

Database Access with JDBC

MSIS 531 – Spring 2006

Overview

2

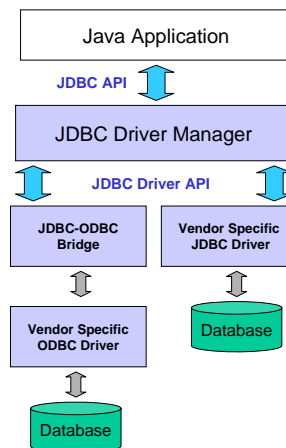
- Overview of JDBC technology
- JDBC drivers
- Seven basic steps in using JDBC
- Retrieving data from a ResultSet
- Using prepared and callable statements
- Manual transaction control

JDBC Introduction

- Standard Java library for accessing relational databases
 - JDBC standardizes:
 - Establishing connection to database
 - Approach to initiating queries
 - Methods for creating stored (parameterized) queries
 - The data structures for query results (table)
 - Determining the number of columns, metadata, etc.
 - API does not standardize SQL syntax
 - JDBC is not embedded SQL
 - JDBC classes are in the java.sql package

JDBC Drivers

- JDBC consists of two parts:
 - The JDBC API, a purely Java-based API
 - JDBC Driver Manager, which communicates with vendor-specific drivers that interact with the database
 - Point: translation to vendor format is performed on the client
 - No changes needed to server
 - Driver (translator) needed on client



JDBC Data Types

JDBC Type	Java Type
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	
BINARY	byte[]
VARBINARY	
LONGVARBINARY	
CHAR	String
VARCHAR	
LONGVARCHAR	

JDBC Type	Java Type
NUMERIC	BigDecimal
DECIMAL	
DATE	java.sql.Date
TIME	java.sql.Timestamp
TIMESTAMP	
CLOB	Clob*
BLOB	Blob*
ARRAY	Array*
DISTINCT	mapping of underlying type
STRUCT	Struct*
REF	Ref*
JAVA_OBJECT	underlying Java class

*SQL3 data type supported in JDBC 2.0

Steps in Using JDBC

1. Load the JDBC driver
2. Define the Connection URL
3. Establish the Connection
4. Create a Statement object
5. Execute a query
6. Process the results
7. Close the connection

JDBC: Details of Process

1. Load the driver

```
try {
    Class.forName("connect.microsoft.MicrosoftDriver");
    Class.forName("oracle.jdbc.driver.OracleDriver");
} catch { ClassNotFoundException cnfe) {
    System.out.println("Error loading driver: " cnfe);
}
```

2. Define the Connection URL

```
String host = "dbhost.yourcompany.com";
String dbName = "someName";
int port = 1234;
String oracleURL = "jdbc:oracle:thin:@" + host +
    ":" + port + ":" + dbName;
```

JDBC: Details of Process (cont.)

3. Establish the Connection

```
String username = "jay_debese";
String password = "secret";
Connection connection =
    DriverManager.getConnection(oracleURL, username,
        password);
```

- Optionally, look up information about the database

```
DatabaseMetaData dbMetaData =
    connection.getMetaData();
String productName =
    dbMetaData.getDatabaseProductName();
System.out.println("Database: " + productName);
```

JDBC: Details of Process (cont.)

4. Create a Statement

```
Statement statement =
    connection.createStatement();
```

5. Execute a Query

```
String query =
    "SELECT col1, col2, col3 FROM sometable";
ResultSet resultSet =
    statement.executeQuery(query);
```

- To modify the database, use `executeUpdate`, supplying a **UPDATE**, **INSERT**, or **DELETE** statement
- Use `setQueryTimeout` to specify a max wait delay

JDBC: Details of Process (cont.)

6. Process the Results

```
while(resultSet.next()) {
    System.out.println(resultSet.getString(1) + " " +
        resultSet.getString(2) + " " +
        resultSet.getString(3));
}
```

- First column has index 1, not 0
- The `ResultSet` provides `getXxx` methods that take a column index *or column name* and returns the data
- Can also access result meta data (column names, etc.)

7. Close the Connection

```
connection.close();
```

- Since opening a connection is expensive, postpone this step if additional database operations are expected

Using Microsoft Access via JDBC

- JDK 1.4 includes a driver that “bridges” JDBC and ODBC, allowing access to any ODBC data source
 - Not for production or high-volume use, but nice for proof of concept
- Use handout to download Northwind and set up an ODBC connection
- NorthwindTest sample application gives a command-line method for testing the connection
 - Note use of try/catch blocks for driver load, SQL execution

Using JDBC MetaData

- Metadata: “data about the data”
 - Not “How many customers have sales over \$100K?” but “How many columns have a datatype of BIT (boolean)?”
- System-wide data
 - `connection.getMetaData().getDatabaseProductName()`
 - `connection.getMetaData().getDatabaseProductVersion()`
- Table-specific data
 - `resultSet.getMetaData().getColumnCount()`
 - When using the result, remember that the index starts at 1, not 0
 - `resultSet.getMetaData().getColumnName()`

