

ACTIVIDAD

Diseño de programas para la gestión de bases de datos relacionales y persistencia de objetos

Desarrollo de Aplicaciones Web /
Desarrollo de Aplicaciones Multiplataforma
Programación





¿Cómo lo hago?

Actividad

Diseño de programas para la gestión de bases de datos relacionales y persistencia de objetos



Objetivos

- Gestionar información almacenada en bases de datos relacionales.
- Programar conexiones a bases de datos.
- Escribir código para almacenar información en bases de datos, así como editarla y consultarla.
- Programar aplicaciones que almacenen objetos en bases de datos objeto-relacionales.
- Realizar programas para recuperar, actualizar y eliminar objetos de las bases de datos objeto-relacionales.

- 
- 
1. Rellena los datos que se piden en la tabla “Antes de empezar”.
 2. Haz uso de fuentes comunes como Arial, Calibri, Times New Roman etc.
 3. Utiliza el color negro para desarrollar tus respuestas y usa otros colores para destacar contenidos o palabras que creas necesario resaltar.
 4. Recuerda entregar la actividad en formato PDF a no ser que el profesor o profesora indique lo contrario.
 5. Recuerda nombrar el archivo siguiendo estas indicaciones:
 - Ciclo_Módulo o crédito_Tema_ACT_númeroactividad_Nombre y apellido
 - Ejemplo: AF_M01_T01_ACT_01_Maria Garcia

Antes de empezar...

Nombre	GARA
Apellidos	GONZÁLEZ SOSA
Módulo/Crédito	M03 PROGRAMACIÓN
UF (solo ciclos LOE)	UF6
Título de la actividad	Diseño de programas para la gestión de bases de datos relacionales y persistencia de objetos.





Se debe entregar un zip que contenga todos los archivos .java que has creado y el código SQL para crear la base de datos. Para poder aprobar un ejercicio, éste debe poder ejecutarse sin errores. Crea los archivos .java dentro de una carpeta de nombre actividad07

1- Creadentro de un package de nombre “**actividad07.libreria**” un programa para gestionar información sobre libros almacenando la información en MySQL.

1. Crea la clase **LibreriaMain** que muestra al usuario un menú con 6 opciones:
 - a. Restablecer la base de datos: esta opción debe:
 - i. Borrar la base de datos si ya existe
 - ii. Crear la base de datos
 - iii. Crear una tabla libro para almacenar: título, autor, país, paginas, género y un identificador autoincremental.
 - iv. Crear una tabla autor para almacenar: nombre y los apellidos del autor y un identificador autoincremental
 - v. Añadir dos autores
 - vi. Para cada uno de los dos autores anteriores, añadir un libro suyo.
 - b. Mostrar autores: esta opción debe de mostrar un listado de los autores
 - c. Mostrar libros: esta opción debe mostrar un listado de los libros
 - d. Modificar un autor: esta opción debe
 - i. Mostrar un listado de los autores
 - ii. Pedir qué autor se quiere modificar según su identificador
 - iii. Pedir qué dato se quiere modificar del autor
 - iv. Actualizar en la base de datos la información del autor.
 - e. Eliminar un libro: esta opción debe:
 - i. Mostrar un listado de los libros
 - ii. Pedir qué libro se quiere eliminar según su identificador
 - iii. Borrar el libro de la base de datos



A continuación se muestra un ejemplo de ejecución:



Qué quieres hacer?
1-restablecer la base de dat
2-mostrar autores
3-mostrar libros
4-modificar un autor.
5-eliminar un libro.
0-Salir

4
----LISTADO DE AUTORES---
1 - Manuel Escribano
2 - Juana Escritor

Indica el id del autor a mod
1
Indica qué modificar:
1-nombre
2-apellidos
2
Indica el nuevo valor
Escribnatitos
Qué quieres hacer?
1-restablecer la base de dat
2-mostrar autores
3-mostrar libros
4-modificar un autor.
5-eliminar un libro.

