BASES DE DATOS

2º CURSO - 2ºCUATRIMESTRE

GRUPO L1.01

Carlos Solana Melero NIP: 872815@unizar.es

Daniel Simón Gayán Nip: 870984@unizar.es

Diego Mateo Lorente Nip: 873338@unizar.es

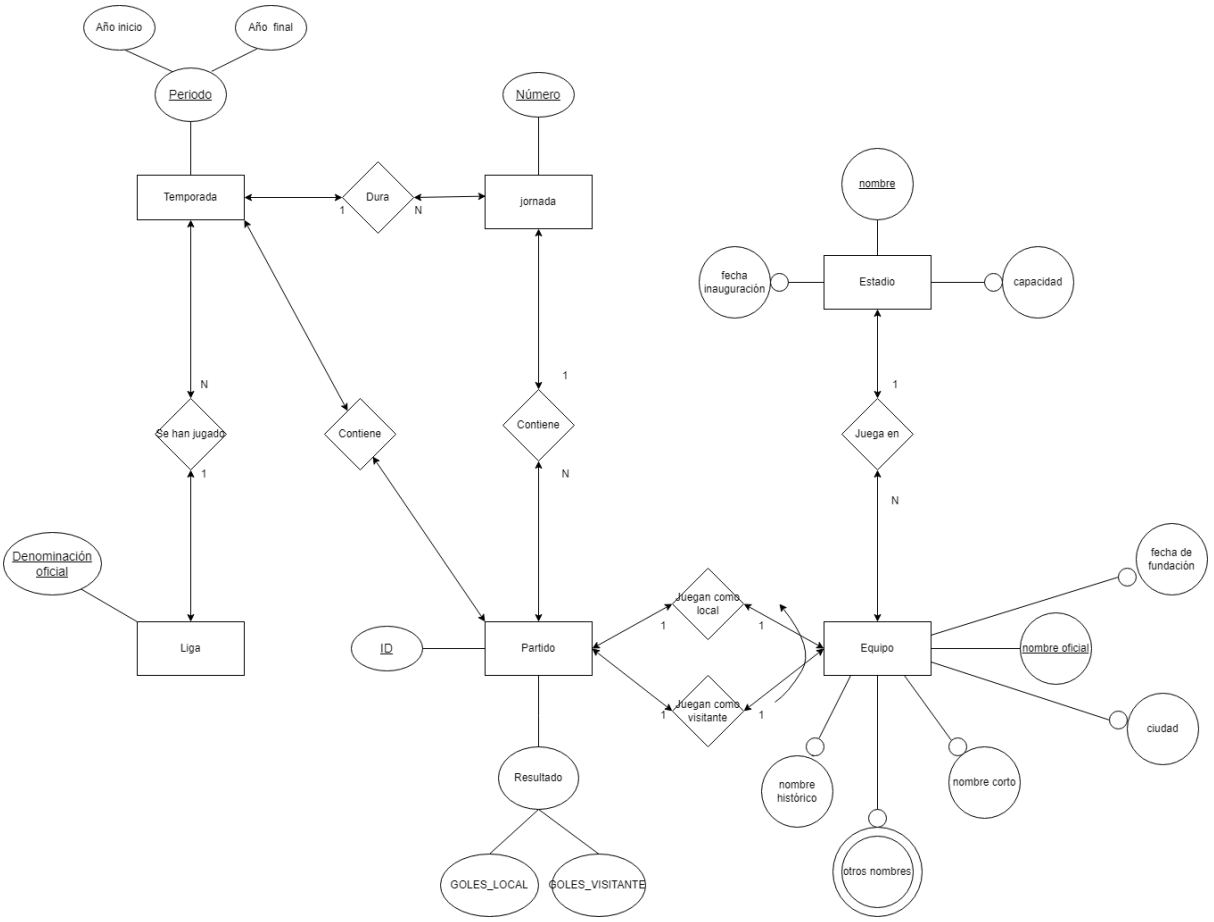
6 marzo de 2024

Parte 1: Creación de una base de datos:

Modelo Entidad - Relación

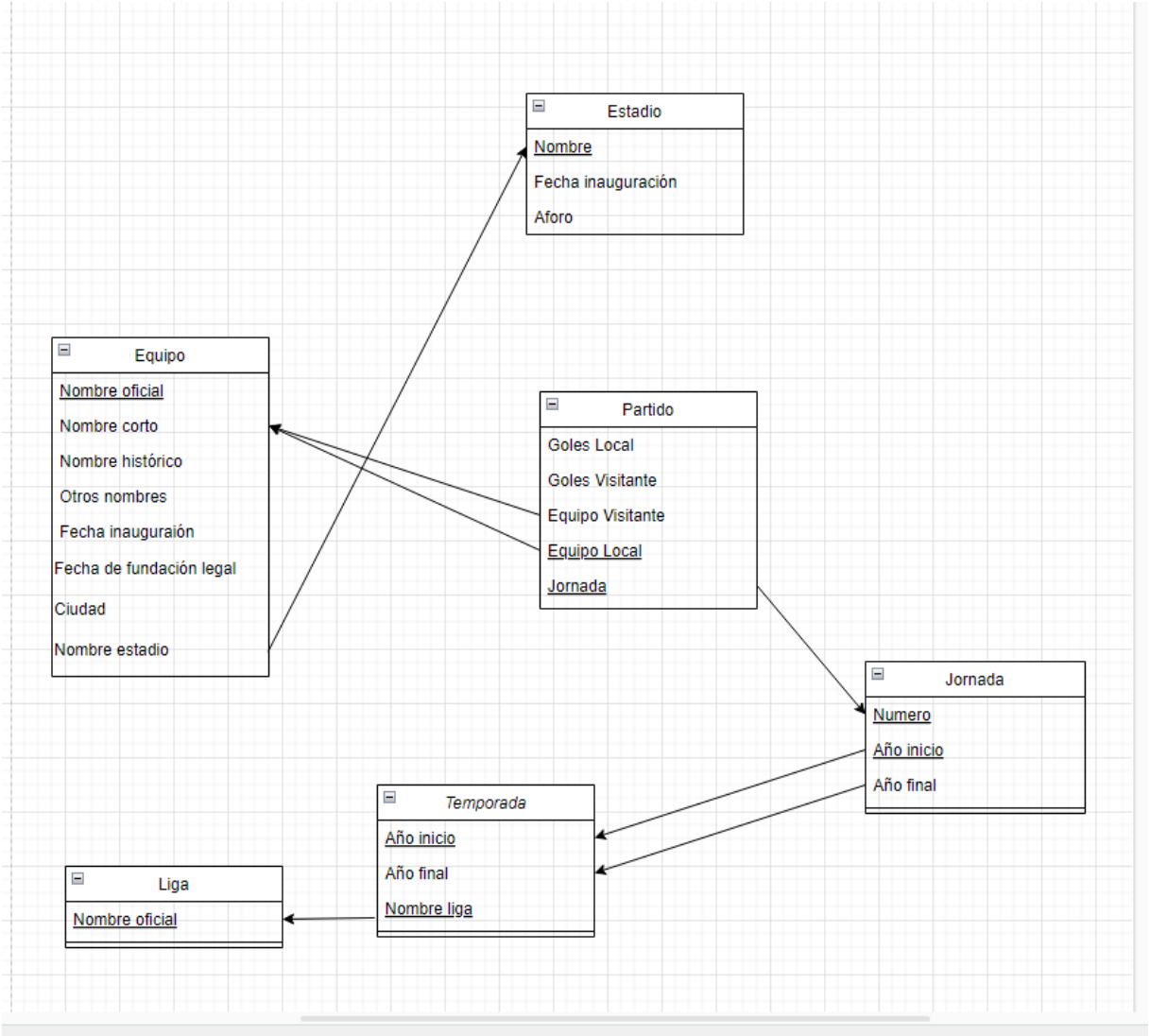
Mientras hemos estado realizando el esquema E/R de la base de datos propuesta, nos hemos encontrado con algunas restricciones. Comenzamos hablando de “Liga”, donde el único atributo es “denominación oficial”, siendo este atributo primario y no hay restricción alguna respecto a su uso. Como solo hay una ocurrencia de la liga por cada temporada, hemos concluido que la relación entre “Liga” y “Temporada”, cuyo nombre es “Se han jugado, ya que se guarda el registro de temporadas de la liga, debe ser 1-N. Esto se debe a que una temporada empieza en un año y termina al año siguiente, por lo que no puede darse el caso en el que la misma liga empiece 2 veces en el mismo año, ya que ninguna liga empieza y termina en el mismo año. Con respecto a “Temporada”, tenemos un único atributo (por lo tanto, primario) que es el periodo en el que se comprende la misma temporada, la cual, como hemos mencionado anteriormente, comienza un año y termina al año siguiente. Por tanto, hemos querido subdividir el atributo “periodo” en 2 atributos, que serán “año inicio” y “año final”. Esta entidad tiene con “Jornada” una relación 1-N cuyo nombre va a ser “Dura”. Esto se debe a que a cada temporada se le atribuyen “N” jornadas, y ninguna jornada puede pertenecer a más de una temporada distinta. Respecto a la entidad “Jornada”, el único atributo que hemos introducido (y es el primario) es “número”, que corresponde al identificador numérico de la jornada en cuestión. Esto es importante ya que una temporada tiene varias jornadas y el número identificativo sirve para distinguir una jornada de otra dentro de una misma temporada. Una restricción de este atributo es que debe ser mayor a 0, ya que no puede haber un identificador negativo para una jornada en cuestión. Esta entidad se relaciona con la entidad “Partido” en una relación 1-N que hemos llamado “Contiene”, ya que una jornada contiene varios partidos que se juegan entre distintos equipos y son únicos, es decir, que un partido no puede repetirse exactamente en 2 jornadas distintas dentro de una temporada. En la entidad “Partido” hemos introducido un único atributo que es “Resultado”, que contiene la cantidad de goles tanto del equipo local como del visitante. Como dicho atributo debe contener 2 valores distintos, hemos decidido subdividir este atributo en 2, que serán “Goles local” y “Goles visitante”, que corresponden a los goles de los 2 equipos que jueguen en partido. Esta entidad va a tener una relación doble con la entidad “Equipo”. Ambas relaciones son 1-1 y sus nombres son “Juegan como local” y “Juegan como visitante”, respectivamente. La entidad “Equipo” tiene como atributo primario el nombre corto, que es el nombre con

el que se corresponderá tanto el equipo local como el equipo visitante. Además de este, tenemos otros atributos: “nombre corto”, “nombre histórico” y “otros nombres”, todos opcionales, ya que puede darse el caso de que solo se conozca el nombre corto del equipo. El atributo “otros nombres” es multivaluado porque pueden existir varios apodos para un equipo. Los otros dos atributos son “fecha de fundación” y “ciudad”, también opcionales porque en algunos casos no se conocen dichos valores. Varios equipos pueden jugar en un mismo estadio, por lo que la entidad tendrá la relación N-1 “Juega en” con “Estadio, que tendrá los atributos “nombre” como primario, “fecha inauguración” y “capacidad”, ambos opcionales. La única restricción la encontraremos en “capacidad”, que debe tener un valor mayor a 0. Comentadas todas las restricciones, así nos queda el diagrama:



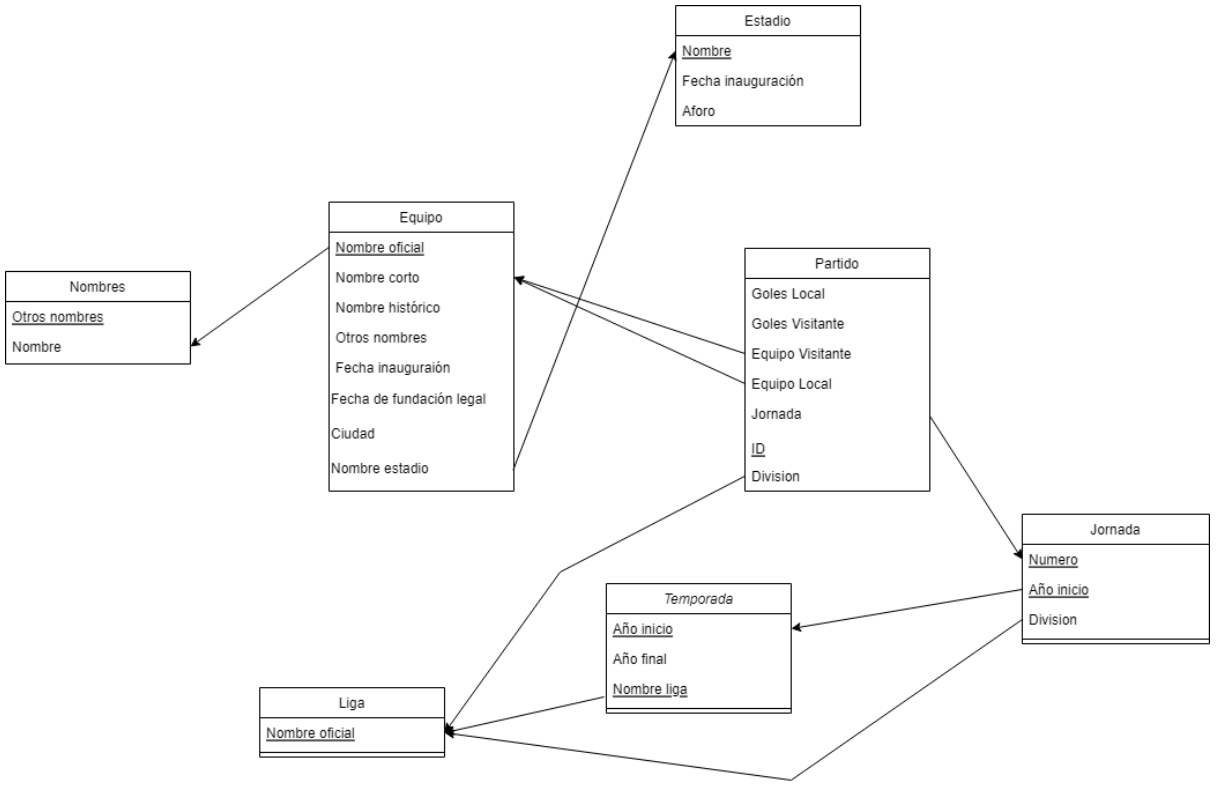
Modelo relacional y normalización

Por otro lado tuvimos que pasar del modelo Entidad/Relación al modelo Relacional para más tarde poder diseñar las sentencias SQL para crear y poblar las tablas. Nuestro modelo inicial fue este, pero más tarde nos dimos cuenta de distintos errores y aspectos a mejorar al aplicar las formas normales.



