

PRESA. UNIDAD 2. ACCESO A BASES DE DATOS. PARTE 1. TRABAJAR CON BASES DE DATOS RELACIONALES

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

Unidad 2. ACCESO A LAS BASES DE DATOS

Parte 1. Trabajar con bases de datos relacionales

Abelardo Martínez

Año 2023-2024

1. Introducción

Esta unidad proporciona una descripción general de cómo las aplicaciones orientadas a objetos utilizan sistemas de gestión de bases de datos (DBMS) para lograr la persistencia de sus instancias. Los DBMS son aplicaciones especializadas en el almacenamiento de datos estructurados, con la capacidad de almacenarlos y recuperarlos de manera consistente y eficiente, independientemente del número de accesos simultáneos. Cabe señalar, sin embargo, que la gestión de los SGBD requiere conocimientos técnicos bastante específicos que dificultan el acceso de usuarios no especializados a sus datos almacenados.

Para simplificar las tareas de almacenamiento, es posible automatizarlas utilizando lenguajes y herramientas de consulta y administración propios de los DBMS. Sin embargo, estas herramientas no son suficientes para permitir que un usuario neófito las utilice sin tener que realizar un importante esfuerzo de aprendizaje y planificación para traducir sus necesidades en una secuencia eficaz de tareas.

La industria del desarrollo de aplicaciones utiliza bases de datos como herramientas para derivar las complejas tareas de almacenamiento para centrar esfuerzos en las dificultades del dominio donde se desarrollarán las aplicaciones, así como en el acceso a la automatización de las tareas de almacenamiento requeridas para que los usuarios puedan reducir el aprendizaje. curva de las aplicaciones que utilizarán, acercándose a formas más intuitivas según sus conocimientos previos específicos.

Para coordinar los lenguajes de programación con el potencial de los DBMS, la industria ha desarrollado un conjunto de herramientas específicas para conectarse a una base de datos determinada y enviarle la secuencia de tareas que las aplicaciones pueden necesitar durante su ejecución.

1.1. Bases de datos relacionales y no relacionales

A pesar de que durante los primeros años de la historia de la informática, la persistencia de los datos ha pasado, de década en década, por diferentes paradigmas de representación y almacenamiento de datos, la tendencia al cambio se ha ido desvaneciendo con el crecimiento del paradigma relacional. Se trata de una tecnología sencilla pero muy eficiente que ha sabido adaptarse a la mayoría de sistemas de datos que demandaban las empresas y a un coste lo suficientemente asequible como para hacerse con la hegemonía absoluta del mercado desde el último cuarto

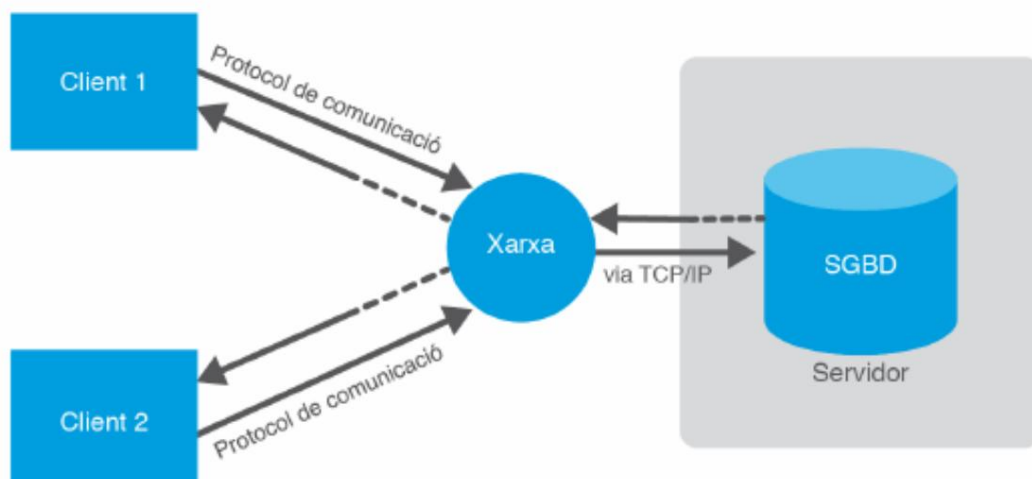
Es cierto que el modelo relacional tiene importantes limitaciones a la hora de representar información poco estructurada, o con estructuras excesivamente dinámicas y complejas, pero a pesar de todo ello, los sistemas gestores de bases de datos relacionales (RDBMS) ofrecen una gran solidez y madurez.

En los últimos años, han surgido DBMS no relacionales para complementar las limitaciones de los sistemas relacionales. Por ejemplo, las bases de datos no relacionales permiten una gran escalabilidad horizontal, por lo que se pueden replicar mucho más fácilmente y a un coste mucho menor. Por otro lado, adolecen de otras limitaciones, como el hecho de que no proporcionan una actualización inmediata de la información. Actualmente se utilizan ampliamente en aplicaciones de redes sociales y almacenamiento de datos históricos.

1.2. Tecnología cliente-servidor

A medida que las teorías de datos relacionales ganaron impulso y las redes se hicieron más populares debido a una mayor eficiencia a precios muy competitivos, se comenzaron a implementar sistemas de gestión de bases de datos basados en tecnología cliente-servidor.

La tecnología cliente-servidor permitió aislar los datos y los programas de acceso específicos del desarrollo de la aplicación. La razón principal de esta división probablemente fue permitir el acceso remoto a los datos desde cualquier computadora conectada a la red. Lo cierto, sin embargo, es que este hecho empujó a los sistemas de bases de datos a desarrollarse de forma aislada y a crear protocolos y lenguajes específicos para poder comunicarse de forma remota con aplicaciones que se ejecutaban en ordenadores externos.

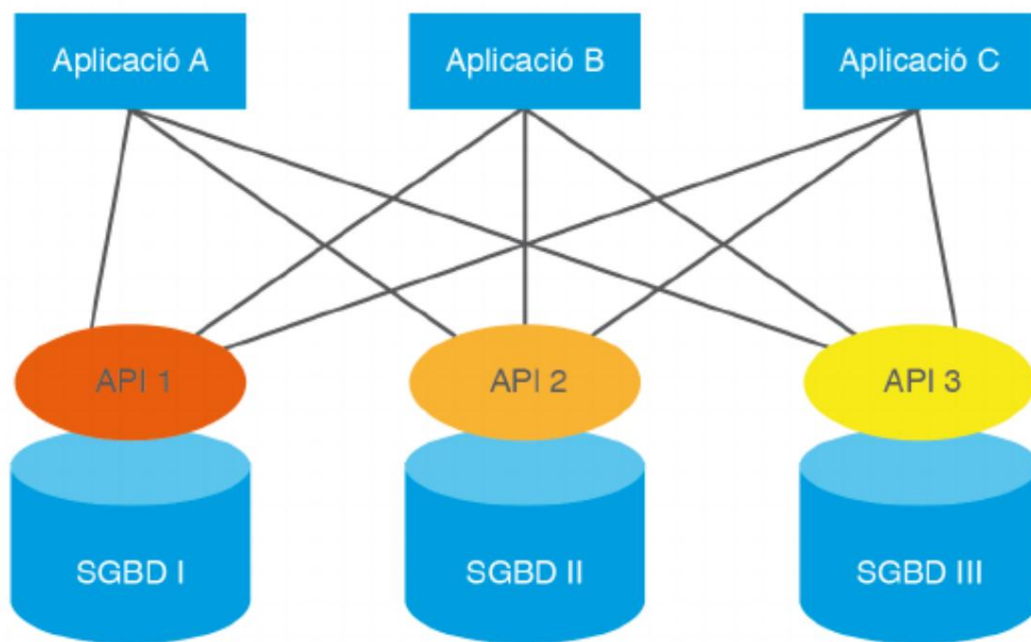


Poco a poco, el software en torno a bases de datos creció espectacularmente intentando responder a un rango máximo de demandas a través de sistemas altamente configurables. Esto es lo que hoy se conoce como middleware o capa de persistencia intermedia. Es decir, el conjunto de aplicaciones, utilidades, bibliotecas, protocolos y lenguajes, ubicados tanto en el lado del servidor como en el del cliente, que permiten la conexión remota a una base de datos para poder configurarla o explotar sus datos.

1.3. La llegada de los estándares

Inicialmente, cada empresa que desarrollaba un DBMS implementó soluciones patentadas específicas para su sistema, pero pronto se dio cuenta de que trabajando juntas podrían sacarle más provecho y progresar mucho más rápido.

Sobre la base de la teoría relacional y algunas implementaciones iniciales de IBM y Oracle, se desarrolló el lenguaje de consulta de datos llamado SQL. Este desafío fue sin duda un gran paso adelante, pero las aplicaciones necesitaban API con funciones que permitieran realizar llamadas desde el lenguaje de desarrollo para enviar consultas realizadas con SQL. Es decir, todavía existía un sistema de conexión propietario, donde cada DBMS tiene su propia conexión y su propia API.



2. Conectores

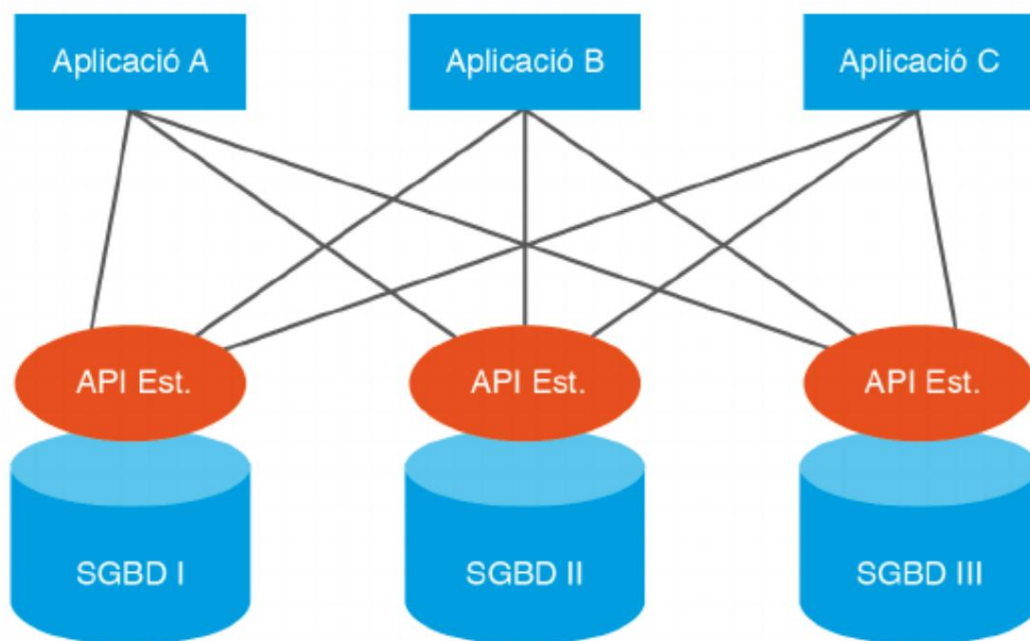
La historia de las bases de datos relacionales ha estado estrechamente ligada a la historia de las aplicaciones de arquitectura distribuida y, en particular, a las arquitecturas cliente-servidor. Hace poco más de un cuarto de siglo, la persistencia de las aplicaciones formaba parte del desarrollo de aplicaciones, que se concebía como un todo monolítico. El almacenamiento y la recuperación de datos se mezclaron y confundieron con la explotación de datos. Las bases de datos a menudo tenían sus propios lenguajes de programación que incorporaban declaraciones y llamadas de acceso a datos específicas.

En realidad, esta situación no representó muchas ventajas ni para las empresas desarrolladoras del SGBD ni para las empresas usuarias. Los primeros descubrieron que mantener el desarrollo de un lenguaje de programación era realmente costoso si no querían quedar obsoletos rápidamente. Las empresas usuarias, en cambio, estaban atadas a un lenguaje de programación que no siempre cumplía con todas sus necesidades. Además, cualquier cambio de sistema de gestión de datos era una tarea imposible, ya que implicaba tener que reprogramar todas las aplicaciones de la empresa. Los lenguajes de desarrollo tuvieron que desacoplarse del DBMS.

