**Java y JPA: consultas avanzadas, rendimiento y modelos complejos**

¿Comenzando en esta etapa? Aquí puedes descargar los archivos del proyecto.

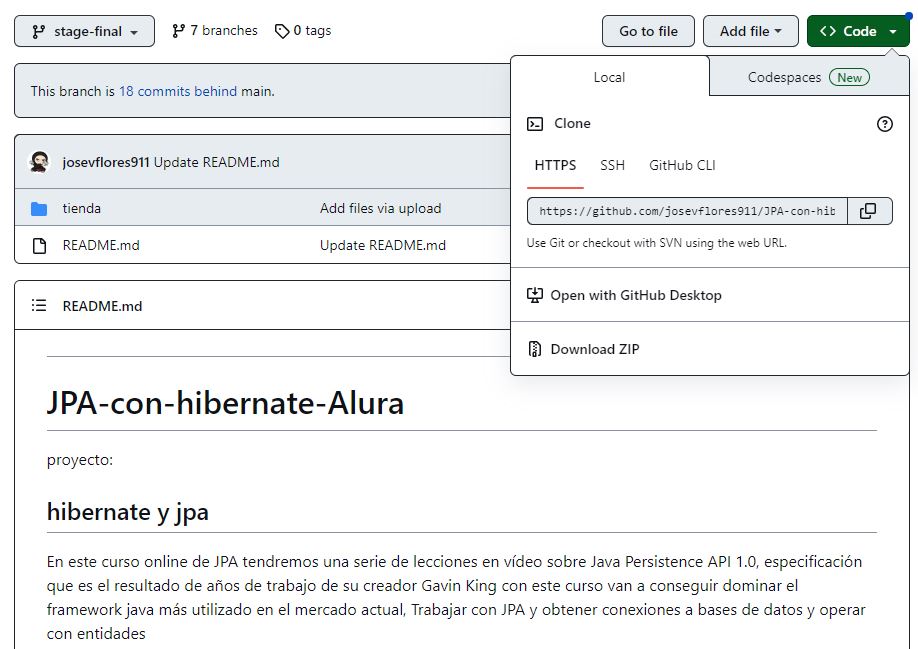
[Descargue los archivos en Github](https://github.com/alura-cursos/JPA-con-hibernate-Alura-/tree/stage-final) o haga clic [aquí](https://github.com/alura-cursos/JPA-con-hibernate-Alura-/archive/refs/heads/stage-final.zip) para descargarlos directamente.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-proyecto-anterior/83953/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83953/next)

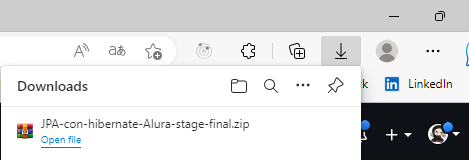
**03Preparando el ambiente**

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83952/next)

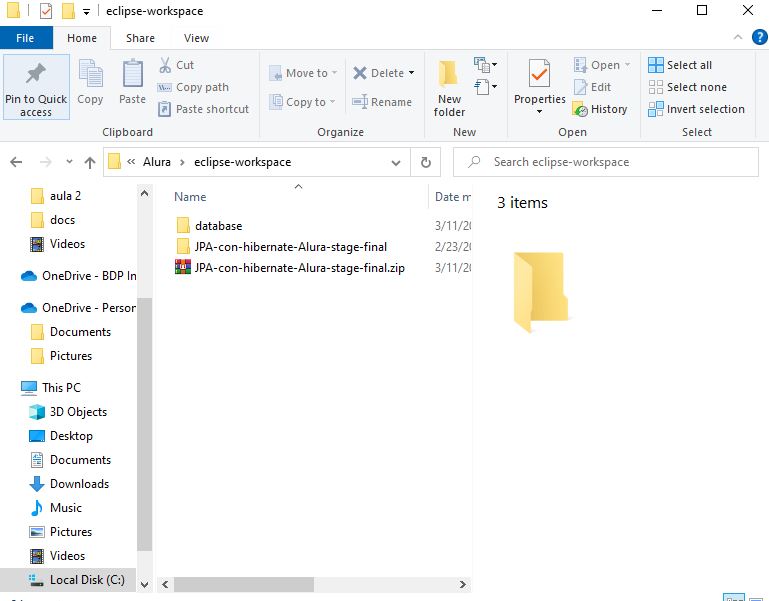
Para comenzar esta segunda parte del curso necesitamos descargar la versión del curso anterior que podemos encontrar en el siguiente [click aqui](https://github.com/alura-cursos/JPA-con-hibernate-Alura-/tree/stage-final" \t "_blank).



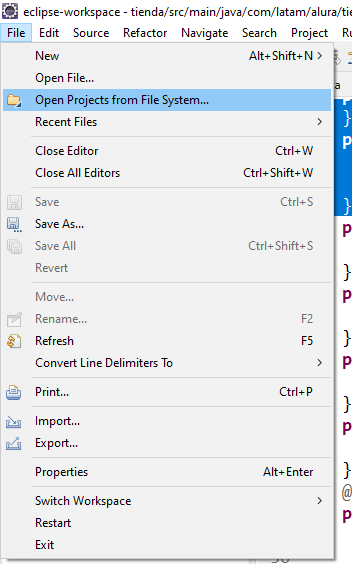
* En la imagen superior podemos ver como se encuentra el repositorio vamos a descargar el archivo .zip.

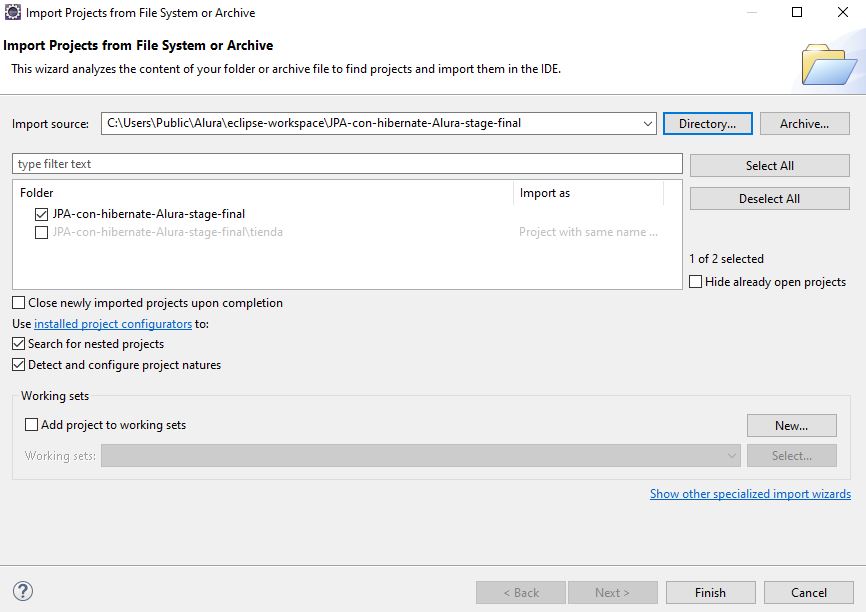


* Luego vamos a extraer este archivo en la carpeta donde se encuentra el resto de las informaciones de jpa para este curso.



