Entrega Práctica 3 - Sistemas de Información

Autores:

Fernando Pastor Peralta (897113)
Juan José Muñoz Lahoz (902677)
Marcos San Julián Fuertes (899849)

Fecha de entrega: 16/10/2025

Grupo: D09

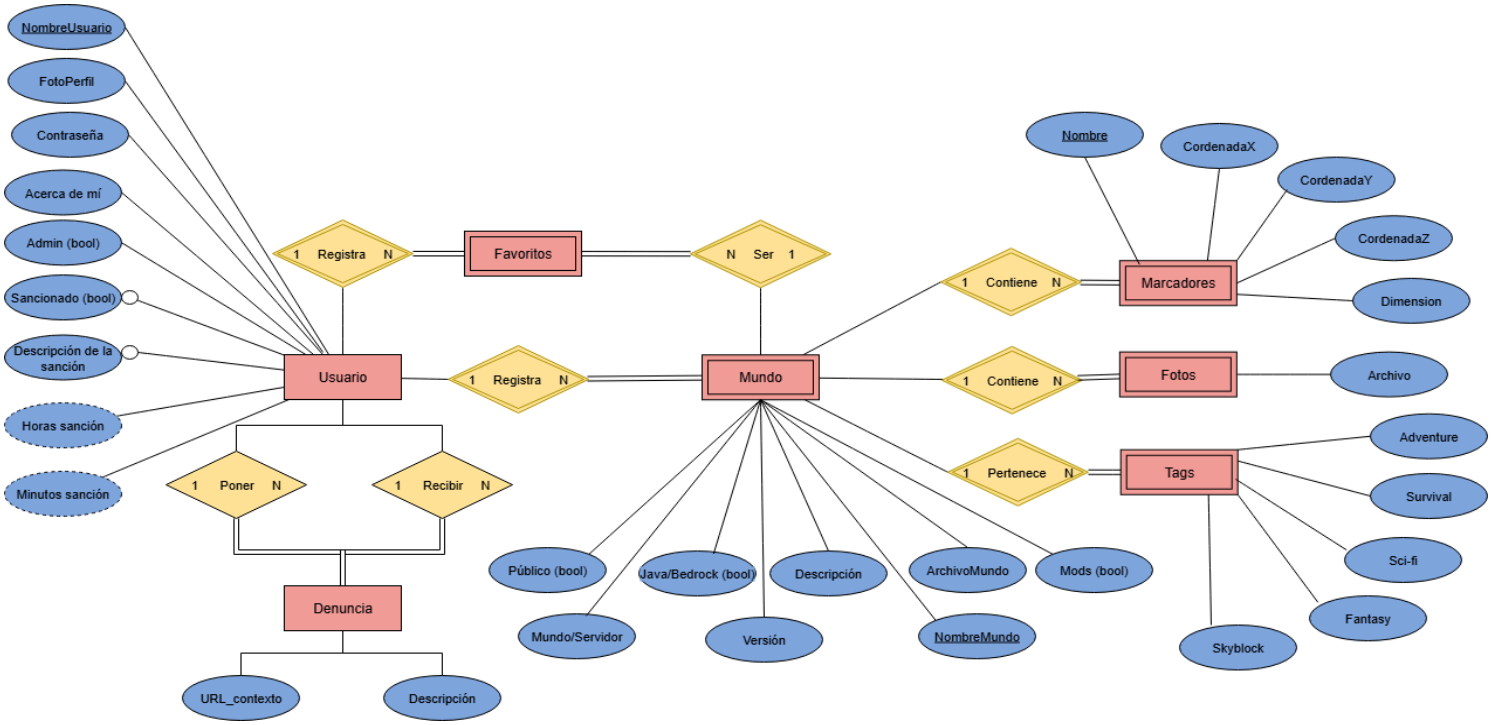
Ingeniería Informática - Universidad de Zaragoza, Curso 2025-2026

