CHAPTER 3

Introductory Example (An Opinion Poll)

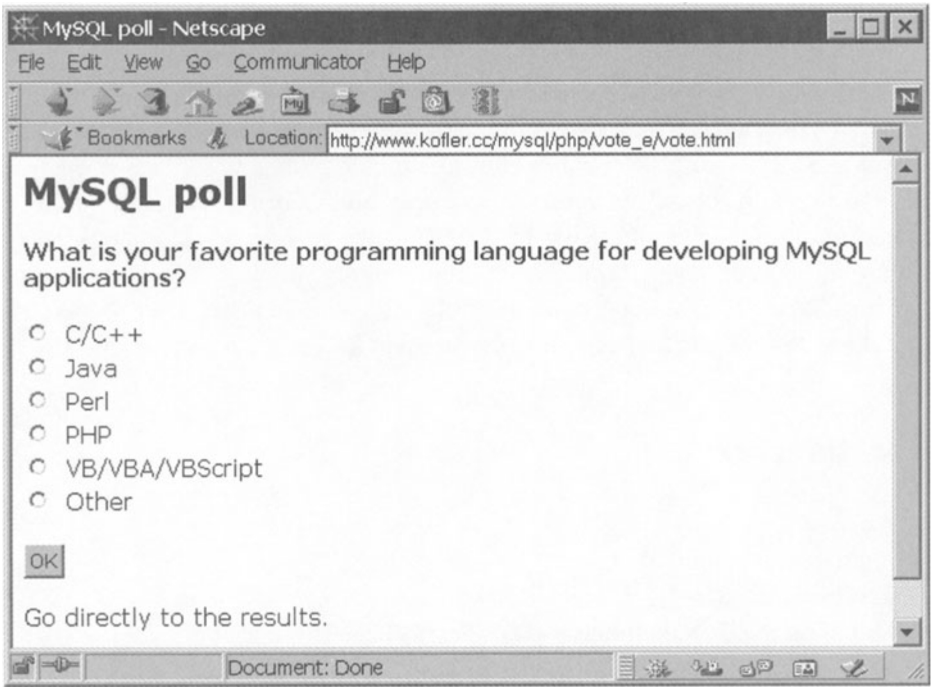
THE BEST WAY OF BECOMING familiar with a new database or development system is to work through a full-fledged example. Thus in this chapter our goal is to create a web site for the purpose of conducting an opinion poll.

To a certain extent this is a trivial example, and its results certainly could be accomplished without the use of MySQL. However, it brings into focus the interplay between MySQL and a script programming language (for this example we have used PHP). Moreover, our example casts light on the entire process of database design from first beginnings right up to the completed application.

Chapter Overview

Overview

Our opinion poll consists of two pages. The file `vote.html` contains a simple questionnaire with a single question: What is your favorite programming language for developing MySQL applications? The question is followed by a selection of choices for a response (see Figure 3-1). After one of the options is selected and OK is clicked, the result page `results.php` is displayed (see Figure 3-2).


MySQL poll - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: http://www.kofler.cc/mysql/php/vote_e/vote.html
OK
Document: Done

MySQL poll

What is your favorite programming language for developing MySQL applications?

- C/C++
- Java
- Perl
- PHP
- VB/VBA/VBScript
- Other

Go directly to the results.

Figure 3-1. The questionnaire

Assumptions

You can try out this example right now. Just wend your way to my web site (`www.kofler.cc`). However, from a pedagogical point of view it would be better for you to attempt to recreate this example for yourself. For this you will need a test environment consisting of Apache/MySQL/PHP that permits you the following:

- creation of a new MySQL database (that is, you need sufficient privileges to be able to execute *CREATE DATABASE*)
- moving files into a directory on the web server
- executing PHP script files

