

PRÁCTICA FINAL

21751 – Sistemas de Gestión
de Bases de Datos



6 FEBRERO

PROYECTO FINAL

Mario Ventura Burgos 43223476-J

Grado en Ingeniería Informática (GIN 3)

CURSO 2023-2024

Índice

Índice	2
Introducción	4
General	5
Modelo conceptual de datos.....	5
Aclaraciones y explicaciones sobre el modelo.....	6
Paso a relacional del modelo conceptual de datos	10
Pasos 1 y 2:	10
Pasos 3 y 4:	11
Paso 5:	11
Normalización	12
Primera forma normal (1FN).....	12
Segunda forma normal (2FN).....	12
Tercera forma normal (3FN)	13
Cuarta forma normal (4FN)	13
Quinta forma normal (5FN).....	13
Desnormalizaciones.....	14
Modelo final.....	15
MySQL	16
Creación de un fichero con la implementación del modelo	16
Creación de un tablespace de 5GB y cota máxima 10GB.....	18
Creación de una base de datos IMPBD para la importación	20
Creación de un usuario IMPBD con privilegios.....	22
Conexión con el usuario IMPBD y crear las tablas	22
Importación de los archivos en las tablas creadas	25
Código de los scripts ODS.....	28
Código del script que trata el fichero CSV.....	30
Creación de una segunda base de datos DEFBD	32
Creación de un usuario MIGBD	33
Creación de esquema de tablas con MIGBD	34
Realización de script SQL para traspaso de información	39
Tabla Ubicación.....	39
Tabla Persona.....	43

Tabla Torero	47
Tabla Apoderat.....	47
Tabla Plaza	48
Tabla Actuación.....	49
Tabla Ramaderia	50
Tabla Toro	51
Tabla Feria	52
Tabla Esdeveniment	53
Tabla R_Esdeveniment_Toro	55
Actualización de fechas de nacimiento de toros.....	56
Realización de análisis para optimización	62
Análisis.....	62
Mejora	63
PostgreSQL	64
Análisis y explicación de las pautas de configuración de FDW	64
Creación de un espacio de almacenamiento llamado Data	69
Creación de una base de datos FET	70
Creación de un esquema temp dentro de la base de datos anterior	70
Creación del usuario UFDW	71
Realización de sentencias SQL.....	72
Media de peso de toros lidiados por año.	72
Número de toros lidiados en el estado español	74
Oracle	76
Configuración del gestor.....	76
Creación del usuario Utest	79
Migración de tablas y ejecución de consulta	82
Migración	82
Consulta	86
Plan de ejecución y operadores algebraicos	88
Plan de ejecución.....	88
Operadores algebraicos.....	89
Conclusiones.....	91
Referencias	93

Introducción

En este proyecto se nos pide realizar una serie de tareas relacionadas con la información recopilada en diferentes archivos sobre la Real federación taurina de España. Esta federación ha ido almacenando sus datos más importantes desde sus inicios y actualmente cuenta con 5 ficheros para almacenar los datos. Estos ficheros son los siguientes:

1. APODERATS.ods
2. TOREROS.ods
3. esdeveniments.csv
4. PLACES BOUS.ods
5. ACTUACIONES.ods

Respecto a estos ficheros y la información que contienen, se pide que se haga el modelo de datos que corresponda y, posteriormente, se pide crear una base de datos en MySQL para migrar los datos a ella. Finalmente, se migrará nuevamente estos datos desde MySQL a PostgreSQL y se llevarán a cabo una serie de actividades con Oracle.

A continuación, la solución propuesta para cada una de las tareas a realizar.

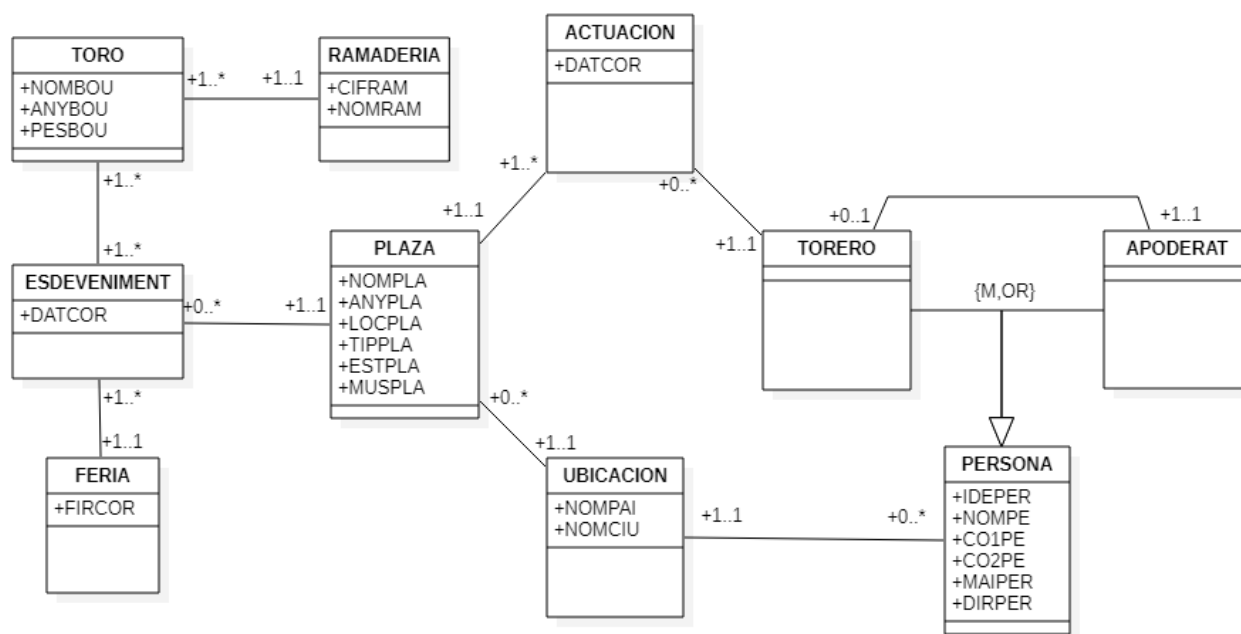
General

En este apartado realizaremos el modelo de datos correspondiente a la información presentada, definiendo todos los elementos necesarios para mantener y asegurar la coherencia y consistencia de la información en una supuesta base de datos. Esto incluye, en caso de que se considere necesarios implementarlos, elementos como procedures, events, triggers, checks, constraints, etc.

Posteriormente, también se comentará el significado de las multiplicidades, relaciones y clases que conforman el modelo, así como el paso a relacional del modelo presentado, la forma normal a la que se llega y las desnormalizaciones aplicadas, si es que hay alguna.

Modelo conceptual de datos

El modelo de datos definitivo creado es el siguiente:



Antes de hacer el paso a relacional, se harán unas aclaraciones sobre el modelo presentado. Después se llevará a cabo paso a relacional para establecer claves, identificar posibles nuevas claves, y estudiar la posibilidad de fusión en las tablas. Finalmente, haremos un proceso de normalización, asegurando la consistencia y coherencia de datos y eliminando posibles redundancias.

Aclaraciones y explicaciones sobre el modelo

Respecto al modelo de datos presentado, es importante tener en cuenta las siguientes consideraciones para asegurar su completa comprensión (se explicará todas las clases y multiplicidades, pero solo se demostrará mediante consultas aquellas multiplicidades que puedan considerarse susceptibles de causar confusión en la comprensión del modelo):

- **Herencia de la clase Persona** → Dado que los toreros y apoderados son ambas personas, y teniendo en cuenta que de ambas clases hay cierta información en común que deseamos almacenar, se ha optado por hacer una herencia creando una clase persona de la cual heredan torero y apoderat. La clase madre (Persona) tendrá la información del nombre, los dos apellidos de la persona, ya sea torero o apoderat, el ID que identifica a la persona, el mail de la persona y la dirección de la persona. Esto se hace debido a que los atributos son realmente los mismos y tenerlos por separado en ambas tablas no es lo óptimo. Estos atributos se añadirán a la clase madre (Persona) y serán heredados por los hijos. La herencia es de tipo {M,OR} ya que se considera que una persona de la base de datos que sea torero podrá ser apoderat en un futuro.
- **Relación entre torero y apoderat** → La relación entre torero y apoderat se debe a que se considera que un torero tiene un (y solo un) apoderado (multiplicidad 1..1), y un apoderado puede ser apoderado de 0 o 1 torero (0..1). Podemos comprobar estas multiplicidades mediante dos consultas hechas sobre el modelo de datos original presentado por el enunciado de la práctica.

```
-- COMPROBACIÓN DE SI UN TORERO TIENE UN ÚNICO APODERADO
-- Se seleccionan Los toreros con más de 1 apoderado
SELECT IDEAPO AS ID, COUNT(IDETOR) AS QTT_APODERADO
FROM TORERO
WHERE IDEAPO IS NULL
GROUP BY ID HAVING COUNT(ID) > 1;

-- COMPROBACIÓN DE SI EXISTEN TOREROS SIN APODERADO
-- Se seleccionan Los toreros sin apoderado
SELECT IDETOR AS ID FROM TORERO WHERE IDEAPO IS NULL;
```

```
mysql> SELECT IDEAPO AS ID, COUNT(IDETOR) AS QTT_APODERADO
-> FROM TORERO
-> WHERE IDEAPO IS NULL
-> GROUP BY ID HAVING COUNT(ID) > 1;
Empty set (0.69 sec)

mysql> SELECT IDETOR AS ID FROM TORERO WHERE IDEAPO IS NULL;
Empty set (0.19 sec)
```

- **Clase actuación** → La clase actuación representa una actuación de un torero en concreto en una plaza en concreto. Teniendo en cuenta que en el fichero ods original esta tabla contaba con únicamente 3 atributos de los cuales 2 hacen referencia al torero y la plaza, se ha optado por hacer una tabla actuación que tenga solo un atributo que hace referencia a la fecha de la actuación.

Se ha relacionado esta clase con la clase plaza de forma que una actuación concreta tiene lugar en 1 plaza concreta y en una plaza puede haber muchas actuaciones (1..1 y 1..*); y con la clase torero, de forma que una actuación va asociada a solamente un torero, y un torero puede hacer varias actuaciones (1..1 y 0..*).

De esta manera se sabe para cada actuación, el torero que la realizó y la plaza donde se realizó.

```
-- COMPROBACIÓN DE MULTIPLICIDAD 0..* DE TORERO CON ACTUACION
-- Se seleccionan los toreros que no tienen actuación asociada
SELECT COUNT(*) FROM TORERO WHERE IDTOR NOT IN (
    SELECT DISTINCT IDTOR AS ID_TORERO
    FROM ACTUACION
);
```

```
mysql> SELECT COUNT(*) FROM TORERO WHERE IDTOR NOT IN (
->     SELECT DISTINCT IDTOR AS ID_TORERO
->     FROM ACTUACION
-> );
+-----+
| COUNT(*) |
+-----+
|    96572 |
+-----+
1 row in set (2.07 sec)
```

- **Clase ubicación** → La clase ubicación se ha creado para eliminar redundancias. Los toreros, apoderados y plazas tienen todos un atributo con el nombre de un país y de una ciudad. Estas redundancias nos llevan a juntar estos atributos en una misma clase a la que llamamos ubicación, y que relacionaremos con estas otras tres clases mencionadas.

La relación entre ubicación y persona se debe a que, como los toreros y apoderados tienen ambos los atributos de ciudad y país, y ambos heredan de la clase madre persona, basta con relacionar la ubicación con una persona. Una persona va asociada a 1 única ubicación ya que esto representa el país y ciudad de residencia de la persona; y en una misma ubicación (mismo país y misma ciudad) puede haber muchas personas o ninguna (0..*).

- **Clase esdeveniment** → Esta clase representa un acontecimiento en concreto, es decir, un evento taurino. Un evento o esdeveniment tiene el atributo *datacor* para representar la fecha.

Un esdeveniment está tiene lugar en una plaza en concreto (1..1), y en una plaza pueden organizarse 0 o muchos esdeveniments (0..*). Además, un esdeveniment en concreto está relacionado con una sola feria, pero en una feria pueden hacerse varios eventos/esdeveniments (multiplicidad 1..1 y 1..*)


```
-- COMPROBACIÓN DE SI EN UNA PLAZA SE PUEDEN haber CERO O VARIOS ESDEVENIMENTS
-- Selección de plazas con más de 1 esdeveniment, limitado a 10 plazas
-- máximo y orden de resultado de mayor a menor
SELECT PLAZA.NOMPLA AS NOMBRE, COUNT(DISTINCT ESDEVENIMENT.FIRCOR) AS NUM_ESD
FROM PLAZA
    LEFT JOIN ESDEVENIMENT
    ON ESDEVENIMENT.NOMPLA = PLAZA.NOMPLA
GROUP BY PLAZA.NOMPLA HAVING COUNT(ESDEVENIMENT.FIRCOR) > 1
ORDER BY NUM_ESD DESC
LIMIT 10;

-- Selección de esdeveniments con 0 plazas asociadas
SELECT PLAZA.NOMPLA FROM PLAZA
    LEFT JOIN ESDEVENIMENT
    ON PLAZA.NOMPLA = ESDEVENIMENT.NOMPLA
WHERE ESDEVENIMENT.NOMPLA IS NULL
LIMIT 10;
```

NOMBRE	NUM_ESD
Cehegín	185
Manzanares	180
Monumental de Ambato	180
Coliseum de La Coruña	179
El Burgo	179
Alberto Balderas	178
Coso de Sta. Eugenia	177
Guadalupe	177
Alcalá de Henares	176
Almería	176

10 rows in set (1 min 37.08 sec)

NOMPLA
Palacio de Vistalegre
Palacio del Arte
Palencia
Palmares
Parla
Pedro Bernardo
Pepe Cáceres
Pino
Plasencia
Pontevedra

- **Clase Toro** → Esta clase representa a un toro y cuenta con los atributos siguientes: nombou, que representa el nombre del toro; pesbou, que representa el peso del toro; y anybou que representa el año de nacimiento del toro.

Esta clase se relaciona con la tabla esdeveniment ya que se considera que un toro en concreto puede haber participado en muchos eventos (1..*) y un evento concreto puede incluir la participación de varios toros, pero mínimo uno (1..*).

- **Clase Ramaderia** → Esta clase representa la ganadería a la que pertenece un toro en concreto. Por ello, esta clase se relaciona con la clase toro con multiplicidad 1..* ya que una ganadería puede tener muchos toros, y un toro está asociado con una sola ganadería ya que pertenece solamente a una. La clase Ramaderia contiene el atributo cifram que representa el cif de la ganadería a la que pertenece el toro, y el atributo nomram con el nombre de la ganadería a la que pertenece el toro.

- **Clase plaza** → La clase plaza representa la plaza de toros en la que tiene lugar el evento taurino. Cuenta con los atributos *nompla* (nombre de la plaza), *anypla* (año de construcción de la plaza), *locpla* (número de localidades o asientos en la plaza), *tippla* (tipo de plaza, que se refiere a si la plaza es fija o móvil), *estpla* (estilos predominantes en la construcción) y *muspla* (indica si la plaza contiene un museo o no).

En una plaza se pueden hacer una o muchas actuaciones y una actuación se hace solo en una plaza (relación 1..* y 1..1 con actuación), una plaza está en una única ubicación (país y ciudad) pero en una misma ubicación puede haber 0 o muchas plazas (relación 1..1 y 0..* con ubicación), y una plaza puede estar asociada a muchos acontecimientos/eventos o ninguno, mientras que un esdeveniment/evento/acontecimiento se realiza en una plaza en concreto (relación 0..* y 1..1 con esdeveniment).

- **Clase feria** → Por último, respecto a la clase feria, esta representa una feria en concreto en la que tiene lugar uno o varios eventos taurinos (esdeveniment). Esta clase se ha creado para eliminar redundancias en la clase esdeveniment, y además permite la tipificación ya que todas las ferias existentes se almacenan en una misma tabla en lugar de referirnos a ellas como atributo dentro de la clase esdeveniment.

Comprobamos la cantidad de apariciones de cada Feria en la tabla esdeveniment y vemos que se trata de una cantidad notable de apariciones. Si se suma las apariciones de entre las ferias que más aparecen, se suma más de medio millón de apariciones, cosa que nos lleva a crear una tabla feria con el atributo *fircor*. Esto presenta ventajas como normalización (hablaremos de normalización más adelante), ahorro de espacio (ya no hay varchar's que aparecen más de 100.000 veces, sino foreign keys que harán referencia a ellos) y, por tanto, mayor rendimiento en consultas.

```
-- COMPROBACIÓN DEL NÚMERO DE APARICIONES DE LAS FERIAS QUE MÁS APARECEN
-- EN LA TABLA ESDEVENIMENT. COMPROBAMOS REDUNDANCIAS CON CLÁUSULA HAVING
-- Selección de las 10 ferias con más apariciones en esdeveniment
SELECT FIRCOR AS NOMBRE_FERIA, COUNT(*) AS NUM_APARICIONES
FROM ESDEVENIMENT
GROUP BY FIRCOR HAVING COUNT(*) > 1
ORDER BY NUM_APARICIONES DESC
LIMIT 10;
```

NOMBRE_FERIA	NUM_APARICIONES
Feria de Barcelona	145584
Feria de Málaga	135594
Feria de Quito	111024
Feria de Tarragona	100224
Feria de Lisboa	96390
Feria de Guadalajara	88344
Feria de Almería	83052
Feria de Alcalá de Henares	82998
Feria de Andújar	82728
Feria de Madridejos	82620

10 rows in set (1 min 54.53 sec)

En una feria pueden realizarse uno o varios esdeveniments, pero los esdeveniments van asociados siempre a una sola feria ya que un evento taurino concreto se realiza en una feria en concreto (no se puede realizar un evento taurino en dos ferias diferentes porque, por lógica, entonces son dos eventos taurinos diferentes).

Paso a relacional del modelo conceptual de datos

Ahora que se ha presentado y explicado el modelo conceptual de datos, haremos el paso a relacional del modelo anterior con el objetivo de obtener el código de creación de las tablas. A continuación, los 5 pasos del paso de conceptual a relacional.

Pasos 1 y 2: Cada tabla del modelo conceptual es una tabla en el modelo relacional. Para cada una de las tablas se escogerá una clave o se creará una. Las tablas del modelo y sus claves son:

- PERSONA(**ideper**, nompe, co1pe, co2pe, maiper, dirper)
- TORERO(**idetor**)
- APODERAT(**ideapo**)
- ACTUACION(**ideact**, datcor)
- PLAZA(**nompla**, anypla, locpla, tippla, estpla, muspla)
- TORO(**idebou**, nombou, anybou, pesbou)
- RAMADERIA(**cifram**, nomram)
- UBICACION(**ideubi**, nompai, nomciu)
- ESDEVENIMENT(**ideesd**, datcor)
- FERIA(**fircor**)

Donde los atributos subrayados y en negrita son la clave primaria (PK) de la tabla.

Se ha supuesto que los atributos NOMPLA y CIFRAM son únicos (unicidad) y por ello podrían actuar como clave de sus respectivas clases sin necesidad de generar una clave nueva.

Además, las clases TORERO y APODERAT heredan de Persona y, por tanto, heredan la clave principal de persona.

Pasos 3 y 4: Cada relación del modelo conceptual es una tabla del modelo relacional. Para cada una de estas tablas seleccionaremos una clave en base a las multiplicidades de la relación, siguiendo la jerarquía siguiente: 1..* y 0..*, 0..1, 1..1. Las tablas en estos pasos son las siguientes:

- R_TORERO_APODERAT(idetor, ideapo)
- R_TORERO_ACTUACION(ideact, idetor)
- R_ACTUACION_PLAZA(ideact, nompla)
- R_PLAZA_UBICACION(nompla, ideubi)
- R_UBICACION_PERSONA(ideper, ideubi)
- R_PLAZA_ESDEVENIMENT(ideesd, nompla)
- R_ESDEVENIMENT_TORO(ideesd, idebou)
- R_TORO_RAMADERIA(idebou, cifram)
- R_ESDEVENIMENT_FERIA(ideesd, fircor)

Paso 5: Ahora estudiaremos la posibilidad de fusionar las tablas de los pasos 1 y 2 con las tablas de los pasos 3 y 4. Las tablas candidatas son las que tienen las mismas claves en los pasos 1/2 y 3/4. Las tablas finales obtenidas son las siguientes:

- Fusión de la tabla TORERO con la tabla R_TORERO_APODERAT: Tienen la misma clave (idetor), por tanto, se fusionan y queda de la siguiente manera:

TORERO(idetor, ideapo).

Donde ideapo es una Foreign Key a la tabla APODERAT.

- Fusión de la tabla ACTUACIÓN con la tabla R_TORERO_ACTUACION y R_ACTUACION_PLAZA: Queda de la siguiente manera:

ACTUACION(ideact, idetor, nompla, datcor).

Donde idetor es una Foreign Key a la tabla TORERO y nompla es una Foreign Key a la tabla PLAZA.

- Fusión de PLAZA con R_PLAZA_UBICACION: El resultado de la fusión es el siguiente:

PLAZA(nompla, ideubi, anypla, locpla, tippla, estpla, muspla).

Donde ideubi es una Foreign Key a la tabla UBICACION.

- Fusión de PERSONA con R_UBICACION_PERSONA:

PERSONA(ideper, ideubi, nompe, co1pe, co2pe, maiper, dirper).

Donde el atributo ideubi es una Foreign Key a la tabla UBICACION.

- Fusión de TORO con R_TORO_RAMADERIA:

TORO(idebou, cifram, nombou, anybou, pesbou)

Donde cifram es una Foreign Key a la tabla RAMADERIA

- Fusión de ESDEVENIMENT con R_PLAZA_ESDEVENIMENT con R_ESDEVENIMENT_FERIA. La fusión es la siguiente:

ESDEVENIMENT(ideesd, nompla, fircor, datcor).

Donde los atributos nompla y fircor son Foreign Key a las tablas PLAZA y FERIA respectivamente.

- Se puede apreciar que en los pasos 3 y 4 aparece una tabla “intermedia” entre ESDEVENIMENT y TORO. Esta tabla se llama R_ESDEVENIMENT_TORO y aparecerá como tabla nueva ya que tiene una clave compuesta que referencia a un toro concreto de un esdeveniment concreto.
- Las tablas que no se han fusionado quedarán igual que en el resultado de los pasos 1 y 2.
- Por último, cabe destacar que dada la etiqueta ({M,OR}) de la herencia donde las clases TORERO y APODERAT heredan de la clase madre PERSONA, esta herencia no puede fusionarse de ninguna manera y por tanto no se fusiona.

Ahora analizaremos la forma normal actual de las tablas que tenemos, y estudiaremos la posibilidad de hacer modificaciones por tal de alcanzar una forma normal que elimine redundancias en la información y mantenga la coherencia, consistencia, integridad y adaptabilidad de los datos.

Normalización

Se pretende alcanzar la tercera forma normal (3FN):

Primera forma normal (1FN)

Para encontrarnos en 1FN debemos asegurarnos de que los valores de las tablas son atómicos, es decir, cada celda de la tabla debe contener un único valor atómico, no debe haber conjuntos, listas o valores compuestos. Cada fila debe ser única, por tanto, no puede haber filas duplicadas.

Esta forma normal se cumple ya que todos los datos son atómicos, no existen celdas con valores compuestos ni filas duplicadas. Podemos asegurar que estamos en 1FN.

Segunda forma normal (2FN)

Respecto a la segunda forma normal, para asegurar que estamos en 2FN debe suceder lo siguiente:

- Estar en 1FN
- Todo atributo no clave depende funcionalmente en forma completa de la clave primaria

Podemos asegurar que estamos en 1FN, y también podemos asegurar que la segunda condición se cumple ya que no existen atributos que no dependan funcionalmente de forma completa de la clave de la relación.

Tercera forma normal (3FN)

Para poder asegurar que estamos en 3FN se debe asegurar que se cumplen las siguientes condiciones:

- Está en 2FN.
- Ningún atributo no clave depende funcionalmente de ningún otro subconjunto de atributos no clave.

Podemos asegurar que estamos en 2FN, pero la segunda condición no se cumple ya que en la clase UBICACIÓN existe el atributo *nompai* que genera una dependencia transitiva. Esto se debe a que el país depende en cierta forma de la ciudad, ya que en un país puede haber muchas ciudades y, por tanto, distintas ubicaciones en la tabla *ubicación* pueden tener valores repetidos en la tabla.

Por otro lado, también podría suceder que ciudades de países diferentes tengan el mismo nombre (por ejemplo, las ciudades de Ámsterdam (Países Bajos) y Ámsterdam (Estados Unidos), no están en nuestra base de datos, pero esto es una posibilidad en el mundo real y se debe tener en cuenta).

Por tanto, alcanzaremos la tercera forma normal creando una tabla nueva que separe estos dos atributos:

- UBICACION(*ideubi*, *nompai*, *nomciu*)
- PAIS(*nompai*)

Cuarta forma normal (4FN)

Para asegurar que alcanzamos la cuarta forma normal debemos asegurarnos de que se cumplen las siguientes condiciones:

- Estar en 3FN
- No se deben tener dependencias multivaluadas independientes, es decir, que para un valor de X haya un conjunto de valores Y asociados que no dependan de otros atributos de la clase.

En este caso puede asegurarse que ambas condiciones se cumplen y, por tanto, se alcanza la cuarta forma normal.

Quinta forma normal (5FN)

Una relación está en 5FN si, y solamente si:

- Está en 4FN
- No existen dependencias de proyección/combinación sin variación de información

Se puede asegurar que estamos en 4FN, pero para poder asegurar la segunda condición tenemos que identificar las posibles dependencias de combinación o proyección. Estas se dan cuando una clase se

puede obtener como resultado de la operación de proyección de otras clases, es decir, cuando se puede descomponer una clase en 2 o más sin causar pérdidas de información. En nuestro caso, existen tablas como PERSONA o PLAZA que cumplen esta condición y, por tanto, nos impiden alcanzar la quinta forma normal en el modelo.

Para poder alcanzar 5FN, se deberían separar estas tablas siguiendo la lógica siguiente:

- PERSONA_NOM(**ideper**, nompe)
- PERSONA_APE(**ideper**, co1pe, co2pe)
- PERSONA_MAI(**ideper**, maiper)
- PERSONA_DIR(**ideper**, dirper)
- PERSONA_CIU(**ideper**, ideubi)

De esta forma, se eliminarían estas dependencias y se podría asegurar que se alcanza la quinta forma normal.

