APPENDIX L

Example Web Scripts

Objectives

In this appendix you will learn:

- The use of client-side JavaScript.
- The use of PHP and PostgresQL.
- The use of CGI and Perl.
- The use of JDBC to retrieve data from a database.
- The use of SQLJ to retrieve data from a database.
- The use of (server-side) JavaServer Pages (JSP).
- The use of (server-side) Active Server Pages and ActiveX Data Objects.
- The use of Oracle's PL/SQL Server Pages (PSPs).

In Chapter 30 we examined in some detail the World Wide Web (Web) and some of the current approaches to integrating databases into the Web environment. In this appendix we provide a number of examples that illustrate some of the concepts covered in that chapter. We assume the reader is familiar with the concepts introduced in Chapter 30. The examples in this appendix are drawn from the *DreamHome* case study documented in Section 11.4 and Appendix A.

L.1 JavaScript

In Section 30.3.1 we introduced the scripting language JavaScript. The following example illustrates the use of client-side JavaScript. We illustrate the use of server-side JavaScript using Microsoft's Active Server Pages in Example L.7.

EXAMPLE L.1 Use of JavaScript to display book details

Create a JavaScript program to emulate the HTML page shown in Figure 30.2(b).

Figure L.1 illustrates the use of client-side JavaScript to create a Web page corresponding to Figure 30.2(b). In this example, an array is used to hold a list of pages that the user can access and another array to hold the corresponding URLs for each page. An HTML form is created with an onChange event specified to call a function (goPage) when the user selects a page.

```
<HTML>
<HEAD>
<TITLE>Database Systems: A Practical Approach to Design, Implementation and Management</TITLE>
<SCRIPT LANGUAGE = "JavaScript">
<!-- Hide script within a comment field to ensure that it is not displayed by old browsers
// that do not recognize JavaScript
// Function to create an array and populate its properties
function makeArray() {
    var args = makeArray.arguments;
    for (var i = 0; i < args.length; i++) {
    this[i] = args[i];
    }
    this.length = args.length;
}
// Array to hold the descriptions and names of the pages
var pages = new makeArray("Select a Page",
                "Table of Contents",
                "Chapter 1 Introduction",
                "Chapter 2 Database Environment",
                "Chapter 3 Database Architectures",
                "Chapter 4 The Relational Model",
// Array to hold the URLs of the pages
var urls = new makeArray("",
                 "http://cis.paisley.ac.uk/conn-ci0/book/toc.html",
                 "http://cis.paisley.ac.uk/conn-ci0/book/chapter1.html",
                 "http://cis.paisley.ac.uk/conn-ci0/book/chapter2.html",
                 "http://cis.paisley.ac.uk/conn-ci0/book/chapter3.html",
                 "http://cis.paisley.ac.uk/conn-ci0/book/chapter4.html",
                 "http://cis·paisley·ac·uk/conn-ci0/book/chapter6.html",
// Function to determine which page has been selected and to go to it.
function goPage(form) {
    i = form.menu.selectedIndex;
    if (i != 0) {
        window.location.href = urls[i];
    }
}
```

(*continued*)

Figure L.1 Example of client-side JavaScript to display Web page shown in Figure 30.2(b).

```
// end hiding contents from old browsers -->
</SCRIPT>
</HEAD>
<BODY bgcolor = # FFFFCC>
<H2>Database Systems: A Practical Approach to Design, Implementation and Management</H2>
<P>Thank you for visiting the Home Page of our database text book. From this page you can view online a
selection of chapters from the book. Academics can also access the Instructor's Guide, but this requires the
specification of a user name and password, which must first be obtained from Addison Wesley Longman.
<BR><BR>
<SCRIPT LANGUAGE = "JavaScript">
<!--
// Display the menu, wait for selection and go there
document.write('<FORM><SELECT NAME = "menu" onChange = "goPage(this.form)">');
      for (var i = 0; i < pages.length; i++) {
            document.write('<OPTION>' + pages[i]);
      }
document.write('</SELECT></FORM>');
//-->
</SCRIPT>
</BODY>
</HTML>
```

Figure L.1 (*Continued*)

L.2 PHP and PostgreSQL

In Section 30.3.3 we introduced the scripting language PHP. The following example illustrates the use of server-side PHP.

