**Guía Completa: JDBC con Java, Gradle, H2 y Operaciones CRUD**

**Índice**

1. [Introducción a JDBC](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#introducci%C3%B3n-a-jdbc)
2. [Configuración del Proyecto](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#configuraci%C3%B3n-del-proyecto)
   * [Requisitos Previos](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#requisitos-previos)
   * [Creación del Proyecto en IntelliJ IDEA](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#creaci%C3%B3n-del-proyecto-en-intellij-idea)
   * [Configuración de Gradle](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#configuraci%C3%B3n-de-gradle)
3. [Configuración de la Base de Datos H2](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#configuraci%C3%B3n-de-la-base-de-datos-h2)
   * [Conexión a la Base de Datos](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#conexi%C3%B3n-a-la-base-de-datos)
   * [Creación de Tablas](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#creaci%C3%B3n-de-tablas)
4. [Estructura del Proyecto](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#estructura-del-proyecto)
5. [Implementación de Operaciones CRUD](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#implementaci%C3%B3n-de-operaciones-crud)
   * [Conexión a la Base de Datos](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#conexi%C3%B3n-a-la-base-de-datos-1)
   * [Crear (Create)](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#crear-create)
   * [Leer (Read)](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#leer-read)
   * [Actualizar (Update)](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#actualizar-update)
   * [Eliminar (Delete)](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#eliminar-delete)
6. [Manejo de Excepciones](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#manejo-de-excepciones)
7. [Buenas Prácticas](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#buenas-pr%C3%A1cticas)
8. [Ejemplo Completo](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#ejemplo-completo)
9. [Recursos Adicionales](https://claude.ai/chat/88d34a2a-a1b7-4791-8505-73e0a2e87562#recursos-adicionales)

**Introducción a JDBC**

JDBC (Java Database Connectivity) es una API de Java que permite la conexión y manipulación de bases de datos desde aplicaciones Java. Con JDBC puedes:

* Establecer una conexión con una base de datos
* Enviar declaraciones SQL
* Procesar los resultados

JDBC proporciona una interfaz uniforme para interactuar con diferentes sistemas de gestión de bases de datos (SGBD), como H2, MySQL, PostgreSQL, entre otros.

**Configuración del Proyecto**

**Requisitos Previos**

Para seguir esta guía, necesitarás:

* JDK 11 o superior instalado
* IntelliJ IDEA (Community o Ultimate)
* Conocimientos básicos de Java y SQL

**Creación del Proyecto en IntelliJ IDEA**

1. Abre IntelliJ IDEA
2. Selecciona **File** > **New** > **Project**
3. Selecciona **Gradle** en el panel izquierdo
4. Configura:
   * Name: jdbc-crud-example
   * Location: Elige una ubicación para tu proyecto
   * Language: Java
   * Build system: Gradle
   * JDK: Selecciona tu versión de JDK (11 o superior)
   * Gradle DSL: Groovy
5. Haz clic en **Next** y luego en **Finish**

**Configuración de Gradle**

Modifica el archivo build.gradle.kts para incluir las dependencias necesarias:

plugins {

id("java")

id("application")

}

group = "com.example"

version = "1.0-SNAPSHOT"

repositories {

mavenCentral()

}

dependencies {

// H2 Database

implementation("com.h2database: h2:2.3.232")

}

application {

mainClass.set("com.example.Main")

}

tasks.test {

useJUnitPlatform()

}

java {

sourceCompatibility = JavaVersion.VERSION\_17

targetCompatibility = JavaVersion.VERSION\_17

}

Después de modificar el archivo build.gradle.kts, asegúrate de actualizar el proyecto:

* Haz clic en el icono de "Reload Gradle Project" (icono de actualización en la ventana de Gradle)
* O ejecuta el comando ./gradlew build desde la terminal

**Configuración de la Base de Datos H2**

H2 es una base de datos SQL escrita en Java que puede funcionar en modo en memoria o en modo archivo. Para nuestro proyecto, usaremos el modo archivo para poder persistir los datos.

**Conexión a la Base de Datos**

Hay varias formas de conectar a una base de datos H2:

1. **Modo de archivo**: Los datos se almacenan en el sistema de archivos
2. jdbc:h2:./data/nombredb
3. **Modo en memoria**: Los datos se pierden cuando la aplicación termina
4. jdbc:h2:mem:nombredb
5. **Modo servidor**: H2 se ejecuta como un servidor independiente
6. jdbc:h2:tcp://localhost/~/nombredb

Para nuestro ejemplo, usaremos el modo de archivo.

