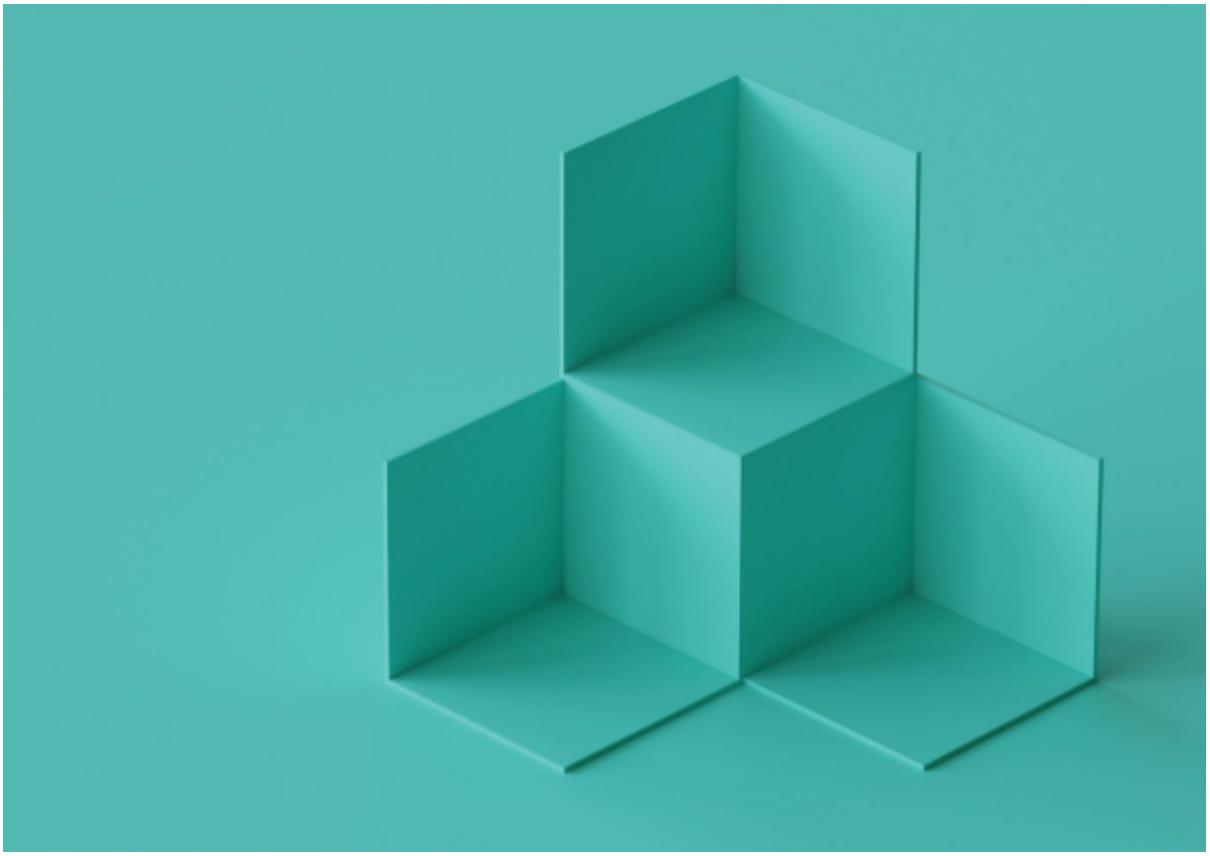
Bases de datos - Práctica 1

Curso 2022

03/04/2022



Participantes

Héctor Toral Pallás - 798095@unizar.es

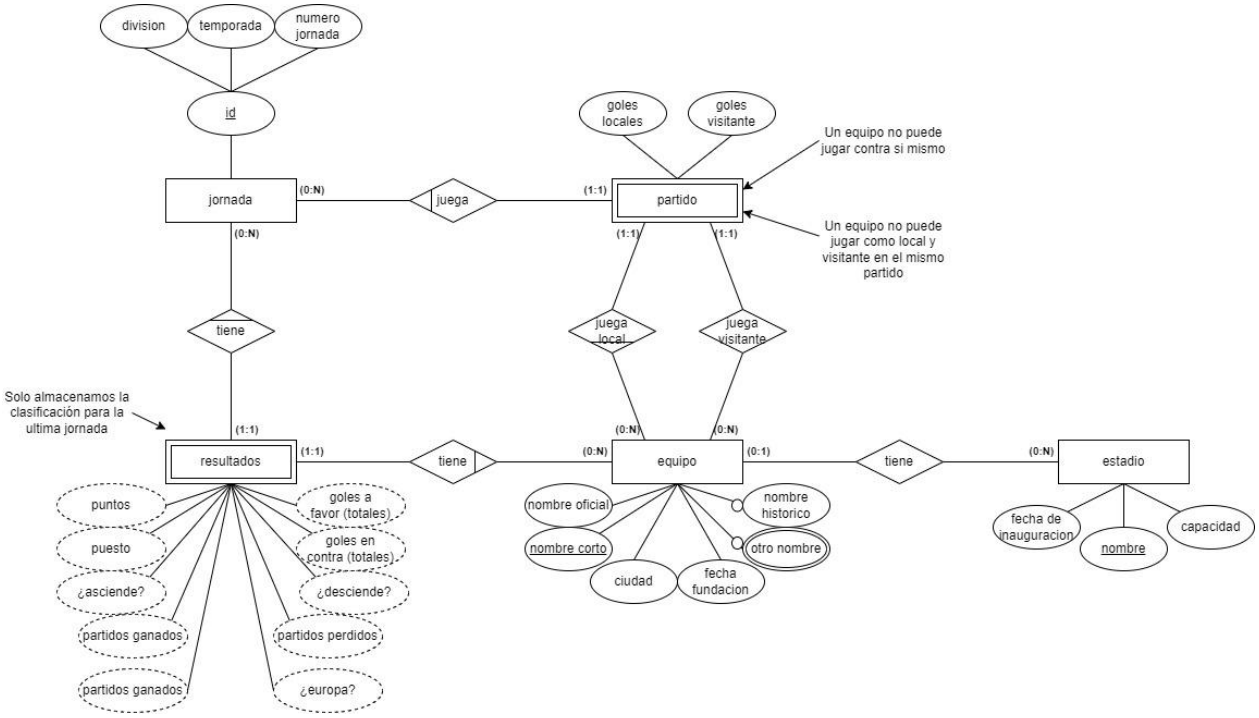
Francisco Javier Pizarro Martinez - 821259@unizar.es

