PART 1

Getting Started with MySQL Development

This part introduces you to concepts in developing and modifying open source systems. Chapter 1 guides you through the benefits and responsibilities of an open source system integrator. It highlights the rapid growth of MySQL and its importance in the open source and database system markets. Chapter 2 covers the basics of what a database system is and how it is constructed. Chapter 3 provides a complete introduction to the MySQL source presented in this chapter along with how to obtain and build the system. Chapter 4 introduces a key element in generating high-quality extensions to the MySQL system. You'll learn about software testing as well as common practices for testing large systems.

CHAPTER 1

MySQL and the Open Source Revolution

Open source systems are rapidly becoming a force that is changing the software landscape. Information technology professionals everywhere are taking note of the high-quality, and in many cases world-class, development and support offered by open source software vendors. Corporations are paying attention because for the first time they have an alternative to the commercial proprietary software vendors. Small businesses are paying attention because open source software can significantly lower the cost of their information systems. Individuals are paying attention because they have more choices with more options than ever before. The majority of the underpinnings that make the Internet what it is today are based on open source software such as Linux, Apache HTTP server, BIND, Sendmail, OpenSSL, MySQL, and many others.

The most common business objective that drives the choice to use open source software is cost. Open source software, by its very nature, reduces the total cost of ownership (TCO) and provides a viable business model on which businesses can build or improve their markets. In the case of open source database systems, this is especially true. The cost of commercial proprietary database systems begins in the multiple thousands of dollars and, by the time you add support costs, can easily go into the tens or hundreds of thousands of dollars.

It used to be that open source software was considered by many to be limited to the hobbyist or hacker bent on subverting the market of large commercial software companies. Although it may be true that some developers feel they are playing the role of David to Microsoft's Goliath, the open source community is not about that at all. The open source community does not profess to be a replacement for commercial proprietary software, but rather they propose the open source philosophy as an alternative solution. As you will see in this chapter, not only is open source a viable alternative to commercial software, but it is also fueling a worldwide revolution of how software is developed and marketed.

Note In this book, the term *hacker* refers to Richard Stallman's definition of hacker: "someone who loves to program and enjoys being clever about," and not the common perception of nefarious villain bent on stealing credit cards and damaging computer systems.

The following section is provided for those who may not be familiar with open source software or the philosophy of MySQL. If you are already familiar with open source software philosophy, you can skip to the section "Developing with MySQL."

What Is Open Source Software?

Open source software grew from a conscious resistance to the corporate property mind-set. While working for the Artificial Intelligence Lab at the Massachusetts Institute of Technology (MIT), Richard Stallman began a code-sharing movement in the 1970s. Fueled by the desire to make commonly used code available to all programmers, Stallman saw the need for a cooperating community of developers. This philosophy worked well for Stallman and his small community—that is, until the industry collectively decided software was property and not something that should be shared with potential competitors. This resulted in many of the MIT researchers being lured away from MIT to work for these corporations. Eventually, the cooperative community faded away.

Fortunately, Stallman resisted the trend and left MIT to start the GNU (GNU Not Unix) project and the Free Software Foundation (FSF). The goal of the GNU project was to produce a free Unix-like operating system. This system would be free (including access to the source code) and available to anyone. The concept of free was to not prohibit anyone from using and modifying the system.

Stallman's goal was to reestablish the cooperating community of developers that worked so well at MIT. However, Stallman had the foresight to realize the system needed a copyright license that guaranteed certain freedoms. (Some have coined Stallman's take on copyright as "copyleft" as it guarantees freedom rather than restricts it.) Stallman created the GNU Public License (GPL). The GPL is a clever work of legal permissions that permits the code to be copied and modified without restriction, and states that derivative works (the modified copies) must be distributed under the same license as the original version without any additional restrictions. Essentially, this uses the copyright laws against copyrights by removing the proprietary element altogether.

Unfortunately, Stallman's GNU project never fully materialized, but several portions have become essential elements of many open source systems. The most successful of these include the GNU compilers for the C programming language (GCC) and the GNU text editor (Emacs). Although the GNU operating system failed to be completed, the pioneering efforts of Stallman and his followers permitted Linus Torvalds to fill the gap with his then-infant Linux operating system in 1991. Linux has become the free Unix-like operating system that Stallman envisioned (see the sidebar "Why Is Linux So Popular?"). Today, Linux is the world's most popular and successful open source operating system.

WHY IS LINUX SO POPULAR?

Linux is a Unix-like operating system built on the open source model. It is therefore free for anyone to use, distribute, and modify. Linux is built using a conservative kernel design that has proven to be easy to evolve and improve. Since its release in 1991, Linux has gained a worldwide following of developers who seek to improve its performance and reliability. Some may even claim Linux is the most well developed of all operating systems. Since its release, Linux has gained a significant market share of the world's server and workstation installations. Linux is often cited as the most successful open source endeavor to date.

There was one problem with the free software movement. Free was intended to guarantee freedom to use, modify, and distribute, not free as in no cost or free-to-a-good home (often explained as "free" as free speech, not "free" as in free beer). To counter this misconception, the Open Source Initiative (OSI) was formed and later adopted and promoted the phrase "open source" to describe the freedoms guaranteed by the GPL; visit the web site at `www.opensource.org`.

The efforts of the OSI changed the free software movement. Software developers were given the opportunity to distinguish between free software that is truly no cost and open software that was part of the cooperative community. With the explosion of the Internet, the cooperative community has become a global community of developers. This global community of developers is what ensures the continuation of Stallman's vision.

Open source software therefore is software that is licensed to guarantee the rights of developers to use, copy, modify, and distribute their software while participating in a cooperative community whose natural goals are the growth and fostering of higher-quality software. Open source does not mean zero cost. It does mean anyone can participate in the development of the software and can, in turn, use the software without incurring a fee. On the other hand, many open source systems are hosted and distributed by organizations that sell support services for the software. This permits organizations that use the software to lower their information technology costs by eliminating startup costs and in many cases saving a great deal on maintenance.

