Módulo 2. Entornos de trabajo: framework. XAMPP. SQL

CÓRDOBA CREA FUTURO

Introducción

Introducción

Un entorno de trabajo en programación es el conjunto de herramientas, *software* y configuraciones que un desarrollador usa para escribir, probar y depurar código, como un editor de texto, compiladores, depuradores y sistemas de control de versiones, usualmente agrupados en un **IDE** (entorno de desarrollo integrado) para hacer el proceso más fluido y eficiente, el cual puede ser local, estar en la nube o ser híbrido, según las necesidades del proyecto.

Características principales de un entorno de desarrollo de programación

Los entornos de desarrollo pueden variar según el lenguaje de programación y el tipo de aplicación que se desea construir. Sin embargo, todos comparten algunas características principales.

- **Editor de código**: es una de las partes más importantes. Permite escribir y editar el código fuente con funciones avanzadas como resaltado de sintaxis, autocompletado y corrección de errores.

- **Compilador o intérprete**: se encarga de traducir el código fuente a un lenguaje que la máquina pueda entender y ejecutar.

- **Depurador**: hace más simple la identificación y corrección de errores en el código.

- **Herramientas de prueba**: algunos entornos incluyen opciones para hacer pruebas automáticas y asegurar que el *software* funcione correctamente.

- **Control de versiones**: permite llevar un seguimiento de los cambios en el código, lo que hace más simple el trabajo en equipo.

Tipos de entornos de desarrollo de programación

Existen diferentes tipos de entornos de desarrollo según su funcionalidad y el tipo de programación que soportan. A continuación, se mencionan algunos de los más comunes.

- **Entornos de desarrollo integrados (IDE)**: son plataformas completas que incluyen todas las herramientas necesarias para programar, como Visual Studio Code, NetBeans, Eclipse o JetBrains IntelliJ IDEA.

- **Editores de texto avanzados**: programas como Sublime Text o Atom que, aunque no son IDE completos, ofrecen funciones avanzadas para la escritura de código.

- **Entornos en la nube**: permiten programar sin necesidad de instalar *software* en la computadora, como Replit o GitHub Codespaces.

- **Entornos especializados**: diseñados para lenguajes o aplicaciones específicas, como Android Studio para desarrollo de aplicaciones móviles en Android.

Ventajas de usar un entorno de desarrollo adecuado

Trabajar con un entorno de desarrollo de programación adecuado trae numerosos beneficios; estos se describen en los próximos párrafos.

- **Mayor productividad**: al contar con herramientas que automatizan tareas repetitivas y facilitan la depuración, se reduce el tiempo de desarrollo.
- **Mejor organización del código**: facilita la estructuración de proyectos y la colaboración entre desarrolladores.
- **Reducción de errores**: gracias a las funciones de corrección y depuración, se reducen los errores en el código.
- **Compatibilidad y escalabilidad**: algunos entornos permiten el desarrollo en múltiples lenguajes y la integración con otras herramientas.

¿Cómo elegir el mejor entorno de desarrollo de programación?

Para seleccionar el mejor entorno de desarrollo, es importante considerar algunos aspectos.

- **El lenguaje de programación**: no todos los entornos son compatibles con todos los lenguajes. Es clave elegir uno que se adapte al lenguaje que se va a usar.

- **La facilidad de uso**: algunos entornos son más intuitivos que otros. Para principiantes, puede ser recomendable optar por un IDE con una interfaz sencilla.

- **La compatibilidad con herramientas externas**: si se requiere integrar bases de datos, librerías o *frameworks*, es importante que el entorno lo permita.

- **El rendimiento y la estabilidad**: un buen entorno debe ser rápido y estable para evitar interrupciones en el desarrollo

IDE vs. *framework*

Saber qué es un entorno de desarrollo de programación es fundamental para cualquier persona que esté interesada en la programación, dado que facilita el proceso de desarrollo de *software* y mejora la calidad del código. Dependiendo de las necesidades de cada programador, existen diversas opciones para elegir la mejor herramienta. Si te interesa aprender más sobre tecnología e informática, te invitamos a explorar los cursos del Instituto IEA y descubrir las oportunidades que ofrecemos en este apasionante campo.

Un **IDE** (entorno de desarrollo integrado) es una aplicación que te da el espacio de trabajo (editor, compilador, depurador) para escribir y gestionar código (por ejemplo, VS Code, Eclipse). Un *framework* es un esqueleto de código predefinido (bibliotecas, patrones) que te da una estructura y herramientas para construir aplicaciones específicas más rápido, lo que evita «reinventar la rueda» (por ejemplo, Django para Python, Bootstrap para web). En resumen, puedes usar un IDE para escribir el código, y dentro de ese IDE puedes aplicar un *framework* para estructurar tu proyecto.

Django es un *framework* web gratuito y de código abierto para Python que permite crear aplicaciones web de manera rápida, maneja bases de datos (ORM), facilita desarrollo *backend*, etcétera.

Bootstrap, por su parte, es un *framework* de código abierto para desarrollo *front-end*, popular para construir sitios web responsivos y modernos rápidamente; proporciona componentes predefinidos (botones, menús, etcétera) y un sistema de cuadrícula adaptable que funciona con HTML, CSS y JavaScript.

CONTINUAR

1. ¿Qué IDE se recomienda usar?

1. ¿Qué IDE se recomienda usar?

Hasta el momento, hemos estado usando una IDE *online* (https://www.online-java.com/), y lo seguiremos haciendo, dado que es de mucha utilidad pedagógica porque permite mostrar, escribir, modificar, ejecutar código fuente Java, etcétera. La ventaja del uso de estas plataformas web es que el alumno no necesita instalar nada en su computadora. La IDE *online*, por su docilidad, puede ser accedida con un navegador web desde una computadora, *tablet*, *notebook* o dispositivo móvil (*smartphone*). Esta herramienta *online*, como se mencionó, se usa aquí con fines pedagógicos, pero para uso profesional se recomienda una IDE que no sea web, es decir, será necesario descargarla en su equipo y aprender a usarla. Si bien todas las IDE tienen un parecido en su uso, gozan de muchas ventajas para desarrollo profesional, lo que permite trabajar colaborativamente en grandes desarrollos con otros profesionales que desarrollan *software*; el uso de un *framework* permite el uso de ciertas librerías que se

pueden descargar y usar. En la modalidad de uso *online*, al no ser los administradores, solo usamos la parte que está permitida o habilitada a usar.

Al existir una variedad de IDE, su elección dependerá del entorno de trabajo que más se use. Por ejemplo, en las universidades nacionales es común ver instalados IDE, tales como Netbeans o Eclipse, pero en otras pueden usar Visual Studio Code (VS Code), lo que demuestra que la elección es una opción. Por esta diversidad, es recomendable interiorizarse de las existentes en todo momento, dado que no todas las empresas usan una en particular. A continuación, se describen algunas, pero se debe considerar por lo mencionado que no se recomienda alguna en particular, tan solo se exponen como una guía.

