

Contents at a Glance

CHAPTER 1: Introducing MySQL	445
Seeing the Purpose of a Database	445
Presenting MySQL	454
Advanced MySQL Features	458
CHAPTER 2: Administering MySQL	465
MySQL Administration Tools	465
Managing User Accounts	477
CHAPTER 3: Designing and Building a Database	489
Managing Your Data	489
Creating Databases	492
Building Tables	500
CHAPTER 4: Using the Database	513
Working with Data	513
Searching for Data	524
Playing It Safe with Data	531
CHAPTER 5: Communicating with the Database from PHP Scripts	541
Database Support in PHP	541
Using the mysqli Library	543
Putting It All Together	554

IN THIS CHAPTER

- » Understanding why you need a database
- » Seeing how MySQL works
- » Exploring the advanced features of MySQL

Chapter 1

Introducing MySQL

Computers are all about storing information. However, unlike that junk drawer in your kitchen that contains multiple shards of paper with names and phone numbers scribbled on them, you want to store your dynamic web application data in an orderly fashion. After all, you wouldn't want to mix up the data from your astrophysics experiments with your bowling league scores!

The MySQL database server provides a user-friendly platform for you to organize your application data, making it simple to identify which data belongs to which application and easy for the application to access the data, all while maintaining security so the right people can only get to the right data. This chapter describes just why you need a database for your dynamic web applications, and why you should choose the MySQL database server.

Seeing the Purpose of a Database

With PHP, you have a few different options for storing persistent data in your application to retrieve at a later time. One method is to use the PHP file system functions to create a standard text file on the server to store the application data, and then read the data back as necessary.

One downside to using standard text files to store your application data is that it's hard to find a specific data item buried in the text file. Standard text files are often

called “flat files” because you can’t create any type of relationships in the data to make searching for specific information easier. Your application must open the text file and read each line until it finds the data it needs. That’s fine for small amounts of data, but for large amounts of data that can be slow, especially if there are thousands of site visitors all trying to access their data from the same file at the same time.

To solve that problem, most web developers have turned to using databases. Databases organize data in a manner making it easier for the database server to insert, find, modify, and delete data. There are lots of different database types available, but one of the most popular is the relational database system. This section describes how relational databases work with data to help speed up your web application.

How databases work

Microsoft Access is by far the most popular end-user database tool developed for commercial use. Many Windows users, from professional accountants to bowling league secretaries, use Access to track data. It provides an easy, intuitive user interface, allowing novice computer users to quickly produce queries and reports with little effort.

However, despite its user-friendliness, Access has its limitations. To fully understand how MySQL differs from Access, you must first understand how database systems are organized.

There is more to a database than just a bunch of data files. Most databases incorporate several layers of files, programs, and utilities, which all interact to provide the database experience. The whole package is referred to as a *database management system* (DBMS).

There are different types of DBMS packages, but they all basically contain the following parts:

- » A database engine
- » One or more database files
- » An internal data dictionary
- » A query language interface

The *database engine* is the heart and brains of the DBMS. It controls all access to the data, which is stored in the database files. Any application (including the DBMS itself) that requires access to data must go through the database engine (see Figure 1-1).

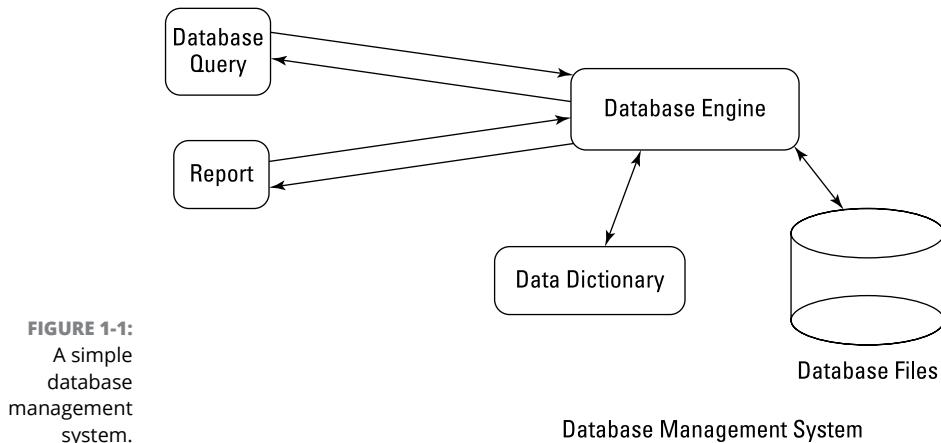


FIGURE 1-1:
A simple
database
management
system.

Database Management System

The database engine uses an internal *data dictionary* to define how the database operates, the type of data that can be stored in the database files, and the structure of the database. It basically defines the rules used for the DBMS. Each DBMS has its own data dictionary.

If you're a user running a simple database on Access, you probably don't even realize you're using a database engine. Access keeps much of the DBMS work under the hood and away from users. When you start Access, the database engine starts, and when you stop Access, the database engine stops.

In MySQL, the database engine runs as a service that is always running in the background on the server. Users run separate application programs that interface with the database engine while it's running. Each application can send queries to the database engine and process the results returned. When the application stops, the MySQL database engine continues to run, waiting for commands from other applications.

Both Access and MySQL require one or more database files to be present to hold data. If you work with Access, you've seen the .mdb database files. These files contain the data defined in tables created in the Access database. Each database has its own .mdb file.

In the Access environment, if two or more applications want to share a database, the database file must be located on a shared network drive available to all the applications. Each application has a copy of the Access database engine program running on the local workstation, which points to the common database file, as shown in Figure 1-2.

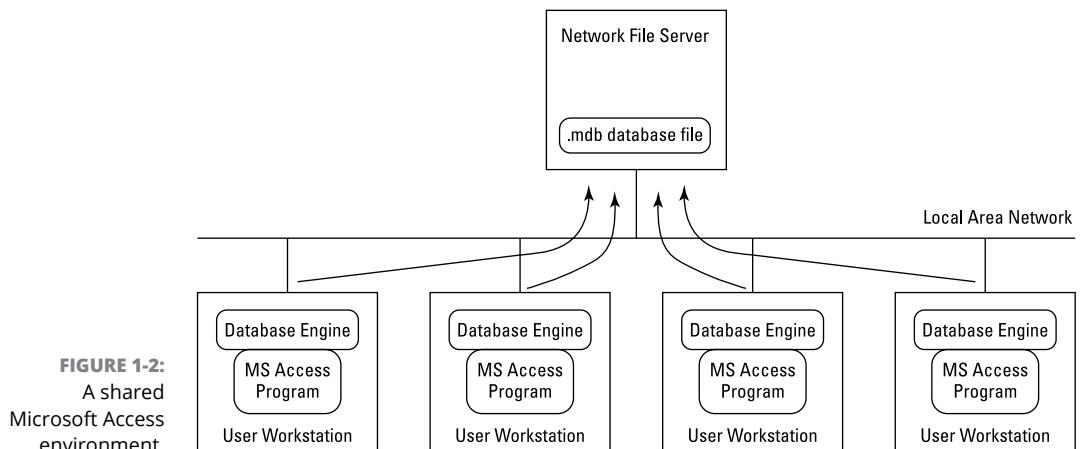


FIGURE 1-2:
A shared
Microsoft Access
environment.

Where this model falls apart is that there are multiple database engines, all trying to access the database files across a network environment. This generates large amounts of data on the network and slows down the performance of the individual database engines.

In the MySQL model, the database engine and database files are always on the same computer. Queries and reports run from separate applications, but they all send requests to the common database engine, as shown in Figure 1-3.

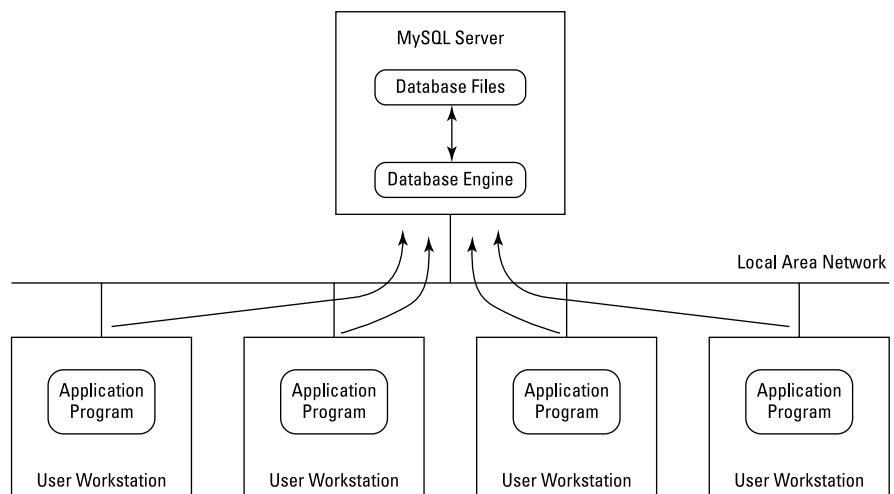


FIGURE 1-3:
A multiuser
MySQL
environment.

As you can see from Figure 1-3, the MySQL database engine accepts data requests from multiple users across the network. All the database access is performed on the local system running the MySQL server, so the data interaction with the

database files stays on the local system. The database engine only sends the query or report results across the network to the applications.

This feature alone makes using MySQL a better database choice in multiuser database projects.

Relational databases

Databases are all about arranging data to make finding information faster. *Relational database theory* arranges data in three levels: databases, tables, and data fields.

Databases

A *database* groups related data into a single container. The database is the highest level or grouping of data on the relational database server. The server allows you to create multiple databases, all accessible from the same server service running on the server.



TIP

To help keep things organized, it's a good idea to create a separate database for each application you're hosting on the server. This helps to separate data elements and eliminates accidents caused by accessing the wrong data from the wrong application.



REMEMBER

Each database you create must have a unique name on the server. To help with the organization process, it's usually a good idea to somehow relate the database name to the name of the application.

Table

The *table* is a subset of data within the database, which contains a grouping of similar data items. For example, if a company wants to track data on employees, customers, and products, instead of having just one group of mixed-up data elements, the company would create four separate tables to hold the data:

- » An Employees table to hold data related to employees
- » A Customers table to hold data related to customers
- » A Products table to hold data related to products
- » An Orders table to track which products are in individual customer orders

The process of grouping application data into tables is called *data normalization*. Grouping similar data into its own table gives you more control over the data. For

example, if you have a program that interfaces only with customer orders, you can give it permissions to only the Customers, Products, and Orders tables, leaving the Employees table safe from accidental exposure.

Data fields

You use *data fields* to hold individual data elements within a table. For example, the Employees table might contain data fields for an employee ID number, first name, last name, home address, salary, and employment start date. The data fields are the core of the application because they're where the application actually stores data.

The table groups data fields into *data records*. Each data record is a single occurrence of values for each of the data fields. Figure 1-4 shows a diagram of how the Employees table might look.

employeeid	lastname	firstname	address	salary	startdate
143	Smith	John	123 Main St; Chicago, IL	45000	4/30/1986
219	Jones	Fred	33 Oak Road; New York, NY	37500	1/4/1990
312	Brown	Sara	221 Pine St; Miami, FL	67000	4/13/1993

FIGURE 1-4:
An example of
an Employees
table layout.

Figure 1-4 shows the data fields as table headings. Each data record appears as a single line of data in the table (in this case, the information for a single employee). Because data is often displayed this way in a table, you'll often hear the word *row* used to reference a single data record.

Database data types

Just as with variables in programming languages, databases need to identify the type of data stored in each data field so that it knows how much space to reserve to store the data, and how to handle the data. Table 1-1 shows the basic data types found in most relational database systems.

TABLE 1-1 Standard Database Data Types

Data Type	Description
int	A whole number between -2,147,483,648 and 2,147,483,647
float	A floating point number between -3.40283466E+38 and +3.40283466E+38
bool	A Boolean true or false value
date	A day value in the YYYY-MM-DD format
datetime	A day and time value displayed in YYYY-MM-DD HH:MM:SS format
char(<i>x</i>)	A fixed-length character string with <i>x</i> characters
varchar(<i>x</i>)	A variable-length character string with <i>x</i> or fewer characters
text	A variable-length character string of up to 65,536 characters stored as a binary value

Many relational database servers provide variations of these standard data types, such as small integer values or large text values, to help you customize exactly how much space to reserve for each data field. Unfortunately, these customized data types aren't necessarily standardized between relational databases.

Data constraints

Besides the data field name and value, a data field can be marked with special *data constraints*. Relational databases use data constraints to control how you place data into a data field. The most popular data constraint is the *primary key*.

A primary key defines the table data field(s) that uniquely identify each individual data record in the table. For example, if you're retrieving an employee data record and your company has two employees named John Smith, you'll run into a problem trying to get the correct data for the correct employee. To solve this problem,

relational databases allow you to add a special data field to tell you which John Smith each data record refers to.

To do this, you must create an employee ID data field and assign a unique ID number to each employee. Because the new employee ID data field uniquely identifies each employee record, you can specify it as the primary key for the Employees table. The database server creates a separate hidden table relating the primary key values to data record numbers, and then uses it as an index to quickly retrieve the correct data record based on the primary key value.

Another popular data constraint you'll run across is the `is null` restriction. If you set a data field with the `is null` data constraint, the database server will prevent you from entering a data record without a value in that data field.

Structured Query Language

The Structured Query Language (SQL) is a language for interacting with relational database systems that been around since the early 1970s. Over the years, other database vendors have tried to mimic or replace SQL with their own query languages. But despite their attempts, SQL still provides the easiest interface for both users and administrators to interact with any type of relational database system.

In 1986, the American National Standards Institute (ANSI) created the first SQL standards. The U.S. government adopted them as a federal standard and named it ANSI SQL89. Most commercial database vendors now use this SQL standard to interface with their products.



TIP

The SQL standard has been evolving over the years, with new standards being released to support new advanced database features. At the time of this writing, the most current standard is SQL:2016.

The SQL language specifies a format that you use to send commands to the database server. The SQL command format consists of:

» A keyword: SQL *keywords* define the action the database server takes based on the SQL statement. The SQL standard defines lots of different keywords for performing lots of different actions. However, you'll find yourself just using a few standard keywords in your database programming, so it's not all that hard to remember them. Table 1-2 lists the popular ones you'll get to know.

TABLE 1-2

SQL Keywords

Keyword	Description
DELETE	Removes a data record from a table
DROP	Removes a table or database
INSERT	Adds a new data record to a table
SELECT	Retrieves data records from a table
UPDATE	Modifies data within an existing data record in a table

- » **An identifier:** The SQL command *identifier* defines the database object used in a command. This is most often a database name, table name, or the names of data fields. The SQL identifiers help you select which data elements to retrieve from the database and which table to select them from.
- » **One or more literals (optional):** SQL command *literals* define specific data values referenced by the keyword. Literals are constant values, such as data values to insert into a table or data values used to search within the table data. You must enclose string literals in quotes (either single or double quotes), but you can use numerical values without quotes.

The most common SQL command you'll use in your web applications is the *query*. A query is a SQL SELECT statement that searches the database for specific data records. Here's the basic format of a SELECT statement:

```
SELECT datafields FROM table
```

The *datafields* parameter is a comma-separated list of the data field names you want the query to return. If you want to retrieve all the data field values for the data records, you use an asterisk as a wildcard character.

You must also specify the specific *table* you want the query to search. To get meaningful results, you must match your query data fields with the proper table.



TIP

SQL keywords are often identified with all capital letters in a SQL statement. MySQL allows you to use either uppercase or lowercase for keywords. I use all capitals in this book to help you identify the keywords within the SQL statements.

By default, the SELECT statement returns all the data records in the specified table. You can use one or more *modifiers* to define how MySQL returns the data requested by the query. Table 1-3 shows the more popular modifiers you'll run into with SQL queries.

TABLE 1-3

SQL Query Modifiers

Modifier	Description
LIMIT	Displays only a subset of the returned data records
ORDER BY	Displays data records in a specified order
WHERE	Displays a subset of data records that meet a specified condition

The WHERE clause is the most common SELECT statement modifier. It allows you to specify conditions to filter data from the table. For example:

```
SELECT lastname FROM Employees WHERE salary > 100000;
```

This SELECT statement only returns the last name of the employees with a salary of over \$100,000.



TIP

Having to use SQL to interact with a database server can seem a bit overwhelming at first — you have to learn an entirely new programming language besides the languages you’re learning to build your dynamic web application. Don’t fret, though. There are really only a handful of SQL statements that you’ll regularly use during the course of your application development. You’ll start remembering them in no time.

Presenting MySQL

The specific relational database server that I discuss in this book is the MySQL database server. The MySQL server is the most popular database server used in web applications — and for good reason. The following sections describe the features of the MySQL server that make it so popular.

MySQL features

The MySQL database server was created by David Axmark, Allan Larsson, and Michael Widenius as an upgrade to the mSQL database server and was first released for general use in 1996. It’s now owned and supported by Oracle but released as open-source software.

MySQL was originally created to incorporate indexing data to speed up data queries in the mSQL database server, by using the indexed sequential access method (ISAM). It did this by incorporating a special data management algorithm called the MyISAM storage engine. This proved to be a huge success.

MySQL was initially recognized for its speed of accessing data. The MyISAM data storage and indexing method proved to be a game changer in speeding up data access from other types of DBMS packages. It wasn't long before the Internet world took notice, and MySQL became the DBMS package of choice for high-volume web applications.

These days, MySQL has evolved to do more than just fast data queries. Development is continually ongoing to add new features to MySQL. A short list of features includes the following:

- » It was written in C and C++ and has been compiled to run on many different platforms.
- » It incorporates a modular design approach to create a multi-layer server design.
- » It supports multi-threading, making it easily scalable to incorporate multiple CPUs if available.
- » It uses a thread-based memory allocation system.
- » It implements hash tables in memory to increase performance.
- » It supports client/server and embedded server environments.
- » It supports multiple data storage engines.
- » It implements all SQL functions using a class library.
- » It includes support for all standard SQL data types.
- » It offers a security system that supports both user-based and host-based verification.
- » It includes support for large databases using more than 5 billion rows of data.
- » It provides application programming interfaces (APIs) for many common programming languages (including PHP).
- » It incorporates many different character sets, allowing it to support many different languages.
- » It provides both command line and graphical tools for common database management.

Of these features, let's take a closer look at two specific features to demonstrate the versatility of MySQL. The following sections dive into the ability for MySQL to support different database storage types, as well as how MySQL handles user authentication.

Storage engines

As shown in the preceding section, the MySQL server uses a modular approach to building the database server. One of those modules is how it stores and accesses database data. This is called the *storage engine*.

The storage engine is the gatekeeper to your data and all requests to your data go through it. The MySQL server incorporates several different types of storage engines, shown in Table 1-4.

TABLE 1-4 The MySQL Storage Engines

Storage Engine	Description
Archive	Produces a special-purpose table for inserting and retrieving data, but not updating or deleting it.
Blackhole	Accepts data but does not store it. Used for development testing.
CSV	Stores data in a comma-separated file format.
Federated	Allows data access from a remote server without using replication.
Example	A storage engine that does nothing. Used as a template for storage engine developers.
InnoDB	An advanced storage engine that balances high reliability and high performance.
Memory	Stores all data in memory for fast performance, but it doesn't retain the data.
MyISAM	The initial MySQL storage engine, known for being fast with few advanced features.

The MyISAM storage engine is what made MySQL famous, but it's no longer being developed by Oracle. The default and recommended storage engine for MySQL is now the InnoDB storage engine.

The InnoDB storage engine supports many advanced database features found in commercial databases, but initially it was known for not being all that fast. Developers had to decide which was more important to their application: performance or fancy database features.

However, work has been done by the MySQL developers to increase the performance of the InnoDB storage engine so that it comes close to the performance of the MyISAM storage engine. This gives you the best of both worlds — advanced database features and a high-performance storage engine, all as open-source software!

Data permissions

The MySQL database server handles access to database data using a two-tiered approach:

- » The user account assigned to a user
- » The location from where the user connects to the server

MySQL considers your identity from both the user account you use to log into the system, as well as the host from which you connect. That means you can control access to your data not only to specific user accounts, but from where the users happen to be when they log into the database server. For example, you can give a user account full access to a database when she logs in from the local server but restricted read-only access to the database when she logs in from a remote server.

MySQL does this by using an *access control list* (ACL) to define permissions to databases, tables, and special features based on the identities. When you create an identity in MySQL, you not only create a user account, but also specify the location from which the access control applies. You can use wildcards to allow users to have the same permissions from multiple locations.

MySQL uses a two-stage approach to verifying your database connection. First, MySQL accepts or rejects the connection request based on the user ID/password combination provided and whether the account is locked on the system. Then, if the connection is granted, MySQL accepts or rejects the access request based on database and table permissions.

A user account can have access to the database server, but not every database on the server. You can create separate user accounts for each application database that you create on the MySQL server. If your application requires more control, you can even create separate user accounts that have access to only certain tables within the same database!

As the database administrator you also have the ability to grant system-level privileges to user accounts, such as the ability to create new databases or even new user accounts.



WARNING

The MySQL server has a single main administration user account named *root*. If you forget the password to the root user account that may or may not be recoverable, depending on your server setup and environment. It's always a good idea to keep track of the root user account's password, but also to protect it so that no one else can use it. If your system requires multiple administrators, give them each a separate user account and grant those user accounts elevated privileges on the database server so they can create databases and user accounts as needed.

Advanced MySQL Features

When you use the default InnoDB storage engine in MySQL, you have a wealth of advanced database feature available for your applications to utilize. This section walks through the more advanced features that the InnoDB storage engine brings to the MySQL world.

Handling transactions

All database servers allow users to enter database commands to query and manipulate data. What separates good database servers from bad ones is the way they handle commands.

The database engine processes commands as a single unit, called a *transaction*. A transaction represents a single data operation on the database. Most simplistic database servers treat each command received — such as adding a new record to a table or modifying an existing record in a table — as a separate transaction. Groups of commands create groups of transactions.

However, some advanced database servers (such as the MySQL with the InnoDB storage engine) allow you to perform more complicated transactions. In some instances, it's necessary for an application to perform multiple commands as a result of a single action.



REMEMBER

In a relational database, tables can be related to one another. This means that one table can contain data that is related to the data in another table. In the store example, the Orders table relied on data in both the Customers and Products tables. Although this makes organizing data easier, it makes managing transactions more difficult. A single action may require the database server to update several data values in several different tables.

In the store example, if a new customer comes into the store and purchases a laptop computer, the database server must modify three tables:

- » Add a new data record to the Customers table
- » Add a new data record to the Orders table
- » Modify the Products table to subtract one from the laptop inventory value

For the action to be complete, all three of these actions must succeed. If any one of the actions fails, the data will become corrupt. In an advanced database server, you can combine all these actions into a single transaction. If any one of the actions fails, the database server rolls back the other two actions to return

the database to the previous condition. This feature is crucial to have available for your web applications!

Making sure your database is ACID compliant

Over the years, database experts have devised rules for how databases should handle transactions. The benchmark for all professional database systems is the ACID test. No, we're not throwing the server into an acid bath; the ACID test is actually an acronym for a set of database features defining how the database server should support transactions:

- » Atomicity
- » Consistency
- » Isolation
- » Durability

The following sections examine these four features and discuss how MySQL implements them.

Atomicity

The atomicity feature states that for a transaction to be considered successful, all steps within the transaction must complete successfully. Either all the steps should be applied to the database, or none of them should. A transaction should not be allowed to complete partway.

To support atomicity, MySQL uses a system called *commit and rollback*. Database actions are only temporarily performed during a transaction. When it appears that all the actions in a transaction would complete successfully, the transaction is committed (the server applies all the actions to the database). If it appears that any one of the actions would fail, the entire transaction is rolled back (any previous steps that were successful are reversed). This ensures that the transaction is completed as a whole.

MySQL uses the two-phase commit approach to committing transactions. The two-phase commit performs the transaction using two steps (or phases):

- » **Prepare phase:** A transaction is analyzed to determine if the database is able to commit the entire transaction.
- » **Commit phase:** The transaction is physically committed to the database.

The two-phase commit approach allows MySQL to test all transaction commands during the prepare phase without having to modify any data in the actual tables. Table data is not changed until the commit phase is complete.

Consistency

The concept of consistency is a little more difficult than atomicity. The consistency feature states that every transaction should leave the database in a valid state. The tricky part here is what is considered a “valid state.”

Often, this feature is applied to how a database server handles unexpected crashes. If the database takes a power hit in the middle of the commit phase of a multi-action transaction, can it leave the tables in a state where the data makes sense?

MySQL utilizes two features to accomplish consistency:

- » **Double-write buffering:** With double-write buffering, before MySQL writes data to the actual tables, it stores the data in a buffer area. Only after all the transaction data is written to the buffer area will MySQL write the buffer area data to the actual table data files.
- » **Crash recovery:** If there is a system crash before the buffer area is completely written to the table files, MySQL can recover the buffer area using the crash recovery feature, which recovers submitted transactions from a transaction log file.

Isolation

The isolation feature is required for multiuser databases. When there is more than one person modifying data in a database, odd things can happen. If two people try to modify the same data value at the same time, who's to say which value is the final value?

When more than one person tries to access the same data, the DBMS must act as the traffic cop, directing who gets access to the data first. Isolation ensures that each transaction in progress is invisible to any other transaction in progress. The DBMS must allow each transaction to complete and then decide which transaction value is the final value for the data. It accomplishes this task using a feature called *locking*.

Locking does what it says: It locks data while a transaction is being committed to the database. While the data is locked, other users can't access the data, not even for queries. This prevents multiple users from querying or modifying the data while it's in a locked mode.

There are two basic levels of locking that MySQL uses to support isolation:

- » **Table-level locking:** With table-level locking, any time a user requires a modification to a data record in a table, the DBMS locks the entire table, preventing other users from even viewing data in the table. As you can guess, this has an adverse effect on database performance, especially in environments where there is a lot of change to the data in the database. Early DBMS implementations used table-level locking exclusively.
- » **Row-level locking:** To solve the problems of table-level locking, many DBMS implementations (including the MySQL InnoDB storage engine) now incorporate row-level locking. With row-level locking, the DBMS locks only the data record that's being modified. The rest of the table is available for other users to access.

Durability

The durability feature states that when a transaction is committed to the database, it must not be lost. This sounds like a simple concept, but in reality durability is often harder to ensure than it sounds.

Durability means being able to withstand both hardware and software failures. A database is useless if a power outage or server crash compromises the data stored in the database.

MySQL supports durability by incorporating multiple layers of protection. The same double-write buffer and crash recovery features mentioned for the consistency feature also apply to the durability feature. MySQL writes all transactions to a log file, writes the changes to the double-write buffer area, and then writes them to the actual database files. If the system crashes during this process, most of the time MySQL can recover the transaction within the process.



TIP

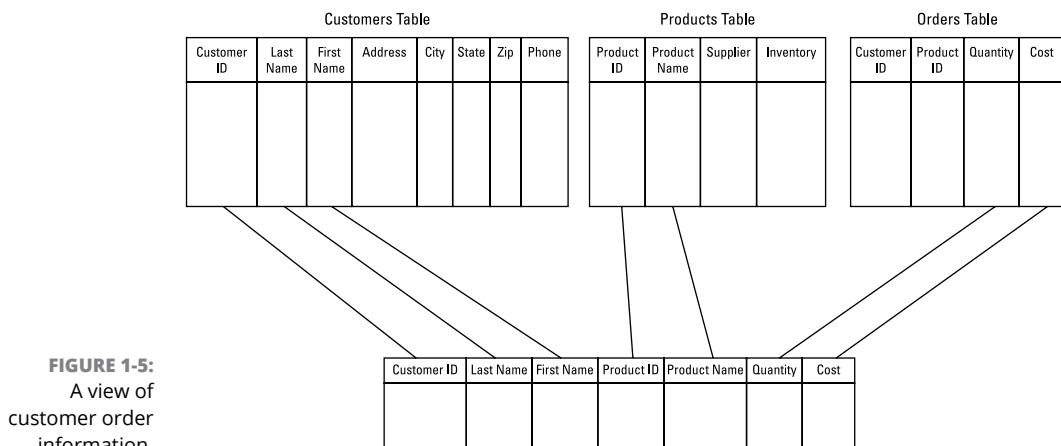
The onus of durability also rests on the database administrator. Having a good uninterrupted power supply (UPS) for your database server, as well as performing regular database backups, is crucial to ensuring your database tables are safe.