All open source systems today draw their lineage from the foundations of the work that Stallman and others produced in an effort to create a software utopia in which Stallman believed organizations should generate revenue from selling services, not proprietary property rights. There are several examples of Stallman's vision becoming reality. The GNU/Linux (henceforth referred to as Linux) movement has spawned numerous successful (and profitable) companies, such as Red Hat and Slackware, that sell customized distributions and support for Linux. Another example is MySQL, which has become the most successful open source database system.

Although the concept of a software utopia is arguably not a reality today, it is possible to download an entire suite of systems and tools to power a personal or business computer without spending any money on the software itself. No-cost versions of software ranging from operating systems and server systems such as database and web servers to productivity software are available for anyone to download and use.

Why Use Open Source Software?

Sooner or later, someone is going to ask why using open source software is a good idea. To successfully fend off the ensuing challenges from proponents of commercial proprietary software, you should have a solid answer. The most important reasons for adopting open source software are

- Open source software costs little or nothing to use. This is especially important for nonprofits, universities, and community organizations whose budgets are constantly shrinking and that must do more with less every year.
- Open source software permits you to modify it to meet your specific needs.
- The licensing mechanisms available are more flexible than commercial licenses.
- Open source software is more robust (tested) than commercial proprietary software.
- Open source software is more reliable and secure than commercial proprietary software.

Although it is likely you won't be challenged or asked to demonstrate any of these reasons for adopting open source software, you are more likely to be challenged by contradiction. That is, proponents of commercial proprietary software (opponents of open source) will attempt to discredit these claims by making statements about why you shouldn't use open source software for development. Let's examine some of the more popular reasons not to use open source software from a commercial proprietary software viewpoint and refute them with the open source view.

Myth 1: Commercial Proprietary Software Fosters Greater Creativity

The argument goes: most enterprise-level commercial proprietary software provide application programming interfaces (API) that permit developers to extend their functionality, thus making them more flexible and ensuring greater creativity for developers.

Portions of this statement are true. APIs do permit developers to extend the software, but they often do so in a way that strictly prohibits developers from adding functionality to the base software. These APIs often force the developer into a sandbox, further restricting her creativity.

■ **Note** Sandboxes are often created to limit the developer's ability to affect the core system. The main reason for doing this has to do with security. The more open the API is, the more likely it is for villainous developers to create malicious code to damage the system or its data.

Open source software may also support and provide APIs, but open source provides developers with the ability to see the actual source code of the core system. Not only can they see the source code, they are free (and encouraged) to modify it! Some of the reasons you may want to modify the core system are when a critical feature isn't available or you need the system to read or write a specific format. Therefore, open source software fosters greater creativity than commercial proprietary software.

Myth 2: Commercial Proprietary Software Is More Secure Than Open Source Software

The argument goes: organizations require their information systems in today's Internet-connected society to be more secure than ever before. Commercial proprietary software is inherently more secure because the company that sells the software has a greater stake in ensuring their products can stand against the onslaught of today's digital predators.

Although the goals of this statement are quite likely to appear on a boardroom wall as a mantra for any commercial software vendor, the realization of this goal, or in some cases marketing claim, is often misleading or unobtainable. Let's consider the Microsoft Windows server operating system. It can be shown that the Windows server operating system is less secure than Linux. While Microsoft has built in a successful and efficient patch system to ensure installations are kept free from exposed vulnerabilities, the fact that these mechanisms are part of everyday server maintenance is reason enough to consider that Microsoft hasn't obtained a level of security that is sufficient to ward off attacks. (Sadly, some would say as long as there is a Microsoft there will be digital predators.)

The main reason why Linux is more secure than Windows is because the global community of developers who have worked on Linux have worked together to ensure the system is protected against attacks (also called *hardening*). In the case of Linux, many developers throughout the

world are working toward hardening the system. The more developers working on the problem, the more creative ways there are to solve it. When new vulnerabilities are discovered in Linux, they are fixed quickly and the door is slammed in the predator's face.

Microsoft, on the other hand, has far fewer developers to devote to hardening Windows and therefore fewer ideas on how to solve the problem. Thus, the hardening of Windows will be a much longer course than Linux. This argument probably isn't true for all open source software, but it does show that open source systems can adapt to threats and become more secure than commercial proprietary software.

Myth 3: Commercial Proprietary Software Is Tested More Than Open Source Software

The argument goes: software vendors sell software. The products they sell must maintain a standard of high quality or customers won't buy them. Open source software is not under any such pressure and therefore is not tested as stringently as commercial proprietary software.

This argument is very compelling. In fact, it sings to the hearts of all information technology acquisition agents. They are convinced paying for something means it is more reliable and freer of defects than software that can be acquired without a fee. Unfortunately, these individuals are overlooking one important concept of open source software.

Open source software is developed by a global community of developers, many of whom consider their role as defect detectives (testers). These individuals pride themselves on finding and reporting defects. In some cases, open source software companies have offered rewards for developers who find repeatable bugs. MySQL AB offers a significant reward for finding bugs in their MySQL database system. At the time of this writing, MySQL AB was offering a free Apple iPod nano to anyone who finds a repeatable bug in their software. Now, that's an incentive!

It is true that software vendors employ software testers (and no doubt they are the best in their field), but more often than not commercial software projects are pushed toward a specific deadline. These deadlines are put in place to ensure a strategic release date or competitive advantage. Many times these deadlines force software vendors to compromise on portions of their software development process—which is usually the later part: testing. As you can imagine, reducing a tester's access to the software (testing time) means they will find fewer defects.

Open source software companies, by enlisting the help and support of the global community of developers, ensure that their software is tested more often by more people. Therefore, open source software is tested more than commercial software.

Myth 4: Commercial Proprietary Systems Have More Complex Capabilities and More Complete Feature Sets Than Open Source Systems

The argument goes: commercial proprietary database systems are sophisticated and complex server systems. Open source systems are neither large nor complex enough to handle mission-critical enterprise data.

Although it is true that some open source systems are good imitations of the commercial systems they mimic, the same cannot be said for a database system such as MySQL. Earlier versions of MySQL did not have all of the features found in commercial proprietary database systems. However, with the release of version 5.0, MySQL has all of the advanced features of the commercial proprietary database systems.