NetBeans

- **Pros**: también de código abierto, fácil de usar, buen soporte para Java y desarrollo multiplataforma.
- Ideal para principiantes y proyectos que necesiten una interfaz intuitiva.

Figura 1: Sitio web de NetBeans

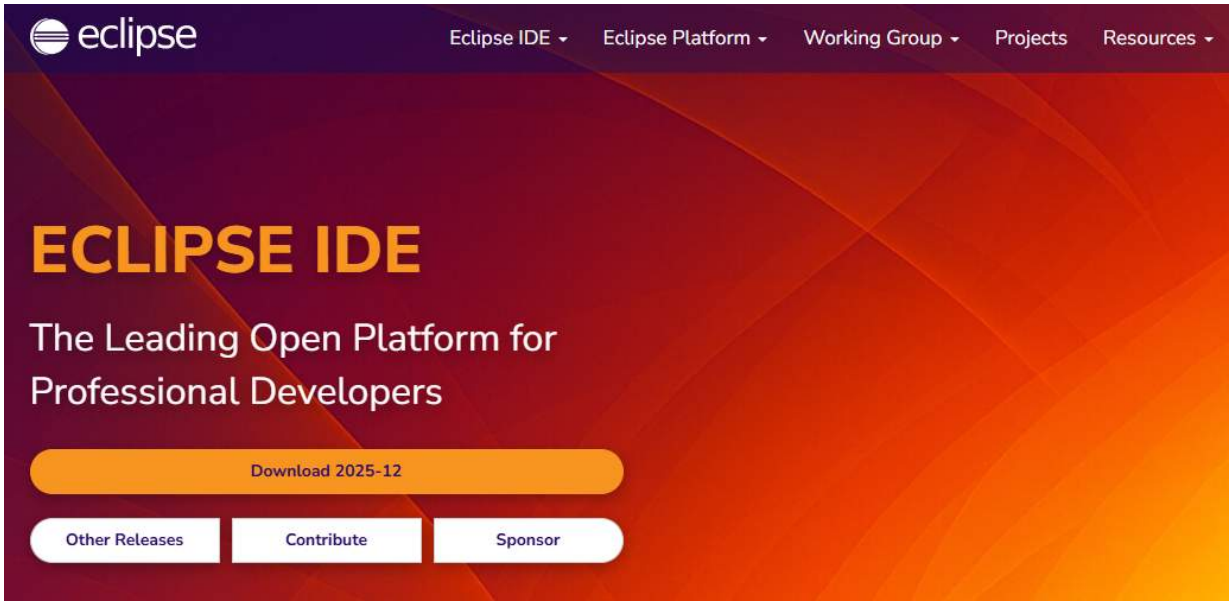
Fuente: captura de pantalla de NetBeans (https://netbeans.apache.org/).

Eclipse

- **Pros**: muy popular, código abierto, gran comunidad y muy extendido.

- Ideal para una alternativa robusta y versátil. Muy usado en entornos académicos y profesionales.

Figura 2: Sitio web de Eclipse

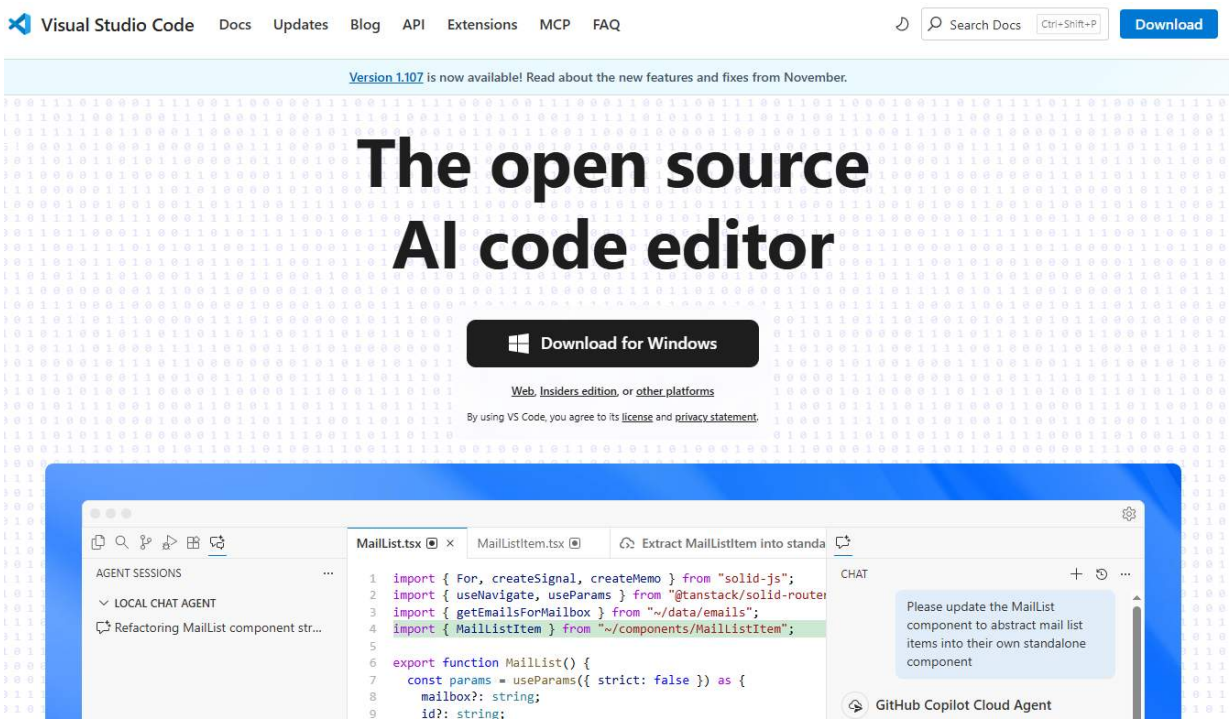


Fuente: captura de pantalla de Eclipse (https://eclipseide.org/).

Visual Studio Code (VS Code con extensiones)

- **Pros**: ligero, muy personalizable con extensiones (Java Extension Pack), versátil para otros lenguajes.

- Ideal para desarrolladores que quieren un editor potente y ligero para múltiples tecnologías, no solo Java.