EXAMPLE L.2 Use of PHP and PostgreSQL

Produce a PHP script to list all records in the Staff table.

The code for the script is shown in Figure L.2. The PHP part of the HTML file is delimited by <?php . . . ?>. The PHP function pg_pconnect is used to connect to the data source, pg_Exec is used to run an SQL query, pg_NumRows is used to determine how many rows are in the result set, and pg_result is used to access each field of the result set. Dynamically generated HTML is created using the echo command.

```
<HTML>
<HEAD>
<TITLE>DreamHome Staff Query</TITLE>
</HEAD>
<BODY>
<CENTER><H2><I>DreamHome</I> Staff Query</H2></CENTER>
```

(*continued*)

Figure L.2 Sample PHP script to connect to PostgreSQL.

```
<?php
// Connect to the PostgreSQL and run the select query
$conn = pg_pconnect("host=www.dreamhome.co.uk user=admin password=dbapass dbname=dreamhomeDB");
if (!$conn) {
  echo "Error connecting to database.\n";
  exit;
}
$sql = "SELECT staffNo, fName, lName, position FROM Staff";
$resultSetID = pg_Exec($conn, $sql);
// Now loop over the records returned and display them
$rows = pg_NumRows($resultSetID);
echo "<TABLE BORDER=1 ALIGN=CENTER>"
echo "<THEAD> <TR><TH>staffNo</TH><TH>fName</TH>
<TH>lName</TH><TH>position</TH></TR></THEAD>"
for ($j=0; $j < $rows; $j++) {
  echo "<TR>"
  echo "<TD>pg_result($resultSetID, $j, "staffNo")</TD>"
  echo "<TD>pg_result($resultSetID, $j, "fName")</TD>"
  echo "<TD>pg_result($resultSetID, $j, "lName")</TD>"
  echo "<TD>pg_result($resultSetID, $j, "position")</TD>"
  echo "</TR>"
}
echo "</TABLE>"
?>
</BODY>
</HTML>
```

Figure L.2 *(Continued)*

L.3 CGI and Perl

In Section 30.3.3 we introduced the high-level interpreted programming language Perl and in Section 30.4 we examined the Common Gateway Interface (CGI) protocol. The following example illustrates the use of CGI and Perl.

EXAMPLE L.3 Use of CGI and Perl

Produce a Perl script to take an arbitrary SELECT statement and write out the results.

The code for the script is shown in Figure L.3. The use statement allows a Perl script to include the contents of predefined libraries. The CGI library is included to provide functionality that makes it easier to write the HTML to be sent to the browser. For example, the CGI library contains keywords that represent HTML tags, which are used in this script to create an HTTP header and footer (print header and print end_html).

```
# query.pl
# Program to query the database and send results to the client.
use Win32::ODBC;
use CGI qw/:standard/;
#Connect to the database and run the query
my $querystring = param(QUERY);
$DSN = "DreamHomeDB";
print header;
if (!($Data = new Win32::ODBC($DSN))) {
  print "Error connecting to $DSN\n";
  print "Error: " . Win32::ODBC::Error() . "\n";
  exit;
}
if ($Data->Sql($querystring)) {
  print "SQL query failed.\n";
  print "Error: " . $Data->Error() . "\n";
  $Data->Close();
  exit;
}
# Now print out each row of the result table
$counter = 0;
print "<TABLE BORDER = 1 ALIGN = CENTER>";
while($Data->FetchRow()) {
  %Data = $Data->DataHash();
  @key_entries = keys(%Data);
  print "<TR>";
  foreach $key(keys(%Data)) {
    print "<TD>$Data{$key}</TD>";
  }
  print "</TR>";
  $counter++;
}
print "</TABLE>";
print "<BR>Your search returned <B>$counter</B> rows.";
print end_html;
$Data->Close();
```

Figure L.3 Sample CGI/Perl script to run an arbitrary query.

