CURSO DE PROGRAMACIÓN FULL STACK

ACCESO A BASES DE DATOS DESDE JAVA: JDBC / JPA

GUÍA DE PERSISTENCIA CON JDBC Y JPA

Conectividad a Bases de Datos de Java (JDBC)

Para utilizar una base de datos desde java es necesario primero contar con un Driver JDBC. JDBC es una API que permite la ejecución de operaciones sobre bases de datos desde el lenguaje de programación Java, independientemente del sistema operativo donde se ejecute o de la base de datos a la cual se accede, utilizando el dialecto SQL del modelo de base de datos que se utilice.

El API JDBC se presenta como una colección de interfaces Java y métodos de gestión de manejadores de conexión hacia cada modelo específico de base de datos. Un manejador de conexiones hacia un modelo de base de datos en particular es un conjunto de clases que implementan las interfaces Java y que utilizan los métodos de registro para declarar los tipos de localizadores a base de datos (URL) que pueden manejar. Para utilizar una base de datos particular, el usuario ejecuta su programa junto con la biblioteca de conexión apropiada al modelo de su base de datos, y accede a ella estableciendo una conexión; para ello provee el localizador a la base de datos y los parámetros de conexión específicos. A partir de allí puede realizar cualquier tipo de tarea con la base de datos a la que tenga permiso: consulta, actualización, creación, modificación y borrado de tablas, ejecución de procedimientos almacenados en la base de datos, etc.

Las clases e interfaces principales de JDBC son:

- java.sql.DriverManager
- java.sql.Connection
- java.sql.Statement
- java.sql.ResultSet
- java.sql.PreparedStatement
- javax.sql.DataSource

COMPONENTES DEL API DE JDBC

DriverManager: Maneja una lista de drivers de bases de datos. Hace coincidir las solicitudes de conexión de la aplicación Java con el controlador de base de datos adecuado utilizando el subprotocolo de comunicación. El primer controlador que reconoce un cierto subprotocolo bajo JDBC se usará para establecer una conexión de base de datos.

- **Driver:** Es el enlace de comunicaciones de la base de datos que maneja toda la comunicación con la base de datos. Normalmente, una vez que se carga el controlador, el desarrollador no necesita llamarlo explícitamente.

- **Connection:** Es una interfaz con todos los métodos para contactar una base de datos. El objeto de conexión representa el contexto de comunicación, es decir, toda la comunicación con la base de datos es solo a través del objeto de Connection.

- **Statement:** Encapsula una instrucción SQL que se pasa a la base de datos para ser analizada, compilada, planificada y ejecutada.

- **ResultSet:** Los ResultSet representan un conjunto de filas recuperadas debido a la ejecución de una consulta.

ACCESO A BASES DE DATOS CON JDBC

JDBC nos permitirá acceder a bases de datos desde Java. Para ello necesitaremos contar con un SGBD (sistema gestor de bases de datos) además de un driver específico para poder acceder a este SGBD. La ventaja de JDBC es que nos permitirá acceder a cualquier tipo de base de datos, siempre que contemos con un driver apropiado para ella.

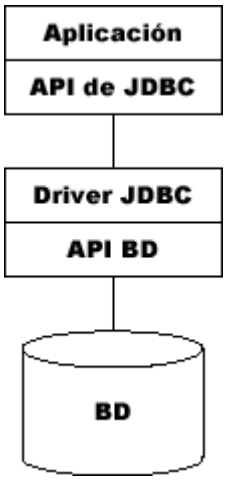


Figura 1: Arquitectura de JDBC

Como se observa en la Figura 1, cuando se construye una aplicación Java utilizando JDBC para el acceso a una base de datos, en la aplicación siempre se utiliza la API estándar de JDBC, y la implementación concreta de la base de datos será transparente para el usuario.

- Conexión a la Base de Datos

Una vez cargado el driver apropiado para nuestro SGBD se debe establecer la conexión con la BD. Para ello se utiliza el siguiente método:

```
Connection con = DriverManager.getConnection(url, login, password);
```

La conexión a la BD está encapsulada en un objeto Connection, y para su creación se debe proporcionar la url de la BD y el login y password para acceder a ella. El formato de la url variará según el driver que se utilice.

- Creación y ejecución de sentencias SQL

Una vez obtenida la conexión a la BD, se puede utilizar para crear sentencias. Estas sentencias están encapsuladas en la clase Statement, y se pueden crear de la siguiente forma:

```
Statement stmt = con.createStatement();
```

Una vez obtenido este objeto se puede ejecutar sentencias utilizando su método executeUpdate() al que se proporciona una cadena con la sentencia SQL que se quiere ejecutar:

```
stmt.executeUpdate(sentenciaSQL);
```

Estas sentencias pueden utilizarse para creación de tablas, y para la inserción, actualización y borrado de datos en ellas. Para ello se utilizan las correspondientes sentencias SQL.

- Obtención de datos

Para obtener datos almacenados en la BD se utiliza una consulta SQL (query). La consulta se puede ejecutar utilizando el objeto Statement, con el método executeQuery() al que le se le pasa una cadena con la consulta SQL. Los datos resultantes se devuelven como un objeto ResultSet.

```
ResultSet result = stmt.executeQuery(query);
```

La consulta SQL devolverá una tabla, que tendrá una serie de campos y un conjunto de registros, cada uno de los cuales consistirá en una tupla de valores correspondientes a los campos de la tabla.

