Portada

Integrantes del grupo:

- Rubén Gómez Olivencia

Índice

1. Introducción

La empresa **KulunSport** quiere abrirse camino en el mundo deportivo que está tan de moda actualmente queriéndose convertir en un referente entre los deportistas del mundo del ciclismo, running y andar, pero de cara a futuro abarcar más deportes como la natación, triatlón, …

Entre los nuevos servicios que quiere ofrecer está un portal web donde los deportistas se puedan registrar e ir registrando sus entrenamientos.

Para que los deportistas federados puedan tener una mejor gestión de las pruebas oficiales que realizan en los distintos campeonatos que dirigen las federaciones, se quiere que puedan tener un histórico de sus tiempos oficiales realizados.

Así mismo, y para abrirse camino entre deportistas no federados, se quiere incentivar el deporte proponiendo retos a estos deportistas, que en caso de ser completados se les entregarán logros virtuales.

2. Necesidades y requisitos del usuario (análisis de requisitos)

Para conocer de manera exacta las necesidades del cliente se han realizado varias entrevistas a las que han asistido:

- **María de los Ángeles Rodríguez**: Responsable de la empresa KulunSport y encargada de negocio.
- **Rubén Gómez Olivencia**: Jefe de proyecto y diseñador de bases de datos.

De esta manera, se ha podido recabar la información necesaria del estado actual de los datos que contiene la empresa, de las necesidades básicas que tienen, de los requisitos exigidos de cara a la terminación de proyecto, así como de sus ideas de futuro.

Toda esta información será explicada en los apartados que vienen a continuación.

2.1. Estado actual de los datos

Actualmente la empresa cuenta con los datos de muy pocos deportistas profesionales, y estos datos están almacenados en una hoja de cálculo sin ningún sistema de respaldo automatizado.

Sobre el calendario de competiciones, se avisa a los deportistas directamente, sin quedar registrado ningún histórico de las pruebas que han realizado.

Es por eso que la empresa quiere aprovechar la idea de la creación del portal web para crear una base de datos para los deportistas profesionales que tienen actualmente, pero también para atraer a nuevos deportistas, tanto profesionales como amateurs.

2.2. Necesidades

Desde KulunSport se nos informa que la necesidad principal del proyecto es la de crear un portal web, y con ello una base de datos, en la que se puedan gestionar deportistas que practican distintos deportes, ya sea de forma federada o no.

La empresa quiere ofrecer a todos los deportistas que utilicen su portal la gestión de sus entrenamientos. De inicio se van a centrar en los deportes de ciclismo, correr y andar, pero quieren dejar abierta la posibilidad de que el diseño permita añadir más deportes de cara a futuro.

Para incentivar al uso de la plataforma, quieren que los deportistas federados puedan registrar sus tiempos oficiales realizados en pruebas de campeonatos.

Para los deportistas no federados quieren crear retos que una vez terminados otorgarán logros virtuales.

Por último, y para el disfrute de todos los deportistas, quieren mantener el registro de asistencia a eventos deportivos y de los tiempos realizados en ellos.

2.3. Tareas sobre datos

Una vez entendida las necesidades generales del cliente, llegó el momento de recabar la información de las tareas que se deben permitir realizar y sobre qué tipo de datos se efectuarán dichas tareas.

Esta información se va a reflejar en la siguiente tabla:

Tarea	Datos necesarios
Registro y edición del deportista	Nombre, apellidos, fecha de nacimiento, género del deportista, usuario y password
Registro de entrenamiento	Deporte del entrenamiento, nombre del entrenamiento, la fecha en la que se ha realizado, duración del mismo y la distancia realizada
Consulta de entrenamientos	Deporte del entrenamiento, nombre del entrenamiento, la fecha en la que se ha realizado, duración del mismo y la distancia realizada

Tarea	Datos necesarios
Registro de número de federado	Usuario, deporte y número de federado
Registro de federación	Nombre, dirección y la web de la federación
Consulta de las federaciones	Nombre, dirección y web de la federación
Registro de deportes	Nombre del deporte
Consultar deportes	Nombre de los deportes y federaciones a las que pertenecen
Registro de pruebas	Nombre y distancia
Registro de campeonato	Nombre del campeonato, fecha de inicio y fecha de finalización
Registrar tiempo en prueba de campeonato	Deportista, prueba de campeonato y tiempo
Consulta de pruebas en competiciones	Deportista, pruebas realizadas y tiempos
Registro de eventos	Nombre del evento, fecha y distancia
Registro de asistencia a eventos	Deportista, evento y tiempo realizado
Consulta de eventos	Eventos, por fecha, por distancia…
Creación de retos	Nombre del reto, fecha inicio, fecha fin, distancia y tiempo
Consulta de retos	Retos por fecha, distancia…
Hacer reto	Fecha unión, fecha completado
Crear logros	Nombre del logro
Obtener logros	Fecha de obtención
Listar logros	Logros por deportista

2.4. Requisitos finales

Tras ver las tareas que se quieren realizar sobre los datos, a continuación se van a desgranar y convertir en requisitos atómicos indicando los datos que tomarán parte.

Requisitos Finales	Datos necesarios
Registro del deportista	Nombre, apellidos, fecha de nacimiento, género del deportista, usuario y password
Edición de los datos del deportista	Nombre, apellidos, fecha de nacimiento, género del deportista, usuario y password
Registro de número de federado	Usuario, deporte ynúmero de federado.
Registro de federación	Nombre, dirección y web de la federación
Consulta de las federaciones	Nombre, dirección y web de la federación
Registro de deportes	Nombre del deporte, federación a la que pertenece
Consulta de deportes	Nombre de los deportes y federaciones a las que pertenecen
Registro de pruebas	Nombre, distancia y federación a la que pertenece
Registro de campeonato	Nombre del campeonato, fecha de inicio, fecha de finalización, y federación que organiza
Registrar tiempo en prueba de campeonato	Deportista, prueba de campeonato y tiempo

Requisitos Finales	Datos necesarios
Consulta de pruebas en competiciones	Deportista, pruebas de campeonatos y tiempos
Registro de eventos	Nombre del evento, fecha y distancia
Registro de asistencia a eventos	Deportista, evento y tiempo realizado
Consulta de eventos	Nombre, fecha, y distancia de eventos
Registro de retos	Nombre del reto, fecha inicio, fecha fin, distancia y tiempo
Consulta de retos	Deportista, retos por fecha, distancia
Hacer reto	Deportista, reto, fecha unión
Completar reto	Deportista, reto, fecha de finalización
Registro de logros	Nombre del logro, evento asociado
Obtener logros	Fecha de obtención
Listar logros	Deportista y logros obtenidos

No se ha quedado fuera ninguna de las tareas propuestas inicialmente por **KulunSport**, y por tanto todos los requisitos finales cumplirán con sus necesidades.

3. Diseño de la base de datos

Una vez realizada la toma de requisitos, habiendo entendido cuáles son las necesidades del cliente y habiendo cerrados los requisitos finales que se van a implantar, procedemos a realizar el diseño de la base de datos.

A la hora de realizar el diseño es necesario realizar varios pasos de manera secuencial para llevar a buen término la generación final de la base de datos. Estos pasos a su vez pueden contar con apartados para realizar tareas atómicas y que de esta manera el diseño cuente con todas las necesidades previamente expuestas.

