MySQL and Java Developer's Guide

Mark Matthews
Jim Cole
Joseph D. Gradecki

Wiley Publishing, Inc.

MySQL and Java Developer's Guide

Mark Matthews
Jim Cole
Joseph D. Gradecki

Wiley Publishing, Inc.

Publisher: Robert Ipsen
Editor: Robert M. Elliott
Managing Editor: Vincent Kunkemueller
Book Producer: Ryan Publishing Group, Inc.

Copyeditor: Elizabeth Welch
Proofreader: Nancy Sixsmith
Compositor: Gina Rexrode

Designations used by companies to distinguish their products are often claimed as trademarks. In all instances where Wiley Publishing, Inc., is aware of a claim, the product names appear in initial capital or ALL CAPITAL LETTERS. Readers, however, should contact the appropriate companies for more complete information regarding trademarks and registration.

This book is printed on acid-free paper. ∞

Copyright © 2003 by Wiley Publishing, Inc. All rights reserved.

Published by Wiley Publishing, Inc., Indianapolis, Indiana

Published simultaneously in Canada.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as permitted under Section 107 or 108 of the 1976 United States Copyright Act, without either the prior written permission of the Publisher, or authorization through payment of the appropriate per-copy fee to the Copyright Clearance Center, Inc., 222 Rosewood Drive, Danvers, MA 01923, (978) 750-8400, fax (978) 750-4470. Requests to the Publisher for permission should be addressed to the Legal Department, Wiley Publishing, Inc., 10475 Crosspoint Blvd., Indianapolis, IN 46256, (317) 572-3447, fax (317) 572-4447, E-mail: permcoordinator@wiley.com.

Limit of Liability/Disclaimer of Warranty: While the publisher and author have used their best efforts in preparing this book, they make no representations or warranties with respect to the accuracy or completeness of the contents of this book and specifically disclaim any implied warranties of merchantability or fitness for a particular purpose. No warranty may be created or extended by sales representatives or written sales materials. The advice and strategies contained herein may not be suitable for your situation. You should consult with a professional where appropriate. Neither the publisher nor author shall be liable for any loss of profit or any other commercial damages, including but not limited to special, incidental, consequential, or other damages.

For general information on our other products and services please contact our Customer Care Department within the United States at (800) 762-2974, outside the United States at (317) 572-3993 or fax (317) 572-4002.

Trademarks: Wiley, the Wiley Publishing logo and related trade dress are trademarks or registered trademarks of Wiley Publishing, Inc., in the United States and other countries, and may not be used without written permission. All other trademarks are the property of their respective owners. Wiley Publishing, Inc., is not associated with any product or vendor mentioned in this book.

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data:

Matthews, Mark.
MySQL™ and Java™ developer's guide / Mark Matthews.
p. cm.
ISBN 0-471-26923-9 (PAPER/WEBSITE)
1. SQL (Computer program language) 2. Java (Computer program language) I. Title.
A76.3.S67M38 2003
005.75'65—dc21

2002155887

Printed in the United States of America

10 9 8 7 6 5 4 3 2 1

Dedication

To my wife Diane, for all her support in my "geeky" endeavors, and to our new daughter Lauren.

I would also like to dedicate this work to Monty, David, and the rest of the fine group of developers at MySQL AB. Without their contribution to the software community and dedication to free software and open source ideals, this book would not have been possible.

--Mark Matthews

I would like to dedicate this book to my parents. Their ever-present love and encouragement have made so many things possible.

—Jim Cole

This book is dedicated to the trinity: God, Jesus Christ, and the Holy Spirit.

—Joseph D. Gradecki

Acknowledgments

I need to acknowledge the patience and support of my beautiful and loving wife and our boys. Thank you for the opportunity to be your husband and father. Tim, thank you for the opportunities. Jim, welcome to this new adventure and I look forward to many more in the future. Thank you to Liz Welch for the excellent review.

Mark Matthews is the creator of Connector/J and its predecessor MM.MySQL, the Java JDBC driver for MySQL. Last year, he joined MySQL AB to further develop Java support in MySQL. Mark specializes in Java, MySQL, XML, and DHTML solutions and has architected major Web applications projects, including a GIS-based retail analytics package. Mark has also taught classes in both Java and UML.

Jim Cole is a senior software engineer specializing in Internet and knowledge management systems. He is an active developer working in Java, C++, Perl, and PHP. He also serves as a system administrator for several Web-based projects, where his duties include custom software development, database management, and security maintenance.