- Optimización de sentencias

Cuando se quiere invocar una determinada sentencia repetidas veces, puede ser conveniente dejar esa sentencia preparada para que pueda ser ejecutada de forma más eficiente. Para hacer esto se utiliza la interfaz PreparedStatement, que podrá obtenerse a partir de la conexión a la BD de la siguiente forma:

```
PreparedStatement ps = con.prepareStatement("UPDATE FROM nombreTabla
                           SET campo1 = 'valor'
                           WHERE campo2>1200 AND campo2<1300");
```

Vemos que, a este objeto, a diferencia del objeto Statement visto anteriormente, se le proporciona la sentencia SQL en el momento de su creación, por lo que estará preparado y optimizado para la ejecución de dicha sentencia posteriormente.

Sin embargo, lo más común es que se necesite hacer variaciones sobre la sentencia, ya que normalmente no será necesario ejecutar repetidas veces la misma sentencia exactamente, sino variaciones de ella. Por ello, este objeto nos permite parametrizar la sentencia. Para ello se deben establecer las posiciones de los parámetros con el carácter '?' dentro de la cadena de la sentencia, tal como se muestra a continuación:

```
PreparedStatement ps = con.prepareStatement("UPDATE FROM nombreTabla
                           SET campo1 = 'valor'
                           WHERE campo2 > ? AND campo2 < ?");
```

En este caso se tienen dos parámetros, que representan un rango de valores en el cual se quiere actualizar. Cuando se ejecute esta sentencia, el *campo1* de la tabla *nombreTabla* se establecerá a *valor1* desde el límite inferior hasta límite superior indicado en el *campo2*.

Para dar valor a estos parámetros se utiliza el método setXXX() donde XXX será el tipo de los datos que asignamos al parámetro, indicando el número del parámetro (que empieza desde 1) y el valor que le queremos dar. Por ejemplo, para asignar valores enteros a los parámetros se debe hacer:

```
ps.setInt(1,1200);
ps.setInt(2,1300);
```

Una vez asignados los parámetros, se puede ejecutar la sentencia llamando al método executeUpdate() del objeto PreparedStatement:

```
int n = ps.executeUpdate();
```

Igual que en el caso de los objetos Statement, se puede utilizar cualquier otro de los métodos para la ejecución de sentencias, executeQuery() o execute(), según el tipo de sentencia que se vaya a ejecutar.

PERSISTENCIA EN JAVA CON JPA

JPA es una API que ofrece Java para implementar un Framework Object Relational Mapping (ORM), que permite interactuar con la base de datos por medio de objetos, de esta forma, JPA es el encargado de convertir los objetos Java en instrucciones para el Manejador de Base de Datos (MDB). El objetivo que persigue el diseño de esta API es no perder las ventajas de la orientación a objetos al interactuar con una base de datos (siguiendo el patrón de mapeo objeto-relacional)

JPA es una especificación, es decir, no es más que un documento en el cual se plasman las reglas que debe de cumplir cualquier proveedor que dese desarrollar una implementación de JPA, de tal forma que cualquier persona puede tomar la especificación y desarrollar su propia implementación de JPA. Existen varios proveedores como lo son los siguientes:

- Hibernate
- ObjectDB
- TopLink
- EclipseLink
- OpenJPA

Java Persistence Query Language (JPQL) es un lenguaje de consulta orientado a objetos independiente de la plataforma definido como parte de la especificación Java Persistence API (JPA). JPQL es usado para hacer consultas contra las entidades almacenadas en una base de datos relacional. Está inspirado en gran medida por SQL, y sus consultas se asemejan a las consultas SQL en la sintaxis, pero opera con objetos entidad de JPA en lugar de hacerlo directamente con las tablas de la base de datos.

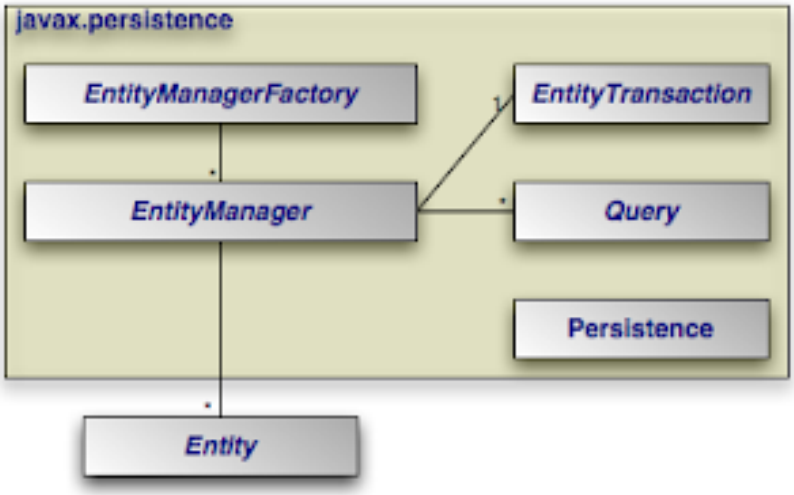
PERSISTENCIA DE OBJETOS

Los entornos de persistencia de objetos se encargan de guardar y recuperar objetos Java en bases de datos:

- El programador no necesita programar código JDBC ni consultas SQL.
- El entorno realiza la conversión entre tipos Java y tipos SQL.
- El entorno crea y ejecuta las consultas SQL necesarias.

ARQUITECTURA JPA

La arquitectura de JPA está diseñada para gestionar Entidades y las relaciones que hay entre ellas. A continuación, detallamos los principales componentes de la arquitectura



Entity: Son los objetos persistidos en Base de datos como registros de una tabla. Podemos decir que cada Entidad corresponderá con una tabla de nuestra Base de Datos

Persistence: Clase con métodos estáticos que nos permiten obtener instancias de EntityManagerFactory

EntityManagerFactory: Es una factoría de EntityManager. Se encarga crear y gestionar múltiples instancias de EntityManager

EntityManager: Es una interface que gestiona las operaciones de persistencia de las Entidades. A su vez trabaja como factoría de Query

Query: Es una interface para obtener la relación de objetos que cumplen un criterio

EntityTransaction: Agrupa las operaciones realizadas sobre un EntityManager en una única transacción de Base de Datos

MAPEO CON ANOTACIONES

Las anotaciones nos permiten configurar el mapeo de una entidad dentro del mismo archivo donde se declara la clase, de este modo, es más fácil de visualizar y mantener.