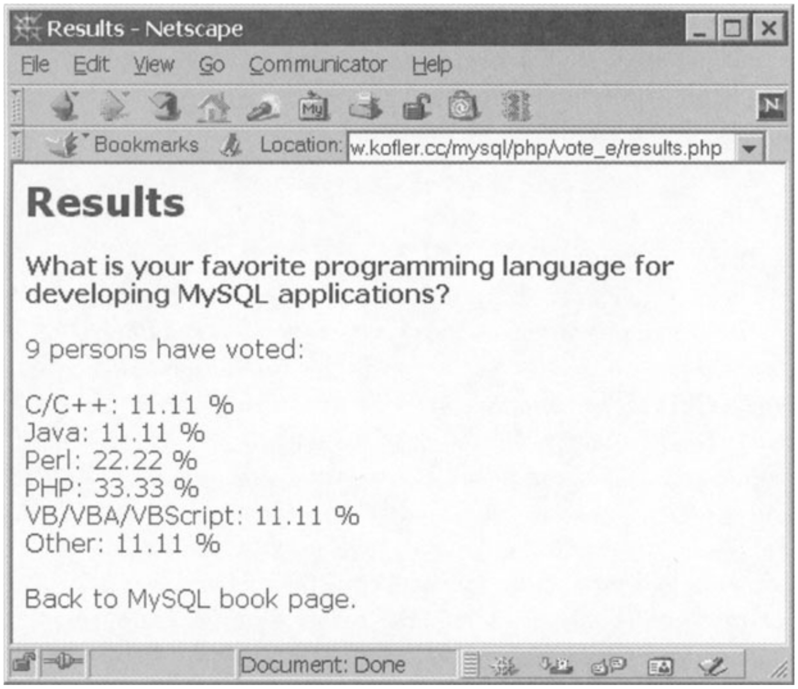
Information on setting up such a test environment on a local computer can be found in the previous chapter.

A complete understanding of our example requires some basic knowledge of databases. If you have never had much, if anything, to do with databases and as a consequence run into difficulties in understanding what is going on, please do not despair. Chapter 5 provides a complete introduction to relational database systems, while Chapter 6 explains in great detail how to use the database query language SQL.

In this example the programming language PHP will be used. The code is rather straightforward; that is, you should have no difficulty in understanding it even you don't know a word of PHP. However, you should know in general how embedded script languages function in HTML. (The Active Server pages developed by Microsoft are based on the same idea.)

Database Development

To save the results of our questionnaire with MySQL you must first set up a database and then place a table in that database. (Every MySQL database consists of tables. A particular feature of this example is that only a single table is required.


Results - Netscape
File Edit View Go Communicator Help
Bookmarks
Location: w.kofler.cc/mysql/php/vote_e/results.php
Document: Done

Results

What is your favorite programming language for developing MySQL applications?

9 persons have voted:

```
C/C++: 11.11 %
Java: 11.11 %
Perl: 22.22 %
PHP: 33.33 %
VB/VBA/VBScript: 11.11 %
Other: 11.11 %
```

Back to MySQL book page.

Figure 3-2. Results of the survey

As a rule, that is, when the requirements of the project are more complex, several linked tables will be used.)

Executing `mysql`

Both operations—generating a database and creating a new table—require that you communicate with MySQL. Under Unix/Linux you execute the command `mysql`. Under Windows you search in Explorer for the program `mysql.exe` and launch it with a double click. (The program should reside in the `bin` directory of the MySQL installation directory, that is, in `C:\Programs\MySQL\bin`.)

