Introduction to Java Database Connectivity (JDBC)

- Objective
  - to give some background on JDBC to help with the lab exercises
Overview

1. What is JDBC?
2. The JDBC-ODBC Bridge
3. Four Kinds of JDBC Drivers
4. JDBC Pseudocode
5. simpJDBC.java

Continued
6. Meta Data
7. **Books.mdb** as an ODBC Data Source
8. Table in **Books.mdb**
9. More Information
1. What is JDBC?

- JDBC is an interface which allows Java code to execute SQL statements inside relational databases
  - the databases must follow the ANSI SQL-2 standard
JDBC in Use

Java program

connectivity
data processing
utilities

JDBC

driver
for Oracle

driver
for Sybase

jdbc-odbc
bridge

odbc
driver
2. The JDBC-ODBC Bridge

- ODBC (Open Database Connectivity) is a Microsoft standard from the mid 1990’s.
- It is an API that allows C/C++ programs to execute SQL inside databases
- ODBC is supported by many products.
The JDBC-ODBC bridge allows Java code to use the C/C++ interface of ODBC. It means that JDBC can access many different database products.

The layers of translation (Java --> C --> SQL) can slow down execution.
The JDBC-ODBC bridge comes free with the JDK:
- called `sun.jdbc.odbc.JdbcOdbcDriver`

The ODBC driver for Microsoft Access comes with MS Office
- so it is easy to connect Java and Access
3. **Four Kinds of JDBC Driver**

- **1. JDBC-ODBC Bridge**
  - translate Java to the ODBC API

- **2. Native API**
  - translate Java to the database’s own API
3. Native Protocol
   - use Java to access the database more directly using its low level protocols

4. Net Protocol
   - use Java to access the database via networking middleware (usually TCP/IP)
   - required for networked applications
A searchable list of drivers (freeware, shareware, and commercial) can be found at:

http://www.javasoft.com/products/jdbc/drivers
4. JDBC PseudoCode

- All JDBC programs do the following:
  - 1) load the JDBC driver
  - 2) Specify the name and location of the database being used
  - 3) Connect to the database with a Connection object

Continued
– 4) Execute a SQL query using a Statement object

– 5) Get the results in a ResultSet object

– 6) Finish by closing the ResultSet, Statement and Connection objects
4.1. **Pseudocode as a Diagram**

- **DriveManager** creates **Connection**
- **Connection** creates **Statement**
- **Statement** creates **ResultSet**
- **DriveManager** makes a link to the **Driver**
- **Driver** interacts with the **database**
- **Database** provides **data** to **ResultSet**
4.2. **DriveManager**

- It is responsible for establishing the connection to the database through the driver.

- e.g.

```java
Class.forName(
    "sun.jdbc.odbc.JdbcOdbcDriver");
Connection conn =
    DriveManager.getConnection(url);
```
4.3. Name the Database

- The name and location of the database is given as a URL
  - the details of the URL vary depending on the type of database that is being used
ODBC Database URL

jdbc:odbc://host.domain.com:2048/data/file

- The comms protocol
- The machine holding the database.
- The port used for the connection.
- The path to the database on the machine

E.g. jdbc:odbc:Books
4.4. Statement Object

- The **Statement** object provides a ‘workspace’ where SQL queries can be created, executed, and results collected.

- e.g.

```java
Statement st = conn.createStatement();
ResultSet rs = st.executeQuery("select * from Authors");
st.close();
```
4.5. ResultSet Object

- Stores the results of a SQL query.

- A ResultSet object is similar to a ‘table’ of answers, which can be examined by moving a ‘pointer’ (cursor).

Continued
- **Cursor operations:**
  - `first()`, `last()`, `next()`, `previous()` etc.