* Luego de haber descargado la información del curso anterior vamos a abrir la IDE de eclipse, seleccionar en la pestaña de nuevo abrir un proyecto existente, seleccionamos el archivo del directorio y finalizar.





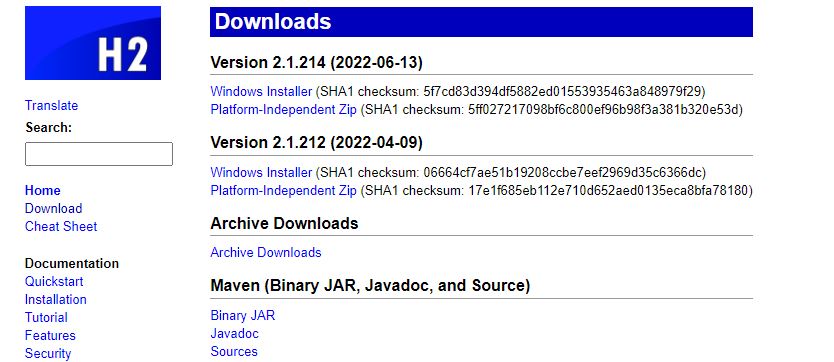
**Configurando H2 database**

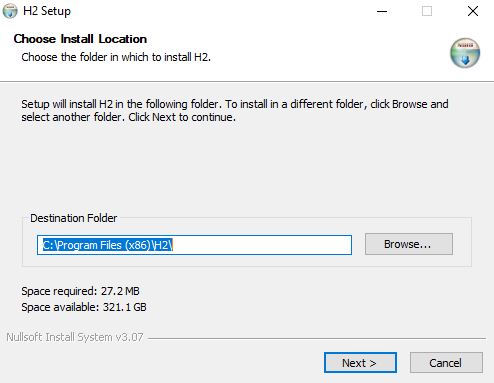
[**https://www.h2database.com/html/download.html**](https://www.h2database.com/html/download.html)

En la parte anterior almacenamos la información en la base de datos H2 en memoria y vimos las querys para cada construcción para esto no precisamos descargar el software de H2.

En esta parte del curso vamos a ver las tablas y los registros almacenados, luego de haber ejecutado nuestra API de tienda\_alura.

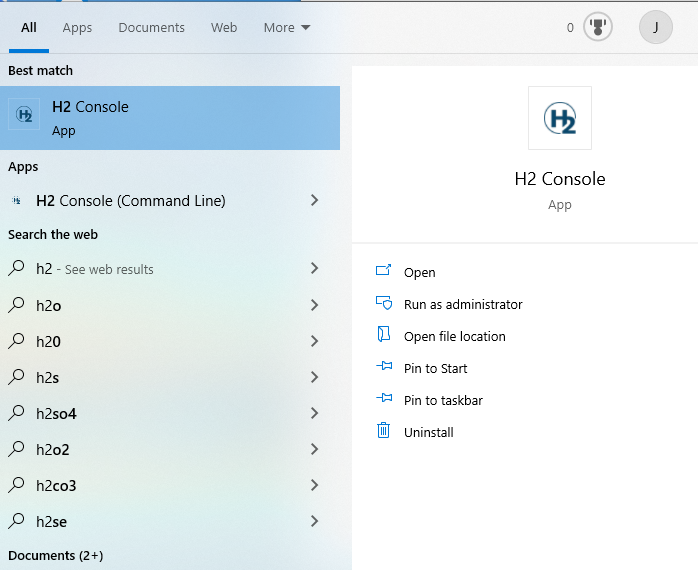
* Para esto necesitamos descargar el software de H2 click aquí en windows seleccionamos windows installer e instalamos el software.



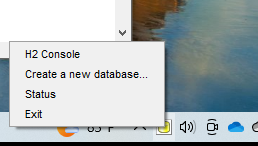


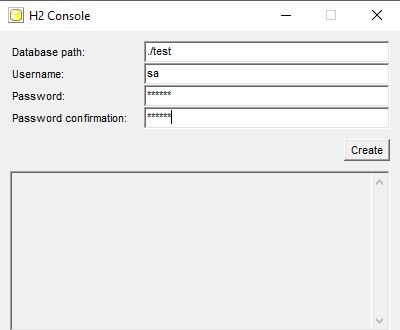
* Para configurar la url de la base de datos vamos a ejecutar el software de h2 y en el panel de inicio, vamos a hacer click derecho sobre el icono de la base de datos seleccionamos crear una nueva base de datos.

**1)**

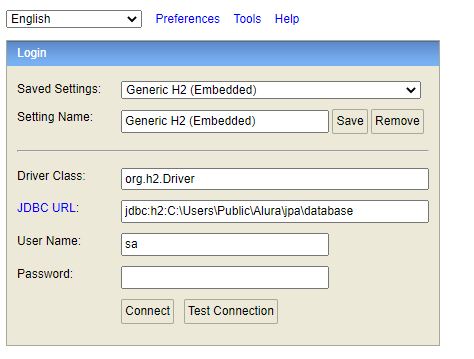


**2)**





* En la ruta de la base de datos el último nombre va a ser el nombre de la base de datos y el restante las carpetas del directorio.
* Luego vamos a ejecutar la aplicación de h2 y modificar los valores con los de la base de datos que acabamos de crear (url,usuario,password) y seleccionamos connect.



[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-preparando-el-ambiente/83952/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83952/next)

# 10Parametro mappedBy

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83942/next)

¿Por qué se debe usar el parámetro mappedBy en la anotación @OneToMany?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Para indicar el nombre de la tabla de Join.

* Alternativa correta

Para indicar que se trata de una relación bidireccional.

Sin este parámetro, JPA considera que es otra relación, asumiendo que hay otra tabla de unión.

* Alternativa correta

Tal parámetro debe establecerse solo en la anotación @ManyToOne.