Joseph D. Gradecki is a software engineer at Comprehensive Software Solutions, where he works on their SABIL product, an enterprise-level securities processing system. He has built numerous dynamic, enterprise applications using Java, AspectJ, servlets, JSPs, Resin, MySQL, BroadVision, XML, and more. He has also built P2P distributed computing systems in a variety of languages including Java/JXTA, C/C++, and Linda. He holds Bachelors and Masters degrees in Computer Science and is currently obtaining his PhD.

Introduction

Have you ever been assigned a project and realized that you had no idea how you were going to accomplish it? Many developers have experienced this feeling when asked to interface their code with a database. With a few exceptions, most developers were busy learning Lisp, linked lists, and big-O notation during their formal education instead of learning the fundamentals of relationship database management systems. When the time comes to interface their code with a database, they turn to a book like the one you are holding.

Your challenge might be to write a Web-based system using servlets and Enterprise JavaBeans (EJBs) to transfer shipping records from the home office in Bend, Oregon, to a satellite shipper in New Jersey. Or perhaps your father just opened his new medical office and you volunteered to create a scheduling system over the weekend.

Whatever the situation, interfacing an application to a database is one of the most fundamental tasks a developer is required to perform. This book is designed for developers who either have a pressing task ahead of them or who are curious about how to read database information into their application.

By combining MySQL, the number-one open source database available, with Java, the most portable language ever developed, you can create an undisputable champion. So, sit back in your desk chair with a hot chocolate and get ready to supercharge your coding.

What's in This Book

The primary goal of *MySQL and Java Developer's Guide* is to provide a comprehensive approach to writing code from a Java application to a MySQL database using the industry standard: JDBC. As you will see later in this Introduction, the chapter titles indicate what area of database connectivity and manipulation they cover. The chapters are ordered to reflect the JDBC specification, but we aren't here to simply describe the specification.

We wrote all of the material in the book to highlight how MySQL's Connector/J JDBC driver achieves the interfacing of MySQL with Java while maintaining the spirit of the specification. With this in mind, we provide example code using all major forms of Java development, including

- Applications
- Applets
- Servlets
- JSPs
- EJBs

As you work in Java and JDBC, you will see the true power of the specification. You can write database access code in a Java application and move the code to a servlet with little if any changes. In the case of EJBs and container-managed persistence, we devoted a full chapter to dealing with database access without the cumbersome details of SQL.

We designed the layout of the book to move you through the entire process of writing Java code needed to access a back-end database. Developing the database is one of the first things that you must accomplish in this process. While we don't delve deeply into the theory of database development, you learn how to create databases in MySQL, administer those databases, and handle such details as granting access permissions. From there, we take you into an examination of the MySQL Connector/J driver and how it accomplishes its goal of portable database access. The remainder of the book steps you through Java code that highlights how to accomplish database tasks such as the following:

- Querying and updating
- Handling ResultSets
- Using transactions
- Handling typing issues between JDBC and MySQL
- Working with metadata
- Addressing efficiency issues

Once you're familiar with these concepts, we present a complete application that pulls it all together. Our application illustrates how you can create to a simple authorization service. Using a combination of JSP, servlets, and EJBs, the service allows new users to create accounts, recall the account, and verify a username/password combination. The system is designed to be interactive using JSP pages, which are handled on the server using servlets. The JSPs can be bypassed using the servlets directly. All of the critical information is kept on the database for persistence and management needs.

After reading this book, you should know how to interface Java to MySQL and be able to use the many examples for reference.

NOTE

All the code and examples in this book can be found on the the support Web site at www.wiley.com/compbooks/matthews.

Who Should Read This Book

This book is written for Java developers who need to interface their code to a back-end database. The book's specifics deal with MySQL and Connector/J, but this doesn't limit the information because JDBC is designed to be portable against many databases. If you aren't using MySQL, you still find valuable information.

You don't need to know much about databases—we have included several chapters that provide all of the basics necessary to create databases and make sure they are operational. Keep in mind that we didn't intend these chapters to replace a good reference on MySQL, though.

We do expect that you are an experienced Java developer who is comfortable with the language. This book explains a combination of Java delivery methods, including applications, applets, beans, and EJBs; you may want to begin with what you know best and expand from there.

