GENERALITAT VALENCIANA
Conselleria d'Educació, Cultura i Esport

ceedcv CENTRE ESPECÍFIC D'EDUCACIÓ A DISTÀNCIA DE LA COMUNITAT VALENCIANA

Unit 2. ACCESS TO DATABASES

Part 1. Working with Relational Databases

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

CFGS Desarrollo de Aplicaciones Multiplataforma (DAM)

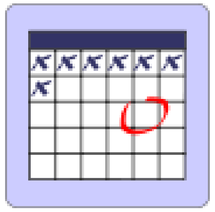
Abelardo Martínez

Year 2023-2024

Credits

- Notes made by Abelardo Martínez.
- Based and modified from Sergio Badal (www.sergiobadal.com).
- The images and icons used are protected by the LGPL licence and have been obtained from:
  - https://commons.wikimedia.org/wiki/Crystal_Clear
  - https://www.openclipart.org

Unit progress



FROM	UNIT	WEEKNO.	DESCRIPTION	ASSESSABLE TASKS
	UNIT 1: ACCESS TO FILES			
18/09/23	UNIT 1	WEEK 1	Introduction, Java review, IDE installation and basics of Java	
25/09/23	UNIT 1	WEEK 2	Files and folders. Class File: methods, exceptions. File types	
02/10/23	UNIT 1	WEEK 3	Access types. Reading and writing operations	
09/10/23	UNIT 1	WEEK 4	Files: XML/XSL	
16/10/23	UNIT 1	WEEK 5	Files: XML/XSL	AT1.PRESENTATION
	UNIT 2: ACCESS TO DATABASES			
23/10/23	UNIT 2	WEEK 1	ACCESS TO RELATIONAL DBS	AT2.PRESENTATION
30/10/23	UNIT 2	WEEK 2	ACCESS TO NON RELATIONAL DBS	AT1.SUBMISSION
06/11/23	UNIT 2	WEEK 3	UNIT 1 AND UNIT 2 REVIEW	AT2.SUBMISSION
13/11/23	CONTENTS REVIEW			

Contents

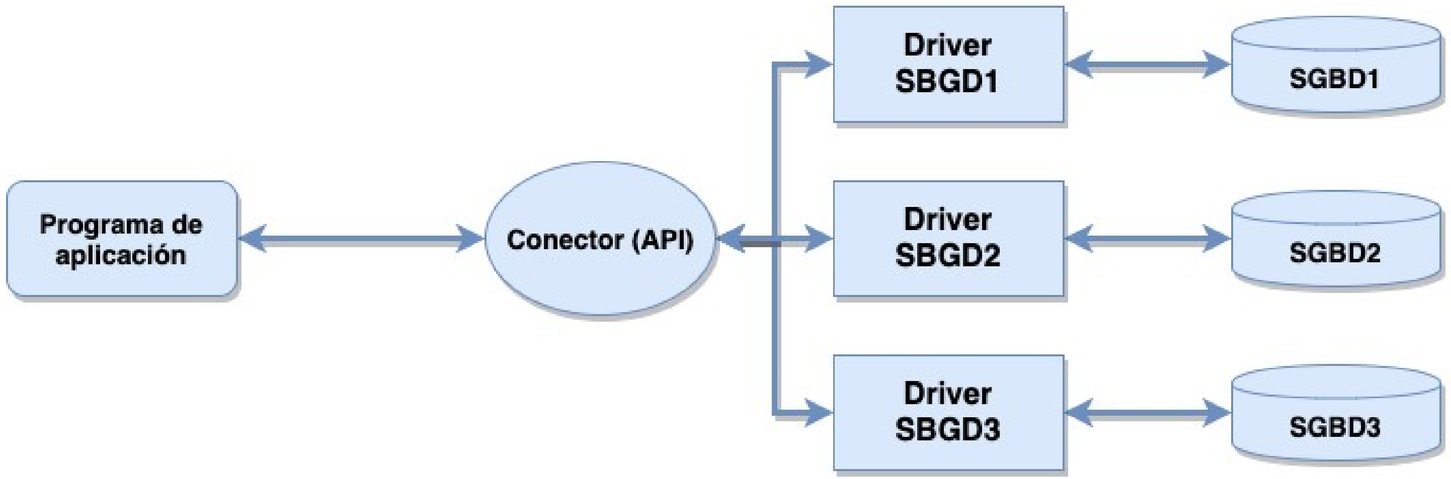
1. CONNECTORS

Connectors

- Database management systems (DBMSs) of different types have their own specialised languages for dealing with the data they store.
- Application programs are written in general purpose programming languages, such as Java.
- In order for these programs to operate with DBMSs, mechanisms are needed to allow the application programs to communicate with the databases in these languages.
- These are implemented in an **API** and are called **connectors**.

Connectors for RDBMS

- To work with relational database management systems (RDBMS), the **SQL language** is used.
- Each RDBMS uses its own version of SQL, with its own peculiarities, which is why we must use its own low-level structures.
- The use of drivers allows the development of a generic architecture in which the connector has a common interface for both the applications and the databases, and the drivers take care of the particularities of the different databases.



Connectors for RDBMS

- In this way, the connector is not just an API, but an architecture, because it specifies interfaces that the different drivers have to implement to access the databases.
- There are different architectures in this sense:
  - ODBC
  - OLE-DB
  - ADO
  - JDBC
- **ODBC** and **JDBC** are the predominant ones today, as almost all DBMSs implement them and allow working with them.
- The benefits provided by driver-based connectors come at the cost of increased programming complexity.