Pablo López Mosqueda - 779739@unizar.es

Parte 1 - Creación de la base de datos

1.1 - Modelo Entidad/Relación

En está práctica se ha de realizar una base de datos para gestionar y almacenar información acerca de la liga de fútbol española desde 1972, para ello, se ha planteado el siguiente modelo E/R:



Restricciones del modelo:

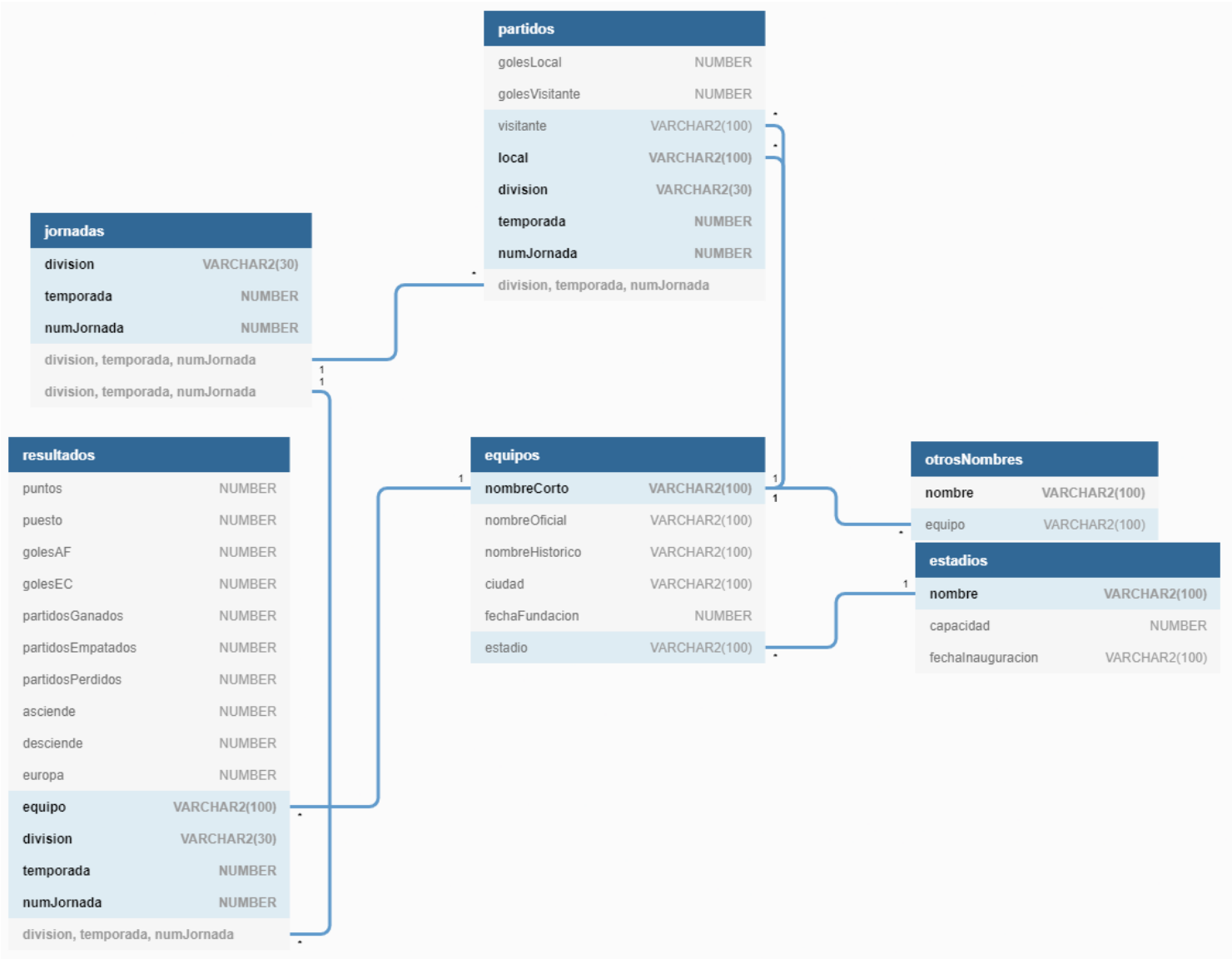
- No se puede introducir un resultado para una temporada dada, si la fecha de fundación del equipo de dicho resultado es posterior a la temporada.
- Un equipo no puede jugar más de un partido en la misma jornada.

Alternativas a nuestro modelo:

Para la tabla de jornadas, se podría representar como una cadena de entidades débiles.

Una alternativa a la doble relación que hay entre equipo y partido hubiera sido hacer una relación 2:N.

1.2 - Modelo Relacional



El modelo se encuentra en la 1ªFN debido a que los atributos multivaluados han sido traducidos al modelo relacional de manera que estos se han repartido a otras tablas (tabla: otros nombres).

El modelo se encuentra en la 2ªFN debido a que todos los atributos que forman parte de las claves compuestas no dependen unos de otros.

El modelo se encuentra en la 3ª FN y 3ªFNBC ya que todos los atributos dependen siempre de la clave.