The Technology Used

In this book, we use the latest Java Developments Kits (JDK) available from Sun at the time of writing. The JDKs we used include J2SE 1.4.0 and J2EE 1.3.1. The Java examples are used in a mixed environment, including Windows 2000/XP, Linux Mandrake, and Linux Slackware. For the most part, we developed the examples using simple text editors and compiled them using the Java command-line compiler. However, all the examples should work just fine in an IDE such as JBuilder.

Two different versions of MySQL are used throughout this book: 4.0.4 and 3.23.52. JDBC connectivity is handled using MySQL's Connector/J driver, and we cover both versions 2.0.14 and development 3.0.1.

Book Organization

The first four chapters of this book provide an overview of databases, JDBC, and installation of the tools you will be using. The remainder of the book is an in-depth guide to building database applications with MySQL, Connector/J, JDBC, and Java.

Chapter 1: An Overview of MySQL

MySQL is one of the most popular open source database systems available today, and it is used as the back-end data storage device for many personal and corporate Web sites. Java is the most portable language in use today and continues to be improved with each new release. In this chapter, we provide a brief overview of each product and begin the discussion of how to interface the two and thus allow Java applications to have access to a vast array of information.

Chapter 2: JDBC and Connector/J

As shown in Chapter 1, JDBC facilitates the interface between Java and MySQL. The JDBC specification defines the interfaces and classes necessary for any Java application, applet, servlet, and so forth to make calls to an underlying database. Because the JDBC specification isn't specific to any one database system, manufacturers create JDBC drivers for their specific database. In this chapter, we discuss the history of JDBC, how it started, and its progress into a version 3.0 specification. We examine in depth the MySQL JDBC driver called Connector/J, and look at its history as the MM.MySQL JDBC driver as well as its future.

Chapter 3: Working with MySQL SQL

Before we delve into the concepts surrounding the interface between Java and MySQL, this chapter provides a basic overview of databases and SQL. Topics include basic concepts behind databases, simple database design, database normalization, and data manipulation.

Chapter 4: Installing MySQL, Java, and Connector/J

All of the coding examples in this book are built using MySQL as the primary database, Java as our coding language, and Connector/J, MySQL's JDBC driver. Although the installation of these components isn't overly difficult, this chapter provides comprehensive instructions for obtaining all of the necessary components and performing a step-by-step installation. We also provide simple examples for testing the installation.

Chapter 5: Using JDBC with Java Applications and Applets

This chapter is the first in a series on the use of Java to access a MySQL database using JDBC. Some of the basic functionality discussed includes loading the JDBC driver, connecting to a local or remote database, building JDBC statements in preparation for queries, executing queries against the MySQL database, working with ResultSets, and accessing MySQL-specific functionality through JDBC.

Chapter 6: Achieving Advanced Connector/J Functionality with Servlets

At this point, you've learned the basics, and it's time to expand into the more advanced topics. This chapter is designed to expand your understanding of SQL, MySQL, and JDBC. The topics include updatable ResultSets, PreparedStatements, date/time types, BLOBs and CLOBs, and joins.

Chapter 7: MySQL Type Mapping

One of the fundamental issues associated with databases and programming language is determining the correct mapping from one to the other. While programming languages have a large variety of types, including simple ones like integer, they also allow more complex ones, like classes. Databases, on the other hand, are limited in their choices for the types of data that can be stored. In the middle of this situation is the JDBC driver. This chapter discusses the types available on the MySQL database, how JDBC interprets those types, and the resulting Java type produced by the mapping.

Chapter 8: Transactions and Table Locking with Connector/J

In a simple world, information is stored in a single table of a database. When you have to update information or insert a new row, you can use a single query. However, most modern databases store information across several different tables to increase the normalization of the tables. In this situation, when you have to update information or insert new rows, you must write two

queries instead of one. This chapter looks at inserting multiple pieces of information into multiple tables, what problems can arise, and how transactions can be used to solve these problems.

Chapter 9: Using Metadata

After a query is performed against a MySQL database, the information is returned in a ResultSet object. This object includes all of the rows and columns specific to the query performed. In many cases, additional information is needed about the data, including the name of the columns in the result, the precision of the data in a float column, the maximum length of a column, and maybe even information about the server from which the data was returned. In this chapter, we discuss pulling metadata about both the database and a ResultSet that contains information from a query.

Chapter 10: Connection Pooling with Connector/J

In many cases, a JDBC driver requires between 4 and 10 different communications with a database application before a connection can be established and returned to the requesting application. If an application is constantly creating connections, doing its business, and then closing the connection, the application suffers in its potential performance. To overcome the connection performance problem, you can use a connection pool. This chapter provides a comprehensive introduction to connection pools, presents valuable statistics for creating database connections, and demonstrates how to use the connection pooling mechanisms within JDBC.

