

JAVA TECHNOLOGY

(503111)

LAB 2

LAB OBJECTIVES

Use the Java Database Connectivity API (JDBC) - a JAVA standard library that allows connecting and working with database systems. The Microsoft SQL Server database is used for example purposes in this lab (other database systems can also be used in a similar way). After completing these exercises, students will know how to use JDBC to connect and interact with mssql databases, perform basic operations such as reading data, adding/inserting data, updating or delete data in the database.

JDBC SETUP

To use the JDBC API, we first need to download the driver corresponding to the database system we want to use. These drivers are usually in the form of *.jar files, they can be downloaded at the database vendor's website (Microsoft, Oracle, Mysql, etc).

- Once you have downloaded the *.jar files, you need to include them in the classpath of your java project. There are different ways to do this, if you work with Eclipse you can create a folder named **libs**, then copy the *.jar libraries into this folder and add them to the classpath by selecting the library , right-click and choose *Build Path --> Add to Build Path* (Figure 1).
- The second way to include the jdbc library to the project is to use the Maven dependency. To do this, you need to create a java maven project (on Eclipse or IntelliJ Idea) then copy the <dependency> script of the jdbc version corresponding to the database into the pom.xml file (Figure 2).

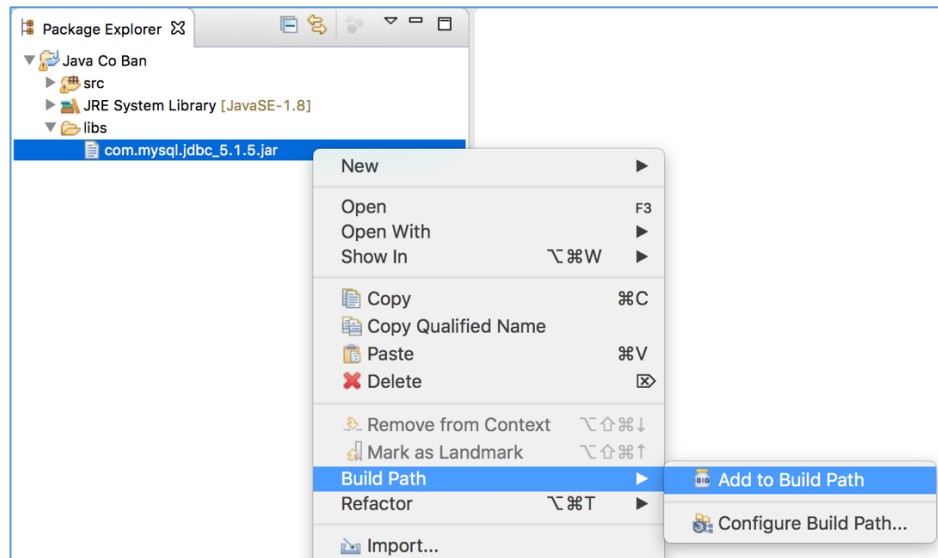


Figure 1. Add *.jar files to classpath of a java project using Eclipse IDE

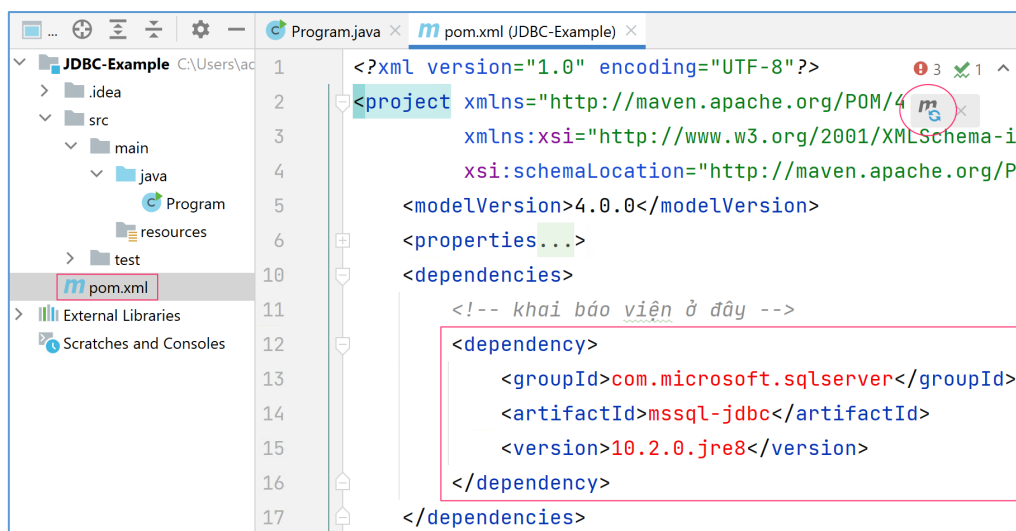


Figure 2. Setup mssql-jdbc dependency in a maven project with IntelliJ Idea tool

<https://mvnrepository.com/artifact/com.microsoft.sqlserver/mssql-jdbc>

USING JDBC API

Steps to connect and interact with a database in JDBC

1. Create and connect to database server
2. Make queries
3. Close connection

1. Create and connect to database server

To connect to a database server in JDBC, a connection string needs to be specified. This connection string contains important information of the server such as: hostname/ip of the server, username and password to login to the server, the name of the database to work with and some other necessary information.

