BASES DE DATOS

Curso 2023 - 2024

Fecha de entrega: 06/03/2024

Grupo: L1.02

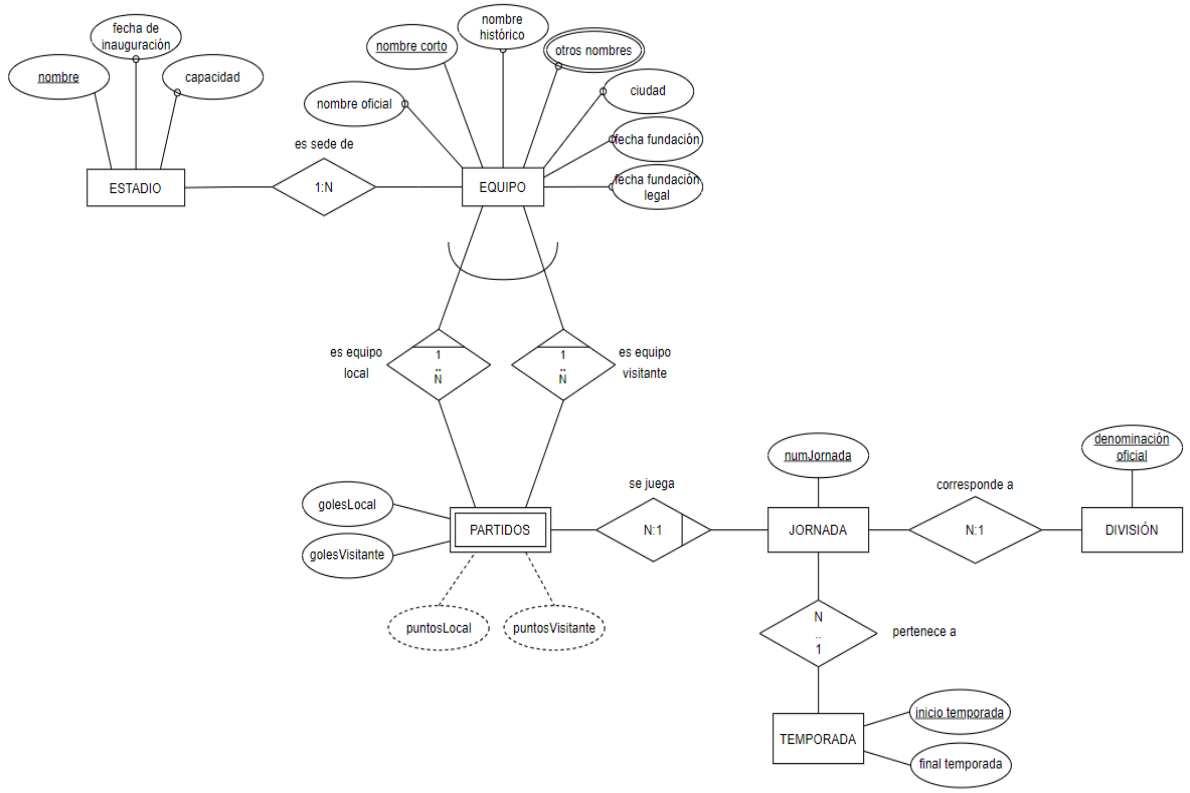
Irene Pascual Albericio 871627@unizar.es

Ariana Porroche Llorén 874055@unizar.es

Iván Vayad Díaz 874762@unizar.es

PARTE 1 - CREACIÓN DE UNA BASE DE DATOS

ESQUEMA E/R



El esquema E/R que hemos diseñado consta de 6 entidades: División, Temporada, Jornada, Partidos, Equipo y Estadio. Ya hemos tenido en cuenta algunas características de los datos proporcionados, como la opcionalidad de los atributos de la entidad Equipo o que la clave primaria de esa misma entidad es el nombre corto, en lugar del nombre oficial.
Restricciones:

- El atributo derivado puntosLocal es igual a 3 si golesLocal es mayor que golesVisitante, sino es 1 si golesLocal es igual a golesVisitante, sino es 0
- El atributo derivado puntosVisitante es igual a 3 si golesLocal es menor que golesVisitante, sino es 1 si golesLocal es igual a golesVisitante, sino es 0
- En la entidad partido, el equipo local tiene que ser distinto del equipo visitante
- En la entidad temporada, la diferencia entre los atributos final temporada e inicio temporada debe ser de 1

