Chapter 23
ADS with ODBC, PHP, and DBI/Perl

Note: This chapter accompanies the book Advantage Database Server: A Developer's Guide, 2nd Edition, by Cary Jensen and Loy Anderson (2010, ISBN: 1453769978). For information on this book and on purchasing this book in various formats (print, e-book, etc), visit: http://www.JensenDataSystems.com/ADSBook10

Note: We want to thank Lance Schmidt from the Advantage team at Sybase. Lance provided us with invaluable assistance with the information in this chapter, and in particular, with the PHP examples used here and in previous editions of this book.

Each of the preceding chapters in this section describes building client applications using an IDE (integrated development environment) such as Delphi, or Visual Studio for .NET. This final chapter introduces you to a collection of related data access mechanisms that are not associated with a particular IDE, or even a specific operating system. These are the Advantage ODBC (open database connectivity) Driver, the Advantage PHP Extension, and the Advantage DBI Driver (for Perl).

As in the previous chapters on writing Advantage client applications, this chapter is not intended to show you how to program in the environments supporting the covered drivers, focusing instead on how to connect to and use Advantage. For information on developing in the languages covered in this chapter, refer to the documentation or a good book on the subject.

Accessing Advantage Using the Advantage ODBC Driver

ODBC (open database connectivity) is based on the Open SQL CLI (call-level interface), a SQL-based standard for accessing data. Advantage supplies ODBC drivers for both Windows and Linux.

ODBC is an API (application programming interface). However, most developers who use ODBC to access their data do not make direct ODBC calls. Instead, they use an IDE that supports ODBC.

A good example of this can be found in the .NET framework, and the classes in the System.Data.ODBC namespace in particular. These classes make use of the ODBC API, through which you can use any installed ODBC driver. However, developers who use ODBC through these classes, including ODBCConnection, ODBCCommand, and ODBCDataAdapter do not make direct ODBC calls. Instead, they use the ADO.NET interfaces of the .NET Framework Class Library (FCL).

While ODBC is an older standard, compared with OLE DB (and the related ActiveX data objects) and ADO.NET, it is easily the most widely supported data access mechanism for Windows and Linux. Nearly every database currently available is supported by at least one, and in some cases several, ODBC drivers. Consequently, every development environment for Windows and Linux that we are aware of provides some support for ODBC. Even Java, with its JDBC-ODBC bridge class 1 driver (Java database connectivity/open database connectivity), provides support for ODBC.

The Advantage ODBC Driver is compliant with the core API and level 1 API for ODBC 2.0. In addition, it supports most of the level 2 API functions. For a complete list of ODBC functions and information on Advantage's ODBC conformance level, see the Advantage help.

Who Should Use the Advantage ODBC Driver

There are two groups of users who should use the Advantage ODBC Driver. The first group consists of those developers who are using development environments for which there are no alternative drivers. For example, if you are using a proprietary development environment that does not support the ACE (Advantage Client Engine) API, Java, .NET, ADO, or VCL (Delphi's Visual Component Library), the Advantage ODBC Driver is your fallback solution.

For those developers for whom there is an alternative to ODBC, ODBC is usually a poor choice for connecting to Advantage. This is because the alternative solutions offer more options than does ODBC. In short, ODBC is the lowest common denominator for data access. All other data access solutions, including the two others covered in this chapter, provide a more extensive API than that supplied by ODBC alone.

The second group of developers who will use ODBC to access Advantage are those that use the Advantage PHP Extension or the Advantage DBI Driver. Both of these drivers, which are used by Web developers through the PHP and Perl languages, respectively, connect to Advantage through the ODBC driver. However, these drivers supply additional support beyond that provided by plain ODBC.

Connecting to Advantage Using the Advantage ODBC Driver

Before you can execute SQL statements against Advantage, you must obtain a connection to Advantage through the ODBC driver. Advantage supports two ODBC API functions for obtaining a connection. These are SQLConnect and SQLDriverConnect.

You use SQLConnect when there is a DSN (data source name) for the directory or data dictionary that you want to connect to. SQLDriverConnect, by comparison, does not require a DSN. Instead, the connection request is passed to SQLDriverConnect in its connection string. How you connect using these functions is described in the following sections.

Connecting Through ODBC Using a Data Source Name

A DSN is a definition that is stored on the workstation, and can be used to connect to data using an ODBC driver. Under the Windows operating system, DSN definitions are stored in the Windows Registry, while in Linux these definitions appear in a configuration file named odbc.ini.

Windows users typically do not add the Windows Registry entries for a DSN manually. Instead, they use an applet found in either the Control Panel (for older Windows installations), or the Administrative Tools page of the Control Panel. The name of this applet depends on which operating system you are using, but it always includes the letters ODBC. In Windows 2000 and later (which includes Vista and Windows 7), it is called Data Sources (ODBC).

To define a DSN manually, run this utility after you have installed the Advantage ODBC Driver on the workstation. If your client application is going to run under an end user account, you can add a user DSN. If your application is going to run under some other account, such as IUSER_MACHINE (used by Microsoft's Internet Information Server), add a system DSN.