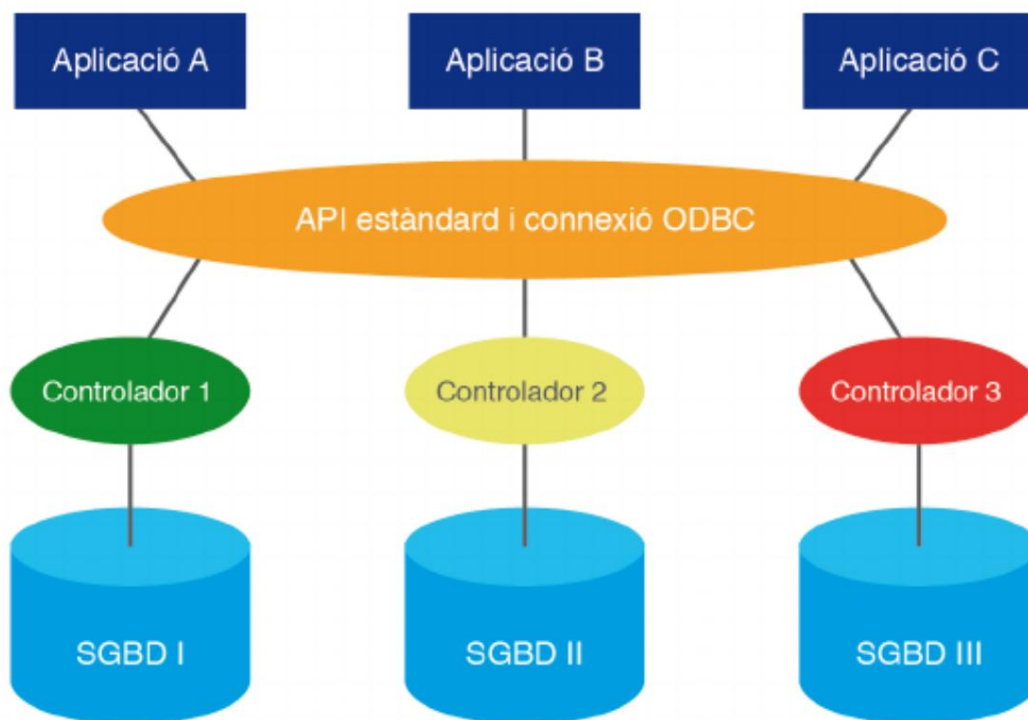
Los sistemas de gestión de bases de datos (DBMS) de diferentes tipos tienen sus propios lenguajes especializados para tratar los datos que almacenan y los programas de aplicación están escritos en lenguajes de programación de propósito general, como Java. Para que estos programas funcionen con DBMS, se necesitan mecanismos que permitan que los programas de aplicación se comuniquen con las bases de datos en estos lenguajes. Estos se implementan en una API y se denominan conectores.

2.1. ODBC

El Grupo SQL Access, del que forman parte prestigiosas empresas del sector como Oracle, Informix, Ingres, DEC, Sun y HP, definió una API universal independientemente del lenguaje de desarrollo y de la base de datos a conectar.



En 1992, Microsoft y Simba implementaron ODBC (Open Data Base Connectivity), una API basada en la definición de SQL Access Group, que se integra en el sistema operativo Windows y permite añadir múltiples conectores a varias bases de datos SQL de una forma muy sencilla y transparente. De esta manera, ya que los conectores son autoinstalables y totalmente configurables desde las mismas herramientas del sistema operativo.



La llegada de ODBC supuso un avance sin precedentes en el camino hacia la interoperabilidad entre bases de datos y lenguajes de programación. La mayoría de los desarrolladores de sistemas de gestión de bases de datos incorporaron controladores de conectividad en las utilidades de sus sistemas y los principales lenguajes de programación desarrollaron bibliotecas específicas para admitir la API ODBC.

Situación presente

Actualmente, ODBC sigue siendo una iniciativa adecuada para conectarse a DBMS relacionales. Su desarrollo sigue liderado por Microsoft, pero existen versiones en otros sistemas operativos ajenos a la compañía, como UNIX/LINUX o MAC. Los lenguajes de desarrollo más populares mantienen actualizadas las bibliotecas de comunicación con las sucesivas versiones que han ido apareciendo y la mayoría de DBMS cuentan con un controlador ODBC básico.

Actualmente, ODBC se estructura en tres niveles:

1er. API principal. Es el nivel más básico correspondiente a la especificación original (basada en SQL Access Group).

2do. API de nivel 1.

3er. API de nivel 2. Tanto el Nivel 1 como el Nivel 2 añaden funcionalidades avanzadas, como llamadas a procedimientos almacenados en el sistema, aspectos de seguridad de acceso, definición de tipos estructurados, etc.

En realidad, ODBC es una especificación de bajo nivel, es decir, funciones básicas de habilitación de conexión que garantizan la atomicidad de las solicitudes, la devolución de información, la encapsulación del lenguaje de consulta SQL o la recuperación de datos obtenidos en respuesta a una solicitud. el bajo nivel

la funcionalidad lo hace adaptable a una gran cantidad de aplicaciones; sin embargo, a costa de un número considerable de líneas de código necesarias para adaptarse a la lógica de cada aplicación. Es por esto que han surgido otras alternativas de persistencia de nivel superior sobre la base de ODBC. Por ejemplo, Microsoft ha desarrollado OLE DB o ADO.NET.

2.2. JDBC

Casi simultáneamente con ODBC, la empresa Sun Microsystems lanzó JDBC en 1997, una API de conector de base de datos, implementada específicamente para su uso con el lenguaje Java. Es una API bastante similar a ODBC en cuanto a funcionalidad, pero adaptada a las especificidades de Java. Es decir, la funcionalidad está encapsulada en clases (ya que Java es un lenguaje totalmente orientado a objetos) y, además, no depende de ninguna plataforma concreta, de acuerdo con la característica multiplataforma que preconiza Java.

Este conector será la API que estudiaremos en detalle en esta unidad, ya que Java no tiene ninguna biblioteca ODBC específica. Las razones dadas por Sun fueron que ODBC no se puede utilizar directamente en Java ya que está implementado en C y no está orientado a objetos. En otras palabras, utilizar ODBC desde el lenguaje Java requeriría una biblioteca completa para adaptar la API de ODBC a los requisitos de Java.

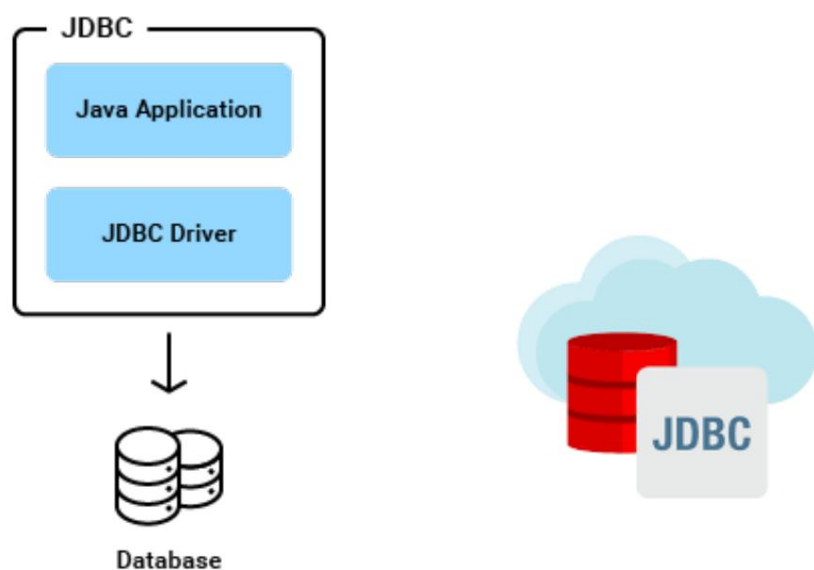
Sun Microsystem optó por una solución que permite hacer ambas cosas al mismo tiempo; por un lado, implementando un conector específico de Java que puede comunicarse directamente con cualquier base de datos mediante controladores de forma muy similar a ODBC, y por otro, incorporando de serie un controlador especial que actúa como adaptador entre la especificación JDBC y la especificación ODBC. Este controlador también suele denominarse puente JDBC-ODBC. Con este controlador puede vincular cualquier aplicación Java a cualquier conexión ODBC.

Actualmente la gran mayoría de DBMS cuentan con controladores JDBC, pero en caso de tener que trabajar con un sistema que no los tiene, si tiene un controlador ODBC, puedes utilizar el puente JDBC-ODBC para lograr la conexión desde Java.

¿Qué es JDBC?

JDBC (Java DataBase Connectivity) es la interfaz común que proporciona Java para poder conectarse a cualquier RDBMS (Relational DataBase Management System) con este lenguaje de programación. Proporciona una completa API (Application Program Interface) para trabajar con Bases de Datos Relacionales de tal forma que sea cual sea el motor al que nos conectemos, la API siempre será la misma.

Simplemente tendremos que conseguir el Driver correspondiente al motor que queremos utilizar, que dependerá enteramente de él. A pesar de ello no supone mucho problema ya que de momento podemos encontrar un controlador JDBC para prácticamente cualquier RDBMS existente.

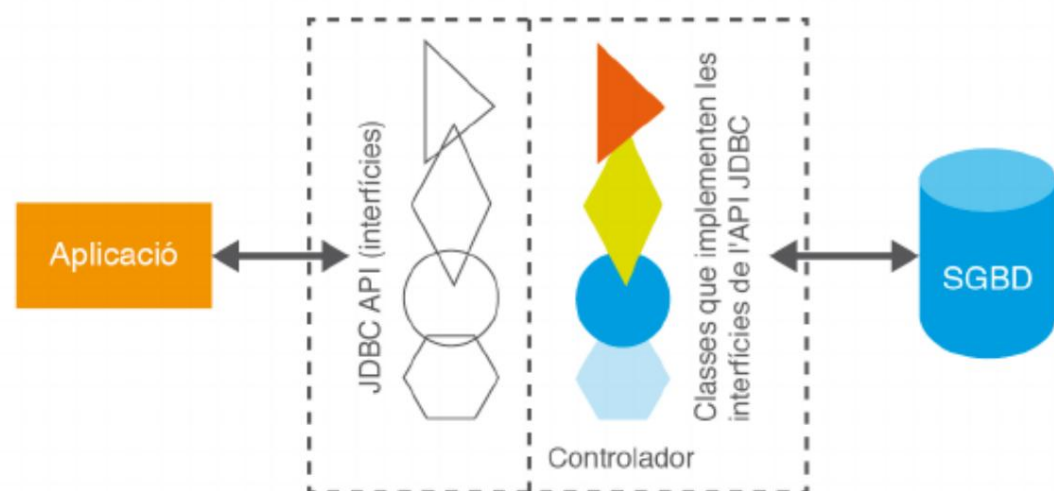


Dado que el controlador es lo único que depende exclusivamente del RDBMS utilizado, es muy fácil escribir aplicaciones cuyo código pueda ser reutilizado si luego tenemos que cambiar el motor de la base de datos o si queremos permitir que la aplicación se conecte a más de un RDBMS para que el usuario no tenga que estar vinculado al mismo RDBMS.

2.2.1. Arquitectura JDBC

Al igual que ODBC, cada desarrollador de sistemas de gestión de bases de datos implementa sus propios controladores JDBC. Para lograr la interoperabilidad entre controladores, la biblioteca estándar JDBC contiene una gran cantidad de interfaces sin las clases que las implementan. El controlador de cada proveedor incorpora las clases que implementan las interfaces API JDBC.

De esta forma, el controlador utilizado será completamente transparente para la aplicación, es decir, durante el desarrollo de la aplicación no será necesario saber qué controlador se utilizará finalmente para la explotación de los datos, ya que la aplicación declarará exclusivamente el JDBC. Interfaces API.



2.2.2. Tipos de controlador

JDBC distingue entre cuatro tipos de controladores:

1. Tipo I. Controladores puente como JDBC-ODBC. Se caracterizan por utilizar una tecnología externa a JDBC y actuar como adaptador entre las especificaciones API de JDBC y la tecnología específica utilizada. El más conocido es el controlador puente JDBC-ODBC, pero existen otros, como JDBC-OLE DB.

Su principal razón de ser es permitir el uso de una tecnología generalizada y así garantizar la conexión a prácticamente cualquier fuente de datos. Por otro lado, hay que decir que es necesario que cada cliente tenga instalada una utilidad de configuración y gestión de fuentes de datos ODBC (o de la tecnología utilizada) además del controlador ODBC específico del SGBD, que debe estar instalado y configurado. El hecho de tener que conectarse mediante un adaptador que hace de puente puede provocar en ocasiones problemas de rendimiento y, por tanto, es recomendable utilizarlo sólo como última alternativa.

2. Tipo II. Controladores Java con API parcialmente nativa (controlador Native-API parcialmente Java). También se les conoce como simplemente nativos. Como su nombre lo indica, constan de una parte codificada en Java y otra parte que utiliza bibliotecas binarias instaladas en el sistema operativo. Este tipo de drivers existen porque algunos sistemas de gestión de datos tienen entre sus utilidades estándar conectores específicos del sistema de gestión de datos. Suelen ser conectores propietarios que no siguen ningún estándar, ya que suelen ser anteriores a ODBC o JDBC, pero se mantienen debido a que suelen estar muy optimizados y eficientes.

Utilizando una tecnología Java llamada JNI es posible implementar clases cuyos métodos invocan funciones de bibliotecas binarias instaladas en el sistema operativo. Los controladores de tipo II utilizan esta tecnología para crear las clases de implementación de la API JDBC. En algunos casos puede requerir una instalación extra de determinadas utilidades del lado del cliente, requeridas por el conector nativo del sistema gestor.

3. Tipo III. Controladores Java vía protocolo de red. Este es un controlador escrito completamente en Java que traduce llamadas JDBC a un protocolo de red contra un servidor intermedio (comúnmente llamado Middleware) que puede estar conectado a varios DBMS. Este tipo de driver tiene la ventaja de que utiliza un protocolo independiente del DBMS y por tanto el cambio de fuente de datos se puede realizar de forma totalmente transparente para los clientes. Esto lo convierte en un sistema muy flexible, aunque, por otro lado, será necesario instalar un servidor intermedio conectado a todos los DBMS necesarios en algún lugar accesible de la red. Este tipo de controladores son bastante útiles cuando hay una gran cantidad de clientes, ya que los cambios en el DBMS no requerirán ningún cambio en los clientes, ni siquiera la adición de una nueva biblioteca.