1.3 - Traducción a SQL

```
CREATE TABLE estadios (
    nombre              VARCHAR2(100),
    capacidad           NUMBER      NOT NULL,
    fechaInauguracion   VARCHAR2(100)   NOT NULL,
    CONSTRAINT pk_Es_nombre         PRIMARY KEY (nombre),
    CONSTRAINT ck_capacidad         CHECK (capacidad >= 0)
);

CREATE TABLE equipos (
    nombreCorto     VARCHAR2(100),
    nombreOficial   VARCHAR2(100)   NOT NULL,
    nombreHistorico VARCHAR2(100),
    ciudad          VARCHAR2(100)   NOT NULL,
    fechaFundacion  NUMBER          NOT NULL,
    estadio         VARCHAR2(100),
    CONSTRAINT pk_Eq_nombreCorto    PRIMARY KEY (nombreCorto),
    CONSTRAINT fk_estadio           FOREIGN KEY (estadio) REFERENCES estadios
(nombre),
    CONSTRAINT ck_fechaFundacion CHECK (fechaFundacion >= 1582)
);

CREATE TABLE otrosNombres (
    nombre  VARCHAR2(100),
    equipo  VARCHAR2(100),
    CONSTRAINT pk_R_nombre  PRIMARY KEY (nombre),
    CONSTRAINT fk_equipo    FOREIGN KEY (equipo) REFERENCES equipos
(nombreCorto)
);

CREATE TABLE jornadas (
    division    VARCHAR2(30),
    temporada   NUMBER,
    numJornada  NUMBER,
    CONSTRAINT pk_Eq_super      PRIMARY KEY (division, temporada, numJornada),
    CONSTRAINT ck_temporada     CHECK (temporada >= 1972),
    CONSTRAINT ck_numJornada    CHECK (numJornada >= 0)
);
```

```
CREATE TABLE partidos (
    golesLocal      NUMBER,
    golesVisitante  NUMBER,
    local           VARCHAR2(100),
    visitante       VARCHAR2(100),
    division        VARCHAR2(30),
    temporada       NUMBER,
    numJornada      NUMBER,
    CONSTRAINT pk_P_super   PRIMARY KEY (local, division, temporada,
numJornada),
    CONSTRAINT fk_local     FOREIGN KEY (local)     REFERENCES equipos
(nombreCorto),
    CONSTRAINT fk_visitante FOREIGN KEY (visitante) REFERENCES equipos
(nombreCorto),
    CONSTRAINT fk_jornada   FOREIGN KEY (division, temporada, numJornada)
REFERENCES jornadas (division, temporada, numJornada),
    CONSTRAINT ck_golesLocal        CHECK (golesLocal >= 0),
    CONSTRAINT ck_golesVisitante    CHECK (golesVisitante >= 0),
    CONSTRAINT ck_distEquipo        CHECK (local <> visitante)
);

CREATE TABLE resultados (
    puntos              NUMBER  DEFAULT 0,
    puesto              NUMBER  NOT NULL,
    golesAF             NUMBER  DEFAULT 0,
    golesEC             NUMBER  DEFAULT 0,
    partidosGanados     NUMBER  DEFAULT 0,
    partidosEmpatados   NUMBER  DEFAULT 0,
    partidosPerdidos    NUMBER  DEFAULT 0,
    asciende            NUMBER  DEFAULT 0,
    desciende           NUMBER  DEFAULT 0,
    europa              NUMBER  DEFAULT 0,
    equipo      VARCHAR2(100),
    division    VARCHAR2(30),
    temporada   NUMBER,
    numJornada  NUMBER,
    CONSTRAINT pk_R_super   PRIMARY KEY (equipo, division, temporada,
numJornada),
    CONSTRAINT fk_R_equipo  FOREIGN KEY (equipo) REFERENCES equipos
(nombreCorto),
    CONSTRAINT fk_R_jornada FOREIGN KEY (division, temporada, numJornada)
REFERENCES jornadas (division, temporada, numJornada),
    CONSTRAINT ck_asciende  CHECK (asciende = 0 OR asciende = 1),
    CONSTRAINT ck_desciende CHECK (desciende = 0 OR desciende = 1),
```

```
    CONSTRAINT ck_europa    CHECK (europa = 0 OR europa = 1),
    CONSTRAINT ck_puntos    CHECK (puntos >= 0),
    CONSTRAINT ck_puesto    CHECK (puesto >= 0),
    CONSTRAINT ck_golesAF   CHECK (golesAF >= 0),
    CONSTRAINT ck_golesEC   CHECK (golesEC >= 0),
    CONSTRAINT ck_partidosGanados   CHECK (partidosGanados >= 0),
    CONSTRAINT ck_partidosEmpatados CHECK (partidosEmpatados >= 0),
    CONSTRAINT ck_partidosPerdidos  CHECK (partidosPerdidos >= 0)
);
```

Parte 2 - Introducción de datos y ejecución de consultas

2.1 - Población de la Base de Datos

Para empezar con la parte de población nos pusimos de acuerdo en usar archivos csv para almacenar y manejar los datos, así como en emplear Python como lenguaje de scripting para la recolección de datos además de su procesado y conversión a inserts en sql.

Antes de empezar con la extracción de datos se procesó el fichero LigaHost.txt con un script para convertir dichos datos a formato csv.

Para la recolección de datos empleamos un par de módulos auxiliares en Python que hacían más sencillo el web scraping, el script encargado de recolectar datos primero anotaba los equipos cuya información necesitábamos, extrayendolos de LigaHost.txt, después utilizando el siguiente enlace:

http://es.wikipedia.org/wiki/Anexo:Clubes_de_fútbol_de_España_por_fundación

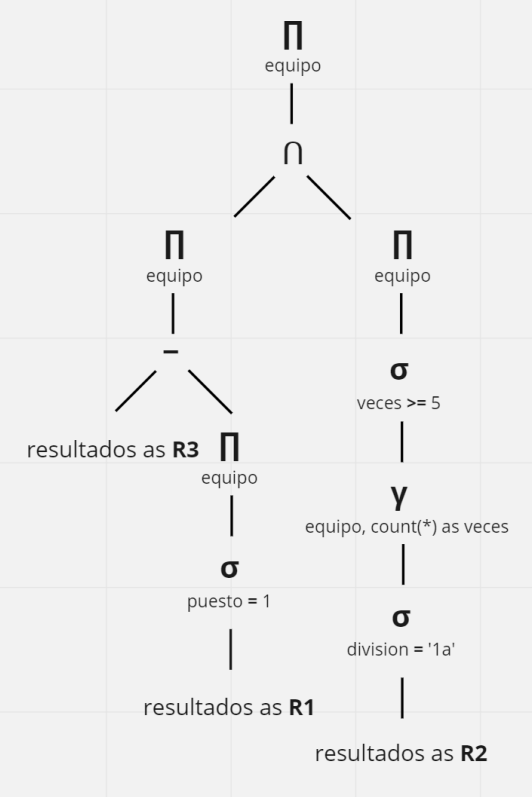
Extrajimos la información de las tablas relacionada con los equipos y la almacenamos en un csv. Posteriormente, otro script usaba los datos recolectados para, utilizando los distintos enlaces de los estadios de los equipos, extraer datos de los mismos y almacenarlos también en un csv. Para generar el sql que poblaba cada una de las tablas se empleó un script distinto para cada uno que simplemente hacía la conversión de nuestros datos de csv a inserts de sql. Adicionalmente se empleó una consulta sql para extraer los datos necesarios para poblar la tabla de resultados.