Once you have decided to add either a user or system DSN (by selecting either the User or System tabs of this applet), click the Add button. Windows responds by displaying the Create New Data Source dialog box shown in Figure 23-1. Select Advantage StreamlineSQL ODBC from this list, and then click Finish.

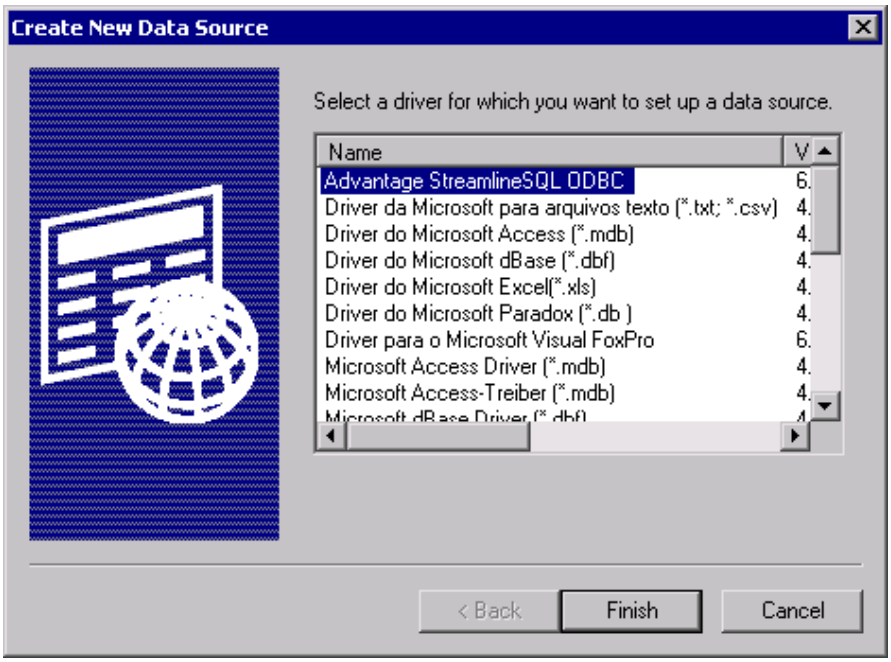

Figure 23-1: The Create New Data Source dialog box in Windows

You will then see the Advantage StreamlineSQL ODBC Driver Setup dialog box, shown in Figure 23-2. You use this dialog box to configure the DSN.

Set Data Source Name to the name you will use to refer to this DSN, and provide an optional description for the Description field.

If you will use this DSN to connect to a data dictionary, check the Data Dictionary checkbox and enter the full path to the data dictionary in the provided field. If you will use this DSN to connect to free tables, enter the data directory path here. In most cases, you will want to use a UNC (universal naming convention) path in this field.

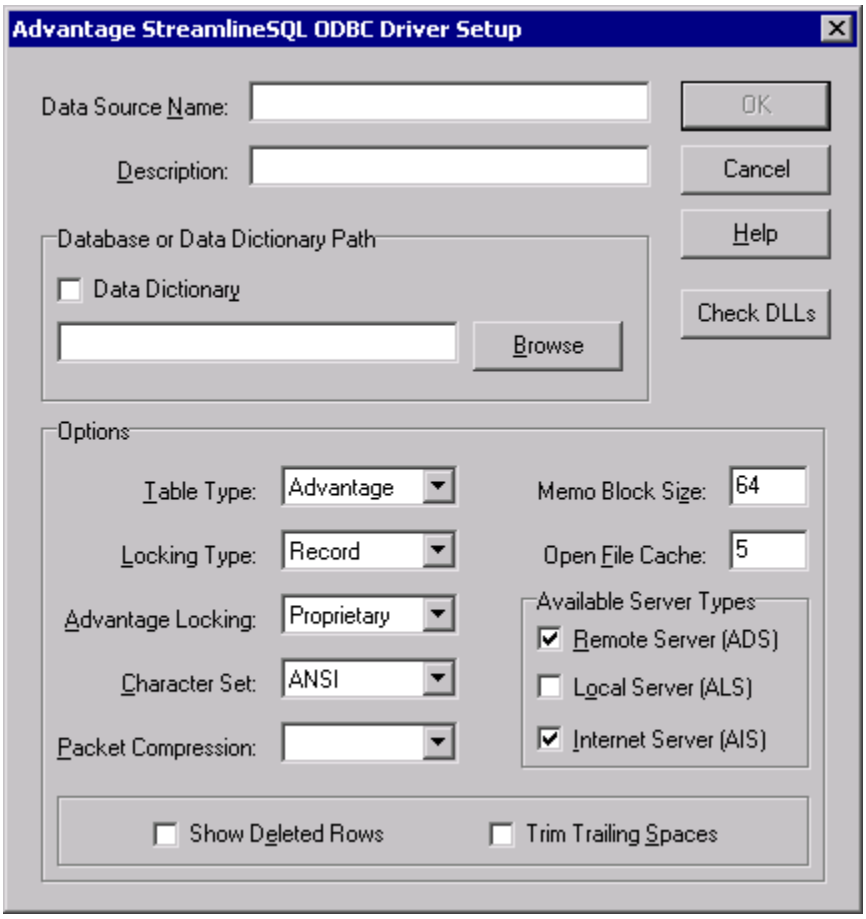

Figure 23-2: The Advantage StreamlineSQL ODBC Driver Setup dialog box

Use the Table Type, Locking Type, Advantage Locking, Character Set, and Packet Compression dropdown lists to define what data you are connecting to and how. To configure the size of memo blocks created by ODBC, or to adjust the number of tables to cache, set the corresponding fields. Also, you can configure the ODBC driver to show deleted rows for DBF tables, as well as trim trailing spaces from character fields.

