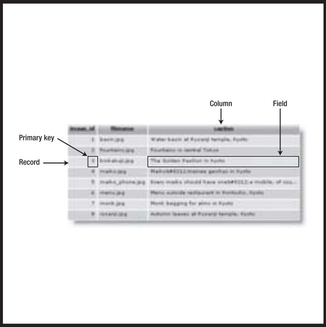
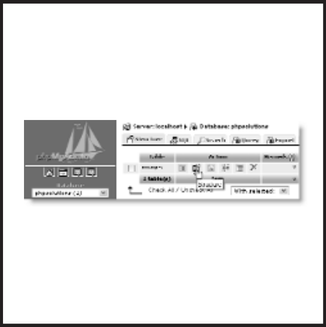
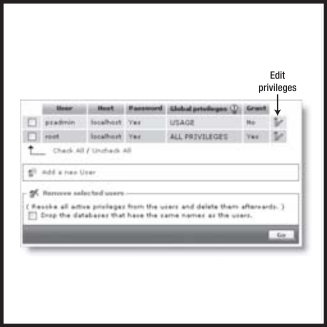
11 GETTING STARTED WITH A DATABASE







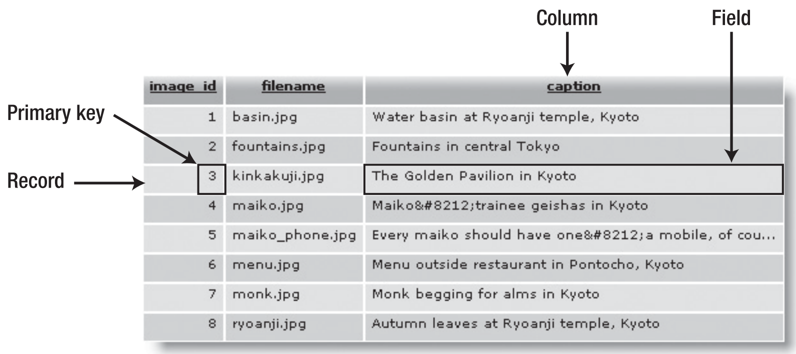
What this chapter covers:

- Creating MySQL user accounts
- Creating a new database
- Defining a database table
- Choosing the right column type
- Using MySQL, MySQLI, and PDO to query a database

When I first started working with databases, one of the greatest frustrations was that all the books and online tutorials I consulted assumed that you already knew the basics of database design and construction, or—if you didn't—that you planned to use Microsoft Access. MySQL is very different from Access, which is intended for use in small office environments. MySQL is not only fast and multiplatform; it's capable of handling a high number of simultaneous connections without any perceptible loss of performance. The differences between MySQL and Access also affect the way that you construct and interact with the database. After describing the basics of a database, I'll show you how to set up MySQL user accounts, create your first database, and connect to it with PHP. I'll also show you how to choose the correct data type to store each piece of information.

How a database stores information

MySQL is a relational database system. All the data is stored in tables, very much in the same way as in a spreadsheet, with information organized into rows and columns. Figure 11-1 shows the database table that you will build later in this chapter, as displayed in phpMyAdmin.



image_id	filename	caption
1	basin.jpg	Water basin at Ryoanji temple, Kyoto
2	fountains.jpg	Fountains in central Tokyo
3	kinkakuji.jpg	The Golden Pavilion in Kyoto
4	maiko.jpg	Maiko&#8212;trainee geishas in Kyoto
5	maiko_phone.jpg	Every maiko should have one&#8212;a mobile, of cou...
6	menu.jpg	Menu outside restaurant in Pontocho, Kyoto
7	monk.jpg	Monk begging for alms in Kyoto
8	ryoanji.jpg	Autumn leaves at Ryoanji temple, Kyoto

Figure 11-1. Information in a database table is stored in rows and columns, just like in a spreadsheet.

Each **column** has a name (`image_id`, `filename`, and `caption`) indicating what it stores.

The rows aren't labeled, but the first column (`image_id`) contains a unique identifier known as a **primary key**, which can be used to identify the data associated with a particular row.

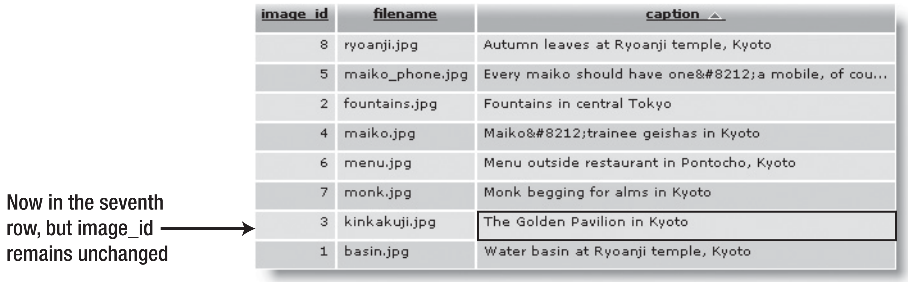
Each row contains an individual **record** of related data. The significance of primary keys is explained in the next section.

The intersection of a row and a column, where the data is stored, is called a **field**. So, for instance, the `caption` field for the third record in Figure 11-1 contains the value "The Golden Pavilion in Kyoto" and the primary key for that record is 3.

> *The terms "field" and "column" are often used interchangeably, particularly by phpMyAdmin. A field holds one piece of information for a single record, whereas a column contains the same field for all records.*

How primary keys work

Although Figure 11-1 shows `image_id` as a consecutive sequence from 1 to 8, they're not row numbers. Figure 11-2 shows the same table with the captions sorted in alphabetical order. The field highlighted in Figure 11-1 has moved to the seventh row, but it still has the same `image_id` and `filename`.



image_id	filename	caption
8	ryoanji.jpg	Autumn leaves at Ryoanji temple, Kyoto
5	maiko_phone.jpg	Every maiko should have one&#8212;a mobile, of cou...
2	fountains.jpg	Fountains in central Tokyo
4	maiko.jpg	Maiko&#8212;trainee geishas in Kyoto
6	menu.jpg	Menu outside restaurant in Pontocho, Kyoto
7	monk.jpg	Monk begging for alms in Kyoto
3	kinkakuji.jpg	The Golden Pavilion in Kyoto
1	basin.jpg	Water basin at Ryoanji temple, Kyoto

Figure 11-2. Even when the table is sorted in a different order, each record can be identified by its primary key.

Although the primary key is rarely displayed, it identifies the record and all the data stored in it. Once you know the primary key of a record, you can update it, delete it, or use it to display data in a separate page. Don't worry about how you find the primary key. You'll see in the next chapter that it's easily done using Structured Query Language (SQL), the standard means of communicating with all major databases. The important thing to remember is that you should assign a primary key to every record.

- A primary key doesn't need to be a number, but *it must be unique*.
- Social security, staff ID, or product numbers make good primary keys. They may consist of a mixture of numbers, letters, and other characters, but are always different.
- MySQL will generate a primary key for you automatically.
- Once a primary key has been assigned, it should never—repeat, never—be changed.

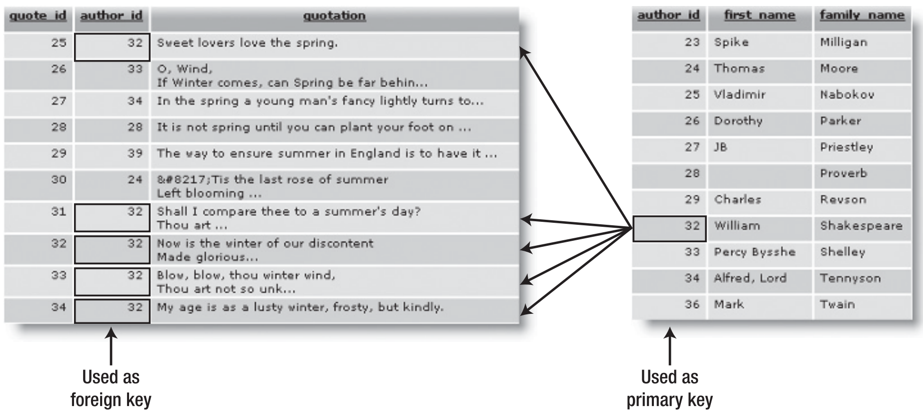
Because a primary key must be unique, MySQL doesn't normally reuse the number when a record is deleted. This leaves holes in the sequence. *Don't even think about renumbering.* Gaps in the sequence are of no importance whatsoever. The purpose of the primary key is to identify the record, and by changing the numbers to close the gaps, you put the integrity of your database at serious risk.

Some people want to remove gaps in the sequence to keep track of the number of records in a table. It's not necessary, as you'll discover later in the chapter.

