The Java Database Connectivity API (JDBC)

Stored Procedures

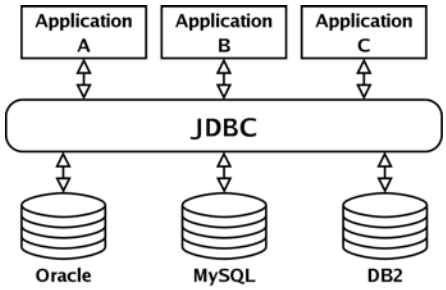
Three-Tier Architecture

Thanks to Stefan Büttcher for initial version of slides

University of Waterloo

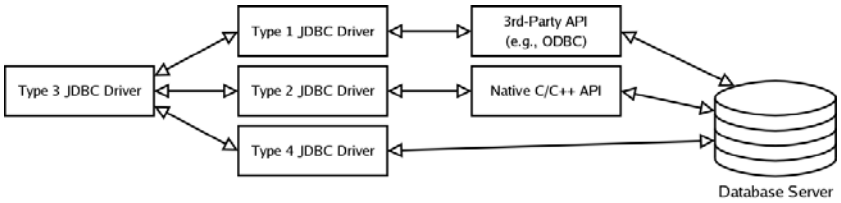
What is JDBC?

JDBC is a set of classes that can be used to uniformly access various databases. For different DBMS, different drivers are available. But from within your application, they all look the same.



Different Types of JDBC Drivers

Different types of drivers are available, communicating with the database server either directly or through other drivers.



How to use JDBC?

You must have the correct driver. Then:

1. Import all classes and interfaces in the java.sql package:

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

2. Register the driver with the JDBC DriverManager:

```
Class c = Class.forName("COM.ibm.db2.jdbc.net.DB2Driver");
DriverManager.registerDriver((Driver)c.newInstance());
```

or:

```
DriverManager.registerDriver(
    new COM.ibm.db2.jdbc.net.DB2Driver());
```

How to use JDBC?

3. Open a connection to the DB server:

```
Connection conn = DriverManager.getConnection(
    “jdbc:db2://rees.math.uwaterloo.ca:50448/cs448”,
    “db2guest”, “password_goes_here”);
```

In general:

```
Connection conn = DriverManager.getConnection(
   PROTOCOL + DATABASE, USERNAME, PASSWORD);
```

4. Send SQL queries to the server and process the results... (this is the actual program)

5. Close the connection to the DB server:

```
conn.close();
```

JDBC Classes

The most important classes (interfaces) in JDBC are:

Connection
Represents a single connection to the database server. A connection may consist of multiple transactions, but transactions cannot span over multiple connections.

Statement
Used to manage information about a single SQL statement.

ResultSet
Used to retrieve the DB server's results to an SQL query (Statement instance).

The Statement Class

The Statement class is used to send SQL queries to the database system.

```
Connection conn = .....
Statement s = conn.createStatement();
s.executeQuery("SELECT * FROM Course");
s.close();
conn.close();
```

But how to get the results???

The ResultSet Class

The ResultSet class is used to retrieve results to a query.

```
Connection conn = ...
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM Course");
while (rs.next()) {
   System.out.print(rs.getString(1) + " ");
   System.out.println(rs.getString(2));
}
s.close();
conn.close();
```

Using the ResultSet Class

Navigation inside a ResultSet instance is done using the methods:

- boolean next() *(returns true iff there are more data)*
- boolean previous() *(analogous to next())*
- void beforeFirst() *(moves cursor to beginning of res.)*
- void afterLast() *(moves cursor to end of result)*

Besides getString(int), you can also use the following methods to extract a value:
getInt, getBoolean, getDate, getBlob, getFloat, getTime,

No *indicator variables.* Use Java nulls.
`rs.wasNull()` after getXXX returns true if the retrieved value was null

Fancy Stuff: ResultSetMetaData

If you want to know how many columns a ResultSet instance contains, you can use the ResultSetMetaData class:

```
Connection conn = ...
Statement s = conn.createStatement();
ResultSet rs = s.executeQuery("SELECT * FROM Course");

ResultSetMetaData metaData = rs.getMetaData();
int colCount = metaData.getColumnCount();
while (rs.next()) {
   for (int i = 1; i <= colCount; i++)
        System.out.print(rs.getString(i) + " ");
   System.out.println("");
}
```

More Fancy Stuff: PreparedStatement

Sometimes, a sequence of very similar statements has to be sent. Query processing can be sped up by using the PreparedStatement class:

```
void printSalaryForEachPerson(String name[], int year[]) {
  PreparedStatement ps = conn.createPreparedStatement(
      "SELECT Salary FROM Income WHERE Name=? AND Year=?");
  for (int i = 0; i < names.length; i++) {
    ps.setString(1, name[i]);
    ps.setInt(2, year[i]);
    ResultSet rs = ps.executeQuery();
    if (rs.next())
      System.out.println(
          name[i] + " (" + year[i] + "): " + rs.getFloat(1));
    rs.close();
  }
  ps.close();
}
```

