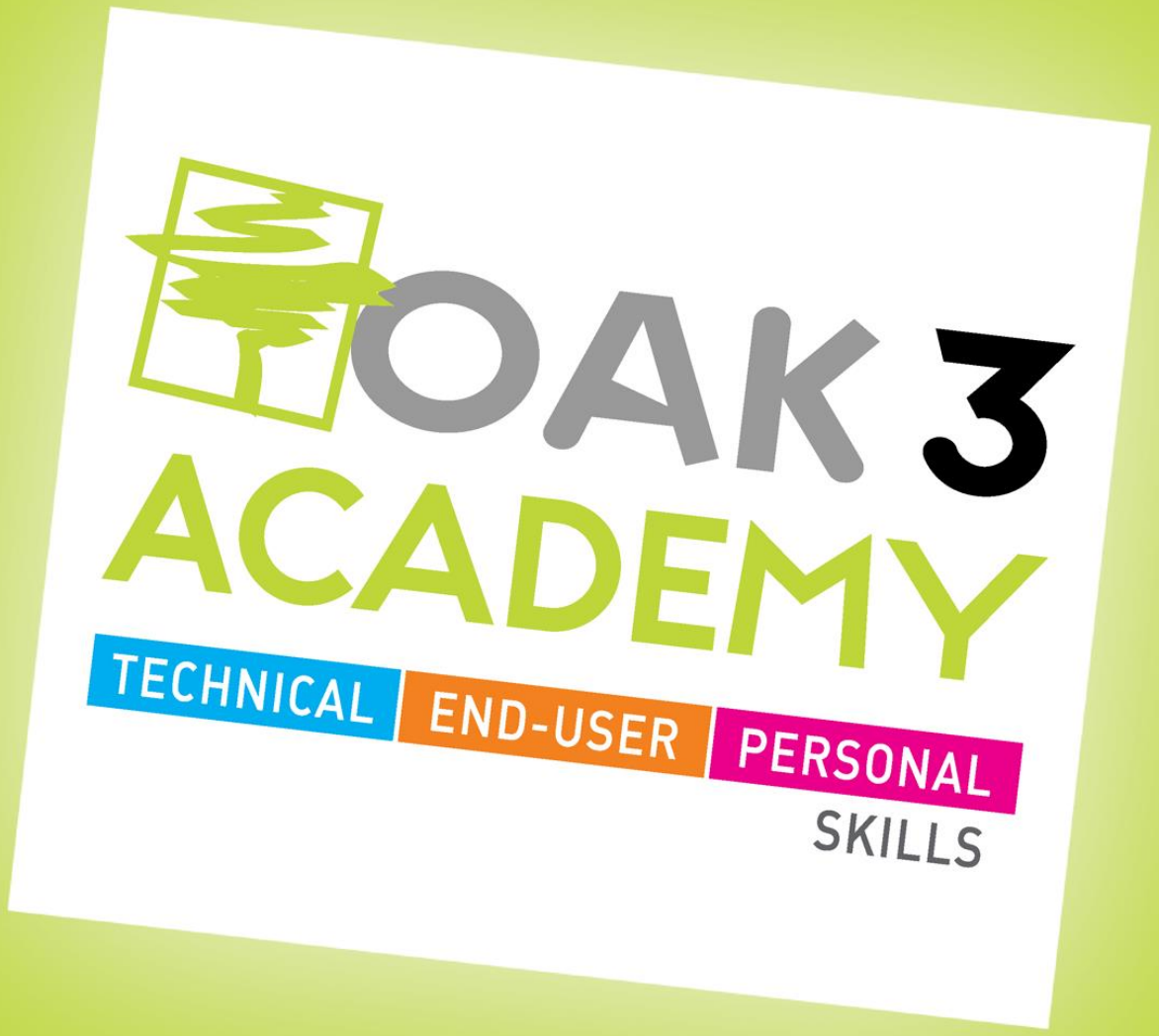

JDBC

Table of contents

Introduction

JDBC = Java DataBase Connectivity

JAVA API: basic support for Connections/ Statements/ ResultSets

Requirements

RDMS: Relational Database Managment System (Oracle, MySQL, MS SQL…)

Connector: connection between JDBC and RDMS

IDE (IntelliJ) – JDK

JDBC Fundamentals

Steps:

1. Load Driver (before Java 6)
2. Define Connection URL
3. Open Connection
4. Create Statement
5. Execute Query
6. Processing Results
7. Close resources

1. Load Driver

```
Before Java 6:
try {
   Class.forName("com.mysql.jdbc.Driver");
} catch (ClassNotFoundException cnfe) {
   System.out.println("Error: " + cnfe);
}
```

Since Java 6: driver loads automatically

2. Connection URL

jdbc:<DBMS>://<HOSTNAME>:<PORT_NUMBER>/DATABASE_NAME

Example:

String DATABASE_URL=
“jdbc:mysql//noelvaes.eu:3306/StudentDB”

3. Open Connection

Connection Object → getConnection() method of DriverManager.

```
String sourceURL = “jdbc:mysql://noelvaes.eu/StudentDB”;
String username   = “student”;
String password   =  “student123”;

Connection con =
DriverManager.getConnection(sourceURL, username, password);
```

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

3. Open Connection

Try with resources → Since JAVA 7 → Closes resource automatically.

```
try (Connection con =
      DriverManager.getConnection(sourceURL, username,
password);) {
…

}
catch (SQLException sqle) {
…
}
```

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

4. Statement object

Send queries to RDMS

```
try (Connection con =
      DriverManager.getConnection(sourceURL, username,
password); Statement stmt= con.createStatement();) {
…

}
catch (SQLException sqle) {
…
}
```

5. Execute query

Statement object has 3 methods:

- executeQuery() → return type ResultSet
- executeUpdate()→ returns type int
- execute → return type boolean

true → ResultSet

false → int

5. Execute query

```
String query = “select * from beers”;
ResultSet rs =
   statement.executeQuery(query);
```

6. Processing Results

```
String query = “select * from beers”;
ResultSet rs =
   statement.executeQuery(query);

while(rs.next()){
   System.out.println(rs.getString(“name”);
}
```

6. Processing Results

- Data is accessible row by row
- Cursor moves only forward! (next())
- At start, cursor is before first row

7. Close Connection

Before java 7: method close();

After java 7: “try-with-resources”

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

Exercise 1

Start MySQL: make connection to localhost.