Inicialmente para cada entidad creamos una tabla cada una con su respectiva primary key, esos campos primarios están subrayados, los demás campos son atributos de la entidad. A parte, lo interesante de este modelo son las claves ajenas que creamos para poder relacionar las tablas.

Al ir pasando las diferentes formas normales fuimos cambiando nuestro modelo relacional, principalmente partido, al principio teníamos Equipo Local y Equipo visitante como claves primarias, pero pronto nos dimos cuenta que no cumplía la segunda forma normal, por lo que seleccionamos Equipo Local y jornada como claves Primarias.



Nuestro modelo final es este en el cual hemos cambiado atributos y relaciones ya que al poblar nos daban errores y hemos tenido que rectificar, los principales cambios que hemos hecho ha sido añadir división como campo en Partido y Jornada, así como crear un campo primario ID para el partido. También hemos añadido nombres como atributo multivaluado por lo que hemos creado una tabla nueva para almacenarlo.

Sentencias SQL de creación de tablas

```
CREATE TABLE Liga (
 DIVISION CHAR(32) PRIMARY KEY
);
```

En esta sentencia definimos la tabla liga cuya única componente es división que indica en qué Clasificación nos encontramos (1, 2, Ascenso, Descenso)

```
CREATE TABLE Temporada (
 INICIO_TEMPORADA CHAR(32),
 FIN_TEMPORADA CHAR(32) NOT NULL,
 DIVISION CHAR(32),
 PRIMARY KEY (INICIO_TEMPORADA, DIVISION),
 FOREIGN KEY (DIVISION) REFERENCES Liga(DIVISION)
);
```

Esta sentencia define la tabla Temporada que se identifica según la clave INICIO_TEMPORADA que indica el año en el que empezó y la clave foránea DIVISION que indica la categoría en la que se jugó esa temporada. Por último tenemos la clave FIN_TEMPORADA que en un principio nos planteamos que fuera clave primaria pero pronto nos dimos cuenta que depende totalmente de INICIO_TEMPORADA, por lo que la hemos dejado como NOT_NULL.

```
CREATE TABLE Jornada (
 INICIO_TEMPORADA CHAR(32),
 JORNADA NUMBER,
 DIVISION CHAR(32),
 PRIMARY KEY (JORNADA, INICIO_TEMPORADA, DIVISION),
 FOREIGN KEY (INICIO_TEMPORADA, DIVISION) REFERENCES Temporada(INICIO_TEMPORADA, DIVISION),
 CONSTRAINT cons_jornada CHECK (JORNADA >= 0)
);
```

En esta sentencia especificamos la tabla Jornada cuyas claves primarias son el número de jornada, la temporada a la que pertenece, y la división en la que se ha jugado. También hemos añadido un constraint que nos indica que el número de la jornada no puede ser negativo. Tuvimos bastantes problemas a la hora de seleccionar las claves primarias para jornada hasta que nos dimos cuenta que teníamos que incluir división o no lograríamos que fuera totalmente única, por lo que lo añadimos como clave foránea, así como INICIO_TEMPORADA.

```
CREATE TABLE Estadio (
 ESTADIO CHAR(50),
 FECHA_INAG CHAR(32),
 AFORO CHAR(32),
 PRIMARY KEY (ESTADIO),
 CONSTRAINT cons_afo CHECK (AFORO >= 0)
);
```

Para crear la tabla estadio hemos definido como clave primaria el nombre del estadio, también tiene los campos de AFORO Y FECHA_INAG que son NULLABLES, hemos puesto como constraint para estadio que no se pueda tener un aforo negativo. Nos planteamos incluir el equipo que jugaba en el estadio como clave foránea pero vimos que no era necesario.

```
CREATE TABLE Equipo (
 NOMBRE_EQUIPO CHAR(50),
 NOMBRE_OFICIAL CHAR(50),
 CIUDAD CHAR(32),
 FUNDACION CHAR(32),
 FUNDACION_LEGAL CHAR(32),
 NOMBRE_HISTORICO CHAR(50),
 ESTADIO CHAR(50),
 PRIMARY KEY (NOMBRE_EQUIPO),
 FOREIGN KEY (ESTADIO) REFERENCES Estadio(ESTADIO)
);
```

En la tabla de equipo tenemos los campos de NOMBRE_EQUIPO que indica su nombre corto y es su clave primaria, NOMBRE_OFICIAL representa el nombre oficial del equipo, NOMBRE_HISTORICO tal y como indica su nombre almacena el nombre histórico del equipo, todos ellos tienen 50 caracteres ya que nos encontramos con problemas si los poníamos de 32, debido a que hay algunos muy largos. FUNDACION_LEGAL y CIUDAD son otros campos con información pero como son más cortos sólo necesitan 32 caracteres. Por último tenemos ESTADIO que es una clave ajena de la tabla Estadio

```
CREATE TABLE Nombres (
 NOMBRE_EQUIPO CHAR(50) NOT NULL,
 OTRO_NOMBRE CHAR(36) PRIMARY KEY,
 FOREIGN KEY (NOMBRE_EQUIPO) REFERENCES Equipo(NOMBRE_EQUIPO)
);
```

La tabla de nombres tiene como clave primaria OTRO_NOMBRE para que solo se cree en el caso de que tengamos otro nombre para el equipo debido a que no todos lo tienen y así permitimos que sea multivaluado, luego tenemos también en NOMBRE_EQUIPO el nombre corto del equipo para poder saber a cuál corresponde.

```
CREATE TABLE Partido (
 ID NUMBER PRIMARY KEY,
 INICIO_TEMPORADA CHAR(32),
 DIVISION CHAR(32),
 JORNADA NUMBER,
 EQUIPO_LOCAL CHAR(50) NOT NULL,
 EQUIPO_VISITANTE CHAR(50) NOT NULL,
 GOLES_LOCAL NUMBER(2) NOT NULL,
 GOLES_VISITANTE NUMBER(2) NOT NULL,
 CONSTRAINT cons_gLocal CHECK (GOLES_LOCAL >= 0),
 CONSTRAINT cons_gVisitante CHECK (GOLES_VISITANTE >= 0),
 FOREIGN KEY (EQUIPO_LOCAL) REFERENCES Equipo(NOMBRE_EQUIPO),
```

```
 FOREIGN KEY (EQUIPO_VISITANTE) REFERENCES Equipo(NOMBRE_EQUIPO),
FOREIGN KEY (INICIO_TEMPORADA,DIVISION,JORNADA) REFERENCES
Jornada(INICIO_TEMPORADA,DIVISION,JORNADA)
);
```

Partido es la tabla más extensa, se identifica cada uno por un ID que es único y es su clave primaria, también tenemos varias claves foráneas que son EQUIPO_LOCAL, EQUIPO_VISITANTE, INICIO TEMPORADA, DIVISION y JORNADA, cada una en referencia a a un equipo o para identificar el momento en el que se jugó el partido, otros campos son GOLES_LOCAL y GOLES_VISITANTE que tienen como constraint que no pueden ser negativos y son NOT NULL, al representar un número pequeño solo tienen dos caracteres de espacio reservado.

