Database systems (DBS1) Spring 2017

Relational modeling
- Normalization of relations

+ basics of JDBC

Bo Brunsgaard

Agenda

- Modeling work from last week
- JDBC: accessing database from Java
- Normalization of relations

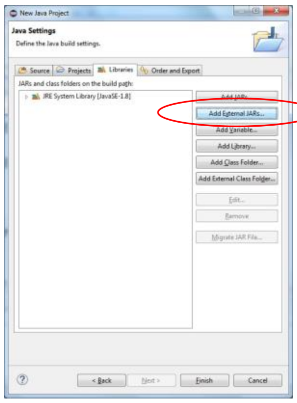
JDBC

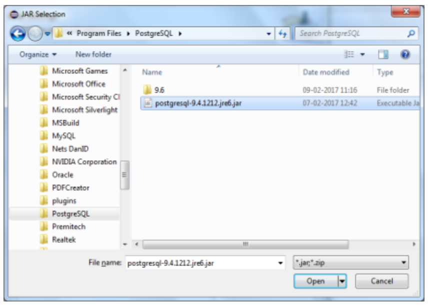
What is JDBC

- Java DataBase Connectivity
- A general interface from Java programs to relational databases
  - (reasonably) product agnostic

Required setup in Eclipse

- JDBC needs a driver to translate JDBC calls to the specific DBMS
- Set on the build path of the Java project

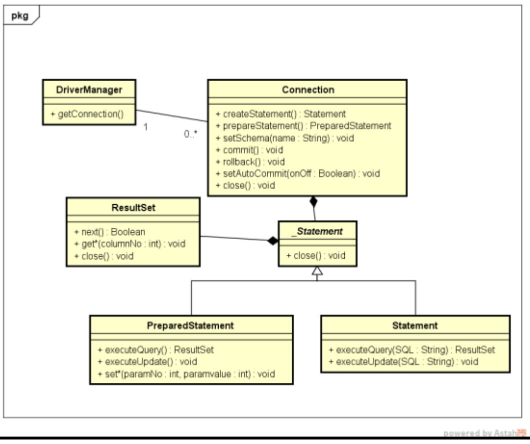




Name	Date modified	Type
9.6	09-02-2017 11:16	File folder
postgresql-9.4.1212.jre6.jar	07-02-2017 12:42	Executable Ja

File name: postgresql-9.4.1212.jre6.jar

.jar;.zip


pkg
DriverManager
+ getConnection()
1
0..*
Connection
+ createStatement() : Statement
+ prepareStatement() : PreparedStatement
+ setSchema(name : String) : void
+ commit() : void
+ rollback() : void
+ setAutoCommit(onOff : Boolean) : void
+ close() : void
ResultSet
+ next() : Boolean
+ get*(columnNo : int) : void
+ close() : void
_Statement
+ close() : void
PreparedStatement
+ executeQuery() : ResultSet
+ executeUpdate() : void
+ set*(paramNo : int, paramvalue : int) : void
Statement
+ executeQuery(SQL : String) : ResultSet
+ executeUpdate(SQL : String) : void
powered by Astah

Errors

- All JDBC calls may throw an error
- They must be enclosed in a try-catch block

```
try {
    PreparedStatement stmt = conn.prepareStatement("something");
    stmt.executeUpdate();
}
catch (SQLException e) {
   System.out.println("error executing insert");
   System.out.println(e.getMessage());
   e.printStackTrace();
   System.exit(e.getErrorCode());
}
```

Includes

- Relevant JDBC classes must be included
- You can either import java.sql.* or be nicer:

```
import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.PreparedStatement;
import java.sql.Statement;
import java.sql.ResultSet;
import java.sql.SQLException;
```

Getting a connection

```
    final String userID = "javademo";
    final String password = “mytopsecretpassword";
    final String database = "dbbbc";
    Connection conn = null;

    try {
1    Class.forName("org.postgresql.Driver");
     conn = DriverManager.getConnection(
2                "jdbc:postgresql://localhost:5432/"+database
                 ,userID
                 ,password);

3    conn.setSchema("javademo");
```

1 Load the DriverManager class associated with postgreSQL

2 Return a connection to the specified DBMS server for the user

3 Set database schema prefix (normally specified on the user in the DB=

Preparing and executing a statement

```
1  PreparedStatement stmt = conn.prepareStatement(
                       "insert into employee "
                     + " (name,salary)"
                     + " values(?,?)");

2  stmt.setString(1, "Professor Snape");
   stmt.setInt(2, 500);

3  stmt.executeUpdate();

   stmt.close();
```

