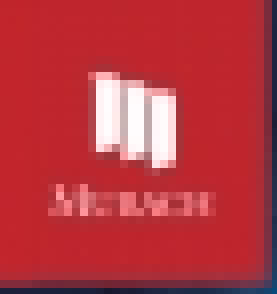


TRAINING & REFERENCE

murach's MySQL

3rd Edition

JOEL MURACH

MASTER THE SQL STATEMENTS
that every application developer needs for retrieving and updating the data in a MySQL database

DESIGN DATABASES LIKE A PRO
and code the SQL statements that create databases, tables, indexes, and more

GAIN PROFESSIONAL SKILLS
like using transactions, stored procedures, functions, triggers, and events

GET STARTED AS A DBA
by learning how to configure the server, manage security, and create backups

murach's MySQL

3rd Edition

Joel Murach

TRAINING & REFERENCE

murach's MySQL

3rd Edition

Joel Murach

Mike Murach & Associates, Inc.
4340 N. Knoll Ave. • Fresno, CA 93722
www.murach.com • murachbooks@murach.com

Editorial team

Author:	Joel Murach
Writer/Editor:	Anne Boehm
Editorial Support:	Steven Mannion
Production:	Samantha Walker

Books on general-purpose programming languages

Murach's Python Programming
Murach's Java Programming
Murach's C++ Programming
Murach's C#
Murach's Visual Basic

Books for web developers

Murach's HTML5 and CSS3
Murach's JavaScript and jQuery
Murach's PHP and MySQL
Murach's Java Servlets and JSP
Murach's ASP.NET Web Programming with C#

Books for database programmers

Murach's MySQL
Murach's SQL Server for Developers
Murach's Oracle SQL and PL/SQL for Developers

For more on Murach books, please visit us at www.murach.com

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1
ISBN: 978-1-943872-36-7

Content

Expanded contents

Chapter 3 How to retrieve data from a single table

Chapter 4 How to retrieve data from two or more tables

Introduction

Since its release in 2000, MySQL has become the world's most popular open-source database. It has been used by everyone from hobbyists to the world's largest companies to deliver cost-effective, high-performance, scalable database applications…the type of applications that the web is built on. In fact, MySQL has been used as the database for many high-profile websites, including Wikipedia, Facebook, and Twitter. So knowing MySQL is a plus for any developer today.

Who this book is for

This book is designed for developers who are new to MySQL, as well as developers who have been using MySQL for years but who still aren't getting the most from it. It shows how to code all the SQL statements that developers need for their applications, and it shows how to code these statements so they run efficiently.

This book is also a good choice for anyone who wants to learn standard SQL. Since SQL is a standard language for accessing database data, most of the SQL code in this book will work with any database management system. As a result, once you use this book to learn how to use SQL to work with a MySQL database, you can transfer most of what you have learned to another database management system, such as Oracle, SQL Server, or DB2.

This book is also the right *first* book for anyone who wants to become a database administrator. Although this book doesn't present all of the advanced skills that are needed by a DBA, it will get you started. Then, when you complete this book, you'll be prepared for more advanced books on the subject.

5 reasons why you'll learn faster with this book

- Unlike most MySQL books, this one starts by showing you how to query an existing database rather than how to create a new database. Why? Because that's what you're most likely to need to do first on the job. Once you master those skills, you can learn how to design and implement a database if you

need to do that. Or, you can learn how to work with other database features like transactions or stored procedures if you need to do that.

- Unlike most MySQL books, this one shows you how to use MySQL Workbench to enter and run your SQL statements. MySQL Workbench is a graphical tool that's an intuitive and user-friendly alternative to the MySQL Command Line Client, a command-line program that has been around since the beginning of MySQL. In our experience, using MySQL Workbench instead of the command line helps you learn more quickly.
- Like all of our books, this one includes hundreds of examples that range from the simple to the complex. That way, you can quickly get the idea of how a feature works from the simple examples, but you'll also see how the feature is used in the real world from the complex examples.
- Like all of our books, this one has exercises at the end of each chapter that give you hands-on experience by letting you practice what you've learned. These exercises also encourage you to experiment and to apply what you've learned in new ways.
- If you page through this book, you'll see that all of the information is presented in "paired pages," with the essential syntax, examples, and guidelines on the right page and the perspective and extra explanation on the left page. This helps you learn more with less reading, and it is the ideal reference format when you need to refresh your memory about how to do something.

What you'll learn in this book

- In section 1, you'll learn the concepts and terms you need for working with any database. You'll learn how to use MySQL Workbench to work with a database and run SQL statements. You'll also learn all the SQL skills for retrieving data from a database and for adding, updating, and deleting that data. These skills are the critical SQL skills that you'll need to get started.
- In section 2, you can learn more SQL skills as you need them. You can learn how to summarize the data that you retrieve. You can learn how to code subqueries. You can learn about the types of data that MySQL supports. And you can learn how to use MySQL functions in your SQL statements. These advanced skills are sure to raise your expertise even if you already have SQL experience.
- In section 3, you'll learn how to design a database. This includes learning how to use MySQL Workbench to create an EER (enhanced entity-relationship) model for your database. Then, you'll learn how to implement that design by using the DDL (Data Definition Language) statements that are a part of SQL. When you're done, you'll be able to design and implement your own database. In addition, you'll gain valuable perspective that will make you a better SQL programmer, even if you never have to design a database.

- In section 4, you'll learn how to use MySQL to create stored procedures, functions, triggers, and events. In addition, you'll learn how to manage transactions and locking. These features allow you to create programs made up of multiple SQL statements that can be stored in the database and accessed as needed, either to run on their own or to use in application programs…a great productivity booster! So once you master these features, you'll have a powerful set of MySQL skills.
- In section 5, you'll learn a starting set of skills for becoming a database administrator (DBA). These skills include how to secure a database, how to back up a database, and how to restore a database.

What software you need for this book

Although you should be able to use this book with most versions of MySQL, we recommend that you use:

- MySQL Community Server 8.0 or higher
- MySQL Workbench 8.0 or higher

Both of these products can be downloaded for free from MySQL's website. And appendixes A (for Windows) and B (for macOS) provide complete instructions for installing them.

Since the MySQL server is backwards compatible, all of the SQL statements presented in this book should also work with future versions of MySQL. In addition, most statements presented in this book work with earlier versions of MySQL, and we have done our best to identify any statements that don't.

If you use MySQL Workbench 8.0, all of the skills presented in this book should work exactly as described. However, MySQL Workbench is being actively developed, so its functionality is improving all the time. As a result, you may want to use a later version of MySQL Workbench. If you do, the skills presented in this book may not work exactly as described, but they should work similarly.

What you can download from our website

You can download all the source code for this book from our website. That includes:

- A script file that creates the three databases used by this book
- The source code for all of the examples in this book
- The solutions to the exercises that are at the end of each chapter

Again, appendixes A (Windows) and B (macOS) provide complete instructions for installing these items on your computer.

Support materials for trainers and instructors

If you're a corporate trainer or a college instructor who would like to use this book for a course, we've created a set of instructional materials that include: (1) a complete set of PowerPoint slides that you can use to review and reinforce the content of the book; (2) instructional objectives that describe the skills a student should have upon completion of each chapter; (3) test banks that measure mastery of those skills; (4) additional exercises that aren't in this book; and (5) solutions to those exercises.

To learn more about these instructional materials, please go to our website at www.murachforinstructors.com if you're an instructor. Or if you're a trainer, please go to www.murach.com and click on the *Courseware for Trainers* link, or contact Kelly at 1-800-221-5528 or kelly@murach.com.

Please let me know how this book works for you

When I started this book, I had two goals. First, I wanted to get you started with MySQL as quickly and easily as possible. Second, I wanted to raise your database development skills to a professional level.

Now, I thank you for buying this book. I wish you all the best with your MySQL development. And if you have any comments about this book, I'd love to hear from you.

Joel Murach, Author
joel@murach.com

Section 1

An introduction to MySQL

Before you begin to learn how to write SQL statements that work with MySQL, you need to understand some concepts and terms related to SQL and relational databases. That's what you'll learn in chapter 1. In addition, you'll need to learn about some of the tools you can use to work with a MySQL database. That's what you'll learn in chapter 2.

After that, you'll be ready to learn about the most important SQL statements. In chapter 3, you'll learn how to use the SELECT statement to retrieve data from a single table. In chapter 4, you'll learn how to use the SELECT statement to retrieve data from two or more tables. And in chapter 5, you'll learn how to use the INSERT, UPDATE, and DELETE statements to add, update, and delete rows. At that point, you'll have all of the background and skills that you need to work with the rest of this book.

1

An introduction to relational databases

This chapter presents the concepts and terms that you should understand before you begin learning how to work with a SQL database such as MySQL. Although this chapter doesn't present the coding details, it does present an overview of the most important types of SQL statements that are presented in this book.

An introduction to client/server systems

In case you aren't familiar with client/server systems, the topics that follow introduce you to their essential hardware and software components. When you use SQL to access a MySQL database, that system is often a client/server system.

The hardware components of a client/server system

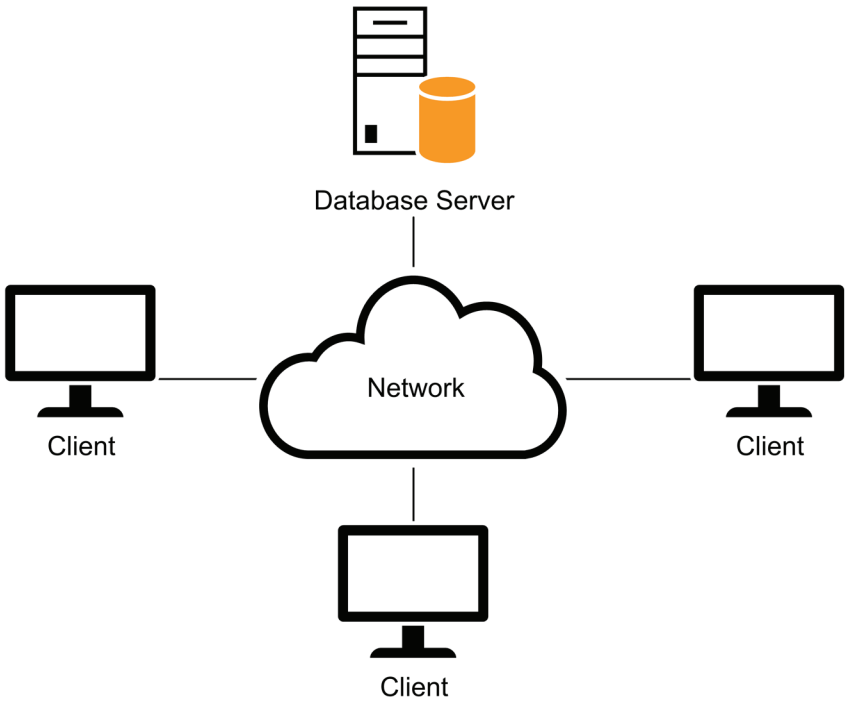
Figure 1-1 presents the three hardware components of a client/server system: the clients, the network, and the server. The *clients* are usually the PCs that are already available on the desktops throughout a company. Clients can also be mobile devices like laptops, tablets, and smartphones. And the *network* is the cabling, communication lines, network interface cards, hubs, routers, and other components that connect the clients and the server.

The *server*, commonly referred to as a *database server*, is a computer that has enough processor speed, internal memory (RAM), and disk storage to store the files and databases of the system and provide services to the clients of the system. This computer can be a high-powered PC, a midrange system like an IBM Power System or Unix system, or even a mainframe system. When a system consists of networks, midrange systems, and mainframe systems, often spread throughout the country or world, it is commonly referred to as an *enterprise system*.

To back up the files of a client/server system, a server usually has a backup disk drive or some other form of offline storage. It often has one or more printers or specialized devices that can be shared by the users of the system. And it can provide programs or services like e-mail that can be accessed by all the users of the system.

In a simple client/server system, the clients and the server are part of a *local area network* (*LAN*). However, two or more LANs that reside at separate geographical locations can be connected as part of a larger network such as a *wide area network* (*WAN*). In addition, individual systems or networks can be connected over the Internet.

A simple client/server system



The three hardware components of a client/server system

- The *clients* are the PCs, Macs, or workstations of the system. They can also be mobile devices like laptops, tablets, and smartphones.
- The *server* is a computer that stores the files and databases of the system and provides services to the clients. When it stores databases, it's often referred to as a *database server*.
- The *network* consists of the cabling, communication lines, and other components that connect the clients and the servers of the system.

Client/server system implementations

- In a simple *client/server system* like the one above, the server is typically a high-powered PC that communicates with the clients over a *local area network* (*LAN*).
- The server can also be a midrange system, like an IBM Power System or a Unix system, or it can be a mainframe system.
- A client/server system can also consist of one or more PC-based systems, one or more midrange systems, and a mainframe system in dispersed geographical locations. This type of system is commonly referred to as an *enterprise system*.
- Individual systems and LANs can be connected and share data over larger private networks, such as a *wide area network* (*WAN*), or a public network like the Internet.

Figure 1-1 The hardware components of a client/server system

The software components of a client/server system

Figure 1-2 presents the software components of a typical client/server system. Here, the server requires a *database management system* (*DBMS*) like MySQL or Microsoft SQL Server. This DBMS manages the databases that are stored on the server.

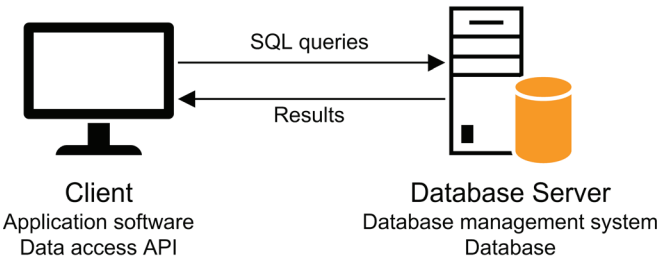
In contrast to a server, each client requires *application software* to perform useful work. This can be a purchased software package like a financial accounting package, or it can be custom software that's developed for a specific application.

Although the application software is run on the client, it uses data that's stored on the server. To do that, it uses a *data access API* (*application programming interface*). Since the technique you use to work with an API depends on the programming language and API you're using, you won't learn those techniques in this book. Instead, you'll learn about a standard language called *SQL* (*Structured Query Language*) that lets any application communicate with any DBMS. (In conversation, SQL is pronounced as either *S-Q-L* or *sequel*.)

Once the software for both client and server is installed, the client communicates with the server via *SQL queries* (or just *queries*) that are passed to the DBMS through the API. After the client sends a query to the DBMS, the DBMS interprets the query and sends the results back to the client.

In a client/server system, the processing is divided between the clients and the server. In this figure, for example, the DBMS on the server processes the requests that are made by the application running on the client. Theoretically, at least, this balances the workload between the clients and the server so the system works more efficiently.

Client software, server software, and the SQL interface



Server software

- To store and manage the databases of the client/server system, each server requires a *database management system* (*DBMS*) like MySQL.
- The processing that's done by the DBMS is typically referred to as *back-end processing*, and the database server is referred to as the *back end*.

Client software

- The *application software* does the work that the user wants to do. This type of software can be purchased or developed.
- The *data access API* (*application programming interface*) provides the interface between the application program and the DBMS. For example, for Java applications, the most common data access API for MySQL is *JDBC* (*Java Database Connectivity*).
- The processing that's done by the client software is typically referred to as *front-end processing*, and the client is typically referred to as the *front end*.

The SQL interface

- The application software communicates with the DBMS by sending *SQL queries* through the data access API. When the DBMS receives a query, it provides a service like returning the requested data (the *query results*) to the client.
- *SQL* stands for *Structured Query Language*, which is the standard language for working with a relational database.

Client/server versus file-handling systems

- In a client/server system, the processing done by an application is typically divided between the client and the server.
- In a file-handling system, all of the processing is done on the clients. Although the clients may access data that's stored in files on the server, none of the processing is done by the server. As a result, a file-handling system isn't a client/server system.

Figure 1-2 The software components of a client/server system

Other client/server architectures

In its simplest form, a client/server system consists of a single database server and one or more clients. Many client/server systems today, though, include additional servers. For example, figure 1-3 shows two client/server systems that include an additional server between the clients and the database server.

