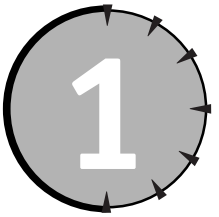
SESSION



Why MySQL?

Session Checklist

- ✔ SQL servers in the development process
- ✔ MySQL versus the competition

30 Min. To Go

To use MySQL effectively, you need to learn the syntax of a new language and grow comfortable with a new set of tools. However, before learning the specifics, you should understand the class of products to which MySQL belongs: database servers. Some of the early sessions of this book describe many of the theories that govern the majority of database servers. For starters, you should understand where database servers (also known as *SQL Servers* or *Relational Database Management Systems* [RDBMS]) fit into the application development process.

SQL Servers in the Development Process

When it comes time to develop an application, chances are you're going to need more than one tool. Whether your application is going to be viewed over the Web in a browser, in a Microsoft Windows environment, or in the MacOS, there's a good chance that multiple development packages will be needed to complete the application.

Developing Web applications

If you're interested in using MySQL, you're probably planning on building a dynamic Web site. MySQL can be used in other environments, but it is used most frequently for the Web. Creating a dynamic Web site takes at least three distinct pieces of software: the SQL server,

the programming/scripting language, and the Web server. An SQL server (in your case MySQL) is a vital piece of the process, but other pieces are equally necessary to get applications up and running. Figure 1-1 gives a diagram of the tools typically used in Web development.

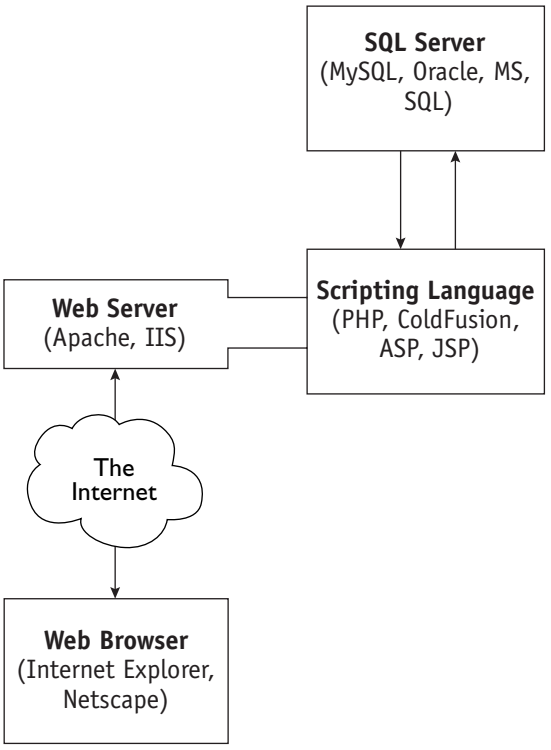


Figure 1-1 *Tools in the Web development process*

In Figure 1-1, you can see the three distinct pieces (the SQL server, the programming/scripting language, and the Web server). Each has its own responsibilities.

The SQL server

Before describing the responsibilities assigned to the SQL server, you should know what SQL stands for: Structured Query Language. SQL is a standard implemented by many companies in many products. In the course of your career, you may have come across products like Microsoft's Access, FoxPro, and SQL Server 7. These are all implementations of the Structured Query Language. Companies like Oracle, Sybase, Informix, and IBM all make billions of dollars

making products that implement this standard. You'll spend a good portion of this book learning the specifics of SQL, but for the sake of this session, all you need to know is what an SQL server's role is in the development process.

What an SQL server can do

An SQL server essentially has two jobs: to store data efficiently and to retrieve that data quickly. The SQL server is where your data — your valuable, crucial, business-dependent data — will live. You have the ability to insert, delete, and alter this information in any way you see fit.

Additionally, and perhaps most importantly, you can *query* this data — pull information out of the SQL server — in about any way you can imagine.

Further, an SQL server can provide statistical information on the data you have stored. If the idea of getting statistical information on your data is a bit vague right now, don't worry. It will become clear as you progress with this weekend-long course. For now, think of this simple example: In MySQL you've stored names and addresses of everyone affiliated with your business. If a time comes when you want to know exactly how many of these people live in a specific state or region of the country, your SQL server is able to quickly and easily return this information.

What an SQL server cannot do

Figure 1-1 illustrates that an SQL server does not interact directly with the Internet or with browsers. In Web-based applications, there are two related but distinct pieces of software that compliment the SQL server: the Web server and the middleware.

When you come down to it, the Web server has a pretty simple job. It is supposed to sit on the Internet and listen for requests made for a specific IP address or domain name. When you type an address into your browser, the Web server responds directly to your request. An SQL server cannot respond to requests that are sent via HTTP (the Hypertext Transfer Protocol), which is the language spoken by browsers and Web servers.

