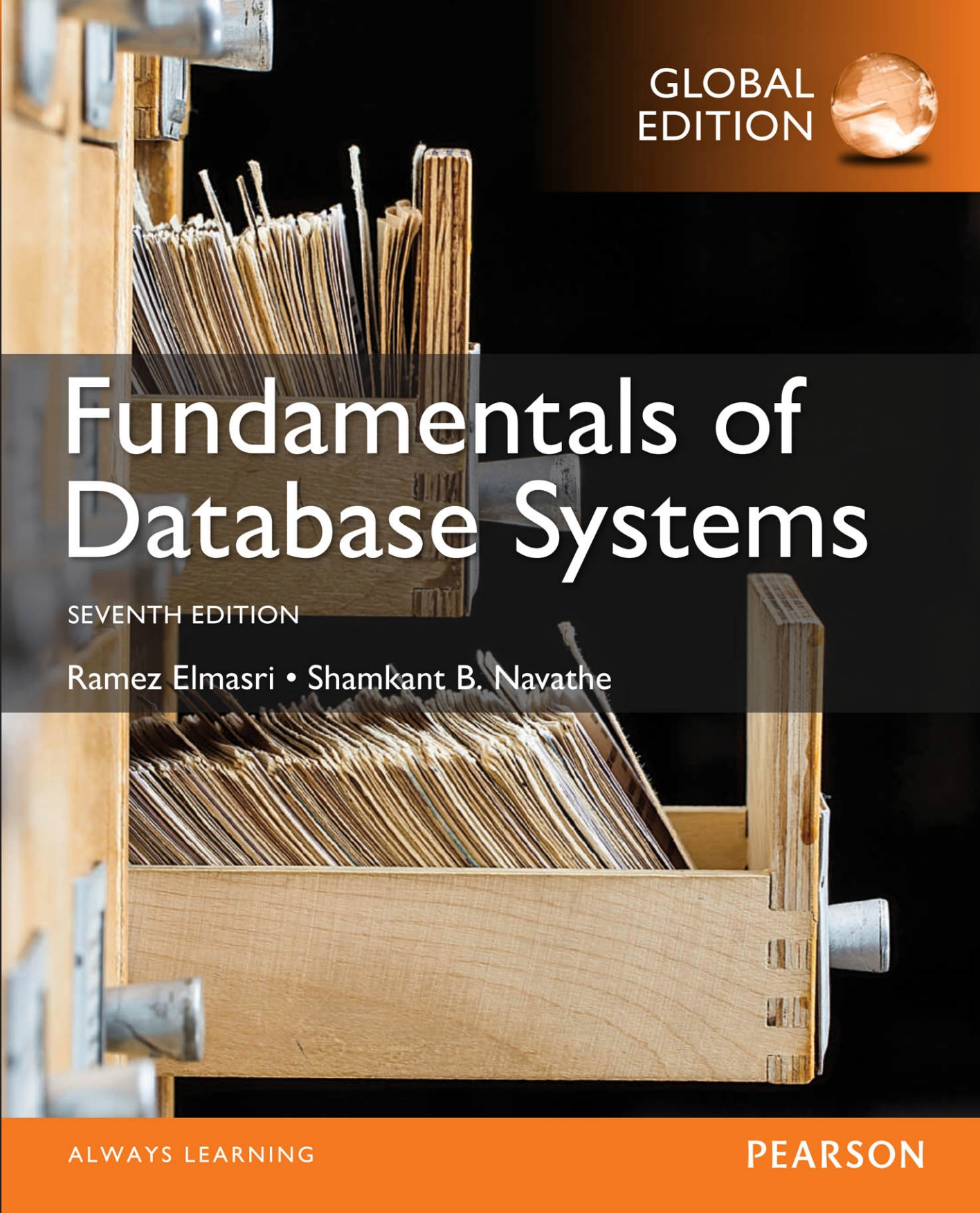
GLOBAL
EDITION
Fundamentals of
Database Systems
SEVENTH EDITION
Ramez Elmasri • Shamkant B. Navathe
ALWAYS LEARNING
PEARSON

FUNDAMENTALS OF

Database Systems

SEVENTH EDITION

GLOBAL EDITION

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) $d = DB::connect('oci8://acct1:pass12@www.host.com/db1');
 2) if (DB::isError($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->setErrorHandling(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 `oc8i` 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

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