema E/R:
- Eliminar la entidad Estadio y poner sus atributos como atributos de la entidad Equipo.
- Eliminar la entidad Clasificación e implementarla como atributo a la relación entre Jornada y Equipo.
- La entidad Clasificación depende de la jornada y no de la temporada.

1.2. Esquema relacional correspondiente al E/R y su normalización

El modelo relacional de nuestra base de datos quedaría de la siguiente forma:

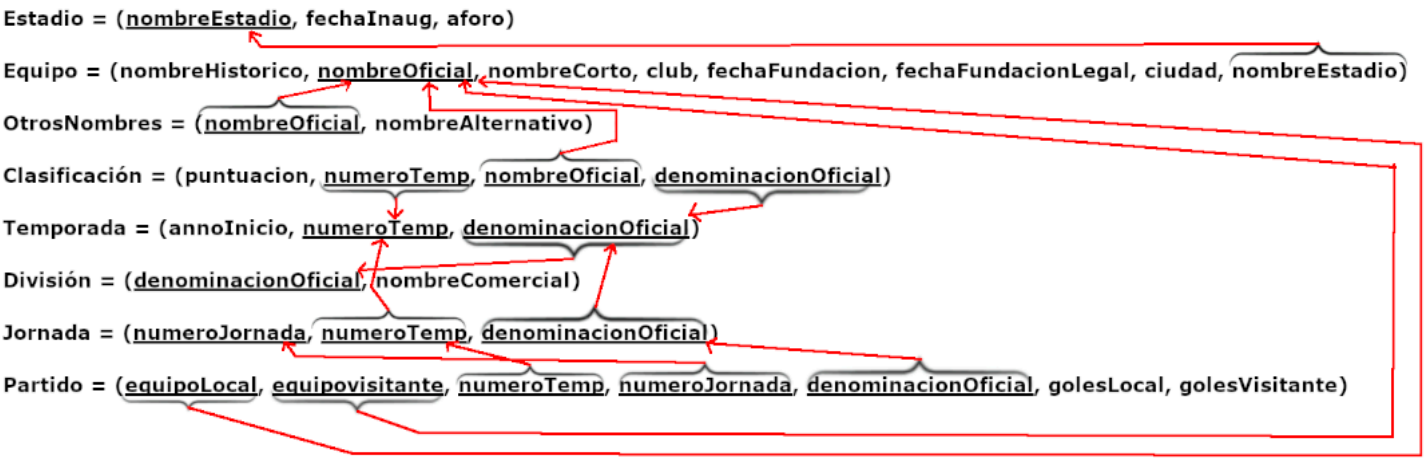


Podemos ver que, de primeras, este esquema de modelo relacional no está normalizado: en la entidad equipo existe un atributo multivaluado llamado otrosNombres. Para normalizar nuestro esquema, hay que deshacerse de ese atributo. Para ello crearemos otra entidad con su mismo nombre, cuyos atributos son nombreOficial como clave primaria y ajena de Equipo, y nombreAlternativo, que indica el nombre alternativo que dicho equipo posee.

Una vez hecho esto, ya tendremos nuestro esquema en 1ª forma normal. Ahora bien, este ha de estar en la Forma Normal de Boyce-Codd, entonces vamos a ver qué hay que modificar para que se cumpla:

- Segunda Forma Normal: : 1FN + No hay dependencias funcionales con parte de la clave.
  Esto se cumple en nuestra base de datos ya que no existe ninguna entidad cuya clave primaria dependa de tan solo una parte de la clave primaria de otra entidad.
- Tercera Forma Normal: 2FN + No hay dependencias funcionales transitivas. Esto se cumple en nuestra base de datos ya que en ninguna tabla existen atributos que dependen de otro atributo que no es clave primaria. Este sería el caso si, por ejemplo, en vez de tener la entidad estadio, guardamos en la tabla de Equipo, en cada entrada de equipo los datos de su estadio, potencialmente repitiendo la información en la base de datos.
- Forma Normal de Boyce-Codd: Todas las dependencias funcionales dependen de la clave. Esto se cumple en nuestra base de datos, ya que todas las entidades cuya clave depende de la clave de otra entidad, siempre dependen de la primaria entera, ni no primaria ni de forma parcial.