The two most popular Web servers are Microsoft's Internet Information Server (IIS) and the Apache Web server. You can use either Web server when creating dynamic Web sites with MySQL. However, chances are that if you are using MySQL, you're using the Apache Web server. Both of these products are open source and are frequently paired in Web applications. (I talk more about open source later in this session.)

As you can see in Figure 1-1, piece of software sits between the Web server and the SQL server, which is the scripting language, also known as *middleware*. The whole point in creating dynamic Web applications is so your Web site can react differently to different users. For example, the user should be able to add information to the site. The user should also have the ability to view information based on his or her particular needs.

Being dynamic requires your site to act differently based on different pieces of information, whether that information is coming from the user or the database. Here's a quick example:

- A viewer of your online real estate listing wishes to see only houses under $200,000 that are within a certain ZIP code. Based on this information supplied by the user, your Web pages show only the listings that fit these criteria.

In cases such as this, middleware is essential. In this example, the user enters information into form elements on an HTML page and submits the form. Figure 1-2 shows what this Web page may look like.

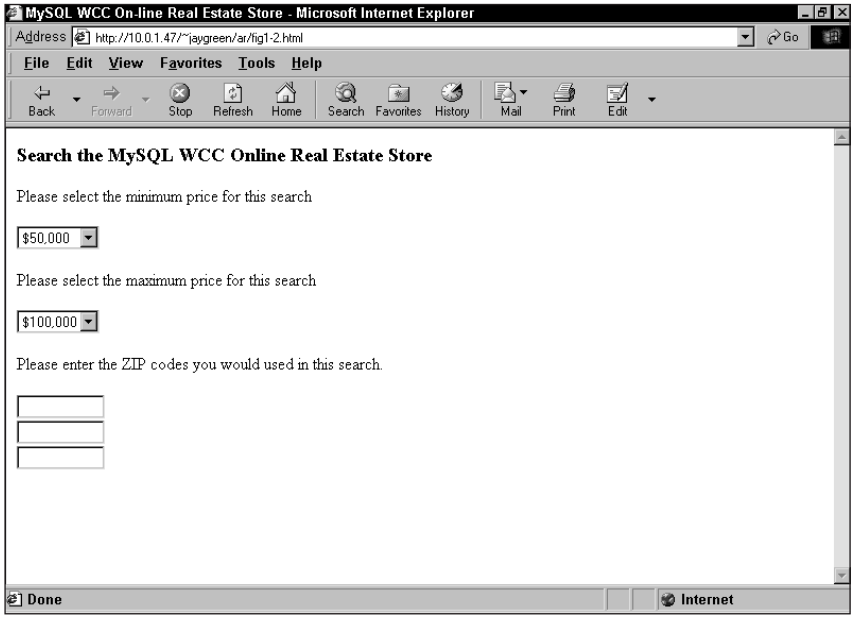


Figure 1-2 *HTML page that gathers user preferences*

The Web server receives the information sent by the browser, which includes the form information. That information needs to be molded into a form that the database server can respond to. Eventually, you need to make a request from the SQL server, but it must be in the format the SQL server is expecting.

This is where your middleware comes in. It can look at the incoming information and perform programmatic tasks on it as necessary. For example, if the user failed to enter location information, such as a ZIP code, the middleware you are using tests for the existence of at least one ZIP. If there isn't a ZIP, the middleware may present the form again with an added message alerting the user of his or her failure to complete the data. (See Figure 1-3 for an example of what this page may look like.)

Similarly, even if the user did fill out all of the form data, there is still a chance that no listing meets the user's criteria. In this case, you want to alert the user and offer him or her a chance to broaden the search parameters (additional ZIPs, higher price range, and so on). (See Figure 1-4 for an example of what this page may look like.)