ODBC and JDBC

ODBC (Open Database Connectivity)

- Defines an API that applications can use to open a connection to the database, send queries, updates, etc.
- Applications can use this API to connect to any compatible database server.
- It is written in C language.

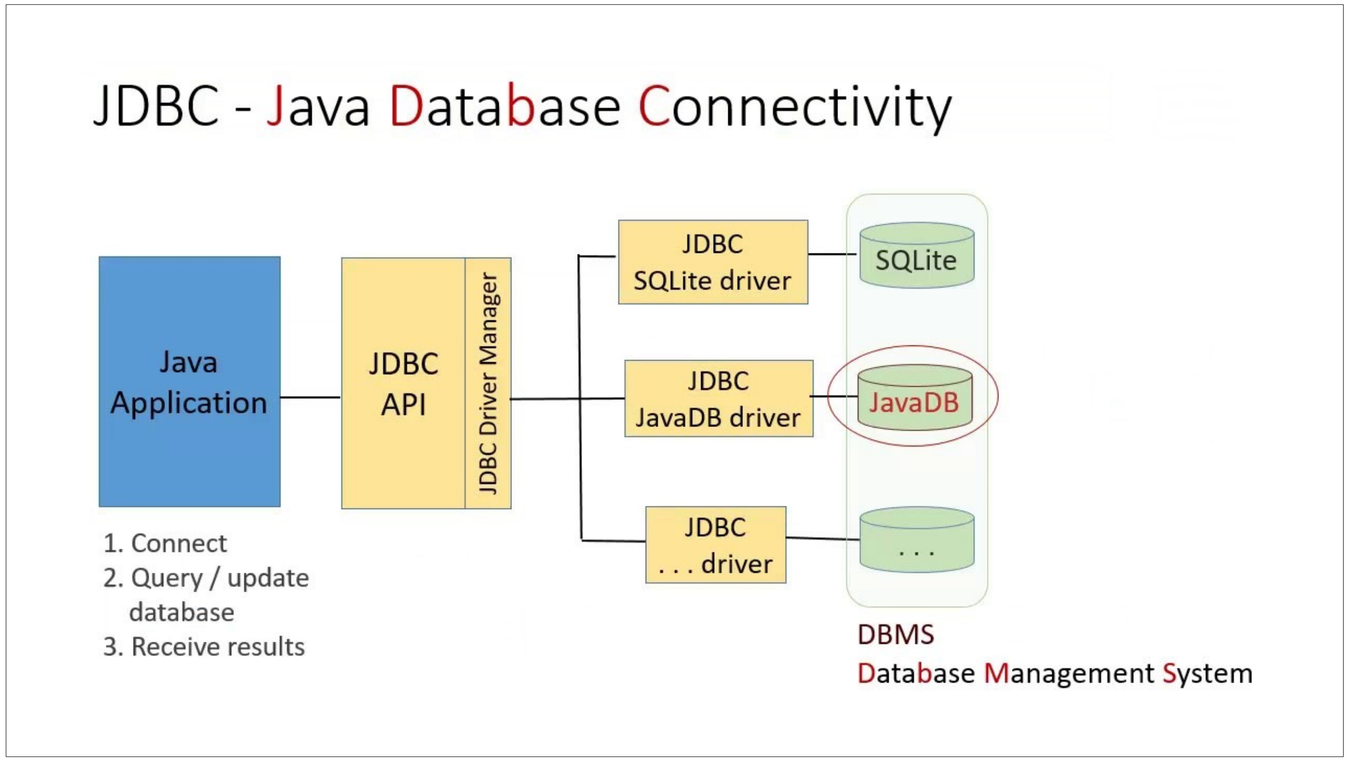
JDBC (Java Database Connectivity)

- This is the quintessential API for connecting databases to Java applications and is the one we will focus on in this unit.

2. WHAT IS JDBC?

What is JDBC?

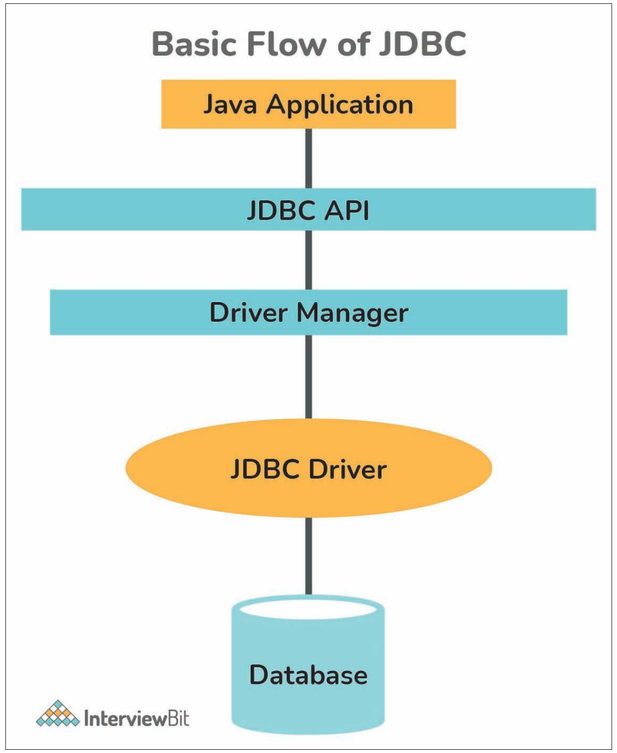
- JDBC provides an API for accessing SQL relational database oriented data sources.
- JDBC has a different interface for each DBMS, this is called **Driver**.
- It provides an architecture that allows vendors to create drivers that allow Java applications to access data.
- This allows calls to Java methods to correspond to the database API.



