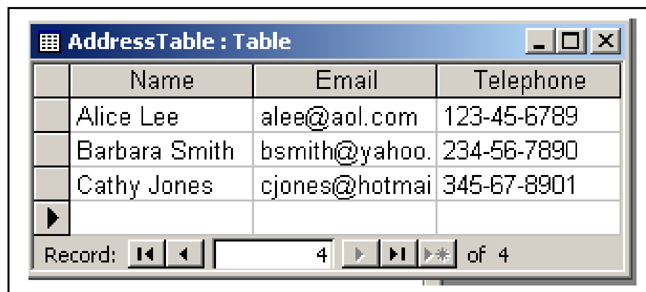
## Connecting to a Database

Web applications usually connect to a database rather than read and write to text files. A relational database consists of one or more tables each of which is made up of rows and columns. Each row defines the information for a single object. The columns then are the fields that describe the object. An example is the address book used before. Each entry will be contained in a separate row. The field names can be Name, Email, and Telephone.

The following is an example of one such table, called AddressTable. It is contained in an *Access* database called *addresses.mdb*. Access is part of Microsoft's Office suite of programs. It is convenient to use, since it can reside on the same local computer as the server.



AddressTable : Table

| Name | Email | Telephone |
| --- | --- | --- |
| Alice Lee | alee@aol.com | 123-45-6789 |
| Barbara Smith | bsmith@yahoo. | 234-56-7890 |
| Cathy Jones | cjones@hotmai | 345-67-8901 |
| | | |

Record: 4 of 4

To connect to a database using a Java program, you must first register the database with the operating system.[1] The connection is done with a jdbc-odbc bridge. Jdbc stands for Java database connectivity API (application programming interface), while the 'O' in Odbc stands for Open. Odbc is a protocol from Microsoft that is based on the X/Open SQL specification.

In a Java program, we create a *Connection* object. The lines of code required are:

```
Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
Connection con = DriverManager.getConnection ("jdbc:odbc:addresses");
```

where *addresses* is the name used for the database in the registration information. Both the Connection and DriverManager objects are contained in the Java package, *java.sql*. This package is a standard part of JDK and so doesn't have to be separately downloaded. But it must be imported into any program that connects to a database.

SQL stands for Structured Query Language and is usually pronounced *sequel.* SQL is the standard way to interact with relational databases and is not part of Java. SQL is not case sensitive, so you can mix upper and lower cases, but commands often use upper case. Some examples of commands are Select, Insert, Delete, and Update. You can also modify commands by using connectors such as Where or Set.

**Select** is used to obtain information from the database. For example, "SELECT * FROM AddressTable" will get all (*) the data from the database. If you don't want all the data, you can add a clause that will further define the rows needed. This is done with the modifier, *Where.* For example, if you just want the names that begin with the letter A, you can use the query

"SELECT * FROM AddressTable WHERE Name LIKE 'A%'"

[1] To connect to the database using a Java program, you must first register the database with the operating system. In Windows 98 this is done by clicking on **Settings/Control Panel/Data Sources (ODBC)**. In Windows 2000 or XP, you will find this same file in **Settings/Control Panel/Administrative Tools.** Select **Add/Microsoft Access Driver (*.mdb),** and from there **Browse** to find the location of your database.

The 'A%' combination is used to indicate a pattern that begins with the letter A. Note the single quotes.

When sending a query to a database from a program, you first create a Statement object for the query, and then you execute it. If the database is able to execute the query, it returns a *ResultSet* with the data. A query that returns all the data in the AddressTable uses the following code:

```
    Statement stmt = con.createStatement ();
    String query = "SELECT * FROM AddressTable";
    ResultSet rs = stmt.executeQuery (query);
```

The data is returned as a sequence of rows. The particular fields in the row can be obtained using *get* methods. Since all the items in this database are strings, the get method used here is *getString*. The readData method is replaced by the following getData method. It throws a SQLException back to the method that is used to create the list. Note that when we read from a file, the exception thrown was an IOException.

```
// getData gets data from the database and stores it in the Data object.
protected void getData (ResultSet rs) throws SQLException
{
    name = rs.getString ("Name");
    email = rs.getString ("Email");
    phone = rs.getString ("Telephone");
} // getData
```