Recolectando los datos nos encontramos con una serie de problemas, equipos que no aparecían en la tabla principal sino que aparecían en la tabla de equipos extintos, así como datos atípicos con carácteres especiales o filas en la tabla que tenían un número distinto de de columnas(tanto en LigaHost.txt como en Wikipedia), otro problema que nos encontramos es estadios cuya página web en la wikipedia existía pero estaba vacía. Algunos de los problemas se solucionaron programando cómo se gestionaban los distintos errores y otros mediante la recolección manual de los datos restantes.

2.2 - Consultas de la Base de Datos

1. Equipo(s) que han estado en primera división un mínimo de cinco temporadas y que no han ganado ninguna liga.

```sql
SELECT DISTINCT equipo
FROM resultados
WHERE equipo NOT IN (
    -- Equipos que han ganado al menos una liga
    SELECT DISTINCT equipo
    FROM resultados
    WHERE puesto = 1
) AND equipo IN (
    -- Equipos que han estado en primera al menos 5 temporadas
    SELECT equipo
    FROM (
        SELECT COUNT(*) AS veces, equipo
        FROM resultados
        WHERE division = '1ª'
        GROUP BY equipo
    ) A
    WHERE A.veces >= 5
)
ORDER BY equipo;
```

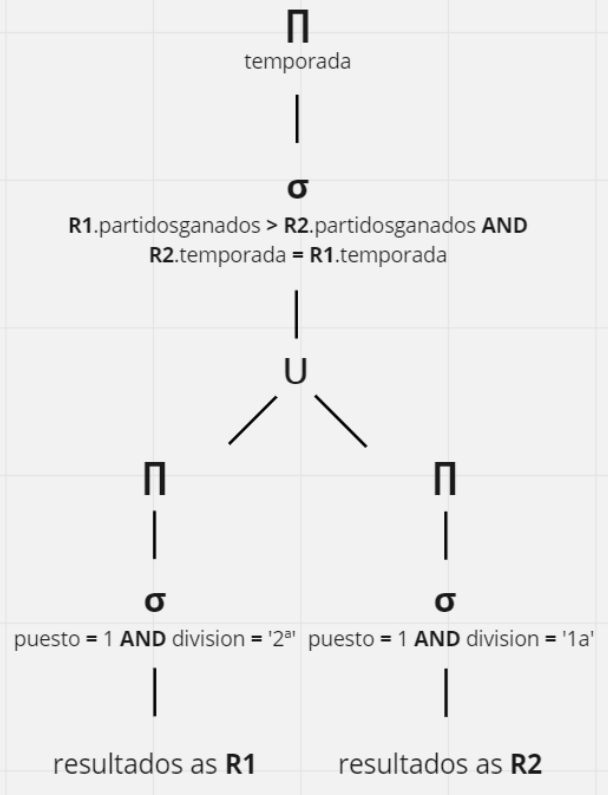


Equipo			
Alavés	Albacete	Ath. Bilbao	Burgos
Castellón	Dptivo. Coruña	Getafe	Granada
Hércules	Las Palmas	Levante	Logroñés
Málaga (C.D.)	Mallorca	Osasuna	Oviedo
Rayo Vallecano	Real Sociedad	Rec. Huelva	Salamanca
Sporting Gijón	Valencia	Villarreal	Zaragoza

2. Temporada(s) en las que el ganador de segunda división ha ganado más partidos que el ganador de primera división.

```sql
SELECT T1.temporada
FROM resultados T1
WHERE T1.division = '2ª' AND T1.puesto = 1
    AND EXISTS (
    -- Devuelve una "tupla"/"flag" cuando se cumple que el ganador de segunda
    -- ha ganado más partidos que el de (primera/temporada)
    SELECT 1
    FROM resultados T2
    WHERE T2.division = '1ª' AND T2.puesto = 1
    AND T1.partidosganados > T2.partidosganados
    AND T2.temporada = T1.temporada
)
ORDER BY T1.temporada;
```

Árbol descriptivo del álgebra relacional para esta consulta:

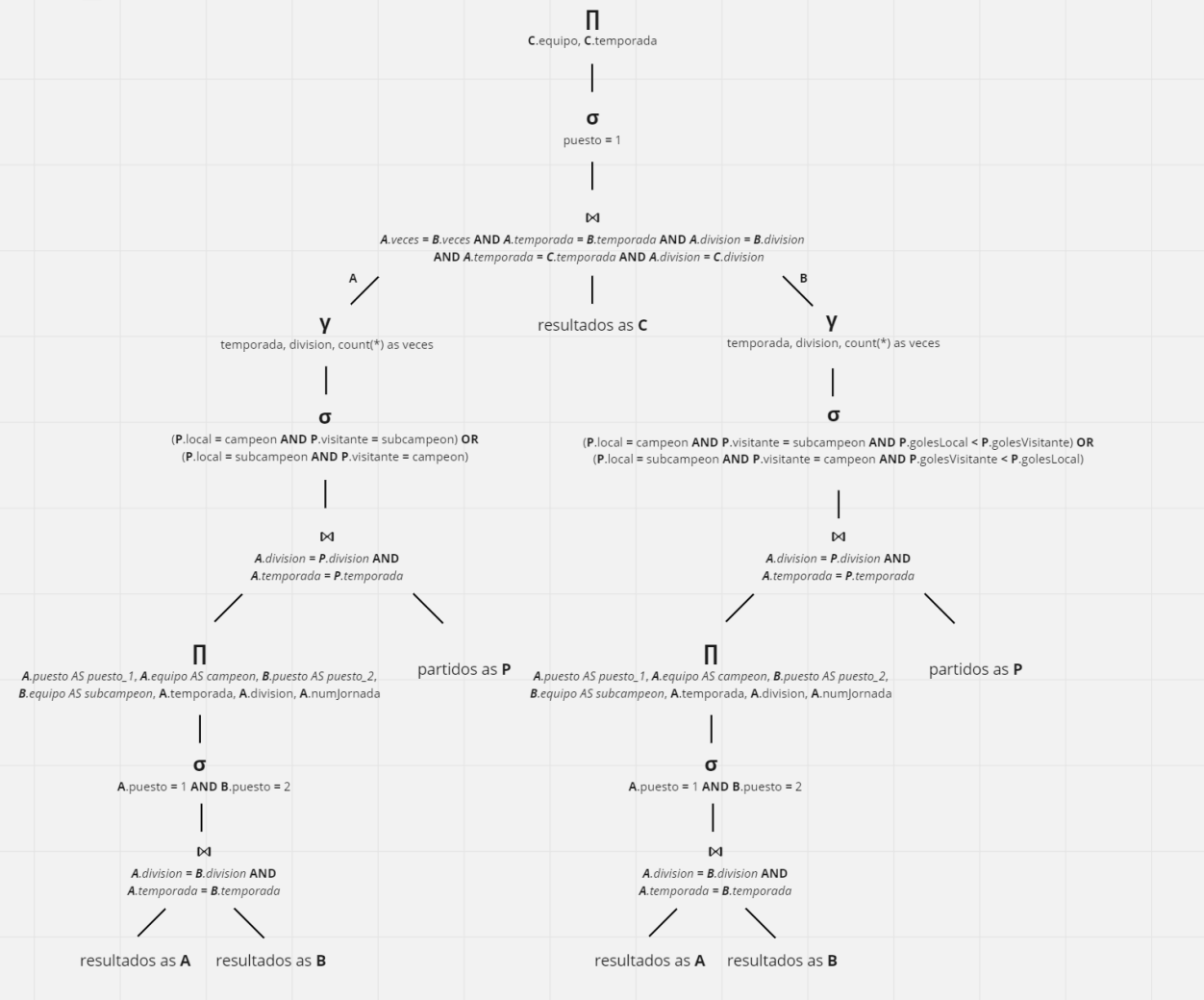


Temporada
1973
1979
1985
1994
1999
2001
2003
2011

3. Equipo(s) y temporada(s) donde dicho equipo ha ganado dicha temporada, habiendo perdido todos los partidos contra el equipo que quedó en segunda posición.

```
SELECT C.equipo, C.temporada
FROM (
    SELECT COUNT(*) AS veces, R.temporada, R.division
    FROM partidos P, (
        -- Devuelve pares de ganador, subcampeon
        SELECT DISTINCT A.puesto AS puesto_1, A.equipo AS campeon,
               B.puesto AS puesto_2, B.equipo AS subcampeon, A.temporada,
               A.division, A.numJornada
        FROM resultados A, resultados B
        WHERE A.puesto = 1 AND B.puesto = 2 AND A.division = B.division AND
              A.temporada = B.temporada
    ) R
    WHERE ((P.local = R.campeon AND P.visitante = R.subcampeon) OR
           (P.local = R.subcampeon AND P.visitante = R.campeon)) AND
            P.division = R.division AND P.temporada = R.temporada
    GROUP BY R.temporada, R.division
) A, (
    SELECT COUNT(*) AS veces, R.temporada, R.division
    FROM partidos P, (
        -- Devuelve pares de ganador, subcampeon
        SELECT DISTINCT A.puesto AS puesto_1, A.equipo AS campeon,
               B.puesto AS puesto_2, B.equipo AS subcampeon, A.temporada,
               A.division, A.numJornada
        FROM resultados A, resultados B
        WHERE A.puesto = 1 AND B.puesto = 2 AND A.division = B.division AND
              A.temporada = B.temporada
    ) R
    WHERE ((P.local = R.campeon AND P.visitante = R.subcampeon AND
            P.goleslocal < P.golesvisitante) OR (P.local = R.subcampeon AND
            P.visitante = R.campeon AND P.goleslocal > P.golesvisitante)) AND
            P.division = R.division AND P.temporada = R.temporada
    GROUP BY R.temporada, R.division
) B, resultados C
WHERE A.veces = B.veces AND A.temporada = B.temporada AND
      A.division = B.division AND A.temporada = C.temporada AND
      A.division = C.division AND puesto = 1
ORDER BY A.temporada;
```

Árbol descriptivo del álgebra relacional para la consulta:



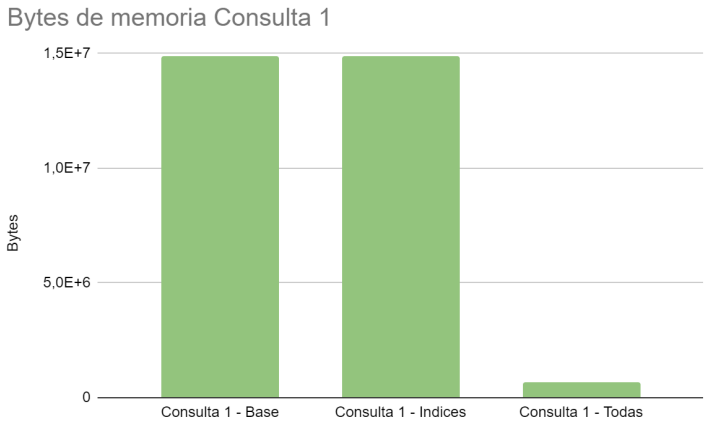
Equipo	Temporada
At. Madrid	1978
At. Madrid	1982
Celta	1987
Rac. Santander	1993
Málaga	2008
Celta	2012
At. Madrid	2015