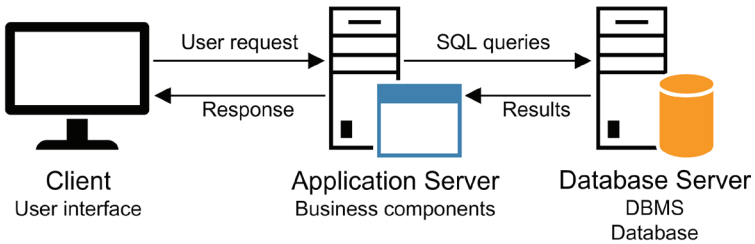
The first illustration is for a simple networked system. With this system, only the user interface for an application runs on the client. The rest of the processing that's done by the application is stored in one or more *business components* on the *application server*. Then, the client sends requests to the application server for processing. If the request involves accessing data in a database, the application server formulates the appropriate query and passes it on to the database server. The results of the query are then sent back to the application server, which processes the results and sends the appropriate response back to the client.

Similar processing is done by a web-based system, as illustrated by the second example in this figure. In this case, though, a *web browser* running on the client is used to send requests to a *web application* running on a *web server* somewhere on the Internet. The web application, in turn, can use *web services* to perform some of its processing. Then, the web application or web service can pass requests for data on to the database server.

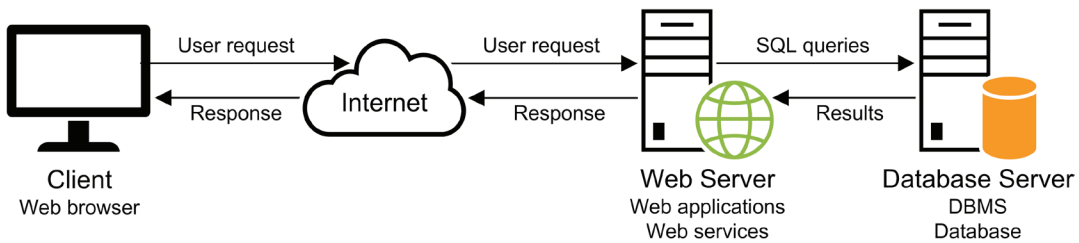
Although this figure should give you an idea of how client/server systems can be configured, you should realize that they can be much more complicated than what's shown here. For example, business components can be distributed over any number of application servers, and those components can communicate with databases on any number of database servers. Similarly, the web applications and services in a web-based system can be distributed over numerous web servers that access numerous database servers. In most cases, though, you don't need to know how a system is configured to use SQL.

Before I go on, you should know that client/server systems aren't the only systems that support SQL. For example, traditional mainframe systems and newer *thin client* systems also use SQL. Unlike client/server systems, though, most of the processing for these types of systems is done by a mainframe or another high-powered machine. The terminals or PCs that are connected to the system do little or no work.

A networked system that uses an application server



A simple web-based system



Description

- In addition to a database server and clients, a client/server system can include additional servers, such as *application servers* and *web servers*.
- Application servers are typically used to store *business components* that do part of the processing of the application. In particular, these components are used to process database requests from the user interface running on the client.
- Web servers are typically used to store *web applications* and *web services*. Web applications are applications that are designed to run on a web server. Web services are like business components, except that, like web applications, they are designed to run on a web server.
- In a web-based system, a *web browser* running on a client sends a request to a web server over the Internet. Then, the web server processes the request and passes any requests for data on to the database server.
- More complex system architectures can include two or more application servers, web servers, and database servers.

Figure 1-3 Other client/server architectures

An introduction to the relational database model

In 1970, Dr. E. F. Codd developed a model for a new type of database called a *relational database.* This type of database eliminated some of the problems that were associated with standard files and other database designs. By using the relational model, you can reduce data redundancy, which saves disk storage and leads to efficient data retrieval. You can also view and manipulate data in a way that is both intuitive and efficient. Today, relational databases are the de facto standard for database applications.

How a table is organized

The model for a relational database states that data is stored in one or more *tables*. It also states that each table can be viewed as a two-dimensional matrix consisting of *rows* and *columns*. This is illustrated by the relational table in figure 1-4. Each row in this table contains information about a single vendor.

In practice, the rows and columns of a relational database table are often referred to by the more traditional terms, *records* and *fields*. In fact, some software packages use one set of terms, some use the other, and some use a combination. In this book, I use the terms *rows* and *columns* because those are the terms used by MySQL.

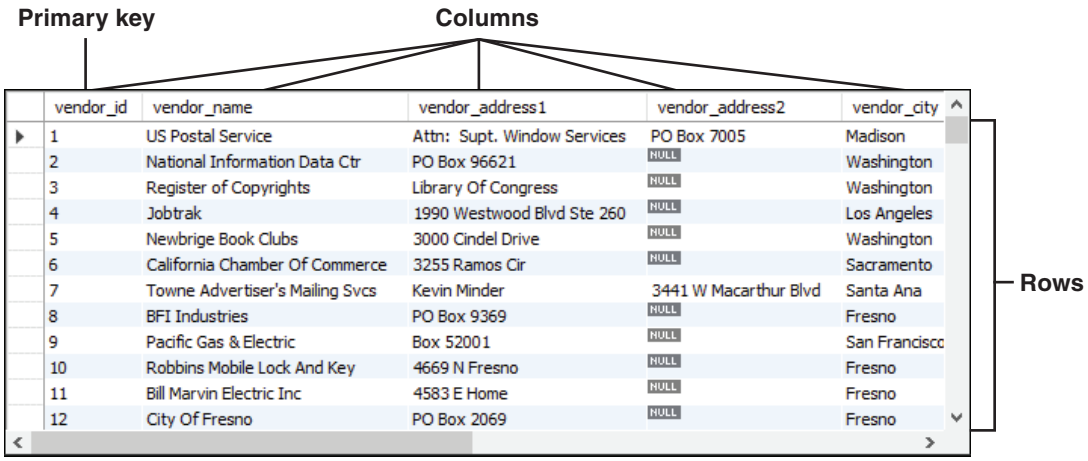
In general, each table is modeled after a real-world entity such as a vendor or an invoice. Then, the columns of the table represent the attributes of the entity such as name, address, and phone number. And each row of the table represents one instance of the entity. A *value* is stored at the intersection of each row and column, sometimes called a *cell*.

If a table contains one or more columns that uniquely identify each row in the table, you can define these columns as the *primary key* of the table. For instance, the primary key of the Vendors table in this figure is the vendor_id column. In this example, the primary key consists of a single column. However, a primary key can also consist of two or more columns, in which case it's called a *composite primary key*.

In addition to primary keys, some database management systems let you define additional keys that uniquely identify each row in a table. If, for example, the vendor_name column in the Vendors table contains unique data, it can be defined as a *non-primary key*. In MySQL, this is called a *unique key*.

Indexes provide an efficient way of accessing the rows in a table based on the values in one or more columns. Because applications typically access the rows in a table by referring to their key values, an index is automatically created for each key you define. However, you can define indexes for other columns as well. If, for example, you frequently need to sort the Vendor rows by zip code, you can set up an index for that column. Like a key, an index can include one or more columns.

The Vendors table in an Accounts Payable database



vendor_id	vendor_name	vendor_address1	vendor_address2	vendor_city
1	US Postal Service	Attn: Supt. Window Services	PO Box 7005	Madison
2	National Information Data Ctr	PO Box 96621	NULL	Washington
3	Register of Copyrights	Library Of Congress	NULL	Washington
4	Jobtrak	1990 Westwood Blvd Ste 260	NULL	Los Angeles
5	Newbrige Book Clubs	3000 Cindel Drive	NULL	Washington
6	California Chamber Of Commerce	3255 Ramos Cir	NULL	Sacramento
7	Towne Advertiser's Mailing Svcs	Kevin Minder	3441 W Macarthur Blvd	Santa Ana
8	BFI Industries	PO Box 9369	NULL	Fresno
9	Pacific Gas & Electric	Box 52001	NULL	San Francisco
10	Robbins Mobile Lock And Key	4669 N Fresno	NULL	Fresno
11	Bill Marvin Electric Inc	4583 E Home	NULL	Fresno
12	City Of Fresno	PO Box 2069	NULL	Fresno

Concepts

- A *relational database* consists of *tables*. Tables consist of *rows* and *columns*, which can also be referred to as *records* and *fields*.
- A table is typically modeled after a real-world entity, such as an invoice or a vendor.
- A column represents some attribute of the entity, such as the amount of an invoice or a vendor's address.
- A row contains a set of values for a single instance of the entity, such as one invoice or one vendor.
- The intersection of a row and a column is sometimes called a *cell*. A cell stores a single *value*.
- Most tables have a *primary key* that uniquely identifies each row in the table. The primary key is usually a single column, but it can also consist of two or more columns. If a primary key uses two or more columns, it's called a *composite primary key*.
- In addition to primary keys, some database management systems let you define one or more *non-primary keys*. In MySQL, these keys are called *unique keys*. Like a primary key, a non-primary key uniquely identifies each row in the table.
- A table can also be defined with one or more *indexes*. An index provides an efficient way to access data from a table based on the values in specific columns. An index is automatically created for a table's primary and non-primary keys.

Figure 1-4 How a database table is organized

How tables are related

The tables in a database can be related to other tables by values in specific columns. The two tables shown in figure 1-5 illustrate this concept. Here, each row in the Vendors table is related to one or more rows in the Invoices table. This is called a *one-to-many relationship*.

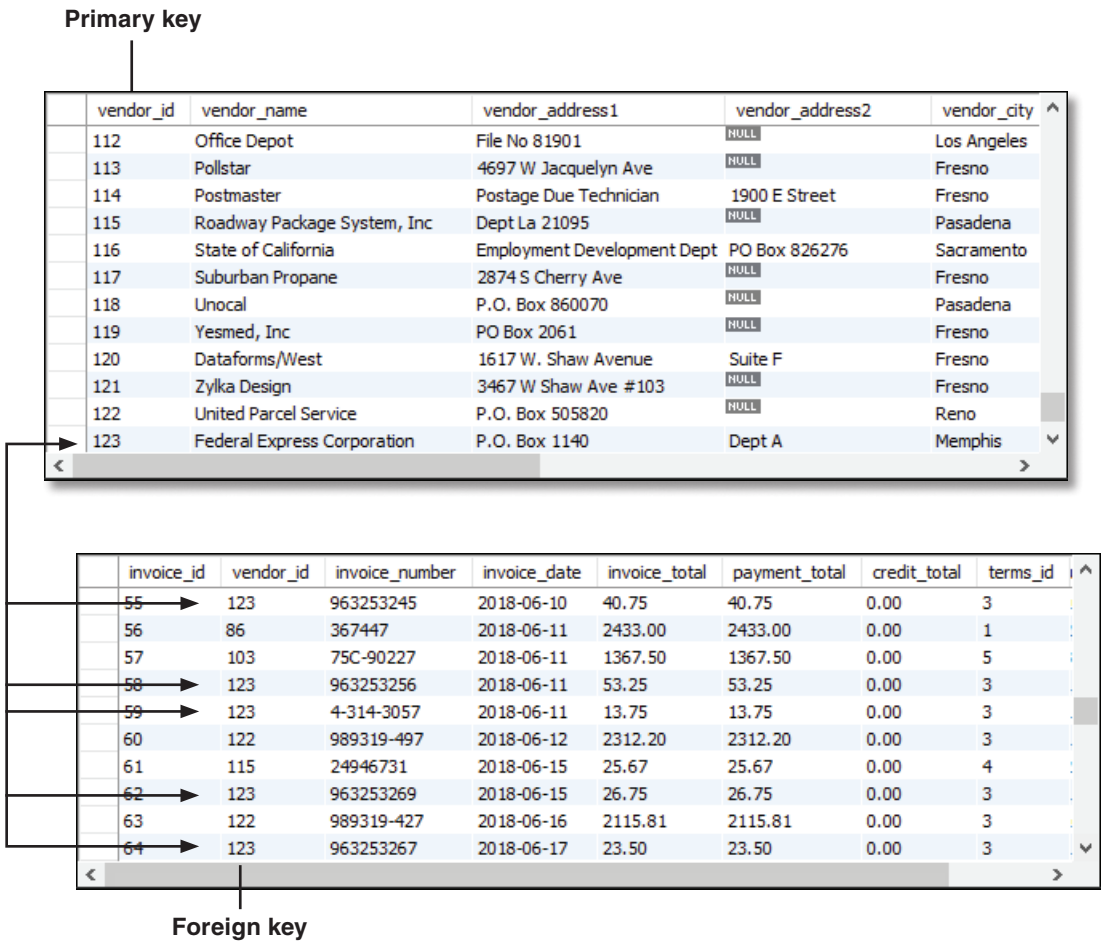
Typically, relationships exist between the primary key in one table and the *foreign key* in another table. The foreign key is simply one or more columns in a table that refer to a primary key in another table. In this figure, for example, the vendor_id column is the foreign key in the Invoices table and is used to create the relationship between the Vendors table and the Invoices table.

Although one-to-many relationships are the most common, two tables can also have a one-to-one or many-to-many relationship. If a table has a *one-to-one relationship* with another table, the data in the two tables could be stored in a single table. Because of that, one-to-one relationships are used infrequently.

In contrast, a *many-to-many relationship* is usually implemented by using an intermediate table that has a one-to-many relationship with the two tables in the many-to-many relationship. In other words, a many-to-many relationship can usually be broken down into two one-to-many relationships.

If you define a foreign key for a table in MySQL, you can have the foreign key enforce *referential integrity*. When MySQL enforces referential integrity, it makes sure that any changes to the data in the database don't create invalid relationships between tables. For example, if you try to add a row to the Invoices table with a vendor_id value that doesn't exist in the Vendors table, MySQL won't add the row and will display an error. This helps to maintain the integrity of the data that's stored in the database.

The relationship between the Vendors and Invoices tables in the database



Concepts

- The vendor_id column in the Invoices table is called a *foreign key* because it identifies a related row in the Vendors table. A table may contain one or more foreign keys.
- When you define a foreign key for a table in MySQL, you can have that foreign key enforce *referential integrity*. Then, MySQL makes sure that any changes to the data in the database don't create invalid relationships between tables.
- The most common type of relationship is a *one-to-many relationship* as illustrated by the Vendors and Invoices tables. A table can also have a *one-to-one relationship* or a *many-to-many relationship* with another table.

Figure 1-5 How tables are related

How columns are defined

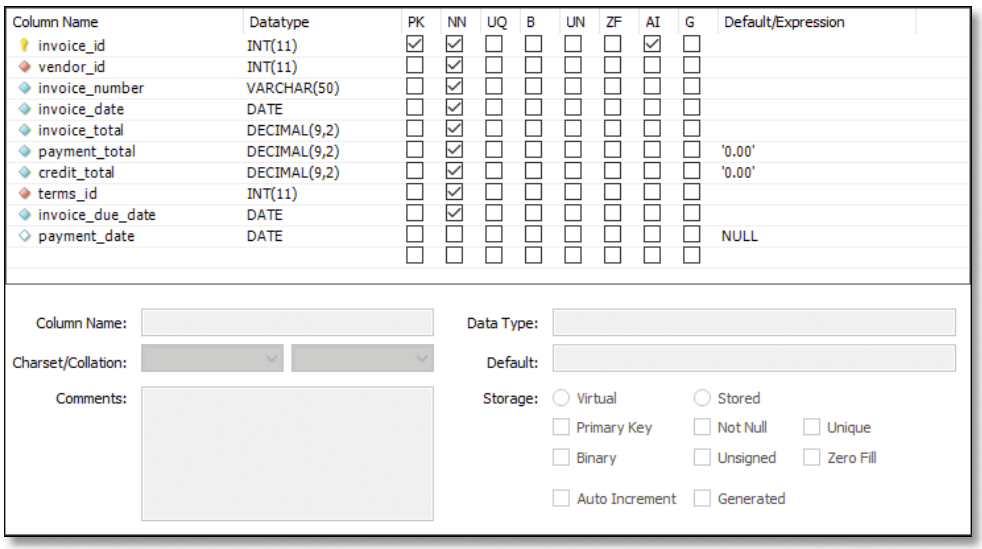
When you define a column in a table, you assign properties to it as indicated by the design of the Invoices table in figure 1-6. The most critical property for a column is its data type, which determines the type of information that can be stored in the column. With MySQL, you can choose from the *data types* listed in this figure as well as several other data types that are described in chapter 8. As you define each column in a table, you generally try to assign the data type that minimizes the use of disk storage because that improves the performance of the queries later.

In addition to a data type, you must identify whether the column can store a *null value* (or just *null*). A null represents a value that's unknown, unavailable, or not applicable. In this figure, the columns that have the NN (not null) box checked don't allow null values. If you don't allow null values for a column, you must provide a value for that column when you store a new row in the table.

You can also assign a *default value* to each column. Then, that value is assigned to the column if another value isn't provided. As you can see, three of the columns of the Invoices table have a default value. You'll learn more about how to work with null and default values later in this book.