4. Tipo IV. Puro Java Controladores tipo 100% Java. También se denominan controladores de protocolo nativo. Son controladores escritos íntegramente en Java. Las llamadas al sistema de gestión se realizan siempre a través del protocolo de red utilizado por el propio sistema de gestión y, por tanto, no se necesita código nativo en el cliente ni un servidor intermedio para conectarse a la fuente de datos. Se trata, por tanto, de un driver que no requiere ningún tipo de instalación ni requisito, lo que lo convierte en una alternativa muy valorada y que en los últimos tiempos se ha vuelto cada vez más popular. De hecho, la mayoría de los fabricantes han acabado creando un controlador de tipo IV aunque siguen manteniendo también los otros tipos.

3. ¿Qué es Maven?

La palabra maven proviene del yiddish meyn, que significa "el que entiende".

Pero para ser un experto hay que algo más que comprender un tema: hay que conocer sus entresijos. A menudo, los expertos son las personas a las que recurre como expertos en un campo. No te conviertes en un experto de la noche a la mañana. Ese tipo de experiencia viene con una acumulación de conocimientos a lo largo de los años.

Fuente: <https://www.vocabulary.com/dictionary/maven>

3.1. experto

La creación de un proyecto de software normalmente consta de tareas como descargar dependencias, colocar archivos jar adicionales en un classpath, compilar código fuente en código binario, ejecutar pruebas, empaquetar código compilado en artefactos implementables como archivos JAR, WAR y ZIP, e implementar estos artefactos. a un servidor o repositorio de aplicaciones.

Apache Maven automatiza estas tareas, minimizando el riesgo de que los humanos cometan errores mientras construyen el software manualmente y separan el trabajo de compilar y empaquetar nuestro código del de construcción del código.

Las características clave de Maven son:

- Configuración simple del proyecto que sigue las mejores prácticas. Maven intenta evitar la mayor cantidad de configuración posible proporcionando plantillas de proyecto (llamadas arquetipos).
- Gestión de dependencia. Incluye actualización automática, descarga y validación de la compatibilidad, así como informar los cierres de dependencias (conocidas también como dependencias transitivas).
- Aislamiento entre dependencias y complementos del proyecto. Con Maven, las dependencias del proyecto se recuperan de los repositorios de dependencias, mientras que las dependencias de cualquier complemento se recuperan de los repositorios de complementos, lo que genera menos conflictos cuando los complementos comienzan a descargar dependencias adicionales.
- Sistema de repositorio central. Las dependencias del proyecto se pueden cargar desde el archivo local. sistema o repositorios públicos, como Maven Central.

Para aprender cómo instalar Maven en su sistema, consulte este tutorial: <https://www.baeldung.com/install-maven-on-windows-linux-mac>



¿Por qué es útil Maven?

Fuente: <https://stackabuse.com/search/?q=maven>

Maven ha sido un proyecto de código abierto bajo Apache desde 2003, comenzando en Sonatype

antes de que. Dado su fuerte respaldo y su inmensa popularidad, Maven es muy estable y rico en funciones, y proporciona numerosos complementos que pueden hacer cualquier cosa, desde generar versiones en PDF de la documentación de su proyecto hasta generar una lista de cambios recientes de su SCM. Y todo lo que se necesita para agregar esta funcionalidad es una pequeña cantidad de XML adicional o un parámetro de línea de comando adicional.

¿Tienes muchas dependencias? Ningún problema. Maven se conecta a repositorios remotos (o puede configurar sus propios repositorios locales) y descarga automáticamente todas las dependencias necesarias para construir su proyecto.

3.2. ¿Cómo funciona Maven?

Se basa en un archivo central, pom.xml, el modelo de objetos del proyecto (POM) escrito en XML, donde defines todo lo que tu proyecto necesita. Maven gestiona las dependencias del proyecto, compila, empaqueta y ejecuta las pruebas. A través de complementos, te permite hacer mucho más, como generar mapas de Hibernate a partir de una base de datos, implementar la aplicación, etc.

Lo más útil de Maven es el manejo de dependencias. Antes de Maven, tenías que descargar manualmente el archivo jar que necesitabas en tu proyecto y copiarlo manualmente en el classpath. Fue muy tedioso. Con Maven esto se acabó. Solo necesita definir en su pom.xml las dependencias y maven las descarga y las agrega al classpath.

Más información sobre el archivo POM:

<https://maven.apache.org/guides/introduction/introduction-to-the-pom.html>

3.3. Maven y artefactos

a) Bibliotecas y limitaciones

El concepto de biblioteca es un concepto que en ocasiones resulta limitado. Por ejemplo, es posible que queramos utilizar la biblioteca A en nuestro proyecto. Sin embargo, no bastará simplemente con querer utilizar la biblioteca A, sino que también necesitaremos saber qué versión exacta de la misma necesitamos (A v1, A v2, etc.).

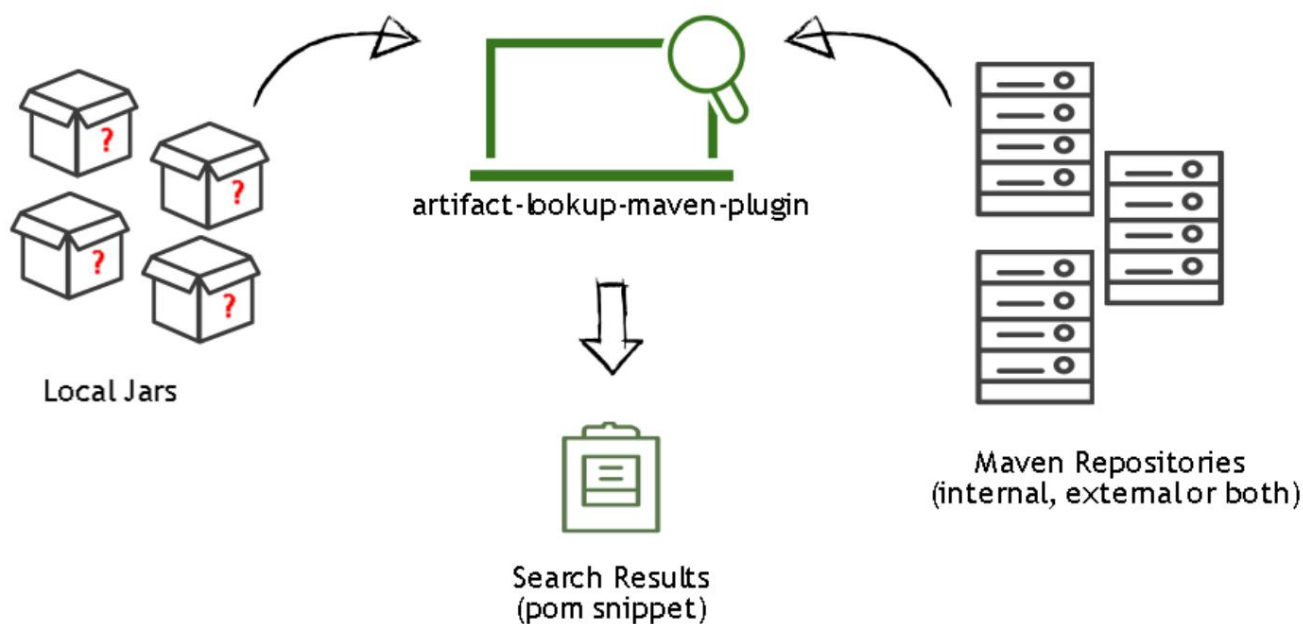
b) solución Maven

Maven resuelve este problema mediante el concepto de Artefacto. Un Artefacto puede verse como una biblioteca con satélites (aunque agrupa más conceptos). Contiene las clases propias de la biblioteca, pero también incluye toda la información necesaria para su correcta gestión (grupo, versión, dependencias, etc.).

Un artefacto es un objeto creado por el hombre que tiene algún tipo de significado cultural. Si encuentras un jarrón del siglo XII, es un artefacto de esa época. ¡No lo dejes caer!

Artefacto es una combinación de dos palabras latinas, arte, que significa "por habilidad" y factum que significa "hacer". Por lo general, cuando se utiliza la palabra artefacto, se describe algo elaborado que se utilizó para un propósito particular en una época mucho anterior.

Fuente: <https://www.vocabulary.com/dictionary/artifact>



3.4. Artefactos y archivo POM

Para definir un Artefacto necesitamos crear un archivo pom.xml (Project Object Model) que se encarga de almacenar toda la información que hemos comentado anteriormente. Este podría ser un ejemplo. Sólo echa un vistazo, no necesitas hacer nada ahora.

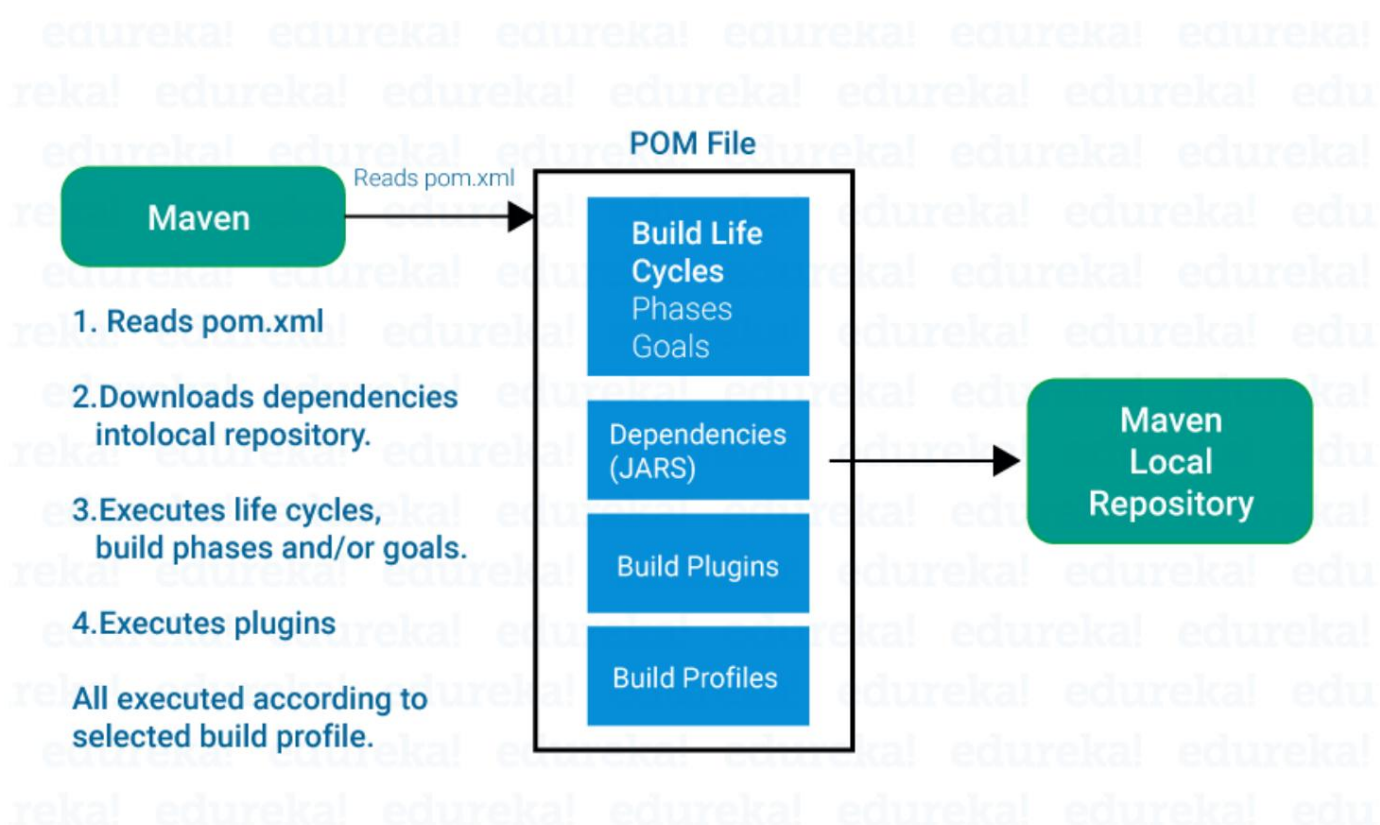
```
<project xmlns="http://maven.apache.org/POM/4.0.0"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://maven.apache.org/POM/4.0.0
    http://maven.apache.org/xsd/maven-4.0.0.xsd">
  <modelVersion>4.0.0</modelVersion>
  <groupId>com.genbetadev.project1</groupId>
  <artifactId>project1</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <packaging>jar</packaging>
  <dependencies>
    <dependency>
      <groupId>log4j</groupId>
      <artifactId>log4j</artifactId>
      <version>1.2.17</version>
    </dependency>
  </dependencies>
</project>
```



La estructura del archivo puede ser muy compleja y puede depender de otros POM. En este ejemplo estamos viendo el archivo más simple posible. En él definimos el nombre del Artefacto (artifactID) el tipo de empaquetado (jar) y también las dependencias que tiene (log4j). De esta forma nuestra biblioteca queda definida de una forma mucho más clara.

3.5. Repositorio y artefactos de Maven

Una vez definidos correctamente todos los Artefactos que necesitamos, Maven nos proporciona un Repositorio donde alojarlos, mantenerlos y distribuirlos. Permittiéndonos una correcta gestión de nuestras bibliotecas, proyectos y dependencias. El uso de Maven es hoy en día una necesidad en cualquier proyecto Java/Java EE de determinada entidad.



