

Database Access from a Programming Language:

Java's JDBC

Werner Nutt

Introduction to Databases

Free University of Bozen-Bolzano

Database Access from a Programming Language

2

Two Approaches

1. **Embedding SQL** into programming language

e.g., “Embedded SQL” for C and C++

2. DB access **via API** (= “*call level interface*”)

e.g., JDBC, ODBC

~> *How do they work?*

Approach 1: Embedded SQL

- *SQL code occurs in program*, separated by markers:

```
EXEC SQL SELECT ranking INTO :r
        FROM   sailors
        WHERE  sailors.sid = 15765;

r++;
EXEC SQL UPDATE sailors S
        SET    ranking = :r
        WHERE  sailors.sid = 15765;
```

- *Transfer of values* between PL and SQL:
use of **host language variables** in SQL (prefixed with “:”)
- *Compilation in two steps:*
 1. **Preprocessor** translates *SQL fragments* into *function calls* of SQL run time library (= pure C/C++ code, depends on DBMS)
 2. **Regular compiler** for C/C++ produces executable

Approach 2: Call Level Interfaces

4

Principles of JDBC

- *Contact* between **Java application** (= client) and **DBMS** (= server) is brokered by a **driver**
 - *Application* invokes **server commands** by sending **strings**
 - *Driver* translates command **strings** into DBMS procedure **calls**
 - Drivers are **vendor specific**
e.g., drivers for PostgreSQL, Oracle, DB2, SQL Server, ...
 - A **driver manager** chooses the *right driver* for each DBMS
- ~> *compiled client does not contain compiled SQL code*
- ~> *one client can communicate with many DBMS's,
even from different vendors*

Schema of a JDBC Application

5

- *Load* the **driver** for a specific *DBMS*
(e.g., the PostgreSQL “JDBC Driver”)
- *Establish* a **connection** to a specific *database*
(e.g., the PostgreSQL database `wdb` on the server `database.inf.unibz.it`)
- *Create* an abstract **statement**, to be sent over the connection
- **Execute** the *statement* by sending a Java string
(e.g., `"SELECT ranking FROM sailors WHERE sid = 15765"`)

↪ *returns an object of class* `ResultSet`

- *Process* the **result set** with methods of `ResultSet`
- **Close** statement and connection

JDBC Example Code: Parameters

6

```
import java.io.*;
import java.sql.*;

//This class collects the parameters for the example ...
public class PostgreSQLAccess{
    static PrintWriter screen = new PrintWriter(System.out,true);
    static BufferedReader keyboard =
        new BufferedReader(new InputStreamReader(System.in));

    //Driver name: set CLASSPATH so that Java can find it!
    static String driverName = "org.postgresql.Driver";

    //URL of the DB: specifies access protocol and location of the DB
    static String dburl = "jdbc:postgresql://database.inf.unibz.it/mydb";

    //User name: wnutt
    static String user = "wnutt";

    //Password for the database
    static String passwd = "cheerio";
}
```

JDBC Example Code: Queries are String

7

```
import java.sql.*;
```

```
import java.io.*;
```

```
public class Select extends PostgreSQLAccess{
```

```
    //Our example query
```

```
    static String selectQuery =
```

```
        "SELECT    ename, sal " +
```

```
        "FROM      emp " +
```

```
        "WHERE     hiredate > '01-JAN-2002'" +
```

```
        "ORDER BY  ename";
```

This is the query that we want to send to the database!

JDBC Example Code: Accessing the Database

8

```
public static void main(String args[]) throws Exception {  
  
    //Load the driver  
    Class.forName(driverName);  
  
    //Establish a connection to the database via the driver  
    Connection con =  
        DriverManager.getConnection(dburl, user, passwd);  
  
    //Create an abstract statement for the connection  
    Statement stmt = con.createStatement();  
  
    //Execute the query and retrieve the set of results  
    ResultSet rs = stmt.executeQuery(selectQuery);  
  
    //Process results (see below)  
    printResults(rs,screen);  
  
    //Close statement and connection  
    stmt.close();  
    con.close();  
}
```


JDBC Example Code: Processing the Result Set

9

```
//Process the result set with methods of class ResultSet
public static void printResults(ResultSet rs, PrintWriter pw) {
    try{
        //Move cursor over the result set
        while (rs.next()) {
            //Fetch components of result tuples
            String ename = rs.getString(1);
            int sal = rs.getInt(2);

            //Feed components into computation
            screen.println(ename + " \t earns   " + sal + "\t per month");}

            //Close result set
            rs.close();

            //Catch exceptions
        }catch (Exception e) {
            screen.println(e.toString());}
    }
}
```

JDBC Example Code: the Output

10

```
>> java Select
```

Adams	earns	1100	per month
Allen	earns	1600	per month
Blake	earns	2850	per month
Clark	earns	2450	per month
Ford	earns	3000	per month
James	earns	950	per month
Jones	earns	2975	per month
King	earns	5000	per month
Martin	earns	1250	per month
Miller	earns	1300	per month
Scott	earns	3000	per month
Smith	earns	800	per month
Turner	earns	1500	per month
Ward	earns	1250	per month

JDBC Example Code: Updates (1)

11

```
import java.sql.*;
```

```
import java.io.*;
```

```
public class Update extends PostgreSQLAccess{
```

```
    //Our example update
```

```
    static String update =
```

```
        "INSERT INTO emp VALUES (" +
```

```
        "8492, 'MacGregor', 'Clerk', 7902, " +
```

```
        "'15 NOV 2001', 1800, null, 200)";
```

JDBC Example Code: Updates (2)

12

```
public static void main(String args[]) throws Exception {  
  
    Class.forName(driverName);  
    Connection con =  
        DriverManager.getConnection(dburl, user, passwd);  
    Statement stmt = con.createStatement();  
  
    //Execute the update  
    stmt.executeUpdate(update);  
  
    stmt.close();  
    con.close();  
}  
}
```

JDBC: Prepared Statements

13

This is a way to define **patterns** of queries and updates:

//Define pattern with question marks as place holders

```
static String updatePattern =  
    "UPDATE emp SET sal = ? WHERE ename = ?";
```

//Create instance of class PreparedStatement on connection

```
PreparedStatement updateEmpSalary =  
    con.prepareStatement(updatePattern);
```

//Fill in values for question marks

```
updateEmpSalary.setInt(1,2000);  
updateEmpSalary.setString(2,"MacGregor");
```

//Execute the instantiated pattern

```
updateEmpSalary.executeUpdate();
```

//Close the prepared statement

```
updateEmpSalary.close();
```

JDBC: More Features

- **Prepared statements:** *updates* and *queries*,
with methods `executeUpdate()`, `executeQuery()`
- **Navigation** in *result sets*:
forward, backward, skip *n* tuples, ...
- **Transactions**
 - defined for *connections*
 - methods `commit()`, `rollback()`, `setSavepoint().`,...
- **Error handling** with classes
 - `SQLException`
 - `SQLWarning`

References

15

These slides are partly based on learning material provided by SUN Microsystems at

<http://java.sun.com/docs/books/tutorial/jdbc/TOC.html>.

The examples on the slides have been run on the faculty Linux machines and reflect our local setup.