- **Typical code:**
  ```java
  while( rs.next() ) {
    // process the row;
  }
  ```
// simpJDBC.java
// Displays the firstnames and lastnames
// of the Authors table in the Books db.

import java.sql.*;

public class simpJDBC {

    public static void main(String[] args) {
        // The URL for the Books database.
        // 'Protected' by a login and password.
        String url = "jdbc:odbc:Books";
        String username = "anonymous";
        String password = "guest";
        :
    }
}
try {
    // load the JDBC-ODBC Bridge driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // connect to db using DriverManager
    Connection conn =
        DriverManager.getConnection( url,
                        username, password );

    // Create a statement object
    Statement statement = conn.createStatement();

    // Execute the SQL query
    ResultSet rs = statement.executeQuery(
        "SELECT lastName, firstName FROM Authors" );
}
// Print the result set
while( rs.next() )
    System.out.println(
        rs.getString("lastName") + ", " +
        rs.getString("firstName")
    );

// Close down
statement.close();
conn.close();
catch ( ClassNotFoundException cnfex ) {
    System.err.println("Failed to load JDBC/ODBC driver.");
    cnfex.printStackTrace();
    System.exit( 1 );  // terminate program
}

catch ( SQLException sqlex ) {
    System.err.println( sqlex );
    sqlex.printStackTrace();
}

} // end of main()

} // end of simpJDBC class
Output

```
C>javac simpJDBC.java

C>java simpJDBC
Deitel, Harvey
Deitel, Paul
Nieto, Tem

C>
```
5.1. Username & Password

- The database’s link to the outside (e.g. its ODBC interface) must be configured to have a login and password
  - details for ODBC are given later
5.2. Accessing a ResultSet

- The ResultSet class contains many methods for accessing the value of a column of the current row
  - can use the column name or position
  - e.g. get the value in the lastName column:
    \[ \text{rs.getString("lastName")} \]
The ‘tricky’ aspect is that the values are SQL data, and so must be converted to Java types/objects.

There are many methods for accessing/converting the data, e.g.
- `getString()`, `getDate()`, `getInt()`, `getFloat()`, `getObject()`
6. **Meta Data**

- Meta data is the information *about* the database:
  - e.g. the number of columns, the types of the columns
  - meta data is the *schema* information

<table>
<thead>
<tr>
<th>ID</th>
<th>Name</th>
<th>Course</th>
<th>Mark</th>
</tr>
</thead>
<tbody>
<tr>
<td>007</td>
<td>James Bond</td>
<td>Shooting</td>
<td>99</td>
</tr>
<tr>
<td>008</td>
<td>Aj. Andrew</td>
<td>Kung Fu</td>
<td>1</td>
</tr>
</tbody>
</table>
6.1. Accessing Meta Data

- The `getMetaData()` method can be used on a `ResultSet` object to create its metadata object.

- e.g.

```java
ResultSetMetaData md = rs.getMetaData();
```
6.2. Using Meta Data

```java
int numCols = md.getColumnCount();

for (int i = 0; i <= numCols; i++) {
    if (md.getColumnType(i) == Types.CHAR)
        System.out.println(
            md.getColumnName(i)
        );
}
```
6.3. More Meta Data Methods

- `getTableName()`
- `getPrecision()`
  - number of decimal digits in the column
- `isSigned()`
  - returns true if column has signed numbers
- `isCurrency()`
- `etc.`
7. Books.mdb as an ODBC Data Source

1. Click on “32 bit ODBC” in the Control Panel. This displays the ODBC Data Sources Administrator.

An ODBC User data source stores information about how to connect to the indicated data provider. A User data source is only visible to you and can only be used on the current machine.

![Create New Data Source dialog](image)
3. Type in a source name, description, and press “Select” to browse to set the path to the Books.mdb file.

