# Acceso a datos

## Actividad UT 05 - 02
## Uso de Spring Web y Spring Data
## para acceder a base de datos embebida H2

**CONTENIDO**

## 1.- Objetivos

- Ampliar los conocimientos sobre ORM en Java, en concreto Spring Data JPA
- Practicar el desarrollo de modelos de clases siguiendo un enfoque “code first”
- Practicar la creación de relaciones entre clases, y la navegación a través de estas relaciones
- Practicar el uso de repositorios de Spring Data
- Practicar la personalización de consultas
- Practicar la creación de servicios REST / JSON con Spring Web

## 2.- Visión general de la actividad

La actividad está enmarcada dentro de la UT 05 (Herramientas de mapeo objeto-relacional). En esta actividad se creará un modelo de clases, mapeado con JPA y el ORM Hibernate, para luego realizar una API Spring que permita probar el modelo.

## 3.- Equipos de trabajo

Esta actividad se realizará de forma individual. No hay equipos de trabajo.

## 4.- Requisitos previos

- IntelliJ Idea. Se podrá usar cualquier otro entorno, pero las explicaciones y los ejemplos de clase se basan en este entorno. No se recomienda Eclipse, a no ser que sea la versión específica para Spring (Spring Tool Suite). NetBeans tiene herramientas de ayuda al ORM similares a las de IntelliJ.

## 5.- Entregas y evaluación

Se entregará el proyecto completo comprimido en zip. En este tipo de proyecto puede haber ficheros fuera de la carpeta SRC necesarios para la correcta ejecución (ej:pom.xml). También se entregará un documento, un “cuaderno de trabajo” en el que se explique la forma en la que se realizó la actividad.

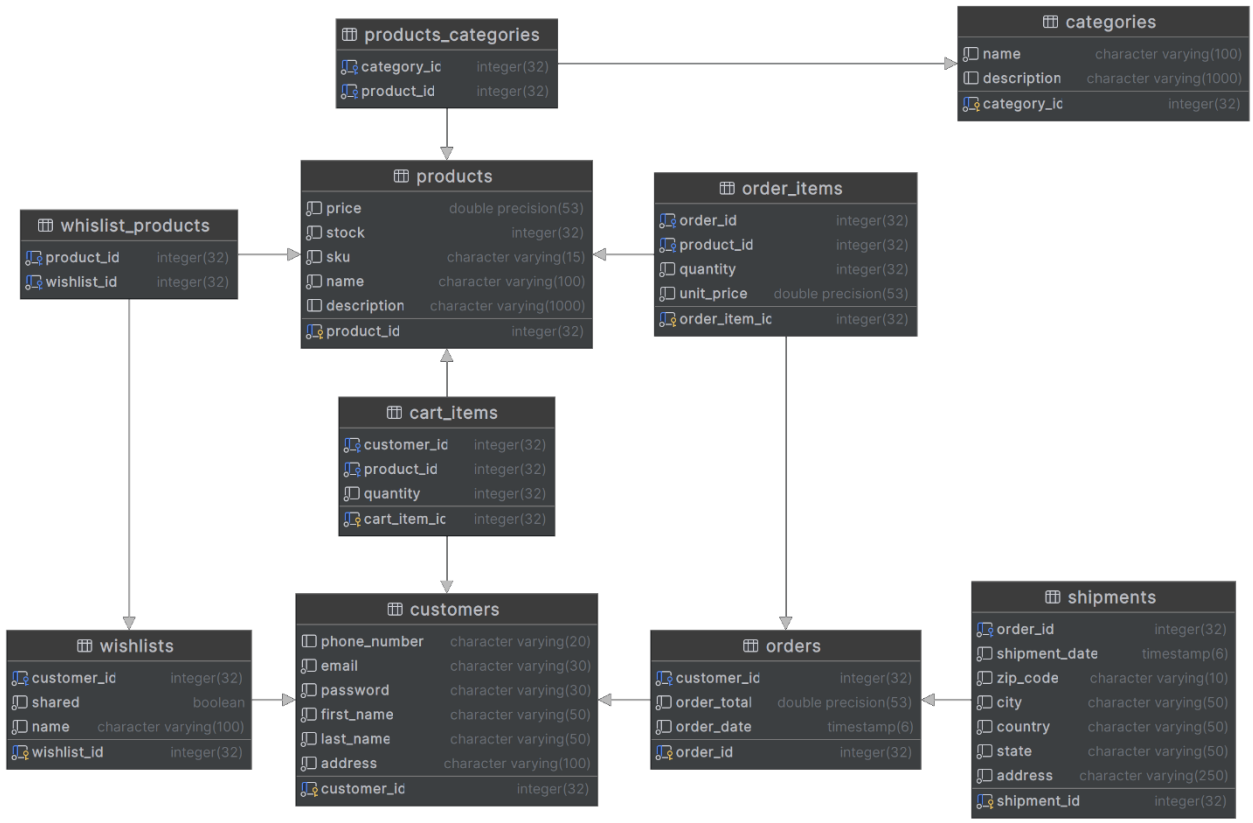
## 6.- Modelo de base de datos

Como se ha indicado, se seguirá un enfoque code first para el desarrollo de esta actividad. Esto implica que primero se creará un modelo de clases, de entidades, y relaciones entre estas entidades, y se dejará que el ORM cree la estructura de bases de datos necesaria.

