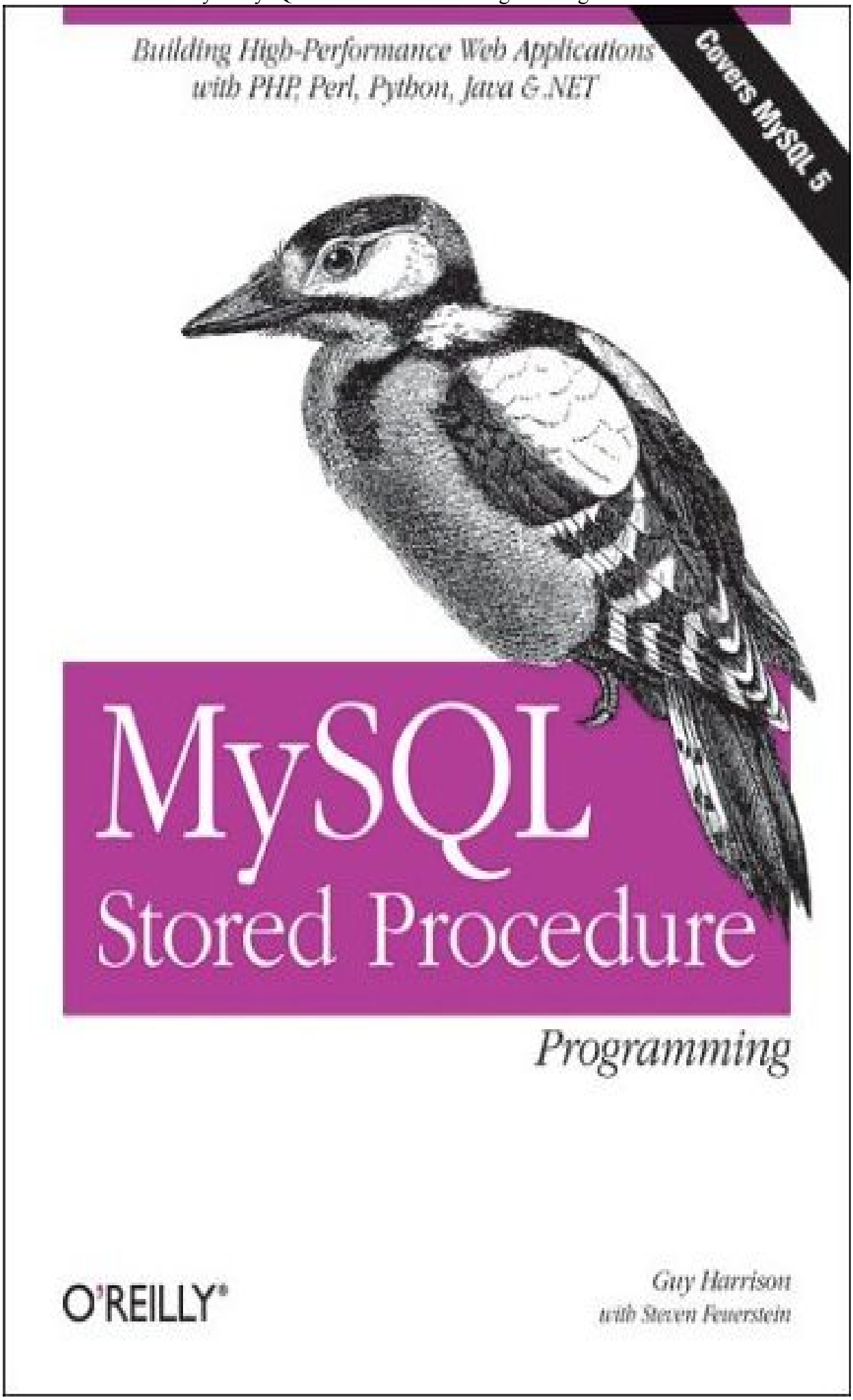

Building High-Performance Web Applications
with PHP, Perl, Python, Java & .NET
Covers MySQL 5
MySQL
Stored Procedure
Programming
O'REILLY®
Guy Harrison
with Steven Feuerstein

MySQL Stored Procedure Programming
By Steven Feuerstein, Guy Harrison
.....
Publisher: **O'Reilly**
Pub Date: **March 2006**
Print ISBN-10: **0-596-10089-2**
Print ISBN-13: **978-0-59-610089-6**
Pages: **636**

Table of Contents | Index

The implementation of stored procedures in MySQL 5.0 a huge milestone -- one that is expected to lead to widespread enterprise adoption of the already extremely popular MySQL database. If you are serious about building the web-based database applications of the future, you need to get up to speed quickly on how stored procedures work -- and how to build them the right way. This book, destined to be the bible of stored procedure development, is a resource that no real MySQL programmer can afford to do without.

In the decade since MySQL burst on the scene, it has become the dominant open source database, with capabilities and performance rivalling those of commercial RDBMS offerings like Oracle and SQL Server. Along with Linux and PHP, MySQL is at the heart of millions of applications. And now, with support for stored procedures, functions, and triggers in MySQL 5.0, MySQL offers the programming power needed for true enterprise use.

MySQL's new procedural language has a straightforward syntax, making it easy to write simple programs. But it's not so easy to write secure, easily maintained, high-performance, and bug-free programs. Few in the MySQL world have substantial experience yet with stored procedures, but Guy Harrison and Steven Feuerstein have decades of combined expertise.

In MySQL Stored Procedure Programming, they put that hard-won experience to good use. Packed with code examples and covering everything from language basics to application building to advanced tuning and best practices, this highly readable book is the one-stop guide to MySQL development. It consists of four major sections:

- MySQL stored programming fundamentals -- tutorial, basic statements, SQL in stored programs, and error handling
- Building MySQL stored programs -- transaction handling, built-in functions, stored functions, and triggers
- MySQL stored programs in applications -- using stored programs with PHP, Java, Perl, Python, and .NET (C# and VB.NET)
- Optimizing MySQL stored programs -- security, basic and advanced SQL tuning, optimizing stored program code, and programming best practices

A companion web site contains many thousands of lines of code, that you can put to use immediately.

