

PRESA. UNIDAD 3. ACCESO MEDIANTE MAPEO RELACIONAL DE OBJETOS (ORM). ACCESO A BASES DE DATOS RELACIONALES USANDO DAO

PRESA. Acceso a Datos (ADA) (a distancia en inglés)

Unidad 3. ACCESO UTILIZANDO MAPEO RELACIONAL DE OBJETOS (ORM)

Acceso a bases de datos relacionales usando DAO

Abelardo Martinez

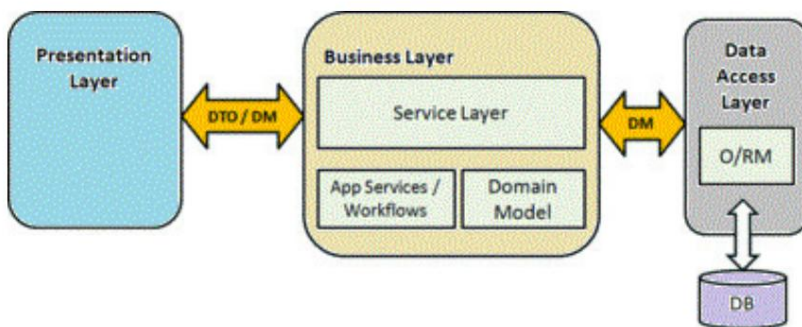
Basado y modificado de Sergio Badal (www.sergiobadal.com)

Año 2023-2024

1. ¿Qué es DAO?

Hasta ahora nos hemos conectado a la base de datos usando solo una clase (principal) que contenía tanto la conexión como la declaración de selección. Ahora separaremos el código según su funcionalidad para realizar un acceso diferenciado por capas con el objetivo de aumentar la reutilización del código.

El patrón Data Access Object (DAO) es un patrón estructural que nos permite trabajar con una base de datos utilizando una API abstracta. La API oculta a la aplicación toda la complejidad de realizar operaciones CRUD (Crear, Leer, Actualizar, Eliminar) en el mecanismo de almacenamiento subyacente. Esto permite que ambas capas evolucionen por separado sin saber nada entre sí.



1.1. Ejemplo

Para comprender cómo funciona DAO, echemos un vistazo al siguiente código.

- Como puede ver, estamos trabajando con una base de datos sin utilizar instrucciones SQL, accediendo a clases y elementos Java/objetos en lugar de tablas o cualquier elemento/objeto de la base de datos.
- Podemos decir que estamos usando CAPAS para ocultar la complejidad de la base de datos o, en otras palabras, podríamos decir:

"El patrón de objeto de acceso a datos (DAO) es un patrón estructural que nos permite aislar la capa de aplicación/negocio de la capa de persistencia (generalmente una base de datos relacional, pero podría ser cualquier otro mecanismo de persistencia) utilizando una API abstracta".

Ejemplo de: <https://www.baeldung.com/java-dao-pattern>

Aplicación de usuario de clase pública {

Dao estático privado <Usuario> jpaUserDao;

público estático vacío principal (String [] argumentos) {

// Leer usuario con ID = 1

Usuario usuario1 = getUser(1);

System.out.println(usuario1);

// Actualizar usuario con ID = 1

updateUser(usuario1, new String[]{"Jake", "jake@dominio.com"}); // Insertar nuevo

usuario saveUser(new

User("Mónica", "monica@dominio.com"));

// Elimina ese nuevo usuario

eliminarUsuario(getUsuario(2));

// Leer todos los usuarios (SELECCIONAR)

getAllUsers().forEach(usuario -> System.out.println(user.getName())); }

Usuario público estático getUser (identificación larga) {

Opcional<Usuario> usuario = jpaUserDao.get(id);

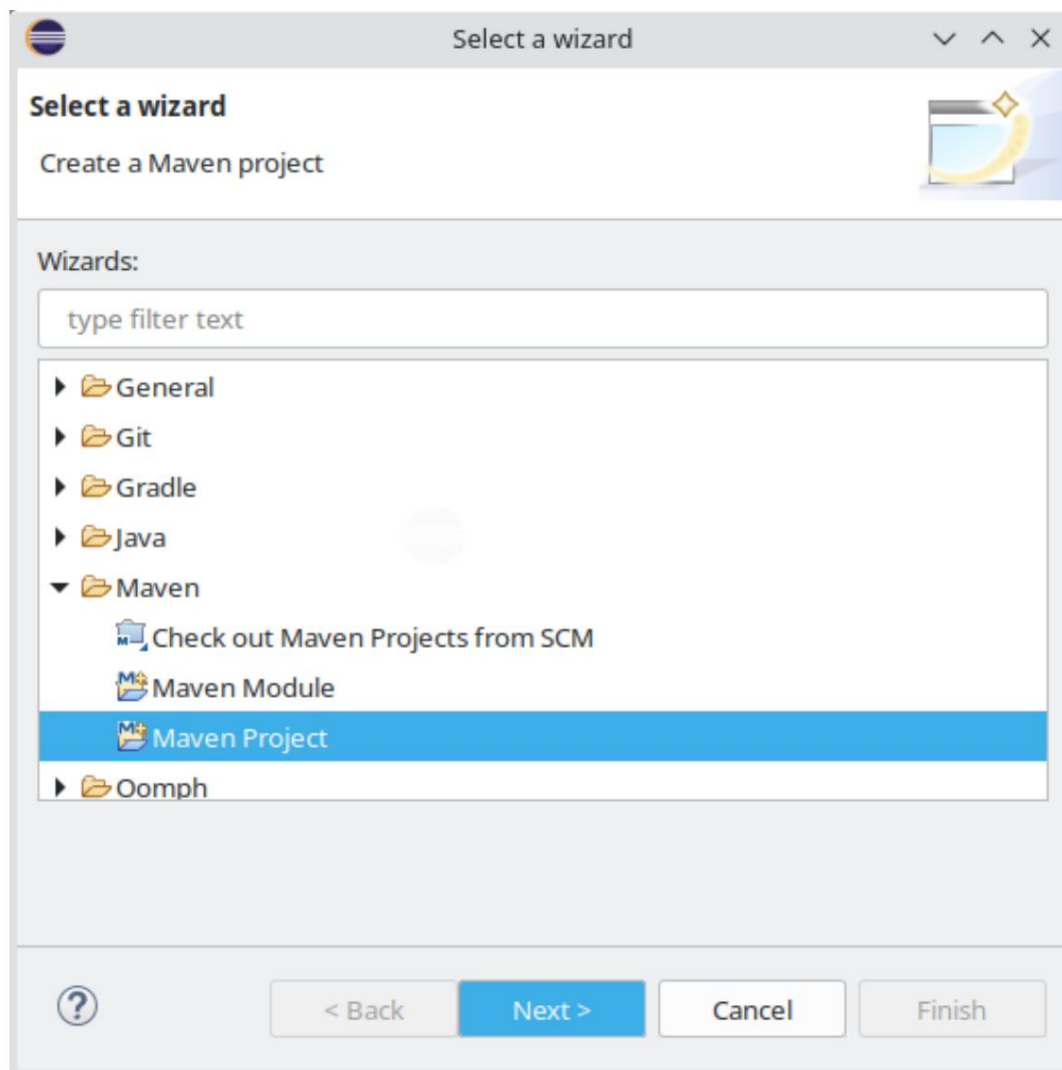
```
        devolver usuario.orElseGet(  
            () -> nuevo Usuario("usuario no existente", "sin correo electrónico"));  
    }  
  
    Lista estática pública <Usuario> getAllUsers() {  
        devolver jpaUserDao.getAll();  
    }  
  
    usuario de actualización de vacío estático público ( usuario usuario, parámetros de cadena [] ) {  
        jpaUserDao.update(usuario, parámetros);  
    }  
  
    public static void saveUser(Usuario usuario)  
    { jpaUserDao.save(usuario);  
    }  
  
    public static void eliminarUsuario(Usuario usuario)  
    { jpaUserDao.delete(usuario);  
    }  
}
```