DB access via JDBC

JDBC consists of a set of interfaces and classes that allow us to write Java applications to manage the following tasks with a relational database:

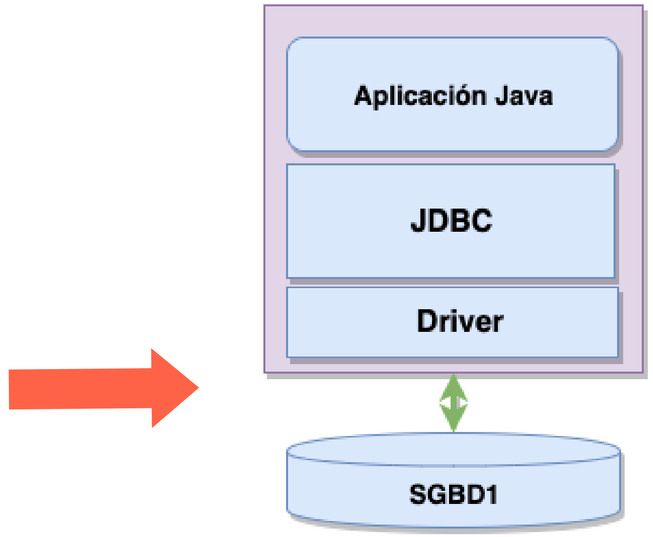
- Connect to a database.
- Send queries and update instructions to a database.
- Retrieve and process the results received from the database in response to those queries.



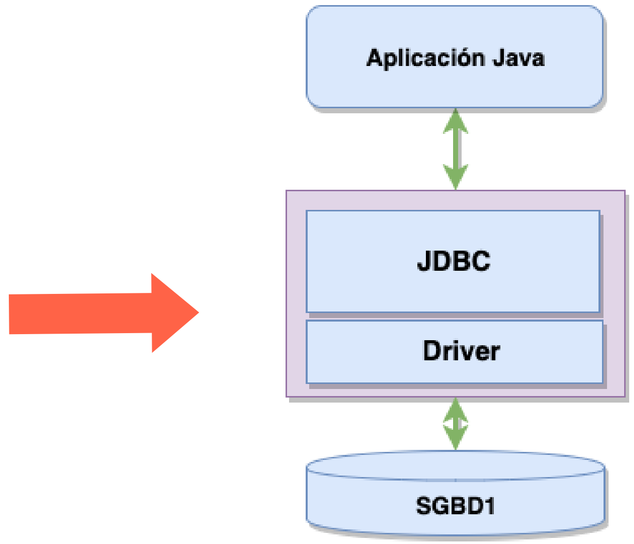
JDBC. Access models

The JDBC API supports two access models:

1) **Two-layer model** → The Java application talks directly to the database. This requires a JDBC driver residing in the same space as the application.
   - SQL statements are sent from the Java program to the DBMS for processing and the DBMS sends the results back to the program.
   - The DBMS can be located in the same or in another machine and the requests are made through the network.
   - The driver is in charge of managing the connections transparently to the programmer.



2) **Three-layer model** → Commands are sent to an intermediate layer which is responsible for sending the SQL commands to the database and collecting the results.



JDBC. Types of drivers

Type	Description
JDBC-ODBC Bridge	Enables JDBC DBMS access via ODBC protocol.
Native	Controller written partly in Java and in native database code. Translates Java API calls into DBMS calls.
Network	Pure Java driver that uses a network protocol to communicate with the database server.
Thin	Pure Java driver with native protocol. Translates API calls to native calls of the network protocol used by the DBMS.

JDBC. Classes and interfaces

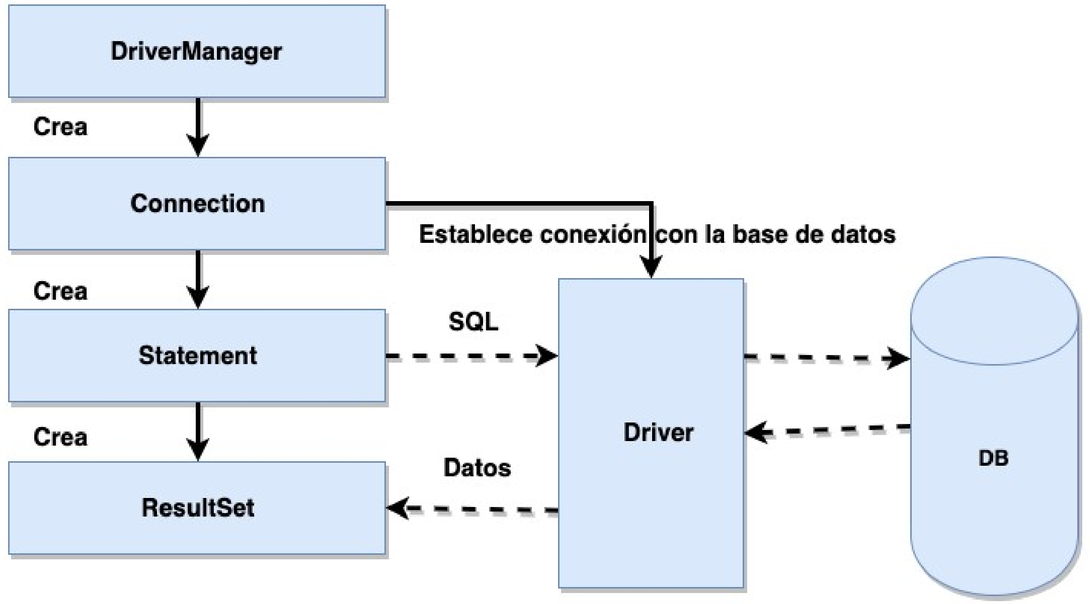
JDBC defines **classes and interfaces** in the java.sql package. The following table shows the most relevant ones:

Class or interface	Description
Driver	Allows to connect to a database.
DriverManager	Allows you to manage the drivers installed in the system.
DriverPropertyInfo	Provides driver information.
Connection	Represents a connection to the database.
DatabaseMetadata	Provides information from the database.
Statement	Allows SQL statements to be executed without parameters.
PreparedStatement	Allows to execute SQL statements with parameters.
CallableStatement	Allows SQL statements with input and output parameters to be executed.
ResultSet	Contains the result of the queries.
ResultSetMetadata	Allows to get information from a ResultSet.

JDBC. Operation steps

The operation of a JDBC program follows the following steps:

- Import the necessary classes
- Configure POM file to load the JDBC Driver
- Identify the data source
- Create a Connection object
- Create a Statement object
- Execute the query with the Statement object
- Retrieve the result in a ResultSet
- Release the ResultSet object
- Release the Statement object
- Release the Connection object



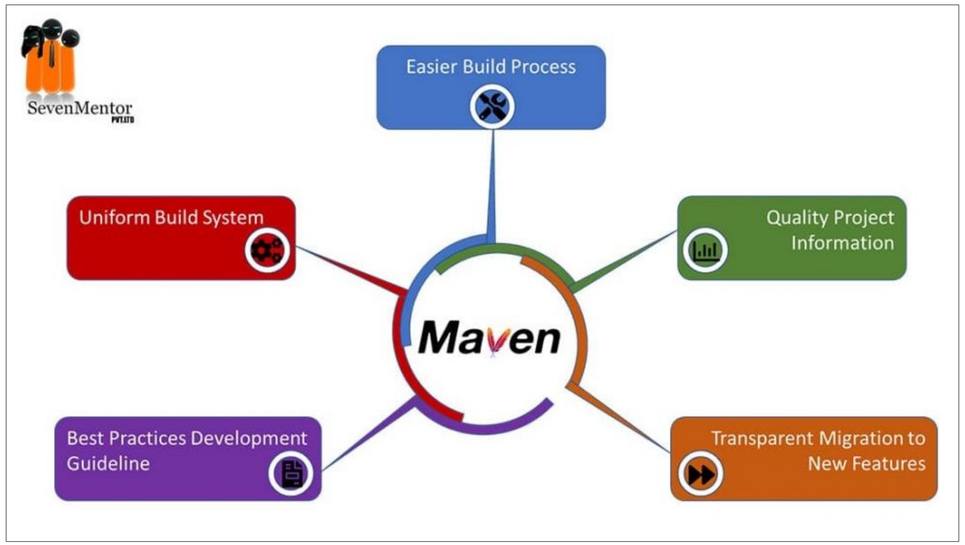
3. WHAT IS MAVEN?

What is Maven?

Maven has been an open source project under Apache since 2003. Maven is very stable and feature-rich, providing numerous plugins that can do anything from generate PDF versions of your project's documentation to generating a list of recent changes from your SCM. And all it takes to add this functionality is a small amount of extra XML or an extra command line parameter.

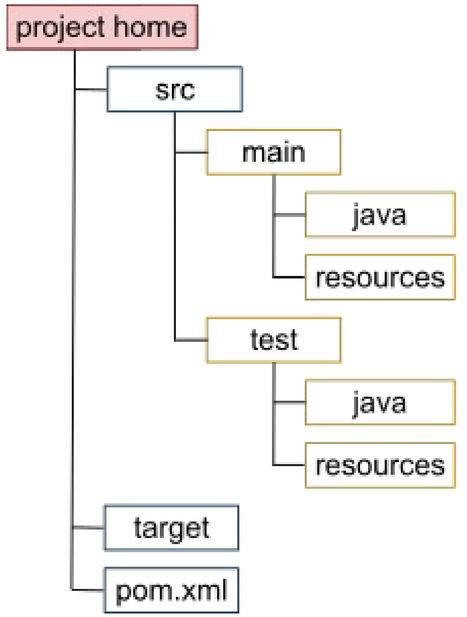
Have a lot of dependencies? No problem. **Maven connects to remote repositories** (or you can set up your own local repos) **and automatically downloads all of the dependencies needed to build your project**.

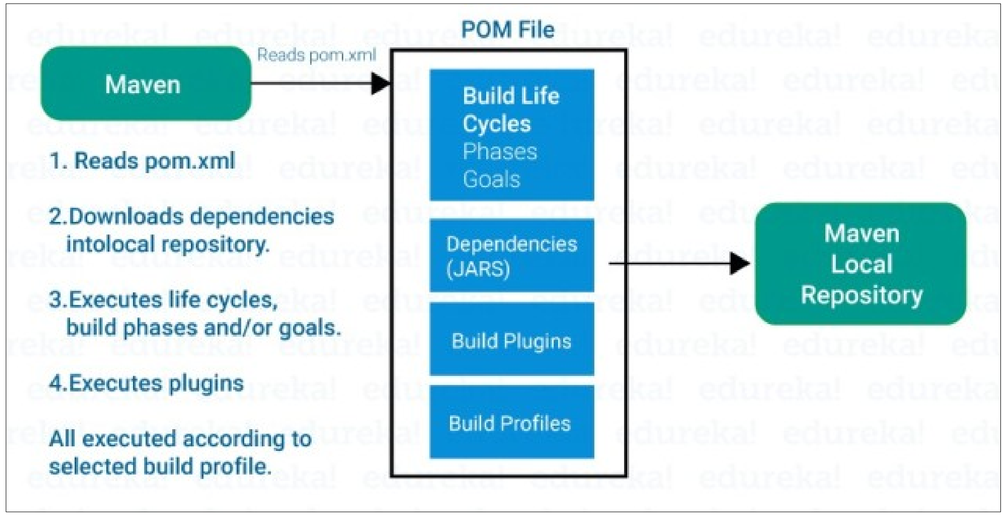
Source: https://stackabuse.com/search/?q=maven



How Maven works?

