Ordering Information:

Perl How to Program
The Complete Perl Training Course

- View the complete **Table of Contents**
- Download the **Code Examples**

To view all the Deitel products and services available, visit the Deitel Kiosk on InformIT at **www.informIT.com/deitel**.

To follow the Deitel publishing program, sign-up now for the *DEITEL™ BUZZ ON-LINE* e-mail newsletter at **www.deitel.com/newsletter/subscribeinformIT.html**.

To learn more about our **Perl programming courses** or any other Deitel instructor-led corporate training courses that can be delivered at your location, visit **www.deitel.com/training**, contact our Director of Corporate Training Programs at (978) 461-5880 or e-mail: **christi.kelsey@deitel.com**.

Note from the Authors: This article is an excerpt from Chapter 15, Sections 15.5 and 15.6 of *Perl How to Program*. This article introduces manipulating databases using the Perl programming language. Readers should be familiar with Perl programming and SQL. In the article we provide an example of using Perl to select information from a database created in MS Access. The code examples included in this article show readers programming examples using the DEITEL™ signature LIVE-CODE™ Approach, which presents all concepts in the context of complete working programs followed by the screen shots of the actual inputs and outputs.

15.5 Introduction to DBI

Many of today's Web sites provide their users with services such as the ability to purchase items online, store files remotely and access e-mail. These Web sites require databases to authenticate the users that access the sites and to maintain data about those users. Databases have become a crucial part of *distributed applications*. A distributed application is a program that divides the work needed to be done across multiple computer systems. For instance, one computer might be responsible for managing a Web site and another for a database management system. A distributed application uses both computers to perform the task of retrieving a result set from a database and displaying those results on another computer—typically called a client.

The Perl *Database Interface (DBI)* provides a means of accessing relational databases from Perl programs. There are many different implementations of relational databases (e.g., MySQL, Microsoft Access, Oracle, etc.). A software program—called a *driver*—helps programs access a database. Each database implementation requires its own driver and each driver can have different syntax for its use in a program. To make using all these different databases easier, an *interface* was created to provide uniform access to all databases. This interface is known as DBI. The database vendors create drivers for their databases that can receive interactions through DBI and process those interactions in a database-specific manner. DBI is database independent, so it allows for easy migration from one DBMS to another. While DBI is not the only interface available for database connectivity in Perl, it is the most widely used.

DBI uses an object-oriented interface. The DBI objects are known as *handles*. There are three different handle types—*driver handles*, *database handles* and *statement handles*. Driver handles encapsulate the driver for the database; they are rarely used in a script. Database handles encapsulate a specific connection to a database. They can be used to send SQL statements to a database. Statement handles encapsulate specific SQL statements and the results returned from them. Any number of database handles can be created with a driver handle and any number of statement handles can be created with a database handle.

Testing and Debugging Tip 15.1

DBI functions do not use the standard predefined Perl error variables. Generally, when DBI function calls fail they return **`undef`** *and store the error string in* **`$DBI::errstr`**. *The error number is stored in* **`$DBI::err`**. *Each handle also stores its error information, which can be accessed through method calls* **`errstr`** *and* **`err`** *on the handle.*

15.6 Working with DBI

In this article, we demonstrate a basic example using DBI. We will be accessing an ODBC database, specifically, a Microsoft Access database, **`Employee.mdb`**, which contains data about various employees at a company. Note that this database needs to be registered as a valid ODBC source first. Also, the **`DBI.pm`** module and the ODBC driver must be installed before the programs that interact with Microsoft Access databases can be executed. The Perl resources posted at our Web site, **`www.deitel.com`**, include step-by-step instructions on registering a Microsoft Access database as an ODBC data source on a Windows system, installing the **`DBI.pm`** module and installing the ODBC database driver

(**DBD::ODBC**). You can find the resources for all our books by clicking the **Downloads/Resources** link.

In this DBI program (Fig. 15.18), the contents of the employee database are output. The **use DBI** statement on line 7 loads the DBI module and line 8 loads the database driver, **DBD::ODBC**.