Furthermore, MySQL has been shown to provide the reliability, performance, and scalability that large enterprises require for mission-critical data. Indeed, many well-known organizations use MySQL for mission-critical data. Therefore, MySQL is one example of an open source

system that offers all of the features and capabilities of the best commercial proprietary database systems.

Myth 5: Commercial Proprietary Software Vendors Are More Responsive Because They Have a Dedicated Staff

The argument goes: when a software system is purchased, the software comes with the assurances that the company that produced the software is available for assistance or to help solve problems. Open source systems, by the very nature that no one "owns" it, means that it is far more difficult to contact anyone for assistance.

Most open source software is built by the global community of developers. However, the growing trend is to base a business model on the open source philosophy and build a company around it selling support and services for the software that they oversee. In fact, most of the major open source products are supported in this manner. For instance, MySQL AB owns the source code for their MySQL product. (For a complete description of MySQL's open source license, see `www.mysql.com/company/legal/licensing/opensource-license.htm`.) MySQL AB provides a wide range of support options, including 24×7 coverage and response times as low as 30 minutes.

Developers who develop open source software respond much more quickly to issues and problems than commercial developers. In fact, it can be nearly impossible to talk to a commercial software developer directly. Microsoft has a comprehensive support mechanism in place and can meet the needs of just about any organization. However, if you want to talk to a developer of a Microsoft product, you will have to go through proper channels. This requires talking to every stage of the support hierarchy—and even then are you not guaranteed contact with the developer.

Open source developers, on the other hand, use the Internet as their primary form of communication. Since they are already on the Internet, they are much more likely to see your question appear in a forum or news group. Additionally, open source companies like MySQL AB actively monitor their community and can respond quickly to their customers.

Therefore, it is not true that purchasing commercial proprietary software guarantees you quicker response times than open source software. It has been shown that in many cases open source software developers are more responsive (reachable) than commercial software developers.

What If They Want Proof?

I've listed just a few of the arguments that are likely to cause you grief as you attempt to adopt open source software in your organization. Several researchers have attempted to prove arguments such as these. One researcher, James W. Paulson, has conducted an empirical study of open source and commercial proprietary software (he calls it "closed"), which examines the preceding arguments and proves that open source software development can demonstrate measurable improvements over commercial proprietary software development. See Paulson's article, "An Empirical Study of Open-Source and Closed-Source Software Products," in the April 2004 issue of *IEEE Transactions on Software Engineering*.

Is Open Source Really a Threat to Commercial Software?

Until recently, open source software was not considered a threat to the commercial proprietary software giants. The two largest commercial competitors to MySQL AB are beginning to exhibit

the classic signs of competitive threat. Microsoft continues to speak out against open source software, denouncing MySQL as a world-class database server while passively ignoring the threat. However, Oracle is taking a considerably different tactic.

Oracle has recently gone on a corporate spending spree, purchasing open source companies SleepyCat and Innobase. Both companies provide solutions that are part of the MySQL system. While support agreements are in place and no immediate consequences are expected from this maneuver, industry pundits agree that despite Oracle's claim of innocent diversification, the database giant is hedging its bets and staking a claim in the open source database segment. With an estimated $12 billion database server market projected for 2007 the stakes are clearly profit and market share.

Perhaps the most telling betrayal of Oracle's misdirected innocence is its recent attempt to purchase MySQL AB. What better example of a threat can one find than one's closest competitor desiring to own what you have? MySQL AB deserves great praise in standing their ground and refusing to sell their endeavors. Few would blame them for cashing in and enjoying their fortunes. However, the strength of the philosophy that is the open source world has prevailed and the CEOs of MySQL AB felt there is more to be gained by continuing their quest for becoming the world's best database system.

The pressure of competition isn't limited to MySQL versus proprietary database systems. At least one open source database system, Apache Derby, is touting itself as an alternative to MySQL and has recently tossed its hat into the ring as a replacement for the "M" in the LAMP stack (see the sidebar "What Is the LAMP Stack?"). Proponents for Apache Derby cite licensing issues with MySQL and feature limitations. Neither has deterred the MySQL install base, nor have these "issues" limited MySQL's increasing popularity.

WHAT IS THE LAMP STACK?

LAMP stands for Linux, Apache, MySQL, and PHP/Perl/Python. The LAMP stack is a set of open source servers, services, and programming languages that permit rapid development and deployment of high-quality web applications. The key components are

- *Linux*: A Unix-like operating system. Linux is known for its high degree of reliability and speed as well as its vast diversity of supported hardware platforms.
- *Apache*: A web application server known for its high reliability and ease of configuration. Apache runs on most Unix operating systems.
- *MySQL*: The database system of choice for many web application developers. MySQL is known for its speed and small execution footprint.
- *PHP/Perl/Python*: These are all scripting languages that can be embedded in HTML web pages for programmatic execution of events. These scripting languages represent the active programming element of the LAMP stack. They are used to interface with system resources and back-end database systems to provide active content to the user. While most LAMP developers prefer PHP over the other scripting languages, each can be used to successfully develop web applications.

There are many advantages to using the LAMP stack for development. The greatest advantage is cost. All of the LAMP components are available as no-cost open source licenses. Organizations can download, install, and develop web applications in a matter of hours with little or no initial cost for the software.

An interesting indicator of the benefits of offering an open source database system is the recent offering of "free" versions from some of the proprietary database vendors. Microsoft, which has been a vocal opponent of open source software, now offers a no-cost version of its SQL Server 2005 database system called SQL Server Express. Although there is no cost for downloading the software and you are permitted to distribute the software with your application, you are not permitted to see the source code or modify it in any way. Oracle also offers a "free" version of its database system called Oracle Database Express Edition. Like Microsoft, Oracle grants you a no-cost download and the right to distribute the server with your application, but does not permit modification or access to the source code. Both of these products have reduced features (Oracle more so) and are not scalable to a full enterprise-level database server without purchasing additional software and services.

Clearly, the path that MySQL AB is blazing with its MySQL server products demonstrates a threat to the proprietary database market—a threat that the commercial proprietary software industry is taking seriously. Whatever the facts concerning Oracle's recent open source spending spree (we may never know), it is clear they are reacting to the threat of MySQL AB. Although Microsoft continues to try to detract from the open source software market, they too are starting to see the wisdom of no-cost software.

