


B A S E S D E D A T O S

Base de datos de fútbol

C U R S O 2 2 / 2 3

G R U P O T 1 3

1 3 D E M A R Z O D E 2 0 2 3

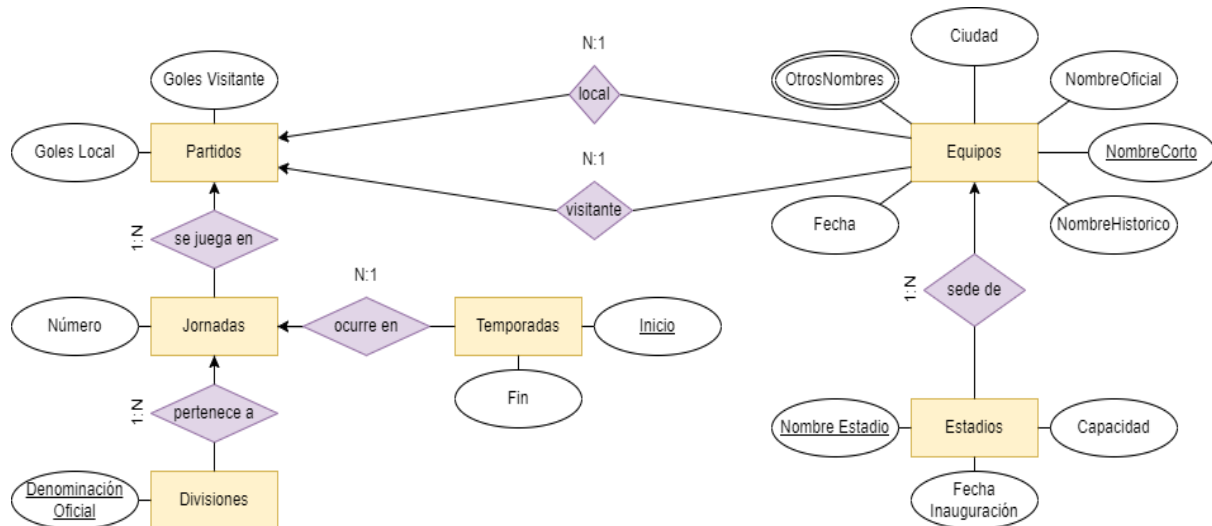


Lizer Bernad Ferrando, 779035@unizar.es
Guillermo Bajo Laborda, 842748@unizar.es
Axel Isaac Pazmiño Ortega, 817627@unizar.es

Parte I

Creación de una base de datos

Esquema E/R y restricciones:



En el esquema E/R aparecen un total de 6 entidades distintas: *Divisiones*, *Temporadas*, *Jornadas*, *Partidos*, *Equipos* y *Estadios*, cada cual con sus respectivos atributos. Estos atributos son principalmente los que se solicitaban en el enunciado, se distinguen atributos de distintos tipos: únicos (como el nombre de un estadio), multivaluados (como otros nombres de un equipo) y regulares. Estos nos proporcionan información sobre cada una de las entidades, así como los nombres de un equipo, la ciudad, la fecha de fundación, etc... A su vez, existen diversas relaciones entre las entidades (todas 1:N), que nos sirven para relacionar las entidades entre sí.

A continuación, mostramos las principales restricciones encontradas:

Divisiones:

- Un equipo solo puede participar en una división por temporada.
- No puede haber dos divisiones con la misma Denominación Oficial.

Temporadas:

- No puede haber dos temporadas con el mismo año de inicio.
- Si al año de finalización le restamos 1 debe de dar el año de inicio.

Jornadas:

- En una jornada un equipo no puede jugar más de una vez.

Partidos:

- Un partido debe ser jugado por 2 equipos distintos.

Equipos:

- No puede haber dos equipos con el mismo nombre corto.
- Cada equipo tiene un máximo de un estadio y solo se guardará el actual.

Estadios:

- Todo estadio es sede de al menos un equipo.
- No puede haber dos estadios con el mismo nombre.

Ya desde el inicio el esquema E/R es muy similar al diseño final. Sin embargo cabe destacar ciertos problemas o cuestiones menores que han surgido durante el diseño.

Para empezar cada uno de los integrantes del equipo ha diseñado por su cuenta el esquema y al poner en común por primera vez se han discutido los siguientes puntos:

- El uso de NombreCorto en vez de NombreOficial como clave primaria de Equipos.
- El uso de dos relaciones 1:N (local y visitante) para relacionar Equipos y Partidos.
- Organización general de las entidades Temporada y División respecto a Jornada.