Each table can also contain a numeric column whose value is generated automatically by the DBMS. In MySQL, a column like this is called an *auto increment column*. You'll learn more about defining auto increment columns in chapter 11. For now, just note that the primary key of both the Vendors and Invoices tables—vendor_id and invoice _id—are auto increment columns.

The columns of the Invoices table



Common MySQL data types

Type	Description
`CHAR, VARCHAR`	A string of letters, symbols, and numbers.
`INT, DECIMAL`	Integer and decimal numbers that contain an exact value.
`FLOAT`	Floating-point numbers that contain an approximate value.
`DATE`	Dates and times.

Description

- The *data type* that's assigned to a column determines the type of information that can be stored in the column.
- Each column definition also indicates whether or not it can contain *null values*. A null value indicates that the value of the column is unknown.
- A column can also be defined with a *default value*. Then, that value is used if another value isn't provided when a row is added to the table.
- A column can also be defined as an *auto increment column*. An auto increment column is a numeric column whose value is generated automatically when a row is added to the table.

Figure 1-6 How columns are defined

How to read a database diagram

When working with relational databases, you can use an *entity-relationship* (*ER*) *diagram* to show how the tables in a database are defined and related. Or, you can use a newer version of an ER diagram known as an *enhanced entity-relationship* (*EER*) *diagram*. In figure 1-7, for example, you can see an EER diagram for the AP (Accounts Payable) database that's used throughout this book. This diagram shows that the database contains five related tables: Vendors, Terms, Invoices, Invoice_Line_Items, and General_Ledger_Accounts.

For each table, this diagram shows how the columns are defined. For example, it shows that the Vendors table has 12 columns. It shows the name and data type for each column. It uses a key icon to show that the primary key for this table is the vendor_id column. And it uses a dark diamond icon to show that the table has two columns that are foreign keys: default_terms_id and default_account_number.

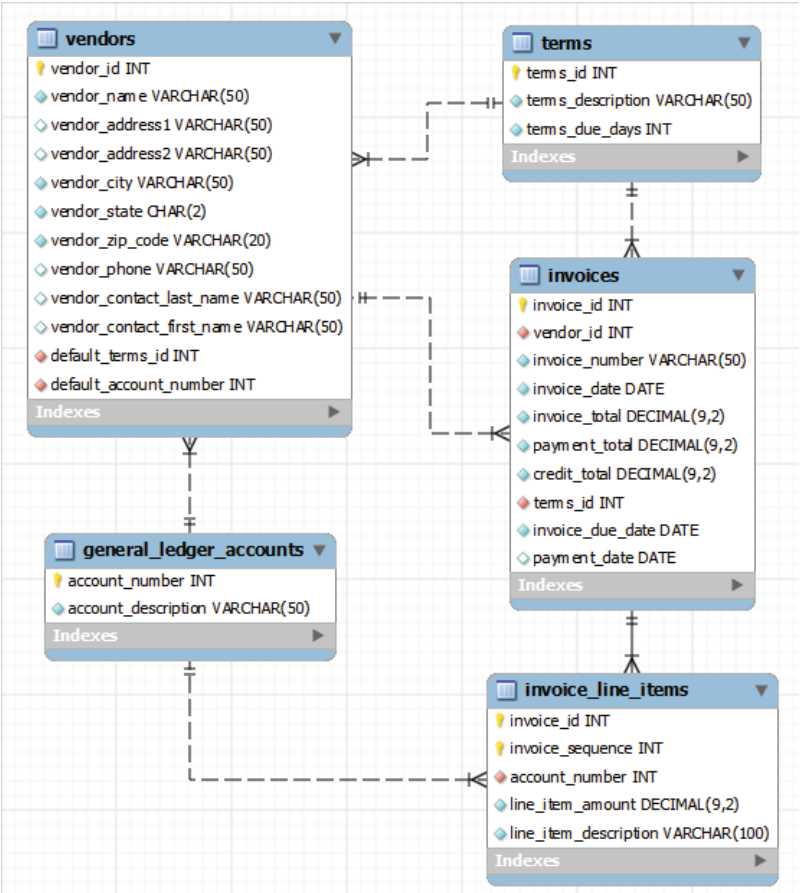
This diagram also shows how the tables are related. To do that, it places a connector symbol between the tables. For example, the connector between the Vendors and Invoices table shows that these tables have a one-to-many relationship. On this connector, the symbol closest to the Invoices table indicates that many invoices can exist for each vendor, and the symbol closest to the Vendors table shows that only one vendor can exist for each invoice. If you study the primary and foreign keys for these tables, you can deduce that these tables are designed to be joined on the vendor_id column that's in both tables.

Similarly, this diagram shows that there's a one-to-many relationship between the Terms and Vendors table. In other words, each terms of payment can have many vendors, but each vendor can only have one default terms of payment. If you study the primary and foreign keys for these tables, you can deduce that these tables are designed to be joined on the terms_id column of the Terms table and the default_terms_id column of the Vendors table.

Most of the tables in this diagram begin with a single column that defines the primary key for the table. However, the Invoice_Line_Items table begins with two columns (invoice_id and invoice_sequence) that define the primary key for this table. In other words, since the invoice_id column doesn't uniquely identify the line item, the primary key must include the invoice_sequence column so it can uniquely identify each row.

In chapter 10, you'll learn how to use MySQL Workbench to create and work with EER diagrams. For now, you just need to understand how to read the diagram presented in this figure so you can understand the relationships between the tables in the AP database.

An EER diagram for the AP (Accounts Payable) database



Description

- An *entity-relationship* (*ER*) *diagram* or *enhanced entity-relationship* (*EER*) *diagram* can be used to show how the tables in a database are defined and related.

Figure 1-7 How to read a database diagram

An introduction to SQL and SQL-based systems

In the topics that follow, you'll learn how SQL and SQL-based database management systems evolved. In addition, you'll learn how some of the most popular SQL-based systems compare.

A brief history of SQL

Prior to the release of the first relational database management system, each database had a unique physical structure and a unique programming language that the programmer had to understand. That all changed with the advent of SQL and the relational database management system.

Figure 1-8 lists the important events in the history of SQL. In 1970, Dr. E. F. Codd published an article that described the relational database model he had been working on with a research team at IBM. Then, in 1979, Relational Software, Inc. released the first *relational database management system*, called *Oracle*. This *RDBMS* ran on a minicomputer and used SQL as its query language. This product was widely successful, and the company later changed its name to Oracle to reflect that success.

In 1982, IBM released its first commercial SQL-based RDBMS, called *SQL/DS* (*SQL/Data System*). This was followed in 1985 by *DB2* (*Database 2*). Both systems ran only on IBM mainframe computers. Later, DB2 was ported to other operating systems, including Unix and Windows. Today, it continues to be IBM's premier database system.

During the 1980s, other SQL-based database systems, including SQL Server, were developed. Although each of these systems used SQL as its query language, each implementation was unique. That began to change in 1989, when the *American National Standards Institute* (*ANSI*) published its first set of standards for a database query language. As each database manufacturer has attempted to comply with these standards, their implementations of SQL have become more similar. However, each still has its own *dialect* of SQL that includes additions, or *extensions*, to the standards.

Although you should be aware of the SQL standards, they will have little effect on your job as a MySQL programmer. The main benefit of the standards is that the basic SQL statements are the same in each dialect. As a result, once you've learned one dialect, it's relatively easy to learn another. On the other hand, porting applications that use SQL from one type of database to another often requires substantial changes.

1995 saw the first release of MySQL, which was used internally by the company that developed it, MySQL AB. In 2000, MySQL became an open-source database. Since then, MySQL has become one of the most popular databases, especially for web applications. In 2008, MySQL was acquired by Sun Microsystems, and in 2010, Oracle acquired Sun.

Important events in the history of SQL

Year	Event
1970	Dr. E. F. Codd developed the relational database model.
1979	Relational Software, Inc. (later renamed Oracle) released the first relational DBMS, Oracle.
1982	IBM released their first relational database system, SQL/DS (SQL/Data System).
1985	IBM released DB2 (Database 2).
1987	Microsoft released SQL Server.
1989	The American National Standards Institute (ANSI) published the first set of standards for a database query language, called ANSI/ISO SQL-89, or SQL1.
1992	ANSI published revised standards (ANSI/ISO SQL-92, or SQL2) that were more stringent than SQL1 and incorporated many new features. These standards introduced levels of compliance, or levels of conformance, that indicated the extent to which a dialect met the standards.
1995	MySQL AB released MySQL for internal use.
1999	ANSI published SQL3 (ANSI/ISO SQL:1999). These standards incorporated new features, including support for objects. Levels of compliance were dropped and were replaced by a core specification that defined the essential elements for compliance, plus nine packages. Each package is designed to serve a specific market niche.
2000	MySQL became an open-source database.
2003	ANSI published SQL4 (ANSI/ISO SQL:2003). These standards introduced XML-related features, standardized sequences, and identity columns.
2006	ANSI published SQL:2006, which defined how SQL can be used with XML. These standards also allowed applications to integrate XQuery into their SQL code.
2008	Sun Microsystems acquired MySQL.
2008	ANSI published SQL:2008. These standards introduced INSTEAD OF triggers and the TRUNCATE statement.
2010	Oracle acquired Sun Microsystems and MySQL. Soon after the acquisition, many of the original developers of MySQL left and begin working on a fork of the open-source code named MariaDB.
2016	ANSI published SQL:2016, which includes support for regular expressions and JavaScript Object Notation (JSON).

Description

- Although SQL is a standard language, each vendor has its own *SQL dialect*, or *variant*, that may include extensions to the standards.

How knowing "standard SQL" helps you

- The most basic SQL statements are the same for all SQL dialects.
- Once you have learned one SQL dialect, you can easily learn other dialects.

How knowing "standard SQL" does not help you

- Any non-trivial application will require modification when moved from one SQL database to another.

Figure 1-8 A brief history of SQL

Soon after Oracle's acquisition of MySQL, many of the original developers of MySQL left and began working on a fork of the open-source code named MariaDB. One of MariaDB's stated goals is to remain free and open-source while maintaining high compatibility with MySQL so it can be used as a drop-in replacement for MySQL. As a result, many large companies, including Google and Wikipedia, have switched from MySQL to MariaDB.

A comparison of Oracle, DB2, Microsoft SQL Server, and MySQL

Although this book is about MySQL, you may want to know about some of the other SQL-based relational database management systems. Figure 1-9 compares MySQL with three other popular databases: Oracle, DB2, and Microsoft (MS) SQL Server.

Oracle has a huge installed base of customers and continues to dominate the marketplace, especially for servers running the Unix or Linux operating system. Oracle works well for large systems and has a reputation for being extremely reliable. However, it also has a reputation for being expensive and difficult to use.

DB2 was originally designed to run on IBM mainframe systems and continues to be the premier database for those systems. It also dominates in hybrid environments where IBM mainframes and newer servers must coexist. Although it has a reputation for being expensive, it also has a reputation for being reliable and easy to use.

SQL Server was designed by Microsoft to run on Windows and is widely used for small- to medium-sized departmental systems. It has a reputation for being inexpensive and easy to use. However, it also has a reputation for not scaling well for systems with a large number of users.

MySQL runs on all major operating systems and is widely used for web applications. MySQL is an *open-source database*, which means that any developer can view and improve its source code. In addition, the MySQL Community Server is free for most users, although Oracle also sells an Enterprise Edition of MySQL that has advanced features.

One of the main differences between MySQL and SQL Server is that MySQL runs under most operating systems including Unix, Linux, Windows, macOS, and IBM's z/OS. In contrast, SQL Server only runs under Windows and Linux, and the Linux version was just released in 2017. Since many developers consider z/OS and Unix to be more stable and secure than Windows, most large companies use z/OS or Unix as the operating system for the servers that store the databases for mission-critical applications. As a result, they can't use SQL Server and must use Oracle, DB2, or MySQL.

If you search the Internet, you'll find that dozens of other relational database products are also available. These include proprietary databases like Informix, Sybase, and Teradata. And they include open-source databases like PostgreSQL and MariaDB.

A comparison of Oracle, DB2, SQL Server, and MySQL

	Oracle	DB2	SQL Server	MySQL
Release year	1979	1985	1987	2000
Platforms	Unix/Linux	OS/390, z/OS	Windows	Unix/Linux
	OS/390, z/OS	Unix/Linux	Linux	Windows
	Windows	Windows		macOS
	macOS	macOS		z/OS

Description

- Oracle is typically used for large, mission-critical systems that run on one or more Unix servers.
- DB2 is typically used for large, mission-critical systems that run on legacy IBM mainframe systems using the z/OS or OS/390 operating system.
- Microsoft (MS) SQL Server is typically used for small- to medium-sized systems that run on one or more Windows servers.
- MySQL is a popular *open-source database* that runs on all major operating systems and is commonly used for web applications.

Figure 1-9 A comparison of Oracle, DB2, SQL Server, and MySQL

The SQL statements

In the topics that follow, you'll learn about some of the SQL statements provided by MySQL. You can use some of these statements to manipulate the data in a database, and you can use others to work with database objects. Although you may not be able to code these statements after reading these topics, you should have a good idea of how they work. Then, you'll be better prepared to learn the details of coding these statements when they're presented in the rest of this book.

An introduction to the SQL statements

Figure 1-10 summarizes some of the most common SQL statements. These statements can be divided into two categories. The statements that work with the data in a database are called the *data manipulation language* (*DML*). These statements are presented in the first group in this figure, and these are the statements that application programmers use the most.

The statements that create databases and work with the objects within a database are called the *data definition language* (*DDL*). On large systems, these statements are used exclusively by *database administrators* (*DBAs*). It's the DBA's job to maintain existing databases, tune them for faster performance, and create new databases. On smaller systems, though, the SQL programmer may fill the role of the DBA.

SQL statements used to work with data (DML)

Statement	Description
`SELECT`	Retrieves data from one or more tables.
`INSERT`	Adds new rows to a table.
`UPDATE`	Changes existing rows in a table.
`DELETE`	Deletes existing rows from a table.

SQL statements used to work with database objects (DDL)

Statement	Description
`CREATE DATABASE`	Creates a new database on the server.
`CREATE TABLE`	Creates a new table in a database.
`CREATE INDEX`	Creates a new index for a table.
`ALTER TABLE`	Changes the definition of an existing table.
`ALTER INDEX`	Changes the structure of an existing index.
`DROP DATABASE`	Deletes an existing database and all of its tables.
`DROP TABLE`	Deletes an existing table.
`DROP INDEX`	Deletes an existing index.

Description

- The SQL statements can be divided into two categories: the *data manipulation language (DML)* that lets you work with the data in the database and the *data definition language (DDL)* that lets you work with the objects in the database.
- MySQL programmers typically work with the DML statements, while *database administrators* (*DBAs*) use the DDL statements.

Figure 1-10 An introduction to the SQL statements

How to work with database objects

To give you an idea of how you use the DDL statements shown in the previous figure, figure 1-11 presents some examples. Here, the first example creates a database named AP. Then, the second example selects that database. As a result, the rest of the statements in this figure are run against the AP database.

The third example creates the Invoices table that's used throughout this chapter. If you don't understand all of this code right now, don't worry. You'll learn how to code statements like this in chapter 11. For now, just realize that this statement defines each column in the table, including its data type, whether or not it allows null values, and its default value if it has one.

In addition, the third example defines the primary and foreign key columns for the table. These definitions are one type of *constraint*. Since the Invoices table includes foreign keys to the Vendors and Terms tables, these tables must be created before the Invoices table. Conversely, before you can delete the Vendors and Terms tables, you must delete the Invoices table.

The fourth example in this figure changes the Invoices table by adding a column to it. Like the statement that created the table, this statement specifies the attributes of the new column. Then, the fifth example deletes the column that was just added.

The sixth example creates an index on the Invoices table. In this case, the index is for the vendor_id column, which is used frequently to access the table. Then, the last example deletes the index that was just added.

