Key Skills & Concepts

- Installing and Using the MySQL Workbench
- Understanding SQL
- Using MySQL
- Reviewing MySQL Word Usage
- Using MySQL Operators
- Exploring MySQL Functions

MySQL is a relational database management system that uses the Structured Query Language (SQL) to store, work with, and retrieve information over the Internet. MySQL is called "the world's most popular open-source database," with many millions of websites using it as a web resource that runs on a web server, as does PHP. That is the way we'll discuss it here, but MySQL can also run on a stand-alone server not connected to the Web. MySQL is built to handle large-volume, multiuser database access. mysql.com says "many of the world's largest and fastest-growing organizations including Facebook, Google, Alcatel Lucent, and Zappos rely on MySQL." MySQL's low or no cost, ease of use, speed, reliability, and usability for small to large sites make it easy to see why it has widespread use.

NOTE

The GNU General Public License discussed here is a model license for free software that was developed for the Swedish GNU operating system. "GNU" is not an acronym, but the name of an antelope-like animal.

About MySQL

MySQL was developed and first released in 1995 by MySQL AB, a Swedish company that is now a part of Oracle. Oracle makes the program available at a number of levels, including under a free-to-the-user GNU General Public License (GPL) in addition to proprietary licenses. GPL stipulates that any system that uses MySQL must be distributed under a similar license. It is this arrangement that makes MySQL available free of charge

with the free Zend Server, as well as with WAMP (Windows servers for Apache, MySQL, and PHP) and XAMPP (cross platform servers for Apache, MySQL, PHP, and Perl). As this is written (summer 2014), Zend Server includes MySQL 5.5.23. Oracle offers the MySQL Community Server with MySQL for Windows, Linux, and other operating systems, which can be freely downloaded, but does require some effort to download, set up, integrate, and manage. It also offers MySQL Enterprise, which comes with many tools and technical support to make the job much easier, but for a hefty price. Zend Server, WAMP, and XAMPP make using MySQL relatively easy, free, and a good way to start a project. MySQL is a mature, full-featured, professional relational database management system (RDBMS), as you can see from its website, mysql.com. Many professional web developers believe that PHP and MySQL are the best way to add a database to a website of any size.

TIP

This and the following chapters will require Zend Server to be running and at least one browser to be opened to localhost. With the standard setup you were guided through in earlier chapters, Zend Server with MySQL should automatically start when you start your computer. As described in the last chapter, load a browser and, in the address bar, type **localhost/phpmyadmin**, where you see on the right that MySQL is installed as the database server (see Figure 10-1). On the left, you see that MySQL has its own database with a number of tables that support its use.

TIP

If you have problems seeing your website on http://localhost/, try http://127.0.0.1/.

MySQL gives a website the ability to store, retrieve, and manipulate large amounts of information quickly and efficiently. This allows a website to present a large catalog of items for sale, to provide access to and searchability of a large archive of information, or to register and service a large group of individuals, such as members of an organization.

This chapter will look at the MySQL Workbench as an alternative to phpMyAdmin, SQL as the foundation of MySQL, the structure and use of the MySQL command language, and the basics of working with MySQL directly with a command-line interface and phpMyAdmin.

Installing and Using MySQL Workbench

The MySQL Workbench is a recent graphic interface distributed and maintained by Oracle that allows users to create, manage, and administer MySQL databases. There are two versions of the Workbench: the free Community Edition that can be downloaded from mysql.com, and the extended Standard Edition that can be purchased.

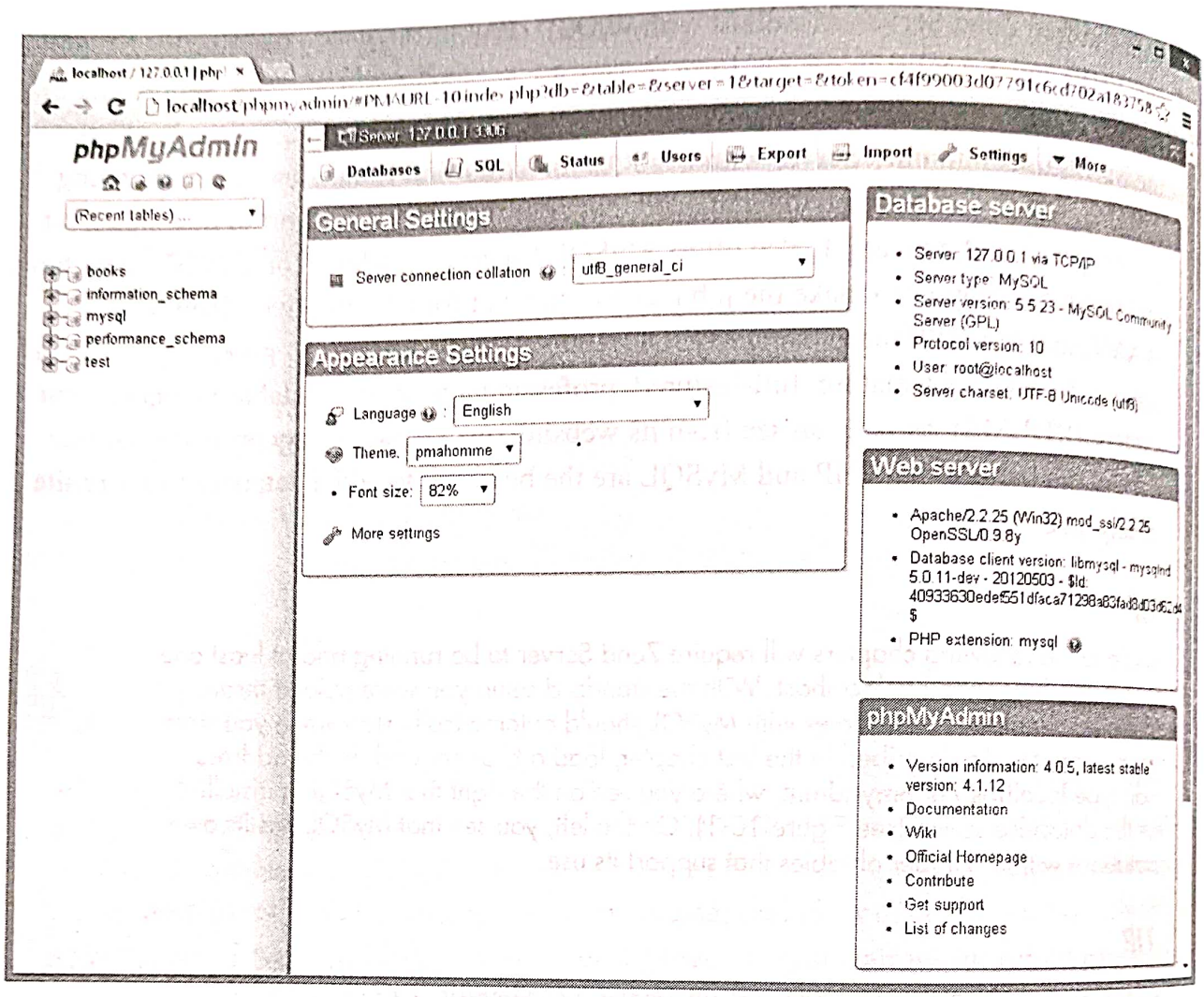


Figure 10-1 "localhost" is the Apache web server running on your computer and provides access to MySQL.

NOTE

phpMyAdmin is an early graphic user interface for MySQL that has been significantly upgraded over time, so today, it is a comprehensive and reliable tool. Because it has been free and open source for some time and included in WAMP, XAMPP, and LAMP (Linux servers for Apache, MySQL, and PHP) server packages, many people (including myself) became familiar with it and continue to use it as the MySQL interface of choice. MySQL Workbench is relatively recent and comes from Oracle. MySQL Workbench is a fully capable interface that many people like. It and phpMyAdmin are much like Ford and Chevy—it depends on your preferences. Try them both and see for yourself (and there are at least half a dozen other MySQL front ends if you want to search them out).

Download and Install MySQL Workbench

Download and install the MySQL Workbench Community Edition with the following steps.

1. Open a browser to mysql.com and click the Downloads tab. Scroll down to MySQL Community Edition and click Download From MySQL Developer Zone.
2. Scroll down to MySQL Workbench and click Download. Scroll down until you see Generally Available (GA) Releases and click Download opposite MSI Installer (assuming you are getting the Windows version).
3. Enter your Oracle user ID and password or sign up for a new one, go through Oracle's questions (part of the reason people stay with phpMyAdmin), and click Download for the third time. Click Run.
4. Click Next three times and then click Install. Click Yes to allow the installation, and when it is done, click Finish. MySQL Workbench will appear as you can see in Figure 10-2.

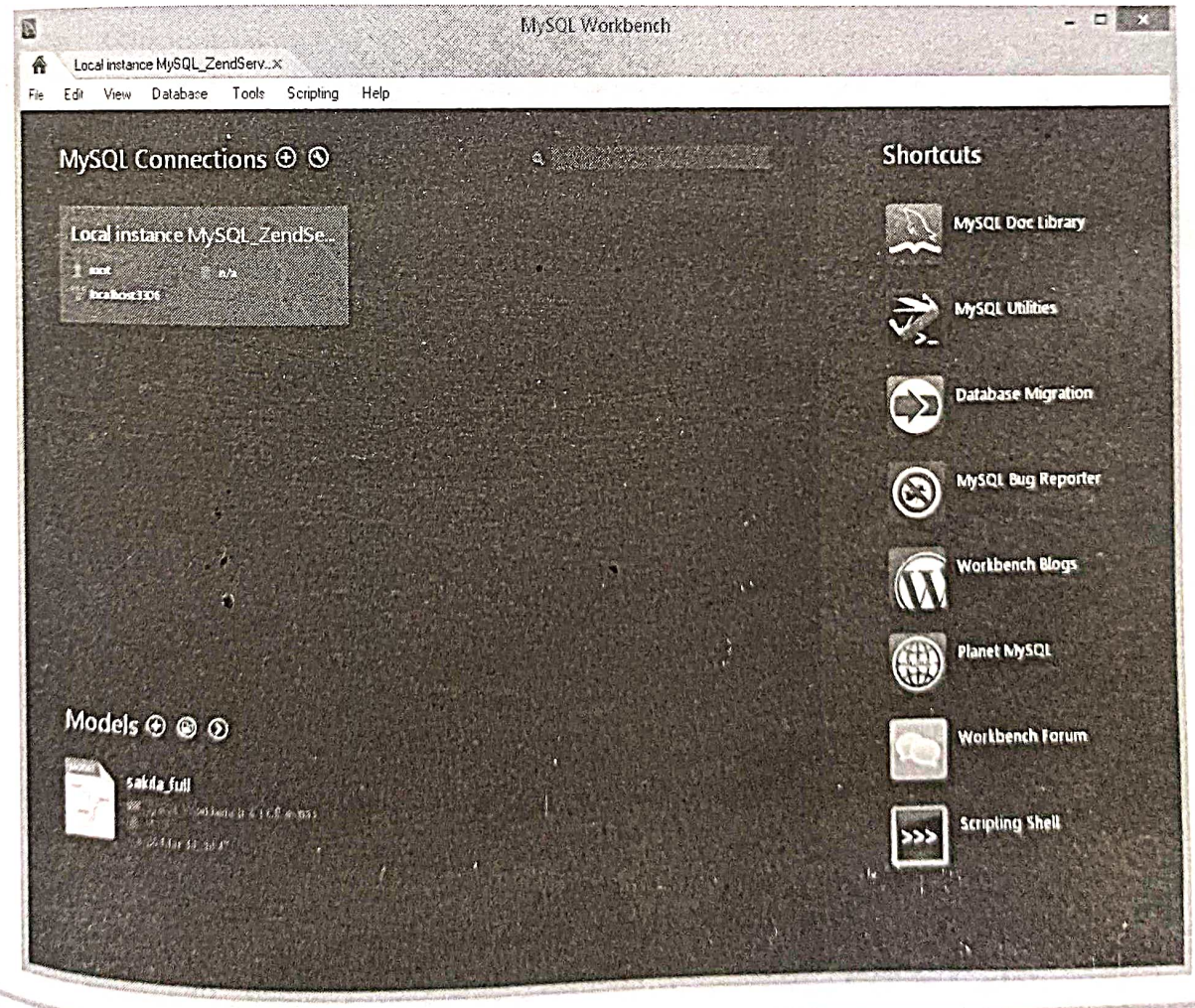


Figure 10-2 MySQL Workbench is desktop app, whereas phpMyAdmin is a web app.

Create a Database Model

One of the strengths of MySQL Workbench is the ability to create a visual model of a database you want to build, look at it graphically, and then convert to an operating database. To compare to what we did with phpMyAdmin, create a model of a relational database with four relational tables.