3.1. Diseño conceptual

A la hora de realizar el diseño conceptual se han seguido los siguientes pasos:

1. Identificar las **entidades** que formarán parte del diseño de datos.
2. Identificar las **relaciones** que existen entre las entidades obtenidas.
3. Identificar los **atributos** que tienen las entidades y las relaciones
4. Determinar el **dominio** de los distintos atributos.
5. Identificar los **atributos únicos** en las entidades.
6. Plasmar el resultado en un diagrama Entidad-Relación

A continuación se detallan cada apartado por separado y los datos obtenidos

3.1.1. Identificar las entidades

Las entidades que van a formar parte de la base de datos son las siguientes:

- **Deportista**: Son los usuarios que van a utilizar la aplicación.
- **Entrenamiento**: Son los entrenamientos que van a realizar los deportistas.
- **Deporte**: Los deportes que van a poder practicar los usuarios.
- **Federación**: Los deportes están reglados por distintas federaciones deportivas.
- **Prueba**: Cada deporte consta de distintas pruebas oficiales.
- **Campeonato**: Las federaciones dirigen campeonatos donde se realizan pruebas oficiales.
- **Evento**: Las federaciones organizan eventos para todo tipo de deportistas.
- **Reto**: Retos propuestos por la empresa KulunSport para incentivar el ejercicio entre los deportistas.
- **Logro**: Logros virtuales obtenidos por los deportistas tras completar retos.

3.1.2. Identificar relaciones

Las relaciones surgidas del análisis de requisito son las siguientes:

- **Realiza**, es una relación entre deportista y entrenamiento.
- **Practíca**, es una relación entre deportista y deporte.
- **Pertenece**, es una relación entre un entrenamiento y deporte.
- **Regla**, es una relación entre la federación y el deporte al que crea las reglas.
- **Tiene**, es una relación entre deporte y pruebas.
- **Dirige**, es una relación entre federación y campeonato.
- **Compite**, es una relación entre deportista, la prueba y el campeonato en el que compite.
- **Organiza**, es una relación entre federación y los eventos.
- **Asiste**, es una relación entre deportista y eventos.
- **Tiene**, es una relación entre deporte y retos.
- **Hace**, es una relación entre deportista y retos.
- **Regala**, es una relación entre retos y logros.
- **Obtiene**, es una relación entre deportista y logros.

Mientras se ha realizado el análisis para obtener dichas relaciones, también se ha aprovechado para tener en cuenta la cardinalidad de las mismas. Toda esa información es plasmada más adelante en el diagrama de Entidad-Relación.

3.1.3. Identificar atributos

Las entidades obtenidas requieren de atributos que son mostrados a continuación:

- **Deportista**: nombre, apellidos, fecha de nacimiento, género, usuario y password
- **Entrenamiento**: nombre, fecha, duracion, distancia
- **Deporte**: nombre
- **Federación**:nombre, direccion, web
- **Prueba**: nombre, distancia
- **Campeonato**: nombre, fecha_inicio, fecha_fin
- **Evento**: nombre, fecha, distancia
- **Reto**: nombre, descripcion, fecha_inicio, fecha_fin, distancia, tiempo

- **Logros**: nombre

Y algunas de las relaciones identificadas previamente también requieren de atributos:

- **Practíca**: federado, numero_federado
- **Compite**: tiempo_realizado
- **Asiste**: tiempo_realizado
- **Hace**: fecha_unirse, fecha_completado
- **Obtiene**: fecha

3.1.4. Identificar dominios

Entre todos los atributos especificados previamente, sólo existe un dominio:

- **Genero**: posibles valores: “masculino”, “femenino” o “nd”

Donde “nd” significa “no disponible” ya que el deportista ha preferido no indicarlo.

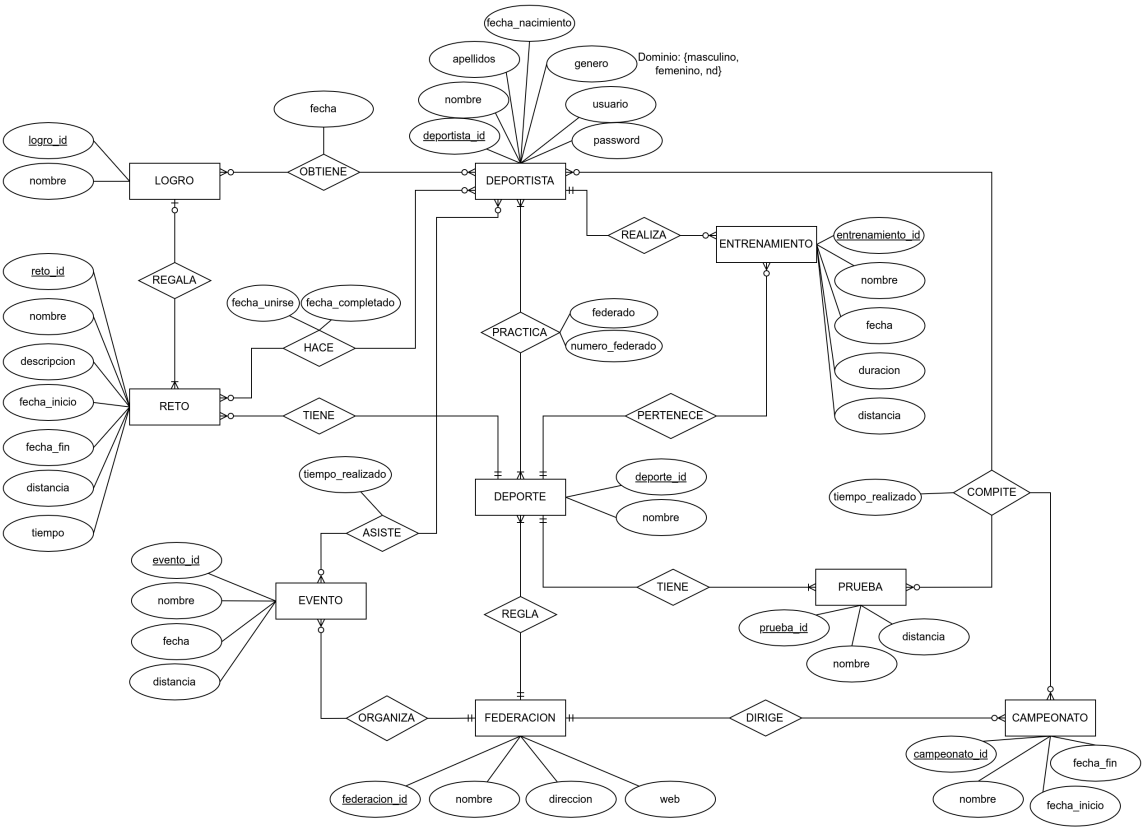
3.1.5. Identificar los atributos únicos en las entidades

Las entidades deben de contar con atributos que deben ser únicos en la base de datos para poder ser identificados. En este caso, hemos optado por añadir un nuevo atributo a las entidades para que sea un identificador entero único, que será el siguiente para cada entidad:

- **Deportista**: deportista_id
- **Entrenamiento**: entrenamiento_id
- **Deporte**: deporte_id
- **Federación**: federacion_id
- **Prueba**: prueba_id
- **Campeonato**: campeonato_id
- **Evento**: evento_id
- **Reto**: reto_id
- **Logros**: logro_id

3.1.6. Diagrama Entidad-Relación

Tras realizar todo el análisis previo, se va a plasmar en un diagrama Entidad-Relación, ya que es la manera más visual de agrupar y mostrar todo lo realizado hasta ahora. Tal como se ha dicho previamente, mientras se identificaban las relaciones se ha analizado la cardinalidad de las mismas, y también son mostradas en el siguiente diagrama.



