

Web Database Programming Using PHP

In the previous chapter, we gave an overview of database programming techniques using traditional programming languages, and we used the Java and C programming languages in our examples. We now turn our attention to how databases are accessed from scripting languages. Many Internet applications that provide Web interfaces to access information stored in one or more databases use scripting languages. These languages are often used to generate HTML documents, which are then displayed by the Web browser for interaction with the user. In our presentation, we assume that the reader is familiar with basic HTML concepts.

Basic HTML is useful for generating *static* Web pages with fixed text and other objects, but most Internet applications require Web pages that provide interactive features with the user. For example, consider the case of an airline customer who wants to check the arrival time and gate information of a particular flight. The user may enter information such as a date and flight number in certain fields of the Web page. The Web interface will send this information to the application program, which formulates and submits a query to the airline database server to retrieve the information that the user needs. The database information is sent back to the Web page for display. Such Web pages, where part of the information is extracted from databases or other data sources, are called *dynamic* Web pages. The data extracted and displayed each time will be for different flights and dates.

There are various techniques for programming dynamic features into Web pages. We will focus on one technique here, which is based on using the PHP open source server side scripting language. PHP originally stood for Personal Home Page, but now stands for PHP Hypertext Processor. PHP has experienced widespread use. The interpreters for PHP are provided free of charge and are written in the C language so

they are available on most computer platforms. A PHP interpreter provides a Hyper-text Preprocessor, which will execute PHP commands in a text file and create the desired HTML file. To access databases, a library of PHP functions needs to be included in the PHP interpreter, as we will discuss in Section 11.3. PHP programs are executed on the Web server computer. This is in contrast to some scripting languages, such as JavaScript, that are executed on the client computer. There are many other popular scripting languages that can be used to access databases and create dynamic Web pages, such as JavaScript, Ruby, Python, and PERL, to name a few.

This chapter is organized as follows. Section 11.1 gives a simple example to illustrate how PHP can be used. Section 11.2 gives a general overview of the PHP language and how it is used to program some basic functions for interactive Web pages. Section 11.3 focuses on using PHP to interact with SQL databases through a library of functions known as PEAR DB. Section 11.4 lists some of the additional technologies associated with Java for Web and database programming (we already discussed JDBC and SQLJ in Chapter 10). Finally, Section 11.5 contains a chapter summary.

11.1 A Simple PHP Example

PHP is an open source general-purpose scripting language. The interpreter engine for PHP is written in the C programming language so it can be used on nearly all types of computers and operating systems. PHP usually comes installed with the UNIX operating system. For computer platforms with other operating systems such as Windows, Linux, or Mac OS, the PHP interpreter can be downloaded from: <http://www.php.net>. As with other scripting languages, PHP is particularly suited for manipulation of text pages, and in particular for manipulating dynamic HTML pages at the Web server computer. This is in contrast to JavaScript, which is downloaded with the Web pages to execute on the client computer.

PHP has libraries of functions for accessing databases stored under various types of relational database systems such as Oracle, MySQL, SQLServer, and any system that supports the ODBC standard (see Chapter 10). Under the three-tier architecture (see Chapter 2), the DBMS would reside at the **bottom-tier database server**. PHP would run at the **middle-tier Web server**, where the PHP program commands would manipulate the HTML files to create the customized dynamic Web pages. The HTML is then sent to the **client tier** for display and interaction with the user.

Consider the PHP example shown in Figure 11.1(a), which prompts a user to enter the first and last name and then prints a welcome message to that user. The line numbers are not part of the program code; they are used below for explanation purposes only:

1. Suppose that the file containing PHP script in program segment P1 is stored in the following Internet location: <http://www.myserver.com/example/greeting.php>. Then if a user types this address in the browser, the PHP interpreter would start interpreting the code and produce the form shown in Figure 11.1(b). We will explain how that happens as we go over the lines in code segment P1.

(a)

```

//Program Segment P1:
0) <?php
1) // Printing a welcome message if the user submitted their name
   // through the HTML form
2) if ($_POST['user_name']) {
3)     print("Welcome, ") ;
4)     print($_POST['user_name']);
5) }
6) else {
7)     // Printing the form to enter the user name since no name has
       // been entered yet
8)     print <<< _HTML_
9)     <FORM method="post" action="$_SERVER['PHP_SELF']">
10)    Enter your name: <input type="text" name="user_name">
11)    <BR/>
12)    <INPUT type="submit" value="SUBMIT NAME">
13)    </FORM>
14)    _HTML_;
15) }
16) ?>

```

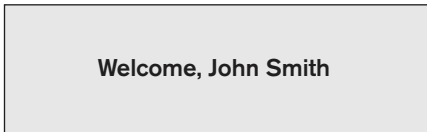
(b)



(c)



(d)


Figure 11.1

(a) PHP program segment for entering a greeting.
 (b) Initial form displayed by PHP program segment.
 (c) User enters name *John Smith*. (d) Form prints welcome message for *John Smith*.