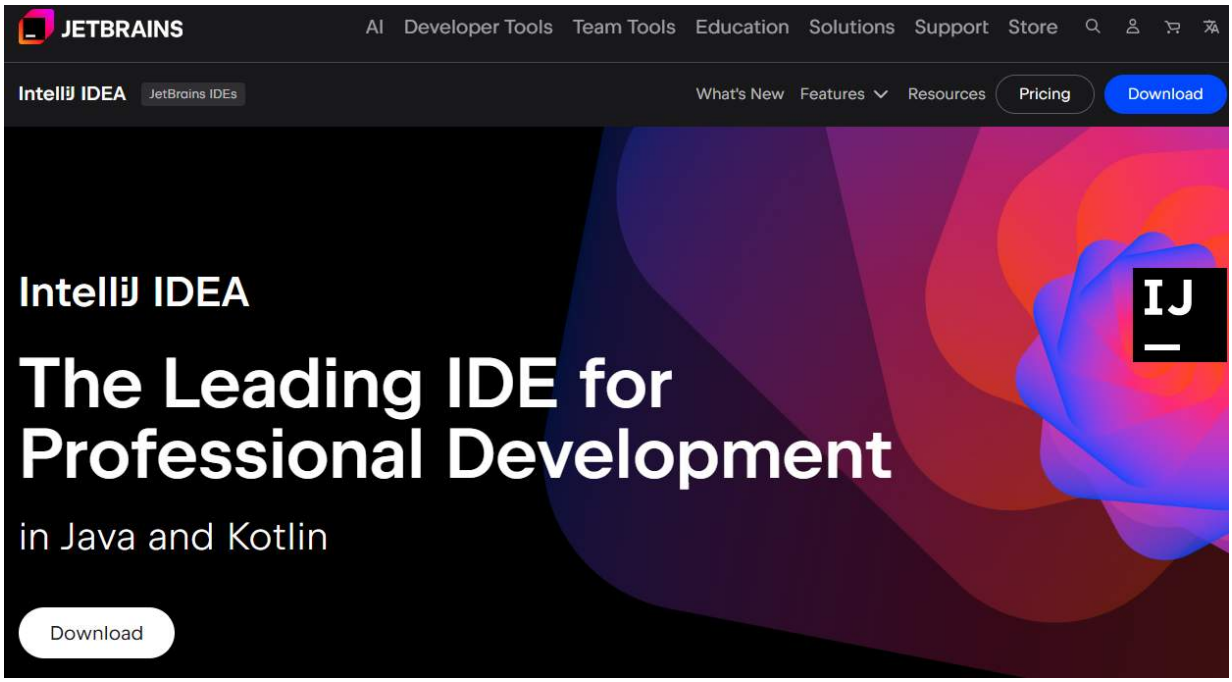
Fuente: captura de pantalla de Visual Studio Code (https://code.visualstudio.com/).

IntelliJ IDEA (JetBrains)

- **Pros**: muchos lo consideran el mejor. Es muy productivo, presenta gran integración con *frameworks* como Spring; depuración fluida y excelentes herramientas.

- Ideal para desarrollo profesional, proyectos grandes y modernos.

Figura 4: Sitio web de JetBrains



Fuente: captura de pantalla de JetBrains (https://www.jetbrains.com/idea/).

CONTINUAR

2. Base de datos relacional

2. Base de datos relacional

Una base de datos relacional (RDB) es un sistema que organiza la información en tablas (como hojas de cálculo) compuestas por filas (registros) y columnas (atributos), lo que permite establecer vínculos o relaciones entre ellas mediante claves para consultar datos interconectados de manera eficiente usando el lenguaje SQL. Su principal fortaleza es estructurar datos complejos, lo que hace más simple su gestión, recuperación y análisis.

Figura 5: Datos de formulario que se guardan en una base de datos

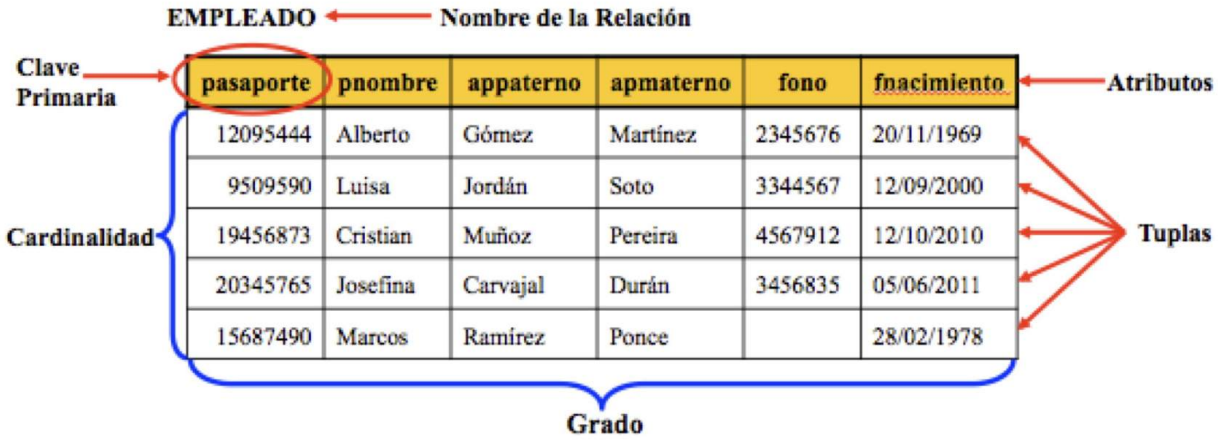
Fuente: elaboración propia.

Componentes principales

- **Tablas**: contenedores para tipos específicos de datos (como, por ejemplo, clientes, productos).
- **Filas (registros)**: cada fila representa una instancia única de la entidad (como, por ejemplo, un cliente específico).
- **Columnas (atributos)**: definen qué información se guarda para cada registro (como, por ejemplo, nombre, dirección, ID).

- **Claves (*keys*)**: identificadores únicos (clave primaria) y valores que conectan tablas (claves foráneas) para crear las relaciones.

Tabla 1: Tabla de una base de datos relacional



pasaporte	pnombre	appaterno	apmaterno	fono	fnacimiento
12095444	Alberto	Gómez	Martínez	2345676	20/11/1969
9509590	Luisa	Jordán	Soto	3344567	12/09/2000
19456873	Cristian	Muñoz	Pereira	4567912	12/10/2010
20345765	Josefina	Carvajal	Durán	3456835	05/06/2011
15687490	Marcos	Ramírez	Ponce		28/02/1978

Fuente: elaboración propia.

¿Cómo funciona?

- **Estructura**: los datos se dividen en tablas lógicas, lo que evita la redundancia y permite una mejor organización.
- **Relaciones**: se establecen conexiones entre tablas (por ejemplo, un "Cliente" puede tener varias "Órdenes")

mediante valores compartidos, como un ID de cliente.

- **Consultas (SQL)**: se usa SQL para pedir a la base de datos que recupere, modifique o inserte datos, incluso combinando información de múltiples tablas relacionadas.