Chapter 11: EJBs with MySQL

Enterprise JavaBeans (EJBs) provide the framework for building applications that can handle the rigors of enterprise-level applications. In addition, EJBs can be distributed across a network or a farm of servers. In this chapter, we cover the basic EJB programming model, using DataSources and JNDI, and building session beans to access MySQL. We also discuss container-managed persistence and bean-managed entity beans.

Chapter 12: Building a General Interface for MySQL

All of the chapters to this point have featured relatively simple examples using Java applications, applets, servlets, and JSP to illustrate the finer points of accessing a MySQL database using Java and Connector/J. This chapter pulls it

all together using a Certificate Authority application. Using JSP, servlets, and EJB, the application shows how to create new accounts, request certificates, and enable the verification of certificates. All of the information, including the binary certificate, is stored in a MySQL database with multiple tables.

Chapter 13: Database Administration

Once you have a good knowledge of the MySQL database system as well as the fundamentals described in the previous chapters for accessing the data from Java, you must learn some database administration basics. In this chapter, we examine many of the functions within MySQL that benefit administrators, such as granting/revoking permissions, providing security within the server, and recovering from disasters.

Chapter 14: Performance and Tuning

Once the application is written and the information is safely sitting in a database, the users get the final say on whether or not the application meets their performance requirements. If the application isn't running at an appropriate level, you have a couple of options. First, you can profile the Java code to determine where the application is spending the most time and then rework the code associated with the problem areas. Second, you can tune the MySQL server and create indexes for the database tables. In this chapter, we provide the necessary information on performing these two options.

Appendix A: MySQL Development and Test Environments

We developed and tested all of the code in this book on several different test architectures in order to provide a representative reference. This appendix briefly describes those environments and lists the installed software. In addition, we offer some notes for reproducing the configuration.

Appendix B: Databases and Tables

In this appendix, we list all databases and tables used in the examples throughout this book.

Appendix C: The JDBC API and Connector/J

This appendix is a comprehensive review of the entire JDBC API, with annotations for Connector/J. Code snippets are provided to show at a quick glance how to use the various interfaces, classes, and methods.

Appendix D: MySQL Functions and Operators

The list of MySQL functions and operators in this appendix will help you determine when the database should handle computations versus the application. Each function and operator is described, and an example of its use is given.

Appendix E: Connector/J Late-Breaking Additions

The most current, up-to-date additions to Connector/J as it moves from gamma to production version.

CHAPTER 1

An Overview of MySQL

In this chapter, we explain why you might choose to use a database system with your software. We also provide an overview of the MySQL database server and the Connector/J JDBC driver.

For many years, large corporations have enjoyed the ability to deploy relational database management systems (RDBMSs) across their enterprise. Companies have used these systems to collect vast amounts of data that serve as the "fuel" for numerous applications that create useful business information.

Until recently, RDBMS technology has been out of reach for small businesses and individuals. Widely used RDBMS systems such as Oracle and DB2 require complex, expensive hardware. License fees for these systems are in the tens to hundreds of thousands of dollars for each installation. Businesses must also hire and retain staff with specialized skill sets to maintain and develop these systems. Smaller enterprises have relied on systems like Microsoft Access and FoxPro to maintain their corporate data.

Early on, during the explosive growth of the Internet, open source database systems like mSQL, Postgres (now PostgreSQL), and MySQL became available for use. Over a relatively short amount of time, the developers of these systems have provided a large subset of the functionality provided by the expensive commercial database systems. These open source database systems also run on less-expensive commodity hardware, and have proven in many cases to be easier to develop for and maintain than their commercial counterparts.

Finally, smaller businesses and individuals have access to the same powerful level of software tools that large corporations have had access to for over a decade.

Why Use an RDBMS?

Almost every piece of software that has been developed needs to *persist* or store data. Once data has been persisted, it is natural to assume that this data needs to be retrieved, changed, searched, and analyzed.

You have many options for data persistence in your software, from rolling your own code, to creating libraries that access flat files, to using full-blown RDBMS systems. Factors to consider when choosing a persistence strategy include whether you need multiuser access, how you will manage storage requirements, whether you need transactional integrity, and whether the users of your software need ad hoc query capability. RDBMSs offer all of this functionality.