Poblando la base de datos

En un principio intentamos poblar desde un ctv y modificamos el original para tener una tabla distinta para cada campo, también creamos los archivos ctl para poblar las tablas. Pero a la hora de ejecutarlo todo tuvimos grandes problemas para poblar la jornada ya que insistía en que estábamos insertando un campo vacío en una clave primaria. Después de estar varias tardes decidimos cambiar al método de insertar los datos a una serie de consultas sql para poblar las tablas a partir de la tabla que está ya cargada.

```
INSERT INTO Liga SELECT DISTINCT DIVISION
        FROM datosdb.ligahost
        WHERE DIVISION IS NOT NULL;

INSERT INTO Temporada SELECT DISTINCT INICIO_TEMPORADA, FIN_TEMPORADA,  DIVISION
        FROM datosdb.ligahost
        WHERE INICIO_TEMPORADA IS NOT NULL AND DIVISION IS NOT NULL AND FIN_TEMPORADA IS
NOT NULL;

INSERT INTO Jornada SELECT DISTINCT INICIO_TEMPORADA, JORNADA,  DIVISION
        FROM datosdb.ligahost
        WHERE INICIO_TEMPORADA IS NOT NULL AND DIVISION IS NOT NULL AND JORNADA IS NOT
NULL;


INSERT INTO Estadio SELECT DISTINCT ESTADIO, FECHA_INAG, AFORO
        FROM datosdb.ligahost
        WHERE ESTADIO IS NOT NULL;

INSERT INTO Equipo SELECT DISTINCT EQUIPO_LOCAL, Club , CIUDAD, FUNDACION , Fund_legal, Nombre ,
ESTADIO
        FROM datosdb.ligahost
```

```
            WHERE EQUIPO_LOCAL IS NOT NULL;

INSERT INTO Nombres SELECT DISTINCT EQUIPO_LOCAL, EQUIPO
            FROM datosdb.ligahost
 WHERE EQUIPO IS NOT NULL AND EQUIPO_LOCAL IS NOT NULL;

CREATE SEQUENCE sec START WITH 1 INCREMENT BY 1;

 INSERT INTO Partido (ID,INICIO_TEMPORADA, DIVISION, JORNADA, EQUIPO_LOCAL, EQUIPO_VISITANTE,
GOLES_LOCAL, GOLES_VISITANTE)
 SELECT   sec.NEXTVAL,
       INICIO_TEMPORADA,
       DIVISION,
       JORNADA,
       EQUIPO_LOCAL,
       EQUIPO_VISITANTE,
       GOLES_LOCAL,
       GOLES_VISITANTE
 FROM datosdb.ligahost
WHERE INICIO_TEMPORADA IS NOT NULL AND DIVISION IS NOT NULL AND JORNADA IS NOT NULL AND
EQUIPO_LOCAL IS NOT NULL AND EQUIPO_VISITANTE IS NOT NULL AND GOLES_LOCAL IS NOT NULL AND
GOLES_VISITANTE IS NOT NULL;
```

En las primeras sentencias simplemente seleccionamos los campos que queremos introducir en las tablas creadas en el orden correcto, indicamos que queremos extraerlos de datosdb.ligahost y por último ponemos como NOT_NULL todos aquellos campos que lo necesiten. Nuestras dificultades empezaron a llegar cuando nos dimos cuenta que no todos los equipos disponían de un nombre corto, tampoco todos tenían un nombre oficial o un nombre histórico y por lo tanto no podían ser claves primarias. Al final la solución que encontramos fue seleccionar EQUIPO_LOCAL como clave primaria, somos conscientes de que no es una solución perfecta ya que aquellos equipos que no hayan jugado ningún partido no aparecerán en nuestra base de datos pero es la mejor que encontramos. Otro problema con el que nos encontramos fue el seleccionar una clave primaria para partido, ya que hay una serie de partidos del que solo disponemos información sobre quienes lo jugaron y sobre los goles, por lo que nuestro esquema inicial que era añadir JORNADA, INICIO_TEMPORADA y DIVISION a partido y ponerlos como claves primarias, además de EQUIPO_LOCAL no era válido. Por lo que decidimos tener un id único para cada partido y usarlo como clave primaria. Para crearlo creamos una secuencia tal y como aparece en el enunciado llamada sec. Después asignamos al campo id sec.NEXTVAL que hace que aumente en 1 la secuencia para que sea único. Para su correcto funcionamiento cambiamos el select distinct por un select normal y al borrar nos encargamos también de borrar la secuencia.

Consultas

Para la primera consulta, que nos pide imprimir por pantalla el equipo que ha ganado más ligas según la base de datos, hemos realizado una implementación en la que poco a poco vamos realizando vistas para acercarnos poco a poco a la solución.

```
CREATE VIEW partidos_primera_div
AS SELECT INICIO_TEMPORADA, JORNADA, EQUIPO_LOCAL, EQUIPO_VISITANTE, GOLES_LOCAL, GOLES_VISITANTE
  FROM PARTIDO
  WHERE DIVISION='1'
  ORDER BY INICIO_TEMPORADA DESC;
```

Inicialmente, vamos a crear una vista que nos muestre todos y cada uno de los partidos de primera división, mostrando por pantalla el año de inicio de la temporada, la jornada, los 2 equipos enfrentados y el resultado final del partido. Esto nos va a permitir saber cómo distribuir los puntos, que es lo que vamos a hacer a continuación.

```
CREATE VIEW victorias
AS( SELECT INICIO_TEMPORADA, EQUIPO_LOCAL AS EQUIPO FROM partidos_primera_div WHERE GOLES_LOCAL>GOLES_VISITANTE
  UNION ALL
      SELECT INICIO_TEMPORADA, EQUIPO_VISITANTE AS EQUIPO FROM partidos_primera_div WHERE GOLES_VISITANTE>GOLES_LOCAL);

CREATE VIEW empatados
AS( SELECT INICIO_TEMPORADA, EQUIPO_LOCAL AS EQUIPO FROM partidos_primera_div WHERE GOLES_LOCAL=GOLES_VISITANTE
  UNION ALL
     SELECT INICIO_TEMPORADA, EQUIPO_VISITANTE AS EQUIPO FROM partidos_primera_div WHERE GOLES_VISITANTE=GOLES_LOCAL);
```