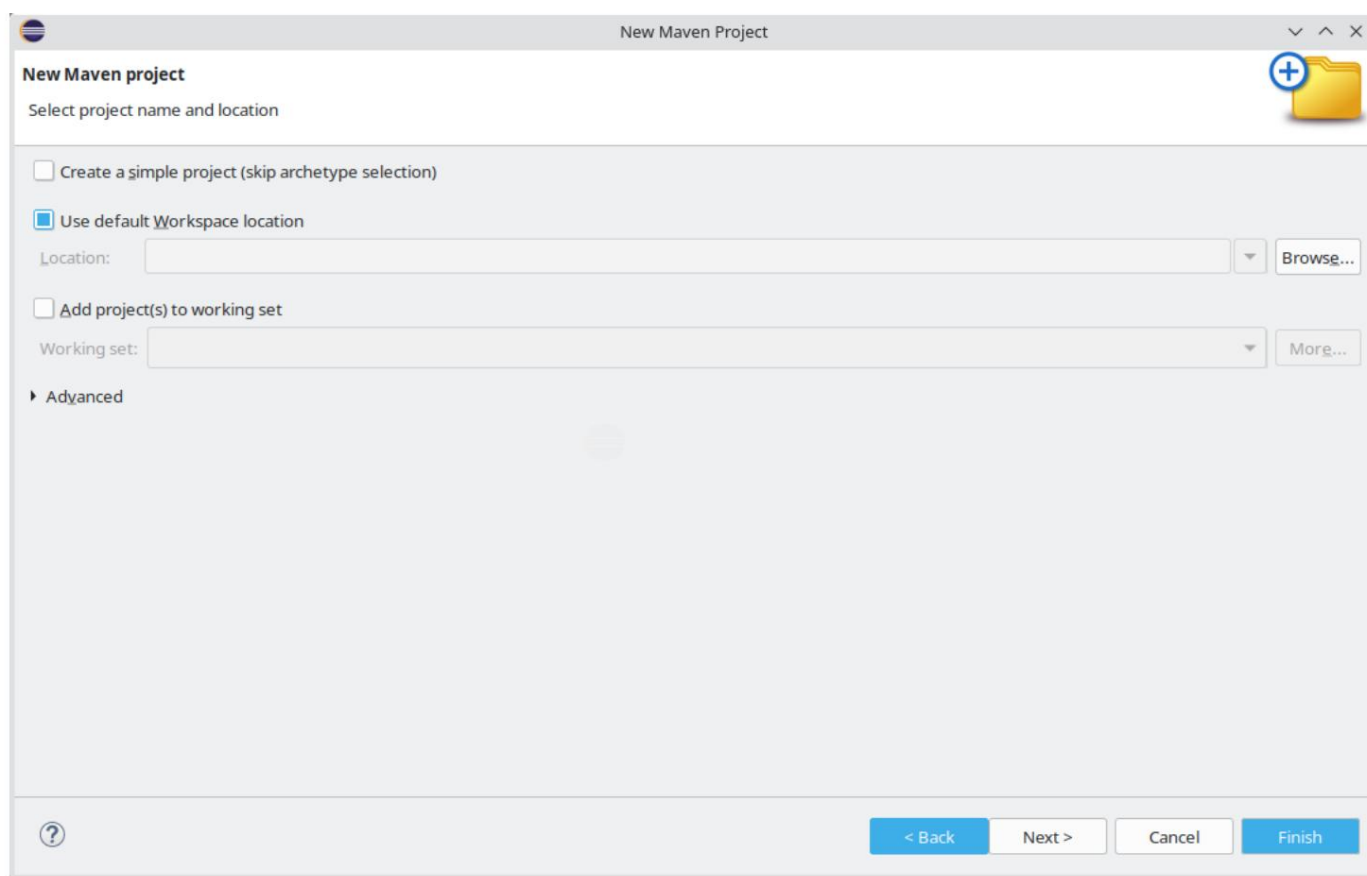
Ahora vamos a crear un proyecto Java para acceder a una base de datos relacional desde cero y ejecutar operaciones CRUD, paso a paso, usando DAO. Nuestro IDE elegido será Eclipse, pero puedes usar cualquier IDE con el que te sientas cómodo.

