Práctica 1

Base de datos de fútbol

Asignatura: Bases de Datos (Grupo L1.01)

Curso 2024/25 – Lunes 8:00 a 10:00

Entrega: Jueves, 13 de marzo de 2025

César Tejedo Manovel (818924@unizar.es)

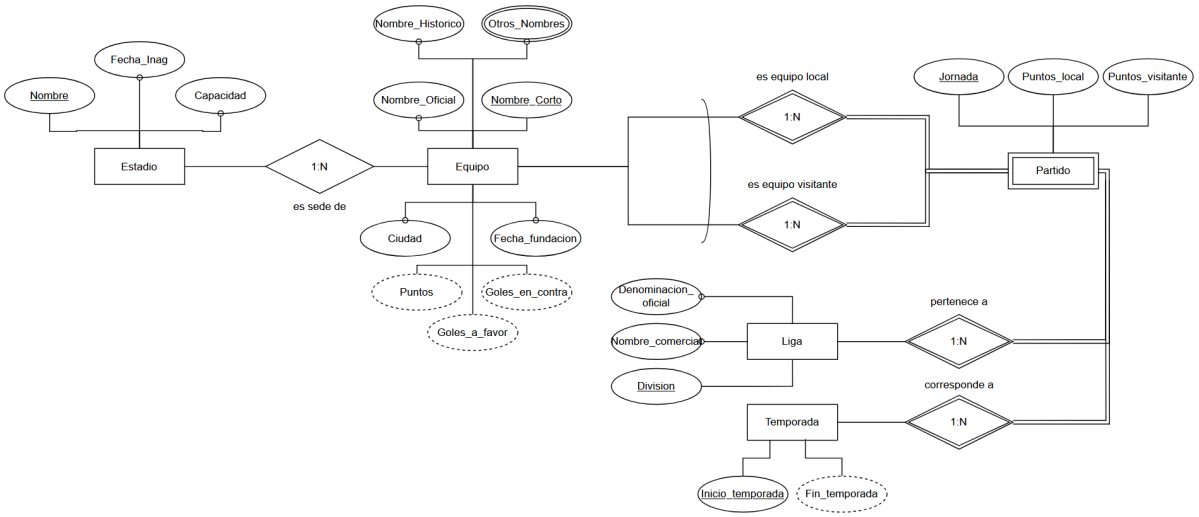
Lucía Vázquez Martín (871886@unizar.es)

Luna Zhou Chen (812103@unizar.es)

ÍNDICE

Parte 1. Creación de una base de datos

1.1. Esquema E/R



El esquema E/R diseñado consta de cinco entidades: *Estadio*, *Equipo*, *Partido*, *Liga* y *Temporada*. En esta primera versión, se han considerado las características de los datos proporcionados, lo que ha influido tanto en la opcionalidad de los atributos de la entidad *Estadio*, *Liga* y *Equipo*, así como en la elección de la clave primaria de esta última entidad. En lugar de utilizar el *Nombre_Oficial*, se hace uso de *Nombre_Corto*.

En cuanto a las restricciones, se han planteado las siguientes:

- El atributo *Inicio_temporada* debe ser posterior o igual al año 1972.
- El atributo derivado *Fin_temporada* es igual a: *Inicio_temporada* + *1*.
- El atributo derivado *Puntos* es igual a: *3* * *Victorias* + *1* * *Empates*, independientemente de si se juega como visitante o como local.
- El atributo derivado *Goles_en_contra* es igual a: *Puntos_local* + *Puntos_visitante* en función de si el equipo contrario juega como local o visitante. Mientras que *Goles_a_favor* es igual a: *Puntos_local* + *Puntos_visitante* en función de si el equipo juega como local o visitante.
- La división debe tener como valor: *1*, *2*, *Promoci* o *Descens*.

Respecto a las soluciones alternativas, se planteó representar Jornada mediante una entidad en vez de un atributo de la entidad *Partido*. En ese caso, dicha entidad estaría relacionada con *Partido*, *Liga* y *Temporada* mediante relaciones 1:N, dado que una jornada puede tener muchos partidos, pero un partido solamente puede estar asociado a una jornada. Además, *Jornada* tendría un atributo que permitiese contabilizar el número de jornada en la que se encuentra. Finalmente, se decidió añadir *jornada* directamente como atributo para simplificar el esquema.