Desnormalizaciones

A continuación, se estudia la posibilidad de aplicar desnormalizaciones con el objetivo de alcanzar un equilibrio entre normalización y rendimiento en la base de datos real implementada. Se puede desnormalizar deliberadamente en algunas ocasiones en las que se considere que implementar una base de datos que alcanza cierta forma normal tendría consecuencias negativas en el rendimiento de esta base de datos.

En nuestro caso, alcanzar la quinta forma normal implicaría una separación de ciertas tablas en otras tablas de “menor tamaño”. El problema es que al hacer esto, en este caso, aumenta notablemente el número de tablas del modelo a implementar. Esto influye gravemente en el rendimiento de las futuras consultas u otras operaciones sobre la base de datos, ya que se deberán realizar diversas operaciones de JOIN para poder llevar a cabo operaciones relativamente simples que nos permitan obtener la información que se busca.

Tener muchas tablas es sinónimo de mayor complejidad y de menor eficiencia, y en este caso, no alcanzar la quinta forma normal y, por tanto, la reducción del número de tablas, se considera que es más beneficiosa en términos de rendimiento de lo que lo será alcanzar 5FN en términos de normalización de información.

Por ello, se decide deliberadamente desnormalizar la quinta forma normal, de forma que se mejore el rendimiento, acceso y análisis de información de la base de datos.

Además, se desnormalizará la separación llevada a cabo en la tabla ubicación (separación en tablas UBICACION y PAIS) para alcanzar la tercera forma normal. Esto se hace con el objetivo de simplificar el futuro diseño de la base de datos.

Modelo final

Una vez hemos tenido en cuenta la normalización, desnormalización u otras consideraciones anteriores, se puede presentar el modelo final, pasado a relacional y posteriormente normalizado hasta donde se ha creído conveniente.

El modelo es el siguiente:

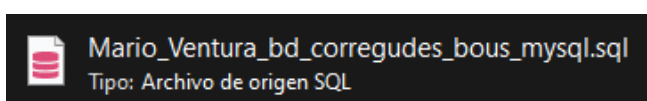
- PERSONA(**ideper**, ideubi, nompe, co1pe, co2pe, maiper, dirper)
- TORERO(**idetor**, ideapo)
- APODERAT(**ideapo**)
- ACTUACION(**ideact**, idetor, nompla, datcor)
- PLAZA(**nompla**, ideubi, anypla, locpla, tippla, estpla, muspla)
- UBICACION(**ideubi**, nompai, nomciu)
- ESDEVENIMENT(**ideesd**, fircor, nompla, datcor)
- R_ESDEVENIMENT_TORO(**ideesd**, **idtoro**)
- TORO(**idtoro**, cifram, nombou, anybou, pesbou)
- RAMADERIA(**cifram**, nomram)
- FERIA(**fircor**)

MySQL

Para las tareas relacionadas con el gestor de bases de datos MySQL se ha optado por usar el sistema operativo Windows 11 Home, ya que es el sistema operativo con el que se cuenta en el equipo que se usará, y de momento no se ve la necesidad de hacer una partición e instalar otro sistema operativo en ella. Aun así, esta posibilidad seguirá abierta de cara a un futuro.

Creación de un fichero con la implementación del modelo

Se genera un archivo .sql con el código de creación de la base de datos (tablas) que ilustra la estructura de datos presentada en el apartado anterior (modelo final). El fichero es el siguiente:



Que contiene el siguiente código:

```
-- TABLA DE APODERAT
CREATE TABLE APODERAT (
    ideapo VARCHAR(10), -- Identificador del apoderado (màxim 10 dígits)
    CONSTRAINT PK_APODERAT PRIMARY KEY (ideapo)
);

-- TABLA DE UBICACION
CREATE TABLE UBICACION (
    ideubi INT(11) AUTO_INCREMENT, -- Identificador de la ubicación
    nompai VARCHAR(40) NOT NULL, -- Nombre del país
    nomciu VARCHAR(50) NOT NULL, -- Nombre de la ciutat
    CONSTRAINT PK_UBICACION PRIMARY KEY (ideubi)
);

-- TABLA DE RAMADERIA
CREATE TABLE RAMADERIA (
    cifram VARCHAR(3), -- CIF de la ramaderia
    nomram VARCHAR(50) NOT NULL, -- Nombre de la ramaderia
    CONSTRAINT PK_RAMADERIA PRIMARY KEY (cifram)
);

-- TABLA DE FERIA
CREATE TABLE FERIA (
    fircor VARCHAR(50), -- Nombre de la festa o feria
    CONSTRAINT PK_FERIA PRIMARY KEY (fircor)
);

-- TABLA DE PERSONA
CREATE TABLE PERSONA (
    ideper VARCHAR(10), -- Identificador de la persona
```

```

ideubi INT(11) NOT NULL,      -- Clave forana que referencia a UBICACION(ideubi)
nompe VARCHAR(50) NOT NULL,   -- Nombre de La persona
co1pe VARCHAR(50) NOT NULL,   -- Primer apellido de la persona
co2pe VARCHAR(50),           -- Segundo apellido de la persona
maiper VARCHAR(100),          -- Mail de La persona
dirper VARCHAR(100) NOT NULL, -- Dirección de La persona
CONSTRAINT PK_PERSONA PRIMARY KEY (ideper),
CONSTRAINT FK_PERSONA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi)
);

-- TABLA DE PLAZA
CREATE TABLE PLAZA (
    nompla VARCHAR(50), -- Nombre de La plaza
    ideubi INT(11) NOT NULL, -- Clave forana que referencia a UBICACION(ideubi)
    anypla DATE NOT NULL, -- Año de construcción de la plaza
    locpla INT NOT NULL, -- Número de asientos de la plaza
    tippla BOOLEAN NULL, -- Si la plaza es fija o móvil
    estpla TEXT, -- Estilos predominantes en la construcción
    muspla BOOLEAN NULL, -- Indica si la plaza contiene un museo
    CONSTRAINT PK_PLAZA PRIMARY KEY (nompla),
    CONSTRAINT FK_PLAZA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi),
    CONSTRAINT CHECK_PLAZA_MUSPLA CHECK (muspla IN (0, 1)),
    CONSTRAINT CHECK_PLAZA_TIPPLA CHECK (tippla IN (0, 1))
);

-- TABLA DE TORERO
CREATE TABLE TORERO (
    idetor VARCHAR(10), -- Identificador del torero (máximo 10 dígitos)
    ideapo VARCHAR(10) NOT NULL, -- Clave forana que referencia a APODERAT(ideapo)
    CONSTRAINT PK_TORERO PRIMARY KEY (idetor),
    CONSTRAINT FK_TORERO_APODERAT FOREIGN KEY (ideapo) REFERENCES APODERAT(ideapo)
);

-- TABLA DE ACTUACION
CREATE TABLE ACTUACION (
    ideact INT(11) AUTO_INCREMENT, -- Identificador de la actuación
    idetor VARCHAR(10) NOT NULL, -- Clave forana que referencia a TORERO(idetor)
    nompla VARCHAR(50) NOT NULL, -- Clave forana que referencia a PLAZA(nompla)
    datcor DATE NOT NULL, -- Fecha de La actuación
    CONSTRAINT PK_ACTUACION PRIMARY KEY (ideact),
    CONSTRAINT FK_ACTUACION_TORERO FOREIGN KEY (idetor) REFERENCES TORERO(idetor),
    CONSTRAINT FK_ACTUACION_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
);

-- TABLA DE TORO
CREATE TABLE TORO (
    idtoro INT(11) AUTO_INCREMENT, -- Identificador del toro (autoincremental)
    cifram VARCHAR(3) NOT NULL, -- Clave forana que referencia a RAMADERIA(cifram)
    nombou VARCHAR(50) NOT NULL, -- Nombre o identificación del toro

```

```

anybou DATE NOT NULL,          -- Año de nacimiento del toro
pesbou DECIMAL(10,2) NOT NULL, -- Peso del toro
CONSTRAINT PK_TORO PRIMARY KEY (idtoro),
CONSTRAINT FK_TORO_RAMADERIA FOREIGN KEY (cifram) REFERENCES RAMADERIA(cifram)
);

CREATE TABLE R_ESDEVENIMENT_TORO (
    idtoro INT(11) NOT NULL,    -- id del toro que hace referencia a la tabla TORO
    ideesd INT(11) NOT NULL,    -- id del esdeveniment que hace referencia a la tabla
    ESDEVENIMENT
    CONSTRAINT PK_ESD_TORO PRIMARY KEY (idtoro, ideesd),
    CONSTRAINT FK_TORO FOREIGN KEY (idtoro) REFERENCES TORO (idtoro),
    CONSTRAINT FK_ESDEVENIMENT FOREIGN KEY (ideesd) REFERENCES ESDEVENIMENT (ideesd)
);

-- TABLA DE ESDEVENIMENT
CREATE TABLE ESDEVENIMENT (
    ideesd INT(11) AUTO_INCREMENT, -- Identificador del esdeveniment
    fircor VARCHAR(50) NOT NULL,    -- Clave forana que referencia a FERIA(fircor)
    nompla VARCHAR(50) NOT NULL,    -- Clave forana que referencia a PLAZA(nompla)
    datcor DATE NOT NULL,          -- Fecha del esdeveniment
    CONSTRAINT PK_ESDEVENIMENT PRIMARY KEY (ideesd),
    CONSTRAINT FK_ESDEVENIMENT_FERIA FOREIGN KEY (fircor) REFERENCES FERIA(fircor),
    CONSTRAINT FK_ESDEVENIMENT_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
);

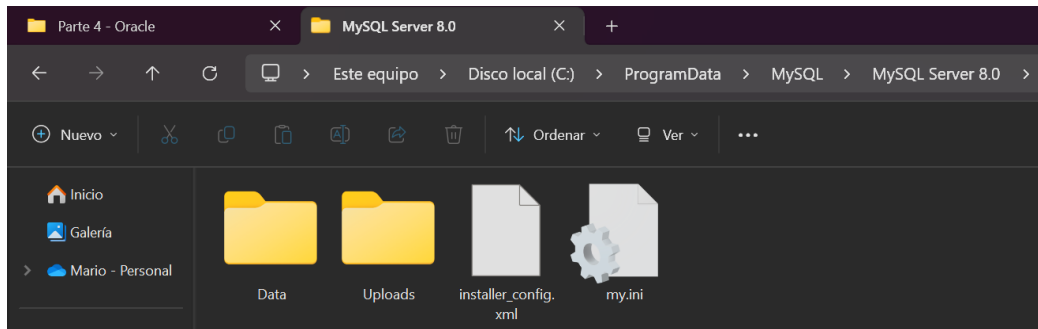
```

Creación de un tablespace de 5GB y cota máxima 10GB

Para crear un tablespace de tamaño inicial 5Gb y tamaño máximo de 10Gb, se debe tener en cuenta que, cuando se usa el motor InnoDB (motor más comúnmente usado ya que NDB no recibe soporte en las versiones 8 o 5.7 de MySQL), la gestión de la memoria, el tamaño y el crecimiento del tablespace es controlado por parte del gestor. Esto se hace en base a parámetros como **innodb_data_file_path** o **innodb_autoextend_increment**, de forma que cada tablespace puede entenderse como un archivo físico en el sistema de archivos, que es controlado por el gestor. No hay ninguna configuración específica o explícita en InnoDB para especificar el tamaño inicial del tablespace, y por eso se trabaja con Datafiles.

Para crear un tablespace con las características especificadas, el planteamiento a seguir deberá combinar sintaxis y configuración, ambas detalladas en la documentación que ofrece MySQL acerca de tablespaces [\[1\]](#).

Se configurará internamente **innodb_data_file_path**. Para ello, es necesario acceder y modificar el archivo de configuración del servidor, que generalmente se llama **my.cnf** o **my.ini** dependiendo de la versión específica de MySQL. En este caso, existe un archivo my.ini en la ruta típica del sistema en la que se suele encontrar este archivo por defecto, que es **C:\ProgramData\MySQL\MySQL Server 8.0**. Cabe mencionar que este directorio suele estar oculto al usuario de forma habitual, por lo que será necesario habilitar la opción de Windows para mostrar objetos ocultos.



Una vez encontrado el archivo `my.ini`, lo abriremos con un editor de texto como el bloc de notas del propio sistema operativo o Notepad. Para abrir el archivo, también será necesario permisos de administrador, de forma que los concederemos haciendo clic derecho y ejecutar como administrador, o simplemente pulsando “Aceptar” al hacer doble click de forma normal en el archivo. Una vez abierto el archivo buscaremos la sección `[mysqld]`, y si no existe, la crearemos nosotros mismos al final del archivo. En este caso sí que se ha encontrado dicha sección.

Ahora se modificará (o agregará) la línea `innodb_data_file_path` para que coincida con la configuración establecida por el enunciado. Para ello, se añadirá la siguiente sentencia de configuración: **`innodb_data_file_path=ibdata1:5G:autoextend:max:10G`**

Donde `ibdata1` es el nombre predeterminado del primer archivo de datos InnoDB, 5G hace referencia a los 5 GB de tamaño inicial del archivo de datos del motor InnoDB. `autoextend` indica que el archivo de datos se puede extender automáticamente según sea necesario, y por último, 10G hace referencia a los 10GB de tamaño máximo que el archivo de datos puede tener, haciendo que cuando se alcance este límite ya no se pueda seguir extendiendo el tamaño automáticamente.

```
# config para tam inicial 5G y tam maximo 10G
innodb_data_file_path=ibdata1:5G:autoextend:max:10G
```

Hecho esto, se reinicia el servicio de MySQL para aplicar los cambios, y se creará el tablespace con el siguiente código:

```
-- Creación del tablespace
CREATE TABLESPACE my_tablespace
INITIAL_SIZE=5G
MAX_SIZE=10G;
```

```
MySQL 8.0 Command Line Cli X + v
Enter password: ****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 11
Server version: 8.0.35 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> -- Creación del tablespace
mysql> CREATE TABLESPACE my_tablespace
      -> INITIAL_SIZE=5G
      -> MAX_SIZE=10G;
Query OK, 0 rows affected (0.04 sec)

mysql>
```

El tablespace se ha creado correctamente.

Creación de una base de datos IMPBD para la importación

Para crear una base de datos de nombre IMPBD haremos uso del comando `CREATE DATABASE` [2]. Se debe establecer el conjunto de caracteres que se usarán en la base de datos, y por ello, se usarán las cláusulas `CHARACTER SET` y `COLLATE` [3].

Se puede comprobar las posibilidades que ofrece el SGBD mediante el siguiente comando:

```
-- Posibles character sets y collations
SHOW CHARACTER SET;
```

Al ejecutar este comando, se puede observar una amplia lista de posibilidades.

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
ascii	US ASCII	ascii_general_ci	1
big5	Big5 Traditional Chinese	big5_chinese_ci	2
binary	Binary pseudo charset	binary	1
cp1250	Windows Central European	cp1250_general_ci	1
cp1251	Windows Cyrillic	cp1251_general_ci	1
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
cp850	DOS West European	cp850_general_ci	1
cp852	DOS Central European	cp852_general_ci	1
cp866	DOS Russian	cp866_general_ci	1
cp932	SJIS for Windows Japanese	cp932_japanese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
eucjpms	UJIS for Windows Japanese	eucjpms_japanese_ci	3
euckr	EUC-KR Korean	euckr_korean_ci	2
gb18030	China National Standard GB18030	gb18030_chinese_ci	4

En este caso, se barajan dos opciones: El CHARACTER SET utf8mb4 y latin1. Dado que nos centramos en el alfabeto español de España, que es el que se trata en los archivos ods y csv ambos conjuntos de caracteres serían suficientes para representar los símbolos necesarios. Si buscamos un mayor ahorro de espacio y solo necesitamos soportar el español, el conjunto de caracteres latin1 podría ser una elección más eficiente en términos de almacenamiento. Sin embargo, utf8 es más ampliamente utilizado y puede ofrecer más flexibilidad si en el futuro si se decide expandir la cobertura de idiomas.

El número máximo de bytes requerido para representar un carácter del conjunto de caracteres latin1 es 1, mientras que para utf8 es 4 (columna maxlen [\[4\]](#)).

Dada la flexibilidad que ofrece utf8mb4, y teniendo en cuenta posibles expansiones, se ha optado por utilizar el conjunto de caracteres (CHARACTER SET) **utf8mb4**, con COLLATE= **utf8mb4_0900_ai_ci** ya que esto nos permite representar todos los caracteres que necesitamos con mayor flexibilidad y adaptabilidad que latin1.

```
-- Creación BD
CREATE DATABASE IMPBD
CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;
```

```
mysql> -- Creación BD
mysql> CREATE DATABASE IMPBD
-> CHARACTER SET utf8mb4
-> COLLATE utf8mb4_0900_ai_ci;
Query OK, 1 row affected (0.01 sec)
```

Tras la creación de la base de datos, podemos comprobar si se ha creado correctamente ejecutando el comando siguiente:

```
-- Mostrar BDs disponibles
SHOW DATABASES;
```

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| information_schema |
| impbd |
| mysql |
| performance_schema |
| psswr |
| sys |
+-----+
6 rows in set (0.00 sec)
```

Por último, debemos asegurarnos de que esta base de datos usa el tablespace creado en el paso anterior. Dado que la asignación al tablespace se hace de forma individual para cada una de las tablas

de la base de datos, seleccionaremos nuestra base de datos con el comando **USE IMPBD** y, posteriormente, añadiremos al CREATE TABLE de cada una de estas tablas la sentencia "TABLESPACE 'my_tablespace'". De esta forma seleccionaremos la base de datos y para cada una de sus tablas se le asignará el tablespace asignado.

Creación de un usuario IMPBD con privilegios

La creación del usuario IMPBD se hará con el comando CREATE USER, que solo se conectará desde el mismo servidor donde se está usando MySQL (parámetro localhost) y se identificará bajo la contraseña "impbd_user_passwd".

```
-- Usuario Impbd
CREATE USER 'IMPBD'@'localhost' IDENTIFIED BY 'impbd_user_passwd';
```

Ahora debemos otorgar permisos mediante "GRANT" al usuario creado. Los permisos que se le quieren asignar a este usuario y su respectiva correspondencia en MySQL son:

- Crear objetos → CREATE
- Insertar filas en tablas → INSERT
- Actualizar filas en tablas → UPDATE
- Borrar filas en tablas → DELETE

Los comandos usados para otorgar estos privilegios al usuario IMPBD son:

```
-- Permisos
GRANT CREATE, INSERT, UPDATE, DELETE ON IMPBD.* TO 'IMPBD'@'localhost';
```

Por último, comprobaremos si estos permisos se han otorgado correctamente al usuario mediante la ejecución del comando **SHOW GRANTS** para el usuario IMPBD. Ejecutamos el comando y obtenemos lo siguiente:

```
mysql> -- Comprobar los permisos del usuario
mysql> SHOW GRANTS FOR 'IMPBD'@'localhost';
+-----+
| Grants for IMPBD@localhost |
+-----+
| GRANT USAGE ON *.* TO `IMPBD`@`localhost` |
| GRANT INSERT, UPDATE, DELETE, CREATE ON `impbd`.* TO `IMPBD`@`localhost` |
+-----+
2 rows in set (0.00 sec)
```

Conexión con el usuario IMPBD y crear las tablas

En primer lugar, para conectarnos a la base de datos con el usuario IMPBD, abriremos la línea de comandos Windows (no la consola de MySQL donde hemos estado trabajando hasta ahora), y navegaremos mediante el comando **cd** hasta la ruta 'C:\Program Files\MySQL\MySQL Server 8.0\bin'.

Cuando nos encontremos en esta ruta, haremos uso del comando **mysql -u IMPBD -p** para realizar la conexión con el usuario IMPBD. Se nos pedirá la contraseña, que es "impbd_user_passwd". Tras introducirla realizaremos finalmente la conexión a la base de datos.

```
C:\Windows\System32>cd ..
C:\Windows>cd ..
C:\>cd Program Files
C:\Program Files>cd MySQL
C:\Program Files\MySQL>cd MySQL Command Line 8.0
El sistema no puede encontrar la ruta especificada.
C:\Program Files\MySQL>cd MySQL Server 8.0
C:\Program Files\MySQL\MySQL Server 8.0>cd bin
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u IMPBD -p
Enter password: *****
```

(Navegamos por el sistema hasta el directorio bin de MySQL Server 8.0)

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u IMPBD -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 34
Server version: 8.0.35 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> USE IMPBD;
Database changed
mysql> |
```

(seleccionamos la base de datos IMPBD con el comando USE)

A continuación, una vez nos conectamos con el usuario IMPBD y seleccionamos la base de datos, crearemos unas tablas necesarias para poder exportar los datos que tenemos en los ficheros ods y csv. Estas tablas son temporales, no son el esquema de tablas del modelo definitivo.

El código usado para la creación de las tablas es el siguiente:

```
-- TABLA DE APODERADO
CREATE TABLE APODERAT(
    -- No ponemos ID porque en el modelo original no hay ID y se pondria en un poste-
    -- rior paso a relacional
    IDEAPO VARCHAR(10),
    NOMAPO VARCHAR(50),
    CO1APO VARCHAR(50),
```

```

CO2APO VARCHAR(50),
MAIAPO VARCHAR(100),
DIRAPO VARCHAR(100),
CIUAPO VARCHAR(50),
PAIAPO VARCHAR(40)) TABLESPACE my_tablespace;

-- TABLA DE PLAZA
CREATE TABLE PLAZA(
    NOMPLA VARCHAR(50),
    ANYPLA DATE,
    LOCPLA INT,
    TIPPLA BOOLEAN,
    CIUPLA VARCHAR(50),
    PAIPLA VARCHAR(50),
    ESTPLA TEXT NULL,
    MUSPLA BOOLEAN) TABLESPACE my_tablespace;

-- TABLA DE TORERO
CREATE TABLE TORERO(
    IDETOR VARCHAR(10),
    NOMTOR VARCHAR(50),
    CO1TOR VARCHAR(50),
    CO2TOR VARCHAR(50),
    MAITOR VARCHAR(100),
    DIRTOR VARCHAR(100),
    CIUTOR VARCHAR(50),
    PAITOR VARCHAR(40),
    IDEAPO VARCHAR(10)) TABLESPACE my_tablespace;

-- TABLA DE ACTUACION
CREATE TABLE ACTUACIO(
    IDETOR VARCHAR(10),
    DATCOR DATE,
    NOMPLA VARCHAR(50)) TABLESPACE my_tablespace;

-- TABLA DE ESDEVENIMENT
CREATE TABLE ESDEVENIMENT(
    FIRCOR VARCHAR(50),
    DATCOR DATE,
    NOMPLA VARCHAR(50),
    NOMBOU VARCHAR(50),
    ANYBOU DATE,
    PESBOU DECIMAL(10,2),
    CIFRAM VARCHAR(3),
    NOMRAM VARCHAR(50)) TABLESPACE my_tablespace;

```

```
mysql> -- TABLA DE ESDEVENIMENT
mysql> CREATE TABLE ESDEVENIMENT(
  -> -- No ponemos ID porque en el modelo original no hay ID y se pondria en un posterior paso a relacional
  -> FIRCOR VARCHAR(50),
  -> DATCOR DATE,
  -> NOMPLA VARCHAR(50),
  -> NOMBOU VARCHAR(50),
  -> ANYBOU DATE,
  -> PESBOU DECIMAL(10,2),
  -> CIFRAM VARCHAR(3),
  -> NOMRAM VARCHAR(50)) TABLESPACE my_tablespace;
Query OK, 0 rows affected (0.05 sec)
```

(Ejemplo de creación de una de las tablas)

Importación de los archivos en las tablas creadas

La importación de datos en sus respectivas tablas de la base de datos es ciertamente compleja. Para poder insertar los datos, tenemos que poder escribirlos en el formato correcto de una sentencia INSERT en MySQL, pero estos datos vienen en una serie de archivos en formato .csv y formato .ods. Además, la gran cantidad de datos que estos archivos contienen dificulta la escritura manual de los inserts.

Por ello, se ha decidido implementar **cinco scripts** en el lenguaje **Python** (uno para cada tabla) que sean capaces de leer los archivos, conectarse a la base de datos, e insertar los datos en sus respectivas tablas. Se han implementado dos porque de esta forma tenemos un script que nos sirve para los archivos en formato csv y uno para los archivos en formato ods. Todo esto se hace con el fin de agilizar y automatizar un proceso que tomaría demasiado tiempo si se hiciese de forma manual.