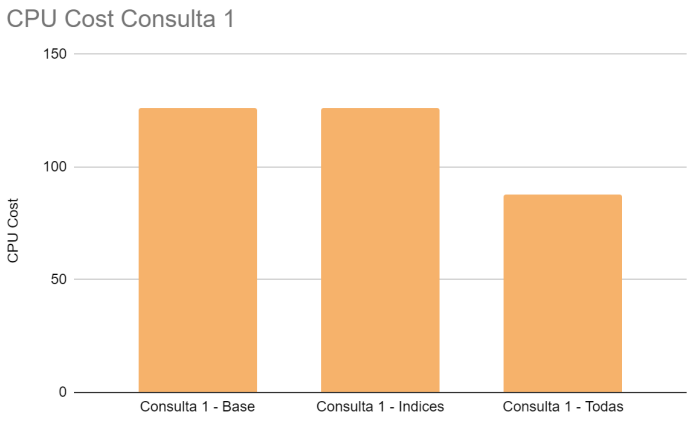
Parte 3 - Evaluación de rendimiento y triggers

3.1 - Evaluación de rendimiento

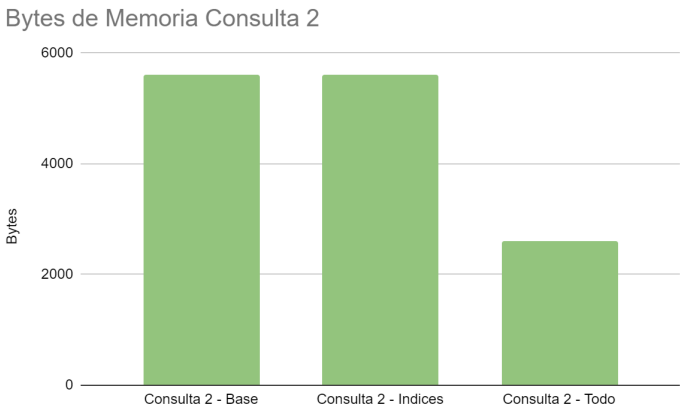
En la primera consulta el coste recae sobre las condiciones del where y en los group by.

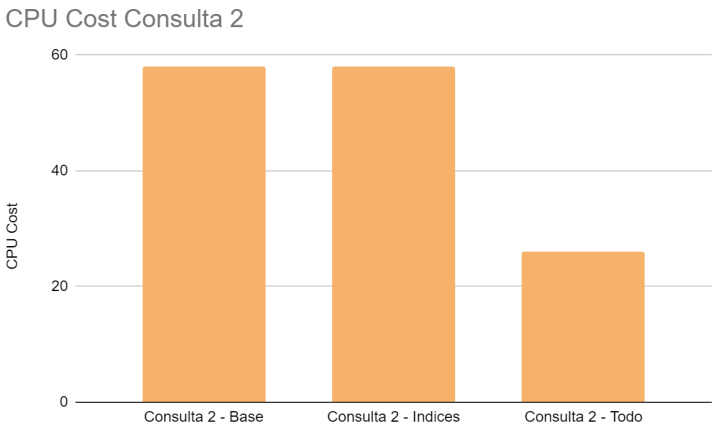
Las pruebas realizadas han sido indexar el puesto y crear una vista materializada del número de apariciones de un equipo de primera división en los resultados.





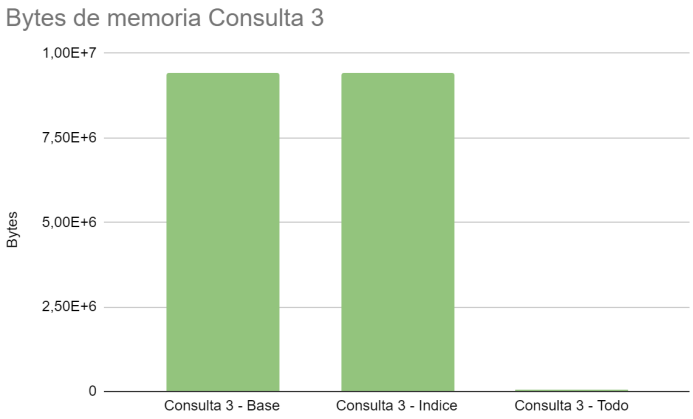
En la segunda consulta el coste es mínimo, no podemos esperar una gran mejora como en la anterior, de todas formas, repitiendo las pruebas anteriores, nos damos cuenta de que también existe una diferencia notable.

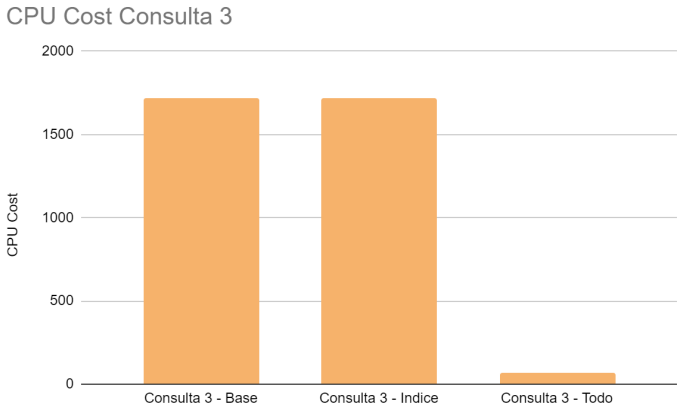




En la tercera consulta el coste es muy alto, especialmente el acceso completo a la tabla de partidos. Además, gran parte de la consulta ocurre 2 veces por lo que se puede optimizar bastante.

Hemos probado a usar un índice, lo cual no ha mejorado el rendimiento, también hemos probado a crear una vista materializada de resultados y otra de los partidos en los que el equipo campeón y subcampeón se enfrentaban.





Como conclusiones de la optimización podemos decir que tener los datos necesarios para llegar y coger, favorecen en gran medida al rendimiento de la consulta.

3.2 - Triggers

Inicialmente desarrollamos 2 triggers sencillos y efectivos que realmente se usarían en producción en la base de datos, adicionalmente creamos un tercer trigger para aprender a desarrollar triggers más complejos, no obstante este último no debería ser empleado en producción debido a su gran coste.

Nos aseguramos al insertar un partido que ninguno de los equipos participantes haya jugado un partido en esa misma jornada de dicha temporada y división, a pesar de que un equipo no puede figurar 2 veces como local o 2 veces como visitante por qué son las-que identifican la fila del partido y por lo tanto no se pueden repetir, si que es posible que se de la configuración en la que el equipo A juega un partido como equipo local y un partido como equipo visitante el mismo día.