A statement that creates a new database

```
CREATE DATABASE ap
```

A statement that selects the current database

```
USE ap
```

A statement that creates a new table

```
CREATE TABLE invoices
(
  invoice_id          INT             PRIMARY KEY    AUTO_INCREMENT,
  vendor_id           INT             NOT NULL,
  invoice_number      VARCHAR(50)     NOT NULL,
  invoice_date        DATE            NOT NULL,
  invoice_total       DECIMAL(9,2)    NOT NULL,
  payment_total       DECIMAL(9,2)                   DEFAULT 0,
  credit_total        DECIMAL(9,2)                   DEFAULT 0,
  terms_id            INT             NOT NULL,
  invoice_due_date    DATE            NOT NULL,
  payment_date        DATE,
  CONSTRAINT invoices_fk_vendors
    FOREIGN KEY (vendor_id)
    REFERENCES vendors (vendor_id),
  CONSTRAINT invoices_fk_terms
    FOREIGN KEY (terms_id)
    REFERENCES terms (terms_id)
)
```

A statement that adds a new column to a table

```
ALTER TABLE invoices
ADD balance_due DECIMAL(9,2)
```

A statement that deletes the new column

```
ALTER TABLE invoices
DROP COLUMN balance_due
```

A statement that creates an index on the table

```
CREATE INDEX invoices_vendor_id_index
  ON invoices (vendor_id)
```

A statement that deletes the new index

```
DROP INDEX invoices_vendor_id_index
  ON invoices
```

Figure 1-11 Typical statements for working with database objects

How to query a single table

Figure 1-12 shows how to use a SELECT statement to query a single table in a database. To start, this figure shows some of the columns and rows of the Invoices table. Then, in the SELECT statement that follows, the SELECT clause names the columns to be retrieved, and the FROM clause names the table that contains the columns, called the *base table*. In this case, six columns will be retrieved from the Invoices table.

Note that the last column, balance_due, is calculated from three other columns in the table. In other words, a column by the name of balance_due doesn't actually exist in the database. This type of column is called a *calculated value*, and it exists only in the results of the query.

In addition to the SELECT and FROM clauses, this SELECT statement includes a WHERE clause and an ORDER BY clause. The WHERE clause gives the criteria for the rows to be selected. In this case, a row is selected only if it has a balance due that's greater than zero. Finally, the returned rows are sorted by the invoice_date column.

This figure also shows the *result set* (or *result table*) that's returned by the SELECT statement. A result set is a logical table that's created temporarily within the database. When an application requests data from a database, it receives a result set.

The Invoices base table

invoice_id	vendor_id	invoice_number	invoice_date	invoice_total	payment_total	credit_total	terms_id
1	122	989319-457	2018-04-08	3813.33	3813.33	0.00	3
2	123	263253241	2018-04-10	40.20	40.20	0.00	3
3	123	963253234	2018-04-13	138.75	138.75	0.00	3
4	123	2-000-2993	2018-04-16	144.70	144.70	0.00	3
5	123	963253251	2018-04-16	15.50	15.50	0.00	3

A SELECT statement that retrieves and sorts selected columns and rows from the Invoices table

```
SELECT invoice_number, invoice_date, invoice_total,
    payment_total, credit_total,
    invoice_total - payment_total - credit_total AS balance_due
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
ORDER BY invoice_date
```

The result set defined by the SELECT statement

invoice_number	invoice_date	invoice_total	payment_total	credit_total	balance_due
39104	2018-07-10	85.31	0.00	0.00	85.31
963253264	2018-07-18	52.25	0.00	0.00	52.25
31361833	2018-07-21	579.42	0.00	0.00	579.42
263253268	2018-07-21	59.97	0.00	0.00	59.97
263253270	2018-07-22	67.92	0.00	0.00	67.92

Concepts

- You use the SELECT statement to retrieve selected columns and rows from a *base table*. The result of a SELECT statement is a *result table*, or *result set*, like the one shown above.
- A result set can include *calculated values* that are calculated from columns in the table.
- A SELECT statement is commonly referred to as a *query*.

Figure 1-12 How to query a single table

How to join data from two or more tables

Figure 1-13 presents a SELECT statement that retrieves data from two tables. This type of operation is called a *join* because the data from the two tables is joined together into a single result set. For example, the SELECT statement in this figure joins data from the Invoices and Vendors tables.

An *inner join* is the most common type of join. When you use an inner join, rows from the two tables in the join are included in the result table only if their related columns match. These matching columns are specified in the FROM clause of the SELECT statement. In the SELECT statement in this figure, for example, rows from the Invoices and Vendors tables are included only if the value of the vendor_id column in the Vendors table matches the value of the vendor_id column in one or more rows in the Invoices table. If there aren't any invoices for a particular vendor, that vendor won't be included in the result set.

Although this figure shows only how to join data from two tables, you can extend this syntax to join data from three or more tables. If, for example, you want to include line item data from a table named Invoice_Line_Items in the results shown in this figure, you can code the FROM clause of the SELECT statement like this:

```
FROM vendors
    INNER JOIN invoices
        ON vendors.vendor_id = invoices.vendor_id
    INNER JOIN invoice_line_items
        ON invoices.invoice_id = invoice_line_items.invoice_id
```

Then, in the SELECT clause, you can include any of the columns in the Invoice_Line_Items table.

In addition to inner joins, most relational databases including MySQL support other types of joins such as *outer joins*. An outer join lets you include all rows from a table even if the other table doesn't have a matching row. You'll learn more about the different types of joins in chapter 4.

A SELECT statement that joins data from the Vendors and Invoices tables

```
SELECT vendor_name, invoice_number, invoice_date, invoice_total
FROM vendors INNER JOIN invoices
    ON vendors.vendor_id = invoices.vendor_id
WHERE invoice_total >= 500
ORDER BY vendor_name, invoice_total DESC
```

The result set defined by the SELECT statement

vendor_name	invoice_number	invoice_date	invoice_total
Federal Express Corporation	963253230	2018-07-07	739.20
Ford Motor Credit Company	9982771	2018-07-24	503.20
Franchise Tax Board	RTR-72-3662-X	2018-05-25	1600.00
Fresno County Tax Collector	P02-88D77S7	2018-05-03	856.92
IBM	Q545443	2018-06-09	1083.58
Ingram	31359783	2018-06-03	1575.00
Ingram	31361833	2018-07-21	579.42
Malloy Lithographing Inc	0-2058	2018-05-28	37966.19

Concepts

- A *join* lets you combine data from two or more tables into a single result set.
- The most common type of join is an *inner join*. This type of join returns rows from both tables only if their related columns match.
- An *outer join* returns rows from one table in the join even if the other table doesn't contain a matching row.

Figure 1-13 How to join data from two or more tables

How to add, update, and delete data in a table

Figure 1-14 shows how you can use the INSERT, UPDATE, and DELETE statements to modify the data in a table. In this figure, for example, the first statement is an INSERT statement that adds a row to the Invoices table. To do that, the INSERT clause names the columns whose values are supplied in the VALUES clause.

In chapter 5, you'll learn more about specifying column names and values. For now, just note that you have to specify a value for a column unless it's a column that allows null values or a column that's defined with a default value.

The two UPDATE statements in this figure show how to change the data in one or more rows of a table. The first statement, for example, assigns a value of 35.89 to the credit_total column of the invoice in the Invoices table with invoice number 367447. The second statement adds 30 days to the invoice due date for each row in the Invoices table whose terms_id column has a value of 4.

To delete rows from a table, you use the DELETE statement. For example, the first DELETE statement in this figure deletes the invoice with invoice number 4-342-8069 from the Invoices table. The second DELETE statement deletes all invoices with a balance due of zero. However, since the Invoices table has a foreign key that references the Invoice_Line_Items table, these DELETE statements won't work unless the invoice doesn't contain any line items. One way to get these DELETE statements to work is to delete the corresponding rows from the Invoice_Line_Items table first.

A statement that adds a row to the Invoices table

```
INSERT INTO invoices
  (vendor_id, invoice_number, invoice_date,
   invoice_total, terms_id, invoice_due_date)
VALUES
  (12, '3289175', '2018-07-18', 165, 3, '2018-08-17')
```

A statement that changes the value of the credit_total column for a selected row in the Invoices table

```
UPDATE invoices
SET credit_total = 35.89
WHERE invoice_number = '367447'
```

A statement that changes the values in the invoice_due_date column for all invoices with the specified terms_id

```
UPDATE invoices
SET invoice_due_date = DATE_ADD(invoice_due_date, INTERVAL 30 DAY)
WHERE terms_id = 4
```

A statement that deletes a selected invoice from the Invoices table

```
DELETE FROM invoices
WHERE invoice_number = '4-342-8069'
```

A statement that deletes all paid invoices from the Invoices table

```
DELETE FROM invoices
WHERE invoice_total - payment_total - credit_total = 0
```

Concepts

- You use the INSERT statement to add rows to a table.
- You use the UPDATE statement to change the values in one or more rows of a table based on the condition you specify.
- You use the DELETE statement to delete one or more rows from a table based on the condition you specify.

Warning

- If you're new to SQL statements, please don't execute the statements above until you read chapter 5 and understand the effect that these statements can have on the database.

Figure 1-14 How to add, update, and delete data in a table

SQL coding guidelines

SQL is a freeform language. That means that you can include line breaks, spaces, and indentation without affecting the way the database interprets the code. In addition, SQL isn't case-sensitive like some languages. That means that you can use uppercase or lowercase letters or a combination of the two without affecting the way the database interprets the code.

Although you can code SQL statements with a freeform style, we suggest that you follow the coding recommendations presented in figure 1-15. The examples in this figure illustrate the value of these coding recommendations. The first example presents an unformatted SELECT statement that's difficult to read. In contrast, this statement is much easier to read after our coding recommendations are applied as shown in the second example.

The third example illustrates how to code a *block comment*. This type of comment is typically coded at the beginning of a group of statements and is used to document the entire group. Block comments can also be used within a statement to describe blocks of code, but that's not common.

The fourth example in this figure includes a *single-line comment*. This type of comment is typically used to document a single statement or line of code. A single-line comment can be coded on a separate line as shown in this example, or it can be coded at the end of a line of code. In either case, the comment is delimited by the end of the line.

Although many programmers sprinkle their code with comments, that shouldn't be necessary if you write your code so it's easy to read and understand. Instead, you should use comments only to clarify sections of code that are difficult to understand. Then, if you change the code, you should be sure to change the comments too. Otherwise, the comments won't accurately represent what the code does, which will make the code even more difficult to understand.

A SELECT statement that's difficult to read

```
select invoice_number, invoice_date, invoice_total,
payment_total, credit_total, invoice_total - payment_total -
credit_total as balance_due from invoices where invoice_total -
payment_total - credit_total > 0 order by invoice_date
```

A SELECT statement that's coded with a readable style

```
SELECT invoice_number, invoice_date, invoice_total,
    payment_total, credit_total,
    invoice_total - payment_total - credit_total AS balance_due
FROM invoices
WHERE invoice_total - payment_total - credit_total > 0
ORDER BY invoice_date
```

A SELECT statement with a block comment

```
/*
Author: Joel Murach
Date: 8/22/2018
*/
SELECT invoice_number, invoice_date, invoice_total,
    invoice_total - payment_total - credit_total AS balance_due
FROM invoices
```

A SELECT statement with a single-line comment

```
-- The fourth column calculates the balance due
SELECT invoice_number, invoice_date, invoice_total,
    invoice_total - payment_total - credit_total AS balance_due
FROM invoices
```

Coding recommendations

- Capitalize all keywords, and use lowercase for the other code in a SQL statement.
- Separate the words in names with underscores, as in invoice_number.
- Start each clause on a new line.
- Break long clauses into multiple lines and indent continued lines.
- Use *comments* only for portions of code that are difficult to understand. Then, make sure that the comments are correct and up-to-date.

How to code a comment

- To code a *block comment*, type /* at the start of the block and */ at the end.
- To code a *single-line comment*, type -- followed by the comment.

Description

- Line breaks, white space, indentation, and capitalization have no effect on the operation of a statement.
- Comments can be used to document what a statement does or what specific parts of the code do. They are not executed by the system.

Figure 1-15 SQL coding guidelines

How to use SQL from an application program

This book teaches you how to use SQL from within the MySQL environment. However, SQL is commonly used from application programs too. So in the topics that follow, you'll get a general idea of how that works.

As you'll see, there's a lot involved in accessing a MySQL database from an application program. That's why most application programmers use a framework that makes it easier to execute SQL statements against a database. In some cases, application programmers create their own framework by writing utility classes and data access classes. In other cases, application programmers use an existing framework that provides the classes they need.

Common options for accessing MySQL data

Figure 1-16 shows three ways to access a MySQL database when you use a programming language to write a custom application. The technique that's used varies depending on the language that's used to develop the application. However, most modern languages provide an API that allows you to connect to a MySQL database.

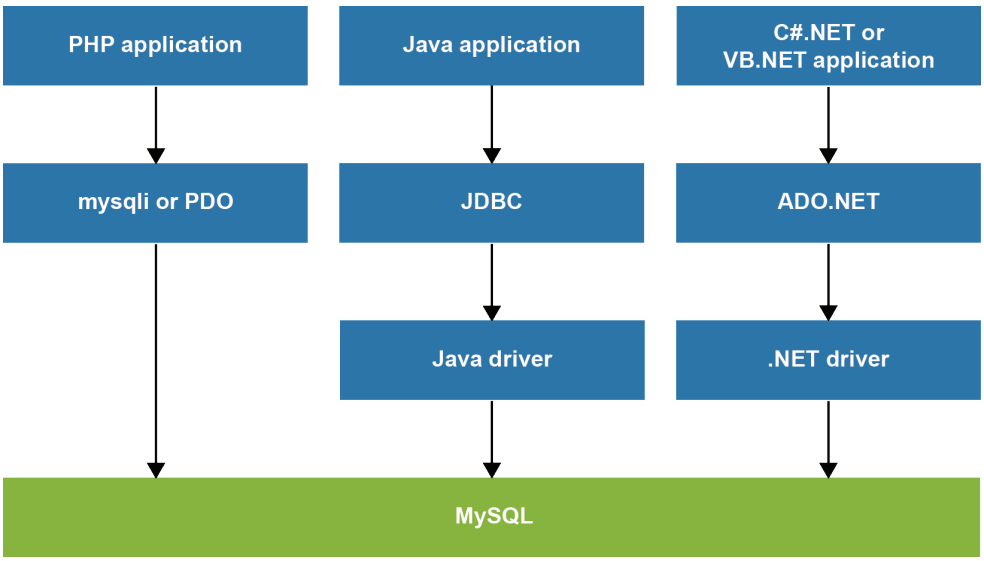
An API uses a piece of software known as a *database driver* to communicate with the database. For some languages, the database driver is built in. For others, you need to download and install a database driver.

To access a MySQL database from a PHP application, for example, you typically choose from two APIs. Some programmers prefer to use the *mysqli* (*MySQL Improved Extension*) API. Other programmers prefer to use the newer *PDO* (*PHP Data Objects*) API. Neither of these APIs requires a database driver, since that driver is typically included as part of the PHP language.

On the other hand, to access a MySQL database from a Java application, you typically use the *JDBC* (*Java Database Connectivity*) API. This API requires a driver to communicate with MySQL. In most cases, you can use the Connector/J driver that's available from the MySQL website to connect a Java application to a MySQL database.

Although it's more common to use MySQL with non-Microsoft languages such as PHP and Java, it's possible to use MySQL with Microsoft .NET languages such as C# and Visual Basic. However, the .NET platform doesn't include a database driver by default, so you typically need to download and install the Connector/Net driver that's available from the MySQL website. Then, you can use the *ADO.NET* API to access a MySQL database.

Common options for accessing MySQL data



Two commonly used MySQL drivers

Name	Description
`Connector/J`	Connects Java applications to a MySQL database.
`Connector/Net`	Connects .NET applications to a MySQL database.

Description

- To work with a MySQL database, an application uses a data access API. For example, PHP uses the *mysqli* API or the *PDO* API, Java uses the *JDBC* API, and .NET languages like C# and Visual Basic use the *ADO.NET* API.
- Most modern programming languages provide an API that you can use to access MySQL.
- Some programming languages include a piece of software known as a *database driver* for the API that it uses to access MySQL. For example, PHP includes a MySQL driver for both the mysqli and PDO APIs. As a result, you typically don't need to install a database driver when you use PHP.
- Some programming languages don't provide a database driver to communicate with a MySQL database. For example, Java doesn't include a MySQL driver for the JDBC API. As a result, you typically need to install a database driver such as the Connector/J driver before you can use Java to access MySQL.

Figure 1-16 Common options for accessing MySQL data

PHP code that retrieves data from MySQL

Figure 1-17 presents PHP code that uses the PDO API to execute a SQL statement against a MySQL database. This code displays information from the Vendors and Invoices tables. It creates the PDO objects used by the application and then uses them to display the data that's retrieved.

If you have some PHP programming experience, you shouldn't have much trouble understanding this code. If you don't have PHP experience, that's fine too. In that case, focus on how this code uses the PDO API to execute SQL against a MySQL database. If you want to learn more about using PHP to work with a database, we recommend *Murach's PHP and MySQL*.

The code in this figure begins by defining a PHP script. Within this script, the first statement stores a SELECT statement in a variable named $query. Then, the next three statements create variables that store the information that's needed to connect to a MySQL database named AP that's running on the same computer as the PHP application. That includes variables that specify a username of "ap_tester" and a password of "sesame".

