# Lab: Database Apps Introduction

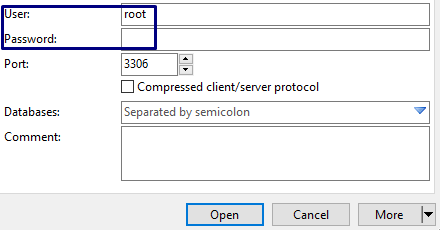
This document defines the lab assignments for the [“Databases Frameworks” course at Software University](https://softuni.bg/courses/databases-advanced-hibernate).

## Part 1: Accessing Database via Simple Java Application – Demo

You are given a simple java program that opens a connection to your local database server and retrieves data by user input - first and last name from the “**soft\_uni**” database by salary criteria. It shows briefly the usage of the **java.sql** package and **MySQL Connector/J** driver.

### Connection and Connection Properties

To this point we have used the HeidiSQL GUI to connect to our local instance of MySQL server.



From now on we will achieve that with **Java code** using the **JDBC API**.

Properties props = **new** Properties();  
props.setProperty(**"user"**, user);  
props.setProperty(**"password"**, password);  
  
Connection connection = DriverManager.*getConnection*(**"jdbc:mysql://localhost:3306/soft\_uni"**, props);

The following code shows how a connection is created:

Lets look closely on what is happening there. A Properties class in being instantiated, that holds the username and password of the user (given by input before this peace of code above). They are passed as an argument to the DriverManager static class’s method getConnection.

But before even getting to them, we need to specify the connection string. A connection string is a string, that holds information about a data source and consists of a database name and other parameters needed to establish the initial connection. In our case such parameters are:

* host and port of our local server – **localhost:3306**
* database name – can be diverse, but by the means of our assignment it is **soft\_uni**
* user – by default **root**, or any other **specified by the user**
* password – user password for connection if there is one

Last two are contained in our Properties instance.

Look back at the **getConnection** method. It is a static method that establishes a connection, but before that the DriverManager selects the appropriate driver – the **MySQL Connector/J** driver. If one is included, the connection is ready.

### Preparing and Executing Statements

SQL execution is done via the Statement Interface. To prevent SQL Injection, PreparedStatement is being used:

**PreparedStatement** stmt =

connection.prepareStatement(**"SELECT \* FROM employees WHERE salary > ?"**);

String salary = sc.nextLine();

stmt.**setDouble**(1, Double.parseDouble(salary));

The user is asked to input the salary criteria, by which we will filter the results. Then we use the connection to prepare a PreparedStatement and pass an unfinished SQL query to append to the input value of the criteria. The question mark is a placeholder for that value. The setDouble method accepts two parameters – index of the parameter to be replaced and the value. If we had done plain concatenation using just Statement and append the value of salary to the query, we make our program vulnerable to SQL Injection, because we do not check the value. It can be another SQL query, that in the worst case might harm our database. Thus we use PreparedStatement, setDouble and Double.parseDouble methods, to insure value as expected – a double number.

### Iterating over the Result

Finally, if data in the database have matched the given criteria, we will receive a ResultSet of table rows, otherwise it wil be empty.

ResultSet rs = stmt.executeQuery();

The ResultSet object is a table of data. It is not updatable and it can be read only from first to last, thus it remains cursor pointing to the rows of the table. Field values can be accessed via **getter** methods, such as getString, which accepts column name:

while(rs.next()) {

System.out.println(rs.**getString**(**"first\_name"**) + " " + rs.**getString**(**"last\_name"**));

}

## Part 2: Writing your own data retrieval application

Now it’s time for you write a similar program. Follow the steps to write a java application that retrieves information about the users, their games and duration. We are going to use the “**diablo**” database from the previous course.

### Result

|  |  |
| --- | --- |
| **Input** | **Output** |
| nakov | User: nakov  Svetlin Nakov has played 6 games |
| destroyer | No such user exists |

### Establish a connection to the database

Get the user’s **username** and **password** and change the connection string according to the name of the database.



### Username and statement

Ask the user for username by which you will retrieve the desired info. Write a proper SQL statement and get the result.



### Output

Consider that the input may be invalid – **user with given username might not exist** and you will receive an empty ResultSet. If so, print “**No such user exists**”, otherwise print the user\_name, **first** and **last names** and the total count of games a user has played.