- Trigger 1:

```
CREATE OR REPLACE TRIGGER CHECK_INTEGRIDAD
BEFORE INSERT ON partidos
FOR EACH ROW
DECLARE
   flag NUMBER;
BEGIN
   SELECT COUNT(*) INTO flag
   FROM partidos
   WHERE temporada = :NEW.temporada AND division = :NEW.division AND
         numJornada = :NEW.numJornada AND
         (local = :NEW.visitante OR visitante = :NEW.visitante);

   IF flag >= 1 THEN
       RAISE_APPLICATION_ERROR (-20000,
       'Un equipo no puede jugar como local y visitante en la misma jornada.');
   END IF;
END;
/
```

En este trigger nos aseguramos de que no se introduce en resultados para una temporada dada, un resultado cuyo equipo fue fundado después de dicha temporada.

- Trigger 2:

```
CREATE OR REPLACE TRIGGER DATES_OK
BEFORE INSERT ON resultados
FOR EACH ROW
DECLARE
    fechaFUN NUMBER;
BEGIN
    SELECT fechaFundacion
    INTO fechaFUN FROM equipos
    WHERE :NEW.equipo = nombreCorto;

    IF :NEW.temporada < fechaFUN
    THEN
        RAISE_APPLICATION_ERROR (-20003,
        'El equipo no existe aún en esta temporada');
    END IF;
END;
/
```

En el trigger 3, se persiguen principalmente 2 objetivos:

1. Que las tuplas insertadas en partidos se inserten en un orden específico. Para una temporada y división se insertarán primero todos los partidos de la jornada 1, luego los de la 2 y así sucesivamente.
2. Una vez bien insertadas las tuplas en partidos, se procederá a calcular los atributos calculados, esto supondrá el coste que se genere al realizar las siguientes operaciones: (2 inserts o 2 updates) y un merge sobre una consulta que actualizará toda la tabla para la temporada, división insertadas sobre la tabla partidos.
   El problema de este trigger radica en que supondrá un elevado coste al tener que realizar estos pasos cada vez que un partido sea insertado.

- Trigger 3:

```
CREATE OR REPLACE TRIGGER UPT_RESULTADOS
FOR INSERT ON partidos
COMPOUND TRIGGER
    filaLocal resultados%ROWTYPE; filaVisitante resultados%ROWTYPE;
    puntosLocal NUMBER; puntosVisitante NUMBER; ganaLocal NUMBER;
    ganaVisitante NUMBER; empate NUMBER; existeParLocA NUMBER;
    existeParVisA NUMBER; existeResLoc NUMBER; existeResVis NUMBER;

    -- Garantiza una insercion de los partidos en orden
    BEFORE EACH ROW IS
    BEGIN
        -- El partido que juega local en la jornada anterior no existe
        SELECT COUNT(*) INTO existeParLocA
        FROM partidos
        WHERE (local = :NEW.local OR visitante = :NEW.local) AND
            temporada = :NEW.temporada AND division = :NEW.division AND
            numJornada = :NEW.numJornada - 1;

        -- El partido que juega visitante en la jornada anterior no existe
        SELECT COUNT(*) INTO  existeParVisA
        FROM partidos
        WHERE (local = :NEW.visitante OR visitante = :NEW.visitante) AND
              temporada = :NEW.temporada AND division = :NEW.division AND
              numJornada = :NEW.numJornada - 1;

        IF  :NEW.numJornada <> 1 AND NOT (1 <= existeParLocA) OR
             NOT (1 <= existeParVisA)
        THEN
            RAISE_APPLICATION_ERROR (-20003,
            'Esta tupla esta siendo insertada antes de lo que deberia');
        END IF;
    END BEFORE EACH ROW;

    -- Actualiza la tabla de resultados
    AFTER EACH ROW IS
    BEGIN
        -- captura los resultados antiguos de los 2 equipos insertados
        SELECT * INTO filaLocal
```

```
        FROM resultados
        WHERE equipo = :NEW.local AND temporada = :NEW.temporada AND division = :NEW.division;
        SELECT * INTO filaVisitante
        FROM resultados
        WHERE equipo = :NEW.visitante AND temporada = :NEW.temporada AND division = :NEW.division;

        -- Establece los puntos nuevos
        -- Gana el local
        IF :NEW.golesLocal > :NEW.golesVisitante THEN
            puntosLocal := 3;  puntosVisitante := 0;  ganaLocal := 1;  ganaVisitante := 0;  empate := 0;
        -- Gana el visitante
        ELSIF :NEW.golesLocal < :NEW.golesVisitante THEN
            puntosLocal := 0;  puntosVisitante := 3;  ganaLocal := 0;  ganaVisitante := 1;  empate := 0;
        -- Empatan el partido
        ELSE
            puntosLocal := 1;  puntosVisitante := 1;  ganaLocal := 0;  ganaVisitante := 0;  empate := 1;
        END IF;


        -- si existe un resultado para el equipo local
        SELECT COUNT(*) INTO existeResLoc
        FROM resultados
        WHERE equipo = :NEW.local AND temporada = :NEW.temporada AND division = :NEW.division;
        -- si existe un resultado para el equipo local
        SELECT COUNT(*) INTO existeResVis
        FROM resultados
        WHERE equipo = :NEW.visitante AND temporada = :NEW.temporada AND division = :NEW.division;

        IF (1 >= existeResLoc AND 1 >= existeResVis) THEN
            -- Actualiza los resultados del equipo local
            UPDATE RESULTADOS
            SET
                PUNTOS  = filaLocal.puntos  + puntosLocal,
                GOLESAF = filaLocal.golesAF + :NEW.golesLocal,
                GOLESEC = filaLocal.golesEC + :NEW.golesVisitante,
                PARTIDOSGANADOS     = filaLocal.partidosGanados   + ganaLocal,
                PARTIDOSEMPATADOS   = filaLocal.partidosEmpatados + empate,
                PARTIDOSPERDIDOS    = filaLocal.partidosPerdidos  + ganaVisitante,
                NUMJORNADA          = :NEW.numJornada
            WHERE EQUIPO = :NEW.local AND DIVISION = :NEW.division AND TEMPORADA = :NEW.temporada;

            -- Actualiza los resultados del equipo visitante
            UPDATE RESULTADOS
            SET
                PUNTOS  = filaVisitante.puntos  + puntosVisitante,
                GOLESAF = filaVisitante.golesAF + :NEW.golesVisitante,
                GOLESEC = filaVisitante.golesEC + :NEW.golesLocal,
                PARTIDOSGANADOS     = filaVisitante.partidosGanados   + ganaVisitante,
                PARTIDOSEMPATADOS   = filaVisitante.partidosEmpatados + empate,
                PARTIDOSPERDIDOS    = filaVisitante.partidosPerdidos  + ganaLocal,
                NUMJORNADA          = :NEW.numJornada
            WHERE EQUIPO = :NEW.visitante AND DIVISION = :NEW.division AND TEMPORADA = :NEW.temporada;
        ELSE
            -- Inserta el primer resultado del equipo local
            INSERT INTO RESULTADOS (PUNTOS, GOLESAF, GOLESEC, PARTIDOSGANADOS, PARTIDOSEMPATADOS,
                                    PARTIDOSPERDIDOS, EQUIPO, DIVISION, TEMPORADA, NUMJORNADA)
            VALUES (puntosLocal, :NEW.golesLocal, :NEW.golesVisitante, ganaLocal, empate, ganaVisitante,
                   :NEW.local, :NEW.division, :NEW.temporada, 1);
            -- Inserta el primer resultado del equipo visitante
            INSERT INTO RESULTADOS (PUNTOS, GOLESAF, GOLESEC, PARTIDOSGANADOS, PARTIDOSEMPATADOS,
```

```
                                PARTIDOSPERDIDOS, EQUIPO, DIVISION, TEMPORADA, NUMJORNADA)
            VALUES (puntosVisitante, :NEW.golesVisitante, :NEW.golesLocal, ganaVisitante, empate, ganaLocal,
                   :NEW.visitante, :NEW.division, :NEW.temporada, 1);
        END IF;

        -- Zona para recalcular los puestos y ascensos
        -- Consulta que calcula los datos buenos a usar con un update para actualizar la parte de la tabla que
        nos interese...
        MERGE INTO resultados
        USING (
            SELECT ROW_NUMBER() OVER (ORDER BY puntos DESC) AS puesto, R1.partidosGanados,
                   R1.partidosEmpatados, R1.partidosPerdidos, R1.golesAF, R1.golesEC, R1.puntos, R1.equipo,
                   R1.division, R1.temporada, R1.numJornada,
            CASE WHEN puesto <= 3 AND division = '2ª' THEN 1 END AS asciende,
            CASE WHEN puesto  >= R2.numEquipos - 1 AND division = '1ª' THEN 1 END AS desciende,
            CASE WHEN puesto <= 5 AND division = '1ª' THEN 1 END AS europa
            FROM resultados R1, (
                SELECT COUNT(*) AS numEquipos
                FROM resultados
                WHERE temporada = :NEW.temporada AND division = :NEW.division
            ) R2
            WHERE R1.temporada = :NEW.temporada AND R1.division = :NEW.division
            ORDER BY R1.division, R1.puesto
        ) N
        ON (resultados.equipo = N.equipo AND resultados.temporada = :NEW.temporada AND
            resultados.numJornada = :NEW.numJornada AND resultados.division = :NEW.division)
        WHEN MATCHED THEN UPDATE
        SET resultados.puesto   = N.puesto,    resultados.asciende = N.asciende,
            resultados.desciende= N.desciende, resultados.europa   = N.europa;
    END AFTER EACH ROW;
END UPT_RESULTADOS;
```