Examining the views

The SQL programming language provides developers with the ability to create some pretty complex queries, retrieving data from multiple tables in a single SQL statement. However, for queries that span more than a couple of tables, the SQL statement can become overly complex.

To help simplify complex query statements, some DBMS packages (including MySQL) allow administrators to create views. A view allows you to see (or view) data contained in separate database tables as if it were in a single table. Instead of having to write a sub-select query to grab data from multiple places, all the data is available in a single table view.

To a query, the view looks like any other database table. The DBMS can query views just like normal tables. A view does not use any disk space in the database files, because the DBMS generates the data in the view “on the fly” when a query tries to access the data. When the query is complete, the view data disappears. Figure 1-5 shows a sample view that you could create from the store database example.



The view shown in Figure 1-5 incorporates some of the customer data from the Customers table, product data from the Products table, and order data from the Orders table. Queries can access all the fields in the view as if they belonged to a single table.



WARNING

You can always use a view to read data, but you may or may not be able to use the view to insert new data or update existing data. It depends on the relationship between the data fields in the view. Data fields related in a one-to-one relationship can be inserted or updated, but data fields related in a one-to-many relationship can't.

Working with stored procedures

A *stored procedure* is a set of SQL statements that are commonly used by applications. Instead of each application needing to submit the multiple SQL statements,

you can create a stored procedure that contains the SQL statements and each application just needs to run the stored procedure.

Stored procedures can also help with the performance of the application, because less information needs to be sent between the client and the server (especially for long procedures). Stored procedures also allows you to create your own library of common functions in the database server to share among multiple applications. This helps you performance-tune queries and ensure all the applications use the same procedure to retrieve the data.

Pulling triggers

A *trigger* is a set of instructions that the DBMS performs on data based on an event in the table that contains the data. Events that can trigger the instructions are inserts, updates, or deletions of data contained in one or more tables. Here are the most common triggers you'll see:

- » AFTER DELETE: Perform the set of instructions after a data record has been deleted from the table.
- » BEFORE DELETE: Perform the set of instructions before a data record is deleted from the table.
- » AFTER INSERT: Perform the set of instructions after a data record has been inserted into the table.
- » BEFORE INSERT: Perform the set of instructions before a data record is inserted into a table.
- » AFTER UPDATE: Perform the set of instructions after a data record is updated in the table.
- » BEFORE UPDATE: Perform the set of instructions before a data record is updated in the table.

Triggers help you maintain data integrity within your database tables by monitoring when data is changed and having the ability to change related data at the same time.

Working with blobs

Most database users are familiar with the common data types that you can store in a database. These include integers, floating point numbers, Boolean values, fixed-length character strings, and variable-length character strings. However, in the modern programming world, support for lots of other data types is necessary. It's

not uncommon to see web applications that are used to store and index pictures, audio clips, and even short videos. This type of data storage has forced many professional databases to devise a plan to store different types of data.

MySQL uses a special data type called the binary large object (BLOB) to store any type of binary data. You can enter a BLOB into a table the same as any other data type. This allows you to include support for any type of multimedia storage within applications and still use all the fast retrieval and indexing methods of the database.



WARNING

Just because you can save large binary files in your tables doesn't mean that it's necessarily a good idea to do it. Large binary files can quickly fill a database disk space and slow down normal database queries. You'll need to analyze your particular application requirements to determine if it's better to store binary data inside the database or store the binary data outside as standard files, with just a pointer to the filename stored in the database.

IN THIS CHAPTER

- » Working from the command line
- » Using MySQL Workbench
- » Administering the server from the web
- » Creating user accounts
- » Assigning database privileges to users

Chapter 2

Administering MySQL

As you can tell from the previous chapter, the MySQL database server is a crucial component in your dynamic web applications. It's important that you know how to interact with the MySQL database server to create the database objects and user accounts required for your application. This chapter examines the different methods you have available for interacting directly with the MySQL database server in your application environment.

MySQL Administration Tools

There are lots of different tools available for interacting with a MySQL server to help manage your database environment. Over the years, three particular tools have risen to the top to be the most popular:

- » The MySQL command-line utilities
- » The MySQL Workbench graphical tool
- » The phpMyAdmin web-based tool

All these methods allow you to create, modify, and remove database objects in the server, manage user accounts and privileges, and perform standard database maintenance tasks such as backups and restores. They just all happen to use different environments to do that.

This section walks through the basics of these tools, showing you how to use them to perform basic administration functions on the MySQL database server.

Working from the command line

Just about everything these days uses some type of graphical interface, but the MySQL project still provides a method for interacting with the database directly from a text command line in the Windows, Mac, and Linux environments. That may seem old-fashioned, but the command line can often provide a handy interface for quickly entering commands. It's also great to use in emergencies — you never know when you'll find yourself working in a situation where the command line is all you have to work with!

This section walks through how to perform standard database administration functions with the MySQL server using just the command line.

Command-line tools

MySQL offers many scripts and programs that provide different ways for you to interact with the MySQL server in a command-line environment. Table 2-1 lists the command-line tools you'll find in your MySQL server installation.

TABLE 2-1 MySQL Command-Line Tools

Tool	Description
<code>innoschecksum</code>	Checks for damaged MyISAM storage engine files
<code>l4z_decompress</code>	Expands a <code>mysqlpump</code> archive file
<code>myisam_ftdump</code>	Displays information about full text indexes in MyISAM files
<code>myisamchk</code>	Repairs corrupt MyISAM storage engine files
<code>myisamlog</code>	Displays the contents of the MyISAM log file
<code>myisampack</code>	Compresses MyISAM storage engine table files
<code>mysql</code>	Provides an interactive command-line interface to the MySQL server
<code>mysqld</code>	The main MySQL database server program
<code>mysqld_multi</code>	Manages multiple <code>mysqld</code> server processes on a server
<code>mysqld_safe</code>	MySQL server startup script for Linux and Unix systems
<code>mysql.server</code>	MySQL server startup script for Mac systems
<code>mysqladmin</code>	Command-line administration tool

Tool	Description
<code>mysqlbinlog</code>	Parses binary log files
<code>mysqlcheck</code>	Analyzes, optimizes, and repairs MySQL tables
<code>mysqldump</code>	Performs a database backup
<code>mysqldumpslow</code>	Parses the MySQL slow query log
<code>mysqlimport</code>	Loads data from a file into a database
<code>mysqlpump</code>	Generates a SQL file to migrate a database to another SQL server
<code>mysqlsh</code>	A MySQL shell for creating scripts
<code>mysqlshow</code>	Displays database, table, and data field information
<code>mysqlslap</code>	Emulates client load on a MySQL server
<code>perror</code>	Displays a text description from a MySQL error code number
<code>zlib_decompress</code>	Expands compressed output from the <code>mysqlpump</code> command

As you can see in Table 2-1, there are quite a few command-line utilities that MySQL provides to help you out as a database administrator. Most likely, you'll never use most of them, but it's good to know they're there (and what they do) in case you ever need them.

For most normal interactions with the MySQL server, you'll use the `mysql` command-line program, which is discussed in the next section.

Exploring the MySQL client tool

The `mysql` command provides a text-based interactive interface (commonly called a *command-line interface*, or CLI) to the MySQL server. When you start the command, you'll get an interactive prompt:

```
C:\xampp\mysql\bin>mysql --user=root --password
Enter password:
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 4
Server version: 10.1.28-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]>
```

From the > prompt, you can submit SQL statements directly to the server to interact with the databases contained on the server. There are also some built-in commands available in the CLI to help manage your database objects.

The mysql command provides a lot of command-line parameters that allow you to customize what it does when you start it. You can use the -? parameter to display all the available parameters and what they do. There are lots of parameters that provide a lot of features that, again, you'll most likely never use. Usually the only parameters you'll need to worry about are --user and --password.

These parameters allow you to specify the user ID and password to use when connecting to the MySQL server (by default the mysql command attempts to connect to the MySQL server using the user account of the currently logged-in user). These days it's not a good idea to enter your password in plain text on the command line. If you use the --password parameter by itself (without a specified value), the mysql command prompts you to enter your password as a hidden entry, as I did in the preceding example.

Occasionally, you may find yourself in an environment where you need to connect to a remote MySQL server. If that's the case, add the --host parameter to specify the host name or address of the remote server.

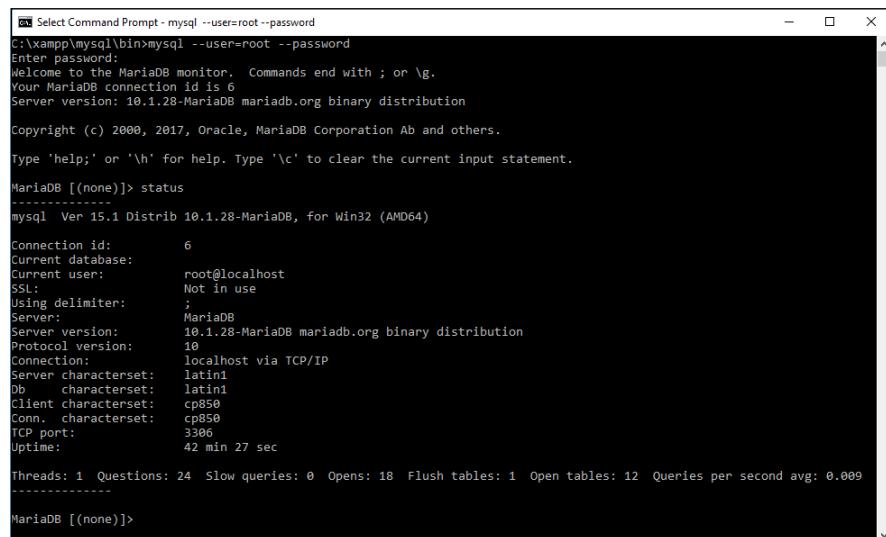
Besides standard SQL statements, the mysql command has quite a few special internal commands of its own. These commands help you set features within the CLI that regulate how it behaves. Each command has a full-name version and a shortcut-character version. If you want to use the shortcut, precede the shortcut character with a backslash. Table 2-2 lists the commands and their shortcuts that are currently available.

TABLE 2-2 The mysql Commands

Command	Shortcut	Description
charset	\C	Switch to another character set for the output
connect	\r	Reconnect to the server with a specified database
delimiter	\d	Set the delimiter used between SQL statements (the default is a semicolon)
edit	\e	Edit the command using the default editor
ego	\G	Send command to the MySQL server and display results
exit	\q	Exit the command-line interface
go	\g	Send the command to the MySQL server
help	\h	Display available commands

Command	Shortcut	Description
nopager	\n	Disable the pager and send output to the standard output
notee	\t	Don't redirect output to an output file
nowarning	\w	Don't display MySQL warning messages
pager	\P	Define a program to use to page output (such as more)
print	\p	Print the current command
prompt	\R	Change the command-line prompt
quit	\q	Quit the command-line interface
rehash	\#	Rebuild the command-line completion hash
source	\.	Execute the specified SQL script file
status	\s	Retrieve status information from the MySQL server
tee	\T	Redirect output to specified output file as well as the display
use	\u	Use another database as the default database
warnings	\W	Display MySQL warnings after each command

After you enter the command, the `mysql` program processes it and displays the results within the CLI environment. Figure 2-1 demonstrates the output from using the `status` command.



```

Select Command Prompt - mysql --user=root --password
C:\xampp\mysql\bin>mysql --user=root --password
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 6
Server version: 10.1.28-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> status
-----
mysql Ver 15.1 Distrib 10.1.28-MariaDB, for Win32 (AMD64)

Connection id:          6
Current database:       -
Current user:           root@localhost
SSL:                  Not in use
Using delimiter:        ;
Server:                MariaDB
Server version:         10.1.28-MariaDB mariadb.org binary distribution
Protocol version:       10
Connection:             localhost via TCP/IP
Server characterset:    latin1
Db     characterset:    latin1
Client characterset:   cp850
Conn. characterset:    cp850
TCP port:              3306
Uptime:                42 min 27 sec

Threads: 1 Questions: 24 Slow queries: 0 Opens: 18 Flush tables: 1 Open tables: 12 Queries per second avg: 0.009
-----
MariaDB [(none)]>

```

FIGURE 2-1:
The `status` command output.

To exit the `mysql` command prompt environment, just type `exit`.



TIP

As you can see from my examples, some all-in-one Apache/MySQL/PHP packages (such as XAMPP) use the MySQL sister application, MariaDB, instead of the original MySQL server package. The MariaDB package is a spinoff from the original MySQL package done by the original developers after Oracle took over development of the MySQL package. They created MariaDB to be a complete replacement for the MySQL server. To maintain complete compatibility between the two packages, the MariaDB developers use the same `mysql` commands, but insert the MariaDB signature in the command output, as you can see in Figure 2-1.

Using MySQL Workbench

Working from the command line can make you feel like a hard-core administrator, but relying on a graphical interface doesn't make you any less of a true administrator. Many administrators prefer to work with graphical tools, especially if they're already working in a graphical desktop environment.

The MySQL project includes a great graphical administration tool called Workbench. It's not often installed by default in most MySQL setups, but it's not hard to download and install yourself. This section walks through that process and shows you some of the features available in the graphical interface.

Installing the Workbench package

You can get the Workbench tool directly from the MySQL website (<https://dev.mysql.com/downloads/workbench>). The Download page is shown in Figure 2-2.

Scroll to the bottom of the page to see the section for downloading the program installation package. Just follow these steps to download and install Workbench:

1. Select the OS where you plan to run Workbench.

Because it's a binary program, you need to download a separate package for each OS environment where you plan to use it.

2. Click the Download button to start the download.

If you're a Windows user, be careful — there are two download options available. One option downloads the complete MySQL server along with Workbench. If you already have an all-in-one package installed, you don't need to download the MySQL server, only the Workbench client.

3. When the download completes, run the download package and follow the step-by-step instructions on installing Workbench on your workstation.



WARNING

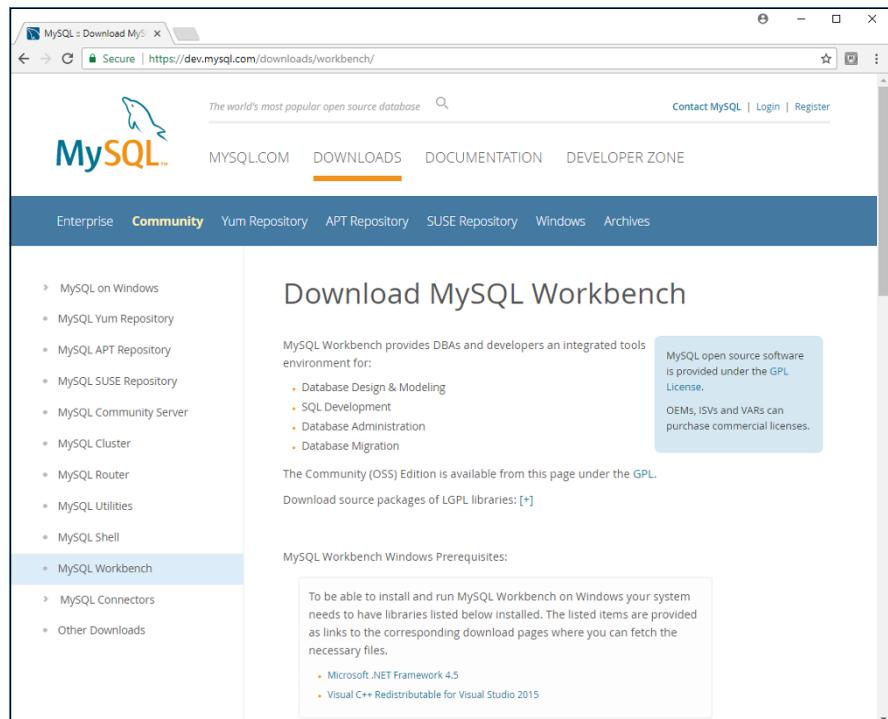


FIGURE 2-2:
The MySQL
Workbench
download
web page.



WARNING

If you're using Workbench on a Windows workstation, you'll need to have both the Microsoft .NET 4.5 and Visual C++ 2015 Redistributable libraries installed. You can find both of these library packages on the Microsoft developer website. The Workbench installation provides the URLs that you need to get them.

Exploring the Workbench options

After you install Workbench on your workstation and launch it, you'll be greeted by the main window, shown in Figure 2-3.

Before you can get started with your database administration, you'll need to tell Workbench how to find and log into your MySQL server. To do that, follow these steps:

1. Click the Plus sign next to the MySQL Connections heading to add a new connection.

This opens the Setup New Connection dialog box, shown in Figure 2-4.

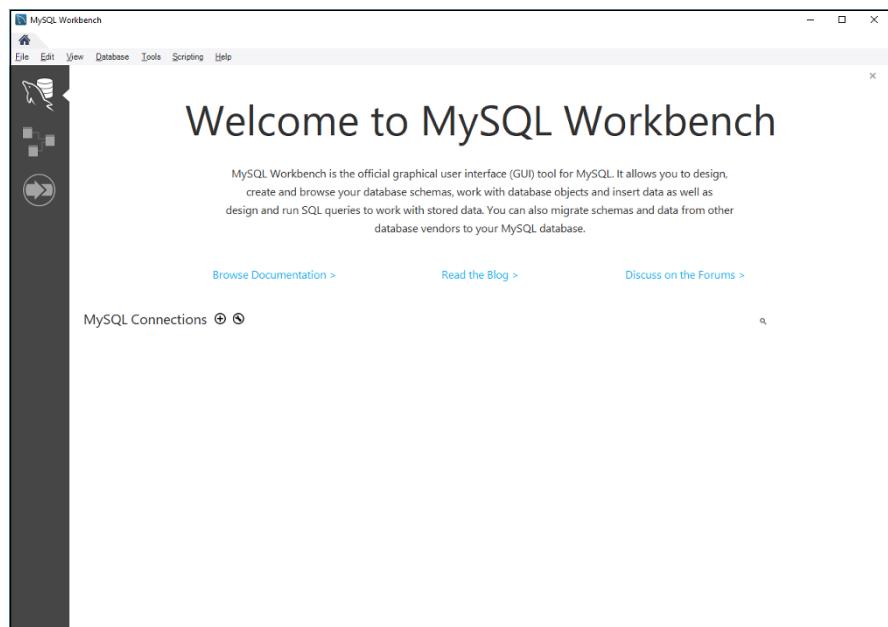


FIGURE 2-3:
The main
Workbench
window.

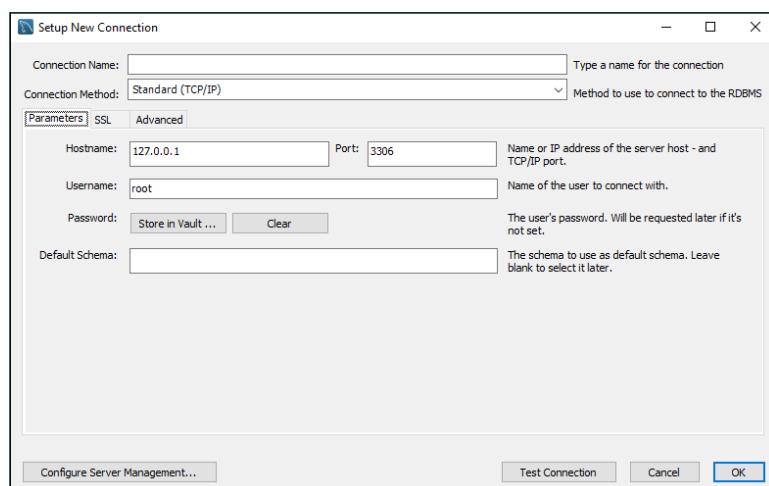


FIGURE 2-4:
The Workbench
Setup New
Connection
dialog box.

2. Enter a unique name for the connection in the Connection Name text box.
3. Enter the IP address or hostname for the MySQL server in the Hostname textbox.

If you're running MySQL server on your workstation (such as if you're using the XAMPP package), keep the default IP address of 127.0.0.1.

4. Enter the user account you use to log into the MySQL server in the **Username** textbox.

For full administration privileges, use the **root** user account.

5. Click **OK** to save the connection information.

After you create the connection, it appears as an option in the main Workbench window. Click that entry to start the connection to the database.



WARNING

The MySQL Workbench tool assumes you're working with the latest version of MySQL server. If your MySQL installation isn't the latest version (as is usually the case with all-in-one packages), Workbench will display a warning message when you connect, informing you that not all the features will be supported. Just click the Continue Anyway button to continue with the connection.

When Workbench establishes the connection, it produces the main administration window, as shown in Figure 2-5.

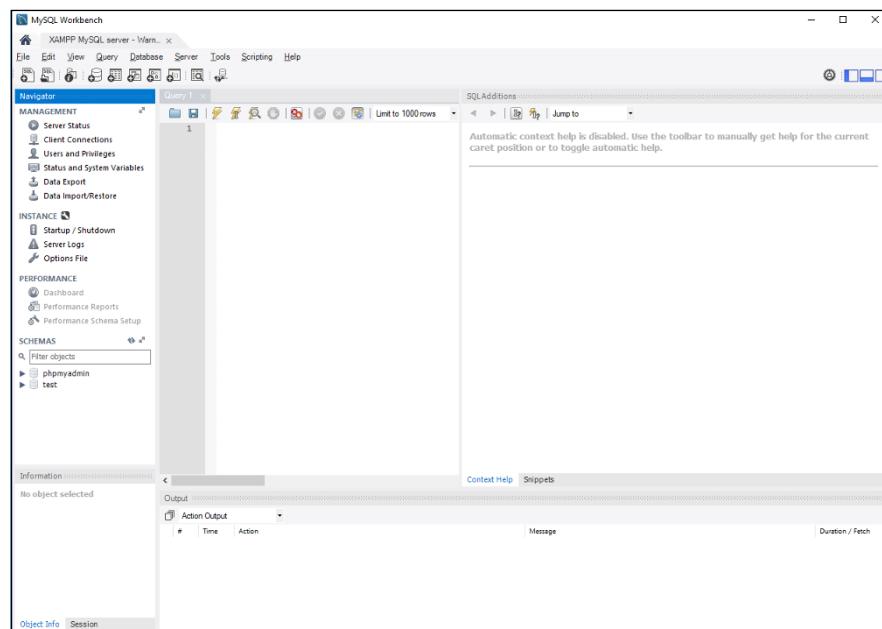


FIGURE 2-5:
The Workbench administration window.

The main administration window consists of five sections:

- » **Navigation:** Provides links to start and stop the server, monitor client connections, administer user accounts, export and import data, watch the

performance of the databases and tables, and add, modify, or remove databases and tables from the system schema.

- » **Query1:** Submit SQL queries directly to the server for testing.
- » **SQL Additions:** Provides online help with SQL statements, showing the context help for the SQL statements you enter into the Query1 panel.
- » **Information:** Displays information on the connection session or an individual object that you select from the Query1 panel.
- » **Action Output:** Displays the status of any actions you submit to the server, such as queries.

When you submit a query (or group of queries) from the Query1 panel, a new panel appears under the Query1 panel, showing the results from the transaction. If the transaction was a SELECT statement, the data records from the result set appear in a grid, as shown in Figure 2–6.

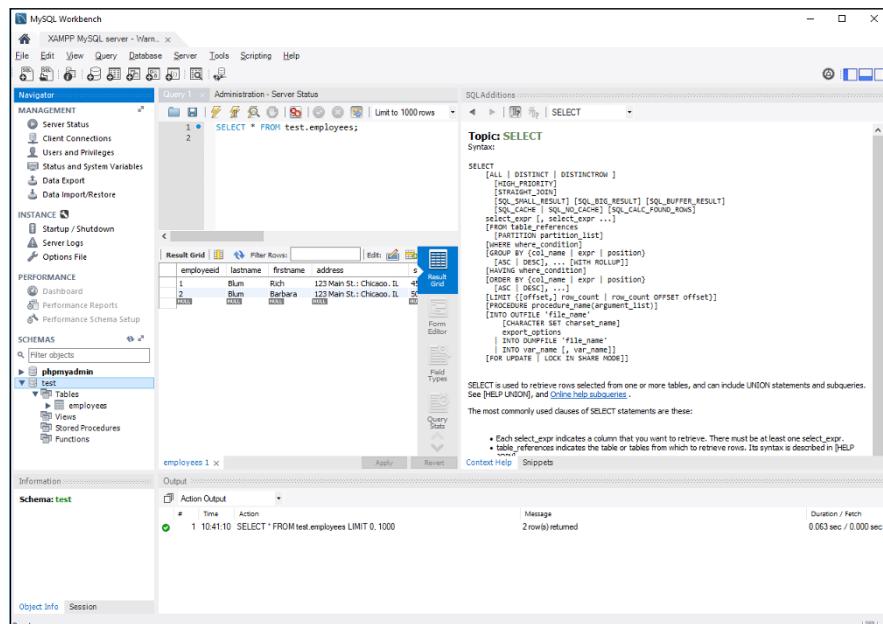


FIGURE 2-6:
Submitting
a query
using MySQL
Workbench.

From the Workbench interface, you can perform all the same functions that you can from the MySQL command-line interface but with a fancy graphical twist, making it a snap to manage your MySQL server!

Using the phpMyAdmin tool

Quite possibly the most popular graphical tool for working with MySQL servers is the web-based phpMyAdmin tool. As the name suggests, the phpMyAdmin tool is a PHP web application that interfaces with a MySQL server to provide a wealth of administration functions, all as a website that you can access from any browser!

The phpMyAdmin tool has become so popular that it's usually installed by default in many Apache/MySQL/PHP packages, such as XAMPP, MAMP, and LAMP, as well as supported by most commercial web hosting companies.

Since it's a website, to launch the phpMyAdmin application you need to open your browser and enter the URL that points to the package on your server. For most installations that's just `http://localhost/phpmyadmin`.

If you had to move the Apache web server to an alternative TCP port in your installation, you need to include that port in the URL: `http://localhost:8080/phpmyadmin`.

Depending on your particular environment, the phpMyAdmin package may be configured by default to automatically log into the MySQL server when you start it (such as in XAMPP and MAMP). If not, you'll be greeted by a login form to enter a MySQL user account and password. For full access privileges, log in using the root user account.

When you're logged into phpMyAdmin, you'll see the main window, as shown in Figure 2-7.

The main phpMyAdmin window displays the existing databases on the server on the left-hand side of the window. At the top is the navigation area, allowing you to select from 12 different options:

- » **Databases:** Create and manage databases.
- » **SQL:** Submit SQL statements directly to the server.
- » **Status:** Displays the status of connections, server processes, and database queries.
- » **User accounts:** Create and manage user accounts.
- » **Export:** Create a backup of one or more databases.
- » **Import:** Restore one or more databases.