Se proporciona el modelo de base de datos que se desea crear. Se trata de un modelo físico, con tablas y relaciones, claves ajenas, etc.

En el diseño de las clases, se recomienda la siguiente equivalencia entre tipos H2 (los que aparecen en el modelo de base de datos) y tipos Java:

| Tipo H2 | Tipo Java |
|---|---|
| **integer** | Integer o int |
| **character varying** | String |
| **Timestamp** | LocalDateTime |
| **double precision(nn)** | Double o double |




Leyenda:

- Tabla
- Clave primaria
- Clave ajena.
- Columna obligatoria
- Columna opcional

La actividad debe desarrollarse de forma que se consiga generar una base de datos exactamente igual a la del diagrama. En concreto, se espera:

- Que se respeten los nombres de tablas. La BD generada debe tener las mismas tablas y con el mismo nombre que el modelo.
- Que se respeten los nombres de columnas. Las tablas de las BD generada deben tener las mismas columnas, con el mismo nombre, que el modelo.
- Que se respeten los tipos de datos. Las columnas de cada tabla deben tener el tipo de datos representado en el modelo.
- Que se respeten las columnas obligatorias (not null) y opcionales (nulables). Todas las columnas de todas las tablas son obligatorias, salvo:
    - Tabla “customers”, columna “pone_number”.
    - Tabla “products”, columna “description”.
    - Tabla “categories”, columna “description”.
- Que se creen las relaciones (claves ajenas) indicadas en el modelo. En el modelo las relaciones se representan con una flecha que va de la tabla “dependiente” a la tabla “principal”.
- Que todas las relaciones sean obligatorias, salvo:

  - Tabla “orders”. Un pedido (order) puede no haberse enviado, por lo que no estará relacionado con un envío (shipments) hasta que se haga el envío. Es una relación 1 – 0..1.
- Que se creen algunas restricciones / valores por defecto que no aparecen en el modelo:
  - Tabla “cart_items”: debe tener un índice único para que no se pueda añadir dos veces el mismo producto al carrito de un cliente. Si se intenta añadir de nuevo un producto que ya está en el carrito, se debe incrementar el número de unidades.
  - Tabla “order_items”: debe tener un índice único para que no se pueda añadir dos veces el mismo producto al pedido de un cliente.
  - Tabla “orders”: la columna “order_date” debe tomar por defecto, al crearse, el valor de la fecha y hora del momento de la creación del pedido.
  - Tabla “shipments”: la columna “shipment_date” debe tomar por defecto, al crearse, el valor de la fecha y hora del momento del envío del pedido.
- Todas las columnas de clave primaria serán autoincrementales / identity.

## 7.- Actividades a realizar

### *7.1.- Creación de proyecto Spring Boot*

Crear un proyecto Spring Boot con Spring Initializr, IntelliJ o cualquier otro entorno. El proyecto debe cumplir con los siguientes requisitos:

- Lenguaje Java
- Versión de Java 17, 19, o 23. No usar una versión distinta a las indicadas.
- SDK Amazon Coretto, de la versión adecuada. No usar un SDK distinto al indicado (OpenJDK, Oracle SDK, etc.).
- Dependencias necesarias:
  - Spring Web (servicios REST)
  - Spring Data JPA
  - H2
- Dependencias recomendadas:
  - Lombok
- Estructura de paquetes. Como mínimo, el proyecto debe contener paquetes independientes para separar y organizar los siguientes elementos:
  - Entidades
  - Repositorios
  - Servicios
  - Controladores
  - DTO, si se considera necesario el uso de alguna clase de DTO
  - Pueden añadirse más paquetes si se considera necesario
- La nomenclatura de las clases y paquetes debe ser la adecuada. Respetar convenciones, y utilizar los nombres más adecuados para el modelo de base de datos propuesto.