The complete program follows. You might note where it is different from the file reading program that was done before.

```
/*  Addresses is a Java program that gets data from a database and stores it in an array. It then sends a
copy of the list to the client in a second web page.*/

package client_server;

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

/*  Addresses is the main class. It gets the output page and a connection, instantiates the List class and
calls its methods. */
public class Addresses extends WebRequestProcessor
{
    private List list;
    private PrintWriter out;

    public void process (Request request, Response response)
    {
        try
        {
            response.setContentType ("text/html");
            out = response.getWriter ();

            // Get a jdbc-odbc bridge and connect to addresses.mdb.
            Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");
            Connection con = DriverManager.getConnection ("jdbc:odbc:addresses");
```

```
			Page.createHeader (out, "Email Address List Example");
			list = new List ();
			list.getDataFromDatabase (con, out);
			list.displayData (out);
			Page.createFooter (out);
			con.close ();
		} catch (ClassNotFoundException e)
		  {System.out.println ("<h3>Class Not Found exception.</he>");}
		  catch (SQLException e){System.out.println ("<h3>SQL Exception<\h3>");}
	} // process
} // class Addresses

//	The Data class is used to store name, email and phone data.
class Data
{
	private String name, email, phone;

	// getData gets data from the database and stores it in the Data object.
	protected void getData (ResultSet rs) throws SQLException
	{
		name = rs.getString ("Name");
		email = rs.getString ("Email");
		phone = rs.getString ("Telephone");
	} // getData

	// displayData displays the data in the object in a table on the response page.
	protected void displayData (PrintWriter out)
	{
		out.println ("<tr><td>" + name + "</td>");
		out.println ("<td>" + email + "</td>");
		out.println ("<td>" + phone + "</td></tr>");
	} // displayData
} // class Data

/*	The List class instantiates an array, gets data from the database and puts it into the array.  It then
displays the contents of the array on a web page sent to the client.*/
class List
{
	private Data [] list;
	private int listSize;
	final int maxSize = 20;

	List ()
	{
		list = new Data [maxSize];
		listSize = 0;
	} // constructor

	/* getDataFromDatabase creates a select query to get all the data from the database.  It stores the data
	in an array of objects called list. */
```

```
	public void getDataFromDatabase (Connection con, PrintWriter out)
	{
		try
		{
			Statement stmt = con.createStatement ();
			String query = "SELECT * FROM AddressTable";
			ResultSet rs = stmt.executeQuery (query);
			while ((rs.next ())&& (listSize < maxSize))
			{
				Data data = new Data ();
				data.getData (rs);
				list [listSize] = data;
				listSize ++;
			}
			stmt.close ();
		} catch (SQLException es) {out.println ("SQL Exception");}
	} // getDataFromDatabase

	// displayData sends a copy of the list to the client formatted as a html table.
	public void displayData (PrintWriter out)
	{
		out.println ("<h3>Address Book</h3>");
		out.println ("<table border='1' bordercolor='#000000' cellspacing='10'>");
		out.println ("<tr><td>Name</td><td>Email Address</td><td>Telephone Number</td></tr>");

		for (int count = 0; count < listSize; count++)
			list [count].displayData (out);
		out.println ("</table>");
	} // displayData
} // class List
```

**More SQL**

There are many kinds of queries that can be constructed using SQL. The basic ones are Select, Insert, Update, and Delete. We have used Select above. A brief description of the other three follows.

**Insert** is used to add a row to the database. An example that adds a row to the AddessTable is

```
"INSERT INTO AddressTable VALUES ('Diana Chen', 'dchen@earthlink.com', '456-78-9012')"
```

Note that the individual values that are to be inserted are contained in single quotes. Single quotes are used for strings, but not for numeric data types. They must always be part of an insert or update query that uses strings. So when we get data from the client and include it in a query, the complete query string must contain single quotes around each string entry. Sometimes a single quote is the only thing inside a set of double quotes in order to put a single quote in the place where it is needed. The same query string as before would look like the following if the data came from the Data object rather than consisting only of constants. Note the single quotes (in bold face) used to surround each variable.

```
"INSERT INTO AddressTable VALUES ('" + name + "', '" + email + "', '" + phone + "')"
```

The html form used to get the data from the client follows:

```
<form method = "get" action="http://127.0.0.1:8080/client_server.Addresses">
	<br /><input name = "name"  type = "text"  value = ""  size = "30" /> Name
```

```
    <br /><input name = "email"  type = "text"  value = ""  size = "30" /> E-Mail
    <br /><input name = "phone"  type = "text" value = ""  size = "30" /> Telephone
    <p><input type = "submit"  value = "Send" /></p>
</form>
```