- It is based on a central file, **pom.xml - the Project Object Model (POM) written in XML-** where you define everything your project needs. **Maven manages the project dependencies, compiles, packages and runs the tests**.
- Through plugins, it allows you to do much more, such as generating Hibernate maps from a database, deploying the application, etc.
- Before Maven, you had to manually download the jar that you needed in your project and copy them manually in the classpath.
- With Maven **you only need to define in your pom.xml the dependencies** and Maven downloads them and adds them to the classpath.





Maven. POM file example

- In this example we define the POM file to access to a MySQL database.

```xml
<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
    https://maven.apache.org/xsd/maven-4.0.0.xsd">
    <modelVersion>4.0.0</modelVersion>
    <groupId>mx.com.gm.jdbc</groupId>
    <artifactId>ADA.U2W1.ExampleJDBC.Eclipse</artifactId>
    <version>0.0.1-SNAPSHOT</version>
    <dependencies>
        <dependency>
            <groupId>mysql</groupId>
            <artifactId>mysql-connector-java</artifactId>
            <version>8.0.20</version>
        </dependency>
        <dependency>
            <groupId>org.xerial</groupId>
            <artifactId>sqlite-jdbc</artifactId>
            <version>3.8.7</version>
        </dependency>
    </dependencies>
</project>
```

Source: https://stackabuse.com/search/?q=maven

4. CONNECTING TO THE DATABASE

Step 1. Create the Database

Let's see an example of a MySQL database connection and query.

1. Install MySQL: https://dev.mysql.com/doc/mysql-installation-excerpt/8.0/en/
2. Use the provided code ahead to create a new database called testDB and user called mavenUser with all privileges (to make it easier) and password maven1234X.
3. We'll use the following database schema.
   - DB Name → Company
   - Table Name → Employees
   - Fields:
     - Tax ID Varchar(9) PK
     - First Name Varchar(100)
     - Last Name Varchar(100)
     - Salary Decimal (9,2)
4. Add five random people to that database/table.

Step 1. Create the Database

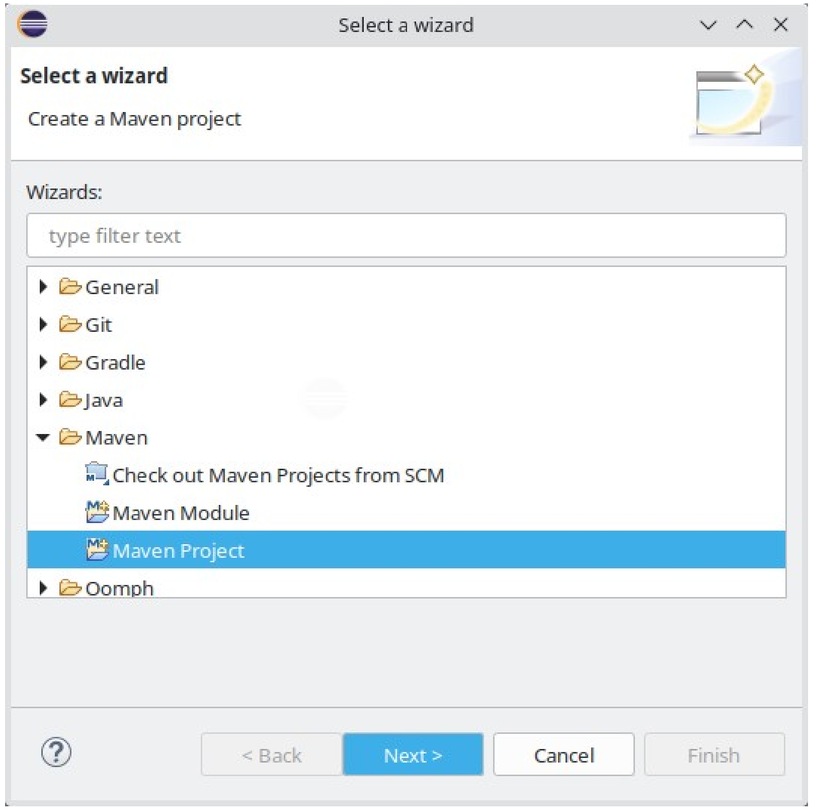
This could be a good example of how to achieve those steps:

```
-- Create database https://matomo.org/faq/how-to-install/faq_23484/
CREATE DATABASE DBCompany;

-- Create a user if you are using MySQL 5.7 or MySQL 8 or newer
CREATE USER 'mavenuser'@'localhost' IDENTIFIED WITH mysql_native_password BY 'ada0486';
-- Or if you are using an older version such as MySQL 5.1, 5.5, 5.6:
-- CREATE USER 'mavenuser'@'localhost' IDENTIFIED BY 'ada0486';
GRANT ALL PRIVILEGES ON DBCompany.* TO 'mavenuser'@'localhost';

-- Select the database to use
USE DBCompany;

-- Create the employee's table
CREATE TABLE Employee (
   TaxID        VARCHAR(9),
   Firstname    VARCHAR(100),
   Lastname     VARCHAR(100),
   Salary       DECIMAL(9,2),
   CONSTRAINT emp_tid_pk PRIMARY KEY (TaxID)
);
-- Insert random employees
INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES ('11111111A', 'José', 'Salcedo López', 1279.90);
INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES ('22222222B', 'Juan', 'De la Fuente Arqueros', 1100.73);
INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES ('33333333C', 'Antonio', 'Bosch Jericó', 1051.45);
INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES ('44444444D', 'Ana', 'Sanchís Torres', 1300.02);
INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES ('55555555E', 'Isabel', 'Martí Navarro', 1051.45);
```

Step 2. Create the project within the IDE

The first step is to open Eclipse, which comes with the integrated Maven environment.