2. Line 0 shows the PHP start tag `<?php`, which indicates to the PHP interpreter engine that it should process all subsequent text lines until it encounters the PHP end tag `?>`, shown on line 16. Text outside of these tags is printed as is. This allows PHP code segments to be included within a larger HTML file. Only the sections in the file between `<?php` and `?>` are processed by the PHP preprocessor.
3. Line 1 shows one way of posting comments in a PHP program on a single line started by `//`. Single-line comments can also be started with `#`, and end at the end of the line in which they are entered. Multiple-line comments start with `/*` and end with `*/`.
4. The **auto-global** predefined PHP variable `$_POST` (line 2) is an array that holds all the values entered through form parameters. Arrays in PHP are

dynamic arrays, with no fixed number of elements. They can be numerically indexed arrays whose indexes (positions) are numbered (0, 1, 2, ...), or they can be associative arrays whose indexes can be any string values. For example, an associative array indexed based on color can have the indexes {"red", "blue", "green"}. In this example, `$_POST` is associatively indexed by the name of the posted value `user_name` that is specified in the name attribute of the input tag on line 10. Thus `$_POST['user_name']` will contain the value typed in by the user. We will discuss PHP arrays further in Section 11.2.2.

5. When the Web page at <http://www.myserver.com/example/greeting.php> is first opened, the `if` condition in line 2 will evaluate to false because there is no value yet in `$_POST['user_name']`. Hence, the PHP interpreter will process lines 6 through 15, which create the text for an HTML file that displays the form shown in Figure 11.1(b). This is then displayed at the client side by the Web browser.
6. Line 8 shows one way of creating **long text strings** in an HTML file. We will discuss other ways to specify strings later in this section. All text between an opening `<<<_HTML_` and a closing `_HTML_;` is printed into the HTML file as is. The closing `_HTML_;` must be alone on a separate line. Thus, the text added to the HTML file sent to the client will be the text between lines 9 and 13. This includes HTML tags to create the form shown in Figure 11.1(b).
7. PHP **variable names** start with a `$` sign and can include characters, numbers, and the underscore character `_`. The PHP auto-global (predefined) variable `$_SERVER` (line 9) is an array that includes information about the local server. The element `$_SERVER['PHP_SELF']` in the array is the path name of the PHP file currently being executed on the server. Thus, the action attribute of the form tag (line 9) instructs the PHP interpreter to reprocess the same file, once the form parameters are entered by the user.
8. Once the user types the name *John Smith* in the text box and clicks on the **SUBMIT NAME** button (Figure 11.1(c)), program segment *P1* is reprocessed. This time, `$_POST['user_name']` will include the string "John Smith", so lines 3 and 4 will now be placed in the HTML file sent to the client, which displays the message in Figure 11.1(d).

As we can see from this example, a PHP program can create two different HTML commands depending on whether the user just started or whether they had already submitted their name through the form. In general, a PHP program can create numerous variations of HTML text in an HTML file at the server depending on the particular conditional paths taken in the program. Hence, the HTML sent to the client will be different depending on the interaction with the user. This is one way in which PHP is used to create *dynamic* Web pages.

11.2 Overview of Basic Features of PHP

In this section we give an overview of a few of the features of PHP that are useful in creating interactive HTML pages. Section 11.3 will focus on how PHP programs can access databases for querying and updating. We cannot give a comprehensive

discussion of PHP; there are many books that focus solely on PHP. Rather, we focus on illustrating certain features of PHP that are particularly suited for creating dynamic Web pages that contain database access commands. This section covers some PHP concepts and features that will be needed when we discuss database access in Section 11.3.

11.2.1 PHP Variables, Data Types, and Programming Constructs

PHP **variable names** start with the \$ symbol and can include characters, letters, and the underscore character (_). No other special characters are permitted. Variable names are case sensitive, and the first character cannot be a number. Variables are not typed. The values assigned to the variables determine their type. In fact, the same variable can change its type once a new value is assigned to it. Assignment is via the = operator.