L.4 Java and JDBC

In Section 30.7.1 we introduced JDBC for Java as one way to connect a Java application to a DBMS. The following example illustrates the use of JDBC.

EXAMPLE L.4 Use of JDBC

Produce a Java application to list all records in the Staff table.

This example demonstrates the use of JDBC within a standalone Java application using the JDBC–ODBC bridge and Microsoft's 32-bit ODBC driver. The code for the Java application is shown in Figure L.4.

```
/*
* JDBCExample.java - Illustrates the use of JDBC.
*/
import java.sql.*;
class JDBCExample {
  static Connection dbCon;            // database connection object
  public static void main(String args[]) throws Exception {
        Class.forName ("sun.jdbc.odbc.JdbcOdbcDriver");       // Load the JDBC-ODBC bridge driver
        open();                       // Open database and display all rows in a table
        select();
        close();                      // Close the database
}
/* Open a database connection */
static void open() throws SQLException {
      /* Set up ODBC data source name and username/password for connection */
      String url = "jdbc:odbc:DreamHomedbDSN";
      String user = "admin";
      String password = "dbapass";
      /* Call the Driver Manager to connect to the database */
      dbCon = DriverManager.getConnection(url, user, password);
}
/* Close the database connection */
static void close() throws SQLException {
      dbCon.close();
}
/* Issue a SQL query with a WHERE clause */
static void select() throws SQLException {
      Statement stmt;               // SQL statement object
      String query;                 // SQL select string
      ResultSet rs;                 // SQL query results
      boolean more;                 // "more rows found" switch

/* Set up the query, then create a statement and then execute the query */
      query = "SELECT staffNo, fName, lName, position FROM Staff WHERE staffNo = 'SG37' ";
      stmt = dbCon.createStatement();
      rs = stmt.executeQuery(query);
 /* Check to see if any rows were returned */
      more = rs.next();                                                    (continued)
```

Figure L.4 Sample Java application using JDBC.

```
        if (!more) {
                System.out.println("No rows found.");
                return;
        }
/* Loop through the rows retrieved from the query and write out the data */
        while (more) {
                System.out.println("Staff Number: "  + rs.getString ("staffNo"));
                System.out.println("Name: " + rs.getString("fName") + " " + rs.getString("lName"));
                System.out.println("Position: " + rs.getString("position"));
                System.out.println("");
                more = rs.next();
        }
        rs.close();
        stmt.close();
    }
}
```

Figure L.4 (*Continued*)

L.5 Java and SQLJ

In Section 30.7.2 we briefly introduced SQLJ as an alternative approach to accessing databases from a Java application. The following example illustrates the use of SQLJ.

EXAMPLE L.5 Using SQLJ to retrieve data

Produce an SQLJ program to print out the properties for a particular member of staff.

The template program is shown in Figure L.5. The function of the program is similar to that of Example I.3, which used (static) embedded SQL. In SQLJ, embedded statements begin with the string "#sql". The program declares a connection-context class, *DreamHome*DB, for an object representing the database where the SQL statements will execute. In this example, the SQL query can return multiple rows, and so a form of cursor is used to allow iteration through the individual records. The method shown here binds the attributes of the PropertyForRent table to an iterator type called propertyDetails. With this approach, we can access each attribute as a method of the iterator object, propertyIterator.

```
import java.sql.*
import oracle.sqlj.runtime*;
public class staffProperties {
  public static void main(String args[]) throws SQLException {
/* open database connection */
  Oracle.connect(staffProperties.class, "connect.DreamHomeDB");
/* define iterator for query */
  #sql iterator propertyDetails (String propNo, String st);
       propertyDetails propertyIterator = null;       //define iterator object
  #sql propertyIterator = {SELECT propertyNo AS propNo, street AS st
                  FROM PropertyForRent  WHERE staffNo = "SG37" ORDER BY propertyNo};
/* Loop over each record in result and print out details using method based on attribute name */
  while (propertyIterator.next() ) {
            System.out.println ("Property number: ", propertyIterator.propNo());
            System.out.println ("Street: ", propertyIterator.st());
       }
 propertyIterator.close()
 }
}
```

Figure L.5 Multirow query using SQLJ.

