CS 580 Client-Server Programming
Fall Semester, 2002
Doc 23 JDBC
Contents

SQL and Java 2
JDBC 4
  JDBC Architecture 8
  Using JDBC 11
  JDBC Drivers 14
  Queries 19
  Threads & Connections 25
  SQL Data Types and Java 26
  Transactions 27
  PreparedStatement 29
  CallableStatement 31

References

http://java.sun.com/j2se/1.4/docs/guide/jdbc/index.html Sun's on-line JDBC Tutorial & Documentation

Client/Server Programming with Java and CORBA, Orfali and Harkey, John Wiley and Sons, Inc. 1997

PostgreSQL JDBC Documentation, http://www.ca.postgresql.org/users-lounge/docs/7.2/postgres/jdbc.html

SQL and Java
Some Jargon

SQL Access Group (SAG) - multivendor "Standards" group

SQL Call Level Interface (CLI)
  SAG standard for remote connects to a database

CLI uses drivers to the database

Program uses a driver manager to talk to the driver

The driver is database specific

In 1994 X/Open adopted SQL CLI to produce X/Open CLI

In 1996 X/Open CLI was adapted by ISO to become ISO 9075-3 Call level Interface

Microsoft's Open Database Connectivity (ODBC)

Extension of the SAG CLI

ODBC 2.0 (32 bit) has three conformance levels

- Core
  23 API calls for basic SQL stuff

- Level 1
  19 API calls for large objects (BLOBs) and driver-specific

- Level 2
  19 API calls for scrolling (cursors)

JDBC
Java Database Connectivity

Sun states

JDBC is a trademark and
Not an abbreviation for Java Database Connectivity

JDBC is a portable SQL CLI written in Java.

Versions of JDBC

- JDBC 1.x
- JDBC 2.x
- JDBC 3.0

JDBC 1.x

Basic SQL functionality

JDBC 2.1 Core

Standard part of JDK 1.2

JDBC drivers must implement JDBC 2.x before you can use it

MySQL driver for JDBC 2.x is in pre-beta release

Additional Features

- Scrollable result sets
- Updateable result sets
  Can change the result of a query locally & in database
- Batch updates
- BLOB, CLOB support

JDBC 2.0 Package

Now java.sql

Once was optional Java package javax.sql

- Java Naming and Directory Interface (JNDI) support
- Connection pooling
- Distributed transactions
- JavaBean RowSets
  Access any tabular data (files, spreadsheets)
  Make old drivers scrollable & updateable
  Wraps JDBC driver for use in GUI

