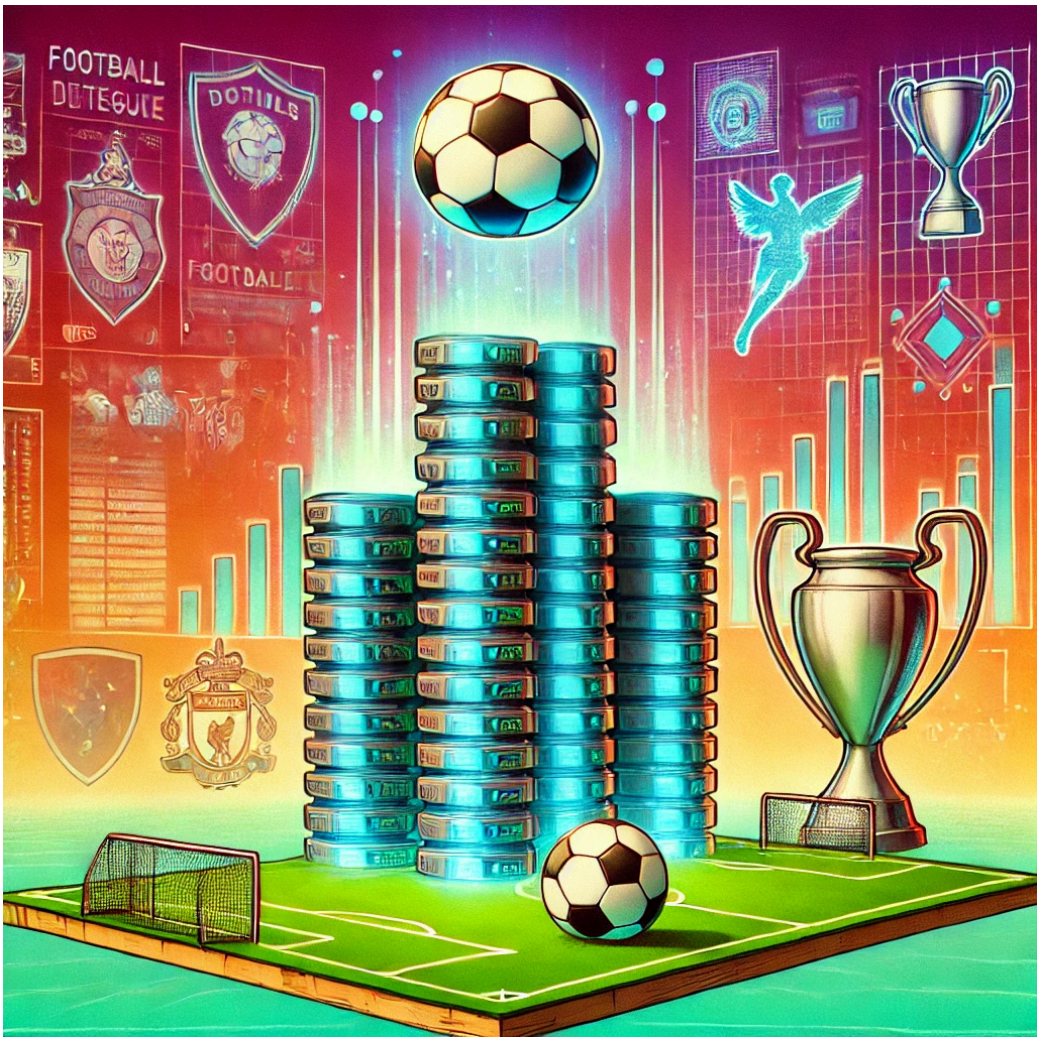
DATABASES

COURSE 2024-2025

Memory First Database: Football League

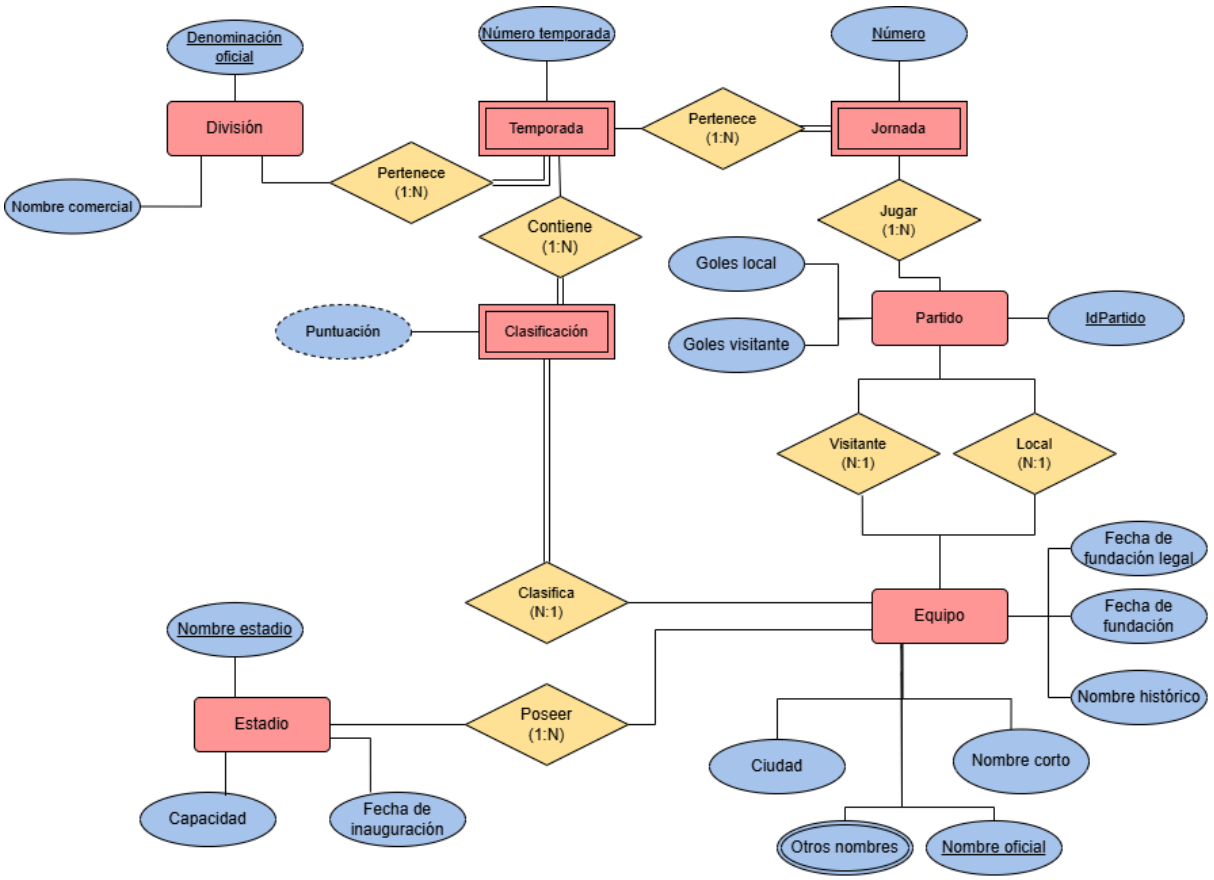


Group:J2.10
Delivery date:14/03/2025
Authors:
- Fernando Pastor Peralta (897113) .....897113@unizar.es
- Juan José Muñoz Lahoz (902677) .....902677@unizar.es
- Marcos San Julián Fuertes (899849) .....899849@unizar.es

INDEX:

Part 1: Creating a database

1.1. Global E/R Scheme and List of Restrictions