Antes de ejecutar los scripts deberemos modificar el conjunto de caracteres del cliente y la conexión, para asegurar que ambos pueden ver los datos en utf8mb4. Podemos comprobar qué valor tienen por defecto mediante los siguientes comandos:

```
-- Antes de ejecutar scripts python
-- USE IMPBD -- si no estamos conectados aún
STATUS;
```

```
mysql> STATUS;
-----
C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe  Ver 8.0.36 for Win64 on x86_64 (MySQL Community Server - GPL)

Connection id:          19
Current database:       impbd
Current user:           root@localhost
SSL:                    Cipher in use is TLS_AES_256_GCM_SHA384
Using delimiter:        ;
Server version:         8.0.36 MySQL Community Server - GPL
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:     utf8mb4
Db characterset:        utf8mb4
Client characterset:     cp850
Conn. characterset:     cp850
TCP port:               3306
Binary data as:         Hexadecimal
Uptime:                 6 hours 21 min 8 sec

Threads: 2  Questions: 246  Slow queries: 5  Opens: 401  Flush tables: 3  Open tables: 317  Queries per second avg: 0.010
-----
```

Como vemos, salen el CHARACTERSET cp850. Mediante una comprobación, podemos ver que este pertenece a un conjunto de caracteres del oeste europeo.

```
-- Comprobamos qué character set es cp850
SHOW CHARACTER SET;
```

```
mysql> SHOW CHARACTER SET;
```

Charset	Description	Default collation	Maxlen
armscii8	ARMSCII-8 Armenian	armscii8_general_ci	1
ascii	US ASCII	ascii_general_ci	1
big5	Big5 Traditional Chinese	big5_chinese_ci	2
binary	Binary pseudo charset	binary	1
cp1250	Windows Central European	cp1250_general_ci	1
cp1251	Windows Cyrillic	cp1251_general_ci	1
cp1256	Windows Arabic	cp1256_general_ci	1
cp1257	Windows Baltic	cp1257_general_ci	1
cp850	DOS West European	cp850_general_ci	1
cp852	DOS Central European	cp852_general_ci	1

Dado que el conjunto de caracteres por defecto es cp850 y queremos modificarlo a utf8mb4, en este caso, optaremos por cambiar el conjunto de caracteres solamente para la sesión actual. Esto se hará mediante el siguiente código:

```
-- Cambiamos character set
SET NAMES 'utf8mb4';
SET CHARACTER SET 'utf8mb4';
```

```
mysql> SET NAMES 'utf8mb4';
Query OK, 0 rows affected (0.00 sec)

mysql> SET CHARACTER SET 'utf8mb4';
Query OK, 0 rows affected (0.00 sec)
```

Ahora, si ejecutamos nuevamente el comando *STATUS* podremos ver que hay cambios.

```
mysql> STATUS;
-----
C:\Program Files\MySQL\MySQL Server 8.0\bin\mysql.exe Ver 8.0.36 for Win64 on x86_64 (MySQL Community Server - GPL)

Connection id:          19
Current database:       impdb
Current user:            root@localhost
SSL:                     Cipher in use is TLS_AES_256_GCM_SHA384
Using delimiter:        ;
Server version:         8.0.36 MySQL Community Server - GPL
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:     utf8mb4
Db characterset:         utf8mb4
Client characterset:     utf8mb4
Conn. characterset:      utf8mb4
TCP port:                3306
Binary data as:         Hexadecimal
Uptime:                  6 hours 36 min 9 sec

Threads: 2  Questions: 253  Slow queries: 5  Opens: 401  Flush tables: 3  Open tables: 317  Queries per second avg: 0.010
-----
```

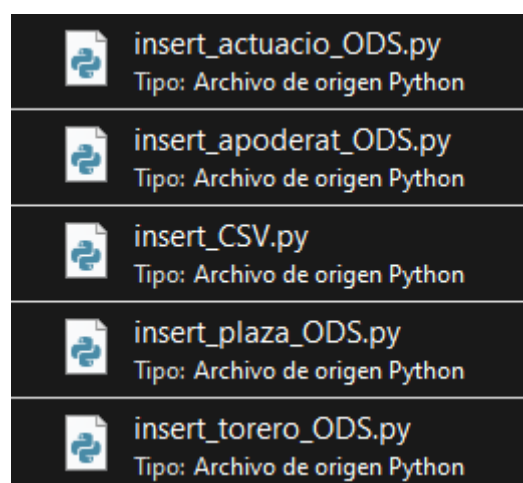
Paralelamente, también podemos realizar estas comprobaciones mediante el siguiente código:

```
SHOW VARIABLES LIKE 'character_set_client';  
SHOW VARIABLES LIKE 'character_set_connection';
```

```
mysql> SHOW VARIABLES LIKE 'character_set_client';  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| character_set_client | utf8mb4 |  
+-----+-----+  
1 row in set, 1 warning (0.09 sec)  
  
mysql> SHOW VARIABLES LIKE 'character_set_connection';  
+-----+-----+  
| Variable_name | Value |  
+-----+-----+  
| character_set_connection | utf8mb4 |  
+-----+-----+  
1 row in set, 1 warning (0.00 sec)
```

Los scripts se adjuntarán con este documento, y cuentan con prints de debugging que muestran lo que va sucediendo en cada momento de la ejecución del script. Si se quiere entender en detalle el código implementado basta con prestar atención a los comentarios que describen cada instrucción o bloque de instrucciones. El funcionamiento general es simple, pero los comentarios permitirán profundizar más en la razón y objetivo de cada una de las acciones del script.

Cabe mencionar que, debido a la gran cantidad de filas que se deben insertar en la tabla esdeveniment (fichero csv), se considera que hacerlos todos “seguidos” o juntos puede ser ineficiente ya que si por ejemplo el script lleva el 80% de los inserts realizados (+8.000.000 de filas) y hay un fallo del sistema, al no haber consolidado estos valores, se podrían perder todos los inserts realizados y esto resultaría ineficiente teniendo en cuenta que se prevé que realizar todas estas inserciones puede llevar horas. Por ello, se harán inserts por lotes, cuyo tamaño viene determinado por el valor de la variable **tamano_lote** (el valor por defecto es 250.000 inserciones).



A continuación, se mostrará el código de los scripts. De los scripts que tratan los archivos ODS se mostrará solo el código de uno de ellos (insert_torero_ODS.py), ya que son todos iguales con la única diferencia del nombre de la tabla sobre la que se trabaja. Se ha optado por tener varios scripts para ficheros ODS ya que, de esta manera, no hace falta modificar el código del script cada vez que se trabaja en una tabla diferente.

Código de los scripts ODS

```
# =====
# Mario Ventura Burgos - 43223476J
# Sistemas de gestión de bases de datos - proyecto final
# Script de tratamiento de fichero ODS para inserción en base de datos
# =====
# ===== TABLA SOBRE LA QUE SE TRABAJARÁ ---> TORERO
# =====

# Importar las bibliotecas necesarias
from pandas_ods_reader import read_ods
import mysql.connector
from mysql.connector import errorcode
from unicode import unicode

# Función para generar la sentencia INSERT en SQL
def generate_insert(nameDB, path):
    # Leer el archivo ODS
    df = read_ods(path)
    df = df.loc[:, ~df.columns.str.contains('^unnamed')]
    # Obtener los nombres de todas las columnas
    nombres_columnas = df.columns.tolist()
    print(nombres_columnas)
    # Construir la sentencia INSERT en SQL
    sql = f"INSERT INTO {nameDB} ({', '.join(nombres_columnas)}) VALUES " + "\n"
    # Reemplazar comillas simples en el DataFrame
    df = df.replace("'", "\'", regex=True)
    # Iterar sobre las filas del DataFrame
    for index, row in df.iterrows():
        insert = "("
        # Iterar sobre las columnas
        for column in nombres_columnas:
            value = row[column]
            # Verificar el tipo del valor y construir la parte de la sentencia INSERT
            if value is None or value == " ":
                insert += f"NULL",'
            elif isinstance(value, str):
                insert += f"'{value}',"
            else:
                insert += f"{int(value)},"
```

```

        # Eliminar la coma final y agregar el paréntesis de cierre
        insert = insert.rstrip(',') + ")"
        sql += insert + ',' + '\n' if index < len(df) - 1 else insert + '\n'
        # Eliminar la coma final y agregar el punto y coma al final de la sentencia INSERT
        sql = sql.rstrip(',') + ";"
        # Escribir la sentencia SQL en un archivo o realizar alguna operación
        writeSQL(sql)
# Función para escribir la sentencia SQL y ejecutarla
def writeSQL(sql):
    global cursor, cnx
    try:
        # Conectar a la base de datos MySQL
        cnx = mysql.connector.connect(user='IMPBD', password='impbd_user_passwd',
host='localhost', database='IMPBD')
        cursor = cnx.cursor()
        # Ejecutar la sentencia SQL
        cursor.execute(sql)
        cnx.commit()
    except mysql.connector.Error as err:
        # Manejar los errores de conexión o ejecución de SQL
        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Algo está mal con tu nombre de usuario o contraseña")
        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("La base de datos no existe")
        else:
            print(err)
    finally:
        # Imprimir un mensaje de éxito y cerrar la conexión y el cursor
        print("Operación completada con éxito")
        cursor.close()
        cnx.close()

# Ruta del archivo ODS
path = "TOREROS.ods"
table_name = "TORERO"
# Llamar a la función para generar la sentencia INSERT y escribir en la base de datos
generate_insert(table_name, path)

```

```

C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\1 SEMESTRE\SGBD\Practica Final Recu\Ficheros\Datos>python insert_plaza_ODS.py
['NOMPLA', 'ANYPLA', 'LOCPLA', 'TIPPLA', 'CIUPLA', 'PAIPLA', 'ESTPLA', 'MUSPLA']
EXITO

C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\1 SEMESTRE\SGBD\Practica Final Recu\Ficheros\Datos>python insert_torero_ODS.py
['IDETOR', 'NOMTOR', 'CO1TOR', 'CO2TOR', 'MAITOR', 'DIRTOR', 'CIUTOR', 'PAITOR', 'IDEAPO']
EXITO

C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\1 SEMESTRE\SGBD\Practica Final Recu\Ficheros\Datos>python insert_apoderat_ODS.py
['IDEAPO', 'NOMAPO', 'CO1APO', 'CO2APO', 'MAIAPO', 'DIRAPO', 'CIUAPO', 'PAIAPO']
EXITO

C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\1 SEMESTRE\SGBD\Practica Final Recu\Ficheros\Datos>python insert_actuacio_ODS.py
['IDETOR', 'DATCOR', 'NOMPLA']
EXITO

```


Código del script que trata el fichero CSV

```
# =====
# Mario Ventura Burgos - 43223476J
# Sistemas de gestión de bases de datos - proyecto final
# Script de tratamiento de fichero CSV para inserción en base de datos
# =====
# ===== TABLA SOBRE LA QUE SE TRABAJARÁ ---> ESDEVENIMENT
# =====

import time
import pandas as pd
import mysql.connector
from mysql.connector import errorcode

# Función para generar la sentencia INSERT en SQL con carga por lotes
def generate_insert(nameDB, path):
    tamaño_lote = 250000 # Tamaño del lote de inserción
    contador = 0

    # Iterar sobre el archivo CSV en lotes
    for chunk in pd.read_csv(path, chunksize=tamaño_lote):
        # Obtener los nombres de todas las columnas
        nombres_columnas = chunk.columns.tolist()
        chunk = chunk.replace("'", "\\'", regex=True)
        # Construir la sentencia INSERT en SQL
        sql = f"INSERT INTO {nameDB} ({','.join(nombres_columnas)}) VALUES " + "\n"
        count = 0 # Contador de filas procesadas en el lote
        # Iterar sobre las filas del lote
        for index, row in chunk.iterrows():
            insert = "("
            # Iterar sobre las columnas
            for column in nombres_columnas:
                value = row[column]
                # Verificar el tipo del valor y construir la parte de la sentencia IN-
                if pd.isna(value) or value == " ":
                    insert += "NULL,"
                elif isinstance(value, str):
                    insert += f"'{value}',"
                else:
                    insert += f"{int(value)},"
            # Eliminar la coma final y agregar el paréntesis de cierre
            count += 1
            insert = insert.rstrip(',') + ")"
            sql += insert + ', ' + '\n' if count < len(chunk) else insert + '\n'

SERT
```

```

# Eliminar la coma final y agregar el punto y coma al final de la sentencia
INSERT

sql = sql.rstrip(',') + ";"
# Escribir la sentencia SQL y realizar operaciones adicionales
writeSQL(sql)
#print(sql)
time.sleep(5) # Simular una pausa de 5 segundos entre lotes
contador+=1
print("LOTE NUMERO ",contador, " INSERTADO. TOTAL APROXIMADO INSERTADO: ",
contador*tamano_lote)

# Función para escribir la sentencia SQL y ejecutarla
def writeSQL(sql):
    global cursor, cnx
    try:
        # Conectar a la base de datos MySQL
        cnx = mysql.connector.connect(user='IMPBD', password='impbd_user_passwd',
host='localhost', database='IMPBD')
        cursor = cnx.cursor()
        # Ejecutar la sentencia SQL
        cursor.execute(sql)
        cnx.commit()
    except mysql.connector.Error as err:
        # Manejar los errores de conexión o ejecución de SQL
        if err.errno == errorcode.ER_ACCESS_DENIED_ERROR:
            print("Algo está mal con tu nombre de usuario o contraseña")
        elif err.errno == errorcode.ER_BAD_DB_ERROR:
            print("La base de datos no existe")
        else:
            print(err)
    finally:
        # Imprimir un mensaje de éxito y cerrar la conexión y el cursor
        print("Operación completada con éxito")
        cursor.close()
        cnx.close()

# Ruta del archivo CSV
path = "esdeveniments.csv"
table_name = "esdeveniment"

# Llamar a la función para generar la sentencia INSERT y escribir en la base de datos
generate_insert(table_name, path)

```

Si ejecutamos el script, podemos ver prints de debugging como los mostrados a continuación.

```
C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\
EXITO
LOTE NUMERO 1 INSERTADO. TOTAL APROXIMADO INSERTADO: 250000
EXITO
LOTE NUMERO 2 INSERTADO. TOTAL APROXIMADO INSERTADO: 500000
EXITO
LOTE NUMERO 3 INSERTADO. TOTAL APROXIMADO INSERTADO: 750000
```

...

```
EXITO
LOTE NUMERO 39 INSERTADO. TOTAL APROXIMADO INSERTADO: 9750000
EXITO
LOTE NUMERO 40 INSERTADO. TOTAL APROXIMADO INSERTADO: 10000000
EXITO
LOTE NUMERO 41 INSERTADO. TOTAL APROXIMADO INSERTADO: 10250000
C:\Users\mvent\OneDrive\Escritorio\Mario\Universidad\4 CARRERA\1 SEM
```

Como podemos ver, se han realizado un total aproximado de 10.000.000 inserciones mediante el script. Realizando una consulta del tipo `'SELECT COUNT(*) FROM esdeveniment;'` podemos saber la cantidad exacta de filas insertadas.

El resto de tablas cuentan con la siguiente cantidad de filas:

```
mysql> SELECT COUNT(*) FROM PLAZA;
+-----+
| COUNT(*) |
+-----+
|      266 |
+-----+
1 row in set (0.00 sec)

mysql> SELECT COUNT(*) FROM ACTUACIO;
+-----+
| COUNT(*) |
+-----+
|   535990 |
+-----+
1 row in set (0.12 sec)
```

```
mysql> SELECT COUNT(*) FROM torero;
+-----+
| COUNT(*) |
+-----+
|   106640 |
+-----+
1 row in set (0.07 sec)

mysql> SELECT COUNT(*) FROM apoderat;
+-----+
| COUNT(*) |
+-----+
|   132189 |
+-----+
1 row in set (0.09 sec)
```

Creación de una segunda base de datos DEFBD

La creación de esta segunda base de datos se hará mediante la sentencia `CREATE DATABASE`. Teniendo en cuenta el objetivo de esta base de datos (importar de IMPBD a DEFBD), se creará también con el mismo `CHARACTER SET` y `COLLATION` que la base de datos IMPBD, que es `utf8mb4`. Esto se hará para mantener consistencia en el conjunto de caracteres usados y para prevenir posibles errores.

```
-- Creación de la base de datos DEFBD;
CREATE DATABASE DEFBD
CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;
```

```
mysql> CREATE DATABASE DEFBD
-> CHARACTER SET utf8mb4
-> COLLATE utf8mb4_0900_ai_ci;
Query OK, 1 row affected (0.04 sec)
```

Posteriormente comprobaremos si se ha creado correctamente mediante la sentencia SHOW DATABASES, cuyo resultado nos mostrará las bases de datos existentes.

```
-- Mostrar BDs disponibles
SHOW DATABASES;
```

```
mysql> SHOW DATABASES;
+-----+
| Database |
+-----+
| defbd    |
| impbd    |
| information_schema |
| mysql    |
| performance_schema |
| sys      |
+-----+
6 rows in set (0.01 sec)
```

Se puede ver como la base de datos se ha creado correctamente. Esta base de datos contendrá las tablas normalizadas de nuestro modelo de datos. Ahora crearemos las tablas subyacentes al modelo generado

Creación de un usuario MIGBD

Para crear un nuevo usuario en la base de datos, seguiremos los mismos pasos que seguimos cuando se creó el usuario IMPBD. Haremos uso de la sentencia CREATE USER en combinación con la sentencia GRANT para otorgar los privilegios necesarios a este usuario. El usuario MIGBD debe tener los permisos necesarios para poder crear, consultar e insertar datos en las tablas.

```
-- Usuario Impbd
CREATE USER 'MIGBD'@'localhost' IDENTIFIED BY 'migbd_user_passwd';
-- Permisos
GRANT CREATE, INSERT, UPDATE, DELETE, SELECT, REFERENCES ON DEFBD.* TO 'MIGBD'@'localhost';
```

```
mysql> -- Usuario Impbd
mysql> CREATE USER 'MIGBD'@'localhost' IDENTIFIED BY 'migbd_user_passwd';
Query OK, 0 rows affected (0.06 sec)
```

```
mysql> GRANT CREATE, INSERT, UPDATE, DELETE, SELECT, REFERENCES ON DEFBD.* TO 'MIGBD'@'localhost';
Query OK, 0 rows affected (0.02 sec)
```

Finalmente, actualizamos los privilegios mediante FLUSH PRIVILEGES para no salir de la sesión, y comprobaremos que los permisos se hayan otorgado correctamente mediante la sentencia SHOW GRANTS.

```
-- Comprobar los permisos del usuario
FLUSH PRIVILEGES;
SHOW GRANTS FOR 'MIGBD'@'localhost';
```

```
mysql> SHOW GRANTS FOR 'MIGBD'@'localhost';
+-----+-----+
| Grants for MIGBD@localhost |
+-----+-----+
| GRANT USAGE ON *.* TO `MIGBD`@`localhost` |
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, REFERENCES ON `defbd`.* TO `MIGBD`@`localhost` |
+-----+-----+
2 rows in set (0.00 sec)
```

Cabe mencionar que en la sentencia CREATE USER MIGBD se especifica la contraseña del usuario (migbd_user_passwd) y la máquina de la cual el usuario se conectará a la instancia, que es localhost.

Creación de esquema de tablas con MIGBD

Para crear el esquema de tablas del primer apartado con el usuario MIGBD, debemos conectarnos a la instancia de mysql con el usuario migbd. Para conectarnos a la base de datos con el usuario MIGBD, abriremos la línea de comandos de Windows al igual que hicimos en su momento con el usuario impbd y navegaremos mediante el comando **cd** hasta la ruta 'C:\Program Files\MySQL\MySQL Server 8.0\bin'.

Cuando nos encontremos en esta ruta, haremos uso del comando **mysql -u MIGBD -p** para realizar la conexión con el usuario MIGBD. Se nos pedirá la contraseña, que es "migbd_user_passwd". Tras introducirla realizaremos finalmente la conexión a la base de datos.

```
C:\Program Files\MySQL\MySQL Server 8.0\bin>mysql -u MIGBD -p
Enter password: *****
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 49
Server version: 8.0.35 MySQL Community Server - GPL

Copyright (c) 2000, 2023, Oracle and/or its affiliates.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> |
```

Una vez nos hayamos conectado con el usuario, accederemos a la base de datos con el comando USE DEFBD;

```
-- Acceso con MIGBD y accedemos a la bd
-- mysql -u MIGBD -p
USE DEFBD;
```

```
mysql> USE DEFBD;
Database changed
mysql>
```

A continuación, realizaremos los CREATE TABLE, las tablas descritas en el documento adjunto “Mario_Ventura_bd_corregudes_bous_mysql.sql”. El código de creación de las tablas es el siguiente:

```
-- =====
-- ===== Mario Ventura Burgos - 43223476J =====
-- ===== Sistemas de gestión de base de datos =====
-- ===== Código de creación de tablas del modelo de datos definitivo =====
-- =====

-- TABLA DE APODERAT
CREATE TABLE APODERAT (
    ideapo VARCHAR(10), -- Identificador del apoderado (màxim 10 dígits)
    CONSTRAINT PK_APODERAT PRIMARY KEY (ideapo)
) TABLESPACE my_tablespace;

-- TABLA DE UBICACION
CREATE TABLE UBICACION (
    ideubi INT(11) AUTO_INCREMENT, -- Identificador de la ubicación
    nompai VARCHAR(40) NOT NULL, -- Nombre del país
    nomciu VARCHAR(50) NOT NULL, -- Nombre de la ciutat
    CONSTRAINT PK_UBICACION PRIMARY KEY (ideubi)
) TABLESPACE my_tablespace;

-- TABLA DE RAMADERIA
CREATE TABLE RAMADERIA (
    cifram VARCHAR(3), -- CIF de la ramaderia
    nomram VARCHAR(50) NOT NULL, -- Nombre de la ramaderia
    CONSTRAINT PK_RAMADERIA PRIMARY KEY (cifram)
) TABLESPACE my_tablespace;

-- TABLA DE FERIA
CREATE TABLE FERIA (
    fircor VARCHAR(50), -- Nombre de la festa o feria
    CONSTRAINT PK_FERIA PRIMARY KEY (fircor)
) TABLESPACE my_tablespace;

-- TABLA DE PERSONA
CREATE TABLE PERSONA (
    ideper VARCHAR(10), -- Identificador de la persona
    ideubi INT(11) NOT NULL, -- Clave forana que referencia a UBICACION(ideubi)
    nompe VARCHAR(50) NOT NULL, -- Nombre de la persona
```

```

colpe VARCHAR(50) NOT NULL,      -- Primer apellido de la persona
co2pe VARCHAR(50),              -- Segundo apellido de la persona
maiper VARCHAR(100),            -- Mail de la persona
dirper VARCHAR(100) NOT NULL,   -- Dirección de la persona
CONSTRAINT PK_PERSONA PRIMARY KEY (ideper),
CONSTRAINT FK_PERSONA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi)
) TABLESPACE my_tablespace;

-- TABLA DE PLAZA
CREATE TABLE PLAZA (
    nompla VARCHAR(50), -- Nombre de la plaza
    ideubi INT(11) NOT NULL, -- Clave forana que referencia a UBICACION(ideubi)
    anypla DATE NOT NULL, -- Año de construcción de la plaza
    locpla INT NOT NULL, -- Número de asientos de la plaza
    tippla BOOLEAN NULL, -- Si la plaza es fija o movil
    estpla TEXT, -- Estilos predominantes en la construcción
    muspla BOOLEAN NULL, -- Indica si la plaza contiene un museo
    CONSTRAINT PK_PLAZA PRIMARY KEY (nompla),
    CONSTRAINT FK_PLAZA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi),
    CONSTRAINT CHECK_PLAZA_MUSPLA CHECK (muspla IN (0, 1)),
    CONSTRAINT CHECK_PLAZA_TIPPLA CHECK (tippla IN (0, 1))
) TABLESPACE my_tablespace;

-- TABLA DE TORERO
CREATE TABLE TORERO (
    idetor VARCHAR(10), -- Identificador del torero (máximo 10 dígitos)
    ideapo VARCHAR(10) NOT NULL, -- Clave forana que referencia a APODERAT(ideapo)
    CONSTRAINT PK_TORERO PRIMARY KEY (idetor),
    CONSTRAINT FK_TORERO_APODERAT FOREIGN KEY (ideapo) REFERENCES APODERAT(ideapo)
) TABLESPACE my_tablespace;

-- TABLA DE ACTUACION
CREATE TABLE ACTUACION (
    ideact INT(11) AUTO_INCREMENT, -- Identificador de la actuación
    idetor VARCHAR(10) NOT NULL, -- Clave forana que referencia a TORERO(idetor)
    nompla VARCHAR(50) NOT NULL, -- Clave forana que referencia a PLAZA(nompla)
    datcor DATE NOT NULL, -- Fecha de la actuación
    CONSTRAINT PK_ACTUACION PRIMARY KEY (ideact),
    CONSTRAINT FK_ACTUACION_TORERO FOREIGN KEY (idetor) REFERENCES TORERO(idetor),
    CONSTRAINT FK_ACTUACION_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
) TABLESPACE my_tablespace;

-- TABLA DE TORO
CREATE TABLE TORO (
    idtoro INT(11) AUTO_INCREMENT, -- Identificador del toro (autoincremental)
    cifram VARCHAR(3) NOT NULL, -- Clave forana que referencia a RAMADERIA(cifram)
    nombou VARCHAR(50) NOT NULL, -- Nombre o identificación del toro
    anybou DATE NOT NULL, -- Año de nacimiento del toro
    pesbou DECIMAL(10,2) NOT NULL, -- Peso del toro

```



```

    CONSTRAINT PK_TORO PRIMARY KEY (idtoro),
    CONSTRAINT FK_TORO_RAMADERIA FOREIGN KEY (cifram) REFERENCES RAMADERIA(cifram)
) TABLESPACE my_tablespace;

-- TABLA DE ESDEVENIMENT
CREATE TABLE ESDEVENIMENT (
    ideesd INT(11) AUTO_INCREMENT, -- Identificador del esdeveniment
    fircor VARCHAR(50) NOT NULL, -- Clave forana que referencia a FERIA(fircor)
    nompla VARCHAR(50) NOT NULL, -- Clave forana que referencia a PLAZA(nompla)
    datcor DATE NOT NULL, -- Fecha del esdeveniment
    CONSTRAINT PK_ESDEVENIMENT PRIMARY KEY (ideesd),
    CONSTRAINT FK_ESDEVENIMENT_FERIA FOREIGN KEY (fircor) REFERENCES FERIA(fircor),
    CONSTRAINT FK_ESDEVENIMENT_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
) TABLESPACE my_tablespace;

-- TABLA DE RELACION ENTRE ESDEVENIMENT Y TORO
CREATE TABLE R_ESDEVENIMENT_TORO (
    idtoro INT (11) NOT NULL, -- id del toro que hace referencia a la tabla TORO
    ideesd INT(11) NOT NULL, -- id del esdeveniment que hace referencia a la tabla
ESDEVENIMENT
    CONSTRAINT PK_ESD_TORO PRIMARY KEY (idtoro, ideesd),
    CONSTRAINT FK_TORO FOREIGN KEY (idtoro) REFERENCES TORO (idtoro),
    CONSTRAINT FK_ESDEVENIMENT FOREIGN KEY (ideesd) REFERENCES ESDEVENIMENT (ideesd)
) TABLESPACE my_tablespace;

```

```

mysql> -- TABLA DE TORERO
mysql> CREATE TABLE TORERO (
  ->   idetor VARCHAR(10), -- Identificador del torero (máximo 10 dígitos)
  ->   ideapo VARCHAR(10) NOT NULL, -- Clave forana que referencia a APODERAT(ideapo)
  ->   CONSTRAINT PK_TORERO PRIMARY KEY (idetor),
  ->   CONSTRAINT FK_TORERO_APODERAT FOREIGN KEY (ideapo) REFERENCES APODERAT(ideapo)
  -> );
Query OK, 0 rows affected (0.10 sec)

mysql>
mysql> -- TABLA DE ACTUACION
mysql> CREATE TABLE ACTUACION (
  ->   ideact INT(11) AUTO_INCREMENT, -- Identificador de la actuación
  ->   idetor VARCHAR(10) NOT NULL, -- Clave forana que referencia a TORERO(idetor)
  ->   nompla VARCHAR(50) NOT NULL, -- Clave forana que referencia a PLAZA(nompla)
  ->   datcor DATE NOT NULL, -- Fecha de la actuación
  ->   CONSTRAINT PK_ACTUACION PRIMARY KEY (ideact),
  ->   CONSTRAINT FK_ACTUACION_TORERO FOREIGN KEY (idetor) REFERENCES TORERO(idetor),
  ->   CONSTRAINT FK_ACTUACION_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
  -> );
Query OK, 0 rows affected, 1 warning (0.11 sec)

mysql>
mysql> -- TABLA DE TORO
mysql> CREATE TABLE TORO (
  ->   idtoro INT(11) AUTO_INCREMENT, -- Identificador del toro (autoincremental)
  ->   cifram VARCHAR(3) NOT NULL, -- Clave forana que referencia a RAMADERIA(cifram)
  ->   nombou VARCHAR(50) NOT NULL, -- Nombre o identificación del toro
  ->   anybou DATE NOT NULL, -- Año de nacimiento del toro
  ->   pesbou DECIMAL(10,2) NOT NULL, -- Peso del toro
  ->   CONSTRAINT PK_TORO PRIMARY KEY (idtoro),
  ->   CONSTRAINT FK_TORO_RAMADERIA FOREIGN KEY (cifram) REFERENCES RAMADERIA(cifram)
  -> );
Query OK, 0 rows affected, 1 warning (0.08 sec)

```

Ahora comprobamos que las tablas se hayan creado correctamente.

```

-- Creación de las tablas
-- Comprobación de que las tablas se han creado
SHOW TABLES;

```

```

mysql> SHOW TABLES;
+-----+
| Tables_in_defdb |
+-----+
| actuacion        |
| apoderat         |
| esdeveniment     |
| feria            |
| persona          |
| plaza            |
| r_esdeveniment_toro |
| ramaderia        |
| torero           |
| toro             |
| ubicacion        |
+-----+
11 rows in set (0.00 sec)

```

Realización de script SQL para traspaso de información

Para realizar el traspaso de información de las tablas creadas con el usuario IMPBD hacia las tablas creadas con el usuario MIGBD, debemos darle a este último los permisos de lectura necesarios para que pueda leer el contenido de las tablas creadas por IMPBD. Para otorgarle los permisos de lectura, haremos uso una vez más de la cláusula GRANT, otorgándole permisos de SELECT a MIGBD.

```
-- Para realizar scripts de traspaso de info, se necesitan privilegios de lectura en IMPBD
GRANT SELECT ON impbd.* TO 'MIGBD'@'localhost';
-- Actualizamos privilegios por seguridad
FLUSH PRIVILEGES
-- Comprobamos que se ha añadido el permiso
SHOW GRANTS FOR 'MIGBD'@'localhost';
```

```
mysql> GRANT SELECT ON impbd.* TO 'MIGBD'@'localhost';
Query OK, 0 rows affected (0.04 sec)
```

```
mysql> -- Comprobamos que se ha añadido el permiso
mysql> SHOW GRANTS FOR 'MIGBD'@'localhost';
+-----+
| Grants for MIGBD@localhost |
+-----+
| GRANT USAGE ON *.* TO `MIGBD`@`localhost` |
| GRANT SELECT, INSERT, UPDATE, DELETE, CREATE, REFERENCES ON `defbd`.* TO `MIGBD`@`localhost` |
| GRANT SELECT ON `impbd`.* TO `MIGBD`@`localhost` |
+-----+
3 rows in set (0.00 sec)
```

Ahora ya podemos comenzar a trabajar en las sentencias SQL necesarias para realizar la importación de datos. Para la importación de los datos de IMPBD en DEFBD, se hará un subapartado sobre cada tabla, de forma que se entienda bien qué planteamiento se ha seguido para hacer la migración de cada tabla.