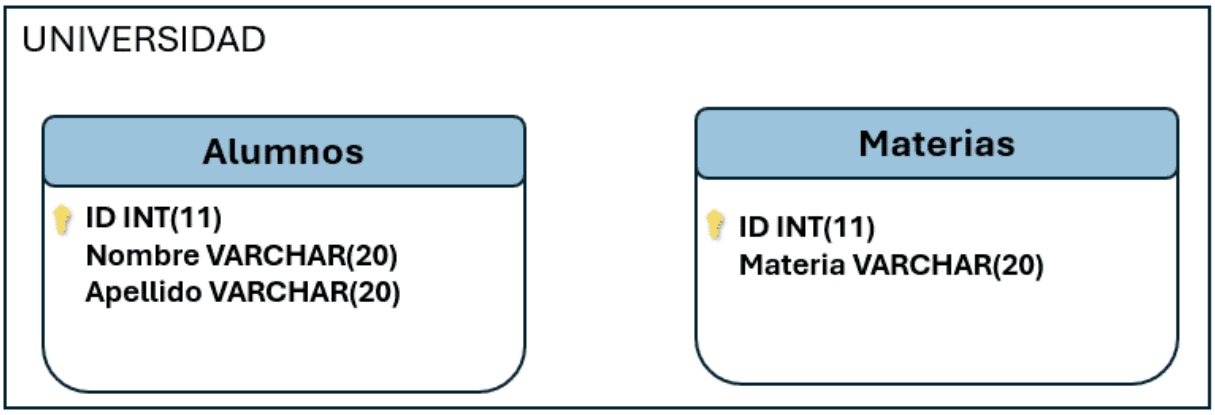
Ejemplos de uso

- Sistemas de **ERP** (planificación de recursos empresariales) como SAP.
- Aplicaciones de **CRM** (gestión de relaciones con clientes) como Salesforce.
- Plataformas de **comercio electrónico** para gestionar productos, clientes y pedidos.
- Sistemas financieros y bancarios que requieren alta integridad de datos.

Ejemplo

Supongamos que tenemos una base de datos llamada universidad que contiene dos tablas, alumnos y materias.

Figura 6: Tablas de una base de datos



Fuente: elaboración propia.

Supongamos que luego de las inscripciones de alumnos y materias que hizo la universidad, tenemos cargados los siguientes datos en ambas tablas.

Tabla 2: Tablas con datos cargados

Tabla Alumnos

ID	Nombe	Apellido
1	Walter	Agüero
2	Esteban	Agüero
3	María del Carme	Godoy

Tabla Materias

ID	Materia
1	Inteligenci Artificial
2	Programación Orientada a Objetos

Fuente: elaboración propia.

En esta simulación, el siguiente proceso es la inscripción de los alumnos que cursarán materias.

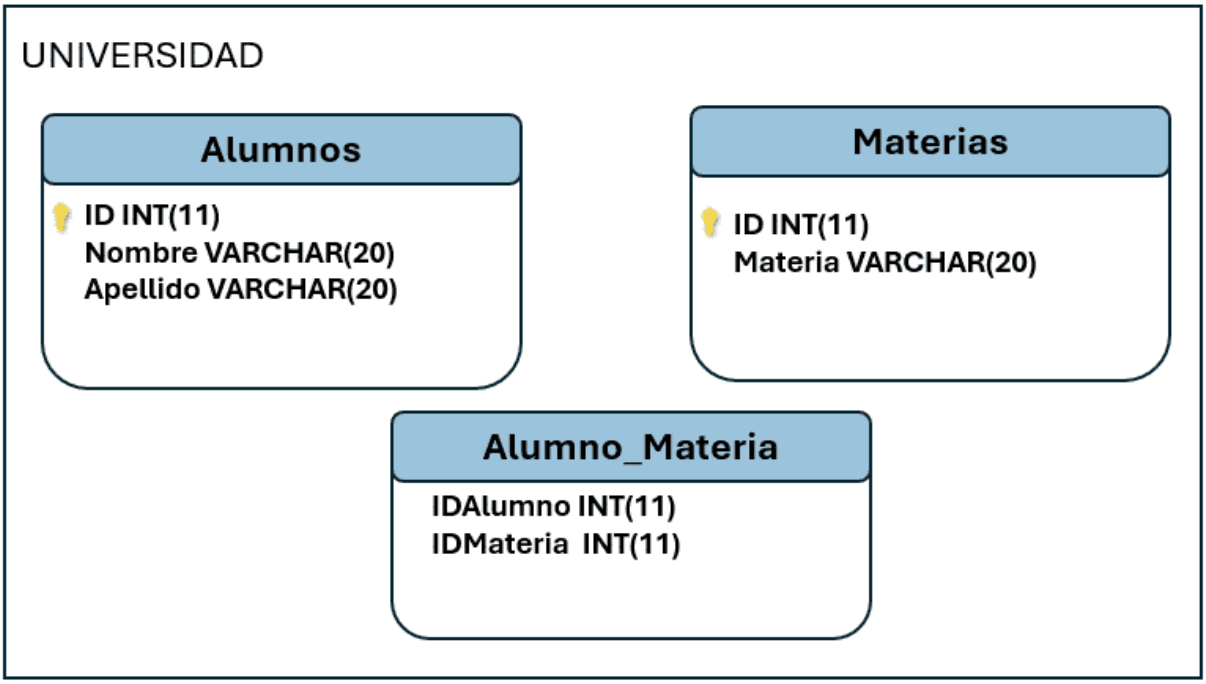
Tabla 3: Tabla que relaciona a las tablas alumnos con materias

Alumno	Materia
Walter Agüero (ID 1)	Inteligencia artificial (**ID** 1)
Walter Agüero(ID 1)	Programación orientada a objetos (**ID** 2)
Esteban Agüero (ID 2)	Programación orientada a objetos (**ID** 2)
María del Carmen Godoy (ID 3)	Inteligencia artificial (**ID** 1)

Fuente: elaboración propia.

La tabla relacional que vincula alumno con materia quedará especificada con sus correspondientes ID. La estructura total de la nueva tabla y su carga queda como se muestra en la siguiente imagen donde la nueva tabla se incorpora a la base de datos y su estructura se puede observar en la figura a continuación.

Figura 7: Tabla que vincula alumno y materia



Fuente: elaboración propia.

Los datos reflejados en la tabla quedan relacionados en la tabla nueva Alumno_Materia.

Figura 8: Datos cargados en la tabla que relaciona alumnos con materia

Tabla Alumnos

ID	Nombe	Apellido
1	Walter	Agüero
2	Esteban	Agüero
3	María del Carme	Godoy

Tabla Materias

ID	Materia
1	Inteligenci Artificial
2	Programación Orientada a Objetos

IDAlumno	IDMateria
1	1
1	2
2	2
3	1

Tabla Alumno_Materia

Fuente: elaboración propia.

De esta manera, se ha mostrado cómo quedan impactadas las tablas en la base de datos cuando se simula su carga. Por supuesto que un usuario cargó estos datos por medio de un sistema. La aplicación, seguramente, además podrá visualizar consultas del siguiente tipo: qué alumnos están inscritos en una materia o un alumno en qué materias se inscribió. Estas consultas se hacen a las tablas de la base de datos por medio de consultas SQL que se verán a continuación.

