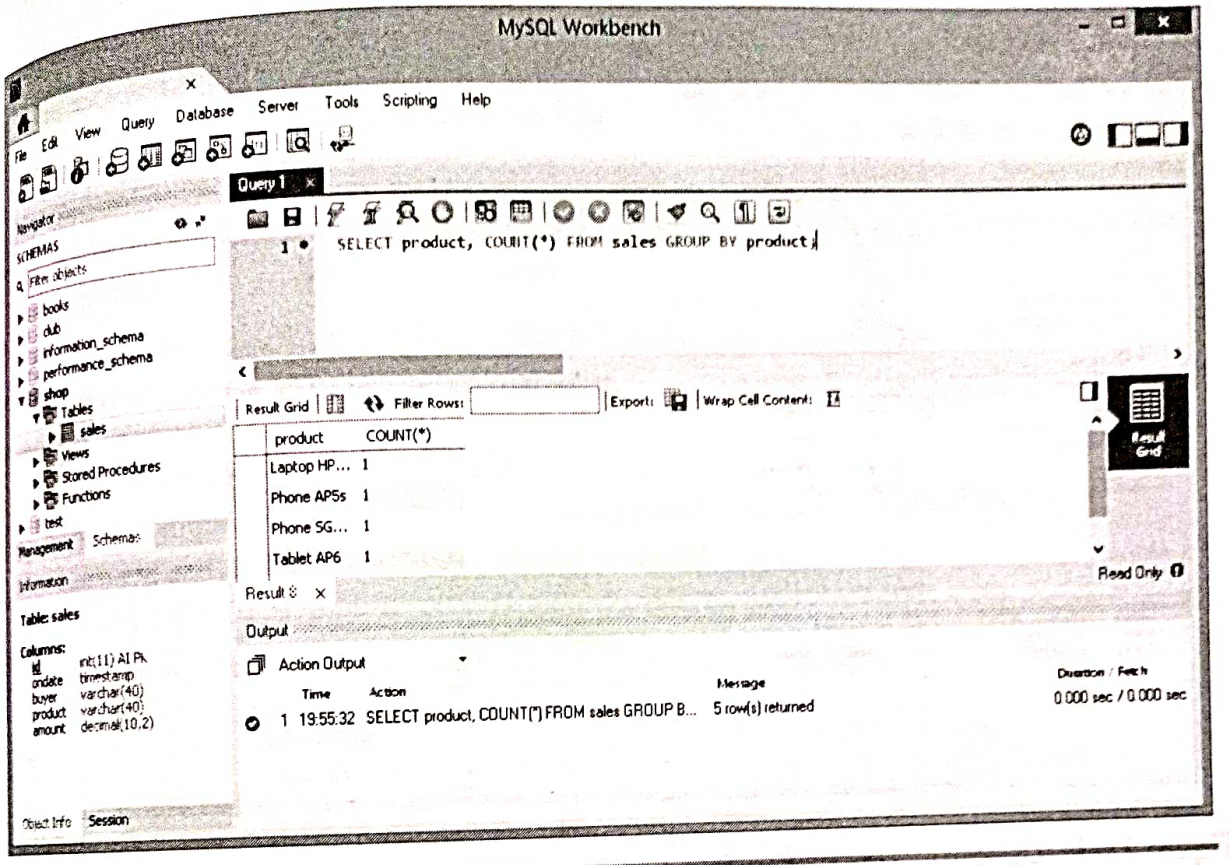


Figure 11-6 Using COUNT and GROUP BY

The third example demonstrates the use of HAVING and LIKE in place of WHERE, although you can use LIKE with WHERE just as well. Figure 11-7 shows the result after clicking Execute.

```
SELECT buyer, product FROM sales HAVING product LIKE 'Tablet%';
```

Replacing Data

The REPLACE command is almost exactly like the INSERT command—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 REPLACE command can use one of the following three options:

```
REPLACE INTO table_name (column1_name,... columnn_name) VALUES (column1_
value1,... columnn_value1),...(column1_valuen,... columnn_valuen);
```

or

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

or

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

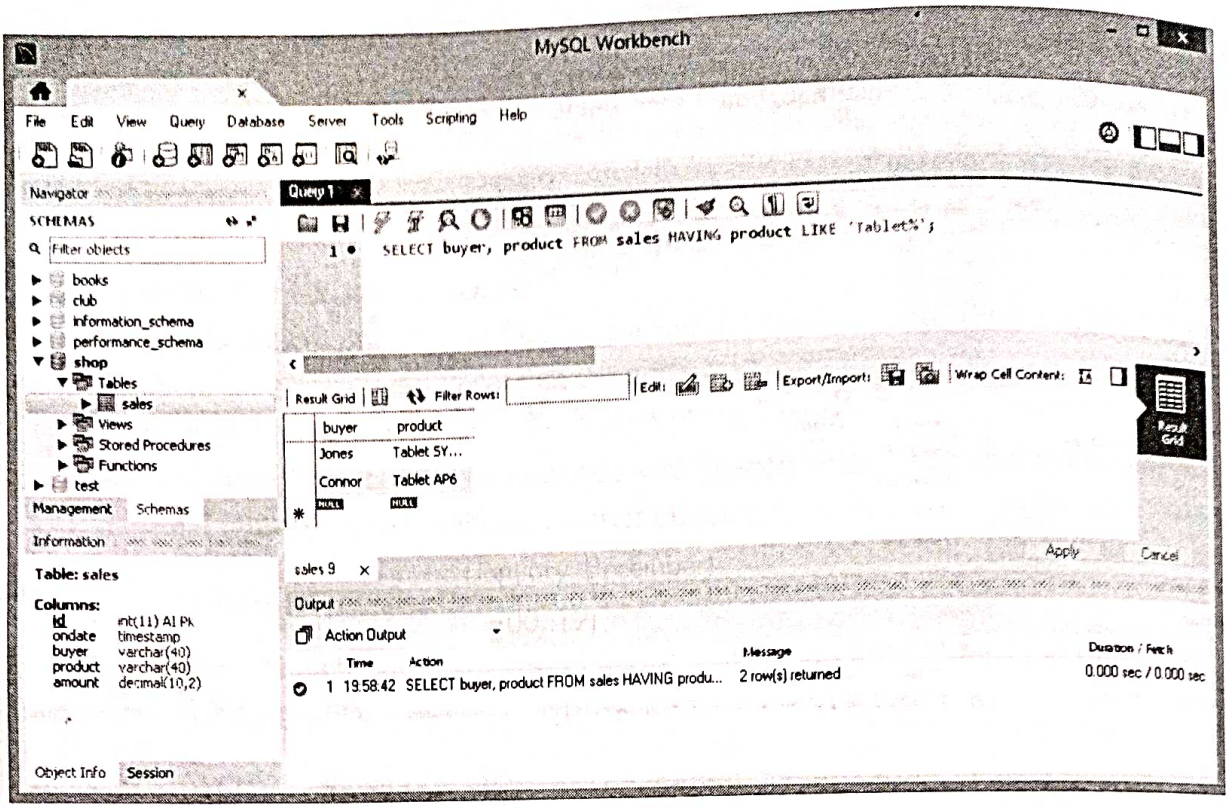


Figure 11-7 Using `HAVING` and `LIKE`

The first and second options, which are the most common, insert explicit values in specified columns. The third option uses the `SELECT` command to select values from other table(s). (See "Selecting Data" earlier in this chapter.) In the first and third options, 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.

With `REPLACE` you include either the primary key or unique index in the list of column names and in the list of values. If there is no match with an existing primary key or unique index value, then the `REPLACE` acts exactly like `INSERT` and a new record is added to the table. If there is a match with an existing primary key or unique index value, then the existing record is deleted and its replacement is added to the table.

One `REPLACE` statement can have multiple rows of values, each row enclosed in parentheses and separated by commas.

Keywords Used with `REPLACE`

The common keywords (or phrases) that can be used with `REPLACE` 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.

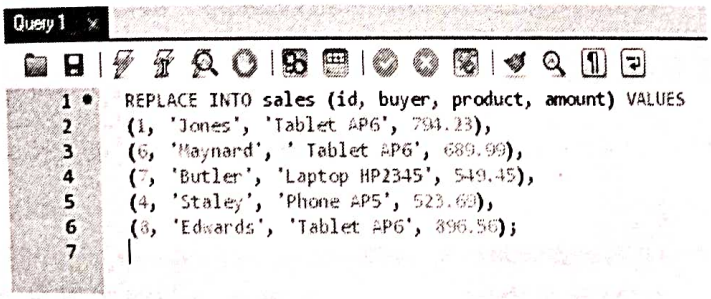
- DELAYED puts the replaced values into a buffer and waits until the table is not in use to add the rows.
- INTO is an optional keyword for readability.
- LOW PRIORITY waits to replace the rows and prevents the client from doing anything else until no one is reading from the table. This may be a long time. In contrast, DELAYED lets the client go on about their work.
- VALUES or VALUE, which are synonymous, provides a list of values for, and in the order of, the columns in the columns list.

Replace in the Example Table

For the example sales table, the data to be replaced is shown in Table 11-3.

The id and the ondate columns are automatically created when the record is added.

Perform the following steps to replace the data in the sales table of the shop database in a MySQL Workbench query.