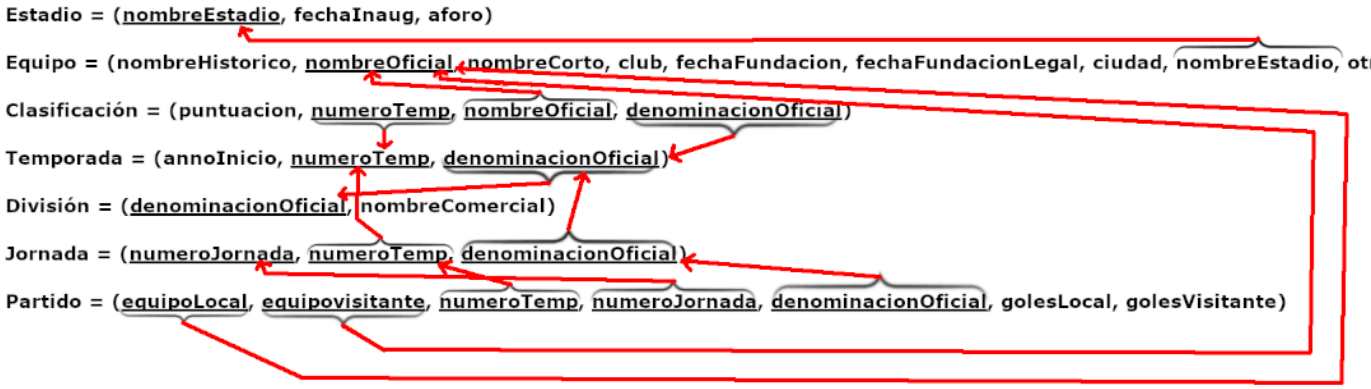


Restrictions:
- It can't be done face a teamagainst himself.
- It will not be possible to insert a match that is played in a season whose initial year has not yet arrived at the time of insertion.
- The score will be calculated as (games won*3 + games tied).
- The positions can be queried using an SQL query.
- A team can only have one stadium.
- You cannot include seasons with a year earlier than 1972.
+ restrictions in the section 3.2.

