JAVA DAYS ´98

JDBC 2.0

Marc Van Cappellen
Development Manager
marc_vancappellen@intersolv.com

Sun microsystems

JDBC is the premiere database connectivity API for Java

- JDBC is there for about 3 years
- A dozen of drivers
  - Most DBMSs are accessible
  - Multiple drivers for all major DBMSs
- 100 % Pure Java certification
  - INTERSOLV - SequeLink Java Edition
  - Sybase - JConnect
  - and more ...

JDBC is the premiere database connectivity API for Java

- **Universal Access**
  - **Web browser**
  - **Desktop application**
  - **Application server**
  - **Database server**
- **Foundation for higher level components**
  - **SQLJ (Embedded SQL for Java)**
  - **Enterprise JavaBeans**
  - **Java Blend**

JDBC 2.0 Goals

- Compatibility with JDBC 1.0
- Advanced Database Features
  - Scrollable cursors
  - SQL3 data types
  - Java-relational DBMSs
- Better manageability (JNDI support)
- JavaBeans Integration
- Improve Performance

JDBC 2.0 Overview

- **JDBC 2.0 Core API**
  - java.sql
  - Additions to existing JDBC interfaces
  - A few new classes and interfaces
- **JDBC 2.0 Standard Extensions**
  - javax.sql
  - Closely related to other Java Standard Extensions
  - Keep the JDBC Core API small and focused
  - Standard extensions are downloadable
  - Extensions are not tied to a JDK release

JDBC 2.0 Overview

- Result set enhancements — core
- Batch updates — core
- Advanced data types — core
- JNDI and data sources — ext
- Rowsets — ext
- Connection Pooling — ext
- Distributed transaction support * — ext

* not covered

Sun microsystems

JDBC 2.0 Overview

- Result set enhancements
- **Batch updates**
- **Advanced data types**
- **JNDI and data sources**
- **Rowsets**
- **Connection Pooling**
- **Distributed transaction support**

Sun microsystems

Result set enhancements

- **New functionality for result sets**
  - **Scrolling**
  - **Updatability**
- **Result set types**
  - **forward-only**
  - **scroll-insensitive**
    - **static view of the data**
    - **membership, order and values are fixed**
  - **scroll-sensitive**
    - **dynamic view of the data**
    - **membership and order may be fixed**
- **Concurrency types**
  - **read-only**
  - **updatable**

Most important methods

- **Scrolling**
  - next ()
  - beforeFirst ( )
  - afterLast ()
  - first ()
  - last ()
  - previous ()
  - absolute (int)
  - relative (int)
- **Updating**
  - updateXXX ()
  - updateRow ()
  - deleteRow ()
  - moveToInsertRow ()
  - insertRow ()
- **Miscellaneous**
  - wasDeleted ()
  - refreshRow ()
  - getType ()
  - getConcurrency ()

Code example


```
// Create a scrollable and update statement
stmt =
con.createStatement(ResultSet.TYPE_SCROLL_INSENSI
TIVE,
                     ResultSet.CONCUR_UPDATABLE);
rs = stmt.executeQuery(
             “SELECT ename, salary FROM emp”);


// Create an equivalent prepared statement
pStmt = con.prepareStatement(
             “SELECT ename, salary FROM emp”,
             ResultSet.TYPE_SCROLL_INSENSITIVE,
             ResultSet.CONCUR_UPDATABLE);
rs = pStmt.executeQuery();
```

Code example


```
// Move backwards through a result set
rs.afterLast();
while (rs.previous()) {
      System.out.println(rs.getString(1));
      System.out.println(rs.getFloat(2));
}

// Update all records in the result set
rs.beforeFirst();
while (rs.next()) {
      Float salary = rs.getFloat(2));
      salary = salary * 1.1;
      rs.updateFloat(2, salary);
      rs.updateRow();
}
```

Result set enhancements

- Alternative to positioned update and delete
- Result sets can be downgraded by the driver
- Database metadata
- Scrollable cursors are not for free!