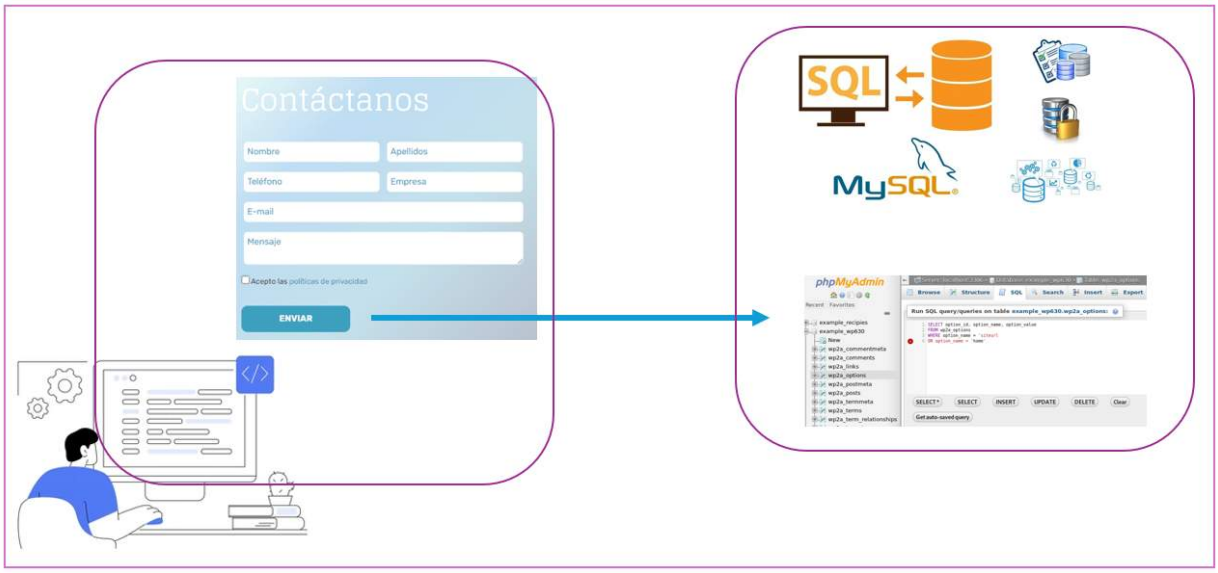
CONTINUAR

3. SQL

3. SQL

SQL (Structured Query Language) es un lenguaje estandarizado para gestionar y manipular bases de datos relacionales, lo cual permite consultar, insertar, actualizar y eliminar datos, lo que lo hace esencial para empresas que manejan grandes volúmenes de información, desde bancos hasta servicios *online*, y funciona con sistemas como MySQL, Oracle y SQL Server.

Figura 9: Los datos del formulario se guardan en la base de datos



Fuente: elaboración propia.

El programa Java, así como cualquier otro programa, puede comunicarse con la base de datos usando las credenciales de usuario, contraseña y dirección IP del lugar en que esta se encuentra. Es decir, una aplicación hecha con una IDE en nuestro equipo local (computadora, *notebook*, *tablet*, celular, etcétera) puede trabajar con una base de datos localizándola por medio de su número IP.

Aprender a usar SQL es un desafío necesario y, por tal motivo, se presenta brevemente una introducción, pero se dejan referencias para aprender ciertos aspectos, como, por ejemplo, el tutorial *online* de W3shools (https://www.w3schools.com/sql/).

Por ejemplo, tomando la tabla Alumnos para consultar por todos los alumnos, la sentencia a usar sería la siguiente:

```
SELECT * FROM Alumnos;
```

El resultado se puede observar en la tabla a continuación.

Tabla 4: Resultado luego de ejecutar la sentencia SQL

ID	Nombe	Apellido
1	Walter	Agüero
2	Esteban	Agüero
3	María del Carme	Godoy

Fuente: elaboración propia.

Otro ejemplo podría ser listar todas las personas que sean de apellido Agüero (es probable que deba escribirse Agüero sin diéresis [“Aguero”]).

```
SELECT * FROM Alumnos

WHERE Apellido='Agüero”;
```

El resultado se puede observar en la tabla a continuación.

Tabla 5: Resultado luego de ejecutar la sentencia SQL

ID	Nombe	Apellido
1	Walter	Agüero
2	Esteban	Agüero

Fuente: elaboración propia.

CONTINUAR

4. XAMPP

4. XAMPP

XAMPP es un entorno de desarrollo web gratuito y de código abierto que permite crear y probar sitios web en tu propia computadora (servidor local), incluyendo Apache (servidor web), MariaDB/MySQL (base de datos), PHP y Perl (lenguajes de *scripting*). Es multiplataforma (funciona en Windows, Linux y macOS), fácil de instalar y es ideal para desarrolladores principiantes y experimentados, lo que hace más simple el trabajo sin necesidad de conexión a internet antes de publicar un proyecto en un servidor real.

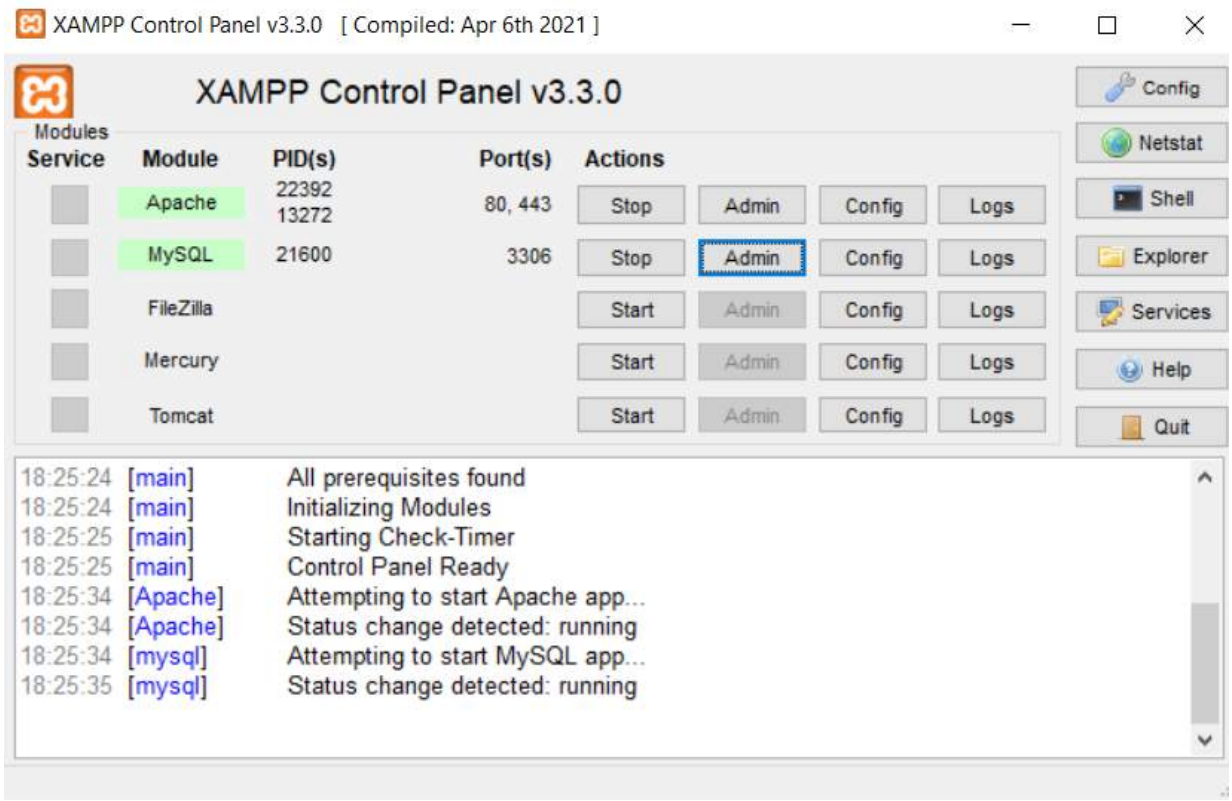
Es un acrónimo que representa sus componentes principales.