Other alternative options for creating the E/R scheme:
- Remove the Stadium entity and put its attributes as attributes of the Team entity.
- Remove the Classification entity and implement it as an attribute to the relationship between Day and Team.
- The Entity ClassificationIt depends on the day, not the season.

1.2. Relational schema corresponding to the E/R and its normalization

The relational model of our database would look like this:

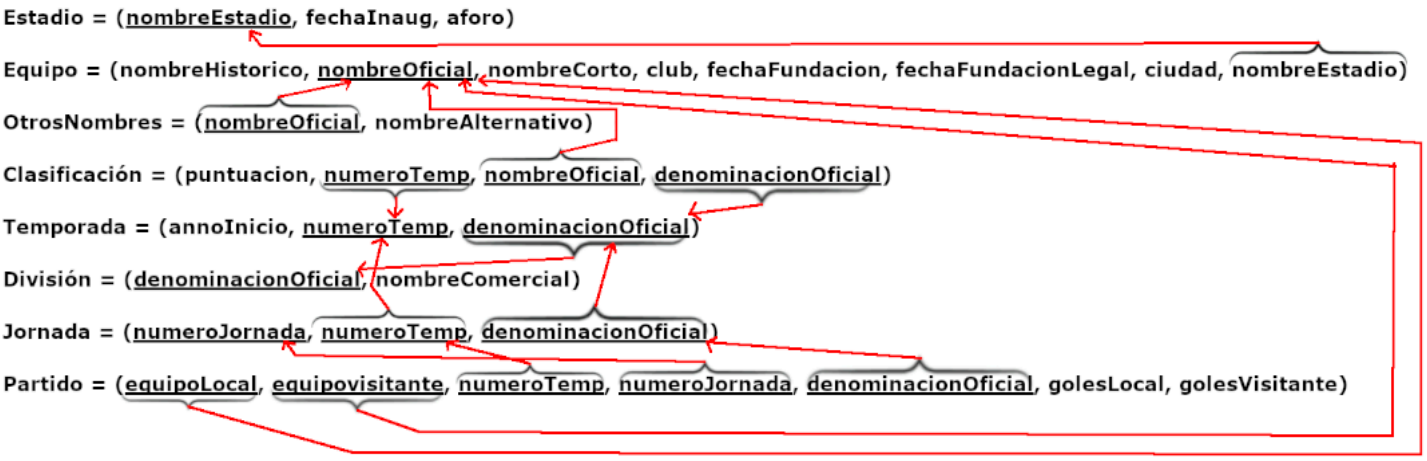


We can see that, at first glance, this relational model scheme is not normalized: in the team entity there is a multivalued attribute calledotherNamesTo normalize our schema, we need to get rid of that attribute. To do this, we will create another entity with the same name, whose attributes areOfficial nameas the primary and external key of the Team, andalternativeName, which indicates the alternative name that said team has.

Once this is done, we will have our schema in 1st normal form. Now, this must be in Boyce-Codd Normal Form, so let's see what needs to be modified to achieve this:

- Second Normal Form: 1NF + No functional dependencies on part of the key.
  This is true in our database since there is no entity whose primary key depends on only a part of the primary key of another entity.
- Third Normal Form: 2NF + There are no transitive functional dependencies. This holds true in our database since no table contains attributes that depend on another attribute that is not a primary key. This would be the case if, for example, instead of having the stadium entity, we stored the stadium data in the Team table for each team entry, potentially repeating the information in the database.
- Boyce-Codd Normal Form: All functional dependencies depend on the key. This holds true in our database, since all entities whose key depends on the key of another entity always depend on the entire primary key, neither a non-primary key nor a partial one.

After normalizing the database, the relational schema would look like this:



1.3. SQL statements for creating tables

```
/* Creationof the tables */
CREATE TABLE Estadio (
	stadium name		VARCHAR(50) PRIMARY KEY,
	dateInaug		NUMBER,
	forum			NUMBER
);

CREATE TABLE Team (
	historical name		VARCHAR(70),
	Official name		VARCHAR(30) PRIMARY KEY,
	shortname		VARCHAR(30),
	Foundation date		NUMBER,
	LegalFoundationDate	NUMBER,
	city			VARCHAR(50),
	stadium name	VARCHAR(30),
	FOREIGN KEY (stadium name) REFERENCES Stadium(stadium name)
);

CREATE TABLE OtrosNombres (
	Official name		VARCHAR(30) PRIMARY KEY,
	alternativeName		VARCHAR(30),
	FOREIGN KEY (Official name) REFERENCES Team(Official name)
);

CREATE TABLE Division (
	Official designation	VARCHAR(10) PRIMARY KEY,
	trade name		VARCHAR(70)
);

CREATE TABLE Season (
	annoInicio		NUMBER,
	numeroTemp		NUMBER, -- annoInicio- 1972 (year the first season
began)
	Official designation VARCHAR(10),
	PRIMARY KEY (numeroTemp, Official designation),
	FOREIGN KEY (Official designation) REFERENCES Division(Official designation)
);

CREATE TABLE Classification (
	punctuation		NUMBER,
	numeroTemp		NUMBER,
	Official name		VARCHAR(70),
	Official designation	VARCHAR(10),
	PRIMARY KEY(numeroTemp, Official designation, Official name),
	FOREIGN KEY (numeroTemp, Official designation) REFERENCES Season(numeroTemp,
Official designation),
	FOREIGN KEY (Official name) REFERENCES Team(Official name)
);
```

```
CREATE TABLE Journal (
      issue of the day          NUMBER,
      numeroTemp                NUMBER,
      Official designation      VARCHAR(10),
      PRIMARY KEY(issue of the day, numeroTemp, Official designation),
      FOREIGN KEY (numeroTemp, Official designation) REFERENCES Season(numeroTemp,
Official designation)
);

CREATE TABLE Match (
      local team                VARCHAR(30),
      Visiting team             VARCHAR(30),
      numeroTemp                NUMBER,
      issue of the day          NUMBER,
      Official designation      VARCHAR(10),
      golesLocal                NUMBER,
      Visitor goals             NUMBER,
      PRIMARY KEY(local team,VisitingTeam,TempNumber,issue of the day,Official
designation),
      FOREIGN KEY (local team) REFERENCES Team(Official name),
      FOREIGN KEY (Visiting team) REFERENCES Team(Official name),
      FOREIGN KEY (numeroJornada, numeroTemp, Official designation) REFERENCES
Day(issue of the day, numeroTemp, Official designation)
);

CREATE TABLE tablaTemporal (
    annoInicio                  NUMBER,
    yearFin                     NUMBER,
    Official designation        VARCHAR(10),
    issue of the day                  NUMBER,
    local team                  VARCHAR(30),
      Visiting team             VARCHAR(30),
    golesLocal                  NUMBER,
      Visitor goals             NUMBER,
    alternativeName             VARCHAR(70),
    city                        VARCHAR(50),
    Foundation date             NUMBER,
    LegalFoundationDate         NUMBER,
    historical name             VARCHAR(70),
    stadium name                VARCHAR(50),
      dateInaug                 NUMBER,
forum                   NUMBER,
    shortname           VARCHAR(70)
);
```

Part 2: Data Entry and Query Execution

2.1. Database population

For the database population, a temporary table was created in which the data from the csv file was entered.
We created the scriptpoblarT.sql, in which we transfer the necessary data from each column of the temporary table to their corresponding columns in our tables.
However, we later observed that this was not a method contemplated in the statement of the practice for populating the database, and that there was already a table that had exactly the same purpose as our temporary table (datosdb.ligahost). Once we saw the problem, in order not to have to redo all the population statements we had already written, we decided to populate the temporary table with the data from the table 'datosdb.ligahost'And that's how we solved the problem.

Let's break down the different decisions made to populate each of our tables:

- Stadium, Team andDivision, Other Namesand Party:
  We directly retrieve the columns from the table using SELECT DISTINCT (thus avoiding duplicates), ensuring that the PRIMARY KEY is not NULL (obtaining the attributes in this way is repeated in the rest of the tables, leaving aside the peculiarities of each of the others).

- Season:
  In this case, the attributenumeroTempIt has the peculiarity that we will calculate it by means of a subtraction: "(t.annoInicio - 1971) AS numeroTemp"(same in the rest of the tables)."
  In addition, we will use a "FROMtablaTemporal t INNER JOIN Divisiond ONt.denominacionOficial = d.officialName"which ensures the selection of those records with the same official name in both tables.

- Day:
  To ensure that only days from a specific, already registered season are inserted, we use "FROMtablaTemporal t JOIN Temporada temp ON (t.annoInicio - 1971) = temp.numeroTemp AND t.denominacionOficial = temp.denominacionOficial;".

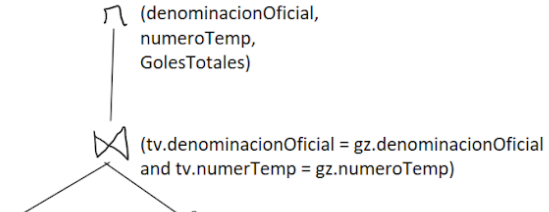
- Classification:
  We seek to give value to the score attribute.
  Basically, we created 4 views,Individual matches won and Tied matches IndividualThey list each match in which a team has won or drawn respectively, and then through the viewsmatches wonAnd in Drawn matches we count how many times a team has won or drawn respectively.
  Next, a JOIN is performed between the viewsMatches Won and Tied matchesto match information by season, official name and team, and the score is calculated using "(pg.gamesWon*3+pe.matchesTied) as the score".

The problems we have encountered during the population survey have been:
- Initial problems adapting to SQL syntax.
- The difficulty in dealing with the merging of common data in tables such as Season or Matchday.
- Many difficulties in focusing the score calculation onClassification.