Las anotaciones comienzan con el símbolo "@" seguido de un identificador. Las anotaciones son utilizadas antes de la declaración de clase, propiedad o método. A continuación, se detallan las principales:

@Entity. Declara la clase como una Entidad

@Table: Declara el nombre de la Tabla con la que se mapea la Entidad

@Id: Declara que la propiedad es la clave primaria de la Tabla

@GeneratedValue: Declara como la propiedad va a ser inicializada. Manualmente, Automático o a partir de una secuencia

@SecuenceGenerator: Declara una secuencia que será usada para asignar un valor a la propiedad que representa la clave primaria de la tabla

@Column: Declara que la propiedad se mapea con una columna de la tabla

@Transient: Declara que la propiedad no debe persistirse en Base de Datos. Se utiliza para propiedad auto calculadas.

@Enumerated: Declara que la propiedad es de alguno de los valores definidos en un Enumerado (lista de valores constantes). Los valores de un tipo enumerado tienen asociado implicitamente un tipo ordinal que será asociada a la propiedad de este tipo.

@Temporal. Declara que se está tratando de un atributo que va a trabajar con fechas, entre paréntesis, debemos especificarle que estilo de fecha va a manejar: @Temporal(TemporalType.DATE), @Temportal(TemporalType.TIME), @Temporal(TemporalType.TIMESTAMP)

LAS RELACIONES

A partir de estos conceptos definimos 4 tipos principales de relaciones entre entidades:

@ManyToOne identifica una relación de este tipo y debe aparecer en la entidad origen. La anotación @ManyToOne identifica siempre el lado propietario de la relación, así que si hay una "columna de unión" la anotación @JoinColum debe aparecer en esta misma entidad. El lado propietario de la relación se identifica porque lleva la anotación @JoinColum y el lado inverso o no propietario se identifica porque no lleva esta anotación.

@OneToOne se caracterizan porque solo puede existir una y solo una relación con la Entidad de destino, de esta forma, la entidad marcada como @OneToOne deberá tener una referencia a la Entidad destino y por ningún motivo podrá ser una colección. De la misma forma, la Entidad destino no podrá pertenecer a otra Instancia de la Entidad origen. Las relaciones @OneToOne se utilizan cuando existe una profunda relación entre la Entidad origen y destino, de tal forma que la entidad destino le pertenece a la Entidad origen y solo a ella, por ende, la existencia de la entidad destino depende de la Entidad origen.

@OneToMany se caracterizan por Entidad donde tenemos un objeto principal y colección de objetos de otra Entidad relacionados directamente. Estas relaciones se definen mediante colecciones, pues tendremos una serie de objetos pertenecientes al objeto principal.

@ManyToMany se caracterizan por Entidades que están relacionadas con a muchos elementos de un tipo determinado, pero al mismo tiempo, estos últimos registros no son exclusivos de un registro en particular, si no que pueden ser parte de varios, por lo tanto, tenemos una Entidad A, la cual puede estar relacionada como muchos registros de la Entidad B, pero al mismo tiempo, la Entidad B puede pertenecer a varias instancias de la Entidad A. Algo muy importante a tomar en cuenta cuando trabajamos con relaciones @ManyToMany, es que en realidad este tipo de relaciones no existen físicamente en la base de datos, y en su lugar, es necesario crear una tabla intermedia que relaciones las dos Entidades.

JPA VS JDBC

Ventajas JPA

- Permite desarrollar mucho más rápido.
- Permite trabajar con la base de datos por medio de entidades en vez de Querys.
- Ofrece un paradigma 100% orientado a objetos.
- Elimina errores en tiempo de ejecución.
- Mejora el mantenimiento del software.

Desventajas JPA

- No ofrece toda la funcionalidad que ofrecería tirar consultas nativas.
- El performance es mucho más bajo que realizar las consultas por JBDC.
- Puede representar una curva de aprendizaje más grande.

Ventajas JDBC

- Ofrece un performance superior ya que es la forma más directa de mandar instrucciones la base de datos.
- Permite explotar al máximo las funcionalidades de la base de datos.

Desventajas JDBC

- El mantenimiento es mucho más costoso.
- Introduce muchos errores en tiempo de ejecución.
- El desarrollo es mucho más lento.

Probablemente se pueda pensar que JPA son mejores ya que ofrece mayores ventajas que desventajas, sin embargo, las desventajas que tiene son muy serias y pueden ser cruciales a la hora de decidir que tecnología utilizar. Si tenemos una aplicación muy buena y fácil de mantener pero que tarda demasiado para consultar datos puede llegar a ser algo muy malo.

Como conclusión si lo que se requiere es una aplicación donde el rendimiento sea el factor más importante conviene utilizar JDBC, pero por otra parte, si el rendimiento es algo que no es tan importante, se puede utilizar JPA.