Linking tables with primary and foreign keys

A major difference between a spreadsheet and a relational database like MySQL is that most databases store data in lots of smaller tables, rather than in one huge table. The main reason for doing this is to prevent duplication and inconsistency. Let's say you're building a database of your favorite quotations. Instead of typing out the name of the author each time, it's more efficient to put the authors' names in a separate table, and store a reference to an author's primary key with each quotation.

Storing a primary key from one table in another table is known as creating a **foreign key**. As you can see in Figure 11-3, every record in the left-hand table identified by `author_id 32` is a quotation from William Shakespeare. Because the name is stored in only one place, it guarantees that it's always spelled correctly. And if you do make a spelling mistake, just a single correction is all that's needed to ensure that the change is reflected throughout the database.



quote_id	author_id	quotation
25	32	Sweet lovers love the spring.
26	33	O, Wind, If Winter comes, can Spring be far behin...
27	34	In the spring a young man's fancy lightly turns to...
28	28	It is not spring until you can plant your foot on ...
29	39	The way to ensure summer in England is to have it ...
30	24	&#8217;Tis the last rose of summer Left blooming ...
31	32	Shall I compare thee to a summer's day? Thou art ...
32	32	Now is the winter of our discontent Made glorious...
33	32	Blow, blow, thou winter wind, Thou art not so unk...
34	32	My age is as a lusty winter, frosty, but kindly.

author_id	first_name	family_name
23	Spike	Milligan
24	Thomas	Moore
25	Vladimir	Nabokov
26	Dorothy	Parker
27	JB	Priestley
28		Proverb
29	Charles	Revson
32	William	Shakespeare
33	Percy Bysshe	Shelley
34	Alfred, Lord	Tennyson
36	Mark	Twain

Figure 11-3. Foreign keys are used to link information stored in separate tables.

Using foreign keys to link information in different tables is one of the most powerful aspects of a relational database. It can also be difficult to grasp in the early stages, so we'll work with single tables until Chapter 14, which covers foreign keys in detail. In the meantime, bear the following points in mind:

- When used as the primary key of a table, the identifier must be unique. So each `author_id` in the table on the right is used only once.
- When used as a foreign key, there can be multiple references to the same identifier. So `32` appears several times in the `author_id` column in the table on the left.

As long as `author_id` *remains unique in the table where it's the primary key, you know that it always refers to the same person.*

Breaking down information into small chunks

You may have noticed that the table on the right in Figure 11-3 has separate columns for each author's first name and family name. This is an important principle of a relational database: *break down complex information into its component parts, and store each part separately*.

It's not always easy to decide how far to go with this process. In addition to first and last name, you might want separate columns for title (Mr., Mrs., Ms., Dr., and so on) and for middle names or initials. Addresses are best broken down into street, town, county, state, zip code, and so on. Although it may be a nuisance to break down information into small chunks, you can always use SQL and/or PHP to join them together again. However, once you have more than a handful of records, it's a major undertaking to try to separate complex information stored in a single field.

Checkpoints for good database design

There is no *right* way to design a database—each one is different. However, the following guidelines should point you in the right direction:

- Give each record in a table a unique identifier (primary key).
- Put each group of associated data in a table of its own.
- Cross-reference related information by using the primary key from one table as the foreign key in other tables.
- Store only one item of information in each field.
- Stay DRY (don't repeat yourself).

In the early stages, you are likely to make design mistakes that you later come to regret. Try to anticipate future needs, and make your table structure flexible. You can add new tables at any time to respond to new requirements.

That's enough theory for the moment. Let's get on with something more practical by building a database for the Japan Solutions website from Chapters 4 and 5.

Setting up the phpsolutions database

MySQL is a relational database management system (RDMS), which can support a large number of databases. In a local testing environment, there's no limit to the number of databases that you can create, and you can call them whatever you like. I am going to assume that you are working in a local testing environment and will show you how to set up a database called `phpsolutions`, together with two user accounts called `psquery` and `psadmin`.

On shared hosting, you may be limited to just one database set up by the hosting company. If you don't have the freedom to set up a new database and user accounts, substitute the name and username allocated by your hosting company for `phpsolutions` *and* `psadmin` *respectively throughout the rest of this book.*

MySQL naming rules

The basic MySQL naming rules for databases, tables, and columns are as follows:

- Names can be up to 64 characters long.
- Legal characters are numbers, letters, the underscore, and $.
- Names can begin with a number, but cannot consist exclusively of numbers.

Some hosting companies seem blissfully ignorant of these rules and assign clients databases that contain one or more hyphens (an illegal character) in their name. If a database, table, or column name contains spaces or illegal characters, you must always surround it by backticks (`` ` ``) in SQL queries. Note that this is not a single quote (`'`), but a separate character.

When choosing names, you might accidentally choose one of MySQL's many reserved words (`http://dev.mysql.com/doc/refman/5.0/en/reserved-words.html`), such as `date` or `time`. One technique to avoid this is to use compound words, such as `arrival_date`, `arrival_time`, and so on. Alternatively, surround all names with backticks. phpMyAdmin does this automatically, but you need to do this manually when writing your own SQL in a PHP script.

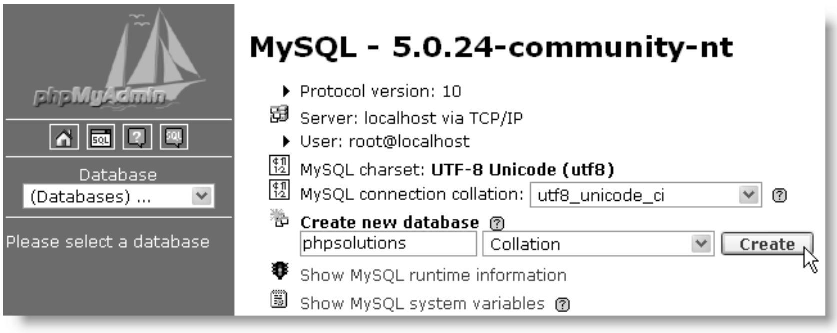
Case sensitivity of names

Windows and Mac OS X treat MySQL names as case-insensitive. However, Linux and Unix servers respect case sensitivity. To avoid problems when transferring databases and PHP code from your local computer to a remote server, I strongly recommend that you use lowercase exclusively in database, table, and column names. When building names from more than one word, join them with an underscore.

Using phpMyAdmin to create a new database

Creating a new database in phpMyAdmin is easy.

1. Launch phpMyAdmin in a browser, as described in the previous chapter.
2. Type the name of the new database (phpsolutions) into the field labeled Create new database. Leave the Collation drop-down menu at its default setting, and click Create, as shown in the following screenshot:



MySQL - 5.0.24-community-nt

- Protocol version: 10
- Server: localhost via TCP/IP
- User: root@localhost
- MySQL charset: **UTF-8 Unicode (utf8)**
- MySQL connection collation: utf8_unicode_ci
- **Create new database**: phpsolutions, Collation

Collation determines the sort order of records. Unless you are using a language other than English, Swedish, or Finnish, you never need to change its value. Even if you use a different language, you should use the Collation option only if your remote server uses MySQL 4.1 or higher.

3. The next screen should confirm that the database has been created and offer you the opportunity to create your first table. Before creating any tables in a new database, it's a good idea to create user accounts for it. Leave phpMyAdmin open, as you'll continue using it in the next section.

Creating database-specific user accounts

At the moment, your installation of MySQL has only one registered user—the superuser account called "root," which has complete control over everything. The root user should *never* be used for anything other than top-level administration, such as the creation and removal of databases, creating user accounts, and exporting and importing data. Each individual database should have at least one—preferably two—dedicated user accounts with limited privileges.

When you put a database online, you should grant users the least privileges they need, and no more. There are four important privileges—all named after the equivalent SQL commands:

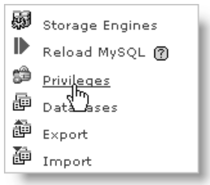
- **SELECT**: Retrieves records from database tables
- **INSERT**: Inserts records into a database

- **UPDATE**: Changes existing records
- **DELETE**: Deletes records, but not tables or databases (the command for that is DROP)

Most of the time, visitors need only to retrieve information, so the psquery user account will have the SELECT privilege only. However, for user registration or site administration, you need all four privileges. These will be made available to the psadmin account.

Granting user privileges

1. Return to the main phpMyAdmin screen by clicking either the little house icon at the top left of the left frame or Server: localhost at the top left of the main frame.
2. Click the Privileges link toward the bottom of the left column of the main screen.