REMARK *If* `mysql` *immediately terminates, possibly with an error message such as* Access denied for user xy, *then either access to MySQL is completely denied to user xy, or it is protected by a password. In either case you must invoke* `mysql` *with the options* `-u name` *and* `-p`*:*

```
> mysql -u username -p
Enter password: xxx
```

Furthermore, `mysql` *has dozens of other options. The most important of these are described in Chapter 4, while all of them are collected in Chapter 14. Background information on the MySQL security system (access protection, passwords, privileges) can be found in Chapter 7.*

REMARK *The distinction between MySQL and* `mysql` *is somewhat confusing. MySQL denotes the database server, which normally is launched automatically at system startup. The server runs continuously in the background. (This chapter assumes that the server is up and running.) The program name of the server is, depending on the operating system,* `mysqld` *(Unix/Linux) or* `mysqld.exe` *or* `mysqld-nt.exe` *(Windows).*

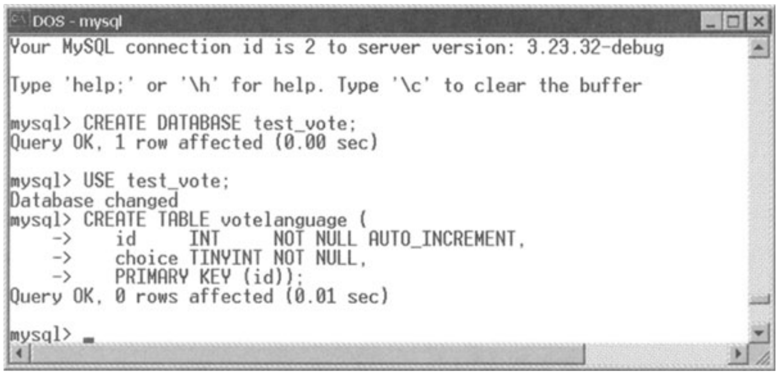
In contrast to these we have the auxiliary programs `mysql` *and* `mysql.exe`*. These programs come into play when administration or maintenance are to be carried out interactively. The program* `mysql` *has the task of transmitting interactive commands to the server and displaying the results of these commands on the monitor. The official name of* `mysql` *is MySQL Monitor, but the functionality and mode of action are more reminiscent of a command line interpreter.*

POINTER *Some alternatives to* `mysql`*, such as the more convenient HTML interface phpMyAdmin, will be introduced in Chapter 4.*

In `mysql` you can now input commands that will be transmitted to the database server (see Figure 3-3). To test whether a connection can even be made, execute the command *STATUS*. The result should be the display of various pieces of status information about the database (such as the version number).

```
mysql> STATUS;
mysql Ver 11.12 Distrib 3.23.32, for Win95/Win98 (i32)
  Connection id:          3
  Current database:
  Current user:           ODBC@localhost
  Server version:         3.23.32-debug
  Protocol version:       10
  Connection:             localhost via TCP/IP
  Client characterset:    latin1
  Server characterset:    latin1
  TCP port:               3306
  Uptime:                 1 min 5 sec

Threads: 2 Questions: 8 Slow queries: 0 Opens: 7
Flush tables: 1 Open tables: 2 Queries per second avg: 0.123
Memory in use: 8332K Max memory used: 8348K
```



```
Your MySQL connection id is 2 to server version: 3.23.32-debug

Type 'help;' or '\h' for help. Type '\c' to clear the buffer

mysql> CREATE DATABASE test_vote;
Query OK, 1 row affected (0.00 sec)

mysql> USE test_vote;
Database changed
mysql> CREATE TABLE votelanguage (
    ->     id     INT     NOT NULL AUTO_INCREMENT,
    ->     choice TINYINT NOT NULL,
    ->     PRIMARY KEY (id));
Query OK, 0 rows affected (0.01 sec)

mysql> _
```

Figure 3-3. The MySQL monitor

POINTER *If problems arise in starting up* `mysql` *or in executing* `status`*, the most probable cause is that the database server hasn't even been started or that access has been denied to you. More information on installation can be found in Chapter 2, while information on access and security can be found in Chapter 7.*

Setting Up the Database

To set up the new database *test_vote,* in `mysql` execute the command *CREATE DATABASE.* Note that that you must end the command with a semicolon. In the following two lines your input appears in boldface.

```
mysql> CREATE DATABASE test_vote;
Query OK, 1 row affected (0.01 sec)
```

The reply delivered by `mysql` may look a bit weird. The output `1 row affected` indicates that in the list of all databases, which internally, of course, is in the form of a MySQL table, one row was changed. What is important here is only that the *CREATE DATABASE* command was executed correctly.