Al normalizar la base de datos, el esquema relacional quedaría de la siguiente forma:



1.3. Sentencias SQL de creación de tablas

```
/* Creacion de las tablas */
CREATE TABLE Estadio (
	nombreEstadio		VARCHAR(50) PRIMARY KEY,
	fechaInaug		NUMBER,
	aforo			NUMBER
);

CREATE TABLE Equipo (
	nombreHistorico		VARCHAR(70),
	nombreOficial		VARCHAR(30) PRIMARY KEY,
	nombreCorto		VARCHAR(30),
	fechaFundacion		NUMBER,
	fechaFundacionLegal	NUMBER,
	ciudad			VARCHAR(50),
	nombreEstadio		VARCHAR(30),
	FOREIGN KEY (nombreEstadio) REFERENCES Estadio(nombreEstadio)
);

CREATE TABLE OtrosNombres (
	nombreOficial		VARCHAR(30) PRIMARY KEY,
	nombreAlternativo	VARCHAR(30),
	FOREIGN KEY (nombreOficial) REFERENCES Equipo(nombreOficial)
);

CREATE TABLE Division (
	denominacionOficial	VARCHAR(10) PRIMARY KEY,
	nombreComercial		VARCHAR(70)
);

CREATE TABLE Temporada (
	annoInicio		NUMBER,
	numeroTemp		NUMBER, -- annoInicio - 1972 (año inicio de la
primera temporada)
	denominacionOficial VARCHAR(10),
	PRIMARY KEY (numeroTemp, denominacionOficial),
	FOREIGN KEY (denominacionOficial) REFERENCES Division(denominacionOficial)
);

CREATE TABLE Clasificacion (
	puntuacion		NUMBER,
	numeroTemp		NUMBER,
	nombreOficial		VARCHAR(70),
	denominacionOficial	VARCHAR(10),
	PRIMARY KEY(numeroTemp, denominacionOficial, nombreOficial),
	FOREIGN KEY (numeroTemp, denominacionOficial) REFERENCES
Temporada(numeroTemp, denominacionOficial),
	FOREIGN KEY (nombreOficial) REFERENCES Equipo(nombreOficial)
);
```

```
CREATE TABLE Jornada (
	numeroJornada		NUMBER,
	numeroTemp		NUMBER,
	denominacionOficial	VARCHAR(10),
	PRIMARY KEY(numeroJornada, numeroTemp, denominacionOficial),
	FOREIGN KEY (numeroTemp, denominacionOficial) REFERENCES
Temporada(numeroTemp, denominacionOficial)
);

CREATE TABLE Partido (
	equipoLocal		VARCHAR(30),
	equipoVisitante		VARCHAR(30),
	numeroTemp		NUMBER,
	numeroJornada		NUMBER,
	denominacionOficial	VARCHAR(10),
	golesLocal		NUMBER,
	golesVisitante		NUMBER,
	PRIMARY
KEY(equipoLocal,equipoVisitante,numeroTemp,numeroJornada,denominacionOficial),
	FOREIGN KEY (equipoLocal) REFERENCES Equipo(nombreOficial),
	FOREIGN KEY (equipoVisitante) REFERENCES Equipo(nombreOficial),
	FOREIGN KEY (numeroJornada, numeroTemp, denominacionOficial) REFERENCES
Jornada(numeroJornada, numeroTemp, denominacionOficial)
);

CREATE TABLE tablaTemporal (
    annoInicio			NUMBER,
    annoFin			NUMBER,
    denominacionOficial		VARCHAR(10),
    numeroJornada		NUMBER,
    equipoLocal			VARCHAR(30),
       equipoVisitante		VARCHAR(30),
    golesLocal			NUMBER,
       golesVisitante		NUMBER,
    nombreAlternativo		VARCHAR(70),
    ciudad			VARCHAR(50),
    fechaFundacion		NUMBER,
    fechaFundacionLegal		NUMBER,
    nombreHistorico		VARCHAR(70),
    nombreEstadio		VARCHAR(50),
       fechaInaug		NUMBER,
    aforo			NUMBER,
    nombreCorto			VARCHAR(70)
);
```

Parte 2: Introducción de datos y ejecución de consultas

2.1. Población de la base de datos

Para la población de la base de datos, se ha optado por la creación de una tabla temporal en la que se metieron los datos del archivo csv.
Creamos el script de poblarT.sql, en el que transportamos los datos necesarios de cada columna de la tabla temporal a sus columnas correspondientes en nuestras tablas.
Sin embargo, posteriormente, observamos que este no era un método contemplado en el enunciado de la práctica para poblar la base de datos, y que ya existía una tabla que tenía exactamente el mismo propósito que tenía nuestra tabla temporal (datosdb.ligahost). Una vez que vimos el problema, para no tener que rehacer todas las sentencias de población que ya habíamos escrito, decidimos poblar la tabla temporal con los datos de la tabla 'datosdb.ligahost' y así resolvimos el problema.

Vamos a desglosar las distintas decisiones tomadas para poblar cada una de nuestras tablas:

- Estadio, Equipo y Division, OtrosNombres y Partido:
  Tomamos directamente las columnas de la tabla mediante SELECT DISTINCT (evitando así duplicados), asegurándonos de que los PRIMARY KEY no sean NULL (la obtención de los atributos de esta forma se repite en el resto de tablas, dejando de lado las peculiaridades de cada una de las demás).