* Alternativa correta

Para indicar el nombre de la columna de Join.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-parametro-mappedby/83942/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83942/next)

# 13Parámetro de cascade

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83943/next)

¿Cuál es el uso del parámetro cascade?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Propagar las operaciones realizadas en una entidad en su relación.

Cascade crea un efecto dominó en las operaciones realizadas en una entidad.

* Alternativa correta

Indicar qué operaciones se pueden realizar en una relación.

Las operaciones posibles son indicadas en el DAO.

* Alternativa correta

Indicar el atributo inverso en una relación bidireccional.

El atributo inverso lo indica el parámetro mappedBy.

* Alternativa correta

Indicar que una relación es bidireccional.

Mapeamentos bidireccionales se hacen a traves del parametro mappedBy en la anotacion @oneToMany en conjunto con @manyToOne .

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-parametro-de-cascade/83943/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83943/next)

# 14Haga lo que hicimos en aula: más sobre relacionamientos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83959/next)

En esta primera aula realizamos una modificación en el archivo persistence.xml donde configuramos las propiedades que nos dan acceso a la base de datos. En el curso anterior habíamos configurado la base de datos en la memoria, en esta parte vamos a almacenar los datos en un archivo local que nos va a permitir observar las modificaciones que estamos realizando.

<persistence-unit name="tienda" transaction-type="RESOURCE\_LOCAL">

<properties>

<property name="javax.persistence.jdbc.driver" value="org.h2.Driver"/>

<property name="javax.persistence.jdbc.url" value="jdbc:h2:C:\Users\Public\Alura\jpa\database"/>

<property name="javax.persistence.jdbc.user" value="sa"/>

<property name="javax.persistence.jdbc.password" value="1234"/>

<property name="hibernate.dialect" value="org.hibernate.dialect.H2Dialect"/>

<property name="hibernate.show\_sql" value="true"/>

<property name="hibernate.format\_sql" value="true"/>

<property name="hibernate.hbm2ddl.auto" value="create-drop"/>

</properties>COPIA EL CÓDIGO

* Vamos a crear dos nuevas entidades que nos van a permitir ampliar nuestro modelo inicial donde registramos un producto por categoría, ahora podremos registrar los pedidos que realizan diversos clientes de estos productos, por lo tanto vamos a crear la entidad Pedido y la entidad cliente que estarán relacionados con la anotación ManyToOnedonde un cliente tendrá muchos pedidos similar a la entidad Producto con Cliente.

@Entity

@Table(name="pedidos")

public class Pedido {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

private LocalDate fecha=LocalDate.now();

@Column(name="valor\_total")

private BigDecimal valorTotal;

@ManyToOne

private Cliente cliente;

@OneToMany(mappedBy="pedido", cascade=CascadeType.ALL)

private List<ItemsPedido> items=new ArrayList<>();

public void agregarItems(ItemsPedido item) {

item.setPedido(this);

this.items.add(item);

}

public Pedido(Cliente cliente) {

this.cliente = cliente;

}

public Pedido() {}

//getters / setters

}

@Entity

@Table(name="clientes")

public class Cliente {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

private String nombre;

private String dni;

public Cliente() {}

public Cliente(String nombre, String dni) {

this.nombre=nombre;

this.dni=dni;

}

//getters / setters

}COPIA EL CÓDIGO

* Para conseguir relacionar los pedidos de los clientes a los productos podemos crear un atributo con la entidad @ManyToMany y jpa crea automaticamente una nueva entidad, en nuestro caso tenemos que crear una nueva entidad y realizar un relacionamento bidireccional para que esa entidad intermedia contenga atributos auxiliares.

@Entity

@Table(name="items\_pedido")

public class ItemsPedido {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

private int cantidad;

private BigDecimal precioUnitario;

@ManyToOne

private Producto producto;

@ManyToOne

private Pedido pedido;

public ItemsPedido() {}

public ItemsPedido(int cantidad, Producto producto, Pedido pedido) {

this.cantidad = cantidad;

this.producto = producto;

this.pedido = pedido;

this.precioUnitario=producto.getPrecio();

}

}COPIA EL CÓDIGO

* Como vamos a guardar nuevos registros debemos crear clases DAO para estas nuevas entidades excepto para ItemPedido donde JPA reconoce que es un relacionamiento bidireccional que conecta un grupo de entidades con otro.

public class PedidoDao {

private EntityManager em;

public PedidoDao(EntityManager em) {

this.em = em;

}

public void guardar(Pedido pedido) {

this.em.persist(pedido);

}

…

COPIA EL CÓDIGO

public class ClienteDao {

private EntityManager em;

public ClienteDao(EntityManager em) {

this.em = em;

}

public void guardar(Cliente cliente) {

this.em.persist(cliente);

}

…

COPIA EL CÓDIGO

public class RegistroDePedido {

public static void main(String[] args) throws FileNotFoundException {

registrarProducto();

EntityManager em = JPAUtils.getEntityManager();

ProductoDao productoDao = new ProductoDao(em);

Producto producto = productoDao.consultaPorId(1l);

ClienteDao clienteDao = new ClienteDao(em);

PedidoDao pedidoDao = new PedidoDao(em);

Cliente cliente = new Cliente("Juan","k6757kjb");

Pedido pedido = new Pedido(cliente);

pedido.agregarItems(new ItemsPedido(5,producto,pedido));

em.getTransaction().begin();

clienteDao.guardar(cliente);

pedidoDao.guardar(pedido);

em.getTransaction().commit();

}

…}

COPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-mas-sobre-relacionamientos/83959/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83959/next)

# 15Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83964/next)

En esta aula, aprendiste:

* Mapeo de nuevas entidades en la aplicación según el modelado de la base de datos.
* Mapeo de una relación con cardinalidad de muchos a muchos.
* Mapeo de una relación bidireccional.
* Cómo persistir entidades que tienen relaciones bidireccionales.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-lo-que-aprendimos/83964/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83964/next)

# 01Proyecto del aula anterior

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83954/next)

¿Comenzando en esta etapa? Aquí puedes descargar los archivos del proyecto que hemos avanzado hasta el aula anterior.

[Descargue los archivos en Github](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/tree/stage_1) o haga clic [aquí](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/archive/refs/heads/stage_1.zip) para descargarlos directamente.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-proyecto-del-aula-anterior/83954/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83954/next)

# 05 Generando informes / Relatórios

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83944/next)

Analice el siguiente código:

select new br.com.alura.tienda.dto.VentasPorDia(producto.nombre, sum(pedido.valorTotal), pedido.fecha)

from Pedido pedido

join pedido.items items join items.producto producto

group by produto.fecha, pedido.fechaCOPIA EL CÓDIGO

Y la siguiente clase DTO:

public class VentasPorDia {

private String nombreDelProduto;

private BigDecimal valorVendido;

private LocalDate FechaDeVentas;

//getters e setter de todos os atributosCOPIA EL CÓDIGO

¿Por qué la consulta anterior no funcionará correctamente?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

La clase DTO debe contener un constructor default.