REMARK *The database name* test_vote *was not chosen quite arbitrarily. In the default setting of MySQL access privileges, every user is permitted to create databases on a local computer that begin with the word "test." In particular, when you yourself are not the MySQL administrator (but rely on the help of a system administrator), a name of the form* test_xy *can save any number of e-mails or telephone calls.*

The drawback of the name test_xy *is that every user of the local computer can edit, or even delete, the database. This is no problem for this introductory example, but in the case of a real-life application you probably will want to give your database a bit more security. Necessary information on this can be found in Chapter 7.*

Creating Tables

The database *test_vote* has been created, but it is not yet possible to store any information. For this you need tables. To create a new table within the database *test_vote* use the command *CREATE TABLE.*

Before you execute this command, however, you must specify the database into which the table is to be placed. The requisite command for this is *USE.* It determines the default database to which further commands are to be applied. (MySQL is managing other databases besides the newly created database *test_vote.*)

```
mysql> USE test_vote;
Database changed
mysql> CREATE TABLE votelanguage (
    -> id INT NOT NULL AUTO_INCREMENT,
    -> choice TINYINT NOT NULL,
    -> PRIMARY KEY (id));
Query OK, 0 rows affected (0.01 sec)
```

The *CREATE TABLE* command may seem a bit strange at first. But just go ahead and input the boldface commands listed above line for line. (You can terminate each command with `Return`. The semicolon indicates to `mysql` that the end of a command has been reached.)

TIP *If you should make a typing error during the input of a command, MySQL will usually inform you of this fact with an error message. You must now repeat the entire command. Using the cursor keys ↑ and ↓ you can recall and correct previously input lines.*

If MySQL has accepted CREATE TABLE *in spite of a typographical error (because the command, though semantically not what you had in mind, is nonetheless syntactically correct), you can delete the incorrectly defined table with* DROP TABLE votelanguage;. *That accomplished, you can repeat the command* CREATE TABLE.

Now let us explain what is actually going on. With *CREATE TABLE* you have brought into being a table with two columns, *id* and *choice.* Once the table is filled with data (namely, the results of the survey), the content of the table can be displayed something like this:

```
id  choice
1   4
2   5
3   4
4   3
5   3
 ...
```

The interpretation of these data is that the first person to respond to the survey chose the programming language PHP, while the second chose VB, and the third selected PHP. The next respondents chose, in order, Perl, Other, Perl, and C. The column *id* thus contains a running identification number that identifies the lines (the data set). The column *choice* contains, in coded form, the selection made by the survey participant, where the numbers 1 through 6 correspond to the programming languages C, Java, Perl, PHP, VB, and Other.

> TIP *Like any other specialized subject, the database world has its own argot. Database vocabulary includes the term* data record *for each line of the table above. Instead of* columns *(here* `id` *and* `choice`*) one often speaks of* fields.

In order to generate a table with the two columns *id* and *choice* the following command would suffice:

```
CREATE TABLE votelanguage (id INT, choice TINYINT);
```

The result is that the column *id* is declared with the data type *INT*, and *choice* is declared to be of data type *TINYINT*. That means that in theory, 2^{31} individuals (2,147,483,648, that is) could participate in our survey before the range for *id* were exhausted. (If *id* were declared as type *UNSIGNED INT*, then the number of potential participants would be doubled.) In *choice*, on the other hand, there are 2^{16} different values available. (The data types *INT* and *TINYINT* are discussed in Chapter 5 together with the other MySQL data types.)

