**Programare Avansata pe Obiecte  
Laborator 11**

**Pahontu Bogdan-Ionut**

**E-mail:** [**pahontubogdan@gmail.com**](mailto:pahontubogdan@gmail.com)

Contents

[1. JDBC 3](#_Toc8684705)

[1.1. Writing basic SQL statements 3](#_Toc8684706)

[1.2. JDBC Interfaces 3](#_Toc8684707)

[1.3. Connecting to the database 4](#_Toc8684708)

[1.4. Obtaining a statement 5](#_Toc8684709)

[1.5. Executing a statement 5](#_Toc8684710)

[1.6. PreparedStatement 6](#_Toc8684711)

[1.7. Getting data from a ResultSet 7](#_Toc8684712)

[1.8. Getting data for a column – Primitives 8](#_Toc8684713)

[1.9. Scrolling a ResultSet 8](#_Toc8684714)

[1.10. Closing Data Resources 9](#_Toc8684715)

[1.11. Add Resources 10](#_Toc8684716)

[2. Tasks 10](#_Toc8684717)

# JDBC

A relational database is a database that is organized into tables, which consist of rows and columns.

There are two main ways to access a relational database from Java:

* Java Database Connectivity Language (JDBC): Accesses data as rows and columns.
* Java Persistence API (JPA): Accesses data through Java objects using a concept called object-relational mapping (ORM). The idea is that you don’t have to write as much code, and you get your data in Java objects.
* A relational database is accessed through Structured Query Language (SQL).
* In addition to relational databases, there is another type of database called a NoSQL database. This is for databases that store their data in a format other than tables.

## Writing basic SQL statements

* INSERT: Add a new row to the table
* SELECT: Retrieve data from the table
* UPDATE: Change zero or more rows in the table
* DELETE: Remove zero or more rows from the table

## JDBC Interfaces

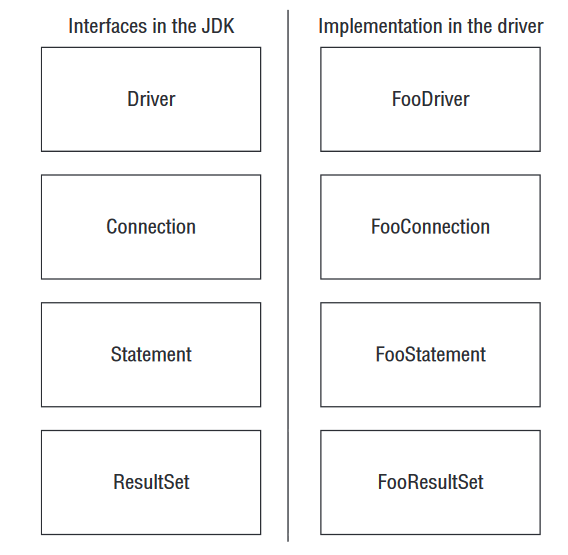


Figure – JDBC Interfaces

* **Driver**: Knows how to get a connection to the database
* **Connection**: Knows how to communicate with the database
* **Statement**: Knows how to run the SQL
* **ResultSet**: Knows what was returned by a SELECT query
* All database classes are in the package java.sql

## Connecting to the database

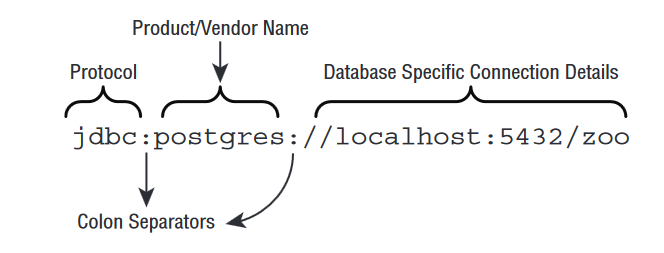


Figure – Connection string

* The first piece is always the same. It is the protocol jdbc.
* The second part the name of the database such as derby, mysql or postgres.
* The third part typically contains the location and the name of the database.