Import BeerDB: https://github.com/KennethVG/Products_English --> Execute script in MySQL!

Create BeerApp to connect to the BeerDB.

EXTRA: Use a property file!

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

Exercise 2

- Show the names of the beers where the alcohol is below 6°.
- Sort the beers on price as well.

Exercise 3

- Change the stock of all Beers to 50.
- Print out how many Beers are changed
- Add a new Beer to the DB

Scrollable ResultSets

- Accessible forward, backward and directly.
  - next()
  - previous()
  - first()
  - last()
  - absolute(rowNr)
  - relative(rows)

Scrollable ResultSets

Statement stmt = con.createStatement(ResultSet.***TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_READ_ONLY))***

- ***TYPE_FORWARD_ONLY: default***
- ***TYPE_SCROLL_INSENSITIVE***
- ***TYPE_SCROLL_SENSITIVE***

INSENSITIVE VS SENSITIVE

TYPE_SCROLL_INSENSITIVE:

The result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set is insensitive to changes made to the underlying data source while it is open. It contains the rows that satisfy the query at either the time the query is executed or as the rows are retrieved.

TYPE_SCROLL_SENSITIVE:

The result can be scrolled; its cursor can move both forward and backward relative to the current position, and it can move to an absolute position. The result set reflects changes made to the underlying data source while the result set remains open.

Scrollable ResultSets

Example:

```
rs = stmt.executeQuery("select * from Beers");


resultset.absolute(23); // jump to record 23
```

Updatable ResultSets

Statement stmt = con.createStatement(ResultSet.***TYPE_SCROLL_INSENSITIVE, ResultSet.CONCUR_UPDATABLE))***

- ***CONCUR_READ_ONLY: default***
- ***CONCUR_UPDATABLE***

Scrollable ResultSets

Example:

```
while (rs.next()) {
String beername = rs.getString("Name");
    if (beername.equalsIgnoreCase("Aardbeien
        witbier")) {
            rs.updateString(2, "Aardbeien BlondBier");
            rs.updateRow();
    }
}
```

Exercise 5

Show the beers from row 50 until row 60.

OAK 3 ACADEMY
TECHNICAL END-USER PERSONAL SKILLS

Statements

Statement:

Need to be compiled by the server everytime.

PreparedStatement:

Compiled once → Much faster!

Can accept params

CallableStatement:

Access Stored Procedures/functions from the database.

Can accept params

PreparedStatement

Example:

```
PreparedStatement ps =
con.prepareStatement( “select * from
planten where kleur = ?”);

ps.setString(1,”geel”);
ResultSet rs= ps.executeQuery();
```

PreparedStatement

Example 2:

```
ps = con.prepareStatement (
"insert into beers (name,stock) values (?,?)" );

ps.setString(1,“Jupiler");
ps.setInt(2,100);

count = ps.executeUpdate();
```

Exercise 6

Search for all beers with “bier” in the name and where the alcohol ° is below 7.

Use a PreparedStatement.

Exercise 7

List<Beer> getBeersByName(String name);

int updateBeerWithId(int id, Beer newBeer);

Transactions

Execute a number of queries at the same time. Use **commit()** method.

→If one query fails, all the other will not be executed!

rollback(): all queries that are not committed will not be executed

setAutoCommit(false)

Transactions

During transaction: db is read only!

Some issues:

- **Dirty Read:** T1 changes row, T2 reads before changes are committed
- **Non-Repeatable read:** T1 reads, T2 changes and commits, T1 rereads and get different values.
- **Phantom read:** T1 reads, T2 inserts or delete some rows, T1 rereads and get different number of rows.

Transactions Isolation Level

con.getTransActionIsolation() → Default

Level	Dirty read	Non-repeatable read	Phantom read
TRANSACTION_NONE	No transactions!		
TRANSACTION_READ_UNCOMMITTED	☺	☺	☺
TRANSACTION_READ_COMMITED	✗	☺	☺
TRANSACTION_REPEATABLE_READ	✗	✗	☺
TRANSACTION_SERIALIZABLE	✗	✗	✗

Batches

= Transaction BUT if one query fails the other will still be executed!

Batches

```
con.setAutoCommit( false );
stmt = con.createStatement( );

stmt.addBatch("insert into…. values(...)");
stmt.addBatch("insert into…. values(...)");
stmt.addBatch("insert into…. values(...)");
...

int [ ] updateCounts = stmt.executeBatch( );

System.out.println("All " + updateCounts.length + " are
succesfully executed!");
con.commit( );
```

Exercise 7: **Bankruptcy**

Brewer 1 → bankruptcy

- Beers with alcohol ° above 7.5 are taken over by Brewer 2.
- All other beers are taken over by Brewer 3.
- At the end Brewer1 will be deleted!

Do everything with one transaction.

Show how many rows are updatet!