4. Acceso a RDBMS con Maven

Veamos un ejemplo completo de cómo acceder a bases de datos relacionales usando Maven.

4.1. Preparar el entorno y la base de datos.

Elegiremos el DBMS, crearemos las estructuras necesarias y poblaremos la base de datos. Puedes consultar un ranking de los DBMS más utilizados en el mundo, actualizado cada mes: <https://db-engines.com/es/ranking>

En este caso elegimos MySQL. Se deben seguir los siguientes pasos:

1) Instalar MySQL

- <https://www.digitalocean.com/community/tutorials/how-to-install-mysql-on-ubuntu-20-04-es>
- <https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/>

2) Utilice el código proporcionado a continuación para crear una nueva base de datos llamada "DBCompany" y un usuario llamado "mavenuser" con todos los privilegios (para hacerlo más fácil) y contraseña "ada0486". Puede utilizar la terminal o el entorno gráfico MySQL WorkBench.

- Matomo. ¿Cómo creo una nueva base de datos y un usuario de base de datos en MySQL?
https://matomo.org/faq/how-to-install/faq_23484/

3) Dentro de esa base de datos, crea una nueva tabla llamada "Empleado" con los siguientes campos:

- ID de impuesto VARCHAR(9)
- Nombre VARCHAR(100)
- Apellido VARCHAR(100)
- Salario DECIMAL(9,2)

4) Agregue cinco empleados aleatorios a esa base de datos/tabla.

Este podría ser un buen ejemplo de cómo lograr esos pasos:

- Crear base de datos https://matomo.org/faq/how-to-install/faq_23484/

CREAR BASE DE DATOS DBCompany;

-- Cree un usuario si está utilizando MySQL 5.7 o MySQL 8 o posterior

CREAR USUARIO 'mavenuser'@'localhost' **IDENTIFICADO CON** mysql_native_password **POR**

-- O si estás usando una versión anterior como MySQL 5.1, 5.5, 5.6:

```
-- CREAR USUARIO 'mavenuser'@'localhost' IDENTIFICADO POR 'ada0486';
```

```
CONCEDA TODOS LOS PRIVILEGIOS EN DBCompany.* A 'mavenuser'@'localhost';
```

```
-- Seleccione la base de datos a utilizar
```

```
UTILICE DBCompany;
```

```
- Crear la tabla del empleado.
```

```
CREAR TABLA Empleado (  
    Identificación del impuesto          VARCHAR(9),  
    Nombre de pila          VARCHAR(100),  
    Apellido                VARCHAR(100),  
    Salario                 DECIMALES(9,2),  
    RESTRICCIÓN emp_tid_pk CLAVE PRIMARIA (ID de impuesto)  
);
```

```
- Insertar empleados aleatorios
```

```
INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALORES ('11111111A',  
INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALORES ('22222222B',  
INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALORES ('33333333C',  
INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALORES ('44444444D',  
INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALORES ('55555555E',
```

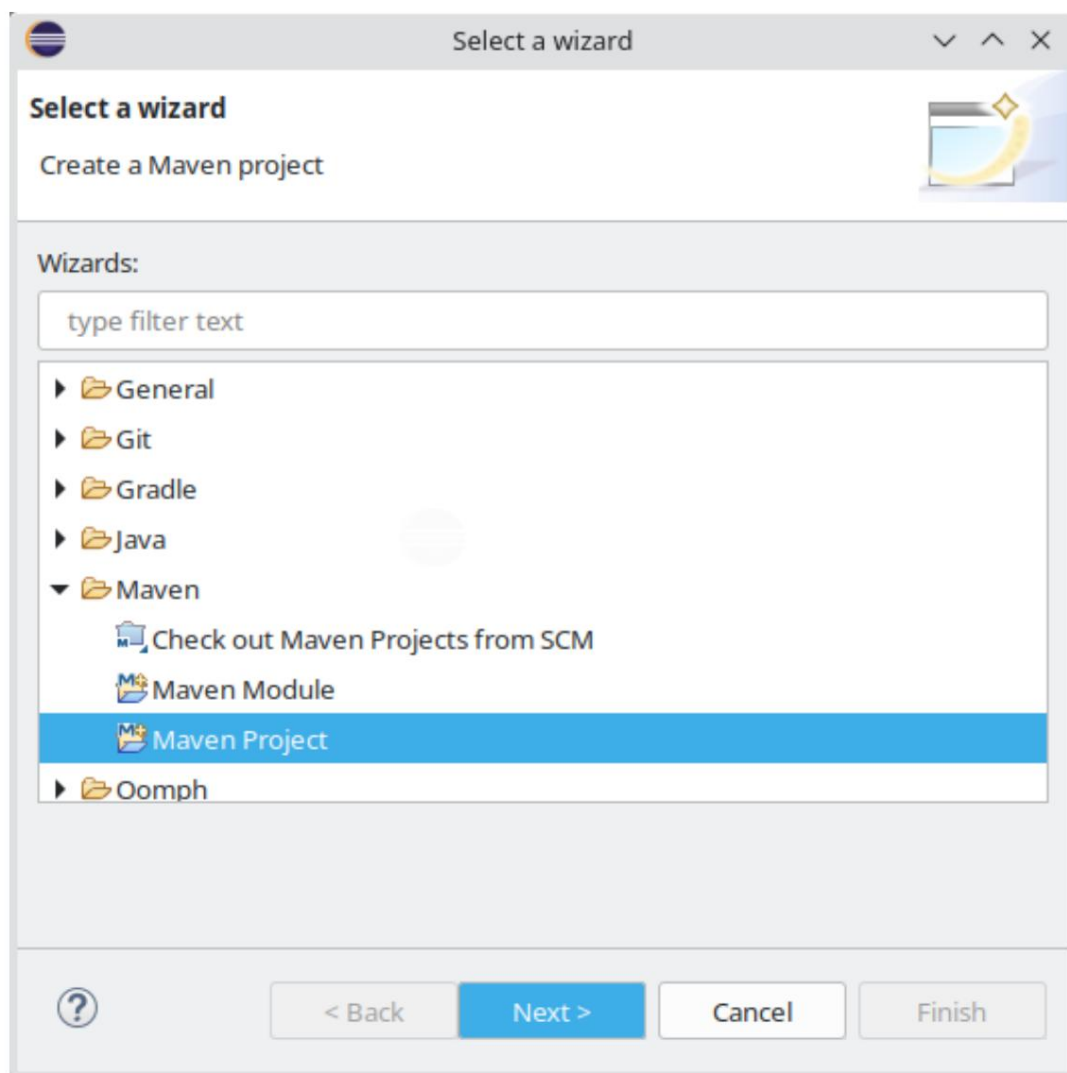
4.2. Crear un nuevo proyecto Maven Java

El primer paso es abrir Eclipse, que viene con el entorno Maven integrado.

- Vaya al menú Archivo, opción Nuevo → Proyecto.

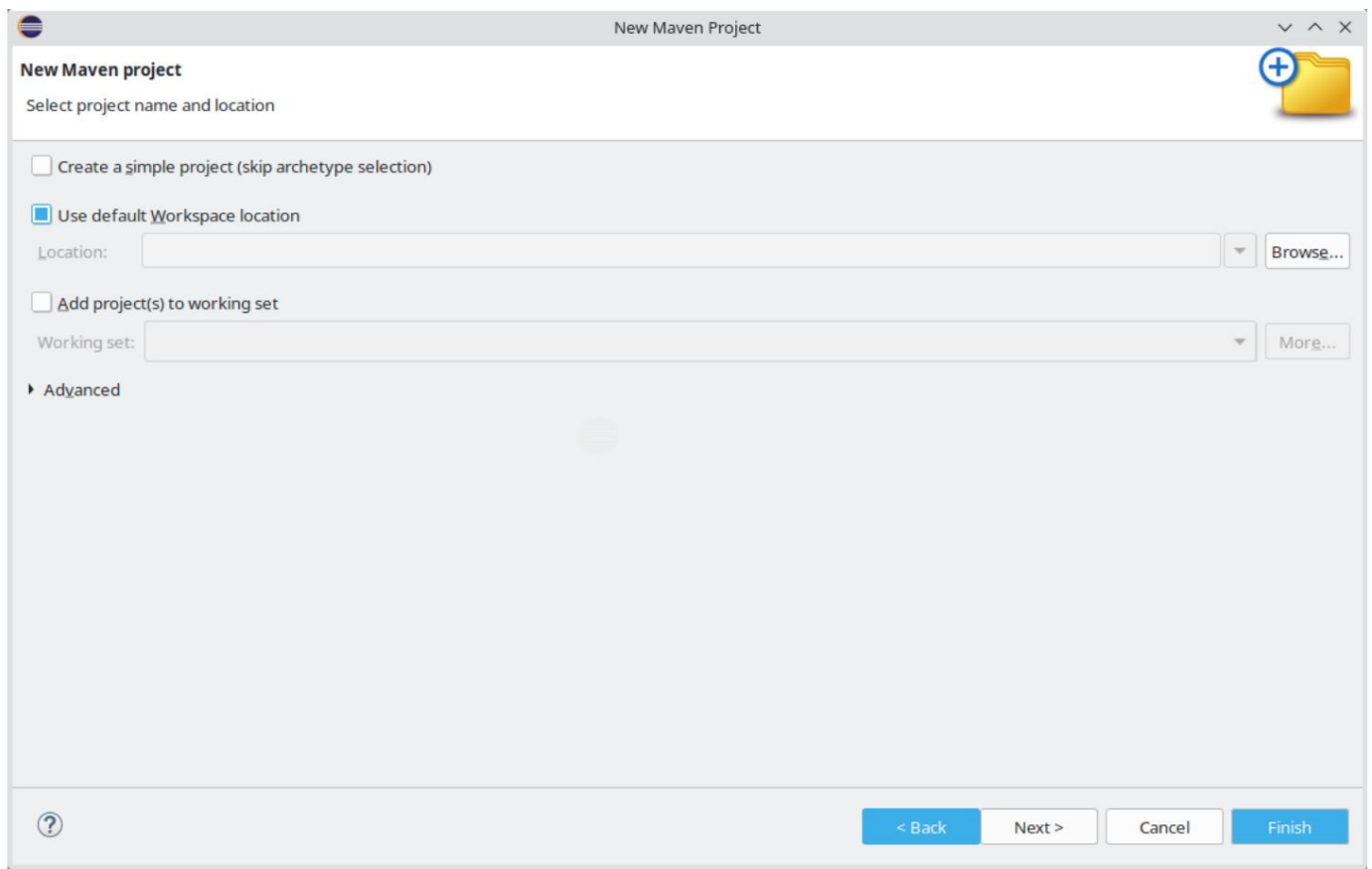
El cuadro de diálogo que aparece en la pantalla mostrará diferentes tipos de proyectos.

- Seleccione la opción Proyecto Maven.
- Haga clic en Siguiente.



Un cuadro de diálogo aparecerá. Seleccione el espacio de trabajo predeterminado.

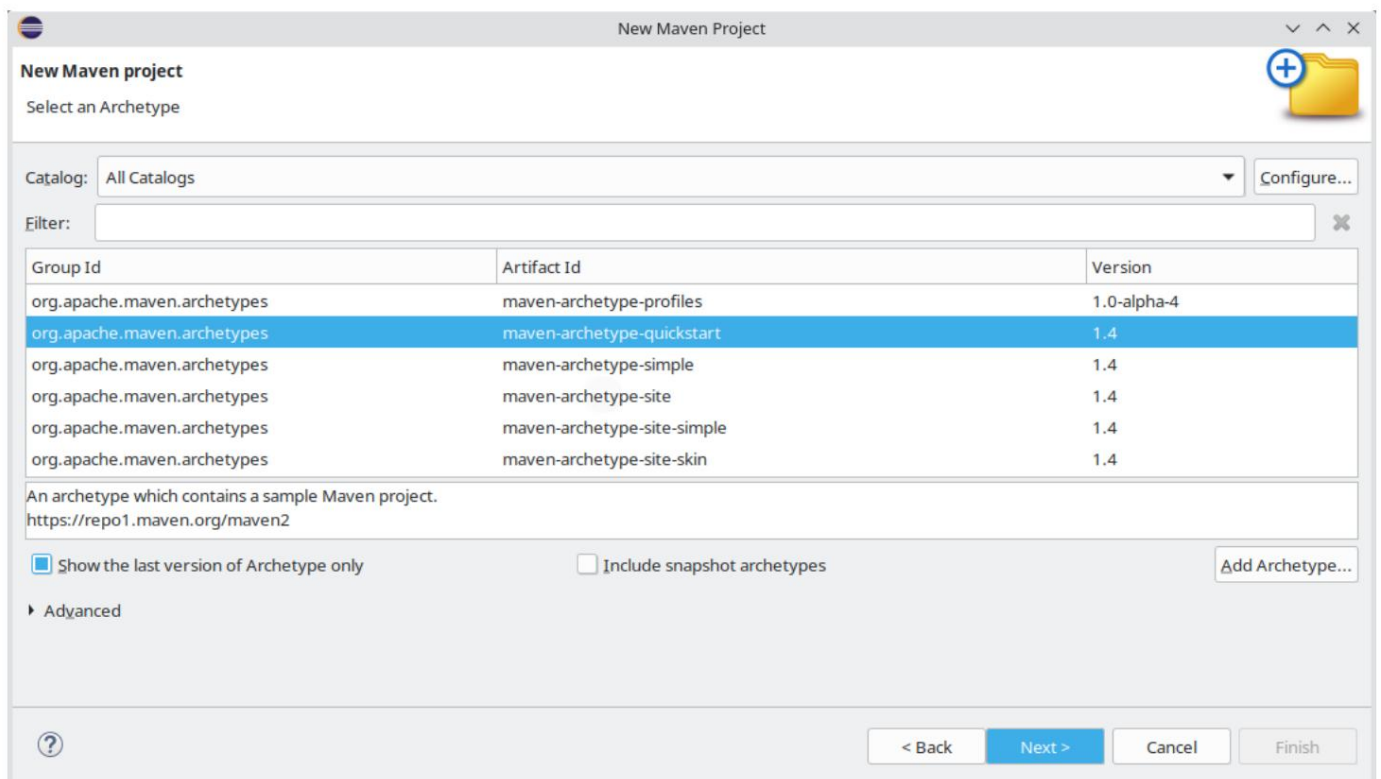
- Haga clic en "Siguiente".