- 
- 
- 1) Creamos la conexión con la base de datos con los atributos: `datosConexion`, `baseDatos`, `usuario`, `contraseña`.
 - 2) Nos conectamos al `mysql` con el `DriverManager`.
 - 3) En el método `crearBDD`:
Creamos la base de datos con la query `"create database if not exists "` y los datos de conexión.

En el `main`, hacemos una instancia de la base de datos para poder acceder a los métodos de la clase. Hacemos el menú con el `switch /case`.

-CASE1:

- 1) Llamamos al método `borrarBDD` que ejecuta la query que va a restablecer la base de datos.
- 2) Llamamos también al método que `crearBDD`.
- 3) Llamamos a `crearTablaAutor()`. Con su query e incluimos los elementos que lo van a componer, la clave identificador incremental, el nombre y apellido del autor. Ejecutamos la query.
- 4) Llamamos al método `crearTablaLibro()`. En esta, a diferencia de la otra, le vamos añadir una clave foránea para vincular esta tabla con la tabla `Autor`.
- 5) Creamos 2 autos y 2 libros pasándole los datos a sus constructores.

-CASE2:

Llamamos al método `mostrarAutores()`. Este contiene un `ArrayList` de autores que habremos hecho previamente con la query `select`, seleccionando de la tabla `autores`. Y añadimos los datos de nuestra tabla, `id`, `nombre` y `apellidos`.

-CASE 3:

Llamamos a `mostrarLibros()`. Lo primero de todo, igual que en el `case2`, es listar los libros, añadiendo los elementos de la tabla `libro`: `id`, `titulo`, `autor`, `país`, `género`, `páginas`, `autorId`. Y los mostramos.

-CASE4:

Llamamos a `modificar autor` y le pasamos la entrada del scanner por parámetro. En el método preguntaremos al usuario cuál es el `id` del autor que quiere modificar, el campo a modificar (`nombre/apellidos`) y el nuevo valor. Con la query `"update Autor Set" ... where id= ..."`, actualizaremos el valor.



-CASE5:

Llamamos a eliminarLibro y también le vamos a pasar por parámetro la entrada del Scanner ya que vamos a interactuar con el usuario. Este debe elegir el id del libro que quiere eliminar, y luego con la query “delete from Libro where id=””; lo borraremos.

2-Crea dentro de un package de nombre “**actividad07.libreria**” un programa paragestionar información en **objectDB**sobre tiendas y sus empleados.

En la base de datos se debe almacenar la siguiente información

1. Sobre un Empleado:
 - a. nombre : nombre del empleado
 - b. apellido: apellido del empleado
2. Sobre una tienda
 - a. id: campo autogenerado
 - b. direccion : la dirección de la tienda
 - c. ventas : el número de ventas que ha efectuado una tienda en el transcurso de la semana.
 - d. empleados : conjunto de empleados que tiene una tienda

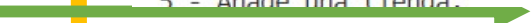
Crea la clase **GestionaTiendas** con el método *main* para que:

1. Inicialmente cree tres tiendas y tres empleados y los almacene en la base de datos (los datos estarán preestablecidos por vosotros).
2. Muestre un menú con las siguientes opciones programada cada una en una función aparte. El programa solo finaliza al seleccionar la opción de salir:
 1. Mostrar los empleados
 2. Mostrar las tiendas
 3. Mostrar tiendas ordenadas por ventas
 4. Editar un empleado
 5. Crear una nueva tienda
 0. Salir

A continuación, se muestra un ejemplo de ejecución del programa:

Menú inicial:

Introduzca la operación a realizar del siguiente menú

- 1 - Muestra los empleados
 - 2 - Muestra las tiendas
 - 3 - Mostrar tiendas ordenadas por ventas
 - 4 - Modificar un empleado.
 - 5 - Añade una tienda.
- 

Mostrar los empleados:

```
1
index:[0] Autor: [ id=4, nombre=Black,apellido=Orchid]
index:[1] Autor: [ id=5, nombre=Voldo,apellido=Soul]
index:[2] Autor: [ id=6, nombre=Blanca,apellido=Street]
```