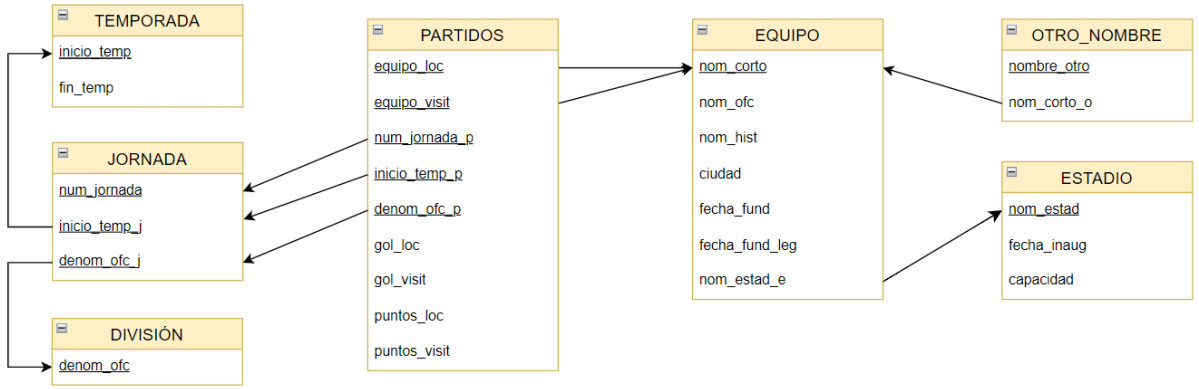
En cuanto a las soluciones alternativas, dudamos especialmente a la hora de gestionar el puesto de cada equipo en la clasificación. Al principio añadimos una entidad Clasificación que almacenase el puesto de cada equipo en cada temporada, pero al final optamos por eliminarlo y calcularlo posteriormente para la primera consulta ya que únicamente lo necesitamos ahí.

ESQUEMA RELACIONAL

Para poder representar el esquema relacional de nuestro modelo E/R, tuvimos que realizar un par de cambios:

- Primero transformamos el atributo multi-valuado otros nombres de la entidad Equipo. Para ello, añadimos una nueva entidad Otro_nombre, la cual se relaciona con la entidad Equipo mediante una relación 1:N. De este modo, eliminamos el atributo otros nombres de la entidad Equipo y lo añadimos como clave primaria a la entidad Otro_nombre. Así conseguimos que estuviese en 1FNBC.
- A continuación, comprobamos que estuviese en 2FNBC y en 3FNBC. Para ello, nos aseguramos de que ningún atributo que no era clave dependiese funcionalmente de parte de la clave o de un conjunto de atributos no clave

Finalmente, tras las transformaciones realizadas, obtuvimos el siguiente esquema relacional:



SENTENCIAS SQL DE CREACIÓN DE TABLAS

Necesitamos crear 7 tablas: Temporada, Division, Jornada, Estadio, Equipo, Otro_nombre y Partidos, en el mismo orden en el que están descritas. Para ello, utilizamos las siguientes sentencias SQL:

```
CREATE TABLE Temporada (
      inicio_temp     NUMBER(4) PRIMARY KEY,
      fin_temp        NUMBER(4)
);
```

```
CREATE TABLE Division (
      denom_ofc       VARCHAR(7) PRIMARY KEY
);
```

```
CREATE TABLE Jornada (
      num_jornada     NUMBER(2),
      inicio_temp_j    NUMBER(4),
      denom_ofc_j VARCHAR(7),
      FOREIGN KEY (inicio_temp_j) REFERENCES Temporada(inicio_temp),
      FOREIGN KEY (denom_ofc_j) REFERENCES Division(denom_ofc),
      PRIMARY KEY (num_jornada,inicio_temp_j,denom_ofc_j)
);
```

```
CREATE TABLE Estadio (
      nom_estad       VARCHAR(50) PRIMARY KEY,
      fecha_inaug     NUMBER(4),
      capacidad       NUMBER(6)
);
```

```
CREATE TABLE Equipo (
      nom_corto       VARCHAR(70) PRIMARY KEY,
      nom_ofc         VARCHAR(70),
      nom_hist        VARCHAR(70),
      ciudad          VARCHAR(40),
      fecha_fund      NUMBER(4),
      fecha_fund_leg  NUMBER(4),
      nom_estad_e     VARCHAR(50),
      FOREIGN KEY (nom_estad_e) REFERENCES Estadio(nom_estad)
);
```

```
CREATE TABLE Otro_nombre (
      nom_otro    VARCHAR(70) PRIMARY KEY,
      nom_corto_o VARCHAR(70),
      FOREIGN KEY (nom_corto_o) REFERENCES Equipo(nom_corto)
);
```

```
CREATE TABLE Partidos (
      equipo_loc        VARCHAR(70),
      equipo_visit      VARCHAR(70),
      num_jornada_p     NUMBER(2),
      inicio_temp_p     NUMBER(4),
      denom_ofc_p       VARCHAR(7),
      gol_loc           NUMBER(2) NOT NULL,
      gol_visit         NUMBER(2) NOT NULL,
      puntos_loc        NUMBER(1),
      puntos_visit      NUMBER(1),
      FOREIGN KEY (equipo_loc) REFERENCES Equipo(nom_corto),
      FOREIGN KEY (equipo_visit) REFERENCES Equipo(nom_corto),
      FOREIGN KEY (num_jornada_p,inicio_temp_p,denom_ofc_p) REFERENCES
Jornada(num_jornada,inicio_temp_j,denom_ofc_j),
      PRIMARY KEY
(equipo_loc,equipo_visit,num_jornada_p,inicio_temp_p,denom_ofc_p)
);
```

PARTE 2 - INTRODUCCIÓN DE DATOS Y EJECUCIÓN DE CONSULTAS

POBLACIÓN DE TABLAS

Para poblar la base de datos, hemos optado por hacer un programa en c++ que lea el archivo .csv y cree los distintos archivos de población de las tablas.

Primero, nos dimos cuenta de que había algunos equipos al final de la hoja Excel que no tenían nombre corto, pero sí que añadían información sobre los distintos nombres del equipo o del estadio en el que jugaba dicho equipo. El hecho de no tener el nombre corto del equipo hacía que no pudiésemos añadir dichas tuplas a nuestras tablas, ya que el nombre corto era la clave primaria de la tabla de equipos. Entonces decidimos que el programa en c++, en esos casos, copiase el nombre oficial en el nombre corto.

Una vez tenida en cuenta esa pequeña modificación de los datos en el programa, ejecutamos en lab000 el programa en c++ llamado “lecturaLiga.cpp” que leía el fichero Excel llamado “Liga.csv” y generaba las distintas sentencias de inserción en el archivo correspondiente para poblar las tablas. Así obtuvimos los siguientes ficheros: insert_division.sql, insert_equipo.sql, insert_estadio.sql, insert_jornada.sql, insert_otro_nombre.sql, insert_partidos.sql e insert_temporada.sql.

Por último, nos conectamos a Oracle y fuimos ejecutando todos los ficheros en el siguiente orden: @insert_division.sql, @insert_temporada.sql, @insert_jornada.sql, @insert_estadio.sql, @insert_equipo.sql, @insert_otro_nombre.sql y @insert_partidos.sql.