La migración de datos de las tablas de IMPBD a las tablas de DEFBD se hará mediante consultas sobre IMPBD que generen un resultado apto para ser insertado en DEFBD. Es decir, se seguirá un planteamiento del tipo “INSERT INTO DEFBD (SELECT INTO IMPBD)”.

A continuación, la explicación correspondiente al proceso seguido para la migración en cada tabla.

Tabla Ubicación

Para insertar datos en la tabla UBICACION de DEFBD, deberemos hacer consultas sobre todas las tablas de IMPBD que contengan información sobre ubicaciones que deban ser almacenadas. El problema que esto puede producir es que, si leemos dos tablas con información sobre ubicaciones e insertamos lo leído en la tabla UBICACIÓN de DEFBD, debemos asegurarnos de que no insertemos lo mismo dos veces. A continuación, un ejemplo simplificado sobre tablas inventadas de cómo esto podría suceder:

UBICACIONES EN PERSONA
Ubicación 1
Ubicación 2

UBICACIONES EN PLAZA
Ubicación 1
Ubicación 3

UBICACIONES EN UBICACIÓN (tras realizar importaciones)
Ubicación 1
Ubicación 2
Ubicación 1
Ubicación 3

Problema: Si insertamos todas las ubicaciones leídas en todas las tablas que contienen ubicaciones, es posible que se inserte una ubicación (u otra fila con otra información como personas, plazas, toros...) más de una vez en la misma tabla, dando lugar a duplicidades.

Solución: En cada inserción que se haga en una tabla de DEFBD, gestionaremos la posibilidad de que esto suceda, e insertaremos solamente las filas que no estén ya introducidas en ese momento. Esto se verá en la condición “*NOT IN*” del “*WHERE*” de las sentencias SQL usadas.

Este mismo razonamiento se usará para el resto de las tablas en las que sea necesario.

A continuación, las sentencias SQL usadas:

```
-- TABLA UBICACIÓN
-- Insert en ubicación desde plaza
INSERT INTO UBICACION (NOMPAI, NOMCIU)
  SELECT DISTINCT PAIPLA, CIUPLA
  FROM IMPBD.PLAZA
  -- Comprobación de que no esté insertado en DEFBD
  WHERE (PAIPLA, CIUPLA) NOT IN (
    SELECT NOMPAI, NOMCIU
```

```

        FROM DEFBD.UBICACION
    );

-- Insert en ubicación desde torero
INSERT INTO UBICACION (NOMPAI, NOMCIU)
    SELECT DISTINCT PAITOR, CIUTOR
    FROM IMPBD.TORERO
    -- Comprobación de que no esté insertado en DEFBD
    WHERE (PAITOR, CIUTOR) NOT IN (
        SELECT NOMPAI, NOMCIU
        FROM DEFBD.UBICACION
    );

-- Insert en ubicación desde apoderat
INSERT INTO UBICACION (NOMPAI, NOMCIU)
    SELECT DISTINCT PAIAPO, CIUAPO
    FROM IMPBD.APODERAT
    -- Comprobación de que no esté insertado en DEFBD
    WHERE (PAIAPO, CIUAPO) NOT IN (
        SELECT NOMPAI, NOMCIU
        FROM DEFBD.UBICACION
    );

```

```

mysql> -- Insert en ubicación desde plaza
mysql> INSERT INTO UBICACION (NOMPAI, NOMCIU)
->     SELECT DISTINCT PAIPLA, CIUPLA
->     FROM IMPBD.PLAZA
->     -- Comprobación de que no esté insertado en DEFBD
->     WHERE (PAIPLA, CIUPLA) NOT IN (
->         SELECT NOMPAI, NOMCIU
->         FROM DEFBD.UBICACION
->     );
Query OK, 248 rows affected (0.10 sec)
Records: 248  Duplicates: 0  Warnings: 0

```

Una vez hecho esto, podemos comprobar si la importación se ha realizado correctamente y sin duplicidades realizando una consulta como la siguiente:

```

-- Comprobación de si hay duplicidades
SELECT NOMPAI, NOMCIU, COUNT(*) AS QTT
FROM UBICACION
GROUP BY NOMPAI, NOMCIU
HAVING COUNT(*) > 1;
-- Debería salir Empty Set (conjunto vacío)

-- Comprobación de cuantas filas hay en la tabla
SELECT COUNT(*) FROM UBICACION;

```

```

mysql> SELECT NOMPAI, NOMCIU, COUNT(*) AS QTT
-> FROM UBICACION
-> GROUP BY NOMPAI, NOMCIU
-> HAVING COUNT(*) > 1;
Empty set (0.03 sec)

```

```
mysql> SELECT COUNT(*) FROM UBICACION;
+-----+
| COUNT(*) |
+-----+
|      5911 |
+-----+
1 row in set (0.03 sec)
```

Como puede verse, en la tabla ubicación contiene ahora 5911 ubicaciones distintas. Se sabe que son distintas debido a que la consulta desarrollada demuestra que no existe ninguna ubicación que, pese a que pueda tener id diferente, tenga exactamente el mismo nombre de país y ciudad.

Para insertar las ubicaciones se ha accedido a todas las tablas que contienen información sobre ubicaciones en el modelo original descrito en las tablas de la base de datos IMPBD. Estas tablas son la tabla Torero, Apoderat y Plaza.

Además, podemos comprobar si la cantidad de inserciones es correcta realizando la siguiente consulta sobre IMPBD:

```
-- Comprobación de si la cantidad es correcta
SELECT COUNT(*) FROM (
    SELECT DISTINCT CIUTOR, PAITOR FROM TORERO
    UNION
    SELECT DISTINCT CIUAPO, PAIAPO FROM APODERAT
    UNION
    SELECT DISTINCT CIUPLA, PAIPLA FROM PLAZA
) AS DIFFERENT_LOCATIONS;
```

```
mysql> -- Comprobación de si la cantidad es correcta
mysql> SELECT COUNT(*) FROM (
->     SELECT DISTINCT CIUTOR, PAITOR FROM TORERO
->         UNION
->     SELECT DISTINCT CIUAPO, PAIAPO FROM APODERAT
->         UNION
->     SELECT DISTINCT CIUPLA, PAIPLA FROM PLAZA
-> ) AS DIFFERENT_LOCATIONS;
+-----+
| COUNT(*) |
+-----+
|      5911 |
+-----+
1 row in set (1.63 sec)
```

Puede verse que la cantidad de inserciones realizada coincide con la cantidad de resultados ofrecidos por la consulta.

Tabla Persona

Para la tabla persona se ejecuta la sentencia SQL siguiente:

```
-- TABLA PERSONA
-- Insert en Persona desde Torero
INSERT INTO PERSONA (ideper, ideubi, nompe, colpe, co2pe, maiper, dirper)
  SELECT IDETOR, UBI.ideubi, NOMTOR, CO1TOR, CO2TOR, MAITOR, DIRTOR
  FROM IMPBD.TORERO TOR
    JOIN DEFBD.UBICACION UBI
    ON UBI.NOMPAI = TOR.PAITOR
    AND UBI.NOMCIU = TOR.CIUTOR
  -- Comprobación de que no esté insertado en DEFBD
  WHERE TOR.IDETOR NOT IN (
    SELECT IDEPER FROM PERSONA
  );

-- Insert en Persona desde Apoderat
INSERT INTO PERSONA (ideper, ideubi, nompe, colpe, co2pe, maiper, dirper)
  SELECT IDEAPO, UBI.ideubi, NOMAPO, CO1APO, CO2APO, MAIAPO, DIRAPO
  FROM IMPBD.APODERAT APO
    JOIN DEFBD.UBICACION UBI
    ON UBI.NOMPAI = APO.PAIAPO
    AND UBI.NOMCIU = APO.CIUAPO
  -- Comprobación de que no esté insertado en DEFBD
  WHERE APO.IDEAPPO NOT IN (
    SELECT IDEPER FROM PERSONA
  );
```

Sin embargo, el segundo INSERT nos genera un error que dice que existe una persona con segundo apellido nulo (es decir, una persona sin segundo apellido que tiene el atributo co2pe a NULL) y nuestra base de datos DEFBD no permite que esto suceda ya que en un primer momento se puso la restricción NOT NULL al atributo co2pe en los CREATE TABLE.

```
mysql> -- Insert en Persona desde Torero
mysql> INSERT INTO PERSONA (ideper, ideubi, nompe, colpe, co2pe, maiper, dirper)
  ->   SELECT IDETOR, UBI.ideubi, NOMTOR, CO1TOR, CO2TOR, MAITOR, DIRTOR
  ->   FROM IMPBD.TORERO TOR
  ->     JOIN DEFBD.UBICACION UBI
  ->     ON UBI.NOMPAI = TOR.PAITOR
  ->     AND UBI.NOMCIU = TOR.CIUTOR
  ->   -- Comprobación de que no esté insertado en DEFBD
  ->   WHERE TOR.IDETOR NOT IN (
  ->     SELECT IDEPER FROM PERSONA
  ->   );
Query OK, 106640 rows affected (21.54 sec)
Records: 106640 Duplicates: 0 Warnings: 0

mysql> INSERT INTO PERSONA (ideper, ideubi, nompe, colpe, co2pe, maiper, dirper)
  ->   SELECT IDEAPO, UBI.ideubi, NOMAPO, CO1APO, CO2APO, MAIAPO, DIRAPO
  ->   FROM IMPBD.APODERAT APO
  ->     JOIN DEFBD.UBICACION UBI
  ->     ON UBI.NOMPAI = APO.PAIAPO
  ->     AND UBI.NOMCIU = APO.CIUAPO
  ->   -- Comprobación de que no esté insertado en DEFBD
  ->   WHERE APO.IDEAPPO NOT IN (
  ->     SELECT IDEPER FROM PERSONA
  ->   );
ERROR 1048 (23000): Column 'co2pe' cannot be null
```

Para solucionar este error, modificaremos esta columna de la tabla PERSONA, permitiendo que tenga atributos nulos y, por tanto, que contenga información de personas que solo cuentan con 1 apellido.

```
-- Modificar tabla persona para que co2pe pueda ser NULL
ALTER TABLE PERSONA
MODIFY COLUMN co2pe VARCHAR(50) NULL;
```

Al intentar hacer esto, surge el siguiente problema:

```
mysql> ALTER TABLE PERSONA
-> MODIFY COLUMN co2pe VARCHAR(50) NULL;
ERROR 1142 (42000): ALTER command denied to user 'MIGBD'@'localhost' for table 'persona'
mysql> |
```

Esto nos indica que el usuario MIGBD no tiene los permisos necesarios para alterar la tabla. Lo que haremos será darle los permisos necesarios, modificar la tabla, y revocarle los permisos nuevamente al usuario. Esto se podría hacer con el usuario root desde otra sesión de MySQL Command Line, pero se ha optado por hacerlo de esta manera para tratar de primera mano los permisos otorgados a los usuarios y familiarizarse un poco más si cabe con la sintaxis asociada a los permisos. Cabe mencionar que es muy importante revocarle los permisos al usuario ya que un usuario no debe tener más permisos de los que son estrictamente necesarios para él. Cabe mencionar, además, que un usuario no puede otorgarse privilegios/permisos a sí mismo, de forma que para darle permisos a MIGBD deberemos hacerlo desde el usuario root de MySQL.

```
-- Darle permiso ALTER (necesario usuario root)
USE DEFBD;
GRANT ALTER ON DEFBD.* TO 'MIGBD'@'localhost';
FLUSH PRIVILEGES;
```

```
mysql> USE DEFBD
Database changed
mysql> GRANT ALTER ON DEFBD.* TO 'MIGBD'@'localhost';
Query OK, 0 rows affected (0.03 sec)

mysql> FLUSH PRIVILEGES;
Query OK, 0 rows affected (0.04 sec)

mysql> |
```

```
-- Modificar tabla persona
ALTER TABLE PERSONA
MODIFY COLUMN co2pe VARCHAR(50) NULL;
```

```
mysql> ALTER TABLE PERSONA
-> MODIFY COLUMN co2pe VARCHAR(50) NULL;
Query OK, 0 rows affected (1.70 sec)
Records: 0 Duplicates: 0 Warnings: 0
```


Para revocarle los privilegios al usuario MIGBD haremos uso de la cláusula REVOKE [\[5\]](#).

```
-- Quitamos permisos a MIGBD
REVOKE ALTER ON *.* FROM 'MIGBD'@'localhost';
```

```
mysql> REVOKE ALTER ON *.* FROM 'MIGBD'@'localhost';
Query OK, 0 rows affected (0.00 sec)
```

Ahora sí, tras hacer esto, se ha podido realizar las inserciones en la tabla PERSONA. Se ha tenido que volver a ejecutar la segunda de las inserciones detalladas previamente.

Cabe destacar que para esto funcione, se ha tenido que hacer la misma modificación en la columna maiper de la tabla PERSONA, que tampoco debe tener la restricción de no ser nula. El procedimiento es el mismo: se modifica la columna mediante código antes de hacer la inserción.

```
-- Se observa lo mismo para maiper
ALTER TABLE PERSONA
MODIFY COLUMN maiper VARCHAR(100) NULL;
```

```
mysql> -- Insert en Persona desde Apoderat
mysql> INSERT INTO PERSONA (ideper, ideubi, nompe, colpe, co2pe, maiper, dirper)
-> SELECT IDEAPO, UBI.ideubi, NOMAPO, CO1APO, CO2APO, MAIAPO, DIRAPO
-> FROM IMPBD.APODERAT APO
-> JOIN DEFBD.UBICACION UBI
-> ON UBI.NOMPAI = APO.PAIAPO
-> AND UBI.NOMCIU = APO.CIUAPO
-> -- Comprobación de que no esté insertado en DEFBD
-> WHERE APO.IDEAPO NOT IN (
-> SELECT IDEPER FROM PERSONA
-> );
Query OK, 132044 rows affected (14.94 sec)
Records: 132044 Duplicates: 0 Warnings: 0
```

Una vez se ha realizado con éxito la inserción, podemos comprobar si hay filas duplicadas mediante una consulta que busque las filas cuyas columnas se repiten. La consulta es la siguiente:

```
-- Comprobación de si hay duplicidades
SELECT IDEPER, NOMPE, COUNT(*) AS QTT
FROM PERSONA
GROUP BY IDEPER, NOMPE
HAVING COUNT(*) > 1;
```

Y Podemos observar que no hay ninguna duplicidad ya que se obtiene un conjunto vacío en el resultado de la consulta.

```
mysql> -- Comprobación de si hay duplicidades
mysql> SELECT IDEPER, NOMPE, COUNT(*) AS QTT
-> FROM PERSONA
-> GROUP BY IDEPER, NOMPE
-> HAVING COUNT(*) > 1;
Empty set (3.75 sec)
```

Por último, la cantidad de filas en la tabla Persona se puede comprobar usando un *COUNT(*)*.

```
-- Comprobación de cuantas filas hay en la tabla
SELECT COUNT(*) FROM PERSONA;
```

```
mysql> -- Comprobación de cuantas filas hay en la tabla
mysql> SELECT COUNT(*) FROM PERSONA;
+-----+
| COUNT(*) |
+-----+
|  238684 |
+-----+
1 row in set (0.04 sec)
```

Para comprobar si esta cantidad de personas insertadas es correcta, vamos a realizar la siguiente consulta sobre la base de datos original IMPBD:

```
-- Comprobación del número de personas únicas
SELECT COUNT(DISTINCT ide)
FROM (
    SELECT IDETOR AS ide
    FROM TORERO
    WHERE IDETOR IS NOT NULL
    UNION
    SELECT IDEAPO
    FROM APODERAT
    WHERE IDEAPO IS NOT NULL
) AS DIFFERENT_PERSONS;
```

Que nos permite comprobar que la cantidad de personas insertadas es la correcta:

```
mysql> -- Comprobación del número de personas únicas
mysql> SELECT COUNT(DISTINCT ide)
-> FROM (
->     SELECT IDETOR AS ide
->     FROM TORERO
->     WHERE IDETOR IS NOT NULL
->     UNION
->     SELECT IDEAPO
->     FROM APODERAT
->     WHERE IDEAPO IS NOT NULL
-> ) AS DIFFERENT_PERSONS;
+-----+
| COUNT(DISTINCT ide) |
+-----+
|          238684 |
+-----+
1 row in set (4.11 sec)
```

Tabla Torero

Para la tabla torero la inserción es simple ya que en la base de datos original IMPBD ya se cuenta con una tabla que contiene la información de los toreros. Se hará una consulta simple y se insertará el resultado en la tabla torero de DEFBD.

```
-- TABLA TORERO
INSERT INTO TORERO (IDETOR, IDEAPO)
  SELECT IDETOR, IDEAPO
  FROM IMPBD.TORERO;
```

```
mysql> -- TABLA TORERO
mysql> INSERT INTO TORERO (IDETOR, IDEAPO)
->     SELECT IDETOR, IDEAPO
->     FROM IMPBD.TORERO;
Query OK, 106640 rows affected (23.08 sec)
Records: 106640  Duplicates: 0  Warnings: 0
```

En este caso no hace falta hacer un *COUNT(*)* para contar el número de filas de la tabla ya que la propia terminal nos lo dice. La tabla torero cuenta con 106640 filas, con 0 duplicidades.

Aun así, se puede comprobar si esta cantidad de inserciones es correcta realizando un *SELECT COUNT* en la tabla TORERO de IMPBD:

```
SELECT COUNT(*) FROM TORERO;
```

```
mysql> SELECT COUNT(*) FROM TORERO;
+-----+
| COUNT(*) |
+-----+
|   106640 |
+-----+
1 row in set (0.04 sec)
```

El número de filas insertadas coincide con la cantidad de datos de la tabla original en IMPBD.

Tabla Apoderat

Para la tabla apoderat se hace otra inserción simple ya que toda la información de los apoderados se encuentra en la tabla apoderat de la base de datos IMPBD. La tabla cuenta con 132189 filas con 0 duplicados, y la sentencia SQL usada es la siguiente:

```
-- TABLA APODERAT
INSERT INTO APODERAT (IDEAPO)
  SELECT DISTINCT IDEAPO
  FROM IMPBD.APODERAT;
```

```
mysql> -- TABLA APODERAT
mysql> INSERT INTO APODERAT (IDEAPO)
-> SELECT DISTINCT IDEAPO
-> FROM IMPBD.APODERAT;
Query OK, 132189 rows affected (7.15 sec)
Records: 132189 Duplicates: 0 Warnings: 0
```

Al igual que en el caso anterior, puede comprobarse si la cantidad de inserciones es la correcta mediante una simple consulta del tipo SELECT COUNT(*).

```
SELECT COUNT(*) FROM APODERAT;
```

```
mysql> SELECT COUNT(*) FROM APODERAT;
+-----+
| COUNT(*) |
+-----+
| 132189 |
+-----+
1 row in set (0.22 sec)
```

Tabla Plaza

Para la tabla plaza se realiza una consulta con todos sus atributos sobre la tabla PLAZA de la base de datos IMPBD. La consulta necesita necesariamente el id de la ubicación en la que se encuentra la plaza, y por ello, accederemos mediante un JOIN a la tabla ubicación de la base de datos DEFBD.

```
-- TABLA PLAZA
INSERT INTO PLAZA (nompla, ideubi, anypla, locpla, tippla, estpla, muspla)
SELECT NOMPLA, UBI.ideubi, ANYPLA, LOCPLA, TIPPLA, ESTPLA, MUSPLA
FROM IMPBD.PLAZA PLA
JOIN UBICACION UBI
ON UBI.NOMCIU = PLA.CIUPLA
AND UBI.NOMPAI = PLA.PAIPLA;
```

```
mysql> -- TABLA PLAZA
mysql> INSERT INTO PLAZA (nompla, ideubi, anypla, locpla, tippla, estpla, muspla)
-> SELECT NOMPLA, UBI.ideubi, ANYPLA, LOCPLA, TIPPLA, ESTPLA, MUSPLA
-> FROM IMPBD.PLAZA PLA
-> JOIN UBICACION UBI
-> ON UBI.NOMCIU = PLA.CIUPLA
-> AND UBI.NOMPAI = PLA.PAIPLA;
Query OK, 266 rows affected (0.05 sec)
Records: 266 Duplicates: 0 Warnings: 0
```

En este caso, para comprobar si la cantidad de inserciones es correcta, haremos la siguiente consulta:

```
SELECT COUNT(*) FROM PLAZA;
```

```
mysql> SELECT COUNT(*) FROM PLAZA;
+-----+
| COUNT(*) |
+-----+
|      266 |
+-----+
1 row in set (0.01 sec)
```

Esto nos demuestra que la cantidad de inserciones es correcta

Tabla Actuación

Para la tabla ACTUACION, se realiza la inserción de datos mediante la siguiente sentencia SQL:

```
-- TABLA ACTUACION
INSERT INTO ACTUACION (idetor, nompla, datcor)
  SELECT TOR.idetor, PLA.nompla, ACT.DATCOR
  FROM IMPBD.ACTUACIO ACT
    JOIN DEFBD.TORERO TOR
    ON TOR.IDETOR = ACT.IDETOR
    JOIN DEFBD.PLAZA PLA
    ON PLA.NOMPLA = ACT.NOMPLA;
```

```
mysql> -- TABLA ACTUACION
mysql> INSERT INTO ACTUACION (idetor, nompla, datcor)
->      SELECT TOR.idetor, PLA.nompla, ACT.DATCOR
->      FROM IMPBD.ACTUACIO ACT
->      JOIN DEFBD.TORERO TOR
->      ON TOR.IDETOR = ACT.IDETOR
->      JOIN DEFBD.PLAZA PLA
->      ON PLA.NOMPLA = ACT.NOMPLA;
Query OK, 535990 rows affected (1 min 7.42 sec)
Records: 535990  Duplicates: 0  Warnings: 0
```

Se realiza una consulta en la tabla ACTUACIO de la base de datos IMPBD y el resultado se inserta en la tabla ACTUACION de la base de datos DEFBD. Para insertar datos en DEFBD necesitamos datos de la tabla PLAZA y de la tabla TORERO, y por ello es necesario hacer dos operaciones de JOIN para pasar por esas tablas. Sin embargo, estas operaciones de JOIN se hacen sobre las tablas de DEFBD, que no están vacías porque ya se han realizado inserciones previamente.

Para comprobar que la cantidad de actuaciones insertada es correcta haremos un SELECT COUNT.

```
SELECT COUNT(*) FROM ACTUACIO;
```

```
mysql> SELECT COUNT(*) FROM ACTUACIO;
+-----+
| COUNT(*) |
+-----+
|    535990 |
+-----+
1 row in set (0.50 sec)
```

Tabla Ramaderia

Para realizar la inserción en la tabla RAMADERIA se ha hecho una consulta sobre la tabla esdeveniment de la base de datos IMPBD, que busca todos los cif diferentes y su respectivo nombre, y verifica que no exista una fila en la tabla RAMADERIA con el valor de cifram igual al valor de cifram en ESDEVENIMENT. La subconsulta SELECT 1 FROM RAMADERIA se encarga de devolver al menos una fila si esto sucede.