Here, the same script that created the database also created the ap_tester. This user has limited privileges. In particular, it can only access the AP database, not other databases. In addition, it can only work with the data in the database, not modify the structure of the database. As a result, when writing code, it's more secure to connect to the database as the ap_tester than to connect as a global user such as the root user that has all privileges on all databases.

After specifying the connection information, this code uses these variables to create a PDO object that represents a connection to the database. If this code isn't able to create a PDO object, an error known as a PDOException occurs, and the application displays an error message and ends. Otherwise, this code uses the PDO object to prepare the SELECT statement. Then, it executes that statement, gets all rows from the result set, and stores them in a variable named $rows.

At this point, the HTML tags begin displaying an HTML page. Within the <body> tag, a PHP script loops through each row in the result set and displays that data on the HTML page. In particular, it displays the vendor_name, invoice_number, and invoice_total columns. Here, the PHP function named number_format is used to apply formatting to the invoice_total column.

Although this code may seem complicated, there's only one statement in this figure that uses SQL. That's the statement that specifies the SELECT statement to be executed. Of course, if an application updates data, it can execute INSERT, UPDATE, and DELETE statements as well. With the skills that you'll learn in this book, though, you won't have any trouble coding the SQL statements you need.

PHP code that retrieves data from MySQL

```
<?php
    $query =
        "SELECT vendor_name, invoice_number, invoice_total
         FROM vendors INNER JOIN invoices
             ON vendors.vendor_id = invoices.vendor_id
         WHERE invoice_total >= 500
         ORDER BY vendor_name, invoice_total DESC";

    $dsn = 'mysql:host=localhost;dbname=ap';
    $username = 'ap_tester';
    $password = 'sesame';

    try {
        $db = new PDO($dsn, $username, $password);
    } catch (PDOException $e) {
        $error_message = $e->getMessage();
        echo $error_message;
        exit();
    }

    $statement = $db->prepare($query);
    $statement->execute();
    $rows = $statement->fetchAll();
?>
<!DOCTYPE html>
<html>
    <head>
        <title>DB Test</title>
    </head>
    <body>
        <h1>Invoices with totals over 500:</h1>

        <?php foreach ($rows as $row) : ?>
        <p>
            Vendor: <?php echo $row['vendor_name']; ?><br/>
            Invoice No: <?php echo $row['invoice_number']; ?><br/>
            Total: $<?php echo number_format($row['invoice_total'], 2); ?>
        </p>
        <?php endforeach; ?>

    </body>
</html>
```

Note

- For this code to run correctly with MySQL 8.0 or later, you must be using a recent version of PHP that includes a PDO driver that supports MySQL 8.0's new default authentication plug-in (caching_sha2_password), or you must use MySQL's older authentication plug-in (mysql_native_password).

Figure 1-17 PHP code that retrieves data from MySQL

Java code that retrieves data from MySQL

Figure 1-18 presents Java code that uses the JDBC API to execute a SQL statement against a MySQL database. This code displays information from the Vendors and Invoices tables.

If you have some Java programming experience, you shouldn't have much trouble understanding this code. If you don't have Java experience, that's fine too. In that case, focus on how this code uses an API to execute SQL against a MySQL database. If you want to learn more about using Java to work with a database, we recommend *Murach's Java Programming* and *Murach's Java Servlets and JSP*.

Before this code can be executed, a database driver must be installed. To do that, you can download the Connector/J database driver from the MySQL website. Then, you can add the JAR file for that driver to the libraries that are available to your application.

The code in this figure begins by importing all classes in the java.sql package. These classes define JDBC objects like the Connection object that are used to access a MySQL database.

Within the main method, the first statement stores a SQL SELECT statement in a variable named query. Then, the next three statements create variables that store the information that's needed to connect to a MySQL database named AP that's running on the same computer as the Java application on port 3306. That includes variables that specify a username of "ap_tester" and a password of "sesame".

Like the previous figure, the code in this figure connects as the ap_tester instead of the root user for security reasons. For more information about the ap_tester, please refer to the previous figure.

After specifying the connection information, this code uses a try-with-resources statement to create the Connection, PreparedStatement, and ResultSet objects that are needed to display the data. Since the try-with-resources statement was introduced with Java SE 7, it won't work with earlier versions of Java. If this statement isn't able to create these objects, an error known as a SQLException occurs, and the application prints an error message and ends. Otherwise, this code uses the Connection and PreparedStatement objects to execute the SELECT statement, and it stores the result set in a ResultSet object.

Next, this code uses the get methods of the ResultSet object to retrieve the values that are stored in the vendor_name, invoice_number, and invoice_total columns. Here, the getString method is used to get the VARCHAR data and the getDouble method is used to get the DECIMAL data. Finally, the NumberFormat class is used to apply currency formatting to the invoice_total column, and the values are printed to the console.

Java code that retrieves data from MySQL

```
package murach.ap;

import java.sql.*;
import java.text.NumberFormat;

public class DBTestApp {

    public static void main(String args[]) {
        String query
                = "SELECT vendor_name, invoice_number, invoice_total "
                + "FROM vendors INNER JOIN invoices "
                + "    ON vendors.vendor_id = invoices.vendor_id "
                + "WHERE invoice_total >= 500 "
                + "ORDER BY vendor_name, invoice_total DESC";

        String dbUrl = "jdbc:mysql://localhost:3306/ap";
        String username = "ap_tester";
        String password = "sesame";

        // define common JDBC objects
        try (Connection connection = DriverManager.getConnection(
                    dbUrl, username, password);
             PreparedStatement ps = connection.prepareStatement(query);
             ResultSet rs = ps.executeQuery()) {

            // Display the results of a SELECT statement
            System.out.println("Invoices with totals over 500:\n");
            while (rs.next()) {
                String vendorName = rs.getString("vendor_name");
                String invoiceNumber = rs.getString("invoice_number");
                double invoiceTotal = rs.getDouble("invoice_total");

                NumberFormat currency = NumberFormat.getCurrencyInstance();
                String invoiceTotalString = currency.format(invoiceTotal);

                System.out.println(
                        "Vendor:     " + vendorName + "\n"
                      + "Invoice No: " + invoiceNumber + "\n"
                      + "Total:      " + invoiceTotalString + "\n");
            }
        } catch (SQLException e) {
            System.out.println(e.getMessage());
        }
    }
}
```

Description

- Before you can use Java to work with MySQL, you must install a database driver. To do that, you can download the JAR file for the driver and add it to the libraries that are available to your Java application.
- To execute a SQL statement from a Java application, you can use JDBC objects such as the Connection, PreparedStatement, and ResultSet objects.

Figure 1-18 Java code that retrieves data from MySQL

Perspective

To help you understand how SQL is used from an application program, this chapter has introduced you to the hardware and software components of a client/server system. It has also described how relational databases are organized and how you use some of the SQL statements to work with the data in a relational database. With that as background, you're now ready to start using MySQL. In the next chapter, then, you'll learn how to use some of the tools for working with a MySQL database.

Terms

client
server
database server
network
client/server system
local area network (LAN)
enterprise system
wide area network (WAN)
database management system (DBMS)
back end
application software
API (application programming interface)
data access API
JDBC (Java Database Connectivity)
front end
SQL (Structured Query Language)
query
query results
application server
web server
business component
web application
web service
web browser
thin client
relational database
table
row
column
record
field
cell
value
primary key
composite primary key
non-primary key
unique key
index
foreign key
one-to-many relationship
one-to-one relationship
many-to-many relationship
referential integrity
data type
null value
default value
auto increment column
entity-relationship (ER) diagram
enhanced entity-relationship (EER) diagram
relational database management system (RDBMS)
SQL dialect
SQL extension
open-source database
data manipulation language (DML)
data definition language (DDL)
database administrator (DBA)
constraint
base table
result set
calculated value
join
inner join
outer join
comment
block comment
single-line comment
database driver
mysqli
PDO
ADO.NET

2

How to use MySQL Workbench and other development tools

In the last chapter, you learned about some of the SQL statements that you can use to work with the data in a relational database. Before you learn the details of coding these statements, however, you need to learn how to use MySQL Workbench to enter and execute SQL statements. In addition, you should learn how to use the MySQL Reference manual, and you should at least be familiar with the MySQL Command Line Client.

An introduction to MySQL Workbench

MySQL Workbench is a free graphical tool that makes it easy to work with MySQL. We recommend using this tool as you work through this book. This chapter shows how to work with version 8.0. However, with some minor variations, the skills presented in this chapter should work for later versions as well.

The Home page of MySQL Workbench

When you start MySQL Workbench, it displays its Home page as shown in figure 2-1. This page is divided into three tabs: Welcome, Models, and Migration.

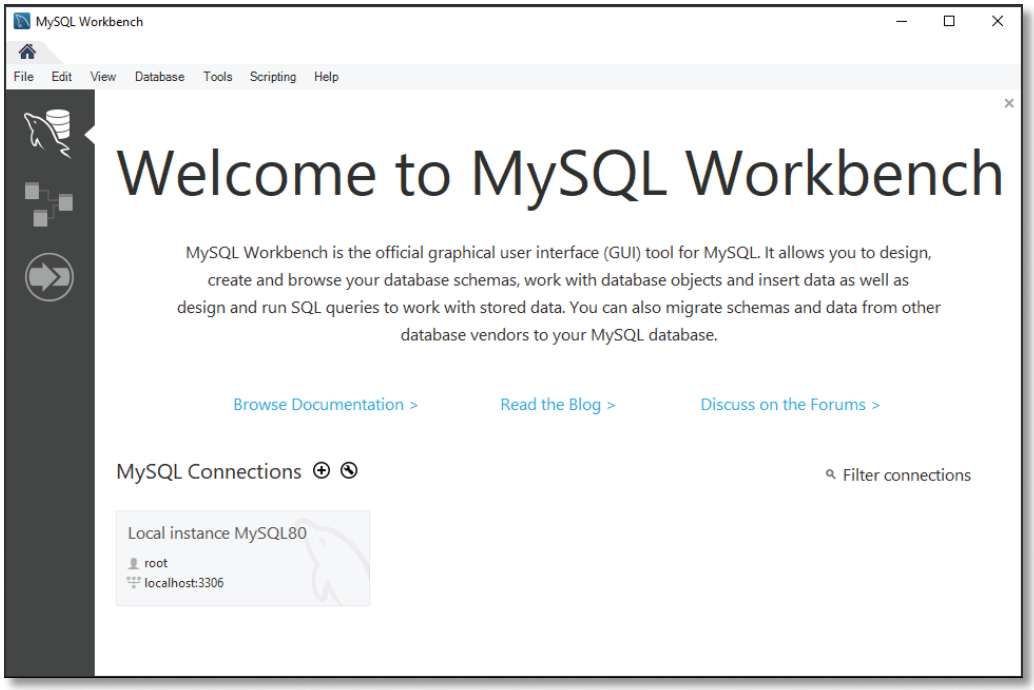
The MySQL Connections section of the Welcome tab contains links that you can use to open a connection to a MySQL server. Then, you can use that connection to code and run SQL statements. By default, this tab contains one connection that allows you to connect as the root user to a MySQL server that's running on the local computer. In this book, this is the only connection you will need. However, if necessary, you can click the ⊕ icon to the right of MySQL Connections to create other connections.

The Welcome tab also contains links to MySQL Workbench documentation, blogs, and forums. This book doesn't show how to use these links, but you may find them useful, especially after you have learned the basic skills for working with MySQL that are described in this book.

The Models tab contains links that let you create a database diagram from a type of data model known as an EER model. You can also use this tab to open existing EER models or to create new ones. Then, you can work with EER diagrams that correspond with these models. To learn more about this, you can read chapter 10.

You can return to the Home page by clicking on the tab with the house icon on it near the top left corner of the Workbench window. In this figure, the Home tab is the only tab that's shown, but you'll see some other tabs in the next few figures.

The Home page of MySQL Workbench



Description

- The Home page of MySQL Workbench is divided into three tabs displayed at the left side of the window: Welcome, Models, and Migration.
- You can use the MySQL Connections section of the Welcome tab to start and stop the database server and to code and run SQL statements.
- You can use the links on the Welcome tab to view the documentation for using MySQL Workbench, view the MySQL Workbench blog, and view and join in the MySQL Workbench forum.
- You can use the Models tab to create and work with EER models.
- You can use the Migration tab to migrate other databases to MySQL and to copy a database from one instance of MySQL to another.
- You can return to the Home page by clicking the tab with the house icon. This tab is always displayed in the top left corner of the Workbench window.

Note

- In some cases, you'll get an "Unsupported Operating System" message when you start MySQL Workbench. This happens, for example, when you start MySQL Workbench 8.0 on Windows 7. If you click the OK button when this message is displayed, MySQL Workbench should work fine. This is a known bug that should be fixed in a future release of Workbench.

Figure 2-1 The Home page of MySQL Workbench

How to open a database connection

Before you can work with a database, you need to connect to the database server. When you start MySQL Workbench, the MySQL Connections section displays a list of saved connections.

By default, MySQL Workbench has one saved connection in this list. This connection is named "Local instance MySQL80", and it connects as the root user to a MySQL server that's running on port 3306 of the local host computer. (This assumes that you're using MySQL version 8.0. If you're using another version, the number at the end of the connection name will be different.)

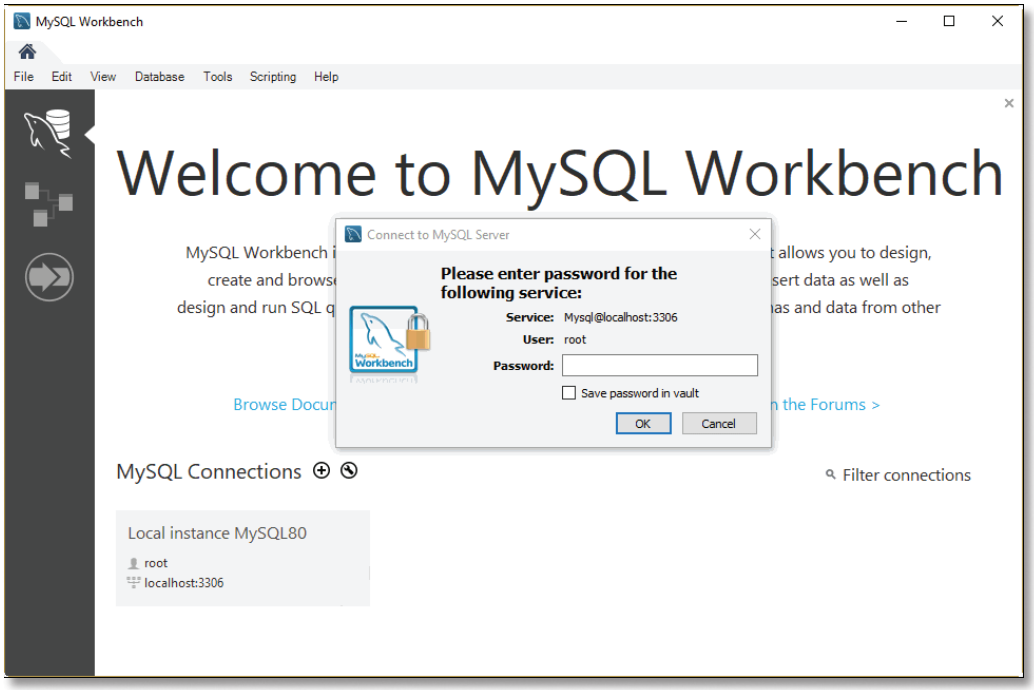
Since this is what you want when you're first getting started, you typically use this connection to connect to the server. To do that, click the connection and enter the password for the root user if you're prompted for it. If you installed MySQL Workbench following the directions in appendix A (Windows) or B (macOS), the password for the root user is "sesame80."

Figure 2-2 shows the dialog box that MySQL Workbench displays to prompt for a password. This dialog box shows that it's attempting to use the root user to connect to a MySQL server running on port 3306 of the local host. In addition to entering a password in this dialog box, you can select the "Save password in vault" option to save the password so you don't have to enter it every time you connect to this server. Then, if you ever want to clear the password from the vault, you can right-click the connection, select the Edit Connection item, and click the Clear button.

If you need to connect as another user, or if you need to connect to a MySQL server running on a different computer, you can use MySQL Workbench to edit the connection parameters for a connection. To do that, right-click the connection and select the Edit Connections item. This displays a dialog box that lets you specify the parameters for the connection such as the username, hostname, and port number.

If you want to add a new connection to the Home tab, you can click the ⊕ icon to the right of MySQL Connections, enter a name for the connection, and specify the parameters for the connection. Then, this connection appears in the list of connections, and you can click it to use it.