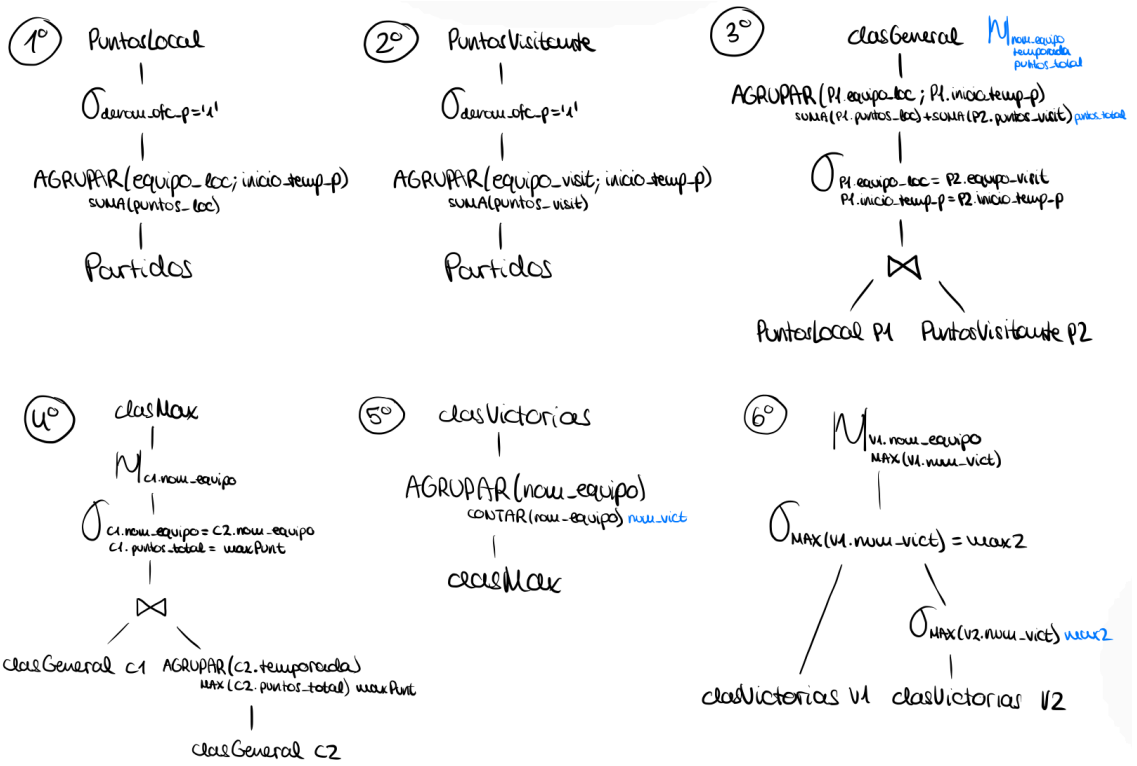
Al principio, no seguimos el orden correcto de población de las tablas, lo que nos provocó algunos errores. Pero finalmente, siguiendo todos los pasos indicados aquí, conseguimos poblar las tablas correctamente.

Para comprobar que todas las tuplas se habían insertado, comparamos el número de líneas de cada fichero SQL con el valor que nos devolvía la consulta SELECT count(*) FROM <nombre_tabla>.

CONSULTAS SQL

CONSULTA 1 - EQUIPO QUE MÁS LIGAS DE PRIMERA DIVISIÓN HA GANADO

Para la primera consulta, hemos creado 5 vistas para mejorar la legibilidad. Primero, hacemos la suma de los puntos de cada equipo como local en cada temporada de primera división. Repetimos lo mismo con los puntos de cada equipo visitante. Luego hacemos la suma de los puntos como local y visitante de cada equipo en cada temporada y así obtenemos los puntos totales. A continuación, nos quedamos con el equipo que más puntos tiene de cada temporada, es decir, con los campeones de primera división. Seguidamente, contamos el número de veces que cada equipo ha ganado la liga. Y por último, proyectamos el equipo que ha ganado la liga de primera el máximo número de veces.