El resultado es que se insertan en la tabla RAMADERIA de DEFBD las filas de la tabla ESDEVENIMENT en IMPBD que no existen previamente en la tabla RAMADERIA y que tienen combinaciones únicas de cif y nombre de la Ramaderia.

```
-- TABLA RAMADERIA
INSERT INTO DEFBD.RAMADERIA (CIFRAM, NOMRAM)
  SELECT DISTINCT ESD.CIFRAM, ESD.NOMRAM
  FROM IMPBD.ESDEVENIMENT ESD
  WHERE NOT EXISTS (
    SELECT 1
    FROM RAMADERIA RAM
    WHERE RAM.CIFRAM = ESD.CIFRAM
  );
```

```
mysql> -- TABLA RAMADERIA
mysql> INSERT INTO DEFBD.RAMADERIA (CIFRAM, NOMRAM)
-> SELECT DISTINCT ESD.CIFRAM, ESD.NOMRAM
-> FROM IMPBD.ESDEVENIMENT ESD
-> WHERE NOT EXISTS (
-> SELECT 1
-> FROM RAMADERIA RAM
-> WHERE RAM.CIFRAM = ESD.CIFRAM
-> );
Query OK, 362 rows affected (1 min 17.92 sec)
Records: 362 Duplicates: 0 Warnings: 0
```

Se han insertado un total de 362 ramaderias diferentes sin duplicidades. Para comprobar si esta cantidad es la correcta, haremos la siguiente consulta sobre la base de datos IMPBD:

```
-- Ramaderias en la tabla esdeveniment de IMPBD
SELECT COUNT(DISTINCT CIFRAM) AS QTT FROM ESDEVENIMENT;
```

```
mysql> -- Ramaderias en la tabla esdeveniment de IMPBD
mysql> SELECT COUNT(DISTINCT CIFRAM) AS QTT FROM ESDEVENIMENT;
+-----+
| QTT |
+-----+
| 362 |
+-----+
1 row in set (44.61 sec)
```

Tabla Toro

Para insertar los datos en la tabla TORO, se deberá buscar la información en la tabla ESDEVENIMENT de la base de datos IMPBD. No hace falta insertar un id al toro ya que el id es un entero de carácter auto incremental y se añadirá automáticamente sin necesidad de nuestra intervención. Mediante la cláusula “NOT IN” se comprueba que los datos seleccionados en la tabla ESDEVENIMENT de IMPBD estén ya insertados en la tabla TORO de DEFBD. El código utilizado es el siguiente:

```
-- TABLA TORO
INSERT INTO TORO (CIFRAM, NOMBOU, ANYBOU, PESBOU)
  SELECT DISTINCT ESD.CIFRAM, ESD.NOMBOU, ESD.ANYBOU, ESD.PESBOU
  FROM IMPBD.ESDEVENIMENT ESD
  WHERE (ESD.CIFRAM, ESD.NOMBOU, ESD.ANYBOU) NOT IN (
    SELECT TOR.CIFRAM, TOR.NOMBOU, TOR.ANYBOU
    FROM DEFBD.TORO TOR
  );
```

```
mysql> -- TABLA TORO
mysql> INSERT INTO TORO (CIFRAM, NOMBOU, ANYBOU, PESBOU)
-> SELECT DISTINCT ESD.CIFRAM, ESD.NOMBOU, ESD.ANYBOU, ESD.PESBOU
-> FROM IMPBD.ESDEVENIMENT ESD
-> WHERE (ESD.CIFRAM, ESD.NOMBOU, ESD.ANYBOU) NOT IN (
-> SELECT TOR.CIFRAM, TOR.NOMBOU, TOR.ANYBOU
-> FROM DEFBD.TORO TOR
-> );
Query OK, 5800581 rows affected (38 min 4.37 sec)
Records: 5800581 Duplicates: 0 Warnings: 0
```

Para comprobar que la cantidad de toros introducidos es la correcta, vamos a realizar la siguiente consulta:

```
-- Toros distintos en IMPBD
SELECT COUNT(DISTINCT NOMBOU, ANYBOU, PESBOU) AS QTT_TOROS FROM ESDEVENIMENT;
```

```
mysql> -- Toros distintos en IMPBD
mysql> SELECT COUNT(DISTINCT NOMBOU, ANYBOU, PESBOU) AS QTT_TOROS FROM ESDEVENIMENT;
+-----+
| QTT_TOROS |
+-----+
|    5800581 |
+-----+
1 row in set (2 min 12.03 sec)
```

Tabla FERIA

Para la tabla feria la inserción es simple ya que esta tabla cuenta con solamente un atributo llamado fircor. Este atributo se encuentra en la tabla ESDEVENIMENT de la base de datos IMPBD, de forma que con una consulta simple sobre esta tabla podremos insertar lo obtenido en la tabla FERIA de DEFBD. Cabe destacar que es necesaria la cláusula DISTINCT ya que una misma feria puede aparecer varias veces en la tabla ESDEVENIMENT, por tanto, si no se usara la cláusula, la tabla FERIA tendría duplicados.

```
-- TABLA FERIA
INSERT INTO FERIA (fircor)
  SELECT DISTINCT ESD.fircor
  FROM IMPBD.esdeveniment ESD;
```

```
mysql> -- TABLA FERIA
mysql> INSERT INTO FERIA (fircor)
  ->     SELECT DISTINCT ESD.fircor
  ->     FROM IMPBD.esdeveniment ESD;
Query OK, 239 rows affected (35.35 sec)
Records: 239  Duplicates: 0  Warnings: 0
```

Se han insertado un total de 239 ferias diferentes, y no existen duplicados en la tabla. Para comprobar si esta cantidad de inserciones es la correcta haremos la siguiente consulta:

```
-- Cantidad de ferias distintas en IMPBD
SELECT COUNT(DISTINCT fircor) AS QTT_FERIAS FROM ESDEVENIMENT;
```

```
mysql> -- Cantidad de ferias distintas en IMPBD
mysql> SELECT COUNT(DISTINCT fircor) AS QTT_FERIAS FROM ESDEVENIMENT;
+-----+
| QTT_FERIAS |
+-----+
|         239 |
+-----+
1 row in set (1 min 19.02 sec)
```


Tabla Esdeveniment

Para insertar en la tabla esdeveniment, se hace una consulta simple en la tabla ESDEVENIMENT de IMPBD, seleccionando todos los atributos e insertándolos en la tabla ESDEVENIMENT de DEFBD. Una vez más, se usa la cláusula DISTINCT para evitar duplicidades.

```
-- TABLA ESDEVENIMENT
INSERT INTO ESDEVENIMENT (FIRCOR, NOMPLA, DATCOR)
  SELECT DISTINCT ESD.FIRCOR, PLA.NOMPLA, ESD.DATCOR
  FROM IMPBD.ESDEVENIMENT ESD
    INNER JOIN PLAZA PLA
    ON PLA.NOMPLA = ESD.NOMPLA;
```

```
mysql> -- TABLA ESDEVENIMENT
mysql> INSERT INTO ESDEVENIMENT (FIRCOR, NOMPLA, DATCOR)
->      SELECT DISTINCT ESD.FIRCOR, PLA.NOMPLA, ESD.DATCOR
->      FROM IMPBD.ESDEVENIMENT ESD
->      INNER JOIN PLAZA PLA
->      ON PLA.NOMPLA = ESD.NOMPLA;
Query OK, 125443 rows affected (2 min 39.17 sec)
Records: 125443  Duplicates: 0  Warnings: 0
```

Como se puede ver en la imagen, se insertan un total de 125.443 esdeveniments diferentes entre sí, ya que no existen duplicidades en la tabla.

Esta cantidad llama la atención ya que en la tabla original ESDEVENIMENT de IMPBD había más de 10 millones de filas, y ahora hemos obtenido solamente 125.443, es decir, unas 100 veces menos. Esto se debe a que en la tabla original había una gran cantidad de esdeveniments que aparecían varias veces porque se han celebrado en las mismas fechas, pero en plazas distintas. Por ello, podemos comprobar mediante una consulta el número de esdeveniments que existen bajo la condición de que se celebren el mismo día, pero en diferentes plazas.

```
-- Esdeveniments celebrados el mismo día en plazas distintas
SELECT FIRCOR, DATCOR, COUNT(*) AS QTT
FROM ESDEVENIMENT
GROUP BY FIRCOR, DATCOR
HAVING COUNT(*)>1;
```

Feria de Zaragoza	1989-08-18	54
Feria de Zaragoza	1990-05-31	54
Feria de Zaragoza	1991-05-19	108
Feria de Zaragoza	1992-12-23	54
Feria de Zaragoza	1993-12-16	54
Feria de Zaragoza	1994-07-01	108
Feria de Zaragoza	1995-11-27	108
Feria de Zaragoza	1996-06-13	54
Feria de Zaragoza	1997-12-20	54
Feria de Zaragoza	1998-06-13	54
Feria de Zaragoza	1999-01-11	108
Feria de Zaragoza	2000-05-19	54
Feria de Zaragoza	2001-09-13	108
Feria de Zaragoza	2002-09-26	162
Feria de Zaragoza	2003-07-19	162
Feria de Zaragoza	2004-09-03	216
Feria de Zaragoza	2005-06-09	54
Feria de Zaragoza	2006-10-26	54
Feria de Zaragoza	2007-01-02	54
Feria de Zaragoza	2008-07-12	54
Feria de Zaragoza	2009-10-16	216
Feria de Zaragoza	2010-04-11	54
Feria de Zaragoza	2011-04-30	324
Feria de Zaragoza	2012-09-03	54
Feria de Zaragoza	2013-02-21	54

86136 rows in set (4 min 28.61 sec)

Como podemos apreciar en la imagen anterior, existen un total de 86.136 esdeveniments que se han celebrado en misma fecha, pero en plazas diferentes. Teniendo en cuenta el grado de normalización de la tabla original de IMPBD, es lógico comprender como un mismo esdeveniment podría aparecer numerosas veces debido a que en cada una de sus apariciones se debe hacer referencia a una plaza distinta de las usadas en ese esdeveniment. Como consecuencia, si un esdeveniment se celebra en, por ejemplo, 17 plazas diferentes, aparece 17 veces en la tabla original de IMPBD, con la diferencia de que la columna NOMPLA cambiará en estas 17 filas, con los nombres de las 17 plazas implicadas.

Si realizamos una consulta sobre la tabla ESDEVENIMENT de la base de datos IMPBD, pero hacemos uso de la cláusula DISTINCT, podemos enumerar la cantidad de esdeveniments diferentes como tal, y esta debería coincidir con la cantidad de esdeveniments insertados en la tabla ESDEVENIMENT de DEFBD. La consulta es la siguiente:

```
SELECT COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) FROM ESDEVENIMENT;
```

```
mysql> SELECT COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) FROM ESDEVENIMENT;
+-----+
| COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) |
+-----+
| 125443 |
+-----+
1 row in set (1 min 42.36 sec)
```

El total contado coincide exactamente con la cantidad de esdeveniments insertados en la tabla ESDEVENIMENT de la base de datos DEFBD. Esto es un indicativo de que esta inserción es correcta.

Por último, podemos cuantificar la cantidad de esdeveniments diferentes en la tabla IMPBD para comprobar si la cantidad de inserciones es la correcta:

```
-- Cantidad de esdeveniments diferentes en IMPBD
SELECT COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) FROM ESDEVENIMENT;
```

```
mysql> -- Cantidad de esdeveniments diferentes en IMPBD
mysql> SELECT COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) FROM ESDEVENIMENT;
+-----+
| COUNT(DISTINCT FIRCOR, DATCOR, NOMPLA) |
+-----+
| 125443 |
+-----+
1 row in set (2 min 28.72 sec)
```

Tabla R_Esdeveniment_Toro

Finalmente, para insertar la información en esta tabla se hace la una consulta similar a las anteriores:

```
-- Insertar datos en R_ESDEVENIMENT_BOU
INSERT INTO R_ESDEVENIMENT_TORO (idtoro, ideesd)
  SELECT DISTINCT idtoro, ideesd
  FROM DEFBD.TORO TOR
    -- Acceso a esdeveniment de IMPBD
    JOIN IMPBD.ESDEVENIMENT ESD
    ON ESD.NOMBOU = TOR.NOMBOU
    AND ESD.ANYBOU = TOR.ANYBOU
    AND ESD.PESBOU = TOR.PESBOU
    -- Acceso a esdeveniment de DEFBD
    JOIN DEFBD.ESDEVENIMENT E
    ON E.FIRCOR=ESD.FIRCOR
    AND E.NOMPLA=ESD.NOMPLA
    AND E.DATCOR=ESD.DATCOR;
```

```
mysql> -- Insertar datos en R_ESDEVENIMENT_BOU
mysql> INSERT INTO R_ESDEVENIMENT_TORO (idtoro, ideesd)
-> SELECT DISTINCT idtoro, ideesd
-> FROM DEFBD.TORO TOR
-> -- Acceso a esdeveniment de IMPBD
-> JOIN IMPBD.ESDEVENIMENT ESD
-> ON ESD.NOMBOU = TOR.NOMBOU
-> AND ESD.ANYBOU = TOR.ANYBOU
-> AND ESD.PESBOU = TOR.PESBOU
-> -- Acceso a esdeveniment de DEFBD
-> JOIN DEFBD.ESDEVENIMENT E
-> ON E.FIRCOR=ESD.FIRCOR
-> AND E.NOMPLA=ESD.NOMPLA
-> AND E.DATCOR=ESD.DATCOR;
Query OK, 10006641 rows affected (8 hours 16 min 6.37 sec)
Records: 10006641 Duplicates: 0 Warnings: 0
```

Podemos observar que se ha insertado un total de 10.006.641 filas. Para comprobar si esta cantidad es correcta podemos hacer un SELECT COUNT sobre la tabla ESDEVENIMENT de IMPBD y comprobar si el número coincide.

```
-- Comprobación de cantidades en IMPBD
SELECT COUNT(*) AS QTT_ESD FROM ESDEVENIMENT;
```

```
mysql> -- Comprobación de cantidades en IMPBD
mysql> SELECT COUNT(*) AS QTT_ESD FROM ESDEVENIMENT;
+-----+
| QTT_ESD |
+-----+
| 10006641 |
+-----+
1 row in set (10.66 sec)
```

Actualización de fechas de nacimiento de toros

Antes de actualizar la información con los toros de una edad menor a dos años y medio, se hará una consulta para tener una idea de si existen muchos toros por actualizar o no. Si esta consulta nos diese *empty set* como resultado, indicaría que no existen toros de menos de dos años y medio de edad, por tanto, podríamos ahorrarnos hacer la actualización. Dado que lo único que nos interesa es saber si existen toros cuya edad se deba actualizar, añadiremos un límite al resultado de la consulta, de forma que si hay muchos toros que se deben actualizar, no perderemos el tiempo mostrando los resultados.

En este caso concreto hemos optado por poner un límite de 100 filas en el resultado, mediante el uso de la cláusula LIMIT X. La consulta es la siguiente:

```
-- Seleccionamos los toros con menos de dos años y medio
SELECT DISTINCT TOR.IDTORO, TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
FROM TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
  ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
  ON ET.IDEESD = ESD.IDEESD
  -- Como son 2,5 años, debemos contar por meses o días
  -- En este caso haremos uso de la función DATEDIFF usando días
WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912 -- 365 * 2,5 = 912,5 = 912
ORDER BY IDTORO
-- Limitamos a 100 el resultado como número orientativo
LIMIT 100;
```

473	Alfonso	1671-11-07	1673-04-15
485	Alfonso	1970-08-10	1953-05-08
487	Alfreda	1632-01-08	1633-12-06
488	Alfreda	1652-11-23	1654-03-07
491	Alfreda	1820-11-07	1823-04-25
493	Alfreda	1825-10-18	1826-11-05
502	Ali	1612-06-11	1614-06-13
504	Ali	1683-06-23	1685-01-06
526	Alíka	1764-03-01	1766-07-06
554	Alisa	1883-11-16	1886-03-28
558	Alisa	1923-12-09	1925-03-08
560	Alisa	1995-02-17	1965-08-28
561	Allegra	1612-06-11	1614-05-04

-----+-----+-----+-----+
100 rows in set (0.07 sec)

Podemos comprobar como sí que existen toros cuya fecha de nacimiento debe ser actualizada. Para llevar a cabo la actualización, haremos un UPDATE sobre la base de datos DEFBD.

Antes de actualizar la base de datos DEFBD, se debe tener en cuenta que esta operación es crítica, ya que si el UPDATE no funciona como se espera, ya no habrá vuelta atrás y se deberá borrar toda la información de esta tabla para volver a insertarla y volver a hacer el UPDATE. Además, se presenta la dificultad de que la cantidad de años a actualizar es 2'5, de forma que existen varios planteamientos a seguir: 2 años + 6 meses, 2'5 años, 912 días, 30 meses, etc...

Teniendo en cuenta la criticidad de esta actualización y las distintas posibilidades que existen, se ha optado por hacer una base de datos temporal con la misma tabla TORO que la de DEFBD, sobre la cual se insertarán una serie de toros de prueba y se intentará actualizar sus fechas de nacimiento siguiendo algún planteamiento concreto.

A continuación, una base de datos de prueba con su respectiva tabla e inserciones:

```
-- CREACIÓN DE BD DE PRUEBA
CREATE DATABASE TOROS_PRUEBA
CHARACTER SET utf8mb4
COLLATE utf8mb4_0900_ai_ci;

-- ACCESO A LA BD
USE TOROS_PRUEBA;

-- TABLA DE TORO (ADAPTADA SIN FK'S)
CREATE TABLE TORO (
  idtoro INT(11) AUTO_INCREMENT, -- Identificador del toro (autoincremental)
  cifram VARCHAR(3) NOT NULL,     -- Clave forana que referencia a RAMADERIA(cifram)
  nombou VARCHAR(50) NOT NULL,    -- Nombre o identificación del toro
  anybou DATE NOT NULL,           -- Año de nacimiento del toro
  pesbou DECIMAL(10,2) NOT NULL,  -- Peso del toro
  CONSTRAINT PK_TORO PRIMARY KEY (idtoro)
```

```
);

-- INSERTS DE EJEMPLO
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('ABC', 'Toro1', '2020-01-01', 800.50);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('DEF', 'Toro2', '2019-02-15', 750.25);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('GHI', 'Toro3', '2022-05-20', 900.75);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('JKL', 'Toro4', '2021-12-10', 820.00);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('MNO', 'Toro5', '2020-08-03', 880.30);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('PQR', 'Toro6', '2019-11-25', 720.80);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('STU', 'Toro7', '2022-03-15', 950.00);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('VWX', 'Toro8', '2021-07-08', 870.60);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('YZA', 'Toro9', '2020-09-30', 810.40);
INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('BCD', 'Toro10', '2019-04-12', 760.90);

-- ACTUALIZACIÓN DE TOROS MEDIANTE PLANTEAMIENTO
-- PLANTEAMIENTO: INTERVALO 2 AÑOS + INTERVALO 6 MONTHS
UPDATE TORO TOR
SET ANYBOU = DATE_SUB(TOR.ANYBOU, INTERVAL 2 YEAR) - INTERVAL 6 MONTH;
```

```
mysql> INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('VWX', 'Toro8', '2021-07-08', 870.60);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('YZA', 'Toro9', '2020-09-30', 810.40);
Query OK, 1 row affected (0.00 sec)

mysql> INSERT INTO TORO (cifram, nombou, anybou, pesbou) VALUES ('BCD', 'Toro10', '2019-04-12', 760.90);
Query OK, 1 row affected (0.00 sec)
```

```
mysql> UPDATE TORO TOR
->      -- JOIN R_ESDEVENIMENT_BOU RB ON TOR.IDBOU = RB.IDBOU
->      -- JOIN ESDEVENIMENT E ON RB.IDESDV = E.IDESDV
-> SET ANYBOU = DATE_SUB(TOR.ANYBOU, INTERVAL 2 YEAR) - INTERVAL 6 MONTH;
Query OK, 10 rows affected (0.01 sec)
Rows matched: 10  Changed: 10  Warnings: 0
```

Se pueden apreciar los siguientes cambios en las fechas de nacimiento de los toros insertados:

Antes

```
mysql> SELECT * FROM TORO;
```

idtoro	cifram	nombou	anybou	pesbou
1	ABC	Toro1	2020-01-01	800.50
2	DEF	Toro2	2019-02-15	750.25
3	GHI	Toro3	2022-05-20	900.75
4	JKL	Toro4	2021-12-10	820.00
5	MNO	Toro5	2020-08-03	880.30
6	PQR	Toro6	2019-11-25	720.80
7	STU	Toro7	2022-03-15	950.00
8	VWX	Toro8	2021-07-08	870.60
9	YZA	Toro9	2020-09-30	810.40
10	BCD	Toro10	2019-04-12	760.90

```
10 rows in set (0.00 sec)
```

Después

```
mysql> SELECT * FROM TORO;
```

idtoro	cifram	nombou	anybou	pesbou
1	ABC	Toro1	2017-07-01	800.50
2	DEF	Toro2	2016-08-15	750.25
3	GHI	Toro3	2019-11-20	900.75
4	JKL	Toro4	2019-06-10	820.00
5	MNO	Toro5	2018-02-03	880.30
6	PQR	Toro6	2017-05-25	720.80
7	STU	Toro7	2019-09-15	950.00
8	VWX	Toro8	2019-01-08	870.60
9	YZA	Toro9	2018-03-30	810.40
10	BCD	Toro10	2016-10-12	760.90

```
10 rows in set (0.00 sec)
```

Si se miran las filas individualmente, se puede comprobar como las fechas han cambiado en 2 años y medio. Por ejemplo, el primer toro (idtoro = 1) ha pasado de tener anybou=2020-01-01 a tener 2017-07-01. Esto confirma que es correcto el planteamiento de usar dos intervalos, uno de 2 años y otro de 6 meses, mediante la cláusula *INTERVAL* usada con la función *DATE_SUB* [6].

Dado que ahora sabemos que el planteamiento es correcto, podemos hacer esta actualización en la base de datos de DEFBD, añadiendo las correspondientes operaciones de JOIN y la condición que establece que solo se le deben cambiar las fechas de nacimiento a unos toros específicos.

La operación de actualización es la siguiente:

```
-- Actualización de toros con menos de 2,5 años de edad
UPDATE TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
  ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
  ON ET.IDEESD = ESD.IDEESD
-- Se usa el planteamiento de usar dos intervalos: 2 YEAR + 6 MONTH
SET ANYBOU = DATE_SUB(ESD.DATCOR, INTERVAL 2 YEAR) - INTERVAL 6 MONTH
-- 365 dias x 2'5 = 912 dias
WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912;
```

```
mysql> -- Actualización de toros con menos de 2,5 años de edad
mysql> UPDATE TORO TOR
  -> JOIN R_ESDEVENIMENT_TORO ET
  -> ON TOR.IDTORO = ET.IDTORO
  -> JOIN ESDEVENIMENT ESD
  -> ON ET.IDEESD = ESD.IDEESD
  -> -- Se usa el planteamiento de usar dos intervalos: 2 YEAR + 6 MONTH
  -> SET ANYBOU = DATE_SUB(ESD.DATCOR, INTERVAL 2 YEAR) - INTERVAL 6 MONTH
  -> -- 365 dias x 2'5 = 912 dias
  -> WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912;
Query OK, 1075158 rows affected (42 min 47.89 sec)
Rows matched: 1075313 Changed: 1075158 Warnings: 0
```

Una vez realizada la actualización, procedemos a comprobar que estos datos se han actualizado correctamente. Para ello, podemos seleccionar arbitrariamente algunos de los toros que han salido como resultado de la consulta previa al UPDATE, y comprobar si sus fechas se han cambiado correctamente. En nuestro caso hemos escogido 3 toros llamados Alfonso, Ali y Allegra (todos ellos aparecen en la imagen posterior a la consulta que se hizo para comprobar si existían toros cuya fecha debía ser actualizada). Se genera las siguientes consultas (cada una contiene un comentario con la fecha de nacimiento del toro antes del UPDATE) para comprobar su fecha de nacimiento tras la actualización:

```
-- Comprobación de actualización correcta
SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
FROM TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
    ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
    ON ET.IDEESD = ESD.IDEESD
-- Este toro tenia fecha de nacimiento 1671-11-07
WHERE TOR.NOMBOU = 'Alfonso' AND DATE(ESD.DATCOR) = '1673-04-15'
-- Limitamos a 1 el resultado
LIMIT 1;

SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
FROM TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
    ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
    ON ET.IDEESD = ESD.IDEESD
-- Este toro tenia fecha de nacimiento 1612-06-11
WHERE TOR.NOMBOU = 'Ali' AND DATE(ESD.DATCOR) = '1614-06-13'
-- Limitamos a 1 el resultado
LIMIT 1;

SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
FROM TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
    ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
    ON ET.IDEESD = ESD.IDEESD
-- Este toro tenia fecha de nacimiento 1612-06-11
WHERE TOR.NOMBOU = 'Allegra' AND DATE(ESD.DATCOR) = '1614-05-04'
-- Limitamos a 1 el resultado
LIMIT 1;
```

Y podemos comprobar que los resultados demuestran que ahora estos toros cuentan con, al menos, 2 años y medio de edad en el momento en el que se produjo el esdeveniment al que están asociados (ESDEVENIMENT.DATCOR):


```
mysql> -- Comprobación de actualización correcta
mysql> SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
-> FROM TORO TOR
-> JOIN R_ESDEVENIMENT_TORO ET
-> ON TOR.IDTORO = ET.IDTORO
-> JOIN ESDEVENIMENT ESD
-> ON ET.IDEESD = ESD.IDEESD
-> -- Este toro tenia fecha de nacimiento 1671-11-07
-> WHERE TOR.NOMBOU = 'Alfonso' AND DATE(ESD.DATCOR) = '1673-04-15'
-> -- Limitamos a 1 el resultado
-> LIMIT 1;
```

NOMBOU	ANYBOU	DATCOR
Alfonso	1670-10-15	1673-04-15

1 row in set (0.01 sec)

```
mysql> SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
-> FROM TORO TOR
-> JOIN R_ESDEVENIMENT_TORO ET
-> ON TOR.IDTORO = ET.IDTORO
-> JOIN ESDEVENIMENT ESD
-> ON ET.IDEESD = ESD.IDEESD
-> -- Este toro tenia fecha de nacimiento 1612-06-11
-> WHERE TOR.NOMBOU = 'Ali' AND DATE(ESD.DATCOR) = '1614-06-13'
-> -- Limitamos a 1 el resultado
-> LIMIT 1;
```

NOMBOU	ANYBOU	DATCOR
Ali	1611-12-13	1614-06-13

1 row in set (0.00 sec)

```
mysql> SELECT TOR.NOMBOU, TOR.ANYBOU, ESD.DATCOR
-> FROM TORO TOR
-> JOIN R_ESDEVENIMENT_TORO ET
-> ON TOR.IDTORO = ET.IDTORO
-> JOIN ESDEVENIMENT ESD
-> ON ET.IDEESD = ESD.IDEESD
-> -- Este toro tenia fecha de nacimiento 1612-06-11
-> WHERE TOR.NOMBOU = 'Allegra' AND DATE(ESD.DATCOR) = '1614-05-04'
-> -- Limitamos a 1 el resultado
-> LIMIT 1;
```

NOMBOU	ANYBOU	DATCOR
Allegra	1611-11-04	1614-05-04

1 row in set (0.01 sec)

Por tanto, podemos comprobar que, efectivamente, la actualización se ha llevado a cabo de forma correcta.