Example:

try {

Connection conn = DriverManager.getConnection("jdbc:sqlite:test.db");

} catch (SQLException e) {

System.out.println(e.getMessage());

}

If the JDBC Driver is not added to the classpath the following Exception will be thrown:

Exception in thread "main" java.sql.SQLException: No suitable driver found for jdbc:sqlite:test.db

at java.sql.DriverManager.getConnection(Unknown Source)

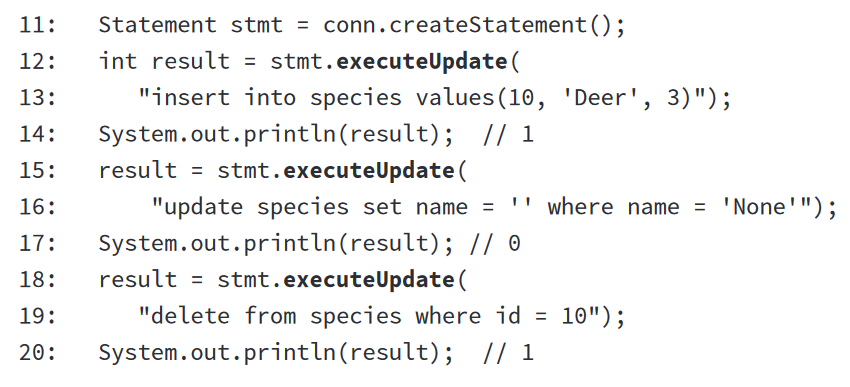
at java.sql.DriverManager.getConnection(Unknown Source)

## Obtaining a statement

Statement stmt = conn.createStatement();

## Executing a statement

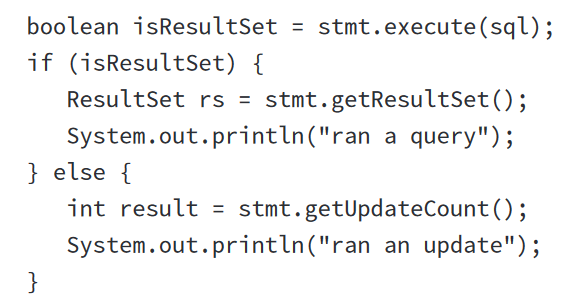
* **executeUpdate()** - takes the SQL statement to run as a parameter. It returns the number of rows that were inserted, deleted, or changed.



* **executeQuery() -** used when we have an SQL statement that begins with **SELECT.**



* **execute() -** can run either a query or an update. It returns a Boolean so that we know whether there is a ResultSet.