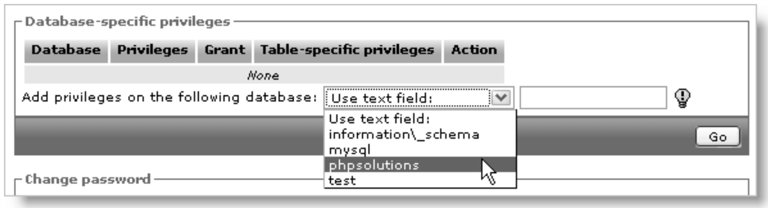


Most links and tabs in phpMyAdmin are context-sensitive. It's important to go back to the main screen and click the Privileges *link rather than the* Privileges *tab at the top of the previous screen. The link on the phpMyAdmin main screen lets you set up new user accounts. The* Privileges *tab at the top of a page only provides information about existing accounts.*

3. This opens the User overview screen. If you have just installed MySQL, there should be only one user: root. Click the Add a new User link halfway down the page.
4. In the page that opens, enter psadmin (or the name of the user account that you want to create) in the User name field. Select Local from the Host drop-down menu. This automatically enters localhost in the field alongside. Selecting this option allows the psadmin user to connect to MySQL only from the same computer. Then enter a password in the Password field, and type it again for confirmation in the Re-type field.

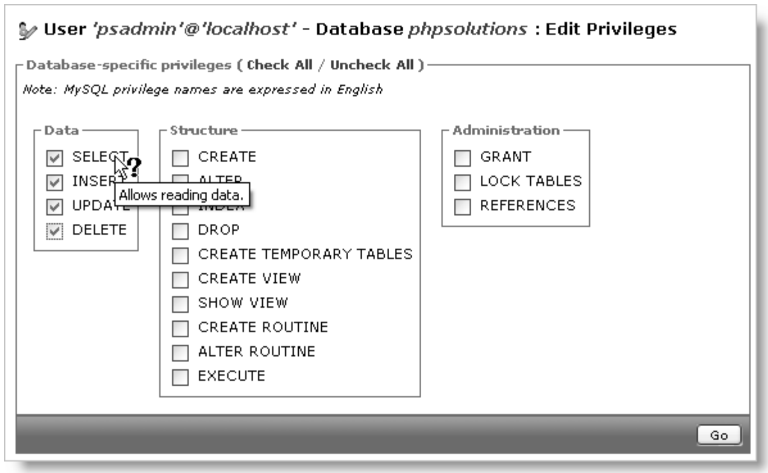
In the download files for this book, I've used a simple password (kyoto), but for a database on the Internet, you should choose a password that's hard to guess. MySQL passwords are case-sensitive.

5. Beneath the Login Information table is one labeled Global privileges. These give a user privileges on all databases, including the mysql one, which contains sensitive information. Granting such extensive privileges is insecure, so leave the Global privileges table unchecked, and click the Go button right at the bottom of the page.
6. The next page confirms that the psadmin user has been created and displays many options, beginning with the Global privileges table again. Scroll down below this to the section labeled Database-specific privileges. Activate the drop-down menu, as shown here, to display a list of all databases on your system. Select phpsolutions.



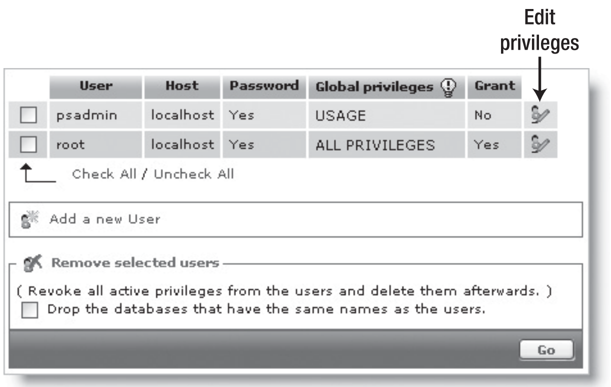
A new installation of MySQL 5 contains three databases: information_schema (phpMyAdmin escapes the underscore by preceding it with a backslash), mysql, and test. The first, information_schema, is a virtual database that contains details of all other databases on the same server. You can view the contents in phpMyAdmin, but you can't edit them. The mysql database contains details of all user accounts and privileges. You should never edit it directly unless you're sure what you're doing. Always use the Privileges link on the main phpMyAdmin page to manage user accounts, privileges, and passwords. The test database is empty.

7. The next screen allows you to set the privileges for this user on just the phpsolutions database. You want psadmin to have all four privileges listed earlier, so click the check boxes next to SELECT, INSERT, UPDATE, and DELETE. (If you hover your mouse pointer over each option, phpMyAdmin displays a tooltip describing what the option is for, as shown.) After selecting the four privileges, click the top Go button. (Always click the Go button at the foot of or alongside the section with the options you want to set.)



8. phpMyAdmin presents you with confirmation that the privileges have been updated for the psadmin user account: the page displays the Database-specific privileges table again, in case you need to change anything. Click the Privileges tab at the top of the page. You should now see psadmin listed with root in the User overview.

If you ever need to make any changes to a user's privileges, click the Edit Privileges icon to the right of the listing, as shown. To delete a user, select the check box to the left of the User column, and then click Go in the Remove selected users section.



	User	Host	Password	Global privileges	Grant	
☐	psadmin	localhost	Yes	USAGE	No	
☐	root	localhost	Yes	ALL PRIVILEGES	Yes	

Check All / Uncheck All

Add a new User

Remove selected users

(Revoke all active privileges from the users and delete them afterwards.)

☐ Drop the databases that have the same names as the users.

Go

9. Click Add a new User, and repeat steps 4 through 8 to create a second user account called `psquery`. This user will have much more restricted privileges, so when you get to step 7, check only the SELECT option. The password I used for `psquery` is `fuji`. Again, for an online database, you should choose something more robust.

Creating a database table

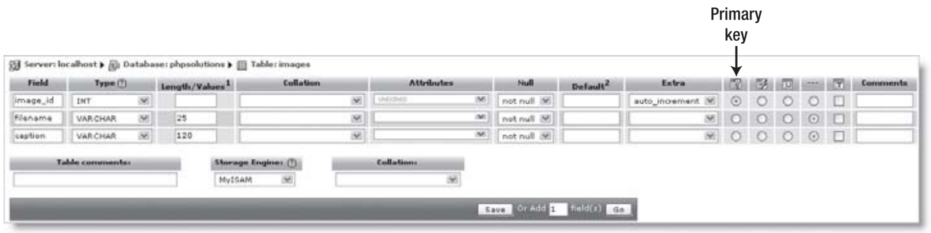
Now that you have a database and dedicated user accounts, you can begin creating tables. Let's begin by creating a table to hold the details of images, as shown in Figure 11-1. Before you can start entering data, you need to define the table structure. This involves deciding the following:

- The name of the table
- How many columns it will have
- The name of each column
- What type of data will be stored in each column
- Whether the column must always have data in each field
- Which column contains the table's primary key

If you look at Figure 11-1, you can see that the table contains three columns: `image_id` (primary key), `filename`, and `caption`. Because it contains details of images, that's a good name to use. There's not much point in storing a filename without a caption, so every column must contain data. Great! Apart from the data type, all the decisions have been made. I'll explain the data types as we go along.

Defining the images table

1. Launch phpMyAdmin in a browser, if it's not already open, and select phpsolutions from the Database drop-down menu in the left frame. Type the name of the new table (images) in the field labeled Create new table on database phpsolutions, and enter 3 as the Number of fields. (As mentioned before, phpMyAdmin refers to columns as fields. What it means is how many fields each record has.) Then click the Go button.
2. The next screen is where you define the table. Unless you have a large monitor, you will probably need to scroll horizontally to see all of it. The following screenshot shows all the fields filled in, but the values may be difficult to read, so they are also listed in Table 11-1 (Collation and Default are omitted, as they are both left blank).



Server: localhost ▸ Database: phpsolutions ▸ Table: images

Field	Type	Length/Values¹	Collation	Attributes	Null	Default²	Extra						Comments
image_id	INT			UNSIGNED	not null		auto_increment	●	○	○	○	☐	
filename	VARCHAR	25			not null			○	○	○	●	☐	
caption	VARCHAR	120			not null			○	○	○	●	☐	

Table comments:
Storage Engine: MyISAM
Collation:

Save Or Add 1 field(s) Go

Table 11-1. Settings for the images table

Field	Type	Length/Values	Attributes	Null	Extra	Primary key
image_id	INT		UNSIGNED	not null	auto_increment	Selected
filename	VARCHAR	25		not null		
caption	VARCHAR	120		not null		

The first column, image_id, is defined as type INT, which stands for integer. Its attribute is set to UNSIGNED, which means that only positive numbers are allowed. It's also set to auto_increment, and is the table's primary key, so MySQL automatically inserts in this column the next available number (starting at 1) whenever a new record is inserted.