Now click on “Advanced”.
4. Type in a username and password (guest). Click “Ok”
8. Table in Books.mdb

<table>
<thead>
<tr>
<th>Publishers</th>
<th>Authors</th>
</tr>
</thead>
<tbody>
<tr>
<td>PublisherID</td>
<td>AuthorID</td>
</tr>
<tr>
<td>PublisherName</td>
<td>FirstName</td>
</tr>
<tr>
<td>ISBN</td>
<td>ISBN</td>
</tr>
<tr>
<td>Title</td>
<td>Title</td>
</tr>
<tr>
<td>EditionNumber</td>
<td>EditionNumber</td>
</tr>
<tr>
<td>YearPublished</td>
<td>YearPublished</td>
</tr>
<tr>
<td>Description</td>
<td>Description</td>
</tr>
<tr>
<td>PublisherID</td>
<td>AuthorISBN</td>
</tr>
</tbody>
</table>

- Publishers table:
  - PublisherID
  - PublisherName

- Titles table:
  - ISBN
  - Title
  - EditionNumber
  - YearPublished
  - Description
  - PublisherID

- AuthorISBN table:
  - ISBN
  - AuthorID

- Authors table:
  - AuthorID
  - FirstName
  - LastName
  - YearBorn
9. More Information

- *Java: How to Program*
  Deitel and Deitel, 3rd Ed.
  Chapter 18.

- *Beginning Java 2*
  Ivor Horton
  Chapters 18 and 19

*Continued*
Current Information:

http://www.javasoft.com/products/jdbc

The JDK Documentation and tutorial

- the JDBC ‘trail’ is very good
Java Database Connectivity (JDBC) 2
Overview

1. SQL Statements
2. Executing DDL and DML
3. PreparedStatement Objects
4. Database Metadata
5. More Information
1. SQL Statements

- They can be defined into two types:
  - those using the *Data Definition Language* (DDL)
    - create, delete tables
  - those using the *Data Manipulation Language* (DML)
    - select
    - others, including insert, update, delete
1.1. create Example

create table studInfo (  
    studID int not null primary key,  
    lastname char(25) not null,  
    firstname char(40) not null,  
    address char(25),  
    email char(25)  
);
1.2. **insert Example**

```sql
insert into studInfo
(studID, lastname, firstname, email)
values (7, 'Bond', 'James',
       'bond007@spy.uk')
```

- **Unspecified rows will be filled with SQL null values.**
- **The** `studID`, `lastname`, **and** `firstname` **must be supplied.**
1.3. update and delete Examples

- update studInfo set lastname = 'Bland'
  where studID = 7

- delete from studInfo where studID = 7
2. Executing DDL and DML

- From last time: select statements are executed from JDBC with `executeQuery()`.
- The method returns a table of results (a `resultSet` object).
- e.g.

```java
ResultSet rs = statement.executeQuery(
    "SELECT lastName, firstName
    FROM Authors" );
```

continued
But most SQL statements do not return a table

- DDL statements, e.g. create, drop
- most DML statements, e.g. insert, update

If you try to execute these kinds of statements with `executeQuery()`, an `SQLException` will occur.
2.1. `executeUpdate()`

- Used to execute SQL statements that change the database, table, or row
  - e.g. DDL: create, drop
  - e.g. DML: insert, update, delete
2.2. BuildTables.java

- This program uses create and insert to build a new table, called urlInfo, inside Books.mdb.
import java.sql.*;

public class simpJDBC {

    public static void main(String[] args) {
        // The URL for the Books database.
        // 'Protected' by a login and password.
        String url = "jdbc:odbc:Books";
        String username = "anonymous";
        String password = "guest";
        :
    }
}
// SQL table creation and insertion
String[] SQLStats = {
    "create table urlInfo (id int, Name char(48),
        url char(80))",
    "insert into urlInfo values(1, 'Andrew D',
        'http://fivedots.coe.psu.ac.th/~ad')",
    "insert into urlInfo values(2, 'JavaSoft Home Page',
        'http://www.javasoft.com')",
    "insert into urlInfo values(3, 'PSU',
        'http://www.psu.ac.th')"};

try {
    // load the JDBC-ODBC Bridge driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");
    // connect to db using DriverManager
    Connection conn =
        DriverManager.getConnection( url, username, password );