Legal Issues and the GNU Manifesto

Commercial proprietary software licenses are designed to limit your freedoms and to restrict your use. Most commercial licenses state clearly that you, the purchaser of the software, do not own the software but are permitted to use the software under very specific conditions. In almost all cases, this means you cannot copy, distribute, or modify the system in any way. These licenses also make it clear that the source code is owned exclusively by the licenser and you, the licensee, are not permitted to see or reengineer it.

Open source systems are generally licensed using a GNU-based license agreement (GNU stands for GNU, not Unix). Most permit free use of the original source code with a restriction that all modifications be made public or returned to the originator as legal ownership. Furthermore, most open source systems use the GPL agreement, which states that it is intended to guarantee your rights to copy, distribute, and modify the software. It is interesting to note that the GPL does not limit your rights in how you use the software. In fact, the GPL specifically grants you the right to use the software however you want. The GPL also guarantees your right to have access to the source code. All of these rights are specified in the GNU Manifesto and the GPL agreement (`www.gnu.org/licenses/gpl.html`).

What is most interesting, the GPL specifically permits you to charge a distribution fee (or media fee) for distribution of the original source and provides you the right to use the system in whole or modified in order to create a derivative product, which is also protected under the same GPL. The only catch is you are required to make your modified source code available to anyone who wants it.

These limitations do not prohibit you from generating revenue from your hard work. On the contrary, as long as you turn over your source code by publishing it via the original owner, you can charge your customers for your derivative work. Some may argue that this means you can never gain a true competitive advantage because your source code is available to everyone. However, the opposite is true in practice. Vendors such as Red Hat and MySQL AB have profited from business models based on the GPL.

The only limitations of the GPL that may cause you pause is the limitation on warranties and the requirement to place a banner in your software stating the derivation (original and license) of the work.

A limitation on expressed warranties isn't that surprising if you consider that most commercial licenses include similar clauses. The part that makes the GPL unique is the concept of nonliable loss. The GPL specifically frees the originator and you, the modifier (or distributor), from loss or damage as a result of the installation or use of the software. Stallman did not want the legal industry to cash in should there ever be a question of liability of open source software. The logic is simple. You obtained the software for free and you did not get any assurances for its performance or protection from damages as a result of using the software. In this case, there is no quid pro quo and thus no warranty of any kind.

Opponents of the open source movement will cite this as a reason to avoid the use of open source software, stating that it is "use at your own risk" and therefore introduces too much risk. While that's true enough, the argument is weakened or invalidated when you purchase support from open source vendors. Support options from open source vendors often include certain liability rights and further protections. This is perhaps the most compelling reason to purchase support for open source software. In this case, there is quid pro quo and in many cases a reliable warranty.

The requirement to place a banner in a visible place in your software is not that onerous. The GPL simply requires a clear statement of the software's derivation and origination as well as marking the software as protected under the GPL. This informs anyone who uses this software of their rights (freedoms) to use, copy, distribute, and modify the software.

Perhaps the most important declaration contained in the GNU manifesto is the statements under the heading, "How GNU Will Be Available." In this section, the manifesto states that although everyone is permitted to modify and redistribute GNU, no one is permitted to restrict its redistribution further. This means no one can take an open source system based on the GNU manifesto and turn it into a proprietary system or make proprietary modifications.

Property

A discussion of open source software licensing would be incomplete if the subject of property were not included. Property is simply something that is owned. We often think of property as something tangible, something we can touch and see. In the case of software, the concept of property becomes problematic. What exactly do we mean when we say software is property? Does the concept of property apply to the source code, the binaries (executables), documentation, or all of them?

The concept of property is often a sticky subject when it comes to open source software. Who is the owner if the software is produced by the global community of developers? In most cases, open source software begins life as a project someone or some organization has developed. The project becomes open source when the software is mature enough to be useful to someone. Whether this is at an early stage when the software is unrefined or later when the software reaches a certain level of reliability is not important. What *is* important is the fact that someone started the project. That someone is considered the owner. In the case of MySQL, the company, MySQL AB, originated the project and therefore they are the owners of the MySQL system.

According to the GPL that MySQL adheres to, MySQL AB owns all the source code and any modifications made under the GPL. The GPL gives you the right to modify MySQL, but it does not give you the right to claim the source code as your property.

The Ethical Side

Everyone dreads the 12-headed dragon called ethics. Ethical dilemmas abound when you first start working with open source software. For example, open source software is free to download, but you have to turn over any improvements you make to the original owner. So how can you make any money off something you have to give away?

To understand this, you must consider the goals that Stallman had in mind when he developed the GNU license model. His goals were to make a community of cooperation and solidarity among developers throughout the world. He wanted source code to be publicly available and the software generated to be free for anyone to use. Your rights to earn (to be paid) for your work are not restricted. You can sell your derivative work. You just can't claim ownership of the source code. You are ethically (and legally!) bound to give back to the global community of developers.

Another ethical dilemma with open source software arises when you consider what should occur if you modify open source software for your own use. For example, you download the latest version of MySQL and add a feature that permits you to use your own abbreviated shortcuts for the SQL commands because you're tired of typing out long SQL statements (I am sure someone somewhere has already done this).

In this case, you aren't modifying the system in a way that could be beneficial to anyone but yourself. So why should you turn over your modifications? Although this dilemma is probably not an issue for most of us, it could be an issue for you if you persist in using the software with your personal modifications and eventually create a derivative work. Care must be taken whenever you modify the source code no matter what the reason. Basically, any productive and meaningful modification you make must be considered property of the originator regardless of its use or limits of its use.

However, if you are modifying the source code as an academic exercise (as I will show you how to do later in this book), the modifications should be discarded once you have completed your exercises or experiments. Some open source software makes provisions for these types of uses. Most consider the exploration and experimentation of the source code a "use" of the software and not a modification. It is therefore permissible to use the source code in academic pursuits.

Let the Revolution Continue!

Freedom is a right that many countries have based their government philosophies on. It is freedom that drove Richard Stallman to begin his quest to reform software development. Although freedom was the catalyst for the open source movement, it has become a revolution because organizations now have an opportunity to avoid obsolescence at the hands of their competitors by investing in lower-cost software systems while maintaining the revenue to compete in their markets.

