

0101010
0100101
1101010

UD12.3-

CAPA DE ACCESO A DATOS


Acceso a más de un DAO (repositorios)

Programación – 1 DAW/DAM

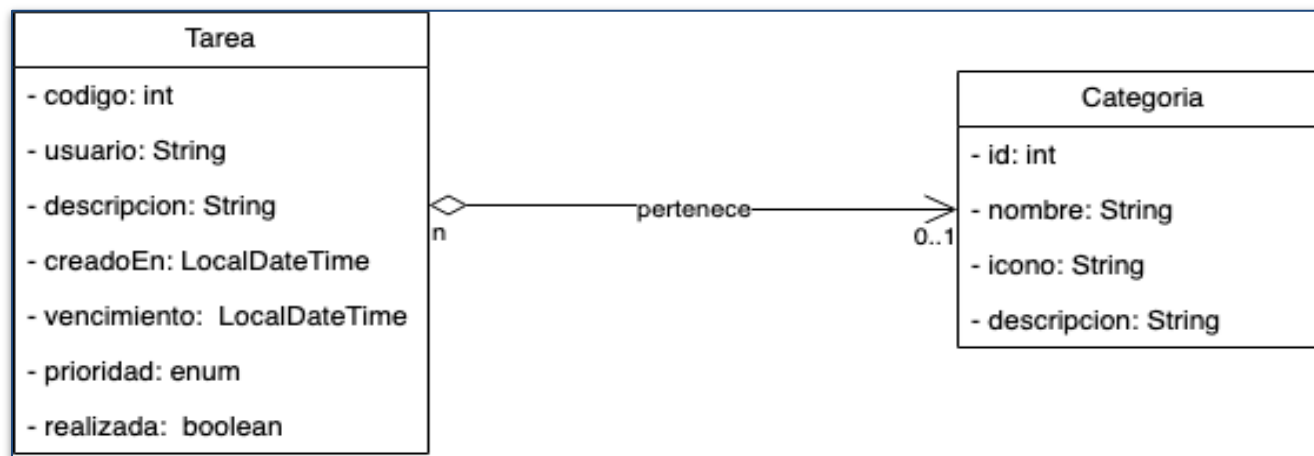
0. ÍNDICE

- INTRODUCCIÓN
- DAO vs REPOSITORY
- REPOSITORIO DE OBJETOS
 - EJEMPLO: REPOSITORIO DE TAREAS
 - DAO (DATA ACCESS OBJECT)
 - IMPLEMENTACIÓN)

1. INTRODUCCIÓN

- En muchas ocasiones necesitamos **acceder** o persistir **objetos con entidades agregadas**
 - Vendedores → Empresa
 - Usuarios → Tweets 
 - Post → Comentario 

En la presentación anterior nos quedó pendiente introducir en el DTO de Tarea su categoría. Este sería otro ejemplo de agregación:



1. INTRODUCCIÓN

```
public class Tarea {  
    private int codigo;  
    private String usuario;  
    private String descripcion;  
    private LocalDateTime creadoEn;  
    private LocalDateTime vencimiento;  
    private Prioridad prioridad;  
    private boolean realizada;  
    private Categoria categoria;
```

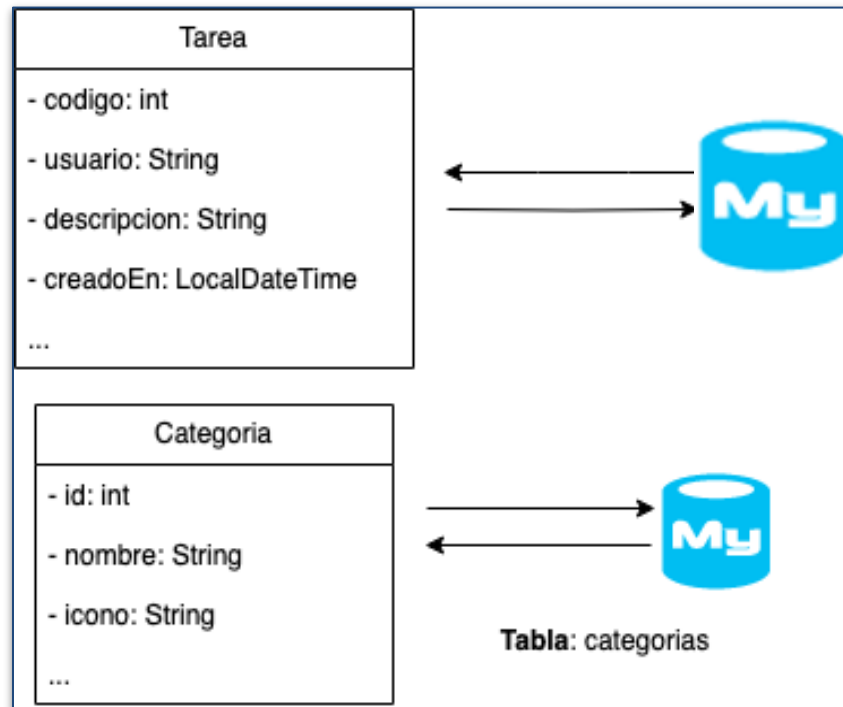
Introducimos una referencia a una categoría en el DTO de Tarea, pero ¿cómo inicializamos la categoría al recuperar una tarea de la BD?

```
    public Tarea(int id, String descripcion, LocalDateTime fechaAlta, boolean  
finalizada, Categoria categoria) {  
        this.id = id;  
        this.descripcion = descripcion;  
        this.fechaAlta = fechaAlta;  
        this.finalizada = finalizada;  
        this.categoria = categoria;  
    }
```