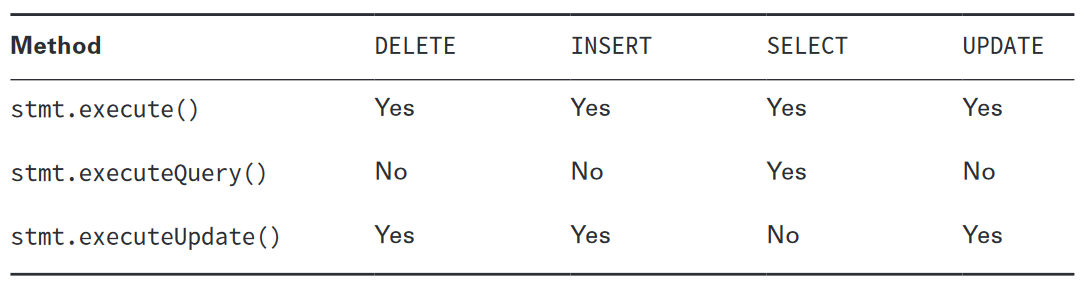


Figure – Executing a statement

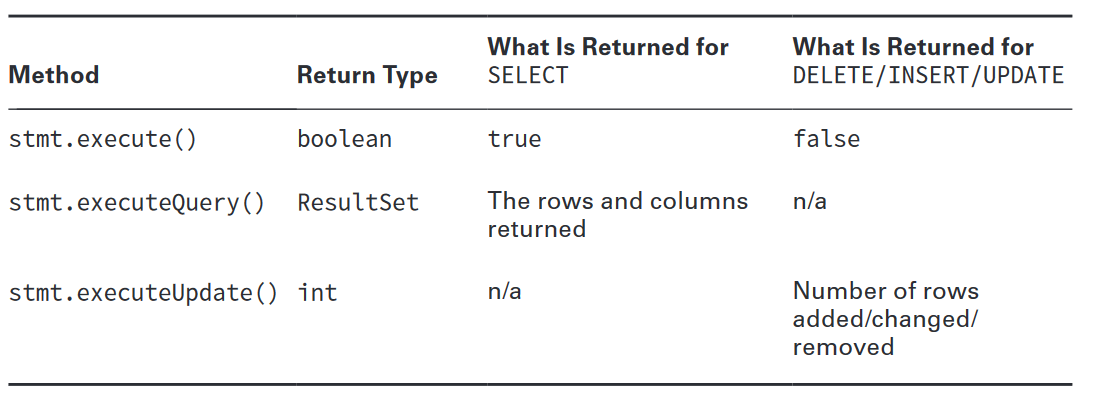


Figure – Executing a statement - return

## PreparedStatement

* In real life, you should not use Statement directly. You should use a subclass called **PreparedStatement**.
* This subclass has three advantages: performance, security, and readability.
* **Performance**: In most programs you run similar queries multiple times. A Prepared-Statement figures out a plan to run the SQL well and remembers it.
* **Security**;
* **Readability**: It’s nice not to have to deal with string concatenation in building a query
* string with lots of variables.

## Getting data from a ResultSet

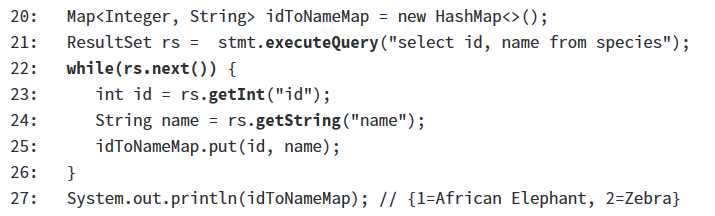


Figure – Getting data from ResultSet

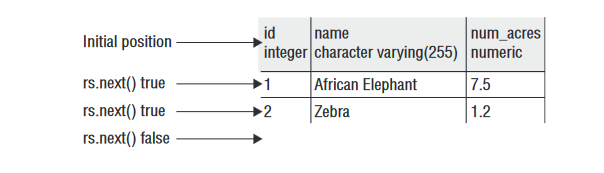


Figure – Getting data from ResultSet (2)

* You can use an index instead of a column name. The column name is better because it is clearer what is going on when reading the code.

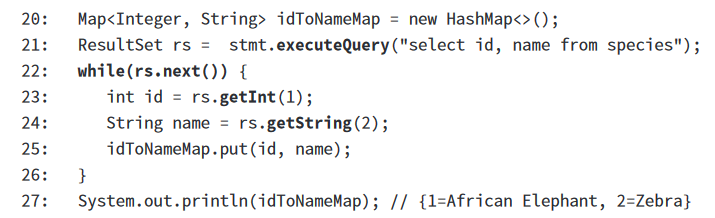


Figure – Getting data from ResultSet (3)

## Getting data for a column – Primitives

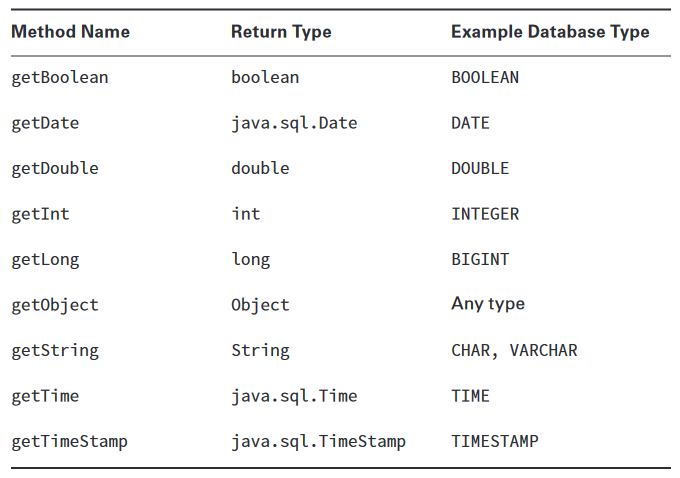


Figure – Getting data : Data Types

* Take care when you get a “Date” type you should convert to a java.time class ;

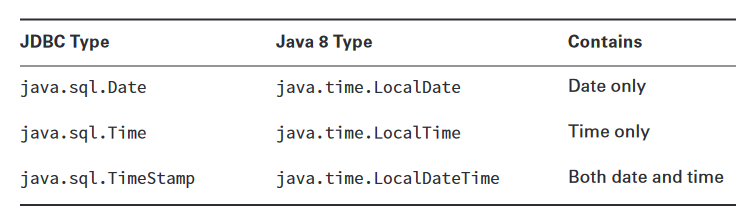


Figure – java.sql vs java.time

## Scrolling a ResultSet

* By default, a ResultSet is in **TYPE\_FORWARD\_ONLY** mode. It allows you to go through the data once in the order in which it was retrieved.
* Other options are: **TYPE\_SCROLL\_INSENSITIVE** and **TYPE\_SCROLL\_SENSITIVE;**
* With **TYPE\_SCROLL\_INSENSITIVE**, you have a static view of what the ResultSet looked like when you did the query. If the data changed in the table, you will see it as it was when you did the query.
* With **TYPE\_SCROLL\_SENSITIVE**, you would see the latest data when scrolling through the ResultSet.

