Database Connectivity
ODBC, JDBC and SQLJ
CS2312

What is ODBC?
- ODBC is (Open Database Connectivity):
  - A standard or open application programming interface (API) for accessing a database.
  - SQL Access Group, chiefly Microsoft, in 1992
- It allows programs to use SQL requests that will access databases without having to know the proprietary interfaces to the databases.
- ODBC handles the SQL request and converts it into a request the individual database system understands.

More on ODBC
- You need:
  - the ODBC software, and
  - a separate module or driver for each database to be accessed. Library that is dynamically connected to the application.
- Driver masks the heterogeneity of DBMS operating system and network protocol.
- E.g. (Sybase, Windows/NT, Novell driver)

ODBC Architecture

What is JDBC?
- JDBC is: Java Database Connectivity
  - is a Java API for connecting programs written in Java to the data in relational databases.
  - consists of a set of classes and interfaces written in the Java programming language.
  - provides a standard API for tool/database developers and makes it possible to write database applications using a pure Java API.
  - The standard defined by Sun Microsystems, allowing individual providers to implement and extend the standard with their own JDBC drivers.
- JDBC:
  - establishes a connection with a database
  - sends SQL statements
  - processes the results.

JDBC vs ODBC
- ODBC is used between applications
- JDBC is used by Java programmers to connect to databases
- With a small "bridge" program, you can use the JDBC interface to access ODBC-accessible databases.
- JDBC allows SQL-based database access for EJB persistence and for direct manipulation from CORBA, DJB or other server objects
JDBC API

- The JDBC API supports both two-tier and three-tier models for database access.
- Two-tier model -- a Java applet or application interacts directly with the database.
- Three-tier model -- introduces a middle-level server for execution of business logic:
  - the middle tier to maintain control over data access.
  - the user can employ an easy-to-use higher-level API which is translated by the middle tier into the appropriate low-level calls.

JDBC Architectures

![JDBC Architecture Diagram]

The JDBC Steps

1. Importing Packages
2. Registering the JDBC Drivers
3. Opening a Connection to a Database
4. Creating a Statement Object
5. Executing a Query and Returning a Result Set Object
6. Processing the Result Set
7. Closing the Result Set and Statement Objects
8. Closing the Connection

1: Importing Packages

```java
// Program name: LecExample_1a.java
// Purpose: Basic selection using prepared statement

// Import packages
import java.sql.*; // JDBC packages
import java.math.*;
import java.io.*;
import oracle.jdbc.driver.*;
```

2: Registering JDBC Drivers

```java
class LecExample_1a {
    public static void main (String args [])
        throws SQLException {
            // Load Oracle driver
            DriverManager.registerDriver (new
                oracle.jdbc.driver.OracleDriver());
        }
```

3: Opening connection to a Database

```java
// Prompt user for username and password
String user;
String password;
user = readEntry("username: ");
password = readEntry("password: ");

// Connect to the local database
Connection conn =
    DriverManager.getConnection
    ("jdbc:oracle:thin:@aardvark:1526:teach
    ", user, password);
```
4. Creating a Statement Object

// Query the hotels table for resort = 'palma nova'
// Please notice the essential trim
PreparedStatement pstmt =
    conn.prepareStatement("SELECT
    hotelname, rating FROM hotels WHERE
    trim(resort) = ?");
pstmt.setString(1, "palma nova");

5. Executing a Query, Returning a Result Set Object &
6. Processing the Result Set

ResultSet rset = pstmt.executeQuery();
// Print query results
while (rset.next())
    System.out.println(rset.getString(1) + " " + rset.getString(2));

7. Closing the Result Set and Statement Objects
8. Closing the Connection

// close the result set, statement, and the
connection
rset.close();
pstmt.close();
conn.close();
}

Mapping Data Types

There are data types specified to SQL that need to be
mapped to Java data types if the user expects Java to be
able to handle them.

Conversion falls into three categories:
• SQL type to Java direct equivalents
  SQL INTEGER direct equivalent of Java int data type.
  SQL type can be converted to a Java equivalent.
  SQL CHAR, VARCHAR, and LONGVARCHAR can all
  be converted to the Java String data type.
• SQL data type is unique and requires a special Java
data class object to be created specifically for their SQL
  equivalent.
  SQL DATE converted to the java Date object that is
defined in java.Date especially for this purpose.

What is SQLJ?

• SQLJ is a set of programming extensions that
  allow a programmer using the Java
programming language to embed statements
that provide SQL database requests.
• SQLJ is similar to existing extensions for SQL
  that are provided for C, FORTRAN, and other
programming languages.
• IBM, Oracle, and several other companies
  are proposed SQLJ as a standard and as a
simpler and easier-to-use alternative to
JDBC.