3.2 Diseño lógico

Con el diseño conceptual previamente realizado, vamos a pasar a realizar el diseño lógico de datos.

Para la realización del diseño lógico los pasos realizados han sido los expuestos a continuación.

3.2.1. Entidades a tablas

El primer paso es el de transformar las entidades con todos sus atributos en tablas. De esta manera, vamos a formar tablas en las que deberemos asegurar que realizamos:

- Identificar cuál es la clave primaria
- Identificar si existen claves ajenas y a qué tabla y atributo va a referenciar
- Determinar el dominio de los atributos

3.2.2. Relaciones de N a M a tablas

La transformación de las relaciones “N a M” del diseño conceptual al lógico es el método más sencillo, ya que de esta relación surgirá una nueva tabla. En estas nuevas tablas tenemos que realizar los siguientes pasos:

- Identificar cuáles son las claves primarias
- Determinar las claves foráneas y a qué tabla y atributo van a referenciar
- Añadir si existen otros atributos mencionados en el esquema conceptual

3.2.3. Relaciones 1 a N

En las relaciones “1 a N” del esquema conceptual lo habitual es tener que pensar dónde se sitúa la clave ajena y a qué tabla y atributo va a referenciar.

Teniendo esto en cuenta, y con el diagrama Entidad-Relación previamente creado, se han añadido como claves ajenas atributos en las tablas correspondientes que aparecen en el diseño lógico final que se ha añadido más adelante.

3.2.4. Relaciones 1 a 1

Por último, las relaciones 1 a 1. Debemos pensar cuál de las entidades que forman parte de la relación es fuerte o débil, o si ambas son fuertes.

Es importante realizar el análisis previo, ya que va a determina dónde se sitúa la clave ajena (en cuál de las dos tablas de entidades) y a qué tabla y atributo va a referenciar.

3.2.5. Añadir reglas de integridad

Tras todo lo visto hasta ahora, es el momento en el que vamos a tener que revisar todas las claves foráneas existentes y determinar para cada una de ellas las reglas de integridad que va a tener:

- Si la clave foránea va a aceptar nulos o no
- Indicar qué regla va a pasar al ser borrada la clave referenciada:
  - **restringir**: se previene el intento de borrado de la clave referenciada.
  - **propagar**: al borrar la clave referenciada, se borrarán los registros de la tabla que referencia.
  - **poner nulo**: al borrar la clave referenciada, se pondrá "nulo" como clave foránea.
  - **poner por defecto**: al borrar la clave referenciada, se pondrá el valor por defecto como clave foránea.
- Regla al ser modificada la clave referenciada (igual que el caso anterior).

3.2.6. Normalización

Al tratarse del diseño de una base de datos relacional, el último paso es el de realizar la normalización y confirmar que el diseño final cumple al menos las primeras tres formas normales. Para entender este paso, y qué es lo que se debe cumplir en cada forma normal, se detalla a continuación

- **Primera forma normal**: se produce cuando ningún atributo de la relación es una relación en sí mismo ni se puede descomponer; los atributos entonces deben de ser atómicos.
- **Segunda forma normal**: solo cuando está ya en la primera forma normal y todo atributo que no forma parte de una clave candidata depende completamente de cada una de las claves candidatas de la relación.
- **Tercera forma normal**: solo cuando está ya en segunda forma normal y ningún atributo que forma parte de una clave candidata depende de un conjunto de atributos

que contiene algunos que no forman parte de una clave candidata.

En nuestro caso, hemos realizado todos estos pasos para asegurar y confirmar que la normalización se ha llevado a cabo de manera correcta en el diseño lógico final, y no ha sido necesario desnormalizar.

3.2.7. Diseño lógico resultante

Tras la realización de los pasos comentados previamente, se ha concluído con el siguiente diseño lógico, en el que comenzamos con las tablas de las entidades para posteriormente pasar a las tablas de las relaciones obtenidas.

```
DEPORTISTA(deportista_id, nombre, apellidos,
fecha_nacimiento, genero, usuario, password)
    PK: {deportista_id}
    Dominio genero: [masculino, femenino, nd]


FEDERACION(federacion_id, nombre, direccion, web)
    PK:{federacion_id}


DEPORTE(deporte_id,nombre, federacion_id)
    PK:{deporte_id}
    FK: DEPORTE.federacion_id → FEDERACION.federacion_id
        ¿Acepta nulos?: NO
        Borrado: PROPAGAR
        Modificación: PROPAGAR


ENTRENAMIENTO(entrenamiento_id, nombre, fecha, duracion,
distancia, deportista_id, deporte_id)
    PK: {entrenamiento_id}
    FK: ENTRENAMIENTO.deportista_id →
DEPORTISTA.deportista_id
        ¿Acepta nulos?: NO
        Borrado: PROPAGAR
        Modificación: PROPAGAR
    FK: ENTRENAMIENTO.deporte_id → DEPORTE.deporte_id
        ¿Acepta nulos?: NO
        Borrado: RESTRINGIR
        Modificación: PROPAGAR


CAMPEONATO(campeonato_id, nombre, fecha_inicio, fecha_fin,
federacion_id)
    PK: {campeonato_id}
    FK: CAMPEONATO.federacion_id → FEDERACION.federacion_id
        ¿Acepta nulos?: NO
        Borrado: PROPAGAR
        Modificación: PROPAGAR


PRUEBA(prueba_id, nombre, distancia, deporte_id)
```

PK: {prueba_id}
FK: PRUEBA.deporte_id → DEPORTE.deporte_id
¿Acepta nulos?: NO
Borrado: RESTRINGIR
Modificación: PROPAGAR

LOGRO(logro_id, nombre)
PK:{logro_id}

RETO(reto_id, nombre, descripcion, fecha_inicio, fecha_fin, distancia, tiempo, deporte_id, logro_id)
PK: {reto_id}
FK: RETO.deporte_id → DEPORTE.deporte_id
¿Acepta nulos?: NO
Borrado: RESTRINGIR
Modificación: PROPAGAR
FK: RETO.logro_id → LOGRO.logro_id
¿Acepta nulos?: NO
Borrado: RESTRINGIR
Modificación: PROPAGAR

EVENTO(evento_id, nombre, fecha, distancia, federacion_id)
PK: {evento_id}
FK: EVENTO.federacion_id → FEDERACION.federacion_id
¿Acepta nulos?: NO
Borrado: PROPAGAR
Modificación: PROPAGAR

PRACTICA(deportista_id,deporte_id, federado, numero_federado)
PK: {deportista_id,deporte_id}
FK: PRACTICA.deportista_id → DEPORTISTA.deportista_id
¿Acepta nulos?: NO
Borrado: PROPAGAR
Modificación: PROPAGAR
FK: PRACTICA.deporte_id → DEPORTE.deporte_id
¿Acepta nulos?: NO
Borrado: RESTRINGIR
Modificación: PROPAGAR

ASISTE(deportista_id,evento_id,tiempo_realizado)
PK: {deportista_id,evento_id}
FK: ASISTE.deportista_id → DEPORTISTA.deportista_id
¿Acepta nulos?: NO
Borrado: PROPAGAR
Modificación: PROPAGAR
FK: ASISTE.evento_id → EVENTO.evento_id
¿Acepta nulos?: NO
Borrado: RESTRINGIR
Modificación: PROPAGAR