Guy Harrison is Chief Architect of Database Solutions at Quest Software and a frequent speaker and writer on MySQL topics. Steven Feuerstein is the author of Oracle PL/SQL Programming, the classic reference for Oracle stored programming for more than ten years. Both have decades of experience as database developers, and between them they have authored a dozen books.

MySQL Stored Procedure Programming

by Guy Harrison with Steven Feuerstein

Printed in the United States of America.

Published by O'Reilly Media, Inc., 1005 Gravenstein Highway North, Sebastopol, CA 95472.

O'Reilly books may be purchased for educational, business, or sales promotional use. Online editions are also available for most titles (safari.oreilly.com). For more information, contact our corporate/institutional sales department: (800) 998-9938 or corporate@oreilly.com.

Editor:	Deborah Russell
Production Editor:	Adam Witwer
Production Services:	Argosy Publishing
Cover Designer:	Karen Montgomery
Interior Designer:	David Futato
Illustrators:	Robert Romano, Jessamyn Read, and Lesley Borash

Printing History:	
March 2006:	First Edition.

Nutshell Handbook, the Nutshell Handbook logo, and the O'Reilly logo are registered trademarks of O'Reilly Media, Inc. MySQL Stored Procedure Programming, the image of a middle spotted woodpecker, and related trade dress are trademarks of O'Reilly Media, Inc.

Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and O'Reilly Media, Inc. was aware of a trademark claim, the designations have been printed in caps or initial caps.

While every precaution has been taken in the preparation of this book, the publisher and authors assume no responsibility for errors or omissions, or for damages resulting from the use of the information contained herein.

ISBN: 0-596-10089-2

Advance Praise for MySQL Stored Procedure Programming

"I didn't honestly believe a book could be written on this topic that wouldn't be too dry. But Guy and Steven show the depth of the subject and make the material available to readers. It was a wonderful read."

Brian Aker, Director of Architecture, MySQL AB

"It was a pleasure to work with Guy and the editor at O'Reilly, doing the tech review of many of the chapters for this book. The authors have an excellent grasp of the subject matter. I found the material easy to read, with lots of code examples. MySQL users should find this book an excellent resource."

Arjen Lentz, Community Relations Manager, MySQL AB

"Because MySQL usage is growing so rapidly among modern enterprises, developers and DBAs alike are desperately looking for expert help that shows them how to create high-performance stored procedures and other efficient MySQL code. I doubt that anyone will find better guides than Guy Harrison and Steven Feuerstein when it comes to advice on writing the absolutely best MySQL code."

Robin Schumacher, Director of Product Management, MySQL AB

"This is the first book I've seen that really concentrates on MySQL's stored procedures. I found tips here that I'd never seen before."

Peter Gulutzan, MySQL Software Architect

"MySQL 5.0 opens up a new world to MySQL users, and this book is a great tour guide."

Andy Dustman, Author of MySQL Python API

"Guy and Steven have provided MySQL developers with a gem. They not only cover the nuts and bolts of writing stored procedures in MySQL, but also provide sound advice on designing database applications in the real world. In addition, they write with a sense of humour that makes the book a joy to read."

James Cooper, Technology Consultant, Seattle, WA

Preface

Over the past five years or so, we have seen an explosion in the use of open source software in commercial environments. Linux has almost completely displaced various flavours of Unix as the dominant non-Windows operating system; Apache is by far the most significant web server; Perl and PHP form the foundation for millions of commercial web sites; while JBoss, Hibernate, Spring, and Eclipse are making strong inroads into the Java? and J2EE development and application server markets. Although the world of relational databases continues to be dominated by the commercial players (Oracle, IBM, and Microsoft), the commercial use of open source databases is growing exponentially. MySQL is the dominant open source database management system: it is being used increasingly to build very significant applications based on the LAMP (Linux-Apache-MySQL-PHP/Perl/Python) and LAMJ (Linux-Apache-MySQL-JBoss) open source stacks, and it is, more and more, being deployed wherever a high-performance, reliable, relational database is required.

In the landmark book The Innovators Dilemma,[*] Clayton Christensen provided the first widely accepted model of how open source and other "disruptive" technologies displace more traditional "sustaining" technologies.

[*] The Innovator's Dilemma, Clayton Christensen (New York, 2000), Harper Business Essentials.

When a disruptive technology, Linux for example first appears, its capabilities and performance are typically way below what would be acceptable in the mainstream or high-end market. However, the new technology is highly attractive to those whose requirements or budgets preclude the use of the established commercial alternatives. These very low-end markets are typically associated with low profit margins and low revenues, so the established vendors are more than happy to retreat from these markets and give the disruptive technology this first foothold. As both the sustaining/traditional and disruptive/innovative technologies improve their capabilities, the disruptive technology becomes attractive to a wider segment of the mainstream market, while the established technologies tend to "overshoot" the demands of the average eor even high-end consumer.

For the established vendors, the lower ends of the market are always associated with lower profit margins, and the established vendors make a series of apparently sensible business decisions to successively abandon these markets to the newer disruptive technologies. By the time the disruptive technology is seen as a real threat, the established vendors are unable to compete without cannibalizing the revenues from their established products, and in many cases, they become resigned to losing their market dominance.

Open source in general, and MySQL in particular, shows all the characteristics of the disruptive technology model. Five years ago, the capabilities of MySQL were so far behind the requirements of the majority of business users that the use of MySQL in a business environment was almost unheard of. However, MySQL being free or extremely low cost[*] had a definite appeal for users who were unable to afford a commercial relational database. As with most open source technologies, MySQL has experienced rapid technological development adding transactions, subqueries, and other features normally associated with expensive commercial offerings. By the release of MySQL 4.0, MySQL was being used in a mission-critical manner by an increasing number of high-profile companies, including Yahoo, Google, and Sabre.

[*] MySQL has a dual licensing model that allows for free use in many circumstances but does require a commercial license in some circumstances.

Meanwhile, the commercial database companies have been adding features that, although significant for the very high end of the market, have arguably exceeded the requirements of the majority of database users: they are more concerned with performance, manageability, and stability than with advanced features such as composite object data types, embedded Java Virtual Machines, or complex partitioning and clustering capabilities.