1. Within MySQL Workbench, click the Local Instance of MySQL on the Zend Server, enter your password (if you wish, click Save Password), and click OK.
2. Click the File menu and click New Model. A new database model will appear as shown in Figure 10-3.
3. Right-click mydb and click Edit Schema. (A database schema is the detail structure of a database with its tables, columns, and relationships among tables. In MySQL Workbench, the word "schema" is used synonymously with "database.") Type the name you want to use, like **MyBooks** and then click the X in the tab to close the schema.

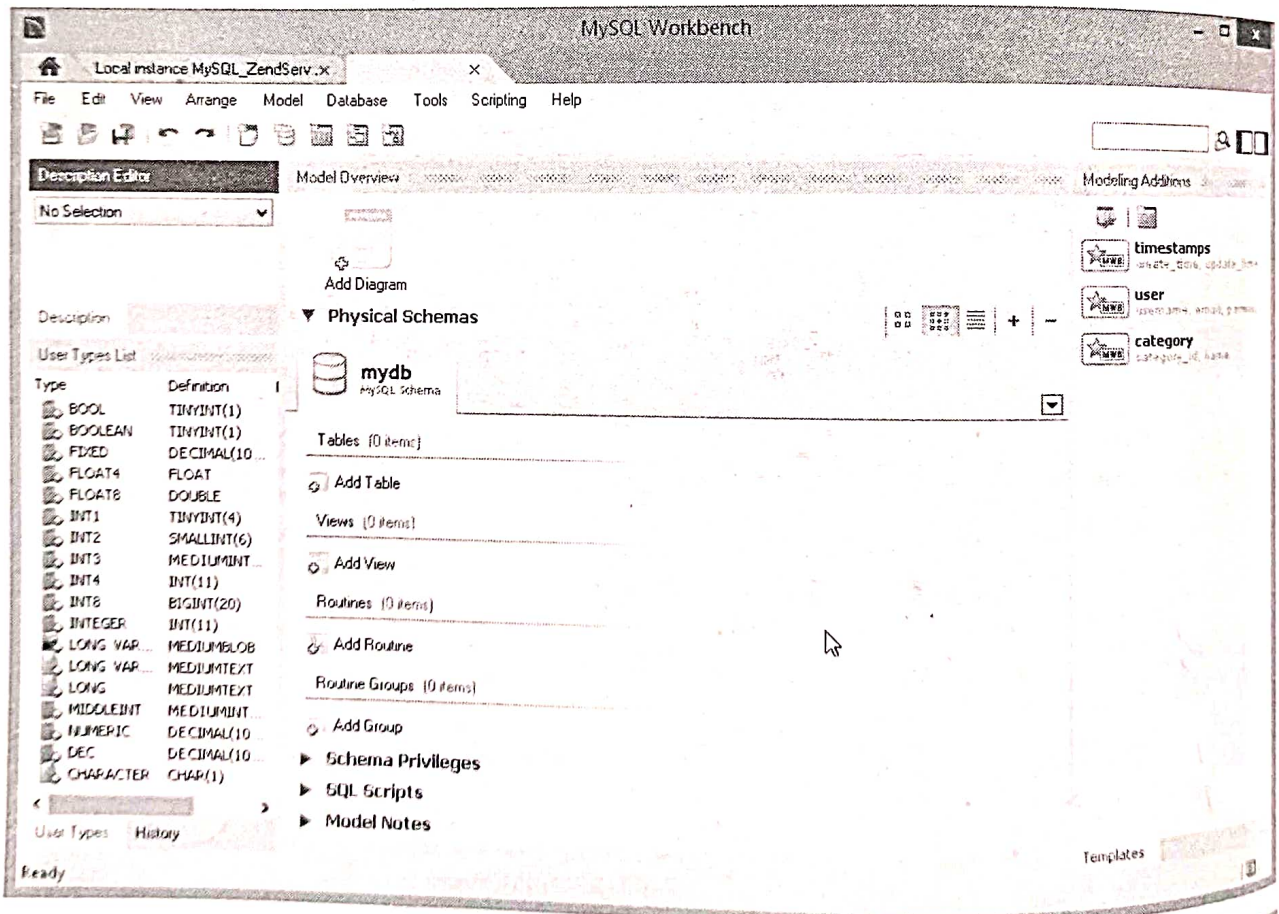


Figure 10-3 Words such as "schema," "triggers," "routines," and "views" pose a bit of a hurdle with MySQL Workbench.

4. Double-click Add Table. Click in the Table Name text box and type **Books**.
5. Click in the Column Name, type **Bkid**, leave INT as the data type, and click in the PK (Primary Key), UQ (Unique Index), and AI (Auto Increment) check boxes by scrolling right. NN (Not Null) will be automatically checked. Repeat this for the following columns:
   - Title, VARCHAR(255)
   - Aid, INT
   - Pid, INT
   - Price, DEC
   - Cid, INT
6. Then click the X in the tab to close the table and add the following three tables with their columns:
   - Authors with Aid, INT, PK, AI; LastName, VARCHAR(30); FirstName, VARCHAR(30)
   - Categories with Cid, INT, PK, AI; Category, VARCHAR(20); Section, VARCHAR(20)
   - Publishers with PID, INT, PK, AI; Publisher, VARCHAR(50); Address, VARCHAR(60); City, VARCHAR(60)
7. Click Add Diagram. From the catalog tree on the left, drag each of the tables to the grid in the middle so they look like the arrangement in Figure 10-4.
8. At the bottom of the list of icons on the left of the Diagram section, click the last icon to place a connector between existing columns, and then click one of the foreign keys (Aid, Pid, and Cid) in the Books table and the related primary key in its table. A line will appear connecting the two tables, as you see in Figure 10-4.
9. Click the File menu, click Save Model As, locate the folder you want to use, give the model a name, and click Save.

The diagramming part of MySQL Workbench is one of its best features.

Understanding SQL

SQL, or the Structured Query Language, is an efficient means of working with data using statements similar to spoken English. SQL was originally developed at IBM and has since become an international standard that is used by a number of RDBMSs. SQL was designed as a structured, declarative query language for relational databases. SQL statements generally begin with a declarative command, such as `CREATE`, `DELETE`,

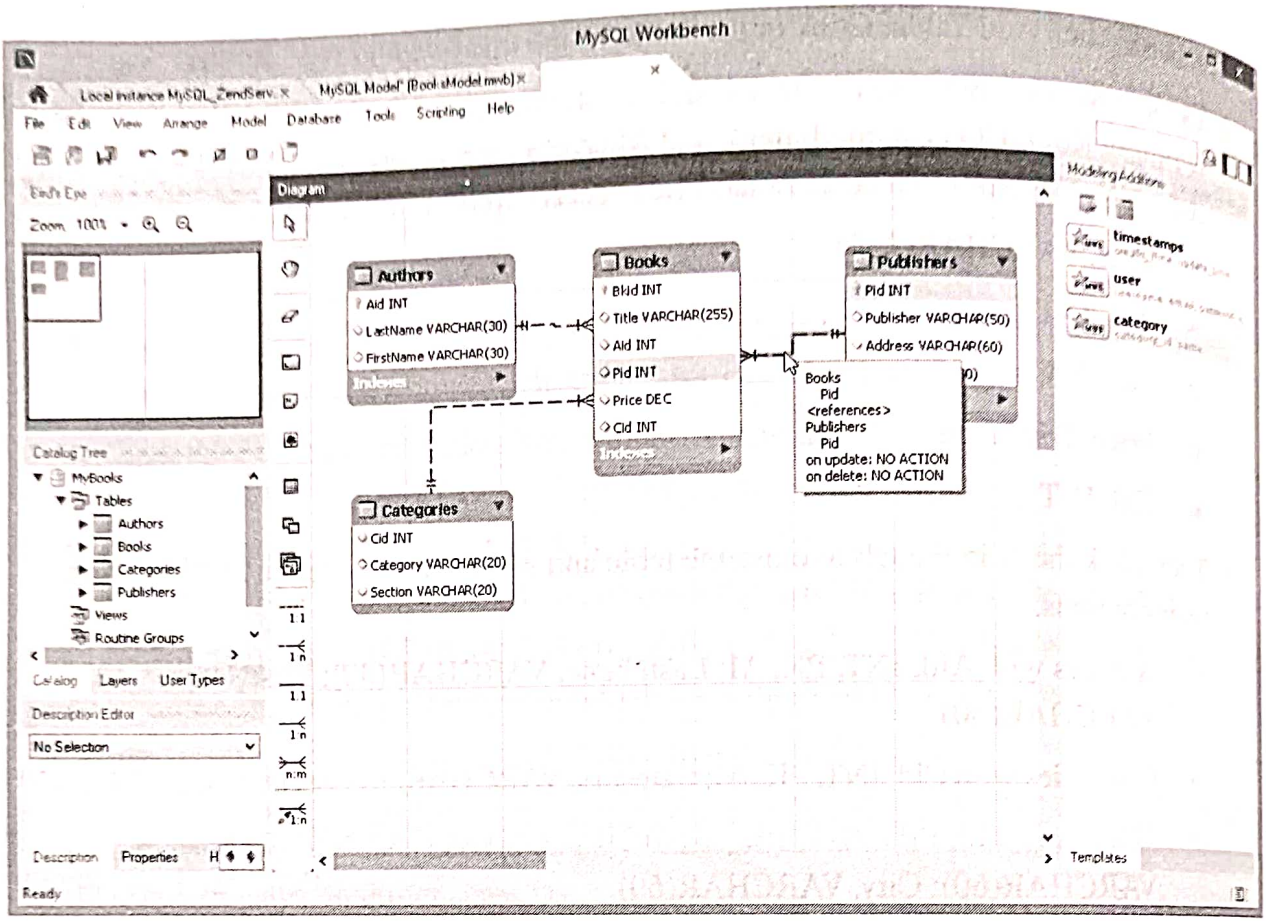


Figure 10-4 The MySQL Workbench diagramming gives you an excellent illustration of your database.

`INSERT`, `OPEN`, `SELECT`, and `UPDATE`. This is then followed by one or more clauses beginning with one or more of these keywords: `FROM`, `GROUP BY`, `HAVING`, `JOIN`, `ORDER BY`, and `WHERE`. For example, using the Books database and table defined in Chapter 9, a query might be:

```
SELECT * FROM books WHERE price > 5.00 ORDER BY title;
```

This will select all fields (the asterisk) in records from the Books table where the Price is greater than $5.00, and order the resulting list by Title. Listing 10-1 shows additional examples of SQL statements that build the Books table used with phpMyAdmin in Chapter 8 with five fields, three of which are variable-character strings of either 255 or 60 maximum characters. The Bkid field is an integer that must have a value (`NOT NULL`) and is the primary key for the table. The data for six books is then inserted into the table. In the first five of them, every field has a value, so the fields are not specified. For the sixth book, two of the fields are missing, so the ones that are present must be specified. Then the table

is updated to change the price for book 3 and to change the title for book 5 to correct a misspelling. Finally, book 6 is removed from the table.

NOTE

There are two SQL and MySQL coding practices that are followed in this book and that are important to understand:

- Here, and in most other publications, you'll see SQL and MySQL commands and keywords in all uppercase. This is not necessary for the commands and keywords to operate correctly—they are not case sensitive—but it helps to identify those words from other text, and so is recommended.
- Because MySQL and some other implementations of SQL allow multiple statements in the same call to the server, each statement must end with a semicolon (;). I believe that all MySQL statements should end with a semicolon to be consistent with PHP and have followed that practice throughout the remaining chapters of this book.

TIP

In this book I have followed the practice of making object names, such as databases, tables, and columns, all lowercase for human readability. On Windows computers, such names are not case sensitive, so they may be any mixture of cases without affecting machine readability. On Apple Mac and Linux computers, these names *are case sensitive*, and MySQL code will not operate properly if you mix cases. For that reason, as well as human readability, it is a good idea to make all names all lowercase and all commands and keywords all uppercase.

Listing 10-1 Examples of SQL Statements

```
CREATE TABLE books (
    bkid int NOT NULL PRIMARY KEY,
    title varchar(255),
    author varchar(60),
    publisher varchar(60),
    price decimal(10,2));
INSERT INTO books VALUES
    (1, 'Nightfall', 'Asimov', 'Bantam', 5.99);
INSERT INTO books VALUES
    (2, 'Patriot Games', 'Clancy', 'Berkley', 4.95);
INSERT INTO books VALUES
    (3, '2010', 'Clarke', 'Ballantine', 3.95);
INSERT INTO books VALUES
    (4, 'Lie Down with Lions', 'Follett', 'Signet', 4.95);
INSERT INTO books VALUES
    (5, 'A Theif of Time', 'Hillerman', 'Harper', 4.95);
```

```
INSERT INTO books (bkid,title,price) VALUES
   (6, 'The Innocent Man',7.99);
UPDATE books SET price=5.99 WHERE bkid = 3;
UPDATE books SET title='A Thief of Time' WHERE bkid = 5;
DELETE FROM books WHERE bkid = 6;
```

There are, of course, many other SQL commands, keywords, and clauses, as well as a variety of data types and functions. The purpose of the book, though, is to explore how to use MySQL with PHP. While MySQL is closely tied to SQL, there are some differences, so we'll leave SQL with this brief introduction.