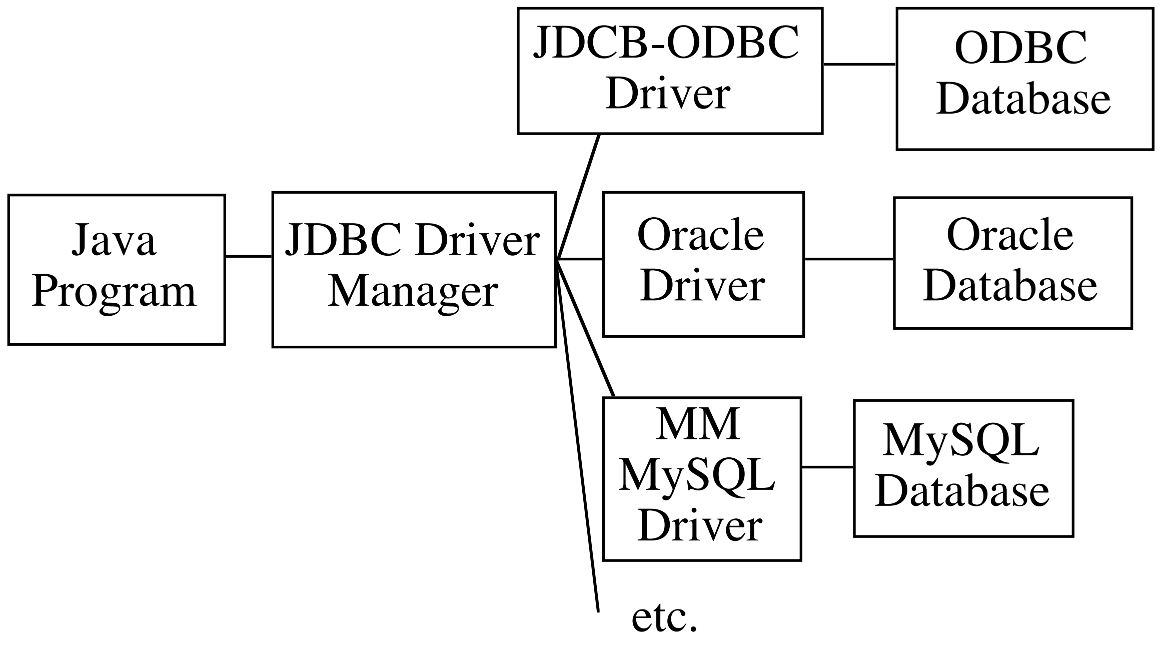
JDBC 3.0

java.sql & javax.sql in JDK 1.4

Most advanced features are in javax.sql

- Set, release, or rollback a transaction to designated savepoints
- Reuse of prepared statements by connection pools
- Connection pool configuration
- Retrieval of parameter metadata
- Retrieval of auto-generated keys
- Ability to have multiple open ResultSet objects
- Passing parameters to CallableStatement objects by name
- Holdable cursor support
- BOOLEAN data type
- Making internal updates to the data in Blob and Clob objects
- Retrieving and updating the object referenced by a Ref object
- Updating of columns containing BLOB, CLOB, ARRAY and REF types
- DATALINK/URL data type
- Transform groups and type mapping
- DatabaseMetadata APIs

JDBC Architecture



JDBC driver provides connections to database via drivers

JDBC Drivers, JDBC Versions & Java API

java.sql.* is mainly interfaces for JDBC Drivers

The driver for the database determines the actual functionality

Sample JDBC Use

```
import java.sql.*;

public class SampleConnection
   {
    public static void main (String args[]) throws Exception
      {
      String dbUrl = "jdbc:postgresql://rugby.sdsu.edu/cs580";
      String user = "whitney";
      String password = "don’tyouwish";
      System.out.println("Load Driver!");

      Class.forName("org.postgresql.Driver");
      Connection rugby;
      rugby = DriverManager.getConnection( dbUrl, user, password);
      Statement getTables = rugby.createStatement();
      ResultSet tableList =
         getTables.executeQuery("SELECT * FROM pg_tables");
      while (tableList.next() )
         System.out.println("Result: " + tableList.getString(1));
      rugby.close();
      }
   }
```

Using JDBC

Step 1. Load the driver(s)

Step 2. Connect to the database

Step 3. Issue queries and receive results

Loading a Driver

The most commonly used way

A well-written JDBC driver is loaded using Class.forName

To load the Oracle driver

```
import java.sql.*;

class JdbcTest
{
  public static void main (String args [])  throws
                    ClassNotFoundException
  {
    Class.forName ("oracle.jdbc.OracleDriver");
  }
}
```

This requires that oracle package be in your path

A properly written driver will register itself with the DriverManager class

Loading a Driver

The Recommended Way

Use the command line to specify the driver

```
java –Djdbc.drivers=org.postgresql.Driver yourProgramName
```

Makes it easier to change database vendors with out recompiling the code

Long command lines need script to run

JDBC Drivers

Java supports four types of JDBC drivers

1. JDBC-ODBC bridge plus ODBC driver
   Java code access ODBC native binary drivers
   ODBC driver accesses databases
   ODBC drivers must be installed on each client

2. Native-API partly-Java driver
   Java code accesses database specific native binary drivers

3. JDBC-Net pure Java driver
   Java code accesses database via DBMS-independent net protocol

4. Native-protocol pure Java driver
   Java code accesses database via DBMS-specific net protocol

JDBC URL Structure

jdbc:<subprotocol>:<subname>

<subprotocol>
Name of the driver or database connectivity mechanism

<subname>
Depends on the <subprotocol>, can vary with vender

If connection goes over Internet subname is to contain net URL

```
jdbc:mysql://fargo.sdsu.edu:5555/WHITNEYR
```

ODBC Subprotocol

```
jdbc:odbc:<data-source-name>[;<attribute-name>=<attribute-value>]*
```