Two methods can also be added to the Data class, one to get the data from the html form and the other to insert the new row into the database.

```
// getRequestData gets the data from the client's html form.
private void getRequestData (Request request)
{
    name = request.getParameter ("name");
    email = request.getParameter ("email");
    phone = request.getParameter ("phone");
} // getRequestData

// This method uses the connection to the database to insert the new data into the database.
public void insertNewData (Request request, Connection con, PrintWriter out)
{
    try
    {
        getRequestData (request);
        Statement stmt = con.createStatement ();
        String query =
        "INSERT INTO AddressTable VALUES ('" + name + "', '" + email + "', '" + phone + "')";
        stmt.executeUpdate (query);
        out.println ("Data inserted.");
        stmt.close ();
    } catch (SQLException es) {out.println ("SQL Insert Exception");}
} // insertNewData
```

The **Update** query is somewhat similar to the Insert query. Update is used to change an entry in the database, such as an e-mail address. The query must include a key to indicate which row is to be changed. For example, to change the email address in some row, we can use the following query:

```
    "UPDATE AddressTable SET Email = '" + email + "' WHERE Name = '" + name + "'"
```

Note that the following method is not much different from the one above.

```
// updateEmail changes the email address in the database in the row given by name.
public void updateEmail (Request request, Connection con, PrintWriter out)
{
    try
    {
        getRequestData (request);
        Statement stmt = con.createStatement ();
        String query =
        "UPDATE AddressTable SET Email = '" + email + "' WHERE Name = '" + name + "'";
        stmt.executeUpdate (query);
        out.println ("Database updated.");
        stmt.close ();
    } catch (SQLException es) {out.println ("SQL Update Exception");}
} // insertNewData
```

**Delete** is used to delete a row in the database. Again a key field is used to determine the appropriate row. The query here is "DELETE FROM AddressTable WHERE Name = '" + name + "'". This deletes the row with the name entered into the html form. The new method follows:

```
public void deleteRow (Request request, Connection con, PrintWriter out)
{
	try
	{
		getRequestData (request);
		Statement stmt = con.createStatement ();
		String query = "DELETE FROM AddressTable WHERE Name = '" + name + "'";
		stmt.executeUpdate (query);
		out.println ("Row deleted.");
		stmt.close ();
	} catch (SQLException es) {out.println ("SQL Delete Exception");}
} // deleteRow
```

**Still More SQL**

Now suppose that the address book is to be used by a club that charges dues. We can add a column to the database using the **Alter** command. We can write

"ALTER TABLE AddressTable Add Dues DOUBLE"

The general form of the Alter command is

"ALTER TABLE table_name ADD column_name datatype"

The data type specifies what type of data the column can hold. The table below contains the most common data types in SQL:

| **Data Type** | **Description** |
|---|---|
| integer(size)<br>int(size)<br>smallint(size) | Hold integers only. The maximum number of digits is specified in parenthesis. |
| decimal(size,d)<br>numeric(size,d) | Hold numbers with fractions. The maximum number of digits is specified in "size". The maximum number of digits to the right of the decimal is specified in "d". |
| float(n)<br>real<br>double | Floating point number with n binary digits of precisions.<br>32-bit floating point number.<br>64-bit floating point number. |
| char(size) | Holds a fixed length string (can contain letters, numbers, and special characters). The fixed size is specified in parenthesis. |
| varchar(size) | Holds a variable length string (can contain letters, numbers, and special characters). The maximum size is specified in parenthesis. |
| date(yyyymmdd) | Holds a date |

(The table above was taken from the web site, http://www.w3schools.com. This site has a number of excellent tutorials on SQL and a number of other things. It is a very good way to learn how to create web applications.)

When a member pays her dues, the amount can be entered using the Update command. To do this, change the update method above to

"UPDATE AddressTable SET Dues = " + dues + " WHERE Name = '" + name + "'"

It is important to note, that the numeric field, dues, is not surrounded by single quotes. Only strings are. So anytime you have a numeric field, leave out the single quotes.

We can also create a new table to be added to the database. The command here is

"CREATE TABLE table_name (column_name1 datatype1, column_name2 datatype2, …)"

If the club wants to create a table with its officers, this can be done with

"CREATE TABLE Officers (Name varchar (30), Office varchar (30))"

This will create a new table in the addresses database with two columns, Name and Office. Both are strings of variable size not to exceed 30 characters. Then to add data, use the Insert command, for example

"INSERT INTO Officers" + " VALUES ('" + name + "', '" + office + "')"

where name and office are request parameters received from the html form.

With two tables, you can perform more complicated queries that get data from both tables. For example, you can get the e-mail address and telephone number of the club president. Since we now have two tables, it is necessary to specify which table each column name refers to. As in Java, this is done using a period to determine the *path*.

```
"SELECT * Officers.Name, AddressTable.Email, AddressTable.Telephone "
+ "FROM AddressTable, Officers"
+ " WHERE  Officers.Office = 'President' "
```