When you are done configuring your DSN, click the OK button to save the DSN definition in the Windows Registry.

It is also possible to create a DSN by writing to the Windows Registry programmatically. This approach is useful if you want to create an automated setup for your client applications, rather than having to enter the DSN information manually on every machine. Note, however, that you should extensively test any code that writes to the Windows Registry, after making a backup of the Registry, as inappropriate changes to the Windows Registry can render a computer unstable or even unusable.

Tip: For guidance on creating Registry entries programmatically, start by adding a DSN using the Data Sources (ODBC) applet. Then, inspect the entries that this applet added to the Registry for the keys, values, and data that you need to insert. You will find these entries in HKEY_CURRENT_USER\ SOFTWARE\ODBC\ODBC.INI for user DSNs, and HKEY_LOCAL_MACHINE\SOFTWARE\ODBC\ ODBC.INI for system DSNs. You can also find information on creating DSNs at http://msdn.Microsoft.com.

In Linux, you create a DSN by adding entries to the odbc.ini file. Refer to the Advantage help for information on working with the odbc.ini file.

Once you have created the DSN that you are going to use, you will be able to call the SQLConnect function of the ODBC API. This function takes seven parameters. The first parameter is a connection handle that you previously allocated by calling SQLAllocHandle. The remaining parameters are string and integer pairs, where you pass the DSN name, user name, and password in the second, fourth, and sixth parameters, respectively; and the lengths of these strings in the third, fifth, and seventh parameters.

Connecting Through ODBC Using a Connection String

The primary drawback to using a DSN is that you must define the DSN on each workstation, which increases the complexity of your client installations. Fortunately, ODBC provides an alternative to using a DSN. This second mechanism employs a connection string.

The ODBC API includes two functions that accept a connection string: SQLDriverConnect and SQLBrowseConnect. Advantage only supports SQLDriverConnect.

SQLDriverConnect takes eight parameters. The first parameter is a connection handle, which you obtain by calling SQLAllocHandle, and the second is the Windows handle of your client application. The third and fifth parameters are used for the input connection string and the completed connection string, respectively. (The completed connection string is the version of the connection string used by ODBC to connect to the database. It includes any parameters that have been expanded by ODBC, as well as any default values not included in the input connection string.) The fourth parameter is the size of the input connection string, and the sixth parameter is the size of the buffer that you have allocated for the completed connection string.

The seventh parameter is the size of the completed connection string that was written to the buffer referenced in the fifth parameter, and the eighth parameter permits you to configure whether or not the ODBC driver manager should prompt the user for additional connection information, if needed. For example, if you pass empty strings in place of the user name and password parameters, and a user name and password are required to connect, you can instruct SQLDriverConnect to prompt the user for this information at runtime.

The connection string itself consists of zero or more parameters that you use to connect to Advantage in the form of name/value pairs. The name and value parts are separated by

an equal sign (=), and individual name/value pairs are separated by semicolons. Table 23-1 shows a complete list of the connection string parameters and the values that you can assign to them.

Parameter	Description
AdvantageLocking	Set to ON to use Advantage proprietary locking, or OFF to use compatibility locking. The default value is ON.
CharSet	Set to either ANSI or OEM. The default is ANSI.
Compression	Set to ALWAYS, INTERNET, NEVER, or empty. If left empty (the default), the ADS.INI file will control the compression setting. This parameter is not used by ALS.
CommType	The communication protocol to use to connect to ADS. Under Windows and Linux, the default is UDP_IP. For Novell Netware, the default is IPX. To use TCP/IP, set ComType to TCP_IP.
DefaultType	Set to FoxPro, Advantage, or Clipper. This parameter is ignored for data dictionary connections, but is required for free tables. The default is Advantage.
Description	This parameter is not used.
DirectoryPath	The path to the directory (for free table connections) or the path to the data dictionary, including the data dictionary name. It is recommended that this path be a UNC path. Data Source is a required parameter.
Language	OEM, ANSI, or a named collation. When this parameter is included, it overrides the Charset parameter.
Locking	FILE or RECORD. Defines what type of lock is applied during an update. The default is RECORD.
MaxTableCloseCache	Set this parameter to the number of underlying tables to hold in cache when cursors are opened and closed. The default value is 25.
MemoBlockSize	Use this parameter to define the block size that ODBC will use for memo fields. This value is always 512 for Clipper-compatible DBF tables (DBF/DBT). The default is 64 for FoxPro-compatible DBF tables (DBF/FPT), and 8 for Advantage proprietary tables (ADT/ADM).
PWD	When connecting to a data dictionary that requires logins, set to the user's password.
RightsChecking	Set to OFF to ignore client rights, or ON to respect them. The default is ON.
Rows	Set to TRUE to display deleted records in DBF files, or FALSE to suppress them. The default is FALSE.
ServerTypes	Set to an integer between 1 and 7 to define the server types the ODBC driver should attempt to connect to. Set to 1 for ALS, 2 for ADS, and 4 for Internet. To attempt to connect to more than one server type, set this parameter to the sum