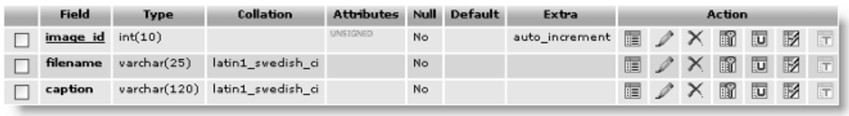
The next column, filename, is defined as type VARCHAR with a length of 25. This means it accepts up to 25 characters of text.

The final column, caption, is also VARCHAR with a length of 120, so it accepts up to 120 characters of text.

All columns are defined as not null, so they must always contain something. However, that "something" can be as little as an empty string. I'll describe the column types in more detail in "Choosing the right column type in MySQL" later in the chapter.

When you have finished, click the Save button at the bottom-center of the screen.

3. The next screen displays the SQL query that phpMyAdmin used to define the images table. Beneath that, you'll see the structure of the table displayed like this:



	Field	Type	Collation	Attributes	Null	Default	Extra	Action
☐	image_id	int(10)		UNSIGNED	No		auto_increment	
☐	filename	varchar(25)	latin1_swedish_ci		No			
☐	caption	varchar(120)	latin1_swedish_ci		No			

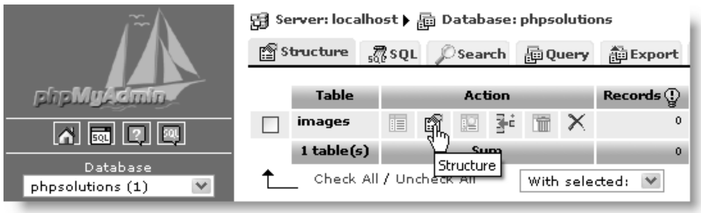
Don't be alarmed by the fact that Collation displays latin1_swedish_ci. MySQL is based in Sweden, and Swedish uses the same sort order as English (and Finnish). The underlining of image_id indicates that it's the table's primary key. To change any settings, click the pencil-like icon in the appropriate row. This opens a version of the previous screen and allows you to change the values. If you made a complete mess and want to start again, click the Drop tab at the top right of the screen, and confirm that you want to drop the table. (In SQL, *delete* refers only to records. You *drop* a table or a database.)

Inserting records into a table

Now that you have a table, you need to put some data into it. Eventually, you'll need to build your own content management system using XHTML forms, PHP, and SQL; but the quick and easy way to do it is with phpMyAdmin. First, I'll show you how to enter a couple of records manually; and then I'll show you how to cheat by loading the entire table from a SQL file.

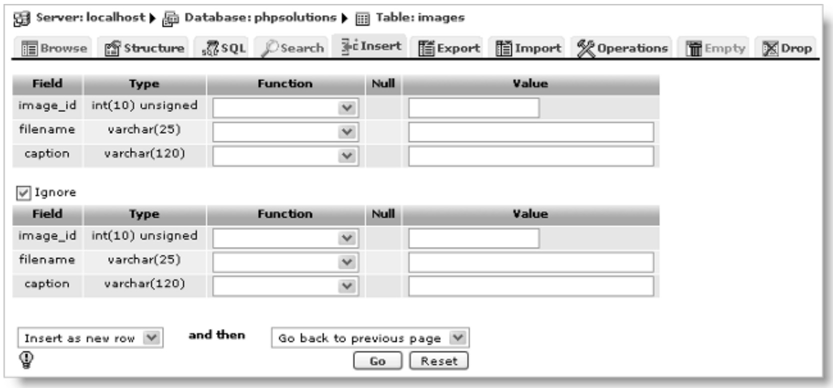
Using phpMyAdmin to insert records manually

1. If phpMyAdmin is still displaying the structure of the images table as at the end of the previous section, skip to step 2. Otherwise, launch phpMyAdmin, and select the phpsolutions database from the drop-down menu in the left frame. Then click the Structure icon alongside images, as shown in the following screenshot:



The breadcrumb trail at the top of the main frame provides the context for the tabs across the head of the page. The Structure tab *at the top left of the preceding screenshot refers to the structure of the* phpsolutions *database. At the moment, it contains only one table,* images. *To access the structure of an individual table, click the* Structure icon *alongside its name. Use your mouse pointer to reveal tooltips for each icon. Some, such as* Browse, *are grayed out because there are no records in the table.*

2. Click the Insert tab in the center top of the page. This displays the following screen, ready for you to insert up to two records:



Field	Type	Function	Null	Value
image_id	int(10) unsigned			
filename	varchar(25)			
caption	varchar(120)			

Field	Type	Function	Null	Value
image_id	int(10) unsigned			
filename	varchar(25)			
caption	varchar(120)			

3. The forms display the names and details of each column. You can ignore the Function fields. MySQL has a large number of functions that you can apply to the values being stored in your table. You'll learn more about them in the following chapters. The Value field is where you enter the data you want to insert in the table.

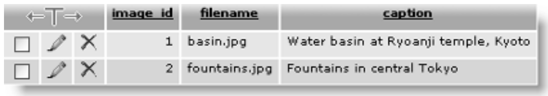
   Because you have defined `image_id` as `auto_increment`, MySQL inserts the next available number automatically. So you *must* leave the first Value field blank. Fill in the next two Value fields as follows:

   - filename: basin.jpg
   - caption: Water basin at Ryoanji temple, Kyoto

4. Deselect the check box labeled Ignore. If you forget to do this, anything entered in the second form won't be inserted into the table.
5. Again, leave the Value field for image_id blank, and fill in the next two fields like this:

   - filename: fountains.jpg
   - caption: Fountains in central Tokyo

6. Click Go. You should be taken back to the table structure page, but the SQL used to insert the records is displayed at the top of the page. I'll explain the basic SQL commands in the remaining chapters, but studying the SQL that phpMyAdmin displays is a good way to learn how to build your own queries. SQL is closely based on human language, so it isn't all that difficult to learn.
7. Click the Browse tab at the top left of the page. You should now see the first two entries in the `images` table, as shown here:



	image_id	filename	caption
	1	basin.jpg	Water basin at Ryoanji temple, Kyoto
	2	fountains.jpg	Fountains in central Tokyo

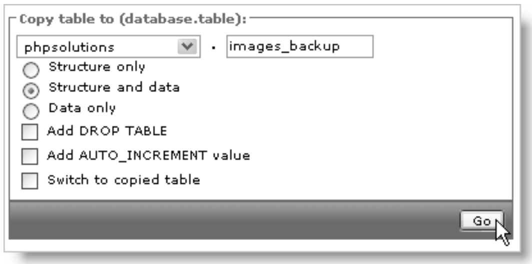
   As you can see, MySQL has automatically inserted 1 and 2 in the image_id fields.

You could continue typing out the details of the remaining six images, but let's speed things up a bit by using a SQL file that contains all the necessary data.

Loading the images records from a SQL file

Because the primary key of the images table has been set to auto_increment, it's necessary to drop the original table and all its data. The SQL file does this automatically and builds the table from scratch. These instructions assume that phpMyAdmin is open at the page in step 7 of the previous section.

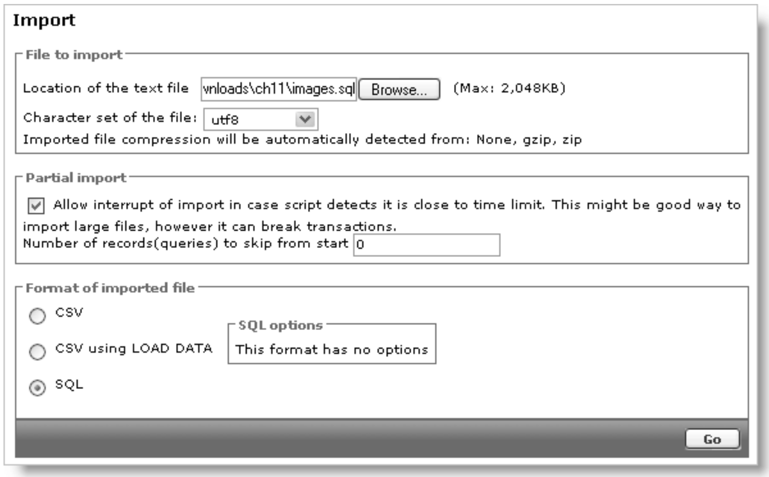
1. If you're happy to overwrite the data in the images table, skip to step 2. However, if you have entered data that you don't want to lose, copy your data to a different table. Click the Operations tab at the top of the page, type the name of the new table in the blank field in the section titled Copy table to (database.table), and click Go. The following screenshot shows the settings for copying the images table to images_backup:



Copy table to (database.table):
phpsolutions . images_backup
[ ] Structure only
[✓ Structure and data]
[ ] Data only
[ ] Add DROP TABLE
[ ] Add AUTO_INCREMENT value
[ ] Switch to copied table
Go

After clicking Go, you should see confirmation that the table has been copied. The breadcrumb trail at the top of the page indicates that phpMyAdmin is still in the images table, so you can proceed to step 2, even though you have a different page onscreen.