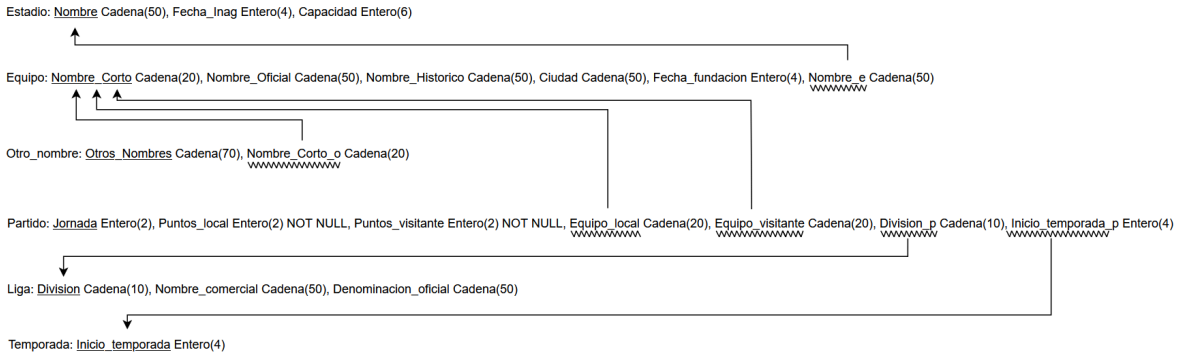
1.2. Esquema relacional

Para representar el esquema relacional y asegurar el cumplimiento de las reglas de normalización, se realizaron una serie de modificaciones.

En primer lugar, para convertir el atributo multivaluado *Otros_Nombres* de la entidad *Equipo* en 1ª FNBC se creó una nueva entidad llamada *Otro_nombre*, relacionada con *Equipo* mediante una relación 1:N. De esta manera, el atributo dejó de formar parte de la entidad *Equipo* y pasó a ser la clave primaria de la nueva entidad.

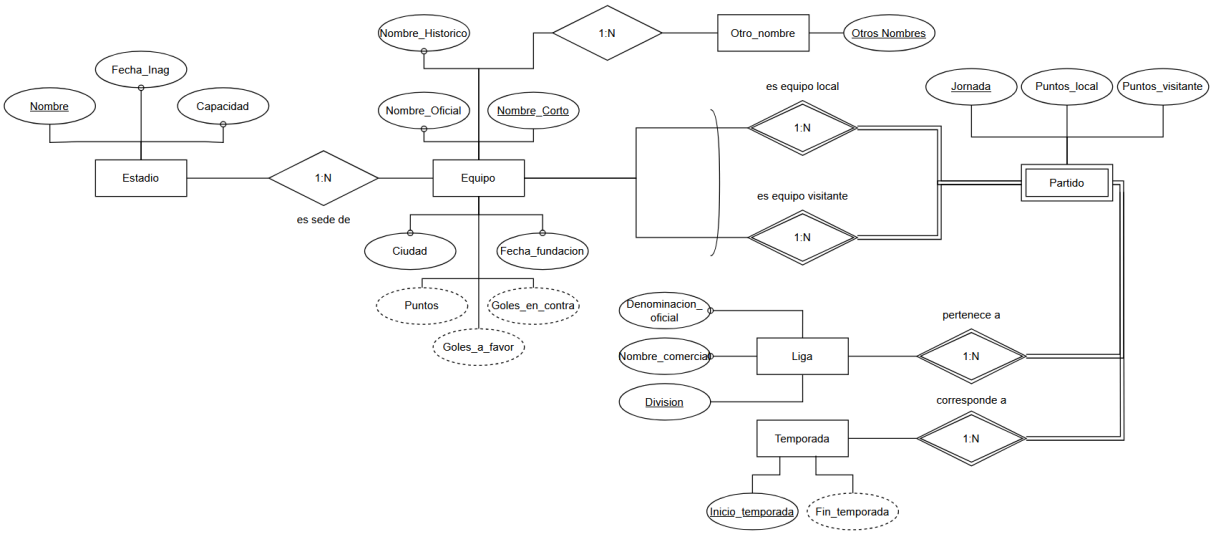
Tras esto, se verificó que ningún atributo no clave dependiera de una parte de la clave primaria ni de un conjunto de atributos no clave. Gracias a esto, se garantizó que el modelo cumpliera con la 2ª FNBC y la 3ª FNBC.

Una vez realizadas las transformaciones mencionadas anteriormente, el esquema relacional fue el siguiente:



Restricciones:

- En un partido, *Equipo_local* debe ser distinto a *Equipo_visitante*.
- En una temporada, *Inicio_temporada* debe ser mayor o igual a 1972.