1 Prepare a statement with placeholders (?) for specific values

2 Set execution-specific values for placeholders

3 Execute statement with values supplied

Preparing and executing a statement returning rows

```
1    PreparedStatement stmt = conn.prepareStatement(
                    "select name , salary from employee "
                 + " where salary < ?");

2    stmt.setInt(1, 120);

3    ResultSet rs = stmt.executeQuery();
```

1. Prepare a statement with placeholders (?) for specific values
2. Set execution-specific values for placeholders
3. Execute statement with values supplied, now returning a ResultSet object

Processing rows returned (if any)

```
1    while (rs.next()) {

        System.out.println(
                 rs.getString("name")
2             + "is paid a measly " + rs.getInt(2));
     }

     rs.close();
3    rs = null;
```

1. The next() method on the ResultSet advances to the next row in the ResultSet, returning false if no next row exists (no rows returned, or at the end of the ResultSet
2. get*Type*() retrieves the value of a column in the resultset for the current row. Note you can specify columns either by name (preferred) or position
3. Your mother doesn't work here, so clean up after you!

DYNAMIC STATEMENTS (AND WHY THEY ARE BAD FOR YOU)

Statement instead of PreparedStatement?

- In tutorials and online forums you will often see the use of `Statement`:*"they are easier to use"*
- <u>This is, put bluntly, a lazy and unprofessional habit: save once, pay many times</u>
- Statement has a very limited place in professional systems' development
  - **It is much more resource demanding**
  - **It is inherently unsafe**

A dynamic statement

```
int salary = 200;

Statement stmt = conn.createStatement();

ResultSet rs = stmt.executeQuery(
         "select name, salary from employee "
       + "where not salary < " + salary );
```

1 Instead of preparing a statement with placeholders, and then supplying values on execution, we just take a `statement` object

2 Values for any parameters are made a part of the SQL statement string

Performance compared

	Execution			Statement processing	
	Time	CPU	IO	Time	CPU
Dynamic	770	55	10289	58 (*)	54
Prepared	220	16	8816	8	14

(*) remainder of difference in elapsed time (500 ms) due to idle wait time generated by dynamic statement processing

Dynamic statements are unsafe

- Dynamic statements allow user input to be interpreted as part of the SQL
- Called SQL Injection, and it is a well-known vulnerability

*Applicant must have experience in areas such as Cross Site Scripting, **SQL-injection**, heap-spray techniques, bufferoverflows and other relevant black hat techniques*

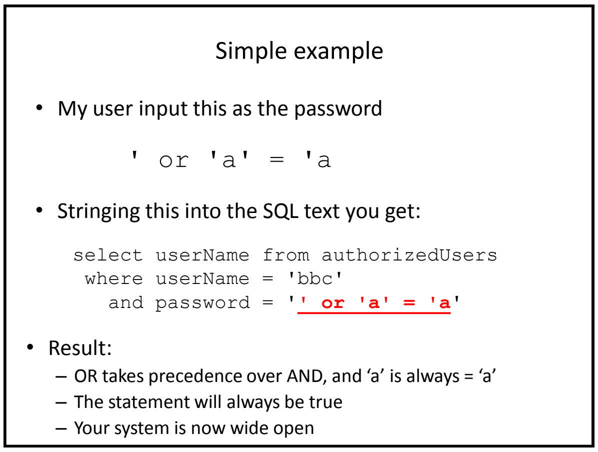
(real-life job advert from the Danish Defense Intelligence Service)

Simple example

```
String sqlText1 = "select userName from authorizedUsers "
                + " where userName = " ;
String sqlText2 = " and password = ";

ResultSet rs = stmt.executeQuery(
                          sqlText1
                        + "'" + userInputName + "'"
                        + sqlText2
                        + "'" + userInputPassword + "'");

if (rs.next()) {
System.out.println("You're in");
}
```



Simple example

- My user input this as the password

```
      ' or 'a' = 'a
```

- Stringing this into the SQL text you get:

```
select userName from authorizedUsers
 where userName = 'bbc'
   and password = '' or 'a' = 'a'
```

- Result:
  - OR takes precedence over AND, and ‘a’ is always = ‘a’
  - The statement will always be true
  - Your system is now wide open