Since PHP is directed toward text processing, there are several different types of string values. There are also many functions available for processing strings. We only discuss some basic properties of string values and variables here. Figure 11.2 illustrates some string values. There are three main ways to express strings and text:

1. **Single-quoted strings.** Enclose the string between single quotes, as in lines 0, 1, and 2. If a single quote is needed within the string, use the escape character (\) (see line 2).
2. **Double-quoted strings.** Enclose strings between double quotes as in line 7. In this case, *variable names appearing within the string* are replaced by the values that are currently stored in these variables. The interpreter identifies variable names within double-quoted strings by their initial character \$ and replaces them with the value in the variable. This is known as **interpolating variables** within strings. Interpolation does not occur in single-quoted strings.
3. **Here documents.** Enclose a part of a document between a <<<DOCNAME and end it with a single line containing the document name DOCNAME.

```

0) print 'Welcome to my Web site.';
1) print 'I said to him, "Welcome Home"';
2) print 'We\'ll now visit the next Web site';
3) printf('The cost is $%.2f and the tax is $%.2f',
   $cost, $tax) ;
4) print strtolower('AbCdE');
5) print ucwords(strtolower('JOHN smith'));
6) print 'abc' . 'efg'
7) print "send your email reply to: $email_address"
8) print <<<FORM_HTML
9) <FORM method="post" action="$ _SERVER[ 'PHP_SELF' ]">
10) Enter your name: <input type="text" name="user_name">
11) FORM_HTML

```

Figure 11.2

Illustrating basic PHP string and text values.

DOCNAME can be any string as long as it used both to start and end the here document. This is illustrated in lines 8 through 11 in Figure 11.2. Variables are also interpolated by replacing them with their string values if they appear inside here documents. This feature is used in a similar way to double-quoted strings, but it is more convenient for multiple-line text.

4. **Single and double quotes.** Single and double quotes used by PHP to enclose strings should be *straight* quotes ("") on both sides of the string. The text editor that creates these quotes should not produce *curly* opening and closing quotes (“ ”) around the string.

There is also a string concatenate operator specified by the period (.) symbol, as illustrated in line 6 of Figure 11.2. There are many string functions. We only illustrate a couple of them here. The function `strtolower` changes the alphabetic characters in the string to all lowercase, whereas the function `ucwords` capitalizes all the words in a string. These are illustrated in lines 4 and 5 in Figure 11.2.

The general rule is to use single-quoted strings for literal strings that contain no PHP program variables and the other two types (double-quoted strings and here documents) when the values from variables need to be interpolated into the string. For large blocks of multiline text, the program should use the *here documents* style for strings.

PHP also has numeric data types for integers and floating points and generally follows the rules of the C programming language for processing these types. Numbers can be formatted for printing into strings by specifying the number of digits that follow the decimal point. A variation of the `print` function called `printf` (print formatted) allows formatting of numbers within a string, as illustrated in line 3 of Figure 11.2.

There are the standard programming language constructs of for-loops, while-loops, and conditional if-statements. They are generally similar to their C language counterparts. We will not discuss them here. Similarly, *any value* evaluates to true if used as a Boolean expression *except for* numeric zero (0) and blank string, which evaluate to false. There are also literal true and false values that can be assigned. The comparison operators also generally follow C language rules. They are `==` (equal), `!=` (not equal), `>` (greater than), `>=` (greater than or equal), `<` (less than), and `<=` (less than or equal).

11.2.2 PHP Arrays

Arrays are very important in PHP, since they allow lists of elements. They are used frequently in forms that employ pull-down menus. A single-dimensional array is used to hold the list of choices in the pull-down menu. For database query results, two-dimensional arrays are used, with the first dimension representing *rows* of a table and the second dimension representing *columns* (attributes) within a row. There are two main types of arrays: numeric and associative. We discuss each of these in the context of single-dimensional arrays next.

A **numeric array** associates a numeric index (or position or sequence number) with each element in the array. Indexes are integer numbers that start at zero and grow incrementally. An element in the array is referenced through its index. An **associative array** provides pairs of (key => value) elements. The value of an element is referenced through its key, and all key values in a particular array must be unique. The element values can be strings or integers, or they can be arrays themselves, thus leading to higher dimensional arrays.

Figure 11.3 gives two examples of array variables: `$teaching` and `$courses`. The first array `$teaching` is associative (see line 0 in Figure 11.3), and each element associates a course name (as key) with the name of the course instructor (as value). There are three elements in this array. Line 1 shows how the array may be updated. The first command in line 1 assigns a new instructor to the course 'Graphics' by updating its value. Since the key value 'Graphics' already exists in the array, no new element is created but the existing value is updated. The second command creates a new element since the key value 'Data Mining' did not exist in the array before. New elements are added at the end of the array.