Esta vista la utilizamos para crear el método por el cual se van a contar los puntos. El sistema es igual al de cualquier liga de fútbol nacional de la actualidad: el equipo que más goles meta, ya sea jugando de local o de visitante, se llevará 3 puntos, mientras que el que meta menos no se llevará ninguno. Por otro lado, si tanto el equipo local como el visitante meten la misma cantidad de goles, ambos se llevarán un punto. De esta forma, podemos saber cuántos puntos se lleva cada equipo.

```
CREATE VIEW puntos
AS  SELECT INICIO_TEMPORADA, EQUIPO, COUNT(*) AS PTOS
   FROM empatados
   GROUP BY INICIO_TEMPORADA,EQUIPO
   UNION ALL
   SELECT INICIO_TEMPORADA, EQUIPO, 3*COUNT(*) AS PTOS
   FROM victorias
   GROUP BY INICIO_TEMPORADA, EQUIPO;
```

Aquí se representa lo anterior mencionado con respecto a la distribución de los puntos. Tuvimos ciertos problemas a la hora de intentar contar los puntos, pero todo se soluciona al utilizar la función COUNT(*), que permite contar el número de filas.

```
CREATE VIEW puntos_por_temporada
AS SELECT INICIO_TEMPORADA, EQUIPO, SUM(PTOS) as PUNTOSTOTALES
   FROM puntos
   GROUP BY INICIO_TEMPORADA, EQUIPO
   ORDER BY PUNTOSTOTALES ASC;
```

Aquí calculamos los puntos que ha ganado cada equipo en cada temporada, utilizando la función SUM(PTOS) para sumar todos los puntos obtenidos en una temporada. Esta vista nos va a venir muy bien, ya que de esta forma podemos calcular quién ha ganado más puntos y, por tanto, ha ganado la liga, y después contar el número de campeonatos ganados de cada equipo y sacar el máximo de ellos. Pero, por el momento, primero debemos saber cual es la cantidad más alta de puntos que se ha ganado en cada temporada.

```
CREATE VIEW max_puntos_temporada
AS SELECT INICIO_TEMPORADA, MAX(PUNTOSTOTALES) AS PUNTOS
   FROM puntos_por_temporada
   GROUP BY INICIO_TEMPORADA;
```

En esta vista, como hemos mencionado anteriormente, calculamos el máximo de puntos de cada temporada. A continuación, haremos una vista que compare los puntos de cada equipo en cada temporada con los puntos máximos de cada temporada y, si hay coincidencia, quiere decir que ese equipo ha ganado la liga de esa temporada. El enunciado menciona que, si 2 equipos consiguen los mismos puntos máximos en una temporada concreta, los 2 equipos serán considerados campeones de liga.

```
CREATE VIEW ganadores_ligas
AS SELECT INICIO_TEMPORADA, EQUIPO, PUNTOSTOTALES
```

```
FROM puntos_por_temporada pts
WHERE pts.PUNTOSTOTALES=(SELECT PUNTOS FROM max_puntos_temporada max WHERE
pts.INICIO_TEMPORADA=max.INICIO_TEMPORADA);
```

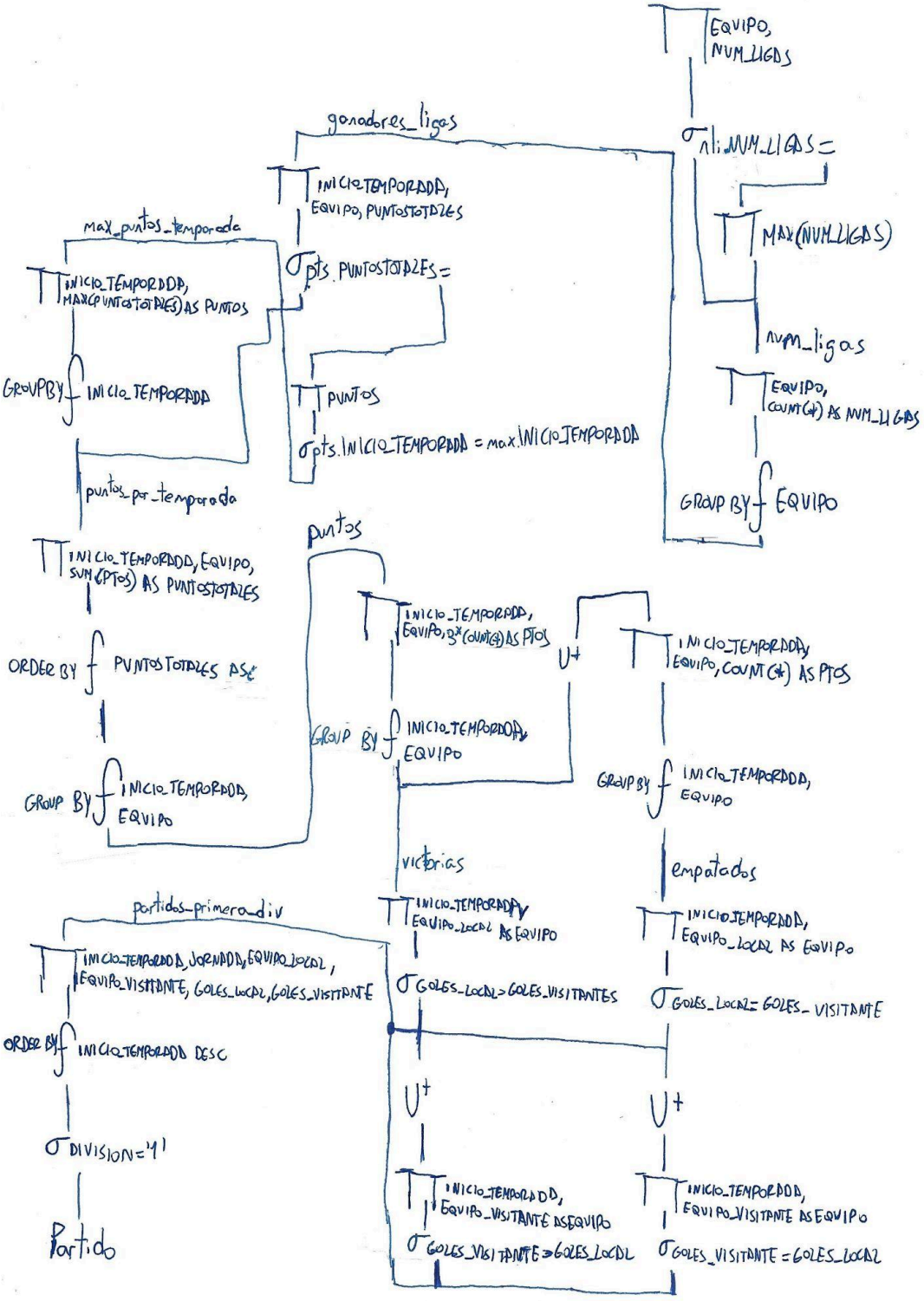
En esta vista vamos a insertar los ganadores de liga cada temporada en primera división. A partir de aquí lo que nos queda por hacer es similar a lo que hicimos con los puntos: calculamos el número de ligas y después mostramos por pantalla el que tiene el máximo número de ellas.

```
CREATE VIEW num_ligas
AS SELECT EQUIPO, COUNT(*) AS NUM_LIGAS
FROM ganadores_ligas
GROUP BY EQUIPO;
```