Organizations that have adopted open source software as part of their own product lines are perhaps the most revolutionary of all. Most have adopted a business model based on the GPL that permits them to gain all of the experience and robustness that come with open source systems while still generating revenue for their own ideas and additions.

Open source software is both scorned and lauded by the software industry. Some despise open source because they see it as an attack against the commercial proprietary software industry. They also claim open source is a fad and will not last. They see organizations that produce, contribute to, or use open source software as being on borrowed time and that sooner rather than later the world will come to its senses and forget about open source software. Some

don't despise open source as much as they see no possibility for profit and therefore dismiss the idea as fruitless. Others see open source software as the savior to rescue us all from the tyrants of commercial proprietary software and that sooner rather than later the giant software companies will be forced to change their property models to open source or some variant thereof. The truth is probably in the middle. I see the open source industry as a vibrant and growing industry of similar-minded individuals whose goals are to create safe, reliable, and robust software.

Whatever your perspective, you must conclude that the open source movement has caused a revolution among software developers everywhere. Now that you have had a sound introduction to the open source revolution, it is your turn to decide whether or not you agree to the philosophies. If you do (and I sincerely hope I have convinced you to), then welcome to the global community of developers. Viva le revolution!

Developing with MySQL

You've taken a look at what open source software is and the legal ramifications of using and developing with open source software. Now you'll learn how to develop products using MySQL. As you'll see, MySQL presents a unique opportunity for developers to exploit a major server software technology without the burden of conforming or limiting their development to a fixed set of rules or limited API suite.

MySQL is owned by MySQL AB. The "AB" is an acronym for the Swedish word "aktiebolag" or "stock company," which translates to the English (US) term "incorporated." What began as a capital venture to build an open source relational database system has become a credible alternative to the commercial database system market. MySQL AB generates revenue by selling commercial licenses, support, and professional development services, including consulting, training, and certification on their products.

MySQL is a relational database management system designed for use in client/server architectures. MySQL can also be used as an embedded database library. Of course, if you have used MySQL before, you are familiar with its capabilities and no doubt have decided to choose MySQL for some or all of your database needs.

MySQL has become the world's most popular and most successful open source database system. This popularity is due in large part to its reliability, performance, and ease of use. There are over 8 million installations of MySQL products worldwide. MySQL AB's success can be attributed to a sound core values statement: "To make superior data management software available and affordable to all." This core values statement is manifested by MySQL AB's key business objectives—to make its database system products

- The world's best and most widely used
- Affordable and available to everyone
- Easy to use
- Continuously improved while maintaining speed and data integrity
- Fun and easy to extend and evolve
- Free from defects

Clearly, MySQL AB has achieved all of these objectives and continues to surprise database professionals everywhere with the quality and performance of their products.

What you may not know is how MySQL came about and how it is constructed. At the lowest level of the system, the server is built using a multithreaded model written in a combination of C and C++. Much of this core functionality was built in the early 1980s and later modified with a Structured Query Language (SQL) layer in 1995. MySQL was built using the GNU C compiler (GCC), which provides a great deal of flexibility for target environments. This means MySQL can be compiled for use on just about any Linux operating systems. MySQL AB has also had considerable success in building variants for the Microsoft Windows and Macintosh operating systems. The client tools for MySQL are largely written in C for greater portability and speed. Client libraries and access mechanism are available for .NET, Java, ODBC, and several others.

MySQL is built using parallel development paths to ensure product lines continue to evolve while new versions of the software are planned and developed. Software development follows a staged development process where multiple releases are produced in each stage. The stages of a MySQL development process are as follows:

1. *Development*—New product or feature sets are planned and implemented as a new path of the development tree.
2. *Alpha*—Feature refinement and defect correction (bug fixes) are implemented.
3. *Beta*—The features are "frozen" (no new features can be added) and additional intensive testing and defect correction is implemented.
4. *Gamma*—Basically, this is a release candidate stage where the code is frozen and final rounds of testing are conducted.
5. *Stable*—If no major defects are found, the code is declared stable and ready for production release.

You'll often see various versions of the MySQL software offered in any of these stages. The parallel development strategy permits MySQL AB to maintain its current releases while working on new features. It is not uncommon to read about the new features in 5.1 while development is continuing in 4.0.10. This may seem confusing because we are used to commercial proprietary software vendors keeping their development strategies to themselves. MySQL version numbers are used to track the releases and contain a two-part number for the product series and a single number for the release. For example, version 5.0.12 is the 12th release of the 5.0 product line.

■ **Tip** Always be sure to include the complete version number when corresponding with MySQL AB. Simply stating the "alpha release" or "latest version" is not clear enough to properly address your needs.

This multiple-release philosophy has some interesting side effects. It is not uncommon to encounter organizations that are using older versions of MySQL. In fact, I have encountered several agencies that I work with who are still using the version 4.*x* product lines. This philosophy has

virtually eliminated the upgrade shell game that commercial proprietary software undergoes. That is, every time the vendor releases a new version they cease development, and in many cases support, of the old version. With major architectural changes, customers are forced to alter their environments and development efforts accordingly. This adds a great deal of cost to maintaining product lines based on commercial proprietary software. The multiple-release philosophy frees organizations from this burden by permitting them to keep their own products in circulation much longer and with the assurance of continued support. Even when new architecture changes occur, as in the case of MySQL version 5.0, organizations have a much greater lead time and can therefore expend their resources in the most efficient manner allowed to them without rushing or altering their long-term plans.

While you are free to download any version of MySQL, you might want to first consider your use of the software. If you plan to use the software as an enterprise server in your own production environment, you may want to limit your download to the stable releases of the product line. On the other hand, if you are building a new system using the LAMP stack or another development environment, any of the other release stages would work for a development effort. Most will download the stable release of the latest version that they intend to use in their environment. For the purposes of the exercises and experiments in this book, any version (stage) of MySQL will work well.

