Para explicar la base de datos según las consignas dadas, podemos estructurar la explicación de la siguiente manera:

1. Tabla principal de productos

La tabla principal de productos en esta base de datos es la tabla `funko`. Esta tabla contiene toda la información esencial sobre los productos que se venden en la tienda:

- **funko**
  - `id`: Identificador único del Funko.
  - `personaje`: Nombre del personaje (redundante, ya que existe `nombre_personaje_id`).
  - `nombre_personaje_id`: Identificador del personaje (relacionado con la tabla `personaje`).
  - `serie_id`: Identificador de la serie (relacionado con la tabla `serie`).
  - `categoria_id`: Identificador de la categoría (relacionado con la tabla `categoria`).
  - `precio`: Precio del Funko.
  - `articulo`: Número de artículo.
  - `origen`: Origen del Funko.
  - `descripcion`: Descripción del Funko.
  - `licencia_id`: Identificador de la licencia (relacionado con la tabla `licencia`).
  - `imagen`: URL de la imagen del Funko.
  - `imagen1`: URL de una segunda imagen del Funko.

2. Tabla relacional con algún parámetro referenciado en la tabla principal que requiera datos propios

Un ejemplo de una tabla relacional en esta base de datos es la tabla `serie`, que está referenciada por la tabla `funko` a través del campo `serie_id`:

- **serie**
  - `id`: Identificador único de la serie.
  - `nombre`: Nombre de la serie.
  - `genero`: Género de la serie.

3. Tabla pívot que genere una relación de muchos a muchos entre la tabla principal y otra tabla

La tabla `etiqueta_x_funko` actúa como una tabla pívot que genera una relación de muchos a muchos entre la tabla `funko` (tabla principal) y la tabla `etiqueta`:

- **etiqueta_x_funko**
  - `id`: Identificador único de la relación etiqueta-Funko.
  - `funko_id`: Identificador del Funko (relacionado con la tabla `funko`).
  - `etiqueta_id`: Identificador de la etiqueta (relacionado con la tabla `etiqueta`).

4. Tabla de usuarios que contenga los nombres de usuarios, emails, contraseñas (correctamente cifradas) y permisos de los clientes

La tabla `usuarios` contiene la información requerida para los usuarios del sistema, incluyendo nombres de usuario, emails, contraseñas cifradas y permisos:

- **usuarios**
  - `id`: Identificador único del usuario.
  - `email`: Correo electrónico del usuario.
  - `nombre_usuario`: Nombre de usuario.
  - `nombre_completo`: Nombre completo del usuario.
  - `password`: Contraseña del usuario (encriptada).
  - `rol`: Rol del usuario (`superadmin`, `admin`, `usuario`).

Explicación detallada

Tabla principal: funko

Esta tabla contiene toda la información relevante de los productos que la tienda vende, como el nombre del personaje, la serie a la que pertenece, la categoría, el precio, el origen, la descripción, la licencia y las imágenes.

Tabla relacional: serie

Esta tabla contiene información sobre las series a las que pertenecen los personajes de los Funkos. Cada Funko está asociado a una serie específica a través del campo `serie_id`.

Tabla pívot: etiqueta_x_funko

Esta tabla establece una relación de muchos a muchos entre los Funkos y las etiquetas. Permite que un Funko tenga múltiples etiquetas y que una etiqueta se aplique a múltiples Funkos.

Tabla de usuarios

La tabla de `usuarios` almacena la información de los usuarios que interactúan con el sistema. Incluye campos para el correo electrónico, nombre de usuario, nombre completo, contraseña cifrada y el rol del usuario. Los roles determinan los permisos y el acceso a diferentes funcionalidades del sistema.

Conclusión

Esta base de datos está bien estructurada para soportar una tienda en línea que vende productos Funko, permitiendo la gestión de productos, personajes, series, categorías, licencias, etiquetas, compras y usuarios. Las relaciones entre las tablas permiten una organización eficiente y la capacidad de realizar consultas complejas para obtener la información necesaria.

Explicación código

1. Propiedades

Las propiedades de la clase `Funkos` son privadas, lo que significa que solo se pueden acceder y modificar desde dentro de la clase. Esto es una buena práctica de encapsulamiento para proteger los datos.

```
private $id;
private $personaje;
private $serie_id;
private $categoria_id;
private $precio;
private $articulo;
private $origen;
private $descripcion;
private $licencia_id;
private $imagen;
private $imagen1;
```

2. Getters

Los getters son métodos públicos que permiten acceder a las propiedades privadas desde fuera de la clase. Cada propiedad tiene su propio getter.

```
public function getId() {
    return $this->id;
}
```

3. Métodos CRUD (Create, Read, Update, Delete)

Estos métodos permiten realizar operaciones básicas de base de datos en la tabla `funko`.

- **Create (Insertar)** El método `insert` agrega un nuevo registro a la tabla `funko`.

```
public function insert($nombre_personaje_id, $personaje, $serie_id, $categoria_id, $precio, $articulo, $origen, $descripcion, $licencia_id, $imagen, $imagen1) {
    $conexion = Conexion::getConexion();

    $query = "INSERT INTO funko (personaje, serie_id, categoria_id, precio, articulo, origen, descripcion, licencia_id, imagen, imagen1)
            VALUES (:personaje, :serie_id, :categoria_id, :precio, :articulo, :origen, :descripcion, :licencia_id, :imagen, :imagen1)";

    $PDOstatement = $conexion->prepare($query);

    $PDOstatement->execute([
        'personaje' => $personaje,
        'serie_id' => $serie_id,
        'categoria_id' => $categoria_id,
        'precio' => $precio,
        'articulo' => $articulo,
        'origen' => $origen,
        'descripcion' => $descripcion,
        'licencia_id' => $licencia_id,
        'imagen' => $imagen,
        'imagen1' => $imagen1
    ]);


    return $PDOstatement->rowCount();
}
```

Resumen y Conceptos Clave

1. **Encapsulamiento:** Las propiedades privadas y getters protegen y controlan el acceso a los datos.
2. **Consultas Preparadas:** Protegen contra inyecciones SQL y manejan datos de manera segura.
3. **Métodos CRUD:** Implementan operaciones básicas de base de datos (crear, leer, actualizar, eliminar).
4. **Manejo de Etiquetas:** Añaden, eliminan y actualizan etiquetas para los funkos.
5. **Manejo de Excepciones:** Captura errores para evitar que el programa se detenga inesperadamente.