2.2. Consultas SQL

2.2.1. First consultation

Syntax tree:



SQL queries:

```
CREATE OR REPLACE VIEW vicIdaYVueltaZgz(denominacionOficial, numeroTemp) AS
SELECT Official designation, numeroTemp
FROM (
    SELECT Official designation, numeroTemp, Visiting Team AS rival
FROM Party
    WHERE local team = 'Zaragoza' AND golesLocal > Visitor goals
    INTERSECT
    SELECT Official designation, numeroTemp, local team AS rival
FROM Party
    WHERE Visiting team = 'Zaragoza' AND Visitor goals > golesLocal
)
GROUP BY Official designation, numeroTemp
HAVING COUNT(rival) >= 4;

CREATE OR REPLACE VIEW golesZgz(Official designation, numeroTemp, golesTotales) AS
SELECT Official designation, numeroTemp, SUM(goals)AS goalsTotals
FROM (
    SELECT Official designation, numeroTemp, golesLocalAS goals
FROM Party
    WHERE local team= 'Zaragoza'
    UNION ALL
    SELECT Official designation, numeroTemp, golesVisitante ASgoals
FROM Party
    WHERE Visiting team= 'Zaragoza'
)
GROUP BY Official designation, numeroTemp;

SELECT tv.denominacionOficial “Division”,
       tv.numeroTemp "Season",
       gz.totalgoals“Goals”
FROM vicIdaYVueltaZgz tv
JOIN golesZgz gz ON tv.denominacionOficial = gz.officialdenomination AND tv.numeroTemp =
gz.numeroTemp;
```

The query works by selecting the requested attributes from a table formed by a JOIN between the two created views:

1. The first one, vicIdaYVueltaZgz, takes from the set of rivals per season that Zaragoza has beaten in the home and away matches, the seasons in which it did so with 4 or more teams.
2. The second,golesZgzIt takes all the matches that Zaragoza has won in each season and adds up all their goals ordered by season.

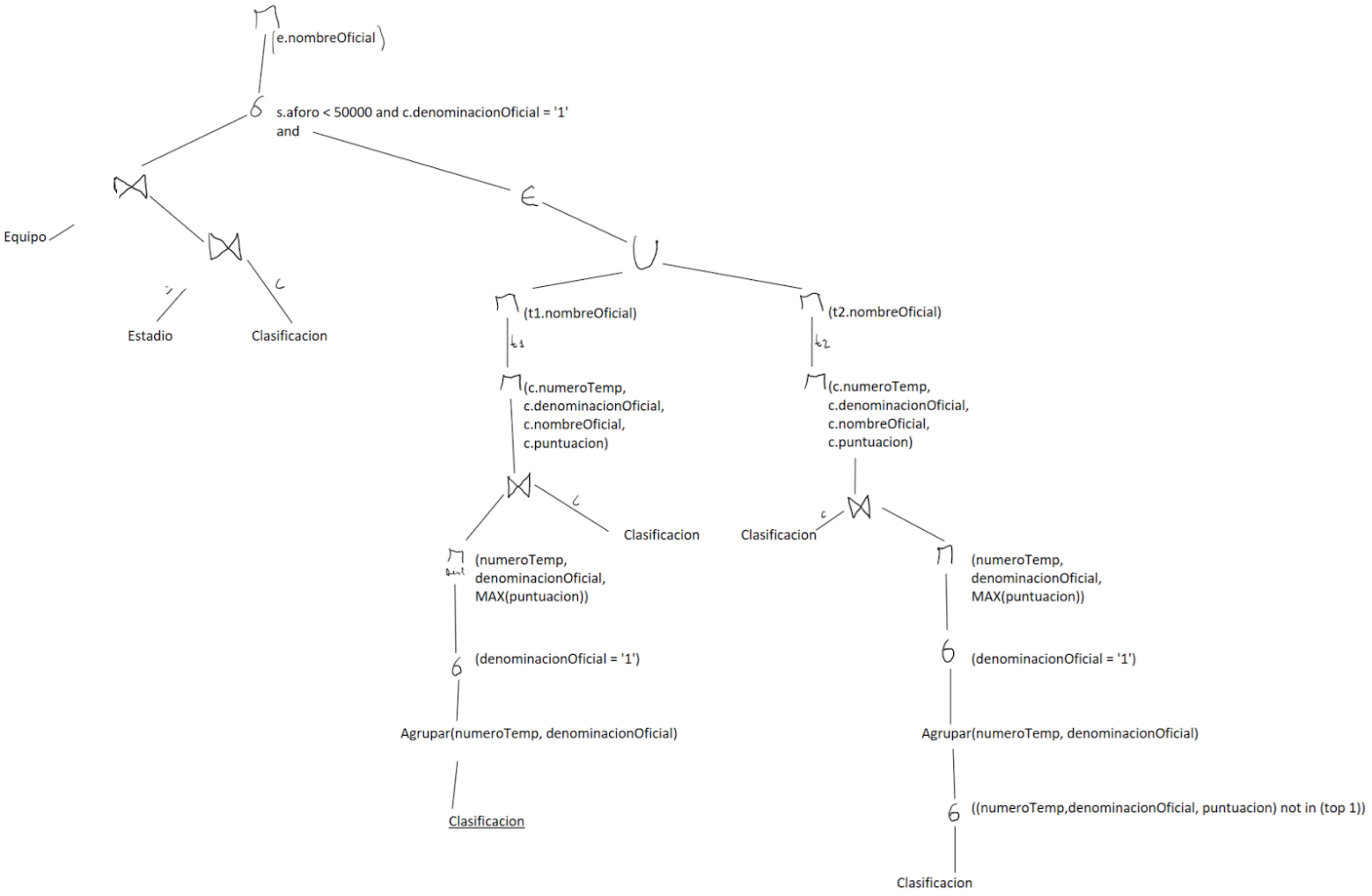
Thus, the result is the division, season, and goals scored (second view) for the seasons in which Zaragoza beat 4 teams in both legs. The most discussed query, with step-by-step explanation, can be found in “consultas.sql” in the zip file of this submission.