L.6 JavaServer Pages

In Section 30.7.5 we examined JavaServer Pages (JSP), a Java-based server-side scripting language that allows static HTML to be mixed with dynamically generated HTML. The following example illustrates the use of JSP.

EXAMPLE L.6 Use of JSP

Use JSP to list all records in the Staff table.

This example demonstrates the use of JSP based on Example L.2. The code for the JSP is shown in Figure L.6 and illustrates the some of the JSP constructs that can be embedded in a page. For example, the PAGE directive defines the underlying language as Java and specifies which classes should be imported for the JSP. Note that in this case, we could replace the database specific code with a JavaBean component, StaffQuery say, and use the Bean with the following code:

```
<jsp:useBean id = "test" class = "StaffQuery.MyBean" />
```

```
<HTML>
<HEAD>
<TITLE>DreamHome Staff Query</TITLE>
</HEAD>
<BODY>
<CENTER><H2><I>DreamHome</I> Staff Query</H2></CENTER>
<%-- Start of JSP scriptlet --%>
<%@page language= "java" import= "java.sql.*" %>
<% Class.forName("sun.jdbc.odbc.JdbcOdbcDriver").newInstance();
  Connection dbConn = DriverManager.getConnection("jdbc:odbc:DreamHomedbDSN", "admin", "dbapass");
  Statement stmt = dbConn.createStatement();
<%-- Set up the query and retrieve the results --%>
  ResultSet recordSet = stmt.executeQuery("SELECT staffNo, fName, lName, position FROM Staff");
  boolean isEmpty = !recordSet.next();
  boolean hasData = !isEmpty;
%>
<TABLE BORDER = "1" ALIGN = "CENTER">
<THEAD> <TR><TH>staffNo</TH><TH>fName</TH>
      <TH>lName</TH><TH>position</TH></TR></THEAD>
<% while (hasData) {
%>
     <TR>
       <TD><%= (String) recordSet.getObject("staffNo") %></TD>
       <TD><%= (String) recordSet.getObject("fName") %></TD>
       <TD><%= (String) recordSet.getObject("lName") %></TD>
       <TD><%= (String) recordSet.getObject("position") %></TD>
     </TR>
<%hasData = recordSet.next();}
%>
</TABLE>
<%
  recordSet.close();
  dbConn.close();
%>
</BODY>
</HTML>
```

Figure L.6 Sample JavaServer Page (JSP).

L.7 Active Server Pages and ActiveX Data Objects

In Section 30.8.2 we examined Microsoft's Active Server Pages (ASP) and ActiveX Data Objects (ADO), which allow server-side scripting to interface to a DBMS. The following example illustrates the use of ASP and ADO.

EXAMPLE L.7 Use of Active Server Pages and ActiveX Data Objects

Create an ASP application that accesses the DreamHome database and returns details about staff and the properties that they manage.

To answer this query in the *DreamHome* Microsoft Office Access database, we could use the following SQL statement:

```
SELECT p.propertyNo, p.street, p.city, p.staffNo, s.fName, s.lName
FROM Staff s INNER JOIN PropertyForRent p ON s.staffNo = p.staffNo;
```