Relación de Muchos a Muchos

El manejo de etiquetas muestra una relación de muchos a muchos entre Funkos y etiquetas. Aquí hay un desglose detallado:

1. **Relación Muchos a Muchos**:
   - Un Funko puede tener múltiples etiquetas.
   - Una etiqueta puede estar asociada con múltiples Funkos.
2. **Tabla Intermedia (`funko_etiquetas`)**:
   - `funko_id`: referencia al ID del Funko en la tabla `funko`.
   - `etiqueta`: la etiqueta asociada al Funko.

El método `actualizarEtiquetas` primero elimina todas las etiquetas actuales de un Funko (usando `clear_etiquetas`) y luego agrega las nuevas etiquetas (usando `add_etiqueta`). Esto asegura que las etiquetas de un Funko se actualicen correctamente.

```
public function actualizarEtiquetas($id, $etiquetas) {
    $this->clear_etiquetas($id);
    foreach ($etiquetas as $etiqueta) {
        $this->add_etiqueta($id, $etiqueta);
    }
}
```

Propiedades de la Clase

```
private $id;

private $nombre;
```

Métodos de la Clase

`lista_completa()`

```
public function lista_completa(): array
{
    $conexion = Conexion::getConexion();
    $query = "SELECT * FROM etiqueta";

    $PDOstatement = $conexion->prepare($query);
    $PDOstatement->setFetchMode(PDO::FETCH_CLASS, self::class);
    $PDOstatement->execute();
    $reusult = $PDOstatement->fetchAll();

    if (!$reusult) {
        return [];
    };

    return $reusult;
}
```

Este método obtiene todas las etiquetas de la base de datos. Devuelve un array de objetos `Etiqueta`.

1. Se conecta a la base de datos.
2. Prepara la consulta SQL para seleccionar todas las filas de la tabla `etiqueta`.
3. Ejecuta la consulta.
4. Devuelve los resultados como un array de objetos `Etiqueta`.

```
/**
 * Sube un archivo de imagen al directorio especificado.
 *
 * @param string $directorio El directorio donde se debe guardar la imagen.
 * @param array $datosArchivo Los datos del archivo de imagen a subir.
 *
 * @return string El nombre del archivo de imagen subido.
 *
 * @throws Exception Si no se pudo subir la imagen.
 */
public function subirImagen($directorio, $datosArchivo): string
{
    if (!empty($datosArchivo['tmp_name'])) {
        //le damos un nuevo nombre
        $og_name = (explode(".", $datosArchivo['name']));
        $extension = end($og_name);
        $filename = time() . rand(1000, 9999) . ".$extension";

        $fileUpload = move_uploaded_file($datosArchivo['tmp_name'], "$directorio/$filename");
```

```
        if (!$fileUpload) {
            throw new Exception("No se pudo subir la imagen");
        } else {
            return $filename;
        }
    }
}
```

El código define una clase `Imagen` que maneja las operaciones relacionadas con la subida y borrado de imágenes en un servidor. A continuación, se explica cada parte del código:

Propiedades de la Clase

php

Copiar código

```
private $error;
```

Esta es una propiedad privada que se podría utilizar para almacenar mensajes de error, aunque en el código actual no se está utilizando.

Métodos de la Clase

`subirImagen()`

php

Copiar código

```
/**
 * Sube un archivo de imagen al directorio especificado.
 *
 * @param string $directorio El directorio donde se debe guardar la imagen.
 * @param array $datosArchivo Los datos del archivo de imagen a subir.
 *
 * @return string El nombre del archivo de imagen subido.
 *
 * @throws Exception Si no se pudo subir la imagen.
 */
public function subirImagen($directorio, $datosArchivo): string
{
    if (!empty($datosArchivo['tmp_name'])) {
        //le damos un nuevo nombre
        $og_name = (explode(".", $datosArchivo['name']));
        $extension = end($og_name);
        $filename = time() . rand(1000, 9999) . ".$extension";

        $fileUpload = move_uploaded_file($datosArchivo['tmp_name'], "$directorio/$filename");

        if (!$fileUpload) {
            throw new Exception("No se pudo subir la imagen");
        } else {
            return $filename;
        }
    }
}
```

Este método sube un archivo de imagen al directorio especificado y devuelve el nombre del archivo subido.

1. **Parámetros**:
    - `$directorio`: Directorio donde se guardará la imagen.

    - `$datosArchivo`: Datos del archivo de imagen a subir (típicamente `$_FILES['campo']`).
2. **Proceso**:
    - Verifica si el archivo temporal (`tmp_name`) no está vacío.
    - Extrae la extensión del archivo original.
    - Genera un nuevo nombre único para el archivo utilizando `time()` y `rand()`.
    - Mueve el archivo desde el directorio temporal al directorio especificado con el nuevo nombre.
    - Si el movimiento falla, lanza una excepción; si tiene éxito, devuelve el nuevo nombre del archivo.