Índice

Diseño del modelo:.....3
Diagrama Entidad-Relación:..... 3
Decisiones de diseño.....3
Modelo relacional:.....4
Implementación del modelo lógico en el SGBD:..... 5
Diseño de la capa de persistencia de datos:..... 6
Creación del DAO/VO:.....6
Gestión de conexiones:..... 6
Manejo de errores:.....7
Programa de pruebas:.....7
Metodología y reparto temporal:.....8
Dificultades encontradas:.....8

Diseño del modelo:

Diagrama Entidad-Relación:

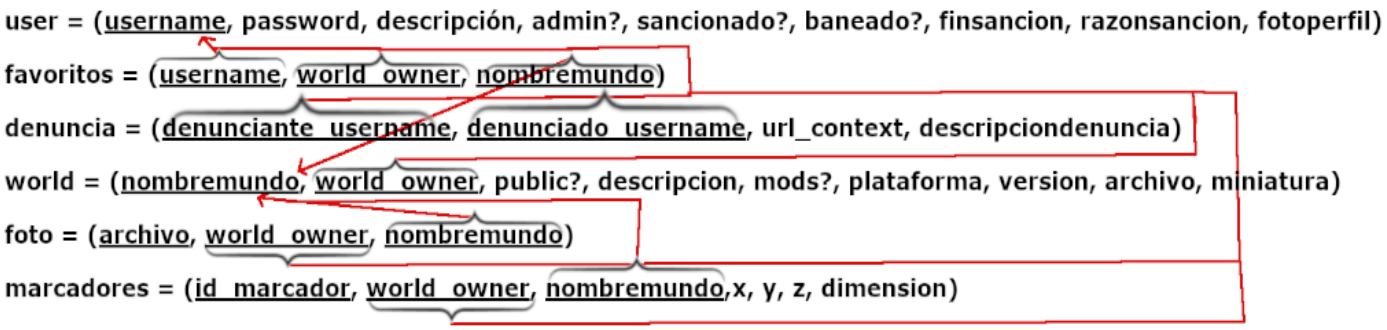


Decisiones de diseño

1. Las tags de los mundos se insertarán en las columnas de cada mundo, para no tener una tabla dedicada a cada tag y así poder consultar de inmediato qué tags tiene activado cada uno de los mundos con una sola consulta.
2. La tabla que registrará todos los favoritos de la base de datos, diferenciará al usuario que ha marcado sus mundos preferidos de los favoritos de los demás, según su clave primaria (la del usuario), y así podrá consultar cualquier mundo que haya marcado. Además, este diseño permitirá consultar cuantas personas han marcado como favorito un mundo en concreto.
3. Las sanciones a los usuarios se guardarán en la misma tabla de usuarios, con una serie de columnas dedicadas a saber si un determinado usuario tiene una sanción o no, por qué y de cuánto tiempo.
4. Las denuncias se diferencian según el denunciante y el denunciado, por lo que solo se podrá poner una denuncia de una persona a otra, si se intenta introducir una nueva denuncia no lo permitiremos hasta que la denuncia anterior sea valorada por un administrador.

Modelo relacional:

El modelo relacional de nuestra base de datos quedaría de la siguiente forma:



Podemos ver que, de primeras, este esquema de modelo relacional parece estar normalizado.
Ahora bien, este ha de estar en la Forma Normal de Boyce-Codd, entonces vamos a ver si hay que modificar algo para que se cumpla:

Primera Forma Normal: Cada uno de los atributos son valores atómicos, es decir, no hay valores multivaluados. Además, cada entidad es única mediante sus respectivas claves primarias. Ambas premisas se cumplen en nuestra base de datos.

Segunda Forma Normal: 1FN + No hay dependencias funcionales con parte de la clave. Esto se cumple en nuestra base de datos ya que no existe ninguna entidad cuya clave primaria dependa de tan solo una parte de la clave primaria de otra entidad.