2. Usando DAO con Java

2.1. Configurando el proyecto

Cree un proyecto Java Maven vacío con estos parámetros como vimos en la UNIDAD 2:





New Maven Project

Select project name and location

☐ Create a simple project (skip archetype selection)

☒ Use default Workspace location

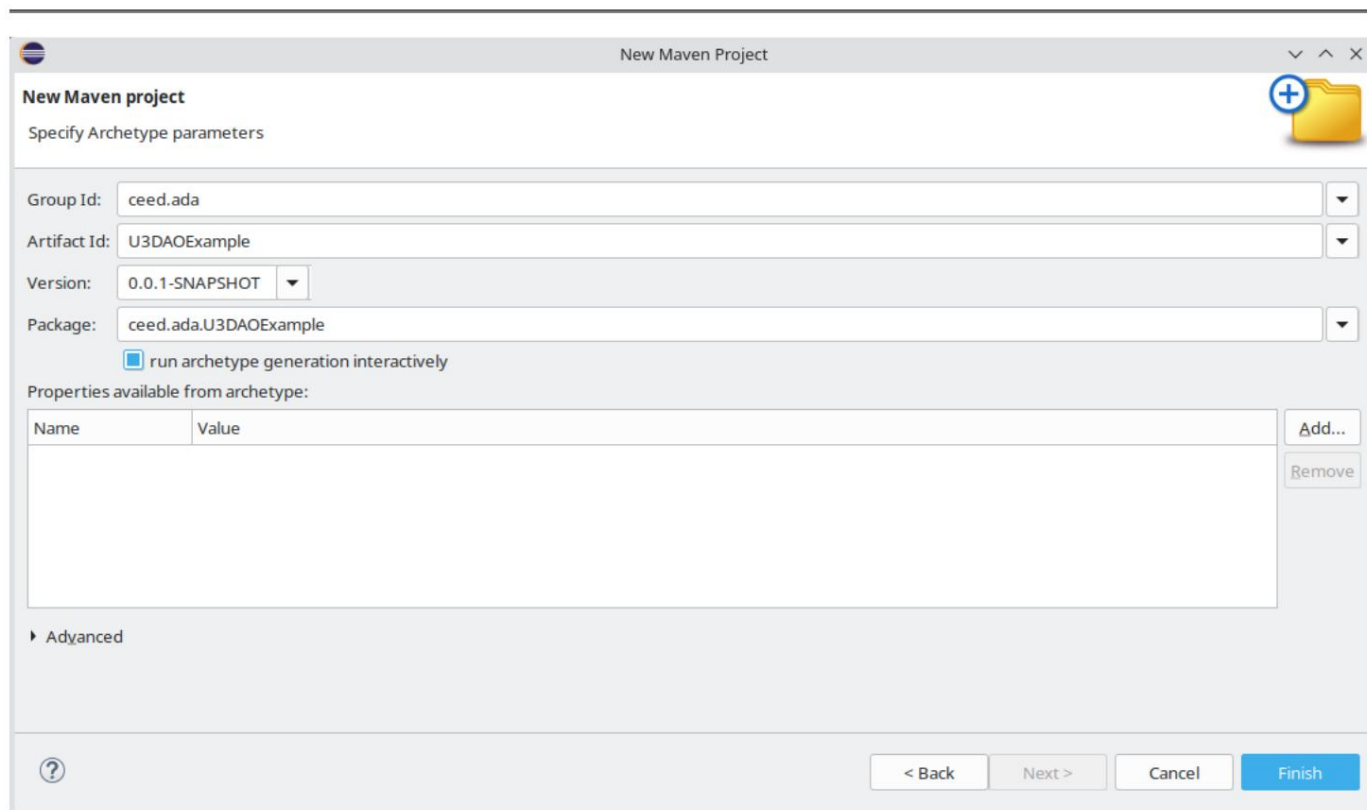
Location: Browse...

☐ Add project(s) to working set

Working set: More...

Advanced

< Back Next > Cancel Finish



New Maven project

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

☒ run archetype generation interactively

Properties available from archetype:

Name	Value
------	-------

Add... Remove

Advanced

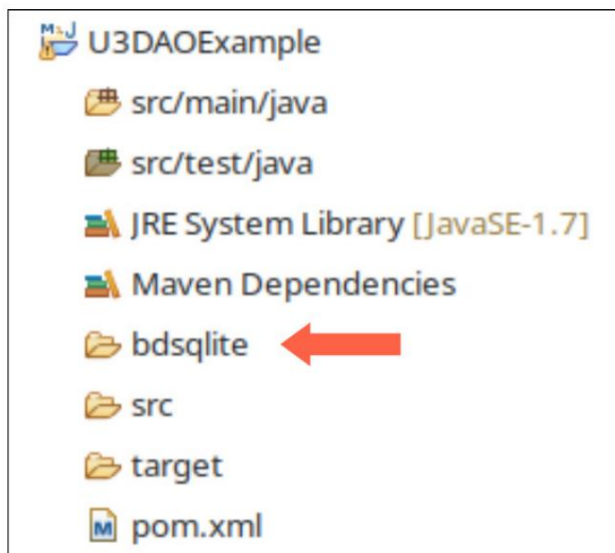
< Back Next > Cancel Finish

2.2. Configurando la base de datos

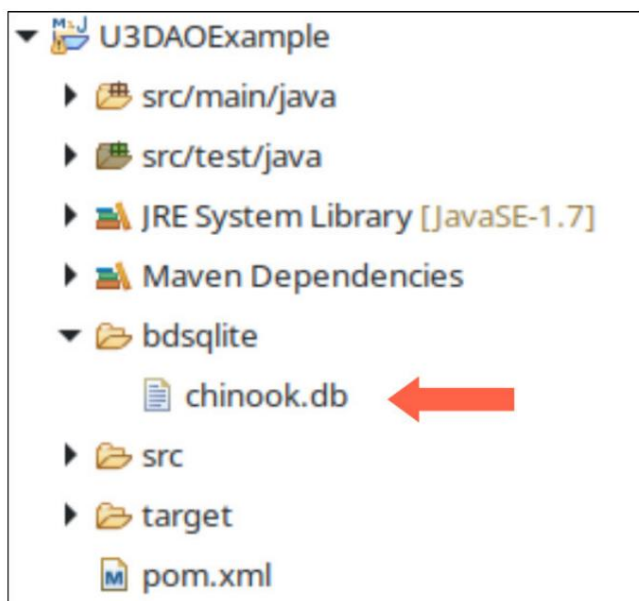
Dado que el objetivo principal de esta unidad es explicar cómo DAO te permite trabajar con cualquier base de datos relacional usando capas centrándose en Java, usaremos una de las bases de datos más simples que puedas encontrar: SQLite.

Siga estos pasos para crear una tabla en una base de datos SQLite existente:

1. Cree una carpeta dentro de su proyecto llamada "bdsqllite".



2. Vaya a esta URL y descargue chinook.db: <https://www.sqlitetutorial.net/sqlite-sample-database/> . Copie la base de datos a la carpeta.



3. Instale SQLite. <https://www.sqlite.org/download.html>

4. Vaya a la consola/terminal y ejecute

```
sqlite3 chinook.db
```

5. Vaya a esta URL si tiene problemas: <https://www.sqlitetutorial.net/download-install-sqlite/>

6. Dentro de la consola sqlite, cree la tabla PERSONAS con este comando:

```
sqlite>CREAR TABLA SI NO EXISTE Personas (  
        personID INTEGER PRIMARY KEY,  
        nombre VARCHAR(255),  
        edad INTEGER);
```

7. Verifique que la tabla se haya creado correctamente con el comando .tables (no agregue "." al final)

```
sqlite> .tables
```

8. Inserte cuatro registros aleatorios dentro de la tabla, como por ejemplo:

```
sqlite>INSERTAR EN VALORES de Personas (1, 'Sergio Fuentes', 20); sqlite>INSERT  
INTO People VALUES (2, 'Juan Hernández', 23); sqlite>INSERT INTO People VALUES  
(3, 'Ana Del Río', 34); sqlite>INSERT INTO People VALUES (4, 'Laura Pérez', 31);
```

9. Verifique que todo salió como se esperaba y cierre la consola SQLite:

```
sqlite> SELECCIONAR * DE  
Personas; sqlite> .salir
```

2.3. Conexión a la base de datos

Siga los pasos descritos en la UNIDAD 2 para crear un proyecto Java Maven para acceder a un sencillo Base de datos SQLite con una tabla llamada Personas con las columnas:

- ID de persona
- nombre
- edad

1. Debe configurar las dependencias en el archivo POM (asegúrese de configurar el número de versión correcto)

```
<?xml versión="1.0" codificación="UTF-8"?>

<proyecto xmlns="http://maven.apache.org/POM/4.0.0"                xmlns:xsi="http://www.w3.org/2001/
    esquemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/m xsi:
<modelVersion>4.0.0</modelVersion>

<groupId>ceed.ada</groupId>
<artifactId>U3DAOExample</artifactId>
<versión>0.0.1-SNAPSHOT</versión>

<name>U3DAOExample</name>
<!-- FIXME cámbielo al sitio web del proyecto --> <url>http://
www.example.com</url>

<propiedades>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
</propiedades>

<dependencias>
```

```
<dependencia>
  <groupId>junit</groupId>
  <artifactId>junit</artifactId> <versión>4.11</
  versión> <scope>test</scope> </
  dependency> <!-- https://
  mvnrepository.com/
  artifact/ org.xerial/sqlite-jdbc --> <dependencia>

  <groupId>org.xerial</groupId>
  <artifactId>sqlite-jdbc</artifactId> <versión>3.39.3.0</
  versión>

</dependencia> </
dependencias> </
proyecto>
```

2. Cree una nueva clase llamada **ConnectToDB** dentro de un nuevo paquete llamado DATA, que contenga:

- La cadena de conexión como variable final estática.
- Un método para conectarse a la base de datos.
- Varios métodos para sobrecargar los métodos predefinidos close() de cada objeto que vamos a utilizar mientras trabajamos con bases de datos como ResultSet, Statement, PreparedStatement y Connection.
- El código podría ser este:



DATOS del paquete ;

```
importar java.sql.Statement;
importar java.sql.Conexión;
```

```
importar java.sql.DriverManager;
importar java.sql.PreparedStatement;
importar java.sql.ResultSet;
importar java.sql.SQLException;

/**
 * =====
 *
 * Programa para gestionar operaciones de conexión y cierre de bases de datos*
 * @autor Abelardo Martínez. Basado y modificado de Sergio Badal.
 * =====
 */

conexiónDB de clase pública { /**
 *
 * =====
 *
 * CONSTANTES Y VARIABLES GLOBALES
 *
 * =====
 */
//Conexión URL
privada estática final String URL = "jdbc:sqlite:bdsqllite/chinook.db"; //versión de Linux //URL de cadena final
estática privada = "jdbc:sqlite:bdsqllite\\chinook.db"; //ventanas

/*
 * =====
 *
 * GESTIÓN DE CONEXIÓN
 *
 * =====
 */
}

*
* Método estático: simplemente intenta conectarte a la base de datos
*/

Conexión estática pública ConnectToDB() {
    intento
    { Conexión cnDB = DriverManager.getConnection(URL);
    devolver cnDB;
    } captura (SQLException sqle) {
        System.out.println("Algo salió mal al intentar conectar con el da
```

```
        sqlc.printStackTrace(System.out);
    }
    devolver nulo;
}

/*
 * Método estático: simplemente intenta desconectarte de la base de datos
 */
cierres público estático vacío (Conexión cnDB) lanza SQLException {
    cnDB.close();
}

/*
 * -----
 * RECURSOS DE CIERRE
 * -----
 */

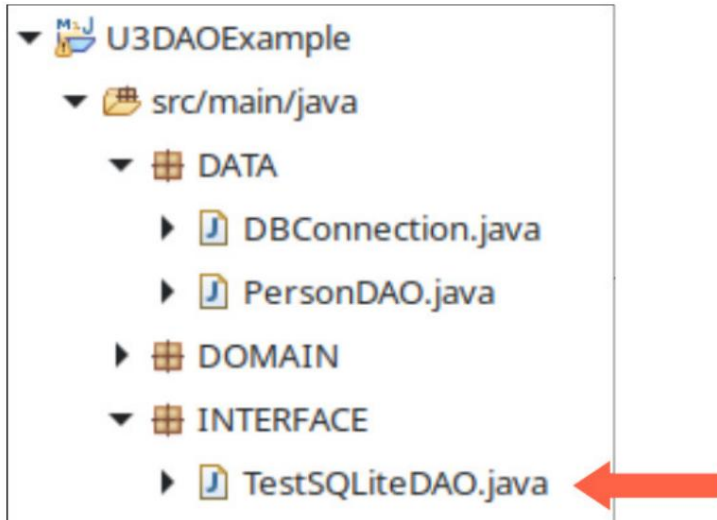
//ResultSet
public static void close(ResultSet rsCursor) lanza SQLException { rsCursor.close();
}

// Declaración
pública estática vacía cerrada (Declaración staSQL) lanza SQLException {
    staSQL.close();
}

//PreparedStatement
public static void close(PreparedStatement pstaSQL) lanza SQLException {
    pstaSQL.close();
}
}
```

3. Finalmente, solo necesitamos un método principal para comprobar que la conexión funciona correctamente.

- Puede crear un nuevo paquete llamado INTERFACE.



- El código podría ser este:

DEBE comprender cada línea de este código antes de continuar con esta UNIDAD 3. Vaya a la documentación ampliada de la UNIDAD 2 para revisar todos los conocimientos requeridos.

```
importar java.sql.*;
importar DATA.DBConnection;

clase pública TestSQLiteDAO { public
    static void main(String[] stArgs) {
        intente
        { Conexión cnDB = DBConnection.ConnectToDB(); Declaración
        staSQL = cnDB.createStatement(); String stSQLSelect =
        "SELECCIONAR * DE Personas"; System.out.println("Ejecutando:
        " + stSQLSelect); ResultSet rsPerson =
        staSQL.executeQuery(stSQLSelect); int iNumItems = 0; mientras
        (rsPerson.next()) {

            iNumItems++;

            System.out.println("ID: " + rs.getInt("personalID"));
            System.out.println("Nombre: " + rs.getString("nombre"));
            System.out.println("Edad: " + rs.getInt("edad"));
```



```
    }  
    si (iNumItems == 0)  
        System.out.println("No se encontraron elementos en la tabla Personas");  
    ConnectToDB.close(rsPersona); //cerrar conjunto de  
    resultados ConnectToDB.close(cnDB); //cerrar conexión a la base de datos  
} captura (SQLException sqle) {  
    System.out.println("¡Algo salió mal mientras leía!");  
    sqle.printStackTrace(System.out);  
}  
  
}  
  
}
```

2.4. Crear clases y leer datos.

Ahora necesitamos crear dos clases para cada entidad/tabla de la base de datos:

A) **Person.java** para almacenar los datos de cada fila de la base de datos.

- Cree esta clase dentro de un nuevo paquete llamado DOMINIO.
- Definir los atributos, setters, getters y 4 constructores:
 - Constructor vacío
 - Constructor que proporciona solo el ID (por ejemplo, para eliminar solo necesitamos tener el personId, cual es la clave principal -PK-)
 - Constructor para insertar un nuevo registro (no necesitamos especificar el personId)
 - Constructor para modificar un registro (necesita todos los atributos)

```
paquete DOMINIO;
```

```
/**  
 *  
 * =====  
 *  
 *Persona Objeto*  
 *  
 * @autor Abelardo Martínez. Basado y modificado de Sergio Badal.  
 *  
 * =====  
 */
```

```
Persona de clase pública {
```

```
/*  
 *  
 * =====  
 *  
 * ATRIBUTOS  
 *  
 * =====  
 */
```

```
identificación interna  
privada ; cadena privada  
stName; edad privada ;
```

```
/*
 * -----

 * MÉTODOS
 * -----

 */

/*
 * Constructor vacío */

Persona pública () { } /

*

* Constructor de identificación. Sólo clave primaria
*/

Persona pública (int iID) {
    this.iID = iID;

} /

* * Constructor sin ID. Todos los campos, excepto la clave principal */

Persona pública (String stName, int iEdad) {
    this.stName = stName;
    this.iEdad = iEdad;

} /*
 * Constructor completo con todos los campos.
 */

Persona pública (int iID, String stName, int iEdad) {
    this.iID = iID;
    this.stName = stName;
    this.iEdad = iEdad;
}

/*
 * -----

 * RECOGEDORES
 * -----
```

```
*/  
  
público int getPersonID() {  
    devolver ID;  
}  
  
public int getAge() { return  
    iAge;  
}  
  
cadena pública getName() {  
    devolver stName;  
}  
  
/*  
 * -----  
  
 * CONFIGURADORES  
 * -----  
  
 */  
  
setPersonID público vacío (int iID) {  
    this.iID = iID;  
}  
  
public void setName(String stName)  
    { this.stName = stName;  
}  
  
setAge público vacío (int iAge) {  
    this.iEdad = iEdad;  
}  
  
/*  
 * -----  
  
 * MÉTODOS DE FORMATO DE DATOS  
 * -----  
  
 */  
  
@Anular
```

```
cadena pública toString() {  
    devolver "Persona [ID = " + iID + ", Nombre = " + stNombre + ", Edad = " + iEdad  
}  
}
```