SQLJ Specifications

• The SQLJ specifications are in several parts:
  • SQLJ: Embedded SQL...Specifications for
    embedding SQL statements in Java
    methods.
  • SQLJ: SQL Routines...Specifications for
calling Java static methods as SQL stored
  procedures and user-defined functions.
  • SQLJ: SQL Types...Specifications for using
    Java classes as SQL user-defined data
    types.
SQLJ Example

```sql
#sql {...} ;
SQL can span multiple lines
Java host expressions in SQL statement
throws java.sql.SQLException
String bug = "spider";
#sql {
    INSERT INTO bugs (name, numLegs)
    VALUES (:bug, :(getNumLegs(bug)))
};
```

JDBC Example

```java
PreparedStatement pstmt =
    conn.createStatement
("INSERT INTO bugs (name, numLegs)
VALUES (?, ?)");
pstmt.setString(1,bug);
pstmt.setInt(2,getNumLegs(bug));
pstmt.executeUpdate();
pstmt.close();
```

Java host expressions in SQL statement

### SQLJ vs JDBC comparison

<table>
<thead>
<tr>
<th>Feature</th>
<th>SQLJ</th>
<th>JDBC</th>
</tr>
</thead>
<tbody>
<tr>
<td>SQL statements</td>
<td>static</td>
<td>dynamic</td>
</tr>
<tr>
<td>Strong typing</td>
<td>yes</td>
<td>no</td>
</tr>
<tr>
<td>Checking</td>
<td>static</td>
<td>runtime only</td>
</tr>
<tr>
<td>Syntax</td>
<td>concise</td>
<td>API</td>
</tr>
<tr>
<td>Standard</td>
<td>ANSI</td>
<td>Sun</td>
</tr>
<tr>
<td>Portable</td>
<td>yes*</td>
<td>yes*</td>
</tr>
<tr>
<td>Object support</td>
<td>yes*</td>
<td>yes*</td>
</tr>
</tbody>
</table>

### Use SQLJ to write your program when
- you want to be able to check your program for errors at translation-time rather than at run-time.
- you want to write an application that you can deploy to another database. Using SQLJ, you can customize the static SQL for that database at deployment-time.
- you are working with a database that contains compiled SQL. You will want to use SQLJ because you cannot compile SQL statements in a JDBC program.

### Use JDBC to write your program when
- your program uses dynamic SQL. For example, you have a program that builds queries on-the-fly or has an interactive component.
- you do not want to have a SQLJ layer during deployment or development. For example, you might want to download only the JDBC Thin driver and not the SQLJ runtime libraries to minimize download time over a slow link.

### SQLJ static and non-static SQL
- The standard covers only static SQL operations
- those that are predefined and do not change in real-time as a user runs the application
- of course the data values that are transmitted can change dynamically!
- Oracle SQLJ offers extensions to support dynamic SQL operations
- those that are not predefined, where the operations themselves can change in real-time.
- It is possible to use dynamic SQL operations through JDBC code or PL/SQL code within a SQLJ application.
- Typical applications contain much more static SQL than dynamic SQL.
Java and SQLJ versus PL/SQL

- Java and PL/SQL are complementary.
- Suited for different kinds of applications.
- PL/SQL is better for SQL-intensive applications.
- Optimized for SQL, and so SQL operations are faster in PL/SQL than in Java.
- Uses SQL datatypes directly, while Java applications must convert between SQL datatypes and Java types.
- Java is better for logic-intensive applications.
- Superior programming model.
- Java's more general type system is better suited than PL/SQL for component-oriented applications.

Interoperability: SQLJ and PL/SQL

- PL/SQL programs
  - transparently call Java stored procedures, enabling you to build component-based Enterprise JavaBeans and CORBA applications.
  - have transparent access to a wide variety of existing Java class libraries through trivial PL/SQL call specifications.
- Java programs
  - call PL/SQL stored procedures and anonymous blocks through JDBC or SQLJ.
  - SQLJ provides syntax for calling stored procedures and functions from within a SQLJ statement, and also supports embedded PL/SQL anonymous blocks within a SQLJ statement.

Further Information

- [http://www.whatis.com/odbc.htm](http://www.whatis.com/odbc.htm)
- [http://www.whatis.com/jdbc.htm](http://www.whatis.com/jdbc.htm)
- [http://www.whatis.com/sqlj.htm](http://www.whatis.com/sqlj.htm)
- Local online JDBC Oracle manual pages

Additional material

ReadEntry method for completeness

```java
// Method: readEntry
// Purpose: to read a string from the user and return it
// Input:  The prompt string
// Output: User entry
static String readEntry (String prompt) {
    try {
        StringBuffer buffer = new StringBuffer ();
        System.out.print (prompt);
        System.out.flush ();
        int c = System.in.read ();
        while (c != '\n' && c != -1) {
            buffer.append ((char)c);
            c = System.in.read ();
        }
        return buffer.toString ().trim ();
    } catch (IOException e) {
        return "";
    }
}
```