Using MySQL

The MySQL server can be used in a variety of ways. In this and the next chapter, we'll look at two of them: directly through the MySQL command-line client and online through the phpMyAdmin user interface. In Chapter 12 and on, we'll look at accessing MySQL programmatically through PHP. In this section, we will briefly look at the command-line client, and in the next major section, we'll explore MySQL with phpMyAdmin.

TIP

The authoritative MySQL manual from Oracle is available for download for free at http://downloads.mysql.com/docs/refman-5.5-en.a4.pdf. I recommend that you add it to your library. The same manual is also available online at http://dev.mysql.com/doc/refman/5.5/en/create-database.html.

The MySQL command-line client is mainly for determining if MySQL is installed and running on a computer, and for doing some quick maintenance on a database. Major setup and maintenance of a database is better done with phpMyAdmin or the MySQL Workbench, while recurring use of a database is done programmatically, which in this book means PHP.

Starting the MySQL Command-Line Client

The MySQL command-line client was placed on your computer when Zend Server was installed and is in the c:/Program Files (86)/zend/mysql55/bin folder.

1. Open Windows Explorer by clicking its icon on the taskbar.
2. Navigate to the c:/Program Files (86)/zend/mysql55/bin folder.
3. Open the MySQL command-line window by double-clicking mysql.exe, as shown in Figure 10-5.

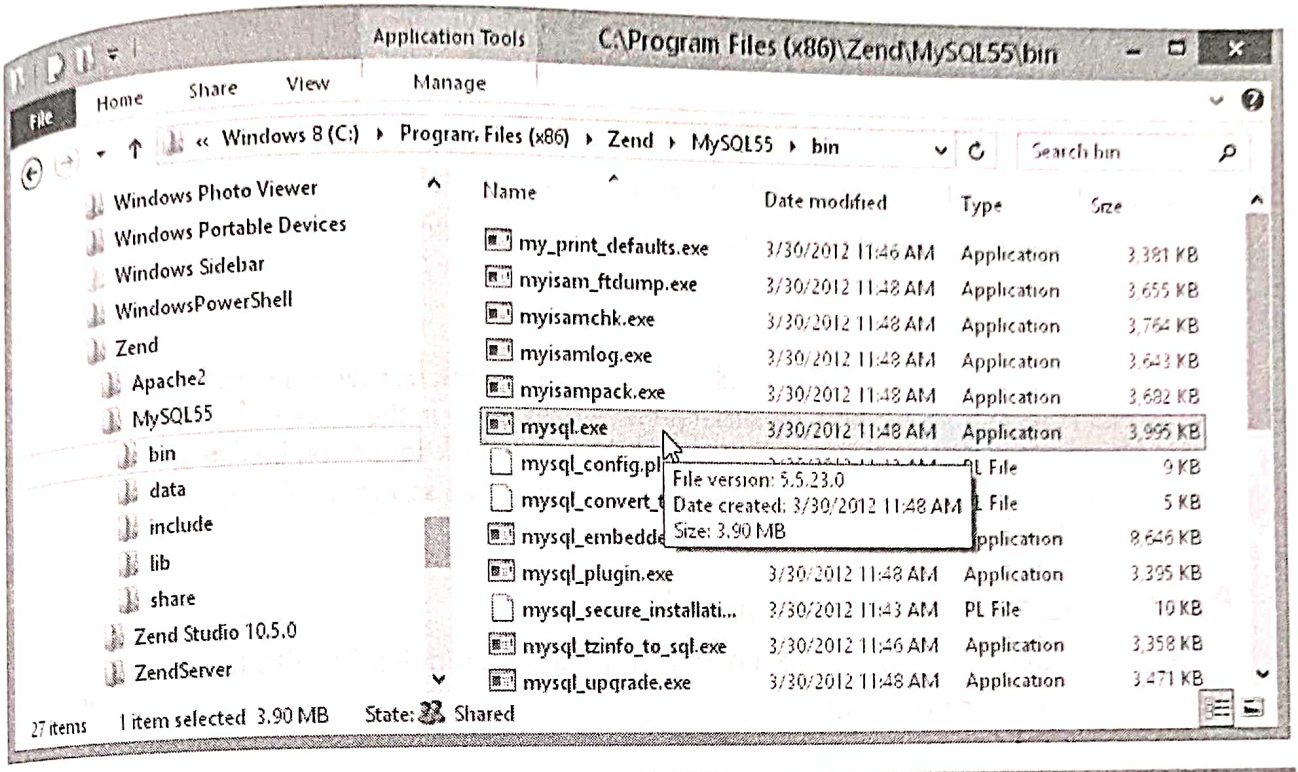
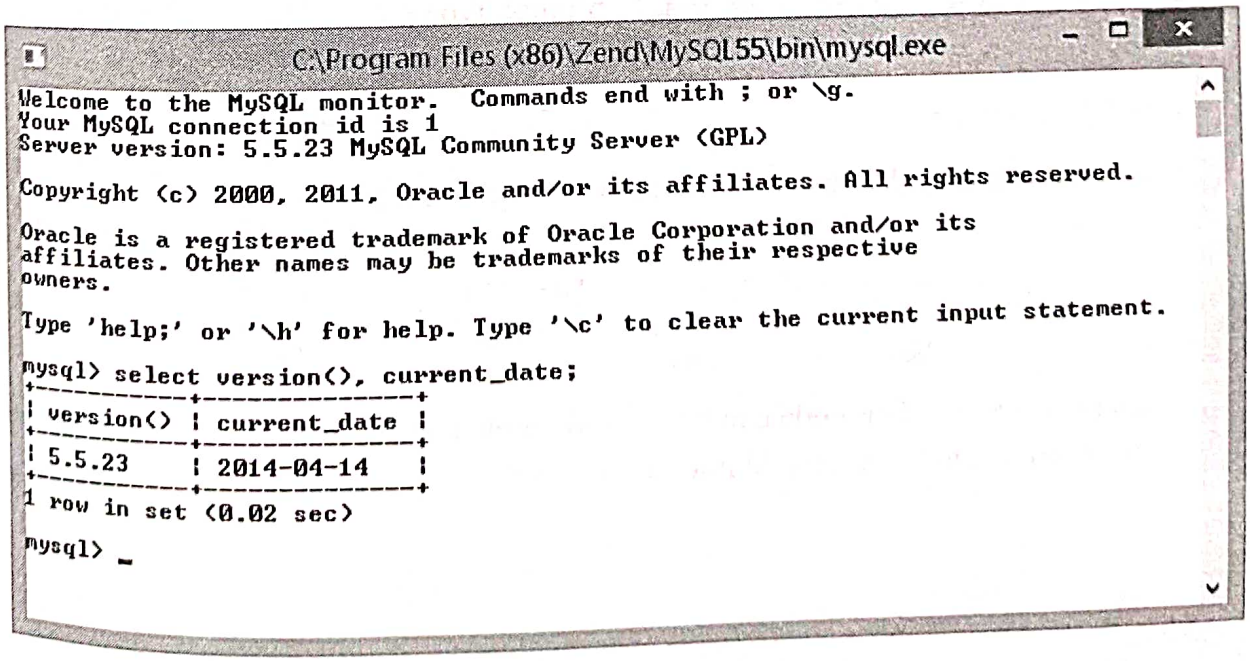


Figure 10-5 Zend Server provides a full suite of web development software, including a full version of MySQL.

4. At the `mysql>` prompt, type

```
select version(), current_date;
```

and press ENTER. Your command window should look like this:



5. If you have been able to do these instructions and get the results shown, then MySQL is properly installed and running. If not, you need to go back over your installation and possibly these instructions to see where the error is.
6. If you are immediately going on to the next section, do so directly. Otherwise, type **exit**, and press ENTER to close the command-line window.

NOTE

Your command-line window starts out with white text on a black background. You can reverse that to what is shown here by clicking the System menu in the upper-left corner, clicking Properties | Colors | Screen Text and clicking the black square on the left of the color bar. Then click Screen Background and click the white square on the right of the color bar. Finally, click OK.

Build a Table on the Command Line

Build a bit of the Books table, do a query on it, and then delete it with these instructions (see Figure 10-6 for the results).

1. If you exited the command-line window, use as much of the immediately preceding instructions as needed to reopen it.
2. To attach the Books table to an existing database, see what databases are available by typing the following at the `mysql>` prompt:

```
SHOW DATABASES;
```

   and pressing ENTER.
3. To use the Test database, at the `mysql>` prompt, type

```
USE test;
```

   and press ENTER.
4. To begin creating the new table, at the `mysql>` prompt, type

```
CREATE TABLE books (
      bkid int NOT NULL PRIMARY KEY,
      title varchar(255),
      author varchar(60));
```

   and press ENTER. This code can be entered all on a single line or, by using SHIFT-ENTER (newline), it can be on several lines as shown here.

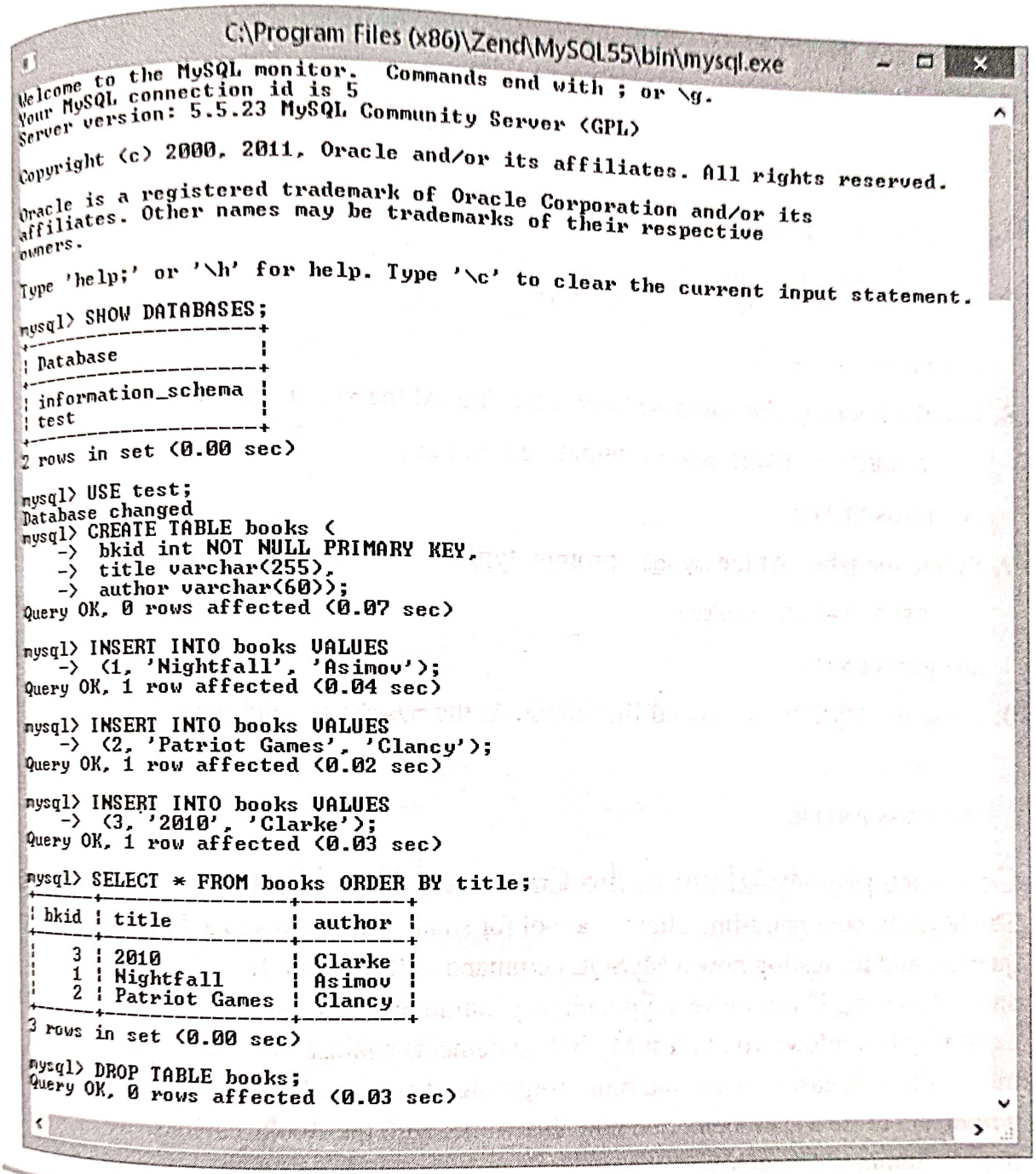


Figure 10-6 The MySQL command-line client is a bit tedious and probably not your first choice for working with MySQL.

5. Add books to the database at the `mysql>` prompt by typing

```
INSERT INTO books VALUES
      (1, 'Nightfall', 'Asimov');
```

and press ENTER. Again, this can be on one or two lines.