	of the values. For example, to attempt to connect to ADS and then to ALS if the ADS connection fails, set ServerType to 3.
SQLTimeout	The maximum number of second after which a SQL statement that has not completed will be aborted.
TrimTrailingSpaces	Set to TRUE to trim trailing spaces from character fields, or FALSE to preserve trailing spaces.
UID	If connecting to a data dictionary that requires logins, set to the user's user name.

Table 23-1: The Advantage ODBC Driver Connection String Parameters

Examples of ODBC connections strings are provided in the following discussion of PHP.

Accessing Advantage Using the Advantage PHP Extension

PHP (the PHP: Hypertext Preprocessor language) is an open source scripting language that can be embedded into HTML documents in order to generate dynamic content for the World Wide Web. PHP can be used with any Web server that supports the PHP scripting engine. Most PHP development is done with Apache server for Linux, but other Web servers, such as Microsoft's IIS (Internet Information Server) can be configured to support PHP.

When a properly configured Web server reads a text file containing PHP commands, it runs the PHP scripting engine to execute those commands. The Web server then replaces the PHP commands with whatever output the scripting instructions call for. In most cases, the scripting commands are replaced either by simple text, HTML, or both.

Unlike client-side scripting, such as browser-executed JavaScript, all of the PHP commands are processed on the server—the client browser never receives them. As a result, end users cannot discover the PHP commands that you include in your PHP files.

After installing the Advantage PHP Extension, you will need to configure your Web server to load the PHP extension. Refer to your Web server documentation or your PHP add-on documentation for information on how to configure your Web server to use the Advantage PHP Extension.

Once you have enabled the Advantage PHP Extension, you can call any of the Advantage extended functions from within your PHP files. Table 23-2 contains a list of these available functions.

ads_autocommit	ads_binmode	ads_close
ads_close_all	ads_columnprivileges	ads_columns
ads_commit	ads_connect	ads_cursor

ads_do	ads_error	ads_errormsg
ads_exec	ads_execute	ads_fetch_array
ads_fetch_into	ads_fetch_object	ads_fetch_row
ads_field_len	ads_field_name	ads_field_num
ads_field_precision	ads_field_scale	ads_field_type
ads_foreignkeys	ads_free_result	ads_gettypeinfo
ads_longreadlen	ads_next_result	ads_num_fields
ads_num_rows	ads_pconnect	ads_prepare
ads_primarykeys	ads_procedurecolumns	ads_procedures
ads_result	ads_result_all	ads_rollback
ads_setoption	ads_specialcolumns	ads_statistics
ads_tableprivileges	ads_tables	

Table 23-2: The Advantage PHP Extended Functions

If you want to see a list of all of the functions supported by the Advantage PHP Extension, create and retrieve the following PHP file from a PHP-enabled Web server (or alternatively, add these commands to a file and pass the filename as a command-line parameter to the php.exe executable):

```
<html><body>
<?$functions = get_extension_funcs('advantage');
echo "Functions available in the Advantage PHP Extension:<P>";
foreach($functions as $func)
  echo $func."<br>";
echo "<br>"; ?>
</body>
</html>
```

The following sections demonstrate how to perform a number of essential tasks with Advantage and PHP. Unlike many of the examples of Advantage access presented in the preceding chapters, these examples do not include administrative operations such as granting rights to a newly created table. While there is nothing to stop you from opening an administrative connection with PHP, these types of operations are rarely performed via a browser interface, as they represent a potential security risk. If you find that you do need to perform administrative tasks using PHP, extrapolate one or more of the examples given in earlier chapters using the PHP extended functions.

Note: In order to run the examples provided in the following section, you must have a PHP-enabled Web browser, and have correctly configured a Web server–accessible directory to hold these PHP executable files. See the documentation for performing these tasks, or visit http://www.php.net for further information.

A Sample PHP Web Site

In order to demonstrate the basic use of Advantage using PHP, we have provided you with a simple PHP-based Web site with this book's code download. The main page of this site is named index.htm, and it produces the page shown in Figure 23-3 when rendered in a Web browser.