Relación de agregación

2. DAO vs REPOSITORY

- Los objetos DAO, están centrados en el **manejo de entidades básicas (DTO's)** mediante el **acceso al fichero y/o la tabla** que almacena sus datos.
- **Tienen una gran dependencia del sistema de persistencia** que utilicemos (BD relacionales,...).



En nuestras aplicaciones, los datos **no** siempre **son tratados** de forma **aislada**. En este caso, hay una dependencia (agregación) entre **Categoria** con **Tarea**.

2. DAO vs REPOSITORY

- A la hora de implementar el nuevo DAO, que interactúe con la base de datos, podemos pensar hacer...

```
public class SQLTareaDAO implements TareaDAO {
    .
    .
    .
    private Tarea mapToTarea(ResultSet rs) throws SQLException {
        int codigo = rs.getInt("codigo");
        String usuario = rs.getString("usuario");
        String descripcion = rs.getString("descripcion");
        LocalDateTime creadoEn = rs.getTimestamp("fechaCreacion").toLocalDateTime();
        Prioridad priority = Prioridad.fromText(resultSet.getString("prioridad"));
        LocalDateTime vencimiento = rs.getTimestamp("vencimiento").toLocalDateTime();
        boolean realizada = rs.getBoolean("realizada");
        int idCategoria = rs.getInt("categoria");
        Categoria categoria = categoriaDAO.findById(idCategoria);
        return new Tarea(codigo, usuario descripcion, creadoEn, finalizado, categoria);
    }
}
```



- No siempre necesitamos obtener la información de las categorías asociadas.