If we only provide values (no keys) as array elements, the keys are automatically numeric and numbered 0, 1, 2, This is illustrated in line 5 of Figure 11.3, by the `$courses` array. Both associative and numeric arrays have no size limits. If some value of another data type, say an integer, is assigned to a PHP variable that was holding an array, the variable now holds the integer value and the array contents are lost. Basically, most variables can be assigned to values of any data type at any time.

There are several different techniques for looping through arrays in PHP. We illustrate two of these techniques in Figure 11.3. Lines 3 and 4 show one method of looping through all the elements in an array using the `foreach` construct, and printing the key and value of each element on a separate line. Lines 7 through 10 show how a traditional for-loop construct can be used. A built-in function `count`

Figure 11.3

Illustrating basic PHP array processing.

```

0) $teaching = array('Database' => 'Smith', 'OS' => 'Carrick',
                    'Graphics' => 'Kam');
1) $teaching['Graphics'] = 'Benson'; $teaching['Data Mining'] = 'Li';
2) sort($teaching);
3) foreach ($teaching as $key => $value) {
4)     print " $key : $value\n";}
5) $courses = array('Database', 'OS', 'Graphics', 'Data Mining');
6) $alt_row_color = array('blue', 'yellow');
7) for ($i = 0, $num = count($courses); i < $num; $i++) {
8)     print "<TR bgcolor=\"" . $alt_row_color[$i % 2] . "\">";
9)     print "<TD>Course $i is</TD><TD>$course[$i]</TD></TR>\n";
10) }
```


(line 7) returns the current number of elements in the array, which is assigned to the variable `$num` and used to control ending the loop.

The example in lines 7 through 10 also illustrates how an HTML table can be displayed with alternating row colors, by setting the two colors in an array `$alt_row_color` (line 8). Each time through the loop, the remainder function `$i % 2` switches from one row (index 0) to the next (index 1) (see line 8). The color is assigned to the HTML *bgcolor* attribute of the `<TR>` (table row) tag.

The count function (line 7) returns the current number of elements in the array. The sort function (line 2) sorts the array based on the element values in it (not the keys). For associative arrays, each key remains associated with the same element value after sorting. This does not occur when sorting numeric arrays. There are many other functions that can be applied to PHP arrays, but a full discussion is outside the scope of our presentation.

11.2.3 PHP Functions

As with other programming languages, **functions** can be defined in PHP to better structure a complex program and to share common sections of code that can be reused by multiple applications. The newer version of PHP, PHP5, also has object-oriented features, but we will not discuss these here because we are focusing on the basics of PHP. Basic PHP functions can have arguments that are *passed by value*. Global variables can be accessed within functions. Standard scope rules apply to variables that appear within a function and within the code that calls the function.

We now give two simple examples to illustrate basic PHP functions. In Figure 11.4, we show how we could rewrite the code segment P1 from Figure 11.1(a) using functions. The code segment P1' in Figure 11.4 has two functions: `display_welcome()` (lines 0 to 3) and `display_empty_form()` (lines 5 to 13). Neither of these functions has arguments; nor do they have return values. Lines 14 through 19 show how we can call these functions to produce the same effect as the segment of code P1 in Figure 11.1(a). As we can see in this example, functions can be used just to make the PHP code better structured and easier to follow.

A second example is shown in Figure 11.5. Here we are using the `$teaching` array introduced in Figure 11.3. The function `course_instructor()` in lines 0 to 8 in Figure 11.5 has two arguments: `$course` (a string holding a course name) and `$teaching_assignments` (an associative array holding course assignments, similar to the `$teaching` array shown in Figure 11.3). The function finds the name of the instructor who teaches a particular course. Lines 9 to 14 in Figure 11.5 show how this function may be used.

The function call in line 11 would return the string: *Smith is teaching Database*, because the array entry with the key 'Database' has the value 'Smith' for instructor. On the other hand, the function call on line 13 would return the string: *there is no Computer Architecture course* because there is no entry in the array with the key

Figure 11.4

Rewriting program segment P1 as P1' using functions.

```
//Program Segment P1':
0) function display_welcome() {
1)     print("Welcome, ") ;
2)     print($_POST['user_name']);
3) }
4)
5) function display_empty_form(); {
6) print <<<_HTML_
7) <FORM method="post" action="$_SERVER['PHP_SELF']">
8) Enter your name: <INPUT type="text" name="user_name">
9) <BR/>
10) <INPUT type="submit" value="Submit name">
11) </FORM>
12) _HTML_;
13) }
14) if ($_POST['user_name']) {
15)     display_welcome();
16) }
17) else {
18)     display_empty_form();
19) }
```