With the 5.0 release, MySQL has arguably crossed one of the last remaining capability thresholds for enterprise credibility. The ability to create stored procedures, functions, triggers, and updateable views removes one of the last remaining objections to using MySQL as a mainstream commercial database. For instance, prior to the introduction of stored procedures, MySQL could not claim Java J2EE certification, because the certification tests include stored procedure routines. While the "commercial" databases still include many features not found in MySQL, these features are often superfluous to the needs of mainstream database applications.

We believe that MySQL will continue to grow in significance as the premier open source RDBMS and that stored programs procedures, functions, and triggers will play a major part in the ongoing MySQL success story.

> First, a note about this book's title and terminology.
>
> The IT industry, the media, and MySQL AB itself generally use the term stored procedures to refer to both stored procedures and stored functions. While this is technically inaccurate (a function is not a procedure), we felt that the title MySQL Stored Procedure Programming would most accurately and succinctly describe the purpose and content of this book. We also felt that the title MySQL Stored Procedure, Function, and Trigger Programming would just be too much of a mouthful!

To avoid any confusion, we use the general term stored program within this book to refer to the set of database routines that includes procedures, functions, and triggers, and to specific types of programs (e.g., stored procedures) when appropriate.

Objectives of This Book

The new capabilities provided by stored procedures, functions, and triggers (we call these, in general, stored programs) require new disciplines for MySQL developers, only some of whom will have prior experience in stored program development using other relational databases. Wise use of stored programs will lead to MySQL applications that are more robust, reliable, and efficient. However, inappropriate use of stored programs, or poorly constructed stored programs, can lead to applications that perform poorly, are hard to maintain, or are unreliable.

Thus, we see the need for a book that will help MySQL practitioners realize the full potential of MySQL stored programs. We hope this book will help you to use stored programs appropriately, and to write stored procedures, functions, and triggers that are reliable, correct, efficient, and easy to maintain.

Best practice stored program development relies on four fundamentals:

Appropriate use

> Used appropriately, stored programs can improve the performance, reliability, and maintainability of your MySQL-based application. However, stored programs are not a universal panacea, and they should be used only where appropriate. In this book, we describe where stored programs can be used to good effect, and we outline some significant patterns (and anti-patterns) involving stored programs.

Reliability

> As with any programming language, the MySQL stored program language allows you to write code that will behave predictably and correctly in all possible circumstances, but the language also allows you to write code subject to catastrophic failure or unpredictable behavior when unanticipated scenarios arise. We outline how to write stored programs that can deal appropriately with error conditions, that fail gracefully and predictably, and that are to the greatest extent possible bug free.

Maintainability

> We have all had that sinking feeling of having to amend some piece of code whether written by a colleague or by ourselvesand finding that the intention, logic, and mechanisms of the code are almost impossible to understand. So-called "spaghetti" code can be written in any language, and MySQL stored programs are no exception. We explain how to construct code that is easily maintained through best practice naming conventions, program structure, commenting, and other mechanisms.

Performance

> Any nontrivial application has to perform to either implicitly or explicitly stated performance requirements. The performance of the database access codeSQL and stored program codeis often the most significant factor in overall application performance. Furthermore, poorly constructed database code often fails to scale predictably or at all when data or transaction volumes increase. In this book, we show you when to use stored programs to improve application performance and how to write stored program code that delivers the highest possible performance. The SQL within a stored program is often the most performance-critical part of the stored program, so we explain in depth how to write high-performance SQL as well.

Structure of This Book

MySQL Stored Procedure Programming is divided into four major sections:

Part I, Stored Programming Fundamentals

This first part of the book introduces the MySQL stored program language and provides a detailed description of the language structure and usage.

- Chapter 1, Introduction to MySQL Stored Programs, asks the fundamental questions: Where did the language come from? What is it good for? What are the main features of the language?
- Chapter 2, MySQL Stored Programming Tutorial, is a tutorial that is designed to get you started with the language as quickly as possible; it shows you how to create basic stored programs of each type and provides interactive examples of major language functions.
- Chapter 3, Language Fundamentals, describes how to work with variables, literals, operators, and expressions.
- Chapter 4, Blocks, Conditional Statements, and Iterative Programming, explains how to implement conditional commands (`IF` and `CASE`) and looping structures.
- Chapter 5, Using SQL in Stored Programming, discusses how SQL can be used within the language.
- Chapter 6, Error Handling, provides the details of how errors can be handled.

Part II, Stored Program Construction

This part of the book describes how you can use the elements described in Part I to build functional and useful stored programs.

- Chapter 7, Creating and Maintaining Stored Programs, outlines the statements available for creating and modifying stored programs and provides some advice on how to manage your stored program source code.
- Chapter 8, Transaction Management, explains the fundamentals of transaction handling in stored programs.
- Chapter 9, MySQL Built-in Functions, details the built-in functions that can be used in stored programs.
- Chapter 10, Stored Functions, describes how you can create and use one particular type of stored program: the stored function.
- Chapter 11, Triggers, describes another special type of stored program the database trigger which is activated in response to DML (Data Manipulation Language) executed on a database table.

Part III, Using MySQL Stored Programs in Applications

Stored programs can be used for a variety of purposes, including the implementation of utility routines for use by MySQL DBAs and developers. However, the most important use of stored programs is within applications, as we describe in this part of the book. Stored programs allow us to move some of our application code into the database server itself; if we do this wisely, we may benefit from an application that will then be more secure, efficient, and maintainable.

- Chapter 12, Using MySQL Stored Programs in Applications, considers the merits of and best practices for using stored programs inside modern typically, web-based applications. The other chapters in this part of the book show you how to use stored procedures and functions from within the development languages most commonly used in conjunction with MySQL.
- Chapter 13, Using MySQL Stored Programs with PHP, describes the use of stored programs from PHP. We primarily discuss the `mysqli` and PDO interfaces recently bundled by MySQL as Connector/PHP and their stored program support.

- Chapter 14, Using MySQL Stored Programs with Java, describes the use of stored programs from Java and includes the use of stored programs using JDBC, Servlets, Enterprise JavaBeans?, Hibernate, and Spring.
- Chapter 15, Using MySQL Stored Programs with Perl, describes the use of stored programs from Perl.
- Chapter 16, Using MySQL Stored Programs with Python, describes the use of stored programs from Python.
- Chapter 17, Using MySQL Stored Programs with .NET, describes the use of stored programs from .NET languages such as C# and VB.NET.