* Alternativa correta

La clase DTO debe contener un constructor compatible con consultas JPQL.

JPA crea instancias de la clase DTO a través de un constructor que recibe parámetros, según la consulta JPQL.

* Alternativa correta

La clase DTO no debe contener métodos getter y setter.

Los métodos getter y setter no interfieren con nada.

* Alternativa correta

La clase DTO debe contener solo atributos primitivos de Java.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-generando-informes-relatorios/83944/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83944/next)

# 07Ventajas de las NamedQueries

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83945/next)

¿Cuál de las siguientes es una de las ventajas de utilizar NamedQueries?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Aumentar el rendimiento de las consultas.

* Alternativa correta

Permitir realizar consultas que filtren por atributos de relaciones de una entidad.

* Alternativa correta

Permitir consultas usando select new.

* Alternativa correta

Acercar las consultas de una entidad a su clase.

Las NamedQueries se declaran en la clase de entidad.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-ventajas-de-las-namedqueries/83945/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83945/next)

# 08Haga lo que hicimos en aula: consultas avanzadas

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83960/next)

En esta aula vamos a realizar consultas utilizando recursos de JPA que nos permiten realizar consultas con funciones de agregación tipo SUM(), AVG(), MIN(),... antes de proceder debemos agregar unos últimos detalles en la clase pedido y ítem pedido que nos permiten obtener el valor total.

* Primero vamos a asignar el nombre de la columna con la anotación @Column(name=”columna”).

@Column(name="valor\_total")

private BigDecimal valorTotal=new BigDecimal(0);COPIA EL CÓDIGO

* Vamos a agregar un nuevo método que nos permite relacionar itemPedido con Pedido, así como Pedido con ItemPedido. Además de las anotaciones tenemos que asignar el valor correcto en nuestro relacionamiento.

public void agregarItems(ItemsPedido item) {

item.setPedido(this);

this.items.add(item);

this.valorTotal= this.valorTotal.add(item.getValor());

}COPIA EL CÓDIGO

* En la clase itemPedido agregamos un método que nos permite calcular el valor total a partir del precio y de la cantidad.

public BigDecimal getValor() {

return this.precioUnitario.multiply(new BigDecimal(this.cantidad));

}COPIA EL CÓDIGO

* Por último en el DAO vamos agregar nuestros métodos de consulta. Vamos a utilizar las funciones de agregación para consultar el valor total en la base de datos con la función SUM().

public BigDecimal valorTotalVendido() {

String jpql= "SELECT SUM(p.valorTotal) FROM Pedido p";

return em.createQuery(jpql,BigDecimal.class).getSingleResult();

}COPIA EL CÓDIGO

* Y también el valor promedio con la función AVG().

public Double valorPromedioVendido() {

String jpql= "SELECT AVG(p.valorTotal) FROM Pedido p";

return em.createQuery(jpql,Double.class).getSingleResult();

}COPIA EL CÓDIGO

* Otro tipo de consulta es cuando queremos obtener en nuestra consulta atributos de múltiples entidades en una única consulta e incluso funciones de agregación para eso tenemos dos(2) métodos un crearemos un método que retorna una lista de Objetos para formar el relatorio.

public List<Object[]> relatorioDeVentas(){

String jpql="SELECT producto.nombre, "

+ "SUM(item.cantidad), "

+ "MAX(pedido.fecha) "

+ "FROM Pedido pedido "

+ "JOIN pedido.items item "

+ "JOIN item.producto producto "

+ "GROUP BY producto.nombre "

+ "ORDER BY item.cantidad DESC";

return em.createQuery(jpql,Object[].class).getResultList();

}COPIA EL CÓDIGO

* Y la forma más recomendada es mediante la construcción de una clase VO (value object) es una clase que nos permite enviar información dentro de nuestra aplicación.

public class RelatorioDeVenta {

private String nombreDelProducto;

private Long CantidadDeProducto;

private LocalDate FechaDeUltimaVenta;

public RelatorioDeVenta(String nombreDelProducto, Long cantidadDeProducto, LocalDate fechaDeUltimaVenta) {

this.nombreDelProducto = nombreDelProducto;

CantidadDeProducto = cantidadDeProducto;

FechaDeUltimaVenta = fechaDeUltimaVenta;

}

public String getNombreDelProducto() {

return nombreDelProducto;

}

public void setNombreDelProducto(String nombreDelProducto) {

this.nombreDelProducto = nombreDelProducto;

}

…

public List<RelatorioDeVenta> relatorioDeVentasVO(){

String jpql="SELECT new com.latam.alura.tienda.vo.RelatorioDeVenta(producto.nombre, "

+ "SUM(item.cantidad), "

+ "MAX(pedido.fecha)) "

+ "FROM Pedido pedido "

+ "JOIN pedido.items item "

+ "JOIN item.producto producto "

+ "GROUP BY producto.nombre "

+ "ORDER BY item.cantidad DESC";

return em.createQuery(jpql,RelatorioDeVenta.class).getResultList();

}COPIA EL CÓDIGO

* En una clase de prueba vamos a conseguir visualizar el resultado de las consultas.

**Método 1)**

BigDecimal valorTotal=pedidoDao.valorTotalVendido();

System.out.println("Valor Total: "+ valorTotal);

List<Object[]> relatorio = pedidoDao.relatorioDeVentasVO();

for(Object[] obj:relatorio){

System.out.println(obj[0]);

System.out.println(obj[1]);

System.out.println(obj[2]);

}COPIA EL CÓDIGO

**Método 2)**

List<RelatorioDeVenta> relatorio = pedidoDao.relatorioDeVentasVO();

relatorio.forEach(System.out::println);COPIA EL CÓDIGO

* Para organizar las consultas podemos utilizar un recurso de JPA @NamedQueries que nos permite colocar una determinada consulta dentro de la entidad para indicar que esa consulta tiene un uso particular.