Figure 11.5

Illustrating a function with arguments and return value.

```
0) function course_instructor ($course, $teaching_assignments) {
1)     if (array_key_exists($course, $teaching_assignments)) {
2)         $instructor = $teaching_assignments[$course];
3)         RETURN "$instructor is teaching $course";
4)     }
5)     else {
6)         RETURN "there is no $course course";
7)     }
8) }
9) $teaching = array('Database' => 'Smith', 'OS' => 'Carrick',
                    'Graphics' => 'Kam');
10) $teaching['Graphics'] = 'Benson'; $teaching['Data Mining'] = 'Li';
11) $x = course_instructor('Database', $teaching);
12) print($x);
13) $x = course_instructor('Computer Architecture', $teaching);
14) print($x);
```

'Computer Architecture'. A few comments about this example and about PHP functions in general:

- The built-in PHP array function `array_key_exists($k, $a)` returns true if the value in variable `$k` exists as a key in the associative array in the variable `$a`. In our example, it checks whether the `$course` value provided exists as a key in the array `$teaching_assignments` (line 1 in Figure 11.5).
- Function arguments are passed by value. Hence, in this example, the calls in lines 11 and 13 could not change the array `$teaching` provided as argument for the call. The values provided in the arguments are passed (copied) to the function arguments when the function is called.
- Return values of a function are placed after the `RETURN` keyword. A function can return any type. In this example, it returns a string type. Two different strings can be returned in our example, depending on whether the `$course` key value provided exists in the array or not.
- Scope rules for variable names apply as in other programming languages. Global variables outside of the function cannot be used unless they are referred to using the built-in PHP array `$GLOBALS`. Basically, `$GLOBALS['abc']` will access the value in a global variable `$abc` defined outside the function. Otherwise, variables appearing inside a function are local even if there is a global variable with the same name.

The previous discussion gives a brief overview of PHP functions. Many details are not discussed since it is not our goal to present PHP in detail.

11.2.4 PHP Server Variables and Forms

There are a number of built-in entries in a PHP auto-global built-in array variable called `$_SERVER` that can provide the programmer with useful information about the server where the PHP interpreter is running, as well as other information. These may be needed when constructing the text in an HTML document (for example, see line 7 in Figure 11.4). Here are some of these entries:

1. **`$_SERVER['SERVER_NAME']`**. This provides the Web site name or the Uniform Resource Locator (URL) of the server computer where the PHP interpreter is running. For example, if the PHP interpreter is running on the Web site `http://www.uta.edu`, then this string would be the value in `$_SERVER['SERVER_NAME']`.
2. **`$_SERVER['REMOTE_ADDRESS']`**. This is the IP (Internet Protocol) address of the client user computer that is accessing the server; for example, `129.107.61.8`.
3. **`$_SERVER['REMOTE_HOST']`**. This is the Web site name (URL) of the client user computer; for example, `abc.uta.edu`. In this case, the server will need to translate the name into an IP address to access the client.
4. **`$_SERVER['PATH_INFO']`**. This is the part of the URL address that comes after a backslash (`/`) at the end of the URL.

5. `$_SERVER['QUERY_STRING']`. This provides the string that holds parameters in a URL after a question mark (?) at the end of the URL. This can hold search parameters, for example.
6. `$_SERVER['DOCUMENT_ROOT']`. This is the root directory that holds the files on the Web server that are accessible to client users.

These and other entries in the `$_SERVER` array are usually needed when creating the HTML file to be sent to the client for display.

Another important PHP auto-global built-in array variable is called `$_POST`. This provides the programmer with input values submitted by the user through HTML forms specified in the HTML `<INPUT>` tag and other similar tags. For example, in Figure 11.4, line 14, the variable `$_POST['user_name']` provides the programmer with the value typed in by the user in the HTML form specified via the `<INPUT>` tag on line 8 in Figure 11.4. The keys to this array are the names of the various input parameters provided via the form, for example by using the name attribute of the HTML `<INPUT>` tag as on line 8. When users enter data through forms, the data values are stored in this array.

11.3 Overview of PHP Database Programming

There are various techniques for accessing a database through a programming language. We discussed some of the techniques in Chapter 10, in the overviews of how to access an SQL database using the C and Java programming languages. In particular, we discussed embedded SQL, JDBC, SQL/CLI (similar to ODBC), and SQLJ. In this section we give an overview of how to access the database using the script language PHP, which is suitable for creating Web interfaces for searching and updating databases, as well as dynamic Web pages.