Aquí lo que hacemos es crear una vista en la que en cada fila aparecerá cada equipo que haya jugado en primera división con el número de ligas ganadas.

```
SELECT EQUIPO, NUM_LIGAS
FROM num_ligas nli
WHERE nli.NUM_LIGAS=(SELECT MAX(NUM_LIGAS) FROM num_ligas);
```

Por último, buscamos en el SELECT el o los equipos cuyo número de ligas coincida con el número máximo de ellas al hacer un recorrido completo de la vista del número de ligas.
Al ejecutar la consulta y realizar todas las operaciones y comprobaciones mencionadas anteriormente, nos sale como resultado que el equipo que más ligas ha ganado en primera división ha sido el Real Madrid, con 19 campeonatos. Esto difiere ligeramente de la realidad, ya que en la vida real ha ganado 17 ligas en el periodo que estudiamos en la base de datos. Esto se debe a que, hasta la temporada 95/96, el conteo de puntos era distinto al actual, ya que cada victoria sumaba 2 en vez de 3 puntos. No hemos tomado en cuenta este aspecto, y esa es la razón por lo que nos sale que el Real Madrid ha ganado 19 ligas en vez de 17.


EQUIPO, NUM_LIGAS
σ n1.NUM_LIGAS =
MAX(NUM_LIGAS)
num_ligas
EQUIPO, COUNT(*) AS NUM_LIGAS
GROUP BY EQUIPO
ganadores_ligas
INICIO_TEMPORADA, EQUIPO, PUNTOSTOTALES
σ pts.PUNTOSTOTALES =
PUNTOS
σ pts.INICIO_TEMPORADA = max.INICIO_TEMPORADA
max_puntos_temporada
INICIO_TEMPORADA, MAX(PUNTOSTOTALES) AS PUNTOS
GROUP BY INICIO_TEMPORADA
puntos_por_temporada
INICIO_TEMPORADA, EQUIPO, SUM(PTOS) AS PUNTOSTOTALES
ORDER BY PUNTOSTOTALES ASC
GROUP BY INICIO_TEMPORADA, EQUIPO
puntos
INICIO_TEMPORADA, EQUIPO, 3*(COUNT(*)) AS PTOS
GROUP BY INICIO_TEMPORADA, EQUIPO
U+
INICIO_TEMPORADA, EQUIPO, COUNT(*) AS PTOS
GROUP BY INICIO_TEMPORADA, EQUIPO
partidos_primera_div
INICIO_TEMPORADA, JORNADA, EQUIPO_LOCAL, EQUIPO_VISITANTE, GOLES_LOCAL, GOLES_VISITANTE
ORDER BY INICIO_TEMPORADA DESC
σ DIVISION='1'
Partido
victorias
INICIO_TEMPORADA, EQUIPO_LOCAL AS EQUIPO
σ GOLES_LOCAL > GOLES_VISITANTES
empatados
INICIO_TEMPORADA, EQUIPO_LOCAL AS EQUIPO
σ GOLES_LOCAL = GOLES_VISITANTE
U+
U+
INICIO_TEMPORADA, EQUIPO_VISITANTE AS EQUIPO
σ GOLES_VISITANTE > GOLES_LOCAL
INICIO_TEMPORADA, EQUIPO_VISITANTE AS EQUIPO
σ GOLES_VISITANTE = GOLES_LOCAL

Para la segunda consulta, que nos pide mostrar por pantalla aquellos equipos que hayan ascendido y después descendido en la siguiente temporada, hemos seguido el esquema de ir poco a poco creando tablas intermedias (vistas) para ir poco a poco acercándonos a la solución que nos pide el enunciado.

```
CREATE VIEW equiposSegunda
AS (SELECT DISTINCT INICIO_TEMPORADA, DIVISION, EQUIPO_LOCAL AS EQUIPO
FROM PARTIDO
WHERE DIVISION='2' AND TO_NUMBER(INICIO_TEMPORADA) >=2006);
```

La siguiente vista que hemos creado contiene todos y cada uno de los equipos que han participado alguna vez en segunda división en las últimas 10 temporadas.

```
CREATE VIEW equiposPrimera
AS (SELECT DISTINCT INICIO_TEMPORADA, DIVISION, EQUIPO_LOCAL AS EQUIPO
FROM PARTIDO
WHERE DIVISION='1' AND TO_NUMBER(INICIO_TEMPORADA) >=2006);
```

La siguiente vista contiene todos los equipos equipos que en las últimas 10 temporadas hayan participado en la primera división al menos una vez.

```
CREATE VIEW ascendidos
AS (SELECT *
FROM equiposSegunda
WHERE EQUIPO IN ( SELECT EQUIPO
FROM equiposPrimera
WHERE TO_NUMBER(INICIO_TEMPORADA)=TO_NUMBER(equiposSegunda.INICIO_TEMPORADA)+1));
```

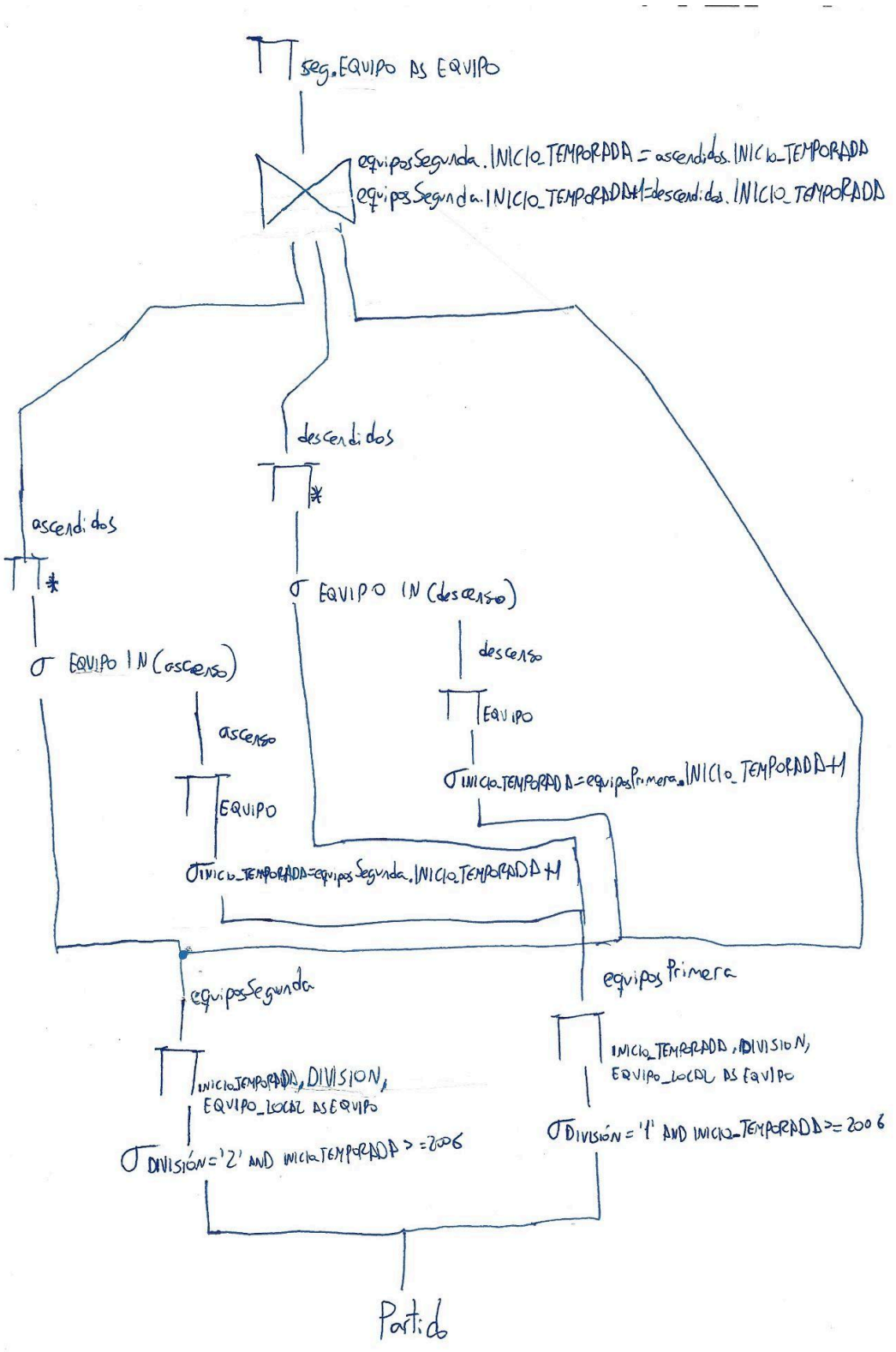
Esta vista contiene los equipos que han ascendido alguna vez durante las últimas 10 temporadas, es decir, aquellos equipos que en una temporada estuvieran en la tabla equiposSegunda y a la siguiente temporada (INICIO_TEMPORADA + 1) están en la tabla equiposPrimera.

```
CREATE VIEW descendidos
AS (SELECT *
FROM equiposPrimera
WHERE EQUIPO IN ( SELECT EQUIPO
FROM equiposSegunda
WHERE TO_NUMBER(INICIO_TEMPORADA)=TO_NUMBER(equiposPrimera.INICIO_TEMPORADA)+1));
```