COMPITE(deportista_id,prueba_id,campeonato_id,tiempo_realizado)

```
        PK: {deportista_id,prueba_id,campeonato_id}
        FK: COMPITE.deportista_id → DEPORTISTA.deportista_id
            ¿Acepta nulos?: NO
            Borrado: PROPAGAR
            Modificación: PROPAGAR
        FK: COMPITE.prueba_id → PRUEBA.prueba_id
            ¿Acepta nulos?: NO
            Borrado: RESTRINGIR
            Modificación: PROPAGAR
        FK: COMPITE.campeonato_id → CAMPEONATO.campeonato_id
            ¿Acepta nulos?: NO
            Borrado: RESTRINGIR
            Modificación: PROPAGAR


    HACE(deportista_id,reto_id,fecha_unirse,fecha_completado)
        PK: {deportista_id,reto_id}
        FK: HACE.deportista_id → DEPORTISTA.deportista_id
            ¿Acepta nulos?: NO
            Borrado: PROPAGAR
            Modificación: PROPAGAR
        FK: HACE.reto_id → RETO.reto_id
            ¿Acepta nulos?: NO
            Borrado: RESTRINGIR
            Modificación: PROPAGAR


    OBTIENE(deportista_id,logro_id,fecha)
        PK: {deportista_id,logro_id}
        FK:OBTIENE.deportista_id → DEPORTISTA.deportista_id
            ¿Acepta nulos?: NO
            Borrado: PROPAGAR
            Modificación: PROPAGAR
        FK:OBTIENE.logro_id → LOGRO.logro_id
            ¿Acepta nulos?: NO
            Borrado: RESTRINGIR
            Modificación: PROPAGAR
```

3.3 Diseño físico

Tras la realización del diseño lógico, es el momento para la realización del diseño físico, y para ello seguiremos los pasos detallados a continuación.

3.3.1. Elección del SGBD a utilizar

Es importante realizar la elección del sistema gestor de base de datos (SGBD) de manera correcta, conociendo las características de este, ya que afectará al diseño físico que tenemos que realizar. Entre los SGBD relacionales, existen ciertas diferencias (tipos de datos, particionado de tablas, posibilidad de realizar replicación, ...) que pueden afectar a la realización del diseño físico y que en otro SGBD relacional no funcionaría.

En nuestro caso, vamos a hacer uso de **PostgreSQL en su versión 13.6** ya que es el sistema gestor de base de datos relacional que mejor se adapta a las necesidades de nuestro cliente.

3.3.2. Transformar los elementos del esquema lógico

En este paso realizaremos el cambio de las tablas, junto con sus atributos, al lenguaje SQL utilizado en las bases de datos relacionales.

Durante el proceso de creación de los atributos, hemos comprobado los tipos de datos que se pueden utilizar en PostgreSQL para utilizar los más adecuados para cada atributo, y tal como hemos dicho previamente, todas las características que mejor se adaptan al proyecto.

3.3.3. Reglas de integridad para claves primarias y ajenas

Junto con el paso anterior, a medida que vamos creando los atributos, para aquellos que hemos indicado en el paso anterior como claves primarias y claves ajenas, tendremos que darle el tratamiento especial que merecen. Es por ello que aplicaremos las reglas de integridad detalladas previamente.

3.3.4. Diseño físico resultante

Tras seguir todos los pasos, a continuación aparece el diseño físico resultante.

```
CREATE DOMAIN genero_deportista as VARCHAR CHECK(
    VALUE IN ('masculino','femenino','nd')
);


CREATE TABLE DEPORTISTA(
    deportista_id SERIAL not null primary key,
    nombre VARCHAR(20) not null,
    apellidos VARCHAR(30),
    fecha_nacimiento date,
    genero genero_deportista,
    usuario VARCHAR (20) not null,
    password VARCHAR (20) not null
);


CREATE TABLE FEDERACION(
    federacion_id SERIAL not null primary key,
    nombre VARCHAR(50),
    direccion VARCHAR(70),
    web VARCHAR(50)
);


CREATE TABLE DEPORTE(
    deporte_id SERIAL not null primary key,
```

```
    nombre VARCHAR(50),
    federacion_id INTEGER not null REFERENCES
federacion(federacion_id) ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE ENTRENAMIENTO(
    entrenamiento_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha TIMESTAMPTZ,
    duracion INTEGER,
    distancia INTEGER,
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id) ON
DELETE RESTRICT ON UPDATE CASCADE
);


CREATE TABLE CAMPEONATO(
    campeonato_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha_inicio TIMESTAMPTZ,
    fecha_fin TIMESTAMPTZ,
    federacion_id INTEGER not null REFERENCES
federacion(federacion_id) ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE PRUEBA(
    prueba_id SERIAL not null primary key,
    nombre VARCHAR(50),
    distancia INTEGER,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id) ON
DELETE RESTRICT ON UPDATE CASCADE
);


CREATE TABLE LOGRO(
    logro_id SERIAL not null primary key,
    nombre VARCHAR(50)
);


CREATE TABLE RETO(
    reto_id SERIAL not null primary key,
    nombre VARCHAR(50),
    descripcion VARCHAR(200),
    fecha_inicio TIMESTAMPTZ,
    fecha_fin TIMESTAMPTZ,
    distancia INTEGER,
    tiempo INTEGER,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id) ON
DELETE RESTRICT ON UPDATE CASCADE,
    logro_id INTEGER not null REFERENCES logro(logro_id) ON DELETE
RESTRICT ON UPDATE CASCADE
);
```

```sql
CREATE TABLE EVENTO(
    evento_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha_inicio TIMESTAMPTZ,
    distancia INTEGER,
    federacion_id INTEGER not null REFERENCES
federacion(federacion_id) ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE PRACTICA(
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id) ON
DELETE RESTRICT ON UPDATE CASCADE,
    federado boolean,
    numero_federado INTEGER,
    PRIMARY KEY (deportista_id,deporte_id)
);


CREATE TABLE ASISTE(
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    evento_id INTEGER not null REFERENCES evento(evento_id) ON
DELETE RESTRICT ON UPDATE CASCADE,
    tiempo_realizado INTEGER,
    PRIMARY KEY (deportista_id,evento_id)
);


-- para la competición se necesita que el tiempo (en segundos)
tenga decimales
CREATE TABLE COMPITE(
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    prueba_id INTEGER not null REFERENCES prueba(prueba_id) ON
DELETE RESTRICT ON UPDATE CASCADE,
    campeonato_id INTEGER not null REFERENCES
campeonato(campeonato_id) ON DELETE RESTRICT ON UPDATE CASCADE,
    tiempo_realizado FLOAT,
    PRIMARY KEY (deportista_id,prueba_id,campeonato_id)
);


CREATE TABLE HACE(
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    reto_id INTEGER not null REFERENCES reto(reto_id) ON DELETE
RESTRICT ON UPDATE CASCADE,
    fecha_unirse TIMESTAMPTZ,
    fecha_completado TIMESTAMPTZ,
    PRIMARY KEY (deportista_id,reto_id)
);
```

```
CREATE TABLE OBTIENE(
    deportista_id INTEGER not null REFERENCES
deportista(deportista_id) ON DELETE CASCADE ON UPDATE CASCADE,
    logro_id INTEGER not null REFERENCES logro(logro_id) ON DELETE
RESTRICT ON UPDATE CASCADE,
    fecha TIMESTAMPTZ,
    PRIMARY KEY (deportista_id,logro_id)
);
```