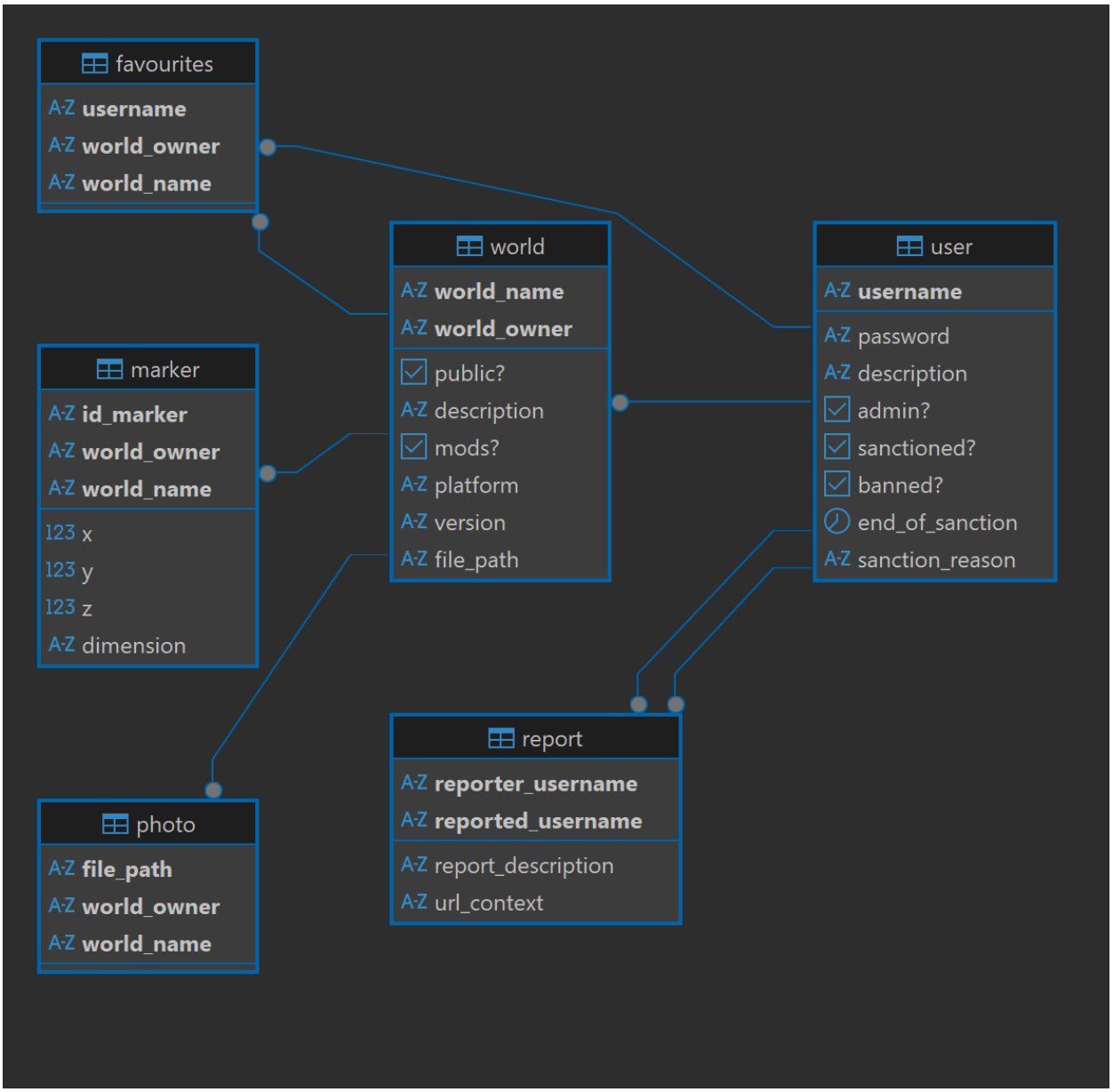
Tercera Forma Normal: 2FN + No hay dependencias funcionales transitivas. Esto se cumple en nuestra base de datos ya que en ninguna tabla existen atributos que dependen de otro atributo que no es clave primaria.

Forma Normal de Boyce-Codd: Todas las dependencias funcionales dependen de la clave. Esto se cumple en nuestra base de datos, ya que todas las entidades cuya clave depende de la clave de otra entidad, siempre dependen de la primaria entera, ni no primaria ni de forma parcial.

Como hemos visto, nuestro esquema relacional ya está normalizado, por ello no necesitaremos realizar una nueva versión del mismo.