The corresponding ASP to return this information to a Web browser is shown in Figure L.7 and the output from the execution of the ASP in Figure L.8.

```
<HTML>
<TITLE>DreamHome Estate Agents</TITLE>
<BODY bgcolor = # FFFFCC>
<CENTER><H2><I>DreamHome</I> Estate Agents</H2></CENTER>
<H3>Query Results for Staff Property Management</H3>
<!-- Check to see whether the user has a connection in place; if not, set one up.  -->
<%
Session.timeout = 3
If IsObject(Session("Dreamhome_conn")) Then
  Set conn = Session("Dreamhome_conn")
Else
  Set conn = Server.CreateObject("ADODB.Connection")
  conn.open "Dreamhome", "", ""
  Set Session("Dreamhome_conn") = conn
End If
%>
<!-- Set up the required SQL command, create the recordset, and open it up -->
<%
  sql = "SELECT p.propertyNo, p.street, p.city, p.staffNo, s.fName, s.lName
        FROM Staff s INNER JOIN PropertyForRent p ON s.staffNo = p.staffNo"
  Set rs = Server.CreateObject("ADODB.Recordset")
  rs.Open sql, conn, 3, 3
%>

<!-- Set up the HTML Table Header -->
<TABLE BORDER=1 BGCOLOR=#ffffff CELLSPACING=0>
<THEAD>
<TR>
<!-- Now output the column names for each of the attributes -->
<% For i = 0 To rs.Fields.Count – 1
%>
```

(*continued*)

Figure L.7 Sample ASP application using ADO.

```
<TH BGCOLOR=#c0c0c0 BORDERCOLOR=#000000 ><FONT SIZE=2 FACE= “Arial”
COLOR=#000000><%= rs(i).Name %></FONT></TH>
<% Next
%>
</TR>
</THEAD>
<!-- Now loop over all the records in the recordset and set up table entry for each one -->
<TBODY>
<%
On Error Resume Next
rs.MoveFirst
Do While Not rs.EOF
 %>
<TR >
<% For i = 0 To rs.Fields.Count – 1
  v = rs(i)
  If isnull(v) Then v = “  ”
%>
<TD VALIGN = TOP BORDERCOLOR=#c0c0c0 ><FONT SIZE=2 FACE= “Arial”
COLOR=#000000><%= v %><BR></FONT></TD>
<% Next
%>
</TR>
<%
rs.MoveNext
Loop
rs.Close
conn.Close
%>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

Figure L.7 *(Continued)*

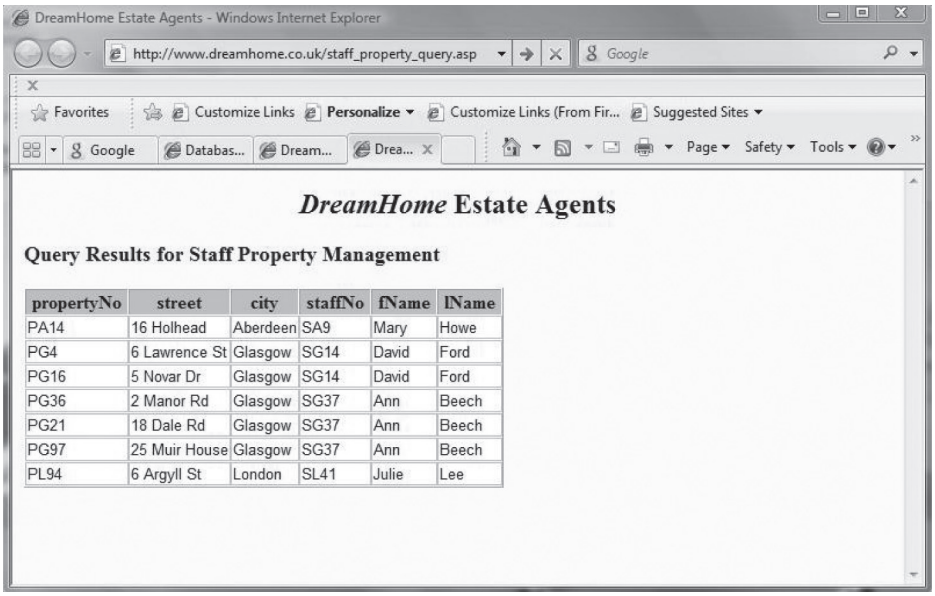

Figure L.8 Output from the Active Server Page shown in Figure L.7.

L.8 ADO.NET

In Section 30.8.5 we examined ADO.NET, a component of the .NET Framework class library. ADO.NET provides a disconnected data access model that is required in the Web environment and at the same time provides extensive support for XML. The following example illustrates the use of ADO.NET.

EXAMPLE L.8 Use of ADO.NET

Create a Windows application that displays staff at branch B003 along with the associated properties and viewings. Also, save the data to XML.

The VB.NET code for this application is shown in Figure L.9. The Windows output is shown in Figure L.10 and part of the XML file output in Figure L.11. Note that in Figure L.10 that each row in this DataGrid has an expanded "—" icon. The reason is that the DataGrid object has detected the relation between the Staff table and the PropertyForRent table. Clicking on the icon reveals all of the relations for which the Staff table is the parent (in this case, the Viewing table appears).

In this example, we have chosen to connect to a Microsoft Office Access database using OLEDB. The connection is achieved using OleDbConnection() and the DataAdapter

used is OleDbDataAdapter. To connect to an SQL Server database, the connection string would be changed to:

```
Dim strConnection As String = "Data Source=localhost;Initial Catalog=DreamHome;" _
    & "Integrated Security=True"
```

and we would use SQLConnection() and SQLDataAdapter in place of the OLEDB ones. The output is achieved by placing a DataGrid called myGrid on the form. Generating the XML file is achieved using the DataSet method WriteXML(). We could read this XML file back into a DataSet using the following statement:

```
Dim ds As New DataSet()
ds.ReadXml("c:\temp.xml")
```

The application also demonstrates how to access individual tables and rows in the DataSet at the end.

```
Imports System.Data.SqlClient
Imports System.Data.OleDb
Public Class Form1
        Inherits System.Windows.Forms.Form

    Private Sub Form1_Load(ByVal sender As Object, ByVal e As System.EventArgs) Handles MyBase.Load

    ' Open a database connection.
    Dim strConnection As String = _
        "Provider=Microsoft.Jet.OLEDB.4.0;Data Source=" _
        & "C:\Data\database\DreamHome.mdb"
    Dim cn As OleDbConnection = New OleDbConnection(strConnection)
    cn.Open()

    ' Set up a data adapter object.
    Dim strSql As String = "SELECT staffNo, fName, lName, branchNo FROM Staff" _
                           & " WHERE branchNo = 'B003'"
    Dim da As OleDbDataAdapter = New OleDbDataAdapter(strSql, cn)

    ' Load a data set.
    Dim ds As DataSet = New DataSet("MyDataSetName")
    da.Fill(ds, "StaffB003")

    ' Set up a new data adapter object.
    strSql = "SELECT propertyNo, street, city, p.staffNo" _
             & " FROM Staff s, PropertyForRent p" _
             & " WHERE (s.staffNo = p.staffNo)" _
             & "  AND (s.branchNo = 'B003')"
    da = New OleDbDataAdapter(strSql, cn)

    ' Load the data set.
    da.Fill(ds, "PropertyForRentB003")

    ' Set up a new data adapter object.
    strSql = "SELECT Viewing.*" _
             & " FROM Viewing, Staff, PropertyForRent" _
             & " WHERE (Staff.staffNo = PropertyForRent.staffNo)" _
             & "  AND (Viewing.propertyNo = PropertyForRent.propertyNo)" _
             & "  AND (Staff.branchNo = 'B003')"
    da = New OleDbDataAdapter(strSql, cn)
```

(continued)

Figure L.9 Sample application using ADO.NET.