@Entity

@Table(name="productos")

@NamedQuery(name="Producto.consultarPrecioPorNombre", query="SELECT P.precio FROM Producto AS P WHERE P.nombre=:nombre")

public class Producto{...COPIA EL CÓDIGO

* Aun cuando se encuentra indicada en la entidad el método continúa en la clase DAO y la consulta se encuentra en la entidad.

public BigDecimal consultarPrecioPorNombreDeProducto(String nombre) {

return em.createNamedQuery("Producto.consultarPrecioPorNombre",BigDecimal.class).setParameter("nombre", nombre).getSingleResult();

}COPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-consultas-avanzadas/83960/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83960/next)

# 09Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83965/next)

En esta aula, aprendiste:

* Realización de consultas utilizando funciones de agregación como min, max, avg y sum;
* Cómo escribir consultas de informes usando JPQL;
* Cómo usar el recurso seleccionado nuevo en consultas JPQL;
* Realización de consultas mediante consultas con nombre.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-lo-que-aprendimos/83965/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83965/next)

# 01Proyecto del aula anterior

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83955/next)

¿Comenzando en esta etapa? Aquí puedes descargar los archivos del proyecto que hemos avanzado hasta el aula anterior.

[Descargue los archivos en Github](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/tree/stage_2) o haga clic [aquí](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/archive/refs/heads/stage_2.zip) para descargarlos directamente.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-proyecto-del-aula-anterior/83955/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83955/next)

# 04Consultas con Lazy / Eager

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83946/next)

Analice el siguiente código:

@Entity

@Table(name = “clientes”)

public class Cliente {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String nombre;

@OneToOne(fetch = FetchType.EAGER)

private Residencia residencia;

@OneToMany(fetch = FetchType.LAZY, mappedBy = “cliente”)

private List<Telefono> telefonos = new ArrayList<>();

}--------------------------------------------------------------------------------------------

@Entity

@Table(name = “residencias”)

public class Residencia{

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String calle;

private String numero;

private String ciudad;

private String estado;

private String barrio;

}--------------------------------------------------------------------------------------------

@Entity

@Table(name = “telefonos”)

public class Telefono {

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String codigoDeArea;

private String numero;

@ManyToOne

private Cliente cliente;

}COPIA EL CÓDIGO

y el siguiente código

Cliente cliente = em.find(Cliente.class, 1l);

System.out.println(cliente.getNombre());COPIA EL CÓDIGO

¿Qué tipo de consulta de base de datos generará el fragmento de código anterior?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Un select en la tabla de clientes que se une a las tablas de direcciones y teléfonos con un Join.

* Alternativa correta

Un select en la tabla de clientes que hace un Join con la tabla direcciones.

La relación con la entidad Dirección es Eager, por lo que la consulta generará un join con la tabla de direcciones.

* Alternativa correta

Un select en la tabla de clientes que hace un Join con la tabla teléfono.

* Alternativa correta

Solo un select en la tabla de clientes.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-consultas-con-lazy-eager/83946/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83946/next)

# 05Ventajas de Join Fetch

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83947/next)

¿Por qué es interesante utilizar el recurso JOIN FETCH en una consulta JPQL?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Para reducir el tiempo de ejecución de una consulta.

* Alternativa correta

Para evitar cargar relaciones en todas las consultas de aplicaciones.

JOIN FETCH permite elegir qué relaciones se cargarán en una consulta determinada, en lugar de cargarlas siempre.

* Alternativa correta

Para forzar la carga de una relación en todas las consultas de la aplicación.

La función de JOIN FETCH solo afecta a las consultas para las que se utiliza.

* Alternativa correta

Para permitir la carga de relaciones bidireccionales.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-ventajas-de-join-fetch/83947/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83947/next)

# 06Haga lo que hicimos en aula: performance de consultas

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83961/next)

Al construir nuestra aplicación mediante el uso de recursos que realizan operaciones que no se encuentran explicitas debemos estudiar la documentación para entender cual es la ciencia detrás del framework o recurso.

Cuando realizamos consultas con la anotación @ManyToOne o @OneToOne detrás de escena JPA aplica una estrategia de cargamento de información llamada Eager o Anticipada o Proactiva realizando JOINS entre tablas. Pero no acaba allí, ya que si esa entidad que tiene el JOIN, tiene otras entidades dentro de sus atributos marcados con la anotación finalizando ToOne, también serán cargadas dentro de la consulta. Esto puede saturar la memoria y afectar seriamente la velocidad de carga, ya para corregir debemos utilizar el parámetro de carga FerchType.LAZY en todas las anotaciones ToOne que le indica a JPA solo cargar la entidad si es solicitada.

public class Producto{

…

@ManyToOne(fetch=FetchType.LAZY)

private Categoria categoria;COPIA EL CÓDIGO

public class Pedido {

…

@ManyToOne(fetch=FetchType.LAZY)

private Cliente cliente;COPIA EL CÓDIGO

public class ItemsPedido {

…

@ManyToOne(fetch=FetchType.LAZY)

private Producto producto;

@ManyToOne(fetch=FetchType.LAZY)

private Pedido pedido;COPIA EL CÓDIGO

* Al realizar esta corrección se presenta un posible inconveniente donde nos encontremos con la necesidad de utilizar ese atributo de entidad. Pero para ese momento ya el EntityManager se puede encontrar cerrado, por lo que tenemos que planear nuestras consultas previniendo, el uso de esa entidad aún cuando se encuentre cerrado el EntityManager.

public Pedido consultarPedidoConCliente(Long id) {

String jpql="SELECT p FROM Pedido p JOIN FETCH p.cliente WHERE p.id=:id";

return em.createQuery(jpql,Pedido.class).setParameter("id", id).getSingleResult();

}

public class PruebaDeDesempenho {

public static void main(String[] args) throws FileNotFoundException {

// LoadRecords.cargarRegistros();

EntityManager em = JPAUtils.getEntityManager();

PedidoDao pedidoDao = new PedidoDao(em);

Pedido pedidoConCliente = pedidoDao.consultarPedidoConCliente(2l);

em.close();

// System.out.println(pedido.getFecha());

// System.out.println(pedido.getItems().size());

System.out.println(pedidoConCliente.getCliente().getNombre());

}

}COPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-performance-de-consultas/83961/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83961/next)

# 07Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83966/next)

En esta aula, aprendiste:

* Cómo funcionan las estrategias EAGER y LAZY, en consultas de entidades que tienen relaciones;
* Por qué JPA podría lanzar LazyInitializationException en ciertas situaciones;
* Buenas prácticas en la carga de entidades con relaciones;
* Cómo realizar consultas programadas con la función de búsqueda de combinación.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-lo-que-aprendimos/83966/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83966/next)

# 01Proyecto del aula anterior

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83956/next)

¿Comenzando en esta etapa? Aquí puedes descargar los archivos del proyecto que hemos avanzado hasta el aula anterior.