There is a PHP database access function library that is part of PHP Extension and Application Repository (PEAR), which is a collection of several libraries of functions for enhancing PHP. The PEAR DB library provides functions for database access. Many database systems can be accessed from this library, including Oracle, MySQL, SQLite, and Microsoft SQLServer, among others.

We will discuss several functions that are part of PEAR DB in the context of some examples. Section 11.3.1 shows how to connect to a database using PHP. Section 11.3.2 discusses how data collected from HTML forms can be used to insert a new record in a database table. Section 11.3.3 shows how retrieval queries can be executed and have their results displayed within a dynamic Web page.

11.3.1 Connecting to a Database

To use the database functions in a PHP program, the PEAR DB library module called `DB.php` must be loaded. In Figure 11.6, this is done in line 0 of the example. The DB library functions can now be accessed using `DB::<function_name>`. The function for connecting to a database is called `DB::connect('string')`,

```

0) require 'DB.php';
1) $d = DB::connect('oci8://acctl:pass12@www.host.com/db1');
2) if (DB::isError($d)) { die("cannot connect - " . $d->getMessage());}
   ...
3) $q = $d->query("CREATE TABLE EMPLOYEE
4)   (Emp_id INT,
5)   Name VARCHAR(15),
6)   Job VARCHAR(10),
7)   Dno INT);" );
8) if (DB::isError($q)) { die("table creation not successful - " .
   $q->getMessage()); }
   ...
9) $d->setErrorHandler(PEAR_ERROR_DIE);
   ...
10) $eid = $d->nextID('EMPLOYEE');
11) $q = $d->query("INSERT INTO EMPLOYEE VALUES
12)   ($eid, $_POST['emp_name'], $_POST['emp_job'], $_POST['emp_dno'])" );
   ...
13) $eid = $d->nextID('EMPLOYEE');
14) $q = $d->query('INSERT INTO EMPLOYEE VALUES (?, ?, ?, ?)',
15) array($eid, $_POST['emp_name'], $_POST['emp_job'], $_POST['emp_dno']) );

```

Figure 11.6

Connecting to a database, creating a table, and inserting a record.

where the string argument specifies the database information. The format for 'string' is:

```
<DBMS software>://<user account>:<password>@<database server>
```

In Figure 11.6, line 1 connects to the database that is stored using Oracle (specified via the string `oci8`). The `<DBMS software>` portion of the 'string' specifies the particular DBMS software package being connected to. Some of the DBMS software packages that are accessible through PEAR DB are:

- **MySQL.** Specified as `mysql` for earlier versions and `mysqli` for later versions starting with version 4.1.2.
- **Oracle.** Specified as `oci8i` for versions 7, 8, and 9. This is used in line 1 of Figure 11.6.
- **SQLite.** Specified as `sqlite`.
- **Microsoft SQL Server.** Specified as `mssql`.
- **Mini SQL.** Specified as `msql`.
- **Informix.** Specified as `ifx`.
- **Sybase.** Specified as `sybase`.
- **Any ODBC-compliant system.** Specified as `odbc`.

The above is not a comprehensive list.

Following the `<DB software>` in the string argument passed to `DB::connect` is the separator `://` followed by the user account name `<user account>` followed by the separator `:` and the account password `<password>`. These are followed by the separator `@` and the server name and directory `<database server>` where the database is stored.

In line 1 of Figure 11.6, the user is connecting to the server at `www.host.com/db1` using the account name `acct1` and password `pass12` stored under the Oracle DBMS `oci8`. The whole string is passed using `DB::connect`. The connection information is kept in the database connection variable `$d`, which is used whenever an operation to this particular database is applied.

Checking for errors. Line 2 in Figure 11.6 shows how to check whether the connection to the database was established successfully or not. PEAR DB has a function `DB::isError`, which can determine whether any database access operation was successful or not. The argument to this function is the database connection variable (`$d` in this example). In general, the PHP programmer can check after every database call to determine whether the last database operation was successful or not, and terminate the program (using the *die* function) if it was not successful. An error message is also returned from the database via the operation `$d->get_message()`. This can also be displayed as shown in line 2 of Figure 11.6.

Submitting queries and other SQL statements. In general, most SQL commands can be sent to the database once a connection is established by using the *query* function. The function `$d->query` takes an SQL command as its string argument and sends it to the database server for execution. In Figure 11.6, lines 3 to 7 send a `CREATE TABLE` command to create a table called `EMPLOYEE` with four attributes. Whenever a query or SQL statement is executed, the result of the query is assigned to a query variable, which is called `$q` in our example. Line 8 checks whether the query was executed successfully or not.