JDBC 2.0 Overview

- **Result set enhancements**
- Batch updates
- **Advanced data types**
- **JNDI and data sources**
- **Rowsets**
- **Connection Pooling**
- **Distributed transaction support**

Sun microsystems

Batch updates

- Multiple statements submitted at once
- Reduces the number of round-trips to the database
- Statement
  - Execute multiple statements at once
- Prepared, CallableStatement
  - Single statement
  - Multiple parameter sets
- Restrictions
  - No result set generating statements
  - No stored procedures with out or in/out parameters (Function results are out parameters)

Code example

```
// Create a Statement
stmt = con.createStatement();

// Add multiple SQL statements to the batch
stmt.addBatch("INSERT INTO employees VALUES
(...)");
stmt.addBatch("INSERT INTO dept VALUES (...)");
stmt.addBatch("UPDATE employees SET ...");
stmt.addBatch("DELETE FROM employees WHERE
...");

// Execute the batch
int [] rowCounts = stmt.executeBatch();
```

Code example

```
// Create a PreparedStatement
pStmt = con.prepareStatement(
            “INSERT INTO employees VALUES
(?,?)”);

// Add multiple parameter sets to the batch
pStmt.setInt(1, 1000);
pStmt.setString(2, “Kelly Kaufmann”);
pStmt.addBatch();
pStmt.setInt(1, 2000);
pStmt.setString(2, “Bill Barnes”);
pStmt.addBatch();

// Execute the batch
int [] rowCounts = pStmt.executeBatch();
```

Miscellaneous

- **Disable autocommit**
- **BatchUpdateException**
  - **subclass of SQLException**
  - **getUpdateCounts ()**
  - **Still catch SQLException**

JDBC 2.0 Overview

- **Result set enhancements**
- **Batch updates**
- Advanced data types
- **JNDI and data sources**
- **Rowsets**
- **Connection Pooling**
- **Distributed transaction support**

Persistence of Java Objects

A Java-relational DBMS is a Java aware database.

It extends the type system of the database with Java object types.

Sun microsystems

Code example

```
// Assume a table personnel, having a column
// employee containing instances of the Java
class
// Employee.

// Fetch an Employee
rs = stmt.executeQuery(
            “SELECT employee FROM personnel”);
rs.next();
Employee emp = (Employee) rs.getObject(1);


// Insert an Employee
// Assume that the Employee ‘emp’ is already
created
pStmt = con.prepareStatement(
            “INSERT into employee VALUES (?)”);
pStmt.setObject(1, emp);
pStmt.executeUpdate();
```

Persistence of Java Objects

- Easy to use, but powerful
- Only JDBC 1.0 classes and methods are used in the previous example
- JDBC does not specify the SQL language extensions
- The bytecodes for the objects, stored in the database, are loaded via the usual Java language mechanism

SQL3 Types

- **BLOB, CLOB**
  - **java.sql.Blob / java.sql.Clob**
  - **LOCATORs**
- **Constructed types**
  - **REF(strutured_type)**
  - **base_type ARRAY [n]**
  - **java.sql.Ref**
  - **java.sql.Array**

SQL3 Types

- **Distinct types**
  - **CREATE TYPE MONEY AS NUMERIC(10,2)**
  - **getXXX ()**
  - **setXXX ()**
- **Structured types**
  - **CREATE TYPE POINT (X FLOAT, Y FLOAT)**
  - **java.sql.Struct**
  - **getObject ()**
  - **setObject ()**
- **Customizable Mapping**
  - **Mapping of SQL types to Java Objects**
  - **Mapping is per connection**
  - **Seamless extension of get/setObject**

JDBC 2.0 Overview

- **Result set enhancements**
- **Batch updates**
- **Advanced data types**
- JNDI and data sources
- **Rowsets**
- **Connection Pooling**
- **Distributed transaction support**

JNDI and data sources

- **JDBC 1.0**
  - Application specifies JDBC Driver and URL
  - JDBC Driver specific
  - Maintainability
  - URLs can be cryptic
- **JDBC data sources**
  - A factory for JDBC connections
  - A JavaBean
  - Properties
    - Description
    - NetworkProtocol
    - UserName
    - DatabaseName
    - ...