1. On line 1 and the needed additional lines, type:

```
REPLACE INTO sales (id, buyer, product, amount) VALUES
(1, 'Jones', 'Tablet AP6', 794.23),
(6, 'Maynard', ' Tablet AP6', 689.99),
(7, 'Butler', 'Laptop HP2345', 549.45),
(4, 'Staley', 'Phone AP5', 523.69),
(8, 'Edwards', 'Tablet AP6', 896.56);
```

2. Click Execute.

D	Buyer	Product	Amount
1	Jones	Tablet AP6	794.23
6	Maynard	Tablet AP6	689.99
7	Nelson	Laptop HP2345	549.45
4	Staley	Phone AP5s	523.69
8	Edwards	Tablet AP 6	896.56

Table 11-3 Data to Be Replaced in the Sales Table

The REPLACE command statement will execute, as you can see in the Action Output panel at the bottom of the MySQL Workbench. Also, if you right-click the sales table and click Select Rows you can see the rows you replaced (see Figure 11-8).

Updating Data

To update data, you enter the command followed by the columns to update from, the keyword FROM, the table name, optionally the keyword WHERE and a conditional expression, and optionally the keyword ORDER BY and a column name. The UPDATE command takes the following general form with several of its optional keywords:

```
UPDATE table_name SET column1_name = value1,... columnn_name = valuen
WHERE conditional_expression ORDER BY column_name;
```

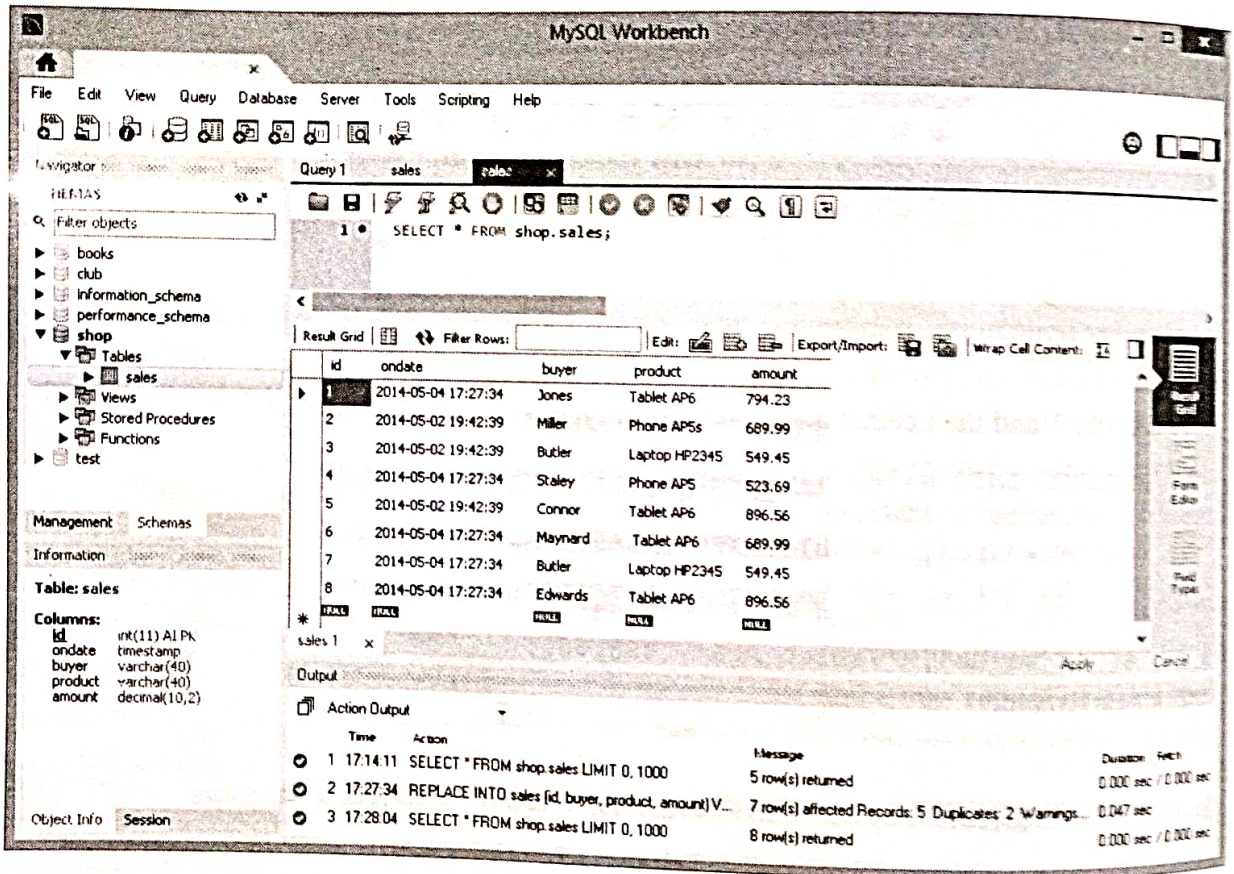


Figure 11-8 Action Output shows the rows replaced.

The first "table name" identifies the table with the columns to be updated. SET identifies one or more columns that are to be updated with the values that follow them. WHERE precedes a conditional expression that selects the rows to be displayed.

Keywords Used with UPDATE

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

- DEFAULT sets a named column to its default value, which is automatically done if you do not name the column.
- IGNORE prevents UPDATE from aborting even if there is an error.
- LIMIT limits the number of rows that are returned. A single number following LIMIT returns that number of rows starting with the first row updated. Two numbers following LIMIT, such as LIMIT a, b, returns b rows beginning with the a^{th} row. There is a hitch, though. Row numbers begin at 0, so LIMIT 1, 4 returns four rows beginning with the *second* row.
- LOW PRIORITY waits to update the rows and prevents the client from doing anything else until no one is reading from the table.
- ORDER BY precedes the column or columns used to specify the order in which the rows are updated.
- SET is used to identify the columns and their new values that will update the values in the existing table.
- WHERE provides the conditional expression that must be satisfied (be TRUE) for a row to be updated.

CAUTION

With no WHERE clause, all rows are updated.

Update the Example Table

UPDATE can be used for both surgical one-off corrections and mass corrections of a number of rows. Here we'll look at both of these using the shop database and sales table.

The first UPDATE example gives everybody who purchased on the second of the month a 5 percent discount:

```
UPDATE sales SET amount=amount*.95 WHERE DAYOFMONTH(ondate)=2;
```

When you enter and execute this statement in a newly installed MySQL Workbench, you get an error message that says you are operating in Safe mode and doing an update

without referencing the primary key in the WHERE clause, which can have negative consequences, although not in this case. The suggested solution is to change the default with these steps.

1. In MySQL Workbench, click the Edit menu, click Preferences | SQL Queries | "Safe Updates" to clear the default check mark.
2. Click OK and then retry to execute the UPDATE statement.

The UPDATE should now complete, but you now get a warning for each of the dollar amounts that were changed because they are truncated at two decimal digits, as you see in Figure 11-9. You could prevent the warnings by putting SET amount=amount*.95 in the ROUND() function with two decimal digits, like this:

```
ROUND(amount=amount*.95, 2)
```

NOTE

No rows will change if you aren't doing these exercises on the second of the month!

NOTE

You can see the change in Figure 11-9 by comparing it to the dollar amounts in Figure 11-8 for Miller, Butler, and Connor.

The second UPDATE example updates a single field in a single record to correct an error.

```
UPDATE sales SET product='Phone AP5s' WHERE buyer='Staley';
```

	id	ondate	buyer	product	amount
	1	2014-05-...	Jones	Tablet AP6	794.23
	2	2014-05-...	Miller	Phone AP5s	655.49
	3	2014-05-...	Butler	Laptop HP...	521.98
	4	2014-05-...	Staley	Phone AP5s	523.69
	5	2014-05-...	Connor	Tablet AP6	851.73
	6	2014-05-...	Maynard	Tablet AP6	689.99
	7	2014-05-...	Butler	Laptop HP...	549.45
	8	2014-05-...	Edwards	Tablet AP6	896.56
*	NULL	NULL	NULL	NULL	NULL

Deleting Data

The DELETE statement deletes rows using FROM to name the table, optionally using WHERE with a conditional expression to select the rows, and optionally the keyword ORDER BY and