Multiuser Access

Many programs use flat files to store data. Flat files are simple to create and change. The files can be used by many tools, especially if they are in comma- or tab-delimited formats. A large selection of built-in and third-party libraries is available for dealing with flat files in Java. The java.util.Properties class included with the Java Development Kit is one example.

Flat file systems can quickly become untenable when multiple users require simultaneous access to the data. To prevent corrupting the data in your file, you must lock the file during changes, and perhaps even during reads. While a file is locked, it cannot be accessed by other users. When the file becomes larger and the number of users increases, this leads to a large bottleneck because the file remains locked most of the time—your users are forced to wait until they can have exclusive access to the data.

RDBMSs avoid this situation by employing a number of locking strategies at varying granularities. Rather than using a single lock, the database system can lock an individual table, an individual page (a unit of storage in the database, usually covering more than one row), or an individual row. This increases throughput when multiple users are attempting to access your data, which is a common requirement in Web-based or enterprise-wide applications.

Storage Transparency

If you use flat files in your software, you are also responsible for managing their storage on disk. You have to figure out where and how to store the data, and

every time the location or layout of the files changes, you are required to change your software. Once the datasets your software is storing become numerous or large, the storage management process becomes cumbersome.

Using a database system gives you "storage transparency." Your software does not care where and how the data is stored. The data can even be stored on some other computer and accessed via networking protocols.

Transactions

When you have more than one user accessing and changing your data, you want to make these changes *transactional*. Transactions group operations on your data into units of work that meet the *ACID* test. The ACID test concept is best illustrated with a commonly used example from the banking industry.

Jack and Jill share a joint checking account with a balance of $1000. They are both performing various operations, such as deposits, withdrawals, and transfers, on the account. Let's see how the four aspects of the ACID test come into play:

- **Atomicity:** All changes made during a transaction are made successfully, or in the case of failure, none are made. If any operation fails during the transaction, then the entire transaction is rolled back, leaving your data in the state it was before the transaction was started. For example, suppose Jack is making a transfer of $500 from his checking account to a savings account. Sometime between the withdrawal of the $500 from the checking account and the deposit of $500 to the savings account, the software running the banking system crashes. Jack's $500 has disappeared! With atomicity, either the entire transfer would have happened, or none of it would have happened, leaving Jack a much happier customer than he is now.
- **Consistency:** All operations transform the database from one consistent state to another consistent state. Consistency is defined by how the database schema is designed and whether integrity constraints such as foreign keys are used. The database management system is responsible for ensuring that transactions do not violate the database schema or integrity constraints. For example, the bank's database developers have declared in the database schema that the balance of an account cannot be empty, or "null." If any transaction attempts to set the balance to an empty value, the transaction will be aborted and any changes rolled back.
- **Isolation:** A transaction's changes are not made visible to other transactions until they are committed under the atomicity rule described earlier. This is best demonstrated by what happens when month-end reports are generated. Let's say that Jack is performing the transfer transaction outlined in the atomicity example, and at the same time you are generating his

monthly statement. Without isolation, the monthly statement might show the withdrawal from the checking account but not the deposit into the savings account. This discrepancy would make it impossible for Jack or the bank to balance their books.

- **Durability:** Once completed, a transaction's changes are never lost through system or hardware crashes. If Jill has paid for $50 worth of groceries with her debit card at the grocery store and the transaction succeeds, even if the database software crashes immediately after the transaction competes, it won't forget that her checking account balance is $50 lower.

Until recently, MySQL did not comply with all components of the ACID test. However, with the new BDB and InnoDB table types (supported in MySQL 3.23 and MySQL 4.0), MySQL can now pass the ACID test.

Not all software requires the robustness (or the associated overhead) of transaction semantics. MySQL is one of the only databases that enable you to decide what level of robustness you need on a table-by-table basis. This becomes important when you are trying to maximize performance, especially when much of the data is read-only (such as in a product catalog).

Searching, Modifying, and Analyzing Data

Any time you store a significant amount of data with your software, your users want to search, modify, and analyze the data you have stored. If you are using flat files, you most likely have to develop this functionality yourself.

As your data stored in flat files takes up more and more space, it takes longer and longer to search. A common solution to this problem is to create an index for your data. Indexes are basically shortcuts to finding a particular piece of data, usually using some sort of key. If you need to develop indexing functionality yourself, you have to learn about data structures, such as hashes and B-trees, and how to store these indexes alongside your data. In addition, you must learn how to implement the index in your software. If you use an RDBMS, you can tell the database system what data you think people will search on, and it does all of the fancy indexing for you.