2. Click the Import tab at the top right of the page. In the next screen, click the Browse (or Choose File) button in File to import, and navigate to images.sql in the download files. Leave all options at their default setting, and click Go at the foot of the page.



Import
File to import
Location of the text file vnloads\ch11\images.sql Browse... (Max: 2,048KB)
Character set of the file: utf8
Imported file compression will be automatically detected from: None, gzip, zip
Partial import
[✓] Allow interrupt of import in case script detects it is close to time limit. This might be good way to import large files, however it can break transactions.
Number of records(queries) to skip from start 0
Format of imported file
[ ] CSV
[ ] CSV using LOAD DATA
[✓ SQL]
SQL options
This format has no options
Go

Use `images323.sql` *for MySQL 3.23 or* `images40.sql` *for MySQL 4.0. Older versions of phpMyAdmin don't have an* Import *tab. Click the* SQL *tab instead. The* File to import *form is at the bottom of the page. It looks slightly different, but works the same way.*

3. phpMyAdmin drops the original table, creates a new version, and inserts all the records. When you see confirmation that the file has been imported, click the Browse button at the top left of the page. You should now see the same data as shown in Figure 11-1 at the beginning of the chapter.

Now that you've got some useful data in your database table, it's time to bring it together with PHP, but first, a quick overview of the main column types in MySQL.

Choosing the right column type in MySQL

You may have received a bit of a shock when selecting Type for the `image_id` column. phpMyAdmin lists all available column types—there are 28 in MySQL 5.0. Rather than confuse you with unnecessary details, I'll explain just the most commonly used. You can find full details of all column types in the MySQL documentation at `http://dev.mysql.com/doc/refman/5.0/en/data-types.html`.

Storing text

The difference between the main text column types boils down to the maximum number of characters that can be stored in an individual field, and whether you can set a default value.

- **CHAR**: A fixed-length character string. You must specify the required length in the Length/Values field. The maximum permitted value in all versions of MySQL is 255. You can define a default value.
- **VARCHAR**: A variable-length character string. You must specify the maximum number of characters you plan to use in the Length/Values field in phpMyAdmin. Prior to MySQL 5.0, the limit is 255. This has been increased to 65,535 in MySQL 5.0. Accepts a default value.
- **TEXT**: Stores text up to a maximum of 65,535 characters (slightly longer than this chapter). Cannot define a default value.

TEXT is convenient because you don't need to specify a maximum size (in fact, you can't). Although the maximum length of VARCHAR is the same as TEXT in MySQL 5.0, other factors may limit the actual amount that can be stored. Keep it simple: use VARCHAR for short text items and TEXT for longer ones.

The term "characters" here refers only to characters in the Latin1 (ISO-8859-1) character set—the default encoding for most Western European languages. If you store your data in UTF-8 (Unicode), the limit is calculated in bytes. Accented characters in Spanish, French, and other Western languages require only one byte in Latin1, but occupy two bytes in UTF-8.

Storing numbers

The most frequently used numeric column types are as follows:

- **INT**: Any whole number (integer) between –2,147,483,648 and 2,147,483,647. If the column is declared as UNSIGNED, the range is from 0 to 4,294,967,295.
- **FLOAT**: A floating-point number. You can optionally specify two comma-separated numbers in the Length/Values field. The first number specifies the number of digits before the decimal point, and the second specifies the precision to which the decimal portion should be rounded. Since PHP will format numbers after calculation, I recommend that you use FLOAT without the optional parameters.
- **DECIMAL**: A floating-point number *stored as a string*. *This column type is best avoided*.

DECIMAL is intended for currencies, but you can't perform calculations with strings inside a database, so it's more practical to use INT. For dollars or euros, store currencies as cents; for pounds, use pence. Then use PHP to divide the result by 100, and format the currency as desired.

Don't use commas or spaces as the thousands-separator. Apart from numerals, the only characters permitted in numbers are the negative operator (-) and the decimal point (.).

Storing dates and times

MySQL stores dates in the format YYYY-MM-DD. This comes as a shock to many people, but it's the standard approved by the ISO (International Organization for Standardization), and avoids the ambiguity inherent in different national conventions. I'll return to the subject of dates in Chapter 14. The most important column types for dates and times are as follows:

- **DATE**: A date stored as YYYY-MM-DD. The supported range is 1000-01-01 to 9999-12-31.
- **DATETIME**: A combined date and time displayed in the format YYYY-MM-DD HH:MM:SS.
- **TIMESTAMP**: A timestamp (normally generated automatically by the computer). Legal values range from the beginning of 1970 to partway through 2037.

MySQL timestamps are based on a human-readable date and, since MySQL 4.1, use the same format as DATETIME. *As a result, they are incompatible with Unix and PHP timestamps, which are based on the number of seconds elapsed since January 1, 1970. Don't mix them.*

Storing predefined lists

MySQL lets you store two types of predefined list that could be regarded as the database equivalents of radio button and check box states:

- **ENUM**: This column type stores a single choice from a predefined list, such as "yes, no, don't know" or "male, female." The maximum number of items that can be stored in the predefined list is a mind-boggling 65,535—some radio-button group!
- **SET**: This column type stores zero or more choices from a predefined list. The list can hold a maximum of 64 choices.

While ENUM is quite useful, SET tends to be less so, mainly because it violates the principle of storing only one piece of information in a field. The type of situation where it can be useful is when recording optional extras on a car or multiple choices in a survey.

Storing binary data

Storing binary data, such as images, isn't a good idea. It bloats your database, and you can't display images directly from a database. However, the following column types are designed for binary data:

- **TINYBLOB**: Up to 255 bytes
- **BLOB**: Up to 64KB
- **MEDIUMBLOB**: Up to 16MB
- **LONGBLOB**: Up to 4GB

With such whimsical names, it's a bit of a letdown to discover that BLOB stands for **binary large object**.

Connecting to MySQL with PHP

One of the great features of PHP is that it supports all the major database systems—and some not so major ones, too. It's also a weakness, because PHP uses dedicated functions for each type of database. This isn't a problem if you use the same database all the time, but it makes code less portable. Consequently, PHP Data Objects (PDO) were introduced in PHP 5.1. The idea is that you write just one set of code, and it will work with any database. Strictly speaking, this isn't 100% true, because there are variations in the way you write SQL for some databases. Nevertheless, it's a major change; and the plan is to move PHP database connection entirely to PDO.

Unfortunately, there's a rather large fly in the ointment . . . Even two years after the release of PHP 5, a large number of hosting companies still offered only PHP 4, and seemed to be in no hurry to upgrade. As a result, if your remote server runs on PHP 4, you still need to use the original MySQL extension. Just to make things more complicated, PHP 5 also offers the MySQL Improved (MySQLI) extension, which is intended for use with MySQL 4.1 and above. So, before you can work with PHP and MySQL on your website, you need to check which versions are running. You have the following options:

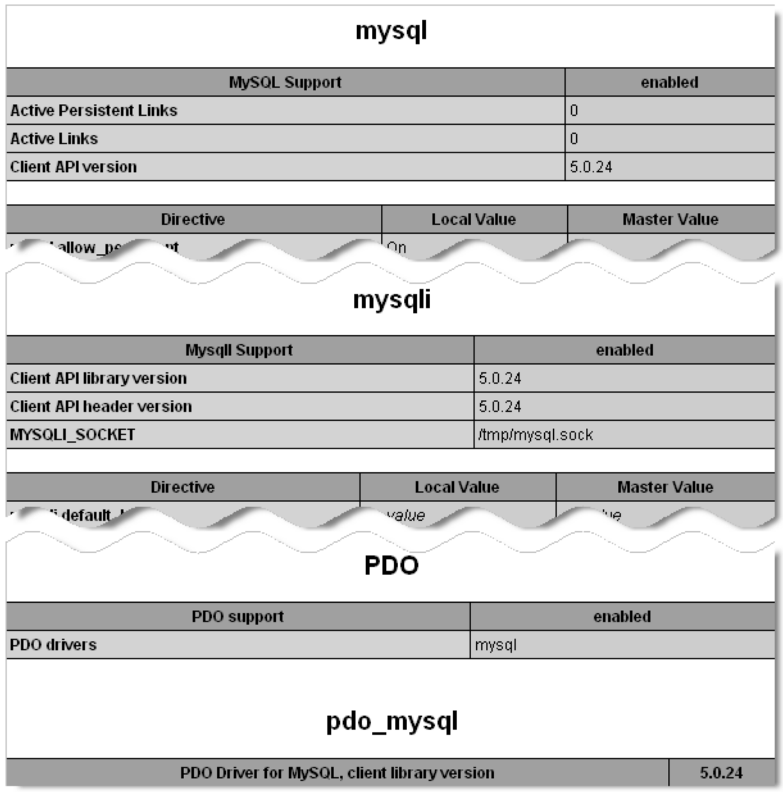
- If your remote server runs PHP 4, you must use the MySQL extension.
- If your remote server runs PHP 5 *and* MySQL 4.1 or above, use the MySQL Improved extension or—if it's available—PDO.