- Podemos tener múltiples agregados
→ No vamos a cargarlos todos

- No es la responsabilidad de la clase SQLTareaDAO

**NO DEBEMOS HACER LOS DAO's
DEPENDIENTES ENTRE ELLOS**

3. REPOSITORIO DE OBJETOS

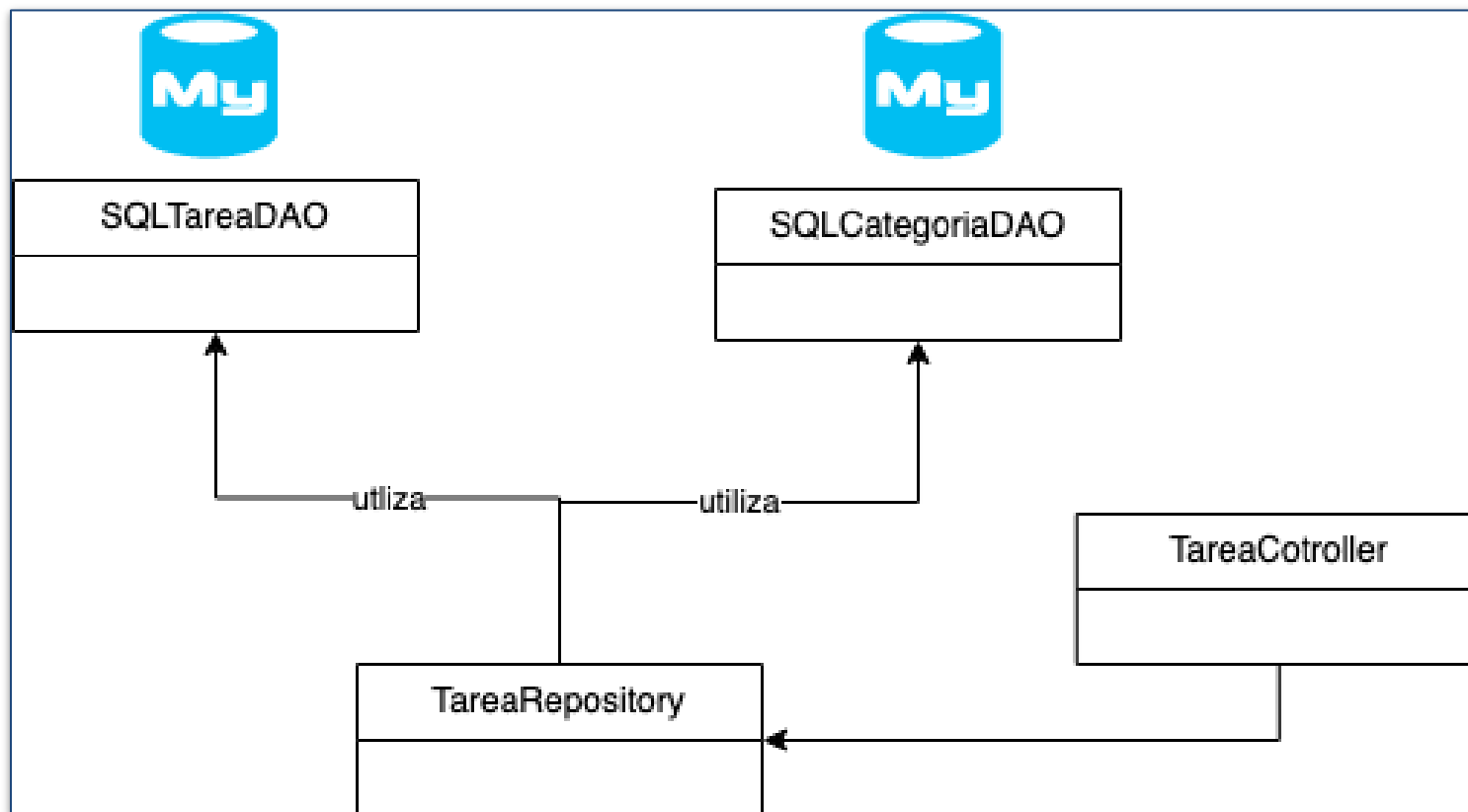
Para llegar a una solución adecuada, vamos a **aclarar el concepto de repositorio**:

*“Un repositorio **encapsula el conjunto de objetos** persistidos en un **almacén de datos** y las **operaciones que podemos realizar sobre ellos**, proporcionando una **concepción más orientada a objetos de la capa de persistencia**”*

- Un **DAO** es **dependiente de la fuente de datos** utilizada (interactúa directamente con ella) y nos permite obtener **objetos simples** haciendo **operaciones básicas CRUD**. Ejemplo.- *SQLTareaDAO*: su fuente es la tabla tareas de una base de datos relacional.
- Los **repositorios** usan los DAO para obtener información, por tanto, están **desacoplados de la fuente de datos**. Permiten construir objetos del dominio junto con sus agregados (datos de las relaciones) y realizar lógica de negocio sencilla. Ejemplo.- *TareaRepository*: ofrece un método save que realizar insert o update según sea el caso. Además, al objeto *Tarea* podremos incluirle la información completa de su relación con *Categoria*.

3.1 EJEMPLO: Repositorio de Tareas

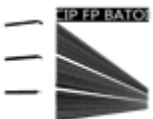
- **Partimos** de que tenemos los **DAO implementados** para usar las entidades (DTO's) de **Tarea** y **Categoria**



3.1.1 DATA ACCESS OBJECT (DAO)

- Al recuperar una tarea en el **DAO** de **Tarea** instanciamos un objeto de la clase **Categoria** que **sólo** contendrá el **id** de la **categoría**. (DEBEREMOS CREAR UN CONSTRUCTOR EN **Categoria** que reciba sólo el id)

```
public class SQLTareaDAO implements TareaDAO {  
  
    . . .  
  
    private Tarea mapToTarea(ResultSet rs) throws SQLException {  
        int codigo = rs.getInt("codigo");  
        String descripcion = rs.getString("descripcion");  
        LocalDateTime fecha = rs.getTimestamp("creadoEn").toLocalDateTime();  
        boolean realizado = rs.getBoolean("realizada");  
        . . .  
        int idCategoria = rs.getInt("categoria");  
        Categoria categoria = new Categoria(idCategoria);  
        return new Tarea(id, descripcion, fecha, finalizado, categoria);  
    }  
}
```