The 'New Maven Project' dialog box is shown. It has a title bar with standard window controls. The main area is titled 'New Maven project' and 'Select project name and location'. It contains several options: 'Create a simple project (skip archetype selection)' (unchecked), 'Use default Workspace location' (checked), a 'Location:' text field with a 'Browse...' button, 'Add project(s) to working set' (unchecked), a 'Working set:' text field with a 'More...' button, and an 'Advanced' section which is currently collapsed. At the bottom, there are four buttons: '< Back', 'Next >', 'Cancel', and 'Finish'.

Luego aparecerán varios ID de grupo, ID de artefacto y versiones.

- Seleccione un complemento allí y haga clic en "Siguiente".



The 'New Maven Project' dialog box is shown at the 'Select an Archetype' step. It features a 'Catalog:' dropdown set to 'All Catalogs' with a 'Configure...' button, and a 'Filter:' text field. Below these is a table of archetypes. The table has three columns: 'Group Id', 'Artifact Id', and 'Version'. The second row, 'org.apache.maven.archetypes:maven-archetype-quickstart:1.4', is highlighted in blue. Below the table, there is a description: 'An archetype which contains a sample Maven project. https://repo1.maven.org/maven2'. There are two checkboxes: 'Show the last version of Archetype only' (checked) and 'Include snapshot archetypes' (unchecked), along with an 'Add Archetype...' button. An 'Advanced' section is collapsed at the bottom. The bottom navigation buttons are '< Back', 'Next >', 'Cancel', and 'Finish'.

Group Id	Artifact Id	Version
org.apache.maven.archetypes	maven-archetype-profiles	1.0-alpha-4
org.apache.maven.archetypes	maven-archetype-quickstart	1.4
org.apache.maven.archetypes	maven-archetype-simple	1.4
org.apache.maven.archetypes	maven-archetype-site	1.4
org.apache.maven.archetypes	maven-archetype-site-simple	1.4
org.apache.maven.archetypes	maven-archetype-site-skin	1.4

En el siguiente cuadro de diálogo que aparece, completará los siguientes pasos:

- Ingrese la ID del grupo. "com.simplilearn"
- Introduzca el ID del artefacto. "proyecto maven"
- La versión aparecerá en la pantalla.
- Todos estos elementos se pueden modificar más adelante si es necesario. Haga clic en "Finalizar"

New Maven Project

Specify Archetype parameters

Group Id:

Artifact Id:

Version:

Package:

☒ run archetype generation interactively

Properties available from archetype:

Name	Value

Advanced

< Back Next > Cancel Finish

La estructura y las dependencias necesarias se crean a continuación.

- Escriba Y en el terminal.

```

/usr/lib/jvm/java-11-openjdk-amd64/bin/java (10 oct 2023, 14:05:00) [pid: 80893]
[INFO] Using property: version = 0.0.1-SNAPSHOT
[INFO] Using property: package = com.simplilearn.U2JDBCExample
Confirm properties configuration:
groupId: com.simplilearn
artifactId: U2JDBCExample
version: 0.0.1-SNAPSHOT
package: com.simplilearn.U2JDBCExample
Y: : Y
    
```

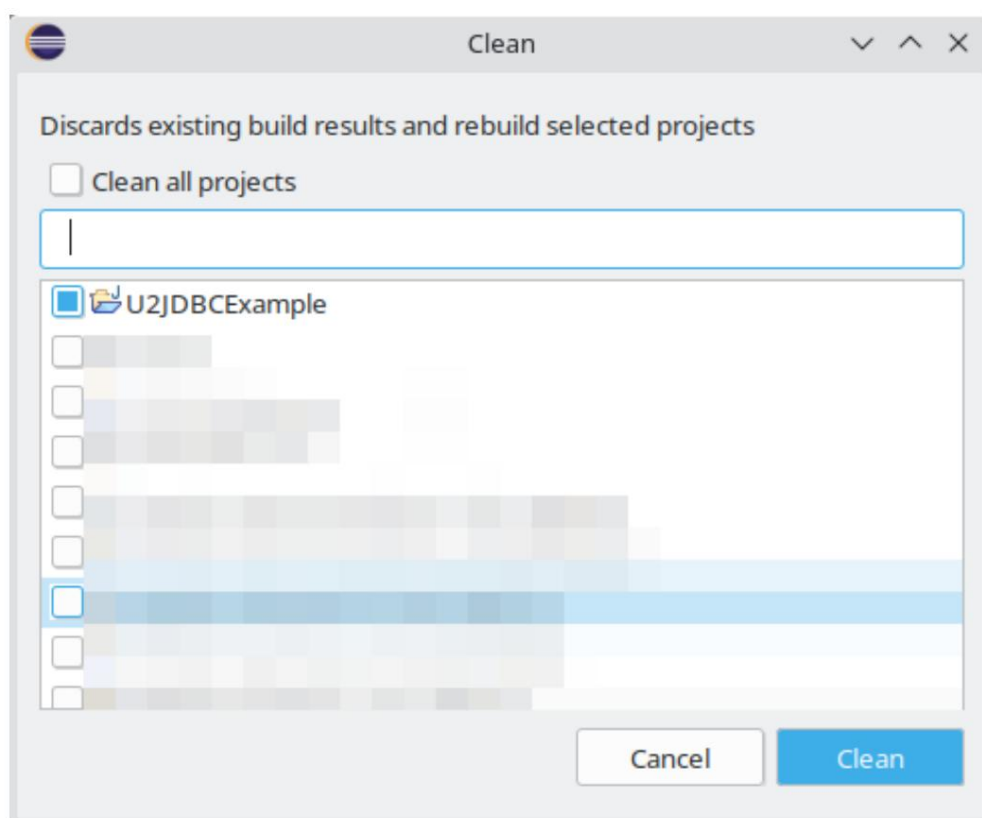
El proyecto ya está creado.

```

Problems @ Javadoc Declaration Console X Progress Error Log
<terminated> /usr/lib/jvm/java-11-openjdk-amd64/bin/java (10 oct 2023, 14:05:00) [pid: 80893]
[INFO] project created from old (1.x) Archetype in dir: /home/ /eclipse-workspace/U2JDBCExample
[INFO] -----
[INFO] BUILD SUCCESS
[INFO] -----
[INFO] Total time: 36.518 s
[INFO] Finished at: 2023-10-10T14:05:39+02:00
[INFO] -----

```

A continuación, seleccionamos Proyecto → Limpiar en nuestro proyecto para que las bibliotecas y archivos necesarios se hayan descargado correctamente. El archivo .jar es el que podemos distribuir en diferentes ordenadores, el cual contiene nuestro proyecto. Así es como se llega allí con Eclipse:



- Abra el archivo pom.xml.

Puede ver toda la información básica que ha ingresado en la pantalla, como el ID de artefacto, el ID de grupo, etc. Puede ver que se han agregado las dependencias de junit. Este proceso se lleva a cabo de forma predeterminada en Eclipse.

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

```
<proyecto xmlns="http://maven.apache.org/POM/4.0.0" xmlns:xsi="http://www.w3.org/2001/
```

```
xsi: esquemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/m
<modelVersion>4.0.0</modelVersion>

<groupId>com.simplilearn</groupId>
<artifactId>U2JDBCEXample</artifactId>
<versión>0.0.1-SNAPSHOT</versión>

<name>U2JDBCEXample</
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>prueba</scope>
  </dependency>
</dependencias>
```


1. Copie y pegue el código de dependencia y agréguelo a su archivo pom.xml dentro de un nuevo nodo llamado <dependencias>, configurando la versión adecuada dentro del nodo <versión>. Su archivo POM debería ser así en Eclipse.

```
<?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>com.simplilearn</groupId>
<artifactId>U2JDBCExample</artifactId>
<versión>0.0.1-SNAPSHOT</versión>
```

```
<name>U2JDBCExample</
name> <!-- FIXME cámbielo al sitio web del proyecto -->
<url>http://www.example.com</url>
```

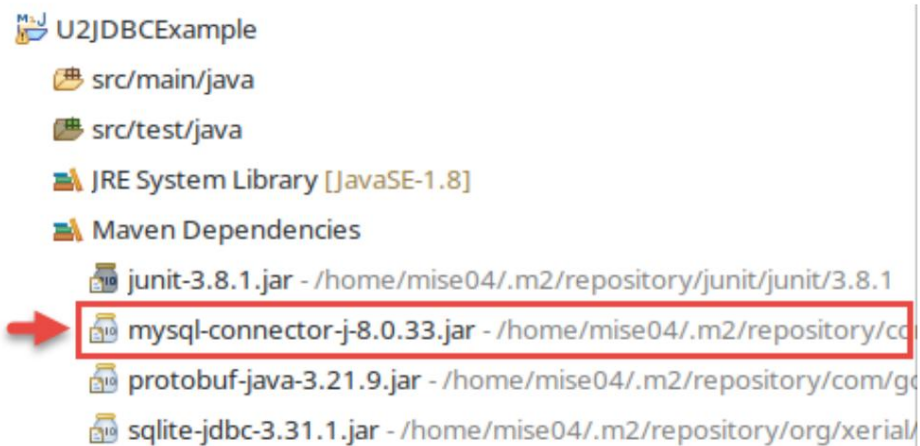
```
<propiedades>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</propiedades>
```

```
<dependencias>
  <dependencia>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <versión>4.11</versión>
    <scope>test</scope> </
  dependency>
  <!-- https://mvnrepository.com/artifact/ com.mysql/mysql-connector-j --> <dependencia>

    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <versión>8.0.33</versión>
```

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

5. Ahora vaya a Proyecto → Limpiar y verifique los archivos JAR que su IDE descargará por usted.
Si no se genera el archivo jar, elimine todo el proyecto (con los archivos) y vuelva al paso 1.



4.4. Estableciendo una conexión

Hay varias formas de conectarse a un RDBMS. Usaremos la clase DriverManager que se explica aquí:

<https://dev.mysql.com/doc/connector-j/8.0/en/connector-j-usagenotes-connect-drivermanager.html>

Cabe recordar que la API JDBC, aparte de algunas clases específicas, se compone principalmente de interfaces que el controlador implementa dándoles la funcionalidad adecuada. Para garantizar la interoperabilidad, las aplicaciones nunca harán referencia a las clases específicas de ningún controlador, excepto a las interfaces API JDBC estándar.

Para lograr esto, la aplicación nunca puede crear una instancia directa de los objetos JDBC con una nueva declaración, sino que se crean indirectamente llamando a un método de una clase u objeto existente que los instancia internamente antes de devolverlos a la aplicación. Por ejemplo, se debe crear una instancia de la interfaz Connection a partir del método estático getConnection de la clase DriverManager.

Conexión representa una conexión a la base de datos, un canal de comunicación entre la aplicación y el DBMS. Los objetos de conexión mantendrán la capacidad de comunicarse con el sistema de gestión mientras permanezcan abiertos. Es decir, desde que se crean hasta que se cierran mediante el método close.

El objeto Conexión está totalmente vinculado a una fuente de datos, por lo que al solicitar la conexión se debe especificar la fuente siguiendo el protocolo JDBC e indicando la url de los datos, y si es necesario el usuario y contraseña.

La URL seguirá el protocolo JDBC, comenzando siempre con la palabra jdbc seguida de dos puntos. El resto dependerá del tipo de controlador utilizado, el host donde está alojado el DBMS, el puerto que utiliza para escuchar las solicitudes y el nombre de la base de datos o esquema con el que queremos trabajar.

DriverManager es una clase API estándar JDBC muy especial, ya que su misión es interceder entre la aplicación y el controlador o controladores JDBC (si hay más de uno).

Desde la URL de conexión a una fuente de datos, la clase DriverManager es capaz de localizar entre todos los controladores que tiene registrados el controlador adecuado, lo que permite realizar una conexión utilizando la URL especificada. Es decir, gracias al DriverManager no necesitamos saber el nombre de la clase principal del driver (clase que tiene que implementar la interfaz java.sql.Driver de la API JDBC estándar).

El método `getConnection` del `DriverManager`, además de buscar y localizar el controlador correspondiente, obtendrá una Conexión del controlador seleccionado y la devolverá activa a la aplicación.