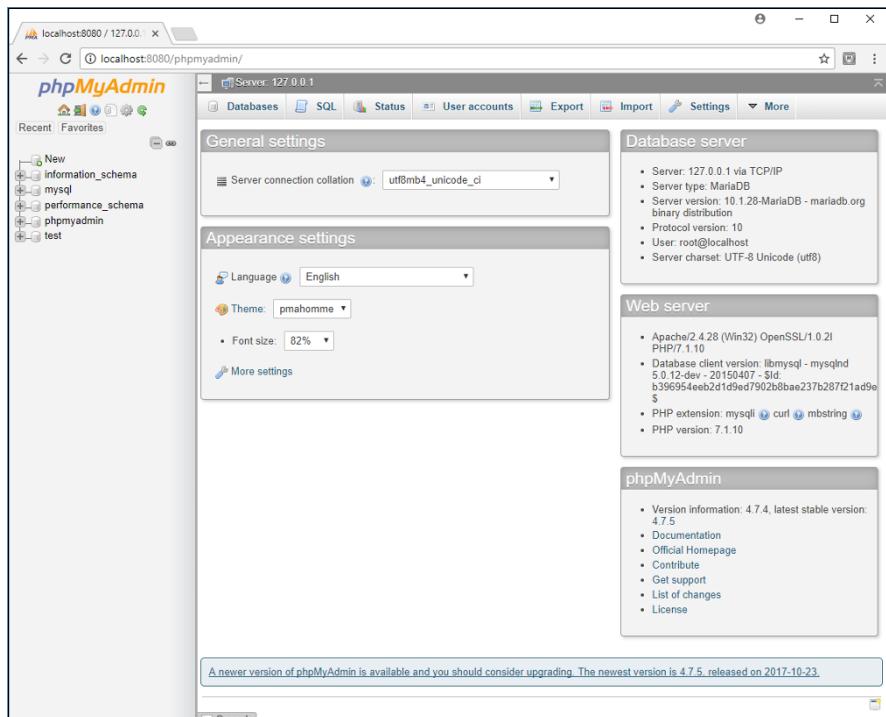


FIGURE 2-7:
The main
phpMyAdmin
window.

- » **Settings:** Manage settings for phpMyAdmin.
- » **Replication:** Control the master and slave replicas if created.
- » **Variables:** Manage the MySQL server configuration settings.
- » **Charsets:** Display the character sets available for the server.
- » **Engines:** Display the storage engines available for the server.
- » **Plugins:** Display plugins installed for phpMyAdmin.

As you can tell, phpMyAdmin also gives you full access to all the server features that you'd need to manage as the MySQL server administrator, all from a simple web interface!

Now that you've seen the three most popular administrator interfaces used for working with a MySQL server, the next sections take a look at doing some basic database administration work using each interface.

Managing User Accounts

One of the basic administration functions you need to perform in your MySQL server is creating and maintaining user accounts. By default, the MySQL server installs with a single user account, root, which has full access to everything on the database server. It's not a good idea to use this user account in web applications to access the databases. If your application should become compromised, the attacker would have full access to the database server, which would definitely cause a bad day for you.

This section walks through how to create and manage MySQL user accounts using each of the three popular MySQL tools.

Creating a user account

It's usually a good idea to create a separate user account for each web application that uses the MySQL server. That way you can restrict each user account to only access the single database used for the application, helping to prevent accidental data access and modification.

Just how you do that depends on the interface you've chosen to use to interact with the MySQL server.

From the MySQL command line

Managing user accounts from the MySQL CLI requires that you know a few SQL statements. To create a new user account, you use the `CREATE USER` statement. Here's the basic format for that:

```
CREATE USER username@location IDENTIFIED BY password;
```

As noted in Chapter 1 of this minibook, the MySQL server tracks user privileges based on a username and the location from where the user logs into the server. The `CREATE USER` statement lists both of these items in the definition. You can create separate `user@location` combinations if you desire to grant different privileges to applications depending on where they're running.

To create a new MySQL server user account, follow these steps:

1. **Open a command-line interface in your OS environment.**

For Windows, that's the Command Prompt tool. For macOS, that's the Terminal utility.

2. Navigate to the folder that contains the MySQL server programs.

If you're using the XAMPP package on Windows, the command is

```
cd \xampp\mysql\bin
```

For the macOS environment, the command is

```
cd /Applications/XAMPP/mysql/bin
```

3. Enter the mysql command to start the CLI, specifying the root user account and prompting for the password:

```
mysql --user root --password
```

For the macOS and Linux environments, you may have to precede the mysql command with the ./ symbol to tell the OS that the program is located in the current folder.

4. Enter the root user account password at the prompt.

For XAMPP, the password is empty, so just press Enter.

5. At the > prompt, type the CREATE USER command to create a new user account:

```
MariaDB [(none)]> CREATE USER user1@localhost IDENTIFIED BY  
'MyL0ngP@ssword';  
Query OK, 0 rows affected (0.08 sec)
```

```
MariaDB [(none)]>
```

6. Type exit to leave the MySQL CLI.

7. Type exit at the command-line prompt to exit the Command Prompt or Terminal session.

Now you have a new user account named user1 that can log into the MySQL server.

Using Workbench

Since the MySQL Workbench is a graphical tool, you don't need to know any SQL statements to create user accounts. You can create new accounts from the graphical interface by simply filling out a form. Follow these steps to do that:

- 1. Launch the MySQL Workbench tool from your workstation environment.**
- 2. Select the option to connect to your MySQL server.**

- 3. From the main Workbench window, click the Users and Privileges link in the Management section under the Navigation pane.**

The Users and Privileges window, shown in Figure 2-8, appears.

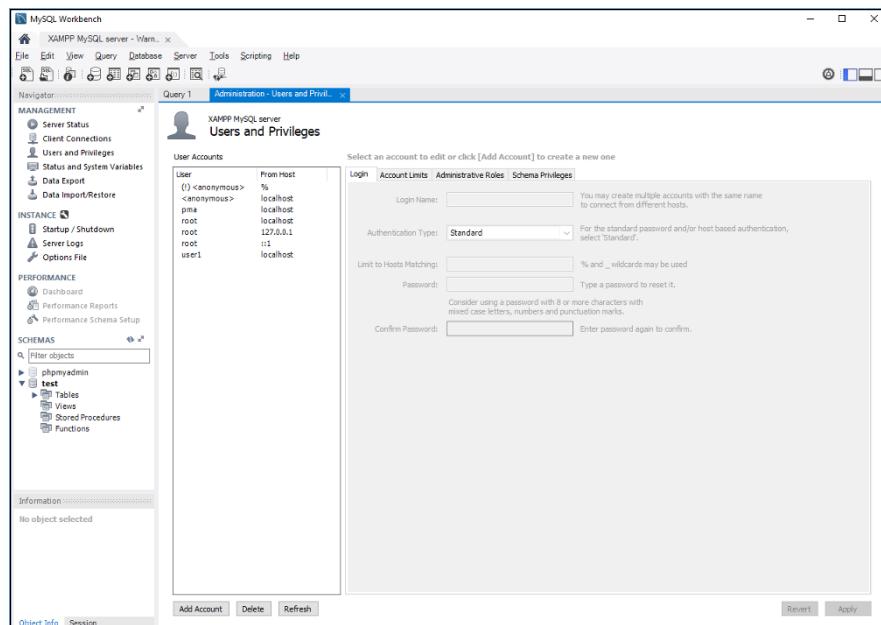


FIGURE 2-8:
The MySQL Workbench Users and Privileges window.

Notice that the window displays a complete list of the user accounts already available for the server. You should see the user1 account you created from the command line, as well as the entry for the root user account.

- 4. To add a new user account, click the Add Account button, located toward the bottom of the window.**
- 5. Fill in the form to specify a new user's login name of user2, the host location of localhost, and a password of MyLoNgP@ssword.**

Figure 2-9 shows this process.

- 6. Click Apply at the bottom of the window to create the user account.**
- 7. Close the MySQL Workbench tool when complete.**

That's all there is to creating a new user account from Workbench.

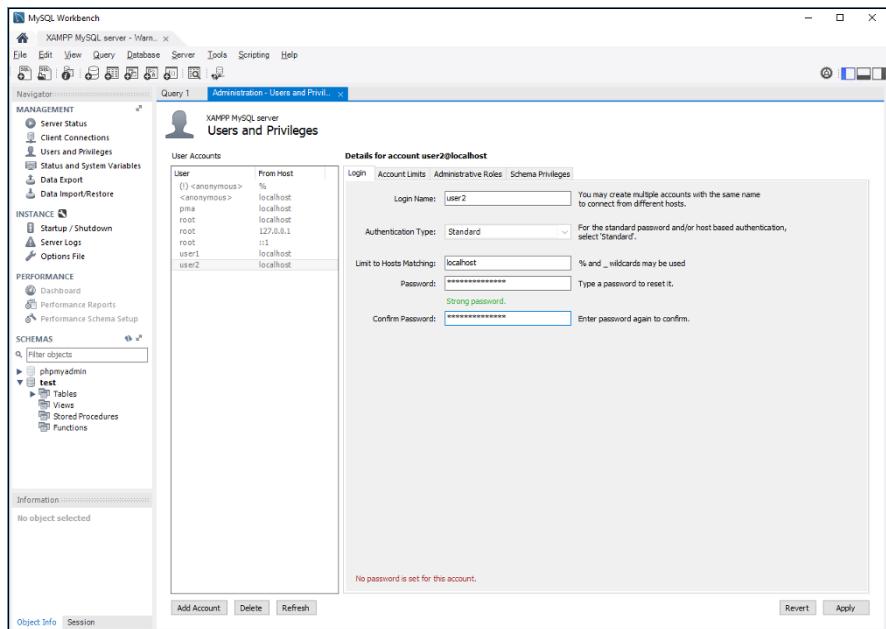


FIGURE 2-9:
Creating a new user account using Workbench.

Using phpMyAdmin

Because the phpMyAdmin tool is also a graphical interface, creating a new user account isn't all that much different from using Workbench, just from a web environment. Follow these steps to create a new user account using phpMyAdmin:

1. **Open your browser and enter the URL to get to the phpMyAdmin tool for your environment.**

If you're using XAMPP, enter the following URL:

```
http://localhost:8080/phpmyadmin
```

You may need to use a different TCP port depending on your web server.

2. **Click the User Accounts button at the top of the main phpMyAdmin web page.**

This produces the User Accounts Overview page, shown in Figure 2-10.

3. **To create a new user account, click the Add User Account link in the New section.**

The Add User Account page appears.

4. **For the username, type user3.**

The screenshot shows the 'User accounts overview' page in phpMyAdmin. The left sidebar lists databases: New, information_schema, mysql, performance_schema, phpmyadmin, and test. The main area displays a table of user accounts:

User name	Host name	Password	Global privileges	User group	Grant	Action
Any	%	No	USAGE		No	Edit privileges Export
Any	localhost	No	USAGE		No	Edit privileges Export
pma	localhost	No	USAGE		No	Edit privileges Export
root	127.0.0.1	No	ALL PRIVILEGES		Yes	Edit privileges Export
root	::1	No	ALL PRIVILEGES		Yes	Edit privileges Export
root	localhost	No	ALL PRIVILEGES		Yes	Edit privileges Export
user1	localhost	Yes	USAGE		No	Edit privileges Export
user2	localhost	Yes	USAGE		No	Edit privileges Export

Below the table are buttons for 'New' (Add user account) and 'Remove selected user accounts'. A note at the bottom says: '(Revoke all active privileges from the users and delete them afterwards.)' and has checkboxes for 'Drop the databases that have the same names as the users.' and a 'Go' button.

FIGURE 2-10:
The User Accounts Overview window in phpMyAdmin.

5. From the Host Name drop-down list, choose Local.
 6. For the password form fields, type MyL0ngP@ssword.
- Figure 2-11 shows what these entries should look like.
7. Scroll to the bottom of the web page and click the Go button.
 8. Click the User Accounts button at the top of the web page to view the user account list to verify the new user account.
 9. Click the Exit icon on the left side of the web page to close the session, and then close your browser window.

As you can see, creating user accounts in the phpMyAdmin environment isn't all that different from the Workbench environment, because they both use similar graphical interfaces to build and submit the CREATE USER SQL statement for you!

Managing user privileges

After you create a user account for your web application, you'll need to grant it privileges to use the database that supports the application. As part of a security feature, MySQL only grants new user accounts the ability to log into the server — they don't have access to any data on the server by default.

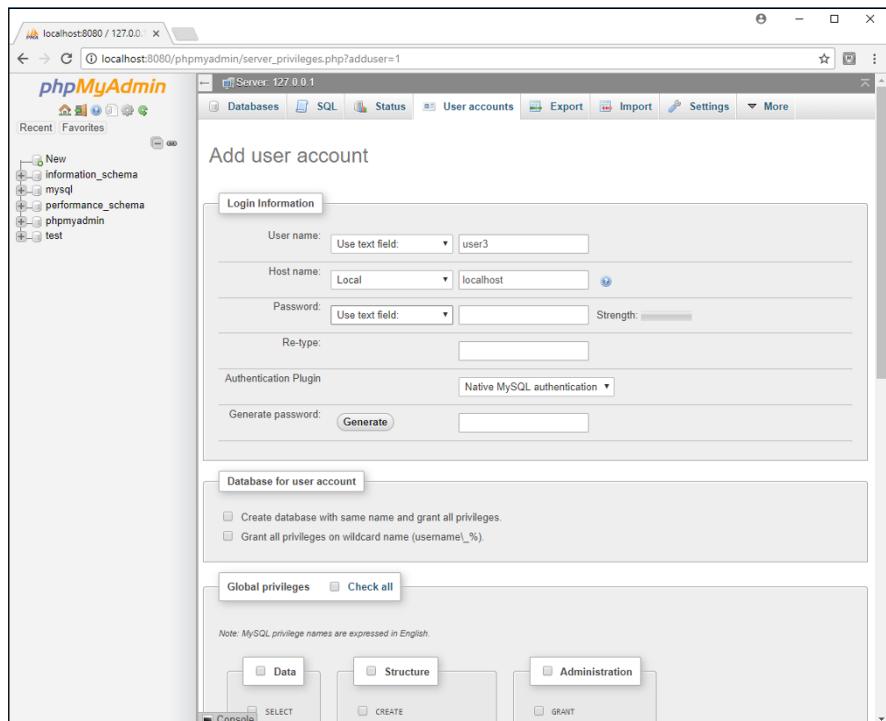


FIGURE 2-11:
Entering a new user account in phpMyAdmin.

To solve that, you need to use the GRANT SQL statement, using either the MySQL CLI or one of the fancy graphical tools you've just learned how to use. The basic format of the GRANT statement is:

```
GRANT privileges ON objects TO user;
```

The *privileges* list controls just what access the user account has on the database objects defined in the *objects* list. MySQL allows you to grant as many or as few privileges to a user account as you need, providing very fine control over database access. Table 2-3 lists the privileges that you can use.

TABLE 2-3 MySQL Privileges

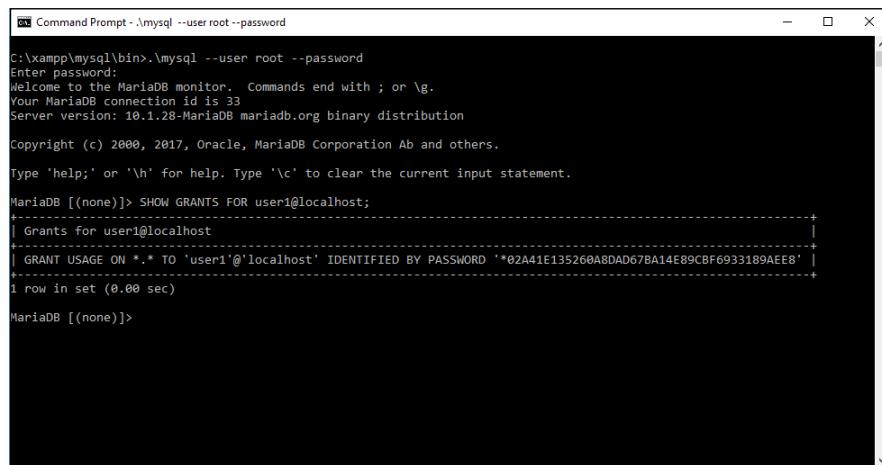
Privilege	Description
ALL	All privileges
ALTER	The ability to change a database or table definition
ALTER ROUTINE	The ability to change or remove a stored routine
CREATE	The ability to create a new database or table within a database

Privilege	Description
CREATE ROUTINE	The ability to create a new stored routine
CREATE TABLESPACE	The ability to create a new database storage area
CREATE TEMPORARY TABLES	The ability to create temporary tables in a database
CREATE USER	The ability to create, rename, or remove user accounts on the server
CREATE VIEW	The ability to create or change a database view
DELETE	The ability to remove data from tables
DROP	The ability to remove databases or tables
EVENT	The ability to use events in the event scheduler
EXECUTE	The ability to run stored routines
FILE	The ability to cause the server to read or write to files
GRANT OPTION	The ability to add or remove privileges to other users
INDEX	The ability to create or remove table indexes
INSERT	The ability to add new data to tables
LOCK TABLES	The ability to lock tables for data access
PROCESS	The ability to see all the database processes
PROXY	The ability to use proxying
REFERENCES	The ability to create and remove foreign key relationships
RELOAD	The ability to force database writes to files
REPLICATION CLIENT	The ability to list replication servers and clients
REPLICATION SLAVE	The ability to enable replication slaves to contact the server
SELECT	The ability to query databases and tables
SHOW DATABASES	The ability to list all the databases on the server
SHOW VIEW	The ability to list all the views on the server
SHUTDOWN	The ability to stop the MySQL server
SUPER	The ability to have administrative control of the MySQL server
TRIGGER	The ability to create and remove triggers
UPDATE	The ability to modify existing data in tables
USAGE	The ability to log into the MySQL server, but no data access

There are two levels of privileges in MySQL:

- » **Global privileges:** Apply to all database objects
- » **Local privileges:** Apply to a specific database or table

To see what global privileges an existing user account has from the CLI, use the SHOW GRANTS statement, as shown in Figure 2-12.



```
Command Prompt - .\mysql --user root --password
C:\xampp\mysql\bin>.\mysql --user root --password
Enter password:
Welcome to the MariaDB monitor. Commands end with ; or \g.
Your MariaDB connection id is 33
Server version: 10.1.28-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> SHOW GRANTS FOR user1@localhost;
+-----+
| Grants for user1@localhost |
+-----+
| GRANT USAGE ON *.* TO 'user1'@'localhost' IDENTIFIED BY PASSWORD '*02A41E135260A8DAD67BA14E89CBF6933189AE8' |
+-----+
1 row in set (0.00 sec)

MariaDB [(none)]>
```

FIGURE 2-12:
Displaying global
user privileges
from the CLI.

The output shows that the user1 user account only has USAGE global privileges, so it can log into the database server, but not access any data.

To grant privileges to a specific database, you must list the privileges in the GRANT statement, along with the specific database:

```
GRANT SELECT ON phpmyadmin.* TO user1@localhost;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]>
```

The wildcard character used in the database object list indicates that the privileges apply to all the tables contained in the phpmyadmin database. If needed, you could apply specific privileges to individual tables within your application.

Now you can log in using the user1 user account and access the phpmyadmin database:

```
C:\xampp\mysql\bin>mysql --user=user1 --password
Enter password: *****
Welcome to the MariaDB monitor.  Commands end with ; or \g.
Your MariaDB connection id is 35
Server version: 10.1.28-MariaDB mariadb.org binary distribution

Copyright (c) 2000, 2017, Oracle, MariaDB Corporation Ab and others.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

MariaDB [(none)]> use phpmyadmin;
Database changed
MariaDB [phpmyadmin]>
```

Doing the same thing from one of the graphical interfaces is similar, but just using a graphical form. Follow these steps to grant database privileges to a user account using Workbench:

- 1. Start the Workbench application and then click the connection icon to connect to your MySQL server.**
- 2. Click the Users and Privileges link in the Navigation section on the right side of the window.**

This displays the Users and Privileges window, as shown in Figure 2-13.

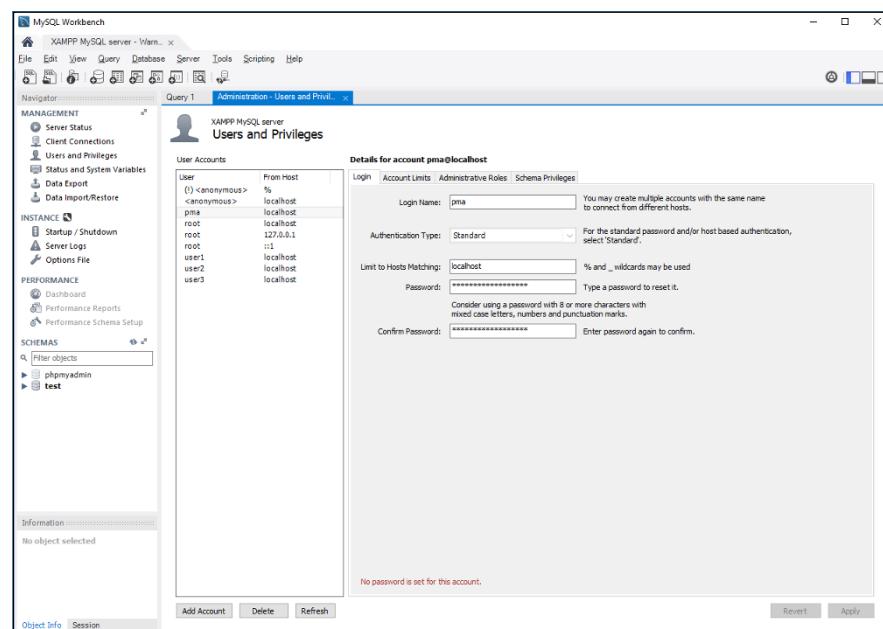


FIGURE 2-13:
The Workbench
Users and
Privileges
window.

- 3. Click the user2 user account in the list.**
- 4. Click the Schema Privileges tab at the top of the display section.**

The current global and database privileges granted to the user account appear.

- 5. Click the Add Entry button.**

The New Schema Privilege Definition form, shown in Figure 2-14, appears.

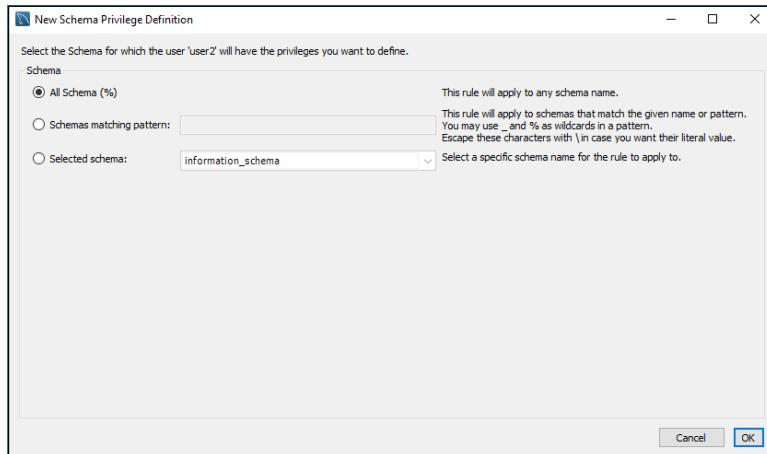


FIGURE 2-14:
The Workbench
form to add
schema
privileges.

- 6. Click the Selected schema radio button, select the phpmyadmin database from the drop-down list, and click OK.**

The Details for Account user2@localhost window, shown in Figure 2-15, appears.

- 7. Check the SELECT check box in the Object Rights section to allow the user2 account access to view and query tables in the phpmyadmin database.**
- 8. Click the Apply button at the bottom of the window to apply the new privileges to the user account.**

As you can probably guess, using the phpMyAdmin tool to grant privileges is very similar to how you did it using the Workbench tool. Follow these steps:

- 1. Click the User Accounts tab at the top of the main phpMyAdmin web page.**

The list of the user accounts currently configured in the MySQL server appears. You should see the user1 and user2 user accounts that you've already created, as shown in Figure 2-16.

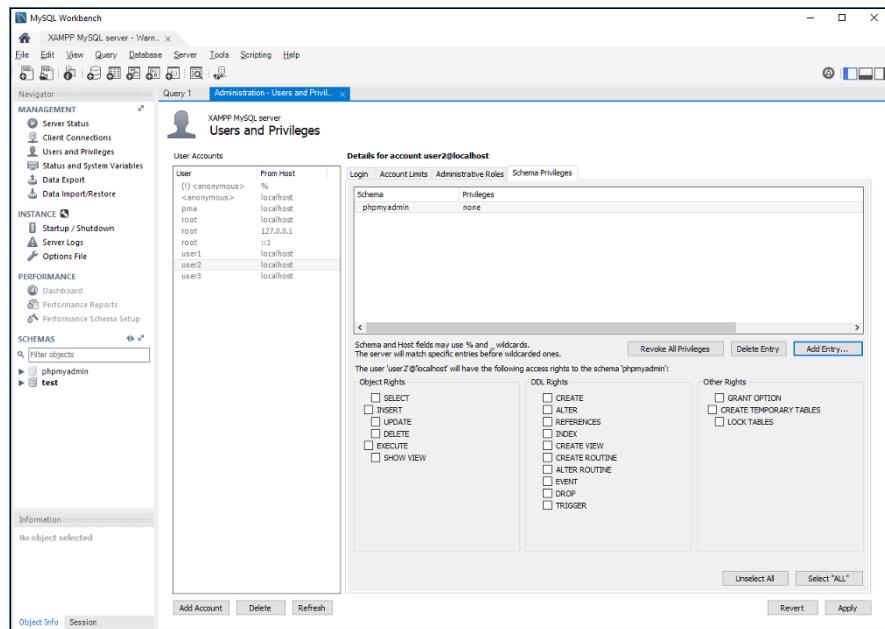


FIGURE 2-15:
Adding schema
privileges using
Workbench.

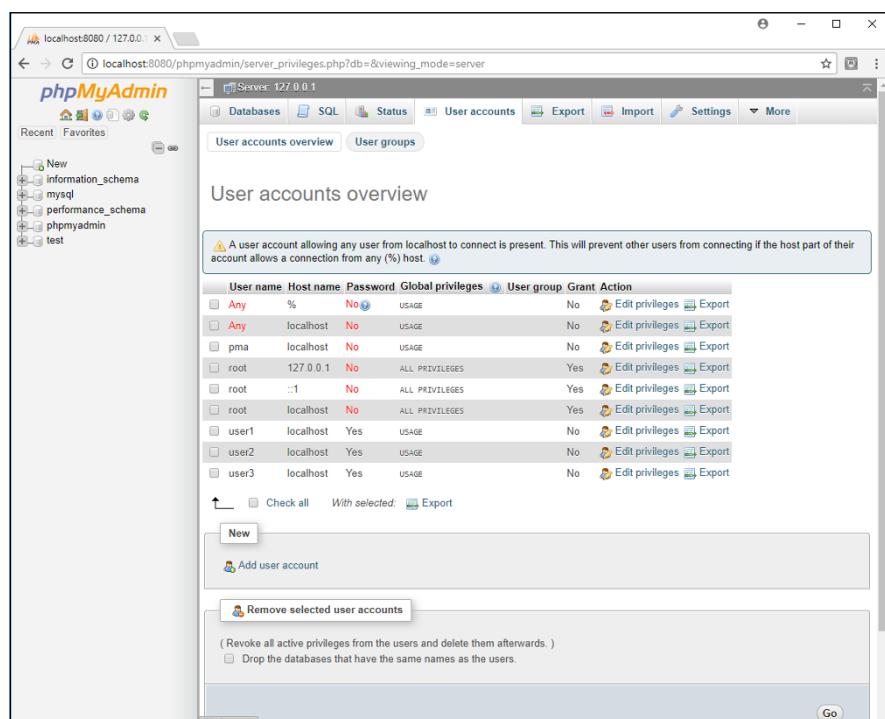


FIGURE 2-16:
Using the
phpMyAdmin
tool to display
user accounts.