3.1.2 IMPLEMENTACIÓN I

- Nuestro TareaRepository ahora manejará no sólo el DAO de Tarea sino también el de Categoría.
- Los **atributos** serán los **DAO de las entidades** a las que **necesitamos acceder**.

```
public class TareaRepository {
```

```
    private TareaDAO tareaDAO;  
    private CategoriaDAO categoriaDAO;
```

Será **el repositorio el encargado de poblar la información** de la/s entidad/**entidades agregadas**

```
    public Tarea getByIdWithCategories(int id) throws NotFoundException {  
        Tarea tarea = tareaDAO.getById(id);  
        Categoria categoria = categoriaDAO.findById(tarea.getCategoria().getId());  
        tarea.setCategoria(categoria);  
        return tarea;  
    }  
}
```

Asignamos la **Categoría** con todos los datos poblados

Recordemos que la Categoría que **inicializa el DAO solo** tiene asignado el **atributo ID**

3.1.2 IMPLEMENTACIÓN II

- Debemos tener en cuenta que **no siempre necesitamos poblar el objeto agregado**. Por ejemplo, **si en un listado no necesitamos** mostrar la categoría y/o la prioridad asociada, simplemente devolvemos el objeto tarea que **sólo contiene el Id**

```
public class TareaRepository {  
  
    private TareaDAO tareaDAO;  
    private CategoriaDAO categoriaDAO;  
  
    public List<Tarea> findAll() {  
        return tareaDAO.findAll();  
    }  
}
```

Id	Descripción	Fecha creación	Realizada
1	Sacar al perro	21/03/2023	Sí
2	Activ. 1 de ED	08/01/2023	Sí
3	Partida al GTA	11/05/2023	No


3.1.2 IMPLEMENTACIÓN III

- Podemos tener **diferentes métodos** o **parametrizar** el método `findAll` para **poblar las categorías de las tareas** solo cuando lo necesitemos.

```
public class TareaRepository {

    private TareaDAO tareaDAO;
    private CategoriaDAO categoriaDAO;

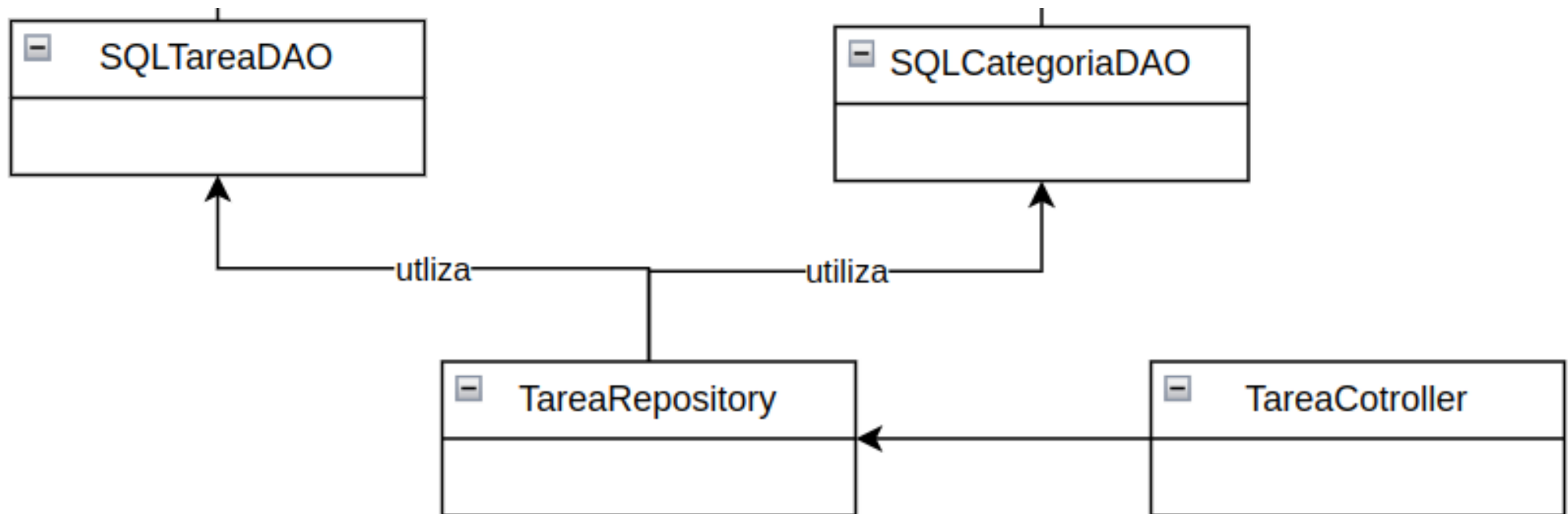
    public List<Tarea> findAllWithCategories() {
        ArrayList<Tarea> tareas = tareaDAO.findAll();
        for (Tarea tarea: tareas) {
            Categoria categoria = categoriaDAO.findById(tarea.getCategoria().getId());
            tarea.setCategoria(categoria);
        }
        return tareas;
    }
}
```