Checking your remote server setup

As always, run the following one-line script to find out the PHP configuration of your remote server:

```
<?php phpinfo(); ?>
```

Scroll down the configuration page, and look for the following sections.



mysql

MySQL Support	enabled
Active Persistent Links	0
Active Links	0
Client API version	5.0.24

Directive	Local Value	Master Value
allow_pe	On	

mysqli

Mysqli Support	enabled
Client API library version	5.0.24
Client API header version	5.0.24
MYSQLI_SOCKET	/tmp/mysql.sock

Directive	Local Value	Master Value
i default	value	ue

PDO

PDO support	enabled
PDO drivers	mysql

pdo_mysql

PDO Driver for MySQL, client library version	5.0.24

All websites should have the first section (mysql), but the mysqli and PDO sections will depend on the server and the version of PHP installed. If you have PDO, you must also make sure that mysql is listed among the PDO drivers.

If your host provides phpMyAdmin, the easiest way to check MySQL is to look at the top left of the main phpMyAdmin screen: the MySQL version number is displayed there. If you don't have phpMyAdmin on your remote server, use `mysql_version.php` in the download files for this chapter. Insert the hostname, username, and password that your hosting company has given you for connecting to MySQL. Save the file, upload it to your site, and view it in a browser. It will display the version running on your server.

After you have checked your remote server settings, remove `mysql_version.php` and the `phpinfo()` script. Although the information may seem harmless, it could be of use to a potential attacker.

How PHP communicates with MySQL

Regardless of whether you use PHP's MySQL functions, the MySQL Improved functions, or PDO, the process always follows this sequence:

1. Connect to MySQL using the hostname, username, and password.
2. Select the database you want to work with (combined with 1 in MySQLI and PDO).
3. Prepare a SQL query.
4. Execute the query and save the result.
5. Extract the data from the result (usually with a loop).

Username and password are straightforward: they're the username and password of the accounts you have just created or the account given to you by your hosting company. But what about hostname? In a local testing environment it's `localhost`. What comes as a surprise is that MySQL normally uses `localhost` even on a remote server. This is because the database server is normally located on the same server as your website. In other words, the web server and MySQL are local to each other. However, if your hosting company has installed MySQL on a separate machine, it will tell you the address to use. The important thing to realize is that the MySQL hostname is not the same as your website domain name.

Let's take a quick look at how you connect to a MySQL server with each of the methods.

When using the original MySQL extension or MySQLI, some commands are followed by the rather foreboding `or die()`. *This stops the script from going any further if the command fails, and displays any error message that you have inserted between the parentheses. PDO requires a different approach because of the way it handles connection errors.*

Connecting with the original MySQL extension

You connect to the MySQL server with the `mysql_connect()` function, which takes three arguments: hostname, username, and password, like this:

```
$conn = mysql_connect($hostname, $username, $password) ➥
or die ('Cannot connect to MySQL server');
```

It doesn't matter whether you pass the arguments as variables or as literal strings. If the connection is successful, the function returns a reference to the connection, which can be stored as a variable.

After connecting, you need to select the individual database using `mysql_select_db()` like this:

```
mysql_select_db('phpsolutions') or die ('Cannot open database');
```

Connecting with the MySQL Improved extension

The MySQL Improved extension has two interfaces: procedural and object-oriented. The procedural interface is designed to ease the transition from the original MySQL functions. Since the object-oriented version is more compact, that's the version adopted here.

To connect to a MySQL server, you create a `mysqli` object by passing four arguments to `new mysqli()`: the hostname, username, password, and the name of the database. The `new` keyword tells PHP that you want to create an object. Don't worry if you're not familiar with object-oriented programming (OOP). For the most part, objects act like ordinary variables. The main difference is that objects have methods (functions) and properties (values), which are accessed using the `->` operator.

So this is how you would connect to the `phpsolutions` database:

```
$conn = new mysqli($hostname, $username, $password, 'phpsolutions') ➥
or die ('Cannot open database');
```

This stores the connection object as `$conn`.

Connecting with PDO

PHP Data Objects are similar to the MySQLI object-oriented interface, but require a slightly different approach. The most important difference is that, if you're not careful, a PDO displays your database username and password onscreen when it can't connect to the database. This is because a PDO uses a type of error handling called **exceptions**, which are new to PHP 5. Unless you **catch the exception**, PHP displays debugging information onscreen. This is great for testing purposes, but a disaster in an online situation. Fortunately, the way you catch an exception is very easy.

To create a connection to the MySQL server, you create a data object by passing the following three arguments to `new PDO()`:

- A string specifying the database type, the hostname, and the name of the database. The string must be presented in the following format:

  `'mysql:host=`*`hostname`*`;dbname=`*`databaseName`*`'`

- The username.
- The user's password.

When creating a PDO—or using any other code that might trigger an exception (the technical term is **throw** an exception)—you need to wrap it in a `try/catch` block. This works in the same way as `if... else`. PHP attempts to execute the code in the `try` block. If it works, fine. If it doesn't, it throws an exception for the `catch` block to . . . well, catch. The code looks like this:

```
try {
  $conn = new PDO("mysql:host=$hostname;dbname=phpsolutions", ➥
  $username, $password);
  }
catch (PDOException $e) {
  echo 'Error: '.$e->getMessage();
  exit;
  }
```

The `catch` keyword is followed by a pair of parentheses, which take two arguments: the type of exception (in this case, `PDOException`) and a variable to catch the exception. The variable can be anything, but the convention is to use `$e`.

In PHP 5.1.6, `PDOException` *is case-insensitive. However, there is a move to make PHP more case-sensitive, so it's a good idea to adhere to this mixture of uppercase and lowercase.*

The `catch` block uses the `->` operator, which tells PHP that you want to use a method (function) or property (variable) with a particular object. The `getMessage()` method (function) gets the error message generated by the exception. Because there's no point going any further, `exit` on the next line terminates the script. It's not obligatory to display the error message generated by the exception. You can put anything you like in the `catch` block. It may be more user-friendly to send the visitor to an error page using `header()`, as described in "Redirecting to another page" in Chapter 5.

Exceptions are an advanced subject that I wouldn't normally cover. However, failure to connect to the database with PDO throws an automatic *exception, so you* must *handle it in this way to prevent the exposure of your username and password. Other PDO errors don't throw exceptions, so this is the only place I'll use a* `try/catch` *block.*

Building a database connection function

Connecting to a database is a routine chore that needs to be performed in every page from now on; and routine tasks are often best left to functions and/or include files. If any of the details change, you need change them in one place only.

PHP Solution 11-1: Making a reusable database connector

The finished script is in the download files for this chapter. There are three versions—one each for the original MySQL extension (conn_mysql.inc.php), MySQL Improved (conn_mysqli.inc.php), and PDO (conn_pdo.inc.php).

1. In a blank file, insert the following code:

```
<?php
function dbConnect($type) {
  if ($type  == 'query') {
    $user = 'psquery';
    $pwd = 'fuji';
    }
  elseif ($type == 'admin') {
    $user = 'psadmin';
    $pwd = 'kyoto';
    }
  else {
    exit('Unrecognized connection type');
    }
  // Connection code goes here
  }
?>
```

   This is the basic skeleton for all versions of a function called dbConnect(), which takes a single argument: the type of connection you want. The if... elseif conditional statement checks the value of the argument and switches between the psquery and psadmin username and password as appropriate.

   If your remote server allows you only one username and password, you can omit the argument and the conditional statement, and just use the following code:

```
<?php
function dbConnect() {
  // Connection code goes here
  }
?>
```

2. Replace the Connection code goes here comment. The code differs according to which connection method you need to use, as described earlier.