3.3.5. Creación de vistas

Al diseño físico anterior se le han añadido dos vistas que serán utilizadas para realizar consultas sobre la base de datos durante el uso de la base de datos en producción.

```
-- Vistas
CREATE VIEW deportista_logro AS
  SELECT d.nombre as deportista_nombre, genero, l.nombre as
logro_nombre, o.fecha
    FROM deportista AS d, logro AS l, obtiene AS o
    WHERE d.deportista_id = o.deportista_id AND o.logro_id =
l.logro_id
;


CREATE VIEW marcas_oficiales AS
  SELECT d.nombre, d.apellidos, c.tiempo_realizado, p.nombre AS
prueba, cmp.nombre AS campeonato
    FROM deportista d, compite c, prueba p, campeonato cmp
    WHERE d.deportista_id = c.deportista_id AND c.campeonato_id =
cmp.campeonato_id AND c.prueba_id = p.prueba_id
;
```

4. Despliegue de la base de datos

Tras realizar todo los pasos explicados previamente para llegar al esquema físico, es el momento de hacer el despliegue de nuestro esquema físico en PostgreSQL.

Para la realización de este despliegue se ha utilizado dos servidores:

- **Servidor local**: Desplegado a través de un contenedor **Docker** exponiendo los puertos necesarios.
- **Servidor remoto**: Servidor desplegado en la nube de Amazon (AWS) a través del proveedor Elephantsql.com

4.1. Conexión a servidores

En este apartado se va a explicar cómo realizar la conexión a servidores PostgreSQL a través del lenguaje de programación **Python**. Antes de realizar la conexión es necesario tener la librería **psycopg2** instalada en nuestro sistema. Para ello realizaremos la instalación mediante el comando de sistema `pip`.

Debido a que la base de datos local usada es a través de un contenedor **Docker**, y por tanto el acceso a las librerías no están accesibles para realizar la compilación, se ha optado por utilizar la misma librería pero en formato binario. De esta manera evitamos tener que realizar la compilación.

In [1]:
```
# Para realizar la conexión necesitamos de la API de python.
# Para realizar la instalación del paquete necesario deberíamos
# ejecutar el siguiente comando que se descarga la librería y la
# compila.
#
# ¿Cuál es el problema? Que requiere del path donde está instalado
# PostgreSQL y por tanto en el entorno local no funciona por estar
# utilizando PostgreSQL en un contenedor Docker
# !pip install psycopg2

# Para poder funcionar en un entorno donde PostgreSQL está en Docker,
# necesitamos instalar la librería en formato binario de esta manera:
!pip install psycopg2-binary
```

```
Defaulting to user installation because normal site-packages is not writeab
le
Requirement already satisfied: psycopg2-binary in /home/yuki/.local/lib/pyt
hon3.10/site-packages (2.9.3)
```

Y tras esto, necesitamos cargar la librería para poder hacer uso de ella:

In [2]:
```
# importamos la librería para poder hacer uso de ella:
import psycopg2
```

Tras importar la librería, vamos a necesitar los siguientes datos para poder realizar la conexión a cada una de las bases de datos:

- **Nombre de la base de datos**: PostgreSQL permite tener distintas bases de datos en el mismo servidor, por lo que debemos de conocer el nombre exacto de la base de datos a la que queremos acceder.
- **Usuario**: Para poder realizar la conexión, es necesario conocer el usuario con el que se puede realizar la conexión, ya que por permisos estará restringido el acceso con otro usuario.
- **Contraseña**: Para asegurar la conexión, y evitar accesos no legítimos, el usuario va acompañado de una contraseña que debemos de añadir en la configuración de conexión.
- **Host**: Nombre **FQDN** (*Fully Qualified Domain Name*) o IP del servidor al que queremos realizar la conexión.
- **Puerto**: Puerto donde está escuchando el serivicio PostgreSQL, que por defecto es el 5432.

Estos datos van a ser utilizados a través del comando "**psycopg2.connect**" para poder realizar las conexiones a los servidores.

A partir de aquí, ya podremos realizar las conexiones que necesitemos, tal como vamos a explicar a continuación.

4.1.1. Conexión local

Tal como se ha dicho previamente, para hacer uso de PostgreSQL en local se ha utilizado el sistema de contenedores **Docker**, haciendo uso de **docker-compose** para levantar los servicios. Para ello se ha utilizado un fichero **compose.yaml** con el siguiente contenido de despliegue:

```
services:
  postgres13:
    image: postgres:13
    restart: always
    environment:
      POSTGRES_PASSWORD: example
    ports:
      - 5432:5432
  pgadmin4:
    image: dpage/pgadmin4:latest
    restart: always
    environment:
      PGADMIN_DEFAULT_EMAIL: user@domain.com
      PGADMIN_DEFAULT_PASSWORD: SuperSecret
    ports:
      - 8088:80
```

Si nuestro sistema está correctamente configurado, si realizamos `docker-compose up` levantará los servicios que hemos configurado:

- **PostgreSQL** versión 13 escuchando en el puerto 5432 del anfitrión.
- **PGAdmin4**: interfaz web para administrar PostgreSQL escuchando en el puerto 8088 del anfitrión.

Para realizar la conexión al servidor local vamos a necesitar conocer los datos de conexión que son:

- **Nombre de la base de datos**: "postgres"
- **Usuario**: "postgres"
- **Contraseña**: ""
- **Host**: "localhost"
- **Puerto**: 5432

Con esos datos, se puede hacer uso del siguiente código python para realizar la conexión:

In [3]:
```
# Para realizar la conexión a servidor local:

# db: el nombre de la base de datos
db = "postgres"

# user: nombre de usuario de la base de datos
user = "postgres"

# password: contraseña de la base de datos
password = "example"

# host: server name
host = "localhost"

# Puerto: puerto de la base de datos: 5432
# (puerto por defecto)
```

```
port = "5432"

# Conectar a la base de datos
con_local = psycopg2.connect(
    dbname=db,
    user=user,
    password=password,
    host=host,
    port=port
)
```

Una vez realizada, vamos a necesitar un "cursor" para poder realizar operaciones en la base de datos:

In [4]:
```
# abrimos un cursor para poder realizar operaciones
cur_local = con_local.cursor()
```

4.1.2. Conexión remota

Para la conexión remota se va a utilizar el proveedor Elephantsql, donde hemos creado una cuenta y se ha realizado el despliegue de una instancia en los servidores de Amazon (AWS).