1.3. Sentencias SQL

Para representar el modelo E/R normalizado mostrado anteriormente, es necesario crear 6 tablas en un determinado orden, que es: *Estadio*, *Equipo*, *Otro_nombre*, *Temporada*, *Liga* y *Partido*.

```sql
CREATE TABLE Estadio (
	Nombre		VARCHAR(50) PRIMARY KEY,
	Fecha_Inag	NUMBER(4),
	Capacidad	NUMBER(6)
);

CREATE TABLE Equipo (
	Nombre_Corto		VARCHAR(20) PRIMARY KEY,
	Nombre_Oficial	VARCHAR(50),
	Nombre_Historico	VARCHAR(50),
	Ciudad			VARCHAR(50),
	Fecha_fundacion	NUMBER(4),
	Nombre_e		VARCHAR(50),
	FOREIGN KEY(Nombre_e) REFERENCES Estadio(Nombre)
);

CREATE TABLE Otro_nombre (
	Otros_Nombres		VARCHAR(70) PRIMARY KEY,
	Nombre_Corto_o VARCHAR(20),
	FOREIGN KEY(Nombre_Corto_o) REFERENCES Equipo(Nombre_Corto)
);

CREATE TABLE Temporada (
	Inicio_temporada NUMBER(4) PRIMARY KEY
);

CREATE TABLE Liga (
	Division			VARCHAR(10) PRIMARY KEY,
	Nombre_comercial		VARCHAR(50),
	Denominacion_oficial	VARCHAR(50)
);

CREATE TABLE Partido (
	Jornada		NUMBER(2),
	Puntos_local		NUMBER(2) NOT NULL,
	Puntos_visitante	NUMBER(2) NOT NULL,
	Equipo_local		VARCHAR(20),
	Equipo_visitante	VARCHAR(20),
	Division_p		VARCHAR(10),
	Inicio_temporada_p	NUMBER(4),
	FOREIGN KEY (Equipo_local) REFERENCES Equipo(Nombre_Corto),
	FOREIGN KEY (Equipo_visitante) REFERENCES Equipo(Nombre_Corto),
	FOREIGN KEY (Division_p) REFERENCES Liga(Division),
	FOREIGN KEY (Inicio_temporada_p) REFERENCES Temporada(Inicio_temporada),
	PRIMARY KEY (Jornada, Equipo_local, Equipo_visitante, Division_p, Inicio_temporada_p)
);
```

Para la tabla de clasificaciones, hemos decidido usar una vista que genere la tabla, ya que todos los datos de la clasificación pueden ser extraídos de los datos del resto de tablas. La instrucción SQL usada es la siguiente:

```sql
CREATE VIEW Clasificacion AS
SELECT INICIO_TEMPORADA_P, DIVISION_P, EQUIPO, SUM(PUNTOS) AS PUNTOS_TOTALES,
SUM(GOLES_RECIBIDOS) AS GOLES_RECIBIDOS
FROM (
    SELECT INICIO_TEMPORADA_P, DIVISION_P, EQUIPO_LOCAL AS EQUIPO, 3 AS PUNTOS,
    PUNTOS_VISITANTE AS GOLES_RECIBIDOS
    FROM PARTIDO
    WHERE PUNTOS_LOCAL > PUNTOS_VISITANTE
    UNION ALL
    SELECT INICIO_TEMPORADA_P, DIVISION_P, EQUIPO_VISITANTE AS EQUIPO, 3 AS PUNTOS,
    PUNTOS_LOCAL AS GOLES_RECIBIDOS
    FROM PARTIDO
    WHERE PUNTOS_VISITANTE > PUNTOS_LOCAL
    UNION ALL
    SELECT INICIO_TEMPORADA_P, DIVISION_P, EQUIPO_LOCAL AS EQUIPO, 1 AS PUNTOS,
    PUNTOS_VISITANTE AS GOLES_RECIBIDOS
    FROM PARTIDO
    WHERE PUNTOS_LOCAL = PUNTOS_VISITANTE
    UNION ALL
    SELECT INICIO_TEMPORADA_P, DIVISION_P, EQUIPO_VISITANTE AS EQUIPO, 1 AS PUNTOS,
    PUNTOS_LOCAL AS GOLES_RECIBIDOS
    FROM PARTIDO
    WHERE PUNTOS_VISITANTE = PUNTOS_LOCAL
)
GROUP BY INICIO_TEMPORADA_P, DIVISION_P, EQUIPO
ORDER BY INICIO_TEMPORADA_P ASC, DIVISION_P ASC, PUNTOS_TOTALES DESC, GOLES_RECIBIDOS ASC;
```