- Temporada:
  En este caso, el atributo numeroTemp tiene la peculiaridad de que lo calcularemos mediante una resta: "(t.annoInicio - 1971) AS numeroTemp" (igual en resto de tablas).
  Además, utilizaremos un "FROM tablaTemporal t INNER JOIN Division d ON t.denominacionOficial = d.denominacionOficial" que asegura la selección de aquellos registros con misma denominación oficial en ambas tablas.

- Jornada:
  Para asegurarnos que solo se insertan jornadas de una determinada temporada ya registrada utilizamos "FROM tablaTemporal t JOIN Temporada temp ON (t.annoInicio - 1971) = temp.numeroTemp AND t.denominacionOficial = temp.denominacionOficial;".

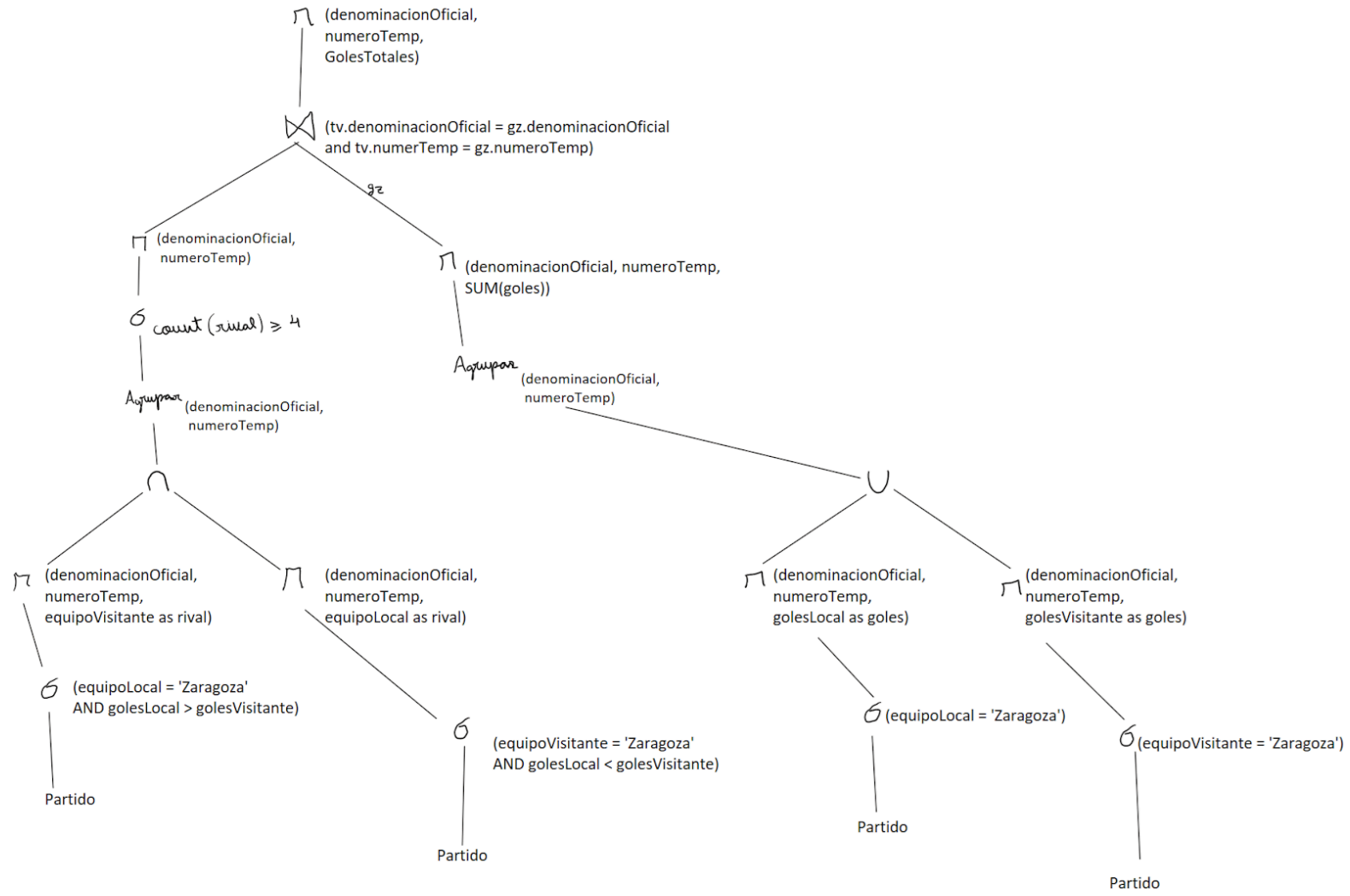
- Clasificacion:
  Buscamos darle valor al atributo puntuación.
  Basicamente, creamos 4 vistas, partidosGanadosIndividuales y partidosEmpatadosIndividuales listan cada partido en el que un equipo ha ganado o empatado respectivamente y luego mediante las vistas partidosGanados y partidosEmpatados contamos cuántas veces a ganado o empatado un equipo respectivamente.
  Después, se realiza un JOIN entre las vistas partidosGanados y partidosEmpatados para emparejar la información por temporada, denominación oficial y equipo, y se calcula la puntuación mediante "(pg.partidosGanados*3+pe.partidosEmpatados) as puntuacion".