Para poder realizar la conexión al servidor remoto debemos conocer los siguientes datos:

- **Nombre de la base de datos**: "aaaesjzo"
- **Usuario**: "aaaesjzo"
- **Contraseña**: ""
- **Host**: "abul.db.elephantsql.com"
- **Puerto**: 5432

Para realizar la conexión a esta instancia, usaremos los datos indicados junto con el siguiente código python:

In [5]:
```
# Para realizar la conexión a servidor remoto en ElephantSQL:

# db: el nombre de la base de datos
db = "aaaesjzo"

# user: nombre de usuario de la base de datos
user = "aaaesjzo"

# password: contraseña de la base de datos
password = "tUJ5LuXYxr5_NykC6h_Pdh2bGa0ZGIcC"

# host: server name
host = "abul.db.elephantsql.com"

# Puerto: puerto de la base de datos: 5432
# (puerto por defecto)
port = "5432"

# Conectar a la base de datos
con_remoto = psycopg2.connect(
    dbname=db,
    user=user,
    password=password,
```

```
    host=host,
    port=port
)
```

Al igual que en la conexión local, necesitamos un cursor para realizar operaciones:

In [6]:
```
# abrimos un cursor para poder realizar operaciones
cur_remoto = con_remoto.cursor()
```

4.2. Creación de tablas y vistas

En este apartado vamos a hacer uso del diseño físico que hemos creado en apartados anteriores para desplegarlo en los dos servidores que tenemos actualmente. Para ello, y con intención de no repetir código, se va a asignar todo el código del diseño físico a una variable:

In [7]:
```
tablas = """
CREATE DOMAIN genero_deportista as VARCHAR CHECK(
    VALUE IN ('masculino','femenino','nd')
);


CREATE TABLE DEPORTISTA(
    deportista_id SERIAL not null primary key,
    nombre VARCHAR(20) not null,
    apellidos VARCHAR(30),
    fecha_nacimiento date,
    genero genero_deportista,
    usuario VARCHAR (20) not null,
    password VARCHAR (20) not null
);


CREATE TABLE FEDERACION(
    federacion_id SERIAL not null primary key,
    nombre VARCHAR(50),
    direccion VARCHAR(70),
    web VARCHAR(50)
);


CREATE TABLE DEPORTE(
    deporte_id SERIAL not null primary key,
    nombre VARCHAR(50),
    federacion_id INTEGER not null REFERENCES federacion(federacion_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE ENTRENAMIENTO(
    entrenamiento_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha TIMESTAMPTZ,
    duracion INTEGER,
    distancia INTEGER,
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);
```

```
CREATE TABLE CAMPEONATO(
    campeonato_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha_inicio TIMESTAMPTZ,
    fecha_fin TIMESTAMPTZ,
    federacion_id INTEGER not null REFERENCES federacion(federacion_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE PRUEBA(
    prueba_id SERIAL not null primary key,
    nombre VARCHAR(50),
    distancia INTEGER,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);


CREATE TABLE LOGRO(
    logro_id SERIAL not null primary key,
    nombre VARCHAR(50)
);


CREATE TABLE RETO(
    reto_id SERIAL not null primary key,
    nombre VARCHAR(50),
    descripcion VARCHAR(200),
    fecha_inicio TIMESTAMPTZ,
    fecha_fin TIMESTAMPTZ,
    distancia INTEGER,
    tiempo INTEGER,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    logro_id INTEGER not null REFERENCES logro(logro_id)
      ON DELETE RESTRICT ON UPDATE CASCADE
);


CREATE TABLE EVENTO(
    evento_id SERIAL not null primary key,
    nombre VARCHAR(50),
    fecha_inicio TIMESTAMPTZ,
    distancia INTEGER,
    federacion_id INTEGER not null REFERENCES federacion(federacion_id)
      ON DELETE CASCADE ON UPDATE CASCADE
);


CREATE TABLE PRACTICA(
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    deporte_id INTEGER not null REFERENCES deporte(deporte_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    federado boolean,
    numero_federado INTEGER,
    PRIMARY KEY (deportista_id,deporte_id)
);


CREATE TABLE ASISTE(
```

```
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    evento_id INTEGER not null REFERENCES evento(evento_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    tiempo_realizado INTEGER,
    PRIMARY KEY (deportista_id,evento_id)
);


-- para la competición se necesita que el tiempo (en segundos)
-- tenga decimales
CREATE TABLE COMPITE(
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    prueba_id INTEGER not null REFERENCES prueba(prueba_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    campeonato_id INTEGER not null REFERENCES campeonato(campeonato_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    tiempo_realizado FLOAT,
    PRIMARY KEY (deportista_id,prueba_id,campeonato_id)
);


CREATE TABLE HACE(
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    reto_id INTEGER not null REFERENCES reto(reto_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    fecha_unirse TIMESTAMPTZ,
    fecha_completado TIMESTAMPTZ,
    PRIMARY KEY (deportista_id,reto_id)
);


CREATE TABLE OBTIENE(
    deportista_id INTEGER not null REFERENCES deportista(deportista_id)
      ON DELETE CASCADE ON UPDATE CASCADE,
    logro_id INTEGER not null REFERENCES logro(logro_id)
      ON DELETE RESTRICT ON UPDATE CASCADE,
    fecha TIMESTAMPTZ,
    PRIMARY KEY (deportista_id,logro_id)
);
"""
```

Y lo mismo con las dos vistas que vamos a necesitar en nuestra plataforma:

In [8]:

```
vistas = """
-- Vistas
CREATE VIEW deportista_logro AS
  SELECT d.nombre as deportista_nombre, genero, l.nombre as logro_nombre,
      o.fecha
    FROM deportista AS d, logro AS l, obtiene AS o
    WHERE d.deportista_id = o.deportista_id AND o.logro_id = l.logro_id
;


CREATE VIEW marcas_oficiales AS
  SELECT d.nombre, d.apellidos, c.tiempo_realizado, p.nombre AS prueba,
      cmp.nombre AS campeonato
    FROM deportista d, compite c, prueba p, campeonato cmp
    WHERE d.deportista_id = c.deportista_id
      AND c.campeonato_id = cmp.campeonato_id
      AND c.prueba_id = p.prueba_id
```

```
;
"""
```

4.2.1 Despliegue en local

Vamos a realizar el despliegue de creación de las tablas en el servidor local haciendo uso del cursor creado previamente:

In [9]:
```
# creamos las tablas haciendo uso de la variable creada previamente
cur_local.execute(tablas)
```

Lo mismo para las vistas:

In [10]:
```
# creamos las vistas haciendo uso de la variable creada previamente
cur_local.execute(vistas)
```

Y por último, es obligatorio hacer persistentes los cambios realizados:

In [11]:
```
# Hacemos persistentes los cambios
con_local.commit()
```

4.2.2 Despliegue en remoto

Al igual que el paso anterior, vamos a realizar el despliegue de las tablas y vistas, pero esta vez en una única celda de código, y lógicamente utilizando las variables de conexión del servidor remoto creadas en el paso anterior:

In [12]:
```
cur_remoto.execute(tablas)
cur_remoto.execute(vistas)
con_remoto.commit()
```

4.3. Inserción de datos

Para finalizar con el despliegue, vamos a insertar datos en las bases de datos desplegadas. De esta manera, tendremos datos de prueba para poder realizar el paso siguiente, que es el de las consultas.