Preguntas de Aprendizaje

1) Responda Verdadero (V) o Falso (F)

	V	F
El proceso de creación y destrucción de una conexión a una base de datos es un proceso rápido por lo que no influye sensiblemente en el rendimiento de una aplicación	( )	( )
La capa de persistencia de una aplicación permite almacenar el estado que necesitan persistir en un sistema gestor de datos, pero no actualizarlo	( )	( )
En el API JDBC, existe sólo un método que puede lanzar la excepción SQLException.	( )	( )
JDBC es la parte de Java que nos va a permitir conectarnos con bases de datos relacionales utilizando el lenguaje SQL	( )	( )
Una sentencia JDBC (objeto Statement) y una sentencia SQL son exactamente lo mismo	( )	( )
Un executeQuery devuelve siempre un objeto de la clase ResultSet	( )	( )
JPA es un API basado en la tecnología ORM para trabajar con entidades persistentes.	( )	( )

JPA se basa en JDBC y es necesario tener en funcionamiento una base de datos SQL (como MySQL). Para usar el API debemos instalar el conjunto de librerías que lo implementan y el archivo de configuración META-INF/persistence.xml. ( ) ( )

Las entidades en JPA son clases Java complejas que se les agregan unas anotaciones para indicar las características de persistencia. ( ) ( )

En el mapeado entidad-relación de JPA las entidades se convierten en tablas y sus atributos en columnas. ( ) ( )

El EntityManager es el elemento fundamental del API. Es el encargado de hacer persistentes las entidades que gestiona y de mantener la sincronización entre el contexto de persistencia (estado en memoria de las entidades) y la base de datos. ( ) ( )

2) **El API estándar de acceso a bases de datos en Java es:**

a) ODBC
b) JDBC
c) JPA/Hibernate
d) Ninguna de las anteriores es correcta

3) **¿Cómo se comprueba en un programa JAVA que la conexión JDBC con una base de datos es correcta?**

a) Si la expresión Connection c = DriverManager.getJDBC(dbUrl, user, password) devuelve una excepción entonces la conexión no es correcta.
b) Si la sentencia Connection c = DriverManager.getConnection(dbUrl, user, Password) devuelve una excepción entonces la conexión es correcta.
c) Si la sentencia Connection c = DriverManager.getConnection(dbUrl, user, password) devuelve una excepción entonces la conexión no es correcta.
d) Ninguna de las anteriores es correcta

4) Dado el siguiente fragmento de código obtenido de un programa perfectamente creado, por tanto, se entiende que existe definición de todas las clases y elementos necesarios para su ejecución, se pide, sabiendo que consulta se define como:

```
ResultSet consulta = null;
```

¿Cuál es el resultado de este código?

```
consulta = statment.executeQuery("SELECT pasword FROM identificadores"+ "
WHERE
identificador="" + identencryp +"");
```

a) Imprime en una ventana nueva el texto Select password form indentificadores.
b) Ejecuta la orden SQL sobre una tabla denominada identificadores.
c) Ejecuta una encriptación de tipo MD5 sobre password
d) Ninguna de las anteriores es correcta

5) La escritura mediante JDBC conlleva

a) Abrir una conexión.
b) Crear una sentencia SQL.
c) Copiar todos los valores de las propiedades de un objeto en la sentencia, ejecutarla y así almacenar el objeto.
d) Todas son correctas.

6) Para operar con una base de datos, y ejecutar las consultas necesarias, nuestra aplicación no deberá hacer:

a) Establecer una conexión con la base de datos.
b) Enviar consultas SQL y procesar el resultado.
c) Liberar los recursos al terminar.
d) Ninguna es correcta, pues todas se deben hacer.

7) La implementación de JPA es proporcionada por?
a) Hibernate
b) Toplink
c) EclipseLink
d) Todos los anteriores

8) ¿Qué es cierto acerca de la siguiente asociación @Entity entre House y Window?

```
@Entity
public class Window {
	@Id
	private int winNo;
	@ManyToOne
	private House aHouse;
}
@Entity
public class House {
	@Id
	private int houseNo;
	@OneToMany(mappedBy="aHouse")
	private List windows;
}
```

a) Es la asociación unidireccional OneToMany
b) Es la asociación bidireccional OneToMany
c) El propietario de la asociación es la clase House
d) El propietario de la asociación es la clase Window

9) Cuál de las siguientes anotaciones permite permite definir aspectos muy importantes sobre las columnas de la base de datos como lo es el nombre, la longitud, constrains, etc.

a) @GeneratedValue
b) @Column
c) @JoinColumn
d) Ninguna de las anteriores

10) find, persist, merge y remove son operaciones de la clase:

e) EntityManager
f) EntityManagerFactory
g) Javax.persistence
h) Ninguna de las anteriores

EJERCICIOS DE APRENDIZAJE

Para la realización de los ejercicios que se describen a continuación, es necesario descargar el archivo *persistencia.zip* que contiene el material necesario para realizar esta práctica. Por otra parte, *se recomienda* consultar el ***Apéndice D*** para obtener más información acerca de las clases, operaciones y atributos de JDBC y JPA.

1. Estancias en el extranjero

Nos han pedido que hagamos una aplicación Java de consola para una pequeña empresa que se dedica a organizar estancias en el extranjero dentro de una familia. El objetivo es el desarrollo del sistema de reserva de casas para realizar estancias en el exterior, utilizando el lenguaje JAVA, una base de datos MySQL y JDBC para realizar la ejecución de operaciones sobre la base de datos (BD).

Creación de la Base de Datos MySQL

La información que se desea almacenar en la base de datos es la siguiente:

- Se tienen contactos con familias de diferentes países que ofrecen alguna de las habitaciones de su hogar para acoger algún chico (por un módico precio). De cada una de estas familias se conoce el nombre, la edad mínima y máxima de sus hijos, número de hijos y correo electrónico.

- Cada una de estas familias vive en una casa, de la que se conoce la dirección (calle, numero, código postal, ciudad y país), el periodo de disponibilidad de la casa (fecha_desde, fecha_hasta), la cantidad de días mínimo de estancia y la cantidad máxima de días, el precio de la habitación por día y el tipo de vivienda.

- Se dispone también de información de los clientes que desean mandar a sus hijos a alguna de estas familias: nombre, dirección (calle, numero, código postal, ciudad y país) y su correo electrónico.