B) **PersonaDAO.java** para gestionar las operaciones entre la base de datos y la clase anterior

- Cree esta clase dentro de un nuevo paquete llamado datos
- Definir las declaraciones de acceso a la base de datos al comienzo de la clase como variables estáticas (bueno práctica)
- Crear un método para leer todos los elementos de esta entidad

```
importar java.sql.Conexión;  
importar java.sql.PreparedStatement;  
importar java.sql.ResultSet;  
importar java.sql.SQLException;  
importar java.util.ArrayList;  
  
importar DOMINIO.Persona;  
  
/**  
 * =====  
 * Programa para gestionar operaciones CRUD a la BD  
 * @autor Abelardo Martínez. Basado y modificado de Sergio Badal.  
 * =====  
 */  
  
clase pública PersonaDAO {  
  
    /**  
     * =====  
     * CONSTANTES Y VARIABLES GLOBALES  
     * =====  
     */  
}
```

```

/*
 * -----

 * CONSULTAS SQL
 * -----

 */

Cadena final estática privada SELECT = "SELECT * FROM People";

/*
 * -----

 * SELECCIONAR
 * -----

 */

public ArrayList<Persona> SelectAllPeople() lanza SQLException {

    Conexión cnDB = nula;
    PreparedStatement pstasqlSelect = nulo;
    Conjunto de resultados rsPerson = nulo;
    Persona objPerson = nulo;
    ArrayList<Persona> arlPerson = new ArrayList<Persona>();

    intentar {
        cnDB = DBConnection.ConnectToDB(); //conectar a la base de datos
        pstasqlSelect = cnDB.prepareStatement(SELECT); // instrucción de selección
        rsPerson = pstasqlSelect.executeQuery(); //ejecutar seleccionar mientras
        (rsPerson.next()) {
            int iID = rsPerson.getInt("personalID"); String
            stName = rsPerson.getString("nombre"); int iEdad =
            rsPerson.getInt("edad"); objPerson = nueva
            Persona(iID, stName, iEdad); arlPerson.add(objPerson);

        }
    } captura (SQLException sqle) {
        System.out.println("Algo salió mal al leer de la tabla de personas" sqle.printStackTrace(System.out));
    } finalmente {
        DBConnection.close(rsPerson); //cerrar conjunto de resultados
        DBConnection.close(pstasqlSelect); //cerrar declaración preparada
    }
}

```

```

        DBConnection.close(cnDB); //cerrar conexión a la base de datos
    }
    devolver arlPersona;
}
}

```

Finalmente, necesitamos usar la clase TestSQLiteDAO para crear una instancia de PersonDAO y simplemente enumerar los registros de la tabla Personas. Tenga en cuenta que no se requiere manejo de excepciones ya que se manejan dentro de las clases DAO.

```

importar java.sql.*;
importar java.util.ArrayList;

datos de importacion .*;
importar DOMINIO.*;

clase pública TestSQLiteDAO {
    /*
     * -----
     * PROGRAMA PRINCIPAL
     * -----
     */

    public static void main(String[] stArgs) lanza SQLException {

        PersonaDAO objPersonDAO = nueva PersonaDAO();
        ArrayList<Persona> arlPersona;

        //
        SELECCIONAR System.out.println("**** LISTA DE REGISTROS
        INICIALES"); arlPerson = objPersonDAO.SelectAllPeople();
        for (Persona objPerson : arlPerson)
            { System.out.println("Persona: " + objPerson);
            }
    }
}

```

```
}
```


2.5. Insertando datos

Para permitir la inserción de datos solo necesitamos crear otro método dentro de la clase DAO y otra oración SQL. La gran diferencia ahora es que necesitamos establecer algunos "marcadores" para agregar valores más adelante:

```
cadena final estática privada INSERT = "INSERT INTO Personas (nombre, edad) VALORES (?,?)

/*
 * -----
 * INSERTAR
 * -----
 */

public int InsertPerson(Persona objPerson) lanza SQLException {

    Conexión cnDB = nula;
    PreparedStatement pstaSQLInsert = nulo;
    int iNumreg = 0;

    intente
    { cnDB = DBConnection.ConnectToDB(); // conectarse a la base
      de datos pstaSQLInsert = cnDB.prepareStatement(INSERT); //insertar declaración
      pstaSQLInsert.setString(1, objPerson.getName());
      pstaSQLInsert.setInt(2, objPerson.getAge());
      iNumreg = pstaSQLInsert.executeUpdate(); //ejecutar inserción
    } captura (SQLException sqle) {
        System.out.println("¡Algo anda mal al insertar!");
        sqle.printStackTrace(System.out);

    } finalmente {
        DBConnection.close(pstaSQLInsert); //cerrar declaración preparada
        DBConnection.close(cnDB); //cerrar conexión a la base de datos
    }
}
```

```
        devolver iNumreg;  
    }
```

2.6. Eliminar datos

Para permitir la eliminación de datos solo necesitamos crear otro método dentro de la clase DAO y otra oración SQL. También necesitamos establecer algunos "marcadores" para agregar valores más adelante:

```
cadena final estática privada DELETE = "ELIMINAR DE Personas DONDE personID =?"

/*
 * -----
 * BORRAR
 * -----
 */

public int DeletePerson (Persona objPerson) lanza SQLException {

    Conexión cnDB = nula;
    PreparedStatement pstaSQLDelete = nulo;
    int iNumreg = 0;

    intente
    { cnDB = DBConnection.ConnectToDB(); // conectarse a la base
      de datos pstaSQLDelete = cnDB.prepareStatement(DELETE); //eliminar declaración
      pstaSQLDelete.setInt(1, objPerson.getPersonID()); iNumreg =
      pstaSQLDelete.executeUpdate(); //ejecutar eliminar
    } captura (SQLException sqle) {
        System.out.println("¡Algo anda mal al eliminar!");
        sqle.printStackTrace(System.out);
    } finalmente {
        DBConnection.close(pstaSQLDelete); //cerrar declaración preparada
        DBConnection.close(cnDB); //cerrar conexión a la base de datos
    } devolver iNumreg;
}
```