Gestión del grupo

Reuniones

28/02 Reunión inicial, para organizarnos.

02/03 Comparación de los distintos modelos E/R planteados.

22/03 Probar a poblar la base de datos entera. Reunión de control, reparto de tareas.

27/03 Planteamiento de las consultas y modelo relacional.

29/03 Codificación de las consultas.

30/03 Planteamiento de los triggers.

01/04 Codificación de los triggers.

02/04 Elaboración de la memoria.

03/04 Elaboración de la presentación.

División del trabajo

Dado que cada integrante del grupo tiene distintos horarios, tratábamos de organizar las tareas de manera individual o por parejas para poder trabajar sin tener problemas de horarios. El modelo E/R fue realizado por todos, de manera independiente y después llegamos a una solución final poniéndolo en común. Posteriormente fue revisado en una tutoría para garantizar que era correcto, tras esto elaboramos el modelo relacional y normalizamos todo. A partir de este momento nos repartimos las tablas a poblar y los distintos datos a obtener así como procesar, entre los distintos integrantes del grupo. Una vez poblada la base nos dedicamos a plantear los árboles algebraicos de las consultas y posteriormente a programarlas. Con las consultas ya funcionando, nos dedicamos a pensar que triggers podíamos implementar para aportar una mayor robustez a la base de datos. De cara a la parte final del trabajo, concretamente la documentación y las optimizaciones trabajamos todos conjuntamente.

Dado que uno de los integrantes del grupo ya había cursado la asignatura y conocía perfectamente el calibre de la práctica así como la metodología del trabajo, la sintaxis, etc..., dicho integrante se ha encargado de ser el líder del grupo y de marcarnos los puntos de control, así de cómo asegurar la integridad de los distintos aportes que los otros integrantes realizabamos, esto ha repercutido muy positivamente en los tiempos así como en la manera de organizarnos internamente como grupo.

Problemas de coordinación

El principal problema ha sido que cada uno de los integrantes del grupo tenía horarios muy diferentes, lo cual dificulta mucho el poder trabajar conjunta y simultáneamente los 3. Esto dificultaba la coordinación.

Otro problema que tuvimos inicialmente fue que dado que uno de los integrantes ya había cursado la asignatura y tenía por lo tanto conocimientos previos, a los otros integrantes nos costaba entender ciertos conceptos porque eran nuevos para nosotros durante los primeros días de proyecto. Esto nos forzó a aprender más rápido por nuestra cuenta para poder alcanzar su ritmo de trabajo.