**Creación de Tablas**

Crearemos una tabla simple para gestionar productos:

CREATE TABLE IF NOT EXISTS productos (

id INT AUTO\_INCREMENT PRIMARY KEY,

nombre VARCHAR(100) NOT NULL,

descripcion VARCHAR(255),

precio DECIMAL(10,2) NOT NULL,

stock INT DEFAULT 0

);

**Estructura del Proyecto**

Para un proyecto JDBC bien organizado, recomendamos la siguiente estructura:

src/main/java/com/example/

├── Main.java # Punto de entrada de la aplicación

├── model/

│ └── Producto.java # Clase modelo para la entidad Producto

├── dao/

│ ├── ProductoDAO.java # Interfaz DAO para operaciones CRUD

│ └── ProductoDAOImpl.java # Implementación de la interfaz DAO

├── util/

│ └── DatabaseUtil.java # Clase para gestionar conexiones a la BD

└── config/

└── DatabaseConfig.java # Configuraciones de la base de datos

**Implementación de Operaciones CRUD**

**Conexión a la Base de Datos**

Primero, creemos una clase utilitaria para gestionar la conexión a la base de datos:

package com.example.util;

import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;

import java.sql.Statement;

public class DatabaseUtil {

private static final String DB\_URL = "jdbc:h2:./data/productosdb";

private static final String DB\_USER = "sa";

private static final String DB\_PASSWORD = "";

// Método para obtener una conexión

public static Connection getConnection() throws SQLException {

return DriverManager.getConnection(DB\_URL, DB\_USER, DB\_PASSWORD);

}

// Método para cerrar recursos

public static void closeResources(Connection conn, Statement stmt) {

try {

if (stmt != null) stmt.close();

} catch (SQLException e) {

e.printStackTrace();

}

try {

if (conn != null) conn.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

// Método para inicializar la base de datos

public static void initDatabase() {

Connection conn = null;

Statement stmt = null;

try {

conn = getConnection();

stmt = conn.createStatement();

// Crear tabla productos si no existe

String sql = "CREATE TABLE IF NOT EXISTS productos (" +

"id INT AUTO\_INCREMENT PRIMARY KEY, " +

"nombre VARCHAR(100) NOT NULL, " +

"descripcion VARCHAR(255), " +

"precio DECIMAL(10,2) NOT NULL, " +

"stock INT DEFAULT 0)";

stmt.execute(sql);

System.out.println("Base de datos inicializada correctamente");

} catch (SQLException e) {

e.printStackTrace();

} finally {

closeResources(conn, stmt);

}

}

}

**Clase Modelo**

Ahora, definamos la clase modelo para representar un producto:

package com.example.model;

public class Producto {

private int id;

private String nombre;

private String descripcion;

private double precio;

private int stock;

// Constructor vacío

public Producto() {

}

// Constructor sin ID (para nuevos productos)

public Producto(String nombre, String descripcion, double precio, int stock) {

this.nombre = nombre;

this.descripcion = descripcion;

this.precio = precio;

this.stock = stock;

}

// Constructor completo

public Producto(int id, String nombre, String descripcion, double precio, int stock) {

this.id = id;

this.nombre = nombre;

this.descripcion = descripcion;

this.precio = precio;

this.stock = stock;

}

// Getters y Setters

public int getId() {

return id;

}

public void setId(int id) {

this.id = id;

}

public String getNombre() {

return nombre;

}

public void setNombre(String nombre) {

this.nombre = nombre;

}

public String getDescripcion() {

return descripcion;

}

public void setDescripcion(String descripcion) {

this.descripcion = descripcion;

}

public double getPrecio() {

return precio;

}

public void setPrecio(double precio) {

this.precio = precio;

}

public int getStock() {

return stock;

}

public void setStock(int stock) {

this.stock = stock;

}

@Override

public String toString() {

return "Producto{" +

"id=" + id +

", nombre='" + nombre + '\'' +

", descripcion='" + descripcion + '\'' +

", precio=" + precio +

", stock=" + stock +

'}';

}

}

**Interfaz DAO**

A continuación, creemos una interfaz que defina las operaciones CRUD:

package com.example.dao;

import com.example.model.Producto;

import java.util.List;

public interface ProductoDAO {

// Create

int crear(Producto producto);

// Read

Producto buscarPorId(int id);

List<Producto> listarTodos();

// Update

boolean actualizar(Producto producto);

// Delete

boolean eliminar(int id);

}

**Implementación DAO**

Ahora implementaremos la interfaz ProductoDAO con las operaciones CRUD:

package com.example.dao;

import com.example.model.Producto;

import com.example.util.DatabaseUtil;

import java.sql.\*;

import java.util.ArrayList;

import java.util.List;

public class ProductoDAOImpl implements ProductoDAO {

@Override

public int crear(Producto producto) {

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

int generatedId = -1;

try {

conn = DatabaseUtil.getConnection();