To create a connection to a database, we pass a *data source name* (*DSN*) to method ***connect*** (line 10). A data source name tells **connect** where to find the database and is constructed in the following format for ODBC databases:

interface name:*database driver*:*data source name*

```
1  #!/usr/bin/perl
2  # Fig. 15.18: fig15_18.pl
3  # Program to query a database and display the contents in a table
4
5  use warnings;
6  use strict;
7  use DBI;
8  use DBD::ODBC;
9
10 my $dbh = DBI->connect( "DBI:ODBC:employeeDB", "", "" ) or
11    die( "Could not make connection to database: $DBI::errstr" );
12
13 my $sth = $dbh->prepare( q{ SELECT * FROM employee } ) or
14    die( "Cannot prepare statement: ", $dbh->errstr(), "\n" );
15
16 $sth->execute() or
17    die( "Cannot execute statement: ", $sth->errstr(), "\n" );
18
19 my @array;
20
21 while ( @array = $sth->fetchrow_array() ) {
22    write();
23 }
24
25 # Check to see if fetch terminated early
26 warn( $DBI::errstr ) if $DBI::err;
27 $dbh->disconnect();
28 $sth->finish();
29
30 format STDOUT =
31 @<<<<<<@<<<<<<<<<<@<<<<<<<<<<@<<<<<@<<<<<<<<<<<
32 $array[ 0 ], $array[ 1 ], $array[ 2 ], $array[ 3 ], $array[ 4 ]
33 .
```

```
0004   Michael   Black      1965  222-44-8888
0001   Jim       Blue       1943  999-85-3698
0002   Kate      Green      1977  111-21-7454
0003   Wendy     White      1959  000-84-3196
```

Fig. 15.18 Using DBI to query a database.

In this program, the *interface name* is **DBI**, the *database driver* is **ODBC** and the *data source name* is **employeeDB**. Method **connect** returns a database handle that is assigned to **$dbh**. The second and third arguments to **connect** represent the username and password. We use empty strings here, because the database does not have a username and password. As with every DBI method call, the remainder of the statement (line 11) determines if the connection was successful and prints an error message and terminates the program if not.

Line 13 creates a *statement handle* by calling the database handle's ***prepare*** method. This method prepares the database driver for a statement. which can be executed multiple times later. The SQL query is passed to the **prepare** statement as a string. The SQL in this case is selecting all the fields from the table. The statement handle returned by method **prepare** is assigned to **$sth**.

After a statement has been prepared, and before the results can be processed, the statement must be executed. This is done by calling the statement handle's **execute** method (line16). The result set generated by the query is stored with the statement handle. Each row of the result set is retrieved from the object and placed into an array by calling method **fetchrow_array** (line 21). We use function **write** to print each row using the format defined on lines 30–33. Method **fetchrow_array** returns *false* after all the rows have been read, thus ending the **while** loop. Other functions for extracting the results of a query are shown in Fig. 15.19.

Because **fetchrow_array** returns false when there are no more rows and when there is an error fetching data, we need to check for an error after the end of the loop (line 26). If an error occurs during the fetching of the data, **$DBI::err** is defined. So, we warn the user that an error did occur on line 26. Once we are finished with a database connection, we close it by using method ***disconnect*** (line 27). If you do not need to fetch all the data from a statement handle, you should call method **finish** to indicate when you are done. When all the data of a result set is fetched, method **finish** is called automatically.

Good Programming Practice 15.1

*Perl will close statement and database handles for you when they go out of scope. It is, however, best to close them explicitly by using the **finish** or **disconnect** method when you are done using them.*

Function Name	Return Type	Description
fetchrow_array	array	Returns a single row in an array.
fetchrow_arrayref	array ref	Returns a single row in an array reference.
fetchrow_hashref	hash ref	Returns a single row in a hash reference with fieldname value pairs.
fetchall_arrayref	array ref	Returns the whole result set in a reference to an array. The array consists of references to arrays that hold the rows of data.

Fig. 15.19 Functions for extracting the results of a query.