Al igual que se ha realizado en el paso anterior, vamos a hacer uso de una variable donde guardaremos los comandos de inserción de datos para no repetir código:

In [13]:
```
inserts = """
INSERT INTO federacion (nombre, direccion, web) VALUES
 ('Real Federación Española de Atletismo',
   'Av. Valladolid, 81, 1º - 28008 Madrid',
   'https://www.rfea.es/'),
 ('Real Federación Española De Ciclismo',
   'C/ Ferraz 16 - 5º Derecha - 28008, Madrid',
   'https://rfec.com/')
;


INSERT INTO deporte (nombre, federacion_id) VALUES
  ('Footing', 1),
  ('Andar', 1),
```

```
  ('Ciclismo', 2),
  ('Atletismo', 1)
;


INSERT INTO deportista (nombre, apellidos, fecha_nacimiento, genero,
    usuario, "password")
  VALUES
  ('Rubén', 'Gómez Olivencia', '1983-03-30', 'masculino',
    'ruben', 'password'),
  ('Asier', 'Gómez Olivencia', '1979-08-22', 'masculino',
    'asier', 'password'),
  ('Rafael', 'Gómez Millán', '1954-04-19', 'masculino',
    'rafa', 'password'),
  ('Carmen', 'Olivencia Cardenas', '1952-02-21', 'femenino',
    'carmen', 'password')
;


INSERT INTO entrenamiento (nombre, fecha, duracion, distancia,
    deportista_id, deporte_id)
  VALUES
  ('Entrenamieto de mañana', '2022-05-15 11:00', 53, 8, 1, 1),
  ('Monte de tarde', '2022-05-16 19:15', 95, 10, 1, 2),
  ('Ciclismo de mañana', '2022-05-17  11:00', 83, 27, 1, 3),
  ('Entrenamieto de mañana', '2022-05-15  11:00', 53, 8, 2, 2),
  ('Monte de tarde', '2022-05-13  11:00', 95, 10, 2, 2),
  ('Ciclismo de tarde', '2022-05-10  19:00', 83, 22, 2, 3),
  ('Arariz', '2022-05-15  8:00', 97, 8, 3, 2),
  ('Monte caramelo', '2022-05-16 8:00', 95, 7, 3, 2),
  ('Arraiz', '2022-05-17 8:00', 83, 6, 3, 2),
  ('Andar', '2022-05-15 11:00', 15, 1, 4, 2),
  ('Andar', '2022-05-16 11:00', 20, 1, 4, 2),
  ('Andar', '2022-05-17 11:00', 28, 3, 4, 2)
;


INSERT INTO campeonato (nombre, fecha_inicio, fecha_fin, federacion_id)
  VALUES
  ('Campeonato de España', '2022-07-15', '2022-07-20', 1),
  ('Campeonato de España cubierto', '2022-11-15', '2022-11-20', 1),
  ('Vuelta a España', '2022-08-30', '2022-09-20', 2)
;


INSERT INTO prueba (nombre, distancia, deporte_id) VALUES
  ('100 metros lisos', 100, 4),
  ('110 metros vallas', 110, 4),
  ('Maratón', 42195, 4),
  ('30  kilómetros velódromo', 3000, 3)
;


INSERT INTO logro (nombre) VALUES
  ('Día de la tierra'),
  ('Empezando bien el año')
;


INSERT INTO reto (nombre, descripcion, fecha_inicio, fecha_fin,
    distancia, tiempo, deporte_id, logro_id)
  VALUES
  ('10 kilómetros en menos de 1 hora',
    'Realizar 10 kilómetros a paso rápido antes de una semana',
```

```
    '2022-05-18', '2022-05-25', 10000, 60, 1, 1),
  ('Correr 5 km', 'Correr 5 kilómetros antes de que termine el mes',
    '2022-05-01', '2022-05-31', 5000, 0, 4, 1),
  ('200 kilómetros en bici',
    'Andar 200 kilómetros antes de que termine el mes',
    '2022-05-01', '2022-05-31', 200000, 0, 3, 1),
  ('10 kilómetros de inicio de año',
    'Realizar 10 kilómetros para empezar bien el año',
    '2022-01-01', '2022-01-07', 10000, 60, 1, 2)
;


INSERT INTO evento (nombre, fecha_inicio, distancia, federacion_id)
  VALUES
  ('Marcha popular', '2022-02-01', 10000, 1),
  ('Carrera popular', '2022-03-01', 5000, 1),
  ('Ciclismo popular', '2022-04-01', 25000, 2),
  ('Ciclismo por la seguridad en carretera', '2022-05-01', 35000, 2)
;


INSERT INTO practica (deportista_id, deporte_id, federado,
    numero_federado)
  VALUES
  (1, 4, true, 1264),
  (2, 3, true, 6745)
;


INSERT INTO asiste (deportista_id, evento_id, tiempo_realizado) VALUES
  (1, 1, 900),
  (1, 2, 240),
  (1, 3, 3600),
  (2, 1, 800),
  (2, 3, 300),
  (3, 1, 1200)
;


INSERT INTO compite (deportista_id, prueba_id, campeonato_id,
    tiempo_realizado)
  VALUES
  (1, 1, 1, 10.432),
  (1, 2, 1, 11.98),
  (1, 1, 2, 10.12),
  (1, 2, 2, 12.04)
;

INSERT INTO hace (deportista_id, reto_id, fecha_unirse,
    fecha_completado)
  VALUES
  (1, 1, '2022-05-19', null),
  (1, 2, '2022-05-02', '2022-05-11'),
  (2, 1, '2022-05-20', '2022-05-21'),
  (3, 1, '2022-05-19', '2022-05-22')
;


INSERT INTO obtiene (deportista_id, logro_id, fecha) VALUES
  (1, 1, '2022-05-19 19:14'),
  (1, 2, '2022-01-03 11:24'),
  (2, 1, '2022-05-21 18:02'),
  (3, 1, '2022-05-22 12:54')
```

```
;
"""
```

4.3.1. Inserción en local

Para realizar la inserción en local se va a hacer uso de las variables previamente creadas, para posteriormente hacer persistentes los cambios

In [14]:
```
# insertamos los datos
cur_local.execute(inserts)
# Hacemos persistentes los cambios
con_local.commit()
```

4.3.2. Inserción en remoto

Se realiza lo mismo para el servidor remoto

In [15]:
```
# insertamos los datos
cur_remoto.execute(inserts)
# Hacemos persistentes los cambios
con_remoto.commit()
```

5. Consultas a base de datos

Una vez realizado el despliegue en los servidores, junto con la inserción de los datos, es el momento de realizar las consultas necesarias para obtener los datos. Las consultas se van a realizar sobre el servidor local, pero dado que en ambos servidores los datos son idénticos, el resultado debe ser el mismo.

5.1. Consultas multitabla

En este apartado se van a realizar consultas utilizando varias tablas, y para ello se va a utilizar el operador **JOIN**.

5.1.1. Mostrar quién ha obtenido un logro concreto

- **Descripción**: En esta consulta queremos mostrar el nombre de los deportistas que han obtenido el logro con identificador **1** y la fecha de cuándo lo han obtenido.
- **Resultado esperado**: El resultado esperado es el de una lista con los nombres de los deportistas que han conseguido el logro con identificador 1, en el que aparece el nombre del logro, y todo ello ordenado por fecha de obtención (de más antiguo a más nuevo).
- **Código de consulta**:

```
SELECT d.nombre, l.nombre as logro, o.fecha
FROM deportista d
  LEFT JOIN obtiene o ON d.deportista_id = o.deportista_id
  LEFT JOIN logro l ON o.logro_id = l.logro_id
```

```
    WHERE l.logro_id  = 1
    ORDER BY fecha;
```

Siendo el resultado el siguiente

```
 nombre |       logro       |         fecha
--------+-------------------+------------------------
 Rubén  | Día de la tierra | 2022-05-19 19:14:00+00
 Asier  | Día de la tierra | 2022-05-21 18:02:00+00
 Rafael | Día de la tierra | 2022-05-22 12:54:00+00
```

En el siguiente paso aparece la consula realizada contra el servidor local

In [16]:
```
# Realizamos la consulta
cur_local.execute("""
SELECT d.nombre, l.nombre as logro, o.fecha
  FROM deportista d
    LEFT JOIN obtiene o ON d.deportista_id = o.deportista_id
    LEFT JOIN logro l ON o.logro_id = l.logro_id
  WHERE l.logro_id  = 1
  ORDER BY fecha;
""")

# Obtenemos los datos
cur_local.fetchall()
```