Los problemas que hemos encontrado durante la población han sido:
- Problemas iniciales para adaptarnos a la sintaxis de SQL.
- La dificultad para afrontar la unión de datos comunes en tablas como Temporada o Jornada.
- Muchas dificultades para enfocar el cálculo de la puntuación en Clasificacion.

2.2. Consultas SQL

2.2.1. Primera consulta

Árbol sintáctico:



Consulta SQL:

```
CREATE OR REPLACE VIEW vicIdaYVueltaZgz(denominacionOficial, numeroTemp) AS
SELECT denominacionOficial, numeroTemp
FROM (
    SELECT denominacionOficial, numeroTemp, equipoVisitante AS rival
    FROM Partido
    WHERE equipoLocal = 'Zaragoza' AND golesLocal > golesVisitante
    INTERSECT
    SELECT denominacionOficial, numeroTemp, equipoLocal AS rival
    FROM Partido
    WHERE equipoVisitante = 'Zaragoza' AND golesVisitante > golesLocal
)
GROUP BY denominacionOficial, numeroTemp
HAVING COUNT(rival) >= 4;

CREATE OR REPLACE VIEW golesZgz(denominacionOficial, numeroTemp, golesTotales) AS
SELECT denominacionOficial, numeroTemp, SUM(goles) AS golesTotales
```

```
FROM (
    SELECT denominacionOficial, numeroTemp, golesLocal AS goles
    FROM Partido
    WHERE equipoLocal = 'Zaragoza'
    UNION ALL
    SELECT denominacionOficial, numeroTemp, golesVisitante AS goles
    FROM Partido
    WHERE equipoVisitante = 'Zaragoza'
)
GROUP BY denominacionOficial, numeroTemp;

SELECT tv.denominacionOficial “Division”,
       tv.numeroTemp “Temporada”,
       gz.golesTotales “Goles”
FROM vicIdaYVueltaZgz tv
JOIN golesZgz gz ON tv.denominacionOficial = gz.denominacionOficial AND tv.numeroTemp =
gz.numeroTemp;
```

El funcionamiento de la consulta consiste en seleccionar los atributos pedidos de una tabla formada por un JOIN entre las dos vistas creadas:

1. La primera, vicIdaYVueltaZgz, coge del conjunto de rivales por temporada a los que el Zaragoza ha ganado en la ida y vuelta, las temporadas en las que lo hizo con 4 equipos o más.
2. La segunda, golesZgz, coge todos los partidos que el Zaragoza ha ganado en cada temporada y hace la suma de todos sus goles ordenados por temporada.