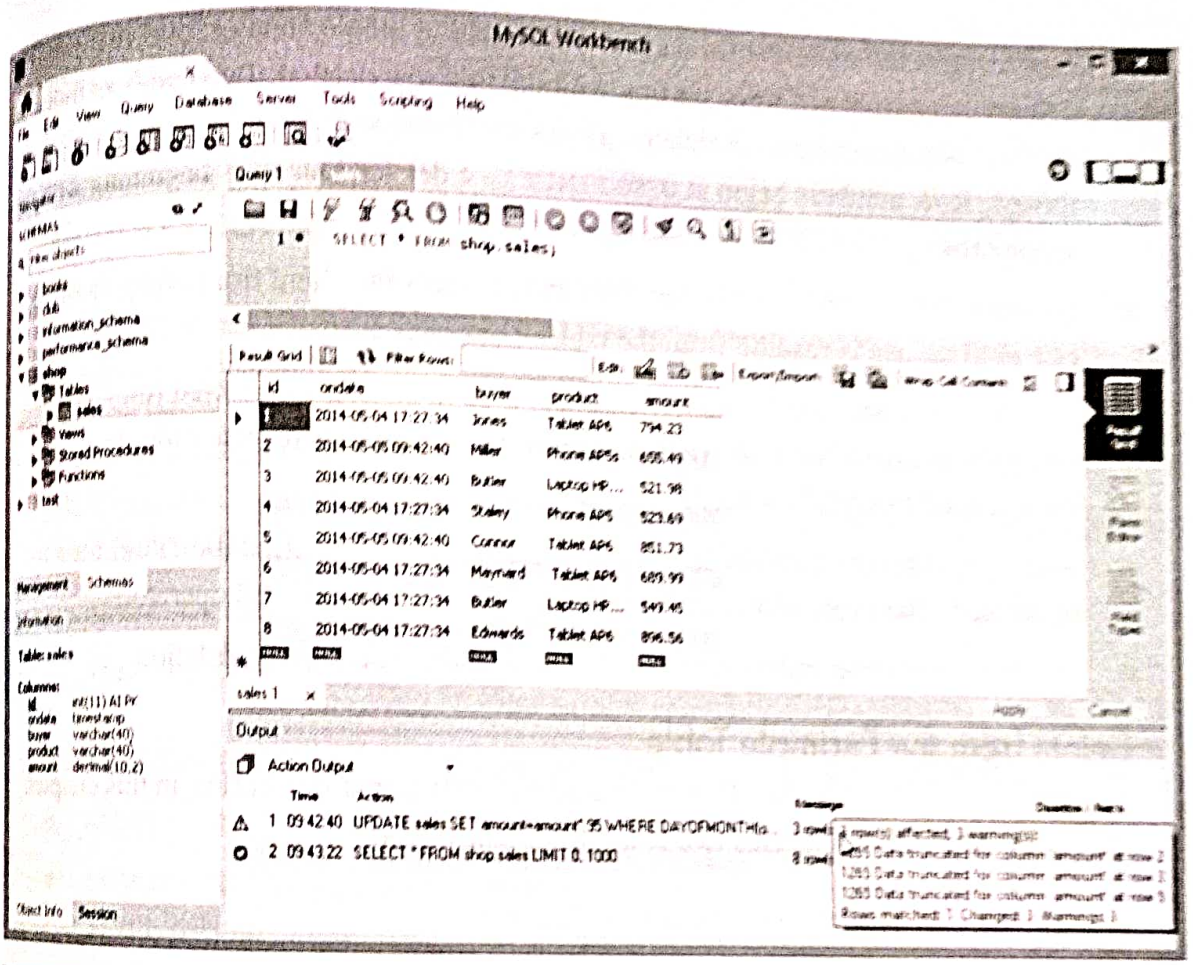


Figure 11-9 Doing an UPDATE and getting a truncation warning

a column name to specify the order in which the deletion occurs. You can also optionally use LIMIT to limit the deletion to a certain number of rows. The DELETE command takes the following general form with several of its optional keywords:

```
DELETE FROM table_name WHERE conditional_expression ORDER BY column_
name LIMIT number_of_ rows;
```

If there is not a WHERE clause, all rows are deleted.

Keywords Used with DELETE

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

- FROM precedes the table in which the deletion will occur.
- IGNORE prevents DELETE from aborting even if there is an error.

- `LIMIT` limits the number of rows that are deleted. A single number following `LIMIT` deletes that number of rows starting with the first row deleted. Two numbers following `LIMIT`, such as `LIMIT a, b`, deletes b rows beginning with the a^{th} row. There is a hitch, though. Row numbers begin at 0, so `LIMIT 1, 4` deletes four rows beginning with the *second* row.
- `LOW PRIORITY` waits to delete the rows and prevents the client from doing anything else until no one is reading from the table.
- `ORDER BY` precedes the column or columns used in sorting the deleted rows. Normal sorting is in ascending order (a to z, 0 to 9). You can sort in reverse order by adding the keyword `DESC` following the column name in `ORDER BY`.
- `WHERE` provides the conditional expression that must be satisfied (be `TRUE`) for a row to be deleted. The expression can contain conditional functions and operators.
- `QUICK` pauses some indexing operations to slightly speed up the deletion.

Delete from the Example Table

`DELETE` is very similar to the other command statements you saw earlier in this chapter. Here is an example that deletes two rows of the sales table:

```
DELETE FROM sales WHERE amount < 525.00;
```

NOTE

If your dates don't match mine, only one row is going to be deleted with this query because the other one didn't get the 5 percent price reduction.

	id	ondate	buyer	product	amount
	1	2014-05-...	Jones	Tablet AP6	794.23
	2	2014-05-...	Miller	Phone AP5s	655.49
	5	2014-05-...	Connor	Tablet AP6	851.73
	6	2014-05-...	Maynard	Tablet AP6	689.99
	7	2014-05-...	Butler	Laptop HP...	549.45
	8	2014-05-...	Edwards	Tablet AP6	896.56
*	NULL	NULL	NULL	NULL	NULL

Alter, Rename, and Drop Tables and Databases

In the previous section we talked about querying, changing, and deleting data within one or more tables. Here we'll talk about changing and deleting tables and databases themselves. While these actions are important and you will most likely need to use them

they are not generally part of your everyday use of MySQL like SELECT, REPLACE, and UPDATE are. As a result, I'll be briefer with these descriptions and you may need to use the MySQL Manual if you need to delve deeply into these commands.

Altering Tables and Databases

Altering tables and databases allows you to change the structure of your database. There is a lot of difference between what you can change in a table and in a database, so they will be discussed separately.

Altering Tables

Altering tables is really altering a table's columns, adding, changing, and dropping (deleting) columns. The general form of an ALTER TABLE statement is

```
ALTER TABLE table_name alteration ;
```

There are a number of types of alterations; among these are

- ADD COLUMN *column_name column_definition* FIRST or AFTER *column_name*
- ALTER COLUMN *column_name* SET DEFAULT *literal* or DROP DEFAULT
- CHANGE COLUMN *old_column_name new_column_name column_definition* FIRST or AFTER *column_name*
- DROP COLUMN *column_name*
- MODIFY COLUMN *column_name column_definition* FIRST or AFTER *column_name*
- RENAME TO *new_table_name*

NOTE

In the ALTER table alterations, the word "COLUMN" is optional and can be left out. I believe, though, that it helps in human understanding of what is being done, so I recommend that it be included.

The *column_name column_definition* clause is the same as that used in CREATE TABLE described earlier in this chapter. Also, the keyword IGNORE, described earlier, can be used with ALTER TABLE. MODIFY COLUMN is used to change only the *column_definition* including the data type.

The default is that new columns are added at the end after the last existing column. You can change this with the FIRST or AFTER *column_name* clause. FIRST will make the new column the first column; AFTER *column_name* places the new column after the named column. In the CHANGE and MODIFY keywords, this clause will reorder an existing column.

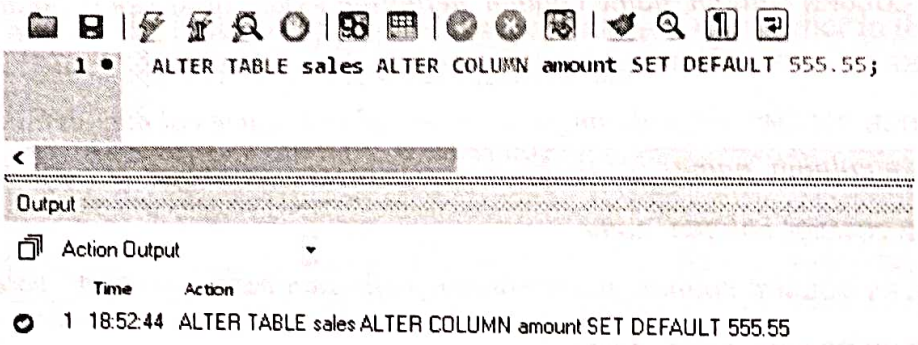
You can use several ADD, ALTER, CHANGE, and DROP alteration clauses in the same ALTER TABLE statement if you separate them with commas.