6. Continue adding books, and at the `mysql>` prompt, type

```
INSERT INTO books VALUES
      (2, 'Patriot Games', 'Clancy');
```

and press ENTER.

7. At the `mysql>` prompt, type

```
INSERT INTO books VALUES
      (3, '2010', 'Clarke');
```

and press ENTER.

8. List the books in the database, sorted by title. At the `mysql>` prompt, type

```
SELECT * FROM books ORDER BY title;
```

and press ENTER.

9. Delete the table. At the `mysql>` prompt, type

```
DROP TABLE books;
```

and press ENTER.

10. Close the MySQL command-line client. At the `mysql>` prompt, type

```
exit;
```

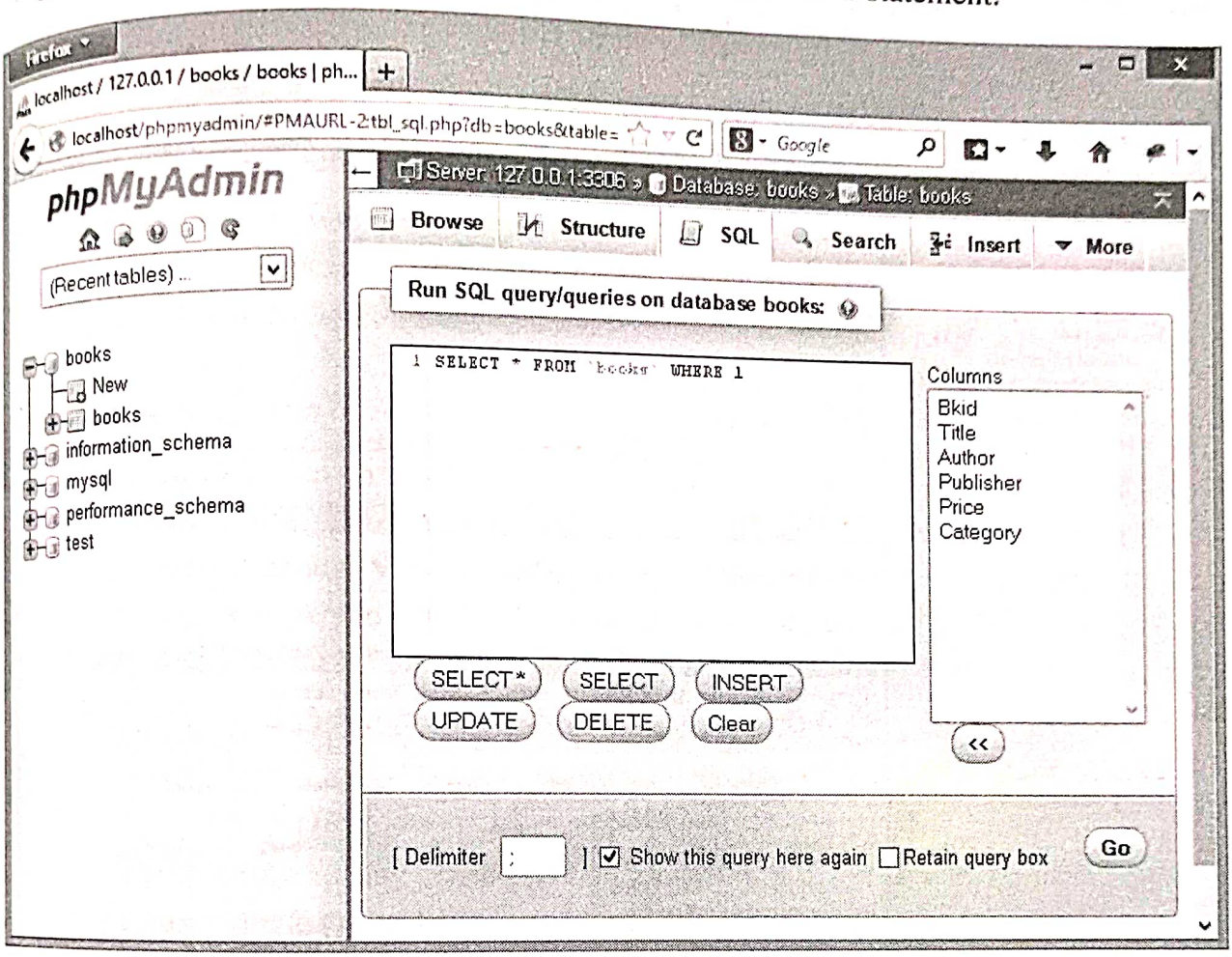
and press ENTER.

Compare phpMyAdmin to the Command-Line Client

The MySQL command-line client is a tool for small, quick fixes to a database; one-time queries; and for testing how a MySQL command will respond. It is also tedious to use and unforgiving if you make a typo. phpMyAdmin also has a similar, but more powerful, capability that allows you to test MySQL statements against a database and work with the results. It is easier to use and more forgiving of typos, allowing you to correct errors within and reuse a statement. See how this works with the Books database we built with phpMyAdmin in Chapter 9.

1. Open a browser, type **localhost\phpmyadmin** in the address bar, and press ENTER.
2. Open the Books database on the left and click the Books table to do a search of that table.

3. Click the SQL tab. You will see the beginning of a SELECT statement.



4. Select the 1 at the end of the statement and double-click Author in the Columns list on the right. `'Author'` (the Author column) will replace the 1.

5. Click immediately after `'Author'`, type a space, and then type **= 'Clancy';**. We now have a statement for a MySQL query of the Books table that says "Select all fields in the records in the Books table where the author is Clancy."

```
1 SELECT * FROM `books` WHERE `Author` =
  'Clancy'
```

6. Click Go. The result is shown in Figure 10-7. It not only answers the query, but also allows you to edit, copy, and delete what was selected.

7. Click Show Query Box at the top to go back and change the query—for example, changing `'Clancy'` to `'Clarke'`. Clicking Go again displays results that reflect the query change.

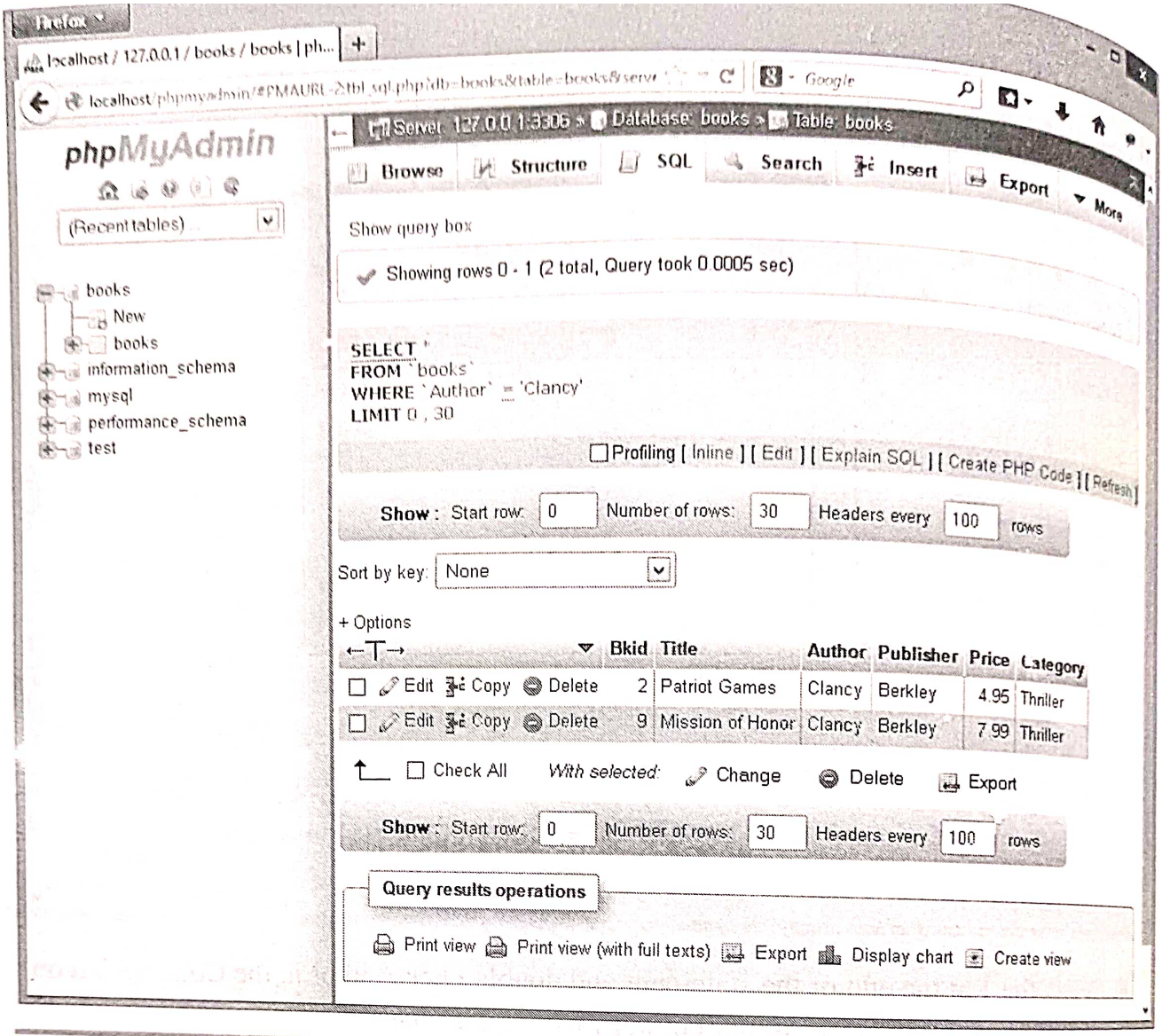


Figure 10-7 phpMyAdmin provides a powerful platform for testing MySQL statements.

Exploring the MySQL Language

MySQL has a number of commands and keywords, a complete set of math and comparison operators, and a number of functions that can be used to build its statements. In the following sections, we'll look at the most important of the operators and functions and demonstrate how they are used. In Chapter 11, we'll explore MySQL commands and other keywords and how all the MySQL elements are put together in statements. It needs to be emphasized that this is not a complete list of every aspect of the language. That is left for the MySQL manual and its over 3,000 pages. We'll begin, though, by looking at MySQL's word usage.

Reviewing MySQL Word Usage

MySQL's language is meant to be human readable and reasonably easy to understand. To do that, MySQL defines various types of words and sets out the rules for their use:

- **Commands** are declarative words that have a specific meaning within MySQL, such as `CREATE`, `SELECT`, and `UPDATE`. They are, by custom, in all capital letters for human readability, but that is not necessary for their proper operation.
- **Keywords** are supporting words for the commands to which MySQL has given a specific meaning. They include `FROM`, `GROUP BY`, `ORDER BY`, and `WHERE`. Once again, they are, by custom, presented in all capital letters, but that is not necessary.
- **Names or identifiers** are words used for specific elements such as databases, tables, and columns. They can contain the letters of the alphabet in either upper or lowercase, the numbers 0 through 9, the dollar sign ($), and the underscore (_). They cannot be over 64 characters in length. They can be within single (') or double quotes ("). Multipart or qualified names are separated by a period (.)—for example, `database_name.table_name.column_name` or `books.books.authors`.
- **Literals** are sets of alphabetic and numeric characters used as values in a statement. There are several types:
  - **String values** are any combination of alphabetic and numeric characters enclosed in either single (') or double (") quotes. Escape sequences can be embedded in a string.
  - **Numeric values** are sets of numeric characters that form an integer, decimal, or floating point number. The number may be preceded by either a plus (+) or minus (-) sign. Floating point numbers can be in scientific notation (`4.6E2`).
  - **Date and time values** are either strings or numbers in a context where a date is expected. Allowable formatting includes the following:
    - Date strings with delimiters in one of two forms: `'YYYY-MM-DD'` or `'YY-MM-DD'` with most punctuation marks allowable as delimiters, such as `'2016/04/19'` or `'16*04*19'`.
    - Date strings without delimiters in one of two forms: `'YYYYMMDD'` or `'YYMMDD'`, such as `'20160419'` or `'160419'`.

- Date numbers in either of two forms: `YYYYMMDD` or `YYMMDD`, such as `20160419` or `160419`.
- Two-digit years 70 to 99 are interpreted as 1970 to 1999, while years 00 to 69 are interpreted as 2000 to 2069.
- Date and time strings with delimiters in one of two forms: `'YYYY-MM-DD HH:MM:SS'` or `'YY-MM-DD HH:MM:SS'` with most punctuation marks allowable as delimiters, such as `'2016/04/19 11:24:30'` or `'16*04*19 11:24:30'`. The separator between date and time can be either a space or the capital letter `T`.
- Date and time strings without delimiters in one of two forms: `'YYYYMMDDHHMMSS'` or `'YYMMDDHHMMSS'`, such as `'20160419112430'` or `'160419112430'`.
- Date and time numbers in one of two forms: `YYYYMMDDHHMMSS` or `YYMMDDHHMMSS`, such as `20160419112430` or `160419112430`.
- **Boolean values** are constants expressed by the words `TRUE`, `true`, `FALSE`, or `false`.