Answer obtained:

```
Division Season Goals
---------- ---------- ----------
2                  37         79
1                  11         59
1                  27         57
2                  31         54
1                  14         51
```

2.2.2. Second consultation

Syntax tree:



SQL queries:

```
CREATE OR REPLACE VIEW top1 AS
SELECT  c.numeroTemp, c.officialName, c.Official Number, c.score
FROM Classification c
JOIN (
    SELECT DISTINCT numeroTemp, Official designation, MAX(score) ASpuntosMax
FROM classification
    GROUP BY numeroTemp, Official designation
    HAVING Official designation = '1'
) pm
ON c.numeroTemp = pm.numeroTemp
AND c.officialName = pm.denominacionOficial
AND c.score = pm.puntosMax;

CREATE OR REPLACE VIEW top2 AS
SELECT c.numeroTemp, c.officialName, c.Official Number, c.score
FROM Classification c

JOIN (
```

```
    SELECT numeroTemp, Official designation, MAX(score) ASpuntosMax
    FROM Classification
    WHERE (numeroTemp, Official designation, punctuation) NOT IN (
        SELECT numeroTemp, Official designation, punctuation
        FROM top1
    )
    GROUP BY numeroTemp, Official designation
    HAVING Official designation = '1'
) pm
ON c.numeroTemp = pm.numeroTemp
AND c.officialName = pm.denominacionOficial
AND c.score = pm.puntosMax;

SELECT DISTINCT e.officialName“Team Name”
FROM Equipo and
JOIN Estadio s ON e.nameStadium = s.nameStadium
JOIN Classification c ON e.officialName = c.Official Number
WHERE s.aforo < 50000
  AND e.officialName IN (SELECT t1.officialName FROM top1 t1 UNION SELECT t2.officialName FROM
top2t2);
```

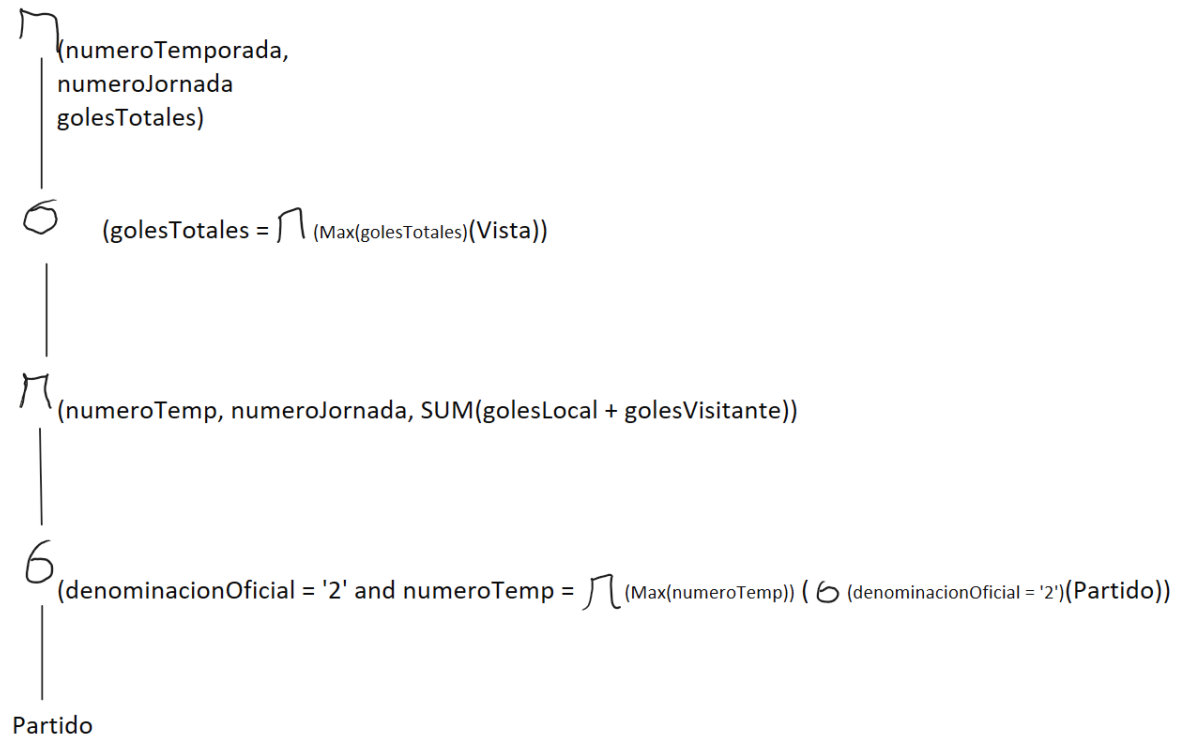
The query displays the names of teams that share a set of attributes collected in a JOIN between the Team table and the Stadium table. This allows us to relate each team to its corresponding stadium capacity and standings. Subsequently, we set the condition (in the WHERE clause) that a team's stadium capacity must be less than 50,000 people and that the team name must be included within the join of the two views.
These two views give a list of all the teams that have finished in first or second place in a season of the first division, (t1 gives first place and t2 gives second place).
Therefore, if a team's name is within these views and its stadium capacity is less than 50,000 people, its name will be projected, in accordance with the statement.
Both views, as well as the query as a whole, are commented in much more detail in the code within the delivered .zip file.