Realización de análisis para optimización

Análisis

Si realizamos un EXPLAIN UPDATE de la actualización llevada a cabo en el apartado anterior, podemos ver cómo se procesa el UPDATE. Al realizarlo observamos lo siguiente:

```
-- ANÁLISIS
EXPLAIN UPDATE TORO TOR
  JOIN R_ESDEVENIMENT_TORO ET
  ON TOR.IDTORO = ET.IDTORO
  JOIN ESDEVENIMENT ESD
  ON ET.IDEESD = ESD.IDEESD
SET ANYBOU = DATE_SUB(ESD.DATCOR, INTERVAL 2 YEAR) - INTERVAL 6 MONTH
WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912;
```

```
--> WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	UPDATE	TORO	NULL	ALL	PRIMARY	NULL	NULL	NULL	5808702	100.00	NULL
1	SIMPLE	ET	NULL	ref	PRIMARY, FK_ESDEVENIMENT	PRIMARY	4	defbd.TOR.idtoro	1	100.00	Using index
1	SIMPLE	ESD	NULL	eq_ref	PRIMARY	PRIMARY	4	defbd.ET.ideesd	1	100.00	Using where

3 rows in set, 1 warning (0.03 sec)

Se observa para la tabla TORO que el número aproximado de filas que se examinarán, es decir, la columna *rows*, tiene el valor **5.808.702**, y esto evidencia que la consulta se puede mejorar, ya que recorrer casi 6 millones de filas tiene un coste computacional alto. Además, la columna *type* nos muestra que se escanea toda la tabla TORO, por eso aparece el valor *type=ALL*. La columna *possible_keys* nos muestra las recomendaciones sobre índices que se podrían aplicar, y la columna *key* nos revela gracias a su valor NULL que no se ha usado ningún índice. *Key_len* muestra la longitud del índice usado, que evidentemente es nulo ya que no se usa índice.

Respecto a la tabla R_ESDEVENIMENT_TORO (alias ET) se puede ver como la columna *type* tiene el valor *ref*, indicando que se hace un recorrido de todas las entradas del índice y accede a la tabla para obtener las columnas adicionales en caso de ser necesario. La columna *key_len* indica una longitud de 4 en la clave, y la columna *rows* indica que únicamente se espera recorrer 1 fila.

Por último, respecto a la tabla ESDEVENIMENT, se aprecia *type=eq_ref*, indicando que el índice retorna una fila indexada y se accede a la tabla a recuperar las columnas adicionales si es necesario. Esto ocurre cuando se usa una PK o UNIQUE no nula. La indexación se considera como una constante. Esto concuerda con el valor observado en la columna *rows*, ya que se espera recorrer únicamente una fila.

Por tanto, en base a este análisis y con el fin de optimizar la consulta anterior, se plantea la creación de índices en las columnas implicadas en este proceso con el objetivo de agilizar la búsqueda, acceso y comparación de estos valores. Esto debería servir de ayuda ya que los índices ordenan e indexan la información, y esto permitirá al motor de la base de datos reducir el número de operaciones necesarias para encontrar los valores que cumplan la condición establecida.

Una buena práctica para mejorar consultas con el uso de índices es crearlos en las columnas implicadas en las operaciones de JOIN y en las condiciones del WHERE. Además, cabe mencionar que SQL crea índices automáticamente en las claves de las tablas aunque estos no sean creados manualmente, de forma que si creamos índices en estas columnas no servirá de nada ya que estas ya cuentan con índices.

Todas las columnas implicadas en las operaciones de JOIN ya son claves de por sí, de forma que sql ya ha creado índices en ellas. Por ello, para notar una mejora real en el rendimiento de la consulta, se crearán índices en las columnas implicadas en el WHERE: la columna DATCOR de la tabla ESDEVENIMENT y la columna ANYBOU de TORO.

Mejora

A continuación, se aplicarán las medidas de mejora explicadas anteriormente.

```
-- Crear índice en la columna DATCOR de la tabla ESDEVENIMENT
CREATE INDEX idx_ESD_DATCOR ON ESDEVENIMENT (DATCOR);
-- Crear índice en la columna ANYBOU de la tabla TORO
CREATE INDEX idx_TORO_ANYBOU ON TORO (ANYBOU);
```

```
mysql> -- Crear índice en la columna DATCOR de la tabla ESDEVENIMENT
mysql> CREATE INDEX idx_ESD_DATCOR ON ESDEVENIMENT (DATCOR);
Query OK, 0 rows affected (0.66 sec)
Records: 0 Duplicates: 0 Warnings: 0

mysql> -- Crear índice en la columna ANYBOU de la tabla TORO
mysql> CREATE INDEX idx_TORO_ANYBOU ON TORO (ANYBOU);
Query OK, 0 rows affected (22.69 sec)
Records: 0 Duplicates: 0 Warnings: 0
```

Tras la creación de estos índices, hacemos otra vez el EXPLAIN UPDATE y observamos lo siguiente:

```
mysql> -- ANÁLISIS
mysql> EXPLAIN UPDATE TORO TOR
-> JOIN R_ESDEVENIMENT_TORO ET
-> ON TOR.IDTORO = ET.IDTORO
-> JOIN ESDEVENIMENT ESD
-> ON ET.IDEESD = ESD.IDEESD
-> SET ANYBOU = DATE_SUB(ESD.DATCOR, INTERVAL 2 YEAR) - INTERVAL 6 MONTH
-> WHERE DATEDIFF(ESD.DATCOR, TOR.ANYBOU) < 912;
```

id	select_type	table	partitions	type	possible_keys	key	key_len	ref	rows	filtered	Extra
1	SIMPLE	ESD	NULL	index	PRIMARY	idx_ESD_DATCOR	3	NULL	122130	100.00	Using index
1	SIMPLE	ET	NULL	ref	PRIMARY, idx_ideesd	idx_ideesd	4	defbd.ESD.ideesd	75	100.00	Using index
1	UPDATE	TOR	NULL	eq_ref	PRIMARY	PRIMARY	4	defbd.ET.idtoro	1	100.00	Using where

3 rows in set, 1 warning (0.02 sec)

Tras la creación de los índices, se puede apreciar en la columna *rows* como se ha pasado de una estimación de 5.808.702 filas recorridas a solamente **122.130 filas recorridas**. Esto supone una mejora en porcentaje del $100 \cdot (1 - (5808702/122130)) = 100 \cdot (1 - 0,021) = 100 \cdot 0,9789 = 97,89\%$.

Esto demuestra una enorme mejora que optimizará el rendimiento de la consulta anterior.

PostgreSQL

En este punto, se plantea el siguiente contexto: una vez que se tiene el gestor de bases de datos MySQL bien trabajado y en producción, el responsable técnico cree necesario cambiar de gestor y usar PostgreSQL, ya que considera que presenta más ventajas de cara a futuros retos que le han sido marcados desde la dirección. Para llevar a cabo esta tarea, el equipo de arquitectos propone usar un *Foreign Data Wrapper* (FDW) mientras se estudia cómo se pasará los datos de un gestor a otro.

Para las tareas relacionadas con PostgreSQL, se ha optado por cambiar de sistema operativo y usar Linux. Esta decisión se debe a que la facilidad de uso que presenta el bash de Linux (más concretamente, Ubuntu) facilitará las tareas a realizar, especialmente las relacionadas con la configuración del FDW, que se sabe que en Windows presenta más dificultades.

Existen dos opciones para usar Linux (distribución Ubuntu):

- **Habilitar el WSL en Windows:** Subsistema de Linux en Windows, desde el cual podemos usar PostgreSQL siguiendo las pautas detalladas en la documentación de Microsoft [\[7\]](#).
- **Hacer una partición del disco** e instalar el sistema operativo, que convivirá con Windows en el mismo equipo, pero nos permitirá una experiencia completa de uso del SSOO.

En nuestro caso, optaremos por la segunda opción, ya que se entiende que es mejor trabajar sobre el propio sistema operativo en lugar de hacerlo sobre el subsistema habilitado en Windows.

Análisis y explicación de las pautas de configuración de FDW

Un *Foreign Data Wrapper* (FDW) es una herramienta que permite a un sistema de gestión de bases de datos interactuar con datos externos como si fueran parte de la base de datos local. PostgreSQL soporta FDWs, lo que le permite conectarse y consultar datos desde otras bases de datos o fuentes externas. Por ejemplo, puedes configurar un FDW para acceder a datos en una base de datos MySQL, Oracle, o incluso para conectarte a un servicio web que proporciona datos.

Todos los comandos usados en el desarrollo de las tareas de este apartado, de PostgreSQL, se desarrollan en el archivo “*PostgreSQL_code.sql*”.

En primer lugar, ya que, vamos a usar Linux, nos aseguraremos de que el bash y los comandos están actualizados.

```
Unpacking libstdc++-7-dev:amd64 (7.5.0-3ubuntu1~18.04) ...
Selecting previously unselected package g++-7.
Preparing to unpack .../07-g++-7_7.5.0-3ubuntu1~18.04_amd64.deb ...
Unpacking g++-7 (7.5.0-3ubuntu1~18.04) ...
Selecting previously unselected package g++.
Preparing to unpack .../08-g++_4%3a7.4.0-1ubuntu2.3_amd64.deb ...
Unpacking g++ (4:7.4.0-1ubuntu2.3) ...
Selecting previously unselected package libdpkg-perl.
Preparing to unpack .../09-libdpkg-perl_1.19.0.5ubuntu2.4_all.deb ...
Unpacking libdpkg-perl (1.19.0.5ubuntu2.4) ...
Selecting previously unselected package dpkg-dev.
Preparing to unpack .../10-dpkg-dev_1.19.0.5ubuntu2.4_all.deb ...
Unpacking dpkg-dev (1.19.0.5ubuntu2.4) ...
Progress: [ 18%] [#####]
```

Una vez nuestro entorno está configurado, pasamos a configurar un Foreign Data Wrapper.

Las pautas de configuración del FDW para MySQL a PostgreSQL son las siguientes:

1. Instalación de FDW en PostgreSQL

Para instalar el módulo Foreign Data Wrapper para MySQL en PostgreSQL versión 16, ejecutaremos el siguiente comando:

```
-- 1. Instalación de FDW en PostgreSQL
sudo apt-get install postgresql-16-mysql-fdw
```

```
mario@LaptopMario: ~$ sudo apt-get install postgresql-16-mysql-fdw
Leyendo lista de paquetes... Hecho
Creando árbol de dependencias... Hecho
Leyendo la información de estado... Hecho
postgresql-16-mysql-fdw ya está en su versión más reciente (2.9.1-2.pgdg22.04+1).
0 actualizados, 0 nuevos se instalarán, 0 para eliminar y 198 no actualizados.
mario@LaptopMario: ~$
```

2. Creación de un servidor FDW en PostgreSQL

Ahora que hemos instalado `mysql_fdw`, es necesario cargar esto en el gestor de PostgreSQL y realizar unas configuraciones para crear un servidor que apunte a la base de datos MySQL. Debemos tener en cuenta que el host es, al igual que en el apartado de MySQL, "localhost". También será necesario saber el puerto en el que se encuentra el servidor MySQL, y para ello usaremos el siguiente comando:

```
SHOW GLOBAL VARIABLES LIKE 'PORT';
```

```
mysql> SHOW GLOBAL VARIABLES LIKE 'PORT';
+-----+-----+
| Variable_name | Value |
+-----+-----+
| port          | 3306  |
+-----+-----+
```

Esto nos muestra que el puerto en el que está el servidor MySQL es el 3306. A continuación, accederemos a PostgreSQL y crearemos una extensión `mysql_fdw` mediante el siguiente código:

```
CREATE EXTENSION mysql_fdw;
```

```
postgres=# CREATE EXTENSION mysql_fdw;
CREATE EXTENSION
```

Por último, crearemos un servidor `mysql_server` y le asignaremos la extensión del Foreign Data Wrapper “`mysql_fdw`”, añadiendo las especificaciones de `host=“localhost”` y `port=3306`.

```
CREATE SERVER mysql_server
FOREIGN DATA WRAPPER mysql_fdw
OPTIONS (host 'localhost', port '3306');
```

```
postgres=# CREATE SERVER mysql_server
FOREIGN DATA WRAPPER mysql_fdw
OPTIONS (host 'localhost', port '3306');
CREATE SERVER
```

El servidor se ha creado correctamente.

3. Creación de un usuario de mapeo en PostgreSQL

El próximo paso será la creación de un usuario de mapeo. Este usuario debe poder identificarse en el servidor de MySQL desde PostgreSQL, de forma que se le dará la información del usuario MIGBD de MySQL. De esta forma podrá acceder al servidor.

```
-- 3. Creación de un usuario de mapeo en PostgreSQL
CREATE USER MAPPING FOR postgres
SERVER mysql_server
OPTIONS (username 'MIGBD', password 'migbd_user_paswd');
```

```
postgres=# CREATE USER MAPPING FOR postgres
SERVER mysql_server
OPTIONS (username 'MIGBD', password 'migbd_user_paswd');
CREATE USER MAPPING
postgres=#
```

4. Importación de las tablas de MySQL

Por último, crearemos una serie de tablas que actuarán como correspondencia de las tablas de la base de datos original DEFBD en el gestor MySQL. Para cada tabla de la base de datos en MySQL, se creará una tabla de representación con la sintaxis “`CREATE FOREIGN TABLE`”. Además, al igual que en MySQL asignamos el tablespace a nivel de tablas añadiendo “`TABLESPACE my_tablespace`” después de cada `CREATE`, ahora debemos asignar a estas tablas el servidor creado en el paso 2, añadiendo “`SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'UBICACION');`” después de cada `CREATE`.

Teniendo todas estas consideraciones en cuenta, las tablas implementadas son las siguientes:

```
-- 4.Importación de las tablas de MySQL
-- Tabla APODERAT
CREATE FOREIGN TABLE APODERAT (
  ideapo VARCHAR(10) -- Identificador del apoderado (màxim 10 dígit)
```

```

) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'APODERAT');

-- Tabla UBICACION
CREATE FOREIGN TABLE UBICACION (
    ideubi INTEGER,          -- Identificador de la ubicación
    nompai VARCHAR(40),      -- Nombre del país
    nomciu VARCHAR(50)       -- Nombre de la ciutat
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'UBICACION');

-- Tabla RAMADERIA
CREATE FOREIGN TABLE RAMADERIA (
    cifram VARCHAR(3),       -- CIF de la ramaderia
    nomram VARCHAR(50)       -- Nombre de la ramaderia
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'RAMADERIA');

-- Tabla FERIA
CREATE FOREIGN TABLE FERIA (
    fircor VARCHAR(50)       -- Nombre de la festa o feria
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'FERIA');

-- Tabla PERSONA
CREATE FOREIGN TABLE PERSONA (
    ideper VARCHAR(10),      -- Identificador de la persona
    ideubi INTEGER,          -- id de la ubicacion
    nompe VARCHAR(50),       -- Nombre de la persona
    colpe VARCHAR(50),       -- Primer apellido de la persona
    co2pe VARCHAR(50),       -- Segundo apellido de la persona
    maiper VARCHAR(100),     -- Mail de la persona
    dirper VARCHAR(100)      -- Dirección de la persona
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'PERSONA');

-- Tabla PLAZA
CREATE FOREIGN TABLE PLAZA (
    nompla VARCHAR(50),      -- Nombre de la plaza
    ideubi INT,              -- id de la ubicacion
    anypla DATE,             -- Año de construcción de la plaza
    locpla INT,              -- Número de asientos de la plaza
    tippla BOOLEAN,          -- Si la plaza es fija o movil
    estpla TEXT,             -- Estilos predominantes en la construcción
    muspla BOOLEAN           -- Indica si la plaza contiene un museo
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'PLAZA');

-- Tabla TORERO
CREATE FOREIGN TABLE TORERO (
    idetor VARCHAR(10),      -- Identificador del torero (máximo 10 dígitos)
    ideapo VARCHAR(10)       -- id del apoderado del torero
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'TORERO');

-- Tabla ACTUACION

```



```

CREATE FOREIGN TABLE ACTUACION (
    ideact INT,           -- Identificador de la actuación
    idetor VARCHAR(10),   -- id del torero que actua
    nompla VARCHAR(50),   -- nombre de la plaza de la actuacion
    datcor DATE           -- Fecha de la actuación
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'ACTUACION');

-- Tabla TORO
CREATE FOREIGN TABLE TORO (
    idtoro INT,           -- Identificador del toro
    cifram VARCHAR(3),    -- ramaderia a la que pertenece el toro
    nombou VARCHAR(50),   -- Nombre o identificación del toro
    anybou DATE,          -- Año de nacimiento del toro
    pesbou DECIMAL(10,2)  -- Peso del toro
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'TORO');

-- Tabla ESDEVENIMENT
CREATE FOREIGN TABLE ESDEVENIMENT (
    ideesd INT,           -- Identificador del esdeveniment
    fircor VARCHAR(50),   -- feria en la que se celebren el esdeveniment
    nompla VARCHAR(50),   -- nombre de la plaza del esdeveniment
    datcor DATE           -- Fecha del esdeveniment
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'ESDEVENIMENT');

-- Tabla R_ESDEVENIMENT_TORO
CREATE FOREIGN TABLE R_ESDEVENIMENT_TORO (
    idtoro INT,           -- id del toro
    ideesd INT            -- id del esdeveniment
) SERVER mysql_server OPTIONS (dbname 'DEFBD', table_name 'R_ESDEVENIMENT_TORO');

```

Una vez hecho esto, ya podemos acceder desde PostgreSQL a los datos de la base de datos de MySQL usando un FDW. Para comprobarlo, hagamos un par de consultas:

```

-- Comprobación con consultas
-- Primeros 10 esdeveniments por fecha
SELECT *
FROM ESDEVENIMENT
ORDER BY datcor DESC
LIMIT 10;

-- Id del torero con apoderado 00002777V
SELECT TOR.idetor AS ID
FROM TORERO TOR
WHERE TOR.ideapo = '00002777V';

-- Cantidad de filas en apoderat
SELECT COUNT(*) FROM APODERAT;

```



```
postgres=# SELECT * FROM ESDEVENIMENT ORDER BY datcor DESC LIMIT 10;
 idsdv | datcor | nompla | fircor
-----+-----+-----+-----
 125443 | 2013-12-30 | Ibiza | Feria de Ibiza
 125442 | 2013-12-28 | Monumental de Playas de Tijuana | Feria de Tijuana
 125441 | 2013-12-25 | La Pradera | Feria de Sogamoso
 125440 | 2013-12-25 | La Pradera | Feria de Setúbal
 125438 | 2013-12-24 | Coliseo El Llano de Tovar | Feria de Pontevedra
 125439 | 2013-12-24 | Coliseo El Llano de Tovar | Feria de Tarragona
 125436 | 2013-12-23 | Monumental Señor de los Milagros | Feria de Girón
 125437 | 2013-12-23 | Monumental Señor de los Milagros | Feria de Villafranca de la Sierra
 125435 | 2013-12-22 | Los Remolinos | Feria de Tortosa
 125434 | 2013-12-21 | Cuenca | Feria de Cuenca
(10 filas)
```

```
postgres=# SELECT TOR.idetor AS ID
FROM TORERO TOR
WHERE TOR.ideapo = '00002777V';
 id
-----
 94558054V
(1 fila)
```

```
postgres=# SELECT COUNT(*) FROM APODERAT;
 count
-----
 132189
(1 fila)
```

Como se puede ver, se ha podido hacer diferentes consultas de distinto tipo en varias tablas de la base de datos de MySQL.

Creación de un espacio de almacenamiento llamado Data

La creación de un espacio de almacenamiento se refiere a la creación de un tablespace de nombre Data. Para ello se necesita gozar de permisos, ya que de lo contrario saldrá un aviso como el siguiente:

```
ERROR: no se pudo definir los permisos del directorio «/tablespace»: Operación no permitida
postgres=#
```

Los permisos serán otorgados mediante el siguiente código:

```
-- Manejo de permisos
sudo chown postgres:postgres /tablespace
ls -ld /tablespace
```

Una vez hecho esto, se permitirá la creación del tablespace:

```
-- Creación tablespace
CREATE TABLESPACE Data
OWNER postgres
```

```
LOCATION '/tablespace';
```

```
postgres=# CREATE TABLESPACE Data
OWNER postgres
LOCATION '/tablespace';
CREATE TABLESPACE
```

Creación de una base de datos FET

A continuación, se creará una base de datos llamada FET (Federación Española de Tauromaquia). Se deberá asignar a esta base de datos el tablespace creado en el apartado anterior, que a diferencia del gestor MySQL, PostgreSQL permite la asignación del tablespace en el momento de la creación de la base de datos. Esto asegurará que se guarde en el tablespace Data con el mismo conjunto de caracteres que la base de datos original de MySQL:

```
-- CREACIÓN DE LA BD FET
CREATE DATABASE FET
WITH ENCODING 'UTF8'
LC_COLLATE 'es_ES.utf-8'
TEMPLATE template0
TABLESPACE data;
```

```
postgres=# CREATE DATABASE FET
postgres=# WITH ENCODING 'UTF8'
postgres=# LC_COLLATE 'es_ES.utf-8'
postgres=# TEMPLATE template0
postgres=# TABLESPACE Data;
CREATE DATABASE
postgres=#
```

La base de datos se ha creado correctamente.

Creación de un esquema temp dentro de la base de datos anterior

Para crear un esquema de nombre Temp en la base de datos primero debemos acceder a esta, para posteriormente crearlo y que este se cree dentro de la BD. Nos conectaremos mediante el comando `\c [nombre_bd];` y ejecutaremos el siguiente comando:

```
-- Creación del esquema temp
CREATE SCHEMA Temp;
```

El esquema creado actuará como contenedor lógico para las tablas y los datos contenidos en ellas. A continuación, debemos realizar el siguiente comando para asegurar de que en un futuro las operaciones se realicen dentro del esquema Temp creado anteriormente:

```
-- Operaciones dentro del esquema
```

```
SET SEARCH_PATH = Temp;
```

Este comando asegura que en un futuro las operaciones se harán en el esquema gracias a asignar este esquema a SEARCH_PATH. Por último, comprobaremos si SEARCH_PATH se ha establecido de forma correcta mediante el comando siguiente:

```
-- Comprobación de path establecido correctamente
```

```
SHOW SEARCH_PATH;
```

```
postgres=# \c fet;
You are now connected to database "fet" as user "postgres".
fet=# CREATE SCHEMA Temp;
CREATE SCHEMA
fet=# SET SEARCH_PATH = Temp;
SET
fet=# SHOW SEARCH_PATH;
 search_path
-----
 temp
(1 row)

fet=# |
```

Se puede apreciar que la creación y configuración del esquema son correctos.

Creación del usuario UFDW

Para crear un usuario de nombre ufdw ejecutaremos el comando CREATE USER, y estableceremos, en este caso, que la contraseña asociada a este usuario será 'ufdw_user_passwd'.

```
-- Creación del usuario UFDW
```

```
CREATE USER UFDW WITH PASSWORD 'ufdw_user_passwd';
```

```
postgres=# -- Creación del usuario UFDW
postgres=# CREATE USER UFDW WITH PASSWORD 'ufdw_user_passwd';
```

A continuación, tendremos que gestionar los permisos del usuario, otorgándole los siguientes permisos:

1. Permiso para conectarse a la base de datos.
2. Permiso para creación y uso del schema Temp.
3. Permisos para insertar y seleccionar en todas las tablas del schema Temp

Esto se llevará a cabo con el código SQL detallado a continuación.

```
-- Otorgar permisos de conexión
```

```
GRANT CONNECT ON DATABASE FET TO UFDW;
```

```
-- Creación y uso del esquema
```

```
GRANT CREATE, USAGE ON SCHEMA Temp TO UFDW;
```

```
-- Otorgar permisos para seleccionar e insertar en todas las tablas del esquema
```

```
GRANT INSERT, SELECT ON ALL TABLES IN SCHEMA Temp TO UFDW;
```

```
postgres=# GRANT CONNECT ON DATABASE FET TO UFDW;
GRANT
postgres=# GRANT CREATE, USAGE ON SCHEMA Temp TO UFDW;
GRANT
postgres=# GRANT INSERT, SELECT ON ALL TABLES IN SCHEMA Temp TO UFDW;
GRANT
```

Ahora que el usuario ha sido creado, se creará con el usuario UFDW una serie de tablas, y se insertará en ellas el contenido materializado de los datos que lee el Foreign Data Wrapper desde PostgreSQL hasta el gestor MySQL.

Realización de sentencias SQL

En este apartado debemos realizar unas consultas mediante sentencias SQL. Estas sentencias se lanzarán desde el gestor PostgreSQL hacia el gestor MySQL a través del FDW. Las sentencias son las detalladas a continuación.

Media de peso de toros lidiados por año.

En esta consulta se pide extraer la media de peso de los toros agrupados por año de lidia. Es decir, se debe saber, para cada año del cual se tiene algún registro, cual era la media de peso de los toros que se lidiaron ese año. Para ordenar el resultado, añadiremos una ordenación con ORDER BY, que ordenará por año de lidia de forma descendiente. Cabe mencionar que el uso de la función EXTRACT se debe a que PostgreSQL, a diferencia de MySQL, no contiene la función YEAR() [\[8\]](#) para extraer el año de un atributo de tipo DATE. Por ello, se debe usar la función EXTRACT con el fin de extraer esa parte del atributo datcor. La sintaxis de EXTRACT es “EXTRACT(field FROM source)” [\[9\]](#). Por otro lado, se usa la función AVG() [\[10\]](#) para calcular el promedio de los pesos, ya que esta función realiza un cálculo de una media aritmética.