- **User-defined variables**, or just "user variables," are names that can be assigned values within a MySQL statement and then used in other statements. User variables begin with an `@` (at sign) followed by a name you give it. The name can contain any alphanumeric characters plus `.` (period), `_` (underscore), and `$` (dollar sign), and are not case sensitive. User variables are assigned a value with the assignment character `:=` (semicolon equal sign) or, in the `SET` clause with just an `=` (equal sign). Examples of user-defined variables are
  - `@book := 'Nightfall'`
  - `SET @price = 7.99`
- **Expressions** are a word or phrase, which can include names, literals, user variables, operators, and functions that provides a value in a MySQL statement.
- **Reserved words** are those for which MySQL has a special usage. There are a large number of reserved words, too many to list here. For that reason, it is a good practice to enclose names, such as database, table, and column names, in either single or double quotes whenever it is possible that a name could be misinterpreted. Also, if you are getting errors for which you cannot find another reason, try putting names in quotes.

- **Comments** are in one of three forms:
  - Begin with a `--` (double-dash) to comment to the end of the line.
  - Begin with a # (hash mark) to comment to the end of the line.
  - Begin with `/*` (slash asterisk) to comment until the following `*/`. Since this is consistent with PHP, it is often used.
- **Escape sequences** are almost the same as those for PHP, as you can see in Table 10-1. All escape sequences begin with a backslash (\) or escape character.

Using MySQL Operators

Within MySQL statements, you can use expressions that include arithmetic, comparison, and assignment operators. For example, in the statement `SELECT * FROM books WHERE price > 5.00`, the expression `price > 5.00` uses the greater-than comparison operator. As you'll see in later sections of this chapter, there a number of situations where expressions and operators can be used. The operators that MySQL has available are very similar to PHP and include

- Arithmetic operators
- Comparison operators
- Logical operators
- Assignment operators

Escape Sequence	Used to Generate the Following Character
\0	Null
\'	Single quote (')
\"	Double quote (")
\b	Backspace
\n	Newline
\r	Carriage return
\t	Tab
\z	End of file on Windows
\\	Backslash
\%	% in a pattern-matching context
_	_ in a pattern-matching context

Table 10-1 Escape Sequences That Can Be Used in MySQL

Arithmetic Operators

Arithmetic operators, shown in Table 10-2, allow you to perform arithmetic operations within a MySQL statement.

If both values are integers, the calculation will be done with 64-bit precision. If one of the values is unsigned (without a plus or minus), the result will be unsigned. If you are doing arithmetic with very large or very small numbers and the precision of the results is important, read the applicable sections of the manual (12.6.1) carefully. Arithmetic operators apply only to numbers. For dates, you must use the date functions.

Comparison Operators

Comparison operators, shown in Table 10-3, allow you to do comparisons within a MySQL statement.

The result of comparison operations is `TRUE (1)`, `FALSE (0)`, or `NULL`, where `NULL` is the absence of a value or is unknown.

TIP

If you want to assign `NULL` to a value (a variable or an expression), use "`IS NULL`" and not "`=NULL.`" The reason is that "`=NULL`" assigns the literal characters "`NULL`" to the values, while "`IS NULL`" assigns `'0'` to the value, which is, and can be tested for, "FALSE."

Logical Operators

Table 10-4 lists MySQL logical operators that allow you to include logical expressions within statements.

Operator	Description
`+`	Addition, the sum of two values
`-`	Subtraction, the difference between two values
`*`	Multiplication, the product of two values
`/`	Division, the quotient of two values
`%` or `MOD`	Modulus division, the remainder after dividing two values
`DIV`	Integer division, the integer quotient of two values, which do not have to be integers

Table 10-2 MySQL Arithmetic Operators

Operator	Description (Is a Value...)
=	Equal to another value
>=	Greater than or equal to another value
>	Greater than another value
<=	Less than or equal to another value
<	Less than another value
!= or <>	Not equal to another value
IS NULL	Absent
IS NOT NULL	Not absent
BETWEEN ... AND ...	Between two other values
NOT BETWEEN ... AND ...	Not between two other values

Table 10-3 MySQL Comparison Operators

The result of a logical expression is TRUE (1), FALSE (0), or NULL, where NULL is the absence of a value or is unknown. Examples of logical expressions where x is TRUE and y is FALSE are

- x AND y is FALSE, but if y is TRUE, x AND y is TRUE.
- x OR y is TRUE, but if y is TRUE, x OR y is still TRUE.
- x XOR y is TRUE, but if y is TRUE, x XOR y is FALSE.
- x AND NOT y is TRUE, but if y is TRUE, x AND NOT y is FALSE.

Operator	Description
AND or &&	Logical AND. Both values must be TRUE for the result to be TRUE.
NOT or !	Negates value. NOT TRUE is FALSE.
OR or ||	Logical OR. Either or both values may be TRUE for the result to be TRUE.
XOR	Logical exclusive OR. Either *but not both* values may be TRUE for the result to be TRUE.

Table 10-4 MySQL Logical Operators

Assignment Operators

Assignment operators allow you to assign a value to a user variable. := is the primary assignment operator, but in the SET clause, you can use =. In all cases other than in the SET clause, the = is considered a comparison operator. Examples of assignment operators are

- `@name := 'Michael'`
- `SET @date = 160419`

Exploring MySQL Functions

MySQL has a large number of functions that you can use within MySQL statements. The following sections group the more common functions into:

- Arithmetic functions
- Comparison functions
- Control flow functions
- Date and time functions
- Encryption and compression functions
- String functions
- System and information functions

Many of MySQL functions are similar to PHP functions. The benefit of the MySQL functions within a MySQL statement is that they are immediately evaluated by the MySQL server without having to go out to the PHP server.

Function names are immediately followed by a set of parentheses that contains the function's arguments. There can be no space between the function name and the parentheses following it. Functions may have zero or more arguments.

Arithmetic Functions

Arithmetic functions are used in expressions within MySQL statements to perform arithmetic operations. They include the functions shown in Table 10-5.

Examples of arithmetic functions are (use the Books table that was built in Chapter 9):

- `ABS(-64)` returns 64
- `SELECT AVG(Price) FROM books;` returns 5.96778

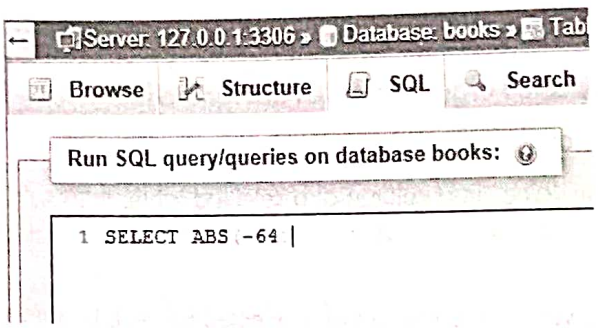
Function	The Result Is:
ABS()	The absolute value of the argument—the removal of any sign.
AVG()	The average value of the values in a column of a selected table. If DISTINCT is added to the arguments, repeated values in the column are ignored.
CEILING() or CEIL()	The smallest integer greater than the argument.
COUNT()	The number of rows in a selected table that contain the argument. If the argument is * then it is a simple count of rows. If DISTINCT is added to the arguments, rows with repeated argument values are ignored.
FLOOR()	The largest integer less than the argument.
MAX()	The maximum value of the values in a column of a selected table. If DISTINCT is added to the arguments, repeated values in the column are ignored, but the result is the same.
MIN()	The minimum value of the values in a column of a selected table. If DISTINCT is added to the arguments, repeated values in the column are ignored, but the result is the same.
RAND()	A floating point random number between 0 and 1. If an integer constant is added as an argument, it acts as a seed to produce a repeatable set of random numbers.
ROUND(a,b)	Rounds argument *a* to *b* number of decimal places. If *b* is missing, it is assumed to be 0.
SUM()	The sum of the values in a column of a selected table. If DISTINCT is added to the arguments, repeated values in the column are ignored.
TRUNCATE(a,b)	Truncates argument *a* to *b* number of decimal places. If *b* is missing, it is assumed to be 0.

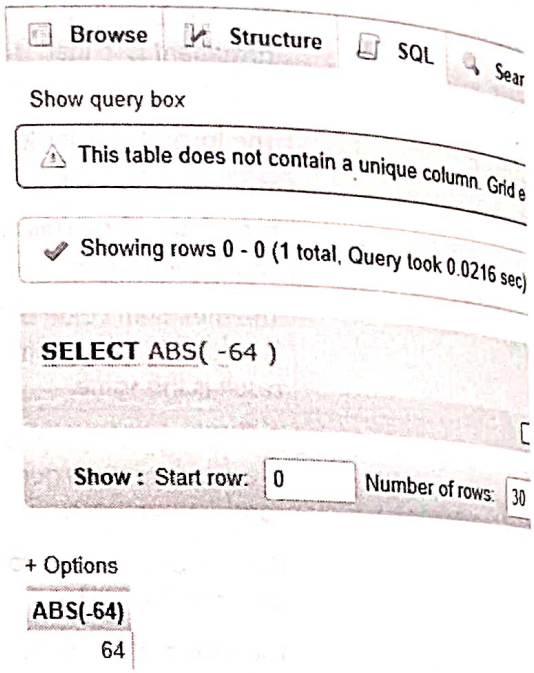
Table 10-5 MySQL Arithmetic Functions

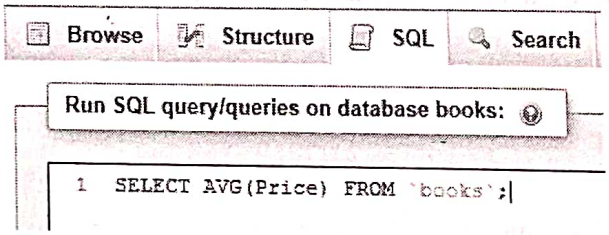
- `SELECT CEILING(4.56);` returns 5
- `SELECT COUNT(*) FROM books;` returns 9
- `SELECT FLOOR(4.56);` returns 4
- `SELECT MAX(Price) FROM books;` returns 7.99
- `SELECT MIN(Price) FROM books;` returns 3.95
- `SELECT RAND();` returns 0.355812035832379
- `SELECT ROUND(7.54);` returns 8
- `SELECT SUM(Price) FROM books;` returns 53.71
- `SELECT TRUNCATE(7.54);` returns 7

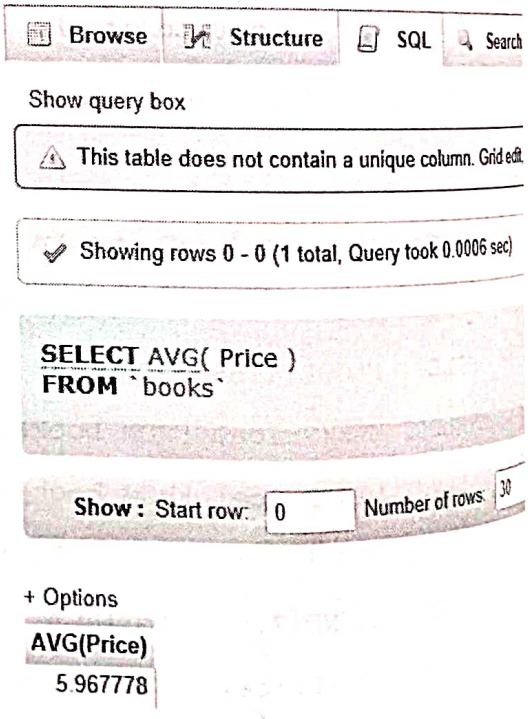
NOTE

You can try out any of these functions using phpMyAdmin, selecting the SQL tab and typing the functions following the `SELECT` command, and clicking Go as you see in the first and third illustrations next. phpMyAdmin then shows the result as you see in the second and fourth illustrations next.









Comparison Functions

Comparison functions are used in expressions within MySQL statements to compare values. They include the functions shown in Table 10-6.

Examples of comparison functions are.

- `COALESCE(NULL, NULL, 1492)` returns 1492
- `GREATEST(4.53, 7.21, 10.83)` returns 10.83
- `5.95 IN(3.99, 4.34, 5.95)` returns 1 or TRUE
- `INTERVAL(10, 4, 8, 12, 15)` returns 2
- `LEAST(4.53, 7.21, 10.83)` returns 4.53
- `'MartyMatthews' LIKE '%Matthews'` returns 1 or TRUE
- `5.95 NOT IN(3.99, 4.34, 5.95)` returns 0 or FALSE
- `STRCMP('Marty', 'MartyMatthews')` returns -1

```
SELECT STRCMP( 'Marty', 'MartyMatthews' )
```

STRCMP('Marty', 'MartyMatthews')
-1

Function	The Result Is:
`COALESCE()`	The first non-NULL argument
`GREATEST()`	The largest argument
`IN()`	TRUE if a value is in the set of arguments
`INTERVAL()`	The index or position of the argument that is not less than the first argument, counting from 0 (if the first argument is less than the second argument, the result is 0)
`LEAST()`	The smallest argument
`LIKE`	TRUE if a value matches a specified pattern, which can contain the % wildcard character to replace any number of characters or the _ wildcard character to replace a single character
`NOT IN()`	TRUE if a value is not in a set of values
`NOT LIKE`	TRUE if a value does not match a specified pattern
`STRCMP()`	`-1` if the first string is in the second string, `0` if the first string is equal to the second string, and `1` otherwise

Table 10-6 MySQL Comparison Functions

TIP

Do not mix quoted and unquoted arguments. For example, replace `(7, 5, 3, 'run')` with `('7', '5', '3', 'run')`.