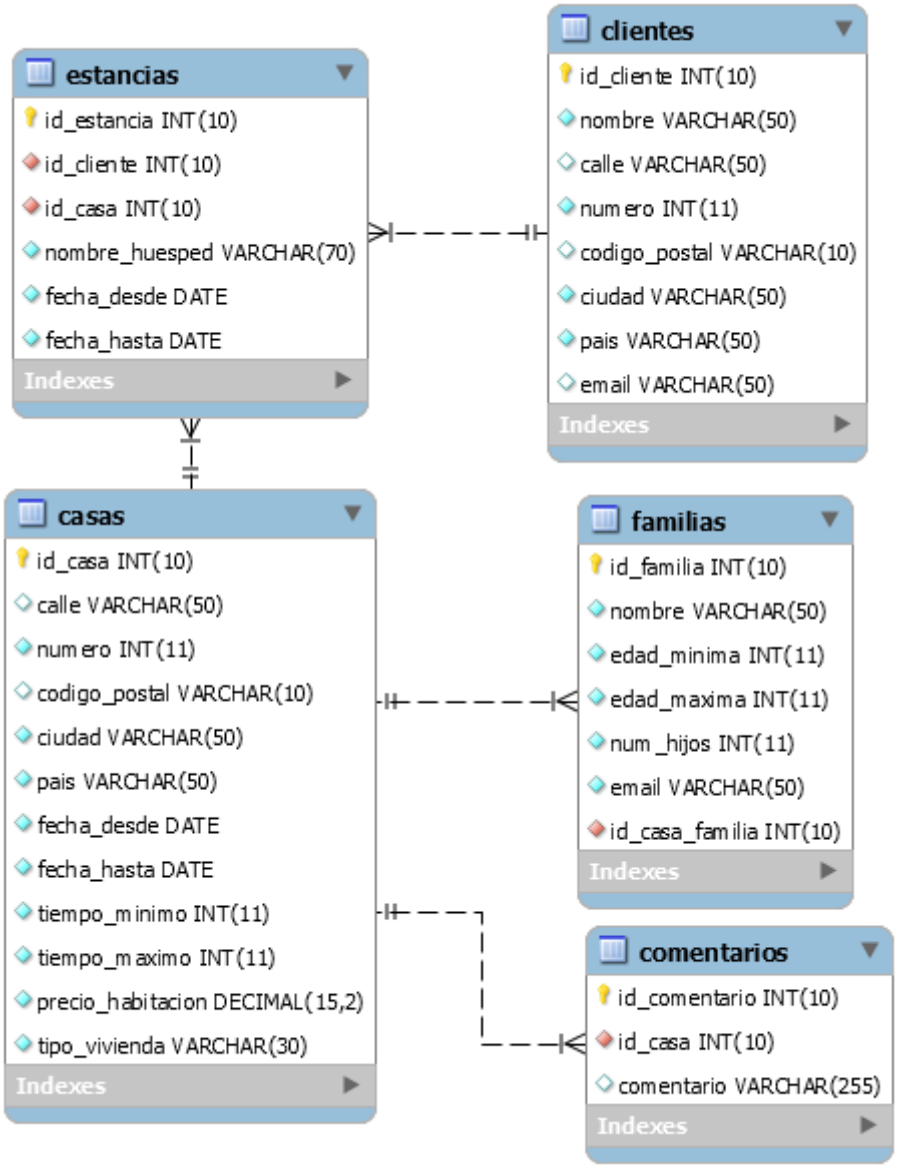
- En la BD se almacena información de las reservas y estancias realizadas por alguno de los clientes. Cada estancia o reserva la realiza un cliente, y además, el cliente puede reservar varias habitaciones al mismo tiempo (por ejemplo para varios de sus hijos), para un periodo determinado (fecha_llegada, fecha_salida).

- El sistema debe también almacenar información brindada por los clientes sobre las casas en las que ya han estado (comentarios).

Según todas estas especificaciones se debe realizar:

- Utilizar el lenguaje MySQL para crear la base de datos llamada estancias_exterior con las tablas correspondientes y las claves foráneas necesarias. Recuerda que las tablas sean de tipo InnoDB. Por comodidad, escribe todas las sentencias en un archivo con extensión .sql y ejecútalo luego en Workbench para generar toda la base de datos.

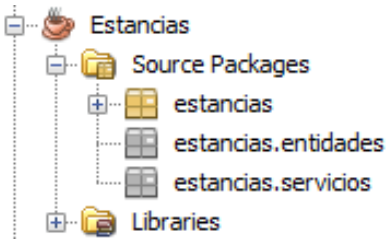
- Generar el modelo de entidad relación de la base de datos. Se debe obtener un modelo como el que se muestra a continuación:



- A continuación, ejecutar en Workbench el script “estancias” para agregar algunos datos a la base de datos creada.

Paquetes del Proyecto Java

- Crear un nuevo proyecto en Netbeans del tipo "*Java Application*" con el nombre *Estancias* y agregar dentro dos paquetes, a uno se lo llamará *entidades* y al otro se lo llamará servicios:



- Agregar en "*Libraries*" la librería "*MySQL JDBC Driver*" para permitir conectar la aplicación de Java con la base de datos MySQL.

Paquete entidades:

Dentro de este paquete se deben crear todas las clases necesarias que queremos persistir en la base de datos. Por ejemplo, una de las clases a crear dentro de este paquete es la clase "Familia.java" con los siguientes atributos:

- private int id;
- private String nombre;
- private int edad_minima;
- private int edad_maxima;
- private int num_hijos;
- private String email;

Agregar a cada clase el/los constructores necesarios y los métodos públicos getters y setters para poder acceder a los atributos privados de la clase.