}
// Create a statement object
Statement statement =
    conn.createStatement();

// Create urlInfo table
for (int i = 0; i < SQLStats.length; i++) {
    statement.executeUpdate(SQLStats[i]);
    System.out.println("Processed: " + SQLStats[i]);
}

// Close down
statement.close();
conn.close();
catch ( ClassNotFoundException cnfex ) {
    System.err.println("Failed to load JDBC/ODBC driver.");
    cnfex.printStackTrace();
    System.exit(1);  // terminate program
}

catch ( SQLException sqlex ) {
    System.err.println(sqlex);
    sqlex.printStackTrace();
}

} // end of main()

} // end of BuildTables class
Notes

- The 4 SQL commands are executed inside the single statement object.

- `executeUpdate()` returns an integer
  - the number of rows affected
  - not used in `BuildTables.java`
3. `PreparedStatement` Objects

- `PreparedStatement` objects are created in a similar way to `Statement` objects

- e.g.

  ```java
  PreparedStatement ps = conn.prepareStatement(" SQL query...");
  ```

*continued*
SQL commands inside a PreparedStatement object are *compiled*

- they will execute faster than SQL in a Statement object
- but the compilation takes time

PreparedStatements are often used for SQL commands that will be executed repeatedly.
3.1. **PlaceHolders**

- SQL commands in a `PreparedStatement` can contain placeholders—these can be changed at run time to vary the meaning of the command.

- Placeholders are represented by ‘?’ in the SQL command.
Example:

```java
String newName = "update authors
    set lastName = ?
    where AuthorID = ? ";
PreparedStatement ps =
    conn.prepareStatement(newName);
```

The ‘?’s must be replaced by real values before the statement is executed.
The ‘?’s can be replaced using a wide range of “set” methods for different types.

e.g.

- `setString()`, `setInt()`, `setFloat()`, `setDate()`, ...
3.2. **PlaceHolders.java**

- **Update the** `URLInfo` **table**
  - replace “Andrew D” by “Andrew Davison”

- **Use a** `PreparedStatement` **object with two placeholders**
  - this simple example could be coded using `Statement` **objects**
import java.sql.*;

public class PlaceHolders {

    public static void main(String[] args) {
        // The URL for the Books database.
        // It is 'protected' by a login and password.
        String url = "jdbc:odbc:Books";
        String username = "anonymous";
        String password = "guest";
    }
}
try {
    // load the JDBC-ODBC Bridge driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // connect to db using DriverManager
    Connection conn =
        DriverManager.getConnection(url, username, password);

    :

// Create PreparedStatement
String changeAD =
    "update urlInfo set name = ?
    where id = ?";
PreparedStatement ps =
    conn.prepareStatement(changeAD);

// Fill in the '?'s
ps.setString(1, "Andrew Davison");
ps.setInt(2, 1); // his id is '1'

// make the change
int noRowsUpdated = ps.executeUpdate();
System.out.println("No. of Rows updated: "
    + noRowsUpdated);

;
// Close down
conn.close();
}
catch ( ClassNotFoundException cnfex ) {
    System.err.println("Failed to load JDBC/ODBC driver.");
    cnfex.printStackTrace();
    System.exit(1); // terminate program
}
catch ( SQLException sqlex ) {
    System.err.println(sqlex);
    sqlex.printStackTrace();
}
} // end of main()

} // end of PlaceHolders class
Execution Results

- c> java PlaceHolderson
  No. of Rows updated: 1

```
C>
```
4. **Database Metadata**

- Metadata for the database is information about all the tables, and all the columns in each table (i.e. their *schema*).