Parte 2. Introducción de datos y ejecución de consultas

2.1. Pasos seguidos para poblar la BD

Antes de poblar la base de datos, fue necesario modificar los datos de *LigaHost.csv* debido a que el equipo de Málaga había cambiado su *nombre_Corto* a lo largo de los años, –*Málaga (C.D.)*, *Málaga*–. Cómo lo utilizamos como clave primaria en la entidad *Equipo*, era imprescindible unificarlo para evitar que se interpretara como dos equipos distintos. Por ello, se tomó la decisión de reemplazar todas las variantes del nombre corto por *Málaga*.

Con respecto a la población, primero se realizaron las sentencias SQL. Posteriormente, se crearon los ficheros .ctl y .csv para cada entidad, donde fue necesario seleccionar las columnas de *LigaHost.csv* que nos interesaban para la población de cada una de las tablas. Además, fue necesario la creación del fichero *Liga.csv* en base a datos de Internet, debido a que esos datos no estaban contenidos en *LigaHost.csv*.

Poblamos las tablas en el mismo orden en que fueron creadas: *Estadio*, *Equipo*, *Otro_nombre*, *Temporada*, *Liga*, *Partido*.

Los primeros datos cargados fueron de la tabla *Estadio*, por lo que comprobamos que se hubiesen rellenado correctamente. Sin embargo, nos dimos cuenta que había menos datos de los que aparecían en el fichero .csv. La razón fue porque las filas que contenían valores que estaban en blanco eran ignorados, puesto que tenían que ser tratados como valores nulos. Así pues, se tuvo que añadir la siguiente línea en los ficheros .ctl: *trailing nullcols*

La instrucción *trailing nullcols* en los ficheros .ctl permite que las filas con valores en blanco al final no sean ignoradas y sean tratadas como valores nulos en la carga de datos. Esto asegura que todos los registros del fichero.csv se importen correctamente en la tabla.

Otro problema encontrado fue la inconsistencia entre los nombres de atributos, debido a tildes y mayúsculas entre las tablas creadas y cargadas. Por ejemplo, en la creación de la tabla *Equipo* se usó *Fecha_fundacion* mientras que en la carga usamos *Fecha_Fundación*.

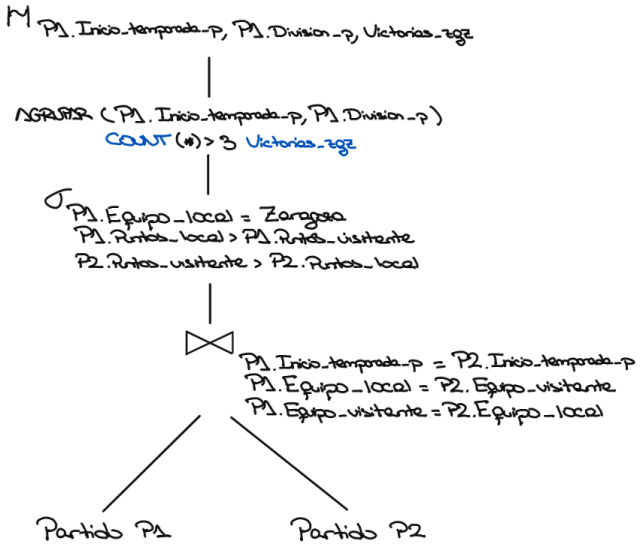
Por último, el mayor problema que surgió fue en la carga de la tabla *Partido*, pues la modificación de los distintos ficheros se realizaron en el S.O Windows, el cual guardó los saltos de línea del .csv con el formato CR-LF. Cómo *lab000* está basado en Linux, usa el formato LF, por lo que no cargaba correctamente el último dato de cada fila. Otro problema que hubo fue el charset usado. Al crear el fichero .csv con los datos de la liga, este se guardó como *UTF-8 BOM*. El BOM *(Byte Order Mark)* son cuatro bytes iniciales que sirven para indicar el *endianness* en el que el fichero está codificado. Esto provocaba que, a la hora de cargar los datos de la liga, la primera fila, que contiene la 1ª división, no la guardase como división “1”, sino como “????1” (Siendo los interrogantes los bytes iniciales), lo que hacía imposible cargar los partidos que estaban en 1ª división. Esto lo arreglamos borrando dichos bytes iniciales, ya que no son necesarios para nuestro fichero.