NOTE

`MAX()` and `GREATEST()` and `MIN()` and `LEAST()` are easy to confuse. `MAX()` and `MIN()` are looking for the maximum or minimum value in a column in a table, while `GREATEST()` and `LEAST()` are looking for the largest or smallest value within the function's arguments.

Control Flow Functions

Control flow functions allow you to direct the flow within a MySQL statement based on values and conditions that are present. There are four control flow functions, one of which has two distinct versions, as shown in Table 10-7.

Examples of control flow functions are

- `CASE 'Marty' WHEN 'Marty' THEN 'Marty Matthews' WHEN 'Carole' THEN 'Carole Matthews' ELSE 'Michael Matthews' END` returns 'Marty Matthews'
- `@var:=56, CASE WHEN @var>50 THEN @var ELSE 50 END` returns 56

```
SELECT @var :=56,
CASE WHEN @var >50
THEN @var
ELSE 50
END
```

@var:=56	CASE WHEN @var>50 THEN @var ELSE 50 END
56	56

Function	Description
`CASE value WHEN comparison1 THEN result1 WHEN comparison2 THEN result2... ELSE result3 END`	Tests each of the comparisons against the initial value and returns the result when the comparison equals the value. If no comparison equals the value, then the result following `ELSE` is returned.
`CASE WHEN condition1 THEN result1 WHEN condition2 THEN result2... ELSE result3 END`	Tests each of the conditions and returns the result of the first condition that is `TRUE`. If no condition is `TRUE`, then the result following `ELSE` is returned.
`IF(condition, true_result, else_result)`	Tests *condition* and returns the *true_result* if the condition is `TRUE`; otherwise, the *else_result* is returned.
`IFNULL(value1, value2)`	If *value1* is not `NULL`, *value1* is returned; otherwise, *value2* is returned.
`NULLIF(value1, value2)`	Returns `NULL` if *value1* equals *value2*; otherwise, *value1* is returned.

Table 10-7 MySQL Control Flow Functions

- `@var:=56, IF(@var>50, @var, 50)` returns 56
- `IFNULL(NULL, 450)` returns 450
- `NULLIF(450, 450)` returns NULL

Date and Time Functions

MySQL has a number of date and time functions that can be used in expressions within MySQL statements to work with dates and times. Dates and times in MySQL in their default form look as they did in PHP, with year, month, day, hour, minute, and second divisions in the form YYYY-MM-DD HH:MM:SS. You can see this with the `NOW()` function to get the full set of values to the current second.

```
SELECT NOW( )
```

NOW()
2014-04-23 14:31:15

Table 10-8 lists some of the more commonly used date and time functions with some of the synonyms. If you are looking for a function and don't see it here, look at Section 12.13 of the MySQL Manual.

Function	Description
`ADDDATE(date, INTERVAL value unit)` or `DATE_ADD(date, INTERVAL value unit)`	Adds *value* in the units indicated to the *date*. The units can be YEAR, QUARTER, MONTH, WEEK, DAY, HOUR, MINUTE, SECOND, and MICROSECOND and various combinations.
`ADDDATE(date, days)`	Adds *days* to *date*.
`ADDTIME(time1, time2)`	Adds two times.
`CURDATE()` or `CURRENT_DATE()`	Returns the current date as YYYY-MM-DD.
`CURTIME()` or `CURRENT_TIME()`	Returns the current time as HH:MM:SS.
`DATE(datetime)`	Returns the date portion of *datetime* expression.
`DATE_FORMAT(date, format)`	Returns *date* as a string formatted using the codes in Table 10-9.
`DATE_SUB(date, INTERVAL value unit)` or `SUBDATE(date, INTERVAL value unit)`	Subtracts the value in the units indicated to the date. See `ADDDATE()` for *unit*.

Table 10-8 MySQL Date and Time Functions *(continued)*

Function	Description
DATEDIFF(*date1, date2*)	Returns the number of days after subtracting *date2* from *date1*.
DAYNAME(*date*) or DAY(*date*)	Returns the name of the day for *date* (Sunday..Saturday).
DAYOFMONTH(*date*)	Returns the day of the month for *date* (1 to 31).
DAYOFWEEK(*date*)	Returns the day of the week for *date* (1 to 7 with 1 = Sunday..7 = Saturday).
DAYOFYEAR(*date*)	Returns the day of the month for *date* (1 to 366).
Extract(*unit* FROM *date*)	Returns *unit* that is in *date*. See ADDDATE() for *unit*.
HOUR(*time*)	Returns the hour value that is in *time*.
LAST_DAY(*date*)	Returns the last day of the month in *date*.
MINUTE(*time*)	Returns the minute value that is in *time*.
MONTH(*date*)	Returns the month value (1 to 12) that is in *date*.
MONTHNAME(*date*)	Returns the month name that is in *date*.
NOW() or CURRENT_TIMESTAMP() or LOCALTIME()	Returns the current date as YYYY-MM-DD HH:MM:SS.
PERIOD_ADD(*period, months*)	Returns the period in the form YYYMM or YYMM after adding *months* to *period*.
PERIOD-DIFF(*period1, period2*)	Returns the number of months between *period1* and *period2*, which are in the form YYYMM or YYMM.
QUARTER(*date*)	Returns the quarter value (1 to 4) that is in *date*.
SECOND(*time*)	Returns the second value that is in *time*.
TIME_FORMAT(*time, format*)	Returns *time* as a string formatted using the time codes in Table 10-9.
WEEK(*date*)	Returns the week value (0 to 53) that is in *date*, assuming Sunday begins the week and week 1 is the first week with a Sunday in the year. You can optionally add a *mode* argument to change the assumptions; see the MySQL Manual.
YEAR(*date*)	Returns the year value (1000 to 9999) that is in *date*.

Table 10-8 MySQL Date and Time Functions

Examples of date and time functions are

- `ADDDATE(NOW(), 22)` returns 2014-05-15 14:23:47 where `NOW()` is as shown earlier (plus several minutes)

```
SELECT ADDDATE( NOW( ) , 22 )
```

ADDDATE(NOW(), 22)
2014-05-15 14:33:37

- `DATEDIFF(CURDATE(),'2014-12-25')` returns –246 since the second date is subtracted from the first
- `DATE_FORMAT(NOW(),'%M %D, %Y %h:%i:%s %P')` returns April 23rd, 2014 02:56:26 P

DATE_FORMAT(NOW(), '%M %D, %Y %h:%i:%s %P')
April 23rd, 2014 02:55:26 P

- `DAYNAME(CURDATE())` returns Wednesday
- `EXTRACT(MONTH FROM CURDATE())` returns 4
- `LAST_DAY(CURDATE())` returns 2014-04-30

Date and Time Formatting Codes

MySQL provides a number of date and time formatting codes, shown in Table 10-9, that can be used in the `DATE_FORMAT()` function. The codes must each begin with a % and are presented as a string with the appropriate delimiter or space between codes. For example: `'%M %D, %Y'` generates 'April 22nd, 2014'. Zero is an acceptable value for months, days, hours, minutes, seconds, and microseconds because MySQL allows storing incomplete dates. See the example for `DATE-FORMAT()` in the previous section.

Encryption and Compression Functions

You can encrypt, decrypt, compress, and uncompress strings within a MySQL statement using the functions shown in Table 10-10. The string resulting from encryption and compression is best stored using the `VARBINARY` or `BLOB` binary string data types to avoid character set conversion and space removal that can change values with other data types. Several older MySQL encryption functions may have become compromised and are not included in the discussion here.

Function	Description
%a	Three-character day of the week name (Sun..Sat)
%b	Three-character month name (Jan..Dec)
%c	Numeric month (0..12)
%D	Day of the month with English suffix (1st, 2nd, 3rd...)
%d	Numeric day of the month (00..31)
%H	Hour (00..23)
%h or %I	Hour (01..12)
%i	Numeric minutes (00..59)
%j	Day of the year (001..366)
%M	Month name (January..December)
%m	Numeric month (00..12)
%p	A or P for AM or PM
%r	12-hour time with AM or PM (hh:mm:ss AM/PM)
%S or %s	Seconds (00..59)
%T	24-hour time (hh:mm:ss)
%U	Week number (00..53) where Sunday is the first day of the week
%W	Full name of the day of the week (Sunday..Saturday)
%w	Numeric day of the week (Sunday=0, Saturday=6)
%Y	Numeric four-digit year
%y	Numeric two-digit year

Table 10-9 MySQL Date and Time Formatting Codes

For examples of encryption and compression functions, a new table was created in th Books database named "users" with three columns: "id," "user," and "pswd."

- `INSERT INTO users (user, pswd) VALUES ('marty', AES_ENCRYPT('passphrase', 'key'));`
  `SELECT user, AES_DECRYPT(pswd, 'key') FROM users;` returns "marty passphrase"

```
SELECT user, AES_DECRYPT( pswd, 'key' )
FROM `users`
LIMIT 0 , 30
```

user	AES_DECRYPT(pswd, 'key')
marty	passphrase

Function	Description
AES_DECRYPT(*string*, *key*)	Decrypts *string* that has been encrypted with the Advanced Encryption Standard (AES) using *key*
AES_ENCRYPT(*string*, *key*)	Encrypts *string* using the AES with *key*
COMPRESS(*string*)	Compresses *string* creating a binary string
DECODE(*string*, *key*)	Decodes *string* that has been encoded using *key*
ENCODE(*string*, *key*)	Encodes *string* using *key*
SHA2(*string*)	Returns a checksum for *string* using the Secure Hash Algorithm 2 (SHA2)
UNCOMPRESS(*string*)	Uncompresses *string* from a compressed binary string
UNCOMPRESS_LENGTH(*string*)	Returns the uncompressed length of a compressed *string*

Table 10-10 MySQL Encryption and Compression Functions

- `UNCOMPRESS(COMPRESS('amoderatelylargestring'));` returns `"amoderatelylargestring"`

CAUTION

MySQL encryption functions with their keys are sent to the server unencrypted unless a Secure Sockets Layer (SSL) connection is used.

String Functions

The MySQL string functions allow you to manipulate string values within a MySQL statement. Table 10-11 shows the most commonly used string functions.

Examples of string functions are

- `CHAR_LENGTH('Seattle')` returns 7
- `CONCAT('Buggs', 'Bunny')` returns "BuggsBunny"
- `FORMAT(9876.5432, 2)` returns 9,876.54
- `INSERT('6718 W 18th Street', 8, 2, '24')` returns "6718 W 24th Street"
- `LOCATE('car', 'Ft Carson')` returns 4
- `LTRIM('  something')` returns "something"
- `RIGHT('something', 5)` returns "thing"
- `SUBSTR('something', 3, 3)` returns "met"
- `UPPER('something')` returns "SOMETHING"

Function	The Result Is:
CHAR_LENGTH() or CHARACTER_LENGTH()	The number of characters in the argument.
CHAR()	The character represented by each integer in the argument.
CONCAT(*string1, string2,.. stringn*)	The concatenation of several strings.
CONCAT_WS(separator, *string1, string2, ..stringn*)	The concatenation of several strings with *separator* between them.
FIELD()	The position of the first argument in the remaining arguments.
FORMAT(*value, places*)	*Value* formatted with comma separators to the number of *places*.
INSERT(*string1, position, length, string2*)	*Length* characters of *string2* placed at *position* in *string1*.
INSTR(*string1, string2*)	The position of *string2* in *string1*.
LEFT(*string, length*)	The leftmost *length* characters in *string*.
LCASE() or LOWER()	All characters changed to lowercase.
LOCATE(*string1, string2*[, *position*]) or POSITION(*string1* IN *string2*)	The position of *string 1* in *string2*, optionally after *position*.
LTRIM(string)	*String* with any leading spaces removed.
QUOTE(*string*)	*String* enclosed in single quote marks; place a backslash before backslashes and single quote marks to escape them.
REPEAT(*string, count*)	*String* repeated *count* times.
REPLACE(*string, find, replace*)	*String* with all occurrences of *find* (which is case sensitive) replaced with *replace*.
REVERSE(*string*)	*String* with the order of the characters reversed.
RIGHT(*string, length*)	The rightmost *length* characters in *string*.
RTRIM(*string*)	*String* with trailing spaces removed.
SPACE(*length*)	A string of *length* spaces.
SUBSTR(*string, position*[, *length*]) or SUBSTRING(*string, position*[, *length*]) or MID(*string, position*[, *length*])	A substring formed from *string* beginning at *position* and optionally of length *length* characters.
TRIM([BOTH | LEADING | TRAILING [*character*] FROM] *string*)	*String* with both leading and trailing spaces removed. Optionally, you can specify *character* to be removed in place of spaces, and you can specify that spaces or *character* are removed from leading, trailing, or both positions. If *character* is specified, then FROM is needed in front of *string*.
UCASE() or UPPER()	All characters changed to uppercase.