[Descargue los archivos en Github](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/tree/stage_3) o haga clic [aquí](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/archive/refs/heads/stage_3.zip) para descargarlos directamente.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-proyecto-del-aula-anterior/83956/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83956/next)

# 03 Consulta dinamica

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83948/next)

Analice el siguiente código:

public List<Cliente> buscarClientes(String nombre, LocalDate fechaNacimiento) {

String jpql = "SELECT c FROM Cliente c WHERE ";

if (nombre != null && !nombre.trim().isEmpty()) {

jpql += "AND c.nombre = :nombre ";

}

if (fechaNacimiento != null) {

jpql += " AND c.fechaNacimiento = :fechaNacimiento ";

}

TypedQuery<Cliente> query = em.createQuery(jpql, Cliente.class);

if (nombre != null && !nombre.trim().isEmpty()) {

query.setParameter("nombre", nombre);

}

if (fechaNacimiento != null) {

query.setParameter("fechaNacimiento", fechaNacimiento);

}

return query.getResultList();

}COPIA EL CÓDIGO

¿Cuál es el resultado esperado de llamar a tal método?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

Se lanzará una excepción solo si se informa el segundo parámetro.

La consulta / querie será incorrecta independientemente de los parámetros, lo que provocará una excepción.

* Alternativa correta

Se lanzará una excepción solo si se informa el primer parámetro.

La consulta / querie será incorrecta independientemente de los parámetros, lo que provocará una excepción.

* Alternativa correta

Solo se lanzará una excepción si no se informa ninguno de los parámetros.

La consulta / querie será incorrecta independientemente de los parámetros, lo que provocará una excepción.

* Alternativa correta

Se lanzará una excepción, independientemente de los parámetros ingresados.

La consulta / querie contiene un error en la cláusula where.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-consulta-dinamica/83948/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83948/next)

# 05Desventajas de Criteria API

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83949/next)

¿Cuál de las siguientes es una desventaja de Criteria API?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

No permite Join con otras tablas.

* Alternativa correta

Solo funciona con Hibernate como implementación JPA.

* Alternativa correta

Código más complejo de entender y mantener.

Criteria API hace que el código sea más difícil de entender.

* Alternativa correta

Código más estrechamente acoplado con la base de datos.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-desventajas-de-criteria-api/83949/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83949/next)

# 06Haga lo que hicimos en aula: Criteria API

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83962/next)

Cuando queremos realizar consultas con múltiples parámetros nos encontramos con el problemas que todos ellos deben ser obligatorios, de lo contrario consultaria elementos nulos en tabla. Para evitar este error tenemos que usar parámetros dinámicos que nos permiten realizar consultas con múltiples parámetros y en caso de que alguno de estos sea nulo la consulta simplemente ignorará este parámetro y realizará la consulta con los parámetros existentes. Y en caso de no existir ningún parámetro, realizará la consulta de todos los elementos en la tabla.

public List<Producto> consultarPorParametros(String nombre, BigDecimal precio,LocalDate fecha){

StringBuilder jpql=new StringBuilder("SELECT p FROM Producto p WHERE 1=1 ");

if(nombre!=null && !nombre.trim().isEmpty()) {

jpql.append("AND p.nombre=:nombre ");

}

if(precio!=null && !precio.equals(new BigDecimal(0))) {

jpql.append("AND p.precio=:precio ");

}

if(fecha!=null) {

jpql.append("AND p.fechaDeRegistro=:fecha");

}

TypedQuery<Producto> query = em.createQuery(jpql.toString(),Producto.class);

if(nombre!=null && !nombre.trim().isEmpty()) {

query.setParameter("nombre", nombre);

}

if(precio!=null && !precio.equals(new BigDecimal(0))) {

query.setParameter("precio", precio);

}

if(fecha!=null) {

query.setParameter("fechaDeRegistro", fecha);

}

return query.getResultList();

}COPIA EL CÓDIGO

* Adicionalmente podemos realizar la misma consulta dinámica utilizando la API de Criteria que es un poco más compleja y recomendamos documentarse sobre ella pero simplifica la cantidad de condiciones en nuestra aplicación.

public List<Producto> consultarPorParametrosConAPICriteria(String nombre, BigDecimal precio,LocalDate fecha){

CriteriaBuilder builder = em.getCriteriaBuilder();

CriteriaQuery<Producto> query = builder.createQuery(Producto.class);

Root<Producto> from = query.from(Producto.class);

Predicate filtro = builder.and();

if(nombre!=null && !nombre.trim().isEmpty()) {

filtro=builder.and(filtro,builder.equal(from.get("nombre"), nombre));

}

if(precio!=null && !precio.equals(new BigDecimal(0))) {

filtro=builder.and(filtro,builder.equal(from.get("precio"), precio));

}

if(fecha!=null) {

filtro=builder.and(filtro,builder.equal(from.get("fechaDeRegistro"), fecha));

}

query=query.where(filtro);

return em.createQuery(query).getResultList();

}

public class PruebaDeParametros {

public static void main(String[] args) {

cargarBancoDeDatos();

EntityManager em = JPAUtils.getEntityManager();

ProductoDao productoDao =new ProductoDao(em);

List<Producto> resultado = productoDao.consultarPorParametros("FIFA", new BigDecimal(10000), null);

System.out.println(resultado.get(0).getDescripcion());

EntityManager em = JPAUtils.getEntityManager();

ProductoDao productoDao =new ProductoDao(em);

List<Producto> resultado = productoDao.consultarPorParametrosConAPICriteria("X", null, null);

System.out.println(resultado.get(0).getDescripcion());

}

private static void cargarBancoDeDatos() {

Categoria celulares = new Categoria("CELULARES");

Categoria videoJuegos = new Categoria("VIDEO\_JUEGOS");

Categoria electronicos = new Categoria("ELECTRONICOS");

Producto celular = new Producto("X", "producto nuevo", new BigDecimal(10000), celulares);

Producto videoJuego = new Producto("FIFA", "2000", new BigDecimal(10000), videoJuegos);

Producto memoria = new Producto("memoria ram", "30 GB", new BigDecimal(10000), electronicos);

EntityManager em = JPAUtils.getEntityManager();

ProductoDao productoDao = new ProductoDao(em);

CategoriaDao categoriaDao = new CategoriaDao(em);

em.getTransaction().begin();

categoriaDao.guardar(celulares);

categoriaDao.guardar(videoJuegos);

categoriaDao.guardar(electronicos);

productoDao.guardar(celular);

productoDao.guardar(videoJuego);

productoDao.guardar(memoria);

em.getTransaction().commit();

em.close();

}

}COPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-criteria-api/83962/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83962/next)

# 07Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83967/next)

En esta aula, aprendiste:

* Cómo realizar consultas JPQL con parámetros opcionales;
* Cómo funciona la API de criterios JPA;
* Cómo realizar una consulta con parámetros opcionales a través de Criteria API.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-lo-que-aprendimos/83967/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83967/next)

# 01Proyecto del aula anterior

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83957/next)

¿Comenzando en esta etapa? Aquí puedes descargar los archivos del proyecto que hemos avanzado hasta el aula anterior.