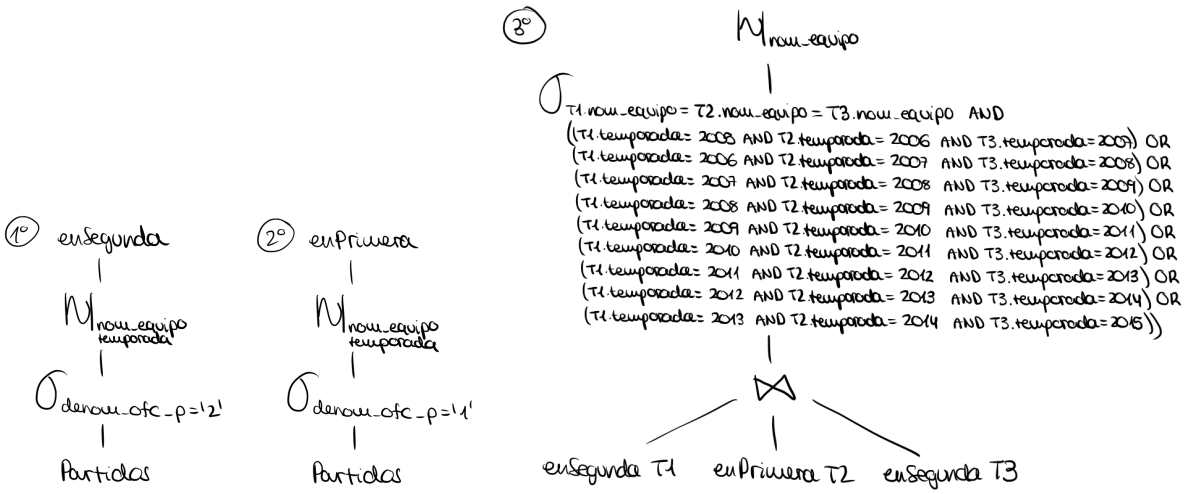


El resultado que hemos obtenido, tras la creación de las 5 vistas y antes de la eliminación de dichas vistas es el siguiente:

EQUIPO	NUM_VICTORIAS
Real Madrid	19

CONSULTA 2 - EQUIPOS DE SEGUNDA DIVISIÓN QUE HAN ASCENDIDO A PRIMERA Y AL AÑO SIGUIENTE HAN VUELTO A DESCENDER EN LAS ÚLTIMAS 10 TEMPORADAS

Para esta consulta, hemos creado 2 vistas, una con el nombre de todos los equipos de segunda división de cada temporada, y otra con el nombre de los equipos de primera división de cada temporada. Para obtener el resultado que queremos, seleccionamos los equipos que en un determinado año entre 2005 y 2013 estuvieron en segunda división, al año siguiente ascendieron a primera división y al siguiente año volvieron a descender a segunda, y proyectamos su nombre.



El resultado que obtenemos tras crear las 2 vistas que hemos descrito anteriormente, y antes de eliminarlas, es el siguiente:

NOM_EQUIPO

Gimn. Tarragon

Murcia

Numancia

Tenerife

Xerez

Hércules

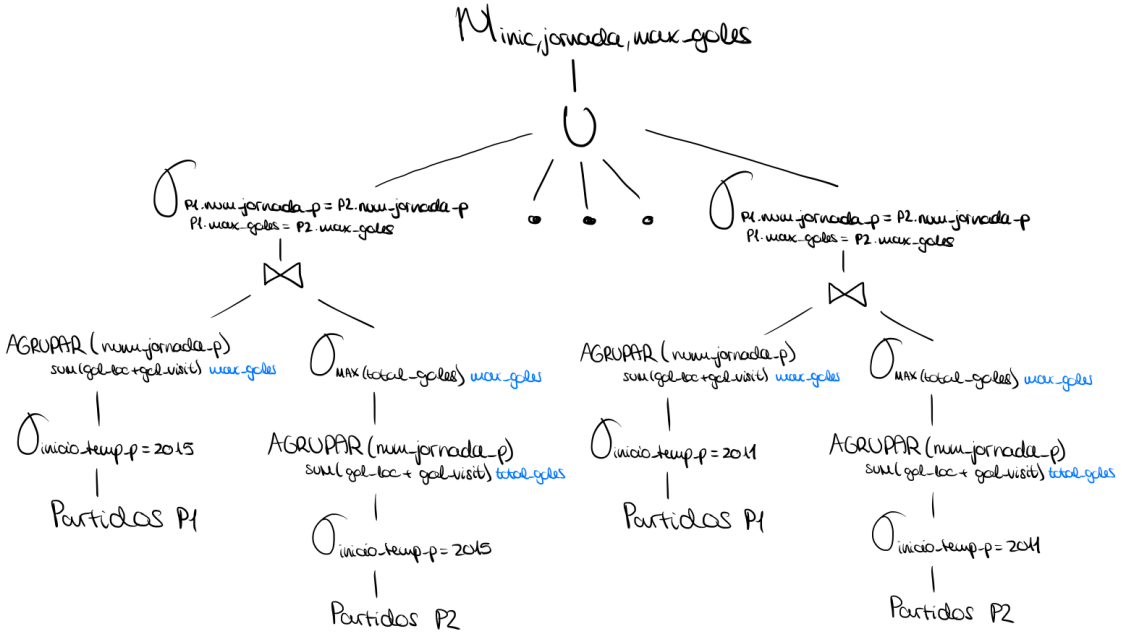
Dptivo. Coruña

Córdoba

CONSULTA 3 - JORNADAS DE LAS ÚLTIMAS CINCO TEMPORADAS DONDE SE HAN MARCADO MÁS GOLES

Para esta consulta, hemos hecho la unión de 5 consultas, una por cada temporada que solicita el enunciado. Primero, seleccionamos de la tabla Partidos sólo las tuplas de la temporada que nos interesa. Luego realizamos la suma de los goles como local y visitante de cada jornada de la temporada. Al mismo tiempo, calculamos el máximo de goles de cada jornada de la temporada concreta. Entonces seleccionamos sólo aquellas tuplas jornada, num_goles, tales que el número de goles sea el máximo de la temporada, y la jornada sea aquella en la que se ha marcado el máximo de goles. Este proceso lo repetimos con las 5 últimas temporadas y hacemos la unión de los resultados obtenidos para obtener el resultado final.

Es una consulta bastante sencilla, por lo que no hemos necesitado crear vistas auxiliares, aunque es muy redundante, ya que repetimos lo mismo 5 veces (en el siguiente apartado optimizaremos la consulta).



El resultado final que obtenemos tras realizar la consulta es el siguiente:

INIC	JORNADA	MAX_GOLES
2015	8	66
2015	9	66
2014	38	77
2013	14	81
2012	30	74
2011	29	73

PARTE 3 - DISEÑO FÍSICO

PROBLEMAS DE RENDIMIENTO

La primera consulta es la que menos coste tiene de entre todas las otras consultas, siendo el máximo de 283. Por ello, hemos creído más importante centrarnos en mejorar las demás consultas.

Para la segunda consulta, antes creábamos 2 vistas auxiliares y ahora, simplemente cogemos la tabla de partidos 3 veces y hacemos un join, además de las condiciones impuestas. Así hemos conseguido reducir el coste de 7149 a 1987. El nuevo código de la consulta es:

```
SELECT DISTINCT T1.equipo_loc
FROM Partidos T1, Partidos T2, Partidos T3
WHERE (T1.equipo_loc=T2.equipo_loc AND T2.equipo_loc=T3.equipo_loc)
AND T1.denom_ofc_p='2' AND T2.denom_ofc_p='1' AND T3.denom_ofc_p='2'
AND ((T1.inicio_temp_p=2005 AND T2.inicio_temp_p=2006 AND T3.inicio_temp_p=2007)
OR (T1.inicio_temp_p=2006 AND T2.inicio_temp_p=2007 AND T3.inicio_temp_p=2008)
OR (T1.inicio_temp_p=2007 AND T2.inicio_temp_p=2008 AND T3.inicio_temp_p=2009)
OR (T1.inicio_temp_p=2008 AND T2.inicio_temp_p=2009 AND T3.inicio_temp_p=2010)
OR (T1.inicio_temp_p=2009 AND T2.inicio_temp_p=2010 AND T3.inicio_temp_p=2011)
OR (T1.inicio_temp_p=2010 AND T2.inicio_temp_p=2011 AND T3.inicio_temp_p=2012)
OR (T1.inicio_temp_p=2011 AND T2.inicio_temp_p=2012 AND T3.inicio_temp_p=2013)
OR (T1.inicio_temp_p=2012 AND T2.inicio_temp_p=2013 AND T3.inicio_temp_p=2014)
OR (T1.inicio_temp_p=2013 AND T2.inicio_temp_p=2014 AND T3.inicio_temp_p=2015));
```

Para la tercera consulta, el principal problema es que estamos duplicando el código y realizando las mismas consultas 5 veces en temporadas distintas. Para mejorar el coste de 346 a 140, hemos cambiado el código de la consulta completa al siguiente:

```
CREATE VIEW numGoles(jornada,temporada,goles) AS
	SELECT num_jornada_p,inicio_temp_p,SUM(gol_loc+gol_visit)
	FROM Partidos
	WHERE inicio_temp_p BETWEEN 2011 AND 2015
	GROUP BY num_jornada_p,inicio_temp_p;
SELECT N1.jornada,N1.temporada,N1.goles
FROM numGoles N1
WHERE (N1.temporada,N1.goles) IN (
	SELECT temporada,MAX(goles)
	FROM numGoles
	GROUP BY temporada
)
ORDER BY N1.temporada;
DROP VIEW numGoles;
```