The dialog box for opening database connections



Description

- To connect as the root user to an instance of MySQL that's running on the local host computer, click the stored connection named "Local instance MySQL80", and enter the password for the root user if prompted.
- To save the password for a connection so you don't have to enter it every time, check the "Save password in vault" option when you're prompted for your password.
- To clear the password from the vault so you are prompted for your password, right-click the connection, select the Edit Connection item, click the Clear button for the password, and click the Close button.
- To edit the connection parameters for a connection, right-click the connection, select the Edit Connection item, enter the connection parameters, and click the Close button. This lets you specify the username, the host address, the port number, and other connection parameters.
- To add a new connection to the Welcome tab of the Home page, click the ⊕ icon to the right of MySQL Connections, enter the connection parameters, and click the OK button. Then, the connection appears in the list of connections.

Figure 2-2 How to open a database connection

How to start and stop the database server

If you installed MySQL on your computer as described in appendix A (Windows) or B (macOS), the *database server* starts automatically when you start your computer. This piece of software is sometimes referred to as the *database service* or *database engine*. It receives SQL statements that are passed to it, processes them, and returns the results.

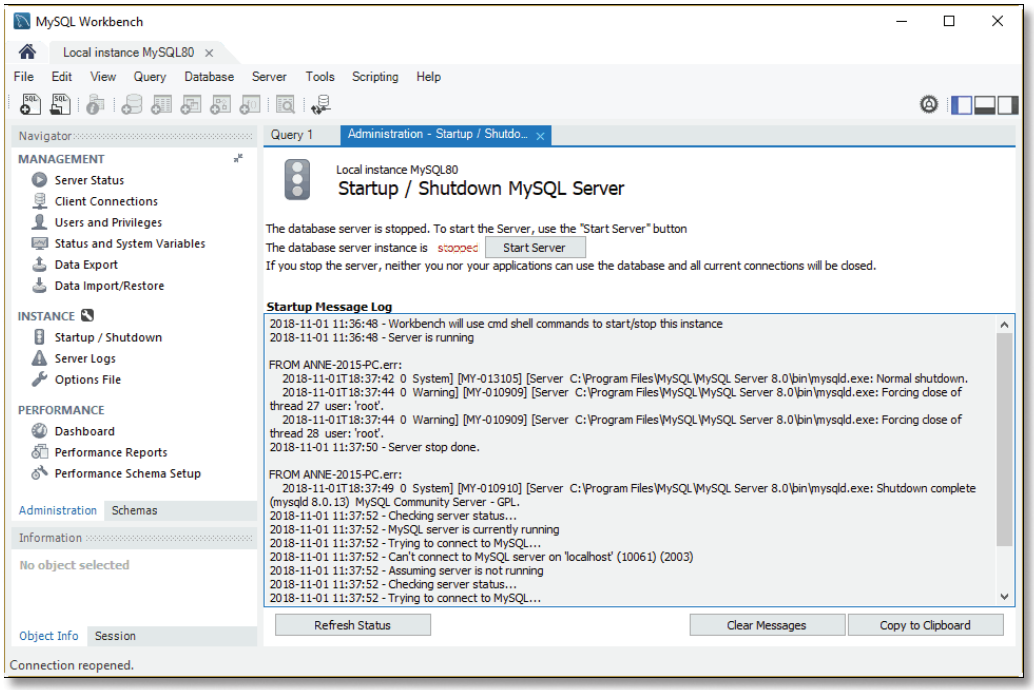
Before you can work with a MySQL database, the database server must be started. To check whether the MySQL database server is running on your computer, you can use the Startup/Shutdown option of MySQL Workbench as shown in figure 2-3. Then, if the server isn't already running, you can start it by clicking on the Start Server button. When you do that, MySQL Workbench displays a message that indicates the status of the MySQL server, and it displays the Stop Server button.

You may also want to stop the database server from time to time. For example, you can stop the server if you aren't going to be using it and you want to free the resources on your computer. Or, you can stop the server if the port that is being used by the MySQL database server conflicts with another program. Then, when you want to work with the database server again, you can start it.

The easiest way to stop the database server is to use the Stop Server button that's available from the Startup/Shutdown option of the Navigator window of MySQL Workbench as described in this figure. When you click this button, MySQL Workbench displays a message when the MySQL server has successfully stopped, and it displays the Start Server button.

When you're running the MySQL database server on your own computer for training purposes, you can stop the database server whenever you want. However, if a database server is running in a production environment, you should make sure that all users are logged off and that no applications are using the database server before you stop it.

The Startup/Shutdown option of MySQL Workbench



How to stop and start the database server

1. Display the Welcome tab of the MySQL Workbench Home page.
2. Click the connection to the local server. This should connect you to the local MySQL server as the root user. If necessary, enter the password for the root user.
3. In the Navigator window, if necessary, click on the Administration tab. Then, select the Startup/Shutdown option from the Instance category.
4. Click the Stop Server button to stop the database server. Or, click the Start Server button to start it.

Description

- After you install MySQL, the *database server* usually starts automatically each time you start your computer.
- The database server can also be referred to as the *database service* or the *database engine*.
- If you aren't able to use Workbench to start and stop the database server, you may need to edit your connection so it points to the correct instance of MySQL. To do that, right-click the connection on the Welcome tab of the Home page, select the Edit Connection item, click the System Profile tab, and edit the service name. For MySQL 8.0 on Windows, the service name is typically "MySQL80".

Figure 2-3 How to start and stop the database server

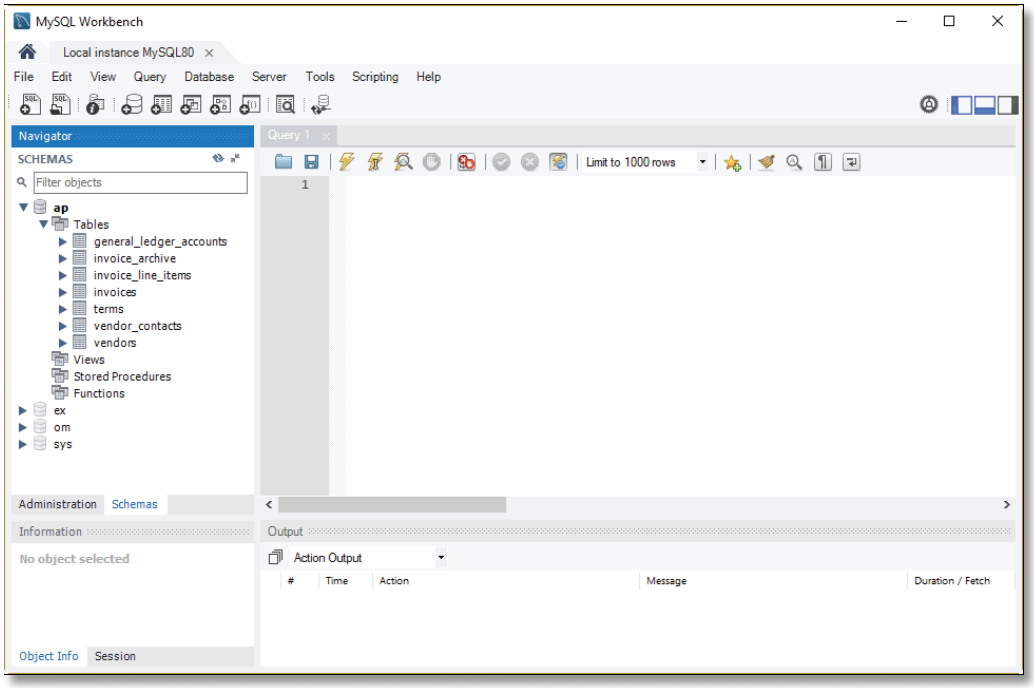
How to navigate through the database objects

After you connect to a database server, you can use the Schemas category of the Navigator window to navigate through the *database objects* in the databases on the server, as shown in figure 2-4. As you can see, these objects include tables, views, stored procedures, and functions. For this chapter, however, you can focus on the tables. Later in this book, you'll learn more about views, stored procedures, and functions.

In this figure, I double-clicked the node for the AP database (*schema*) in the Schemas tab of the Navigator window to select it and view the database objects it contains (tables, views, stored procedures, and functions). Then, I expanded the Tables node to view all of the tables in the AP database.

To work with a node or an object, you can right-click it to display a context-sensitive menu. Then, you can select a command from that menu. For example, you can right-click the node for the AP database to display a list of commands for working with that database.

The tables available for the AP database



Description

- Each database (or *schema*) provides access to the *database objects* that are available. These database objects include tables, views, stored procedures, and functions.
- On some systems, the Navigator window provides Administration and Schemas tabs that you can use to display the Administration and Schemas categories. On other systems, the Navigator window displays the Administration category above the Schemas category.
- To display the databases for the current connection, you can use the Navigator window to view the Schemas category.
- To navigate through the database objects for a database, click the arrows to the left of each of the nodes in the Navigator window to expand or collapse the node.
- To work with a node or an object, right-click the node or object and select a command from the resulting menu.

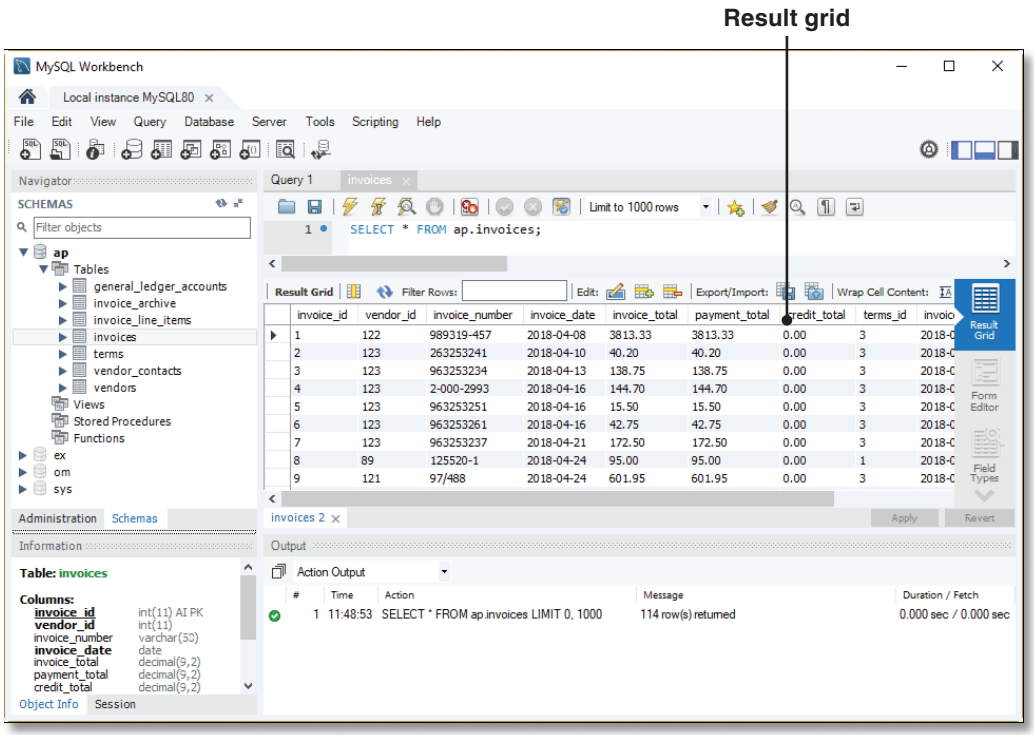
Figure 2-4 How to navigate through the database objects

How to view and edit the data for a table

To view the data for a table, you can right-click the table name and select the Select Rows - Limit 1000 command. In figure 2-5, for example, I selected this command for the Invoices table. This displayed the data for the table in a Result grid. In addition, it displayed information about the SELECT statement that was used to retrieve the data in the Output tab.

To insert, edit, and delete the rows in the table, you can use the buttons at the top of the Result grid. Then, to apply the changes to the table, you can click the Apply button at the bottom of the Result grid. Or, if you want to cancel the changes, you can click the Revert button.

The data for the Invoices table displayed in the Result grid



Description

- To view the data for a table, right-click the table in the Navigator window and select the Select Rows - Limit 1000 command to display it in a Result grid.
- To edit the data for a table, view the data. Then, you can use the buttons at the top of the Result grid to insert, update, and delete rows.
- To apply the changes to the table, click the Apply button at the bottom of the tab. To cancel the changes, click the Revert button.

Figure 2-5 How to view and edit the data for a table

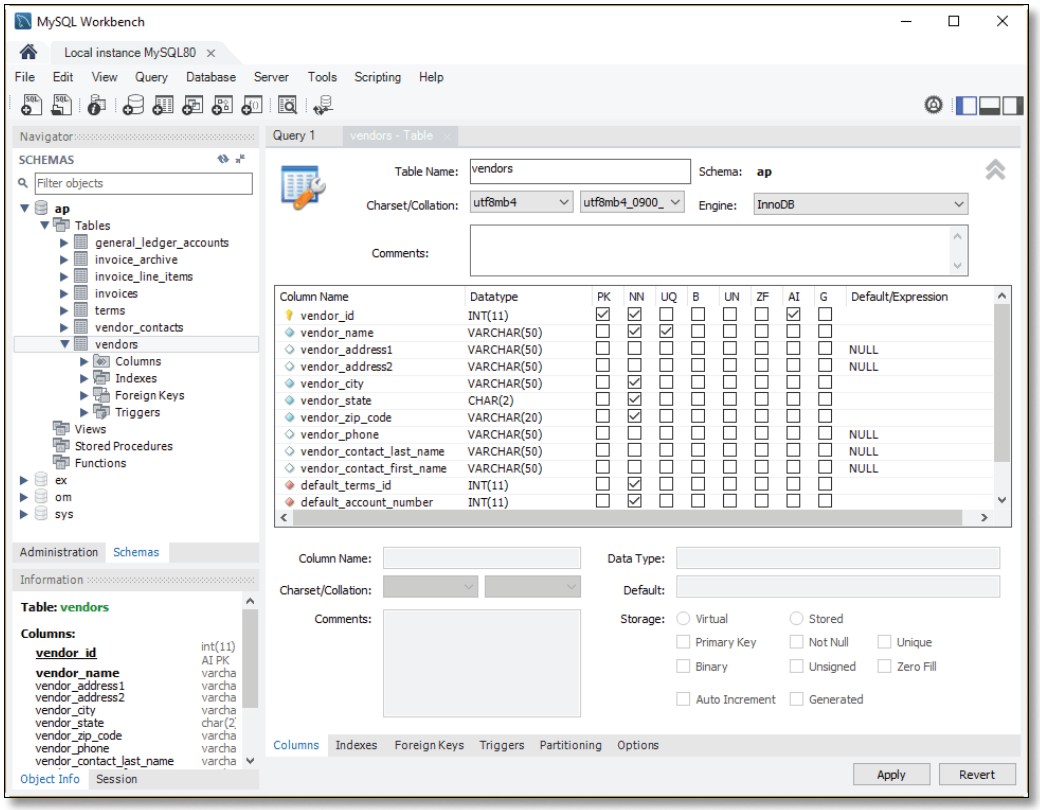
How to view and edit the column definitions for a table

If you want to edit a column definition for a table, you can use the technique described in figure 2-6 to display the column definitions for the table. In this figure, for example, the column definitions for the Vendors table are displayed. At this point, you can view information about each column of the table such as its name and data type.

Once you display the column definitions for a table, you can use the Columns tab to add a column, delete a column, or modify a column. For example, you can add a new column by entering it at the bottom of the list. You can delete a column by right-clicking on it and selecting the Delete command. You can change the name of a column by selecting the column and then clicking on the name and editing it. You can change the data type of a column by selecting the column and then clicking on its data type and selecting another data type from the drop-down list that appears. And so on.

Most of the time, you won't want to use MySQL Workbench to edit the column definitions for a table. Instead, you'll want to edit the scripts that create the database so you can easily recreate the database later. In chapter 11, you'll learn more about creating and modifying the column definitions for a table using both techniques.

The column definitions for the Vendors table



Description

- To view the column definitions for a table, right-click the table name in the Navigator window and select the Alter Table command. Then, select the Columns tab at the bottom of the window that's displayed to view the column definitions for the table.
- To edit the column definitions for a table, view the column definitions. Then, you can use the resulting window to add new columns and modify and delete existing columns.
- For more information about creating and modifying tables, see chapter 11.

Figure 2-6 How to view and edit the column definitions

How to use MySQL Workbench to run SQL statements

Besides letting you review the design of a database, MySQL Workbench is a great tool for entering and running SQL statements.