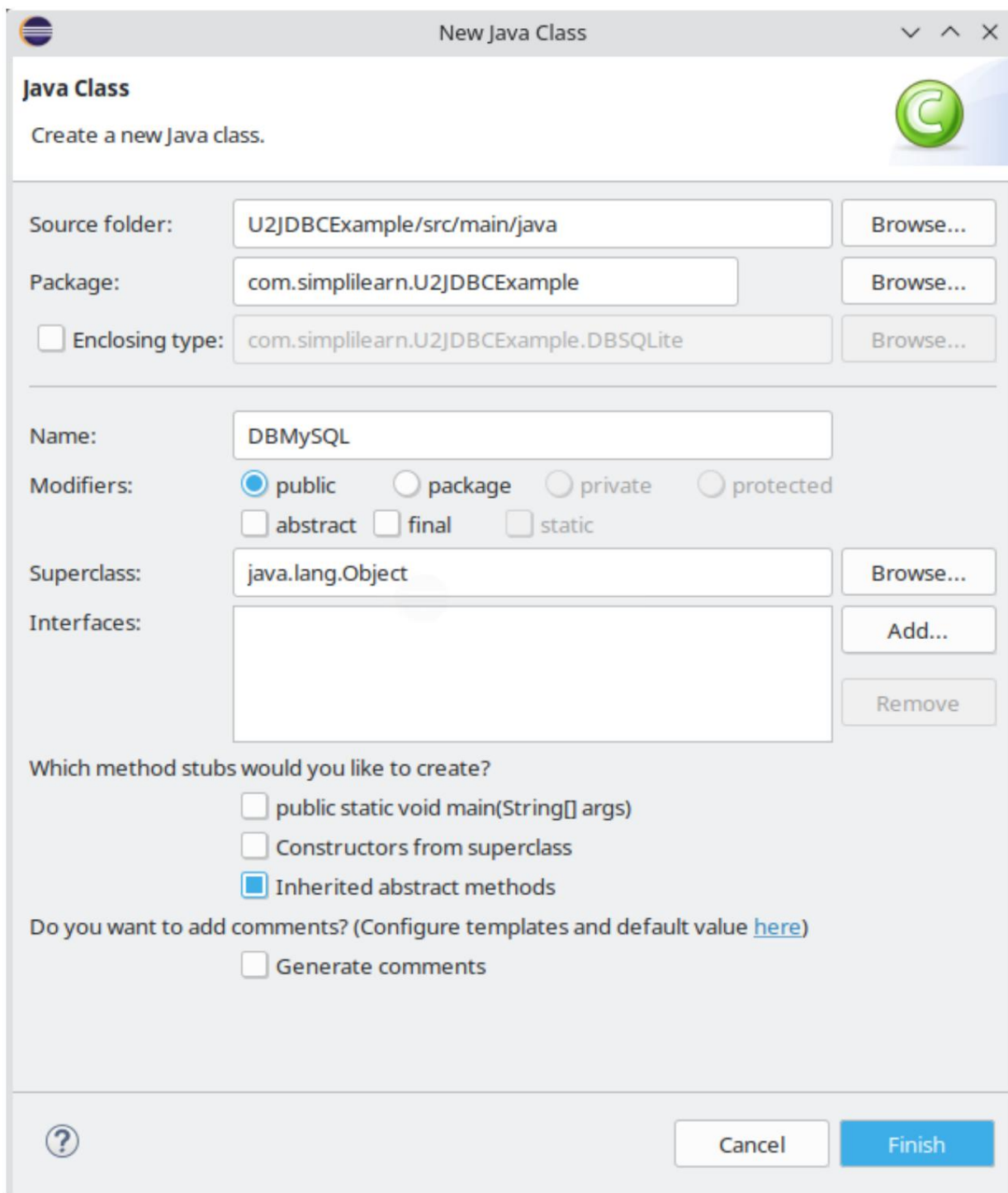
Este método puede generar excepciones y el compilador nos obligará a proteger el código incluyéndolo en una declaración `try-catch`.

Finalmente, antes de empezar a trabajar con sentencias SQL, es importante cerrar todas las conexiones abiertas antes de salir de la aplicación, ya que si la conexión no se cierra, el sistema de gestión mantendrá la conexión en memoria, desperdiciando recursos. Los objetos de conexión tienen los métodos `isClosed` y `close` para gestionar el cierre. Invocando al primero sabremos si la conexión sigue operativa o si ya se ha cerrado.

El segundo es el método que realiza el cierre.

4.4.1. Agregue la cadena de conexión a nuestro RDBMS

- Primero, necesitamos crear una nueva clase. Por ejemplo:



New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

- Segundo, tenemos que construir nuestra cadena de conexión para MYSQL (donde DBNAME corresponde al nombre de la base de datos):

```
URL="
```

```
jdbc:mysql://localhost:3306/NOMBREDDB
```

```
?useSSL=falso
```

```
&useTimezone=verdadero
```

```
&serverTimezone=UTC
```

```
&allowPublicKeyRetrieval=verdadero
```

```
"};
```

- En tercer lugar, implementamos el código:

```
importar java.sql.*;
```

```
/**
```

```
 *
```

```
=====
```

```
*Ejemplo de acceso a una base de datos relacional MySQL con JDBC* @autor
```

```
Abelardo Martínez
```

```
 *
```

```
=====
```

```
*/
```

```
clase pública DBMySQL {
```

```
    /**
```

```
     *
```

```
=====
```

```
    * VARIABLES Y CONSTANTES GLOBALES
```

```
* -----  
  
*/  
  
// Constantes DB  
static final String DBUSER = "mavenuser"; //usuario static final  
String DBPASSWORD = "ada0486"; //contraseña //cadena para  
conectarse a MySQL static final  
String URL = "jdbc:mysql://localhost:3306/DBCompany?useSSL=false&use  
  
/**  
* -----  
  
* PROGRAMA PRINCIPAL  
* -----  
  
*/  
  
principal vacío estático público ( cadena [] stArgs )  
{  
    prueba {  
        //establece la conexión con DBCompany Connection  
        cnDBCompany = DriverManager.getConnection(URL, DBUSER, DBPASSW  
  
        cnDBCompany.close(); //cieramos la conexión a la BD  
    } captura (SQLException sqle) {  
        sqle.printStackTrace(System.out);  
    }  
}  
}
```

4.5. Ejecutar sentencias SQL

Para escribir sentencias SQL, JDBC proporciona objetos `Statement`. Estos son objetos instanciados por `Connection`, que pueden enviar declaraciones SQL al sistema de gestión conectado para que se ejecuten invocando el método `executeQuery` o `executeUpdate`.

- El método `executeQuery`, que veremos más adelante, se utiliza para ejecutar sentencias que se espera que devuelvan datos, es decir, consultas.
- El método `executeUpdate`, por otro lado, es específicamente para declaraciones que modifican la base de datos conectada, pero no se espera que devuelvan ningún dato.

consultas SQL

Las consultas son las sentencias SQL que nos permiten recuperar, total o parcialmente, los datos almacenados en la base de datos. En este caso, estamos recuperando datos de la única tabla que tenemos.

Con los objetos `Statements` también podemos gestionar consultas SQL, pero se debe invocar el método `executeQuery`, porque este método devuelve el conjunto de datos correspondientes a la consulta realizada. Los datos se devuelven mediante un objeto `ResultSet`. Por lo tanto, la ejecución de consultas será similar a la siguiente:

```
ResultSet rs = declaración.executeQuery(sentenciaSQL);
```

El objeto `ResultSet` contiene el resultado de la consulta organizado por filas. Se pueden visitar todas las filas una a la vez llamando al método `next`, ya que cada invocación de `next` avanzará a la siguiente fila.

Inmediatamente después de una ejecución, el `ResultSet` devuelto se coloca justo antes de la primera fila, por lo que para acceder a la primera fila debe invocar `next` una vez.

Cuando se agoten las filas, el siguiente método devolverá falso.

Desde cada fila se puede acceder al valor de sus columnas utilizando la diversidad de métodos disponibles dependiendo del tipo de dato a devolver y pasando como parámetro el número de la columna que queremos obtener. El nombre de los métodos sigue un patrón bastante simple: usaremos `get` como prefijo y el nombre del tipo como sufijo. Entonces, si quisiéramos recuperar la segunda columna, sabiendo que es un tipo de datos `String`, tendríamos que invocar:

```
rs.getString(2);
```

Hay que tener en cuenta que las columnas empiezan a contar desde el valor 1 (no cero). La mayoría de DBMS soportan la posibilidad de pasar el nombre de la columna como parámetro, pero al no poder garantizar un correcto funcionamiento en ningún sistema, normalmente se elige siempre el parámetro numérico.

Por último, cabe destacar que los objetos `ResultSet` también deben cerrarse del mismo modo que las Sentencias o las conexiones. Sin embargo, tenga en cuenta que los `ResultSets` son los primeros en cerrarse.

4.5.1. Consultar datos de la tabla.

Ejecutar la consulta

Realizamos la consulta a través de la interfaz Statement.

1. Para obtener un objeto Declaración llamamos al método `createStatement()` de una Conexión válida objeto.
2. El objeto Statement tiene el método `executeQuery()` que ejecuta una consulta en la base de datos. Este método recibe como parámetro un String que debe contener un SQL declaración.
3. El resultado se recopila en un objeto ResultSet, este objeto es una especie de matriz o lista que incluye todos los datos devueltos por el DBMS.

Atravesando el resultado

Una vez que hemos obtenido los resultados de la consulta y los hemos almacenado en nuestro ResultSet solo nos queda recorrerlos y tratarlos como mejor nos parezca.

- ResultSet ha implementado un puntero que apunta al primer elemento de la lista.
- Mediante el método `next()` el puntero avanza al siguiente elemento.
 - El método `next()`, además de avanzar al siguiente registro, devuelve verdadero si hay más registros y falso si ha llegado al final.
 - Los métodos `getString()` y `getDouble()` se utilizan para obtener los valores de las diferentes columnas. Estos métodos reciben el nombre de la columna como parámetro.

Liberar recursos

Finalmente, debemos liberar todos los recursos utilizados para asegurar la correcta ejecución del programa.

Las instancias de Conexión y Declaración almacenan, en memoria, mucha información relacionada con las ejecuciones realizadas. Además, mientras permanecen activos, mantienen en el DBMS un importante conjunto de recursos abiertos, destinados a atender eficientemente las solicitudes de los clientes. Cerrar estos objetos libera recursos tanto del cliente como del servidor.

De hecho, las Declaraciones son objetos íntimamente vinculados al objeto Conexión que las instancia y si no se cierran específicamente, permanecerán activas mientras la conexión permanezca activa, incluso después de que la variable que las referenciaba haya desaparecido.

Además, la complejidad de estos objetos significa que aunque la conexión haya sido

cerrado, los objetos Statement que no se han cerrado expresamente permanecerán en la memoria por más tiempo que los objetos cerrados previamente, ya que el recolector de basura de Java tendrá que realizar más comprobaciones para garantizar que ya no tenga dependencias internas o externas y que pueda eliminarse. Por eso se recomienda cerrarlo siempre manualmente mediante la operación de cierre. Los objetos de declaración de cierre garantizan la liberación inmediata de recursos y la eliminación de dependencias.

Si en el mismo método tenemos que cerrar un objeto Statement y la conexión que lo instancia, tendremos que cerrar primero la instancia Statement y luego la instancia Connection. Si lo hacemos al revés, al intentar cerrar el Statement obtendremos una excepción de tipo SQLException, ya que el cierre de la conexión la habría dejado inaccesible.

Ejemplo

Nos conectamos a la base de datos y obtenemos todas las filas:

```
importar java.sql.*;

/**
 * =====
 *
 * Ejemplo de acceso a una base de datos relacional MySQL con JDBC* @autor
 * Abelardo Martínez
 * =====
 */

clase pública DBMySQL {

    /**
     * -----
     *
     * VARIABLES Y CONSTANTES GLOBALES
     * -----
     */

    // Constantes DB
    static final String DBUSER = "mavenuser"; //usuario static final
    String DBPASSWORD = "ada0486"; //contraseña //cadena para
    conectarse a MySQL static final
    String URL = "jdbc:mysql://localhost:3306/DBCompany?useSSL=false&use
```



```
/**
 * -----
 * PROGRAMA PRINCIPAL
 * -----
 */

principal vacío estático público ( String[] stArgs )
{
    prueba {
        //establecemos la conexión con DBCompany
        Conexión cnDBCompany = DriverManager.getConnection(URL, DBUSER, DBPASSW
        Declaración staSQLquery = cnDBCompany.createStatement();

        /**
         * -----
         * Sentencias SQL. Definición
         * -----
         */

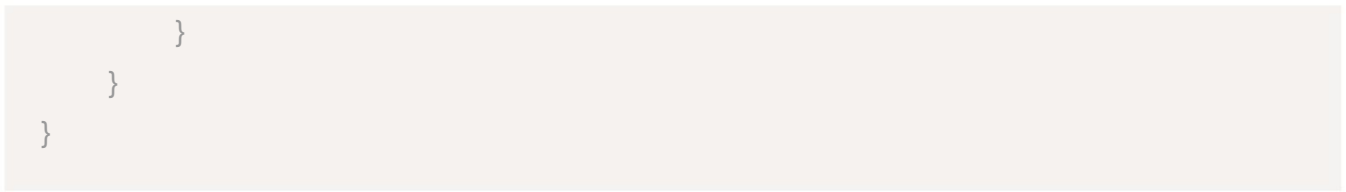
        //seleccionamos todos los registros
        String stSQLSelect = "SELECCIONAR * DEL Empleado";

        /**
         * -----
         * Sentencias SQL. Ejecución
         * -----
         */

        //recupera datos y los muestra en pantalla. sentencia de selección SQL
        ResultSet rsEmployee = staSQLquery.executeQuery(stSQLSelect);
        mientras (rsEmpleado.siguiente()) {
            System.out.println("TaxID:" + rsEmployee.getString("TaxID"));
            System.out.println(" Nombre: " + rsEmployee.getString("Nombre")
            System.out.println(" Apellido: " + rsEmpleado.getString("Apellido")
            System.out.println("Salario: " + rsEmployee.getBigDecimal("Salario")

        }

        //liberar recursos
        rsEmpleado.close(); //cerrar el conjunto de resultados
        staSQLquery.close(); //cerrar la declaración
        cnDBCompany.close(); //cierramos la conexión a la BD
    } captura (SQLException sqle) {
        sqle.printStackTrace(System.out);
    }
}
```



Y este podría ser un resultado:

Identificación fiscal: 11111111A

Nombre: José

Apellido: Salcedo López

Salario: 1279.90

Identificación fiscal: 22222222B

Nombre: Juan

Last name: De la Fuente Arqueros Salary:

1100.73

Identificación fiscal: 33333333C

Nombre: Antonio

Last name: Bosch Jericó

Salario: 1051,45

Identificación fiscal: 44444444D

Nombre: Ana

Last name: Sanchís Torres

Salario: 1300.02

NIF: 55555555E

Nombre: Isabel

Last name: Martí Navarro

Salario: 1051,45

5. Ejecutar sentencias DML

En SQL las sentencias DML son aquellas que nos permiten trabajar con los datos de una base de datos.