Table 10-11 MySQL String Functions

Function	Result Is:
CHARSET()	The character set of the argument
COLLATION()	The collation of the string argument
CURRENT_USER() or USER()	The user name and host name
DATABASE() or SCHEMA()	The name of the current database
DEFAULT()	The default value for a column
FOUND_ROWS()	The number of rows that would be returned if there were no LIMIT clause in a SELECT
LAST_INSERT_ID()	The value of the AUTOINCREMENT column for the last INSERT
ROW_COUNT()	The number of rows updated
VALUES()	The values to be inserted into a table with INSERT
VERSION()	The MySQL version

Table 10-12 MySQL System and Information Functions

System and Information Functions

System and information functions allow you to get information about the MySQL environment and the elements being used. Table 10-12 lists some of the applicable functions.

Examples of system and information functions are

- `CHARSET('something')` returns "utf8"
- `DATABASE()` returns "books"
- `SELECT SQL_CALC_FOUND_ROWS * FROM books WHERE Price > 5.00 LIMIT 2;`
  `SELECT FOUND_ROWS();` returns 4. `SQL_CALC_FOUND_ROWS` requests that MySQL keep track of the total number of rows found so that `FOUND_ROWS()` can report it.
- `INSERT INTO books (title, author, publisher, price, category) VALUES('The Crystal City', 'Card', 'Thor', '7.99', 'Sci. Fic. ');` demonstrates the use of the `VALUES()` function
- `LAST_INSERT_id()` returns 10 after the `INSERT` statement earlier
- `ROW_COUNT()` returns 1 after the `INSERT` statement earlier

This chapter has introduced MySQL and discussed many of the supporting elements to MySQL statements, including operators and functions. In the next chapter, we'll look in detail at the commands and keywords that are central to MySQL statements. Then we'll look at how to put together and use complete statements. This will then lead to Chapter 12, where we will bring PHP back into the picture to utilize the MySQL statements.

Try This 10-1 Create and Use a MySQL Database

Using either the MySQL command line or phpMyAdmin (phpMyAdmin is probably easier) and the examples in this chapter:

1. Build a small database with one table and four fields, including an index, a name, a dollar amount, and a password. Insert four or five records into your table.
2. Do a query or two, for example: dues < 25, names alphabetical order.
3. In phpMyAdmin, write and execute a statement that generates a random number, multiplies it by 10, and rounds it to produce an integer.
4. Select a name from your database with a name like "Sal" using a wildcard.
5. Generate the current date and time and then format it.
6. Insert a new person in your database that includes an encrypted password. Then retrieve the record and show the password.

Chapter 10 Self-Test

The following questions are intended to help reinforce your comprehension of the concepts covered in this chapter. The answers can be found in the accompanying online Appendix A, "Answers to the Self-Tests."

1. What are three tools that can be used to work directly with MySQL other than programmatically with PHP or other languages?
2. What is SQL and how does it relate to MySQL?
3. How should commands and keywords be presented, and is this mandatory?
4. What are at least four parts of the MySQL language and what do they do?
5. What are the three types of literals?

6. What are at least three ways a date can be represented in a MySQL statement?
7. What are three types of operators with examples of them?
8. What are four types of functions with two examples of each?
9. What is the difference between `ROUND` and `TRUNCATE`?
10. What are the two wildcard characters that can be used in MySQL expressions and what do they mean?
11. What formatting characters would I use to get a date that looks like May 15th, 2014?
12. What field types should be used to store passwords and encrypted fields?

Key Skills & Concepts

- Understanding the MySQL Command Structure
- Preparing the MySQL Workbench
- Creating and Using a Database
- Creating a Table and Its Columns
- Reviewing MySQL Data Types
- Working with Data
- Altering Tables and Databases
- Renaming Tables
- Dropping Tables and Databases
- Using Events, Views, and Triggers

In previous chapters you have learned a bit about relational databases, SQL, and MySQL as well as some of the tools that you can use to work with and create a database with them. Other than in some examples where they were required to demonstrate other elements, MySQL commands and keywords have been ignored. In this chapter, we'll discuss and demonstrate commands and keywords and the statements that marry them together, along with operators and functions, to provide the full power of MySQL. We'll briefly revisit MySQL data types, make extensive use of the MySQL Workbench, and, to a lesser degree, phpMyAdmin and the MySQL command line to demonstrate how MySQL statements, their commands, and keywords operate.

MySQL Commands

MySQL commands are declarative words in MySQL statements that identify an action that is to take place. For example, the words `CREATE`, `INSERT`, `SELECT`, and `UPDATE` are all MySQL commands that, respectively,

- Create databases and tables and other elements
- Insert information into a table

- Select information from a table
- Update information in a table

MySQL commands can be grouped into the types of statements that use them:

- **Data definition statements** that work with databases, tables, and other elements and include the commands `ALTER`, `CREATE`, and `DROP`
- **Data manipulation statements** that work with the information in tables and include the commands `DELETE`, `INSERT`, and `SELECT`
- **Other statements** that are transactional and administrative in nature and include the commands `LOCK TABLES`, `EXECUTE`, and `SHOW`

In this section we'll review a number of the more frequently used commands, but, as I've said elsewhere in this book, if you don't see a command here that you need, go to the MySQL Manual that was described in Chapter 10.

Understanding the MySQL Command Structure

The MySQL command structure is set up to facilitate the creation and use of a relational database. To do that, it provides for the:

- Creation of a database
- Creation of tables within that database
- Identifying columns and their type, size, and characteristics within a table
- Inserting information into a row of columns
- Indexing the information in a database
- Altering and deleting the columns, tables, and databases
- Replacing and updating information in the table
- Selecting the information in the database and acting on the results
- Joining multiple tables and working with them together

Commands define MySQL statements and are generally the leading word. The remainder of the statement contains keywords, functions, and operators in clauses that modify the command. For example, the statement `SELECT NOW();` uses the command `SELECT` (probably the most commonly used command) and the function `NOW()` to return the current date and time. You've seen a more complex example in earlier chapters:

```
SELECT * FROM books WHERE price > 5.00;
```

This statement again uses the command `SELECT`, followed by:

- An asterisk (`*`) to indicate that all columns or fields ("columns" and "fields" are synonymous) are to be selected
- The keyword `FROM`
- The table name `books`
- The keyword `WHERE`
- The field name `price`
- The operator `>`
- The literal `5.00`

To repeat what I said in previous chapters, while there are a few exceptions where it isn't needed, most statements need a semicolon (`;`) at the end. Also, while it isn't needed at all, a good practice for human readability that is generally followed is to put commands and keywords in all capital letters and to put database, table, column, and other object names in lowercase or leading caps. Barring my own fallibility, which is reasonably likely, and that of the amazing technical editor, which is not very likely, all statements in this book will end with a semicolon, all commands and keywords will be in all caps, and all names will be lowercase.

Most keywords are aligned to certain commands and will be discussed with that command. In the next set of sections, we'll discuss the most commonly used commands along with their keywords in the following order, which I believe is how they are most commonly used:

- `CREATE` databases and tables
- `INSERT` data into tables
- `SELECT` data from tables
- `REPLACE` data in tables
- `UPDATE` data in tables
- `DELETE` data in tables
- `ALTER` tables and databases
- `RENAME` tables
- `DROP` tables and databases
- `CREATE` and use events, views, and triggers

Preparing the MySQL Workbench

In previous chapters we've relied heavily on phpMyAdmin, and to a lesser extent, on the MySQL command line to demonstrate what was being discussed. In this chapter we'll use the MySQL Workbench for that purpose. After downloading and installing MySQL Workbench, as described in Chapter 10, it should have left you with an icon on your desktop, as shown on the right. Use that icon to start MySQL Workbench and the following steps to prepare it to demonstrate the discussions in this chapter.

1. In the MySQL Workbench Home screen, click Database in the toolbar and then click Connect To Database to open a dialog box.
2. Leave Default Schema blank and click OK. (In MySQL Workbench, "schema" is synonymous with "database.")
3. In the Query1 screen that has opened, run your mouse around to the various menus and icons, as shown in Figure 11-1, and read the tooltips to get a feeling for what you can do.

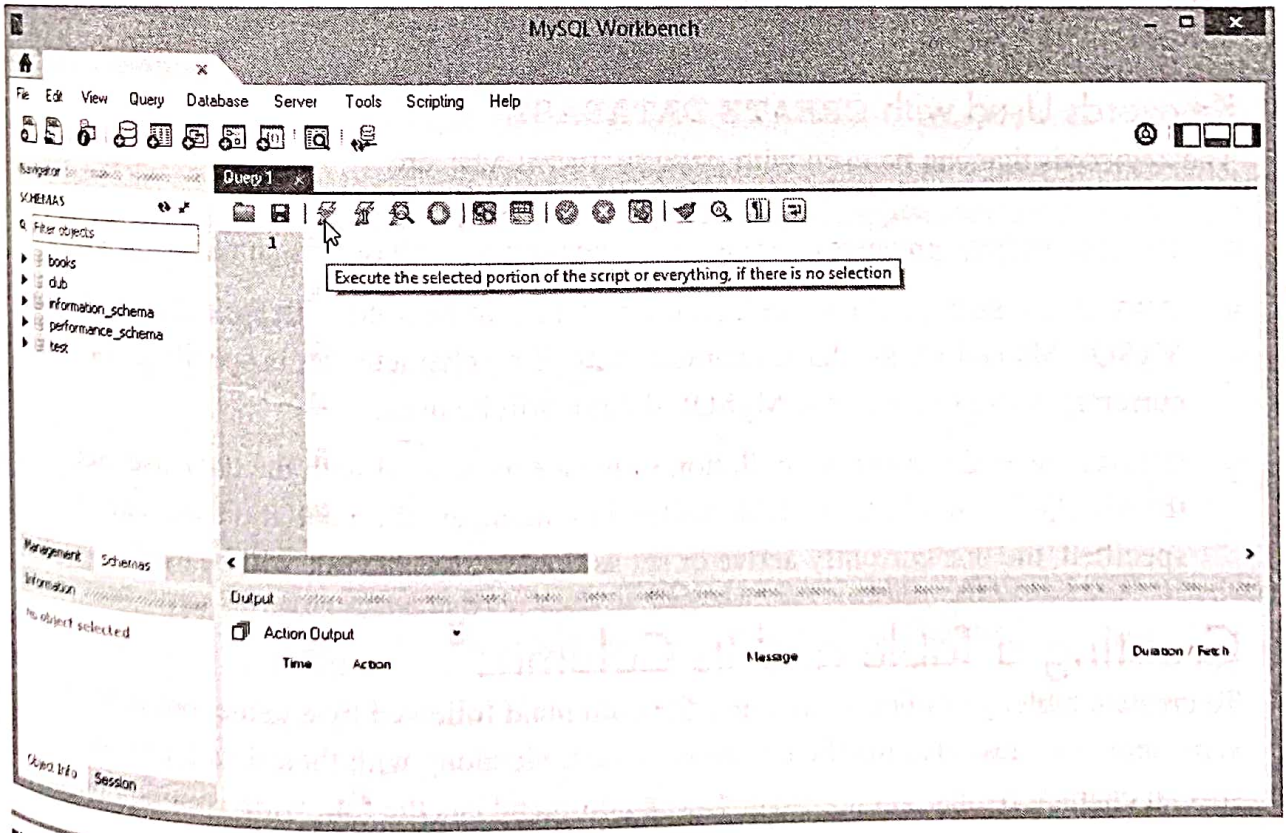


Figure 11-1 MySQL Workbench provides a Query window in which it is easy to try out MySQL statements with existing databases and tables.

In this chapter we'll give the Books database a rest and create a new database named "shop" with initially a single table named "sales." The table will initially have five columns named "id," "ondate," "buyer," "product," and "amount." Later in the chapter we'll add a "tax" column.

Create Databases and Tables

In previous chapters we have created databases and tables using the commands and tools built into phpMyAdmin or MySQL Workbench. Since you only infrequently need to do this, it makes sense to use these tools. MySQL commands, though, can directly create databases and tables, and it is informative to understand how this is done. The MySQL Workbench makes this easy.