Here are three examples of ALTER TABLE command statements:

```
ALTER TABLE sales ADD COLUMN tax DECIMAL(10,2);
```

	id	ondate	buyer	product	amount	tax
	1	2014-05-...	Jones	Tablet AP6	794.23	NULL
	2	2014-05-...	Miller	Phone AP5s	655.49	NULL
	5	2014-05-...	Connor	Tablet AP6	851.73	NULL
	6	2014-05-...	Maynard	Tablet AP6	689.99	NULL
	7	2014-05-...	Butler	Laptop HP...	549.45	NULL
	8	2014-05-...	Edwards	Tablet AP6	896.56	NULL
*	NULL	NULL	NULL	NULL	NULL	NULL

```
ALTER TABLE sales ALTER COLUMN amount SET DEFAULT 555.55;
```




```
ALTER TABLE sales DROP COLUMN tax;
```

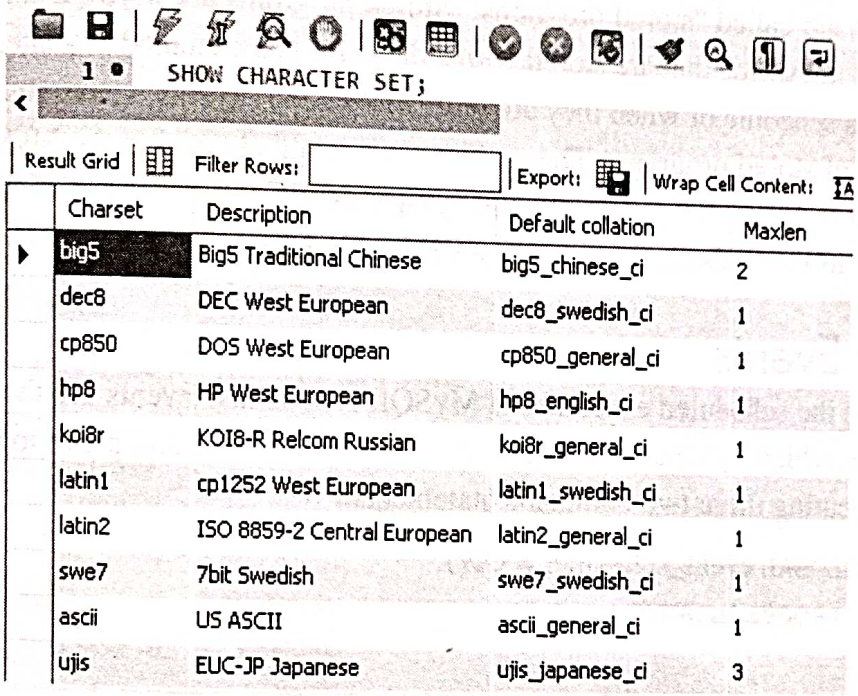
	id	ondate	buyer	product	amount
	1	2014-05-...	Jones	Tablet AP6	794.23
	2	2014-05-...	Miller	Phone AP5s	655.49
	5	2014-05-...	Connor	Tablet AP6	851.73
	6	2014-05-...	Maynard	Tablet AP6	689.99
	7	2014-05-...	Butler	Laptop HP...	549.45
	8	2014-05-...	Edwards	Tablet AP6	896.56
*	NULL	NULL	NULL	NULL	NULL

Altering Databases

ALTER DATABASE allows you to change the default character set and the collation sequence used in the database. The general form of an ALTER DATABASE statement is

ALTER DATABASE *database_name* CHARACTER SET = *character_set_name* or COLLATE = *collation_name*;

The commands SHOW CHARACTER SET; and SHOW COLLATE will provide lists of the character sets and collation sequences that you can use.



```
1 • SHOW CHARACTER SET;
```

Result Grid | Filter Rows: | Export: | Wrap Cell Content:

Charset	Description	Default collation	Maxlen
big5	Big5 Traditional Chinese	big5_chinese_ci	2
dec8	DEC West European	dec8_swedish_ci	1
cp850	DOS West European	cp850_general_ci	1
hp8	HP West European	hp8_english_ci	1
koi8r	KOI8-R Relcom Russian	koi8r_general_ci	1
latin1	cp1252 West European	latin1_swedish_ci	1
latin2	ISO 8859-2 Central European	latin2_general_ci	1
swe7	7bit Swedish	swe7_swedish_ci	1
ascii	US ASCII	ascii_general_ci	1
ujis	EUC-JP Japanese	ujis_japanese_ci	3

Renaming Tables

Only tables can be renamed—databases cannot. Also, while RENAME TABLE is being executed, no other session can access any of the tables in the RENAME. Finally, temporary tables cannot be renamed with RENAME TABLE, but you can use ALTER TABLE RENAME (see "Altering Tables" earlier in this chapter). The general form of an ALTER DATABASE statement is:

RENAME TABLE *table_name1* TO *new_table_name1*, . . . *table_namen* TO *new_table_namen*;

Dropping Tables and Databases

Dropping tables and databases removes or deletes them. This means that all data contained in the table or tables in the database are lost—they are permanently deleted.

DROP TABLE *table_name1*, . . . *table_namen*;
DROP DATABASE *database_name*;

The keyword (phrase) `IF EXISTS` can be used to make sure the table or database to be dropped exists. If it doesn't, the statement stops execution and doesn't produce an error message. Also, you can use the `TEMPORARY` keyword to drop a temporary table. These objects challenge the name "Beginner's Guide," but I think it is important to know that they exist.

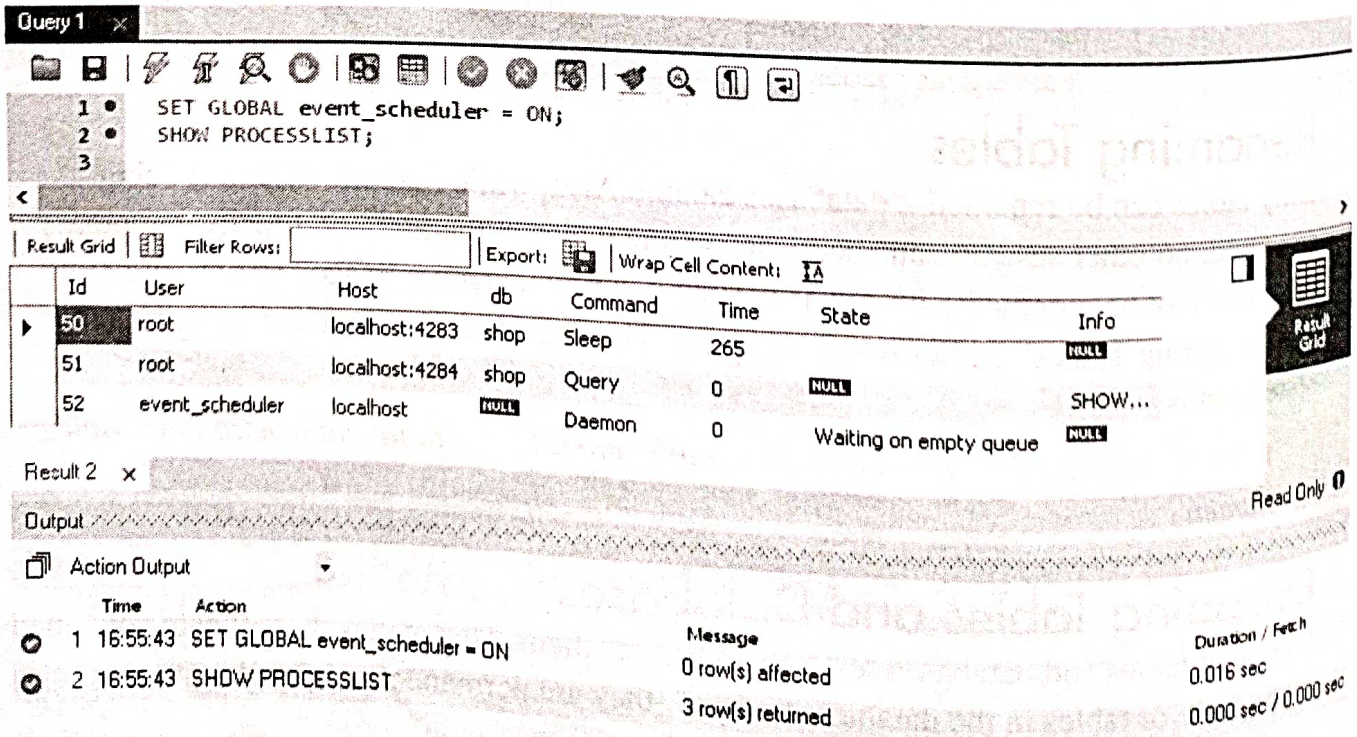
Create and Use Events, Views, and Triggers

MySQL supports a number of other objects, including events, views, and triggers that as a group are called "stored programs." Stored programs are MySQL statements and additional keywords that are stored with a database and executed at a later time, either based on a schedule or when they are otherwise called. Events, views, and triggers are examples of stored programs. These objects are relatively new in the MySQL syntax and are on the borderline of being a "Beginner's Guide" subject, I think, though they are important to know about. As a result, we'll briefly look at them here.

Using Events

Events are the scheduled execution of MySQL statements. Events use the MySQL Event Scheduler, which by default is turned off. You can turn it on and prove to yourself that it is on by executing these two command statements:

```
SET GLOBAL event_scheduler = ON;
SHOW PROCESSLIST;
```



When the Event Scheduler is turned off, it will not show up on the process list. Once turned on, you can turn it off with:

```
SET GLOBAL event_scheduler = OFF;
```

Events can be created, altered, and dropped.

Create Events

The general form of a CREATE EVENT statement is

```
CREATE EVENT event_name ON SCHEDULE schedule DO mysql_statement;
```

All parts of this CREATE EVENT statement are required. The schedule can be

- At a particular date and time timestamp

```
AT CURRENT_TIMESTAMP + INTERVAL 30 MINUTE
```

- On a periodic interval

```
EVERY 10 MINUTE
```

- On or within a start and/or end date and time

```
EVERT 10 MINUTES STARTS CURRENT_TIMESTAMP ENDS CURRENT_TIMESTAMP +
INTERVAL 30 MINUTE
```

NOTE

The time and date values can be in SECOND, MINUTE, HOUR, DAY, WEEK, MONTH, QUARTER, YEAR, and several combinations.

For the mysql_statement, you can use the majority of MySQL statements, but some SELECT and all SHOW statements that simply display information from a table and don't change it have no effect when used in an event, which doesn't display or save information.