VER VIDEOS:

A. Introducción JDBC
B. Clases de Servicio
C. Introducción Capa de Acceso a Datos (DAO)
D. Introducción DAO con Herencia

Paquete servicios:

En este paquete se almacenarán aquellas clases que llevarán adelante lógica del negocio. En general se crea un servicio para administrar cada una de las entidades y algunos servicios para manejar operaciones muy específicas como las estadísticas.

- Dentro de este paquete se debe crear una clase llamada *Conexion.java* para conectar la aplicación con MySQL. Dentro de esta clase escribir un método un método estático llamado obtener(), para que nos devuelva una instancia de conexión a la BD.

VIDEOS:

A) Capa acceso a datos 01
B) Capa acceso a datos 02
C) Capa acceso a datos 03

- Para realizar las consultas con la base de datos, dentro del paquete servicios, creamos las clases para cada una de las entidades con los métodos necesarios para realizar consultas a la base de datos. Una de las clases a crear en este paquete será: FamiliaServicio.java, y en esta clase se implementará, por ejemplo, un método para listar todas las familias que ofrecen alguna habitación para realizar estancias.

Realizar un menú en java a través del cual se permita elegir qué consulta se desea realizar. Las consultas a realizar sobre la BD son las siguientes:

a) Listar aquellas familias que tienen al menos 3 hijos, y con edad máxima inferior a 10 años.
b) Buscar y listar las casas disponibles para el periodo comprendido entre el 1 de agosto de 2020 y el 31 de agosto de 2020 en Reino Unido.
c) Imagínate que, como cliente, estás interesado en mandar a tu hijo a una familia, de la que únicamente recuerdas que su nombre familiar terminaba en 'y'. Escribe la consulta que te recupere las familias que cumplan tus restricciones.
d) Encuentra todas aquellas familias cuya dirección de mail sea de Hotmail.
e) Consulta la BD para que te devuelva aquellas casas disponibles a partir de una fecha dada y un número de días específico.

CONSULTAS EXTRA

Estas consultas son para hacer si terminamos los anteriores, no son obligatorias.

f) Listar los nombre, la fecha desde y la fecha hasta, de los huéspedes que realizaron una estancia.

g) Mostrar todos los clientes donde su mail sea hotmail.

h) Mostrar el numero de casas que tienen un precio de habitación mayor a 45.00.

i) Obtener el numero de clientes que existen para cada uno de los países diferentes.

j) Obtener el número de casas que existen para cada uno de los países diferentes.

k) Debido a la devaluación de la libra esterlina con respecto al euro se desea incrementar el precio por día en un 5% de todas las casas del Reino Unido. Mostrar los precios actualizados.

l) Busca y listar aquellas casas del Reino Unido de las que se ha dicho de ellas (comentarios) que están 'limpias'.

m) Insertar nuevos datos en la tabla estancias verificando la disponibilidad de las fechas.

Para finalizar, pensar junto con un compañero cómo sería posible optimizar las tablas de la BD para tener un mejor rendimiento.

2. Sistema de Reservas de una Librería

Este ejercicio hay que hacerlo con JPA solo si ya se ha dado la clase introductoria de JPA, sino hacerlo con JDBC.

El objetivo de este ejercicio es el desarrollo de un sistema de reserva de libros en JAVA utilizando una base de datos MySQL y JPA como framework de persistencia. *Se recomienda* consultar el *Apéndice D* para obtener más información sobre las anotaciones y relaciones en JPA.

a) Creación de la Base de Datos MySQL

Lo primero que se debe hacer es crear el esquema sobre el que operará el sistema de reservas de libros. Para ello, se debe abrir el IDE de base de datos que se está utilizando (Workbench) y ejecutar la siguiente sentencia:

```
CREATE  DATABASE libreria;
```

De esta manera se habrá creado una base de datos vacía llamada librería.

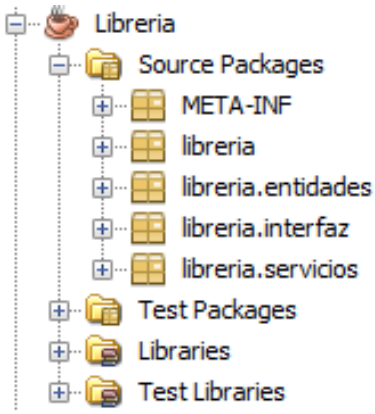
b) Paquetes del Proyecto Java

Los paquetes que se utilizarán para este proyecto son los siguientes:

- **entidades**: en este paquete se almacenarán aquellas clases que se quiere persistir en la base de datos.

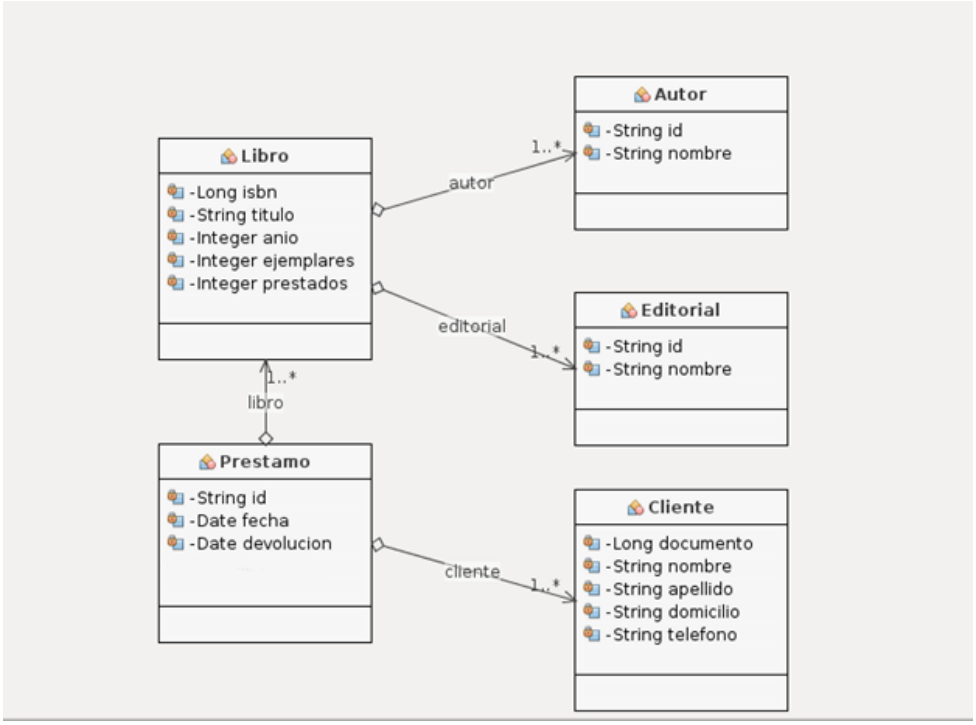
- **interfaz:** en este paquete se almacenarán aquellas clases que se utilizarán como interfaz con el usuario.