Cuando había problemas en la creación de tablas, había que borrarlas para poder actualizar los errores. De modo que el orden para borrarlas es el siguiente: *Otro_nombre*, *Partido*, *Temporada*, *Liga*, *Equipo*, *Estadio*. Esto es así debido a las dependencias entre atributos de cada tabla. De lo contrario, saltarían errores al borrarlo arbitrariamente.

2.2. Consultas SQL

1) Temporadas en las que el Real Zaragoza haya ganado al menos a 4 equipos en ambos partidos de la temporada. Indicar la división, la temporada y el número de goles que marcó el Zaragoza en dicha temporada.

Álgebra Relacional:



Sentencia SQL:

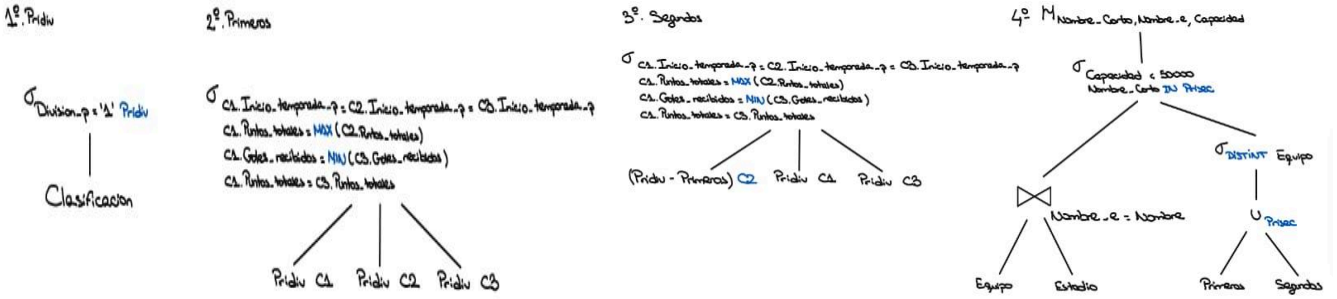
```sql
SELECT P1.INICIO_TEMPORADA_P, P1.DIVISION_P, COUNT(*) AS VICTORIAS_ZGZ FROM PARTIDO P1
JOIN PARTIDO P2 ON P1.INICIO_TEMPORADA_P=P2.INICIO_TEMPORADA_P AND
P1.EQUIPO_LOCAL=P2.EQUIPO_VISITANTE AND P1.EQUIPO_VISITANTE=P2.EQUIPO_LOCAL
--Unimos los partidos ida y vuelta de cada temporada
WHERE P1.EQUIPO_LOCAL='Zaragoza' AND P1.PUNTOS_LOCAL > P1.PUNTOS_VISITANTE AND
P2.PUNTOS_VISITANTE > P2.PUNTOS_LOCAL
--Filtramos los partidos que haya participado el zaragoza y que haya ganado en ida y vuelta
GROUP BY P1.INICIO_TEMPORADA_P, P1.DIVISION_P
--Agrupamos los resultados por temporada y división
HAVING COUNT(*) > 3
--Filtramos las temporadas donde el zaragoza haya ganado ida y vuelta a 4 o más equipos
ORDER BY P1.INICIO_TEMPORADA_P ASC;
```

Resultados Consulta:

```
INICIO_TEMPORADA_P DIVISION_P VICTORIAS_ZGZ
------------------ ---------- -------------
              1982          1             5
              1985          1             4
              1998          1             4
              2002          2             6
              2008          2             6
```

2) Listado de equipos con estadios de menos de 50.000 plazas que han quedado en primer o segundo puesto de la liga (en primera división) al menos una vez.

Álgebra Relacional:



Sentencia SQL:

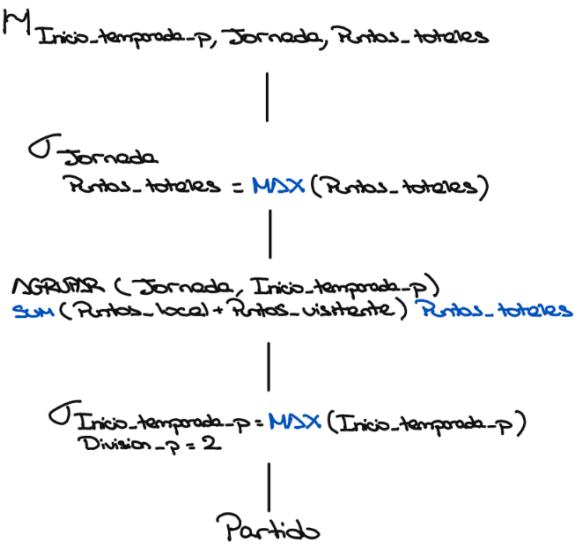
```
WITH PRIDIV AS (SELECT * FROM CLASIFICACION WHERE DIVISION_P = '1'),
--Para buscar solo en la 1ª división
PRIMEROS AS (SELECT * FROM PRIDIV C1 WHERE C1.PUNTOS_TOTALES=(
                    SELECT MAX(C2.PUNTOS_TOTALES) FROM PRIDIV C2
                    WHERE C2.INICIO_TEMPORADA_P = C1.INICIO_TEMPORADA_P )
              AND C1.GOLES_RECIBIDOS = (SELECT MIN(C3.GOLES_RECIBIDOS) FROM PRIDIV C3
                    WHERE C3.INICIO_TEMPORADA_P = C1.INICIO_TEMPORADA_P
                    AND C3.PUNTOS_TOTALES = C1.PUNTOS_TOTALES )),
--Obtenemos los 1º de cada temp, y si hay empate, el que tenga menos goles recibidos
SEGUNDOS AS (SELECT * FROM PRIDIV C1 WHERE C1.PUNTOS_TOTALES=(SELECT MAX(C2.PUNTOS_TOTALES)
                    FROM (SELECT * FROM PRIDIV MINUS SELECT * FROM PRIMEROS) C2
                    WHERE C2.INICIO_TEMPORADA_P=C1.INICIO_TEMPORADA_P)
              AND C1.GOLES_RECIBIDOS=(SELECT MIN(C3.GOLES_RECIBIDOS) FROM PRIDIV C3
                    WHERE C3.INICIO_TEMPORADA_P=C1.INICIO_TEMPORADA_P
                    AND C3.PUNTOS_TOTALES=C1.PUNTOS_TOTALES))
--Repetimos, pero eliminando el primero de cada temp, para obtener como "nuevo 1º" el equipo
--en 2ª pos de cada temp
SELECT NOMBRE_CORTO, NOMBRE_E, CAPACIDAD FROM EQUIPO JOIN ESTADIO ON NOMBRE_E = NOMBRE
WHERE CAPACIDAD < 50000 AND NOMBRE_CORTO IN ( SELECT DISTINCT EQUIPO FROM PRIMEROS UNION
SELECT DISTINCT EQUIPO FROM SEGUNDOS );
--Filtramos aquellos cuyo estadio tiene un aforo de menos de 50K y este en la lista de 1º o 2º
```

Resultado Consulta:

```
NOMBRE_CORTO    NOMBRE_E      CAPACIDAD
--------------- ------------- ----------
Dptivo. Coru??a Abanca-Riazor 32490
Real Sociedad   Reale Arena   39313
Valencia        Mestalla      49430
Zaragoza        La Romareda   33608
```

3) Jornada en la que se han marcado más goles de la última temporada de segunda división. Indicar temporada, jornada y número de goles.

Álgebra Relacional:



Sentencia SQL:

```
SELECT INICIO_TEMPORADA_P, JORNADA, SUM(PUNTOS_LOCAL + PUNTOS_VISITANTE) AS PUNTOS_TOTALES
FROM PARTIDO
WHERE INICIO_TEMPORADA_P = (SELECT MAX(INICIO_TEMPORADA_P) FROM PARTIDO) AND DIVISION_P = '2'
--Filtramos por la última temporada (Aquella con mayor año), y de 2ª división
GROUP BY JORNADA, INICIO_TEMPORADA_P
--Agrupamos por jornada y temporada y sumamos los goles marcados en todos los partidos de cada
--temporada
HAVING SUM(PUNTOS_LOCAL+PUNTOS_VISITANTE) = (SELECT MAX(TOTAL_G) AS PUNTOS_TOTALES
       FROM (SELECT SUM(PUNTOS_LOCAL + PUNTOS_VISITANTE) AS TOTAL_G FROM PARTIDO
              WHERE INICIO_TEMPORADA_P = (SELECT MAX(INICIO_TEMPORADA_P) FROM PARTIDO) AND
              DIVISION_P = '2'
              GROUP BY JORNADA));
--Repetimos el proceso y nos quedamos con la jornada que tiene  el máximo de goles --marcados.
```

Resultado Consulta:

```
INICIO_TEMPORADA_P JORNADA PUNTOS_TOTALES
------------------ ------- --------------
              2015       1             37
```

Parte 3. Diseño físico

3.1. Problemas de rendimiento

Para evaluar el rendimiento de las consultas en la base de datos Oracle, se han utilizado herramientas de análisis como *EXPLAIN PLAN* y *DBMS_XPLAN.DISPLAY()*. Esto ha permitido evaluar el coste de ejecución y detectar posibles mejoras en su rendimiento.