   If you are using the original MySQL extension (PHP 4 and/or MySQL prior to version 4.1), use this:

```
$conn = mysql_connect('localhost', $user, $pwd) ➥
or die ('Cannot connect to MySQL server');
mysql_select_db('phpsolutions') or die ('Cannot open database');
return $conn;
```

If you are using MySQL Improved (PHP 5 and MySQL 4.1 or later), use this:

```
$conn = new mysqli('localhost', $user, $pwd, 'phpsolutions') ➥
or die ('Cannot open database');
return $conn;
```

If you are using PDO (PHP 5.1 and MySQL 4.1 or later), use this:

```
try {
  $conn = new PDO('mysql:host=localhost;dbname=phpsolutions', ➥
$user, $pwd);
  return $conn;
  }
catch (PDOException $e) {
  echo 'Cannot connect to database';
  exit;
  }
```

The script for each version simply encapsulates the connection code described in the preceding section and returns $conn, which is a reference to the database connection (MySQL) or the connection object (MySQLI and PDO).

3. Because this is an include file, make sure there are no new lines or whitespace before or after the PHP tags. Save the page in the includes folder. You can either use the same name as for the download file for your particular version or call it connection.inc.php.

Throughout the rest of the book, in scripts that are not specific to one particular connection method, I use the generic filename connection.inc.php *to refer to the file that contains the* dbConnect() *function. Make sure that you use the correct version for the database connection functions you're using.*

To use this function, include connection.inc.php, and call the function like this for the psquery user:

```
$conn = dbConnect('query');
```

For the psadmin user, call it like this:

```
$conn = dbConnect('admin');
```

Regardless of whether you are using the original MySQL extension, MySQLI, or PDO, $conn contains the correct type of connection to the phpsolutions database. To adapt this for any other database, change the username, password, and database name in connection.inc.php.

Finding the number of results from a query

Counting the number of results from a database query is useful in several ways. It's necessary for creating a navigation system to page through a long set of results (you'll learn how to do that in the next chapter). It's also important for user authentication (covered in Chapter 15). If you get no results from matching a username and password, you know that the login procedure should fail.

The original MySQL extension and MySQL Improved both have a convenient method of finding out the number of results returned by a query. However, this isn't available with PDO, so you need to take a different approach. If you're using PDO, skip ahead to PHP Solution 11-3.

PHP Solution 11-2: Counting records in a result set (MySQL and MySQLI)

As you work through this PHP Solution, you'll see just how similar the code is for the original MySQL extension and MySQL Improved. This makes transferring from one to the other very easy, but you also need to be careful not to mix the two styles. I have indicated the differences clearly in steps 4 and 5.

1. Create a new folder called `mysql` in the `phpsolutions` site root, and create a new file called `mysql.php` inside the folder. The page will eventually be used to display a table, so it should have a `DOCTYPE` declaration and an XHTML skeleton.
2. Include the appropriate connection file for MySQL or MySQLI above the `DOCTYPE` declaration, and create a connection to MySQL like this:

   ```
   <?php
   include('../includes/connection.inc.php');
   // connect to MySQL
   $conn = dbConnect('query');
   ?>
   ```

   You don't need administrative privileges for this exercise, so I have used the account that has only `SELECT` privileges.
3. Next, prepare the SQL query. Add this code immediately after the previous step (but before the closing PHP tag):

   ```
   // prepare the SQL query
   $sql = 'SELECT * FROM images';
   ```

   This means "select everything from the `images` table." The asterisk (*) is shorthand for "all columns."
4. Now execute the query.

   The original MySQL extension uses a function called `mysql_query()`, which takes the SQL query as an argument. The code looks like this (it goes immediately after step 3):

   ```
   // submit the query and capture the result
   $result = mysql_query($sql) or die(mysql_error());
   ```

The code for MySQL Improved is very similar. You apply the `query()` method to the connection object ($conn) using the -> operator, and pass the SQL query as an argument like this:

```
// submit the query and capture the result
$result = $conn->query($sql) or die(mysqli_error());
```

Note that both methods store the result in a variable, which I have imaginatively named `$result`. If there is a problem, the database server returns an error message, which can be retrieved using `mysql_error()` or `mysqli_error()`, depending on your method of connection. By placing this function between the parentheses of `or die()`, the script comes to a halt if there's a problem and displays the error message.

5. Assuming there's no problem, the variable `$result` now holds a reference to the number of records found by the SQL query.

If you're using the original MySQL extension, pass the variable holding the result to `mysql_num_rows()` like this (put the code immediately after the preceding step):

```
// find out how many records were retrieved
$numRows = mysql_num_rows($result);
```

In MySQL Improved, the number of results is held in the `num_rows` property of the result object (`$result`). You access it with the -> operator like this:

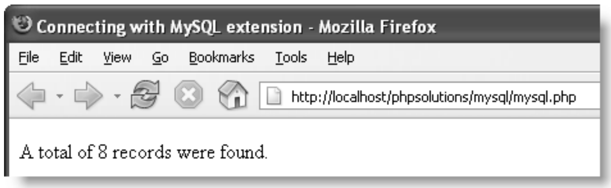
```
// find out how many records were retrieved
$numRows = $result->num_rows;
```

There are no parentheses following `num_rows` *in the MySQLI version. This is because it's a property (or value) of the result object. Functions and methods are followed by parentheses, but variables and properties are not.*

6. You can now display the value of `$numRows` in the body of the page like this:

```
<p>A total of <?php echo $numRows; ?> records were found.</p>
```

7. Save the page and load it into a browser. You should see the following result:


Connecting with MySQL extension - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/phpsolutions/mysql/mysql.php

A total of 8 records were found.

Check your code, if necessary, with `mysql01.php` or `mysqli01.php` in the download files.

PHP Solution 11-3: Counting records in a result set (PDO)

Because PDO doesn't have an equivalent of the MySQLI `num_rows` property or the MySQL function `mysql_num_rows()`, you need to use a SQL function called `COUNT()`.

1. Create a new folder called `mysql` in the `phpsolutions` site root, and create a new file called `pdo.php` inside the folder. The page will eventually be used to display a table, so it should have a `DOCTYPE` declaration and an XHTML skeleton.
2. Include the PDO connection file above the `DOCTYPE` declaration, and create a connection to MySQL like this:

```
<?php
include('../includes/conn_pdo.inc.php');
// connect to MySQL
$conn = dbConnect('query');
?>
```

   You don't need administrative privileges for this exercise, so I have used the account that has only `SELECT` privileges.
3. Next, prepare the SQL query. Add this code immediately after the previous step (but before the closing PHP tag):

```
// prepare the SQL query
$sql = 'SELECT COUNT(*) FROM images';
```

   This means "count every record in the `images` table." The asterisk (*) is shorthand for "all columns." The `COUNT()` function gets the total number of records. Make sure you don't leave a space between `COUNT` and the opening parenthesis, as this generates a SQL error.
4. Now execute the query and store the result in a variable like this (the code goes immediately after the code in step 3):

```
// submit the query and capture the result
$result = $conn->query($sql);
$error = $conn->errorInfo();
if (isset($error[2])) die($error[2]);
```

   `$conn` is the variable that you used to create the connection, so `$conn->query()` means "run this query with my connection." The result is stored in a variable, which I've named, rather predictably, `$result`.

   PDO uses `errorInfo()` to build an array of error messages from the database. The third element of the array is created only if something goes wrong. I've stored the result of `$conn->errorInfo()` as `$error`, so you can tell if anything went wrong by using `isset()` to check whether `$error[2]` has been defined. If it has, `die()` brings the script to a halt and displays the error message.
5. The SQL query in step 3 returns only one piece of information: the number of records found, so you can use the `fetchColumn()` method with `$result` to retrieve

it, and store the number of rows found like this (put the code immediately after the preceding step):

```
// find out how many records were retrieved
$numRows = $result->fetchColumn();
```

6. You can now display the value of $numRows in the body of the page like this:

```
<p>A total of <?php echo $numRows; ?> records were found.</p>
```

7. Save the page and load it into a browser. You should see the same result as shown in step 8 of PHP Solution 11-2. Check your code, if necessary, with pdo01.php.

Displaying the results of a query

In spite of the different ways that MySQL, MySQLI, and PDO communicate with a database, they all produce a result that contains all the information sent back from the database (and stored as $result). In PHP Solution 11-2, $result contains every field in every record. In PHP Solution 11-3, a different query was used because PDO handles the counting of records differently; but if you run the same query with PDO, $result also contains every field in every record.

It's tempting to think of this result as an array. In one sense, it is; but you can't use it in the same way as arrays that you have encountered so far. To extract the information, you need to deal with one record at a time. The most common way is to use a loop in combination with a function (or method) to extract the current record into a temporary array, which you can then use to display the information it holds.

With the MySQL extension, you do it like this:

```
while ($row = mysql_fetch_assoc($result)) {
  // do something with the current record
  }
```

With MySQLI, instead of passing $result to a function, you use the -> operator like this:

```
while ($row = $result->fetch_assoc()) {
  // do something with the current record
  }
```

PDO handles it slightly differently. You can use the query() method directly inside a foreach loop to create an array for each record like this:

```
foreach ($conn->query($sql) as $row) {
  // do something with the current record
  }
```

In each case, $row is an associative array containing every field in the current record. So, in the case of the images table, $row contains these three elements: $row['image_id'], $row['filename'], and $row['caption']. In other words, each element is named after the corresponding column in the table.