- 2.** Click the **Edit Privileges** link for the user3 user account.
- 3.** Click the **Database** button at the top of the **Edit Privileges** page.
- 4.** Select the **phpmyadmin** database from the list and then click the **Go** button.
- 5.** Check the **SELECT** check box and then click **Go**.

Figure 2–17 shows the web page for displaying the privileges set for the user3 user account on the phpmyadmin database.

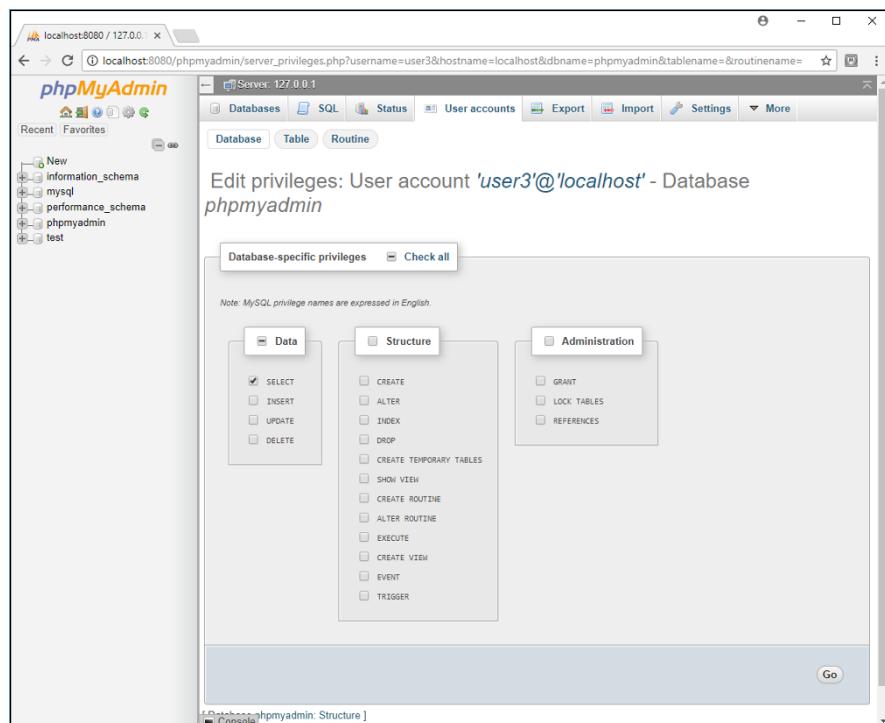


FIGURE 2-17:
Setting
database
privileges
using the
phpMyAdmin
tool.

All three methods produce the exact same results, so feel free to use whichever tool you prefer!

IN THIS CHAPTER

- » Understanding how to design a database
- » Creating a database in MySQL
- » Building tables using different tools

Chapter 3

Designing and Building a Database

In the preceding chapter, you learned your way around the MySQL server interface tools. The next step in the process of building a dynamic web application is to create a database and tables for the data required for your application.

In this chapter, I show you how to determine just what data is required for an application and how to divide it into tables to manage the data. Then I show you how to create databases using the popular MySQL server interface tools. Finally, I explain how to create the tables by using each of the tools, so that you can manage the data in your applications.

Managing Your Data

When you start out a new dynamic web application, your first decision, before you even start any coding, is how to handle the application data. Often you're faced with a myriad of data elements you need to track, such as employee, customer, and product information for a store. The trick to successfully managing all that information is in how to sort it all out.

The process of structuring application data into tables is called *database normalization*. The key to database normalization is to build your database so that your application can quickly and easily add, modify, delete, and search for data contained in the tables, and do it with a minimum amount of server overhead. For large applications, that can be easier said than done!

Fortunately, many very smart people have worked out some standard rules you can follow for organizing the data in your applications. These rules are called *normal forms*. Each normal form defines a set of standards to follow to organize and protect the data in your application. Each normal form builds on the other normal forms to provide a tiered approach to organizing data. Although there are many different normal forms, for most applications you just need to follow three: the first, second, and third normal forms. These are described in the following sections.

The first normal form

In the *first normal form*, the idea is to organize the application data to find related data elements and group them into tables, identify the unique data elements with a key to make them easier to find, and eliminate any redundant data stored in tables.

The first part of the rule specifies to group related data into separate tables. In the store example from Chapter 1 of this minibook, you create three tables for a store application by grouping employee information into an Employees table, customer information into a Customers table, and product information into a Products table. That covers the first part of the first normal form!

The second part of the rule specifies that you should provide a way to uniquely identify each individual data record in each of the tables. You do that by defining a *primary key* data field for each table. Sometimes that can be done using existing data elements; other times it requires that you add new data elements.

In the Employees table, you can't necessarily use one of the existing data values to point to a specific employee — there could be multiple John Smiths working at the company, or there could be multiple employees with the same address. It's even possible to have multiple employees with the same birth date.

The solution is to create a separate data field that assigns a unique value to each employee. The application assigns a unique `employeeid` to each employee that it can use to find individual employees. This data field is designated as the *primary key* for the Employees table. The primary key guarantees that you'll retrieve the information for a single employee based on a unique `employeeid` value. You then do the same thing for the Customers and Products tables.

The last part of the rule specifies that you should eliminate any redundant data contained in the table. For example, if an employee has multiple phone numbers, it may not be a good idea to have multiple phone data fields in the Employees table. How many should you create? What if you create home and cellphone number data fields, but then need to add an employee's summer house number? You can't just continue adding new data fields to the table all the time.

The solution is to create a separate table with the phone number information. The phone number table can have multiple data records with the same employeeid data values, but each with a different phone number. To find all the phone numbers for an employee, just query the Phone Numbers table with that employeeid. Now you can accommodate as many or as few phone numbers for each employee without wasting data field space in the Employees table.

The second normal form

The *second normal form* specifies that you should create separate tables for data fields that could apply to multiple tables. In the store example, this would apply to how you track customer orders.

In this application, customers place orders, so the orderid value could be tracked by customerid. However, orders contain one or more products, so the order could be tracked by productid. This presents a problem.

Adding the order information directly in the Customers table would be bad. Hopefully, your customers will have multiple orders, so each order data record would need to duplicate the customer's information. That would violate the data redundancy rule. Plus, it wouldn't work putting order information in the Products table, because multiple products could also be in the same order.

The solution is to create a separate Orders table, and relate that table to both the Customers and Products tables. Each order data record would use the customerid and productid primary key values from the Customers and Products tables so that it could relate the order item back to a customer and the products it contains.

The third normal form

The third normal form defines how to work with data fields that don't necessarily depend on the primary key in a table but need to be searched. This level of normalization depends heavily on just how your application uses the data that it stores.

An example of this would be the startdate data field in the Employees table. If your application needs to perform a lot of queries to find employees who've

worked at the company for a specific number of years, it could help the application performance to create a separate table with the startdate values, separate from the Employees table. This helps speed up the query process by reading a smaller table with the one value instead of the entire Employees table. This is often referred to as an *index table*.

The index table contains data that is commonly queried in the application but is separate from the primary key of the table. If your application needs to query the startdate of employees as a primary function, it will help increase the performance of those queries by creating a separate index table of the startdate values contained in the Employees table.

However, index tables come with drawbacks. As you insert each new employee data record, the database system must now make two entries: one in the Employees table and another in the startdate index table. That will slow down the performance of adding new employee data records!

As you can see, this produces a trade-off. If your application queries the startdate of employees a lot, it would help to implement the third normal form rule and create the separate index table. If not, it would be best to ignore the third normal form rule and not create the separate table. It all comes down to knowing how your application and your application users work!

Creating Databases

After you determine the structure required to support your application data, you can start creating it in the MySQL server. The first step in that process is to create a database for the application. This section walks through the different ways to create a new database using the different MySQL tools covered in the previous chapter.

Using the MySQL command line

To create a new database from the MySQL command line interface (CLI) you use the CREATE DATABASE SQL statement. Depending on your needs and environment, this command can be either very simple or very complex. If you just want to create a database that uses the server default character set settings, just specify the name of the database in the command:

```
MariaDB [(none)]> CREATE DATABASE dbtest1;
Query OK, 1 row affected (0.00 sec)
```

```
MariaDB [(none)]>
```

You now have a new database!



WARNING

On Mac, Linux, and Unix systems, the database names are case sensitive; on Windows systems, they're case-insensitive. This can cause all sorts of problems if you migrate your database from one environment to another, so be careful with using mixed-case database names! Your best bet is to stick with the same case for all characters in the database name.

To make sure the database was actually created, use the `SHOW DATABASES` statement at the CLI to display the databases contained on the server:

```
MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database      |
+-----+
| dbtest1       |
| information_schema |
| mysql          |
| performance_schema |
| phpmyadmin     |
| test           |
+-----+
6 rows in set (0.00 sec)
```

```
MariaDB [(none)]>
```

The database you created should appear in the list of databases. If you'd like to see a little more detail about the new database, use the `SHOW CREATE DATABASE` statement:

```
MariaDB [(none)]> SHOW CREATE DATABASE dbtest1;
+-----+
| Database | Create Database |
+-----+
| dbtest1  | CREATE DATABASE `dbtest1` /*!40100 DEFAULT CHARACTER SET latin1*/|
+-----+
1 row in set (0.00 sec)
```

```
MariaDB [(none)]>
```

The output from the `SHOW CREATE DATABASE` statement indicates that the database is using the `latin1` character set. The character set defined for your database may be different, depending on the default settings in your MySQL server. If you need to create a database using a specific character set, you can specify that in the `CREATE DATABASE` statement:

```
MariaDB [(none)]> CREATE DATABASE dbtest1
    -> CHARACTER SET latin1
    -> COLLATE latin1_general_cs;
Query OK, 1 row affected (0.00 sec)
MariaDB [(none)]>
```

This statement creates the `dbtest1` database using the `latin1` character set, and the `latin1_general_cs` collation.

CHARACTER SETS AND COLLATIONS

MySQL supports different character sets and collations for storing data. The *character set* defines the binary code MySQL uses to store character text, while the *collation* defines the algorithms used to compare text values. MySQL uses a cascading method of assigning character sets and collations. If you define a default character set and collation for the server, they'll be used when you create new data objects that don't specify a character set or collation. If you define a default character set and/or collation for a database, those will override the server defaults. If you then create a new table, it will use the character set and collation defined for the database by default. If you define a character set and/or collation for a table, those will override any database or server defaults.

The `latin1` character set supports Western European languages. If your application needs to support text from other languages, use the `utf8` character set. Likewise, the `latin1_general_ci` collation compares text based on the `latin1` character set. The `ci` part of the collation name indicates that comparisons are made in case-insensitive mode, so uppercase and lowercase letters will match. If your application needs to support case-sensitive comparisons, you'll want to specify a collation that ends with `cs`, such as the `latin1_general_cs` collation.

You can see what character sets your particular MySQL server supports by using the `SHOW CHARACTER SET` statement. This lists the character sets and the default collation that MySQL will use with that character set. To see the collations that are available, use the `SHOW COLLATION` statement.

If you need to remove a database from the MySQL server, you use the `DROP` statement:

```
MariaDB [(none)]> DROP DATABASE dbtest1;
Query OK, 0 rows affected (0.00 sec)

MariaDB [(none)]> SHOW DATABASES;
+-----+
| Database      |
+-----+
| information_schema |
| mysql          |
| performance_schema |
| phpmyadmin     |
| test           |
+-----+
5 rows in set (0.00 sec)

MariaDB [(none)]>
```



WARNING

Be careful using the `DROP` and `DELETE` SQL statements! The `DROP` statement removes the entire object, while the `DELETE` statement removes the data but keeps the object.

Using MySQL Workbench

The MySQL Workbench tool provides a nice graphical environment for you to easily create databases. The Schemas section of the Navigator pane displays the current databases created on the server. (*Remember:* Workbench refers to databases as schemas.)

To create a new database using Workbench, follow these steps:

1. Right-click in the Schemas section and select **Create schema** from the pop-up menu.

A New schema form opens in the left-hand section of the window, as shown in Figure 3-1.

2. Enter the name of the database in the Name text box.
3. Select a character set and appropriate collation from the Collation drop-down menu.

You can leave the Server Default value to use the default character set and collation settings for the server.

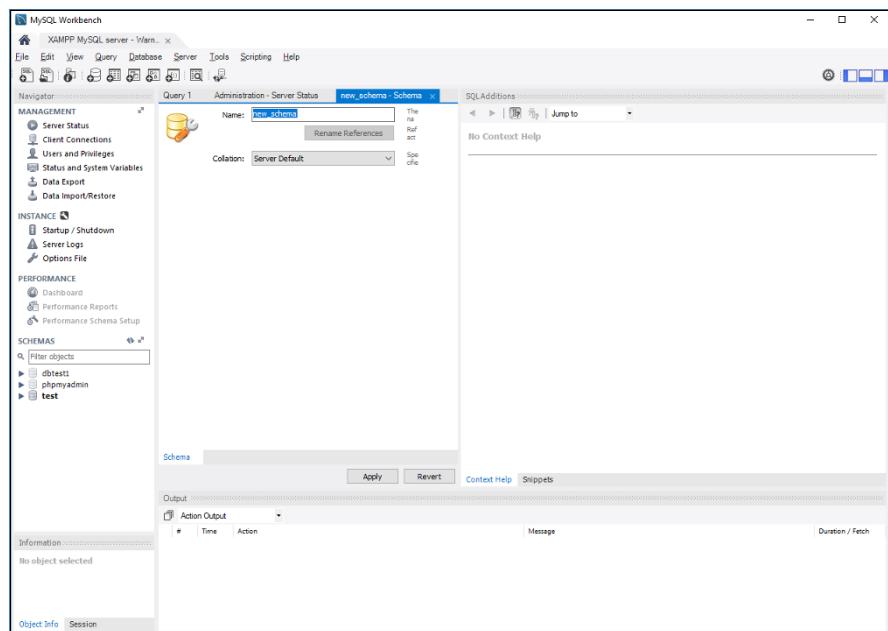


FIGURE 3-1:
Creating a new database using Workbench.

4. Click Apply.

The Workbench Create Database Wizard appears, which walks you through the database creation process. First, the CREATE DATABASE statement generated by the information you entered into the form appears, as shown in Figure 3-2.

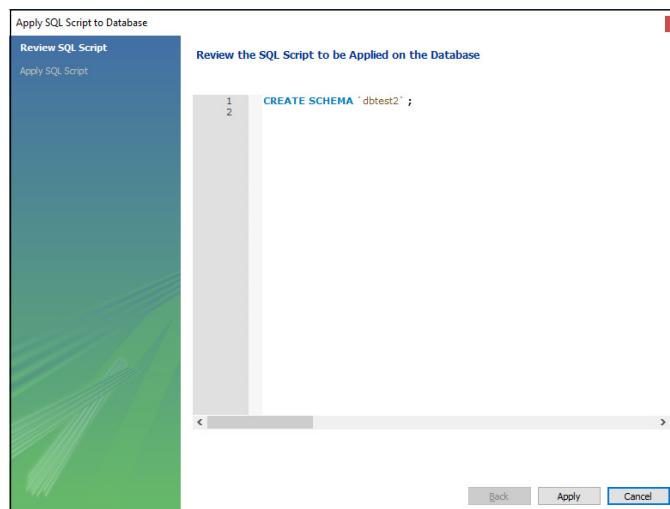


FIGURE 3-2:
The Workbench Create Database Wizard.

5. Click **Apply** to submit the generated SQL statement to the MySQL server to create the database.
6. When the MySQL server runs the statement, the wizard displays the results, as shown in Figure 3-3.

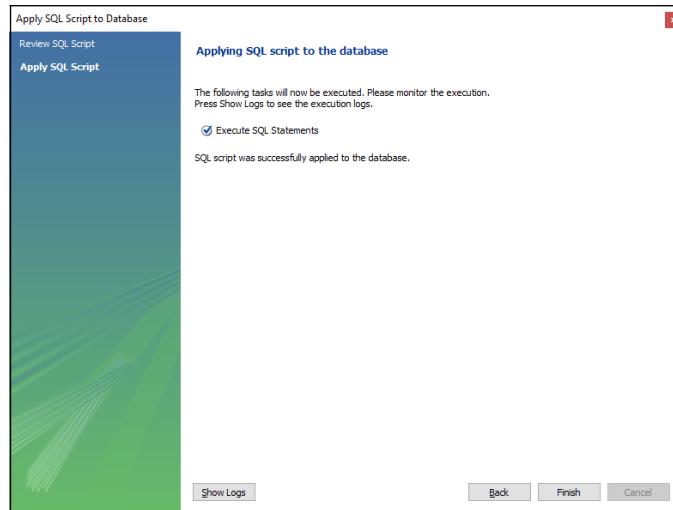


FIGURE 3-3:
The results of
the Workbench
Create Database
Wizard.

If the SQL submission was successful, the new database will appear under the **Schemas** section in the left-hand section of the window.

If you need to remove a database using Workbench, simply right-click the database entry in the **Schemas** list and then select the **Drop schema** menu entry. Simple!

Using phpMyAdmin

As you might guess, creating a database in the phpMyAdmin web-based graphical tool is similar to using Workbench. Here are the steps to do that:

1. After you open the phpMyAdmin tool in your browser, click the **Databases** button in the top Navigation bar.

Figure 3-4 shows the form that appears.

The Databases page displays the existing databases on the MySQL server, along with a form to create a new database.

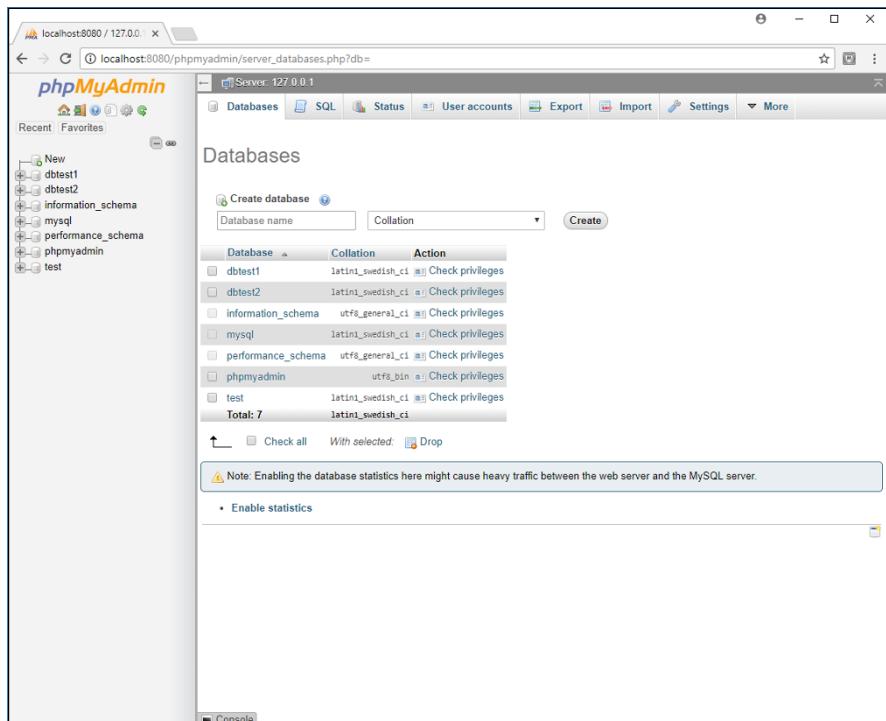


FIGURE 3-4:
The phpMyAdmin
Databases page.

- 2. Enter the name of the new database in the Database name text box.**
- 3. Select the character set and collation from the Collation drop-down menu.**

If you want to use the server default values, just leave the drop-down box empty.

- 4. Click the Create button to submit the SQL to create the database.**

If the database creation was successful, phpMyAdmin automatically takes you to the database interface web page, prompting you to create a new table in the database, as shown in Figure 3-5.

Removing a database using phpMyAdmin is a little more complex than in Workbench. Here are the steps to remove an existing database:

- 1. Click the database you want to remove in the left-hand list of databases.**
- 2. Click the Operations tab at the top of the database web page.**

The database operations web page, shown in Figure 3-6, appears. On the operations web page, you can rename the database, copy the database, create tables in the database, and of course, remove the database.

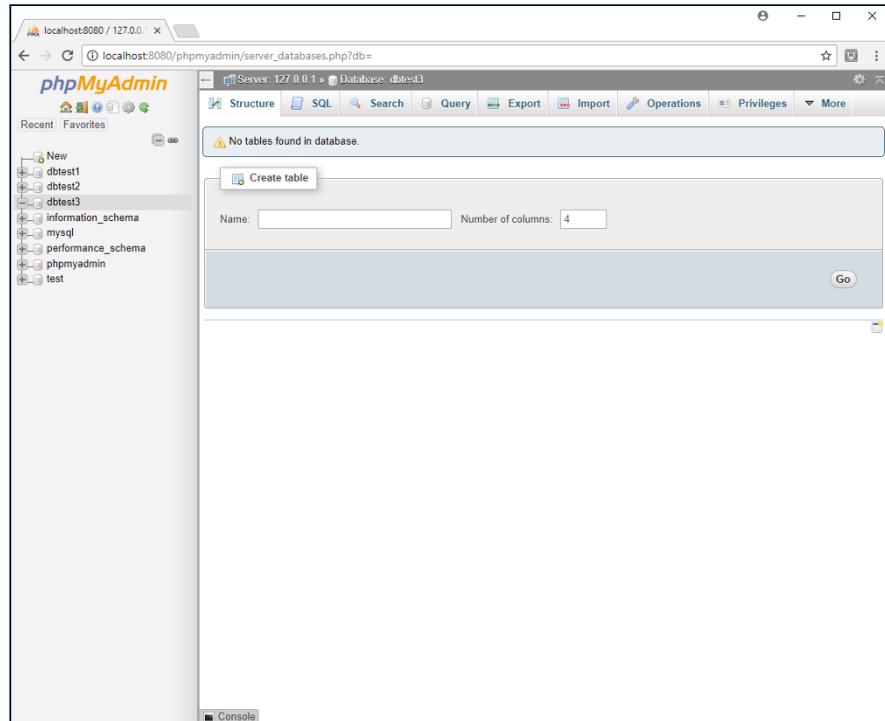


FIGURE 3-5:
The phpMyAdmin
database
web page.

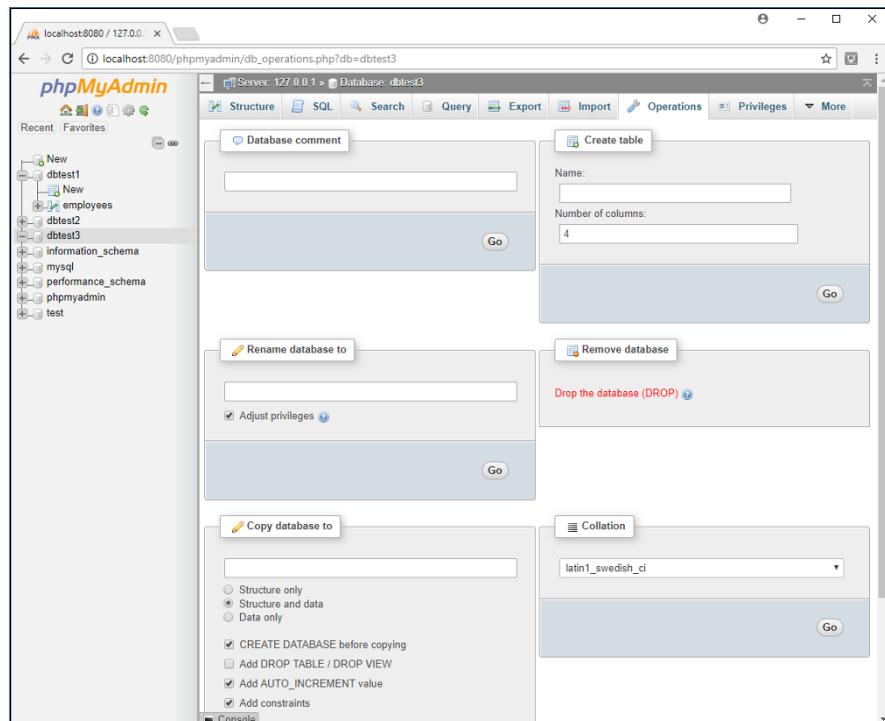


FIGURE 3-6:
The phpMyAdmin
database
operations
web page.

3. Click the Drop the Database (DROP) link that appears on the right-hand side of the page (in red font) to remove the database.

After you've created your application database, you can move onto the next step: creating the tables to hold the application data.

Building Tables

In a relational database model, tables are what hold all the actual application data. As mentioned at the start of this chapter, it's important that you take time to plan your table layout and structure before you try creating any tables.

Each data table definition must specify the individual data elements contained in the table, along with all the properties for those data elements. That includes

- » The data field name
- » The data field data type
- » Any indexes required for the data field (such as the primary key)
- » Whether any foreign keys need to be defined for the table
- » Any data constraints required for the data field

All this information can make creating a table from the command line require a lot of typing! Fortunately, the graphical tools available make the process a little easier, but before you get to the easy stuff, let's take a look at how to create tables using the command line so you can learn the SQL format for the statements.

Working with tables using the command-line interface

In the MySQL CLI, you use the `CREATE TABLE` statement to build a new table. Here's the basic format of the `CREATE TABLE` statement:

```
CREATE TABLE name (field1 datatype constraints, field2 datatype constraints...);
```

You must define each individual data field, specifying the data field name, data type, and any data constraints applied to the data field. For tables with lots of data fields, this can become quite a long statement!

Instead of trying to include all the information required to create a table in one CREATE statement, database administrators often utilize the ALTER TABLE statement. This statement alters the definition of an existing table, allowing you to add, modify, or remove data fields, data field types, and of course, data field constraints. So, you can build a base definition of the table using the CREATE TABLE statement and then add additional elements using ALTER TABLE statements.

The following sections go through the process of creating a table and adding additional elements to the base table.

Defining the base table

For the basic table definition, just define the table name and the individual data fields and their required data types. For tables with lots of data fields, even just this primary information can make for a long CREATE TABLE statement! To help keep your sanity, you can use the command completion feature of the MySQL CLI. Just press Enter in the middle of the statement, and you'll get a prompt to complete the statement. By default, the MySQL CLI won't process the statement until it sees a semicolon.

Follow these steps to create a simple base table:

1. Open the MySQL CLI and log into the MySQL server.
2. Use the dbtest1 database as the default database by entering the USE command:

```
MariaDB [(none)]> use dbtest1;
Database changed
MariaDB [dbtest1]>
```

The CLI prompt shows the default database selected.

3. Enter the start of the CREATE TABLE statement defining the Employees table, along with the opening parenthesis to start the data field definition:

```
MariaDB [dbtest1]> CREATE TABLE employees (
    ->
```

The CLI prompt changes, indicating that it's waiting for the completion of the SQL statement.