Part IV, Optimizing Stored Programs

This final part of the book hopes to take you from "good" to "great." Getting programs to work correctly is hard enough: any program that works is probably a good program. A great program is one that performs efficiently, is robust and secure, and is easily maintained.

- Chapter 18, Stored Program Security, discusses the unique security concerns and opportunities raised by stored procedures and functions.
- Chapter 19, Tuning Stored Programs and Their SQL. This chapter, along with Chapters 20 through 22, covers the performance optimization of stored programs. This chapter kicks off with a general discussion of performance tuning tools and techniques.
- Chapter 20, Basic SQL Tuning. The performance of your stored programs will be largely dependent on the performance of the SQL inside them, so this chapter provides guidelines for tuning basic SQL.
- Chapter 21, Advanced SQL Tuning. This chapter builds on Chapter 20, describing more advanced tuning approaches.
- Chapter 22, Optimizing Stored Program Code, covers the performance tuning of the stored program code itself.
- Chapter 23, Best Practices in MySQL Stored Program Development, wraps up the book with a look at best practices in stored program development. These guidelines should help you write stored programs that are fast, secure, maintainable, and bug free.

You'll find that a significant proportion of the book includes material that pertains not only to stored program development, but also to development in other languages such as PHP or Java. For instance, we believe that you cannot write a high-performance stored program without tuning the SQL that the program contains; therefore, we have devoted significant coverage to SQL tuning material that would also be of benefit regardless of the language in which the SQL is embedded. Likewise, some of the discussions around transaction design and security could be applicable in other languages.

What This Book Does Not Cover

This book is not intended to be a complete reference to MySQL. It focuses on the stored program language. The following topics are therefore outside the scope of this book and are not covered, except in an occasional and peripheral fashion:

The SQL language

> We assume that you already have a working knowledge of the SQL language, and that you know how to write `SELECT`, `UPDATE`, `INSERT`, and `DELETE` statements.

Administration of MySQL databases

> While DBA’s can use this book to learn how to write the code needed to build and maintain databases, this book does not explore all the nuances of the DDL (Data Definition Language) of MySQL's SQL.

Conventions Used in This Book

The following conventions are used in this book:

Italic

Used for URLs and for emphasis when introducing a new term.

Constant width

Used for MySQL and SQL keywords and for code examples.

`Constant width bold`

In some code examples, highlights the statements being discussed.

`Constant width italic`

In some code examples, indicates an element (e.g., a filename) that you supply.

UPPERCASE

In code examples, generally indicates MySQL keywords.

lowercase

In code examples, generally indicates user-defined items such as variables, parameters, etc.

punctuation

In code examples, enter exactly as shown.

indentation

In code examples, helps to show structure but is not required.

In code examples, begins a single-line comment that extends to the end of a line.

/* and */

In code examples, delimit a multi-line comment that can extend from one line to another.

In code examples and related discussions, qualifies a reference by separating an object name from a component name.

[ ]

In syntax descriptions, enclose optional items.

{ }

In syntax descriptions, enclose a set of items from which you must choose only one.

In syntax descriptions, separates the items enclosed in curly brackets, as in {TRUE | FALSE}.

...

In syntax descriptions, indicates repeating elements. An ellipsis also shows that statements or clauses irrelevant to the discussion were left out.

Which Version?

This book describes the stored program language introduced in MySQL 5.0. At the time the book went to press, MySQL 5.0.18 was the most recently available binary Community edition, although we were working with versions up to 5.1.7 built directly from source code.

Resources Available at the Book's Web Site

We have provided all of the code included in this book on the book's O'Reilly web site. Go to:

http://www.oreilly.com/catalog/mysqlspp

and click on the Examples link to go to the book's web companion.

To find the code for a specific example, look for the file corresponding to the example or figure in which that code appeared. For instance, to obtain the code for Example 3-1, you would access the file *example0301.sql*.

At this web site you will also be able to download a dump file containing the sample database used throughout the book, the source code to some utilities we used during our development of the examples, errata, and addenda to the book's content.

In particular, we will use this web site to keep you posted on the status of any restrictions or problems relating to stored programs in MySQL or other tools. Because the MySQL stored program language is relatively new, MySQL AB will be refining the behaviour and capabilities of the language in each new release of the MySQL server. Also, support for stored programs in other languages (PHP, Perl, Python, Hibernate) was sometimes only partially completed as this book went to press; we'll keep you updated with the status of these languages at the web site.

Using Code Examples

This book is here to help you get your job done. In general, you may use the code in this book in your programs and documentation. You do not need to contact us for permission unless you're reproducing a significant portion of the code. For example, writing a program that uses several chunks of code from this book does not require permission. Selling or distributing a CD-ROM of examples from O'Reilly books does require permission. Answering a question by citing this book and quoting example code does not require permission. Incorporating a significant amount of example code from this book into your product's documentation does require permission.

We appreciate, but do not require, attribution. An attribution usually includes the title, author, publisher, and ISBN. For example: "MySQL Stored Procedure Programming by Guy Harrison with Steven Feuerstein. Copyright 2006 O'Reilly Media, Inc., 0-596-10089-2."

If you feel that your use of code examples falls outside fair use or the permission given here, feel free to contact us at permissions@oreilly.com.

Safari® Enabled

When you see a Safari® Enabled icon on the cover of your favourite technology book, that means the book is available online through the O'Reilly Network Safari Bookshelf.

Safari offers a solution that's better than e-books. It's a virtual library that lets you easily search thousands of top tech books, cut and paste code samples, download chapters, and find quick answers when you need the most accurate, current information. Try it for free at http://safari.oreilly.com.