How to enter and execute a SQL statement

When you first connect to a MySQL server in MySQL Workbench, a SQL Editor tab is automatically opened. Figure 2-7 shows how to use the SQL editor to enter and execute a SQL statement. The easiest way to open a SQL Editor tab is to click the Create New SQL Tab button in the SQL Editor toolbar or press the Ctrl+T keys.

Once you open a SQL tab, you can use standard techniques to enter or edit a SQL statement. As you enter statements, you'll notice that MySQL Workbench automatically applies colors to various elements. For example, it displays keywords in blue. This makes your statements easier to read and understand and can help you identify coding errors.

To execute a single SQL statement like the one in this figure, you can press Ctrl+Enter or click the Execute Current Statement button in the SQL Editor toolbar. If the statement returns data, that data is displayed below the SQL editor in a corresponding Result grid. In this figure, for example, the result set returned by the SELECT statement is displayed. If necessary, you can adjust the height of the Result grid by dragging the bar that separates the SQL Editor tab from the Result grid.

Before you execute a SQL statement, make sure you've selected a database by double-clicking the database in the Navigator window. Otherwise, you'll get an error message like this:

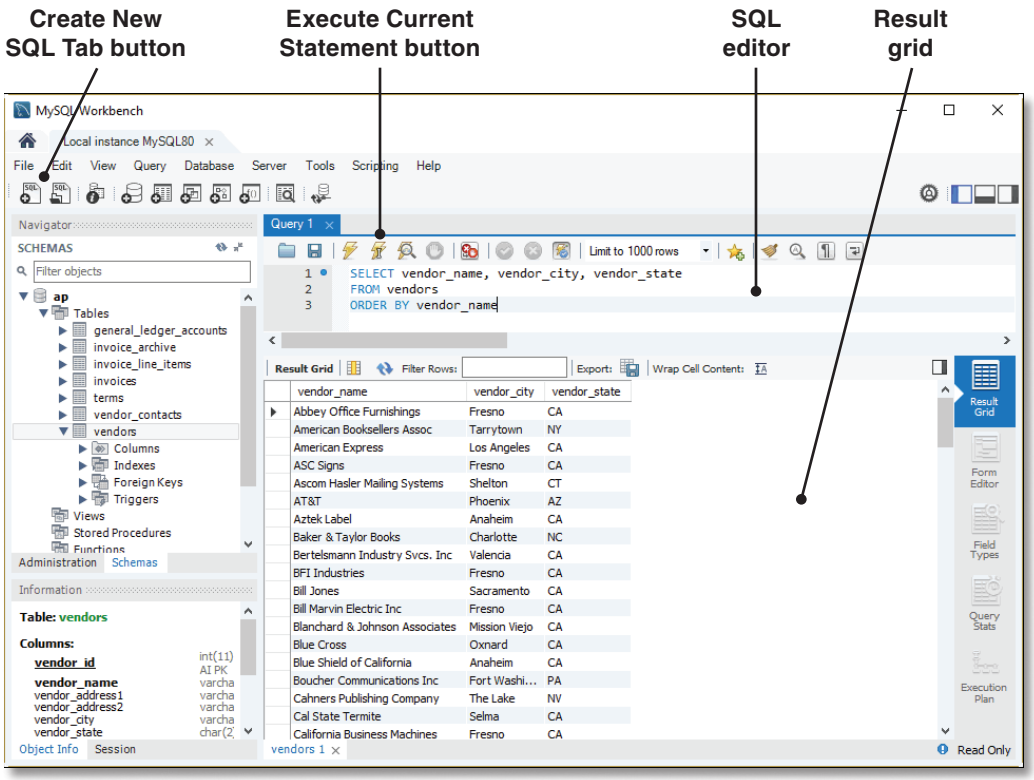
```
Error Code: 1046. No database selected
```

Similarly, if you haven't selected the correct database, you'll get an error message that says the table doesn't exist. For example, if the EX database is selected when you attempt to retrieve data from the Vendors table, you'll get an error message like this:

```
Error Code: 1146. Table 'ex.vendors' doesn't exist
```

To fix this, you can double-click the AP database to select it.

A SELECT statement and its results



Description

- To open a new SQL tab, press Ctrl+T or click the Create New SQL Tab button ( ) in the SQL Editor toolbar.
- To select the current database, double-click it in the Schemas tab of the Navigator window. This displays the selected database in bold.
- To enter a SQL statement, type it into the SQL editor.
- As you enter the text for a statement, the SQL editor applies color to various elements, such as SQL keywords, to make them easy to identify.
- To execute a SQL statement, press Ctrl+Enter or click the Execute Current Statement button ( ) in the SQL Editor toolbar. If the statement retrieves data, the data is displayed in a Result grid below the SQL tab.

Figure 2-7 How to enter and execute a SQL statement

How to use snippets

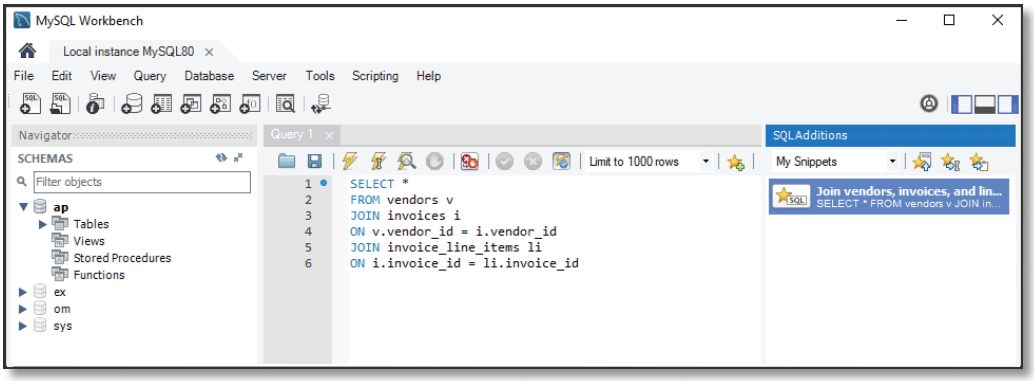
You can think of the *snippets* that come with MySQL Workbench as a library of SQL syntax. This library is divided into statements that you can use to manage a database, define objects in a database, and manipulate the data in a database. You can also create your own snippets that provide custom code. In fact, you're more likely to create your own snippets than you are to use the built-in snippets. That's because the syntax that's provided for the built-in snippets is much more complex than what you typically need.

Figure 2-8 shows how to use snippets. To start, if the SQL Additions window isn't displayed, you can display it by clicking on the rightmost button at the right side of the SQL Editor tab. Then, you can display the snippets tab and use the drop-down list at the top of the tab to select a category of snippets. In this figure, for example, the My Snippets category is displayed. From here, you can select a snippet and then click the Insert Snippet button to enter the snippet into the SQL Editor tab. Finally, you can edit the snippet code so it's appropriate for your SQL statement.

In this figure, the snippet contains code that I wrote for joining the vendors, invoices, and invoice_line_items tables. To create this snippet, I entered it into a SQL Editor tab and then clicked the Add New Snippet button. By saving this statement as a snippet, I can now use it anytime I want to join these three tables instead of having to type it each time.

For now, don't worry if you don't understand the SQL statement presented in this figure. The main point is that you can use the Snippets tab to save and retrieve a variety of SQL code. As you learn more about SQL statements, you'll see how useful this can be.

The SQL Additions tab with a snippet created by a user



Description

- The SQL Additions window contains context help and *snippets*. Snippets contain the syntax for many common SQL statements. You can use the snippets to guide you as you create a SQL statement. You can also create your own snippets and save them for later use.
- The SQL Additions window is displayed to the right of the SQL Editor tab by default. If this window isn't displayed, you can display it by clicking the rightmost button ( ) at the right side of the SQL Editor toolbar. Then, you can click the Snippets tab to display the available snippets.
- The snippets are organized into categories. To display any category of snippets, select the category from the drop-down list at the top of the Snippets tab.
- To enter a snippet into a SQL editor, select the snippet and then click the Insert Snippet button ( ) at the top of the Snippets tab. Then, edit the snippet code so it's appropriate for your SQL statement.
- To replace code in the SQL editor with a snippet, select the code, select the snippet you want to replace it with, and then click the Replace Current Text button ( ).
- To create your own snippet, enter the code for the snippet into a SQL editor tab. Then, select the category where you want to save the snippet, click the Save Snippet button ( ) in the SQL Editor toolbar, and enter a name for the snippet.
- To delete a snippet, right-click it in the Snippets tab and select the Delete Snippet item.

Figure 2-8 How to use the Snippets tab

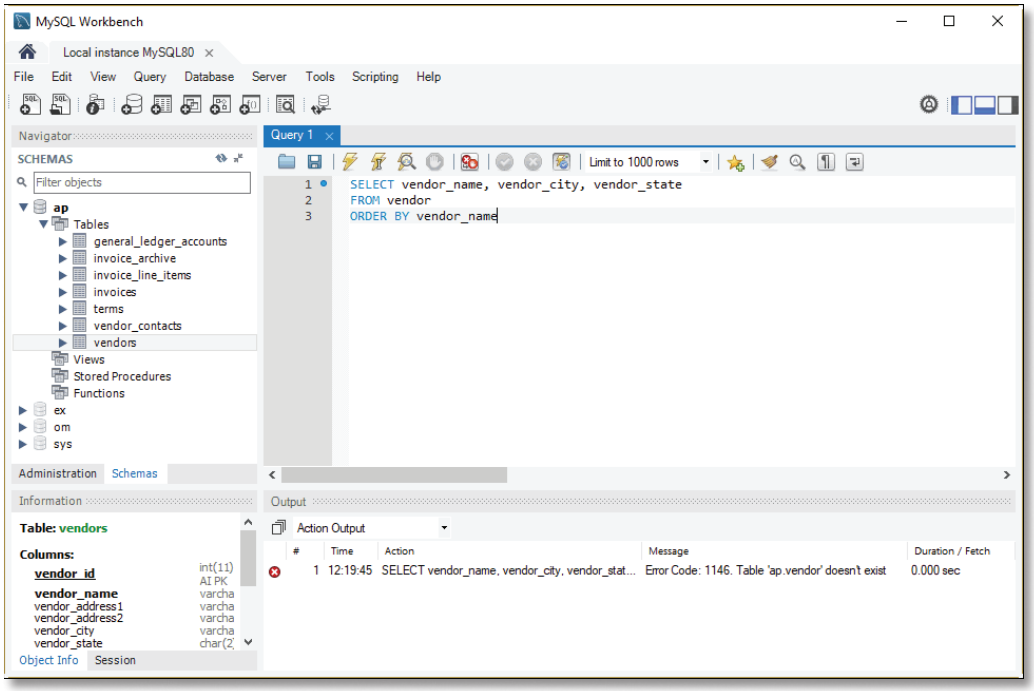
How to handle syntax errors

If an error occurs during the execution of a SQL statement, MySQL Workbench displays a message that includes the error number and a brief description of the error. In figure 2-9, for example, the message displays an error number of 1146 and a brief description that says "Table ap.vendor doesn't exist."

In this example, the problem is that the Vendor table doesn't exist in the database. To fix the problem, you need to edit the SQL statement so the table is Vendors instead of Vendor. Then, you should be able to successfully run the SQL statement.

This figure also lists some other common causes of errors. As you can see, most errors are caused by incorrect syntax. However, it's also common to get an error if you have selected the wrong database. If, for example, you have selected the EX database and you try to run a statement that refers to tables in the AP database, you will get an error. Regardless of what's causing the problem, you can usually identify and correct the problem without much trouble. In some cases, though, it may be difficult to figure out the cause of an error. Then, you can usually get more information about the error by searching the Internet or by searching the MySQL Reference Manual as described later in this chapter.

How to handle syntax errors



Common causes of errors

- Having the wrong database selected
- Misspelling the name of a table or column
- Misspelling a keyword
- Omitting the closing quotation mark for a character string

Description

- If an error occurs during the execution of a SQL statement, MySQL Workbench displays a message in the Output tab that includes an error code and a brief description of the error.
- Most errors are caused by incorrect syntax and can be corrected without any additional assistance. Otherwise, you can usually get more information about an error by searching for the error code or description in the MySQL Reference Manual or on the Internet.

Figure 2-9 How to handle syntax errors

How to open and save SQL scripts

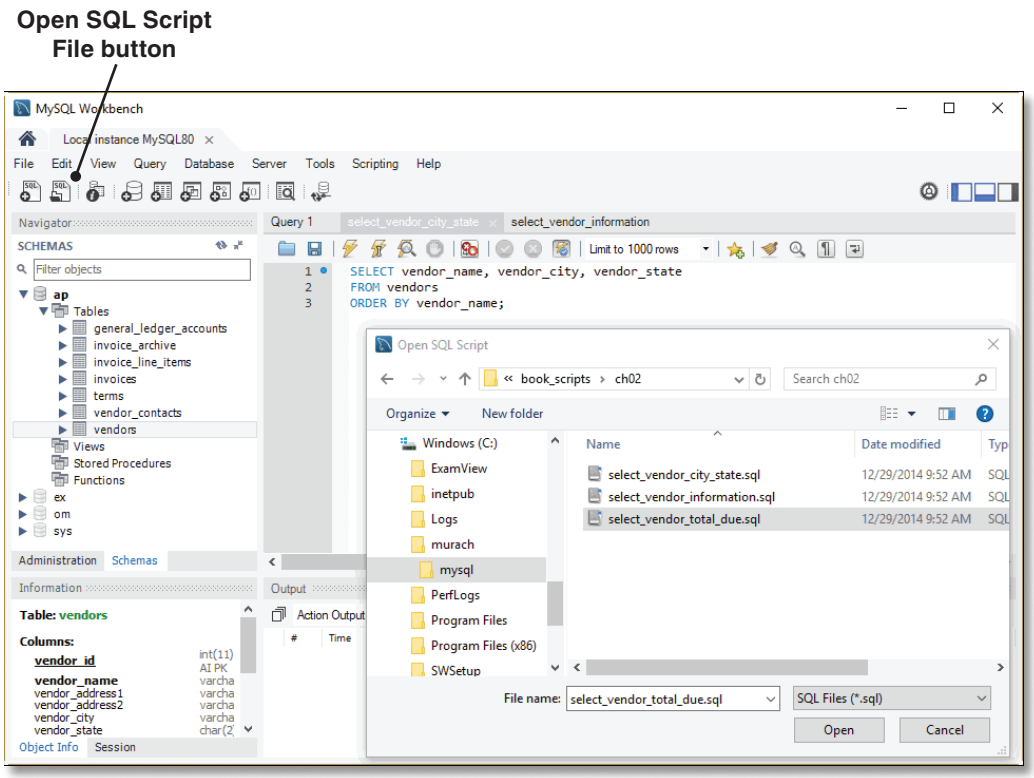
In MySQL, a *script* is a file that contains one or more SQL statements. To create a script, you enter the statements you want it to include into a SQL Editor tab. You'll learn more about that in the next figure. Then, you can click the Save button or press Ctrl+S to save the script as described in figure 2-10.

Once you've saved a script, you can open it later. To do that, you can click the Open SQL Script File button in the SQL Editor toolbar, or you can press Ctrl+Shift+O. In this figure, the dialog box that's displayed shows the script files that have been saved for chapter 2. These files are created when you download and install the source code for this book. Note that the names of these files have the .sql extension. (If you're using Windows 10 and the file extensions aren't displayed, you can display them by opening the File Explorer, displaying the View tab, and selecting the "File name extensions" option in the Show/hide group.)

Once you open a script, you can run it as shown in the next figure. You can also use it as the basis for a new SQL script. To do that, just modify it any way you want. Then, you can save it as a new script by pressing the Ctrl+Shift+S keys or selecting the File→Save Script As command.

The screen in this figure shows the tabs for two script files that have been opened. After you open two or more scripts, you can switch between them by clicking on the appropriate tab. Then, you can cut, copy, and paste code from one script to another.

The Open SQL Script dialog box



Description

- A *SQL script* is a file that contains one or more SQL statements.
- To open a file that contains a SQL script, click the Open SQL Script File button in the SQL Editor toolbar or press the Ctrl+Shift+O keys. Then, use the Open SQL Script dialog box to locate and open the SQL script.
- When you open a SQL script, MySQL Workbench displays it in its own SQL Editor tab. To switch between open scripts, select the appropriate tab.
- To cut, copy, and paste code from one SQL script to another, use the standard techniques.
- To save a SQL statement to a script file, click the Save button in the SQL Editor toolbar or press Ctrl+S. Then, use the Save SQL Script dialog box to specify a location and name for the file.
- To save a script you've modified to a new file, press the Ctrl+Shift+S keys or select the File→Save Script As command.