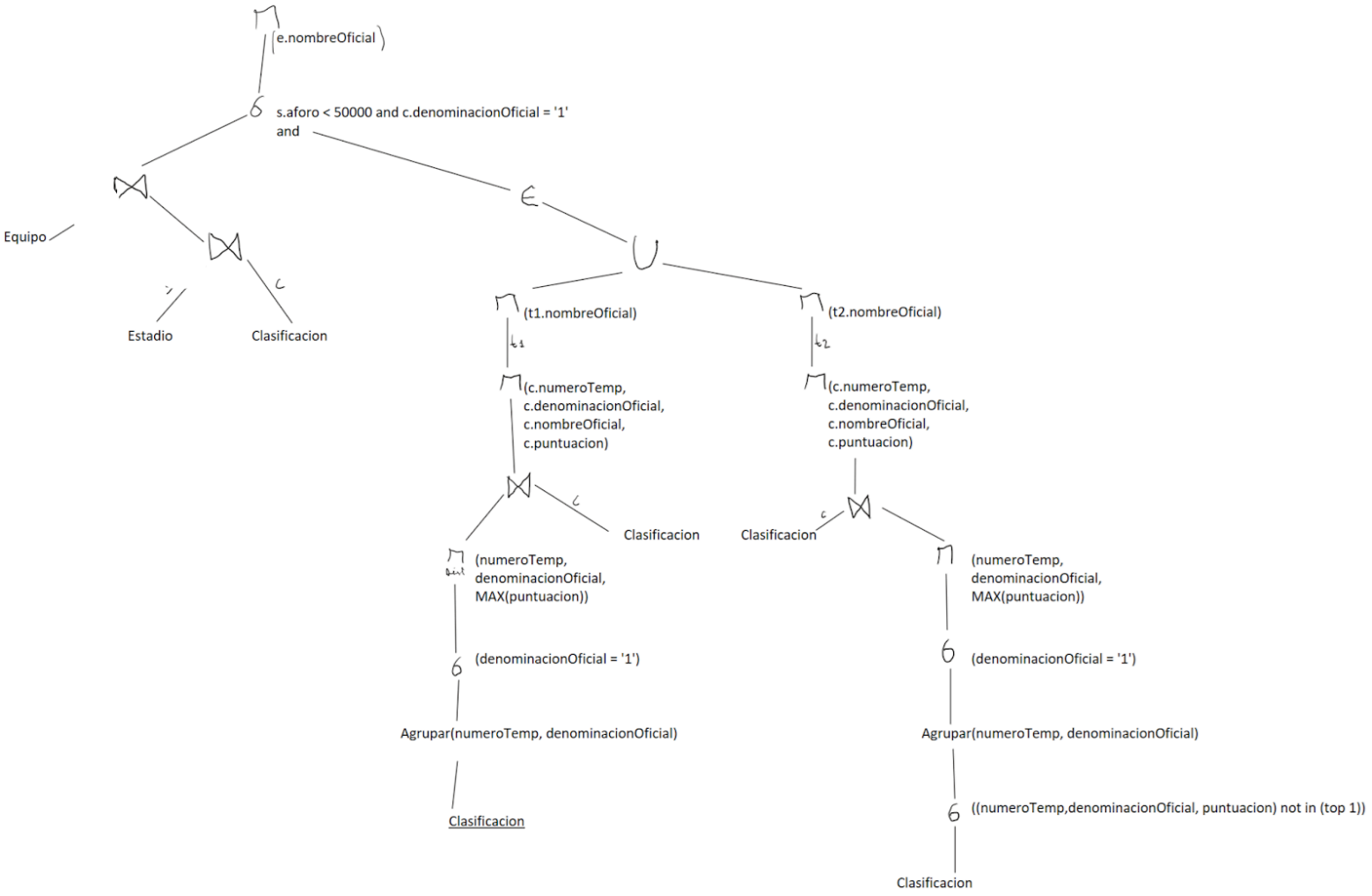
Así, el resultado son la división, temporada y goles marcados (segunda vista) de las temporadas en las que el Zaragoza ganó a 4 equipos tanto en la ida como en la vuelta. En “consultas.sql” del archivo zip de la entrega se encuentra la consulta más comentada paso a paso.

Respuesta obtenida:

```
Division    Temporada      Goles
---------- ---------- ----------
2                  37         79
1                  11         59
1                  27         57
2                  31         54
1                  14         51
```

2.2.2. Segunda consulta

Árbol sintáctico:



Consulta SQL:

```
CREATE OR REPLACE VIEW top1 AS
SELECT  c.numeroTemp, c.denominacionOficial, c.nombreOficial, c.puntuacion
FROM Clasificacion c
JOIN (
    SELECT DISTINCT numeroTemp, denominacionOficial, MAX(puntuacion) AS puntosMax
FROM clasificacion
    GROUP BY numeroTemp, denominacionOficial
    HAVING denominacionOficial = '1'
) pm
ON c.numeroTemp = pm.numeroTemp
AND c.denominacionOficial = pm.denominacionOficial
AND c.puntuacion = pm.puntosMax;

CREATE OR REPLACE VIEW top2 AS
SELECT c.numeroTemp, c.denominacionOficial, c.nombreOficial, c.puntuacion
FROM Clasificacion c

JOIN (
```

```
    SELECT numeroTemp, denominacionOficial, MAX(puntuacion) AS puntosMax
    FROM Clasificacion
    WHERE (numeroTemp, denominacionOficial, puntuacion) NOT IN (
        SELECT numeroTemp, denominacionOficial, puntuacion
        FROM top1
    )
    GROUP BY numeroTemp, denominacionOficial
    HAVING denominacionOficial = '1'
) pm
ON c.numeroTemp = pm.numeroTemp
AND c.denominacionOficial = pm.denominacionOficial
AND c.puntuacion = pm.puntosMax;

SELECT DISTINCT e.nombreOficial “Nombre del equipo”
FROM Equipo e
JOIN Estadio s ON e.nombreEstadio = s.nombreEstadio
JOIN Clasificacion c ON e.nombreOficial = c.nombreOficial
WHERE s.aforo < 50000
  AND e.nombreOficial IN (SELECT t1.nombreOficial FROM top1 t1 UNION SELECT t2.nombreOficial
FROM top2 t2);
```

