



JDBC & JSQL

Java Database Connections

Heimir Þór Sverrisson
Technical Director, Teymi hf.

1



Agenda

Objectives

- JDBC
- JSQL
- Conclusions

2



Objectives

- Give an overview of Java connection to relational databases.
- Compare and contrast the different solutions available to developers.
- Show example applications for both two and three tier architecture i.e. browser based applets, webserver and standalone client apps.

3



Agenda

- Objectives
- JDBC
 - ➡ **Overview**
 - Drivers
 - Examples
- JSQL
- Conclusions

4



JDBC is not ODBC !!

- ODBC APIs provide the lowest common denominator of database functionality.
- Performance
 - Cannot tune performance, like some native interfaces
 - Consumes more client resources.
 - Higher overhead than native interfaces.
- Extensibility
 - No support for OO extensions (e.g. objects, collections, lobes, etc.)

5



JDBC Overview

- A part of JDK-1.1
- Java class library for accessing relational databases
 - Can be used for queries and data manipulation
- Object oriented interface
 - Based on X/Open SQL CLI standard (as is ODBC)
 - Supports SQL92 syntax and types.

6



JDBC Overview 2

- **Extensible**
 - Allows for vendor-specific extensions.
 - Vendors provide JDBC *Drivers* that implement the JDBC interfaces for their particular database.
- **Low Level**
 - Higher level interfaces can be built on top of JDBC (like JSQL and bound datacontrols)

7



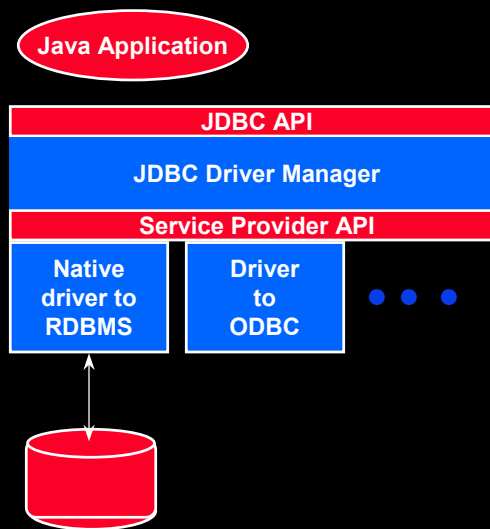
Agenda

- **Objectives**
- **JDBC**
 - Overview
 - ***Drivers***
 - Examples
- **JSQL**
- **Conclusions**

8

JDBC: Driver Based Implementation

- The JDBC class library is a vendor independent layer
- Vendors provide JDBC *Drivers* that implement the JDBC interfaces for their particular database
- Reference implementation is an ODBC driver



JDBC Driver types (1 & 2)

- (1) JDBC-ODBC bridge is built on top of ODBC. Database client code and ODBC must be loaded on each client machine.
- (2) Native-API partly-Java. Converts JDBC calls to the underlying RDBMS API. Native API must be loaded on each client.



JDBC Driver types (3 & 4)

- (3) JDBC-Net pure Java. Translates JDBC calls into RDBMS independent protocol. A corresponding translator from JDBC-Net to vendor RDBMS API must sit on the server
- (4) Native-protocol pure Java. Converts JDBC calls into the network protocol used by the RDBMS vendor.

11

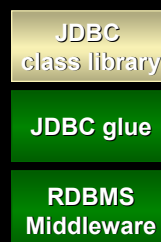


Type 2

- Ideal for Java Applications and Middle-Tier servers (high throughput)
- Can benefit from all RDBMS vendor features: character set conversions, numbers, etc.

BUT

- Not downloadable to applets
- Requires RDBMS middleware on client



12

Type 4

- Pure Java: Ideal for Java applets
- Can be downloaded with applet
- No client installation required
- Rather small (x00 kBytes)
- **BUT**
- Only supports TCP/IP
- Takes time to download