- **Code Fragment:**
  
  ```java
  Connection conn = DriverManager.getConnection( url, username, password );
  DatabaseMetaData metadata = conn.getMetaData();
  ```

*continued*
Once the metadata object has been created, details about the tables and the columns can be extracted with:

- `getTables()`
- `getColumns()`

Both methods return `resultSets` (tables) of information.
4.1. `getTables()`

- ResultSet `trs = metadata.getTables( String catalog, String schemePattern, String tableNamePattern, String[] types);`

- The arguments are used to limit the information which is returned
  - if an argument is not to be used, then set it to null
**Argument Details**

- **String catalog**
  - this specifies the catalog to search for tables

- **String schemePattern**
  - select tables which use the pattern in a schema name
  - a schema pattern is a string. Two special chars:
    - % means ‘any substring’
    - _ means ‘any character’

*continued*
- e.g. “%data” will match schema names:
  “data”, “Mydata”, “Yourdata”
- e.g. “data_” will match schema names:
  “datas”, “data1”

- String tablePattern
  - used to select tables based on their names
  - may use “%” and “_”
String[] types

- select tables based on their types
- possible types:
  - "TABLE",
  - "SYSTEM TABLE",
  - "VIEW"
Get the metadata for all the normal tables:

```java
String[] tableTypes = { "TABLE" };
ResultSet trs = metadata.getTables(null, null, null, tableTypes);
```
**getTables() ResultSet Format**

- The resultSet (e.g. `trs`) has five columns:
  - `TABLE_CAT`:
    - a String specifying the catalog (or null)
  - `TABLE_SCHEM`:
    - a String specifying the table schema (or null)
  - `TABLE_NAME` (the table name)
  - `TABLE_TYPE` (its type)
  - `REMARKS`:
    - a String describing the table
4.2. viewTables.java

// Pull metadata from Books.mdb
// Print the name of every table in Books.mdb
// Uses getTables()

import java.sql.*;

public class viewTables {

    public static void main(String[] args) {
        // The URL for the Books database.
        // It is 'protected' by a login and password.
        String url = "jdbc:odbc:Books";
        String username = "anonymous";
        String password = "guest";
    }
}
try {
    // load the JDBC-ODBC Bridge driver
    Class.forName("sun.jdbc.odbc.JdbcOdbcDriver");

    // connect to db using DriverManager
    Connection conn =
        DriverManager.getConnection( url,
            username, password );

    // Get the database metadata
    DatabaseMetaData metadata =
        conn.getMetaData();
    :
// Extract the table-based metadata
String [] tableTypes = { "TABLE" };
ResultSet tables =
    metadata.getTables(null, null, null, tableTypes);

// Print the table names
String tableName;
while( tables.next() ) {
    tableName =
        tables.getString("TABLE_NAME");
    System.out.println( tableName );
}
// Close down
cnn.close();
}
catch ( ClassNotFoundException cnfex ) {
    System.err.println(
        "Failed to load JDBC/ODBC driver." );
    cnfex.printStackTrace();
    System.exit(1); // terminate program
}
catch ( SQLException sqlex ) {
    System.err.println( sqlex );
    sqlex.printStackTrace();
}
} // end of main()

} // end of viewTables class
Usage

C>javac viewTables.java

C>java viewTables
AuthorISBN
Authors
Publishers
Titles
urlInfo
4.3. **getColumns()**

- `ResultSet crs = metadata.getColumns(String catalog, String schemePattern, String tableNamePattern, String columnPattern);`

- **The first 3 arguments are as in getTables()**

- `columnPattern` selects a column based on the pattern matching its name.
Get the metadata for all the columns in the "URLInfo" table:

```java
ResultSet crs = metadata.getColumns(null, null, null, "URLInfo", null);
```
The resultSet (e.g. `rs`) has 18 columns!

Some of the useful ones:

- `COLUMN_NAME`
- `DATA_TYPE`:
  - type of the data in the column
- `COLUMN_SIZE`
  - num. of chars or the numerical precision in the column