[Descargue los archivos en Github](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/tree/stage_4) o haga clic [aquí](https://github.com/alura-cursos/JPA-con-hibernate-Alura-II/archive/refs/heads/stage_4.zip) para descargarlos directamente.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-proyecto-del-aula-anterior/83957/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83957/next)

# 04Herencia entre entidades

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83950/next)

Analice el siguiente código:

@Entity

public class Persona{

@Id

@GeneratedValue(strategy = GenerationType.IDENTITY)

private Long id;

private String nombre;

private String email;

}

@Entity

@Inheritance(strategy = InheritanceType.JOINED)

public class PersonaFisica extends Persona{

private String cedula;

private LocalDate fechaDeNascimiento;

}

@Entity

@Inheritance(strategy = InheritanceType.JOINED)

public class PersonaJuridica extends Persona {

private String numeroDeInformacionFiscal;

private String inscripcionEstadual

}COPIA EL CÓDIGO

¿Por qué el mapeo es incorrecto?

Elija la alternativa correcta.

Principio del formulario

* Alternativa correta

La anotación de herencia está en la clase incorrecta.

La anotación @Inheritance solo debe agregarse en la clase base.

* Alternativa correta

Las clases secundarias PersonaFisica y PersonaJuridica no definieron el atributo que representa la clave principal de la tabla.

La clave principal solo necesita ser mapeada en la clase base.

* Alternativa correta

La clase base de persona debe ser abstracta.

No se requiere que la clase base sea abstracta.

* Alternativa correta

Más de una clase hereda de la clase base Person.

No hay restricciones en la cantidad de clases secundarias en el mapeo de herencia.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-herencia-entre-entidades/83950/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83950/next)

# 06Herencia entre entidades

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83951/next)

Analice el siguiente código:

@Entity

public class Usuario {

@Embedded

private UsuarioId id;

y

@Embeddable

public class UsuarioId implements Serializable {

private static final long serialVersionUID = 1L;

private String nombre;

private String numeroDeIdentidad;

}COPIA EL CÓDIGO

¿Por qué el mapeo anterior es incorrecto?

Principio del formulario

* Alternativa correta

El atributo id se asignó incorrectamente.

El atributo id debe anotarse con @EmbeddedId en lugar de @Embedded.

* Alternativa correta

El código es correcto, no muestra problemas de mapeo.

El atributo id se anotó incorrectamente.

* Alternativa correta

La clase UsuarioId no heredó la clase User.

La clase que representa la clave compuesta no debe heredar la clase de su entidad.

* Alternativa correta

La clase UsuarioId debería haberse anotado con @EmbeddableId.

No hay tal anotación en JPA.

Final del formulario

¡Enhorabuena, has acertado!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-herencia-entre-entidades/83951/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83951/next)

# 07Haga lo que hicimos en aula: otros tópicos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83963/next)

Por último vamos a mostrar 3 recursos que nos permiten organizar aún mejor nuestro código.

El primero es la anotación @Embeddable y @Embedded que nos permiten agrupar un conjunto de atributos o propiedades dentro de otra clase que no va a ser indicada como una nueva entidad, sino como una clase que va a ser inyectada dentro de nuestra clase entidad. Con esto podemos agrupar atributos como dirección, datos personales, datos familiares, entre otros, sin tener que colocar millones de atributos en una única clase.

Solo hay que notar que al reemplazar los atributos por una clase embutida tenemos que cuidar que los getter y setter así como, otros métodos continúen funcionando correctamente. De lo contrario podemos delegar la función de retornar esa propiedad para la nueva clase construyendo así un método delegate.

@Entity

@Table(name="clientes")

public class Cliente {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

@Embedded

private DatosPersonales datosPersonales;

public Cliente() {}

public Cliente(String nombre, String dni) {

this.datosPersonales=new DatosPersonales(nombre,dni);

}

public Long getId() {

return id;

}

public String getNombre() {

return datosPersonales.getNombre();

}

public void setNombre(String nombre) {

this.datosPersonales.setNombre(nombre);

}

public String getDni() {

return datosPersonales.getDni();

}

public void setDni(String dni) {

this.datosPersonales.setDni(dni);

}

}

@Embeddable

public class DatosPersonales implements Serializable{

private static final long serialVersionUID = 8063180201812979106L;

private String nombre;

private String dni;

public DatosPersonales() { }

public DatosPersonales(String nombre, String dni) {

this.nombre = nombre;

this.dni = dni;

}

//getter and settersCOPIA EL CÓDIGO

El segundo recurso es el mapeamiento de herencia, puede que querramos crear nuevas entidades que compartan propiedades o más especificamente que deriven de una clase madre. Para lograr esto no tenemos que colocar todas la propiedades en la nueva entidad, sino que solo debemos marcar la clase madre con la anotación @Inheritance y construir nuestras nuevas entidades, normalmente, colocando solo las nuevas propiedades y usando la palabra reservada extends para indicar herencia de clase.

* Clase madre:

@Entity

@Table(name="productos")

@NamedQuery(name="Producto.consultarPrecioPorNombre", query="SELECT P.precio FROM Producto AS P WHERE P.nombre=:nombre")

@Inheritance(strategy=InheritanceType.JOINED)

public class Producto{

…COPIA EL CÓDIGO

* Clase derivada libros:

@Entity

public class Libros extends Producto{

private String autor;

private int paginas;

…COPIA EL CÓDIGO

* Clase derivada electrónico:

@Entity

@Table(name="electronicos")

public class Electronico extends Producto{

private String marca;

private String modelo;

public Electronico() { }

public Electronico(String marca, String modelo) {

this.marca = marca;

this.modelo = modelo;

}

…COPIA EL CÓDIGO

Y ya por último, existe un recurso que nos permite identificar elementos en nuestras entidades utilizando más de un único parámetro, generalmente utilizamos un parámetro que puede ser numérico o alfanumérico para identificar las filas. A esto se llama llave primaria única, pero podemos utilizar llaves primarias con parámetros compuestos, para eso JPA nos provee como recurso la anotación @EmbeddedId que nos permite indicarle a nuestra entidad que está siendo inyectada una clase que no es una entidad, sino una clase que compone el ID de esa entidad formado por multiples parametros.