- SELECCIONAR. Para obtener datos de una base de datos.
- INSERTAR. Para insertar datos en una tabla.
- ACTUALIZAR. Modificar datos existentes dentro de una tabla.
- BORRAR. Para eliminar todos los registros de la tabla; no elimina los espacios asignados a los récords.

Para ejecutar cualquiera de estas sentencias debemos utilizar objetos:

- Declaración → Le permite ejecutar declaraciones SQL sin parámetros.
- PreparedStatement → Ejecutar sentencias SQL con parámetros.

5.1. La interfaz de declaración

La interfaz Statement permite la creación de objetos Statement. Al ser una interfaz, no podemos instanciar directamente este tipo de objetos sino que debemos usar el método `createStatement()` de la clase `Connection`. Los principales métodos de esta interfaz son:

Método	Descripción
<code>ResultSet ejecutarQuery (consulta de cadena)</code>	Se utiliza para ejecutar sentencias SQL que recuperan datos. Devuelve un objeto <code>ResultSet</code> con los datos recuperados.
<code>int ejecutarActualización(Cadena Consulta)</code>	Se utiliza para ejecutar declaraciones de manipulación como Insertar, Actualizar y Eliminar. Devuelve un número entero que indica el número de registros que han sido afectados.
<code>ejecución booleana (consulta de cadena)</code>	<p>Se puede utilizar para ejecutar cualquier consulta SQL. En caso de que la declaración devuelva un <code>ResultSet</code>, el método <code>ejecutar()</code> devuelve verdadero y debemos recuperar el objeto <code>ResultSet</code> a través del método <code>getResultSet()</code>.</p> <p>De lo contrario (cualquier otro tipo de declaración) devolverá falso y debemos usar el método <code>getUpdateCount()</code> para recuperar el valor devuelto.</p>

5.2. La interfaz de declaración preparada

Es muy común utilizar variables dentro de una declaración SQL:

- Valores a insertar, actualizar o eliminar.
- Filtros en consultas de selección.
- Etc.

Para este tipo de consultas con parte variable debemos hacer uso de las llamadas Sentencias Preparadas. La interfaz PreparedStatement nos permitirá crear marcadores de posición SQL que representan los datos que se agregarán más adelante (Variables). Estos marcadores de posición están representados por signos de interrogación (?). Por ejemplo:

```
String stSQLInsert = "INSERTAR EN Empleado (ID fiscal, nombre, apellido, salario) VALOR
```

Cada marcador de posición tiene un índice, donde 1 es el primer elemento encontrado en la cadena. Antes de ejecutar un PreparedStatement es necesario asignar valores a cada uno de los marcadores de posición. La ventaja de esto sobre las declaraciones tradicionales es el hecho de que puede preparar consultas (precompilarlas) solo una vez y tener la posibilidad de ejecutarlas tantas veces como sea necesario con diferentes valores de entrada. Ofrecen una gran flexibilidad.

Los principales métodos de PreparedStatement son:

Método	Descripción
<code>Conjunto de resultados ejecutarQuery()</code>	Similar a su contraparte en la interfaz Declaración.
<code>int ejecutarActualizar()</code>	Similar a su contraparte en la interfaz Declaración.
<code>ejecución booleana()</code>	Similar a su contraparte en la interfaz Declaración.

Además, disponemos de varios métodos para asignar valores a cada uno de los marcadores de posición.

Método	tipo SQL
void setString (índice int, valor de cadena)	VARCHAR
void setBoolean(índice int, valor booleano)	POCO
void setByte(índice int, valor de byte)	PEQUEÑO
void setShort(índice int, valor corto)	PEQUEÑO
void setInt(índice int, valor int)	ENTERO
void setLong (índice int, valor int)	EMPEZANDO
void setFloat(índice int, valor flotante)	FLOTAR
void setDouble(índice int, valor doble)	DOBLE
void setBytes(índice int, valor de byte[])	VARBINARIO
void setDate(índice int, valor de fecha)	FECHA
void setTime (índice int, valor de tiempo)	TIEMPO

Para asignar valores NULL a los marcadores de posición debemos usar:

- `setNull(index, int typeSQL)`, donde `typeSQL` es cualquiera de los tipos definidos en `java.sql.Types`.

```
String stSQLInsert = "INSERT INTO Empleado (ID de impuesto, nombre, apellido, salario) VALOR
PreparedStatement pstaSQLQuery = cnDBCompany.createStatement(stSQLInsert);
pstaSQLQuery.setString(1, "11111111A");
pstaSQLQuery.setString(2, "José");
pstaSQLQuery.setString(3, "Salcedo López");
pstaSQLQuery.setFloat(4, 1279.90f);
int iQueryvalue = pstaSQLQuery.executeUpdate();
System.out.println(iQueryvalue);
```

5.3. La clase Conjunto de resultados

A través de un objeto de la clase ResultSet podemos:

- Recopilar los valores devueltos por una consulta de selección.
- Acceder al valor de cualquier columna de la tupla actual a través de su posición o su nombre.
- Obtener información general sobre la consulta (como el número de columnas, su tipo, etc.) a través del método getMetada().

Para acceder a cualquiera de las columnas de la tupla actual usaremos los métodos getXXX():

Método	tipo java
getString(int númeroCol) getString(String nombreCol)	Cadena
getBoolean(int númeroCol) getBoolean(nombre de cadenaCol)	booleano
getBytes(int númeroCol) getBytes (nombre de cadenaCol)	byte
getShort(int númeroCol) getShort (nombre de cadenaCol)	corto
getInt(int númeroCol) getInt(String nombreCol)	En t
getLong(int numeroCol) getLong (nombre de cadenaCol)	largo
getFloat(int númeroCol) getFloat(nombre de cadenaCol)	flotar
getDouble(int númeroCol) getDouble(String nombreCol)	doble
getBytes(int númeroCol) getBytes (nombre de cadenaCol)	byte[]
getFecha(int númeroCol) getDate(String nombreCol)	Fecha
getTime(int numeroCol) getTime (nombre de cadenaCol)	Tiempo
getTimeStamp(int numeroCol) getTimeStamp(String nombreCol)	Marca de tiempo

Para recorrer un ResultSet usaremos el método next() como hemos visto en anteriores

ejemplos.

```
mientras (rsEmpleado.siguiete()) {  
    System.out.println("TaxID:" + rsEmployee.getString("TaxID"));  
    System.out.println(" Nombre: " + rsEmployee.getString("Nombre"));  
    System.out.println("Apellido : " + rsEmployee.getString("Apellido"));  
    System.out.println("Salario: " + rsEmployee.getBigDecimal("Salario"));  
}
```

6. Ejecutar sentencias DDL

Aunque el grueso de las operaciones que se realizan desde una aplicación cliente contra una base de datos son operaciones de manipulación de datos, puede haber casos en los que nos veamos obligados a realizar operaciones de definición.

- Acabamos de instalar una aplicación en un ordenador nuevo y necesitamos crear, de forma de forma automatizada, una base de datos local para su funcionamiento (el caso más común).
- Después de una actualización de software, la estructura de la BD ha cambiado y necesitamos actualizarla desde nuestra aplicación Java.
- Desconocemos la estructura de la BD y queremos realizar consultas dinámicas sobre la misma.

Declaraciones que no devuelven datos

Aunque SQL es esencialmente un lenguaje de consulta, también incluye una serie de comandos imperativos que permiten realizar solicitudes para cambiar las estructuras internas del DBMS donde se almacenarán los datos (instrucciones conocidas como DDL o Lenguaje de definición de datos), otorgar permisos para usuarios existentes o crear otros nuevos (un subconjunto de instrucciones conocido como DCL o Lenguaje de control de datos) o modificar los datos almacenados utilizando las instrucciones de inserción, actualización y eliminación.

Aunque se trata de declaraciones muy diferentes, desde el punto de vista de la comunicación con el DBMS se comportan de forma muy similar, siguiendo el patrón que se muestra a continuación:

1. Creación de instancias desde una conexión activa.
2. Ejecución de una declaración SQL pasada como parámetro al método `ejecutarUpdate`.
3. Cierre del objeto Declaración instanciado.

Las declaraciones DDL tradicionales siguen siendo declaraciones SQL y, por lo tanto, se pueden ejecutar de la forma que ya hemos estudiado a través de la interfaz `Statement/PreparedStatement`.

6.1. Definición y modificación de estructuras.

Mediante este mecanismo podríamos crear bases de datos y sus correspondientes estructuras (tablas, restricciones, vistas, índices, etc.). Es una práctica muy común cuando vamos a utilizar una BD local, y actualmente es muy utilizada en aplicaciones para dispositivos móviles, aplicaciones de gestión local, gestores de contenidos web, etc.

Las declaraciones DDL tradicionales siguen siendo declaraciones SQL y, por lo tanto, se pueden ejecutar de la forma que ya hemos estudiado a través de la interfaz Statement/PreparedStatement.

Podemos hacer uso del modificador IF NOT EXISTS dentro de nuestro código SQL para asegurarnos de que la base de datos se cree antes de iniciar los procesos CRUD de nuestra aplicación.

```
//soltar tabla Empleado si existe

String stSQLDrop = "DROP TABLE SI EXISTE Empleado"; //crear tabla
Empleado
Cadena stSQLCreate = "CREAR TABLA Empleado ("
    + "ID de impuesto      VARCHAR(9), "
    + "Nombre              VARCHAR(100), "
    + "Apellido            VARCHAR(100), "
    + "Salario              DECIMALES(9,2), "
    + "RESTRICCIÓN emp_tid_pk CLAVE PRIMARIA (ID de impuesto))";

String stSQLInsert = "INSERT INTO Empleado (NIF, Nombre, Apellido, Salario) VALOR + "('11111111A', 'José', 'Salcedo López',
    1279.90),"
    + "('22222222B', 'Juan', 'De la Fuente Arqueros', 1100.73),"
    + "('33333333C', 'Antonio', 'Bosch Jericó', 1051.45),"
    + "('44444444D', 'Ana', 'Sanchís Torres', 1300.02),"
    + "('55555555E', 'Isabel', 'Martí Navarro', 1051.45)";

//eliminar el primer empleado

String stSQLDelete1 = "ELIMINAR DEL Empleado DONDE TaxID='11111111A'"; //selecciona todos los
registros String stSQLSelect =
"SELECT * FROM Employee"; // Declaraciones y operaciones SQL

staSQLQuery.executeUpdate(stSQLDrop);
```

```
System.out.println(" Tabla eliminada (si existe) en la base de datos dada...");
staSQLquery.executeUpdate(stSQLCreate);
System.out.println(" Tabla creada en una base de datos
determinada...");
staSQLquery.executeUpdate(stSQLInsert); System.out.println("Tabla poblada con varios
registros en una base de datos determinada..."
staSQLquery.executeUpdate(stSQLDelete1); System.out.println("Se eliminó la primera tabla de registros
```