Using MetaData

```

private void showTable(String driver,
                      String url,
                      String username,
                      String password,
                      String tableName,
                      PrintWriter out) {
    try {
        Class.forName(driver);
        Connection connection =
            DriverManager.getConnection(url, username, password);
        DatabaseMetaData dbMetaData = connection.getMetaData();
        out.println("<UL>");
        String productName = dbMetaData.getDatabaseProductName();
        out.println("  <LI><B>Database:</B> " + productName);
        String productVersion =
            dbMetaData.getDatabaseProductVersion();
        out.println("  <LI><B>Version:</B> " + productVersion);
        out.println("</UL>");
        ...
    }
}

```

Using MetaData (cont.)

```

Statement statement = connection.createStatement();
String query =
    "SELECT * FROM " + tableName;
ResultSet resultSet = statement.executeQuery(query);
out.println("<TABLE BORDER=1>");
ResultSetMetaData resultsMetaData = resultSet.getMetaData();
int columnCount = resultsMetaData.getColumnCount();
out.println("<TR>");
for(int i=1; i<columnCount+1; i++) {
    out.print("<TH>" + resultsMetaData.getColumnName(i));
}
while(resultSet.next()) {
    out.println("<TR>");
    for(int i=1; i<columnCount+1; i++) {
        out.print("<TD>" + resultSet.getString(i));
    }
    out.println();
}
out.println("</TABLE>");

```

Using JDBC Statements

- Overview
 - Through the `statement` object, SQL statements are sent to the database
 - Three types of statement objects are available:
 - `Statement`
 - For executing a **simple SQL statement**
 - `PreparedStatement`
 - For executing a **precompiled SQL statement** passing in parameters
 - `CallableStatement`
 - For executing a database **stored procedure**

Useful Statement Methods

- `executeQuery`
 - Executes the SQL query and returns the data in a table (`ResultSet`)
 - The resulting table may be empty but never null

```
ResultSet results =
statement.executeQuery("SELECT a, b FROM table");
```
- `executeUpdate`
 - Execute for INSERT, UPDATE, or DELETE statements
 - Returns number of rows affected
 - Supports Data Definition Language (DDL) statements CREATE TABLE, DROP TABLE and ALTER TABLE

```
int rows =
statement.executeUpdate("DELETE FROM EMPLOYEES" +
"WHERE STATUS=0");
```

Useful Statement Methods (cont.)

- `execute`
 - Execute stored procedures, prepared statements
 - Execution may or may not return a `ResultSet` (use `statement.getResultSet()`). If the return value is true, two or more result sets were produced
- `getMaxRows/setMaxRows`
 - Maximum number of rows a `ResultSet` may contain
 - Unless explicitly set, the number of rows is unlimited (return value of 0)
- `getQueryTimeout/setQueryTimeout`
 - Specifies the amount of a time a driver will wait for a `STATEMENT` to complete before throwing a `SQLException`

Prepared Statements

- If executing similar SQL multiple times, using “prepared” (parameterized) statements may be more efficient
 - Statement is created in standard form that is sent to the database for compilation before actually being used
 - Each time you use it, you simply replace some of the marked parameters using the `setXxx` methods
- As `PreparedStatement` inherits from `Statement`, the corresponding execute methods have no parameters
 - `execute()`, `executeQuery()`, `executeUpdate()`

Prepared Statement Example

```

Connection connection =
    DriverManager.getConnection(url, user, password);
PreparedStatement statement =
    connection.prepareStatement("UPDATE employees "+
                                "SET salary = ? " +
                                "WHERE id = ?");

int[] newSalaries = getSalaries();
int[] employeeIDs = getIDs();
for(int i=0; i<employeeIDs.length; i++) {
    statement.setInt(1, newSalaries[i]);
    statement.setInt(2, employeeIDs[i]);
    statement.executeUpdate();
}

```

Callable Statements

- Callable statements allow execution of server-side code on the RDBMS server
 - So-called “stored procedures”
 - Code created and maintained on server
 - Benefit is performance, cost is yet another language to learn
- Stored procedures are like Java methods—they take input parameters and can return results (possibly more than one)
 - Setup to pass parameters can be painful

Transactions

- By default, after each SQL statement is executed the changes are **automatically committed** to the database
 - Turn auto-commit off to group two or more statements together into a transaction:
`connection.setAutoCommit(false);`
 - Call **commit** to permanently record the changes to the database after executing a group of statements
 - Call **rollback** if an error occurs

Useful Transaction Methods

- These methods apply to the **Connection**
- `getAutoCommit/setAutoCommit`
 - By default, a connection is set to auto-commit
 - Retrieves or sets the auto-commit mode
- `commit`
 - All changes since last commit to become permanent
 - Database locks held by this **Connection** are released
- `rollback`
 - Drops all changes since the previous call to commit
 - Releases database locks held by this **Connection** object

Transactions: Example

```
Connection connection =
    DriverManager.getConnection(url, username, passwd);
connection.setAutoCommit(false);
try {
    statement.executeUpdate(...);
    statement.executeUpdate(...);

    connection.commit();
} catch (Exception e) {
    try {
        connection.rollback();
    } catch (SQLException sqle) {
        // report problem
    }
} finally {
    try {
        connection.close();
    } catch (SQLException sqle) { }
}
```