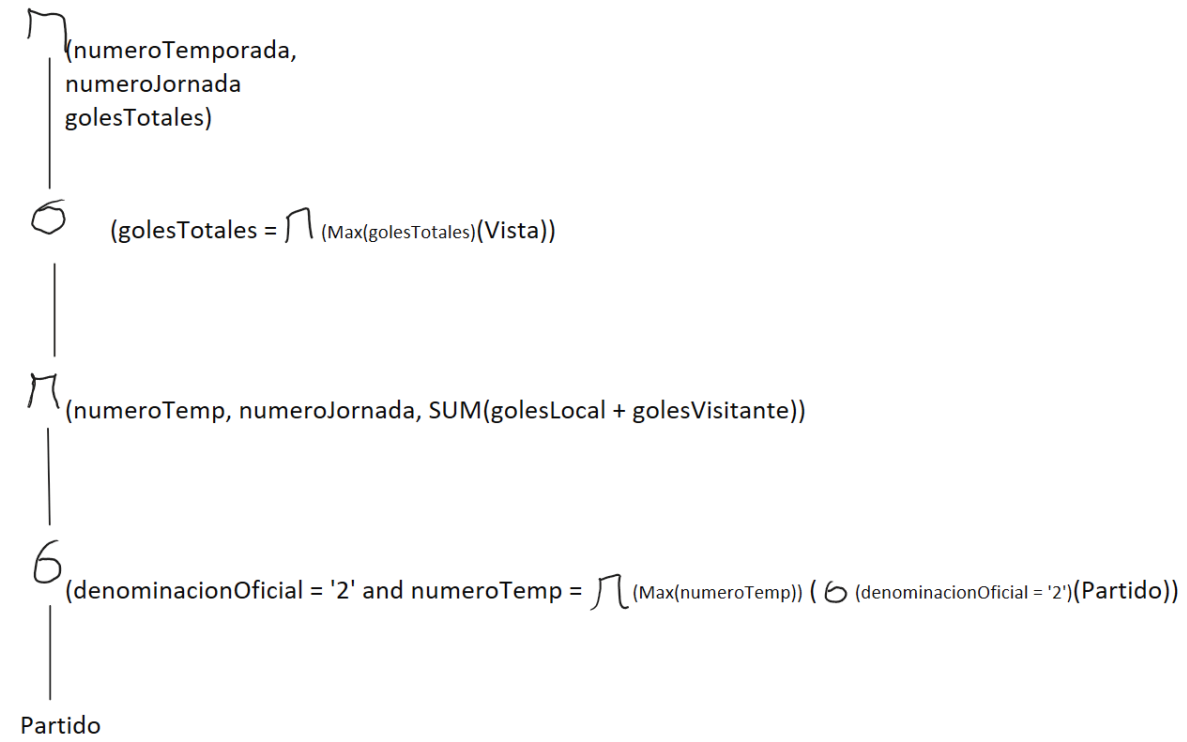
La consulta consiste en mostrar el nombre de los equipos que comparten una serie de atributos recogidos en un JOIN entre la tabla Equipo y la tabla Estadio de esta forma podemos relacionar cada equipo con el aforo de su estadio correspondiente y con Clasificación. Posteriormente ponemos como condición (en el WHERE) que el aforo del estadio de un equipo tiene que ser de menos de 50000 personas y el nombre del equipo tiene que estar contenido dentro de la unión de dos vistas.
Estas dos vistas dan una lista de todos los equipos que han quedado en el primer o el segundo puesto de una temporada de la primera división, (t1 da el primer puesto y t2 el segundo).
De forma que si el nombre de un equipo se encuentra dentro de estas vistas y el aforo de su estadio es de menos de 50000 personas, su nombre se proyectará, cumpliendo con lo especificado en el enunciado.
Ambas vistas así como la consulta en su conjunto están comentadas mucho más en detalle en el código dentro del .zip entregado.

Respuesta obtenida:

```
Nombre del equipo
------------------------------
Zaragoza
Real Sociedad
Dptivo. Coru??a
Valencia
Sevilla
```

2.2.3. Tercera consulta

Árbol sintáctico:



Consulta SQL:

```
CREATE OR REPLACE VIEW vista(numeroTemp, numeroJornada, golesTotales) AS
SELECT numeroTemp, numeroJornada, SUM(golesLocal + golesVisitante) AS golesTotales
FROM Partido
WHERE denominacionOficial = '2'
AND numeroTemp = (
    SELECT MAX(numeroTemp)
    FROM Partido
    WHERE denominacionOficial = '2'
);

SELECT v.numeroTemp "Temporada", v.numeroJornada "Jornada", v.golesTotales "Goles"
FROM vista v
WHERE v.golesTotales=(SELECT MAX(golesTotales) FROM vista);
```

Consiste en seleccionar los atributos pedidos del elemento con el número más grande de goles de la vista. Esta vista crea una tabla con el número de temporada, número de jornada y suma de los goles de los equipos local y visitante de primera división y con la temporada más nueva de la tabla Partido.
El resultado de la consulta es entonces el número de temporada y jornada de la jornada con más goles de la última temporada.En “consultas.sql” del archivo zip de la entrega se encuentra la consulta más comentada paso a paso.