Mostrar las tiendas:

```
2
Tienda [ id=1, direccion=Calle Elm 84, ventas=100,
empleados=[Autor: [ id=4, nombre=Black,apellido=Orchid]]]
Tienda [ id=2, direccion=Calle Malasana 32, ventas=90,
empleados=[Autor: [ id=5, nombre=Voldo,apellido=Soul]]]
Tienda [ id=3, direccion=Calle cloverfield 10, ventas=110,
```



Mostrar tiendas por ventas descendiente

```
3
Ordenar segun ventas Ascendiente o Descendiente?
1- Ascendiente
2-Descendiente
2
Tienda [ id=3, direccion=Calle cloverfield 10, ventas=110,
empleados=[Autor: [ id=5, nombre=Voldo,apellido=Soul], Autor: [ id=6, nombre=Blanca,apellido=Street]]]
Tienda [ id=1, direccion=Calle Elm 84, ventas=100,
empleados=[Autor: [ id=4, nombre=Black,apellido=Orchid]]]
```

Editar un empleado:

```
4
index:[0] Autor: [ id=4, nombre=Black,apellido=Orchid]
index:[1] Autor: [ id=5, nombre=Voldo,apellido=Soul]
index:[2] Autor: [ id=6, nombre=Blanca,apellido=Street]
Id del empleado a modificar
5
El empleado seleccionado es: Autor: [ id=5, nombre=Voldo,
Indica el atributo a modificar:
1 - Nombre
2 - Apellido
0 - Ninguno
6
```

Crear una tienda con el empleado Polo y Blanca:



```
5
Direccion de la tienda
Hyrule Street
ventas
999
index:[0] Autor: [ id=4, nombre=Black,apellid
index:[1] Autor: [ id=5, nombre=Polo,apellid
index:[2] Autor: [ id=6, nombre=Blanca,apelli
Index del empleado que quieras añadir a la ti
1
Empleado añadido correctametne
Quieres añadir un nuevo empleado:
1- SI
0-No
1
Index del empleado que quieras añadir a la ti
2
Empleado añadido correctametne
```

Lo primero de todo es comentar que este ejercicio está a medio hacer. Me hubiera gustado poder hacer que el case4, modificar empleado, funcionara pero no di con la query para poder conectar el empleado con su id correspondiente. Luego con un getNombre y setNombre le hubiera modificado el nombre al empleado. También me faltó dejar mensajitos al usuario tipo, “se ha creado la tienda correctamente” o “ se ha establecido la conexión con la ddbb”.

El case 3, tampoco tuve tiempo de dedicarle el tiempo necesario así que este está totalmente vacío.

En general al ejercicio, le falta insertar try/catch y también me hubiera gustado reestructurar los for-each para que se viera mejor los listados.

Creamos el objeto que va a realizar la conexión a la base de datos, se crea si no existe.

Creamos dos clases, Empleado y Tienda, le añadimos la anotación @Entity para poder guardar la información en la base de datos y la relacionamos con @OneToMany, 1 tienda con varios empleados.

En el main instanciamos los empleados, las tiendas y añadimos los empleados a cada tienda.

Creamos el menú con switch/case.

Case1:

Llamamos a mostrarEmpleado y le pasamos por parámetro el EntityManager em. Con la query "SELECT empleado From Empleado empleado" lo añadimos a una lista, la recorremos y la mostramos.

Case2:

Llamamos a mostrarTiendas, y básicamente igual que el anterior.



Case4:

Ejercicio a medio hacer entre comentarios. Mi intención era con la query, que no pude hacer que funcionara, relacionar el empleado con el id, una vez el usuario hubiera indicado el id del empleado a modificar. Luego con los getters y los setters acceder a los atributos de los empleados y modificarlo, luego mostrar por consola el listado de empleado con el nuevo modificado.

Case5:

Llamamos a crearTienda, creamos una instancia de una tienda y la añadimos. Mostramos tiendas con la nueva creada.