Less Fancy Stuff: executeUpdate

Sometimes, your query does not produce a result set (update operations – INSERT, UPDATE, DELETE).

```
Connection conn = ...
Statement s = conn.createStatement();

int rowsAffected = s.executeUpdate(
     "DELETE FROM Course WHERE C#='CS100'");

System.out.println(
    "" + rowsAffected + " rows deleted.");

s.close();
conn.close();
```

Errors

- Use JAVA exception handling mechanisms.
- The methods used in JDBC may throw `SQLException`
  - You have to catch it.
- Examples of provided methods:
  - SQL state: `exception.getSQLState()`
  - Text Message: `exception.getMessage()`
  - Vendor code: `exception.getErrorCode()`
  - … and much more

Stored Procedures

- What is a stored procedure:
  - Program executed through a single SQL statement
  - Executed in the process space of the server
- Advantages:
  - Can encapsulate application logic while staying "close" to the data
  - Reuse of application logic by different users
  - Avoid tuple-at-a-time return of records through cursors

Stored Procedures: Examples

```
CREATE PROCEDURE ShowNumReservations
   SELECT S.sid, S.sname, COUNT(*)
   FROM Sailors S, Reserves R
   WHERE S.sid = R.sid
   GROUP BY S.sid, S.sname
```

Stored procedures can have parameters:

- Three different modes: IN, OUT, INOUT

```
CREATE PROCEDURE IncreaseRating(
   IN sailor_sid INTEGER, IN increase INTEGER)

UPDATE Sailors
   SET rating = rating + increase
   WHERE sid = sailor_sid
```

Stored Procedures: Examples (Contd.)

Stored procedure do not have to be written in SQL:

```
CREATE PROCEDURE TopSailors(
   IN num INTEGER)
LANGUAGE JAVA
EXTERNAL NAME "file:///c:/storedProcs/rank.jar"
```

Calling Stored Procedures

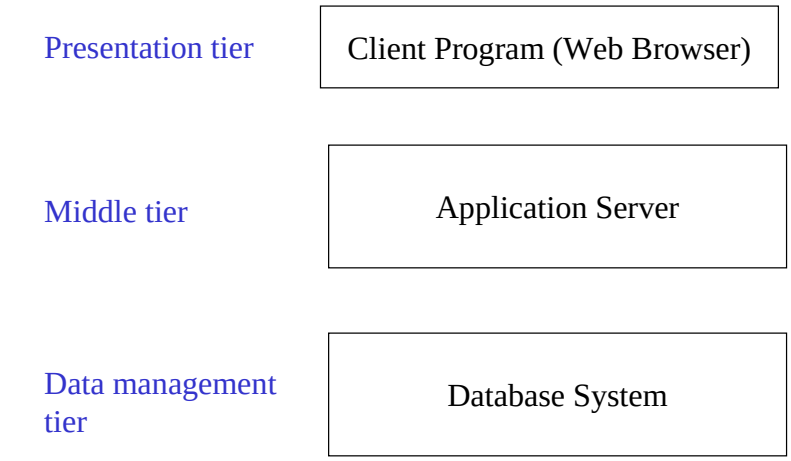
```
EXEC SQL BEGIN DECLARE SECTION
Int sid;
Int rating;
EXEC SQL END DECLARE SECTION

// now increase the rating of this sailor
EXEC CALL IncreaseRating(:sid,:rating);
```

Calling Stored Procedures (Contd.)

JDBC:

```
CallableStatement cstmt=
   con.prepareCall("{call ShowSailors}");
ResultSet rs = cstmt.executeQuery();
while (rs.next()) {
   …
}
```

The Three-Tier Architecture



The Three Layers

Presentation tier

- Primary interface to the user
- Needs to adapt to different display devices (PC, PDA, cell phone, voice access?)

Middle tier

- Implements business logic (implements complex actions, maintains state between different steps of a workflow)
- Accesses different data management systems

Data management tier

- One or more standard database management systems

Example 1: Airline reservations

- Build a system for making airline reservations
- What is done in the different tiers?
- Database System
  - Airline info, available seats, customer info, etc.
- Application Server
  - Logic to make reservations, cancel reservations, add new airlines, etc.
- Client Program
  - Log in different users, display forms and human-readable output

Example 2: Course Enrollment

- Build a system using which students can enroll in courses
- Database System
  - Student info, course info, instructor info, course availability, pre-requisites, etc.
- Application Server
  - Logic to add a course, drop a course, create a new course, etc.
- Client Program
  - Log in different users (students, staff, faculty), display forms and human-readable output