MySQL AB recommends using the latest alpha series for any new development. What they mean is if you plan to add features to MySQL and you are participating in the global community of developers, you should add new features to the alpha stage. This permits the greatest opportunity (exposure) of your code to be tested prior to the last gamma stage (production release). You should also consider that while the stage of the version may indicate its state with respect to new features, you should not automatically associate instability with the early stages or stability with the later. Depending on your use of the software, the stability may be different. For example, if you are using MySQL in a development effort to build a new ecommerce site in the LAMP stack and you are not using any of the new features introduced during the development or alpha stage, the stability for your use is virtually the same as any other stage. The best rule of thumb is to select the version with the features that you need at latest stage of development.

Why Modify MySQL?

Modifying MySQL is not a trivial task. If you are an experienced C/C++ programmer and understand the construction of relational database systems, then you can probably jump right in. For the rest of us, we need take a moment to consider why we would want to modify a database server system and carefully plan our modifications.

There are many reasons why you would want to modify MySQL. Perhaps you require a database server or client feature that isn't available. Or maybe you have a custom application suite that requires a specific type of database behavior and rather than having to adapt to a commercial proprietary system, it is easier and cheaper for you to modify MySQL to meet your needs. It is most likely the case that your organization cannot afford to duplicate the sophistication and refinement of the MySQL database system, but you need something to base your solution on. What better way to make your application world-class than by basing it on a world-class database system?

■ **Note** If a feature is really useful and someone considers it beneficial, the beauty of open source is that the feature will work its way into the product. Someone, somewhere will contribute and build the feature.

Like all effective software developers, you must first begin by planning what you are going to do. Start with the planning devices and materials that you are most comfortable with and make a list of all of the things you feel you need the database server (or client) to do. Spend some time evaluating MySQL to see if any of the features you want already exist and make notes concerning their behavior. After you've completed this research, you will have a better idea of where the gaps are. This "gap analysis" will provide you with a concentrated list of features and modifications needed. Once you have determined the features you need to add, you can begin to examine the MySQL source code and experiment with adding new features.

■ **Warning** Always investigate the current MySQL features thoroughly when planning your modifications. You will want to examine and experiment with all of the SQL commands that are similar to your needs. Although you may not be able to use the current features, examining the existing capabilities will enable you to form a baseline or known behavior and performance that you can use to compare your new feature. You can be sure that the global community of developers will scrutinize any new feature and remove those they feel are best achieved using a current feature.

The best place to start learning the MySQL source code is to keep reading! This book will introduce you to the MySQL source code and provide you with knowledge of how to add new features as well as the best practices for what to change (and what not to change). Later chapters will also detail your options of how to get the source code and how to merge your changes into the appropriate code path (branch). You will also learn the details of MySQL AB's coding guidelines that specify how your code should look and what code constructs you should avoid.

What Can You Modify in MySQL? Are There Limits?

The beauty of open source software is that you have access to its source code for the software (as guaranteed by its respective open source license). This means you have access to all of the inner workings of the entire software. Have you ever wondered how the optimizer works in MySQL? You can find out simply by downloading the source code and working your way through it.

With MySQL, it isn't so simple. The source code in MySQL is often complex and difficult to read and understand. One could say the code has very low comprehensibility. Often regarded by the original developers as having a "genius factor," the source code can be a challenge for even the best C/C++ programmer.

While the challenges of complexities of the C/C++ code may be a concern, it in no way limits your ability or right to modify the software. Most developers modify the source code to add new SQL commands or alter existing SQL commands to get a better fit to their database needs. However, the opportunities are much broader than simply changing MySQL's SQL

behavior. You can change the optimizer, the internal query representation, or even the query cache mechanism.

One of the challenges you are likely to encounter will not be from any of your developers. The challenge may come from your senior technical stakeholders. For example, my recent modifications to the MySQL source code were challenged by senior technical stakeholders because I was modifying foundations of the server code itself. One stakeholder was adamant that my changes "flew in the face of 30 years of database theory and tried and true implementation." I certainly hope you never encounter this type of behavior, but if you do and you've done your research as to what features are available and how they do not meet (or partially meet) your needs, your answer should consist of indisputable facts. If you do get this question or one like it, remind your senior technical stakeholder that the virtues of open source software is that it can be modified and that it frequently *is* modified. You may also want to consider explaining what your new feature does and how it will improve the system as a whole for everyone. If you can do that, you can weather the storm.

Another challenge you are likely to face with modifying MySQL is the question "Why MySQL?" Experts will be quick to point out that there are several open source database systems to choose from. The most popular are MySQL, Firebird, PostgreSQL, and Berkeley DB. The reasons that you would choose to use MySQL in your development projects over some of the other database systems include the following:

- MySQL is a relational database management system that supports a full set of SQL commands. Some open source database systems like PostgreSQL are object relational database systems that use an API or library for access rather than accepting SQL commands. Some open source systems are built using architectures that may not be suited for your environment. For example, Apache Derby is based in Java and may not offer the best performance for your embedded application.

- MySQL is built using C/C++, which can be built for nearly all Linux platforms as well as Microsoft Windows and Macintosh OS. Some open source systems may not be available for your choice of development language. This can be an issue if you must port the system to the version of Linux that you are running.

- MySQL is designed as client/server architecture. Some open source systems are not scalable beyond a client-based embedded system. For example, Berkeley DB is a set of client libraries and is not a stand-alone database system.

- MySQL is a mature database server with a proven track record of stability. Some open source database systems may not have the install base of MySQL or may not offer the features you need in an enterprise database server.

Clearly, the challenges are going to be unique to the development needs and the environment in which the modifications take place. Whatever your needs are, you can be sure that you have complete access to all of the source code and that your modifications are limited only by your imagination.

MySQL Licensing Explained

MySQL is licensed as open source software under the GPL. The server and client software as well as the tools and libraries are all covered by the GPL. MySQL AB has made the GPL a major

focal point in their business model. They are firmly committed to the GNU open source community. Furthermore, all of the venture capitalists who sign on with MySQL AB are required to underwrite the same philosophy and license.

MySQL AB has gained many benefits by exposing their source code to the global community of developers. The source code is routinely evaluated by public scrutiny, third-party organizations regularly audit the source code, the development process fosters a forum of open communication and feedback, and the source code is compiled and tested in many different environments. No other database vendor can make these claims while maintaining world-class stability, reliability, and features.