How to Contact Us

We have tested and verified the information in this book and in the source code to the best of our ability, but given the amount of text and the rapid evolution of technology, you may find that features have changed or that we have made mistakes. If so, please notify us by writing to:

O'Reilly Media, Inc.
1005 Gravenstein Highway North
Sebastopol, CA 95472
800-998-9938 (in the United States or Canada)
707-829-0515 (international or local)
707-829-0104 (fax)

You can also send messages electronically. To be put on the mailing list or request a catalogue, send email to:

info@oreilly.com

To ask technical questions or comment on the book, send email to:

bookquestions@oreilly.com

As mentioned in the earlier section, we have a web site for this book where you can find code, errata (previously reported errors and corrections available for public view), and other book information. You can access this web site at:

http://www.oreilly.com/catalog/mysqlspp

For more information about this book and others, see the O'Reilly web site:

http://www.oreilly.com

Acknowledgments

We'd first like to thank Debby Russell, our editor at O'Reilly Media, for supporting us through this endeavor and for being the organizing force behind the end-to-end project. Many other people at O'Reilly also played a big role in the book's development, including Adam Witwer, the production editor, and Rob Romano, the illustrator; additional production services were provided by Argosy Publishing.

The role of the technical reviewers in the production of this book was absolutely critical. The scope of coverage included not just the MySQL stored program language but also five other development languages and many features of the MySQL 5.0 server itself. Furthermore, the stored program language was evolving as we constructed the book. Without the valuable inputs from our technical reviewers, we would have been unable to achieve any reasonable degree of accuracy and currency across the entire scope. Reviewers included Tim Allwine, Brian Aker, James Cooper, Greg Cottman, Paul DuBois, Andy Dustman, Peter Gulutzan, Mike Hillyer, Arjen Lentz, and Mark Matthews. Thanks guys!

To the open source community in general and to the MySQL development community in particular, we also give thanks. The availability of free (both as in beer and as in speech) software of such quality and innovation is a source of constant amazement and gratification. Many in the MySQL and associated communities contributed to the existence of this in so many ways.

We worked with some of the maintainers of the various open source interfaces to MySQL to ensure that these were able to support some of the new features introduced in MySQL 5.0. Thanks to Wez Furlong, Patrick Galbraith, and Andy Dustman in particular for their help in patching the PHP PDO, Perl DBI, and Python MySQLdb interfaces.

From Guy: On a personal note, I would like to as always thank my wife Jenni and children Christopher, Katherine, Michael, and William for putting up with me during this and other writing projects. Thanks with much love. Also of course thanks to Steven for working with me on this book.

From Steven: I have spent the last 10 years studying, working with, and writing about the Oracle PL/SQL language. That experience has demonstrated very clearly to me the value and importance of stored programs. I was very excited, therefore, when Guy invited me to work with him on a book about MySQL stored programs. I have no doubt that this new functionality will help extend the reach and usefulness of MySQL, and I thank Guy for the opportunity to help MySQL programmers make the most of this key open source relational database.

Part I: Stored Programming Fundamentals

This first part of the book introduces the MySQL stored program language and provides a detailed description of the language structure and usage. Chapter 1 asks the fundamental questions: Where did the language come from? What is it good for? What are the main features of the language? Chapter 2 is a tutorial that is designed to get you started with the language as quickly as possible; it shows you how to create basic stored programs of each type and provides interactive examples of major language functions. Chapters 3 through 6 describe the MySQL stored program language in detail: how to work with variables, how to implement conditional and iterative control structures, how SQL can be used within the language, and how errors can be handled.

Chapter 1, Introduction to MySQL Stored Programs

Chapter 2, MySQL Stored Programming Tutorial

Chapter 3, Language Fundamentals

Chapter 4, Blocks, Conditional Statements, and Iterative Programming

Chapter 5, Using SQL in Stored Programming

Chapter 6, Error Handling

Chapter 1. Introduction to MySQL Stored Programs

When MySQL first emerged into the IT world in the mid-1990s, it had few of the characteristics normally associated with commercial relational databases. Features such as transactional support, subqueries, views, and stored procedures were conspicuously absent. Subsequent releases provided most of the missing features, and now with the introduction of stored procedures, functions, and triggers in MySQL 5 (as well as updateable views and a data dictionary)the feature gap between MySQL and other relational database systems is narrow indeed.

The introduction of stored programs (our generic term for stored procedures, functions, and triggers) has significance beyond simply winning a features war with competitive database systems. Without stored programs, MySQL cannot claim full compliance with a variety of standards, including ANSI/ISO standards that describe how a DBMS should execute stored programs. Furthermore, judicious use of stored programs can lead to greater database security and integrity and can improve overall application performance and maintainability. We outline these advantages in greater detail later in this chapter.

In short, stored programs procedures, functions, and triggers add significantly to the capabilities of MySQL, and a working knowledge of stored programming should be an essential skill for the MySQL professional.

This chapter introduces the MySQL stored program language, its origins, and its capabilities. It also offers a guide to additional resources for MySQL stored program developers and some words of overall development advice.

1.1. What Is a Stored Program?

A database stored program sometimes called a stored module or a stored routine is a computer program (a series of instructions associated with a name) that is stored within, and executes within, the database server. The source code and (sometimes) any compiled version of the stored program are almost always held within the database server's system tables as well. When the program is executed, it is executed within the memory address of a database server process or thread.

There are three major types of MySQL stored programs:

Stored procedures

> Stored procedures are the most common type of stored program. A stored procedure is a generic program unit that is executed on request and that can accept multiple input and output parameters.

Stored functions

> Stored functions are similar to stored procedures , but their execution results in the return of a single value. Most importantly, a stored function can be used within a standard SQL statement, allowing the programmer to effectively extend the capabilities of the SQL language.

Triggers

> Triggers are stored programs that are activated in response to, or are triggered by, an activity within the database. Typically, a trigger will be invoked in response to a DML operation (`INSERT`, `UPDATE`, `DELETE`) against a database table. Triggers can be used for data validation or for the automation of denormalization.

> Other databases offer additional types of stored programs, including packages and classes, both of which allow you to define or collect multiple procedures and functions within a single, named context. MySQL does not currently support such structures in MySQL, each stored program is a standalone entity.

Throughout this book, we are going to use the term stored programs to refer to stored procedures, functions, and triggers , and the term stored program language to refer to the language used to write these programs. Most of the facilities in the stored program language are applicable across procedures, functions, and triggers; however, both functions and triggers have strict limitations on the language features that may be used with them. Thus, we dedicate a chapter to each of these program types in which we explain these limitations.