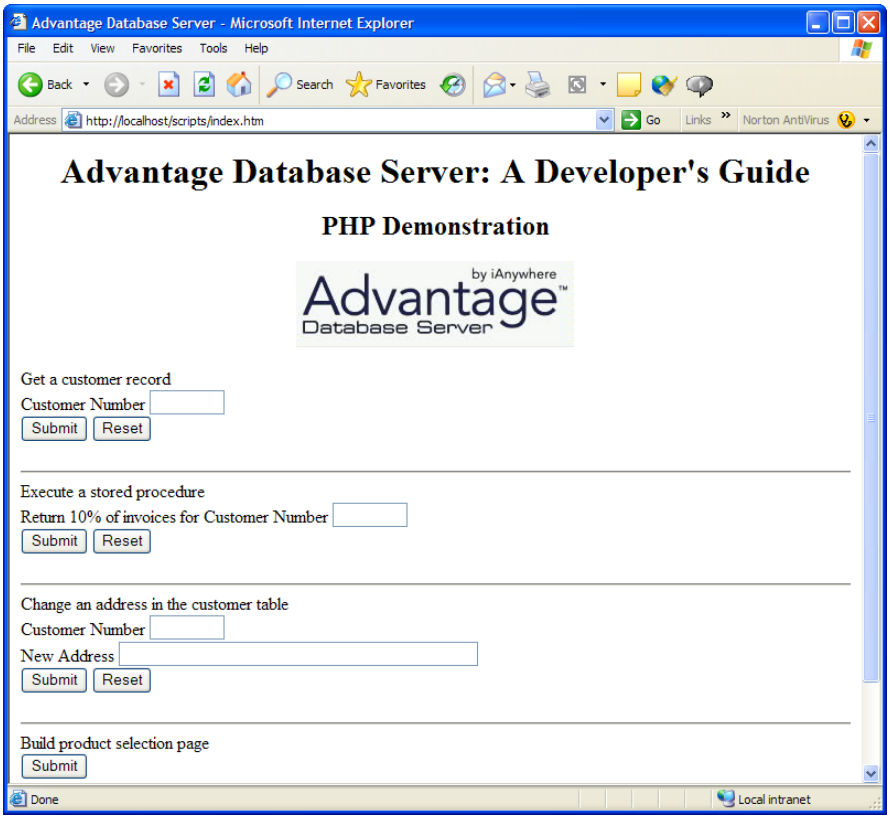

Figure 23-3: The main page of a PHP-based Web site

Code Download: The examples in this chapter can be found in the PHP directory on this book's code download (see Appendix A).

This Web page contains five different HTML forms, each of which submits an HTTP (Hypertext Transfer Protocol) GET request to an associated PHP file. The following is the contents of this HTML file:

```
<html>
<head>
<title>Advantage Database Server</title>
</head>
<body>
```

```
<form method="GET" action="getcustomer.php"
  name="getcustomer">

  <h1 align="center">
   Advantage Database Server: A Developer's Guide</h1>
  <h2 align="center">PHP Demonstration</h2>
  <p align="center">
  <img border="0" src="advantage_logo.jpg"></p>
  Get a customer record<br>
  Customer Number
    <input type="text" name="custnumber" size="7"><br>
  <input type="submit" value="Submit" name="B1">
    <input type="reset" value="Reset" name="B2"><br>
</form>
<form method="GET" action="storedproc.php"
  Name="storedproc">
  <hr>
  Execute a stored procedure<br>
  Return 10% of invoices for Customer Number
    <input type="text" name="custnumber" size="7"><br>
  <input type="submit" value="Submit" name="B1">
    <input type="reset" value="Reset" name="B2"><br>
</form>
<form method="GET" action="changeaddress.php"
  Name="changeaddress">
  <hr>
  Change an address in the customer table <br>
  Customer Number
    <input type="text" name="custnumber" size="7"><br>
  New Address <input type="text" name="newaddress"
  Size="50"><br>
  <input type="submit" value="Submit" name="UpdateAddress">
    <input type="reset" value="Reset" name="B1"><br>
</form>
<form method="GET" action="showproducts.php">
  <hr>
  Build product selection page<br>
  <input type="submit" value="Submit" name="B1"><br>
</form>
<form method="GET" action="showtables.php"
  Name="showtables">
  <hr>
  Show table names<br>
  <input type="submit" value="Submit" name="B1"><br>
</form>

</body>
</html>
```

Note: These forms were submitted using the HTTP GET method so that you can see any submitted data in the URL displayed in your Web browser. Many Web developers prefer to submit forms using the POST method.

There is another characteristic of these examples that is worth noting. Several of these example PHP files expect user input, which is then incorporated into SQL queries. Parameterized queries are used when user input is incorporated into the SQL queries in these examples. As you learned in Chapter 12 (*Introduction to Using Advantage SQL*), since Advantage version 7.0 you can execute SQL scripts that contain two or more SQL statements, separated by semicolons. While this is convenient, it exposes a potential security risk if you do not use parameterized queries. Because all user input is incorporated into queries using parameters, this security risk is eliminated.

Connecting to Advantage Using PHP

You connect to Advantage from PHP by calling the function ads_connect. This function takes either three or four arguments and returns a connection resource. The first argument is an ODBC connection string. The PHP driver connects through the ODBC API by invoking SQLDriverConnect, to which it passes this connection string. See Table 23-1 for the required and valid connection string parameters.