Esta vista contiene los equipos que han descendido alguna vez durante las últimas 10 temporadas, es decir, aquellos equipos que en una temporada estuvieran en la tabla equiposPrimera y a la siguiente temporada (INICIO_TEMPORADA + 1) están en la tabla equiposSegunda.

```
SELECT seg.EQUIPO AS EQUIPO
FROM equiposSegunda seg
WHERE seg.EQUIPO IN(SELECT ascen.EQUIPO
        FROM ascendidos ascen
        WHERE TO_NUMBER(seg.INICIO_TEMPORADA)=TO_NUMBER(ascen.INICIO_TEMPORADA))
AND seg.EQUIPO IN(SELECT descen.EQUIPO
        FROM descendidos descen
        WHERE TO_NUMBER(seg.INICIO_TEMPORADA)+1=TO_NUMBER(descen.INICIO_TEMPORADA));
```

Por último, mostramos en pantalla aquellos equipos que en una temporada hayan ascendido y que en INICIO_TEMPORADA + 1 hayan descendido, comprobando que las temporadas de ascenso y descenso coinciden con las del equipo en cuestión.
Así, después de ejecutar la consulta, nos queda como resultado que los equipos que han ascendido en una temporada y descendido a la siguiente las 10 últimas temporadas son: Córdoba, Tenerife, Xerez. Murcia, Hercules, Deportivo de la Coruña y Numancia.


seg.EQUIPO AS EQUIPO
equiposSegunda.INICIO_TEMPORADA = ascendidos.INICIO_TEMPORADA
equiposSegunda.INICIO_TEMPORADA+1=descendidos.INICIO_TEMPORADA
descendidos
ascendidos
σ EQUIPO IN (descenso)
σ EQUIPO IN (ascenso)
descenso
EQUIPO
ascenso
EQUIPO
σ INICIO_TEMPORADA=equiposPrimera.INICIO_TEMPORADA+1
σ INICIO_TEMPORADA=equiposSegunda.INICIO_TEMPORADA+1
equiposSegunda
equiposPrimera
INICIO_TEMPORADA, DIVISION, EQUIPO_LOCAL AS EQUIPO
INICIO_TEMPORADA, DIVISION, EQUIPO_LOCAL AS EQUIPO
σ DIVISIÓN='2' AND INICIO_TEMPORADA >=2006
σ DIVISIÓN='1' AND INICIO_TEMPORADA>=2006
Partido

Para la tercera y última consulta, que nos pide las jornadas de las últimas 5 temporadas con más goles, hemos visto que tiene varias interpretaciones. Por nuestra parte, hemos decidido entender el enunciado como el muestreo de la/las jornadas de las últimas 5 temporadas que tuvieron más goles, tomando dichas jornadas en solitario y sin agruparlas con otra temporada o división.

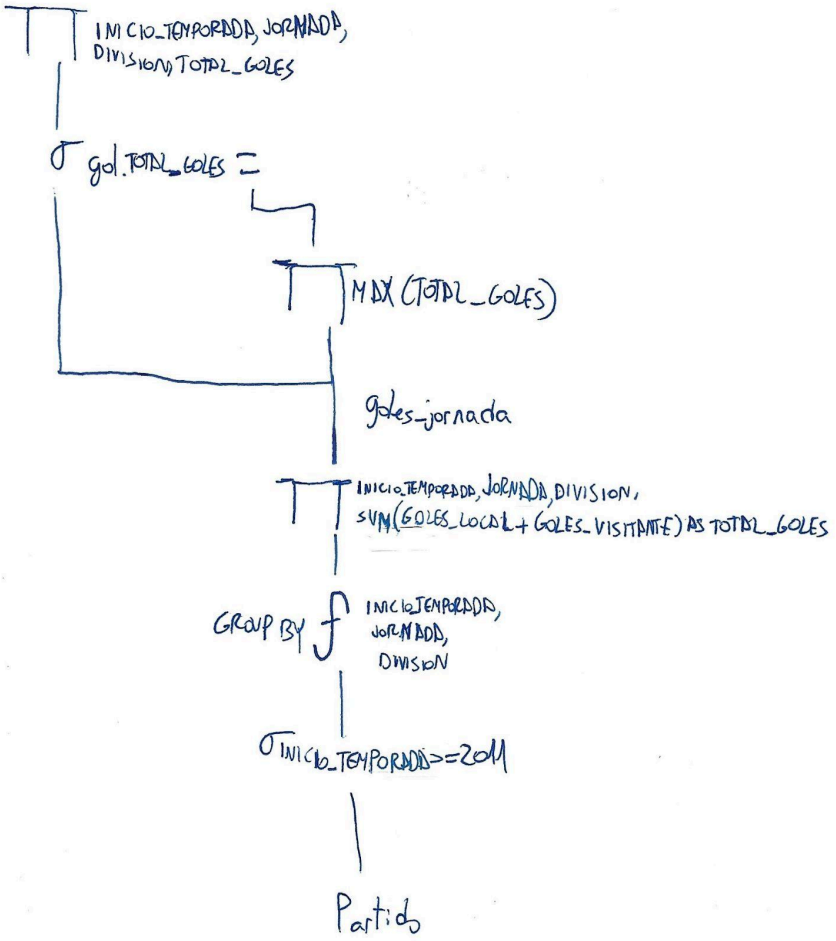
```
CREATE VIEW goles_jornada
AS SELECT INICIO_TEMPORADA, JORNADA, DIVISION, SUM(GOLES_LOCAL + GOLES_VISITANTE) AS TOTAL_GOLES
FROM Partido
WHERE TO_NUMBER(INICIO_TEMPORADA) >= 2011
GROUP BY INICIO_TEMPORADA, JORNADA, DIVISION;
```