- **servicios:** en este paquete se almacenarán aquellas clases que llevarán adelante la lógica del negocio. En general se crea un servicio para administrar cada una de las entidades y algunos servicios para manejar operaciones muy específicas como las estadísticas.



c) Entidades

Crearemos el siguiente modelo de entidades:



Entidad Libro

La entidad libro modela los libros que están disponibles en la biblioteca para ser prestados. En esta entidad, el atributo “ejemplares” contiene la cantidad total de ejemplares de ese libro, mientras que el atributo “prestados” contiene cuántos de esos ejemplares se encuentran prestados en este momento.

Entidad Cliente

La entidad cliente modela los clientes (a quienes se les presta libros) de la biblioteca. Se almacenan los datos personales y de contacto de ese cliente.

Entidad Préstamo

La entidad préstamo modela los datos de un préstamo de libros. Esta entidad registra la fecha en la que se efectuó el préstamo y la fecha en la que se devolvieron los libros. Esta entidad también registra los libros que se llevaron en dicho préstamo y quien fue el cliente al cual se le prestaron.

Entidad Autor

La entidad autor modela los autores de libros.

Entidad Editorial

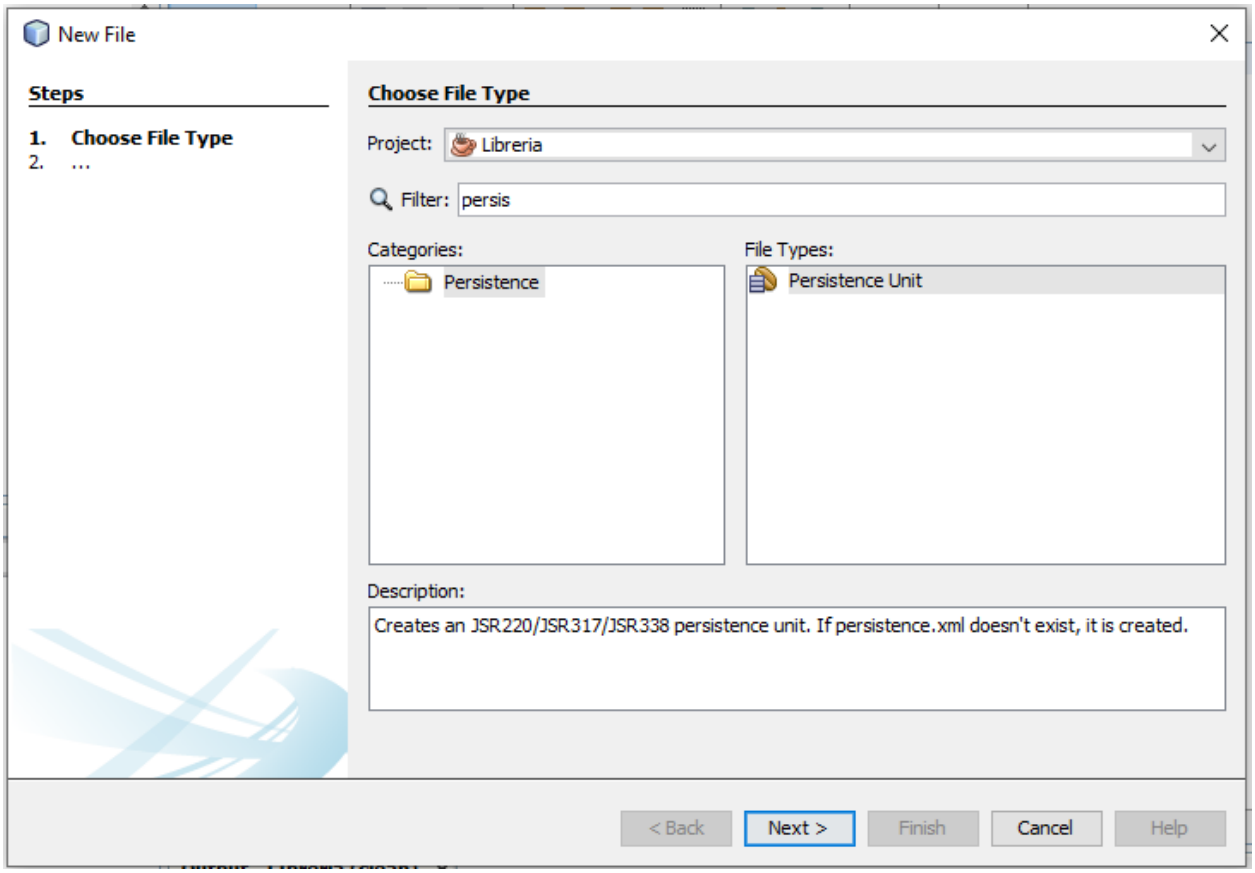
La entidad editorial modela las editoriales que publican libros.