Additional keywords that you can use with CREATE EVENT include

- IF NOT EXISTS prevents the execution of CREATE EVENT if an event of the same name already exists.
- ON COMPLETION PRESERVE, in the ON SCHEDULE clause, preserves the CREATE EVENT statement after it expires; otherwise, it is automatically and immediately dropped.
- COMMENT allows you to add a comment within quotes in the ON SCHEDULE clause.

An example of using an event is:

```
CREATE EVENT addtax ON SCHEDULE AT CURRENT_TIMESTAMP + INTERVAL 2 MINUTE
DO UPDATE sales SET tax = ROUND(amount*.092, 2);
```

This statement allows you to calculate the tax on a group of records on a time-delayed basis, as you can see in Figure 11-10.

Alter and Drop Events

`ALTER EVENT` allows you to make changes to an existing event. It has the following general form:

```
ALTER EVENT event_name ON SCHEDULE schedule DO mysql_statement;
```

This allows you to change any aspect of the event with the contents of the `ALTER EVENT` statement, including using the keyword `RENAME` to change the name of the event.

`DROP EVENT` allows you to delete an existing event. It has this general form:

```
DROP EVENT event_name;
```

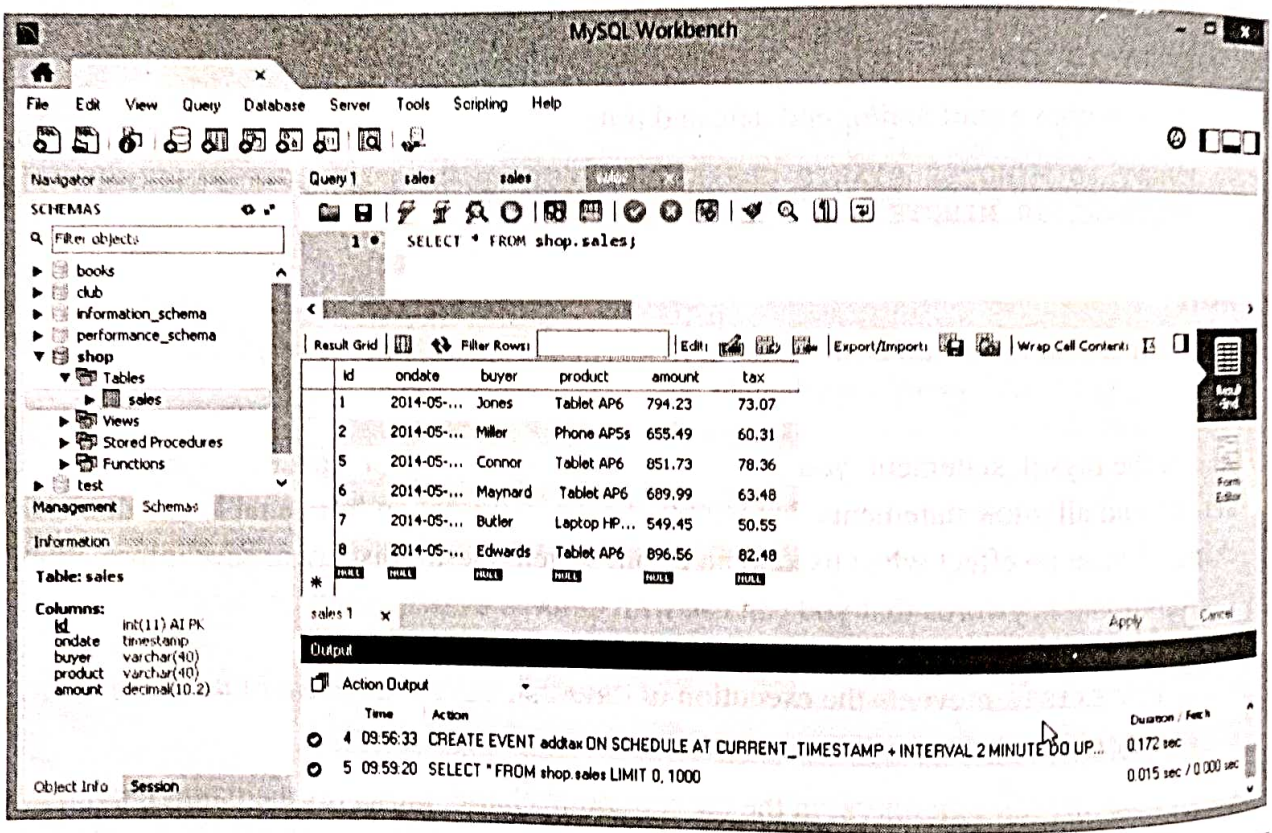


Figure 11-10 Delaying statement execution can be helpful on a heavily used site.

Using Views

Views are, in essence, named `SELECT` statements that you can repeatedly call using only the name without specifying the full `SELECT` statement.

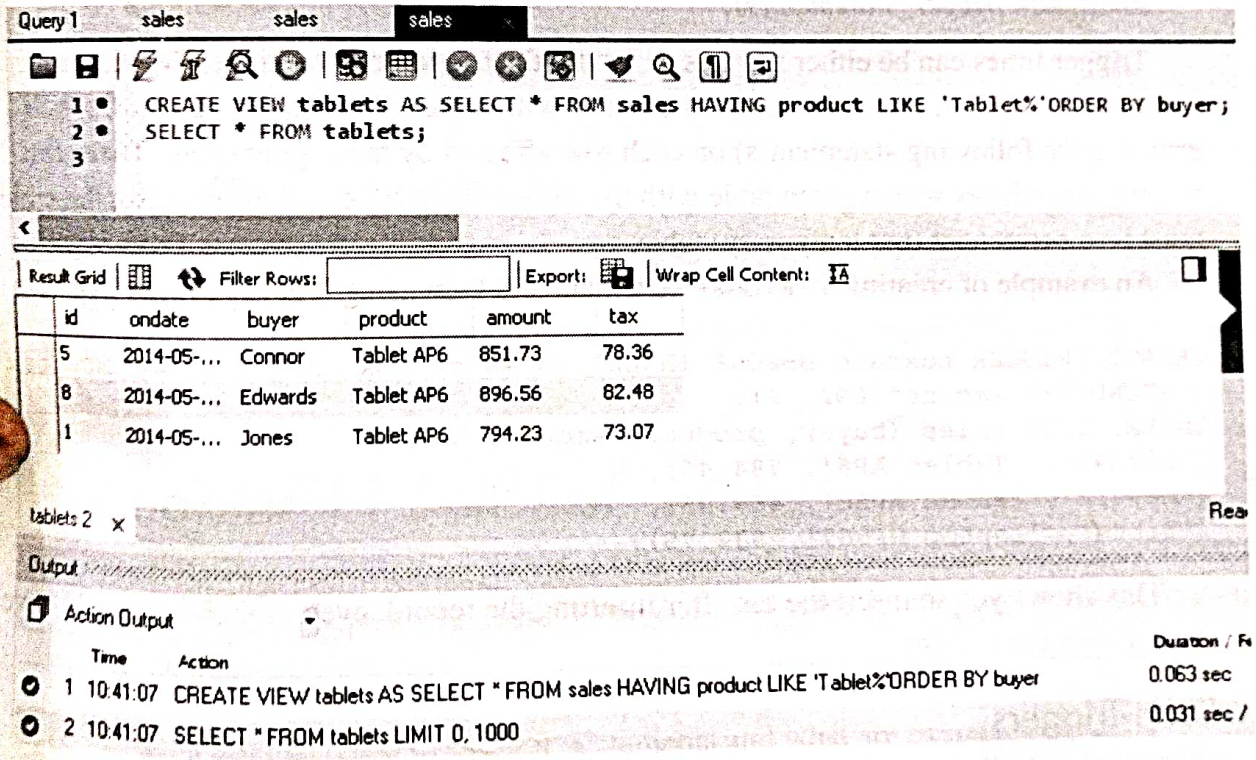
Create Views

The general form of a `CREATE VIEW` statement is

```
CREATE VIEW view_name AS select_statement;
```

An example of creating a `VIEW` statement and then using it is

```
CREATE VIEW tablets AS SELECT * FROM sales HAVING product LIKE
'Tablet%' ORDER BY buyer;
SELECT * FROM tablets;
```



Alter and Drop Views

`ALTER VIEW` allows you to make changes to an existing view. It has the following general form:

```
ALTER VIEW view_name AS select_statement;
```

This allows you to change any aspect of the view with the contents of the `ALTER VIEW` statement.

`DROP VIEW` allows you to delete an existing view. It has this general form:

```
DROP VIEW view_name;
```

Using Triggers

Triggers are one or more statements that are executed on each row of a particular table when an event occurs within the table.

Create Triggers

The general form of a `CREATE TRIGGER` statement is

```
CREATE TRIGGER trigger_name trigger_time trigger_event ON table_name FOR
EACH ROW trigger_statement(s);
```

Trigger times can be either `BEFORE` or `AFTER` the trigger event. Trigger events can be `INSERT`, `UPDATE`, or `DELETE` actions on rows within the named table. `FOR EACH ROW` executes the following statement(s) on each row affected by the trigger event. There can be only one trigger with a given table with the same trigger time and event. Also, the table must be a permanent (not temporary) table.