The second and third parameters are the user name and password to use for the connection, and the optional fourth parameter is used to define what type of cursor you want returned. The valid values for this fourth parameter are:

- 0 (SQL_CURSOR_FORWARD_ONLY)
- 1 (SQL_CURSOR_DYNAMIC)
- 2 (SQL_CURSOR_KEYSET_DRIVEN)

If you omit this parameter, a live (dynamic) cursor is returned.

The following is an example of a call to ads_connect:

```
$rConn = ads_connect("DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password", 1);
```

The connection string in this command attempts to connect to the DemoDictionary data dictionary located on a share named *share* on a server named *server*. In addition, this connection string specifies that it wants to connect to ADS (the license for ALS does not permit you to connect to ALS from a Web server). As is the case with ODBC connections, all parameters not included in the connection string will be expanded using the default parameter values.

Once you are through with the connection, you must close it. You do this by invoking ads_close, passing the connection resource obtained from the call to ads_connect. The following is a simple example of this function call:

```
ads_close( $rConn );
```

Using Parameterized Queries in PHP

Once you have established a connection, you invoke the ads_prepare function to prepare a parameterized query. This function takes two parameters. The first parameter is the connection resource obtained by calling ads_connect, and the second is a string containing the parameterized query.

Once you have prepared your parameterized query, you bind the parameters and execute the query by calling ads_execute. This function takes two parameters: the handle of the prepared statement returned from the ads_prepare function, and an array of values to bind to the parameters, based on their ordinal position within the query.

The execution of a parameterized query is demonstrated in the following listing, which contains the PHP statements from the getcustomer.php file:

```
<?
$rConn = ads_connect( "DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password" );
$rStmt = ads_prepare( $rConn, "SELECT * FROM customer ".
  "WHERE [Customer ID] = ?" );
$aParams = array( 1 => $_GET[ "custnumber" ] );
$rResult = ads_execute( $rStmt, $aParams );
if ( ads_fetch_row( $rStmt ) )
   {
   $strFirstName = ads_result( $rStmt, "First Name" );
   $strLastName = ads_result( $rStmt, "Last Name" );
   $strAddress = ads_result( $rStmt, "Address" );
   echo "Name: " . $strFirstName . " " .
     $strLastName . "<br>";
   echo "Address: " . $strAddress . "<br>";
   }
else
   {
   echo "Invalid Customer Number! <br>";
   }

ads_close( $rConn );
?>
```

As you can see, once the connection is established, and the parameterized query is prepared, a single element array is constructed from the custnumber value passed in the query string of this HTTP GET request (if you used a POST action, you would have read this value using the $_POST PHP function). The query is then executed.

Following the execution of the query, ads_fetch_row is called to retrieve one record from the result set. Individual fields of this record are read using the ads_result function.

If you enter customer number 12037 in the Customer Number field of the Get a customer record HTML form of index.htm and click Submit, your browser will display the page shown in Figure 23-4.

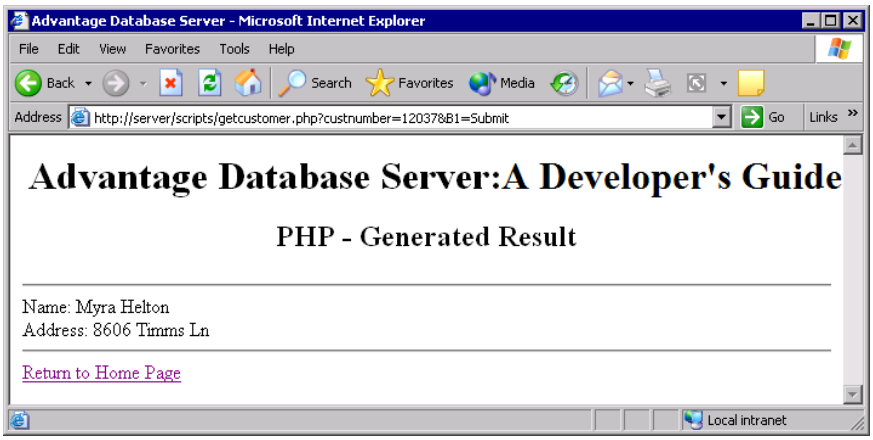

Figure 23-4: Result from the parameterized query example

Getting Tables from Result Sets Using PHP

After executing a query, you can easily display the entire result set by calling ads_result_all. This function takes the handle returned from a call to ads_execute in its first parameter, and an optional string containing HTML table element attributes in the second, and returns a string containing a complete HTML <TABLE> definition that includes one row for each record in the result set. This function is only used when you execute a query that returns one or more records.

The use of ads_result_all is demonstrated in the showtables.php script. The PHP statements from this file are shown in the following listing:

```
<?
$rConn = ads_connect( "DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password" );
$rStmt = ads_prepare( $rConn, "SELECT Name FROM ".
  "system.tables" );
$rResult = ads_execute( $rStmt );
ads_result_all( $rStmt );
ads_close( $rConn );
?>
```

When this script is rendered, it produces the Web page shown in Figure 23-5.