PHP Solution 11-4: Displaying the images table using MySQL

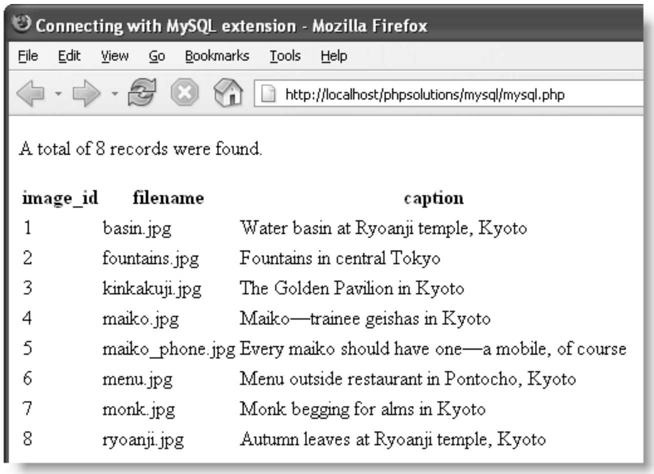
Continue using the file from PHP Solution 11-2. The finished code is in `mysql02.php`.

1. Add the following code to the main body of the page (new code is in bold):

```
<p>A total of <?php echo $numRows; ?> records were found.</p>
<table>
  <tr>
    <th>image_id</th>
    <th>filename</th>
    <th>caption</th>
  </tr>
<?php
while ($row = mysql_fetch_assoc($result)) {
?>
  <tr>
    <td><?php echo $row['image_id']; ?></td>
    <td><?php echo $row['filename']; ?></td>
    <td><?php echo $row['caption']; ?></td>
  </tr>
<?php } ?>
</table>
</body>
```

   The `while` loop iterates through the database result, using `mysql_fetch_assoc()` to extract each record into `$row`. Each element of `$row` is displayed in a table cell. The loop continues until `mysql_fetch_assoc($result)` comes to the end of the result set.

2. Save `mysql.php` and view it in a browser. You should see the contents of the images table displayed as shown in the following screenshot:


Connecting with MySQL extension - Mozilla Firefox
File Edit View Go Bookmarks Tools Help
http://localhost/phpsolutions/mysql/mysql.php

A total of 8 records were found.

image_id	filename	caption
1	basin.jpg	Water basin at Ryoanji temple, Kyoto
2	fountains.jpg	Fountains in central Tokyo
3	kinkakuji.jpg	The Golden Pavilion in Kyoto
4	maiko.jpg	Maiko—trainee geishas in Kyoto
5	maiko_phone.jpg	Every maiko should have one—a mobile, of course
6	menu.jpg	Menu outside restaurant in Pontocho, Kyoto
7	monk.jpg	Monk begging for alms in Kyoto
8	ryoanji.jpg	Autumn leaves at Ryoanji temple, Kyoto

PHP Solution 11-5: Displaying the images table using MySQLI

Continue using the file from PHP Solution 11-2. The finished code is in `mysqli02.php`.

1. Insert the same code into the body of the page as in step 1 of PHP Solution 11-4. However, replace this line:

   ```
   while ($row = mysql_fetch_assoc($result)) {
   ```

   with this:

   ```
   while ($row = $result->fetch_assoc()) {
   ```

2. Save the page and view it in a browser. It should look like the preceding screenshot.

PHP Solution 11-6: Displaying the images table using PDO

Because PDO doesn't have a convenient way of finding the number of records in a result set, you need to submit a second query to the database. It's also necessary to release the database resources associated with the first query.

Continue working with the same file as in PHP Solution 11-3. The finished script is in `pdo02.php`.

1. Amend the last section of code above the `DOCTYPE` declaration to release the database resource after the first query, and store the second query in a variable like this:

   ```
   // find out how many records were retrieved
   $numRows = $result->fetchColumn();
   // free the database resources
   $result->closeCursor();
   // prepare second SQL query
   $getDetails = 'SELECT * FROM images';
   ?>
   ```

   The `closeCursor()` method frees the connection to the database so that further queries can be executed. You apply it to the current result; not to the connection. The second query, stored in `$getDetails`, retrieves all the records in the `images` table.
2. Insert the same code into the body of the page as in step 1 of PHP Solution 11-4. However, replace this line:

   ```
   while ($row = mysql_fetch_assoc($result)) {
   ```

   with this:

   ```
   foreach ($conn->query($getDetails) as $row) {
   ```

3. Save the page and view it in a browser. It should look like the screenshot in PHP Solution 11-4.

MySQL connection crib sheet

Tables 11-2 to 11-4 summarize the basic details of connection and database query for MySQL, MySQLI, and PDO. Some commands will be used in later chapters, but are included here for ease of reference.

Table 11-2. Connection to MySQL with the original MySQL extension

Action	Usage	Comments
Connect	`$conn = mysql_connect($h,$u,$p);`	All arguments optional; first three always needed in practice: hostname, username, password.
Choose DB	`mysql_select_db('dbName');`	Server connection can be second, optional argument.
Submit query	`$result = mysql_query($sql);`	Requires one argument: string containing SQL query. Server connection can be second, optional argument. Returns result set.
Count results	`$numRows = mysql_num_rows($result);`	Takes result set as sole argument.
Extract record	`$row = mysql_fetch_assoc($result);`	Takes result set as sole argument. Extracts current record as associative array.
Extract record	`$row = mysql_fetch_row($result);`	Takes result set as sole argument. Extracts current record as indexed (numbered) array.

As noted in Table 11-2, you can use a reference to the database connection as an optional argument with `mysql_select_db()` and `mysql_query()`. There is no need to do this unless you are using more than one connection (say, with different usernames), because PHP automatically uses the most recent link opened by `mysql_connect()`.

Table 11-3. Connection to MySQL with the MySQL Improved object-oriented interface

Action	Usage	Comments
Connect	`$conn = new mysqli($h,$u,$p,$d);`	All arguments optional; first four always needed in practice: hostname, username, password, database name. Creates connection object.
Choose DB	`$conn->select_db('dbName');`	Use to select different database.
Submit query	`$result = $conn->query($sql);`	Returns result object.
Count results	`$numRows = $result->num_rows;`	Returns number of rows in result object.

Action	Usage	Comments
Release DB resources	`$result->free_result();`	Frees up connection to allow new query.
Extract record	`$row = $result->fetch_assoc();`	Extracts current row from result object as associative array.
Extract record	`$row = $result->fetch_row();`	Extracts current row from result object as indexed (numbered) array.

Table 11-4. Connection to MySQL with PDO

Action	Usage	Comments
Connect	`$conn = new PDO(DSN,$u,$p);`	In practice, requires three arguments: data source name (DSN), username, password. Must be wrapped in `try/catch` block.
Choose DB	See comments	Choice of database is integral part of DSN.
Submit query	`$result = $conn->query($sql);`	Can also be used inside `foreach` loop to extract each record.
Count results	See comments	Use `SELECT COUNT(*)` in SQL query.
Get single result	`$item = $result->fetchColumn();`	Gets first record in first column of result. To get result from other columns, use column number (from 0) as argument.
Get next record	`$row = $result->fetch();`	Gets next row from result set as associative array.
Release DB resources	`$result->closeCursor();`	Frees up connection to allow new query.
Extract records	`foreach($conn->query($sql) as $row) {`	Extracts current row from result set as associative array.

When using PDO with MySQL, the data source name (DSN) is a string that takes the following format:

```
'mysql:host=hostname;dbname=databaseName'
```

If you need to specify a different port from the MySQL default (3306), use the following format, substituting the actual port number:

```
'mysql:host=hostname;port=3307;dbname=databaseName'
```

MySQL Improved and PDO also use prepared statements, which offer greater security when incorporating user input into SQL queries. Prepared statement commands are covered in Chapter 13.

Summary

It's unfortunate that connection to MySQL is in such a transitional phase. Because PDO is so new, it may undergo further changes, but along with MySQLI, it has significant advantages over the original MySQL extension, particularly in improved protection against malicious attacks. However, I suspect that a high proportion of readers will have no option other than to use the traditional method of connecting to MySQL. The good news is that, for the foreseeable future at least, PHP plans to continue support for all three options. This means that even when you move to a server that supports PDO, your MySQL or MySQLI scripts will still work.

In the next chapter, we'll turn those boring lists of filenames and captions into something a lot more attractive—an online mini photo gallery. See you there.