An example of creating a `TRIGGER` statement and then using it is

```
CREATE TRIGGER taxcalc BEFORE INSERT ON sales FOR EACH ROW SET NEW.tax
= ROUND(NEW.amount*.092, 2);
INSERT INTO sales (buyer, product, amount) VALUES
('Linden', 'Tablet AP6', 784.46),
('Meyers', 'Phone AP5c', 469.95),
('Bitts', 'Laptop HP567', 932.75);
```

This allows you to insert the tax after inserting the record, even though you could have done it with the `INSERT`.

Drop Triggers

`DROP TRIGGER` allows you to delete an existing trigger. It has this general form:

```
DROP TRIGGER trigger_name;
```

NOTE

Triggers cannot be altered.

Try This 11-1 Create and Use a Database with MySQL Workbench

Go through this chapter and create and test with MySQL Workbench your own example MySQL statements for each of the following (in MySQL Workbench, try several alternatives to your examples by making simple changes to the statement):

1. Create and use a database.
2. Create a table with various types of columns.
3. Insert data into your table.
4. Select data from your table.
5. Replace data in your table.
6. Update data in your table.
7. Delete data in your table.
8. Alter your table in several ways.
9. Drop a table.
10. Create and use an event.

Chapter 11 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 two types of MySQL statements and what are examples of each?
2. What are the major components of a MySQL statement?
3. What does an asterisk (*) stand for in a MySQL statement?
4. What is a literal?
5. What are the recommended rules to be used in writing MySQL statements?
6. Besides the command CREATE TABLE, what are the other elements in a CREATE TABLE statement?

Key Skills & Concepts

- Database Manipulation
- Database Information
- Database Administration
- Combining PHP MySQL Functions
- Expanding PHP Form Handling
- Facilitating User Interaction
- Interacting with a Database
- Improving Security

PHP and MySQL are like a teacup and a saucer—you can use either one without the other, but they are meant to be used together. Yes, there are other languages that can be used with MySQL and other databases that can be used with PHP, but I believe that PHP and MySQL is the most frequently used combination. That combination is, of course, the purpose of this book, and bringing the two together is the purpose of this chapter.

Bringing PHP and MySQL Together

Bringing PHP and MySQL together is relatively easy. Almost all web-hosting sites provide both PHP and MySQL, and you have Zend Server or WAMP (Windows servers for Apache, MySQL, and PHP) to test the pair on your computer. Creating web apps with MySQL and PHP is straightforward and not difficult. It is an extension of what you have already learned about PHP and MySQL.

PHP works with MySQL through a set of functions that perform the SQL commands and more. These functions can be split into three categories: database manipulation, database information, and database administration.

Database Manipulation

Manipulating an existing MySQL database with PHP can be broken into these steps:

- Connect to MySQL.
- Perform a query.
- Process the results.
- Release the results from memory.
- Disconnect from MySQL.

The last two steps are optional, but they are a good practice and become mandatory if you are working with multiple databases.

PHP's database manipulation functions for MySQL are the heavy lifters of the group, opening and closing the database, and querying and retrieving database contents, as shown in Table 12-1. There are detailed explanations of these functions in the following sections

Function	Description	Arguments and Comments
`mysqli_close()`	Disconnects from MySQL	Connection ID. Returns `TRUE` for success or `FALSE` for failure.
`mysqli_connect()`	Connects to MySQL	Host server name, username, password, database name. Returns the connection ID or `FALSE` for failure.
`mysqli_create_db()`	Creates a MySQL database	Name of `db` to be created. Optionally, the connection ID.
`mysqli_fetch_all()`	Gets all result rows as any type of array	Result of a query, type of array: `mysqli_assoc`, `mysqli_num`, or `mysqli_both` (the default).
`mysqli_fetch_array()`	Gets a row as any type of array	Result of a query, type of array: `mysqli_assoc`, `mysqli_num`, or `mysqli_both` (the default).
`mysqli_fetch_assoc()`	Gets a row as an associative array	Result of a query. Field names are the indexes.
`mysqli_fetch_object()`	Gets one or more fields in a row	Result of a query. Optionally, the class name and an array of parameters.
`mysqli_fetch_row()`	Gets a row as an enumerated array	Result of a query. Field numbers are the indexes, beginning with 0.
`mysqli_query()`	Queries a MySQL database	Connection ID. SQL query statement. Returns the result of the query or `TRUE` if successful, `FALSE` otherwise.

Table 12-1 Database Manipulation Functions

and many examples of their use in this and the following two chapters. Use Table 12-1 as a reference to come back to after you have seen the function explanations and examples of their use.

NOTE

Throughout this and the remaining chapters' discussion of the PHP MySQL functions, we will precede each function with `mysqli`, which is the newer "improved (i)" version of the PHP MySQL functions that uses the latest features of MySQL. There is an earlier set of functions preceded with `mysql` that has been deprecated as of PHP 5.5. There are also two styles of `mysqli` functions, an object-oriented style ("OOP" for object-oriented programming), and a procedural style. In line with the "Beginner's Guide" nature of this book, we will only discuss the procedural style. For further information about the object-oriented style, see php.net/manual/en/book.mysqli.php. For use in this book and for much of your work, the procedural style functions provide everything that is needed to create full-featured web apps.

NOTE

As with the rest of the book, there are many more PHP MySQL functions than can be discussed here. For the full list, see php.net/mysqli.

Connect to MySQL

Connecting to MySQL is really "logging into" a particular MySQL database server with a username, password, and database name. Like this:

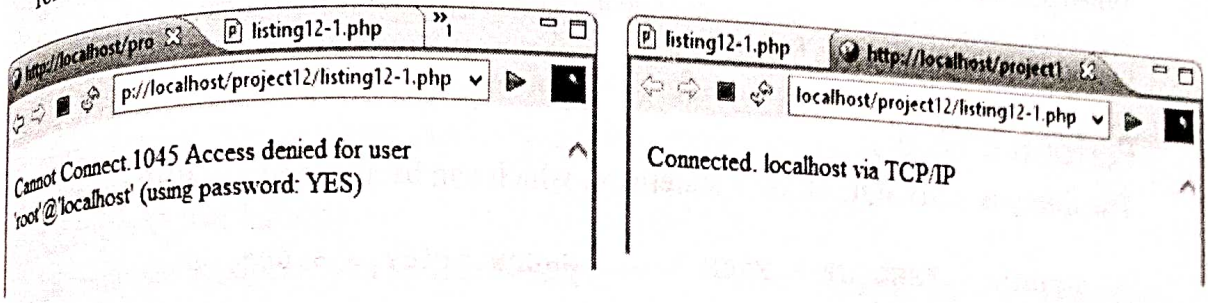
```
mysqli_connect("host_name", "user_name", "password", "database_name");
```

In this book I assume you have installed Zend Server (see Chapter 5) on your computer, used the default username `root`, and set up password of your choosing. If that is correct, then you could create a variable `$conid` like this:

```
$conid = mysqli_connect("localhost", "root", "password", "database_name");
```

If `mysqli_connect` is able to connect, the variable `$conid` contains the connection identifier that is used in the other PHP MySQL functions. This connection remains open for as long as the script is running. When the script ends, the connection is deleted. As a general rule, you need to specify the connection identifier in other PHP MySQL functions. If a connection is not made, then `$conid` contains `FALSE`. Listings 12-1a and 12-1b show how this can be used with the results, first on the left with the words "password" and

"database_name" for those values, and then on the right with my password and "books" for the database name.



Listing 12-1a Connect to a Database

```
<?php
  $conid = mysqli_connect("localhost", "root", "password",
     "database_name");
  if($conid) {
     echo "Connected. " . mysqli_get_host_info($conid);
   }
  else {
    echo "Cannot Connect. " . mysqli_connect_errno() . " " .
       mysqli_connect_error();
   }
  mysqli_close($conid);
?>
```

There are several ways to accomplish the same results. One of the simplest and historically common, and the one that is often used in testing, is shown in Listing 12-1b.

Listing 12-1b Connect to a Database

```
<?php
  $conid = mysqli_connect("localhost", "root", "password", "books")
       or die ("Cannot Connect. " . mysqli_connect_errno() . " " .
       mysqli_connect_error());
  echo "Connected. " . mysqli_get_host_info($conid);
  mysqli_close($conid);
?>
```

The `or die()` clause outputs its message and terminates the script. This is fine for testing, but it should not be used in an active website because it just terminates the current script without giving the user any option but to go to another site.

```
    <h3>List books priced above $5.00</h3>
    <?php
     //Connect to the database Books
    $conid = mysqli_connect("localhost", "root", "password", "books");
    if(!$conid) {
       echo "Couldn't connect. " . mysqli_connect_error();
    }
    //Select books > $5.00
    $query = "SELECT * FROM books WHERE price > 5.00";
    $result = mysqli_query($conid, $query);
    if(!$result) {
       echo "Couldn't do query. " . mysqli_error($conid);
    }
    //Display the books selected
    while($row = mysqli_fetch_row($result)) {
       echo $row[0] . " - ". $row[1] . " - ". $row[2] . " - $".
          $row[4] . "<br />";
    }
    //Free memory and close the database
    mysqli_free_result($result);
    mysqli_close($conid);
    ?>
  </body>
</html>
```

Database Information

As you use a MySQL database with PHP, you will need or want information about the database and its tables, rows, and fields. PHP has a number of functions for this purpose, many of which are shown in Table 12-2.