The PHP PEAR DB library offers an alternative to having to check for errors after every database command. The function

```
$d->setErrorHandling(PEAR_ERROR_DIE)
```

will terminate the program and print the default error messages if any subsequent errors occur when accessing the database through connection `$d` (see line 9 in Figure 11.6).

11.3.2 Collecting Data from Forms and Inserting Records

It is common in database applications to collect information through HTML or other types of Web forms. For example, when purchasing an airline ticket or applying for a credit card, the user has to enter personal information such as name, address, and phone number. This information is typically collected and stored in a database record on a database server.

Lines 10 through 12 in Figure 11.6 illustrate how this may be done. In this example, we omitted the code for creating the form and collecting the data, which can be a variation of the example in Figure 11.1. We assume that the user entered valid values in the input parameters called `emp_name`, `emp_job`, and `emp_dno`. These would be accessible via the PHP auto-global array `$_POST` as discussed at the end of Section 11.2.4.

In the SQL `INSERT` command shown on lines 11 and 12 in Figure 11.6, the array entries `$POST['emp_name']`, `$POST['emp_job']`, and `$POST['emp_dno']` will hold the values collected from the user through the input form of HTML. These are then inserted as a new employee record in the `EMPLOYEE` table.

This example also illustrates another feature of PEAR DB. It is common in some applications to create a unique record identifier for each new record inserted into the database.¹

PHP has a function `$d->nextID` to create a sequence of unique values for a particular table. In our example, the field `Emp_id` of the `EMPLOYEE` table (see Figure 11.6, line 4) is created for this purpose. Line 10 shows how to retrieve the next unique value in the sequence for the `EMPLOYEE` table and insert it as part of the new record in lines 11 and 12.

The code for insert in lines 10 to 12 in Figure 11.6 may allow malicious strings to be entered that can alter the `INSERT` command. A safer way to do inserts and other queries is through the use of **placeholders** (specified by the `?` symbol). An example is illustrated in lines 13 to 15, where another record is to be inserted. In this form of the `$d->query()` function, there are two arguments. The first argument is the SQL statement, with one or more `?` symbols (placeholders). The second argument is an array, whose element values will be used to replace the placeholders in the order they are specified (see lines 13 to 15 in Figure 11.6).

11.3.3 Retrieval Queries from Database Tables

We now give three examples of retrieval queries through PHP, shown in Figure 11.7. The first few lines 0 to 3 establish a database connection `$d` and set the error handling to the default, as we discussed in the previous section. The first query (lines 4 to 7) retrieves the name and department number of all employee records. The query variable `$q` is used to refer to the **query result**. A while-loop to go over each row in the result is shown in lines 5 to 7. The function `$q->fetchRow()` in line 5 serves to retrieve the next record in the query result and to control the loop. The looping starts at the first record.

The second query example is shown in lines 8 to 13 and illustrates a dynamic query. In this query, the conditions for selection of rows are based on values input by the user. Here we want to retrieve the names of employees who have a

¹This would be similar to the system-generated OID discussed in Chapter 12 for object and object-relational database systems.

```

0) require 'DB.php';
1) $d = DB::connect('oci8://acctl:pass12@www.host.com/dbname');
2) if (DB::isError($d)) { die("cannot connect - " . $d->getMessage()); }
3) $d->setErrorHandler(PEAR_ERROR_DIE);
   ...
4) $q = $d->query('SELECT Name, Dno FROM EMPLOYEE');
5) while ($r = $q->fetchRow()) {
6)     print "employee $r[0] works for department $r[1] \n" ;
7) }
   ...
8) $q = $d->query('SELECT Name FROM EMPLOYEE WHERE Job = ? AND Dno = ?',
9)     array($_POST['emp_job'], $_POST['emp_dno']) );
10) print "employees in dept $_POST['emp_dno'] whose job is
    $_POST['emp_job']: \n"
11) while ($r = $q->fetchRow()) {
12)     print "employee $r[0] \n" ;
13) }
   ...
14) $allresult = $d->getAll('SELECT Name, Job, Dno FROM EMPLOYEE');
15) foreach ($allresult as $r) {
16)     print "employee $r[0] has job $r[1] and works for department $r[2] \n" ;
17) }
   ...

