## **JDBC Exercise**

Starter Code: https://github.com/jbrucker/courselist

1. Follow instructions in README.md to create an Sqlite database with data.

2. Create an IDE project from the Java source code.

3. Add lib/sqlite-jdbc-3.36.0.3.jar to your project as a JAR file.

The "courselist" project contains these files

sample.db - SQLite database created by **you**

sample.schema - Schema data to create the database

lib/sqlite-jdbc-3.36.0.3.jar - JDBC driver for Sqlite

data/sample.csv - course data, import into database

src/courselist/Course.java - source code for Course class

Then you can try JDBC interactively in **jshell**. jshell is included in the JDK.

If you get "Not Found" then fix your PATH. Even better: switch to Linux.

**1. Start jshell**

Windows Users: you *might* need to use lib\ instead of lib/ as the path separator.

bash$ **jshell --class-path lib/sqlite-jdbc-3.36.0.3.jar**

Another way: you can add to the classpath while Jshell is running...

jshell> **/env --class-path lib/sqlite-jdbc-3.36.0.3.jar**

**2. Import the JDBC classes**

**jshell> import java.sql.\*;**

**jshell> /imports**

| import java.io.\*

| import java.math.\*

...

| import java.sql.\*

**3. Specify a URL for an embedded SQLite database.** This assumes "sample.db" already exists in your current directory. You can also specify a path to the database file if not in current dir.

jshell> **String url = "jdbc:sqlite:sample.db";**

**4. Get a connection to the database.**

jshell> **Connection connection = DriverManager.getConnection(url);**

connection ==> org.sqlite.jdbc4.JDBC4Connection@76a4d6c

Questions

Q1. Look at the actual connection class. How did JDBC know which class it should use? Can you guess the design pattern used?

Q2. "Connection connection = DriverManager.getConnection()" looks like what design pattern?

**5. Create a Statement,** which is how you issue queries or commands to the database driver.

jshell> **Statement statement = connection.createStatement();**

Questions

Q3. What kind of Statement object did you get?

Q4. What design pattern is this?



**org.sqlite.jdbc4.** JDBC4Connection JDBC4Statement

Actual values depend on the *actual* type of the database **Connection**.

6. How many rows are in the "courses" table?

jshell> **var query = "SELECT count() FROM courses";**

jshell> **ResultSet rs = statement.executeQuery(query);**

7. A ResultSet is a kind of iterator. It contains a "cursor" that points to the next row in the results. Initially, it point before the first row of results.

jshell command completion

ESC ESC - show command completions.

ESC TAB - if many possible completions you must press ESC then TAB

Q4. What design pattern is this?

jshell> **rs.is** [ESC ESC]

isAfterLast() isBeforeFirst() isClosed() isFirst()

isLast() isWrapperFor()

jshell> **rs.isBeforeFirst()**

**true**

**Move the cursor to the next row. This will be the first and only result.**

jshell> **rs.next()**

**8. How many courses (rows) are in the courses table?**

The "ResultSet" contains only a single integer = value of **count()**.

The fields in a ResultSet are indexed, starting at 1. it's up to you, the programmer, to know which index is which result!

jshell> **rs.get** [ESC ESC]

getArray() getBlob() getBigDecimal()

getByte() getDate() getBoolean()

getClass() getInt() getDouble()

getMetaData() getObject() getFloat()

getRow() getRowId() getLong()

...

jshell> **rs.getInt(1)**

61

// Another way:

jshell> **rs.getRow()**

**8. Get a row of course data.**

Each course contains:

**Course**

id

course\_number

title

credits

difficulty

jshell> query = "SELECT \* FROM courses c WHERE c.course\_number = ''";

jshell> **ResultSet rs = statement.executeQuery(query);**

jshell> while( rs.next() ) {

// process the info about this row of ResultSet

String courseNumber = rs.getString("course\_number");

String title = rs.getString("title");

int credits = rs.getInt("credits");

double diff = rs.getDouble("difficulty");

//TODO print it, or create a Course object and print that

}

jshell>

## Create A CourseDao using JDBC

1. Create a package named courselist.persistence for your DAO class.

2. In the courselist.persistence package, create a CourseDao class containing these methods and constructor:

|  |  |
| --- | --- |
| CourseDao(  Connection connection) | Constructor with a java.sql.Connection parameter that is a connection to the database. |
| Course get(int id) | Return a Course object from the database with the given id. |
| void save(Course course) | If the object has not been saved (id == 0) then save it to the database, and assign the id a value equal to the primary key in the courses table.  If the object has already been persisted (id != 0) then update the values stored in the database to match the object's attribute values. |
| void delete(Course course) | Delete the object's data from database, and set the id to zero to indicate it is no longer a persisted object. |
| int count() | Return the number of objects in the courses table. |
| Course findByCourseNumber(   String courseNumber) | Find and return a course having the given course number. |

### How JDBC Works

To use JDBC you first create a connection to a database, using a Connection object. Connection objects are specific to the type of database, e.g. MySQL Connection, Oracle Connection.

Use the Connection object to create a Statement. Statement is a reusable command object that you use to execute SQL commands and get the results. You can specify attributes for a Statement (such as how many results it can hold). Connection can also create another kind of Statement called a PreparedStatement, which has better performance and security.

SQL "SELECT" queries return results as a ResultSet object, which lets you iterate over rows in the result. A ResultSet also contains *metadata* you can use to discover information about the results.



Database

*Connect and authenticate*

*execute*

### How does DriverManager know *what* *database* to use?

The first parameter to DriverManager.getConnection( ) is a URL.

The **url** parameter identifies the location of the database server and the *kind* of database (DB2, MySQL, etc). Here are some examples of URLs:

**jdbc:mysql://se.cpe.ku.ac.th/world** MySQL database on a server

**jdbc:mysql://localhost:3306/world** MySQL database server on this host

**jdbc:sqlite:/database/world** SQLite database on this host

**jdbc:hsqldb:file:/database/world** HSQLDB embedded database on this host

If the URL starts with "jdbc:sqlite" then DriverManager looks for a **registered driver**for "sqlite". It instantiates a Connection class provided by that driver (org.sqlite.jdbc4.JDBC4Connection) and returns it. Your code uses the Connection object to create other objects (Statement, PreparedStatement, etc). The Sqlite JDBC4Connection class returns concrete implementations of Statement, PreparedStatement, etc.