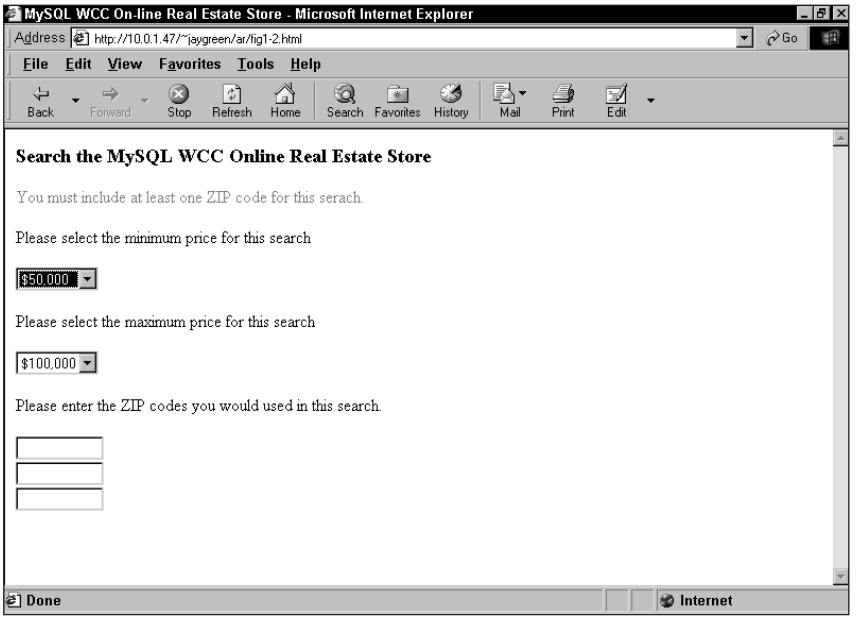


Figure 1-3 *HTML page requesting additional user data*

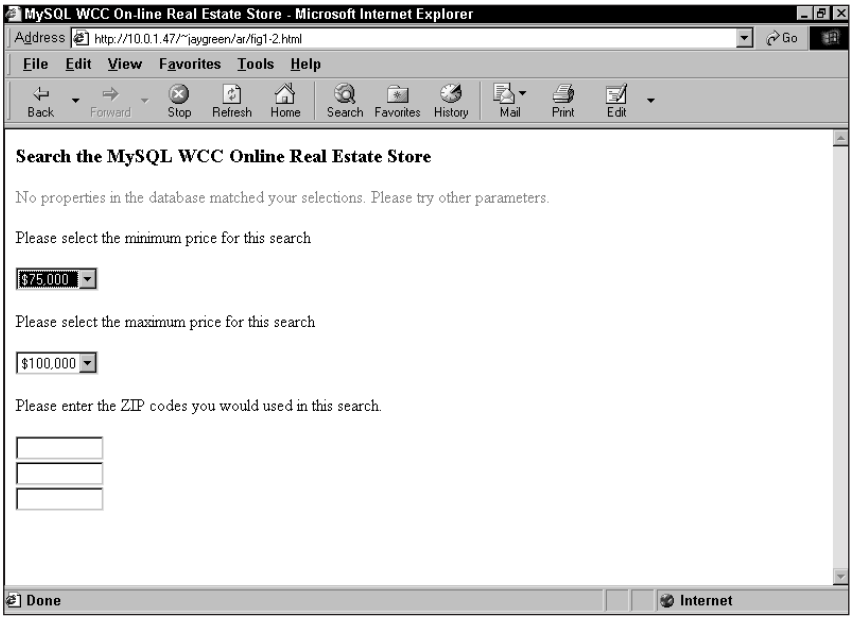


Figure 1-4 *HTML page requesting different parameters*

If the user fills out all of the information correctly and there are matching listings, those listings should be sent to the user's browser. (See Figure 1-5 for an example of what this page may look like.)

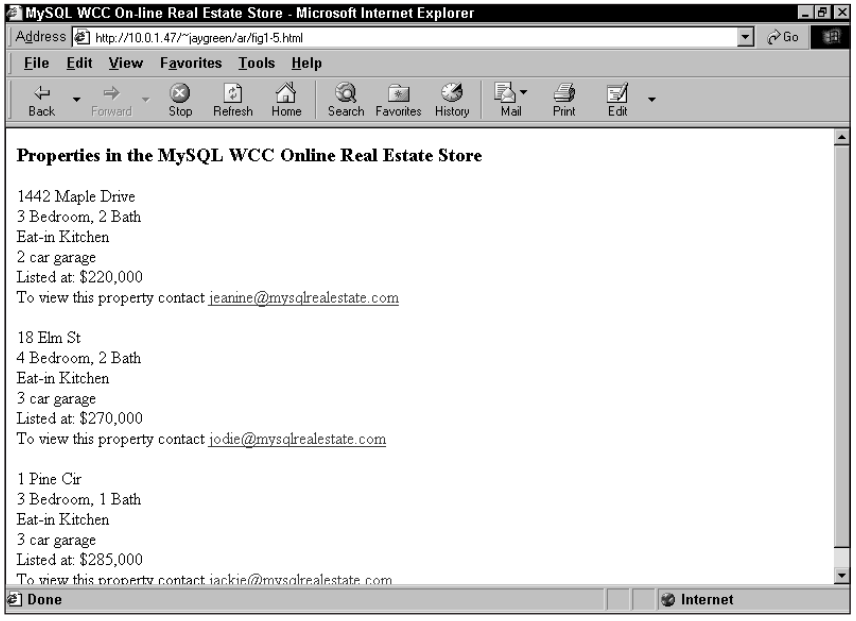


Figure 1-5 *HTML page presenting property listings*

Listing 1-1 shows how the middleware for a page that displays listings may work.