- **X**: cualquiera de los sistemas operativos (Windows, Linux, macOS).
- **A**: Apache (servidor web).

- **M**: MariaDB o MySQL (bases de datos).
- **P**: PHP (lenguaje de programación).
- **P**: Perl (otro lenguaje de programación).

Usando el conector adecuado se puede trabajar con el lenguaje Java para que los datos que se ingresen desde la aplicación impacten modificando, eliminando o guardándose en la base de datos.

Figura 10: XAMPP ejecutándose



Fuente: elaboración propia.

Cuando descarga y ejecuta XAMPP, muestra el mismo ambiente de trabajo, en el cual, si se ejecuta, presiona el botón “Start” en la fila de “Apache” y “MySQL”. Así, deja corriendo los servicios que se necesitan para que, desde la aplicación, los datos ingresados se guarden en la tabla de la base de datos.

Para poder crear una base de datos, agregar una tabla, dar de alto a nuevos registros, etcétera, se debe presionar el botón “Admin” de la fila MySQL; si todo está bien, se abrirá su navegador web mostrando algo parecido a la figura que se presenta a continuación.

Figura 11: Captura de lo mostrado por phpMyAdmin

Fuente: elaboración propia.

CONTINUAR

5. Integrando lo visto

5. Integrando lo visto

En la siguiente referencia, se muestran los pasos para grabar los datos en la base de datos que previamente se generó desde una aplicación hecha por la IDE seleccionada y teniendo XAMPP con los módulos activos mostrados más arriba.

En los próximos párrafos, se presentan algunos de estos pasos que seguir.

Descarga de herramientas

Descargar el conector que permite trabajar con la base de datos desde la aplicación. Se supone que XAMPP ya lo ha descargado anteriormente o que puede hacerlo ahora. Como se explicó más arriba, es necesario ejecutar activando el módulo de Apache y el de MySQL que ejecuta phpmyadmin, como se puede observar en la figura a continuación.

En XAMPP, Apache es el servidor web HTTP de código abierto que permite alojar y ejecutar sitios web y aplicaciones dinámicas (como las de PHP) directamente en tu computadora local, lo cual funciona como un servidor simulado para desarrollo, manejando las solicitudes de los navegadores y entregando el contenido web (HTML, CSS, imágenes) a tu máquina a través de http://localhost.

phpMyAdmin es una herramienta web para administrar bases de datos MySQL (o MariaDB) de forma gráfica, lo que te permite crear, modificar y gestionar tablas y datos sin comandos, todo integrado para facilitar el desarrollo local de aplicaciones web con PH.

Figura 12: Ejecución de phpmyadmin

Fuente: elaboración propia.

Descargar conector

Descargar desde el sitio https://dev.mysql.com/downloads/connector/j/ y luego agregar mysql-connector-java-8.0.33.jar al *classpath* (indicando dentro del programa el paso de la carpeta en la que está instalada).

Conectar con la aplicación

Agregar las siguientes tres líneas al código fuente Java de nuestra aplicación:

```
private static final String URL = "jdbc:mysql://localhost:3306/baseDatos";

private static final String USER = "root"; // Usuario por defecto XAMPP

private static final String PASSWORD = ""; // Contraseña por defecto XAMPP (vacía)
```

A continuación, se explica cada una de las sentencias mostradas.

Conector: indica el conector que se usa, jdbc:mysql.

Dirección IP o URL en la que se encuentra la base de datos y el puerto que se usa por defecto. Al final se pone el nombre de la base que se generó usando la aplicación phpmyadmin:

```
private static final String URL = "jdbc:mysql://localhost:3306/baseDatos";
```

Credencial usuario: poner el nombre de usuario para acceder a la base de datos (lo generó cuando instaló XAMPP o base de datos; si no, por defecto, es “root”).

```
private static final String USER = "root"; // Usuario por defecto XAMPP
```

Credencial *password*: si no lo generó cuando instaló XAMPP, por defecto, es vacío “”.

```
private static final String PASSWORD = ""; // Contraseña por defecto XAMPP (vacía)
```

Seguidamente, se deja un ejemplo completo de conexión a una base de datos agregando este código en nuestra aplicación Java.

Para ello, generamos el siguiente código fuente, y a esa clase que contiene la configuración para conectarse a la base de datos le ponemos por nombre DatabaseConfig.java:

```
package com.ejemplo.db;



public class DatabaseConfig {

  // Configuración para XAMPP (valores por defecto)

        private static final String URL =
"jdbc:mysql://localhost:3306/ejemplo_java";

  private static final String USER = "root"; // Usuario por
defecto XAMPP

    private static final String PASSWORD = ""; //
Contraseña por defecto XAMPP (vacía)
```

```
    public static String getUrl() {
        return URL;
    }

    public static String getUser() {
        return USER;
    }

    public static String getPassword() {
        return PASSWORD;
    }
}
```

El siguiente paso es usar esta configuración para conectarse a la base de datos. Esta clase se llamará DatabaseConnection.java:

```
package com.ejemplo.db;



import java.sql.Connection;

import java.sql.DriverManager;

import java.sql.SQLException;



public class DatabaseConnection {

  private static Connection connection = null;

  // Constructor privado para patrón Singleton

  private DatabaseConnection() {}

    public static Connection getConnection() throws
SQLException {
```

```
        if (connection == null || connection.isClosed()) {

            try {

                    // Cargar el driver (opcional en versiones
recientes)

                Class.forName("com.mysql.cj.jdbc.Driver");

                // Establecer conexión

                connection = DriverManager.getConnection(

                    DatabaseConfig.getUrl(),

                    DatabaseConfig.getUser(),

                    DatabaseConfig.getPassword()

                );



                    System.out.println("✅ Conexión establecida
con MySQL");
```

```
        } catch (ClassNotFoundException e) {
            System.err.println("❌ Error: Driver MySQL no encontrado");
            e.printStackTrace();
            throw new SQLException("Driver no encontrado", e);
        } catch (SQLException e) {
            System.err.println("❌ Error al conectar a la base de datos:");
            System.err.println("URL: " + DatabaseConfig.getUrl());
            System.err.println("Usuario: " + DatabaseConfig.getUser());
            e.printStackTrace();
            throw e;
```

