# Accessing databases in Java using JDBC

#### Introduction

JDBC is an API for Java that allows working with relational databases. JDBC offers the possibility to use SQL statements for DDL and DML statements. This API can be regarded as an abstraction layer between the application and the DBMS, programmers being able to work with the database independent of the underlying DBMS. Thus the code does not suffer modifications if the DBMS is changed. This independence is achieved by the fact that JDBC uses drivers, which translate JDBC API method calls to specific DBMS API. When we change the DBMS it is enough to change the driver, which can be done even at runtime.

(Note: Database notions are assumed to be known by the students, this lesson is focused only on using JDBC).

In addition to executing SQL statements, JDBC allows us to invoke stored procedures which are specific to the used DBMS.

#### **Requirements**

In this exercise we are using an the MySQL relational database that can be downloaded from https://dev.mysql.com/downloads/mysql/. The scripts for database creation will be provided in the tutorial, for easier manipulation of the database you can install MySQL Workbench, a visual tool for database administrators (https://www.mysql.com/products/workbench/)

Install MySQL Community Server with the Developer Default option.

When configuring the server be sure to choose legacy authentication:

## Authentication Method

#### Use Strong Password Encryption for Authentication (RECOMMENDED)

MySQL 8 supports a new authentication based on improved stronger SHA256-based password methods. It is recommended that all new MySQL Server installations use this method going forward.



Attention: This new authentication plugin on the server side requires new versions of connectors and clients which add support for this new 8.0 default authentication (caching\_sha2\_password authentication).

Currently MySQL 8.0 Connectors and community drivers which use libmysqlclient 8.0 support this new method. If clients and applications cannot be updated to support this new authentication method, the MySQL 8.0 Server can be configured to use the legacy MySQL Authentication Method below.

#### Use Legacy Authentication Method (Retain MySQL 5.x Compatibility)

Using the old MySQL 5.x legacy authentication method should only be considered in the following cases:

- If applications cannot be updated to use MySQL 8 enabled Connectors and drivers.
- For cases where re-compilation of an existing application is not feasible.
- An updated, language specific connector or driver is not yet available.

Security Guidance: When possible, we highly recommend taking needed steps towards upgrading your applications, libraries, and database servers to the new stronger authentication. This new method will significantly improve your security.

Use the password "root" for the admin account.

We will use Eclipse as IDE for this example.

#### **Creating the database**

Install MySQL (and optionally MySQL workbench) if you have not done it until this step. Make sure that mysql is running.

The following commands are entered from command line.

mysql -u root -p

Logs in to mysql as admin. You need to enter the password.

CREATE DATABASE jdbcex

Creates a new database called jdbcex.

SHOW DATABASES

Check that the database has been created

We should also create a user that can access this database to avoid the security risk of using the admin user.

CREATE USER 'jdbcuser'@'localhost' IDENTIFIED BY 'password'

GRANT ALL PRIVILEGES ON jdbcex.\* TO 'jdbcuser'@'localhost'

FLUSH PRIVILEGES

Connect to the database

USE jdbcex

We will create a simple database consisting of a Persons table and an Addresses table.

```
CREATE TABLE addresses (
```

id INT NOT NULL AUTO\_INCREMENT,

city VARCHAR(45) NOT NULL,

street VARCHAR(45) NULL,

PRIMARY KEY (id));

CREATE TABLE persons (

id INT NOT NULL AUTO INCREMENT,

name VARCHAR(45) NOT NULL,

address INT NULL,

PRIMARY KEY (id),

INDEX fk address idx (address ASC),

CONSTRAINT fk address

FOREIGN KEY (address)

REFERENCES addresses (id)

ON DELETE NO ACTION

ON UPDATE NO ACTION);

Finally let us introduce some records

INSERT INTO addresses (city, street) VALUES ('Bucuresti',
'Politehnicii');
INSERT INTO addresses (city, street) VALUES ('Ploiesti',

'Independetei');