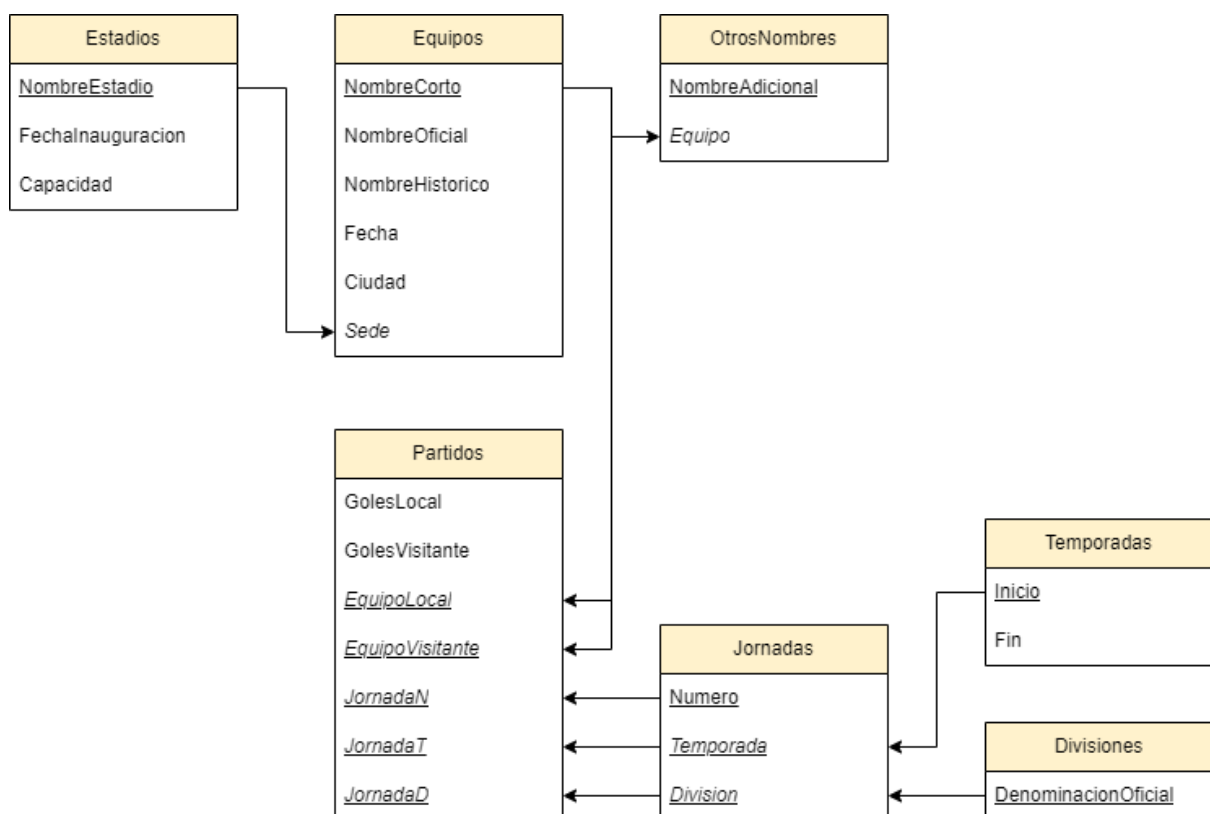
El primer punto se ha resuelto rápidamente tras observar detenidamente los datos. Se ha llegado a la conclusión de que algunos equipos no cuentan con NombreOficial como se esperaba y ante la disyuntiva de eliminar estos equipos o elegir otra clave se ha optado por lo segundo. Esto se debe a que el núcleo de la base de datos son los Partidos y por tanto no podemos simplemente eliminar equipos (y sus respectivos partidos) sin alterar el cálculo de clasificaciones en la base de datos. Se ha elegido como nueva clave primaria NombreCorto pues este es el nombre del EquipoLocal en cada fila de los datos y por tanto en caso de que un Equipo no aparezca ninguna vez en ese campo significa que ese equipo no participa en ningún partido y por lo tanto no es relevante en la base de datos.

El segundo punto también se ha resuelto rápidamente pues ha quedado claro que el uso de dos relaciones 1:N facilita enormemente las consultas y la estructura general de la base de datos.