Users of your software also want to retrieve, modify, and analyze the data you have stored. They expect that your system knows how to compute such values as sums, averages, minimums, and maximums to be used for updating related data or analyzing existing data. They expect that your software will be able to sort the data or group the data by similar attributes. All of this functionality requires you to implement numerous functions and algorithms. If you use an RDMBS, all of these features are built in.

Ad Hoc Queries

It is likely that your software will need to retrieve stored data using arbitrary parameters, otherwise known as *ad hoc* queries. This becomes difficult with flat files because they are not self-describing, and every file layout is different. You also need to consider how you are going to read the data for these queries from your persistent storage mechanism.

Many RDBMSs use SQL (Structured Query Language) for manipulating data. SQL is a declarative language in that you declare *what* data you want, not the procedure for *how* to get it. SQL is also an accepted and widely used standard, so a large set of tools are available (JDBC and Enterprise Java Beans, among them) to help you work with it.

After outlining all of the benefits of an RDBMS, I hope you are ready to consider using one for your software projects. The next question to ask is "Why choose MySQL?"

Why Choose MySQL?

As was the case with many other open source projects, MySQL was first created by someone who needed a better tool to get a specific job done. Monty Widenius and David Axmark started out with another open source project (MSQL), but found that it lacked some features that they needed. They decided to develop their own database system that met their specific requirements. They started building MySQL by using some low-level database storage code they had already developed for other projects and layered a multithreaded server, SQL parser, and client-server protocol on top. They also structured the API for MySQL to appear very similar to MSQL in order to make it easier for developers to port their MSQL-based software to MySQL.

MySQL was eventually released in source-code form, under a proprietary license. Eventually, this license was changed to the GNU General Public License (GPL), which in most cases allows the software to be used without license cost. However, in certain situations you must purchase a commercial license. The exact terms of the license are available in the documentation that ships with MySQL or on the Web at www.mysql.com. Commercial support is also available for those who need it from MySQL-AB, the company that was created by Monty and David to support the continued development of the MySQL software.

The requirements that Monty and David originally had for MySQL were that it be as fast as possible, while still being stable, simple to use, and able to meet the needs of the majority of database developers. Even today, feature requests for future MySQL development are weighed carefully against these original

requirements, and are implemented only when and if the original requirements can be met as much as possible.

Over the years, MySQL has evolved into an RDBMS that has the following core features:

- **Portability:** MySQL runs on almost every flavor of Unix, as well as Windows and MacOS X. You can obtain binaries or source code for the MySQL server as well as the tools that access it. More ports of the software become available every day. It is almost a given that MySQL will run on whatever operating system you have available.
- **Speed:** Using techniques such as efficient indexing mechanisms, in-memory temporary tables, and highly optimized join algorithms, MySQL executes most queries much faster than most other database systems.
- **Scalability:** Because of its modularity and its flexibility in configuration, MySQL can run in systems varying in size from embedded systems to large multiprocessor Unix servers hosting databases with tens of millions of records. This scalability also allows you to run a copy of MySQL on a developer-class machine, and later use the same database system on a larger machine in production. Because it is multithreaded, MySQL efficiently utilizes resources for multiple users, compared to other database servers that start full-fledged processes for each user. It is not uncommon to hear of MySQL installations supporting thousands of concurrent users.
- **Flexibility:** MySQL lets you choose the table types you need to meet your software's requirements, ranging from in-memory heap tables, fast on-disk MyISAM tables, merge tables that group together other sets of tables to form larger "virtual" tables, and transaction-safe tables such as InnoDB. MySQL is also very tunable and includes many parameters that can be changed to increase performance for a given solution. However, MySQL comes with sensible defaults for these parameters, and many users never have to tune MySQL to reach a performance they are happy with.
- **Ease of use:** MySQL is easy to install and administer. While other database systems require special knowledge and training, not to mention special operating system configurations, MySQL can be installed in less than 10 minutes if you've done it before. Even if you are a newcomer, you should be able to install MySQL in under an hour. Once it's installed, MySQL requires little maintenance and administration other than adding or changing user permissions and creating or removing databases.
- **Fine-grained security model:** You can restrict users' rights from an entire database down to the column level based on login name, password, and the hostname that users are connecting from. This allows you to create secure systems by partitioning responsibilities and capabilities of different users and applications to prevent unauthorized modification or retrieval of data.