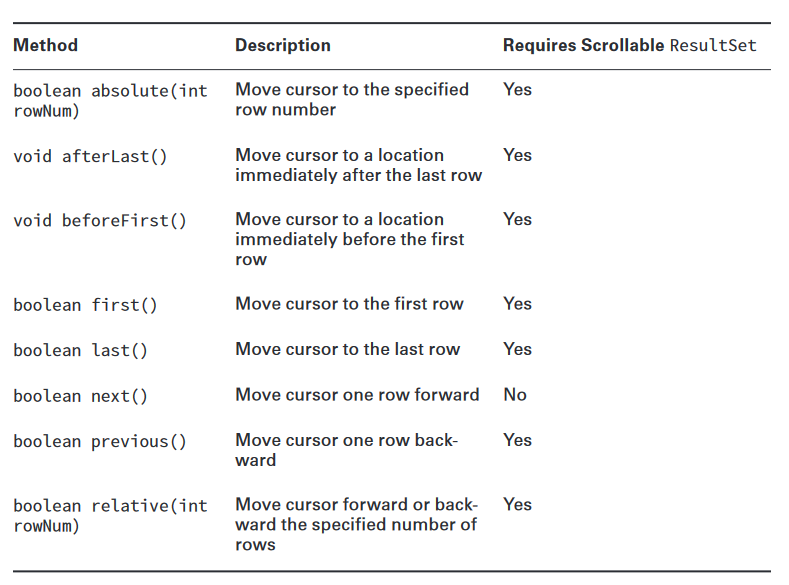


Figure – Scrolling a ResultSet

## Closing Data Resources

* JDBC automatically closes a ResultSet when you run another SQL statement from the same Statement.
* How many resources are closed in this code?

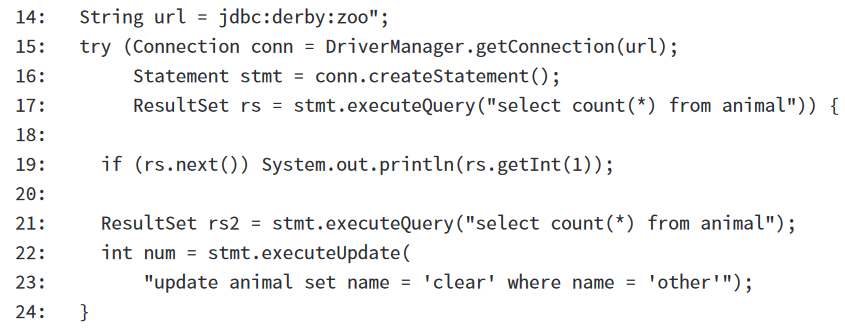


Figure – Closing the resource

R:

* On line 21, rs is closed because the same Statement runs another query.
* On line 23, rs2 is closed because the same Statement runs another SQL statement. This shows you that both a query and an update cause the previous
* ResultSet to be closed.
* Then the try-with-resources statement runs and closes the Statement and Connection objects.
* Closing a Connection also closes the Statement and ResultSet.
* Closing a Statement also closes the ResultSet.

## Resources and examples

* Install Server MySql
  + Link: <https://dev.mysql.com/downloads/installer/>
* JDBC Driver
  + Link: <https://dev.mysql.com/downloads/connector/j/>
  + Download the driver and add in a new created directory named libs;
  + File 🡪 Project Structure 🡪 Libraries 🡪 Add the JAR file;
* **Please check Lab11 JAVA project to see concrete examples;**

# Tasks

* + Install MySQL Server;
  + Download the connector and add it into your project;
  + For one of your entities in the project implement a CRUD on database;
    - Create a ConnectionUtils class (Singleton) that will handle the connection;
    - Create a resource file that will contain the connection string, database username and password;