A continuación, se presentan los costes y tiempos promedios obtenidos para cada consulta:

- Consulta 1: 101, 4 segundos
- Consulta 2: 2008, 15 segundos
- Consulta 3: 138, 4 segundos

Debido al alto coste de la segunda consulta, decidimos darle un enfoque distinto haciendo uso de *JOIN* para simplificar la sintaxis y reducir costes:

```
SELECT NOMBRE_CORTO, NOMBRE_E, CAPACIDAD FROM EQUIPO JOIN ESTADIO ON NOMBRE_E = NOMBRE
WHERE CAPACIDAD < 50000 AND NOMBRE_CORTO IN (SELECT DISTINCT C1.EQUIPO FROM CLASIFICACION C1
JOIN CLASIFICACION C2 ON C1.INICIO_TEMPORADA_P = C2.INICIO_TEMPORADA_P
AND C1.DIVISION_P = C2.DIVISIÓN_P AND (C1.PUNTOS_TOTALES < C2.PUNTOS_TOTALES
OR (C1.PUNTOS_TOTALES=C2.PUNTOS_TOTALES AND C1.GOLES_RECIBIDOS>C2.GOLES_RECIBIDOS))
--Unimos los equipos de C2 que estén en la misma temp y div que C1 y que tengan mas puntos
--que los equipos de C1 ó tengan menos goles recibidos (Están por encima en la clasificación)
WHERE C1.DIVISION_P = '1'
GROUP BY C1.INICIO_TEMPORADA_P, C1.EQUIPO
HAVING COUNT(C2.EQUIPO) < 2);
--Filtramos los equipos que no tengan más de 1 ó 2 equipos por encima de él (Son 1º ó 2º)
```

Ahora el coste de la consulta se reduce a 570, un coste mucho menor a cambio de un incremento en el tiempo de realizar la consulta, de 15 a 25 segundos. Además, durante el análisis, se observó que la inclusión de la cláusula *DISTINCT* al seleccionar los equipos primeros y segundos (ya visible en el código anterior), se redujo el tiempo sin aumentar apreciablemente el coste, de 25 a 17 segundos.

Para evaluar el rendimiento de cada consulta, se han seguido los siguientes pasos:

1. Al inicio de la sesión de SQLPlus, ejecutar el comando *SET TIMING ON* para poder ver el tiempo de ejecución de las consultas.
2. Incluir en la primera línea de la consulta el comando *EXPLAIN PLAN FOR*.
3. Ejecutar la consulta SQL de manera habitual mediante el uso de *@nombreFichero*.
4. Visualizar el plan de ejecución con *SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())*.
5. Analizar el coste de ejecución y proponer mejoras para optimizar la consulta.

Estos análisis han permitido identificar oportunidades de mejora, asegurando que las consultas sean más eficientes en términos de tiempo y recursos utilizados.

3.2. Triggers

En este apartado se identifican restricciones que Oracle no puede verificar automáticamente con la estructura de la base de datos ni con restricciones estándar como *PRIMARY KEY*, *FOREIGN KEY*, *CHECK*…. Para garantizar el cumplimiento de estas reglas es necesario implementar triggers que validen y mantengan la integridad de los datos.

Aquí algunas restricciones que no pueden ser manejadas únicamente con *CREATE TABLE*:

#	Restricción no verificable por Oracle	Explicación
1	Evitar que un equipo juegue contra sí mismo.	CHECK no puede comparar valores dentro de la misma fila.
2	Evitar que dos equipos jueguen entre ellos más de dos veces en una temporada.	Se necesita una validación basada en múltiples filas para contar cuántos enfrentamientos existen entre los mismos equipos en una temporada.
3	Evitar que un equipo juegue más de un partido en una misma jornada.	FOREIGN KEY no permite validar restricciones basadas en múltiples filas.
4	Evitar que el total de puntos de un equipo sea negativo.	Oracle no tiene restricciones para validar valores calculados en función de otras filas.
5	Restringir que en los dos encuentros entre dos equipos en una temporada se juegue una vez como local y otra como visitante.	FOREIGN KEY solo valida que los equipos existan en la tabla, pero no puede controlar cómo se asignan los roles de local y visitante en cada partido.
6	Restringir la eliminación de temporadas si existen partidos asociados.	FOREIGN KEY no permite definir reglas condicionales para la eliminación.
7	Eliminar el estadio de un equipo junto con su referencia.	Se debe actualizar el atributo del equipo a NULL al eliminar el estadio.