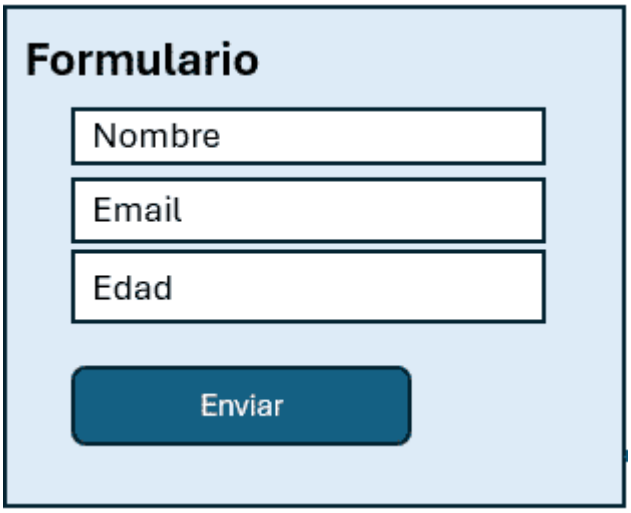
```
            }

        }

        return connection;

    }



    public static void closeConnection() {

        if (connection != null) {

            try {

                connection.close();

                System.out.println("🔌 Conexión cerrada");

            } catch (SQLException e) {

                        System.err.println("❌ Error al cerrar la
conexión");

                e.printStackTrace();
```

```
        }
    }
}

// Método para verificar conexión
public static boolean testConnection() {
    try (Connection conn = getConnection()) {
        return conn != null && !conn.isClosed();
    } catch (SQLException e) {
        return false;
    }
}
}
```

Modelo/entidad

Imaginemos que nuestra aplicación tiene que guardar los datos cargados desde el formulario de la figura que se observa a continuación.

Figura 13: Simulación de formulario armado en la aplicación JAVA que se desarrolla



Fuente: elaboración propia.

El código fuente usando sentencias embebidas de SQL en JAVA que permite tomar los datos cargados en el formulario se muestra seguidamente. A este archivo lo llamaremos Usuario.java:

```
package com.ejemplo.model;
```

```java
import java.sql.Timestamp;


public class Usuario {

  private int id;

  private String nombre;

  private String email;

  private int edad;

  private Timestamp fechaRegistro;


  // Constructores

  public Usuario() {}
```

```
    public Usuario(String nombre, String email, int edad) {
        this.nombre = nombre;
        this.email = email;
        this.edad = edad;
    }

    public Usuario(int id, String nombre, String email, int edad, Timestamp fechaRegistro) {
        this.id = id;
        this.nombre = nombre;
        this.email = email;
        this.edad = edad;
        this.fechaRegistro = fechaRegistro;
```

```
}


// 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 getEmail() { return email; }

  public void setEmail(String email) { this.email =
email; }
```

```
  public int getEdad() { return edad; }

  public void setEdad(int edad) { this.edad = edad; }



    public Timestamp getFechaRegistro() { return
fechaRegistro; }

        public void setFechaRegistro(Timestamp
fechaRegistro) {

    this.fechaRegistro = fechaRegistro;

  }



  @Override

  public String toString() {

              return String.format("Usuario [id=%d,
nombre=%s, email=%s, edad=%d,
fechaRegistro=%s]",
```

```
                id, nombre, email, edad, fechaRegistro);

        }

    }
```

DAO (data *access object*)

En el siguiente código fuente de Java, se toman los datos del paso anterior y se insertan en la base de datos que en este ejemplo aún no se ha generado. A este archivo lo llamaremos UsuarioDAO.java:

```
package com.ejemplo.dao;



import com.ejemplo.db.DatabaseConnection;

import com.ejemplo.model.Usuario;

import java.sql.*;

import java.util.ArrayList;

import java.util.List;
```

```
public class UsuarioDAO {

 // CREATE - Insertar nuevo usuario

     public int insertarUsuario(Usuario usuario) throws
SQLException {

         String sql = "INSERT INTO usuarios (nombre, email,
edad) VALUES (?, ?, ?)";



                                 try (Connection conn =
DatabaseConnection.getConnection();

                                     PreparedStatement pstmt =
conn.prepareStatement(sql,
Statement.RETURN_GENERATED_KEYS)) {



         pstmt.setString(1, usuario.getNombre());

         pstmt.setString(2, usuario.getEmail());
```

```
        pstmt.setInt(3, usuario.getEdad());



        int filasAfectadas = pstmt.executeUpdate();

        // Obtener el ID generado

                                try (ResultSet generatedKeys =
pstmt.getGeneratedKeys()) {

            if (generatedKeys.next()) {

                usuario.setId(generatedKeys.getInt(1));

            }

        }



                    System.out.println(" ✅ Usuario insertado: " +
usuario.getNombre());

        return filasAfectadas;
```

```
    } catch (SQLException e) {
            System.err.println("❌ Error al insertar usuario: " +
e.getMessage());
        throw e;
    }
 }
 // READ - Obtener todos los usuarios
    public List<Usuario> obtenerTodosUsuarios() throws
SQLException {
    List<Usuario> usuarios = new ArrayList<>();
    String sql = "SELECT * FROM usuarios ORDER BY id";
                        try (Connection conn =
DatabaseConnection.getConnection();
```

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

            ResultSet rs = stmt.executeQuery(sql)) {



        while (rs.next()) {

            Usuario usuario = mapearUsuario(rs);

            usuarios.add(usuario);

        }



                System.out.println(" 📋  Usuarios  obtenidos:  " +
usuarios.size());

        return usuarios;



    } catch (SQLException e) {
```

```
            System.err.println("❌ Error al obtener usuarios: " +
e.getMessage());

            throw e;

        }

    }

    // READ - Obtener usuario por ID

    public Usuario obtenerUsuarioPorId(int id) throws
SQLException {

        String sql = "SELECT * FROM usuarios WHERE id = ?";



                                try (Connection conn =
DatabaseConnection.getConnection();

                                    PreparedStatement pstmt =
conn.prepareStatement(sql)) {
```

```
            pstmt.setInt(1, id);



            try (ResultSet rs = pstmt.executeQuery()) {

                if (rs.next()) {

                    System.out.println("🔍 Usuario encontrado: ID " +
id);

                    return mapearUsuario(rs);

                } else {

                    System.out.println("⚠️ Usuario no encontrado: ID
" + id);

                    return null;

                }

            }
```

```
    } catch (SQLException e) {

        System.err.println("❌ Error al obtener usuario por ID:
" + e.getMessage());

        throw e;

    }

 }

 // READ - Buscar usuarios por nombre

   public  List<Usuario>  buscarUsuariosPorNombre(String
nombreBusqueda) throws SQLException {

    List<Usuario> usuarios = new ArrayList<>();

     String sql = "SELECT * FROM usuarios WHERE nombre
LIKE ?";



                              try     (Connection     conn     =
DatabaseConnection.getConnection();
```

```
                                PreparedStatement pstmt =
conn.prepareStatement(sql)) {



        pstmt.setString(1, "%" + nombreBusqueda + "%");



        try (ResultSet rs = pstmt.executeQuery()) {

           while (rs.next()) {

              usuarios.add(mapearUsuario(rs));