Out[16]:
```
[('Rubén',
  'Día de la tierra',
  datetime.datetime(2022, 5, 19, 19, 14, tzinfo=datetime.timezone.utc)),
 ('Asier',
  'Día de la tierra',
  datetime.datetime(2022, 5, 21, 18, 2, tzinfo=datetime.timezone.utc)),
 ('Rafael',
  'Día de la tierra',
  datetime.datetime(2022, 5, 22, 12, 54, tzinfo=datetime.timezone.utc))]
```

5.1.2. Mostrar tiempos oficiales en pruebas de campeonatos

- **Descripción**: Mostrar el nombre y apellidos de los deportistas junto con los tiempos oficiales obtenidos en las distintas competiciones y pruebas que han competido.
- **Resultado esperado**: El resultado esperado es el de una lista con los nombres de los deportistas que han participado en distintas pruebas y campeonatos oficiales junto con el tiempo obtenido en cada uno de ellos.
- **Código de consulta**:

```
SELECT d.nombre, d.apellidos, ca.nombre AS campeonato,
p.nombre AS prueba, c.tiempo_realizado
FROM deportista d
  LEFT JOIN compite c ON d.deportista_id = c.deportista_id
  LEFT JOIN campeonato ca ON c.campeonato_id =
ca.campeonato_id
  LEFT JOIN prueba p ON c.prueba_id = p.prueba_id
WHERE tiempo_realizado IS NOT NULL;
```

Siendo el resultado el siguiente

```
 nombre |    apellidos    |           campeonato          |
prueba       | tiempo_realizado
--------+-----------------+-------------------------------+-
-----------------+------------------
 Rubén  | Gómez Olivencia | Campeonato de España          |
100 metros lisos  |           10.432
 Rubén  | Gómez Olivencia | Campeonato de España          |
110 metros vallas |            11.98
 Rubén  | Gómez Olivencia | Campeonato de España cubierto |
100 metros lisos  |            10.12
 Rubén  | Gómez Olivencia | Campeonato de España cubierto |
110 metros vallas |            12.04
```

En el siguiente paso aparece la consula realizada contra el servidor local

In [17]:
```
# Realizamos la consulta
cur_local.execute("""
SELECT d.nombre, d.apellidos, ca.nombre AS campeonato,
    p.nombre AS prueba, c.tiempo_realizado
  FROM deportista d
    LEFT JOIN compite c ON d.deportista_id = c.deportista_id
    LEFT JOIN campeonato ca ON c.campeonato_id = ca.campeonato_id
    LEFT JOIN prueba p ON c.prueba_id = p.prueba_id
  WHERE tiempo_realizado IS NOT NULL;
""")

# Obtenemos los datos
cur_local.fetchall()
```

Out[17]:
```
[('Rubén',
  'Gómez Olivencia',
  'Campeonato de España',
  '100 metros lisos',
  10.432),
 ('Rubén',
  'Gómez Olivencia',
  'Campeonato de España',
  '110 metros vallas',
  11.98),
 ('Rubén',
  'Gómez Olivencia',
  'Campeonato de España cubierto',
  '100 metros lisos',
  10.12),
 ('Rubén',
  'Gómez Olivencia',
  'Campeonato de España cubierto',
  '110 metros vallas',
  12.04)]
```

5.2. Consultas con función de agregación

A la hora de realizar consultas existen funciones de agregación, como son:

- **COUNT**: para contar el número de registros obtenidos.
- **SUM**: para obtener la suma de los valores obtenidos.
- **MAX**: para obtener el valor máximo de los valores obtenidos.
- **MIN**: para obtener el valor mínimo de los varlores obtenidos.

- **AVG**: para obtener el promedio de los valores obtenidos.

También existen ciertas cláusulas a la hora de realizar las consultas:

- **GROUP BY**: se agrupan los resultados obtenidos por el atributo indicado.
- **HAVING**: sirve para escpecificar una condición que debe cumplir cada grupo.

5.2.1. Listar número de logros obtenidos por los deportistas

- **Descripción**: Conocer cuántos logros se han entregado en total.
- **Resultado esperado**: El resultado esperado es conocer el sumatorio de todos los logros virtuales que se han entregado.
- **Código de consulta**:

```
SELECT COUNT(*) AS Logros_entregados FROM obtiene;
```

Siendo el resultado el siguiente

```
 logros_entregados
-------------------
                  4
```

En el siguiente paso aparece la consula realizada contra el servidor local

In [18]:
```
# Realizamos la consulta
cur_local.execute("""
SELECT COUNT(*) AS Logros_entregados
  FROM obtiene;
""")

# Obtenemos los datos
cur_local.fetchall()
```

Out[18]:
```
[(4,)]
```

5.2.2. Listar tiempo mínimo de una prueba concreta

- **Descripción**: Mostrar el nombre y apellido del deportista, junto con el tiempo mínimo oficial obtenido por todos de una prueba concreta oficial.
- **Resultado esperado**: El resultado esperado es el del nombre y apellidos del deportista que tiene actualmente el record de tiempo mínimo para la prueba de 100 metros lisos (con identificador **1**) entre todos los campeonatos disputados.
- **Código de consulta**:

```
SELECT d.nombre, d.apellidos, MIN(tiempo_realizado) AS record
FROM compite c, deportista d
WHERE prueba_id = 1 AND d.deportista_id = c.deportista_id
GROUP BY (d.nombre, d.apellidos);
```

Siendo el resultado el siguiente

```
 nombre |    apellidos     | record
--------+------------------+--------
```

```
    Rubén  | Gómez Olivencia  |  10.12
```

En el siguiente paso aparece la consula realizada contra el servidor local

In [19]:
```
# Realizamos la consulta
cur_local.execute("""
SELECT d.nombre, d.apellidos, MIN(tiempo_realizado) AS record
  FROM compite c, deportista d
  WHERE prueba_id = 1 AND d.deportista_id = c.deportista_id
  GROUP BY (d.nombre, d.apellidos);
""")

# Obtenemos los datos
cur_local.fetchall()
```

Out[19]:
```
[('Rubén', 'Gómez Olivencia', 10.12)]
```

5.3 Cerrar conexiones a la base de datos

Es importante recordar que tras realizar todas las operaciones sobre la base de datos es obligatorio cerrar las conexiones. A continuación se van a cerrar las conexiones realizadas contra la base de datos local y contra la base de datos remota de ElephantSQL.

In [20]:
```
# cerramos cursor y conexión de base de daots local
cur_local.close()
con_local.close()

# cerramos cursor y conexión de base de datos remota
cur_remoto.close()
con_remoto.close()
```

6. Conclusiones

Tal como se ha podido ver a lo largo de todo el documento, el proceso de creación de una base de datos no comienza con la creación de tablas, sino que empieza mucho antes, realizando la toma de requisitos y entendiendo qué datos usa el cliente, y cuál es la relación que tienen esos datos entre sí.

Es de vital importancia realizar una correcta toma de requisitos, ya que ese es el comienzo del proceso a la hora de diseñar una base de datos. Cualquier error, o falta de información que no se haya obtenido en ese punto, se irá propagando por el resto de los pasos que hemos visto en este documento.

Gracias al trabajo del analista de base de datos, y siguiendo todo el procedimiento detallado en este documento, se ha terminado con una base de datos relacional, junto con los datos insertados, para que la empresa **KulunSport** tenga la base de su próximo portal web para deportistas.