@Entity

@Table(name="categorias")

public class Categoria {

@EmbeddedId

private CategoriaId categoriaId;

public Categoria() {}COPIA EL CÓDIGO

* En la clase que va a componer la llave primaria la anotamos con @Embeddable como lo habíamos hecho anteriormente, pero la anotación @EmbeddedId va a indicarle a la entidad que debe tomar esa clase como ID, adicional todas las clases embutidas sean llaves primarias o no deben implantar la interfaz Serializable ya que sirve para indicarle a la API que van a haber datos transitando dentro de ella.

@Embeddable

public class CategoriaId implements Serializable{

private static final long serialVersionUID = 4198020985304539350L;

private String nombre;

private String password;

public CategoriaId() {

}

public CategoriaId(String nombre, String password) {

this.nombre = nombre;

this.password = password;

}

//getters settersCOPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-otros-topicos/83963/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83963/next)

# 07Haga lo que hicimos en aula: otros tópicos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83963/next)

Por último vamos a mostrar 3 recursos que nos permiten organizar aún mejor nuestro código.

El primero es la anotación @Embeddable y @Embedded que nos permiten agrupar un conjunto de atributos o propiedades dentro de otra clase que no va a ser indicada como una nueva entidad, sino como una clase que va a ser inyectada dentro de nuestra clase entidad. Con esto podemos agrupar atributos como dirección, datos personales, datos familiares, entre otros, sin tener que colocar millones de atributos en una única clase.

Solo hay que notar que al reemplazar los atributos por una clase embutida tenemos que cuidar que los getter y setter así como, otros métodos continúen funcionando correctamente. De lo contrario podemos delegar la función de retornar esa propiedad para la nueva clase construyendo así un método delegate.

@Entity

@Table(name="clientes")

public class Cliente {

@Id

@GeneratedValue(strategy=GenerationType.IDENTITY)

private Long id;

@Embedded

private DatosPersonales datosPersonales;

public Cliente() {}

public Cliente(String nombre, String dni) {

this.datosPersonales=new DatosPersonales(nombre,dni);

}

public Long getId() {

return id;

}

public String getNombre() {

return datosPersonales.getNombre();

}

public void setNombre(String nombre) {

this.datosPersonales.setNombre(nombre);

}

public String getDni() {

return datosPersonales.getDni();

}

public void setDni(String dni) {

this.datosPersonales.setDni(dni);

}

}

@Embeddable

public class DatosPersonales implements Serializable{

private static final long serialVersionUID = 8063180201812979106L;

private String nombre;

private String dni;

public DatosPersonales() { }

public DatosPersonales(String nombre, String dni) {

this.nombre = nombre;

this.dni = dni;

}

//getter and settersCOPIA EL CÓDIGO

El segundo recurso es el mapeamiento de herencia, puede que querramos crear nuevas entidades que compartan propiedades o más especificamente que deriven de una clase madre. Para lograr esto no tenemos que colocar todas la propiedades en la nueva entidad, sino que solo debemos marcar la clase madre con la anotación @Inheritance y construir nuestras nuevas entidades, normalmente, colocando solo las nuevas propiedades y usando la palabra reservada extends para indicar herencia de clase.

* Clase madre:

@Entity

@Table(name="productos")

@NamedQuery(name="Producto.consultarPrecioPorNombre", query="SELECT P.precio FROM Producto AS P WHERE P.nombre=:nombre")

@Inheritance(strategy=InheritanceType.JOINED)

public class Producto{

…COPIA EL CÓDIGO

* Clase derivada libros:

@Entity

public class Libros extends Producto{

private String autor;

private int paginas;

…COPIA EL CÓDIGO

* Clase derivada electrónico:

@Entity

@Table(name="electronicos")

public class Electronico extends Producto{

private String marca;

private String modelo;

public Electronico() { }

public Electronico(String marca, String modelo) {

this.marca = marca;

this.modelo = modelo;

}

…COPIA EL CÓDIGO

Y ya por último, existe un recurso que nos permite identificar elementos en nuestras entidades utilizando más de un único parámetro, generalmente utilizamos un parámetro que puede ser numérico o alfanumérico para identificar las filas. A esto se llama llave primaria única, pero podemos utilizar llaves primarias con parámetros compuestos, para eso JPA nos provee como recurso la anotación @EmbeddedId que nos permite indicarle a nuestra entidad que está siendo inyectada una clase que no es una entidad, sino una clase que compone el ID de esa entidad formado por multiples parametros.

@Entity

@Table(name="categorias")

public class Categoria {

@EmbeddedId

private CategoriaId categoriaId;

public Categoria() {}COPIA EL CÓDIGO

* En la clase que va a componer la llave primaria la anotamos con @Embeddable como lo habíamos hecho anteriormente, pero la anotación @EmbeddedId va a indicarle a la entidad que debe tomar esa clase como ID, adicional todas las clases embutidas sean llaves primarias o no deben implantar la interfaz Serializable ya que sirve para indicarle a la API que van a haber datos transitando dentro de ella.

@Embeddable

public class CategoriaId implements Serializable{

private static final long serialVersionUID = 4198020985304539350L;

private String nombre;

private String password;

public CategoriaId() {

}

public CategoriaId(String nombre, String password) {

this.nombre = nombre;

this.password = password;

}

//getters settersCOPIA EL CÓDIGO

VER OPINIÓN DEL INSTRUCTOR

### Opinión del instructor

Continúe con sus estudios, y si tiene alguna pregunta, ¡no dude en usar nuestro foro!

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-haga-lo-que-hicimos-en-aula-otros-topicos/83963/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83963/next)

# 09Lo que aprendimos

[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83968/next)

En esta aula, aprendiste:

* Usar anotaciones @Embeddable y @Embedded para organizar el código de una entidad;
* Realizar herencia entre entidades con las estrategias SINGLE\_TABLE y JOINED;
* Cómo asignar una clave compuesta con la anotación @EmbeddedId.

[**DISCUTIR EN EL FORO**](https://app.aluracursos.com/forum/curso-java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/exercicio-lo-que-aprendimos/83968/novo)[**PRÓXIMA ACTIVIDAD**](https://app.aluracursos.com/course/java-jpa-consultas-avanzadas-rendimiento-modelos-complejos/task/83968/next)