INSERT INTO persons (name, address) VALUES ('Vasile', 1);

INSERT INTO persons (name, address) VALUES ('John', 1);

INSERT INTO persons (name, address) VALUES ('Emily', 2);

### Accessing a database using JDBC

#### **Basic steps**

#### Create a new Java project in Eclipse. Add the MySQL j-Connector jar. (Properties -> Java Build Path)

Properties for DBTest		$ \Box$ $\times$
type filter text	Java Build Path	
<ul> <li>Resource Builders</li> </ul>	Source Projects Libraries 🗞 Order and Export O Module Dependencies	
Coverage	JARs and class folders on the build path:	
Java Build Path	> 👼 mysql-connector-java-8.0.17.jar - C:\Program Files\Java	Add JARs
> Java Code Style > Java Compiler	> 🛋 JRE System Library [JavaSE-1.8]	Add External JARs
> Java Editor		Add Variable
Project Natures		Add Library
Project References		Add Class Folder
<ul> <li>Task Repository</li> </ul>		Add External Class Folder
Task Tags <ul> <li>Validation</li> </ul>		Edit
WikiText		Remove
		Migrate JAR File
		Apply
		Арріу
?	Apple	y and Close Cancel

We go through the following steps for accessing a database:

1. Making the connection

A connection is identified by an URL with the syntax:

jdbc:<protocol>:<nameOfDatabase>

In our case the URL has the form:

jdbc:mysql://localhost:3306/jdbcex

We also need a username and a password for connecting to the database (In our case username = jdbcuser, and the password is password). This three arguments (nameOfDatabase, user, password) should not be hardcoded in the application but instead saved in an external file (for example using java.util.Properties) so you can have a reusable class for connecting to the database.

So let's save the url in a string:

String url = "jdbc:mysql://localhost:3306/jdbcex";

(note that the port number is 3306 for MySQL)

and then connect to the database:

```
Class.forName("com.mysql.cj.jdbc.Driver");
Connection con = DriverManager.getConnection(url, "jdbcuser",
"password");
```

2. Execute an SQL statement

First we create a Statement from the Connection:

Statement instr=con.createStatement();

then we execute this statement. In case of a SELECT statement we use executeQuery() which returns a ResultSet and in case of an UPDATE, DELETE, INSERT, CREATE TABLE, DROP TABLE we use executeUpdate() which returns the number of affected rows. (except for create and drop where it returns 0).

3. Retrieve values from the ResultSet

JDBC returns results in a ResultSet object:

```
String sql = "SELECT * FROM persons";
ResultSet rs=instr.executeQuery(sql);
while(rs.next()){
System.out.println(rs.getString("name"));
}
```

rs.next() points the cursor of the resultset to the next position (initially it is before the first record). The values are extracted using getx() methods where x is the data type and the argument is the position of the column in the table. We could have also used the name of the column :

4. Close the connection

ResultSet, Statement must be closed first:

```
rs.close();
```

```
instr.close();
```

and then the Connection:

```
con.close();
The complete program :
import java.sql.*;
public class Main {
public static void main(String[] args) throws ClassNotFoundException,
SQLException {
        String url = "jdbc:mysql://localhost:3306/jdbcex";
        Class.forName("com.mysql.cj.jdbc.Driver");
        Connection con = DriverManager.getConnection(url, "jdbcuser",
"password");
        Statement instr = con.createStatement();
        String sql = "SELECT * FROM persons";
        ResultSet rs = instr.executeQuery(sql);
        while (rs.next()) {
            System.out.println(rs.getString("name"));
        }
        rs.close();
     instr.close();
        con.close();
    }
}
```

#### **Using JOIN**

Suppose you want to use more tables to perform a certain operation. In this case you join the tables using a value they have in common. (usually a foreign key). In our case we join persons and addresses using persons.address and addresses.id fields:

```
SELECT p.name, a.city FROM persons p INNER JOIN addresses a ON
p.address = a.id
WHERE a.city = 'Bucuresti'
```