MySQL is also licensed as a commercial product. A commercial license permits MySQL AB to own the source code (as described earlier) as well as own the copyright on the name, logo, and documentation (such as books). This is unique because most open source companies do not ascribe to owning anything; rather, their intellectual property is their experience and expertise. MySQL AB has retained the intellectual property of the software while leveraging the support of the global community of developers to expand and evolve the software. It should be noted that MySQL AB has its own full development team with over 100 employees worldwide. Although it is true that developers from around the world participate in the development of MySQL, MySQL AB employs many of them.

Some would consider this move by MySQL AB as a corruption of the original ideas of Stallman and the FSF. That isn't the case. MySQL AB has created an industry around open source database systems that is driven by the open source philosophy while retaining the ability to employ members of the same development industry. MySQL AB has shown it is possible to give away your ideas and still make money selling them.

This dual-license concept has created some confusion. Specifically, when should you use the GPL versus the commercial license? The GPL is best suited for general use of the software, participation in the global community of developers to add or refine features, and for conducting academic experiments. The commercial license is best suited to situations where you need warranties and assurances of capabilities (support) or when you use the software in mission-critical applications.

The subject of what license to use for modifications is also a source of some confusion. If you are planning features that are of interest to more than your own users, you should consider using the GPL and turn over your changes to MySQL AB. Although this means you are giving away your rights to own those changes, you are gaining the world-class support and all of the other benefits of the MySQL system. If you are making modifications that are of use to only you and your unique needs and you are not repackaging or distributing the changes (in any way), then you can use either license.

If you use the GPL and do not share your modifications, you will not get any support for the modifications and it will be your responsibility to maintain them. This could be a problem if you decide to upgrade to a new version of MySQL. You will have to make all of the modifications all over again. This may not be a difficult challenge, but it is something that will require careful planning. MySQL AB provides a number of support options for users of the GPL. The MySQL web site (`www.mysql.com/support/community_support.html`) has links for subscribing to a variety of free mailing lists, forums, and bug reporting. Consulting services and training are also available for a fee.

If you use the commercial license, you have the option of purchasing support from MySQL AB to assist you in making the modifications. You can even purchase rights that permit you to maintain ownership of the changes. This is especially important if you plan to repackage and redistribute the source code to your own customers. Table 1-1 summarizes the various support

options currently available from MySQL AB. These support packages, called the MySQL Network, are available regardless of which license you choose to use, but may have certain restrictions associated with using the GPL.

Table 1-1. *MySQL Network Support Options*

Feature	Basic	Silver	Gold	Platinum
Software maintenance and upgrades	Yes	Yes	Yes	Yes
Service advisors available	Yes	Yes	Yes	Yes
Access to free knowledge base	Yes	Yes	Yes	Yes
Incident reports	2	unlimited	unlimited	unlimited
Phone support		8×5 (M–F)	24×7	24×7
Initial response time (max)	2 business days	4 hours	2 hours	30 minutes
Emergency response time (max)			30 minutes	30 minutes
Remote troubleshooting			Yes	Yes
Schema review				Yes
Query review				Yes
Performance tuning				Yes
Code reviews (client development)				Yes
Code reviews (user-defined functions)				Yes
Code reviews (server development)				Yes
Dedicated account manager				Option
Indemnification			Option	Option

■ **Note** MySQL has created the indemnification program to assist customers in copyright and patent infringement disputes.

So, Can You Modify MySQL or Not?

You may be wondering after a discussion of the limitations of using open source software under the GNU public license if you can actually modify it after all. The answer is simply, yes, you can!

You can modify MySQL under the GPL provided, of course, that if you intend to distribute your changes you surrender those changes to the owner of the project and thereby fulfill your

obligation to participate in the global community of developers. If you are experimenting or using the modifications for educational purposes, you are not obligated to turn over your changes. Naturally, the truth of the matter comes down to the benefits of the modifications. If you're adding capabilities that can be of interest to someone other than yourself, you should share them.

You can also modify MySQL under the commercial license. In this case, either you're intending to use the modifications for your own internal development or you're bundling MySQL or embedding MySQL in your own commercial product.

Whatever licensing method you choose, the opportunity to modify the system is yours to take.

Guidelines for Modifying MySQL

Take care when approaching a task such as modifying a system like MySQL. A relational database system is a complex set of services layered in such a way as to provide fast, reliable access to data. You would not want to open the source code and start plugging in your own code to see what happens (but you're welcome to try). Instead, you should plan your changes and take careful aim at the portions of the source code that pertain to your needs.

Having modified large systems like MySQL, I want to impart a few simple guidelines that will make your experience with modifying MySQL a positive one.

The first thing you should do is decide which license you are going to use. If you are using MySQL under an open source license already and can implement the modifications yourself, you should continue to use the GPL. In this case, you are obligated to perpetuate the open source mantra and give back to the community in exchange for what was freely offered. Under the terms of the GPL, the developer is bound to make these changes available. If you are using MySQL under the commercial license or need support for the modifications, you should purchase the appropriate MySQL Network support and consult with MySQL AB on your modifications. However, if you are not going to distribute the modifications and can support them for future versions of MySQL, you do not need to change to the commercial license or change your commercial license to the GPL.

Another suggestion is to keep a developer's journal and keep notes of each change you make or each interesting discovery you find. Not only will you be able to record your work step by step, but you can also use the journal as a way to document what you are doing. You will be amazed at what you can discover about your research by going back and reading your past journal entries. I have found many golden nuggets of information scrawled within my engineering notebooks.