TRIGGERS

Tras analizar los posibles problemas en la inserción y actualización de las tablas de la práctica, nos hemos dado cuenta de que hay ciertas situaciones que producirían errores o resultados incorrectos, como pueden ser:

- Si se quiere eliminar una tupla de las tablas Temporada o Division, tanto en la tabla Jornada como en Partidos hay referencias a estos valores, por lo que puede ocurrir que también se tengan que eliminar de dichas tablas. Sin embargo, no nos permite borrar debido a la violación de la restricción de integridad. La solución sería un trigger que eliminase de Partidos y luego de Jornada antes de borrar en Temporada o en Division.
- También puede suceder que se quiera insertar una tupla en Jornada, pero que no exista la la temporada o la división en sus respectivas tablas. En este caso, lo más lógico sería hacer un trigger que antes de insertar en Jornada, insertase el valor de inicio de temporada en Temporada o la denominación oficial en Division.

Estas situaciones no son todas las posibles, pueden darse otras situaciones u otras formas de solucionarlas. Nosotros hemos desarrollado los siguientes 3 triggers:

TRIGGER 1 - VERIFICAR VALORES DE TEMPORADA CORRECTOS

Una de las restricciones que Oracle no puede verificar es que en la Temporada, el valor de inicio de temporada sea un año menor al valor de fin de temporada. Tal y como está se podría añadir cualquier valor posible de temporada, pudiendo ser la diferencia entre el fin y el inicio superior a 1 o incluso negativa.

Con el primer trigger, lo que verificamos es que esta diferencia sea justamente de 1 año. Si no es así, el trigger lanza una excepción y no deja añadir la tupla. Para ello, hemos utilizado el siguiente código:

```
CREATE OR REPLACE TRIGGER trigger1
BEFORE INSERT ON Temporada
FOR EACH ROW
      WHEN (NEW.inicio_temp+1!=NEW.fin_temp)
BEGIN
      raise_application_error(-20000,'el valor de inicio de temporada
debe ser 1 año menor al valor de fin');
END;
/
```

Primero creamos el trigger con el nombre de trigger1. Luego definimos que el trigger se ejecute antes de insertar en la tabla Temporada siempre y cuando en la fila que queramos insertar, la diferencia entre los valores de fin e inicio de temporada no sea de 1 año. Entonces, dada esa situación, lanzamos un mensaje de error y no se añade la tupla.

Al crear el trigger, queda habilitado por defecto, entonces no necesitamos darle permiso.

TRIGGER 2 - ELIMINAR TUPLAS EN ESTADIO QUE PROVOCAN ELIMINAR VALORES EN EQUIPO

La tabla Estadio almacena el nombre, la fecha de inauguración y la capacidad de cada estadio. La tabla Equipo, además de almacenar cierta información sobre el equipo como los nombres o algunas fechas, almacena el nombre del estadio en el que juega el equipo. Este estadio es una referencia al nombre del estadio de la tabla Estadio. Por lo tanto, si eliminamos una tupla en la tabla Estadio, este estadio dejará de existir, por lo que la referencia de la tabla Equipo debe pasar a ser nula.

Esto es lo que garantizamos con el trigger2, que por cada tupla que se borre de la tabla Estadio, se ponga a nulo el campo referido a dicho estadio en la tabla Equipo. Para ello, hemos desarrollado el siguiente código:

```
CREATE OR REPLACE TRIGGER trigger2
AFTER DELETE ON Estadio
FOR EACH ROW
BEGIN
      UPDATE Equipo
      SET Equipo.nom_estad_e=NULL
      WHERE Equipo.nom_estad_e=:OLD.nom_estad;
END;
/
```

Primero, creamos el trigger con el nombre trigger2. Luego definimos que se dispare el trigger tras borrar una tupla de la tabla Estadio para todas las filas. Entonces, dada esa condición, actualizamos el campo nom_estad_e de la tabla Equipo a NULL si el estadio referido por esa tupla de Equipo es el estadio eliminado.