Creating and Using a Database

To create a database, you simply enter the command and give the database a name. Then you must tell MySQL to use that database with the statements that follow. The `CREATE DATABASE` and `USE DATABASE` commands have the following common formats:

```
CREATE DATABASE database_name;
USE database_name;
```

Keywords Used with `CREATE DATABASE`

The keywords that can be used with `CREATE DATABASE` are

- `IF NOT EXISTS` prevents creating the database if a database of that name already exists.
- `CHARACTER SET` specifies a default character set to be used with the database. See the MySQL Manual for available character sets. If no character set is specified, the one currently active or set as the MySQL default will be used.
- `COLLATE` specifies a default collation sequence to be used with the database. See the MySQL Manual for available collation sequences. If no collation sequence is specified, the one currently active or set as the MySQL default will be used.

Creating a Table and Its Columns

To create a table, you once again enter the command followed by a name, but in the same statement you must also list the columns in the table along with their data types, size, and special characteristics. The `CREATE TABLE` command has the following common format:

```
CREATE TABLE table_name (column1_name type(size) characteristics,...
columnn_name type(size) characteristics);
```

Keywords Used with CREATE TABLE

There are three common keywords (or phrases) that you can use with CREATE TABLE:

- IF NOT EXISTS prevents creating the table if a table of that name already exists in the database.
- LIKE *old_table_name* creates a new table with the same column names, types, sizes, and characteristics as another table.
- TEMPORARY creates a table that only exists for the current user and is automatically dropped when that user drops their connection.

Keywords Used with Column Characteristics

As you see in the example table discussed here, column characteristics have their own set of keywords. Among the most common are (many keywords can only be used with specific data types):

- AUTO_INCREMENT automatically increments an integer or floating point data type by 1. There can be only one AUTO_INCREMENT column in a table and it cannot have a DEFAULT value.
- CURRENT TIMESTAMP adds the time and date to a TIMESTAMP column.
- DEFAULT provides a default value that immediately follows the keyword. It must be a constant (not an expression or a variable, so you cannot use NOW() or CURRENT_DATE() as a default for a date column, although you can use DEFAULT to add CURRENT_TIMESTAMP to a TIMESTAMP column) and cannot be used with BLOB and TEXT data types.
- FOREIGN KEY specifies the column contains a link to another table's PRIMARY KEY.
- NOT_NULL specifies that the column must contain a value.
- NULL specifies that a column does not need to have a value. If a column does not have NOT_NULL, then MySQL assumes that it is NULL and does not need to have a value.
- ON_UPDATE takes some action, such as inserting the CURRENT_TIMESTAMP, when the record is updated.
- PRIMARY KEY specifies the column is the index for the table and each value is, by definition, both UNIQUE and NOT NULL.
- UNIQUE cannot duplicate any other entry in the column.

Column Name	Data Type	Size	Special Characteristics
id	INTEGER (INT)	11	PRIMARY KEY, AUTO_INCREMENT
ondate	TIMESTAMP		
buyer	VARCHAR	40	
product	VARCHAR	40	
amount	DECIMAL	10,2	

Table 11-1 Columns in the Sales Table

NOTE

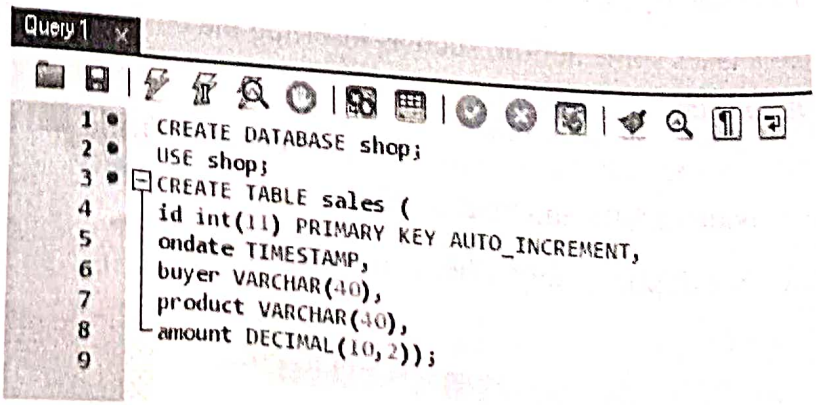
The TIMESTAMP data type by itself automatically inserts the current date and time when a new record is created and it automatically updates the field when the record is updated. This is the same as having both DEFAULT CURRENT_TIMESTAMP and ON_UPDATE CURRENT_TIMESTAMP. If you have only DEFAULT CURRENT_TIMESTAMP the current date and time is added when the record is created but is not updated. If you have only ON_UPDATE CURRENT_TIMESTAMP the current date and time is added when the record is updated but is not when it is created.

Create an Example Table

For the example sales table we're building here, the columns, data types, sizes, and characteristics are shown in Table 11-1.

The id is the index for the table, so you tell MySQL it is the PRIMARY KEY. Also, you want MySQL to automatically create the id by incrementing the previous id in the table with AUTO_INCREMENT. The ondate column is really a date and time column of the TIMESTAMP type that is automatically created when the record is added or updated. You could also use DEFAULT CURRENT_TIMESTAMP or ON UPDATE CURRENT_TIMESTAMP if you want the date added only when the record is created or only upon update. The buyer and product columns use the VARCHAR type with a maximum size of 40 characters each. The amount column uses the DECIMAL type 10 characters long with 2 decimal characters.

To create the database and table, perform the following steps in a MySQL Workbench query.



1. On line 1, type `CREATE DATABASE shop;`.
2. On line 2, type `USE shop;`.
3. On line 3, and the needed additional lines, type
```
CREATE TABLE sales (
id INT(11) PRIMARY KEY AUTO_INCREMENT,
ondate TIMESTAMP,
buyer VARCHAR(40),
product VARCHAR(40),
amount DECIMAL(10,2));
```
4. Click Execute (the leftmost lightning bolt in the Query1 toolbar).

If there are no errors in typing, all three command statements will execute, as you can see in the Action Output panel at the bottom of the MySQL Workbench (see Figure 11-2). Also, if you click Refresh (two circular arrows) opposite Schemas on the left and then open the database, table, and column elements on the left, you can see the results of your work.

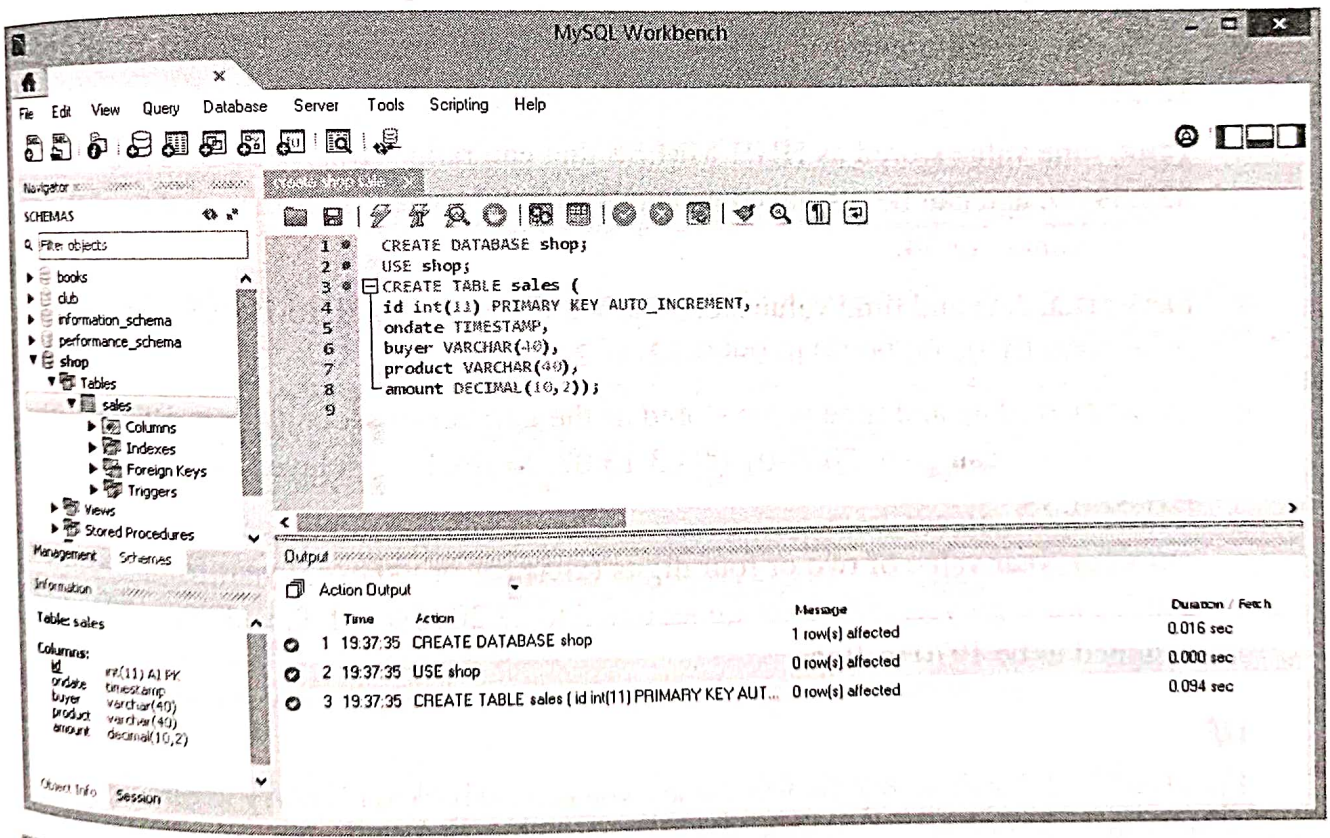


Figure 11-2 The MySQL Workbench provides a very useful set of panes with which to work.

Reviewing MySQL Data Types

Chapter 9 provides a detailed list of common column data types. Here, let's review the most frequently used types for you to keep in mind:

- `INT(n)`, integer number of up to *n* total digits.
- `DECIMAL(n,d)`, floating point number of up to *n* total digits, of which *d* are decimal digits.
- `DOUBLE(n,d)`, double-precision floating point number of *n* total digits, of which *d* are decimal digits.
- `CHAR(n)`, string value of a fixed maximum number of *n* characters padded with blanks on the right, but removed when retrieved.
- `VARCHAR(n)`, string value of a variable number of characters up to a maximum of *n*, not padded.
- `TEXT`, string value of a variable number of characters up to a maximum of 65,535.
- `BLOB`, binary number of up to a maximum of 65,535 bytes.
- `DATE`, date value stored as YYYY-MM-DD that can range from 1000-01-01 to 9999-12-31.
- `TIME`, time value stored as HHH:MM:SS that can range from –838:59:59 to 838:59:59, and can be displayed as D HH:MM:SS, where D is the number of days up to a maximum of 34.
- `DATETIME`, date and time value stored as YYYY-MM-DD HH:MM:SS that can range from 1000-01-01 00:00:00 to 9999-12-31 23:59:59.
- `TIMESTAMP`, date and time value stored as the number of seconds from 1970-01-01 00:00:01 and can go to 2038-01-09 03:13:07. Normally displayed as YYYY-MM-DD HH:MM:SS.
- `YEAR(n)`, year value of two or four digits (`YEAR(2)` or `YEAR(4)`) year from 1901 to 2155. Two-digit years 00 to 69 are assumed to be 2000 to 2069, and years 70 to 99 are assumed to be 1970 to 1999.

TIP

If you leave and shut down MySQL Workbench, you can get back to the query screen by opening the program, clicking Database in the menu bar, clicking Connect to Database, clicking Default Schema, typing **shop**, and clicking OK.

Insert Data into Tables

The next step is to load data into a table using the INSERT command, which creates a new row or record ("row" and "record" are synonymous) in the table. INSERT adds a new row at the end of the existing rows in a table. Instead of INSERT, you can use REPLACE, which will overwrite an existing row if it already exists; otherwise, it will perform like a standard INSERT. See "Replacing Data" later in this chapter.

Inserting Data

To insert data, you enter the command followed optionally, but traditionally, by the keyword INTO, then the table name, the list of column names that will receive values, the keyword VALUES or VALUE, and one or more sets of values. The INSERT command can use one of the following three options:

```
INSERT INTO table_name (column1_name,... columnn_name) VALUES (column1_value1,...
columnn_value1),...(column1_valuen,... columnn_valuen);
```

or

```
INSERT INTO table_name SET column1_name = value1,... columnn_name = valuen;
```

or

```
INSERT INTO table_name (column1_name,... columnn_name) SELECT...;
```

The first format is the most common, although the second is also frequently used. Both of the first two options insert explicit values in specified columns. The third option uses the SELECT command to select values from other table(s). (See "Selecting Data" later in this chapter.) In the first and third option, a value must be provided for each named column. If you do not name the columns, a value must be provided for all columns. If you name the columns and leave one or more columns unnamed, the unnamed columns must have a default value or be automatically set.

Keywords Used with INSERT

The common keywords (or phrases) that can be used with INSERT are

- DEFAULT sets a named column to its default value, which is automatically done if you do not name the column. You can also use DEFAULT(*column_name*) with the same result.
- DELAYED puts the inserted values into a buffer and waits until the table is not in use to add the rows.