JNDI and data sources

- JNDI provides a uniform way to find and access services on a network
- JDBC data sources are registered in the jdbc subcontext
- Data source objects can be created and managed, separately from the JDBC applications

Code example

```
// Creating a data source
SampleDataSource ds = new SampleDataSource();
ds.setDatabasName(“my_database_name”);
cd.setNetworkProtocol(“TCP/IP”);
...
Context cntxt = new InitialContext();
cntxt.bin(“jdbc/Accounting”, ds);


// Connecting to a data source
Context cntxt = new InitialContext();
DataSource ds = (DataSource)cntxt.lookup(
                                “jdbc/Accounting”);
Connection con = ds.getConnection(“scott”,
“tiger”);
```

JDBC 2.0 Overview

- Result set enhancements
- Batch updates
- Advanced data types
- JNDI and data sources
- Rowsets
- Connection Pooling
- Distributed transaction support

Sun microsystems

Rowsets

- A rowset
  - encapsulates a set of rows
  - is a JavaBean component
  - is a ResultSet
  - does accept parameters
- Properties
  - DataSourceName
  - TransactionIsolation
  - Command
  - MaxRows
  - ....
- Events
  - Cursor Movement
  - Insert/Update/Delete
  - Rowset changed

Code example

```
// Suppose we have a RowSet rset

// At design time...
rset.SetDataSourceName("jdbc/SomeDataSourceName");
rset.setCommand("SELECT * FROM emp WHERE ename
like ?");


// At run time...
rset.setString(1, "F%");
rset.execute();
...
rs.beforeFirst();
while (rs.next()) {
       System.out.println(rs.getString(1));
}
```

Rowset Implementations

- **CachedRowSet**
  - disconnected, serializable and scrollable RowSet
  - Augment the JDBC driver capabilities
  - only appropriate for small # records
  - can be passed between components of a distributed application
  - optimistic concurrency
  - multiple cursors over a single set of rows (createShared)
- **JDBCRowSet**
  - connected RowSet
  - A thin JavaBean layer that wraps a JDBC ResultSet
  - No additional functionality is provided
- **WebRowSet**
  - client/server implementation of a RowSet
  - WebRowSet <HTTP> Servlet <JDBC> DBMS

JDBC 2.0 Overview

- Result set enhancements
- Batch updates
- Advanced data types
- JNDI and data sources
- Rowsets
- Connection Pooling
- Distributed transaction support

Sun microsystems

Connection Pooling

- A connection pool is a cache of database connections
  - Connections may be reused
  - Increasing performance, especially in middle-tier server environments
- JDBC 2.0 provides the hooks
  - there are a large # of possible algorithms
  - JDBC driver must provide the hooks using the interfaces ConnectionPoolDataSource and PooledConnection
  - Middle-tier server implements it on top of the JDBC driver using the interface DataSource
  - Refer to the JDBC specs for detailed information

Connection Pooling

- **Connection pooling doesn t impact application code!!!**
- **Use data sources / JNDI as you are used to**

```
// Get a connection, pooling is done internally
DataSource ds = (DataSource)cntxt.lookup(
                                        “jdbc/Accounting”);
Connection con = ds.getConnection(“scott”, “tiger”);

// Do whatever JDBC work...

// Close the connection, the physical database
// connection is returned to the pool.
con.close();
```

JDBC 2.0 Overview

- Result set enhancements
- Batch updates
- Advanced data types
- JNDI and data sources
- Rowsets
- Connection Pooling
- Distributed transaction support