Answer obtained:

```
Team name
------------------------------
Saragossa
Real Sociedad
Dptivo. Coru??a
Valencia
Sevilla
```

2.2.3. Third consultation

Syntax tree:



SQL queries:

```
CREATE OR REPLACE VIEW vista(numeroTemp, issue of the day, golesTotales) AS
SELECT numeroTemp, issue of the day, SUM(golesLocal(Away Goals) AS Total Goals
FROM Party
WHERE Official designation = '2'
AND numeroTemp = (
    SELECT MAX(numeroTemp)
FROM Party
    WHERE Official designation = '2'
);

SELECT v.numeroTemp "Season", v.numberDay"Journey",v.golesTotals"Goals"
FROM vista v
WHERE v.golesTotals=(SELECT MAX(golesTotales) FROM vista);
```

This involves selecting the requested attributes of the element with the highest number of goals in the view. This view creates a table with the season number, matchday number, and total goals for the home and away teams in the first division, using the newest season from the Match table.
The result of the query is then the season number and matchday of the matchday with the most goals of the last season. In “consultas.sql” of the zip file of the delivery you will find the most commented query step by step.

Answer obtained:

```
Season Matchday Goals
---------- ---------- ----------
        44          1         37
```

Part 3: Physical Design

3.1. SQL query performance

When applying the commands

EXPLAIN PLAN FOR <consulta SQL>

SELECT PLAN_TABLE_OUTPUT FROM TABLE(DBMS_XPLAN.DISPLAY())

Regarding the three different queries, we've seen that none of them are particularly costly. Therefore, with the exception of a modification to query 3 mentioned later, we've concluded that the main improvements to database query performance could be made to the database structure itself.

Let's look at the results obtained in terms of time and percentage of CPU usage:

	First consultation	Second consultation	Third consultation
CPU Percentage*	3%	18%	2%
Cost	249	47	138
Execution time*	00:00:01	00:00:01	00:00:01

** Values are approximate.*

We can suggest the following modifications:

1. Using an artificial key in Partido: because of how the database is built, the primary key of Partido is made up of the set of 5 primary keys from different tables (Division(Season, Matchday, Team x2) since all of these are necessary to differentiate one match from another. This is why accessing an element in the Match table takes so long and is the reason why query 2 takes longer than the others; it has to traverse the Match table multiple times.
   Using an artificial key would avoid having to go through so many primary keys to access an element, decreasing the access time to an element in the Match table.

2. Use an attribute set in the tableClassificationTo facilitate the second query by not having to calculate positions 1 and 2 in each execution, an attribute "position" could be added to the Query table indicating the team position associated with that element in the table.Classification, during the season anddivisionindicated.

3. Finally, we optimized the second query by removing one of the joins.whatwe did in the same, which was avoidable (the JOINClassificationc in the consultation), in this way we have managed to reduce the cost from 47 to 42.

3.2. Restrictions solved using triggers

There are a large number of restrictions that Oracle cannot verify when accessing, inserting, or deleting tables in our database. To address this, we will implement triggers, which we will illustrate with three examples after this introduction.