1.1.1. Why Use Stored Programs?

Developers have a multitude of programming languages from which to choose. Many of these are not database languages, which means that the code written in these languages does not reside in, nor is it managed by, a database server. Stored programs offer some very important advantages over more general-purpose languages, including:

- The use of stored programs can lead to a more secure database.
- Stored programs offer a mechanism to abstract data access routines, which can improve the maintainability of your code as underlying data structures evolve.
- Stored programs can reduce network traffic, because the program can work on the data from within the server, rather than having to transfer the data across the network.
- Stored programs can be used to implement common routines accessible from multiple applications possibly using otherwise incompatible frameworks executed either within or from outside the database server.
- Database-centric logic can be isolated in stored programs and implemented by programmers with more specialized, database experience.
- The use of stored programs can, under some circumstances, improve the portability of your application.

While this is an impressive list of advantages (many of which will be explored in greater detail in this book), we do not recommend that you immediately move all your application logic into stored programs. In today's rich and complex world of software technology, you need to understand the strengths and weaknesses of each possible element in your software configuration, and figure out how to maximize each element. We spend most of Chapter 12 evaluating how and where to apply MySQL stored programs.

The bottom line is that, used correctly, stored programs procedures, functions, and triggers can improve the performance, security, maintainability, and reliability of your applications.

Subsequent chapters will explore how to construct MySQL stored programs and use them to best advantage. Before plunging into the details, however, let's look at how the technology developed and take a quick tour of language capabilities.

1.1.2. A Brief History of MySQL

MySQL has its roots in an in-house (non-SQL) database system called Unireg used by the Swedish company TcX that was first developed in the 1980s and optimized for data warehousing. The author of Unireg, Michael "Monty" Widenius, added a SQL interface to Unireg in 1995, thus creating the first version of MySQL. David Axmark, from Detron HB , approached Monty proposing to release MySQL to the world under a "dual licensing" model that would allow widespread free use, but would still allow for commercial advantage. Together with Allan Larsson, David and Monty became the founders of the MySQL company.

The first widely available version of MySQL was 3.11, which was released in mid-1996. Adoption of MySQL grew rapidly paralleling the adoption of other related open source technologies. By the year 2005, MySQL could lay claim to over 6 million installations of the MySQL database.

Version 3 of MySQL, while suitable for many types of applications (particularly read-intensive web applications), lacked many of the features normally considered mandatory in a relational database. For instance, transactions, views, and subqueries were not initially supported.

However, the MySQL system was designed to support a particularly extensible data access architecture, in which the SQL layer was decoupled from the underlying data and file access layer. This allowed custom "storage engines" to be employed in place of or in combination with the native ISAM (Indexed Sequential Access Method) -based MySQL engine. The Berkeley-DB (BDB) database (from Sleepycat Software) was integrated as an optional storage engine in version 3.23.34 in early 2001. BDB provided MySQL with its initial transaction processing capability. At about the same time, the open source InnoDB storage engine became available and quickly became a natively available option for MySQL users.

The 4.0 release in early 2002 fully incorporated the InnoDB option, making transactions easily available for all MySQL users, and also added improved replication capabilities. The 4.1 release in early 2004 built on the 4.0 release and included among many other improvements support for subqueries and Unicode character sets.

With the 5.0 release of MySQL in late 2005, MySQL took an important step closer to functional parity with commercial RDBMS systems; it introduced stored procedures , functions, and triggers , the addition of a data dictionary (the SQL-standard `INFORMATION_SCHEMA`), and support for updateable views.

The 5.1 release, scheduled for the second half of 2006, will add important facilities such as an internal scheduler, table partitioning, row-based replication, and many other significant enhancements.

1.1.3. MySQL Stored Procedures, Functions, and Triggers

MySQL chose to implement its stored program language within the MySQL server as a subset of the ANSI SQL:2003 SQL/PSM (Persistent Stored Module) specification. What a mouthful! Essentially, MySQL stored programs procedures, functions, and triggers comply with the only available open standard for these types of programs the ANSI standard.

Many MySQL and open source aficionados had been hoping for a stored program language implementation based on an open source language such as PHP or Python. Others anticipated a Java?-based implementation. However, by using the ANSI specification the same specification adopted within IBM's DB2 database, MySQL has taken advantage of years of work done by the ANSI committee, which included representatives from all of the major RDBMS companies.

The MySQL stored program language is a block-structured language (like Pascal) that includes familiar commands for manipulating variables, implementing conditional execution, performing iterative processing, and handling errors. Users of existing stored program languages, such as Oracle's PL/SQL or SQL Server's Transact-SQL, will find features of the language very familiar. Programmers familiar with other languages, such as PHP or Java, might consider the language somewhat simplistic, but they will find that it is easy to learn and that it is well matched to the common requirements of database programming.

1.2. A Quick Tour

Let's look at a few quick examples that demonstrate some key elements of both the structure and the functionality of MySQL's stored program language. For a full tutorial, see Chapter 2.

1.2.1. Integration with SQL

One of the most important aspects of MySQL's stored program language is its tight integration with SQL. You don't need to rely on intermediate software "glue," such as ODBC (Open Data Base Connectivity) or JDBC (Java Data Base Connectivity), to construct and execute SQL statements in your stored program language programs. Instead, you simply write the `UPDATE`, `INSERT`, `DELETE`, and `SELECT` statements directly into your code, as shown in Example 1-1.

Example 1-1. Embedding SQL in a stored program

```
1  CREATE PROCEDURE example1(  )
2  BEGIN
3    DECLARE
 l_book_count INTEGER;
4
5    SELECT COUNT(*)
6      INTO l_book_count
7      FROM books
8     WHERE author LIKE '%HARRISON,GUY%';
9
10   SELECT CONCAT('Guy has written (or co-written) ',
11          l_book_count ,
12          ' books.');
13
14    -- Oh, and I changed my name, so...
15    UPDATE books
16       SET author = REPLACE (author, 'GUY', 'GUILLERMO')
17     WHERE author LIKE '%HARRISON,GUY%';
18
19 END
```

Let's take a more detailed look at this code in the following table:

Line(s)	Explanation
1	This section, the header of the program, defines the name (`example1`) and type (`PROCEDURE`) of our stored program.
2	This `BEGIN` keyword indicates the beginning of the program body, which contains the declarations and executable code that constitutes the procedure. If the program body contains more than one statement (as in this program), the multiple statements are enclosed in a `BEGIN-END` block.
3	Here we declare an integer variable to hold the results of a database query that we will subsequently execute.
5-8	We run a query to determine the total number of books that Guy has authored or co-authored. Pay special attention to line 6: the `INTO` clause that appears within the `SELECT` serves as the "bridge" from the database to the local stored program language variables.
10-12	We use a simple `SELECT` statement (e.g., one without a `FROM` clause) to display the number of books. When we issue a `SELECT` without an `INTO` clause, the results are returned directly to the calling program. This is a non-ANSI extension that allows stored programs to easily return result sets (a

Line(s)	Explanation
	common scenario when working with SQL Server and other RDBMSs).
14	This single-line comment explains the purpose of the `UPDATE`.
15-17	Guy has decided to change the spelling of his first name to "Guillermo" he's probably being stalked by fans of his Oracle bookso we issue an `UPDATE` against the `books` table. We take advantage of the built-in `REPLACE` function to locate all instances of "GUY" and replace them with "GUILLERMO".