TIP

Database information functions are particularly useful to debug a script; to find out what is happening in your database; to find what table, row, and field you are accessing; or to find what error messages you are receiving.

How you use the database information functions depends on what you want to do. Here are some examples:

- **`mysqli_affected_rows()`**, which gets the number of rows affected by the last INSERT, UPDATE, REPLACE, or DELETE SQL query, provides this information to use with control functions to iterate through a set of rows. It is for types of queries other than SELECT or SHOW, which get the same information from **`mysqli_num_rows()`**.

Function	Description	Arguments and Comments
`mysqli_affected_rows()`	Gets number of rows affected by previous operation	Connection ID
`mysqli_character_set_name()`	Gets name of character set	Connection ID
`mysqli_connect_errno()`	Connect error code or number	Returns the error code of the last connection if it fails
`mysqli_connect_error()`	Connect error description	Returns the description of the error from the last connection
`mysqli_errno()`	Returns the error number from the previous operation	Connection ID
`mysqli_error()`	Returns the error message from the previous operation	Connection ID
`mysqli_fetch_fields()`	Returns the column name, table name, length of field, or type of a specified field	Result pointer from a query

`mysqli_get_client_info()`
`mysqli_get_host_info()`
`mysqli_get_server_info()`
	Returns the client version, the type of host connection, or the MySQL server version	Connection ID

`mysqli_info()`	Returns detailed information about the last query	Connection ID
`mysqli_insert_id()`	Gets the ID or key last generated by `AUTO_INCREMENT`	Connection ID
`mysqli_num_fields()` `mysqli_num_rows()`	Gets the number of fields or rows in the last query	Result pointer from the last query
`mysqli_stat()`	Gets the current server status	Connection ID

Table 12-2 Database Information Functions

- **`mysqli_error()`**, which returns the textual error message from the most recent MySQL operation, can be displayed with an `or die()` function or an `echo` statement.
- **`mysqli_insert_id()`**, which returns the ID or key that was automatically created through `AUTO_INCREMENT` by the last `INSERT` query, is used to reference the record that was just created.
- **`mysqli_num_rows()`**, which gets the number of rows affected by the last `SELECT` or `SHOW` SQL query, provides this information to use with control functions to iterate through a set of rows. It is for types of queries other than `INSERT`, `UPDATE`, `REPLACE`, or `DELETE`, which get the same information from **`mysqli_affected_rows()`**.

Database Administration

The database administration functions perform several maintenance duties on a database. Several of these functions are shown in Table 12-3.

Examples of how several of the database administration functions might be used are

- `mysqli_data_seek`, which moves the MySQL row pointer to a specific row, can be used with `mysqli_fetch_row` to get a specific row.
- `mysqli_ping`, which checks to see if a script is still connected to a MySQL server and reconnects it if it isn't, can be used after a lengthy idle period to possibly reestablish a connection.
- `mysqli_real_escape_string`, which "escapes" (adds backslashes to) the special characters in a string so they cannot easily include malicious code when included in a SQL statement, should be used with any string that is sent to MySQL that could possibly contain special characters.

NOTE

`mysqli_real_escape_string` takes into account the particular character set being used and can be specific to a particular connection.

Function	Description	Arguments and Comments
`mysqli_data_seek()`	Moves the row pointer to specified row	Result pointer from a query and the new row number (beginning at 0).
`mysqli_field_seek()`	Sets the result pointer to specified field number	Result pointer from the last query and the new field number (beginning at 0).
`mysqli_free_result()`	Frees the memory used by the last query	Result pointer from a query.
`mysqli_ping()`	Checks if a MySQL server connection exists, and if not, tries to reconnect	Connection ID. Returns `TRUE` if connection exists, `FALSE` otherwise.
`mysqli_real_escape_string()`	Escapes special characters in a string for use in SQL	Connection ID. String to be escaped.
`mysqli_set_charset()`	Sets the character set	Connection ID. Character set name.

Table 12-3 Database Administration Functions

Combining PHP MySQL Functions

Listing 12-3 combines a number of the PHP MySQL functions by returning to the Books database used earlier, having the user enter part of a title, and searching for and displaying the full book information. Among the other features are

- Use a form that returns to the PHP in the same script.
- Pause on the form until Submit is clicked.
- Strip the title that is entered of any white space, and make sure it is not blank.
- Escape special characters with `mysqli_real_escape_string`.
- Display the error text on a query error with `mysqli_error`.
- Display the book that is found using an associative array.

Listing 12-3 has several features that need further discussion, which you will see later in this chapter and in future chapters. Both the initial form and the output of this script are displayed in Figure 12-1.

Listing 12-3 Combining PHP MySQL Functions

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="utf-8">
    <title>Listing 12-3</title>
    </head>
    <body>
      <h1>Query the Books Database</h1>
      <h3>Search for a Title</h3>
      <p>Precede and/or follow the entry with % if not a
        complete title.</p>
      <?php
       tryagain:
      //Wait for submit
        if (!$_POST['submit']) {
      //Enter form information
      ?>
        <form action="<?=$_SERVER['PHP_SELF']?>" method="post">
          <p>Book Title: <input type="text" name="title" /></p>
          <p><input type="submit" name="submit" value="Submit" /></p>
        </form>
      <?php
        }
        else {
```

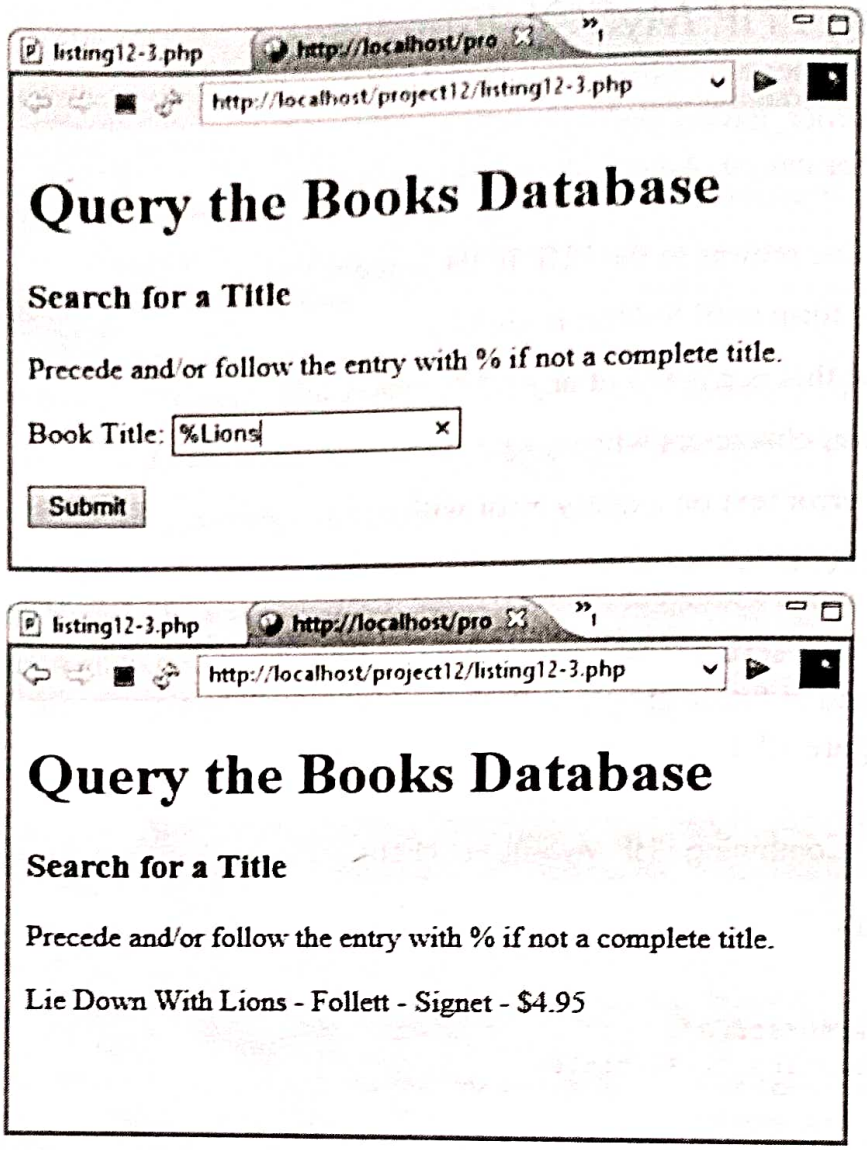


Figure 12-1 The query form disappears when the results are displayed.