- Go to the File menu, option **New → Project**.
- Select the **Maven Project** option.
- We follow the rest of the steps explained in the extended notes.
- Finally, we select **Project → Clean** on our project so the necessary libraries and files have been downloaded correctly.



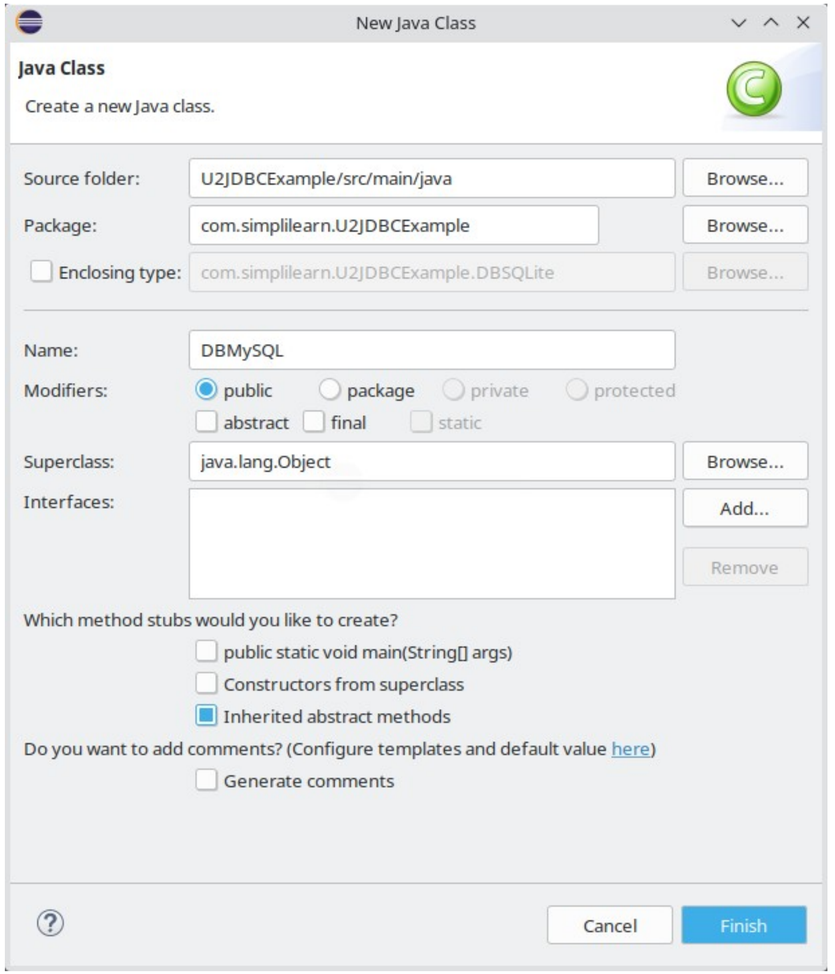
Step 3. Add the dependency to the POM file

```xml
<?xml version="1.0" encoding="UTF-8"?>
<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.simplilearn</groupId>
  <artifactId>U2JDBCExample</artifactId>
  <version>0.0.1-SNAPSHOT</version>
  <name>U2JDBCExample</name>
  <!-- FIXME change it to the project's website -->
  <url>http://www.example.com</url>
  <properties>
    <project.build.sourceEncoding>UTF-8</project.build.sourceEncoding>
    <maven.compiler.source>1.7</maven.compiler.source>
    <maven.compiler.target>1.7</maven.compiler.target>
  </properties>
  <dependencies>
    <dependency>
      <groupId>junit</groupId>
      <artifactId>junit</artifactId>
      <version>4.11</version>
      <scope>test</scope>
    </dependency>
    <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
    <dependency>
        <groupId>com.mysql</groupId>
        <artifactId>mysql-connector-j</artifactId>
        <version>8.0.33</version>
    </dependency>
  </dependencies>
```

Step 4. Create a class and set the connection string

- We create a class called DBMySQL.
- We have to build our connection string for MYSQL (where DBNAME corresponds to the database name):

URL="
jdbc:mysql://localhost:3306/**DBNAME**
?useSSL=false
&useTimezone=true
&serverTimezone=UTC
&allowPublicKeyRetrieval=true
";



Step 5. Connect to the database

Establish the connection

The connection to the DB is established using the DriverManager class and the getConnection method passing as parameters URL, USER and PASSWORD of the DB.

```
//DB constants
static final String DBUSER = "mavenuser"; //user
static final String DBPASSWORD = "ada0486"; //password
//string to connect to MySQL
static final String URL =
 "jdbc:mysql://localhost:3306/DBCompany?useSSL=false&useTimezone=true&serverTimezone=UTC&allowPublicKeyRetrieval=true";
```

```
//establish the connection to DBCompany
Connection cnDBCompany = DriverManager.getConnection(URL, DBUSER, DBPASSWORD);
```

5. DQL QUERIES

Execute DQL statements

The Data Query Language is the sub-language responsible for reading, or querying, data from a database. In SQL, it corresponds to the **SELECT**.