String sql = "INSERT INTO productos (nombre, descripcion, precio, stock) VALUES (?, ?, ?, ?)";

stmt = conn.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS);

stmt.setString(1, producto.getNombre());

stmt.setString(2, producto.getDescripcion());

stmt.setDouble(3, producto.getPrecio());

stmt.setInt(4, producto.getStock());

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

rs = stmt.getGeneratedKeys();

if (rs.next()) {

generatedId = rs.getInt(1);

}

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

// Cerrar ResultSet

if (rs != null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

DatabaseUtil.closeResources(conn, stmt);

}

return generatedId;

}

@Override

public Producto buscarPorId(int id) {

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

Producto producto = null;

try {

conn = DatabaseUtil.getConnection();

String sql = "SELECT \* FROM productos WHERE id = ?";

stmt = conn.prepareStatement(sql);

stmt.setInt(1, id);

rs = stmt.executeQuery();

if (rs.next()) {

producto = new Producto(

rs.getInt("id"),

rs.getString("nombre"),

rs.getString("descripcion"),

rs.getDouble("precio"),

rs.getInt("stock")

);

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

// Cerrar ResultSet

if (rs != null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

DatabaseUtil.closeResources(conn, stmt);

}

return producto;

}

@Override

public List<Producto> listarTodos() {

Connection conn = null;

Statement stmt = null;

ResultSet rs = null;

List<Producto> productos = new ArrayList<>();

try {

conn = DatabaseUtil.getConnection();

stmt = conn.createStatement();

String sql = "SELECT \* FROM productos";

rs = stmt.executeQuery(sql);

while (rs.next()) {

Producto producto = new Producto(

rs.getInt("id"),

rs.getString("nombre"),

rs.getString("descripcion"),

rs.getDouble("precio"),

rs.getInt("stock")

);

productos.add(producto);

}

} catch (SQLException e) {

e.printStackTrace();

} finally {

// Cerrar ResultSet

if (rs != null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

DatabaseUtil.closeResources(conn, stmt);

}

return productos;

}

@Override

public boolean actualizar(Producto producto) {

Connection conn = null;

PreparedStatement stmt = null;

boolean actualizado = false;

try {

conn = DatabaseUtil.getConnection();

String sql = "UPDATE productos SET nombre = ?, descripcion = ?, precio = ?, stock = ? WHERE id = ?";

stmt = conn.prepareStatement(sql);

stmt.setString(1, producto.getNombre());

stmt.setString(2, producto.getDescripcion());

stmt.setDouble(3, producto.getPrecio());

stmt.setInt(4, producto.getStock());

stmt.setInt(5, producto.getId());

int affectedRows = stmt.executeUpdate();

actualizado = (affectedRows > 0);

} catch (SQLException e) {

e.printStackTrace();

} finally {

DatabaseUtil.closeResources(conn, stmt);

}

return actualizado;

}

@Override

public boolean eliminar(int id) {

Connection conn = null;

PreparedStatement stmt = null;

boolean eliminado = false;

try {

conn = DatabaseUtil.getConnection();

String sql = "DELETE FROM productos WHERE id = ?";

stmt = conn.prepareStatement(sql);

stmt.setInt(1, id);

int affectedRows = stmt.executeUpdate();

eliminado = (affectedRows > 0);

} catch (SQLException e) {

e.printStackTrace();

} finally {

DatabaseUtil.closeResources(conn, stmt);

}

return eliminado;

}

}

**Clase Principal**

Finalmente, creemos una clase principal para demostrar cómo usar nuestra implementación:

package com.example;

import com.example.dao.ProductoDAO;

import com.example.dao.ProductoDAOImpl;

import com.example.model.Producto;

import com.example.util.DatabaseUtil;

import java.util.List;

public class Main {

public static void main(String[] args) {

// Inicializar la base de datos

DatabaseUtil.initDatabase();

// Crear instancia del DAO

ProductoDAO productoDAO = new ProductoDAOImpl();

System.out.println("=== DEMOSTRACIÓN DE OPERACIONES CRUD CON JDBC ===");

// CREATE - Crear nuevos productos

System.out.println("\n=== CREATE ===");

Producto nuevoProducto = new Producto("Laptop HP", "Laptop HP Core i5 8GB RAM", 899.99, 10);

int idGenerado = productoDAO.crear(nuevoProducto);

System.out.println("Producto creado con ID: " + idGenerado);

// Crear otro producto

Producto otroProducto = new Producto("Monitor Dell", "Monitor Dell 27 pulgadas 4K", 349.99, 15);

int idOtroProducto = productoDAO.crear(otroProducto);