De estos problemas mencionados, se han seleccionado tres restricciones de distinta naturaleza, las cuales se resuelven a continuación mediante triggers.

Restricción 1: Evitar que un equipo juegue contra sí mismo

La tabla *Equipo* no debe permitir que en la tabla *Partido* se inserten registros donde el equipo local y el equipo visitante sean el mismo.

```
CREATE OR REPLACE TRIGGER trg_BI_equipos_distintos
BEFORE INSERT ON Partido
FOR EACH ROW
WHEN (NEW.Equipo_local = NEW.Equipo_visitante)
--A la hora de insertar un partido, comprobamos si el equipo local es también el equipo
--visitante, y en ese caso lanzamos error.
BEGIN
	raise_application_error (-20000, 'El equipo local y el equipo visitante no pueden ser
	el mismo.');
END;
/
```

Este trigger se ejecuta antes de una inserción o actualización en la tabla *Partido*. Si el equipo local y el visitante son iguales, se lanza un error y se bloquea la operación.

Restricción 2: Evitar eliminación de temporadas si existen partidos asociados

La tabla *Temporada* no debe permitir eliminar una tupla en la que el valor de *Inicio_temporada* tenga algún partido asociado dentro de la tabla *Partido*.

```
CREATE OR REPLACE TRIGGER trg_BD_eliminar_temp
BEFORE DELETE ON Temporada
FOR EACH ROW
DECLARE
	partidosTemp NUMBER;
BEGIN
	SELECT COUNT(*) INTO partidosTemp FROM Partido
	WHERE Partido.Inicio_temporada_p = :OLD.Inicio_temporada;
	--Contamos los partidos cuya temp sea la temp a borrar. Si no es 0, saltamos error
	IF partidosTemp > 0 THEN
		raise_application_error (-20000, 'No se puede borrar una temporada si tiene
		partidos asociados.');
	END IF;
END;
/
```

Antes de eliminar una temporada, cuenta cuántos partidos están registrados en la tabla *Partido* con la misma temporada. Si existen partidos, se lanza un error y se bloquea la eliminación

Restricción 3: Eliminar el estadio de un equipo junto con su referencia

Cuando se elimina un estadio de la tabla *Estadio*, es necesario actualizar la información en la tabla *Equipo* para que los equipos que usaban ese estadio tengan NULL en su *FOREIGN KEY*.

```
CREATE OR REPLACE TRIGGER trg_AD_eliminar_estadio
AFTER DELETE ON Estadio
FOR EACH ROW
BEGIN
        UPDATE Equipo
        SET Equipo.Nombre_e = NULL
        WHERE Equipo.Nombre_e = :OLD.Nombre;
        --Actualizamos el nombre del estadio del equipo a NULL si es el estadio a borrar.
END;
/
```

Después de eliminar un estadio, se actualiza la tabla *Equipo* para asignar NULL a los equipos que tenían ese estadio registrado. Este trigger garantiza que no queden referencias huérfanas en la base de datos.

Anexo 1. Coordinación del grupo de trabajo

La división del trabajo y las horas dedicadas en cada parte no se han repartido entre los tres integrantes, ya que al tratarse del primer contacto con un SGBD, se decidió que cada uno realizase todas las tareas. Luego, se compartían y comparaban los resultados y dudas para tomar una decisión.

Por otra parte, la distribución del trabajo ha ayudado a que todos los integrantes pudiesen organizarse respecto a otras asignaturas. Se incluye el calendario del trabajo semanal, incluyendo las horas dedicadas tanto a la base de datos como al desarrollo de la memoria.

SEMANAS	Tareas realizadas	Horas
SEMANA 1	Esquema E/R, esquema relacional, normalización.	12
SEMANA 2	Sentencias SQL para crear las tablas.	3
SEMANA 3	Poblar las tablas, consultas SQL.	20
SEMANA 4	Análisis de rendimiento, implementación de triggers.	5
		40