```
    ' Load the data set.
    da.Fill(ds, "ViewingB003")

    ' Close the database connection.
    cn.Close()

    ' Create a relation.
    ds.Relations.Add("PropertyForRent", _
            ds.Tables("StaffB003").Columns("staffNo"), _
            ds.Tables("PropertyForRentB003").Columns("staffNo"))

    ds.Relations.Add("Viewing", _
            ds.Tables("PropertyForRentB003").Columns("propertyNo"), _
            ds.Tables("ViewingB003").Columns("propertyNo"))

    ' Bind the data set to a grid.
    myGrid.SetDataBinding(ds, "StaffB003")

    ' Save as XML.
    ds.WriteXml("c:\temp.xml")

    ' Access the data set as tables and rows.
    Dim dt As DataTable = ds.Tables.Item("PropertyForRentB003")
    Dim rowCustomer As DataRow
    For Each rowCustomer In dt.Rows
        Console.WriteLine(rowCustomer.Item("propertyNo"))
        Console.WriteLine(rowCustomer.Item("street"))
        Console.WriteLine(rowCustomer.Item("city"))
    Next

    End Sub
End Class
```

Figure L.9 (*Continued*)

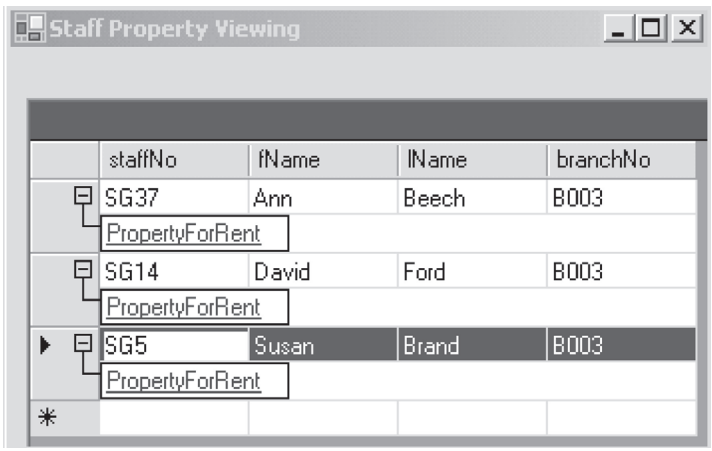

Figure L.10 Output from ADO.NET shown in Figure L.9.

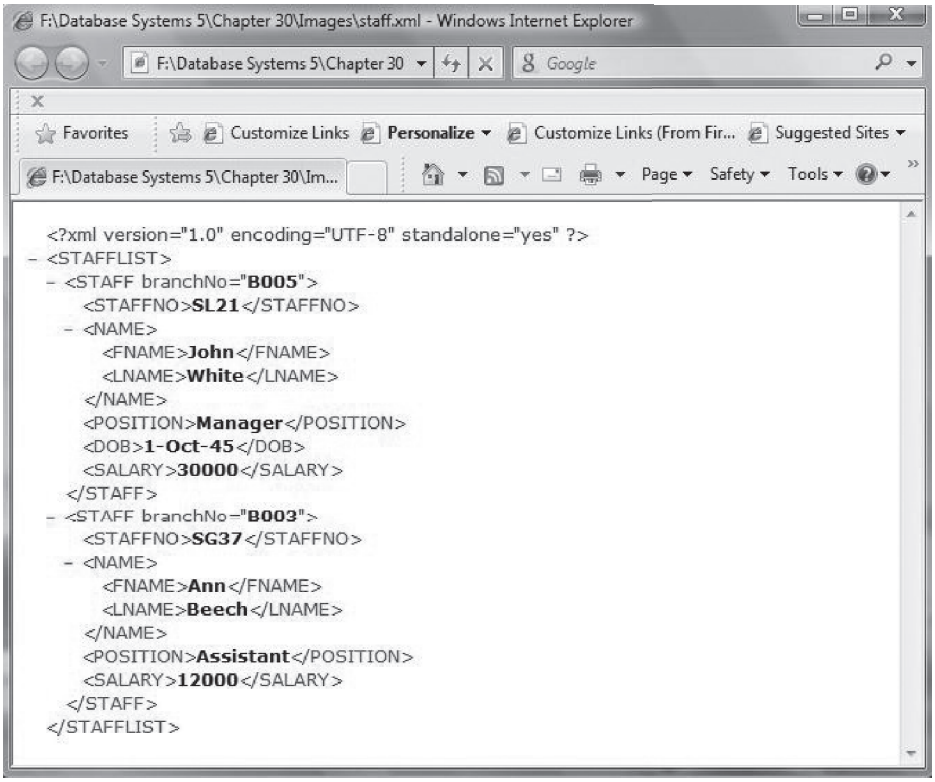

Figure L.11 XML output from ADO.NET shown in Figure L.9.

L.9 Oracle's PL/SQL Server Pages

In Section 30.9 we discussed Oracle's Internet Platform and its support for PL/SQL Server Pages (PSPs), which are analogous to ASPs and JSPs. We illustrate the use of PSPs in the following example.

EXAMPLE L.9 Use of PL/SQL Server Pages

Produce a PL/SQL Server Page to display the names of staff with a salary above a threshold specified by the user.

Assume we have created a Web page with the POST method shown in Figure L.12(a). We can then create the PSP shown in Figure L.12(b) to display the records that satisfy the user-specified minimum salary.

The file for a PL/SQL Server Page must have the extension ".psp" and can contain text and tags interspersed with PSP directives, declarations, and scriptlets. To identify a file as a PL/SQL Server Page, we include a <%@ page language="PL/SQL" %> directive at the beginning of the file. By default, the PL/SQL gateway transmits files as HTML