By this point you may be wondering why I have plagued you with such a complicated command as *CREATE TABLE* when there is a much easier way to achieve the same result. The difference between the complicated and simple variants is the difference between good and bad database creation. (And you wouldn't want me to be leading you astray on our very first example, would you?)

The attribute *AUTO_INCREMENT* for the column *id* has the effect that with each new record the appropriate value for *id* is automatically inserted. Thus when the results of the survey are saved, only *choice* has to be specified, since the database takes care of *id* on its own. This attribute ensures a consistent numbering of the data records, which is important for efficient management of the table.

The attribute *NOT NULL* ensures that actual values must be placed in both columns. It is not permitted to store the data record *NULL* or (in the case of *choice*) not to insert any value at all. Thus this attribute prevents invalid data records from being stored. (Go ahead and try to force MySQL to accept such a data record. It will refuse and present you with an error message.)

PRIMARY KEY (id) has the effect that the column *id* is used to identify the data records. That is the reason that the column was provided for in the first place, but MySQL is not so clever as to be able to read your mind, and it requires that it be informed precisely as to what your wishes are. (In this case *id* is called a *primary key*.) The definition of a primary key has a decisive influence on the speed with which data records can be accessed. That holds especially for linked tables—but let us not get ahead of ourselves. We shall stick for the moment with simple tables, always define a single primary key (and do so, if possible, for an *INT* field with the attributes *NOT NULL* and *AUTO_INCREMENT*).

Why Make It Complicated, When It Could Be So Much Easier?

Perhaps it has occurred to you that we have presented a rather complex solution to an easy problem. It could done in a much simpler fashion. All we need to do is to control six counters, such as in a table of the following form:

id	counter
1	2
2	0
3	7
4	9
5	2
6	1

Such a display would mean that two participants expressed a preference for the C language, none for Java, seven for Perl, nine for PHP, and so on. Each time a preference is registered the corresponding counter is incremented by 1. The database would consist altogether of six lines; one wouldn't have to worry about performance; memory requirements would be essentially zero—nothing but advantages! And all of this is aside from the fact that the six counters could be effortlessly stored in a small text file. In short, MySQL is totally unnecessary for this survey.

All right, then, you are correct! However, such an attitude is helpful only as long as we are dealing with a simple example. What happens when you would like to offer each participant the opportunity to make a comment? What if you allow the participants to fill out the questionnaire only after a login (to rule out the possibility of multiple voting)? What if you wish to record the time and IP address for each participant (again to secure against attempts at manipulating the survey)?

In all these cases you would have to store the responses in a table like the one that we have presented. (The table would have merely to be enlarged by a column or two. The structure of the program can remain essentially the same.) So, if the conception of this example seems a bit overly complicated at first glance, the reason is that we are keeping open the possibility, even in this first example, of a later extension.

The Questionnaire

As we have mentioned already in the introduction, our entire project consists of two web sites. The first page (`vote.html`) contains the questionnaire. This is pure HTML code (without PHP). The second page (`results.php`) carries out two tasks: evaluating the questionnaire and displaying the results.

The HTML code of the questionnaire is given here, so that the code for evaluation in the next section will be understandable. Most important are the attributes *name* and *value* of the elements of the questionnaire. All of the radio buttons have the name *vote*, while *value* contains values between 1 and 6. The OK button has the name *submitbutton*, and as *value* the character string *"OK"* is used.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- php/vote/vote.html -->
<html><head>
  <title>MySQL-poll</title>
</head>
<body>
<h2>MySQL- poll </h2>
<p><b> What is your favorite
programming language for developing MySQL applications?</b>