4. Enter the individual data fields and their data types, with a comma at the end of each line, and press Enter at the end of each data field entry; after

the last data field, add the closing parenthesis and the semicolon to complete the statement:

```
-> employeeid int,
-> lastname varchar(50),
-> firstname varchar(50),
-> departmentcode char(5),
-> startdate date,
-> salary float);
Query OK, 0 rows affected (0.22 sec)

MariaDB [dbtest1]>
```

This creates the basic table defining the table name and the data fields but omits any data constraints and indexes. You can double-check that the table was created by using the SHOW TABLES statement:

```
MariaDB [dbtest1]> SHOW TABLES;
+-----+
| Tables_in_dbtest1 |
+-----+
| employees         |
+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

If you'd like to see the data fields contained in the table, use the SHOW CREATE TABLE statement:

```
MariaDB [dbtest1]> SHOW CREATE TABLE employees;
+-----+-----+
| Table | Create Table           |
+-----+-----+
| employees | CREATE TABLE `employees` (
  `employeeid` int(11) DEFAULT NULL,
  `lastname` varchar(50) DEFAULT NULL,
  `firstname` varchar(50) DEFAULT NULL,
  `departmentcode` char(5) DEFAULT NULL,
  `startdate` date DEFAULT NULL,
  `salary` float DEFAULT NULL
) ENGINE=InnoDB DEFAULT CHARSET=latin1 |
+-----+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

Now that the basic table exists, you can add any required data constraints and indexes.

Adding more table features

After you create a table, you can add, modify, or remove data fields using the `ALTER TABLE` statement. Here's the format of the `ALTER TABLE` statement:

```
ALTER TABLE tablename action
```

The `action` parameter can be one or more SQL commands used to modify the table. MySQL defines lots of actions that you can take on an existing table. Table 3-1 lists and describes the more common commands that you'll probably want to use.

TABLE 3-1

ALTER TABLE Actions

Action	Description
<code>ADD COLUMN name</code>	Add a new column (data field) to the table.
<code>DROP COLUMN name</code>	Remove an existing column from the table.
<code>ALTER COLUMN name MODIFY action</code>	Change the definition of an existing column based on the specified action.
<code>ADD constraint</code>	Add a new data constraint to the table.
<code>DROP constraint</code>	Remove an existing data constraint from the table.
<code>RENAME COLUMN old TO new</code>	Change the name of a table column.
<code>RENAME TO new</code>	Change the table name.

As you can tell from Table 3-1, there are lots of changes you can make to an existing table in MySQL using the `ALTER TABLE` statement! Follow these steps to try out a few of them:

1. Open the MySQL CLI and log in using the root user account.
2. Use the `dbtest1` database as the default by entering the `USE` command:

```
MariaDB [(none)]> use dbtest1;
Database changed
MariaDB [dbtest1]>
```

- 3. Submit an ALTER TABLE statement to add the primary key data constraint to the employeeid data field:**

```
MariaDB [dbtest1]> ALTER TABLE employees add primary key (employeeid);
Query OK, 0 rows affected (0.57 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dbtest1]>
```

- 4. Submit an ALTER TABLE statement to add the NOT NULL data constraint to the lastname data field:**

```
MariaDB [dbtest1]> ALTER TABLE employees MODIFY lastname varchar(50) NOT
NULL;
Query OK, 0 rows affected (0.58 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dbtest1]>
```

- 5. Enter an ALTER TABLE statement to add a new data field named birthdate, using the date data type:**

```
MariaDB [dbtest1]> ALTER TABLE employees ADD COLUMN birthdate date;
Query OK, 0 rows affected (0.30 sec)
Records: 0  Duplicates: 0  Warnings: 0

MariaDB [dbtest1]>
```

To make sure the table changes actually took effect, use the SHOW CREATE TABLE statement again:

```
MariaDB [dbtest1]> SHOW CREATE TABLE employees;
+-----+-----+
| Table | Create Table
|
+-----+-----+
| employees | CREATE TABLE `employees` (
  `employeeid` int(11) NOT NULL,
  `lastname` varchar(50) NOT NULL,
  `firstname` varchar(50) DEFAULT NULL,
  `departmentcode` char(5) DEFAULT NULL,
  `startdate` date DEFAULT NULL,
  `salary` float DEFAULT NULL,
  `birthdate` date DEFAULT NULL,
```

```
    PRIMARY KEY (`employeeid`)
) ENGINE=InnoDB DEFAULT CHARSET=latin1      |
+-----+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

The table definition now shows the updates made to the table — the primary key assigned to the `employeeid` data field, the new `NOT NULL` constraint for the `last-name` data field, and the new `birthdate` data field.

Removing a table using the MySQL CLI requires that you use the `DROP TABLE` statement:

```
DROP TABLE employees;
```

The `DROP` statement removes the entire table structure. If you just need to remove data records, use the `DELETE` statement instead.

Working with tables using Workbench

The graphical environment in Workbench makes creating tables much simpler than the MySQL CLI environment. As you would expect, it's just a matter of filling in a form!

Here are the steps to create a table using the MySQL Workbench tool:

1. Click the arrow icon next to the `dbtest2` database entry in the **Schemas** section of the Navigator pane.
2. Right-click the **Tables** menu item, and select **Create Table**.
The New Table form appears, as shown in Figure 3-7.
3. Enter the table name of `employees` in the **Table Name** text box.
4. Click in the text area that shows the **Column Name** field.
An empty text box appears for the Column Name and Datatype.
5. Enter `employeeid` for the **Column Name**, and select **INT** from the **Datatype** drop-down box.

Notice that Workbench automatically selects the Primary Key and Not Null constraint check boxes in the form for the first data field you enter. Keep those checked.

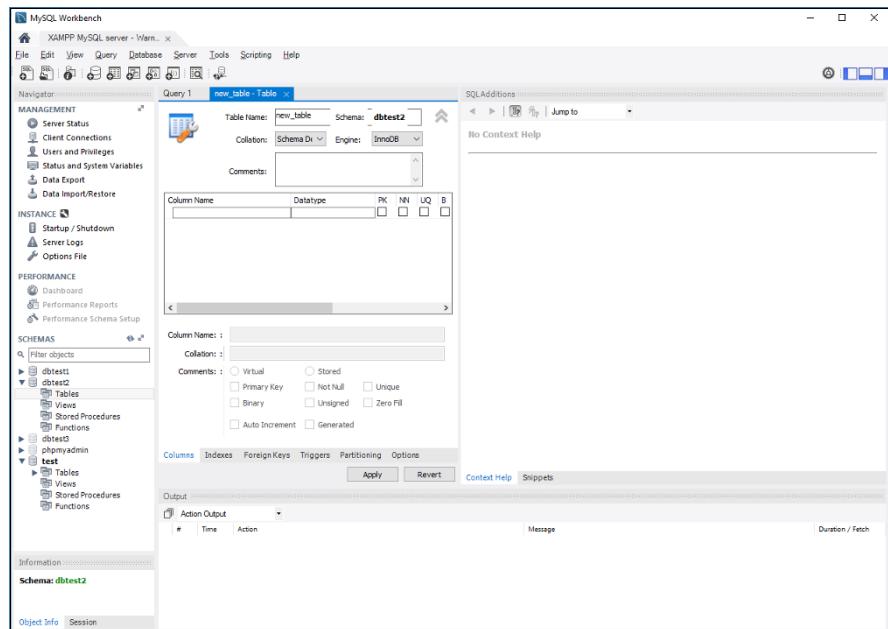


FIGURE 3-7:
Creating a new table using Workbench.

6. Click the empty line under the employeeid data field in the form.

A new data field name of employeescol appears.

7. Double-click the new employeescol name to change it to lastname.

A default data type of varchar(45) appears in the DataType column.

8. Change varchar(45) to varchar(50).

9. Check the Not Null check box in the form.

10. Repeat steps 4 through 9 to add the remainder of the table data fields:

```

lastname varchar(50) Not Null
firstname varchar(50)
departmentcode char(5)
startdate date
birthdate date
salary float

```

When you complete the form, it should look like what's shown in Figure 3-8.

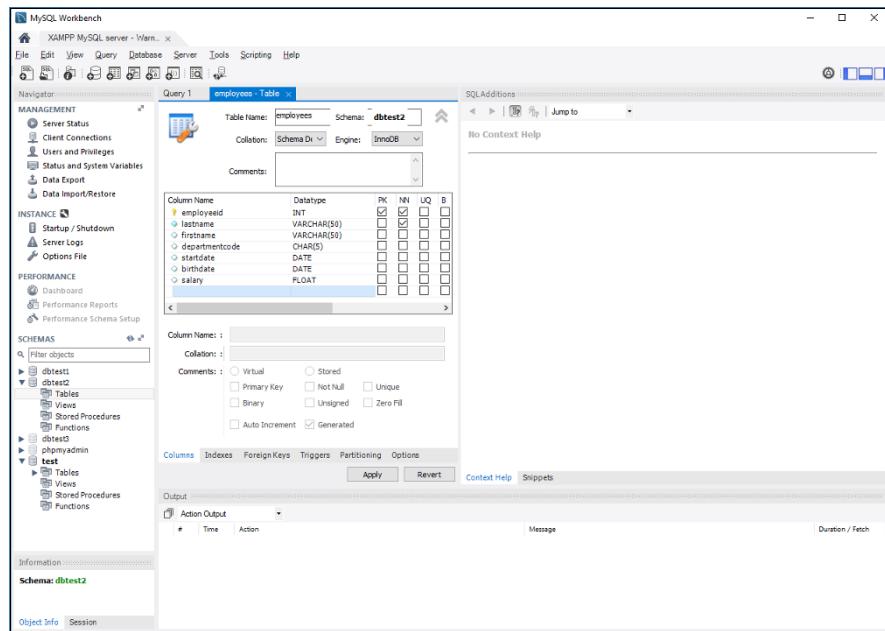


FIGURE 3-8:
The completed
New Table
form for the
Employees
table.

11. Click Apply.

A wizard appears, showing the SQL code generated from the information you entered into the form, as shown in Figure 3-9.

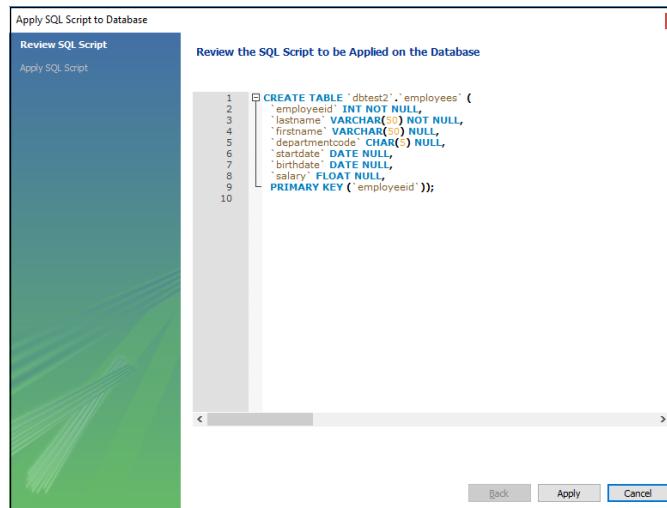


FIGURE 3-9:
The CREATE
TABLE statement
generated by
Workbench.

Notice that the CREATE TABLE statement generated by Workbench to create the table looks just like what you did manually using the MySQL CLI.

12. Click Apply to submit the SQL statement to create the table.

The status of the submitted SQL statement appears.

13. Click the Finish button to close out the wizard.

When you return to the main Workbench interface, click the arrow next to the Tables entry under the dbtest2 database. You should now see the new Employees table added, as shown in Figure 3-10.

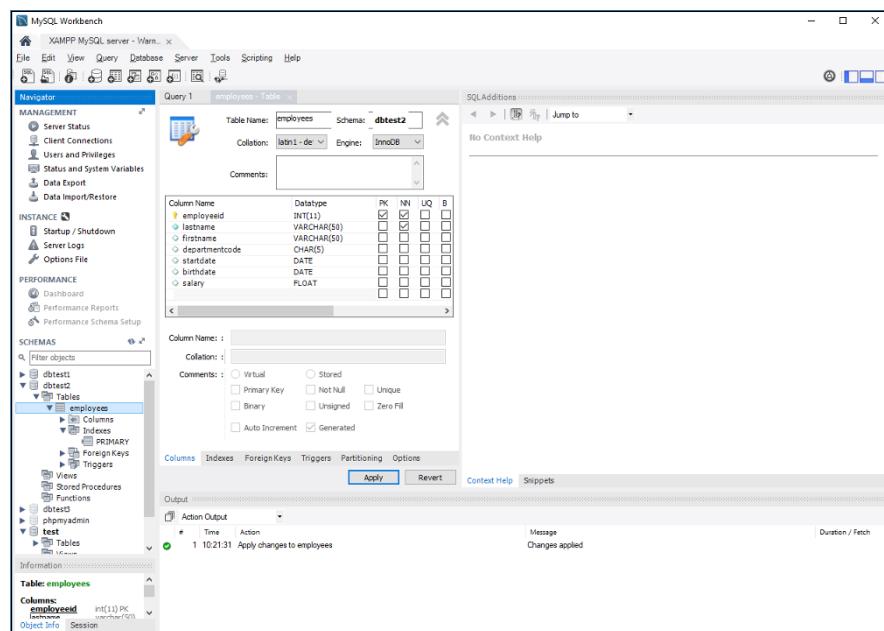


FIGURE 3-10:
Viewing the
Employees
table created
in the dbtest2
database.

From here you can modify any of the data field names, data types, or data constraints. You can remove the table by right-clicking on the table name and then selecting the Drop Table entry from the pop-up menu.

Working with tables in phpMyAdmin

phpMyAdmin also provides a pretty fancy graphical interface for creating your tables. Follow these steps to create a new table using phpMyAdmin:

- 1. After you open the phpMyAdmin web page in your browser, click the dbtest3 database entry from the database list on the left-hand side of the main page.**

This takes you to the database structure page, shown in Figure 3-11.

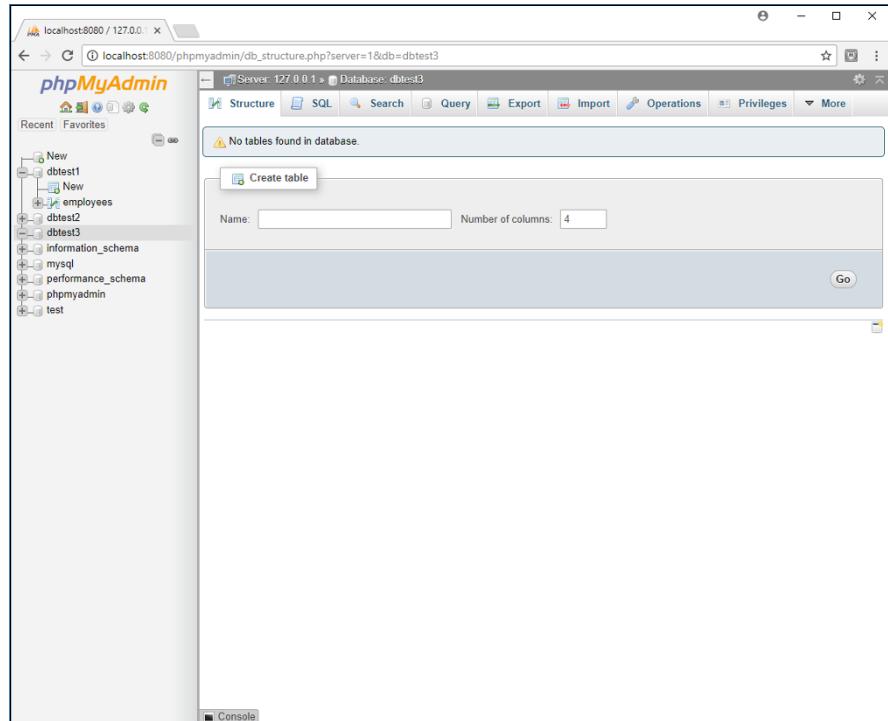


FIGURE 3-11:
The database structure page in phpMyAdmin.

2. Enter employees in the Name text box.
3. Change the number of columns to 7.
4. Click Go.

This produces the table data field form, shown in Figure 3-12.

5. Fill in the top form field with the employeeid data field information.
6. Click the Index drop-down box and select PRIMARY.

A pop-up window appears, prompting you for additional information on the index key, as shown in Figure 3-13.

7. Click Go to accept the default values.

Be careful with the NOT NULL data constraint when using phpMyAdmin. Notice that it provides a Null check box. If you select the check box, that means the data field can have a Null value. Keep the check box empty to apply the NOT NULL data constraint.



WARNING

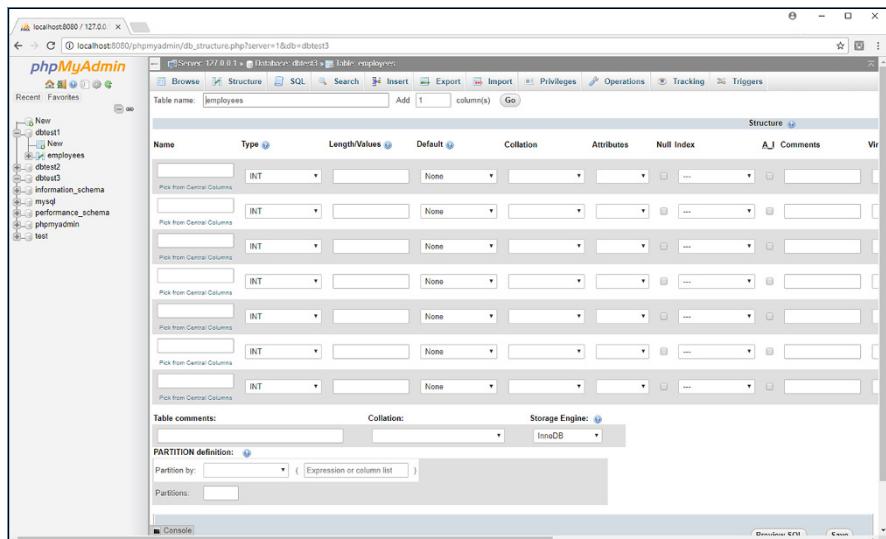


FIGURE 3-12:
The empty new table form in phpMyAdmin.

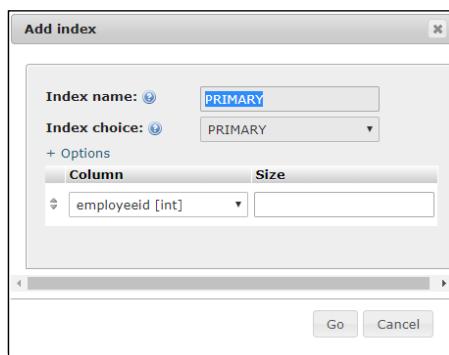


FIGURE 3-13:
The Index dialog box in phpMyAdmin.

8. Complete the form for the rest of the Employees table fields, as shown in Figure 3-14.
9. If you'd like to see the SQL CREATE TABLE statement that the form information would generate ahead of time, click the Preview SQL button at the bottom of the form page.
10. Click the Save button to create the table.

After phpMyAdmin submits the SQL to create the table, it automatically redirects you to the structure page for the new table, as shown in Figure 3-15.

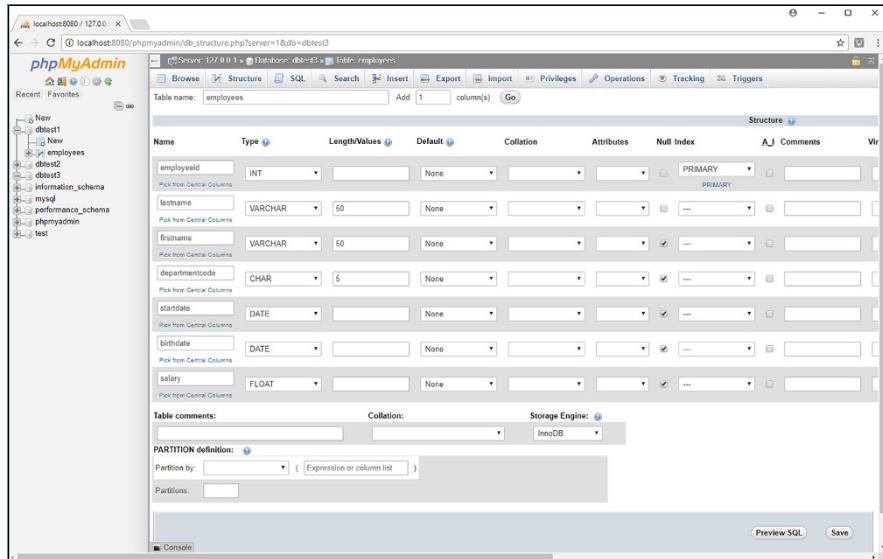


FIGURE 3-14:
The completed new table form in phpMyAdmin.

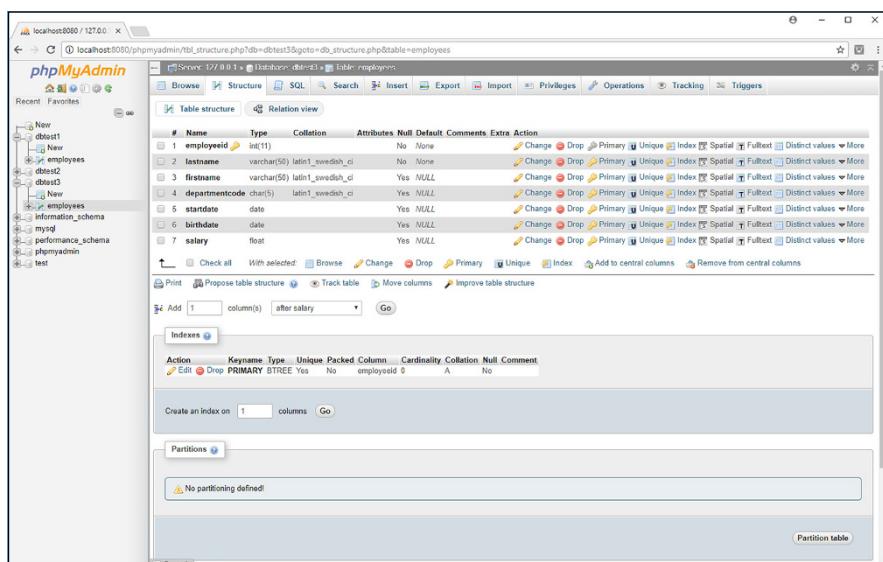


FIGURE 3-15:
The phpMyAdmin table structure page.

The table structure page is a very busy web page! It shows the data fields for the table, along with a series of action icons for each data field. From here, you can change any of the data field properties, along with adding a new data field or removing an existing data field.

If you click the dbtest3 database link on the left side of the web page, phpMyAdmin will take you back to the database Structure page. This time, because you have an existing table in the database, the Structure page shows the table, along with some action icons, as shown in Figure 3-16.

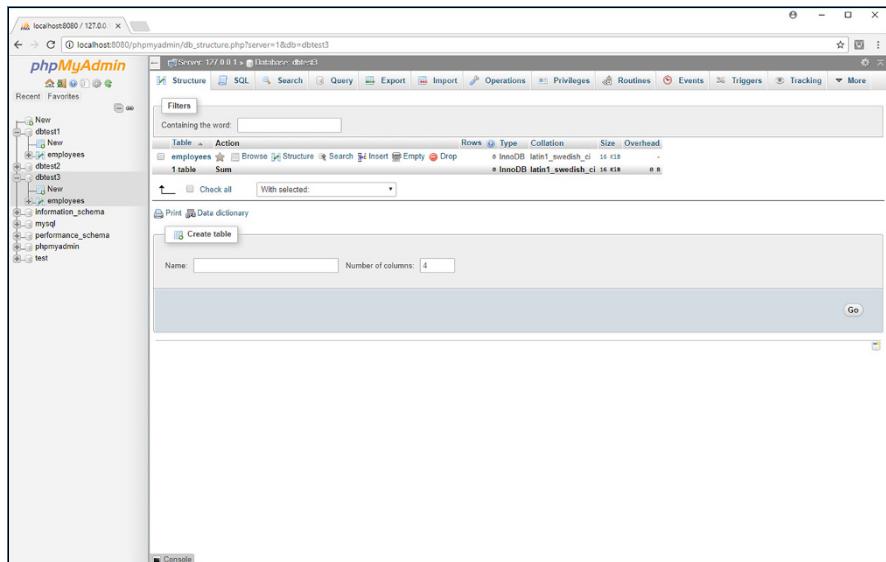
A screenshot of the phpMyAdmin interface. The left sidebar shows a tree view of databases: 'New' (selected), 'dbtest1', 'dbtest2', 'dbtest3' (selected), 'New', 'employees', 'test'. The main content area is titled 'Structure' and shows a table named 'employees'. A table structure is displayed with columns: id (Type: INT, Collation: latin1_swedish_ci, Size: 10 kB, Null: NO, Default: NULL), name (Type: VARCHAR, Collation: latin1_swedish_ci, Size: 10 kB, Null: YES, Default: NULL), age (Type: INT, Collation: latin1_swedish_ci, Size: 10 kB, Null: YES, Default: NULL), and salary (Type: DECIMAL, Collation: latin1_swedish_ci, Size: 10 kB, Null: YES, Default: NULL). Below the table, there are buttons for 'Action', 'Browse', 'Structure', 'Search', 'Insert', 'Empty', 'Drop', 'Sum', 'Print', 'Data dictionary', and 'Create table'. A modal window titled 'Create table' is open at the bottom, asking for 'Name:' and 'Number of columns: 4'.

Table	Action	Rows	Type	Collation	Size	Overhead
employees	Browse Structure Insert Empty Drop Sum	1 table	InnoDB	latin1_swedish_ci	10 kB	0 kB

FIGURE 3-16:
The phpMyAdmin database structure page with an existing table.

From here you can remove the table by clicking the Drop link or delete the data from the table by clicking the Empty link. To get back to the table structure page to view the data fields, click the Structure link.

IN THIS CHAPTER

- » Adding new data to your tables
- » Updating existing data in your tables
- » Finding data quickly
- » Working with backups and restores

Chapter 4

Using the Database

The preceding chapter covers how to create databases and tables for your dynamic web application. That's all well and good, but databases and tables don't really do anything until you start placing data in them.

This chapter explores the different methods you have available for adding, changing, and removing data in your application tables. After that, it walks through possibly the most important feature of any database: how to quickly retrieve the data that your application needs. The chapter closes by discussing the important jobs of backing up and restoring database data.

Working with Data

The ability to easily manage application data is the whole reason dynamic web applications use databases. So it stands to reason that the SQL language has quite a few options for working with data. There are four basic functions that we need to do with the data in our application:

- » Add new data records to tables.
- » Modify existing data records in tables.
- » Remove unwanted data records from tables.
- » Query existing data for specific information.

This section walks through how to accomplish the first three items in this list using the three different MySQL interfaces I cover earlier in this minibook — the MySQL command-line interface (CLI), the graphical MySQL Workbench tool, and the web-based phpMyAdmin tool. Querying data is a complex topic, so I save that for its own section. Let's get started and look at managing the data in your tables.

The MySQL command-line interface

The MySQL CLI uses standard SQL statements to interact with the MySQL server. There are just three basic SQL statements that you need to know to manage data in your database tables:

- » `INSERT`: To add new data records to a table
- » `UPDATE`: To modify existing data records in a table
- » `DELETE`: To remove existing data records from a table

The following sections describe these three statements and show how to use them in your application.

Adding new data

You use the `INSERT SQL` statement to add one or more new data records to a table in the database. A data record consists of a single instance of data values for each data field.



In some MySQL documentation, you'll often see the terms *column* used to refer to a single data field and *tuple* used to refer to an entire data record. I'll use the more generic terms *data field* and *data record* in this book.

Here's the basic format of the `INSERT` statement:

```
INSERT INTO table [(fieldlist)] VALUES (valuelist)
```

The *fieldlist* parameter is optional. By default, the `INSERT` statement tries to load comma-separated values from the *valuelist* into each data field in the table, in the order the data fields appear in the table definition. Chapter 3 of this minibook explains how you can use the `SHOW CREATE TABLE` statement to list the data fields in the table. Another method is to use the `DESCRIBE` SQL statement:

```
MariaDB [dbtest1]> DESCRIBE employees;
+-----+-----+-----+-----+
| Field | Type | Null | Key | Default | Extra |
+-----+-----+-----+-----+
```

```
+-----+-----+-----+-----+
| employeeid | int(11) | NO   | PRI  | NULL   |
| lastname    | varchar(50) | NO   |       | NULL   |
| firstname   | varchar(50) | YES  |       | NULL   |
| departmentcode | char(5) | YES  |       | NULL   |
| startdate   | date    | YES  |       | NULL   |
| salary       | float   | YES  |       | NULL   |
| birthdate   | date    | YES  |       | NULL   |
+-----+-----+-----+-----+
7 rows in set (0.01 sec)

MariaDB [dbtest1]>
```

It doesn't show the exact SQL statement used to create the table, but it produces a quick summary of the data fields contained in the table. That's all you need to see what order the data fields appear in the table for the `INSERT` statement.

Follow these steps to enter a data record into the `employees` table that you created back in Chapter 3 of this minibook. (If you skipped that part, or haven't read it yet, just jump back there and run the `CREATE` statements to do that. I'll wait.)

- 1. Ensure that the MySQL server is started, and then open the MySQL CLI program.**
- 2. Log in as the `root` user account.**
- 3. Specify the `dbtest1` database from Chapter 3 as the default database by entering the `USE` statement:**

```
MariaDB [(none)]> USE dbtest1;
Database changed
MariaDB [dbtest1]>
```

- 4. Add a new data record by entering the `INSERT` statement:**

```
MariaDB [dbtest1]> INSERT INTO employees VALUES
    -> (123, 'Blum', 'Rich', 5, '2020-01-01', 10000, '2000-05-01');
Query OK, 1 row affected (0.12 sec)

MariaDB [dbtest1]>
```



TIP

In the `INSERT` statement, text and date values must be enclosed in quotes to delineate the start and end of the text value. Numeric values don't need to use quotes. Notice that I split the `INSERT` statement into two parts here. That's not necessary, but it can come in handy when you don't want too long of a line for the `INSERT` statement.

The INSERT statement returns a status message indicating how many data record rows were successfully added to the table. (If you need to, you can specify more than one group of data values in the *valuelist*, surrounding each with the parentheses.)

5. Check to ensure the data was added correctly by using the SELECT statement:

```
MariaDB [dbtest1]> SELECT * FROM employees;
+-----+-----+-----+-----+-----+
| employeeid | lastname | firstname | dcode | startdate | salary |
|           |         |           |       |           |        |
+-----+-----+-----+-----+-----+
|      123 | Blum     | Rich      | 5    | 2020-01-01 | 10000 |
|           |           |           |       |           |        |
+-----+-----+-----+-----+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

The SELECT statement shows the data fields in the table (I truncated the departmentcode data field name in this output so it would fit the width of the book page), and then shows the data records contained in the table.