Listing 1-1 *Sample logic for middleware application*

```
if there are no ZIPs or Prices in the HTML forms
    re-display HTML page and notify user of missing data
    then quit the script
    (Figure 1-3)

else if all information has been provided
    request listing from the SQL server, using information
        supplied by user
```

```
        if the SQL server indicates there are no matching results
            tell user about the lack of results and re-display form
            (Figure 1-4)

        else if there are results returned by the SQL server
            while there are listings
                print listing information
                (Figure 1-5)
end if
```

This listing should make the point clear: All the decision logic is done in your middleware. If you have any familiarity with any type of programming, the listing should seem familiar. There are `if` branches and loops (the building blocks of programming.

You are going to be imbedding your database requests in middleware. The SQL server responds only to requests that come from the middleware.

There are many types of middleware available for Web applications. ColdFusion and Microsoft's ASP are two common choices. However, the most common companions to MySQL are PHP and Perl. Sessions 15 and 16 discuss using PHP with MySQL, and Sessions 17 and 18 discuss using Perl with MySQL.

Developing desktop applications

If you are coming to MySQL from a background in Microsoft Access, Paradox, or FileMaker Pro, you may feel that the architecture shown in Figure 1-1 and discussed in the previous pages differs substantially from how you're accustomed to working. After all, you probably think that Access, Paradox, FileMaker, or FoxPro require only one software package to get the job done.

Actually, this is not the case. These applications are packaged in a way that makes it seem that only one piece of software is at work. In all of these packages, a core database engine that is quite separate from the tools is used to build forms and tables for the users of the application. In the case of Access and Paradox, the default database server is known as the Jet engine. In fact, the Jet engine can be reached in many different ways: through Paradox or Access, from a development environment like Visual Basic or Delphi, or over the Web. You just have to let the Web server know where to find the Jet engine for that database.

With Access, you have the option of using something other than the Jet engine. In fact, you can configure a Windows machine so that MySQL serves and stores the information that is viewable through Access forms and Windows. Session 23 shows how you can connect MySQL to Microsoft Access.

Figure 1-6 shows how database servers fit into any applications development process. Notice that a single database server can offer data to many different clients simultaneously.

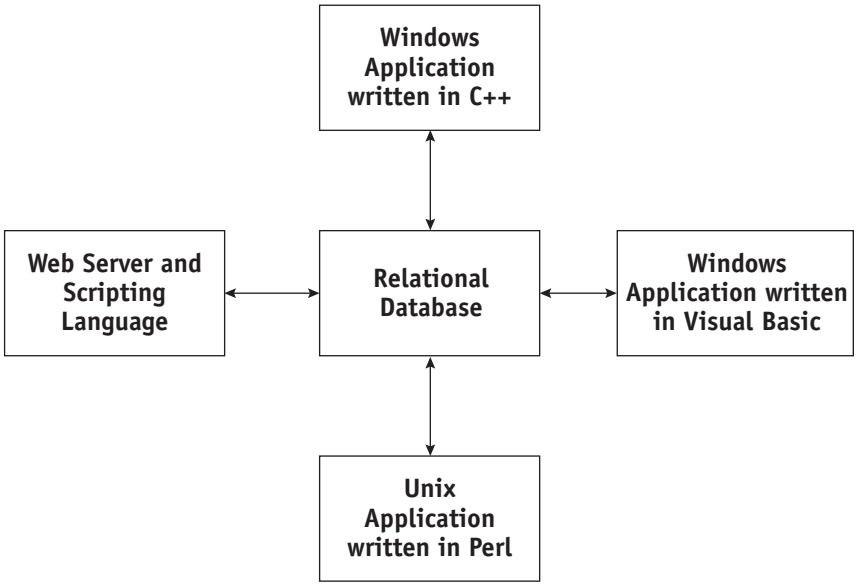


Figure 1-6 *Database servers in desktop applications development environment*

MySQL versus the Competition

You have many choices when it comes to RDBMSs. From Access to Oracle, there are products of many shapes and sizes that can fit your needs.

Before you commit to MySQL, you should know where it excels and where its competitors provide superior alternatives. In order to understand some of the specific advantages and deficiencies of MySQL, you need to understand some of the concepts covered later in this book. For now, I keep this information at a fairly general level.