```
-- Media de peso de toros lidiados en españa y agrupados por año
SELECT EXTRACT(YEAR FROM E.DATCOR) AS AÑO, AVG(T.PESBOU) AS PESO
FROM ESDEVENIMENT E
    JOIN R_ESDEVENIMENT_TORO ET
    ON E.IDEESD = ET.IDEESD
    JOIN TORO T
    ON ET.IDTORO = T.IDTORO
GROUP BY AÑO
ORDER BY AÑO DESC; -- LIMIT 25 (quitar ; del ORDER BY);
```

El resultado obtenido en la consulta es el siguiente:

```
1653 | 635.7299642079053844
1652 | 640.0897276199966783
1651 | 634.0244125846276384
1650 | 631.7976302573760201
1649 | 632.6518082788671024
1648 | 639.5762077294685990
1647 | 640.5422366210703144
1646 | 636.8805695544356451
1645 | 638.7865455689234882
1644 | 635.0841376730265619
1643 | 635.5369543650793651
1642 | 635.1058757355486327
1641 | 637.8117669753086420
1640 | 631.3142846322805423
1639 | 634.0190070166089351
1638 | 634.5583864118895966
1637 | 633.8400910991636798
1636 | 637.0535300925925926
1635 | 639.4742527615334633
1634 | 639.7997343334896078
1633 | 635.5919442418348326
1632 | 637.8991004133236081
1631 | 634.3174189814814815
1630 | 639.5147334147334147
1629 | 637.5253757942042461
1628 | 636.8076153907687385
1627 | 633.4009197667734253
1626 | 636.8785014005602241
1625 | 636.0232949651554303
1624 | 635.0180528691166989
1623 | 637.7606220939554273
1622 | 637.0918226760677527
1621 | 634.9850545566313201
1620 | 634.6072617561381606
1619 | 636.4753215558654889
1618 | 633.2850241545893720
1617 | 635.9941176470588235
1616 | 643.1866682700016033
1615 | 629.6629935720844812
1614 | 633.1755588187306249
1613 | 637.6247744263985563
1612 | 636.3033913431503793
1611 | 641.4210401891252955
(403 filas)

(END)
```

Como se puede ver, el resultado se corta debido a que se tiene información sobre la media de peso de los toros en más de 400 años. Esto no es un error en la consulta, que pese a ser correcta, su resultado se corta debido a la propia naturaleza de la consola de postgres usada en Ubuntu, que además sufre un tipo de bloqueo a causa de esto y no muestra el *prompt* al acabar la consulta. Teniendo en cuenta, además, que si hay más de 400 filas en el resultado no se puede mostrar el resultado en una única captura, modificaremos la consulta y añadiremos una limitación, haciendo que se puedan ver los 25 primeros años. De esta forma, se podrá ver en la misma captura como el resultado mostrado corresponde a la consulta descrita anteriormente. Además, en cuanto se quiera se puede quitar esta limitación y ver el resultado entero si la terminal en la que se ejecuta la consulta lo permite.

Añadiendo el límite mencionado y ordenando los años de forma que salgan primero los años más recientes (esto ya se pensó en la primera consulta), se tiene lo siguiente:

```

postgres=# SELECT EXTRACT(YEAR FROM E.DATCOR) AS AÑO, AVG(T.PESBOU) AS PESO
FROM ESDEVENIMENT E
      JOIN R_ESDEVENIMENT_TORO ET
      ON E.IDEESD = ET.IDEESD
      JOIN TORO T
      ON ET.IDTORO = T.IDTORO
GROUP BY AÑO
ORDER BY AÑO DESC LIMIT 25;
año |          peso
-----+-----
2013 | 637.25606060606061
2012 | 635.9373033163882837
2011 | 636.4346502057613169
2010 | 634.2390643274853801
2009 | 634.1906556447037847
2008 | 635.5252242650324185
2007 | 631.2888486312399356
2006 | 636.3291556198280710
2005 | 637.4501973628957756
2004 | 632.7402959728287239
2003 | 634.5933625091174325
2002 | 642.8280818131564400
2001 | 639.6580108198085726
2000 | 638.6556852257071075
1999 | 635.3692155407077456
1998 | 633.8276537876969842
1997 | 636.4471127041019514
1996 | 634.6129411764705882
1995 | 639.0461760461760462
1994 | 640.5622895622895623
1993 | 636.3119324014846403
1992 | 637.3668797766019988
1991 | 634.8130737575182020
1990 | 641.4717265445316837
1989 | 639.5832955404383976
(25 filas)

```

Ahora se puede apreciar que el resultado de la consulta es el correcto, aunque solo se muestren las primeras 25 filas.

Número de toros lidiados en el estado español

En este caso se nos pide extraer el número de toros lidiados en el estado español. La consulta será del tipo `SELECT COUNT()`, y el resultado de la consulta será una tabla con una fila y una columna, es decir, una sola celda con el número contado por la función `COUNT()`.

Teniendo en cuenta que se quieren contar los toros lidiados, se contará todos los toros con identificador diferente, ya que de lo contrario se podría contar el mismo toro más de una vez y el resultado mostraría más toros de los realmente lidiados como consecuencia. Con el objetivo de obtener los identificadores de todos los toros, se parte desde la tabla `ESDEVENIMENT` y se irá hasta la tabla `TORO` pasando por su tabla intermedia `R_ESDEVENIMENT_TORO`. Además, se irá a la tabla `UBICACIÓ` desde la propia tabla `ESDEVENIMENT`, y pasando necesariamente por la tabla `PLAZA`, para poder establecer la condición de que el nombre de la ubicación (`nompai`) sea España, ya que se pide obtener el número de toros lidiados en el estado español. Esta condición se establecerá en una `AND` del `ON` del `JOIN` con la tabla `UBICACIÓ`, ya que, aunque puede añadirse en el `WHERE` y obtenerse el mismo resultado, si se hiciera esto las tablas intermedias serían mayores y la consulta tendría un mayor coste al tener que procesar más filas.

Con todo esto, la consulta ejecutada es la siguiente:

```
-- Número de toros lidiados en España
SELECT COUNT(DISTINCT T.IDTORO) AS NUM_TOROS
FROM ESDEVENIMENT E
  JOIN R_ESDEVENIMENT_TORO ET
    ON E.IDEESD = ET.IDEESD
  JOIN TORO T
    ON ET.IDTORO = T.IDTORO
  JOIN PLAZA PL
    ON E.NOMPLA = PL.NOMPLA
  JOIN UBICACION UBI
    ON PL.IDEUBI = UBI.IDEUBI AND UBI.NOMPAI = 'España';
```

```
postgres=# -- Número de toros lidiados en España
SELECT COUNT(DISTINCT T.IDTORO) AS NUM_TOROS
FROM ESDEVENIMENT E
  JOIN R_ESDEVENIMENT_TORO ET
    ON E.IDEESD = ET.IDEESD
  JOIN TORO T
    ON ET.IDTORO = T.IDTORO
  JOIN PLAZA PL
    ON E.NOMPLA = PL.NOMPLA
  JOIN UBICACION UBI
    ON PL.IDEUBI = UBI.IDEUBI AND UBI.NOMPAI = 'España';
 num_toros
-----
  3428188
(1 fila)
```

La consulta muestra que hay un total de 3.428.188 toros lidiados en España.

Se ha podido comprobar que las dos consultas se lanzan correctamente desde PostgreSQL hacia MySQL mediante el Foreign Data Wrapper, mostrando en ambas el resultado esperado.

Oracle

En este punto se plantea el siguiente contexto: el equipo de arquitectos ahora considera hacer unas pruebas con el gestor de bases de datos Oracle, y debido a que este es un gestor comercial, optan por hacer dichas pruebas en la versión gratuita que este gestor ofrece. Para ello, usan su imagen Docker en la versión 19.3.0 o superior, usando su arquitectura SingleInstance.

En este caso, para las tareas relacionadas con el gestor Oracle se ha optado por seguir usando el sistema operativo Linux (distribución Ubuntu al igual que en el apartado anterior con PostgreSQL).

Configuración del gestor

Como primer paso debemos configurar el gestor de forma que, siempre que sea posible, se cumplan las siguientes condiciones:

1. Su identificador sea FET (Federación Español de Tauromaquia).
2. El nombre del pluggable database sea FETPDB1
3. El SGA debe ser del 50% del tamaño de la memoria virtual
4. El PGA debe ser del 15% del tamaño de la memoria virtual
5. Se debe activar el archiver

Las pautas para configurar el gestor siguiendo estas condiciones se detallarán a continuación.

Se instalará la imagen Docker de Oracle, usando la versión 19.3.0 Standard 2 y la arquitectura SingleInstance. Para ello, se siguen los pasos detallados en el repositorio Github del enlace proporcionado en el enunciado de la práctica. Estos pasos son:

- **Se clona el repositorio** de Oracle Docker Images ya que contiene los archivos Dockerfile necesarios para la construcción de la imagen. También contiene los scripts necesarios para ello. Para clonar el repositorio se hará uso del comando git clone.

```
-- Clonación del repositorio
git clone https://github.com/oracle/docker-images.git
```

```
mario@LaptopMario:~$ git clone https://github.com/oracle/docker-images.git
Clonando en 'docker-images'...
remote: Enumerating objects: 17510, done.
remote: Counting objects: 100% (2044/2044), done.
remote: Compressing objects: 100% (389/389), done.
remote: Total 17510 (delta 1779), reused 1753 (delta 1643), pack-reused 15466
Recibiendo objetos: 100% (17510/17510), 15.08 MiB | 4.10 MiB/s, listo.
Resolviendo deltas: 100% (10337/10337), listo.
mario@LaptopMario:~$ cd docker-images/OracleDatabase/SingleInstance/dockerfiles
```

Hecho esto, para continuar, es necesario **descargarse el software de Oracle Database**, ya que antes de la construcción de la imagen, es necesaria la descarga manual de este software. Se descargará Oracle Database 19c, en la versión 19.3.0, desde la propia web de Oracle, y se colocará

- A continuación, se construye la imagen Docker mediante el script "buildContainerImage.sh". Se construirá en base a la mencionada versión 19.3.0 Standard. Para la construcción de la imagen Docker se usará al super usuario sudo ("super user do", equivalente a ejecutar como administrador en el ssou Windows) junto con el comando ./ para ejecutar el script:

```
-- Construcción de La imagen
sudo ./buildContainerImage.sh -v 19.3.0 -s
```

Se puede comprobar si la imagen se ha obtenido correctamente mediante el comando docker images ejecutado nuevamente por el super usuario **sudo**:

```
-- Comprobar que La imagen se ha obtenido
sudo docker images
```

```
mario@LaptopMario :~/docker-images/OracleDatabase/SingleInstance/dockerfiles$ sudo docker images
REPOSITORY          TAG                 IMAGE ID            CREATED             SIZE
oracle/database      19.3.0-se2         9348c8f2abca       About a minute ago  6.80GB
```

Una vez que el contenedor esté montado, procederemos a ejecutar el contenedor de Oracle Database, siguiendo los parámetros mencionados al inicio del apartado. Para cumplir con estos requisitos especificados, necesitaremos pasar variables de entorno al contenedor para configurar el DBID (identificador de la base de datos), el nombre de la base de datos pluggable (PDB), el tamaño de la SGA (Server Global Area) y la PGA (Program Global Area), y habilitar el archiver.

Oracle Database no permite cambiar directamente el DBID o el nombre de la PDB mediante variables de entorno al iniciar el contenedor, pero podemos configurar el SID (identificador del sistema) y luego crear una PDB con el nombre deseado después de la instalación. Por otro lado, para la server global area SGA y program global area PGA podemos configurar la memoria total asignada a Oracle, y Oracle ajustará automáticamente la SGA y la PGA dentro de esos límites. Para habilitar el archivador, podemos configurar el modo de archivado de la base de datos.

En este caso, se requiere ajustar el contenedor para configurar la SGA y PGA como porcentajes de la memoria virtual del sistema (50% para la SGA y 15% para la PGA). Por lo tanto, primero necesitaremos calcular la memoria virtual de nuestro sistema. Utilizaremos el siguiente comando, que recopila información sobre la memoria física y la memoria de intercambio utilizando el comando "free", luego utilizaremos "awk" para extraer los valores correspondientes y, finalmente, imprimiremos la suma total de la memoria física y la memoria de intercambio en megabytes, obteniendo de esta manera una representación de la capacidad total en megabytes de la memoria virtual disponible.

El comando usado es el siguiente:

```
-- Representación en Megabytes
free -m | awk '/^Memoria:/ {mem_total=$2} /^Swap:/ {swap_total=$2} END {print "Memoria Virtual Total en MB: " mem_total+swap_total}'
```

```
mario@LaptopMario :~$ free -m | awk '/^Memoria:/ {mem_total=$2} /^Swap:/ {swap_total=$2} END {print "Memoria Virtual Total en MB: " mem_total+swap_total}'
Memoria Virtual Total en MB: 15744
```

Se puede ver que el resultado es de **15744MB** de memoria virtual total. Tomando este valor como referencia, puede hacerse un cálculo de cuantos MB suponen un 50% para el SGA y un 15% para la PGA.

50% para el SGA $\rightarrow 15744 * 0.5 = \mathbf{7872 \text{ MB}}$

15% para el PGA $\rightarrow 15744 * 0.15 = \mathbf{2362 \text{ MB}}$

Ahora que sabemos cuál es el tamaño para el SGA y PGA, ejecutaremos un contenedor Docker usando la imagen Oracle/database:19.30-se2, que corresponden a la imagen standard edition 2.

Ejecutaremos el comando siguiente:

```
-- Ejecutar el contenedor docker
sudo docker run -d --name oracleDatabaseC \
```

Y a este comando le añadiremos los parámetros siguientes para configurar la instancia de base de datos de Oracle:

- **sudo docker run -d --name oracleDatabaseC ** \rightarrow Es el nombre del contenedor, que será oracleDatabaseC.
- **oracle/database:19.3.0-se2** \rightarrow Imagen de docker usada para el contenedor anterior, que como ya se ha mencionado, en este caso es 19.3.0-se2.
- **-p 1521:1521 -p 5500:5500 ** \rightarrow Publica el puerto 1521 al mismo puerto que el host (estándar para conexiones a la base de datos Oracle) y el puerto 5500, que se usa para Oracle Enterprise Manager Express.
- **-e ORACLE_SID=FET ** \rightarrow Es el identificador del sitio: FET.
- **-e ORACLE_PDB=FETPDB1 ** \rightarrow Es el identificador de la pluggable database: FETPDB1.
- **-e INIT_SGA_SIZE=7872 ** \rightarrow Se inicializa SGA SIZE con el valor calculado anteriormente.
- **-e INIT_PGA_SIZE=2362 ** \rightarrow Se inicializa PAG SIZE con el valor calculado anteriormente.
- **-e ENABLE_ARCHIVING=true ** \rightarrow Se pone el archiver en modo Enable para activarlo.

Con todo esto, el comando usado es el siguiente:

```
-- Ejecutar el contenedor docker
sudo docker run -d --name oracleDatabaseC \
  -p 1521:1521 -p 5500:5500 \
  -e ORACLE_SID=FET \
  -e ORACLE_PDB=FETPDB1 \
  -e INIT_SGA_SIZE=7872 \
  -e INIT_PGA_SIZE=2362 \
  -e ENABLE_ARCHIVING=true \
```

```
oracle/database:19.3.0-se2
```

```
mario@LaptopMario: /opt/oracle/images/oracle/database/19c/instants/instants/docker/files$ sudo docker run -d --name oracleDatabaseC \
-p 1521:1521 -p 5500:5500 \
-e ORACLE_SID=FET \
-e ORACLE_PDB=FETPDB1 \
-e INIT_SGA_SIZE=7872 \
-e INIT_PGA_SIZE=2362 \
-e ENABLE_ARCHIVING=true \
oracle/database:19.3.0-se2
```

Se puede comprobar la correcta ejecución mediante el comando **sudo docker ps** para ver los dockers en ejecución.

```
-- Ver dockers en ejecución
sudo docker ps
```

```
mario@LaptopMario: /opt/oracle/images/oracle/database/19c/instants/instants/docker/files$ sudo docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
NAME          oracle/database:19.3.0-se2         "/bin/bash -c 'exec ..." 13 seconds ago Up 13 seconds (health: starting) 0.0.0.0:1521->1521/tcp, :::1521->1521/tcp, 0.0.0.0:5500->5500/tcp, :::5500->5500/tcp
cp           oracleDatabaseC
```

Creación del usuario Utest

En este apartado se debe crear un usuario de nombre Utest y otorgarle los permisos necesarios para crear tablas, llenarlas (es decir, insertar datos en ellas) y poder realizar consultas. Para poder hacer esto, instalaremos la interfaz SQLPlus, que se suele usar en el sgbd Oracle para proporcionar un entorno donde ejecutar los comandos necesarios y procesarlos. Con esto, se podrá acceder e interactuar con la información mediante la ejecución de comandos.

La instalación de SQLPlus se hace siguiendo los pasos establecidos en el siguiente enlace encontrado:

<https://www.geeksforgeeks.org/how-to-install-sqlplus-on-linux/>

Una vez realizado esto, ya es posible interactuar con la base de datos en Oracle. Para ello, primero ejecutaremos el siguiente comando, utilizado para abrir un shell interactivo (bash) dentro del contenedor de Docker llamado OracleDatabaseC (el mismo que se ha ejecutado en la sección anterior):

```
-- Creación del usuario
sudo docker exec -it oracleDatabaseC bash
```

```
mario@LaptopMario: /opt/oracle/instants/instants_21_0$ sudo docker exec -it oracle_bd bash
bash-4.2$ sqlplus / as sysdba

SQL*Plus: Release 19.0.0.0.0 - Production on Tue Feb 6 10:51:36 2024
Version 19.3.0.0.0

Copyright (c) 1982, 2019, Oracle. All rights reserved.

Connected to:
Oracle Database 19c Standard Edition 2 Release 19.0.0.0.0 - Production
Version 19.3.0.0.0

SQL>
```

La creación el usuario Utest deberá hacerse usando el usuario SYS (sysdba) para conectarnos, ya que este usuario tiene permisos de administrador al ser el usuario “sudo” o super usuario de Oracle Database. De esta forma, podremos crear el usuario y darle los permisos necesarios.

```
-- Conexión con SYS
sqlplus / as sysdba
```

Hecho esto, se accede a la terminal SQL de Oracle y se procede a cambiar el contexto de la sesión a un contenedor específico dentro de una base de datos contenedor. En este caso, se cambia la sesión al contenedor denominado FETPDB1, definido en el apartado anterior. Para ello, se usará el comando ALTER SESSION. Hecho esto, se crea el usuario Utest con CREATE USER y se le asigna la contraseña "utest_user_passwd".

```
-- Cambiar sesión
ALTER SESSION SET CONTAINER=FETPDB1;
-- Creación Usuario
CREATE USER Utest IDENTIFIED BY utest_user_passwd;
```

```
SQL> ALTER SESSION SET CONTAINER=FETPDB1;
Session altered.
```

```
SQL> CREATE USER Utest IDENTIFIED BY utest_user_passwd;
User created.
```

Ahora que el usuario ha sido creado, podemos otorgar los permisos correspondientes a este usuario. Recordemos que los permisos eran crear tablas, llenarlas (es decir, insertar datos en ellas) y poder realizar consultas. La asignación de permisos se hará con la sentencia GRANT [\[11\]](#).

Pese a que no se manifiesta de forma explícita en los permisos que se le deben otorgar a Utest, es necesario hacer que este usuario pueda conectarse a la base de datos. Por lo general, si se otorga el permiso CREATE SESSION, ya se incluye automáticamente el privilegio de conexión sin necesidad de hacer GRANT CONNECTION. De todas formas, por estar seguros, especificaremos también este último permiso para asegurarnos, ya que de todas formas no tiene ningún efecto negativo ejecutar ese GRANT después de GRANT CONNECTION. Además, añadiremos los GRANT para el resto de permisos: CREATE TABLE, INSERT Y SELECT.

El código usado es el siguiente:

```
-- Permisos
GRANT CREATE SESSION TO Utest;      -- Crear sesión
GRANT CONNECT TO Utest;             -- Conexión
GRANT CREATE TABLE TO Utest;       -- Crear tablas
```

```
SQL> GRANT CONNECT TO Utest;
Grant succeeded.
```

```
SQL> GRANT CREATE SESSION TO Utest;

Grant succeeded.
```

```
SQL> GRANT CREATE TABLE TO Utest;

Grant succeeded.
```

Sin embargo, los permisos para insertar datos en una tabla y hacer consultas sobre ella, es decir, INSERT y SELECT, en Oracle **se deben hacer a nivel de tabla** y no a nivel de base de datos. Esto significa que se puede otorgar el permiso de inserción a un usuario específico en una tabla específica, y lo mismo sucede para las consultas. Esto permite un control más granular sobre quién puede insertar datos en qué tablas.

Teniendo esto en cuenta y sabiendo que en el próximo apartado se pide llevar a cabo una migración de las tablas necesarias para una consulta, el procedimiento usado en este punto ha sido realizar dicha consulta, mirar qué tablas implica la consulta, hacer el código de creación de estas tablas siguiendo la sintaxis de Oracle, y posteriormente, se podrá otorgar permisos sobre estas tablas debido a que ya habrán sido creadas.

Para no extender en exceso el desarrollo de este apartado, el código de creación de las tablas se encuentra en el siguiente apartado. En lo que queda del desarrollo de este, nos ceñiremos a entender que todas las acciones que se detallarán a continuación se han llevado a cabo después de hacer la creación de las tablas.

se crean las tablas

En este punto, con las tablas ya creadas, podemos otorgar permisos al usuario Utest para que pueda insertar datos en tablas y hacer consultas sobre ellas. El código usado para otorgar los permisos a Utest en las tablas es:

```
-- OTORGAR PERMISOS
-- Tabla APODERAT
GRANT INSERT ON APODERAT TO Utest; -- Insertar
GRANT SELECT ON APODERAT TO Utest; -- Consultar
-- Tabla PERSONA
GRANT INSERT ON PERSONA TO Utest; -- Insertar
GRANT SELECT ON PERSONA TO Utest; -- Consultar
-- Tabla UBICACION
GRANT INSERT ON UBICACION TO Utest; -- Insertar
GRANT SELECT ON UBICACION TO Utest; -- Consultar
-- Tabla RAMADERIA
GRANT INSERT ON RAMADERIA TO Utest; -- Insertar
GRANT SELECT ON RAMADERIA TO Utest; -- Consultar
-- Tabla FERIA
GRANT INSERT ON FERIA TO Utest; -- Insertar
GRANT SELECT ON FERIA TO Utest; -- Consultar
-- Tabla PLAZA
```

```

GRANT INSERT ON PLAZA TO Utest; -- Insertar
GRANT SELECT ON PLAZA TO Utest; -- Consultar
-- Tabla TORERO
GRANT INSERT ON TORERO TO Utest; -- Insertar
GRANT SELECT ON TORERO TO Utest; -- Consultar
-- Tabla ACTUACION
GRANT INSERT ON ACTUACION TO Utest; -- Insertar
GRANT SELECT ON ACTUACION TO Utest; -- Consultar
-- Tabla TORO
GRANT INSERT ON TORO TO Utest; -- Insertar
GRANT SELECT ON TORO TO Utest; -- Consultar
-- Tabla ESDEVENIMENT
GRANT INSERT ON ESDEVENIMENT TO Utest; -- Insertar
GRANT SELECT ON ESDEVENIMENT TO Utest; -- Consultar
-- Tabla R_ESDEVENIMENT_TORO
GRANT INSERT ON R_ESDEVENIMENT_TORO TO Utest; -- Insertar
GRANT SELECT ON R_ESDEVENIMENT_TORO TO Utest; -- Consultar

```

```

SQL> GRANT INSERT ON APODERAT TO Utest;
GRANT SELECT ON APODERAT TO Utest;
GRANT INSERT ON PERSONA TO Utest;
GRANT SELECT ON PERSONA TO Utest;

```

...

```

SQL>
Grant succeeded.

SQL>
Grant succeeded.