If you don't want to assign values to all the data fields in the data record, you must include the *fieldlist* parameter. This specifies the data fields (and the order) that the data values will be placed in:

```
MariaDB [dbtest1]> INSERT INTO employees (employeeid, lastname, firstname)
-> VALUES (124, 'Blum', 'Barbara');
Query OK, 1 row affected (0.10 sec)

MariaDB [dbtest1]>
```



WARNING

Be careful when skipping data fields when adding a new data record. If a data field that uses the NOT NULL data constraint isn't assigned a data value, the server may reject the INSERT statement. I say "may" because it depends on the configuration of the MySQL server. To maintain backward compatibility with older versions of MySQL, by default MySQL won't enforce some data constraints, such as the NOT NULL constraint by default. To enforce it, you must change the `sql_mode` setting, either in the MySQL server configuration, or by setting it in the MySQL connection session. The `sql_mode` setting value of `STRICT_ALL_TABLES` tells MySQL to enforce

all data constraints on all tables. When you do that, you'll get an error message if you don't supply a value for any data field that uses the NOT NULL constraint:

```
MariaDB [dbtest1]> set sql_mode=STRICT_ALL_TABLES;
Query OK, 0 rows affected (0.00 sec)

MariaDB [dbtest1]> INSERT INTO employees (employeeid, firstname) VALUES
-> (126, 'Katie');
ERROR 1364 (HY000): Field 'lastname' doesn't have a default value
MariaDB [dbtest1]>
```

Modifying existing data

If you need to change data that you've already entered into the table, don't worry — all is not lost. You can modify any existing data records in the table, as long as the privileges assigned to your MySQL user account contains the UPDATE privilege.

You use the UPDATE SQL statement for updating one or more data records contained in the table. The UPDATE statement is another of those SQL statements that, though simple in concept, can easily get complex. Here's the basic format for the UPDATE statement:

```
UPDATE table SET datafield = value [WHERE condition]
```

The basic format of this statement specifies a *datafield* in the *table* to change the data value of that data field to the *value* specified. The WHERE clause specifies the condition that a data record must meet to have the change applied to it. However, notice that it's optional, which can cause lots of problems.

Here's the way the scenario often plays out: Suppose you need to go back into the Employees table to change the NULL startdate value for Barbara that wasn't supplied when the data record was created. If you just use the basic format for the UPDATE statement, you'll get a surprise:

```
MariaDB [dbtest1]> UPDATE employees SET startdate = '2020-01-02';
Query OK, 2 rows affected (0.10 sec)
Rows matched: 2  Changed: 2  Warnings: 0

MariaDB [dbtest1]>
```

Your first clue that something bad happened would be the output returned from the MySQL server. The Rows matched and the Changed fields indicate that two

data records were updated — but you just wanted to change one data record. Running a `SELECT` statement will verify your mistake:

```
MariaDB [dbtest1]> select * From employees;
+-----+-----+-----+-----+-----+-----+-----+
| employeeid | lastname | firstname | dcode | startdate | salary | birthdate |
+-----+-----+-----+-----+-----+-----+-----+
|       123 | Blum     | Rich      | 5      | 2020-01-02 | 10000  | 2000-05-01 |
|       124 | Blum     | Barbara   | NULL   | 2020-01-02 | NULL    | NULL      |
+-----+-----+-----+-----+-----+-----+-----+
2 rows in set (0.00 sec)

MariaDB [dbtest1]>
```

The basic `UPDATE` statement changed the `startdate` data field value for all of the data records in the table! This is an all-too-common mistake made by even the most experienced database administrators and programmers when in a hurry. By default, MySQL applies the update to all the table data records.

To solve that problem, you just need to add the `WHERE` clause to specify exactly which data record(s) you intend the change to apply to:

```
UPDATE employees SET startdate = '2020-01-01' WHERE employeeid = 123;
```



TIP

It's a good practice to get in the habit of always including a `WHERE` clause in your `UPDATE` statements, even if you really do want the update to apply to all the data records. That way, you know the update will always be applied to the correct data records and avoid costly mistakes.

Deleting existing data

The `DELETE` statement allows you to remove data from a table but keep the actual table intact (unlike the `DROP` statement, which removes the table and the data). Here's the format for the `DELETE` statement:

```
DELETE FROM table [WHERE condition]
```

This statement works similar to the `UPDATE` statement. Any data records matching the `condition` listed in the `WHERE` clause are deleted. And just like the `UPDATE` statement, if you leave off the `WHERE` clause, the `DELETE` function applies to all the data in the table. Make sure you really mean that before using it!

Here are a couple of examples of using the `DELETE` statement:

```
MariaDB [dbtest1]> DELETE FROM employees WHERE employeeid = 124;
Query OK, 1 row affected (0.08 sec)

MariaDB [dbtest1]> DELETE FROM employees WHERE employeeid = 124;
Query OK, 0 rows affected (0.00 sec)

MariaDB [dbtest1]>
```

In the second example, I tried to delete a data record that I had already deleted. Notice that when the `DELETE` statement fails to find any data records to delete, it does not produce an error message; instead, it just indicates in the return status that the number of data records deleted was zero.

The MySQL Workbench tool

Thanks to its graphical interface, working with table data using MySQL Workbench is a breeze. You don't need to memorize any SQL statements — just fill out a form and apply it to the database. Much like ordering a pizza!

Follow these steps to experiment with the data management features in Workbench:

1. Ensure that the MySQL database server is running, and then open the MySQL Workbench tool.
2. Double-click the `dbtest2` database link in the Navigator pane, under the **Schemas** section.
3. Double-click the **Tables** link under the `dbtest2` link.
4. Hover the mouse pointer over the **Employees** table entry.

Three icons appear:

- An *i* icon, which displays information about the table
- A wrench icon, which allows you to modify the table structure
- A spreadsheet table icon, which allows you to manage data in the table

5. Click the spreadsheet table icon next to the **Employees** table entry.

The Result Grid pane appears under the Query1 pane, as shown in Figure 4-1.

The Result Grid pane shows the existing table data (if any) in a grid layout. Each row in the grid is a data record in the table.

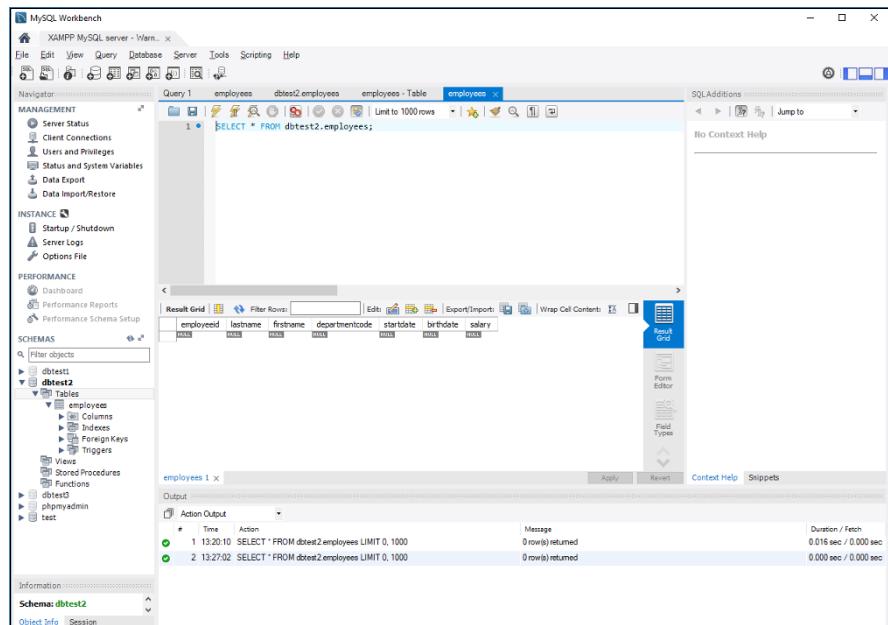


FIGURE 4-1:
The Workbench
Result Grid
for displaying
table data.



TIP

Depending on the size of the Workbench window, the Result Grid area may be truncated on the right-hand side. If that happens, grab the margin line at the right-hand edge of the Result Grid area and drag it to the right to expand the pane.

6. To enter a new data record, either double-click in the empty grid row at the bottom of the table or, if your grid is very long, click the Insert Row icon at the top of the grid to jump to the empty grid row.
7. To modify an existing single data value in a data record, single-click the value in the grid and replace the existing value with the new value.
8. To remove an existing data record, highlight the grid row by clicking the empty cell at the left-hand side of the row, and then click the Delete selected rows icon at the top of the Result Grid pane.
9. To apply the changes to the table, click the Apply button at the bottom of the pane.

The Apply SQL Wizard appears, as shown in Figure 4-2.

The wizard shows the SQL statements generated to add, modify, or delete the data records based on the changes you made in the data grid.

10. Click the Apply button to apply the SQL statements to the table.
11. Click the Finish button to close the wizard.

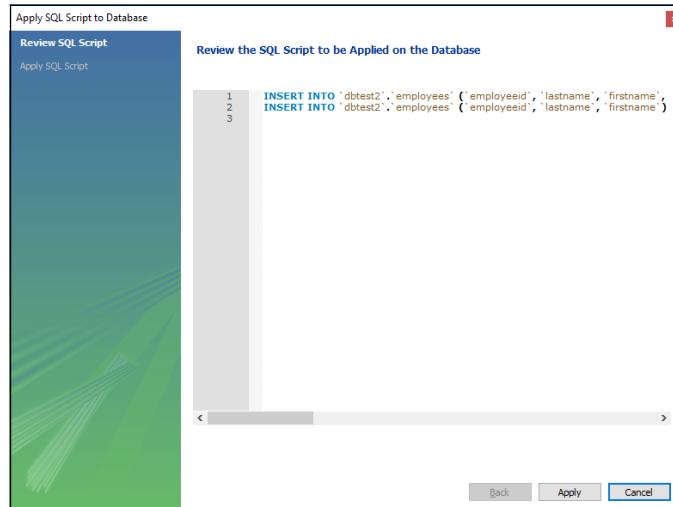


FIGURE 4-2:
The Apply
SQL Wizard in
Workbench.



WARNING

The Result Grid can be a bit misleading. Just making the changes in the grid display doesn't commit them to the table. You have to click the Apply button to run the wizard to commit the changes, or else they'll be gone when you close out the grid!

If you feel a bit restricted by the small area of the result grid, click the Form Editor button on the right-hand side of the pane. That displays a single data record in the table using a form format, as shown in Figure 4-3.

Using the Database

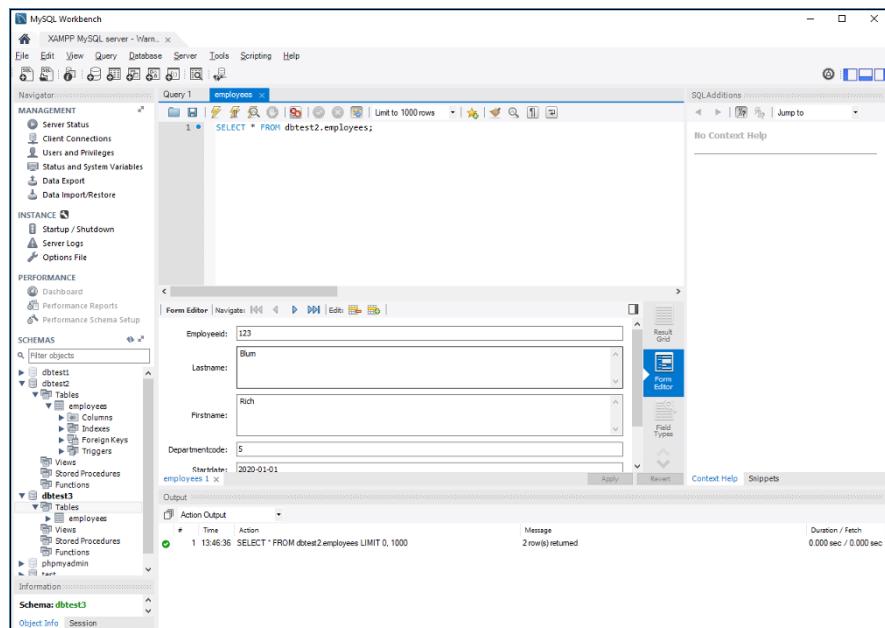


FIGURE 4-3:
Using the
Form Editor in
Workbench to
manage data
records.

The Form Editor does the same thing as the Result Grid but provides a single data record interface, giving you more room for viewing long data fields. Again, if you make any changes in the Form Editor, make sure to click the Apply button at the bottom to commit the changes.

Making changes to data in a table doesn't get any easier than that!

The phpMyAdmin tool

The phpMyAdmin web-based tool also provides a graphical interface for working with your table data, but it's a little more complicated than Workbench. Instead of using a single interface for all data management, phpMyAdmin breaks them up into a couple of different interfaces.

Follow these steps to insert new data using phpMyAdmin:

1. Ensure that the MySQL server is running, and then open your browser and go to the phpMyAdmin URL for your system.

For XAMPP it's `http://localhost:8080/phpmyadmin/`. Note that the TCP port may be different for your server environment.

- 2.** Click the dbtest3 database link on the left-hand side of the main phpMyAdmin web page.

This produces the Database web page, as shown in Figure 4-4.

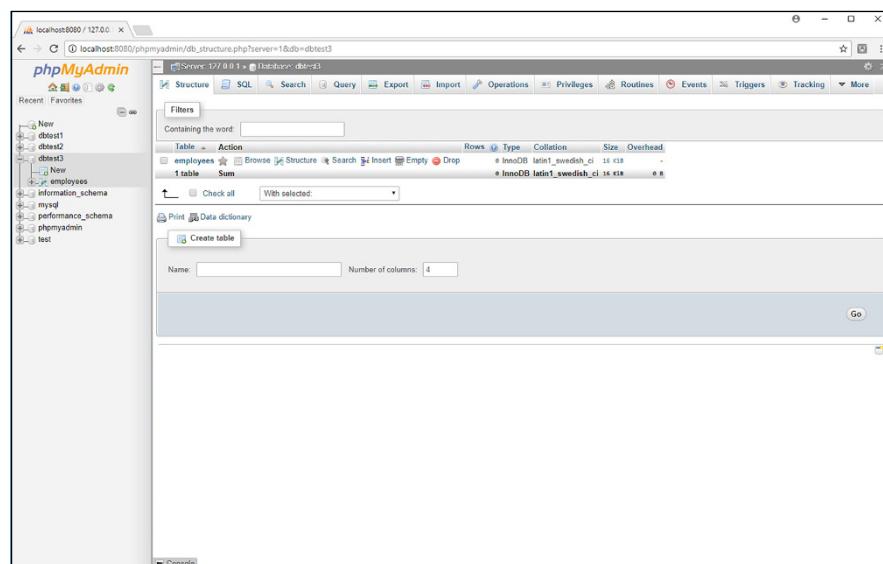


FIGURE 4-4:
The phpMyAdmin
Database
web page.

3. To add a new data record, click the Insert link in the Actions section.

This produces a form to insert one or two new data records, as shown in Figure 4-5.

The screenshot shows the phpMyAdmin interface for the 'employees' table. At the top, there are two sets of input fields for inserting data. Each set includes fields for employee ID (int(11)), last name (varchar(50)), first name (varchar(50)), department code (char(5)), start date (date), birth date (date), and salary (float). Below these fields is a note about ignoring columns. At the bottom of the form, there are 'Go' and 'Cancel' buttons, and a note indicating that clicking 'Insert as new row' will add the data to the table.

FIGURE 4-5:
The INSERT form
in phpMyAdmin.

4. Enter data values in the appropriate data fields, and then click the Go button to add the data record.

When you click the Go button, phpMyAdmin generates the `INSERT` statement and submits the data record to the table. It then takes you back to the Database web page, showing the status for the completed statement.

Managing existing data in a table uses a different interface in phpMyAdmin. Follow these steps to manage the existing data records in the table:

1. Open your browser and enter the following phpMyAdmin URL:

`http://localhost:8080/phpmyadmin/`

- 2. Click the dbtest3 database link on the left-hand side of the main phpMyAdmin web page.**
- 3. Click the Browse icon in the employee table actions section of the Database web page.**

This produces a list of all the data records contained in the table, as shown in Figure 4-6.

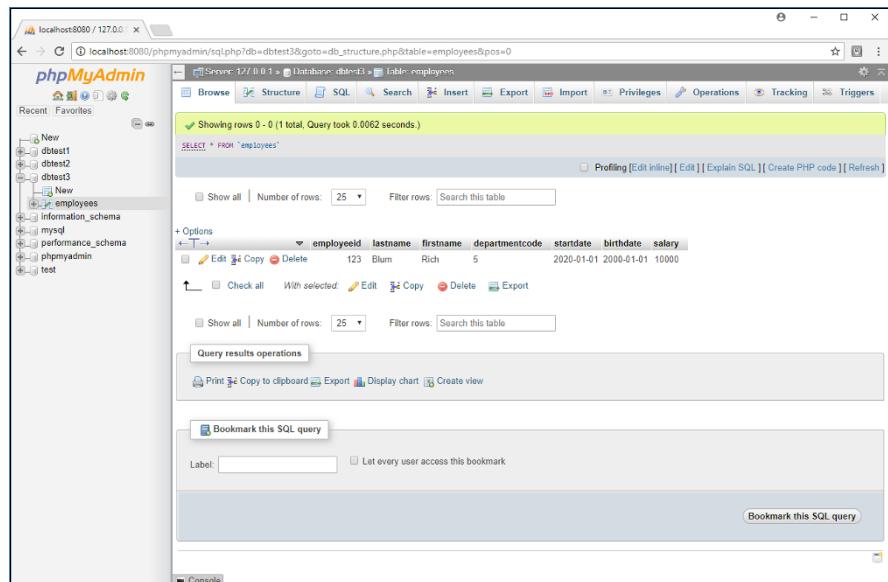


FIGURE 4-6:
The phpMyAdmin window for browsing data records.

4. Click the **Edit** icon for the data record you need to modify or the **Delete** icon for the data record you need to delete.

To delete multiple data records, select the check boxes for those data records, and then click the Delete icon at the bottom of the data record list.

5. Click the **Go** button to confirm editing or deleting the selected data record.

Thanks to the graphical interface in phpMyAdmin, entering and managing data is still a simple process. However, finding specific data records in an application can be somewhat tricky, even when using a graphical interface. The next section tackles that topic.

Searching for Data

Quite possibly the most important function you'll perform in your dynamic web applications is to query existing data in the database. Many web developers spend a great deal of time concentrating on the design layout of the web pages, but the real heart of the application is the behind-the-scenes SQL used to query data to produce the website content. If this code is inefficient, it can cause huge performance problems, and possibly even make the web application virtually useless to customers — no matter how fancy the web pages look.

As a good database application developer, it's essential that you understand how to write good SQL query statements. The SQL statement used for queries is SELECT. Because of its importance, a lot of work has been done on the format of the SELECT statement, to make it as versatile as possible. Unfortunately, with versatility comes complexity.

Because of the versatility of the SELECT statement, the statement format has become somewhat unwieldy and intimidating for the beginner. To try and keep things simple, in this section I walk through the different features of the SELECT statement one piece at a time. The next few sections demonstrate how to use these features of the SELECT statement.

The basic SELECT format

The basic format for the SELECT statement seems simple enough:

```
SELECT fieldlist FROM table
```

The *fieldlist* parameter specifies the data fields that should appear in the output from the *table* you specify. The *fieldlist* can be a comma-separated list of specific data fields in the table, or the wildcard character (the asterisk) to specify all data fields, as shown in the SELECT example I use earlier in this chapter:

```
SELECT * FROM employees;
```

This statement returns all the data field values for all the data records contained in the Employees table. If that's all you need for your application, you don't need to know anything more about the SELECT statement (lucky you)!

However, more than likely, you'll need to customize just what data fields (and data records) need to appear in the output. That's where things start getting complicated. The following sections show some more features that you may need to use in your SELECT statements.

Sorting output data

The output from a SELECT statement is called a *result set*. The result set contains only the data fields specified in the SELECT statement. The result set is only temporary and, by default, is not stored in any tables in the database.

By default, the data records displayed in the result set are not displayed in any particular order. As records are added or removed from the table, MySQL may place new data records anywhere within the table order. Even if you enter data in a particular order using INSERT statements, there is still no guarantee that the records will display in the same order in the result set.

If you need to specify the order in which the data records appear in your output, you must add the ORDER BY clause to the SELECT statement:

```
> SELECT employeeid, lastname, firstname FROM employees ORDER BY firstname;
+-----+-----+-----+
| employeeid | lastname | firstname |
+-----+-----+-----+
|      124 | Blum     | Barbara   |
|      126 | Blum     | Jessica   |
|      125 | Blum     | Katie     |
|      123 | Blum     | Rich      |
+-----+-----+-----+
4 rows in set (0.00 sec)

>
```

In this example, only the data fields specified in the SELECT statement are displayed, ordered by the `firstname` data field. The default order used by the ORDER BY clause is ascending order, based on the data type and collation you select when creating the table. You can change the order to descending by adding the DESC keyword at the end of the ORDER BY clause:

```
ORDER BY firstname DESC;
```

This gives you complete control over how the data records appear in the result set output.

Filtering output data

By default, the SELECT statement places all the data records in the table in the result set output. The power of the database query comes from displaying only a subset of the data that meets a specific condition.

You add the WHERE clause to the SELECT statement to determine which data records to display in the result set output. Now we're getting to the heart of the SELECT statement!

For example, you can check for all the employees who work in the department identified by departmentcode 5 by using the following query:

```
MariaDB [dbtest1]> SELECT * FROM employees WHERE departmentcode = 5;
+-----+-----+-----+-----+-----+-----+
| id  | lname | fname | departmentcode | startdate | salary | birthdate |
+-----+-----+-----+-----+-----+-----+
| 123 | Blum  | Rich  | 5           | 2020-01-02 | 10000  | 2000-05-01 |
```

```

+-----+-----+-----+-----+
| 125 | Blum | Katie | 5   |
+-----+-----+-----+-----+
2 rows in set (0.00 sec)

MariaDB [dbtest1]>

```

The result set only contains the data records from the table that match the WHERE clause condition you specified. In this example, the data field was an integer type, but if the data field you use is a text or date value, you must place quotes around the value to delineate the start and end of the value:

```
SELECT * FROM employees WHERE startdate < "2020-03-01";
```



WARNING

In the WHERE clause condition, the collation you define for the data field is important. MySQL evaluates the specified condition based on the collation defined. If you use a case-insensitive collation, MySQL can't tell the difference between the values Rich and rich. Be very careful in selecting the collation you use for tables, because that plays an important role in just how your application can handle the data contained in the tables.

More advanced queries

Now that you've seen the basics (and the power) of the SELECT statement, let's dive into some more complex topics. The following sections help add to your SELECT querying skills by showing you how to do some pretty complex searches in your database!

Querying from multiple tables

In a relational database, data is split into several tables in an attempt to keep data duplication to a minimum. In Chapter 3 of this minibook, I show you how to apply the second normal form rule of database design to create separate Customers and Orders tables so that the customer information didn't need to be duplicated for every order data record. Although this helps reduce data redundancy, it produces a small problem for your application queries.

When your application needs to generate a report for an order, it most likely will need the customer's address information to place on the report. That means now your program needs to retrieve the order information from the Orders table, and the customer information from the Customers table.

You can do that with two separate queries:

- 1. Query the Orders table with the `orderid` value to retrieve the `customerid`.**
- 2. Query the Customers table with the `customerid` to retrieve the customer address information for that order.**

However, the two separate queries do take some extra processing time, both in your PHP application code and in the MySQL server. A more efficient way of retrieving that information is to submit a single `SELECT` statement that retrieves the data from both tables.