El tercer punto ha sido el que más tiempo ha costado resolver y el que ha sufrido más transformaciones a lo largo del tiempo. En un principio se ha optado por una cadena

de relaciones 1:N División-Temporada-Jornada sin embargo esto nos ha llevado a tener Temporadas repetidas (una por cada División) por lo que nuestra segunda versión tiene Temporada como un atributo de Jornada y que a su vez formaba parte de su clave primaria. Sin embargo al final se llegó a la conclusión de que Jornada, Temporada y División debían tener el mismo peso y ser Entidades para facilitar las consultas. Por tanto para evitar la situación inicial pero lograr el objetivo se ha llegado al diseño actual y final.

Una vez el diseño E/R es satisfactorio se ha realizado la Normalización y el Modelo Relacional. En el proceso de Normalización no ha sido necesario hacer apenas cambios pues el modelo ya cumple con las propiedades de las formas normales exigidas por el enunciado, con una excepción. La excepción es el atributo multivaluado OtrosNombres en la entidad Equipos, el cual viola la 1FN y por tanto ha sido transformado con el método habitual en el modelo Relacional. Tras este pequeño cambio el Modelo Relacional está normalizado hasta la FNBC.



Sentencias SQL de creación de tablas:

La creación de tablas se ha llevado a cabo mediante el fichero crear_bd.sql. En él se encuentran las sentencias necesarias para poblar las 7 tablas que aparecen en el Modelo Relacional Normalizado. Estas tablas tienen relaciones entre sí por lo que se ha seguido un orden específico para crearlas, ya que se generarían problemas si se comenzase creando por ejemplo la tabla Partidos, donde se hace referencia a las tablas Equipos y Jornadas. Es por esto que el orden elegido para crear las tablas ha sido el siguiente: *Estadios, Equipos, OtrosNombres, Divisiones, Temporadas, Jornadas y Partidos*. Para el borrado de las mismas, que se lleva a cabo mediante el fichero borrar_bd.sql, también se debe seguir un orden específico, que es el inverso al de creación.

Las sentencias SQL de creación de tablas tienen una sintaxis específica que se muestra a continuación.

Supongamos la tabla Equipos, para crearla se debe hacer un “CREATE TABLE Equipos”, siendo Equipos el nombre que tiene la tabla. A continuación, se procede a incluir los atributos necesarios con sus respectivas características (UNIQUE, NOT NULL...). Para cada atributo se debe especificar un tipo, en nuestro caso se ha usado únicamente VARCHAR (carácteres) y NUMBER (números). Los tipos se especifican de la siguiente forma: “Nombre_Atributo VARCHAR(x),” siendo x el número de carácteres que tiene el atributo como máximo (lo mismo ocurre con INT). En nuestro caso, se ha especificado atributos de máximo 50 carácteres mientras que los atributos numéricos varían en función del atributo que representan adecuándose a las necesidades.

Para especificar condiciones como NOT NULL o CHECK's, se ha realizado de la siguiente forma: en el caso de GolesLocal, atributo de partidos el cual ha de ser mayor o igual a 0 y no puede ser nulo, se especifica pues “GolesLocal NUMBER(3) NOT NULL CHECK (GolesLocal >= 0)”.

Para el uso de claves ajenas y referenciación a otras tablas se ha usado la sintaxis: “FOREIGN KEY (Atrib) REFERENCES NomTabla(Atributo_Al_Que_Referencia)”. Por ejemplo en la tabla Partidos “FOREIGN KEY (EquipoLocal) REFERENCES Equipos(NombreCorto)”.

Por otra parte, para establecer la clave primaria, en caso de ser un atributo de la propia tabla has sido suficiente con añadir PRIMARY KEY después del tipo de la clave en cuestión. Si se trata de una clave primaria compuesta como en el caso de la tabla

OtrosNombres, se habrá tenido que usar la siguiente sintaxis: “CONSTRAINT Nombre_Clave PRIMARY KEY(Atributos_Clave)”, que en este caso es “CONSTRAINT OtrosNombres_PK PRIMARY KEY(NombreAdicional, Equipo)”.

A priori no ha habido muchos problemas con la creación de tablas. Lo que más ha costado ha sido aprender la sintaxis, en especial la de las claves ajenas, que generó bastantes errores hasta que finalmente se encontró la manera de referenciar correctamente.