2.7. Actualizando datos

Para permitir la actualización de datos solo necesitamos crear otro método dentro de la clase DAO y otra oración SQL. También necesitamos establecer algunos "marcadores" para agregar valores más adelante:

Cadena `final` `estática` `privada` ACTUALIZAR = "ACTUALIZAR Personas SET nombre =?, edad =? DONDE p

```
/*
 * -----
 * ACTUALIZAR
 * -----
 */
```

```
public int UpdatePerson (Persona objPerson) lanza SQLException {
```

```
    Conexión cnDB = nula;
```

```
    PreparedStatement pstaSQLUpdate = nulo;
```

```
    int iNumreg = 0;
```

```
    intente
```

```
        { cnDB = DBConnection.ConnectToDB(); // conectarse a la base
```

```
        de datos pstaSQLUpdate = cnDB.prepareStatement(UPDATE); // declaración de
```

```
        actualización pstaSQLUpdate.setString(1, objPerson.getName());
```

```
        pstaSQLUpdate.setInt(2, objPerson.getAge());
```

```
        pstaSQLUpdate.setInt(3, objPerson.getPersonID()); iNumreg =
```

```
        pstaSQLUpdate.executeUpdate(); //ejecutar eliminar
```

```
    } captura (SQLException sqle) {
```

```
        System.out.println("¡Algo anda mal al actualizar!");
```

```
        sqle.printStackTrace(System.out);
```

```
    } finalmente {
```

```
        DBConnection.close(pstaSQLUpdate); //cerrar declaración preparada
```

```
        DBConnection.close(cnDB); //cerrar conexión a la base de datos
```

```
    } devolver iNumreg;
```

```
}
```

2.8. Operaciones CRUD (clase DAO final)

Si juntamos todos los métodos podemos obtener algo como esto:

```
DATOS del paquete ;

importar java.sql.Conexión; importar
java.sql.PreparedStatement; importar
java.sql.ResultSet; importar
java.sql.SQLException;
importar java.util.ArrayList;

importar DOMINIO.Persona;

/**
 * =====
 *
 *Programa para gestionar operaciones CRUD a la BD*
 @autor Abelardo Martínez. Basado y modificado de Sergio Badal.
 * =====
 */

clase pública PersonaDAO {

    /**
     * -----
     *
     * CONSTANTES Y VARIABLES GLOBALES
     * -----
     *
     * //
     *
     * -----
     *
     * CONSULTAS SQL
     * -----
     *
     */
```

Cadena **final** **estática** **privada** SELECT = "SELECT * FROM People"; cadena
final **estática** **privada** INSERT = "INSERT INTO Personas (nombre, edad) VALORES (?,?)
cadena **final** **estática** **privada** DELETE = "ELIMINAR DE Personas DONDE personID =?" Cadena
final **estática** **privada** ACTUALIZAR = "ACTUALIZAR Personas SET nombre =?, edad =? DONDE p

```
/*
 * -----
 *
 * SELECCIONAR
 * -----
 */
```

```
public ArrayList<Persona> SelectAllPeople() lanza SQLException {
```

```
    Conexión cnDB = nula;
```

```
    PreparedStatement pstaSQLSelect = nulo;
```

```
    Conjunto de resultados rsPerson = nulo;
```

```
    Persona objPerson = nulo;
```

```
    ArrayList<Persona> arlPerson = new ArrayList<Persona>();
```

```
    intente
```

```
    { cnDB = DBConnection.ConnectToDB(); // conectarse a la base
```

```
    de datos pstaSQLSelect = cnDB.prepareStatement(SELECT); // instrucción de
```

```
    selección rsPerson = pstaSQLSelect.executeQuery(); //ejecutar seleccionar
```

```
    mientras (rsPerson.next()) {
```

```
        int iID = rsPerson.getInt("personalID"); String
```

```
        stName = rsPerson.getString("nombre"); int iEdad =
```

```
        rsPerson.getInt("edad"); objPerson = nueva
```

```
        Persona(iID, stName, iEdad); arlPerson.add(objPerson);
```

```
    }
```

```
} catch (SQLException sqle)
```

```
    { System.out.println("Algo salió mal al leer de la tabla de personas"
```

```
    sqle.printStackTrace(System.out);
```

```
} finalmente {
```

```
    DBConnection.close(rsPerson); //cerrar conjunto de resultados
```

```
    DBConnection.close(pstaSQLSelect); //cerrar declaración preparada
```

```
    DBConnection.close(cnDB); //cerrar conexión a la base de datos
```

```
}
```



```

        devolver arlPersona;
    }

    /*
     * -----
     * INSERTAR
     * -----
     */

    public int InsertPerson(Persona objPerson) lanza SQLException {

        Conexión cnDB = nula;
        PreparedStatement pstasqlInsert = nulo;
        int iNumreg = 0;

        intente
        { cnDB = DBConnection.ConnectToDB(); // conectarse a la base
          de datos pstasqlInsert = cnDB.prepareStatement(INSERT); //insertar declaración
          pstasqlInsert.setString(1, objPerson.getName());
          pstasqlInsert.setInt(2, objPerson.getAge()); iNumreg =
          pstasqlInsert.executeUpdate(); //ejecutar inserción
        } captura (SQLException sqle) {
            System.out.println("¡Algo anda mal al insertar!");
            sqle.printStackTrace(System.out);

        } finalmente {
            DBConnection.close(pstasqlInsert); //cerrar declaración preparada
            DBConnection.close(cnDB); //cerrar conexión a la base de datos

        } devolver iNumreg;
    }

    /*
     * -----
     * BORRAR
     * -----
     */

    public int DeletePerson (Persona objPerson) lanza SQLException {