To query data from multiple tables in a single `SELECT` statement, you must specify both tables in the `FROM` clause. Also, because you're referencing data fields from both tables in the data field list, you must indicate which table each data field comes from. That looks like this:

```
MariaDB [dbtest1]> SELECT orders.orderid, customers.name, customers.address
    -> FROM orders, customers
    -> WHERE orderid = 1000 AND orders.customerid = customers.customerid;
+-----+-----+-----+
| orderid | name      | address           |
+-----+-----+-----+
|   1000  | Acme Paper | 134 Main St.; Miami, FL |
+-----+-----+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

As you can see from this example, it doesn't take long for a seemingly simple `SELECT` statement to get complex! Let's walk through just what this statement does.

The first line in the query defines the data fields you want to see in the result set output. Because you're using data fields from two tables, you must precede each data field name with the table it comes from.

In the second line, you have to define which tables the data fields come from in the `FROM` clause. You can list the tables in any order here.

Finally, in the `WHERE` clause, you have to define the condition that filters out the records you want to display. In this example, there are two conditions that must be met:

- » You need the Orders table data record that meets the specific `orderid` value you're looking for.
- » You need the Customers table data record that matches the `customerid` value for that specific order.

You use the logical AND operator to combine the two conditions. The result set contains the data record values that meets both of those conditions.

Using joins

In the previous example, you had to write a lot of code in the WHERE clause to match the appropriate data record from the Customers table to the Orders table data record information. In a relational database, this is a common thing to do. To help programmers, the SQL designers came up with an alternative way to perform this function.

A database *join* matches related data records in relational database tables without your having to perform all the associated checks in your code. Here's the format for using the join in a SELECT statement:

```
SELECT fieldlist FROM table1 jointype table2 ON condition
```

The *fieldlist* parameter lists the data fields from the tables to display in the output as usual. The *table1* and *table2* parameters define the two tables to perform the join on. The *jointype* parameter determines the type of join for MySQL to perform. There are three types of joins available in MySQL:

- » INNER JOIN: Only displays data records found in both tables.
- » LEFT JOIN: Displays all records in *table1* and the matching data records in *table2*.
- » RIGHT JOIN: Displays all records in *table2* and the matching data records in *table1*.

The LEFT and RIGHT join types are also commonly referred to as *outer joins*. The order in which you specify the tables and the join type that you use are very important to the join operation.

Finally, the *ON condition* clause defines the data field relation to use for the join operation.

It's common practice to use the same data field name for data fields in separate tables that contain the same information (such as the `customerid` data field

in the Customers and Orders tables). You can add the NATURAL keyword before the join type to tell MySQL to join using the common data field name. Here's an example of querying the customer information for all the orders using a NATURAL INNER JOIN:

```
MariaDB [dbtest1]> SELECT orders.orderid, customers.name, customers.address
-> FROM orders NATURAL INNER JOIN customers;
+-----+-----+-----+
| orderid | name      | address          |
+-----+-----+-----+
|    1000 | Acme Paper | 134 Main St.; Miami, FL |
|    1001 | Acme Paper | 134 Main St.; Miami, FL |
|    1002 | Acme Machines | 264 Oak St.; Los Angeles, CA |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

Now that's a lot less typing to mess with! The result set shows all the data records in the Orders table that have matching customerid data records in the Customers table.

Another way of doing this is to add the USING clause to a JOIN statement:

```
MariaDB [dbtest1]> SELECT orders.orderid, customers.name, customers.address
-> FROM orders LEFT JOIN customers USING (customerid);
+-----+-----+-----+
| orderid | name      | address          |
+-----+-----+-----+
|    1000 | Acme Paper | 134 Main St.; Miami, FL |
|    1001 | Acme Paper | 134 Main St.; Miami, FL |
|    1002 | Acme Machines | 264 Oak St.; Los Angeles, CA |
+-----+-----+-----+
3 rows in set (0.00 sec)
```

The USING keyword works with the LEFT and RIGHT joins to specify the data field for the join operation.



WARNING

Using joins the wrong way can cause severe performance issues on your MySQL server, especially when working with large amounts of data (joining all the data records in tables with millions of data records can take quite a long time). I strongly suggest testing out your SELECT statements first before coding them into your web application. That will help give you a feel for any performance issues that may occur. In some situations, it's better to submit multiple smaller SELECT statements than to submit a single complex SELECT statement.

Using aliases

Having to specify the table and data field names in SELECT statements can get somewhat cumbersome. To help out, you can use the *table alias* feature, which defines a name that represents the full table name within the SELECT statement. Here's the format for using aliases:

```
SELECT fieldlist FROM table AS alias
```

When you define an alias for a table, you can use the alias anywhere within the SELECT statement to reference the full table name. This is especially handy in the long WHERE clauses when you're working with multiple tables:

```
MariaDB [dbtest1]> SELECT t1.orderid, t2.name, t2.address
    -> FROM orders as t1, customers as t2
    -> WHERE t1.orderid = 1000 AND t1.customerid = t2.customerid;
+-----+-----+
| orderid | name      | address          |
+-----+-----+
| 1000   | Acme Paper | 134 Main St.; Miami, FL |
+-----+
1 row in set (0.00 sec)

MariaDB [dbtest1]>
```

The t1 alias represents the Orders table, and the t2 alias represents the Customers table. Notice that you can use the aliases anywhere in the SELECT statement, even in the data field list!

Playing It Safe with Data

You've worked hard managing the data contained in the database (or at least your application has!). It would be a tragedy if something happened that corrupted the database so that you couldn't access that data. You never know when a catastrophic event will occur in the computer world, so it's always a good idea to have a duplicate copy of your data handy at all times.

The MySQL server provides a few different methods for backing up and restoring database data. This section walks through how to back up and restore database data in the MySQL server environments.

Performing data backups

When backing up a MySQL database server, you have a few different options available:

- » Copy the physical files the MySQL server uses to store data and database information.
- » Use MySQL utilities to extract database and table structure information.
- » Use MySQL utilities to extract table data.
- » Use MySQL utilities to extract both the table structure and data.

If you choose to copy the physical file structure of the MySQL server, you'll need to be careful. MySQL uses file locking to protect data as the server is running, so you may not be able to copy all the files required for the server operation at any given time.

Before you try to manually copy the MySQL server files, it's best to stop the MySQL server process from running to ensure all the data files are available and that you can safely copy them. This is called a *cold backup*.

In a cold backup, because you've stopped the MySQL server, web applications can't access the application data, so your website users won't be able to properly interact with your application. If your application has certain downtimes where website visitors won't use it (such as outside of business hours), this is fine, but for most web applications, your website visitors need access 24 hours a day, seven days a week! In those situations a cold backup just won't work.

The alternative is to perform a *hot backup*, which copies database information while the MySQL server is running and the web applications are still in use. Because the server is still running, the backup process can't lock the data tables, so the MySQL server can still process SQL statements, altering the data contained in the databases.

Because of this, the hot backup methods can't copy any of the files associated with the server operations. Instead, all they can do is take snapshots of the data contained within the database at specific moments in time. This type of backup is called a *data export*.

In a data export hot backup, the backup program exports the table structure and any data contained in the table into a text file that you can then copy to a safe location. The text file formats can differ, from placing data in a comma-separated spreadsheet format, to generating SQL statements that you can feed into the MySQL server to re-create the tables.

Each of the MySQL interfaces that you've been working with support data export hot backups. The following sections describe how to use these options within each of the interfaces.

From the command-line interface

The `mysqldump` command-line utility allows you to quickly and easily export a table structure and data from the command line. The `mysqldump` program is usually included with the other binary programs in MySQL, and you should find it in the same folder as the other MySQL command-line utility files (for XAMPP, that's the `c:\xampp\mysql\bin` folder in Windows, or `/Applications/XAMPP/mysql/bin` in macOS).

Here's the format for the `mysqldump` utility:

```
mysqldump [options] database [tablelist]
```

There are lots of options available for you to customize just how to perform the export. Here are some of the more common ones you may run into:

- » `--add-drop-database`: Add a `DROP DATABASE` statement in the output to replace any existing databases with the same name.
- » `--add-drop-table`: Add a `DROP TABLE` statement in the output to replace any existing tables with the same name.
- » `--all-databases`: Backup all the tables from all the databases on the server.
- » `--databases`: List multiple databases to export.
- » `--lock-tables`: Lock the tables during the export.
- » `--password`: Specify the user password, or if empty, prompt for a password.
- » `--tab`: Produce a tab-separated output for the data instead of SQL statements.
- » `--user`: Specify the user account to log into the MySQL server for the export.

Follow these steps to back up the `dbtest1` database tables using the `mysqldump` utility:

1. Open a command line in Windows or a Terminal session in Linux or macOS.
2. Change to the MySQL folder that contains the MySQL utilities for your installation environment.

For XAMPP on Windows, that's:

```
cd \xampp\mysql\bin
```

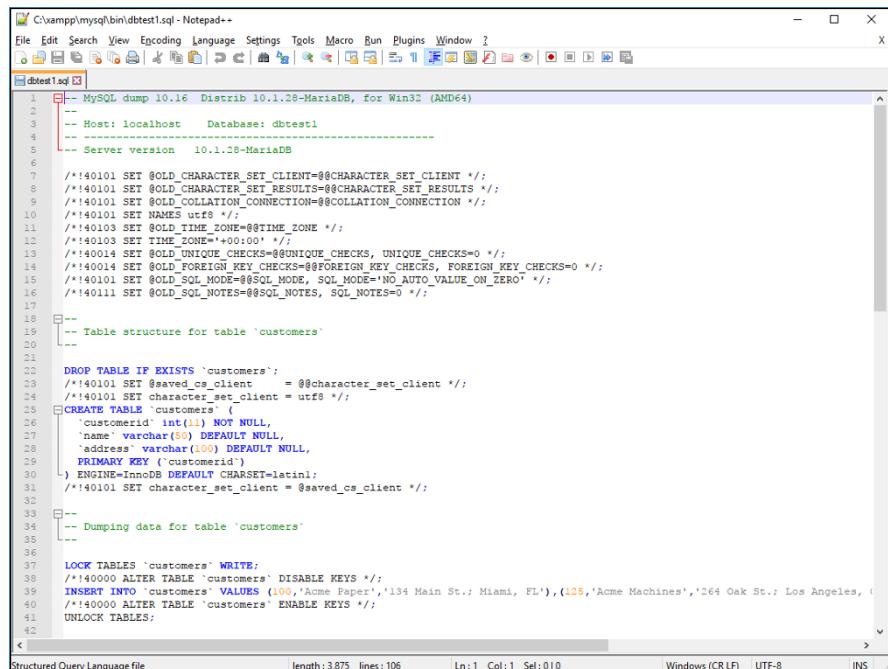
3. Run the mysqldump utility to export the table data from the dbtest1 database.

By default, the mysqldump utility will output the database contents to the screen. To save it to a file, you must redirect the output to a file. Enter this command:

```
C:\xampp\mysql\bin>mysqldump --user=root --password dbtest1 > dbtest1.sql  
Enter password:  
  
C:\xampp\mysql\bin>
```

4. View the generated dbtest1.sql file using your favorite text editor.

Figure 4-7 shows the results from my database.



The screenshot shows a Notepad window titled 'dbtest1.sql'. The content of the file is a MySQL dump script. It starts with the MySQL dump header, followed by various SET statements to restore session variables to their original values. Then, it defines the 'customers' table with its columns and primary key. Finally, it contains an INSERT statement to add a single record into the 'customers' table. The Notepad window has standard Windows-style icons at the top and a status bar at the bottom indicating the file type is 'Structured Query Language file'.

```
-- MySQL dump 10.16 Distrib 10.1.28-MariaDB, for Win32 (AMD64)  
--  
-- Host: localhost Database: dbtest1  
--  
-- Server version 10.1.28-MariaDB  
  
1 /*!40101 SET @OLD_CHARACTER_SET_CLIENT=@@CHARACTER_SET_CLIENT */;  
2 /*!40101 SET @OLD_CHARACTER_SET_RESULTS=@@CHARACTER_SET_RESULTS */;  
3 /*!40101 SET @OLD_COLLATION_CONNECTION=@@COLLATION_CONNECTION */;  
4 /*!40101 SET NAMES utf8 */;  
5 /*!40103 SET @OLD_TIME_ZONE=@TIME_ZONE */;  
6 /*!40103 SET TIME_ZONE='+00:00' */;  
7 /*!40014 SET @OLD_UNIQUE_CHECKS=@UNIQUE_CHECKS, UNIQUE_CHECKS=0 */;  
8 /*!40014 SET @OLD_FOREIGN_KEY_CHECKS=@FOREIGN_KEY_CHECKS, FOREIGN_KEY_CHECKS=0 */;  
9 /*!40101 SET @OLD_SQL_MODE=@SQL_MODE, SQL_MODE='NO_AUTO_VALUE_ON_ZERO' */;  
10 /*!40111 SET @OLD_SQL_NOTES=@SQL_NOTES, SQL_NOTES=0 */;  
  
--  
-- Table structure for table 'customers'  
--  
11 DROP TABLE IF EXISTS `customers`;  
12 /*!40101 SET @saved_cs_client      = @@character_set_client */;  
13 /*!40101 SET character_set_client = utf8 */;  
14 CREATE TABLE `customers` (  
15   `customerid` int(11) NOT NULL,  
16   `name` varchar(50) DEFAULT NULL,  
17   `address` varchar(100) DEFAULT NULL,  
18   PRIMARY KEY (`customerid`),  
19   ENGINE=InnoDB DEFAULT CHARSET=latin1;  
20 /*!40101 SET character_set_client = @saved_cs_client */;  
  
--  
-- Dumping data for table 'customers'  
--  
21 LOCK TABLES `customers` WRITE;  
22 /*!40000 ALTER TABLE `customers` DISABLE KEYS */;  
23 INSERT INTO `customers` VALUES (100,'Acme Paper','134 Main St.; Miami, FL'),(123,'Acme Machines','164 Oak St.; Los Angeles, CA');  
24 /*!40000 ALTER TABLE `customers` ENABLE KEYS */;  
25 UNLOCK TABLES;  
26  
27  
28  
29  
30  
31  
32  
33  
34  
35  
36  
37  
38  
39  
40  
41
```

FIGURE 4-7:
The output from
the mysqldump
utility.

As you peruse through the dbtest1.sql file that the mysqldump utility generated, you'll probably recognize the SQL statements that it uses. For each table in the database, it generates a CREATE TABLE statement to rebuild the table structure; then it generates an INSERT statement to add each data record from the original table.

Using Workbench

The MySQL Workbench graphical program provides a nice form for you to use to pick out the `mysqldump` options for the export. Follow these steps to generate an export file using Workbench:

- 1. Ensure that the MySQL server is running, and then start the Workbench tool.**
- 2. Click the Data Export link from the Management section in the Navigator window pane.**

Figure 4-8 shows what the Data Export interface looks like.

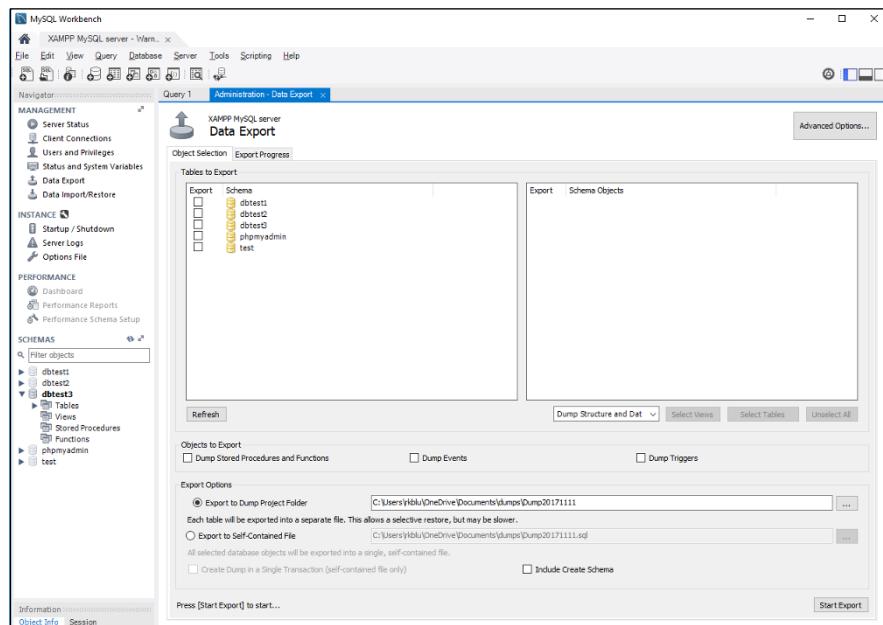


FIGURE 4-8:
The Workbench
Data Export
window.

- 3. Single-click the dbtest2 database entry in the left-hand window of the Tables to Export section of the main window.**

The tables contained in the dbtest2 database appear in the right-hand side window.

- 4. Select the check box for the dbtest2 database in the left-hand window.**

This automatically selects the check boxes for the tables it contains.

5. Under the right-hand side window, ensure that the drop-down box has the **Dump Structure and Data** option selected.
6. In the **Export Options** section, select the **Export to Self-Contained File** radio button and specify the location and name of the **.sql** file that will contain the export.

The default will create a file in your Documents folder under the dump folder.

Alternatively, you can opt to save the export as a project, which generates multiple files for each table. This allows you some more flexibility when restoring the data, but it's more difficult to manage the exported files.

7. Click the **Advanced Options** button at the top of the window.

A complete list of options for customizing the export appears, as shown in Figure 4-9.

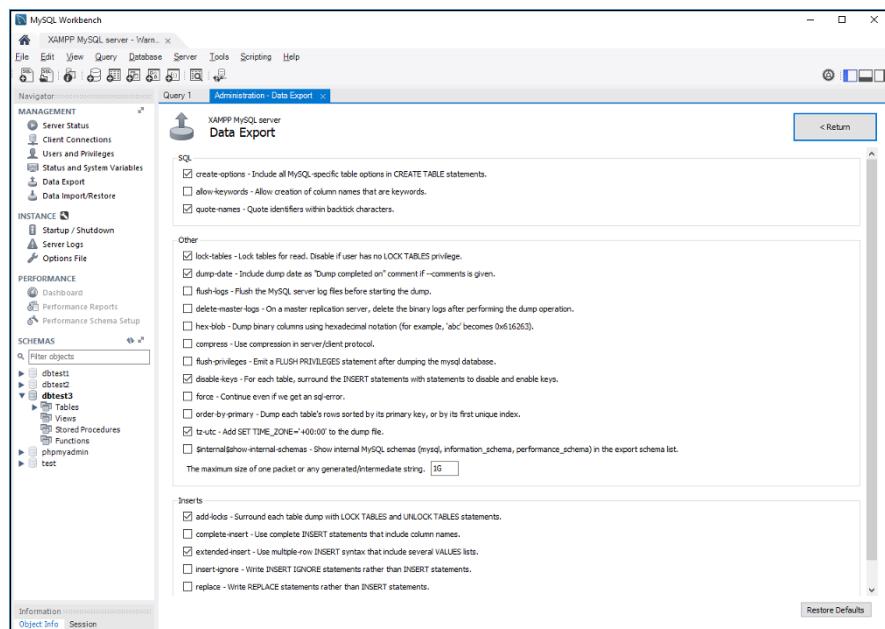


FIGURE 4-9:
The Workbench
Data Export
advanced options
window.

8. Click the **Return** button to return to the main Data Export interface window.
9. Click the **Start Export** button at the bottom of the window.

A dialog box appears, prompting you for the root user account password.

10. For XAMPP, leave it empty and click the OK button.

The Export Progress window appears, showing the progress of the export.

11. Use your favorite text editor to view the .sql file that was generated by the export.

Using a graphical interface certainly makes the data export process much simpler!

Using phpMyAdmin

The phpMyAdmin tool has an excellent graphical interface for handling data exports. After you open the phpMyAdmin tool, click the Export button at the top of the main web page. This produces the interface shown in Figure 4-10.

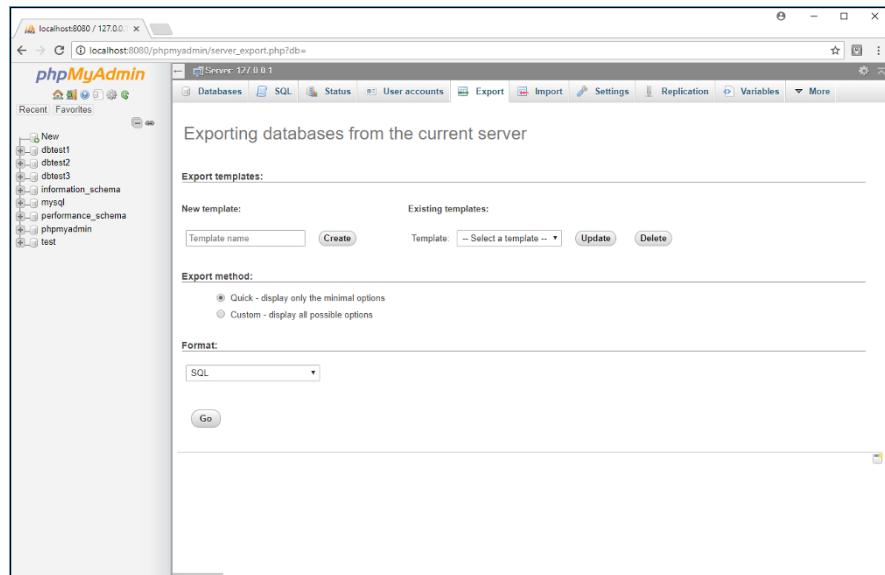


FIGURE 4-10:
The phpMyAdmin export web page.

The main export page allows you to choose from two options:

- » A quick export, which exports all the tables from all the databases using the mysqldump default options.
- » A custom export, which allows you to pick and choose the databases and options for the export.

One nice feature about the phpMyAdmin export interface is that it allows you to select the format of the export file from a long list of options, shown in Table 4-1.

TABLE 4-1

The phpMyAdmin Export Formats

Format	Description
CodeGen	The NHibernate file format
CSV	The comma-separated values format
CSV for Microsoft Excel	The CSV format with customizations for Microsoft Excel
Microsoft Word 2000	The Microsoft 2000 Word document
JSON	The JavaScript Object Notation format
LaTeX	The Lamport TeX format commonly used for academic publications
MediaWiki Table	The Wikipedia table format
OpenDocument Spreadsheet	The open spreadsheet standard format
OpenDocument Text	The open document standard format
PDF	The Adobe Portable Document Format
PHP array	PHP code to create an array of the data
SQL	SQL statements to rebuild the table
Texy!	XHTML formatted data
YAML	A data serialization format that is human-readable

That's a lot of different ways to export your database data!

If you select the Custom export method, you can select the databases to export, the output method (and file type if you save it to a file), the format of the output, and any MySQL directives (such as to add the `DROP DATABASE` or `DROP TABLE` statements). This gives you maximum flexibility when creating your database backups!

Restoring your data

Backups are only good if you have the ability to use them to restore the database. Testing out the restore capabilities of your system before you have a catastrophic event is always a good idea.

Each of the MySQL interface methods provides a different way of restoring data from the backup files. This section walks through each of these methods.

From the command-line interface

To restore a database using the SQL dump file generated by the `mysqldump` utility, just pass the file into the input of the `mysql` command-line tool using the command line redirect symbol (`<`). That looks like this:

```
mysql --user=root --password dbtest1 < dbtest1.sql
```

The MySQL server will process the SQL statements contained in the `dbtest1.sql` file and apply them against the database you specify on the command line. This is a great way to move a database to a new database, either on the same server or on a remote server!

If you opt to save only the table data using either the tab or comma-separated formats, you can read the data into a table by using the `LOAD DATA INFILE SQL` statement:

```
LOAD DATA INFILE filename INTO TABLE table
```

The data fields in the file must match the order in which they appear in the table.

From Workbench

The MySQL Workbench tool provides a graphical interface for loading the backup file. After you open Workbench, click the Data Import/Restore link in the Management section of the Navigator window pane. Figure 4-11 shows what that window looks like.

In the Import Options section, select either the project folder or the export file that you created with the Export feature. Select the database to use for the import in the Default Target Schema drop-down box.

If you opted to save the backup as a project, you can customize the restore by selecting exactly which objects to restore. If you opted to save the backup as a single file as in the example, you must restore all the objects in the export file.

After you've selected the export file and options, click the Start Import button at the bottom of the window to begin the import process. That makes restoring table data almost simple!

From phpMyAdmin

Importing data from an export backup using phpMyAdmin is also a fairly simple process. After you open the phpMyAdmin web tool, click the Import button at the top of the web page. This produces the Import Web page, as shown in Figure 4-12.

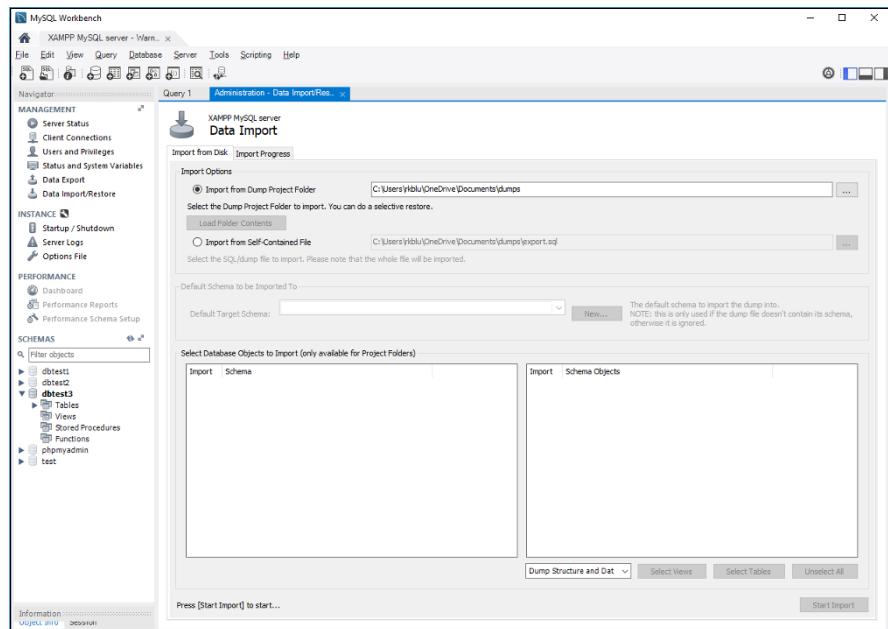


FIGURE 4-11:
The Workbench
Data Import/
Restore window.

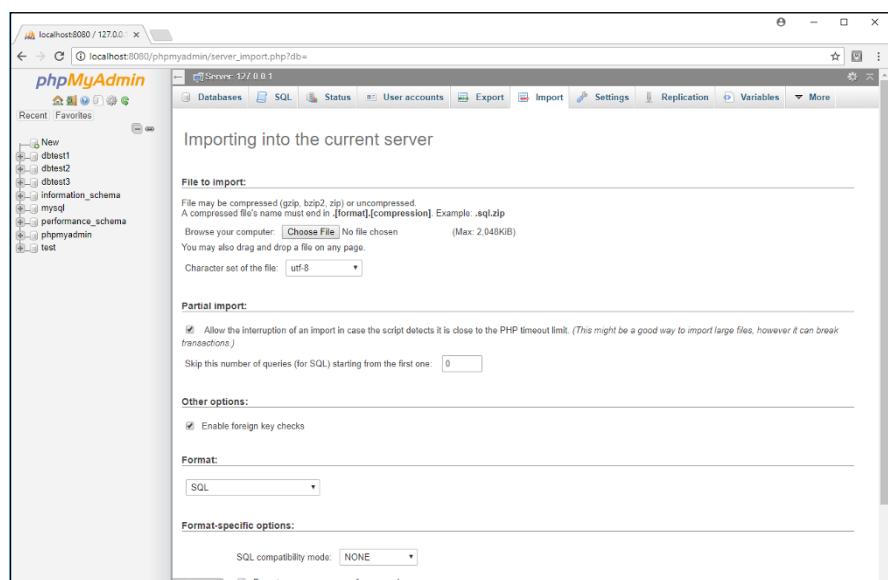


FIGURE 4-12:
The phpMyAdmin
Import web page.

From here, you can browse to find the export file that you generated, along with selecting some options for the import, such as the file format. After you've made your selections, click the Go button at the bottom of the web page to start the import.

IN THIS CHAPTER

- » Examining the PHP database libraries
- » Connecting to the MySQL server
- » Submitting SQL queries
- » Retrieving result set data
- » Exploring a PHP database application

Chapter **5**

Communicating with the Database from PHP Scripts

In the previous chapter, I show you how to insert, delete, and manage data in a MySQL database. Now that you have your content all ready for your application, there's just one more piece to add in the assembly line to complete your dynamic web applications. This chapter explores how you can interact with the MySQL database server from your PHP programs to retrieve the stored data, add new data records, and remove existing data records. This chapter first explores how PHP interacts with databases in general. Then it focuses on the most popular method used for accessing MySQL databases from web applications: the `mysqli` library.