Parte II

Introducción de datos y ejecución de consultas

Población de la base:

Para poblar la base de datos se ha utilizado el csv disponible en moodle como material de apoyo “ligahost.csv”. Inicialmente se han creado sentencias de población de tablas en SQL, sin contratiempos para las tablas de División, Jornada, Temporada y Estadio. A la hora de poblar la tabla Equipos ha habido diversos problemas, principalmente porque se exigía que diversos atributos de la entidad cumplieran la condición *NOT NULL*. Esto se solucionó eliminando dicha condición en atributos como Fecha, Ciudad, Sede, así como diversos nombres (salvo el corto, que es la clave primaria y no puede ser nula). Se optó por esta solución porque era preferible tener datos incompletos (pero no necesarios para las consultas) a eliminar filas enteras de datos. También hubo problemas con el atributo Sede, porque se la consideraba bastante importante para la entidad Equipo, por lo que no se quiso permitir la posibilidad de que fuera *NULL*. Es por esto por lo que se decidió rellenar manualmente los equipos sin estadio con búsquedas en la web de sus estadios auténticos para así completar los datos incompletos.

Sin embargo, finalmente se decidió deshacer los cambios realizados en el csv para partir única y exclusivamente de los datos originales. También se terminó poblando las tablas restantes (incluyendo Equipos y Estadios) con ctl, puesto que el procedimiento resultó bastante más cómodo que la población en SQL.

Se siguió la siguiente metodología para poblar cada una de las tablas restantes en ctl: se partió de ligahost.csv, de donde se crearon múltiples archivos csv, uno para cada tabla, dejando únicamente la información requerida para poblar cada tabla, respectivamente (p.ej para Estadios se eliminaron todas las columnas salvo Estadio, Fecha_Inag y Aforo). Se eliminaron las filas duplicadas para poblar las entidades, ya que no era procedente almacenar datos duplicados. Es importante tener en cuenta que en los archivos .ctl la dirección de los ficheros csv es relativa, en los archivos enviados se ha dejado la dirección el directorio donde se encontraban nuestros archivos .csv, para poblar desde otro dispositivo, esta se deberá modificar. El uso de ctl no supuso mucho problema y se poblaron las tablas restantes con bastante rapidez.

Uno de los únicos problemas que surgieron usando ctl a partir de los fragmentos de csv extraídos, fue que se cargaban los datos en las tablas y salía por pantalla un mensaje diciendo que se habían cargado 0 filas correctamente. Aparentemente el

código, las tablas y el csv estaban bien, pero tras indagar en la web se concluyó que se debía cambiar el formato de fin de línea de 'CR' 'LF' a 'LF'. Una vez realizado este cambio desde el visualizador VSC, todas las filas se cargaron correctamente.

Consulta 1:

```
-- Equipo que más ligas de primera división ha ganado.
-- puntos_temporada (Equipo,puntos,temporada)
CREATE VIEW puntos_temporada AS
    SELECT partidos_puntos_div1.equipo as equipo,
-- los puntos totales de cada equipo en cada temporada en primera division
    sum(partidos_puntos_div1.puntos) as puntos,
-- partidos_puntos_div1 = t1 + tv
    partidos_puntos_div1.temporada as temporada
-- tuplas t1 (nombreEquipoLocal, puntos, jornada y temporada)
    FROM ( SELECT equipolocal as equipo,
        CASE
-- si gana el partido 3 puntos
            WHEN goleslocal > golesvisitante THEN 3
-- si empata 1 punto
            WHEN goleslocal = golesvisitante THEN 1
-- si pierde 0 puntos
            ELSE 0
        END AS puntos,jornadaN as jornada,jornadaT as temporada
    FROM Partidos
    WHERE jornadaD='2'
    UNION ALL
-- tuplas tv (nombreEquipoVisitante, puntos, jornada y temporada)
    SELECT equipovisitante as equipo,
        CASE
            WHEN golesvisitante > goleslocal THEN 3
            WHEN goleslocal = golesvisitante THEN 1
            ELSE 0
        END AS puntos,jornadaN as jornada ,jornadaT as temporada
    FROM Partidos
    WHERE jornadaD='2') partidos_puntos_div1
-- queremos los puntos de cada equipo en cada temporada
GROUP BY partidos_puntos_div1.equipo,partidos_puntos_div1.temporada;
-- num_win_liga (Equipo,numLigasGanadas)
CREATE VIEW num_win_liga AS
-- numero de ligas que ha ganado un equipo de primera division
SELECT win_temp.equipo, count(*) numLigasGanadas
FROM (
    SELECT a.equipo as equipo, a.temporada
    FROM puntos_temporada a
    INNER JOIN
    (
        SELECT temporada,
        max(puntos) as max_puntos
        FROM puntos_temporada
        GROUP BY temporada
-- tupla maximo de puntos en una temporada
    ) max_puntos_temporada
    ON a.temporada = max_puntos_temporada.temporada
-- nos quedamos con los equipos que ganaron en cada temporada
    AND a.puntos = max_puntos_temporada.max_puntos) win_temp
-- agrupamos por equipo para calcular la suma
GROUP BY Equipo;

SELECT equipo
```



```

FROM num_win_liga
WHERE num_win_liga.numLigasGanadas = (SELECT max(num_win_liga.numLigasGanadas)
-- finalmente calculamos el equipo que mas veces ha ganado ligas de primera division
FROM num_win_liga);