<form method="POST" action="results.php">
  <input type="radio" name="vote" value="1">C/C++
  <br><input type="radio" name="vote" value="2">Java
  <br><input type="radio" name="vote" value="3">Perl
  <br><input type="radio" name="vote" value="4">PHP
  <br><input type="radio" name="vote" value="5"> VB/VBA/VBScript
  <br><input type="radio" name="vote" value="6"> Other
  <p><input type="submit" name="submitbutton" value="OK">
</form>
<p>Go directly to the <a href="./results.php">results</a>.
<p>Back to the
<a href="../mysqlbook.html">MySQL book page</a>.
</body>
</html>
```

Questionnaire Evaluation and Displaying Results

One can call `results.php` either directly (for example, via a link) or using the data in the questionnaire. In one case what is shown is only the current state of the questionnaire. In the other case, the data are also evaluated and stored in the database.

Establishing a Link to the Database (`mysql_connectinfo.inc.php`)

The opening lines of `results.php` consist of HTML code. Then comes the first task within the PHP code, which is to create a link to MySQL. For this purpose the PHP function *mysql_pconnect* is used, to which three pieces of information are passed:

- user name
- password
- computer name (host name)

If you have installed MySQL yourself on the local computer and have not yet secured it with a password, then you may simply pass empty character strings as parameters. The specification of the computer name is necessary only if the web server (that is, Apache) is running on a different computer from the one that is running MySQL.

It is fundamentally not a good idea to write the user name and password in plain text in a PHP file. Of course, the visitors to your web site will never see the source text of this file (because the PHP code is first evaluated, and the resulting HTML document contains only the PHP output), but configuration errors can occur. The two most common errors are (1) a change in the Apache configuration (for example after an update) that results in PHP files not being executed and thus being displayed in their original form; (2) an incorrect or slipshod configuration of the FTP server, which allows users to read PHP files on your web site directly (i.e., bypassing Apache and PHP).

To avoid making it too easy for strangers to get at your password (while at the same time not having to write this information in every PHP script, which would cause a great deal of work in the event of a change in the MySQL password), the MySQL login information is usually stored in its own file. This file is stored in a directory of the web server that normally is inaccessible (protected by `.htaccess`). Make certain that it is impossible to get at this file via anonymous FTP.

In the following example this password file has the name `mysql_connectinfo.inc.php`. It is stored in the directory `_private`. (If you have not yet secured your MySQL server, you can simply give an empty password.)

```
<?php
  // <website-root-dir>/_private/mysql_connectinfo.inc.php
  // link data
  $mysqluser="user"; // user name for MySQL access
  $mysqlpasswd="xxx"; // password
  $mysqlhost="localhost"; // name of the computer on which MySQL is running
  $mysqldbname="test_vote"; // name of the database
?>
```

The first PHP instruction in `results.php` is an *include* command, by means of which `mysql_connectinfo.inc.php` is loaded. Depending on the location in the directory hierarchy where `results.php` and `mysql_connectinfo.inc.php` are located, you will have to edit the path name in *include.*

The function `mysql_pconnect` returns an identification number that is stored in the variable *link.* In the present example, *link* is used only to identify possible problems in creating the link. In this case an error message is displayed and the PHP code ended abruptly with *exit.* Otherwise, the active database is selected with *mysql_select_db.* (MySQL usually manages several databases, and the effect of *mysql_select_db* is that the following commands automatically refer to *test_vote.*)

Evaluating the Data and Storing Them in the Database

The next task in `results.php` consists in evaluating the data in the questionnaire and storing them in the database. If `results.php` was called via the questionnaire of `vote.html`, then the variable *submitbutton* contains the character string *"OK"*. (Compare the *name* and *value* attributes in `vote.html`.)

For the sake of security a validation test is carried out for the variable *vote.* (For example, a participant may have forgotten to select one of the choices.) If *vote* contains a valid value, then this value is stored in the database. The SQL command to accomplish this will look something like the following:

```
INSERT INTO votelanguage (choice) VALUES (3)
```

What *INSERT INTO* accomplishes is to insert a new data record (a new row) into the table *votelanguage*. The expression *(choice)* specifies all of the fields of the data record for which values should be passed by means of the command *VALUES (. . .)*. Since MySQL takes responsibility on its own for the field *id* (attribute *AUTO_INCREMENT*; see above), in this example *choice* is the only field affected. Of course, instead of "3" the value of the variable *vote* will be placed, depending on which of the programming languages was selected in the questionnaire. The SQL command thus constructed is then passed along to MySQL with the PHP function *mysql_query*.

POINTER *If you have never had dealings with SQL and thus are unfamiliar with the syntax of SQL commands, fear not. In Chapter 6 you will find an extensive introduction to SQL. The two commands used in this chapter, namely* INSERT *and* SELECT, *are presumably more or less self-explanatory.*

Displaying the Survey Results

Regardless of whether the questionnaire has just been evaluated, the previous results of the survey must be able to be displayed. (If a new vote has been cast, it will be taken into account.)

First a check must be made as to whether the *votelanguage* table contains any data at all. (When the questionnaire is first placed on the Internet, no votes have yet been cast.) The required SQL query looks like this:

```
SELECT COUNT(choice) FROM votelanguage
```

The SQL command is again executed with *mysql_query*. What is new this time is that a link to the result of the query is stored in the variable *result*. The result of a *SELECT* command is, in general, a table. However, in the above example this table consists of merely a single row and a single column. (Furthermore, *result* contains only an ID number, which is used as a parameter in various other *mysql_xxx* functions. It is the task of PHP to take care of the actual management of the result.)

To evaluate the result, the function *mysql_result($result, 0, 0)* is used, by which the element in the table from the first row and first column is read. (With MySQL functions counting begins with 0.)

Provided that the *votelanguage* table is not empty, a loop is executed to report the percentage of votes cast for each programming language. The requisite SQL queries look similar to the one above, only now the number of data records that contain a particular value of *choice* (for example, 3) is counted.

```
SELECT COUNT(choice) FROM votelanguage WHERE choice = 3
```

For evaluation *mysql_result* is used. Then a bit of calculation is necessary to round the percentages to two decimal places.

Program Code (results.php)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- php/vote/results.php-->
<html><head>
<title>Survey Result</title>
</head><body>
<h2> Survey Result </h2>
<?php
  include("../../_private/mysql_connectinfo.inc.php");
// Create Link to Database
  $link =
    @mysql_pconnect($mysqlhost, $mysqluser, $mysqlpasswd);
  if ($link == FALSE) {
    echo "<p><b>Unfortunately, no link to the database
           can be made. Therefore, the results
           cannot be displayed at present. Please try
           again later.
           </body></html>\n";
    exit();
  }
  mysql_select_db($mysqldbname);
  // if questionnaire data are available:
  // evaluate + store
  if($submitbutton=="OK") {
    if($vote>=1 && $vote<=6) {
      mysql_query(
         "INSERT INTO votelanguage (choice) VALUES ($vote)");
    }
    else {
      echo "<p>Not a valid selection. Please vote
               again. Back to
               <a href=\"./vote.html\">questionnaire</a>.
               </body></html>\n";
      exit();
    }
  }
  // display results
  echo "<P><B> What is your favorite programming language
         for developing MySQL applications?</B>\n";

  // Number of votes cast
  $result =
    mysql_query("SELECT COUNT(choice) FROM votelanguage");
  $choice_count = mysql_result($result, 0, 0);

  // Percentages for the individual voting categories
  if($choice_count == 0) {
    echo "<p>No one has voted yet.\n";
  }
else {
    echo "<p>$choice_count individuals have thus far taken part
           in this survey:<br>\n";
    $choicetext = array("", "C/C++", "Java", "Perl", "PHP",
                        "VB/VBA/VBScript", "Other");
    for($i=1; $i<=6; $i++) {
      $result = mysql_query(
         "SELECT COUNT(choice) FROM votelanguage " .
         "WHERE choice = $i");
      $choice[$i] = mysql_result($result, 0, 0);
      $percent = round($choice[$i]/$choice_count*10000)/100;
      print("<br>$choicetext[$i]: $percent %\n");
    }
  }
?>
<p>Back to
<a href="../mysqlexamples.html">MySQL-examplepage</a>.
</body>
</html>
```

The Resulting HTML Code

If you are relatively inexperienced with PHP, you might find it helpful to have a look at the resulting HTML code (that is, what the user finally sees in the browser, as depicted in Figure 3-2) as an aid to understanding the program presented above.