System.out.println("Producto creado con ID: " + idOtroProducto);

// READ - Leer productos

System.out.println("\n=== READ ===");

// Buscar por ID

Producto productoEncontrado = productoDAO.buscarPorId(idGenerado);

System.out.println("Producto encontrado por ID: " + productoEncontrado);

// Listar todos

System.out.println("\nLista de todos los productos:");

List<Producto> todosLosProductos = productoDAO.listarTodos();

for (Producto p : todosLosProductos) {

System.out.println(p);

}

// UPDATE - Actualizar producto

System.out.println("\n=== UPDATE ===");

productoEncontrado.setPrecio(799.99);

productoEncontrado.setStock(20);

boolean actualizado = productoDAO.actualizar(productoEncontrado);

System.out.println("Producto actualizado: " + actualizado);

// Verificar la actualización

Producto productoActualizado = productoDAO.buscarPorId(idGenerado);

System.out.println("Producto después de actualizar: " + productoActualizado);

// DELETE - Eliminar producto

System.out.println("\n=== DELETE ===");

boolean eliminado = productoDAO.eliminar(idOtroProducto);

System.out.println("Producto eliminado: " + eliminado);

// Verificar la eliminación

System.out.println("\nLista de productos después de eliminar:");

todosLosProductos = productoDAO.listarTodos();

for (Producto p : todosLosProductos) {

System.out.println(p);

}

}

}

**Manejo de Excepciones**

Es importante manejar adecuadamente las excepciones en JDBC. Aquí hay algunas mejores prácticas adicionales:

// Ejemplo de manejo de excepciones más robusto para el método crear

public int crear(Producto producto) {

Connection conn = null;

PreparedStatement stmt = null;

ResultSet rs = null;

int generatedId = -1;

try {

conn = DatabaseUtil.getConnection();

// Iniciar transacción

conn.setAutoCommit(false);

String sql = "INSERT INTO productos (nombre, descripcion, precio, stock) VALUES (?, ?, ?, ?)";

stmt = conn.prepareStatement(sql, Statement.RETURN\_GENERATED\_KEYS);

stmt.setString(1, producto.getNombre());

stmt.setString(2, producto.getDescripcion());

stmt.setDouble(3, producto.getPrecio());

stmt.setInt(4, producto.getStock());

int affectedRows = stmt.executeUpdate();

if (affectedRows > 0) {

rs = stmt.getGeneratedKeys();

if (rs.next()) {

generatedId = rs.getInt(1);

}

}

// Confirmar transacción

conn.commit();

} catch (SQLException e) {

try {

// Revertir transacción en caso de error

if (conn != null) {

conn.rollback();

}

} catch (SQLException ex) {

ex.printStackTrace();

}

e.printStackTrace();

} finally {

try {

// Restaurar autocommit

if (conn != null) {

conn.setAutoCommit(true);

}

} catch (SQLException e) {

e.printStackTrace();

}

// Cerrar ResultSet

if (rs != null) {

try {

rs.close();

} catch (SQLException e) {

e.printStackTrace();

}

}

DatabaseUtil.closeResources(conn, stmt);

}

return generatedId;

}

**Buenas Prácticas**

Para trabajar eficientemente con JDBC, considera estas prácticas:

1. **Usa PreparedStatement**: Protege contra inyección SQL y mejora el rendimiento.
2. **Maneja los recursos correctamente**: Cierra Statement, ResultSet y Connection para evitar fugas de memoria.
3. **Usa try-with-resources** para cerrar automáticamente los recursos:
4. try (Connection conn = DatabaseUtil.getConnection(); PreparedStatement stmt = conn.prepareStatement(sql)) { // Código aquí} catch (SQLException e) { e.printStackTrace();}
5. **Implementa transacciones**: Para operaciones que requieren múltiples consultas atómicas.
6. **Usa connection pooling**: Para aplicaciones en producción, considera usar un pool de conexiones como HikariCP.
7. **Separa la lógica de acceso a datos**: Utiliza el patrón DAO como hemos implementado.

**Ejemplo Completo**

El proyecto completo que hemos desarrollado implementa el patrón DAO (Data Access Object) para separar la lógica de acceso a datos de la lógica de negocio. La estructura es clara y mantenible.

**Recursos Adicionales**

* [Documentación oficial de JDBC](https://docs.oracle.com/javase/8/docs/technotes/guides/jdbc/)
* [Tutorial de H2 Database](https://www.h2database.com/html/tutorial.html)
* [Mejores prácticas para JDBC](https://www.oracle.com/technical-resources/articles/java/jdbcbestpractices.html)