10 Min. To Go

The advantages of MySQL

Here are some of MySQL's finer points. Each of the attributes discussed below can be used to describe other products, but put together, these features make MySQL an interesting and unique piece of software.

It's fast

MySQL is undeniably quick. It is a true multi-threaded database server that excels at retrieving information. MySQL was originally designed for the purpose of querying information at an amazingly fast pace.

This design introduced some disadvantages that made MySQL a poor choice for some people. A database is used for two basic actions: inserting/modifying data and *querying* (requesting information) from existing data. The creators of MySQL opted for an architecture that sacrificed some speed in the insert/modifying process to gain speed during queries. So if you were developing a database application in 1999 and you foresaw your application dealing with a lot of inserting and modifying of data, MySQL would have been a poor choice. However, MySQL now offers many configuration options that give you the choice of extreme speed in queries at times when heavy inserts are not expected, or you can opt for options that work better while inserting/modifying data.

For a time, the major criticism of MySQL was that it did not support *transactions*. If at this point you don't know what a transaction is, don't worry — they are explained in detail later in the book. If you do know what a transaction is, you'll be happy to know that MySQL supports this feature.

Just how fast is MySQL compared to the competition? It's very, very difficult to say. Benchmarking of database servers is very tricky business. The only real way to gauge comparative speed is to do a straight-up comparison with another product using the same hardware and the same code. So without giving numbers, which are easily disputable, I'll say what is generally believed: MySQL is fast, and you'll just have to believe that.

It's relatively easy to use

If you work in a Fortune 1000 business, there's an excellent chance that at many places in your organization, some developers are using a recent version of Oracle. You can be sure that the person in charge of the database went through extensive training and/or went to school to learn how to properly maintain this large and very complex database system.

MySQL doesn't offer much of the power of Oracle, but if you don't need all of the features Oracle offers (and most don't), you'll find it far, far easier to get your database and related applications up and running with MySQL. In fact, I believe that this book and the online manual may be all that you need to be a very competent user and administrator of MySQL.

It's widely used

MySQL is growing in popularity. If you're looking to develop a MySQL-powered Web site, you'll have little difficulty finding an ISP that supports MySQL. Additionally, when you run into problems, you will be able to go to one of the many MySQL support forums to get expert troubleshooting advice.

It's open source

MySQL is an open-source product released under the GPL (GNU Public License). At this point, many independent developers work on the code without any monetary compensation because they get great satisfaction from their efforts. In addition, at least two companies (MySQL AB and NuSphere) offer commercial support for MySQL.

With open-source software, the base code is available to any who wants to see and alter it. So if you happen to be a sophisticated C programmer, you are welcome to add your own enhancements.

If you go the open-source route with MySQL, you will find many products that complement your MySQL database server. If you remember back to Figure 1-1, a Web server and scripting language are used to create dynamic sites. In the open-source space, Apache is a well-known and well-respected Web server. As far as scripting languages go, Perl and PHP are two very popular open-source languages that are often used with MySQL.

When combined, these open-source developer tools make for a powerful development environment. This is especially true of Web-based applications.

The disadvantages of MySQL

Are you starting a bank — you know, a Savings & Loan? Or are you trying to create a site with the functionality of, say, Schwab.com? If so, MySQL is the wrong tool. MySQL does not implement features found in many other products. (If you are familiar with RDBMSs, these other products include triggers, stored procedures, and views).

For now, I think it's useful to think of MySQL's missing features in this way: If you are developing a database that will have many different front ends (maybe one written for the Web, one written in Visual Basic or C, and another for Unix), then MySQL isn't a great choice. In situations like these, when your database must react to many different environments, the database administrator needs more control over the data than MySQL offers. In a case like this, you should look at some of the competitive products like Oracle, Sybase, or in the open-source space, PostgreSQL.

Done!

Review

This session introduced you to the category of products known as database servers. It showed how database servers fit into the applications development process for both Web-based and non-Web-based applications. The session then reviewed MySQL's strengths and weaknesses and gave examples of where MySQL is an appropriate choice and where it is a less-than-ideal alternative.

Quiz Yourself

1. In addition to database servers, what two products are needed to create Web-based applications? (See "Developing Web applications.")
2. Which piece of development software would be responsible for alerting a user that a database contained no relevant information? (See "What an SQL server cannot do.")
3. Name three reasons for selecting MySQL. (See "The advantages of MySQL.")
4. In what sort of environment is MySQL a poor choice? (See "The disadvantages of MySQL.")
5. Name three open-source products that are typically used with MySQL in delivering Web-based applications. (See "It's open source.")