1.2.2. Control and Conditional Logic

Of course, real-world applications are full of complex conditions and special cases, so you are unlikely to be able to simply execute a series of SQL statements. The stored program language offers a full range of control and conditional statements so that we can control which lines of our programs actually run under a given set of circumstances. These include:

IF and CASE statements

> Both of these statements implement conditional logic with different structures. They allow you to express logic such as "If the page count of a book is greater than 1000, then . . . ".

A full complement of looping and iterative controls

> These include the simple loop, the `WHILE` loop, and the `REPEAT UNTIL` loop.

Example 1-2, a procedure that pays out the balance of an account to cover outstanding bills, demonstrates some of the control statements of MySQL.

Example 1-2. Stored procedure with control and conditional logic

```
1  CREATE PROCEDURE pay_out_balance
2       (account_id_in INT)
3
4  BEGIN
5
6  DECLARE l_balance_remaining NUMERIC(10,2);
7
8  payout_loop:LOOP
9    SET l_balance_remaining = account_balance(account_id_in);
10
11    IF l_balance_remaining < 1000 THEN
12     LEAVE payout_loop;
13
14    ELSE
15      CALL apply_balance(account_id_in, l_balance_remaining);
16    END IF;
17
18  END LOOP;
19
20 END
```

Let's take a more detailed look at this code in the following table:

Line(s)	Explanation
1-3	This is the header of our procedure; line 2 contains the parameter list of the procedure, which in this case consists of a single incoming value (the identification number of the account).
6	Declare a variable to hold the remaining balance for an account.
8-18	This simple loop (named so because it is started simply with the keyword `LOOP`, as opposed to `WHILE` or `REPEAT`) iterates until the account balance falls below 1000. In MySQL, we can name the loop (line 8, `payout_loop`), which then allows us to use the `LEAVE` statement (see line 12) to terminate that particular loop. After leaving a loop, the MySQL engine will then proceed to the next executable statement following the `END LOOP`; statement (line 18).
9	Call the `account_balance` function (which must have been previously defined) to retrieve the balance for this account. MySQL allows you to call a stored program from within another stored program, thus facilitating reuse of code. Since this program is a function, it returns a value and can therefore be called from within a MySQL `SET` assignment.
11-16	This `IF` statement causes the loop to terminate if the account balance falls below $1,000. Otherwise (the `ELSE` clause), it applies the balance to the next charge. You can construct much more complex Boolean expressions with `ELSEIF` clauses, as well.
15	Call the `apply_balance` procedure. This is an example of code reuse; rather than repeating the logic of `apply_balance` in this procedure, we call a common routine.

1.2.3. Stored Functions

A stored function is a stored program that returns a single value and that can be used whenever a built-in function can be used for example, in a SQL statement. Example 1-3 returns the age of a person in years, when provided with a date of birth.

Example 1-3. A stored function to calculate age from date of birth

```
1 CREATE FUNCTION f_age (in_dob datetime) returns int
2   NO SQL
3 BEGIN
4   DECLARE l_age INT;
5   IF DATE_FORMAT(NOW(  ),'00-%m-%d') >= DATE_FORMAT(in_dob,'00-%m-%d') THEN
6      -- This person has had a birthday this year
7      SET l_age=DATE_FORMAT(NOW(  ),'%Y')-DATE_FORMAT(in_dob,'%Y');
8    ELSE
9      -- Yet to have a birthday this year
10      SET l_age=DATE_FORMAT(NOW(  ),'%Y')-DATE_FORMAT(in_dob,'%Y')-1;
11   END IF;
12   RETURN(l_age);

END;
```

Let's step through this code in the following table:

Lines(s)	Explanation
1	Define the function: its name, input parameters (a single date), and return value (an integer).
2	This function contains no SQL statements. There's some controversy about the use of this clause see

Lines(s)	Explanation
	Chapters 3 and 10 for more discussion.
4	Declare a local variable to hold the results of our age calculation.
5-11	This `IF-ELSE-END IF` block checks to see if the birth date in question has occurred yet this year.
7	If the birth date has, in fact, passed in the current year, we can calculate the age by simply subtracting the year of birth from the current year.
10	Otherwise (i.e., the birth date is yet to occur this year), we need to subtract an additional year from our age calculation.
12	Return the age as calculated to the calling program.

We can use our stored function wherever a built-in function would be permitted within another stored program, in a `SET` statement, or, as shown in Example 1-4, within a SQL statement.

Example 1-4. Using a stored function within a SQL statement (continued)

```
mysql> SELECT firstname,surname, date_of_birth, f_age(date_of_birth) AS age
    ->   FROM employees LIMIT 5;
+-----------+---------+---------------------+------+
| firstname | surname | date_of_birth       | age  |
+-----------+---------+---------------------+------+
| LUCAS     | FERRIS  | 1984-04-17 07:04:27 |   21 |
| STAFFORD  | KIPP    | 1953-04-22 06:04:50 |   52 |
| GUTHREY   | HOLMES  | 1974-09-12 08:09:22 |   31 |
| TALIA     | KNOX    | 1966-08-14 11:08:14 |   39 |
| JOHN      | MORALES | 1956-06-22 07:06:14 |   49 |
+-----------+---------+---------------------+------+
```

1.2.4. When Things Go Wrong

Even if our programs have been thoroughly tested and have no bugs, user input can cause errors to occur in our code. The MySQL stored program language offers a powerful mechanism for handling errors. In Example 1-5, we create a procedure that creates new product codes or if the product code already exists, updates it with a new name. The procedure detects an attempt to insert a duplicate value by using an exception handler. If the attempt to insert fails, the error is trapped and an `UPDATE` is issued in place of the `INSERT`. Without the exception handler, the stored program execution is stopped, and the exception is passed back unhandled to the calling program.