```
<HTML>
<TITLE>DreamHome Estate Agents Salary Query</TITLE>
<BODY bgcolor = # FFFFCC>
<CENTER><H2><I>DreamHome</I> Estate Agents Salary Query</H2></CENTER>
<FORM  METHOD="POST" ACTION="show_staff">
<p>Enter the minimum staff salary for search:
<INPUT TYPE="text" NAME="minsalary">
<INPUT TYPE="submit" VALUE="Submit">
</FORM>
</BODY>
</HTML>
```

Figure L.12(a) POST METHOD to invoke PL/SQL Server Page (PSP).

```
<%@ page language="PL/SQL" %>
<%@ plsql contentType= "text/html" %>
<%@ plsql procedure= "show_staff" %>
<%@ plsql parameter="minsalary" default="15000" %>
<HTML>
<TITLE>DreamHome Estate Agents – Staff Query</TITLE>
<BODY bgcolor = # FFFFCC>
<CENTER><H2><I>DreamHome</I> Estate Agents – Staff Query</H2></CENTER>
<H3>Query Results for Staff With Salary Above Specified Threshold <%= minsalary %></H3>
<!-- Set up the HTML Table Header -->
<TABLE BORDER = "1" ALIGN = "CENTER">
<THEAD> <TR><TH>staffNo</TH><TH>fName</TH>
      <TH>lName</TH><TH>position</TH></TR></THEAD>
<!-- Now loop over all the records in the recordset and set up table entry for each one -->
<TBODY>
<% FOR item IN (SELECT staffNo, fName, lName, salary
                    FROM Staff WHERE salary > minsalary ORDER BY salary)
loop %>
    <TR>
      <TD><%= item.staffNo %></TD>
      <TD><%= item.fName %></TD>
      <TD><%= item.lName %></TD>
      <TD><%= item.salary %></TD>
    </TR>
<% end loop; %>
</TBODY>
<TFOOT></TFOOT>
</TABLE>
</BODY>
</HTML>
```

Figure L.12(b) Sample PL/SQL Server Page.

documents, so that the browser formats them according to the HTML tags. To interpret the document as XML, plain text, or some other document type, a <%@ page contentType=". . ." %> directive can be used, specifying text/html, text/xml, text/plain, or some other MIME type. We have included a directive to indicate that it is HTML.

Each PSP corresponds to a stored procedure within the server. By default, the procedure is given the same name as the original file, with the ".psp" extension removed. The procedure can be given a different name using the directive <%@ page procedure=". . ." %>. To set up parameter passing for a PSP, we have included a <%@ plsql parameter=". . ." %> directive. By default, parameters are of type VARCHAR2 (a different type can used by specifying a type=". . ." attribute within the directive). To set a default value, so that the parameter becomes optional, we have included a default="15000" attribute in the directive.

Compare this script with the ASP and JSP created in earlier examples.