Respuesta obtenida:

```
 Temporada     Jornada       Goles
---------- ---------- ----------
        44          1         37
```

Parte 3: Diseño Físico

3.1. Rendimiento de las consultas SQL

Al aplicar los comandos

EXPLAIN PLAN FOR <consulta SQL>

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())

sobre las 3 diferentes consultas, hemos visto que en general ninguna de ellas es demasiado costosa. Es por esto que, excepto por una modificación en la consulta 3 mencionada más tarde, hemos llegado a la conclusión de que las principales mejoras en cuanto al rendimiento de las consultas de la base de datos se podrían hacer en la propia estructura de la misma.

Veamos los resultados obtenidos en cuanto a tiempo y porcentaje de uso de CPU:

	Primera consulta	Segunda consulta	Tercera consulta
Porcentaje de CPU*	3%	18%	2%
Coste	249	47	138
Tiempo de ejecución*	00:00:01	00:00:01	00:00:01

* *Los valores son aproximados.*

Se nos ocurren las siguientes modificaciones:

1. Utilizar una clave artificial en Partido: por cómo está construida la base de datos, la clave primaria de Partido está formada por el conjunto de 5 claves primarias de diferentes tablas (Division, Temporada, Jornada, Equipo x2) ya que son todas ellas necesarias para diferenciar un partido de otro. Es por ello que el acceso a un elemento de la tabla Partido cuesta tanto tiempo y es el motivo de que a la consulta 2 tarde más tiempo que las otras; tiene que recorrer la tabla de Partido múltiples veces.
   El uso de una clave artificial evitaría tener que pasar por tantas claves primarias para acceder a un elemento, disminuyendo el tiempo de acceso a un elemento de la tabla Partido.

2. Utilizar un atributo puesto en la tabla Clasificacion: con el objetivo de facilitar la segunda consulta al no tener que calcular los puestos 1 y 2 en cada ejecución, se podría añadir en la tabla Consulta un atributo puesto que indicase el puesto del equipo asociado a dicho elemento de la tabla Clasificacion, en la temporada y división indicadas.

3. Finalmente hemos optimizado la segunda consulta gracias a la eliminación de uno de los join que hacíamos en la misma, el cual era evitable (el JOIN Clasificacion c en la consulta), de esta forma hemos conseguido reducir el coste de 47 a 42.

3.2. Restricciones solucionadas mediantes triggers

Hay una gran cantidad de restricciones que Oracle no puede verificar en el acceso, inserción o borrado de las tablas de nuestra base de datos, para ello implementaremos los trigger que después de esta introducción mostraremos mediante 3 ejemplos.

Algunas de las restricciones que podemos encontrar son:

- ➢ No puede jugar un partido un equipo contra sí mismo.
- ➢ No puede insertarse un partido cuya temporada aún no ha empezado.
- ➢ Cuando se borra una división se borra todo lo relacionado con ella.
- ➢ Cuando se borra una temporada se borra todo lo relacionado con ella.
- ➢ Cuando se borra una jornada se borra todo lo relacionado con ella.
- ➢ La capacidad del estadio tiene que ser positiva.
- ➢ No podemos insertar una temporada anterior al año 1972 debido a la estructura interna de la base de datos, para poder hacerlo habría que modificar la columna correspondiente al número de temporada con las nuevas necesidades .
- ➢ Tanto los goles del equipo local como los del equipo visitante tienen que ser mayores o iguales a 0.
- ➢ Cuando se inserta un partido con un equipo no existente, se añade el equipo nuevo a la base de datos.

3.2.1. Primer trigger

Mismo equipo juega un partido:

```
CREATE OR REPLACE TRIGGER mismoEquipoEnPartido
BEFORE INSERT ON Partido
FOR EACH ROW
WHEN NEW.equipoLocal = NEW.equipoVisitante
BEGIN
      RAISE_APPLICATION_ERROR(-20001, 'Un equipo no puede enfrentarse contra si
mismo.');
END;
/
```

Nuestro primer trigger se encarga de uno de los errores más obvios que puede provocar un usuario, y que por ello es bueno cubrirlo. El error ocurre cuando se intenta introducir un partido en el que ambos equipos, local y visitante, son el mismo; es obvio que esto no puede ocurrir, y vamos a evitarlo.

Acerca del funcionamiento del trigger, este es muy básico: el trigger se activa antes de una inserción en la tabla Partido (línea 2), y se lleva a cabo el código que tiene cuando el nombre del equipo local del partido que intentas introducir coincide con el nombre del equipo visitante; es decir, en el partido se enfrenta un equipo contra sí mismo. En código del trigger simplemente imprime el error “'Un equipo no puede enfrentarse contra sí mismo” por pantalla, además de impedir la inserción de dicho partido.