Examples

```
jdbc:odbc:qeor7
jdbc:odbc:wombat
jdbc:odbc:wombat;CacheSize=20;ExtensionCase=LOWER
jdbc:odbc:qeora;UID=kgh;PWD=fooey
```

PostgreSQL Subprotocol

jdbc:postgresql:*database*

jdbc:postgresql://*host*/*database*

jdbc:postgresql://*host*:*port*/*database*

DriverManager.getConnection - Using JDBC URL

Three forms:

```
getConnection(URL, Properties)
getConnection(URL, userName, Password)
getConnection(URLWithUsernamePassword)
```

Form 1

```
static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";

DriverManager.getConnection(ARS_URL, "whitney","secret");
```

Form 2

```
DriverManager.getConnection(
   "jdbc:oracle:whitney/secret@PutDatabaseNameHere");
```

Form 3

```
java.util.Properties info = new java.util.Properties();
info.addProperty ("user", "whitney");
info.addProperty ("password","secret");

DriverManager getConnection (ARS_URL ,info );
```

java.sql.DriverManager

Driver related methods

- deregisterDriver(Driver)
- getDriver(String)
- getDrivers()
- registerDriver(Driver)

Connecting to a database

- getConnection(String, Properties)
- getConnection(String, String, String)
- getConnection(String)

- getLoginTimeout()
- setLoginTimeout(int)

Logging/tracing/Debugging

- getLogStream()
- setLogStream(PrintStream)
- println(String)

  Print a message to the current JDBC log stream

Queries

```
Connection toFargo =
   DriverManager.getConnection(database, user, password);

Statement namesTable = toFargo.createStatement();

ResultSet namesFound =
   namesTable.executeQuery("SELECT * FROM name");
```

executeUpdate
  Use for INSERT, UPDATE, DELETE or SQL that return nothing

executeQuery
  Use for SQL (SELECT) that return a result set

execute
  Use for SQL that return multiple result sets
  Uncommon
  Stored procedures can return

ResultSet - Result of a Query

JDBC returns a ResultSet as a result of a query

A ResultSet contains all the rows and columns that satisfy the SQL statement

A cursor is maintained to the current row of the data

The cursor is valid until the ResultSet object or its Statement object is closed

next() method advances the cursor to the next row

You can access columns of the current row by index or name

ResultSet has getXXX methods that:

- have either a column name or column index as argument
- return the data in that column converted to type XXX

Some Result Set Issues

What happens when we call next() too many time?

What happens before we call next

Example

Name Table

first	last
roger	whitney
pete	stanley
rat	cat

Sample Table

col
a
b

Two Queries

```
public class SampleMySQL {
  public static void main(String[] args) throws Exception {
    Class.forName("org.gjt.mm.mysql.Driver").newInstance();
    String database = "jdbc:mysql://fargo.sdsu.edu:5555/foo";
    Connection toFargo =
      DriverManager.getConnection(database, "foo", "bar");
    Statement namesTable = toFargo.createStatement();

    ResultSet namesFound =
      namesTable.executeQuery("SELECT * FROM name");
    for (int k = 0;k< 3;k++) {
      System.out.println( "first: " + namesFound.getString( 1));
      namesFound.next();
    }
    for (int k = 0;k< 3;k++) {
      sample.next();
      System.out.println( "col: " + sample.getString( 1));
    }
    toFargo.close();
    }
  }
```

Result

```
first: roger
first: roger
first: pete
col: a
col: b
col: b
```

Mixing ResultSets

Can't have two active result sets on same statement

```
Statement namesTable = toFargo.createStatement();

ResultSet namesFound =
   namesTable.executeQuery("SELECT * FROM name");
ResultSet sample =
   namesTable.executeQuery("SELECT * FROM sample");
for (int k = 0;k< 3;k++) {
   namesFound.next();
   sample.next();
   System.out.println( "first: " + namesFound.getString( 1));
   System.out.println( "col: " + sample.getString( 1));
}
```

Result

```
first: roger
col: a
first: roger
col: b
first: roger
col: b
```

Use Two Statements

```
Connection toFargo =
   DriverManager.getConnection(database, user, password);

Statement namesTable = toFargo.createStatement();
Statement exampleTable = toFargo.createStatement();

ResultSet namesFound =
   namesTable.executeQuery("SELECT * FROM name");
ResultSet sample =
   exampleTable.executeQuery("SELECT * FROM sample");
for (int k = 0;k< 3;k++) {
   namesFound.next();
   sample.next();
   System.out.println( "first: " + namesFound.getString( 1));
   System.out.println( "col: " + sample.getString( 1));
}
```