Database Support in PHP

The PHP programming language doesn't have any functions for accessing databases directly built into the language. However, there are plenty of PHP extension libraries available to help out. The PHP extensions provide additional

functionality to the main PHP language by incorporating add-on libraries (see Book 4, Chapter 1).

PHP has a long history of providing library support for accessing different types of databases, making it a popular programming language to use with lots of different database servers. Table 5-1 lists the database server libraries currently available to use with your PHP code.

TABLE 5-1

PHP Database Extension Libraries

Library	Description
CUBRID	An open-source relational database with object extensions
DB++	A non-SQL-based relational database created by Concept asa
dBase	An old proprietary database file format used mostly for microcomputers
FireBird/InterBase	A relational database based on the ISO SQL-2003 standard
IBM DB2	A proprietary IBM relational database format
Informix	An old relational database format acquired by IBM in 2001
Ingres	An open-source relational database designed for large applications
MaxDB	An ANSI SQL-92-compliant relational database used by the SAP software
Mongo	An open-source document-oriented database
mSQL	A lightweight SQL-based database created by Hughes Technologies
MySQL	The open-source MySQL database server
OCI8	The Oracle database server
PostgreSQL	An open-source database based on the original Ingres database
SQLite	An embeddable database system for small environments
SQLite3	An update to the SQLite database system
SQLSRV	The Microsoft SQL database server
tokyo_tyrrant	An open-source distributed database system

In addition to the specific database extensions available in PHP, there are also three abstract database interfaces available:

- » **DBA:** A library for accessing Berkeley DB-style database files
- » **ODBC:** A library for interfacing with the Open Database Connectivity (ODBC) standard, originally developed by Microsoft and popular on Windows platforms
- » **PDO:** The PHP Data Objects (PDO) library, which uses a lightweight interface to access different types of databases, based on an installed driver

The benefit of using an abstract database interface in your PHP code is that you can easily change the underlying database your application uses without having to change any of your PHP code. If you use a specific database extension, such as for MySQL, and you need to run your application using a different database server, you need to recode all your database interactions. However, with the abstract database extension you can keep your code and just change the underlying database driver. If you're developing applications for use in multiple environments, this is definitely the way to go.

Because the MySQL database server is the focus of this book, this chapter shows you how to code PHP to interact with a MySQL database using the MySQL-specific database interface. If you decide to use another database server for your applications, you'll be able to apply the same techniques you learn here to the other PHP database libraries.

Using the mysqli Library

The original MySQL database extension library created for PHP was called *php_mysql*. It had some limitations, but it was widely popular in developing dynamic web applications across the Internet that used the MySQL database.

As time went on, developers worked on improving the limitations of the *php_mysql* library. Eventually, enough changes accumulated to warrant a new library package release. That release was named the *php_mysqli* library (the added *i* stands for “improved”). Starting in PHP version 7, the original *php_mysql* library has been removed, so it's not recommended that any new application development use it. This section walks through how to use the *php_mysqli* library to interface your PHP programs to a MySQL database server.

One of the added features of the *php_mysqli* library is that it supports both procedural language and object-oriented language coding styles. That means you can use the library both as functions in procedural programs and as an object in object-oriented PHP programs.

For the procedural program environment, you just run separate functions for each action. Most of the functions require that you pass the database connection handle along as a parameter. For the object-oriented program environment, you must instantiate a new `mysqli` object, and then run methods using that object. For each action, you need to interact with the MySQL database. In the following sections, I show you how to use both coding methods.

Connecting to the database

The first thing you need to do from your PHP code is to establish a connection to the MySQL database server. The connection creates a database session, from which you can submit queries and retrieve result sets.

In the procedural coding method, you establish the connection using the `mysqli_connect()` function:

```
$con = mysqli_connect(host, user, password, database, port, socket);
```

That's a lot of information to pass for the connection! Fortunately, for most situations, you don't need to include the port or socket values, because those are standard. As you can probably tell from the parameter names, you must specify the host name or IP address of the MySQL server, a userid to log into the server, along with its associated password. The `database` parameter allows you to specify a default database for the session. A typical connection statement would look like this:

```
$con = mysqli_connect("localhost", "user1", "myL0ngP@ssword", "dbtest1");
```

If the connection is successful, the `$con` variable contains what's called the *connection handle*. You must use it to reference the connection session in some of the other `php_mysqli` library functions.

If you need to change the default database used in the connection, use the `mysqli_select_db()` function:

```
mysqli_select_db(handle, database)
```

As you can tell from the function format, for the first parameter you need the connection handle from the original database connection:

```
mysqli_select_db($con, "dbtest2");
```

In object-oriented coding, instead of using the `mysqli_connect()` function, you must instantiate a new instance of the `mysqli` object:

```
$db = new mysqli("localhost", "user1", "myL0ngP@ssword", "dbtest1");
```

The parameters available are the same as for the `mysqli_connect()` function. However, this time the `$db` variable contains an instance of the `mysqli` object instead of a connection handle.

To select a new default database for the session, you run the `select_db()` method from the `$db` object you created:

```
$db->select_db("dbtest2");
```



TIP

Notice that the names of the procedural functions and object-oriented methods are similar. That holds true for the rest of the functions and methods you'll use in the `php_mysqli` library, so it's relatively easy to move back and forth between the two coding methods.



WARNING

In both the procedural and object-oriented coding methods, you must enter the user ID and password to connect to the database. Unfortunately, you must do that in plain text. This is a bit of a security risk. If anyone has access to your source code, they could gain access to the database. First, make sure the user ID configured on the MySQL server is locked down to a specific host location (don't use a wildcard character for the location). The next step is to move the connection statement to an out-of-the-way place, such as in a separate include file; then use the PHP `include()` function to add it to your programs. That at least limits the visibility of the user ID and password information to a single file. If your server environment permits, you may even be able to hide the include file outside of the DocumentRoot folder of your web server.

Closing the connection

When you open a connection to the MySQL server, the connection remains open for the duration of that program. When the PHP server reaches the end of your PHP code for that web page, it automatically closes the connection.

The MySQL server has a limited number of client connections that it supports (defined by the MySQL configuration file). If you're coding in a high-volume environment that has lots of customers accessing your web application at the same time, it may be crucial that your application releases the MySQL server connection as soon as possible.

If you're working in that type of environment, you can manually close the MySQL server connection as soon as you're done using it to help free up more connections as soon as possible. You do that by using the `mysqli_close()` function. Just specify the connection handle as the sole parameter:

```
mysqli_close($con);
```

If you're working with object-oriented programming code, just use the `close()` method on the `mysqli` instance:

```
$db->close();
```

Again, for most normal situations you don't need to worry about closing the MySQL server connection, it's just nice to know you have that option available if you need it!

Submitting queries

After you establish a connection to the MySQL server, you can start submitting SQL statements. You can submit any type of statement, just as if you were working from the MySQL command-line interface.

The procedural function you use to submit a query to the MySQL server is the `mysqli_query()` function. Here's the format:

```
$conresult = mysqli_query(handle, query);
```

The `handle` parameter is the connection handle created when you connected to the MySQL server. The `query` parameter is the text SQL statement. Unless you're submitting a very short SQL statement, it has become common practice to store the SQL statement in a variable so as not to make the `mysqli_query()` statement overly complicated:

```
$query = "SELECT * FROM employees";
$dbresult = mysqli_query($con, $query);
```

The `mysqli_query()` function returns what's called a *resource handle*. You must store that in a variable to be able to access the data in the result set returned by the query.

In the object-oriented coding world, you use the `query()` method of the `mysqli` connection instance to submit SQL statements:

```
$query = "SELECT * FROM employees";
$result = $db->query($query);
```

The `query()` method returns a result set object, which you must store in a variable to be able to access the data contained in the result set.



TIP

The SQL statement that you send to the MySQL doesn't need a semicolon at the end. The semicolon is used by the MySQL command-line interface to indicate the end of the statement. The `mysqli_query()` function already knows the end of the SQL statement because it will only accept one query per text string. That also helps block SQL-injection types of attacks against your system.

Retrieving data

If you submit a SQL statement that will return data (such as a `SELECT` statement), the result set handle returned will point to the data. You can then retrieve the data using other `mysqli` library functions. There are actually quite a few different ways to retrieve that data using different functions or methods. Table 5-2 lists the different functions and methods available.

TABLE 5-2 The `mysqli` Data Retrieval Functions and Methods

Function	Method	Description
<code>mysqli_fetch_all</code>	<code>fetch_all()</code>	Retrieves all the data records in the result set as an array
<code>mysqli_fetch_array</code>	<code>fetch_array()</code>	Retrieves the current data record in the result set as an array
<code>mysqli_fetch_assoc</code>	<code>fetch_assoc()</code>	Retrieves the current data record in the result set as an associative array
<code>mysqli_fetch_field_direct()</code>	<code>fetch_field_direct()</code>	Retrieves the metadata for a specific field in a result set
<code>mysqli_fetch_field()</code>	<code>fetch_field()</code>	Retrieves the metadata for a single field in a result set
<code>mysqli_fetch_fields()</code>	<code>fetch_fields()</code>	Returns the metadata for all the fields in a result set as an array
<code>mysqli_fetch_object()</code>	<code>fetch_object()</code>	Retrieves the current data record of a result set as an object
<code>mysqli_fetch_row()</code>	<code>fetch_row()</code>	Retrieves the current data record of a result set as a numeric array
<code>mysqli_field_seek()</code>	<code>field_seek()</code>	Sets the result set pointer to a specific field in the current data record
<code>mysqli_free()</code>	<code>free()</code>	Releases the memory associated with the result set handle

The `mysqli_fetch` family of statements retrieves the data records from the result set and places them in an array variable. The type of array variable used depends on the function you use:

- » The `mysqli_fetch_array()` function creates an associative array, a numeric array, or both, based on the second parameter (`MYSQLI_ASSOC`, `MYSQLI_NUM`, or `MYSQLI_BOTH`).
- » The `mysqli_fetch_assoc()` function creates an associative array, using the data field names as the array keys.
- » The `mysqli_fetch_row()` function creates a numeric array, using numeric indexes for each data field (starting at 0, and using the data field order specified in the table or `SELECT` statement data field).

The fetch statements are also somewhat unique in that they allow you to walk your way through the result set one data record at a time. Each time you call the fetch statement, it returns the data from the current data record in the result set; then it moves a pointer to the next data record in the result set for the next call. When it reaches the end of the result set data, it returns a `NULL` value, making it ideal to use the fetch statements in a `while()` loop:

```
$query = "SELECT * FROM employees";
$conresult = mysqli_query($con, $query);
while($row = mysqli_fetch_assoc($conresult)) {
    echo "<p>Employee last name: $row['lastname']<br>\n";
    echo "Employee first name: $row['firstname']<br>\n";
    echo "Start date: $row['startdate']<br>\n";
    echo "Salary: $row['salary']</p>\n";
}
```

Or if you're using the object-oriented programming style:

```
$query = "SELECT * FROM employees";
$dbresult = $db->query($query);
while( $row = $dbresult->fetch_assoc()) {
    echo "<p>Employee last name: $row['lastname']<br>\n";
    echo "Employee first name: $row['firstname']<br>\n";
    echo "Start date: $row['startdate']<br>\n";
    echo "Salary: $row['salary']</p>\n";
}
```

The `mysqli_fetch_assoc()` function returns the data record as an associative array value, which you then store in an array variable. The while loop continues

until there are no more data records in the result set; then it drops out so the program can continue.

Being prepared

You can submit any type of SQL statements using the `mysqli_query()` function, but it's not recommended to use that for submitting INSERT statements. All too often, beginning PHP coders retrieve data from an HTML form, place the data directly in an INSERT statement string, then submit the string to the MySQL server, like this:

```
$empid = $_POST['employeeid'];
$lname = $_POST['lastname'];
$fname = $_POST['firstname'];
$start = $_POST['startdate'];
$birth = $_POST['birthdate'];
$salary = $_POST['salary'];
$query = "INSERT INTO employees VALUES ($empid, '$lname', '$fname',
    '$start', '$birth', $salary)";
$dbresult = $db->query($query);
```

This method works, but it's a dangerous way of inserting data into your database! There's no guarantee that the person using the HTML form will enter the correct data into all the data fields (either by accident or on purpose). There's also no guarantee the data submitted in the form won't contain malicious characters meant to cause issues with the database.

The safer way of submitting data in an INSERT statement is to use a *prepared statement*, which defines a template of the query you want to execute on the MySQL server, and then sends the data separate from the template. The MySQL server stores the prepared statement, and then matches the submitted data against the template. This helps filter out malicious data. Plus, it can help speed up executing multiple INSERT statements on the server. You just submit one template statement. Then you can apply multiple data statements against the same template.

With a prepared statement, you create the query string as normal, but instead of including data values, you use a question mark as a placeholder for each value, like this:

```
$query = "INSERT INTO employees VALUES (?, ?, ?, ?, ?, ?, ?)";
```

Then you use the `mysqli_prepare()` function to submit it:

```
$constmt = mysqli_prepare($con, $query);
```

If you're using the object-oriented coding style, it looks like this:

```
$dbstmt = $db->prepare($query);
```

Now the MySQL server has the prepared statement, but it doesn't have any data to plug into it. To do that you use the `mysqli_stmt_bind_param()` function:

```
mysqli_stmt_bind_param($constmt, "issssi", $empid, $lname, $fname, $start,  
$birth, $salary);
```

The first parameter is the result from the prepared statement you submitted. The second parameter is somewhat odd. It defines the data type of each of the data values as a single character in a string value:

- » b: A blob data type value
- » i: An integer data type value
- » d: A double data type value
- » s: A string data type value

In this example, I define the first data value as an integer, the next four as string values, and the final data value as an integer.

After defining the data types, you just list the values for the data fields, in the order they appear in the prepared statement.

If you're following along in the object-oriented code style, here's how to bind the parameters:

```
$dbstmt->bind_param("issssi", $empid, $lname, $fname, $start, $birth, $salary);
```

After you bind the data values to the prepared statement, there's still one more step — you must execute the prepared statement:

```
mysql_bind_execute($constmt);
```

Or if you're using the object-oriented coding style, it looks like this:

```
$dbstmt->execute();
```

This is what links the data to the template in the MySQL server and processes the statement.

Using prepared statements for `INSERT` statements is crucial, but you can also use them for submitting `SELECT` statements if you don't trust the data that you're using in the `WHERE` clause.

This can be the result of allowing your website visitors to enter text to search for in the database. For example, instead of writing this code:

```
$empid = $_POST['employeeid'];
$query = "SELECT * FROM employees WHERE employeeid = $empid";
$result = $db->query($query);
$row = $db->fetch_assoc($result);
```

You can use a prepared statement and write it this way:

```
$empid = $_POST['employeeid'];
$query = "SELECT * FROM employees WHERE empid = ?";
$dbstmt = $db->prepare($query);
$dbstmt->bind_param("i", $empid);
$dbstmt->execute();
$dbstmt->bind_result($empid, $lname, $fname, $start, $birth, $salary);
$dbstmt->fetch();
```

When you submit the `SELECT` statement using a prepared statement, you must bind the PHP variables for the result set data fields using the `bind_result()` method (or `mysqli_stmt_bind_result()` for procedural style coding). Then you can retrieve the data records in the result set using the `fetch()` method (or `mysqli_stmt_fetch()` for procedural style coding).

Checking for errors

Whenever you submit any type of action to the MySQL server, it's always a good idea to ensure that it completed properly before continuing on with your program. There are a couple of different ways to do that.

One method is to test the connection handle or result handle for a `NUL` value. If the connection or query fails, the handle will be set to `NUL`:

```
$con = mysqli_connect("localhost", "user1", "MyL0ngP@ssword", "dbtest1");
if (!$con) {
    echo "Sorry, there was a problem connecting";
    exit;
}
```

Or for procedural-style coding:

```
$db = new mysqli("localhost", "user1", "MyL0ngP@ssword", "dbtest1");
if (!$db) {
    echo "Sorry, there was a problem connecting";
    exit;
}
```

This allows you to halt the program immediately without trying to submit any further statements. The downside is that if the connection fails, the connection statement will generate an ugly PHP error message. If you'd like to suppress the error message, you can precede the code with an ampersand (@), like this:

```
@ $con = mysqli_connect("localhost", "user1", "MyL0ngP@ssword", "dbtest1");
```

Or:

```
@ $db = new mysqli("localhost", "user1", "MyL0ngP@ssword", "dbtest1");
```

The leading ampersand tells PHP to suppress any error output that may be generated from the statement.

The other way of stopping things when the connection fails is to use the PHP `die()` function:

```
@ $con = mysqli_connect("localhost", "user1", "MyL0ngP@ssword", "dbtest1") or
die("Sorry, something went wrong with the connection");
```

The logical OR statement will only trigger if the connection statement fails, running the `die()` function, which displays the string in the web page output.

Sometimes when you submit queries that fail, it helps to be able to see the exact error message the query generated. You can do that using the `mysqli_error()` function:

```
$conresult = mysqli_query($con, $query)
if (!$conresult) {
    echo mysqli_error($con);
}
```

Or for object-oriented programming:

```
$dbresult = $db->query($query);
if (!$dbresult) {
    echo $db->error();
}
```

The `mysqli_error()` function returns the detailed error message generated by the MySQL server when a submitted SQL statement fails. Often that helps shed some additional light on just what went wrong, and can be invaluable when troubleshooting your SQL code!



WARNING

The `mysqli_error()` function is useful for troubleshooting code, but it's also useful for attackers to gain inside information on your database structure. After you're done developing the application code, it's best to remove any `mysqli_error()` functions you have in the code to prevent an attacker from exploiting them.

Miscellaneous functions

Submitting queries to the MySQL server and retrieving the data records from the result set consists of the bulk of your database requirements, but there are a few more handy functions available in the `php_mysqli` library that can be useful in your application coding. Table 5-3 lists some of the more common ones that you may want to use.

TABLE 5-3

Additional mysqli Library Functions

Method	Description
<code>autocommit</code>	Turn on or off the autocommit feature in MySQL, which allows you to submit multi-statement transactions
<code>change_user</code>	Changes the user account for the session
<code>character_set_name</code>	Returns the character set used for the connection
<code>commit</code>	Commits a transaction
<code>more_results</code>	Checks if there are more query results from a multi-query submission
<code>multi_query</code>	Allows you to submit more than one query at a time
<code>next_result</code>	Prepares the next data record result from a multi-query
<code>real_escape_string</code>	Escapes special characters in a string to make them safe to use in an SQL query
<code>rollback</code>	Rolls back the current transaction
<code>set_charset</code>	Sets the default character set used for the session

As you can probably guess, the procedural style names for these methods are the same, just with the `mysqli_` prefix.

The `real_escape_string()` method is useful for cleaning up text input to use in an `INSERT` statement. It places a backslash in front of any character that will cause trouble when sent to the MySQL server. This is especially useful with data that may include single or double quotes.

Though not a method, the `num_rows` property is handy when you just need the number of data records returned in a result set. Because it's a property and not a method, you don't include the parenthesis after it:

```
$query = "SELECT * FROM employees";
$dbresult = $db->query($query);
if ($dbresult->num_rows > 0) {
    echo "There were $dbresult->num_rows data records in the table";
} else {
    echo "Sorry, there weren't any data records returned";
}
```

The procedural style is a little different in that it uses the `num_rows` as a function:

```
if (mysqli_num_rows($conresult) > 0)
```

The `mysqli_num_rows()` function returns the same information, but you just use it as a function instead of a property.

Putting It All Together

Now that you've seen the basics of using the `php_mysqli` library to interact with the MySQL server, let's walk through a short web application that does that. First, you need a database with some data. Follow these steps to create that:

- 1. Ensure that both the Apache and MySQL servers are running.**

If you're using XAMPP, start the Control Panel and click the Start buttons for both Apache and MySQL.

- 2. Open the MySQL command-line interface.**

For Windows, open a Command Prompt session and enter these commands:

```
cd \xampp\mysql\bin
mysql --user=root --password
```

For the macOS environment, open a Terminal session and enter these commands:

```
cd /Applications/XAMPP/mysql/bin  
./mysql --user=root --password
```

- 3. Press the Enter key at the MySQL password prompt.**
- 4. Enter the following statement to create the apptest1 database:**

```
MariaDB [(none)]> CREATE DATABASE apptest1;  
Query OK, 1 row affected (0.00 sec)
```

- 5. Enter the following statement to create the appuser1 user account:**

```
MariaDB [(none)]> CREATE USER appuser1@localhost IDENTIFIED BY  
"MyL0ngP@ssword";  
Query OK, 0 rows affected (0.07 sec)
```

- 6. Enter the following statement to grant privileges to the apptest1 database for the appuser1 user account:**

```
MariaDB [(none)]> GRANT SELECT,INSERT,UPDATE,DELETE ON apptest1.*  
-> TO appuser1@localhost;  
Query OK, 0 rows affected (0.00 sec)
```

- 7. Enter the USE statement to change the default database to the apptest1 database:**

```
MariaDB [(none)]> USE apptest1;  
Database changed
```

- 8. Enter the following statement to create the Bowlers table:**

```
MariaDB [apptest1]> CREATE TABLE bowlers  
-> (bowlerid int primary key,  
-> name varchar(100),  
-> address varchar(200),  
-> phone varchar(20));  
Query OK, 0 rows affected (0.28 sec)
```

- 9. Enter the following statement to create the Games table:**

```
MariaDB [apptest1]> CREATE TABLE games  
-> (gameid int auto_increment primary key,  
-> bowlerid int,  
-> score int);  
Query OK, 0 rows affected (0.22 sec)
```

10. Now you can start entering some data into your tables.

Here's some sample data for the Bowlers table:

```
MariaDB [apptest1]> INSERT INTO bowlers VALUES  
    -> (100, 'Rich', '123 Main St.', '555-1234');  
Query OK, 1 row affected (0.08 sec)  
  
MariaDB [apptest1]> INSERT INTO bowlers VALUES  
    -> (101, 'Barbara', '123 Main St.', '555-5678');  
Query OK, 1 row affected (0.10 sec)  
  
MariaDB [apptest1]> INSERT INTO bowlers VALUES  
    -> (102, 'Katie Jane', '567 Oak St.', '555-0123');  
Query OK, 1 row affected (0.10 sec)  
  
MariaDB [apptest1]> INSERT INTO bowlers VALUES  
    -> (103, 'Jessica', '901 Elm St.', '555-3256');  
Query OK, 1 row affected (0.09 sec)
```

11. Add some data for the Games table for the bowlers:

```
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (100, 110);  
Query OK, 1 row affected (0.10 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (100, 115);  
Query OK, 1 row affected (0.08 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (100, 105);  
Query OK, 1 row affected (0.05 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (101, 110);  
Query OK, 1 row affected (0.11 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (101, 112);  
Query OK, 1 row affected (0.06 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (101, 130);  
Query OK, 1 row affected (0.10 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (102, 115);  
Query OK, 1 row affected (0.11 sec)  
  
MariaDB [apptest1]> INSERT INTO games (bowlerid, score) VALUES (102, 125);  
Query OK, 1 row affected (0.09 sec)
```

```
MariaDB [appptest1]> INSERT INTO games (bowlerid, score) VALUES (102, 140);
Query OK, 1 row affected (0.08 sec)

MariaDB [appptest1]> INSERT INTO games (bowlerid, score) VALUES (103, 135);
Query OK, 1 row affected (0.08 sec)

MariaDB [appptest1]> INSERT INTO games (bowlerid, score) VALUES (103, 138);
Query OK, 1 row affected (0.09 sec)

MariaDB [appptest1]> INSERT INTO games (bowlerid, score) VALUES (103, 130);
Query OK, 1 row affected (0.08 sec)
```

12. Exit the MySQL command-line interface by entering the exit statement:

```
MariaDB [appptest1]> exit;
C:\xampp\mysql\bin>
```

13. Close the Command Prompt or Terminal session.

Now that you have some sample data, you can code an application to use the data! Follow these steps to create a simple application that reads the Bowlers table and then calculates the average score for each bowler on your team:

- 1. Open your favorite text editor, program editor, or integrated development environment (IDE) package.**
- 2. Enter the following code into the editor window:**

```
<!DOCTYPE html>
<html>
<head>
<title>PHP Test Web Page</title>
<style>
table, th, td {
    border: 1px solid black;
    border-collapse: collapse;
    text-align: center;
}
</style>
</head>
<body>
<h1>My Bowling Team</h1>
<CDATATable>
<tr><th>Bowler</th><th>Games Played</th><th>Average</th></tr>
<?php
$db = new mysqli("localhost", "appuser1", "MyL0ngP@ssword", "appitest1");
```

```

if (!$db) {
    echo "Sorry, I could not connect to the database server";
    exit;
}

$query = "SELECT bowlerid, name FROM bowlers ORDER BY name";
$result = $db->query($query);
while($row = $result->fetch_assoc()) {
    $bowlerid = $row['bowlerid'];
    $name = $row['name'];

    $query2 = "SELECT COUNT(score) AS games, AVG(score) AS average FROM
games" .
        " WHERE bowlerid = $bowlerid";
    $result2 = $db->query($query2);
    $row2 = $result2->fetch_assoc();
    $games = $row2['games'];
    $average = $row2['average'];

    echo "<tr><td>$name</td><td>$games</td><td>$average</td>\n";
}
?>
</CDATAtable>
</body>
</html>

```

- 3. Save the file as `phpapptest.php` in the DocumentRoot folder for your web server.**

If you're using XAMPP on Windows, that's `c:\xampp\htdocs`. For XAMPP on macOS, that's `/Applications/XAMPP/htdocs`.

- 4. Ensure that the Apache and MySQL servers are still running.**

- 5. Open your browser and enter the following URL:**

```
http://localhost:8080/phpapptest.php
```

You may need to modify the TCP port in the URL to match your web server.

- 6. Observe the output in the web page.**

- 7. Open the MySQL command-line interface again and add some new data records.**

Create a new bowler, along with some new games scores.

- 8. Refresh the browser window and observe the output.**
- 9. Stop the Apache and MySQL servers, close out the XAPP Control Panel, and close your browser when you're done.**

Figure 5-1 shows the results that you should have seen in your browser window.

The screenshot shows a Microsoft Edge browser window. The title bar reads "PHP Test Web Page". The address bar shows the URL "localhost:8080/phpapptest.php". The main content area has a heading "My Bowling Team" followed by a table with the following data:

Bowler	Games Played	Average
Barbara	3	117.3333
Jessica	3	134.3333
Katie Jane	3	126.6667
Rich	3	110.0000

FIGURE 5-1:
The output
from the
`phpapptest.php`
program.

The PHP code first submitted a SELECT statement to retrieve the bowlerid and name data field values for each bowler in the Bowlers table:

```
$query = "SELECT bowlerid, name FROM bowlers ORDER BY name";
```

Then the code iterates through that result set using the `fetch_assoc()` method:

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

For each iteration in the Bowlers table, the code submits another SELECT statement, using the MySQL COUNT() and AVG() built-in functions to determine the number of data records for the bowler, and the average of the score data field values:

```
$query2 = "SELECT COUNT(score) AS games, AVG(score) AS average FROM games" . "
WHERE bowlerid = $bowlerid";
```

Remember that the dot at the end of the statement is the string concatenation operator, so you can split this long SQL statement into two lines to make it easier to read.

Because this SELECT statement returns only one data record for each bowler, you don't need to iterate through the result set, because there's just one data record in the result set. The code uses the `fetch_assoc()` function to retrieve the data field values (notice that it uses the alias names assigned to the data fields in the SELECT statement) and uses the `echo` statement to output the table row HTML code with the embedded data. Fancy!