### *7.2.- Configuración de BD y consola de H2 en el proyecto*

Configurar el proyecto (application.properties o application.yml) para:

- Que utilice una base de datos en memoria H2. El nombre de la base de datos debe ser “onlineshop”. No se usará usuario o contraseña. Se recomienda configurar la cadena de conexión para que genere los objetos en la base de datos en minúsculas. Por defecto, H2 lo hace en mayúsculas.
- Que se habilite la consola de H2.
- Que se difiera la ejecución del fichero data.sql hasta la creación del modelo (tablas).

### *7.3.- Creación de entidades y verificación del modelo creado*

Crear las entidades necesarias para que se creen las tablas y relaciones de la base de datos.

Verificar que el modelo creado es igual que el que figura en el punto 6, y que se han creado todos los elementos indicados (defaults, índices únicos, etc.).

En el caso de que no se cumplan los requisitos, añadir / personalizar todas las anotaciones necesarias para conseguir un modelo de base de datos igual al indicado. Revisar atentamente los requisitos del punto 6.

### *7.4.- Creación de script de inicialización*

Crear un script SQL para dar valores iniciales a:

- Al menos 4 categorías
- Al menos 20 productos. Los productos, en lo que respecta a su categoría:
  - 10 productos pertenecerán a una categoría.
  - 10 productos pertenecerán a 2 categorías.
- Al menos cuatro clientes.
- Al menos un pedido en uno de los clientes, con tres productos distintos.
- Al menos una lista de deseos en un cliente, con tres productos.
- Al menos cuatro productos añadidos al carro de la compra.

Al ser una base de datos en memoria, cada vez que se arranque el sistema se eliminarán todos los datos. Configurar el proyecto para que al arrancar ejecute el script de inicialización para la carga de datos, de forma que tengamos datos para pruebas.

Se deben usar nombres de productos, categorías y clientes realistas. No usar asdasdas, loren ipsum o similares.

### *7.5.- Implementación de los distintos servicios REST / JSON*

Creación de repositorios, servicios, controladores, DTO, y cualquier otro elemento necesario para poder realizar peticiones para las tareas indicadas a continuación.

A la hora de implementar los servicios hay que tener en cuenta algunas cosas:

- Se debe utilizar una variedad de las técnicas aprendidas en clase: repositorios y sus métodos, transacciones, consultas derivadas, consultas JPQL, consultas SQL nativo, proyecciones, etc. Se creará un documento (más sobre esto en el punto 7.6) donde se registren las técnicas utilizadas.
- Los métodos de los controladores deben usar el método indicado. En cada operación se ha indicado el método que se debe usar.
- Los servicios tienen que devolver el código de respuesta HTTP adecuado. Esto es, por ejemplo:
  - Si se intenta listar los productos del carro de compra de un cliente que no existe, debe devolver un 404 (not found), con un mensaje similar a “No existe el cliente con el código …”. Si lo que ocurre es que el carro de la compra no tiene nada, no es un error, deberá devolver una colección vacía.
  - Si se crea un nuevo pedido, o se crea una lista de deseos, debe devolverse un 201 (created).
  - Si se solicita cualquier clase de información y todo va bien, se deberá devolver un 200 (ok).
  - Si se solicita realizar un envío de un pedido que ya ha sido enviado, se deberá devolver un 400 (bad request) con un mensaje similar a “El pedido … ya había sido enviado”
  - Etc.
- En algunas operaciones no se producirán errores. Por ejemplo, vaciar un carrito que ya está vacío, o quitar un producto del carrito que no está en el carrito, o similares, el método simplemente

funcionará como si se hubiera realizado la operación, y funcionará correctamente. Sólo en los casos en los que esta forma de operar tenga sentido.

### 7.5.1.- Servicios relacionados con productos

Crear un controlador (@RestController), y hacer que responda a las URL que comiencen por “/api/products".

Crear los métodos necesarios para implementar las siguientes funcionalidades:

- Listado paginado de productos (GET /api/products/{page}/{pageSize}).
  - Recibe: datos de paginación: página que se pide, y tamaño de la página.
  - Devuelve: lista con una página de productos, en orden alfabético por su nombre. Se debe incluir en el resultado los datos de las categorías a las que pertenece cada producto.
  - Utiliza un método personalizado (consulta derivada) que se encarga de paginar y ordenar.
- Búsqueda de productos (GET /api/products/search/{query}/{page}/{pageSize}).
  - Recibe: lo mismo que el método anterior (página y tamaño de página), y, además, una cadena de búsqueda.
  - Devuelve: lista con una página de productos, en orden alfabético por su nombre, pero sólo muestra los productos que contengan en su nombre o descripción la búsqueda realizada por el usuario.
  - Utiliza un método personalizado (consulta derivada) que se encarga de buscar, paginar y ordenar.