Result

```
first: roger
col: a
first: pete
col: b
first: rat
col: b
```

Threads & Connections

Some JDBC drivers are not thread safe

If two threads access the same connection results may get mixed up

PostgreSQL driver is thread safe

When two threads make a request on the same connection

- The second thread blocks until the first thread get it its results

Can use more than one connection but

- Each connection requires a process on the database

SQL Data Types and Java

SQL type	**Java type**
CHAR	String
VARCHAR	String
LONGVARCHAR	String
NUMERIC	java.math.BigDecimal
DECIMAL	java.math.BigDecimal
BIT	boolean
TINYINT	byte
SMALLINT	short
INTEGER	int
BIGINT	long
REAL	float
FLOAT	double
DOUBLE	double
BINARY	byte[]
VARBINARY	byte[]
LONGVARBINARY	byte[]
DATE	java.sql.Date
TIME	java.sql.Time
TIMESTAMP	java.sql.Timestamp

Transactions

A transaction consists of one or more statements that have been executed and completed

A transaction ends when a commit or rollback is sent

Connections are opened in auto commit mode:

when a statement is completed, it is committed

Transactions and Concurrency

What happens to data that is changed in a transaction, but not yet committed?

Can other programs access the old or new values?

Use setTransactionIsolation(int) in Connection class to set access levels

Access levels are given as static fields of Connection class

```
TRANSACTION_NONE
TRANSACTION_READ_COMMITTED
TRANSACTION_READ_UNCOMMITTED
TRANSACTION_REPEATABLE_READ
TRANSACTION_SERIALIZABLE
```

Transaction Example

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

class JdbcTest {
  static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";

  public static void main (String args [])  throws
      SQLException, ClassNotFoundException, IOException {
    Class.forName ("oracle.jdbc.OracleDriver");
    Connection ARS;
    ARS =DriverManager.getConnection(ARS_URL,
                "whitney", "secret");

    ARS.setAutoCommit(false);

   String floodProblem = DELETE FROM AirlineSchedule WHERE
                              from = 'FAR';

    String newflight =   INSERT INTO AirlineSchedule VALUES
                     ( 'DE', 'SAN', '8:00', '12:00', '909', 'A');

    Statement schedule = ARS.createStatement ();

    schedule.executeUpdate (floodProblem);
    schedule.executeUpdate (newflight);

    ARS.commit();
    ARS.close();
  }
}
```

PreparedStatement

PreparedStatement objects contain SQL statements that have been sent to the database to be prepared for execution

The SQL statements contains variables (IN parameters) which are given values before statement is executed

Only makes sense to use if database and driver keeps statements open after they have been committed

IN parameters are indicated by a ?

Values are set by position

```
String flightOut =    "SELECT * FROM AirlineSchedule
            WHERE from = ?";
```

PreparedStatement Example

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

class JdbcTest {
  static String ARS_URL = "jdbc:oracle:@PutDatabaseNameHere";

  public static void main (String args [])  throws
      SQLException, ClassNotFoundException, IOException {
    Class.forName ("oracle.jdbc.OracleDriver");
    Connection ARS;
    ARS =DriverManager.getConnection(ARS_URL,
                "whitney", "secret");

    String flightOut =    "SELECT * FROM AirlineSchedule
                WHERE from = ?";

    PreparedStatement schedule;
    schedule = ARS.preparedStatement (flightOut);

    schedule.setObject( 1, "SAN" );
    ResultSet fromSanDiego = schedule.executeQuery ();

    schedule. clearParameters();
    schedule.setObject( 1, "LAX" );
    ResultSet fromLA = schedule.executeQuery ();

  }
}
```

CallableStatement

Some databases have stored procedures (a procedure of SQL statements)

CallableStatement allows a Java program to invoke a stored procedure in a database

DatabaseMetaData

The class DatabaseMetaData allows you to obtain information about the database

The 113 methods in DatabaseMetaData give you more information than you thought possible