We can perform queries through the **Statement** interface.

1. To obtain a Statement object, the createStatement() method of a valid Connection object is called.
2. The Statement object has the executeQuery() method that executes a query in the database. This method receives as parameter a String that must contain a SQL statement.
3. The result is collected in a ResultSet object, this object is a sort of array or list that includes all the data returned by the DBMS.

```
Statement staSQLquery = cnDBCompany.createStatement();
//select all the records
String stSQLSelect = "SELECT * FROM Employee";
//retrieve data and display it on screen. SQL select statement
ResultSet rsEmployee = staSQLquery.executeQuery(stSQLSelect);
```

Scroll through the results

Once we have obtained the results of the query and we have stored them in our ResultSet we only have to traverse them and treat them as we see fit.

a) ResultSet has implemented a pointer that points to the first element of the list.

b) By means of the next() method, the pointer advances to the next element.

- The next() method, in addition to advancing to the next record, returns true if there are more records and false if it has reached the end.
- The getString() and getDouble() methods are used to obtain the values of the different columns. These methods receive the name of the column as a parameter.

```
while (rsEmployee.next()) {
   System.out.println("TaxID:" + rsEmployee.getString("TaxID"));
   System.out.println("First name: " + rsEmployee.getString("Firstname"));
    System.out.println("Last name: " + rsEmployee.getString("Lastname"));
    System.out.println("Salary: " + rsEmployee.getBigDecimal("Salary"));
}
```

Full example

```
import java.sql.*;
/**
 * ======================================================================
 * Example of accessing a MySQL relational database with JDBC
 * @author Abelardo Martínez
 * ======================================================================
 */
public class DBMySQL {
    //DB constants
    static final String DBUSER = "mavenuser"; //user
    static final String DBPASSWORD = "ada0486"; //password
    //string to connect to MySQL
    static final String URL = "jdbc:mysql://localhost:3306/DBCompany?useSSL=false&useTimezone=true&serverTimezone=UTC&allowPublicKeyRetrieval=true";

    public static void main( String[] stArgs )
  {
        try {
           //establish the connection to DBCompany
            Connection cnDBCompany = DriverManager.getConnection(URL, DBUSER, DBPASSWORD);
            Statement staSQLquery = cnDBCompany.createStatement();
            //select all the records
            String stSQLSelect = "SELECT * FROM Employee";
            //retrieve data and display it on screen. SQL select statement
            ResultSet rsEmployee = staSQLquery.executeQuery(stSQLSelect);
            while (rsEmployee.next()) {
                System.out.println("TaxID:" + rsEmployee.getString("TaxID"));
                System.out.println("First name: " + rsEmployee.getString("Firstname"));
                System.out.println("Last name: " + rsEmployee.getString("Lastname"));
                System.out.println("Salary: " + rsEmployee.getBigDecimal("Salary"));
     }
            //release resources
            rsEmployee.close(); //close the ResultSet
            staSQLquery.close(); //close the statement
            cnDBCompany.close(); //close the connection to the DB
        } catch (SQLException sqle) {
            sqle.printStackTrace(System.out);
    }
  }
}
```

```
TaxID:11111111A
First name: José
Last name: Salcedo López
Salary: 1279.90
TaxID:22222222B
First name: Juan
Last name: De la Fuente Arqueros
Salary: 1100.73
TaxID:33333333C
First name: Antonio
Last name: Bosch Jericó
Salary: 1051.45
TaxID:44444444D
First name: Ana
Last name: Sanchís Torres
Salary: 1300.02
TaxID:55555555E
First name: Isabel
Last name: Martí Navarro
Salary: 1051.45
```

6. DML QUERIES

Execute DML sentences

In SQL, DML statements are those that allow us to add, edit or delete data in a database.

- **INSERT**. To insert data into a table.
- **UPDATE**. To modify existing data within a table.
- **DELETE**. To delete all the records in the table; it does not delete the spaces assigned to the records.

To execute any of these statements we must use objects:

- **Statement** → Allows you to execute SQL statements without parameters.
- **PreparedStatement** → Execute SQL statements with parameters.

Example

Insert and delete sentences. Definition:

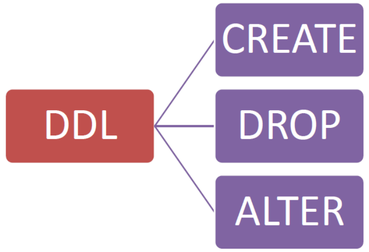
```
//insert employees
String stSQLInsert = "INSERT INTO Employee (TaxID, Firstname, Lastname, Salary) VALUES "
       + "('11111111A', 'José', 'Salcedo López', 1279.90),"
       + "('22222222B', 'Juan', 'De la Fuente Arqueros', 1100.73),"
       + "('33333333C', 'Antonio', 'Bosch Jericó', 1051.45),"
       + "('44444444D', 'Ana', 'Sanchís Torres', 1300.02),"
       + "('55555555E', 'Isabel', 'Martí Navarro', 1051.45)";
//delete first employee
String stSQLDeleteFirst = "DELETE FROM Employee WHERE TaxID='11111111A'";
```

Insert and delete sentences. Execution:

```
staSQLquery.executeUpdate(stSQLInsert);
System.out.println("Populated table with several records in given database...");
staSQLquery.executeUpdate(stSQLDeleteFirst);
System.out.println("Deleted first record table in given database...");
```

7. DDL QUERIES

Execute DDL sentences



DDL (**D**ata **D**efinition **L**anguage) is the sub-language responsible for defining the way in which data are structured in a database.