While experimenting with the source code, you should also make notes in the source code itself. You can annotate the source code with a comment line or comment block before and after your changes. This makes it easy to locate all of your changes using your favorite text parser or search program. The following demonstrates one method for commenting your changes:

```
/* BEGIN MY MODIFICATION */
/* Purpose of modification: experimentation. */
/* Modified by: Chuck */
/* Date modified: 3/19/2006 */
if (something_interesting_happens)
{
  do_something_cool;
}
/* END MY MODIFICATION */
```

Lastly, do not be afraid to explore the free knowledge base and forums on the MySQL web site or seek the assistance of the global community of developers. These are your greatest assets. However, be sure you have done your homework before you post to one of the forums. The fastest way to become discouraged is to post a message on one of the forums only to have someone reply with a curt (but polite) reference to the documentation. Make your posts succinct and to the point. You don't need to elaborate on the many reasons why you're doing what you're doing—just post your question and provide all pertinent information about the issue you're having. Also take care to make sure you are posting to the correct forum. Most forums are moderated and if you are ever in doubt, consult the moderator to ensure you are posting your topic in the correct forum.

A Real-World Example: TiVo

Have you ever wondered what makes your TiVo tick? Would you be surprised to know that it runs on a version of embedded Linux?

Jim Barton and Mike Ramsay designed the original TiVo product in 1997. It was pitched as a home network–based multimedia server serving streaming content to thin clients. Naturally, a device like this must be easy to learn and even easier to use, but most importantly it must operate error free and handle power interruptions (and user error) gracefully.

Barton was experimenting with several forms of Linux and while working at Silicon Graphics (SGI), sponsored a port of Linux to the SGI Indy platform. Due mainly to the stable file system, network, memory handling, and developer tool support, Barton believed it would be possible to port a version of Linux to the TiVo platform and that Linux could handle the real-time performance goals of the TiVo product.

However, Barton and Ramsay faced a challenge from their peers. Many at that time viewed open source with suspicion and scorn. Commercial software experts asserted that open source software would never be reliable in a real-time environment. Furthermore, they believed that basing a commercial proprietary product on the GPL would not permit modification and that if they proceeded, the project would become a nightmare of copyright suits and endless legal haranguing. Fortunately, Barton and Ramsay were not deterred and studied the GPL carefully. They concluded that not only was the GPL viable, it would permit them to protect their intellectual property.

Although the original TiVo product was intended to be a server, Barton and Ramsay decided that the bandwidth wasn't available to support such lofty goals. Instead, they redesigned their product to a client device, called the TiVo Client Device (TCD), which would act like a sophisticated video recorder. They wanted to provide a for-fee service to serve up the television guide and interface with the TCD. This would allow home users to select the shows they wanted in advance and program the TCD to record them. In effect, they created what is now known as a digital video recorder (DVR).

The TCD hardware included a small, embedded computer with a hard drive and memory. Hardware interfaces were created to read and write video (video in and video out) using a MPEG 2 encoder and decoder. Additional input/output (I/O) devices included audio and telecommunications (for accessing the TiVo service). The TCD also had to permit multiprocessing capabilities in order to permit the recording of one signal (channel) while playing back another (channel). These features required a good memory and disk management subsystem. Barton and Ramsay realized these goals would be a challenge for any control system. Furthermore, the video interface must never be interrupted or compromised in any way.

What Barton and Ramsay needed most was a system with a well-developed disk subsystem, supported multitasking, and the ability to optimize hardware (CPU, memory) usage. Linux therefore was the logical choice of operating systems for the TCD. Production goals and budget constraints limited the choice of CPU. The IBM PowerPC 403GCX processor was chosen for the TCD. Unfortunately, there were no ports of Linux that ran on the chosen processor. This meant Barton and Ramsay would have to port Linux to the processor platform.

While the port was successful, Barton and Ramsay discovered they needed some specialized customizations of the Linux kernel to meet the needs and limits of the hardware. For example, they bypassed the file system buffer cache in order to permit faster movement, or processing, of the video signals to and from user space. They also added extensive performance enhancements, logging, and recovery features to ensure that the TCD could recover quickly from power loss or user error.

The application that runs the TCD was built on Linux-based personal computers and ported to the modified Linux operating system with little drama—a testament to the stability and interoperability of the Linux operating system. When Barton and Ramsay completed their porting and application work, they conducted extensive testing and delivered the world's first DVR in March 1999.

The TCD is one of the most widely used consumer product running a customized embedded Linux operating system. Clearly, the TCD story is a shining example of what you can accomplish by modifying open source software. The story doesn't end here, though. Barton and Ramsay published their Linux kernel port complete with the source code. Their enhancements have found their way into the latest versions of the Linux kernel.

CONVINCING YOUR BOSS TO MODIFY OPEN SOURCE SOFTWARE

If you have an idea and a business model to base it on, going the open source route can result in a huge time savings in getting your product to market. In fact, your project may become one that can save a great deal of development revenue and permit you to get the product to market faster than your competition. This is especially true if you need to modify open source software—you have already done your homework and can show the cost benefits of using the open source software.

Unfortunately, many managers have been conditioned by the commercial proprietary software world to reject the notion of basing a product on open source software to generate a revenue case. So how do you change their minds? Use the TiVo story as ammunition. Present to your boss the knowledge you gained from the TiVo story and the rest of this chapter to dispel the myths concerning GPL and reliability of open source software. Be careful, though. If you are like most open source mavens, your enthusiasm can often be interpreted as a threat to the senior technical staff.

Make a list of the technical stakeholders who adhere to the commercial proprietary viewpoint. Engage these individuals in conversation about open source software and answer their questions. Most of all, be patient. These folks aren't as thick as you may think and will eventually come to share your enthusiasm.

Once you've got the senior technical staff educated and bought into the open source mind-set, reengage your management with a revised proposal. Be sure to take along a member of the senior technical staff as a shield (and a voice of reason). Winning in this case is turning the tide of commercial proprietary domination.

Summary

In this chapter, you explored the origins of open source software and the rise of MySQL to a world-class database management system. You learned what open source systems are and how they compare to commercial proprietary systems. You saw the underbelly of open source licensing and discovered the responsibilities of being a member of the global community of developers.

You also received an introduction to developing with MySQL and learned characteristics of the source code and guidelines for making modifications. You read about MySQL AB's dual-license practices and the implications of modifying MySQL to your needs. Finally, you saw an example of a successful integration of an open source system in a commercial product.

In the chapters ahead, you will learn more about the anatomy of a relational database system and how to get started customizing MySQL to your needs. Later in Parts 2 and 3 of this book, you will be introduced to the inner workings of MySQL and the exploration of the most intimate portions of the code.