```

Salida:

Vista creada.

Vista creada.

EQUIPO

Real Madrid

Consulta 2:

```

--Estadios en los que el local ha ganado o empatado más del 85% de las veces
SELECT e.NombreEstadio -- seleccionamos nombres de estadios donde...
FROM Estadios e
WHERE
    (SELECT COUNT(*) FROM Partidos p1, Equipos eq1 -- contamos el total de
partidos ganados
    WHERE e.NombreEstadio = eq1.Sede
    AND eq1.NombreCorto = p1.EquipoLocal
    AND GolesLocal >= GolesVisitante -- los partidos ganados o empatados por
el local
    ) >
-- son mayores que
    (SELECT COUNT(*) FROM Partidos p2, Equipos eq2 -- contamos el total de
partidos jugados
    WHERE e.NombreEstadio = eq2.Sede
    AND eq2.NombreCorto = p2.EquipoLocal) *85/100; -- multiplicamos el total
*0.85%
--seleccionara aquellos en el que los ganados > 85% (total jugados*0.85)

```

Salida:

NOMBREESTADIO

Camp Nou
Santiago Bernabeu
Nuevo Lasesarre
Narcis Sala

Consulta 3:

```
SELECT DISTINCT p.JornadaT, golL + golV
FROM (
    SELECT p1.JornadaT as tl, sum(p1.goleslocal) as golL
    FROM partidos p1
    WHERE p1.EquipoLocal = 'Zaragoza'
    AND p1.goleslocal > p1.GolesVisitante
    AND EXISTS (
        SELECT *
        FROM partidos p2
        WHERE p2.JornadaT = p1.JornadaT
        AND p2.EquipoVisitante = 'Zaragoza'
        AND p2.EquipoLocal = p1.EquipoVisitante
        AND p2.goleslocal < p2.GolesVisitante
    )
    GROUP BY JornadaT
    HAVING COUNT(*) >= 4
),
(
    SELECT p1.JornadaT as tv, sum(p1.GolesVisitante) as golV
    FROM partidos p1
    WHERE p1.EquipoVisitante = 'Zaragoza'
    AND p1.goleslocal < p1.GolesVisitante
    AND EXISTS (
        SELECT *
        FROM partidos p2
        WHERE p2.JornadaT = p1.JornadaT
        AND p2.EquipoLocal = 'Zaragoza'
        AND p1.EquipoLocal = p2.EquipoVisitante
        AND p2.goleslocal > p2.GolesVisitante
    )
    GROUP BY JornadaT
    HAVING COUNT(*) >= 4), partidos p
WHERE p.JornadaT = tl and p.JornadaT = tv;
```

Salida:

JORNADAT GOLL+GOLV

```
-----
1982    25
1998    19
2008    34
1985    21
2002    25
```

Parte III

Diseño físico

Valoración del rendimiento:

Tras haber realizado las consultas, se han utilizado los comandos proporcionados (“EXPLAIN PLAN FOR <consulta>”, que guarda en DBMS_XPLAN el resumen del plan de ejecución de la consulta y su coste, y “SELECT PLAN_TABLE_OUTPUT FROM TABLE”, que muestra el plan de ejecución de la última consulta explicada.).

La consulta 1 tiene un tiempo de CPU que consideramos adecuado, en cambio, consideramos que debido al elevado número de tuplas en la ista partidos_puntos_div se valoraba almacenar los datos en una tabla intermedia para reducir el tiempo de CPU, con su respectivo trigger para gestionar los cambios en la tabla partidos, que afectan directamente a esta.