Figure 2-10 How to open and save SQL scripts

How to enter and execute SQL scripts

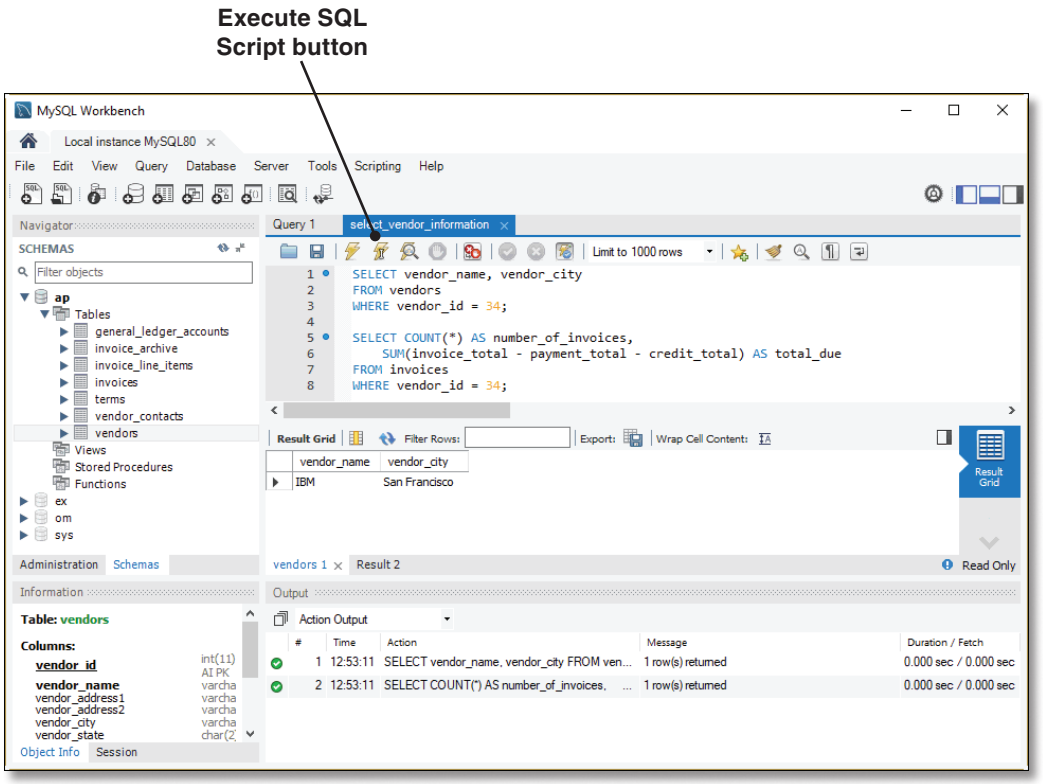
In the last topic, you saw a SQL script that contained a single SQL statement. However, a SQL script typically contains multiple statements. Figure 2-11 shows how to enter and execute scripts like that.

When you code multiple SQL statements within a script, you must code a semicolon at the end of each statement. For example, this figure shows a script that contains two SELECT statements. To execute both of these statements, you can press the Ctrl+Shift+Enter keys, or you can click the Execute SQL Script button in the SQL Editor toolbar. When you do, the results of each script are displayed in a separate Result grid. To switch between Result grids, you can click on the tabs that are displayed below the current Result grid.

If you want to execute a single SQL statement that's stored within a script, you can do that by moving the insertion point into the statement and pressing the Ctrl+Enter keys or clicking the Execute Current Statement button. Then, if the statement retrieves data, the data is displayed in a single Result grid.

If you need to, you can also execute two or more statements in a script. To do that, you select the statements and then press the Ctrl+Shift+Enter keys or click the Execute SQL Script button. This is useful if a script contains many statements and you just want to execute some of them.

A SQL script and its results



Description

- When you code a script that contains more than one statement, you must code a semicolon at the end of each statement.
- To run an entire SQL script, press the Ctrl+Shift+Enter keys or click the Execute SQL Script button ( ) that's located just to the left of the Execute Current Statement button in the SQL Editor toolbar.
- When you run a SQL script, the results of each statement that returns data are displayed in a separate Result grid. To switch between these Result grids, you can click on the tabs that are displayed below the current Result grid.
- To execute one SQL statement within a script, move the insertion point into that statement and press the Ctrl+Enter keys or click the Execute Current Statement button ( ). If the statement retrieves data, the data is displayed in a Result grid.
- To execute two or more statements within a script, select them in the editor and then press the Ctrl+Shift+Enter keys or click the Execute SQL Script button.

Figure 2-11 How to enter and execute SQL scripts

How to use the MySQL Reference Manual

Figure 2-12 shows how to use another useful tool for working with the MySQL database: the *MySQL Reference Manual*. In most cases, you'll use a web browser to view this manual directly from the Internet. That way, you can be sure that the information is always up-to-date. However, you can also download this manual and save it on your hard drive. Either way, you can use the MySQL Reference Manual to quickly look up detailed technical information about the MySQL database, including information about SQL statements and functions.

How to view the manual

You can view the MySQL Reference Manual by using a web browser to go to the web address shown at the top of this figure. Here, the Reference Manual for version 8.0 of MySQL is displayed. However, you can easily select another version by selecting it from the drop-down list at the right side of the page.

How to look up information

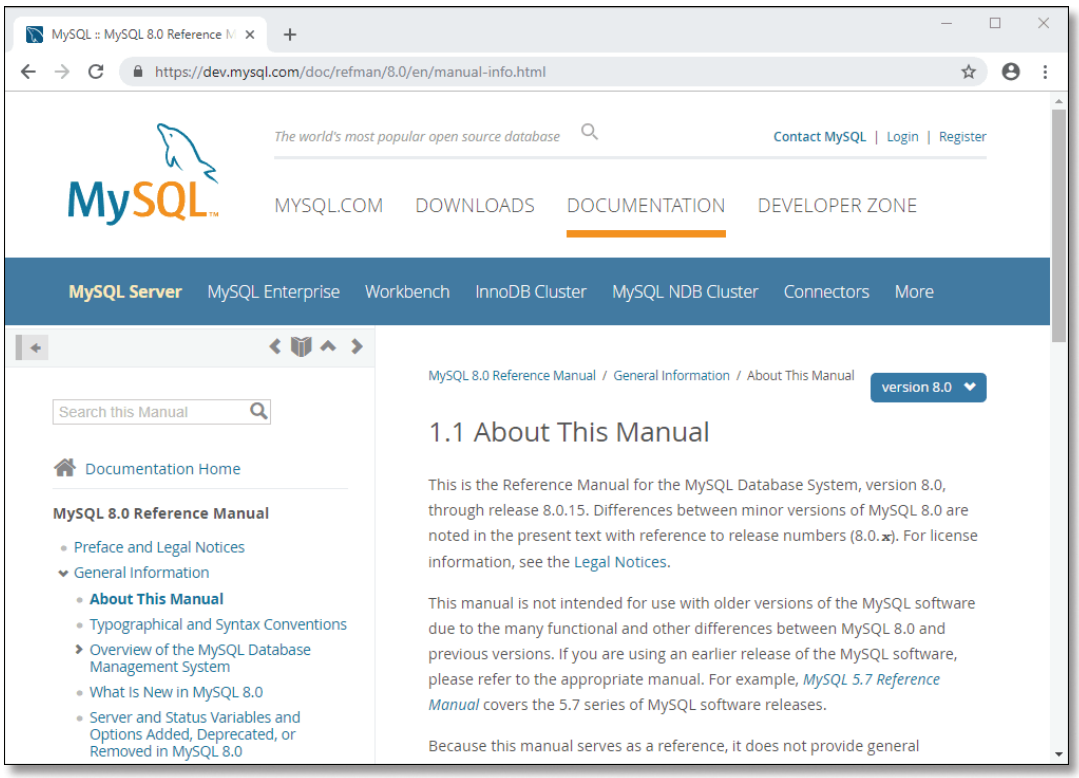
Once you've navigated to the correct version of the MySQL Reference Manual, it's easy to look up information. To do that, you can use the links in the left sidebar to drill down to the information that you're looking for. When you find the topic you want, you can click it to display it in the main part of the window. Then, if you want to navigate back up the hierarchy of information, you can use the breadcrumb links across the top of the page. In this figure, for example, you can click the "MySQL 8.0 Reference Manual" link to return to the Home page for the manual. Or, you can click the "General Information" link to navigate to that page.

Another easy way to look up information is to search for a specific word or phrase. To do that, type the word or phrase in the "Search this Manual" text box located at the top of the sidebar and click the Search icon or press the Enter key. Then, you can click the links in the search results to view information about the search terms.

The web address for the MySQL 8.0 Reference Manual

https://dev.mysql.com/doc/refman/8.0/en/

A web page from the MySQL Reference Manual



Description

- To view the *MySQL Reference Manual*, go to the MySQL website and select the correct version of the manual. The web address for MySQL 8.0 is shown above.
- To view a chapter, click the link for the chapter in the table of contents on the right side of the page.
- To return to the Home page for the manual, click the Start icon ( ) for the manual that's displayed at the top of the left sidebar.
- To search for a particular word or phrase, type the word or phrase in the "Search this Manual" text box in the left sidebar and click the Search icon or press the Enter key. Then, you can scroll through the results and click links to get more information.
- You can also download the MySQL Reference Manual. However, it typically makes sense to use it online.

Figure 2-12 How to use the MySQL Reference Manual

How to use the MySQL Command Line Client

Before MySQL Workbench was available, programmers used a command-line tool known as the *MySQL Command Line Client* to connect to a MySQL server and work with it. This tool is also known as the *MySQL command line*. Although you may never need this tool, you should at least be aware that it exists. This tool comes with MySQL, and it can be useful if MySQL Workbench isn't installed on the system that you're using.

How to start and stop the MySQL Command Line Client

Figure 2-13 shows how to start and stop the MySQL Command Line Client in Windows. Although this figure shows the Command Prompt window that's available from Windows, you can use the MySQL Command Line Client on other operating systems too. In particular, on macOS, you can use the Terminal window to start the MySQL Command Line Client.

When you use Windows, there's an easy way to start the MySQL Command Line Client if you want to log in as the root user for the database server that's running on the local computer. To do that, you just select the MySQL Command Line Client command from the Start menu. Then, MySQL will prompt you for a password. If you enter the password correctly, you will be logged on to the database server as the root user.

In some cases, you'll need to use a command line to start the MySQL Command Line Client instead of using the Start menu. For example, you may need to do that if you want to log into a database that's running on a different computer, if you want to log in as a user other than the root user, or if you're using another operating system such as macOS. In those cases, you can open a command line and change the directory to the bin directory for the MySQL installation. Then, you can execute the mysql command and supply the parameters that are needed to connect to the database server.

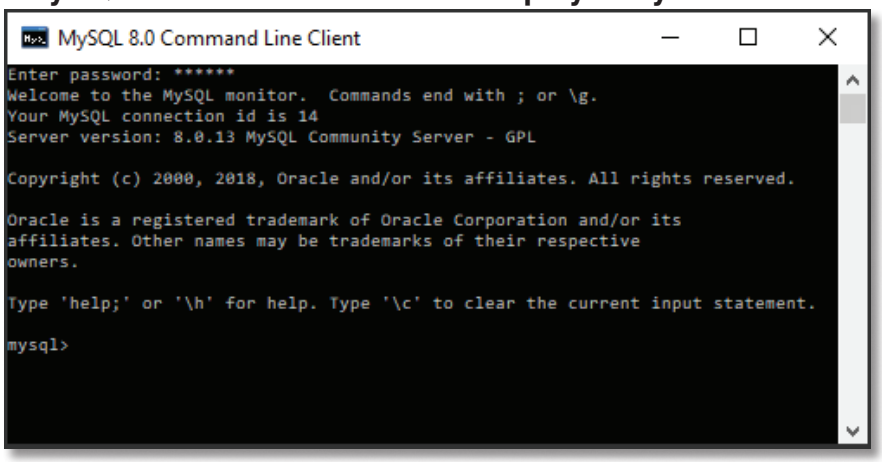
If the MySQL server is located on a remote computer, you can specify -h, followed by the host name of the computer, and -u, followed by a valid username. In addition, you specify -p so MySQL prompts you for a valid password. Although it can take some experimentation to get these connection parameters right, you only need to figure this out once.

Once you enter a valid password for the specified username, the MySQL Command Line Client displays a welcome message and a command line that looks like this:

```
mysql>
```

From this prompt, you can enter any statement that works with MySQL. When you're done, you can exit the MySQL Command Line Client by entering "exit" or "quit" followed by a semicolon.

The MySQL Command Line Client displayed by Windows



How to start the MySQL Command Line Client (Windows only)

Start→All Programs→MySQL→MySQL Server 8.0→MySQL 8.0 Command Line Client

How to start the MySQL Command Line Client from the command line

For Windows

```
cd \Program Files\MySQL\MySQL Server 8.0\bin
mysql -u root -p
```

For macOS

```
cd /usr/local/mysql/bin
./mysql -u root -p
```

How the mysql command works

The syntax

```
mysql -h hostname -u username -p
```

Examples

```
mysql -u ap_tester -p
mysql -h localhost -u root -p
mysql -h murach.com -u ap_tester -p
```

How to exit from the MySQL Command Line Client

```
mysql>exit;
```

Description

- MySQL provides a command-line client program called the *MySQL Command Line Client* that lets you enter SQL statements that work with MySQL databases. This program is also known as the *MySQL command line*.
- For Windows, use a Command Prompt window to start the MySQL Command Line Client.
- For macOS, use a Terminal window to start the MySQL Command Line Client.
- To stop the MySQL Command Line Client, enter "exit" or "quit" at the command line, followed by a semicolon.
- MySQL 8.0 also includes a Unicode version of the command-line client program. For more information on this program, you can refer to section 4.5.1.6.2 of the reference manual.

Figure 2-13 How to start and stop the MySQL Command Line Client

How to use the MySQL Command Line Client to work with a database

Once the MySQL Command Line Client is connected to a database server, you can use it to run SQL statements that work with the databases that are available from that server. When you enter a statement, you must end it with a semicolon. Otherwise, the mysql command line displays a second line when you press the Enter key like this:

```
mysql> show databases
    ->
```

This shows that the MySQL Command Line Client is waiting for you to finish your statement. To finish a statement and execute it, you just type a semicolon and press the Enter key.

Figure 2-14 shows how to execute three SQL statements. Here, I entered all three of these statements in lowercase letters. That's because SQL isn't case-sensitive, and lowercase letters are easier to type.

To list the names of the databases stored on a server, you use the SHOW DATABASES statement as illustrated by the first example. Here, the "ap", "ex", and "om" databases are the databases that are created when you install our downloadable databases as described in appendixes A and B. The "information_schema", "performance_schema", and "mysql" databases are internal databases that are used by the MySQL server. And the "sys" database is a database that comes with MySQL and can be used to interpret data in the "performance_schema" database.

To select the database that you want to work with, you can enter a USE statement as illustrated by the second example. Here, the AP database is selected, and the message after this statement says "Database changed" to indicate that the statement was successful. After you select a database, the commands and statements that you enter will work with that database.

To retrieve data from the database, you use a SELECT statement as illustrated by the third example. Here, the vendor_name column from the Vendors table is displayed. Note, however, that the result set is limited to only the first five rows. When you successfully execute a SELECT statement, the MySQL Command Line Client displays a message giving the number of rows that are included in the result set and the amount of time it took to run the query. In this case, it took less than 1/100 of a second to run the query.

How to list the names of all databases managed by the server

```
mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| ap                 |
| ex                 |
| information_schema |
| mysql              |
| om                 |
| performance_schema |
| sys                |
+--------------------+
7 rows in set (0.00 sec)
```

How to select a database for use

```
mysql> use ap;
Database changed
```

How to select data from a database

```
mysql> select vendor_name from vendors limit 5;
+-----------------------------+
| vendor_name                 |
+-----------------------------+
| Abbey Office Furnishings    |
| American Booksellers Assoc  |
| American Express            |
| ASC Signs                   |
| Ascom Hasler Mailing Systems |
+-----------------------------+
5 rows in set (0.00 sec)
```

Description

- You can use the MySQL Command Line Client to work with any of the databases running on the database server. To do that, you can use any SQL statement that works with a MySQL database.
- To execute a SQL statement, type the statement on the command line, followed by a semicolon. Then, press the Enter key.
- To show a list of all available databases, you can use the SHOW DATABASES statement.
- To select the database that you want to work with, you can use the USE statement.
- SQL statements aren't case-sensitive. As a result, when using the MySQL Command Line Client, most programmers enter their statements in lowercase letters because they're easier to type.

Figure 2-14 How to use the MySQL Command Line Client to work with a database