           }

        }



         System.out.println("🔍 Usuarios encontrados con '" +
nombreBusqueda + "': " + usuarios.size());

        return usuarios;
```

```
    } catch (SQLException e) {

            System.err.println("❌ Error al buscar usuarios: " +
e.getMessage());

        throw e;

    }

 }

 // UPDATE - Actualizar usuario

    public int actualizarUsuario(Usuario usuario) throws
SQLException {

     String sql = "UPDATE usuarios SET nombre = ?, email =
?, edad = ? WHERE id = ?";



                                try (Connection conn =
DatabaseConnection.getConnection();
```

```
                                PreparedStatement pstmt =
conn.prepareStatement(sql)) {



        pstmt.setString(1, usuario.getNombre());

        pstmt.setString(2, usuario.getEmail());

        pstmt.setInt(3, usuario.getEdad());

        pstmt.setInt(4, usuario.getId());



        int filasAfectadas = pstmt.executeUpdate();



        if (filasAfectadas > 0) {

            System.out.println("✏️ Usuario actualizado: ID " +
usuario.getId());

        } else {
```

```
                System.out.println("⚠️ No se actualizó ningún
usuario");

            }



            return filasAfectadas;



        } catch (SQLException e) {

            System.err.println("❌ Error al actualizar usuario: " +
e.getMessage());

            throw e;

        }

    }

    // DELETE - Eliminar usuario

    public int eliminarUsuario(int id) throws SQLException {
```

```
        String sql = "DELETE FROM usuarios WHERE id = ?";

                                try (Connection conn =
DatabaseConnection.getConnection();

                                    PreparedStatement pstmt =
conn.prepareStatement(sql)) {

            pstmt.setInt(1, id);

            int filasAfectadas = pstmt.executeUpdate();

            if (filasAfectadas > 0) {
                System.out.println("🗑️ Usuario eliminado: ID " + id);
            } else {
```

```
                System.out.println("⚠️ No se eliminó ningún usuario");

            }



            return filasAfectadas;



        } catch (SQLException e) {

            System.err.println("❌ Error al eliminar usuario: " + e.getMessage());

            throw e;

        }

    }

    // Método auxiliar para mapear ResultSet a Usuario

    private Usuario mapearUsuario(ResultSet rs) throws SQLException {
```

```
        return new Usuario(
            rs.getInt("id"),
            rs.getString("nombre"),
            rs.getString("email"),
            rs.getInt("edad"),
            rs.getTimestamp("fecha_registro")
        );
    }

    // Método para obtener estadísticas
    public void mostrarEstadisticas() throws SQLException {
        String sql = "SELECT COUNT(*) as total, AVG(edad) as
promedio_edad FROM usuarios";
```

```
                try (Connection conn =
DatabaseConnection.getConnection();

        Statement stmt = conn.createStatement();

        ResultSet rs = stmt.executeQuery(sql)) {



       if (rs.next()) {

          int total = rs.getInt("total");

                                double promedioEdad =
rs.getDouble("promedio_edad");



          System.out.println("\n📊 ESTADÍSTICAS:");

          System.out.println("Total usuarios: " + total);

                    System.out.println("Edad promedio: " +
String.format("%.2f", promedioEdad));

       }
```

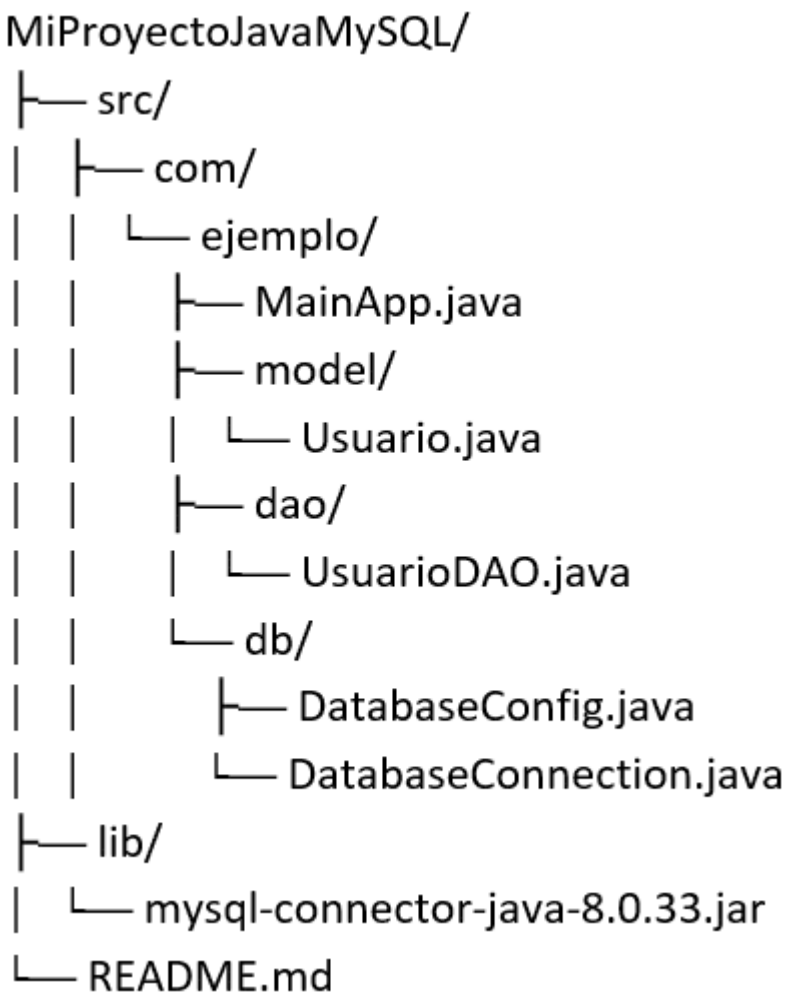
```
        } catch (SQLException e) {

            System.err.println("❌ Error al obtener estadísticas: " +
    e.getMessage());

            throw e;

        }

    }

}
```

Finalmente, la clase que contiene el menú principal se deja en el enlace de *online-java*, dado que es largo (242 líneas de código) y usa los archivos anteriores. Los alumnos pueden copiar y pegar en la aplicación IDE de escritorio que estén usando. A este archivo lo llamaremos **MainApp.java**.

Enlace: https://www.online-java.com/OAn2Ar2vau.

Estructura del proyecto

La estructura que debe tener en su disco duro es la siguiente:



Características del ejemplo

Los puntos que hemos visto en el ejemplo que se ha dejado son los siguientes:

- conexión a MySQL con XAMPP.

- Patrón DAO para separar responsabilidades.
- PreparedStatement para evitar SQL *injection*.
- Manejo adecuado de excepciones.
- Cierre automático de recursos (*try-with-resources*).
- Menú interactivo para probar todas las operaciones CRUD.
- Singleton para la conexión a la base de datos.
- Código limpio y bien documentado.

CONTINUAR

Descarga en PDF