Id	Descripción	Creado en	Realizada	Categoría / Prioridad
1	Sacar al perro	21/03/2023	Sí	Tareas del Hogar
2	Activ. 1 de ED	08/01/2023	Sí	Escuela
3	Partida al GTA	11/05/2023	No	Ocio

3.1.2 IMPLEMENTACIÓN IV

- Gracias a los repositorios, los **cambios** efectuados en los **DAO** serán **transparentes a los controladores** ya que éstos seguirán **accediendo** a los **métodos del repositorio** y nunca a los **de los DAO** para obtener los datos



3.1.2 IMPLEMENTACIÓN V

- Utilizaremos un método u otro en función de las necesidades de la vista y/o función que vamos a llevar a cabo

```
List<Tarea> tareas = tareaRepository.findAll()
```

Id	Descripción	Fecha creación	Realizada
1	Sacar al perro	21/03/2023	Sí
2	Activ. 1 de ED	08/01/2023	Sí
3	Partida al GTA	11/05/2023	No

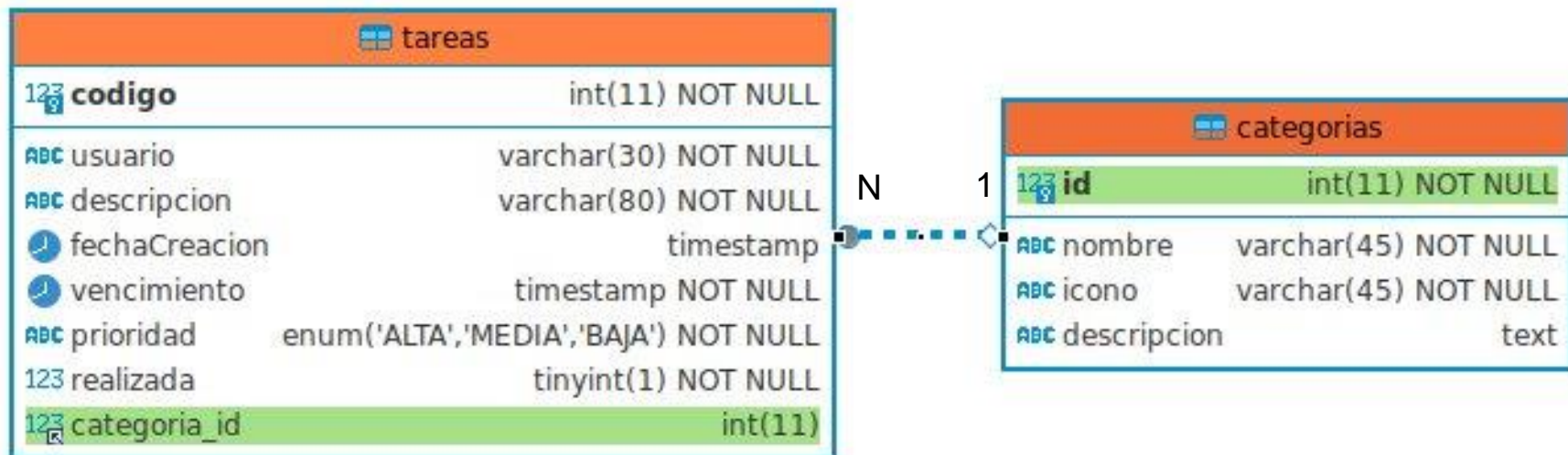
```
List<Tarea> tareas = tareaRepository.findAllWithCategories()
```