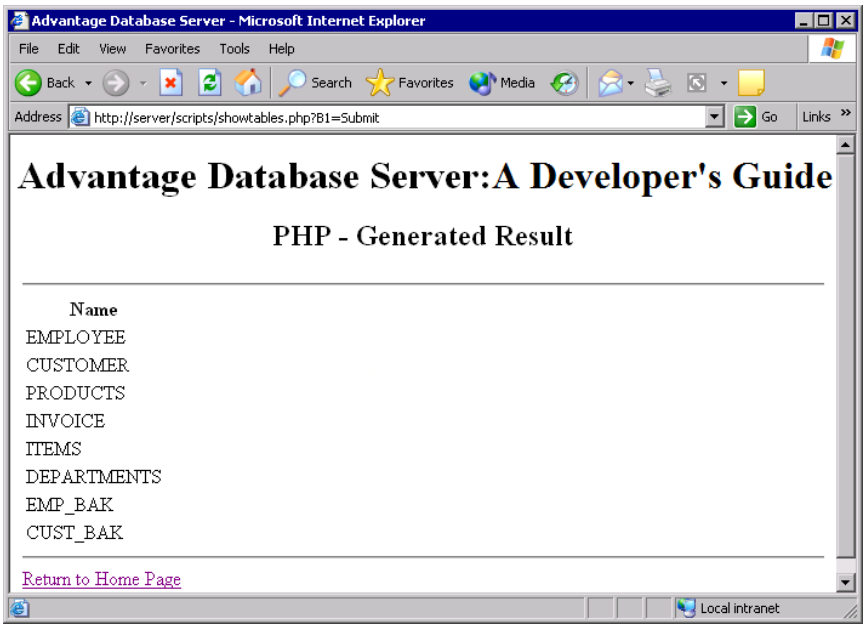

Figure 23-5: The Web page returned by showtables.php

As mentioned earlier, the ads_result_all function can accept a second, optional parameter. You use this parameter to pass a string that will be incorporated into the <TABLE> element. Normally you use this parameter to pass one or more table attributes that will be used to control the table's format and behavior, such as bgcolor, border, onclick, style, id, and width, to name a few.

Editing Data

Since PHP uses ODBC, and ODBC uses SQL to edit data, you change data in your database from a PHP script by executing a SQL UPDATE, INSERT, or MERGE statement. This is demonstrated in the following PHP statements from the script named changeaddress.php. This script expects two values, a customer ID and a string containing a new address, to be passed in the HTTP GET query string. These values are used to execute a parameterized UPDATE query statement. Once the update has been executed, this script performs a SQL SELECT to read the newly updated address from the CUSTOMER table:

```
<?
$rConn = ads_connect( "DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password" );
$rStmt = ads_prepare( $rConn, "UPDATE customer ".
  "SET Address = ? WHERE [Customer ID] = ?" );
$aUpdateParams = array( 1 => $_GET[ "newaddress" ],
                        2 => $_GET[ "custnumber" ] );
$rResult = ads_execute( $rStmt, $aUpdateParams );
$iRowsAffected = ads_num_rows( $rStmt );
if ( $iRowsAffected == 0 )
```

```
   {
   echo "Invalid customer ID!<br><br>\n";
   }
$rStmt = ads_prepare( $rConn, "SELECT * FROM customer ".
  "WHERE [Customer ID] = ?" );
$aSelectParams = array( 1 => $_GET[ "custnumber" ] );
$rResult = ads_execute( $rStmt, $aSelectParams );
if ( ads_fetch_row( $rStmt ) )
   {
   $strFirstName = ads_result( $rStmt, "First Name" );
   $strLastName = ads_result( $rStmt, "Last Name" );
   $strAddress = ads_result( $rStmt, "Address" );
   echo "Address successfully changed!<br><br>";
   echo "The new address is: " . $strAddress . "<br>";
   }
ads_close( $rConn );
?>
```

Scanning Result Sets

Scanning involves the sequential navigation of the records in a result set. Scanning is often done when you want to manually insert HTML based on your result set into the stream that is inserted in place of the PHP commands.

The following PHP commands are located in the showproducts.php script. These commands produce an HTML form that includes one radio button for each product found in the PRODUCT table. Because of the action attribute of the HTML form, the product code of the selected product name will be appended to the query string part of the HTTP GET command that will be submitted to the showproducts.php script.

```
<?
$rConn = ads_connect( "DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password" );
$rStmt = ads_prepare( $rConn, "SELECT [Product Name], ".
  "[Product Code] FROM Products" );
$rResult = ads_execute( $rStmt );
while ( ads_fetch_row( $rStmt ) )
   {
   $strProductName = ads_result( $rStmt, "Product Name" );
   $strProductCode = ads_result( $rStmt, "Product Code" );
   echo "<INPUT Type = \"radio\" Name = \"rb\" Value = \"" .
      trim( $strProductCode ) .  "\" > " .
      $strProductName . "<br>\n";
   }
ads_close( $rConn );
?>
```

When this PHP file is processed, it produces the Web page shown in Figure 23-6.