6.2. Ejemplo completo

Ejemplo completo de un proyecto Maven para acceder a una base de datos MySQL:

```
importar java.sql.*;

/**
 * =====
 *Ejemplo de acceso a una base de datos relacional MySQL con JDBC* @autor
 Abelardo Martínez
 * =====
 */

clase pública DBMySQL {

    /**
     * -----
     * VARIABLES Y CONSTANTES GLOBALES
     * -----
     */

    //Constantes de base de datos
    Cadena final estática DBUSER = "mavenuser"; //usuario static
    final String DBPASSWORD = "ada0486"; //contraseña //cadena para
    conectarse a MySQL static final
    String URL = "jdbc:mysql://localhost:3306/DBCompany?useSSL=false&use

    /**
     * -----
     * PROGRAMA PRINCIPAL
     * -----
     */

    principal vacío estático público ( cadena [] stArgs ) {
```

```
prueba {
    //establecemos la conexión con DBCompany
    Conexión cnDBCompany = DriverManager.getConnection(URL, DBUSER, DBPASSW
    Declaración staSQLQuery = cnDBCompany.createStatement();

    /*
     * -----
     * Sentencias SQL. Definición
     * -----
     */

    //soltar tabla Empleado si existe
    String stSQLDrop = "DROP TABLE SI EXISTE Empleado";

    //crear tabla Empleado
    Cadena stSQLCreate = "CREAR TABLA Empleado ("
        + "ID de impuesto          VARCHAR(9), "
        + "Nombre                  VARCHAR(100), "
        + "Apellido                 VARCHAR(100), "
        + "Salario                  DECIMALES(9,2), "
        + "RESTRICCIÓN emp_tid_pk CLAVE PRIMARIA (ID de impuesto));

    //insertar empleados
    String stSQLInsert = "INSERT INTO Empleado (ID de impuesto, Nombre, Apellido, S
        + "('11111111A', 'José', 'Salcedo López', 1279.90),"
        + "('22222222B', 'Juan', 'De la Fuente Arqueros', 1100.73),"
        + "('33333333C', 'Antonio', 'Bosch Jericó', 1051.45),"
        + "('44444444D', 'Ana', 'Sanchís Torres', 1300.02),"
        + "('55555555E', 'Isabel', 'Martí Navarro', 1051.45));

    //eliminar el primer empleado
    String stSQLDeleteFirst = "ELIMINAR DEL Empleado DONDE TaxID='11111111A'"

    //seleccionamos todos los registros
    String stSQLSelect = "SELECCIONAR * DEL Empleado";

    /*
     * -----
     * Sentencias SQL. Ejecución
     * -----
     */

    //declaraciones y operaciones SQL
    staSQLQuery.executeUpdate(stSQLDrop);

    System.out.println(" Tabla eliminada (si existe) en la base de datos dada..."
```

```

staSQLQuery.executeUpdate(stSQLCreate);
System.out.println(" Tabla creada en una base de datos determinada...");
staSQLQuery.executeUpdate(stSQLInsert);
System.out.println(" Tabla poblada con varios registros en una base de datos determinada
staSQLQuery.executeUpdate(stSQLDeleteFirst);
System.out.println("Se eliminó la primera tabla de registros en una base de datos
determinada..." //recupera datos y los muestra en pantalla. Sentencia de
selección SQL ResultSet rsEmployee = staSQLQuery.executeQuery(stSQLSelect);
while (rsEmployee.next()) {
    System.out.println("TaxID:" + rsEmployee.getString("TaxID"));
    System.out.println(" Nombre: " + rsEmployee.getString("Nombre")
    System.out.println(" Apellido: " + rsEmpleado.getString("Apellido")
    System.out.println("Salario: " + rsEmployee.getBigDecimal("Salario")

} //liberar recursos
rsEmployee.close(); //cerrar el conjunto de resultados
staSQLQuery.close(); //cerrar la declaración
cnDBCompany.close(); //cierramos la conexión a la BD
} catch (SQLException sqle)
{ sqle.printStackTrace(System.out);
}
}
}

```

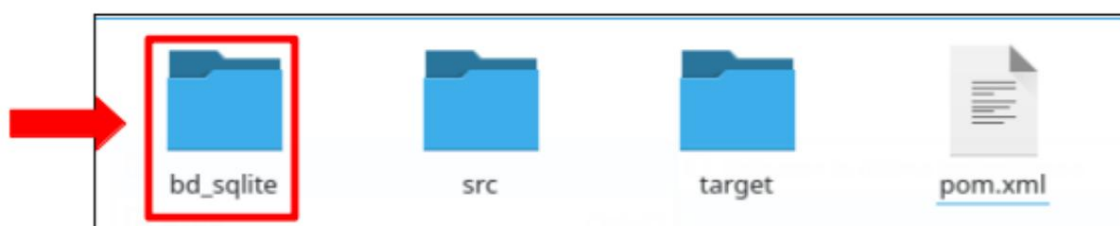
7. Cambiar a otro RDBMS

¿Qué pasa con el uso de otro RDBMS? Es tan fácil como:

1. Agregue sus dependencias al archivo POM.
2. Cambie la cadena de conexión.
3. ¡ESO ES TODO! No hay JAR que descargar, ni ruta que cambiar... ¡nada que configurar! maven lo hace todo.

7.1. Vaya más fácil con SQLite

En primer lugar tenemos que instalar SQLite. Si desea hacerlo más simple, no necesita instalar SQLite. Simplemente descargue un ejemplo de base de datos desde aquí: [https://www.sqlitetutorial.net / sqlite-sample-database/](https://www.sqlitetutorial.net/sqlite-sample-database/). Coloque el archivo .db dentro de la carpeta de su proyecto.



Probemos con SQLite siguiendo estos pasos:

1. Cree una nueva clase llamada DBSQLite.

New Java Class

Create a new Java class.

Source folder:

Package:

☐ Enclosing type:

Name:

Modifiers: ☒ public ☐ package ☐ private ☐ protected
☐ abstract ☐ final ☐ static

Superclass:

Interfaces:

Which method stubs would you like to create?

☐ public static void main(String[] args)

☐ Constructors from superclass

☒ Inherited abstract methods

Do you want to add comments? (Configure templates and default value [here](#))

☐ Generate comments

2. Busque dependencias:

- <https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc>
- Además, puede navegar en Internet para obtener más información. Por ejemplo, puedes escribir “dependencias SQLite del archivo pom”.

○

- <https://database.guide/check-sqlite-version/>

```
<project xmlns="http://maven.apache.org/POM/4.0.0"
esquemaLocation="http://maven.apache.org/POM/4.0.0 http://maven.apache.org/xsd/m
<modelVersion>4.0.0</modelVersion>

<groupId>com.simplilearn</groupId>
<artifactId>U2JDBCExample</artifactId>
<version>0.0.1-SNAPSHOT</version>

<name>U2JDBCExample</name>
<!-- FIXME cámbielo al sitio web del proyecto --> <url>http://
www.example.com</url>
```

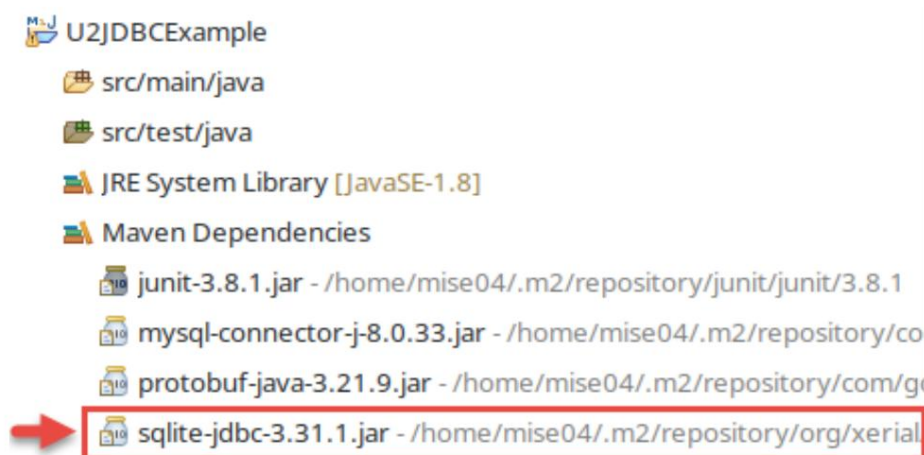
```
<propiedades>
  <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
  <maven.compiler.source>1.7</maven.compiler.source>
  <maven.compiler.target>1.7</maven.compiler.target>
</propiedades>

<dependencias>
  <dependencia>
    <groupId>junit</groupId>
    <artifactId>junit</artifactId>
    <versión>4.11</versión>
    <scope>prueba</scope>
  </dependencia>
  <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j --> <dependencia>

    <groupId>com.mysql</groupId>
    <artifactId>mysql-connector-j</artifactId>
    <versión>8.0.33</versión>
  </dependencia>
  <!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc --> <dependencia>

    <groupId>org.xerial</groupId>
    <artifactId>sqlite-jdbc</artifactId>
    <versión>3.31.1</versión>
  </dependencia>
</dependencias>
```

6. Ahora vaya a Proyecto → Limpiar y verifique los archivos JAR que su IDE descargará por usted.



En este punto, sólo tienes que cambiar la cadena de conexión y los detalles de autenticación en obtener conexión. En nuestro caso, no es necesario establecer un usuario y contraseña. Esto es lo que nosotros

consiguió:

```
importar java.sql.*;
```

```
/**
 * =====
 *
 * Ejemplo de acceso a una base de datos relacional MySQL con JDBC
 * @autor Abelardo Martínez. Basado y modificado de https://www.sqlitetutorial.net/
 * =====
 */
```

```
clase pública DBSQLite {
```

```
/**
 * =====
 *
 * VARIABLES Y CONSTANTES GLOBALES
 * =====
 */
```

```
//Constantes de base de datos
```

```
//cadena para conectarse a MySQL
```

```
URL de cadena final estática = "jdbc:sqlite:bd_sqlite\\chinook.db";
```

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

```

* PROGRAMA PRINCIPAL
* -----

*/

principal vacío estático público (cadena [] stArgs) {

    prueba {
        //establecemos la conexión con chinook.db
        Conexión cnDBChinook = DriverManager.getConnection(URL);
        System.out.println(" Se ha establecido la conexión a la base de datos.")
        Declaración staSQLquery = cnDBChinook.createStatement();

        /*
        * -----

        * Sentencias SQL. Definición
        * -----

        */

        //elimina la tabla ESTUDIANTES si existe
        String stSQLDrop = "BORRAR TABLA SI EXISTEN ESTUDIANTES";
        //crear tabla ESTUDIANTES
        String stSQLCreate = "CREAR TABLA ESTUDIANTES "
            + "(studentid INTEGER no NULL, "
            + " primero VARCHAR(255), "
            + " último VARCHAR(255), "
            + " edad ENTERA, "
            + " CLAVE PRIMARIA (idestudiante));"

        //insertar estudiantes
        String stSQLInsert = "INSERTAR EN ESTUDIANTES (iddeestudiante, nombre, apellido, edad)
            + "(1, 'PETER', 'KENIX', 38),"
            + "(2, 'XENA', 'HARRIS', 28)";

        //eliminar el primer estudiante
        String stSQLDeleteFirst = "ELIMINAR DE ESTUDIANTES DONDE Studentid=1"

        //seleccionamos todos los registros
        String stSQLSelect = "SELECCIONAR ID de estudiante, primero, último, edad DE ESTUDIANTES"

        /*
        * -----

        * Sentencias SQL. Ejecución
        * -----

        */
    }
}

```

```
// Declaraciones y operaciones SQL
staSQLquery.executeUpdate(stSQLDrop);
System.out.println("Tabla eliminada (si existe) en la base de datos dada...");
staSQLquery.executeUpdate(stSQLCreate);
System.out.println(" Tabla creada en la base de datos dada...");
staSQLquery.executeUpdate(stSQLInsert );
System.out.println(" Tabla poblada con varios registros en una base de datos determinada");
staSQLquery.executeUpdate(stSQLDeleteFirst);
System.out.println("Tabla de registros eliminada en una base de datos determinada..."); //
recupera datos y los muestra en la pantalla. Sentencia de selección SQL ResultSet
rsStudent = staSQLquery.executeQuery(stSQLSelect); while (rsStudent.next()) {

    System.out.println(" ID de estudiante:" + rsStudent.getInt("id de estudiante")
    System.out.println("Primero: " + rsStudent.getString("primero")); System.out.println("Último:
    " + rsStudent.getString("último")); System.out.println("Edad: " + rsStudent.getInt("edad"));

}

} //liberar recursos
rsStudent.close(); //cerrar el conjunto de resultados
staSQLquery.close(); //cerrar la declaración
cnDBChinook.close(); //cierramos la conexión a la BD
} captura (SQLException sqle) {
    sqle.printStackTrace(System.out);
}
}
}
```

8. Bibliografía

Fuentes

- Cómo crear un proyecto Maven en Eclipse. <https://www.simplilearn.com/tutorials/maven-tutorial/proyecto-maven-en-eclipse>
- JDBC: ejemplo de creación de tabla. <https://www.tutorialspoint.com/jdbc/jdbc-create-tables.htm>
- Josep Cañellas Bornas, Isidre Guixà Miranda. Acceso a datos. Desarrollo de aplicaciones multiplataforma. Creative Commons. Departamento de Enseñanza, Institut Obert de Catalunya. Depósito legal: B. 29430-2013. <https://ioc.xtec.cat/educacio/recursos>
- Alberto Oliva Molina. Acceso a datos. UD 2. Manejo de conectores. IES Tubalcaín. Tarazona (Zaragoza, España).
- Escuelas W3. RDBMS MySQL. https://www.w3schools.com/mysql/mysql_rdbms.asp
- Conexión a MySQL mediante la interfaz JDBC DriverManager. <https://dev.mysql.com/doc/connector-j/8.1/en/connector-j-usagenotes-connect-drivermanager.html>
- Punto de tutorías. JDBC: ejemplo de inserción de registros. <https://www.tutorialspoint.com/jdbc/jdbc-insertar-registros.htm>
- 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/)