Some examples of connection strings:

RDBMS	JDBC driver name	URL format
MS SQL	com.microsoft.sqlserver.jdbc.SQL ServerDriver	jdbc:sqlserver:// hostname;user=admin;password=111111;databaseName=Lab02;encrypt=true;trustServerCertificate=true;
MySQL	com.mysql.jdbc.Driver	jdbc:mysql:// hostname/dbname? useSSL=false
ORACLE	oracle.jdbc.driver.OracleDriver	jdbc:oracle:thin:@ hostname:port Number:databaseName
DB2	COM.ibm.db2.jdbc.net.DB2Driver	jdbc:db2: hostname:port Number/databaseName
Sybase	com.sybase.jdbc.SybDriver	jdbc:sybase:Tds: hostname: port Number/databaseName

Assuming microsoft sql server is selected to make the connection, then the program to connect to mssql server will be written similar to the following:

```
String url = "jdbc:sqlserver://507-20;user=admin;password=111111;"
            + "databaseName=Lab02;encrypt=true;"
            + "trustServerCertificate=true;";

try {
    Connection conn = DriverManager.getConnection(url);
    System.out.println("Kết nối thành công");

    // thực hiện truy vấn ở đây (đọc, thêm, xóa, cập nhật)

    conn.close();
} catch (SQLException e) {
    System.out.println("Lỗi kết nối: " + e.getMessage());
    e.printStackTrace();
}
```

In case of successful connection, nothing happens. If the connection fails, an exception with details will be printed to the console.

2. Make queries

2.1 Query does not return results

Queries that do not return results can have many different cases. They are sql statements that do not contain the select keyword. Most of these statements make some changes on the server side e.g. adding data, deleting data or updating data.

These queries may or may not contain input parameters, such as a create table statement, a table delete command, etc. With these statements, the [Statement](#) object is used to execute the statement as shown below.

```
// tạo và thực hiện kết nối
Connection conn = (Connection) DriverManager.getConnection(url, username, password);

String sql = "Create table student(name nvarchar(30) primary key, age int)";
Statement stm = (Statement) conn.createStatement(); // tạo statement
stm.executeUpdate(sql); // thực thi câu lệnh

stm.close();
conn.close();
```