SQL>
Grant succeeded.

```

(no se muestran todos ya que son un total de 22 GRANTS y se considera innecesario)

Migración de tablas y ejecución de consulta

Migración

Para hacer la migración se deberá crear las tablas sobre las cuales se insertará la información migrada. El código de creación de las tablas con la sintaxis de Oracle se encuentra detallado en el archivo **Oracle_code.sql**, y es el siguiente:

```

-- CREACIÓN DE TABLAS
-- TABLA UBICACION
CREATE TABLE UBICACION (
    ideubi NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY, -- Identificador de ubi-
ción
    nompai VARCHAR2(40) NOT NULL, -- Nombre del país

```

```

    nomciu VARCHAR2(50) NOT NULL, -- Nombre de la ciutat
    CONSTRAINT PK_UBICACION PRIMARY KEY (ideubi)
);

-- TABLA RAMADERIA
CREATE TABLE RAMADERIA (
    cifram CHAR(3), -- CIF de la ramadería
    nomram VARCHAR2(50) NOT NULL, -- Nombre de la ramadería
    CONSTRAINT PK_RAMADERIA PRIMARY KEY (cifram)
);

-- TABLA FERIA
CREATE TABLE FERIA (
    fircor VARCHAR2(50), -- Nombre de la festa o feria
    CONSTRAINT PK_FERIA PRIMARY KEY (fircor)
);

-- TABLA PLAZA
CREATE TABLE PLAZA (
    nompla VARCHAR2(50), -- Nombre de la plaza
    ideubi NUMBER NOT NULL, -- Identificador de ubicación
    anypla DATE NOT NULL, -- Año de construcción de la plaza
    locpla NUMBER NOT NULL, -- Número de asientos de la plaza
    tippla NUMBER(1) CHECK (TIPPLA IN (0, 1)), -- Tipo de plaza, actua como bool
    estpla CLOB, -- Estilos de construcción de la plaza
    muspla NUMBER(1) CHECK (MUSPLA IN (0, 1)), -- Museo de la plaza, actua como bool
    CONSTRAINT PK_PLAZA PRIMARY KEY (nompla),
    CONSTRAINT FK_PLAZA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi)
);

-- TABLA APODERAT
CREATE TABLE APODERAT (
    ideapo VARCHAR2(10), -- Identificador del apoderado (màxim 10 dígit)
    CONSTRAINT PK_APODERAT PRIMARY KEY (IDEAPO)
);

-- TABLA PERSONA
CREATE TABLE PERSONA (
    ideper VARCHAR2(10), -- Identificador de persona
    ideubi NUMBER NOT NULL, -- Identificador de ubicación
    nompe VARCHAR2(50) NOT NULL, -- Nombre de la persona
    colpe VARCHAR2(50) NOT NULL, -- Primer apellido de la persona
    co2pe VARCHAR2(50), -- Segundo apellido de la persona
    maiper VARCHAR2(100), -- Mail (de la persona)
    dirper VARCHAR2(100) NOT NULL, -- Dirección de la persona
    CONSTRAINT PK_PERSONA PRIMARY KEY (ideper),
    CONSTRAINT FK_PERSONA_UBICACION FOREIGN KEY (ideubi) REFERENCES UBICACION(ideubi)
);

-- TABLA TORERO
CREATE TABLE TORERO (

```



```

idetor VARCHAR2(10),          -- Identificador de torero
ideapo VARCHAR2(10) NOT NULL, -- Identificador de apoderado
CONSTRAINT PK_TORERO PRIMARY KEY (idetor),
CONSTRAINT FK_TORERO_APODERAT FOREIGN KEY (ideapo) REFERENCES APODERAT(ideapo)
);

-- TABLA ACTUACION
CREATE TABLE ACTUACION (
  ideact NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY, -- Id de actuación
  idetor VARCHAR2(10) NOT NULL,          -- Identificador de torero
  nompla VARCHAR2(50) NOT NULL,          -- Nombre de la plaza
  datcor DATE NOT NULL,                  -- Fecha de la actuación
  CONSTRAINT PK_ACTUACION PRIMARY KEY (ideact),
  CONSTRAINT FK_ACTUACION_TORERO FOREIGN KEY (idetor) REFERENCES TORERO(idetor),
  CONSTRAINT FK_ACTUACION_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
);

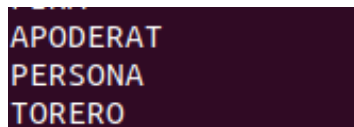
-- TABLA TORO
CREATE TABLE TORO (
  idtoro NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY, -- Identificador de toro
  cifram CHAR(3) NOT NULL,          -- CIF de la ramadería
  nombou VARCHAR2(50) NOT NULL,     -- Nombre o identificación del toro
  anybou DATE NOT NULL,             -- Año de nacimiento del toro
  pesbou NUMBER(10, 2) NOT NULL,    -- Peso del toro
  CONSTRAINT PK_TORO PRIMARY KEY (idtoro),
  CONSTRAINT FK_TORO_RAMADERIA FOREIGN KEY (cifram) REFERENCES RAMADERIA(cifram)
);

-- TABLA ESDEVENIMENT
CREATE TABLE ESDEVENIMENT (
  ideesd NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY, -- Identificador de evento
  fircor VARCHAR2(50) NOT NULL,    -- Clave forana que referencia a FERIA(fircor)
  nompla VARCHAR2(50) NOT NULL,    -- Clave forana que referencia a PLAZA(nompla)
  datcor DATE NOT NULL,            -- Fecha del esdeveniment
  CONSTRAINT PK_ESDEVENIMENT PRIMARY KEY (ideesd),
  CONSTRAINT FK_ESDEVENIMENT_FERIA FOREIGN KEY (fircor) REFERENCES FERIA(fircor),
  CONSTRAINT FK_ESDEVENIMENT_PLAZA FOREIGN KEY (nompla) REFERENCES PLAZA(nompla)
);

-- TABLA R_ESDEVENIMENT_TORO
CREATE TABLE R_ESDEVENIMENT_TORO (
  idtoro NUMBER NOT NULL, -- id del toro que hace referencia a la tabla TORO
  ideesd NUMBER NOT NULL, -- id del esdeveniment que hace referencia a la tabla ES-
  DEVENIMENT
  CONSTRAINT PK_ESDEVENIMENT_TORO PRIMARY KEY (idtoro, ideesd),
  CONSTRAINT FK_TORO FOREIGN KEY (idtoro) REFERENCES TORO(idtoro),
  CONSTRAINT FK_ESDEVENIMENT FOREIGN KEY (ideesd) REFERENCES ESDEVENIMENT(ideesd)
);

```


...



RAMADERIA
APODERAT
PERSONA
TORERO

...

La imagen anterior muestra algunos de los outputs obtenidos al crear las tablas mostradas. Respecto a la sintaxis de los CREATE TABLE, la sintaxis de Oracle es un tanto distinta a la sintaxis de MySQL o de PostgreSQL, gestores usados anteriormente. Se pueden apreciar ciertas diferencias que enumeraremos y explicaremos a continuación:

1. Las cadenas de caracteres son de tipo CHAR cuando sabemos que el atributo que se va a almacenar tiene una longitud fija, como por ejemplo, el atributo cifram en la tabla RAMADERIA. Este hace referencia al CIF que identifica una ganadería, y es un código que siempre es de 3 caracteres.
2. Las cadenas de caracteres son de tipo VARCHAR2 cuando se trata de atributos que pueden contener una longitud variable de información en forma de cadenas de caracteres. Este tipo de datos almacena cadenas de longitud variable, ocupando solamente el espacio necesario sin rellenar con espacios en blanco. Esto nos permitirá optimizar el espacio.
3. Se usó CLOB ("Character Large Object") para almacenar datos de caracteres largos, como texto extenso o grandes bloques de datos de caracteres. Este tipo de variable se ha usado solamente en el atributo *estpla* en la tabla PLAZA (en MySQL era de tipo TEXT).
4. En lugar de INT o INTEGER, se ha usado NUMBER para atributos de tipo numérico, añadiendo también la cláusula NUMBER GENERATED BY DEFAULT ON NULL AS IDENTITY en aquellos atributos de carácter autoincremental.
5. Para los atributos tippla y muspla de la tabla PLAZA, en lugar de booleanos (tipo BOOL en MySQL) se ha usado NUMBER(1) juntamente con restricciones que imponen que estos números pueden ser únicamente 0 o 1, de forma que estos actuarán como booleanos a efectos prácticos.

La migración se hará realizando una copia de la base de datos DEFBD de MySQL y migrando los daots de sus tablas a las tablas creadas anteriormente en Oracle. Para ello, se usará el comando mysqldump con la siguiente sintaxis:

```
sudo mysqldump -u root -p DEFBD > defbdDump.sql
```

Sin embargo, esta migración no se ha podido realizar debido a fallos personales de software en el equipo específico en el que se iba a realizar dicha migración. Se han presentado problemas de compatibilidades entre diferentes versiones del software que forman el entorno necesario para llevar a cabo esta migración. Se ha tratado de usar distintos métodos como el mencionado comando mysqldump, que no

ha dado frutos, y usar SQLDeveloper, que ha generado problemas de compatibilidad y colisiones con otros programas software.

La migración queda incompleta, pero aún así, pese a que no se pueda ejecutar en el entorno de Oracle con los datos migrado de MySQL, se puede obtener la consulta que se pretendía obtener.

Consulta

En esta consulta, se pide la sentencia SQL para poder obtener el nombre y ubicación (se seleccionará todo sobre la ubicación) de todas las plazas a lo largo de la historia, junto con el número de toros realizados en cada una de estas plazas. Para obtener el número de toros usaremos la función COUNT(), que contará identificadores de toros. No se usará "COUNT(DISTINCT ET.IDTORO)" porque no se quiere obtener los toros distintos que hayan sido toreados, sino el número de toros, por tanto, al no usar DISTINCT permitimos la posibilidad de que un mismo toro haya sido toreado en la misma plaza en varias ocasiones. Por último, respecto al COUNT, cabe mencionar que contaremos los identificadores de los toros desde la tabla R_ESDEVENIMENT_TORO ya que esta tabla contiene los toros asociados a algún esdeveniment. Sería ineficiente (por usar una operación de JOIN más) entrar a la tabla TORO cuando ya se tienen los id en la tabla R-ESDEVENIMENT_TORO.

Partiremos desde la tabla PLAZA y haremos 3 operaciones de JOIN:

- JOIN a la tabla UBICACIÓN desde PLAZA para poder acceder a los atributos ideubi, nomciu y nompai y poder seleccionar, para cada ubicación, el identificador, nombre de ciudad y nombre de país respectivamente.
- JOIN a la tabla ESDEVENIMENT desde PLAZA para poder acceder posteriormente a la tabla R_ESDEVENIMENT_TORO con el objetivo de obtener los identificadores de los toros y poder contar.
- JOIN a la tabla R_ESDEVENIMENT_TORO desde ESDEVENIMENT para acceder a los identificadores de los toros, contenidos en el atributo *idtoro* de la tabla.

Con todo esto, la consulta es la siguiente:

```
-- Consulta ejercicio 22
SELECT P.NOMPLA AS "NOMBRE DE LA PLAZA",
       UBI.NOMPAI AS "PAIS",
       UBI.NOMCIU AS "CIUDAD",
       COUNT(ET.IDTORO) AS "NUMERO DE TOREADOS"
FROM PLAZA P
      -- Para poder acceder a nompai y nomciu
      JOIN UBICACION UBI
      ON P.IDEUBI = UBI.IDEUBI
      -- Para poder acceder al id del toro
      JOIN ESDEVENIMENT ESD
      ON ESD.NOMPLA = P.NOMPLA
      JOIN R_ESDEVENIMENT_TORO ET
```

```
ON ET.IDEESD = ESD.IDEESD
```

```
-- Agrupación por plazas para que no salgan duplicados
```

```
GROUP BY P.NOMPLA, PAIS, CIUDAD;
```

En el entorno de Oracle no se puede ejecutar puesto que la migración no se pudo llevar a cabo. Sin embargo, puede comprobarse si esta consulta es correcta ejecutándola en el entorno de MySQL, donde debería dar el mismo resultado que se se esperaría obtener en Oracle si se hubiera hecho la migración.

```
mysql> -- Consulta ejercicio 22
mysql> SELECT P.NOMPLA AS "NOMBRE DE LA PLAZA",
-> UBI.NOMPAI AS "PAIS",
-> UBI.NOMCIU AS "CIUDAD",
-> COUNT(ET.IDTORO) AS "NUMERO DE TOREADOS"
-> FROM PLAZA P
-> -- Para poder acceder a nompai y nomciu
-> JOIN UBICACION UBI
-> ON P.IDEUBI = UBI.IDEUBI
-> -- Para poder acceder al id del toro
-> JOIN ESDEVENIMENT ESD
-> ON ESD.NOMPLA = P.NOMPLA
-> JOIN R_ESDEVENIMENT_TORO ET
-> ON ET.IDEESD = ESD.IDEESD
-> -- Agrupación por plazas para que no salgan duplicados
-> GROUP BY P.NOMPLA, PAIS, CIUDAD;
```

NOMBRE DE LA PLAZA	PAIS	CIUDAD	NUMERO DE TOREADOS
3 de Julio	Ecuador	Santo Domingo de los Colorados	110484
Abarán	España	Abarán	109350
Albacete	España	Albacete	110266
Alberto Balderas	México	Autlán de la Grana	111618
Alberto Balderas de Morelón	México	Morelón	110160
Alcalá de Henares	España	Alcalá de Henares	115830
Alejandra Durango	México	Juriquilla	114372
Algemesi	España	Algemesi	110322
Alicante	España	Alicante	110646
Almazán	España	Almazán	111942
Almendrales	España	Almendrales	113886
Almería	España	Almería	115830
Andújar	España	Andújar	115020
Apizaco	México	Tlaxcala	113562
Arena	España	Madridejos	115830
Arena de Arlés	Francia	Arlés	113886
Arena de Bayona	Francia	Bayona	112104
Arena de Béziers	Francia	Béziers	109836

...

Los Remedios	España	Los Remedios	36000
Luanda	Angola	Luanda	36072
Maestranza César Girón	Venezuela	Maracay	36072
Maestranza César Girón de Maracay	Uruguay	Maracay	38394
Maestranza de Barcelona	Venezuela	Barcelona	37422
Manizales	Colombia	Manizales	36558
Manzanares	España	Manzanares	38826
Maputo	Mozambique	Maputo	36234
Marbella	España	Marbella	36342
Martos	España	Martos	35100
Melilla	España	Melilla	37314
Mérida	España	Mérida	37368
México	México	Ciudad de México	37368
Mitad del Mundo	Ecuador	Quito	38016
Monte Carmelo Huallanca-Bolognesi-Ancash	Perú	Huallanca	38988
Monumental de Aguascalientes	México	Aguascalientes	35910
Monumental de Ambato	Ecuador	Ambato	39312
Monumental de Barcelona	España	Barcelona	36450
Monumental de Huelva	España	Huelva	36882
Monumental de Maracaibo	Venezuela	Maracaibo	37314
Monumental de Morelia	México	Morelia	36126
Monumental de Pamplona	España	Pamplona (Navarra)	37286
Monumental de Playas de Tijuana	México	Tijuana	38178
Monumental de Sevilla	España	Sevilla	37476
Monumental de Tarragona	España	Tarragona	36396
Monumental de Toros de Pueblo Nuevo	Venezuela	San Cristóbal	36666
Monumental de Valencia	Venezuela	Valencia	37314
Monumental de Villahermosa	México	Tabasco	36612
Monumental Las Palomas	España	Algeciras	37368
Monumental Monterrey Lorenzo Garza Monterrey	México	Nuevo León	36882
Monumental Praça de Touros de Póvoa de Varzim	Portugal	Póvoa de Varzim	38556
Monumental Román Eduardo Sandia	Venezuela	Mérida	36288
Monumental Señor de los Milagros	Colombia	Girón	37286
Monumental Zacatecas	México	Zacatecas	37044
Morón	España	Morón de la Frontera	38286
Móstoles	España	Móstoles	38988
Nueva Andalucía	España	Marbella	38772
Nuevo Circo de Caracas	Venezuela	Caracas	37692
Nuevo Progreso	México	Guadalajara	38232
Olot	España	Olot	36018
Orán	Argelia	Orán	37044
Orihuela	España	Orihuela	19222

181 rows in set (40.48 sec)

```
mysql>
```

Se obtiene un total de 181 filas, donde para cada una se observa el nombre de la plaza, el país y ciudad en la que se encuentra y el número de toros realizados en dicha plaza.

Plan de ejecución y operadores algebraicos

A continuación, se pide obtener el plan de ejecución, explicar y analizar cómo se debe llevar a cabo este análisis y obtener también los operadores algebraicos que usará Oracle para realizar la consulta del apartado anterior.

Plan de ejecución

En Oracle se puede extraer el plan de ejecución de una consulta usando la sentencia “EXPLAIN PLAN” [\[12\]](#) al inicio de la consulta cuyo plan se quiere obtener. Si añadimos esta sentencia, la consulta no se ejecutará, simplemente se obtendrá su plan de ejecución. Si añadimos la mencionada sentencia obtenemos el siguiente código:

```
-- Plan de ejecución
EXPLAIN PLAN FOR
SELECT P.NOMPLA AS "NOMBRE DE LA PLAZA",
       UBI.NOMPAI AS "PAIS",
       UBI.NOMCIU AS "CIUDAD",
       COUNT(ET.IDTORO) AS "NUMERO DE TOREADOS"
FROM PLAZA P
      JOIN UBICACION UBI
      ON P.IDEUBI = UBI.IDEUBI
      JOIN ESDEVENIMENT ESD
      ON ESD.NOMPLA = P.NOMPLA
         JOIN R_ESDEVENIMENT_TORO ET
         ON ET.IDEESD = ESD.IDEESD
GROUP BY P.NOMPLA, PAIS, CIUDAD;
```

Una vez hecho esto, podemos usar la función “*DBMS_XPLAN.DISPLAY*” [\[13\]](#) para mostrar el plan de ejecución. Esta función toma el resultado del plan de ejecución y lo muestra de forma legible. Además, podemos usar “*(FORMAT=>'ALL +OUTLINE')*” para obtener una salida más completa y detallada del plan de ejecución, ya que incluye información adicional como la información del contorno (OUTLINE). La información del contorno, en el contexto de Oracle, se refiere a un conjunto de instrucciones que indican al optimizador de consultas cómo ejecutar una consulta específica. Estas instrucciones son parte de un plan de ejecución almacenado en la base de datos y se conocen como contorno de consulta o contorno de plan.

La sintaxis para la función es la siguiente:

```
-- Ver plan de ejecución de forma detallada
SELECT * FROM TABLE(DBMS_XPLAN.DISPLAY (FORMAT=>'ALL +OUTLINE'));
```

Sin embargo, esto no se puede poner en práctica puesto que no se ha conseguido hacer la migración del apartado anterior.

Operadores algebraicos

Los operadores algebraicos para la consulta son los siguientes:

$$R1 \rightarrow (ET \bowtie ESD)_{ET.ideesd=ESD.ideesd}$$

$$R2 \rightarrow (UBI \bowtie R1)_{ESD.nompla=P.nompla}$$

$$R3 \rightarrow (P \bowtie R2)_{P.ideubi=UBI.ideubi}$$

$$R4 \rightarrow \Pi_{P.nompla, UBI.nompai, UBI.nomciu, COUNT(ET.idtoro)}(R3)$$

$$R5 \rightarrow \gamma_{P.nompla, UBI.nompai, UBI.nomciu}(R4)$$

Una vez hecho esto, podemos crear una única expresión algebraica combinando todas las expresiones anteriores. Partiendo desde el final, sustituiremos en R6 el valor de R5, en R5 el valor de R4, en R4 el valor de R3, y así sucesivamente hasta llegar a obtener solo una expresión algebraica. El resultado que obtendríamos es el siguiente.

$$\left(\gamma_{P.nompla, UBI.nompai, UBI.nomciu} \left(\Pi_{P.nompla, UBI.nompai, UBI.nomciu, COUNT(ET.idtoro)}(R3) \right) \right)$$

Donde R3 es:

$$\left(P \bowtie (UBI \bowtie (ET \bowtie ESD)_{ET.ideesd=ESD.ideesd})_{ESD.nompla=P.nompla} \right)_{P.ideubi=UBI.ideubi}$$

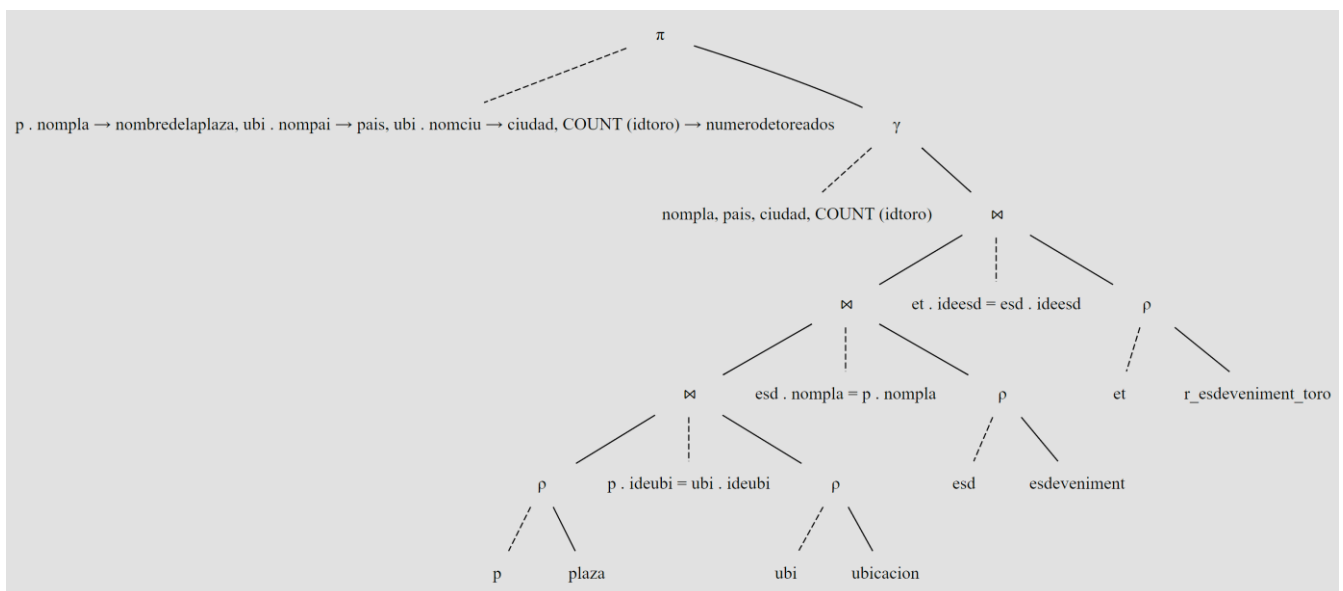
En el paso anterior hemos obtenido una serie de expresiones algebraicas. Estas representan una secuencia de operaciones en álgebra relacional que se utilizan para obtener un conjunto de datos específico a partir de múltiples tablas, aplicando, en cada paso, joins, proyecciones, selecciones... Vamos a analizar cada una de las relaciones obtenidas con el objetivo de entender qué operación algebraica representan, sobre qué tabla se aplica esta operación y con qué finalidad:

- **R1:** Realiza una operación de JOIN entre las tablas R_ESDEVENIMENT_TORO (ET) y ESDEVENIMENT (ESD) bajo la condición de unión siguiente: $ET.ideesd=ESD.ideesd$. La unión viene determinada por el operador \bowtie , y la condición de unión es la que se puede ver en el ON de la consulta original en SQL. Esta condición asegura que solo se incluyan filas donde el *ideesd* en la tabla ET sea igual al *ideesd* en la tabla ESD. El resultado contendrá las columnas de ambas tablas.
- **R2:** Representa otra operación de JOIN, esta vez entre las tablas UBICACIO (UBI) y las dos tablas anteriores descritas en R1. Dado que se une UBI con el resultado obtenido de la expresión R1 se usa el operador \bowtie con UBI y R1. Una vez más, la condición de unión se basa en la simple igualdad de las columnas que tienen en común ambas tablas, es decir, en la igualdad entre la columna *nompla* en P con la columna *nompla* en ESD.
- **R3:** Representa otra operación de JOIN, esta vez entre las tablas PLAZA (P) y el resultado obtenido anteriormente en R2. Dado que se une P con el resultado obtenido de la expresión R2 se

usa el operador \bowtie con P y R2. Otra vez, la condición de unión se basa en la simple igualdad de las columnas que tienen en común ambas tablas, es decir, en la igualdad entre la columna *ideubi* en UBI con la columna *ideubi* en P.

- **R4:** Representa una operación de proyección mediante el SELECT sobre la tabla R3 obtenida anteriormente. Las columnas proyectadas son P.NOMPLA bajo el alias “NOMBRE DE LA PLAZA”, UBI.NOMPAI bajo el alias “PAIS”, UBI.NOMCIU bajo el alias “CIUDAD”, UBI.IDEUBI bajo el alias “ID UBICACION” y COUNT(ET.IDTORO) bajo el alias “NUMERO DE TOREADOS. El objetivo de esto es seleccionar solo las columnas que nos interesan de la tabla resultante de la expresión R3.
- **R5:** La agrupación de los valores que se proyectan se hace en función del valor del nombre de la plaza en primer lugar, del país en segundo lugar, y de la ciudad en tercer lugar. Esta expresión algebraica representa una agrupación (GROUP BY) y por ellos usamos el operador gamma (γ)

El árbol correspondiente a las expresiones algebraicas es el siguiente:



Conclusiones

Tras realizar las operaciones establecidas por el enunciado sobre distintos gestores y tratar con la información ofrecida, se pueden extraer las siguientes conclusiones:

La normalización y el paso a un modelo relacional desempeñan un papel crucial en el mantenimiento de la integridad y consistencia de la información dentro de una base de datos. Al estructurar la base de datos de manera eficiente, se evita la redundancia de datos, asegurando que la información esté organizada de manera coherente y minimizando la posibilidad de errores o inconsistencias. En nuestro caso, la tabla ESDEVENIMENT ha sido separada en varias tablas, de forma que se relaciona un esdeveniment con ferias, toros, y cada toro con su respectiva “ramaderia” (ganadería). De esta forma se mantiene un orden, consistencia e integridad referencial que asegura un uso optimizado y mayor orden en la base de datos.

Por otro lado, la gestión adecuada de permisos es esencial para garantizar la seguridad y privacidad de los datos almacenados. Conceder únicamente los permisos necesarios a cada usuario de la base de datos ayuda a prevenir accesos no autorizados o modificaciones indebidas. Es crucial evaluar de manera constante los privilegios otorgados, asegurándose de no conceder o revocar permisos que no sean estrictamente necesarios para el usuario en cuestión. En nuestro caso pudimos comprobar en el apartado 11, al realizar el traspaso de información a la tabla persona, que era necesario modificar la condición NOT NULL impuesta a dos columnas sobre la tabla Persona. Se optó por jugar con los privilegios para el usuario que estaba realizando la importación (ya que este usuario, MIGBD, es el propietario o creador de dicha base de datos), e inmediatamente después de que la importación acabara, se le revocaron estos permisos ya que nunca se deben tener más de los estrictamente necesarios.

Asimismo, la optimización de consultas resulta ser un elemento clave para mejorar el rendimiento en bases de datos que manejan grandes volúmenes de información. Especialmente en consultas recurrentes, la eficiencia se convierte en un factor determinante para agilizar el acceso a la información. Estrategias como la indexación adecuada, la reescritura de consultas y la monitorización constante permiten optimizar el rendimiento del sistema, mejorando significativamente la velocidad de recuperación de datos.

Además, el uso de un Foreign Data Wrapper en el gestor PostgreSQL ha resultado interesante, ya que es una herramienta de gran utilidad para poder leer datos desde un gestor hacia otro sin necesidad de hacer una migración entre estos gestores. Su utilidad se puede extender a varios ámbitos como entornos empresariales reales donde se usan diferentes bases de datos distribuidas, la consolidación de datos de distintas bases de datos o el acceso a datos externos como archivos CSV o XML, donde se sabe que un FDW puede ser de utilidad.

Por último, cabe destacar la relevancia de que exista una buena documentación ofrecida al usuario por parte de los desarrolladores de los distintos gestores usados. Esta documentación es vital y se ha usado en numerosas ocasiones para comprender la sintaxis y el funcionamiento de algunos aspectos en concreto tales como el tablespace en MySQL, o el Foreign Data Writer (FDW) en PostgreSQL.

En resumen, el trabajo realizado ha exigido la puesta en práctica de competencias de diseño, gestión y manipulación o manipulación de grandes lotes de datos, al igual que ha sido necesaria una abstracción y pensamiento estratégico para poder resolver las cuestiones establecidas y los problemas que iban surgiendo en cada una de ellas.

Referencias

- [\[1\] 15.1.21 CREATE TABLESPACE Statement](#)
- [\[2\] CREATE DATABASE Statement in MySQL](#)
- [\[3\] Character Sets and Collations in MySQL](#)
- [\[4\] SHOW CHARACTER SET Statement](#)
- [\[5\] REVOKE STATEMENT](#)
- [\[6\] 14.7 Date and Time Functions](#)
- [\[7\] Get started with databases on Windows Subsystem for Linux](#)
- [\[8\] 14.7 Date and Time Functions](#)
- [\[9\] PostgreSQL EXTRACT function](#)
- [\[10\] PostgreSQL AVG Function](#)
- <https://www.geeksforgeeks.org/how-to-install-sqlplus-on-linux/>
- [\[11\] GRANT STATEMENT in Oracle](#)
- [\[12\] How to Generate a Useful SQL Execution Plan](#)
- [\[13\] 207 DBMS XPLAN](#)