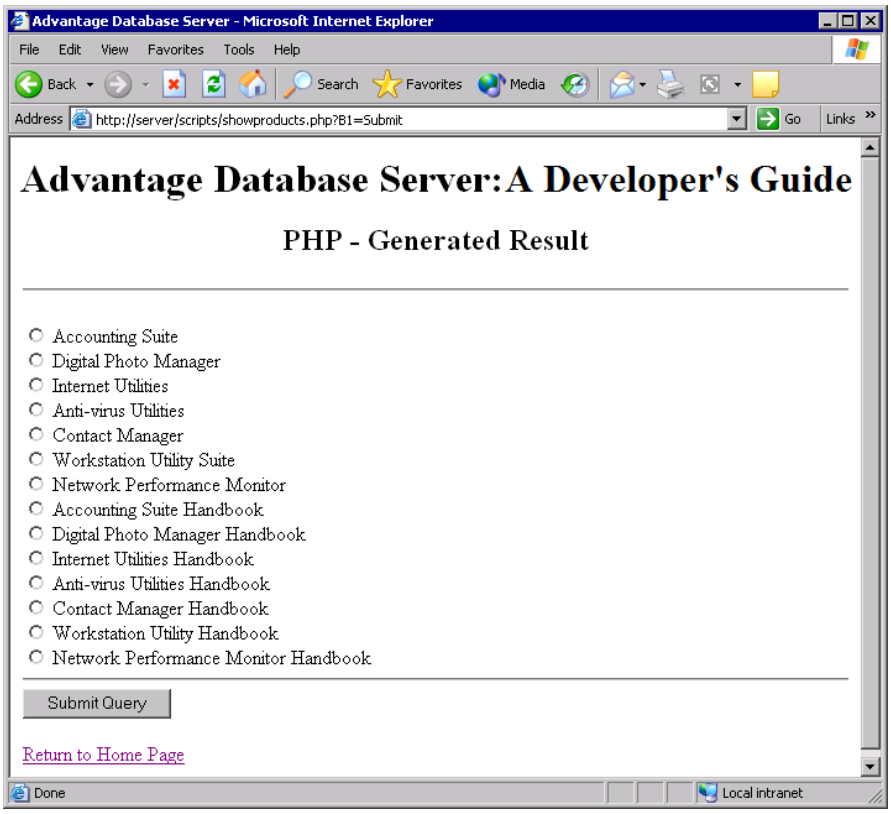

Figure 23-6: The output from showproducts.php

Calling a Stored Procedure

This final example demonstrates the execution of a stored procedure using PHP. Actually, as you can see from these statements, executing a stored procedure is no different than any other type of parameterized query. After a call to ads_prepare, an array is created to hold the parameter values, after which it is passed as an argument to ads_execute. Once again, the ads_result_all function is used to render an HTML table from the query result set. The following PHP commands are located in the storedproc.php script:

```
<?
$rConn = ads_connect( "DataDirectory=\\\\server\\share\\".
  "adsbook\\DemoDictionary.add;ServerTypes=2;",
  "adsuser", "password" );
$rStmt = ads_prepare( $rConn,
  "EXECUTE PROCEDURE Get10PercentSQL ( ? )" );
$aParams = array( 1 => $_GET[ "custnumber" ] );
$rResult = ads_execute( $rStmt, $aParams );
if ( $rResult == FALSE )
   {
   echo ads_errormsg( $rConn ) . "<br>\n";
   }
```

```
else
   {
   ads_result_all( $rStmt );
   }
ads_close( $rConn );
?>
```

Accessing Data Using the Advantage DBI Driver (for Perl)

Accessing data using the Advantage DBI Driver (for Perl) is very similar to that for PHP, since both drivers rely on the Advantage ODBC Driver, but there are important differences. Besides the language difference, there are four primary differences between the Advantage PHP Extension and the Advantage DBI Driver.

The first is associated with the connection string that the DBI driver will send to the ODBC API. This connection string always begins with the dbi:Advantage: string. Everything that follows this string is identical to the parameters listed in Table 23-1 that you use for both the Advantage ODBC Driver and the Advantage PHP Extension.

The following is an example of a connection string that makes a connection similar to that shown in the PHP examples:

```
use DBI;
$dbh = DBI->connect( 'dbi:Advantage:DataDirectory=\\server\share' .
  '\adsbook\DemoDictionary.add;ServerTypes=2;',
  'adsuser', 'password' );
```

The second difference, which you may have already noticed if you examined the preceding example, is that the extended functions for the DBI are similar in name (but not identical) to those used by the PHP Extension. Specifically, while the number and type of parameters are the same, the functions do not use the "ads_" prefix. For example, in PHP you connect to Advantage by calling the ads_connect function, but in Perl you call the connect method of the DBI object.

The third difference is that data access in Perl uses a different paradigm from PHP. This approach to data access should be familiar to any seasoned Perl developer.

And, finally, the fourth difference is that the Advantage DBI Driver can only produce a forward-scrolling cursor. In most Web applications, this is inconsequential since most CGI (common gateway interface) applications don't need to perform any navigation other than forward navigation.

For more information about the Advantage DBI Driver, see the Advantage help.