Some of the restrictions we may encounter are:

- ➢ A team cannot play a match against itself.
- ➢ A match cannot be inserted whose season has not yet started.
- ➢ When a division is deleted, everything related to it is deleted.
- ➢ When a season is deleted, everything related to it is deleted.
- ➢ When a day is deleted, everything related to it is deleted.
- ➢ The stadium capacity has to be positive.
- ➢ We cannot insert a season prior to 1972 due to the internal structure of the database; to do so, the column corresponding to the season number would have to be modified with the new requirements.
- ➢ Both the home team's and the away team's goals must be greater than or equal to 0.
- ➢ When a match is inserted with a non-existent team, the new team is added to the database.

3.2.1. Primer trigger

The same team plays a match:

```
CREATE OR REPLACE TRIGGER sameTeamInMatch
BEFORE INSERT ON Partido
FOR EACH ROW
WHEN NEW.localteam = NEW.visitorTeam
BEGIN
RAISE_APPLICATION_ERROR(-20001, 'A team cannot face againstitself.');
END;
/
```

Our first trigger addresses one of the most obvious errors a user can make, and therefore one that's worth covering. The error occurs when trying to enter a match where both the home and away teams are the same; obviously, this can't happen, and we're going to prevent it.

The trigger's operation is very basic: it activates before an insertion into the Match table (line 2), and its code is executed when the name of the home team in the match you're trying to insert matches the name of the away team; that is, the match is a game between two teams. The trigger code simply prints the error message "'A team cannot play against itself'" to the screen, and also prevents the insertion of that match.

3.2.2. Second trigger

Inserting non-existent teams when inserting a match:

```
CREATE OR REPLACE TRIGGER InsertNuevoEquipoEnPartido
```

```
BEFORE INSERT or UPDATE ON Partido
FOR EACH ROW
DECLARE
    local appearances NUMBER;
    Visitor Appearances NUMBER;
BEGIN
-- Check if the local equipment exists
    SELECT COUNT(*) INTO local appearances FROM Equipo WHERE Official name =
:NEW.localteam;

-- If it doesn't exist, insert it
    IF local appearances = 0 THEN
INSERT INTO Equipo (Official name) VALUES (:NEW.localteam);
    END IF;

-- Check if the visiting team exists
    SELECT COUNT(*) INTO Visitor Appearances FROM Equipo WHERE Official name =
:NEW.visitorTeam;

-- If it doesn't exist, insert it
    IF Visitor Appearances = 0 THEN
INSERT INTO Equipo (Official name) VALUES (:NEW.visitorTeam);
    END IF;
END;
/
```

If you attempt to insert a match where one or both teams don't exist, they are inserted by name. This is because for a match to form its primary key, it needs to borrow the names of the participating teams, and for this to happen, both teams must be part of the Teams table.

Regarding the trigger's operation, it always executes before a match is inserted or updated. First, it checks if the home team is registered in the database by storing its number of appearances in a variable. If this number is 0, then the team doesn't exist, so it inserts it. Then it does the same for the away team.

3.2.3. Third trigger

Erased from an entire season and erased from an entire division:

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


CREATE OR REPLACE TRIGGER cleanDivision
BEFORE DELETE ON Division
FOR EACH ROW
BEGIN
    DELETE FROM Partido WHERE Official designation=:OLD.officialDenomination;
    DELETE FROM Jornada WHERE Official designation=:OLD.officialDenomination;
DELETE FROM Classification WHEREOfficial designation=:OLD.officialDenomination;
    DELETE FROM Temporada WHERE Official designation=:OLD.officialDenomination;

END;
/
```

At this point we created 2 triggers, both very similar.

1. The first one erases a season and everything that season implies, that is, it erases the matches, days and standings that make it up.

2. The second one deletes a division and everything that division implies, that is, it deletes the matches, days, classifications and seasons that make it up.

Both triggers are essential for the normal and logical use of the database; otherwisewe erasedElements related to a season or division would then become elements that depend on something that doesn't exist. The same applies to Matchday; its trigger would be very similar to the two implemented previously.

Group organization

We organized ourselves so that all members of the group focused on the same point, each creating a part of the section. If one of us had problems, we met to solve them together.

The group work has been quite effective and fair.

Tasks performed	Group members involved	Hours dedicated	Problems observed
Global E/R scheme, normalization and list of restrictions.	All members of the group.	2.5 hours	Some difficulty in deciding on relationships with certain entities.
Relational schema.	All members of the group.	2 hours	No problem.
Database creation.	All members of the group.	2 hours	No problem.
Database population.	All members of the group.	25 hours	Problems related to the Classification population, specifically the scoring attribute.
Creation and testing of SQL queries.	All members of the group.	5 hours	No problem.
Relational trees of the previous queries.	All members of the group.	2.5 hours	Initial problems in understanding how to design them.
Query performance.	All members of the group.	1.5 hours	No problem.
Triggers.	All members of the group.	4 hours	No problem.
Creation of memory.	All members of the group.	6 hours	No problem.