```

Figure 11.7

Illustrating database retrieval queries.

specific job and work for a particular department. The particular job and department number are entered through a form in the array variables `$POST['emp_job']` and `$POST['emp_dno']`. If the user had entered 'Engineer' for the job and 5 for the department number, the query would select the names of all engineers who worked in department 5. As we can see, this is a dynamic query whose results differ depending on the choices that the user enters as input. We used two ? placeholders in this example, as discussed at the end of Section 11.3.2.

The last query (lines 14 to 17) shows an alternative way of specifying a query and looping over its rows. In this example, the function `$d=>getAll` holds all the records in a query result in a single variable, called `$allresult`. To loop over the individual records, a `foreach` loop can be used, with the row variable `$r` iterating over each row in `$allresult`.²

As we can see, PHP is suited for both database access and creating dynamic Web pages.

²The `$r` variable is similar to the cursors and iterator variables discussed in Chapters 10 and 12.

11.4 Brief Overview of Java Technologies for Database Web Programming

The parts of the PHP scripting language that we discussed run on the application server and serve as a conduit that collects client user input through forms, formulates database queries and submits them to the database server, and then creates dynamic HTML Web pages to display query results. The Java environment has components that run on the server and other components that can run on the client machine. It also has standards for exchanging data objects. We briefly discuss some of these components here that are related to Web and database access. We already discussed JDBC and SQLJ in some detail in Chapter 10.

Java Servlets. Servlets are Java objects that can reside on the Web server machine and manage interactions with the client. They can store information that was submitted by the client during a session, so that this information can be used to generate database queries. Servlet objects can also store query results so that parts of these results can be formatted as HTML and sent to the client for display. The servlet object can maintain all the information produced during a particular client interaction until the client session is terminated.

Java Server Pages (JSP). This allows scripting at the server to produce dynamic Web pages to be sent at the client in a manner somewhat similar to PHP. However, it is associated with the Java language and the scripting can be combined with Java code.

JavaScript. JavaScript is a scripting language that is different from the Java programming language and was developed separately. It is widely used in Web applications, and it can run on the client computer or on the server.

Java Script Object Notation (JSON). This is a text-based representation of data objects, so that data can be formatted in JSON and exchanged between clients and servers over the Web in text format. It can be considered as an alternative to XML (see Chapter 13) and represents objects using attribute-value pairs. JSON has also been adopted as the data model by some newer database systems known as NOSQL systems, such as MongoDB (see Chapter 24).

11.5 Summary

In this chapter, we gave an overview of how to convert some structured data from databases into elements to be entered or displayed on a Web page. We focused on the PHP scripting language, which is becoming very popular for Web database programming. Section 11.1 presented some PHP basics for Web programming through a simple example. Section 11.2 gave some of the basics of the PHP language, including its array and string data types that are used extensively. Section 11.3 presented an overview of how PHP can be used to specify various types of database commands, including creating tables, inserting new records, and retrieving database records. PHP runs at the server computer in comparison to some other scripting languages that run on the client computer. Section 11.4 introduced some of the technologies associated with Java that can be used in similar contexts.

We gave only a very basic introduction to PHP. There are many books as well as many Web sites devoted to introductory and advanced PHP programming. Many libraries of functions also exist for PHP, as it is an open source product.

Review Questions

- 11.1. Why are scripting languages popular for programming Web applications? Where in the three-tier architecture does a PHP program execute? Where does a JavaScript program execute?
- 11.2. What type of programming language is PHP?
- 11.3. Discuss the different ways of specifying strings in PHP.
- 11.4. Discuss the different types of arrays in PHP.
- 11.5. What are PHP auto-global variables? Give some examples of PHP auto-global arrays, and discuss how each is typically used.
- 11.6. What is PEAR? What is PEAR DB?
- 11.7. Discuss the main functions for accessing a database in PEAR DB, and how each is used.
- 11.8. Discuss the different ways for looping over a query result in PHP.
- 11.9. What are placeholders? How are they used in PHP database programming?

Exercises

- 11.10. Consider the LIBRARY database schema shown in Figure 4.6. Write PHP code to create the tables of this schema.
- 11.11. Write a PHP program that creates Web forms for entering the information about a new BORROWER entity. Repeat for a new BOOK entity.
- 11.12. Write PHP Web interfaces for the queries specified in Exercise 6.18.

Selected Bibliography

There are many sources for PHP programming, both in print and on the Web. We give two books as examples. A very good introduction to PHP is given in Sklar (2005). For advanced Web site development, the book by Schlossnagle (2005) provides many detailed examples. Nixon (2014) has a popular book on web programming that covers PHP, Javascript, JQuery, CSS and HTML5.