Execute a statement with no parameters and no results with a Statement object

2.2 Check the result of the query

For sql statements accept parameters, we use `PreparedStatement` to execute the statement. Inserting data into a table can be classified to statements that does not return data. However, in fact we still need to know the status of the statement execution. More specifically, we need to know how many lines (records) have been added/removed/updated by the last statement. This can be checked through the result returned from the Prepared Statement object's `executeUpdate()` method.

```
String sql = "insert into student values(?,?)";
PreparedStatement ptm = (PreparedStatement) conn.prepareStatement(sql);

ptm.setString(1, "Le Minh");
ptm.setInt(2, 23);

int rows = ptm.executeUpdate();

if (rows == 1) {
    System.out.println("Them thanh cong");
}
```

Execute a statement with arguments using a Prepared Statement object. Although the statement does not return the data of the table, we also need to know how many rows have been added/deleted/updated as a result

3.3 Queries return data

For queries that return data (mostly select statements), we can use Statement or Prepared Statement, depending on whether the statement contains parameters or not. In both cases, the `executeQuery()` method needs to be called to execute the statement and get the data through the ResultSet object.

ResultSet is an object that contains references to data lines in the result of sql statement execution. Each time we read a stream of data by calling the `.next()` method.

```
String sql = "select * from student";
Statement stm = (Statement) conn.createStatement();

// truy vấn
ResultSet data = stm.executeQuery(sql);

List<Student> studentList = new ArrayList<Student>();

// trích xuất dữ liệu
while (data.next()) {
    String name = data.getString(1);
    int age = data.getInt(2);
    Student s = new Student(name, age);
    studentList.add(s);
}

// dữ liệu đã được lưu vào ArrayList
for (Student s: studentList) {
    System.out.println(s);
}

stm.close();
conn.close();
```

In JDBC, the position of the parameters starts from 1

PRACTICAL EXERCISES

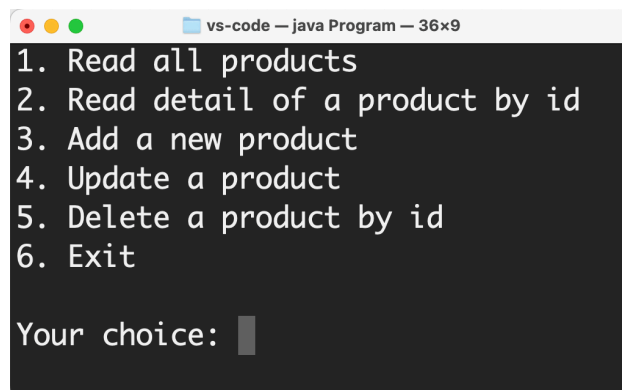
Write a java program that uses jdbc to connect to any database system (mysql or ms sqlserver). **The program must accept the connection url from the command line arguments**, you must not hardcode the url in the source code. This connection url contains all the information to the server and the necessary login information.

When the connection is successful, the program performs the following initial functions:

- Create a database named **ProductManagement** if the database doesn't already exist.
- If the **Product** table already exists, delete the table and then recreate it from scratch.

Next, the program displays a menu for the user to choose the functions they want to. The user enters the number to select the function he/she want to run, the program executes that function, then prints the results to the console (if any) and then display the original menu again until the user selects the "exit" menu item to stop the program. The list of functions that the program supports includes:

1. Read product list
2. Read a product by input id
3. Add a new product, the result is the product id (auto increment)
4. Update a product
5. Delete a product
6. Exit



```
vs-code — java Program — 36x9
1. Read all products
2. Read detail of a product by id
3. Add a new product
4. Update a product
5. Delete a product by id
6. Exit

Your choice: █
```

Example of menu displayed in the program

The program must be written in the following files:

- **Repository.java**: the given file, **do not** edit this file.
- **Product.java**: A POJO file that contains descriptive information for a product.
- **ProductDAO**: this class implements the Repository interface. You should write database connection functions here.
- **Program.java**: contains the main method and use the above classes to form a complete program.

Additional classes can be added if necessary for the exercise.