Although the bulk of the operations that are carried out from a client application against a database are data manipulation operations, there may be cases in which we are forced to carry out definition operations:

- We have just installed an application on a new computer and we need to create, in an automated way, a local database for its operation (the most common case).
- After a software update, the structure of the DB has changed and we need to update it from our Java application.
- We do not know the structure of the DB and we want to make dynamic queries on it.

Traditional DDL statements are still SQL statements and can therefore be executed in the way we have already studied through the **Statement/PreparedStatement** interface.

Example

We can make use of the IF NOT EXISTS modifier within our SQL code to ensure that the database is created before starting the CRUD processes of our application.

```
//drop table Employee if exists
 String stSQLDrop = "DROP TABLE IF EXISTS Employee";
//create table Employee
 String stSQLCreate = "CREATE TABLE Employee ("
        + "TaxID    VARCHAR(9), "
        + "Firstname    VARCHAR(100), "
        + "Lastname     VARCHAR(100), "
        + "Salary       DECIMAL(9,2), "
        + "CONSTRAINT emp_tid_pk PRIMARY KEY (TaxID))";
```

DDL sentences. Execution:

```
//SQL statements and operations
staSQLquery.executeUpdate(stSQLDrop);
System.out.println("Dropped table (if exists) in given database...");
staSQLquery.executeUpdate(stSQLCreate);
System.out.println("Created table in given database...");
```

8. SWITCHING RDBMS

What about using another RDBMS?

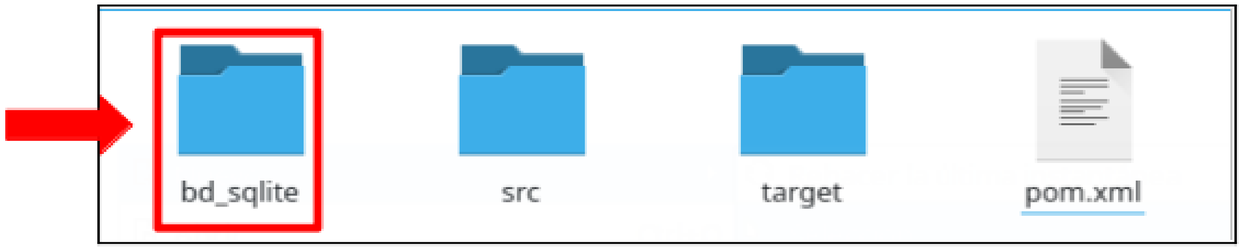
It's as easy as:

1) Add its dependencies to the POM file
2) Change the connection string

THAT'S ALL! No JAR to download, no path to change… nothing to set! **Maven does it all**.

Switching to SQLite

- Install SQLite. https://www.sqlite.org/download.html
- But, if you want to make it simpler, you don’t need to install SQLite. Just download a database example from here: https://www.sqlitetutorial.net/sqlite-sample-database/
- Place the .db file inside your project folder.



Add dependencies to the POM file

- Go to the Maven repository: https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc
- Check the SQLite version you have installed. https://database.guide/check-sqlite-version/
- Copy and paste the dependency code and add it to your pom.xml file inside a new node called <dependencies>, setting the proper version inside <version> node.
- Your POM file should be like this:

```
<dependencies>
   <dependency>
     <groupId>junit</groupId>
     <artifactId>junit</artifactId>
     <version>4.11</version>
     <scope>test</scope>
   </dependency>
  <!-- https://mvnrepository.com/artifact/com.mysql/mysql-connector-j -->
  <dependency>
     <groupId>com.mysql</groupId>
     <artifactId>mysql-connector-j</artifactId>
     <version>8.0.33</version>
  </dependency>
  <!-- https://mvnrepository.com/artifact/org.xerial/sqlite-jdbc -->
  <dependency>
      <groupId>org.xerial</groupId>
      <artifactId>sqlite-jdbc</artifactId>
      <version>3.31.1</version>
  </dependency>
 </dependencies>
```

Connection string

- We take advantage of the previously created class.
  - Create a new package and class called DBSQLite.
  - Copy & paste the code of the DBMySQL class.
- Finally, we change the connection string.
- For example:

```
//DB constants
//string to connect to MySQL
static final String URL = "jdbc:sqlite:bd_sqlite\\chinook.db";

/**
 * ------------------
 * MAIN PROGRAMME
 * ------------------
 */
public static void main(String[] stArgs) {

      try {
        //establish the connection to chinook.db
          Connection cnDBChinook = DriverManager.getConnection(URL);
```

9. ACTIVITIES FOR NEXT WEEK

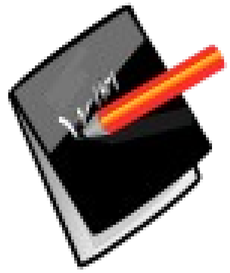
Proposed activities

Check the suggested exercises you will find at the “Aula Virtual”. **These activities are optional and non-assessable but** understanding these non-assessable activities is essential to solve the assessable task ahead.

Shortly you will find the proposed solutions.

10. BIBLIOGRAPHY

Resources



- Josep Cañellas Bornas, Isidre Guixà Miranda. Accés a dades. Desenvolupament d’aplicacions multiplataforma. Creative Commons. Departament d’Ensenyament, Institut Obert de Catalunya. Dipòsit 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).