TRIGGER 3 - CALCULAR LOS PUNTOS DEL EQUIPO LOCAL Y VISITANTE TRAS LA ACTUALIZACIÓN DE UNA TUPLA O SU INSERCIÓN Y ERROR ANTE GOLES NEGATIVOS

Este trigger abarca numerosas situaciones que suceden en la tabla Partidos.

1. La primera de ellas se produce cuando alguien inserta una tupla en la tabla Partidos sin insertar los puntos ni del equipo local ni del equipo visitante. El trigger en este caso establece los puntos en función de los goles siguiendo las reglas establecidas en el modelo E/R.
2. Por otro lado, cabe la posibilidad de que alguien inserte la tupla completa en Partidos pero se confunda con los puntos del equipo local y/o del equipo visitante, que no se correspondan con los goles y la restricción del modelo E/R. El trigger en este caso también establece los puntos en función de los goles y la restricción explicada anteriormente.
3. Por último, se trata de que alguien inserte algún valor negativo en los goles del equipo local o del visitante, situación que en la realidad no puede ocurrir, nunca los goles pueden ser negativos. En este caso, el trigger lo que hace es lanzar un mensaje de error explicando que deben ser valores positivos.

Para las 3 situaciones descritas, utilizamos el siguiente trigger:

```
CREATE OR REPLACE TRIGGER trigger3
BEFORE INSERT OR UPDATE ON Partidos
FOR EACH ROW
BEGIN
  IF(:NEW.gol_loc<0 OR :NEW.gol_visit<0) THEN
    raise_application_error(-20000,'Los valores deben ser
positivos');
  END IF;
  IF :NEW.gol_loc > :NEW.gol_visit THEN
    :NEW.puntos_loc:=3;
    :NEW.puntos_visit:=0;
  ELSIF :NEW.gol_loc < :NEW.gol_visit THEN
    :NEW.puntos_loc:=0;
    :NEW.puntos_visit:=3;
  ELSE
    :NEW.puntos_loc:=1;
    :NEW.puntos_visit:=1;
  END IF;
END;
/
```

Primeramente, creamos el trigger con el nombre de trigger3. Definimos que se dispare el trigger antes de insertar o actualizar cualquier tupla de la tabla Partidos. Entonces, si los goles del equipo local o los goles del equipo visitante son negativos, se muestra un mensaje de error explicando que ‘Los valores deben ser positivos’. Por otro lado, calcula los puntos del equipo local y del equipo visitante siguiendo las reglas establecidas en el modelo E/R. Si los goles del equipo local son mayores que los del visitante, los puntos del equipo local son 3, mientras que los del visitante son 0, y viceversa. Si sucede un empate, ambos equipos obtienen 1 punto. Así nos aseguramos de que siempre estén bien calculados los puntos y de que ese campo no es nulo en la tabla.

PARTE FINAL

DATOS TRABAJO

El reparto del trabajo ha sido bastante equitativo. Tanto en prácticas como en las horas que quedamos para avanzar, los 3 estuvimos haciendo las mismas tareas a la vez para así tener un conocimiento general sobre las partes más importantes del trabajo. El modelo E/R es la parte en la que más horas invertimos, comparado con las demás partes. Luego fuimos repartiendo el trabajo y dividiéndonos para avanzar más, aunque siempre poniendo en común tanto las soluciones como las dudas que se nos fueron planteando.

En total, cada uno dedicamos aproximadamente las siguientes horas:

Irene Pascual Albericio	38h
Ariana Porroche Llorén	39h
Iván Vayad Díaz	36h

Los principales problemas que encontramos fueron:

- Hacer el modelo E/R. Aún no teníamos mucha experiencia en el campo de las bases de datos, por lo que esta parte abarcó una buena parte del tiempo dedicado al trabajo. Especialmente lo que nos costó fue decidir las relaciones entre las distintas entidades.
- Normalizar el modelo relacional. Pasar del modelo E/R al relacional no nos fue complicado, pero normalizarlo hizo que se nos plantearan algunas pequeñas dudas que fuimos resolviendo poco a poco.
- Disparar los triggers. Tuvimos que probar varias formas hasta que averiguamos cómo se creaban los disparadores.
- Averiguar el rendimiento de las consultas. También nos costó un poco interpretar los datos obtenidos tras consultar el plan de ejecución de cada consulta, así como encontrar una manera de mejorar su rendimiento.