Id	Descripción	Creado en	Realizada	Categoría / Prioridad
1	Sacar al perro	21/03/2023	Sí	Tareas del Hogar
2	Activ. 1 de ED	08/01/2023	Sí	Escuela
3	Partida al GTA	11/05/2023	No	Ocio

4. APP TODOLIST CON BASES DE DATOS

- Vamos a añadir a **la versión** de la **app “ToDoList”** que nos quedó tras finalizar la presentación anterior (recuerda que ésta **obviaba el dato de la categoría en las tareas**).
 - El **código fuente** con el **proyecto del que vamos a partir** puedes descargarlo [aquí](#)

4. APP TODOLIST CON BASES DE DATOS I



4. APP TODOLIST CON BASES DE DATOS II

- Crearemos la **nueva entidad** Categoria y la **añadiremos** como **atributo** a la clase Tarea.

```
public class Categoria {  
    private int id;  
    private String nombre;  
    private String descripcion;  
    private String icono;  
    ...  
}
```

```
public class Tarea {  
    private int codigo;  
    private String usuario;  
    private Categoria categoria;  
    private String descripcion;  
    private LocalDateTime creadoEn;  
    ...  
}
```

4. APP TODOLIST CON BASES DE DATOS III

- Creamos la interfaz de **CategoriaDAO** con los **nuevos métodos** que nos permitan **recuperar las categorías** .

```
public interface CategoriaDao {  
    ArrayList<Categoria> findAll();  
    Categoria findById(int id);  
    Categoria getById(int codigo) throws NotFoundException;  
}
```

4. APP TODOLIST CON BASES DE DATOS IV

- Se implementan los **nuevos DAO**, ahora sobre **bases de datos SQL** **SQLTareaDAO** y **SQLCategoriaDAO**.
 - Definimos el método `findAllWithCategories` en **TareaRepository** de forma que:
 - A través de **SQLTareaDAO** se obtendrán las tareas sin categorías
 - Se completarán con **SQLCategoriaDAO**.

```
private Tarea mapToTarea(ResultSet resultSet) throws SQLException {  
    int cod = resultSet.getInt("codigo");  
    ...  
    Categoria categoria = new Categoria(rs.getInt("categoria_id"));  
    return new Tarea(cod, usuario, descripcion, ... categoria);  
}
```

SQLTareaDAO

```
public ArrayList<Tarea> findAllWithCategories() {  
    ArrayList<Tarea> tareas = findAll();  
    for (Tarea item: tareas) {  
        Categoria categoriaPoblada = categoriaDao.findById(item.getCategoria().getId())  
        item.setCategoria(categoriaPoblada);  
    }  
    return tareas;  
}
```

TareaRepository

4. APP TODOLIST CON BASES DE DATOS V

- Si necesitamos mostrar un **listado con las categorías** llamamos al **nuevo método**.
 - Si **no necesitamos la categoría**, seguimos llamando al **anterior**.

Listado de tareas

[Añadir Tarea](#)

Codigo	Usuario	Descripción	Vencimiento	Realizada	Categoría	Acciones
2	Juan	Estudiar Programación	18-05-2023 22:00	No	Escuela	Borrar Detalle
3	Batoi	Hacer la comida	12-05-2023 22:00	Sí	Tareas del Hogar	Borrar Detalle
4	Elena	Podar los setos del jardín	10-05-2023 09:52	No	Jardinería	Borrar Detalle

[Volver al menú principal](#)

Debemos **modificar la vista** para que muestre la tarea incluyendo la nueva columna

```
<td th:text="{{item.categoria.nombre}}"></td>
```

4. APP TODOLIST CON BASES DE DATOS VI

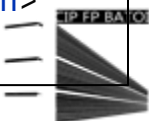
- Necesitamos **añadir a la vista de añadir tareas** un elemento select para **seleccionar la categoría** a la que pertenece.
- Para **mostrar el select** deberemos **pasar a la vista** todas las **categorías disponibles**

```
@GetMapping(value = "/tareas/categorias")
public String tareaListWithCategoryAction(Model model) {
    ArrayList<Categoria> categorias = tareaRepository.findAllCategories();
    model.addAttribute("categorias", categorias);
    return "list_tarea_with_categories_view";
}
```

TareaController

```
<select name="categoria_id" required>
    <option th:each="item : ${categorias}" th:value="${item.id}" th:text="${item.nombre}"></option>
</select>
```

lista_tarea_with_categories_view



- Y ahora sí,