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.0//EN">
<!-- results.php -->
<html><head><title>Umfrageergebnis</title></head><body>
<h2>Umfrageergebnis</h2>
```

```
<P><B>What is your favorite programming language
      for developing MySQL applications?</B>
<p>6 individuals have thus far taken part in this survey:<br>
<br>C/C++: 0 %
<br>Java: 0 %
<br>Perl: 16,67 %
<br>PHP: 50 %
<br>VB/VBA/VBScript: 33,33 %
<br>Andere: 0 %
<p>back to
<a href="../mysqlexample.html">MySQL-examplepage</a>.
</body></html>
```

Ideas for Improvements

Layout

It may have occurred to you that I have not paid much attention to the appearance of things in this example. Naturally, the results of the survey could be presented in a nicely formatted table or perhaps in a colorful bar graph. However, that has nothing to do with the subject of this book, namely, MySQL. Introductions to the attractive presentation of HTML documents, to the PHP programming of various graphics libraries, and the like can be found in quantity in a number of books on HTML and PHP (and, of course, on the Internet).

Questionnaire and Results on a Single Page

It is certainly possible to execute all the elements of our survey—questionnaire, evaluation, and presentation of results—with a single PHP script. However, as a rule it is a good idea for the participants not to see the previous results before casting a vote.

Options

You could make the survey more interesting or more informative by offering additional opportunities for input. In our example there could be a text field for the input of other programming languages or perhaps a text field for optional comments or for information about the professional background of the participants. However, you must consider that the more you ask, the more superfluous data you will collect. (If someone does not want to answer a required question, you may get a deliberately false response.)

Multiple Selection

In principle, you could construct the questionnaire in such a way that a participant could select more than one programming language. In the HTML form you would use check boxes instead of radio buttons. However, the necessary changes in the design of the database are somewhat more complicated. The table *votes* must now be able to store in each data record an arbitrary collection of programming languages. It would be easy, though inefficient, to reserve a separate column in the table for each programming language and then store the value 1 or 0 depending on whether or not the language was voted for. If you do not want the *votes* table to take up any more space in memory than necessary, then you could store a multiple selection as a combination of bits. For this purpose MySQL offers the data type (*SET*). But this option would make the program code for the storage and evaluation of the survey results considerably more complicated.

Protection Against Manipulation

In most Internet surveys the goal is not to obtain information but to promote interest, and thereby obtain a large number of accesses to one's web page. In such cases there is not much point in protecting the survey against misuse. However, if you wish to provide such protection, you have a number of possibilities:

- You could place a "cookie" containing a random number on the participant's computer and store this same value in the database. In this way an attempt to cast additional votes could be easily caught. (The disadvantage is that many Internet users are opposed to cookies and delete all cookies at startup or else prohibit their placement altogether.)
- You could require a login with e-mail address and password. Only those who have registered may vote. You store information in the database on who has already voted. (The disadvantage is that Internet users tend to have little patience for logins. Not many Internet users would give their e-mail address simply for the privilege of filling out your questionnaire.)

It is true in general that you cannot completely prevent such manipulation of a survey. That is, every system of security can be gotten around by someone who puts enough effort into the attempt.

POINTER *There are many complete PHP solutions for questionnaires available, both with and without MySQL support. Before you set about reinventing the wheel, you might want to take a squint at some of the PHP sites on the Internet, for example the following:*

`http://www.hotscripts.com/PHP/`