VIDEOS:

A. Introducción JPA – Parte 1
B. Introducción JPA – Parte 2

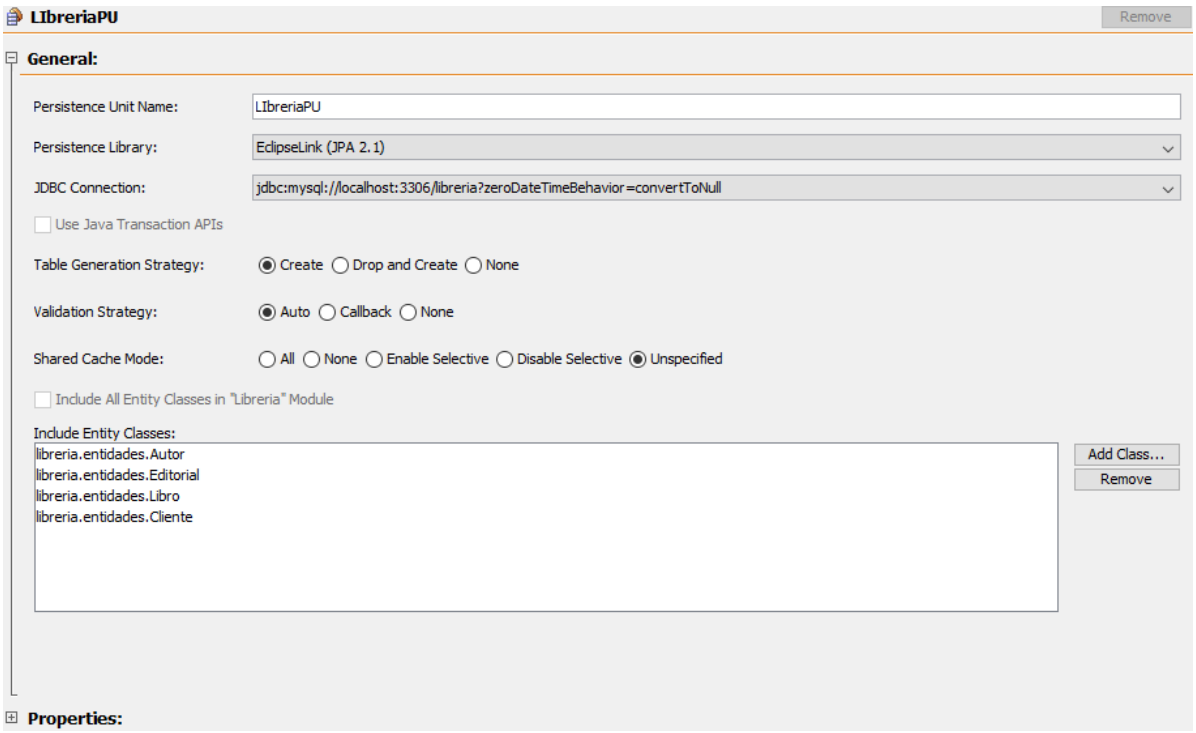
d) Unidad de Persistencia

Para configurar la unidad de persistencia del proyecto recuerde hacer click con el botón derecho sobre el proyecto y seleccionar nuevo. A continuación, se debe seleccionar la opción de *Persistence Unit* como se indica en la siguiente imagen.



Base de Datos

Al configurar la unidad de persistencia, se configura la conexión a la base de datos y se informan todas las entidades que serán manejadas por esas entidades.



Generación de Tablas

La estrategia de generación de tablas define lo que hará JPA en cada ejecución, si debe crear las tablas faltantes, si debe eliminar todas las tablas y volver a crearlas o no hacer nada. Recomendamos en este proyecto utilizar la opción: "*Create*"

Librería de Persistencia

Se debe seleccionar para este proyecto la librería "EclipseLink".

e) Servicios

AutorServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar autores (consulta, creación, modificación y eliminación).

EditorialServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar editoriales (consulta, creación, modificación y eliminación).

LibroServicio

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para administrar libros (consulta, creación, modificación y eliminación).

VIDEO: Capa de acceso a datos

f) Main

Esta clase tiene la responsabilidad de llevar adelante las funcionalidades necesarias para interactuar con el usuario. En esta clase se muestra el menú de opciones con las operaciones disponibles que podrá realizar el usuario.

g) Tarea a Realizar

Al alumno le toca desarrollar, las siguientes funcionalidades:

1) Crear base de datos Librería
2) Crear unidad de persistencia
3) Crear entidades previamente mencionadas (excepto Préstamo)
4) Generar las tablas con JPA
5) Crear servicios previamente mencionados.
6) Crear los métodos para persistir entidades en la base de datos librería

7) Crear la interfaz InterfazLibrería para llamar los métodos de persistencia de datos
8) Crear los métodos para borrar o editar dichas entidades
9) Búsqueda de un libro por ISBN
10) Búsqueda de un libro por Título
11) Creación de un Cliente nuevo
12) Crear entidad Préstamo
13) Registrar el préstamo de un libro.
14) Agregar más de un libro a un préstamo.
15) Devolución de un libro
16) Agregar validaciones a todas las funcionalidades de la aplicación.
17) Validar campos obligatorios.
18) No ingresar datos duplicados.
19) No generar condiciones inválidas. Por ejemplo, no se debe permitir prestar más ejemplares de los que hay, ni devolver más de los que se encuentran prestados. No se podrán prestar libros con fecha anterior a la fecha actual, etc.