```
//Connect to the Book database
   $conid = mysqli_connect("localhost", "root", "password",
      "books");
   if(!$conid) {
      echo "Couldn't connect. " . mysqli_connect_error();
   }
//Remove white space, check for blank, and remove special
  characters
   if (($title = trim($_POST['title'])) == '') {
      echo "Please enter a title.";
      $_POST['submit']=NULL;
      goto tryagain;
   }
   else {
```

```
    //Escape special characters in the title
        $title = mysqli_real_escape_string($conid, $_POST['title']);
      }
    //Select the book based on a partial title
      $query = "SELECT * FROM books WHERE Title LIKE '$title'";
      $result = mysqli_query($conid, $query);
      if(!$result) {
         echo "Couldn't do query. " . mysqli_error($conid);
      }
    //Display book with associative array
      while($row = mysqli_fetch_assoc($result)) {
         echo $row['Title']. " - ". $row['Author']. " - ".
            $row['Publisher']. " - $". $row['Price']. "<br />";
      }
      mysqli_free_result($result);
      mysqli_close($conid);
    }
    ?>
  </body>
</html>
```

NOTE

There is a large body of thought in the programming community that says that using `goto` is a bad practice because it can lead to disjointed code that goes off in multiple directions that can't be easily followed, and can generally be replaced with a compact loop. I don't disagree with that as a general rule, and, like many general rules, there are exceptions that make sense. Listing 12-3 is an example where using `goto` simplifies the code and is easy to follow. When tempted to use `goto` ask yourself if using it would simplify your code, or if a loop would be more understandable.

Working with Forms and Databases

In Chapters 2, 4, and 7 you have been introduced to forms in HTML, JavaScript, and PHP, respectively. In this chapter we'll expand on PHP form handling and then focus on the MySQL aspects of forms. In particular, we'll look at how PHP and MySQL collect and store data in a database table, and then how they retrieve data from a database and display it in a web page.

Expanding PHP Form Handling

PHP acts as both a partner and an intermediary among HTML, JavaScript, and MySQL. Listing 12-3 is a good example of this:

- You start in HTML to set up the web page with its titling.
- Go into PHP to check if the form has been submitted.

- Jump back to HTML to display the form.
- Go back to PHP to:
  - Connect to the MySQL database.
  - Clean up what was entered on the HTML form.
  - Check to make sure something was entered.
  - Query the MySQL database to see if the requested information is there.
  - Fetch the information from the MySQL database.
  - Display the information on the HTML web page.
  - Handle all the possible error conditions.

  You can see that PHP is playing several critical roles, including:
- Facilitating the dynamic user interaction on a static HTML web page.
- Interacting with a MySQL database.
- Providing error handling and security.

Facilitating User Interaction

When working with forms, HTML defines what the user sees—the form itself—and PHP processes the information that is entered into the form. It is a partnership that requires communication between them. Some of the ways this happens is described in the following sections, which uses Listing 12-3 as an example.

HTML `<form method`

On the HTML side, the HTML `<form` tag `method` attribute initializes either the `$_POST` or `$_GET` PHP superglobal array that is used to transfer information entered on the form to PHP using the field names that are defined by HTML in the form. For example, HTML `<form method="post">` and `<input name="title">` allow PHP to use `$_POST['title']` to pick up the value entered into the title field on the form.

HTML `<input type="submit"`

HTML `<input type="submit"` creates a button on the form that the user can click to indicate they have completed entry. The name and value of the button do not have to be "submit" and can be anything the web page creator wants. The type, of course, must be "submit." The name is what is used in the superglobal array, and the value is what is actually displayed on the button. When Submit is selected on the form, the `<form action` takes over and PHP can learn of this using the `$_POST['submit']` superglobal element.

HTML `<form action`

The HTML `<form` tag also uses the `action` attribute or its absence to determine where the focus—in this instance, PHP—should go to continue processing the form after the form has been submitted by the user. If there is no `action` attribute, then the focus is transferred to the first PHP code at the beginning of the script. This can be before or after the start of the HTML. The `action` attribute can also have the URL (web address) of a script to which the focus is transferred. Finally, as is the case in Listing 12-3, the `action` attribute can have a snippet of PHP code to transfer the focus to PHP in the current script. This is exactly the same as *not* having an `action` attribute. The only reason for having `action="<?=$_SERVER['PHP_SELF']?>"` is that it is more obvious to the human reader what is happening.

PHP `$_POST['submit']`

PHP, for its part, watches the superglobal array entry (`$_POST['submit']` in Listing 12-3), which is initially `NULL` or `FALSE`, and goes to work when it sees that the `submit` entry has something in it. PHP does this by asking if `$_POST['submit']` is `FALSE`. In other words, the form has not been submitted. If correct, HTML displays the form, and PHP waits until the form is submitted, when it can then go on and interact with MySQL and the database.

Interacting with a Database

When PHP sees that the form has been submitted, it connects with the database, removes any white space in the entry, and determines if the entry is blank. If so, it asks for a title to be entered, sets the superglobal array entry `$_POST['submit']` to `NULL` and goes back to watching for the form to be submitted. When that happens, and the title is found to not be blank, PHP looks for and escapes (places backslashes in front of) any special characters. (This may seem like it's not an important step, but just adding this one line of code greatly increases the security of your program.) Finally, the Books table is queried for the title and, if it is found, the row or record in the table with that title is fetched and displayed.

Improving Security

The user can enter anything into a web form that they want to. It can be malicious, silly, or just typing errors. PHP provides several functions and features to improve the safety of handling user input, but no matter how much you do, you need to be aware that *there is nothing you can do to be 100 percent safe.* That said, you should still do what you can in an unobtrusive way to protect your website and its users. Some of what you can do

is relatively simple, as you'll see here, but some of it is beyond the scope of this book, and you are encouraged to continue to explore this subject as your knowledge of web programming grows. The PHP Manual has an extensive section on security (php.net/manual/en/security.php) and is worth a review. Here we'll focus on handling user input.

PHP provides functions that work toward cleaning up user input, as shown in Table 12-4.

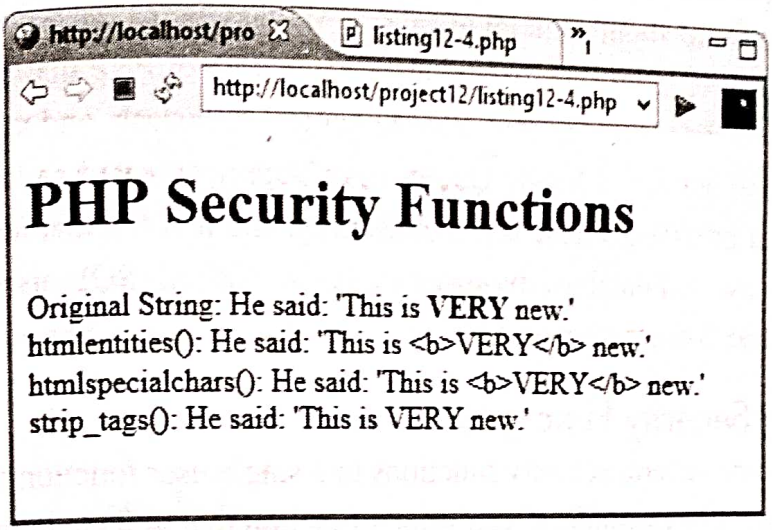
HTML codes that a user places in his or her form input operate as they are designed in HTML to do. If the user places "Super" in their entry, then the bold tags turn the word "Super" into bold. Obviously, turning a word bold is not a security issue, but there are other sets of tags that might be. `htmlentities()` and `htmlspecialcharacters()` convert HTML tags into their HTML entities, so "Super" becomes `<b>Super</b>` and is displayed as "Super." `strip_tags()` simply removes the HTML tags altogether, so "Super" becomes "Super."

`htmlentities()` and `htmlspecialcharacters()` are the same except that `htmlspecialcharacters()` limits itself to the five characters: ampersand (&), double quote ("), single quote ('), less than (<), and greater than (>). Both functions will convert these characters into &amp, &quot, &#039 or &apos, &lt, and &gt. By default, both functions will not convert single quotes, although they will convert double quotes. You can use the flag `ENT_QUOTES` to convert both types of quotes. Both functions are dependent on the character set being used. The default is UTF-8, but you can add an optional third argument to specify other character sets.

Function	Description	Arguments and Comments
`htmlentities()`	Converts HTML code into their HTML entities	String to convert. Optionally, flags on quote handling.
`htmlspecial characters()`	Converts specific characters into their HTML entities	String to convert. Converts only &, ", ', <, and >. Optionally, flags for additional handling.
`strip_tags()`	Removes all HTML and PHP tags	String to strip. Optionally, a list of tags to keep.
`trim()`	Removes white space on either end of a string	String to be trimmed. Strips blanks, tabs, newline, carriage return, NULL, vertical tab.
`mysqli_real_ escape_ string()`	Escapes (adds slashes in front of) special characters in a MySQL query	Connection ID. String to be escaped. Must be connected to a MySQL database.
`addslashes()`	Escapes (adds slashes in front of) special characters in a string	String to be escaped.

Table 12-4 Security-Related Functions

Listing 12-4 shows examples of these functions in use, with the results shown next.



Listing 12-4 Examples of PHP and MySQL Security Functions

```
<!DOCTYPE html>
<html>
  <head>
  <meta charset="utf-8">
  <title>Listing 12-4</title>
  </head>
  <body>
    <h1>PHP Security Functions</h1>
    <?php
        $astr = "He said: 'This is <b>VERY</b> new.'" ;
        $new1 = htmlentities($astr);
        $new2 = htmlspecialchars($astr, ENT_QUOTES);
        $new3 = strip_tags($astr);
        echo "Original String: ", $astr, "<br/>";
        echo "htmlentities(): ", $new1, "<br/>";
        echo "htmlspecialchars(): ", $new2, "<br/>";
        echo "strip_tags(): ", $new3 ;
    ?>
  </body>
</html>
```

NOTE

Don't let what `echo` displays confuse you. `htmlentities()` and `htmlspecialcharacters()` do convert the HTML and quotes as requested, but what you see echoed may not look that way.