```

```

Conexión cnDB = nula;
PreparedStatement pstaSQLDelete = nulo;
int iNumreg = 0;

intente
{
    cnDB = DBConnection.ConnectToDB(); // conectarse a la base
    de datos pstaSQLDelete = cnDB.prepareStatement(DELETE); //eliminar declaración
    pstaSQLDelete.setInt(1, objPerson.getPersonID()); iNumreg =
    pstaSQLDelete.executeUpdate(); //ejecutar eliminar
} captura (SQLException sqle) {
    System.out.println("¡Algo anda mal al eliminar!");
    sqle.printStackTrace(System.out);
} finalmente {
    DBConnection.close(pstaSQLDelete); //cerrar declaración preparada
    DBConnection.close(cnDB); //cerrar conexión a la base de datos

} devolver iNumreg;
}

/*
 * -----
 * ACTUALIZAR
 * -----
 */

public int UpdatePerson (Persona objPerson) lanza SQLException {

    Conexión cnDB = nula;
    PreparedStatement pstaSQLUpdate = nulo; int
    iNumreg = 0;

    intentar {
        cnDB = DBConnection.ConnectToDB(); // conectarse a la base de
        datos pstaSQLUpdate = cnDB.prepareStatement(UPDATE); // declaración de
        actualización pstaSQLUpdate.setString(1, objPerson.getName());
        pstaSQLUpdate.setInt(2, objPerson.getAge());
        pstaSQLUpdate.setInt(3, objPerson.getPersonID());
    }
}

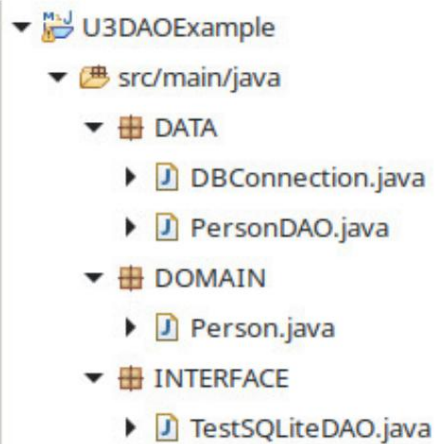
```

```
        iNumreg = pstaSQLUpdate.executeUpdate(); //ejecutar eliminar
    } captura (SQLException sqle) {
        System.out.println("¡Algo anda mal al actualizar!");
        sqle.printStackTrace(System.out);
    } finalmente {
        DBConnection.close(pstaSQLUpdate); //cerrar declaración preparada
        DBConnection.close(cnDB); //cerrar conexión a la base de datos

    } devolver iNumreg;
}

}
```

Tu proyecto debería verse así:



3. Ejemplo completo de CRUD usando DAO

Ahora que tenemos todos los métodos definidos en la clase DAO, podemos operar con la base de datos. Eche un vistazo a este sencillo ejemplo de CRUD:

```
paquete INTERFAZ;

importar java.sql.*;
importar java.util.ArrayList;

datos de
importacion .*; importar DOMINIO.*;

/**
 * =====
 *
 * Programa para gestionar
 * PERSONAS* @autor Abelardo Martínez. Basado y modificado de Sergio Badal.
 * =====
 */

clase pública TestSQLiteDAO {
    /*
     * -----
     *
     * PROGRAMA PRINCIPAL
     * -----
     */

    public static void main(String[] stArgs) lanza SQLException {

        PersonaDAO objPersonDAO = nueva PersonaDAO();
        ArrayList<Persona> arlPersona;

        //
        SELECCIONAR System.out.println("**** LISTA DE REGISTROS
        INICIALES"); arlPerson = objPersonDAO.SelectAllPeople();
        for (Persona objPerson : arlPerson)
            { System.out.println("Persona: " + objPerson);
```

```
}

//
INSERTAR System.out.println("**** INSERTAR NUEVOS
REGISTROS"); Persona objPerson1 = nueva Persona("Herminia
Fuster", 55); objPersonDAO.InsertPerson(objPerson1);
Persona objPerson2 = nueva Persona("Mario Caballero", 25);
objPersonDAO.InsertPerson(objPerson2);

// BORRAR
System.out.println("**** BORRAR EL PRIMER REGISTRO");
Persona objPersonToDelete = nueva Persona(1);
objPersonDAO.DeletePerson(objPersonToDelete);

//
ACTUALIZAR System.out.println("**** ACTUALIZAR EL
SEGUNDO REGISTRO"); Persona objPersonToUpdate = nueva Persona(2, "Juan Manuel
Hernández", 22); objPersonDAO.UpdatePerson(objPersonToUpdate);

//RESULTADO DE LOS
DATOS FINALES System.out.println("**** LISTA DE
REGISTROS FINALES"); arlPerson =
objPersonDAO.SelectAllPeople(); for (Persona
    objPerson : arlPerson) { System.out.println("Persona: " + objPerson);
}
}
}
```

4. Bibliografía

Fuentes

- Cómo crear un proyecto Maven en Eclipse. <https://www.simplilearn.com/tutorials/maven-tutorial/proyecto-maven-en-eclipse>
- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. desenvolupament d'aplicacions multiplataforma. Comunes creatives. Departament de Ensenyament, Institut Obert de Catalunya. Dipòsit legal: B. 29430-2013. <https://ioc.xtec.cat/educacio/recursos>
- Alberto Oliva Molina. Acceso a datos. UD 3. Herramientas de mapeo de objetos relacionales (ORM). IES Tubalcaín. Tarazona (Zaragoza, España).
- Tutorial SQLite. Base de datos de muestra SQLite. <https://www.sqlitetutorial.net/sqlite-sample-base-de-datos/>



Licenciado bajo la [licencia Creative Commons Attribution Share Alike 4.0](https://creativecommons.org/licenses/by-sa/4.0/)