En la siguiente vista, hemos almacenado en ella todos y cada uno de los goles de cada jornada de primera y segunda división en las últimas 5 temporadas. Esto nos deja prácticamente todo el trabajo hecho, ya que lo único que nos falta por hacer el seleccionar el máximo. La selección que nos falta sigue el esqueleto de sacar el número máximo de ligas de la primera consulta.

```
SELECT INICIO_TEMPORADA, JORNADA, DIVISION, TOTAL_GOLES
FROM goles_jornada gol
WHERE gol.TOTAL_GOLES = (SELECT MAX(TOTAL_GOLES) FROM goles_jornada);
```

En esta selección cogemos las jornadas cuyo número de goles sea el mismo que el máximo al recorrer todos los goles de todas las jornadas, y así conseguimos la solución buscada.
Al ejecutar la consulta, la solución que nos sale es la siguiente: El número máximo de goles ha sido de 42 goles, y lo han logrado 4 jornadas:

- Jornada 18 temporada 14/15 de la segunda división.
- Jornada 38 temporada 14/15 de la primera división.
- Jornada 4 temporada 14/15 de la primera división.
- Jornada 14 temporada 13/14 de la primera división.


INICIO_TEMPORADA, JORNADA, DIVISION, TOTAL_GOLES
σ gol.TOTAL_GOLES =
MAX (TOTAL_GOLES)
goles_jornada
INICIO_TEMPORADA, JORNADA, DIVISION, SUM(GOLES_LOCAL + GOLES_VISITANTE) AS TOTAL_GOLES
GROUP BY
INICIO_TEMPORADA, JORNADA, DIVISION
σ INICIO_TEMPORADA >= 2011
Partido

Análisis rendimiento

En la primera consulta usamos una gran cantidad de tablas intermedias por lo que el coste en memoria es muy alto y sería preferible optimizarlo. Intentamos varias maneras de reducirlo intentando juntar las tablas de max_puntos_por temporada y puntos por temporada con un JOIN, pero finalmente no pudimos conseguir que funcionara correctamente por lo que lo dejamos igual. Esta consulta tiene un gran número de instrucciones (98) pero la mayoría son relativamente rápidas

Las operaciones que más memoria consumen son
Tipo: HASH GROUP BY
Filas: 21048
Bytes: 1068K

Tipo: HASH GROUP BY
Filas: 21048
Bytes: 2034K

En la segunda consulta la mayoría de los costos están asociados con operaciones de acceso completo (TABLE ACCESS FULL), lo que indica que la consulta está escaneando grandes cantidades de datos. Esta consulta tiene un número reducido de instrucciones (14)

Las operaciones que más cpu consumen son

Tipo: HASH JOIN SEMI

Costo: 1364

Tipo: TABLE ACCESS FULL en la tabla PARTIDO (varias veces)
Costo: 272 (para varias instancias)

La última consulta es la más corta y por lo tanto tiene muy pocas instrucciones (8). Pese a ello, usa bastante memoria, ya que tiene que recorrer y almacenar grandes tablas. Casi todas tienen el mismo coste de 316kB de memoria. Pensamos en optimizarla usando la operación LIMIT que encontramos en internet para truncar las tablas pero al no haberla dado en clase preferimos no usarla.

Tipo : HASH GROUP BY
Costo: 273
Número Estimado de Filas: 3032
Bytes Estimados: 316K

Operación: TABLE ACCESS FULL
Tabla: GOLES_JORNADA
Número Estimado de Filas: 3032
Bytes Estimados: 316K
Costo: 273

Triggers

Los triggers son muy útiles a la hora de la implementación de base de datos si sabes que va a ser modificada a lo largo del tiempo. En esta primera base de datos hemos creado 3 triggers muy simples, algunos son equivalentes a un constraint pero podemos sacar mensajes de error por ejemplo.

El primero es un caso del que acabo de hablar. Hacemos que sea antes que la inserción o la actualización de la base de datos. Además, lo hacemos para cada fila de la tabla Temporada. Si intentamos introducir una temporada la cual su año de inicio es igual o posterior que su año final, nos saltará un mensaje de error personalizado, ya que usamos un número de código de error negativo entre -20000 y -20999.

```
CREATE OR REPLACE TRIGGER trigger_fecha_temporada
BEFORE INSERT OR UPDATE ON Temporada
FOR EACH ROW
DECLARE
BEGIN
   IF :NEW.INICIO_TEMPORADA >= :NEW.FIN_TEMPORADA THEN
        RAISE_APPLICATION_ERROR(-20001, 'La fecha de inicio no puede ser posterior o igual a la fecha de fin de
temporada.');
   END IF;
END;
```

El segundo trigger es más útil, ya que lo usamos para borrar de la tabla Nombres todos los nombres asociados a un equipo. Este trigger se dispara cuando borramos un equipo de la tabla Equipo. A diferencia del anterior trigger que se ejecutaba antes del cambio, este se ejecuta después, es decir, borramos el equipo que queramos e inmediatamente después se borrarán todos sus nombres asociados.

```
CREATE OR REPLACE TRIGGER trigger_borrar_nombres
AFTER DELETE ON Equipo
FOR EACH ROW
BEGIN
   DELETE FROM Nombres WHERE NOMBRE_EQUIPO = :OLD.NOMBRE_EQUIPO;
END;
/
```

Por último, el tercer y último trigger verifica que, al introducir un nuevo partido, sus nombres no sean iguales, ya que no sería lógico. Se hace obviamente antes de insertarlo y salta otro mensaje de error distinto al del primer trigger, ya que podrían dar lugar a confusiones. Usamos NEW, que son pseudo registros haciendo referencia a los nuevos datos a introducir, y comprobamos como un if si son iguales, en ese caso saltará el mensaje de error.

```
CREATE OR REPLACE TRIGGER
BEFORE INSERT ON Partido
FOR EACH ROW
BEGIN
   IF :NEW.EQUIPO_LOCAL = :NEW.EQUIPO_VISITANTE THEN
      RAISE_APPLICATION_ERROR(-20002, 'Los nombres de los equipos local y visitante no pueden ser iguales.');
   END IF;
END;
```

Anexo I	Carlos Solana	Diego Mateo	Daniel Simón
Diseño E/R	2	2	2
Diseño relacional y normalización	4	4	4
Creación de las tablas	2	1	4
Poblar las tablas	9	6	8
Consultas	3	8	1
Triggers	2	1	3
Memoria	4	4	3
Horas totales	26	26	25