En el grupo se ha meditado implementar esta mejora pero finalmente la decisión final ha sido de no hacerlo puesto que se ha considerado que no se reduciría notoriamente el tiempo de CPU.

Respecto a las consultas 2 y 3, se aprecia que el tiempo de CPU es muy bajo, tanto que no merece la pena realizar modificaciones en la base de datos. Esto se debe a que se considera que el plan de ejecución de la consulta se ajusta a nuestras expectativas

Restricciones y creación de triggers:

Se han encontrado diversidad de restricciones a tener en cuenta, algunas son las siguientes:

- Una temporada debe de concluir al año siguiente de haber comenzado
- Un equipo no puede aparecer dos veces en el mismo partido
- Un equipo no puede jugar en dos partidos de la misma jornada
- No puedes jugar contra un equipo como local en dos ocasiones en la misma temporada
- No puedes participar en dos partidos de distintas divisiones en la misma jornada
- Un equipo no puede jugar en partidos de dos divisiones distintas en la misma temporada

De esta forma, se han implementado los siguientes triggers:

1. Un equipo no puede aparecer dos veces en el mismo partido

```
-- Verifica que el equipo local no sea el mismo que el visitante
CREATE OR REPLACE TRIGGER LocalNoVisitante
BEFORE INSERT OR UPDATE ON Partidos
FOR EACH ROW
-- se dispara cuando ambos equipos son iguales
WHEN (NEW.EquipoLocal = NEW.EquipoVisitante)
BEGIN
    RAISE_APPLICATION_ERROR(-20001, 'Un equipo no puede
enfrentarse a si mismo.');
```

END LocalNoVisitante;

/

2. Un equipo no puede aparecer dos veces en el mismo partido

```
-- Verifica que la temporada insertada sea valida
CREATE OR REPLACE TRIGGER TempValida
BEFORE INSERT OR UPDATE ON Temporadas
FOR EACH ROW
-- se dispara cuando la temporada no acaba al año siguiente de
comenzar
WHEN (NEW.Fin - NEW.Inicio != 1)
BEGIN
    RAISE_APPLICATION_ERROR(-20002, 'Temporada invalida.');
```

END TempValida;

/

3. Un equipo no puede jugar en dos partidos de la misma jornada

```
-- Verifica que ninguno de los equipos introducidos haya jugado ya en
esa jornada
CREATE OR REPLACE TRIGGER RepiteEnJornada
BEFORE INSERT OR UPDATE ON Partidos
FOR EACH ROW
DECLARE
    jugados NUMBER;
BEGIN
    SELECT COUNT(*)
    INTO jugados
    FROM Jornadas, Equipos
    WHERE JornadaT = :NEW.JornadaT
    AND JornadaD = :NEW.JornadaD
    AND JornadaN = :NEW.JornadaN
    AND (NombreCorto = :EW.EquipoLocal OR NombreCorto =
NEW.EquipoVisitante);

    -- si uno de los dos equipos ya ha jugado en esa jornada, salta el
error
    IF (jugados > 0) THEN
        RAISE_APPLICATION_ERROR(-20003, 'Cada equipo puede jugar un
maximo de una vez por jornada.');
```

END IF;

END RepiteEnJornada;
/

Consideraciones Organizativas Finales:

Para la realización de este proyecto han sido necesarias 6 horas de trabajo conjunto en los horarios de prácticas y alrededor de 10 y 15 horas de trabajo individual por cada miembro del equipo.

Respecto al reparto de trabajo, se ha intentado que todos los miembros participaran en todos los apartados para así tener varios puntos de vista y poder debatir sobre las decisiones tomadas a lo largo del desarrollo de la base de datos. Aunque es cierto que ciertos apartados han sido llevados a cabo en mayor medida por un integrante u otro de forma más bien voluntaria que por asignación de trabajo propiamente. En definitiva, se podría decir que todos los integrantes del equipo han participado de forma bastante equitativa en la elaboración del proyecto.

Por tanto no ha habido problemas de organización como tal excepto tal vez cuando se tuvo que comenzar a hacer el Modelo Relacional. Esto se debe a que cuando el siguiente paso a seguir en el proyecto era realizar el Modelo Relacional y su Normalización aún no se habían visto los contenidos teóricos necesarios en clase, lo cual paralizó durante un corto periodo de tiempo el desarrollo del proyecto.