Implementación del modelo lógico en el SGBD:

Hemos usado PostgreSQL sobre un Docker, para implementar el modelo relacional antes descrito (las sentencias SQL utilizadas para la creación de tablas, están adjuntas con el código en el .zip en el que se encontraba este mismo PDF). Además, para poder consultar el estado actual de la estructura de la base de datos hemos utilizado DBeaver:



Diseño de la capa de persistencia de datos:

Creación del DAO/VO:

Hemos creado un total de 3 ficheros nuevos:

- UserVO.java: en este fichero se define el VO para la entidad usuario, mediante una clase. En esta clase, encontramos todos sus atributos de SQL representados en java, cada uno con una función getter, encargada de devolver el valor del atributo; y un setter, encargada de establecer el valor de un atributo al indicado.

- JdbcUserDao.java: contiene la clase la cual representa el DAO de la entidad usuario. Está compuesta por 4 funciones, una para cada acción del CRUD:
    - createUser(UserVO user): recibe un dato de tipo UserVO y lo introduce en la base de datos (cada atributo del VO pasa a su columna correspondiente en la BD), devuelve true si el INSERT ha sido correcto, por el contrario, devuelve false.
    - findUser(String username): busca todos los datos de un usuario según el nombre del usuario y los devuelve en un UserVO.
    - updateUser(String username, UserVO user): le pasamos el nombre del usuario al que le queremos cambiar los datos (clave primaria para identificar la fila) y cambiamos todos los datos del usuario por los que contiene el VO user. Devuelve true si el UPDATE ha sido correcto
    - deleteUser(String username): busca un usuario según su nombre de usuario (clave primaria) y borra su fila. Devuelve true si la operación ha sido correcta, false si no.

  Cada una de estas funciones utiliza PreparedStatement para evitar SQL Injection.

Gestión de conexiones:

Estas se realizan gracias a la clase ConnectionManager:

- ConnectionManager.java: abstrae las conexiones con la base de datos, usando el JBDC driver, con la URL que redirecciona a nuestra base de datos y mediante nuestras credenciales (usuario:postgres, contraseña:sisInf, en nuestro caso). Permite obtener conexiones (getConnection) y liberarlas (releaseConnection).

Manejo de errores:

A la hora de crear el DAO para la tabla de usuarios, hemos incluido un bloque try catch que permite capturar las excepciones lanzadas por las funciones de acceso a la base de datos en caso de que la sentencia SQL ejecutada falle. Además, en las funciones que han de devolver un VO de usuario, hemos encapsulado el objeto a devolver dentro de una clase Optional. Así, podemos comprobar si ha fallado o no sin tener que capturar la excepción, solo tenemos que comprobar si el VO devuelto está vacío o no.

Programa de pruebas:

Para probar que nuestra implementación es correcta hemos diseñado un pequeño programa de prueba en el que probamos la creación de los VO user y su DAO, además del uso de los métodos del mismo. Por ejemplo, probamos las funciones de createUser(), findUser(), updateUser() y deleteUser(), las cuales hacen llamadas a los getters de los atributos del VO. En el main() del programa de prueba llamamos a un setter y por último el DAO hace uso del ConnectionManager. Por tanto, todo el código diseñado está probado.

```java
final public class DAO_VO_test {
    public static void main(String[] args) {
        UserVO usuario1 = new UserVO("paco_garcia", "12321", "Soy paco",
false, false, false, null, null, null);
        UserVO usuario2 = new UserVO("maria_gonzalez", "12321", "Soy
maria", true, false, false, null, null, null);
        JdbcUserDao dao = new JdbcUserDao();

        //Comprobamos si los usuarios existen para eliminarlos si es así
y así poder crearlos sin fallos
        Optional<UserVO> u = dao.findUser(usuario1.getUsername());
        if(!u.isEmpty()) {
            dao.deleteUser(usuario1.getUsername());
        }
        u = dao.findUser(usuario2.getUsername());
        if(!u.isEmpty()) {
            dao.deleteUser(usuario2.getUsername());
        }

        dao.createUser(usuario1);
        dao.createUser(usuario2);
```

```
        if(dao.findUser(usuario2.getUsername()).isEmpty()) {
            System.out.println("Error al crear usuario dato.");
        }
        UserVO usuario3 = dao.findUser("paco_garcia").get();

        Optional<UserVO> usuario4 = dao.findUser("maria_gonzalez");
        if(usuario4.isEmpty()) {
             System.out.println("Error al buscar el usuario4");
        }
        usuario1.setBanned(true);
        dao.updateUser(usuario1.getUsername(), usuario1);
        usuario3 = dao.findUser(usuario1.getUsername()).get();
        if(!usuario3.getBanned()) {
            System.out.println("Error al actualizar atributo de
usuario.");
        }
        usuario4 = dao.findUser("francisco_garcia");
        if(usuario4.isEmpty()) {
             usuario1.setUsername("francisco_garcia");
               dao.updateUser(usuario3.getUsername(), usuario1);
               usuario4 = dao.findUser("francisco_garcia");
               if(usuario4.isEmpty()) {
                    System.out.println("Error al actualizar el nombre de
usuario");
               }
        }else {
             dao.deleteUser(usuario4.get().getUsername());
             usuario1.setUsername("francisco_garcia");
               dao.updateUser(usuario3.getUsername(), usuario1);
               usuario4 = dao.findUser("francisco_garcia");
               if(usuario4.isEmpty()) {
                    System.out.println("Error al actualizar el nombre de
usuario");
               }
        }



        dao.deleteUser(usuario2.getUsername());
        if(!dao.findUser(usuario2.getUsername()).isEmpty()) {
            System.out.println("Error al eliminar usuario dato.");
        }
    }
}
```

Metodología y reparto temporal:

Para empezar diseñamos el modelo de nuestra base de datos. Tras hablar sobre algunas decisiones de diseño, empezamos creando el esquema E/R, que costó relativamente poco gracias a los wireframes que creamos en la sesión anterior y a que teníamos una idea bastante completa de lo que queríamos diseñar. Después, desarrollamos el modelo relacional, que como ya comentamos más arriba no nos dio grandes problemas. Tras eso hicimos la implementación; para ello, levantamos un Docker en el que hemos creado la base de datos mediante PostgreSQL…

Para el diseño de la capa de persistencia, usamos Java en el entorno de desarrollo Eclipse para definir el VO y el DAO para la entidad usuario de nuestra base de datos. Después de eso hemos implementado un programa de prueba para verificar el correcto funcionamiento de las 3 clases; el VO, el DAO y el ConnectionManager. Para ello realizamos diferentes operaciones en la base de datos mediante las operaciones del DAO, que hace uso de las operaciones de las otras 2.

Hemos organizado el trabajo de forma que todos hemos trabajado en conjunto, pero de forma que cada uno nos especializamos en alguna parte concreta, por ejemplo, durante la realización de la base de datos, dos íbamos diseñando el esquema E/R, mientras el otro creaba la base de datos dentro del Docker.

En cuanto al tiempo dedicado al trabajo, en conjunto se habrán dedicado alrededor de unas 25 a 30 horas en la realización de la práctica, incluyendo creación de la base de datos, diseño de la capa de persistencia de datos y redacción de la memoria, con un reparto del tiempo más o menos similar entre los integrantes del grupo.

Dificultades encontradas:

1. Búsqueda, análisis y aprendizaje de las tecnologías usadas para el desarrollo de la práctica.
2. Dificultades iniciales en la creación de nuestro primer Docker.
3. Se nos ocurrían cosas faltantes en la base de datos una vez ya estábamos en la fase de diseño de la capa de persistencia de datos, lo que nos hacía modificar

(aunque de forma mínima) todas las partes que formaban la base de datos (esquema E/R, modelo relacional e implementación).
4. Problemas relacionados con desconocimiento básico de algunas funciones de las tecnologías que usábamos, que con una simple búsqueda en Internet se podían suplir.
5. Dudas sobre el diseño de la función update en el DAO, ya que no teníamos claro si hacer que se modificaran todos los atributos de un objeto al mismo tiempo aunque sólo quisieras cambiar uno, o si hacerlo uno a uno. Al final nos decantamos por modificar todos de golpe.