3.2.2. Segundo trigger

Inserción de equipos inexistentes a la hora de insertar un partido :

```
CREATE OR REPLACE TRIGGER InsertNuevoEquipoEnPartido
BEFORE INSERT or UPDATE ON Partido
FOR EACH ROW
DECLARE
    aparicionesDeLocal NUMBER;
    aparicionesDeVisitante NUMBER;
BEGIN
    -- Verificar si el equipo local existe
    SELECT COUNT(*) INTO aparicionesDeLocal FROM Equipo WHERE nombreOficial =
:NEW.equipoLocal;

    -- Si no existe, insertarlo
    IF aparicionesDeLocal = 0 THEN
        INSERT INTO Equipo (nombreOficial) VALUES (:NEW.equipoLocal);
    END IF;

    -- Verificar si el equipo visitante existe
    SELECT COUNT(*) INTO aparicionesDeVisitante FROM Equipo WHERE nombreOficial =
:NEW.equipoVisitante;

    -- Si no existe, insertarlo
    IF aparicionesDeVisitante = 0 THEN
        INSERT INTO Equipo (nombreOficial) VALUES (:NEW.equipoVisitante);
    END IF;
END;
/
```

En caso de intentar hacer una inserción de un partido en el que alguno o los dos equipos no existen, los inserta por su nombre. Esto se debe a que para que un partido pueda formar su clave primaria, este necesita tomar prestado a los equipos que participan en él su nombre, y para ello ambos equipos deben formar parte de la tabla Equipos.

En cuanto al funcionamiento del trigger, este se ejecuta siempre antes de una inserción o actualización de un partido. Entonces, primero comprueba si el equipo local está registrado en la base de datos almacenando en una variable su número de apariciones en la misma. Si dicho número es 0, entonces no existe, por lo que lo inserta. Después hace lo mismo pero con el equipo visitante.

3.2.3. Tercer trigger

Borrado de toda una temporada y borrado de toda una división:

```
CREATE OR REPLACE TRIGGER limpiarTemporada
BEFORE DELETE ON Temporada
FOR EACH ROW
BEGIN
    DELETE FROM Partido WHERE numeroTemp=:OLD.numeroTemp;
    DELETE FROM Jornada WHERE numeroTemp=:OLD.numeroTemp;
    DELETE FROM Clasificacion WHERE numeroTemp=:OLD.numeroTemp;
END;
/


CREATE OR REPLACE TRIGGER limpiarDivision
BEFORE DELETE ON Division
FOR EACH ROW
BEGIN
    DELETE FROM Partido WHERE denominacionOficial=:OLD.denominacionOficial;
    DELETE FROM Jornada WHERE denominacionOficial=:OLD.denominacionOficial;
    DELETE FROM Clasificacion WHERE denominacionOficial=:OLD.denominacionOficial;
    DELETE FROM Temporada WHERE denominacionOficial=:OLD.denominacionOficial;

END;
/
```

En este punto creamos 2 triggers, ambos muy parecidos.

1. El primero, borra una temporada y todo lo que implica esa temporada, es decir, borra los partidos, jornadas y clasificaciones que la componen.

2. El segundo, borra una división y todo lo que implica esa división, es decir, borra los partidos, jornadas, clasificaciones y temporadas que la componen.

Ambos trigger son básicos para el uso habitual y lógico de la base de datos, si no borrásemos los elementos relacionados a una temporada o división se nos quedarían elementos que dependen de algo que no existe. Lo mismo sucede con Jornada, su trigger sería muy similar a los 2 implementados anteriormente.

Organización del grupo

Nos organizamos de manera que todos los integrantes del grupo nos dedicamos a un mismo punto, cada uno creando una parte del apartado. En caso de que uno tuviese problemas, nos reunimos con el objetivo de solucionarlo en conjunto.

El trabajo en grupo ha sido bastante efectivo y ecuánime.

Tareas realizadas	Miembros del grupo implicados	Horas dedicadas	Problemas observados
Esquema E/R global, normalización y listado de las restricciones.	Todos los miembros del grupo.	2,5 horas	Cierta dificultad a la hora de decidir las relaciones con ciertas entidades.
Esquema relacional.	Todos los miembros del grupo.	2 horas	Sin problemas.
Creación de la base de datos.	Todos los miembros del grupo.	2 horas	Sin problemas.
Población de la base de datos.	Todos los miembros del grupo.	25 horas	Problemas relacionados con la población de Clasificación, en concreto el atributo de puntuación.
Creación y pruebas de las consultas SQL.	Todos los miembros del grupo.	5 horas	Sin problemas.
Árboles relacionales de las consultas anteriores.	Todos los miembros del grupo.	2,5 horas	Problemas iniciales en la comprensión de cómo diseñarlos.
Rendimiento de las consultas.	Todos los miembros del grupo.	1,5 horas	Sin problemas.
Triggers.	Todos los miembros del grupo.	4 horas	Sin problemas.
Creación de la memoria.	Todos los miembros del grupo.	6 horas	Sin problemas.