JDBC
class library

T4 Driver

Java sockets

13

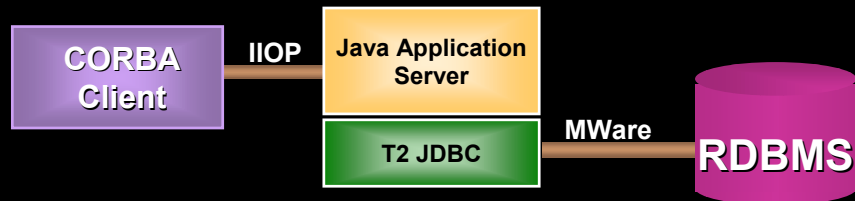
Agenda

- Objectives
- JDBC
 - Overview
 - Drivers
-  **Examples**
- JSQL
- Conclusions

14



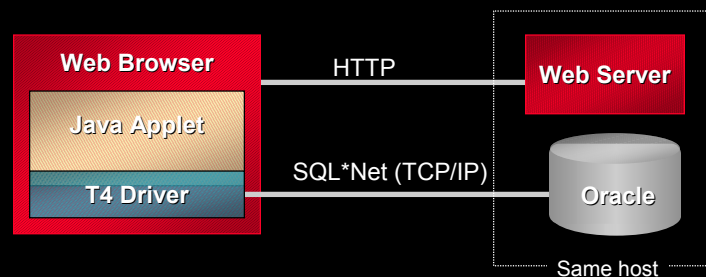
Type 2 for App Servers



Java Enabled Application servers can access the a database with JDBC

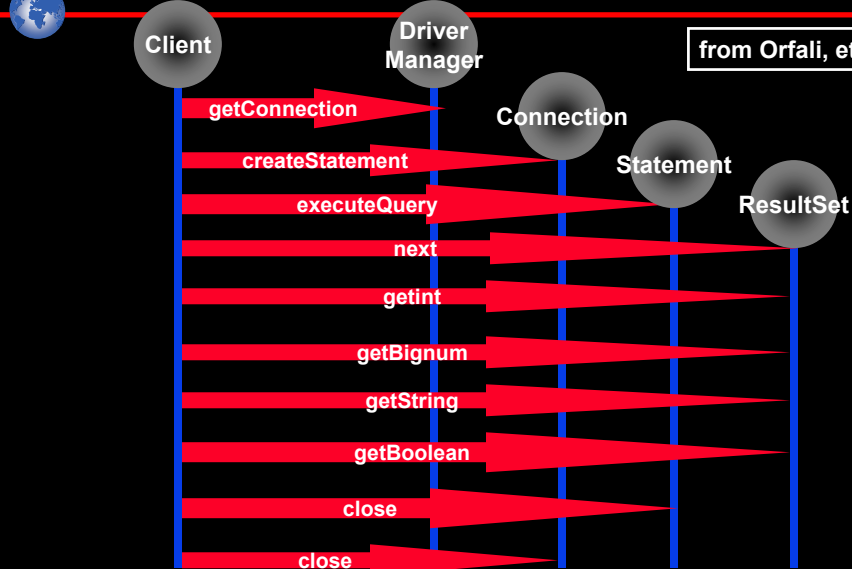


Type4 JDBC for a Java Applet



- Enables downloadable applets to connect directly to a database
- Java security requires that database be on the same host as the Web Server (restriction lifted for JDK 1.1 signed applets)

Invoking a Simple SQL Query



JDBC: Simple Example

```
Connection conn =
    DriverManager.getConnection
        ("jdbc:oracle:oci:scott/tiger@oracle");

Statement stmt = conn.createStatement ();
ResultSet rset =
    stmt.executeQuery ("select EMPNO from EMP");

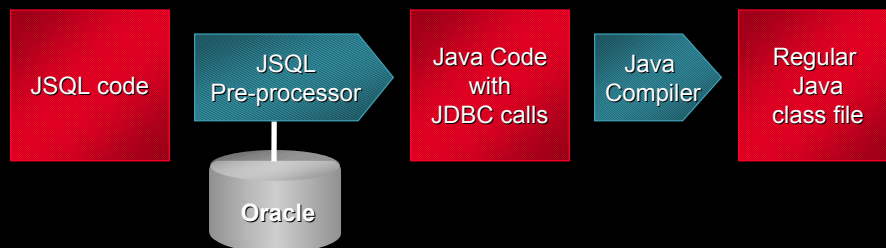
while (rset.next ()) {
    // Print the name
    System.out.println (rset.getInt ("EMPNO"));
}
```

Agenda

- Objectives
- JDBC
- JSQL
 - ➔ Overview
 - Examples
- Conclusions

19

JSQL



- JSQL allows developers to embed SQL in Java code
 - JSQL translates embedded SQL into Java code with JDBC calls.
 - Similar pre-processing mechanism as precompilers for C/C++ and Cobol.

20



JSQL: Advantages

- **Easier to write and maintain programs**
 - More compact code than JDBC
 - Compile-time type checking of SQL statements
 - Strongly typed cursors
- **Standards-compliant**
 - Resulting code is 100% compliant with Javasoft JDBC specification
 - Oracle driving a joint proposal with IBM and Tandem to define a standard for mixing Java and SQL

21



Agenda

- Objectives
- JDBC
- JSQL
 - Overview
- ➡ **Examples**
- Conclusions

22



More concise than JDBC

```
// J/SQL
#sql {create table EMP(EMPNO NUMBER (5))};

// JDBC
Statement stmt = conn.createStatement ();
stmt.execute("create table EMP(EMPNO NUMBER (5))");
```

- J/SQL supports a *default* connection
- JDBC Statement objects are hidden
- SQL statements can span several lines



Much more concise!

```
int n = 17950;

// J/SQL
#sql {insert into EMP values (:n)};

// JDBC
Statement stmt =
    conn.prepareStatement("insert into EMP values (?)");
stmt.setInt (1, n);
stmt.execute ();
```

- J/SQL can use Java variables in SQL statements



Early Type Checking

```
Date d = new Date ("11/4/1993");  
// J/SQL  
#sql {insert into EMP values (:d)};
```

- **Pre-processor checks Java types against SQL types**
 - Pre-processor raises a type mismatch error in this case
 - Equivalent JDBC program would only have raised a runtime error



Strongly typed Cursors

```
// Declare a Java type Empnos for a J/SQL cursor  
#sql cursor Empnos (int EMPNO);  
// Create a Java variable of type Empnos  
Empnos ecurs =  
    (Empnos) #sql cursor {select empno from EMP};  
// Use strongly typed accessors for columns  
while (ecurs.next ()) {  
    int n = ecurs.EMPNO ();  
    ...  
}
```

- **Strongly typed Cursors provide stronger typing**



Visual programming demo

27



Visual example

```
try {
    queryDataSet1.close();
    queryDataSet1.setQuery (
        new QueryDescriptor(queryDataSet1.getDatabase(),
            textFieldControl1.getText())
    );
    gridControl1.setDataSet(queryDataSet1);
}
catch (Exception e1) {
    e1.printStackTrace();
}
}
```

28



Agenda

- Objectives
- JDBC
- JSQL



Conclusions

29



Conclusions

- **Java can use JDBC to connect to Relational Database systems in a portable and efficient way**
 - High performance for Application servers
 - Pure Java versions for applets
- **Higher level interfaces such as JSQL can be built on top of JDBC for higher productivity**

30



Agenda

- Objectives
- JDBC
- JSQL
- Conclusions

 **Q&A**

31



ORACLE®
Enabling the Information Age™