### 7.5.2.- Servicios relacionados con el carrito de compra

Crear un controlador (@RestController), y hacer que responda a las URL que comiencen por “/api/cart".

Crear los métodos necesarios para implementar las siguientes funcionalidades:

- Listado del carrito de compra (GET /api/cart/{customerId}).
  - Recibe: el id del cliente.
  - Devuelve: lista con todos los productos en el carrito de la compra del usuario, incluido el número de unidades, ordenado por nombre del producto. El resultado no debe ser sólo una lista de productos, con precio, unidades, etc. Tiene que incluir también el importe total del carrito.
  - Utiliza un método personalizado (consulta derivada) que se encarga de buscar y ordenar. Realizará proyección sobre un DTO específico.
- Añadir un producto al carrito del usuario (POST /api/cart/{customerId})
  - Recibe: id del cliente (URL), DTO con id de producto y las unidades (@RequestBody).
  - Devuelve: lo mismo que el listado del carrito de compra.
  - Añade cierta cantidad de unidades del producto al carrito. Si el producto ya está en el carrito, incrementa las unidades en la cantidad indicada.
- Quitar producto del carrito (DELETE /api/cart/{customerId}/{productId}).
  - Recibe: id del cliente, id del producto a quitar.
  - Devuelve: lo mismo que el listado del carrito de compra.
  - Elimina un producto del carrito del usuario.
- Vaciar carrito (POST /api/cart/empty/{customerId}).
  - Recibe: el id del cliente (@PathVariable).
  - Devuelve: lo mismo que el listado del carrito de la compra.
  - Vacía completamente el carrito del usuario.

### 7.5.3.- Servicios relacionados con listas de deseos

Crear un controlador (@RestController), y hacer que responda a las URL que comiencen por "/api/wishlists".

Crear los métodos necesarios para implementar las siguientes funcionalidades:

- Listado de listas de deseos (GET /api/wishlists/list/{customerId}).
  - Recibe: el id del cliente.
  - Devuelve: una lista con todas las listas de deseos del usuario.
- Crear lista de deseos (PUT /api/wishlists/{customerId}).
  - Recibe: el id del cliente (URL), DTO con nombre lista, y si es compartida (@RequestBody).
  - Devuelve: la lista de deseos creada.
  - Crea una nueva lista de deseos vinculada al cliente.
- Eliminar lista de deseos (DELETE /api/wishlists/{wishlistId}):
  - Recibe: el id de la lista de deseos que se quiere eliminar.
  - Devuelve: nada. Sólo un código HTTP de éxito o de error.
  - Elimina la lista de deseos. Si la lista no está vacía tendrá que lanzar un error.
- Listado de lista de deseos (GET /api/wishlists/{wishlistId}).
  - Recibe: el id de la lista de deseos
  - Devuelve: lista con todos los productos en la lista de deseos.

### 7.5.4.- Servicios relacionados con pedidos

Crear un controlador (@RestController), y hacer que responda a las URL que comiencen por "/api/orders".

Crear los métodos necesarios para implementar las siguientes funcionalidades:

- Completar pedido (POST /api/orders/create/{customerId}).
  - Recibe: el id del cliente
  - Devuelve: el pedido creado
  - Convierte el carrito de compra en pedido. Se generará un registro en "orders" y varios registros en "order_items". El registo de "orders" quedará pendiente de envío, sin relacionarse con un registro de "shipments".
  - Este proceso vacía el carrito de la compra tras haber generado el pedido.
  - También modifica el stock de los productos, para reducir la cantidad disponible. Debe verificar que hay suficientes unidades de cada producto para poder completar el proceso.
  - Los datos que necesite (p. ej. precio de productos) los tendrá que obtener de la BD, no los puede recibir como entrada.
  - Este proceso se tiene que hacer en transacción, no se pueden convertir "medio" carrito de la compra. O se convierte todo o no se convierte. Y, por ejemplo, tampoco se puede completar el pedido si no se puede cambiar el stock de los productos.
- Enviar pedido (POST /api/orders/send/{orderId}).
  - Recibe: el id de un pedido (URL). Un DTO con los datos necesarios para hacer el envío: dirección, código postal, ciudad, estado, etc. (@RequestBody)
  - Devuelve: el envío generado.
  - Genera un envío (shipment) para un pedido del usuario. El envío creado debe quedar asociado al pedido.

## *7.6.- Cuaderno de trabajo*

Se debe entregar, junto con el proyecto, un documento en formato PDF en el que se recoja una explicación de como se ha desarrollado la práctica, estructura del proyecto, qué técnicas se han utilizado, en qué clases y métodos, y por qué se ha decidido utilizarlas. La extensión del documento no es importante, pero tiene que ser claro y redactado correctamente.