Example 1-5. Error handling in a stored program

```
1  CREATE PROCEDURE sp_product_code
2      (in_product_code VARCHAR(2),
3       in_product_name VARCHAR(30))
4
5  BEGIN
6
7    DECLARE l_dupkey_indicator INT DEFAULT 0;
8    DECLARE duplicate_key CONDITION FOR 1062;
9    DECLARE CONTINUE HANDLER FOR duplicate_key SET l_dupkey_indicator =1;
10
11   INSERT INTO product_codes (product_code, product_name)
12   VALUES (in_product_code, in_product_name);
```

```
13
14   IF l_dupkey_indicator THEN
15     UPDATE product_codes
16        SET product_name=in_product_name
17      WHERE product_code=in_product_code;
18   END IF;
19
20 END
```

Let's take a more detailed look at the error-handling aspects of this code:

Line(s)	Explanation
1-4	This is the header of the stored procedure, accepting two `IN` parameters: product code and product name.
7	Declare a variable that we will use to detect the occurrence of a duplicate key violation. The variable is initialized with a value of 0 (false); subsequent code will ensure that it gets set to a value of 1 (true) only if a duplicate key violation takes place.
8	Define a named condition, `duplicate_key`, that is associated with MySQL error 1062. While this step is not strictly necessary, we recommend that you define such conditions to improve the readability of your code (you can now reference the error by name instead of by number).
9	Define an error handler that will trap the duplicate key error and then set the value of the variable `l_dupkey_indicator` to 1 (true) if a duplicate key violation is encountered anywhere in the subsequent code.
11-12	Insert a new product with the user-provided code and name.
14	Check the value of the `l_dupkey_indicator` variable. If it is still 0, then the `INSERT` was successful and we are done. If the value has been changed to 1 (true), we know that there has been a duplicate key violation. We then run the `UPDATE` statement in lines 15-17 to change the name of the product with the specified code.

Error handling is a critical aspect of writing robust, maintainable MySQL stored programs. Chapter 6 takes you on an extensive tour of the various error-handling mechanisms in MySQL stored programs.

1.2.5. Triggers

A trigger is a stored program that is automatically invoked in response to an event within the database. In the MySQL 5 implementation, triggers are invoked only in response to DML activity on a specific table. The trigger can automatically calculate derived or denormalized values. Example 1-6 shows a trigger that maintains such a derived value; whenever an employee salary is changed, the value of the `contrib_401K` column is automatically set to an appropriate value.

Example 1-6. Trigger to maintain a derived column value

```
1  CREATE TRIGGER employees_trg_bu
2       BEFORE UPDATE ON employees
3       FOR EACH ROW
4    BEGIN
5      IF NEW.salary <50000 THEN
6        SET NEW.contrib_401K=500;
7     ELSE
```

```
8       SET NEW.contrib_401K=500+(NEW.salary-50000)*.01;
9     END IF;
10  END
```

The following table explains this fairly simple and short trigger:

Line(s)	Explanation
1	A trigger has a unique name. Typically, you will want to name the trigger so as to reveal its nature. For example, the "`bu`" in the trigger's name indicates that this is a `BEFORE UPDATE` TRigger.
2	Define the conditions that will cause the trigger to fire. In this case, the trigger code will execute prior to an `UPDATE` statement on the `employees` table.
3	`FOR EACH ROW` indicates that the trigger code will be executed once for each row being affected by the DML statement. This clause is mandatory in the current MySQL 5 trigger implementation.
4-10	This `BEGIN-END` block defines the code that will run when the trigger is fired.
5-9	Automatically populate the `contrib_401K` column in the `employees` table. If the new value for the `salary` column is less than 50000, the `contrib._401K` column will be set to 500. Otherwise, the value will be calculated as shown in line 8.

There is, of course, much more that can be said about the MySQL stored program language which is why you have hundreds more pages of material to study in this book! These initial examples should, however, give you a good feel for the kind of code you will write with the stored program language, some of its most important syntactical elements, and the ease with which you can write and read the stored program language code.

1.3. Resources for Developers Using Stored Programs

The introduction of stored programs in MySQL 5 is a significant milestone in the evolution of the MySQL language. For any new technology to be absorbed and leveraged fully, users of that technology need lots of support and guidance in how best to utilize it. Our objective is to offer in this book complete and comprehensive coverage of the MySQL stored program language.

We are certain, however, that you will need help in other ways, so in the following sections we describe additional resources that either complement this book (by providing information about other MySQL technologies) or provide community-based support or late-breaking news. In these sections we provide quick summaries of many of these resources. By taking full advantage of these resources, many of which are available either free or at a relatively low cost, you will greatly improve the quality of your MySQL development experience and your resulting code.

1.3.1. Books

Over the years, the MySQL series from O'Reilly has grown to include quite a long list of books. Here we list some of the books currently available that we feel could be pertinent to the MySQL stored program developer, as well as relevant books from other publishers. Please check out the MySQL area of the O'Reilly OnLAMP web site (http://www.onlamp.com/onlamp/general/mysql.csp) for more complete information.

MySQL Stored Procedure Programming, by Guy Harrison with Steven Feuerstein

This is the book you are holding now (or maybe even viewing online). This book was designed to be a complete and comprehensive guide to the MySQL stored program language. However, this book does not attempt complete coverage of the MySQL server, the SQL language, or other programming languages that you might use with MySQL. Therefore, you might want to complement this book with one or more other topics from the O'Reilly catalogue or even heaven forbid from another publisher!

MySQL in a Nutshell, by Russell Dyer

This compact quick-reference manual covers the MySQL SQL language, utility programs, and APIs for Perl, PHP, and C. This book is the ideal companion for any MySQL user (O'Reilly).

Web Database Applications with PHP and MySQL, by Hugh Williams and David Lane

This is a comprehensive guide to creating web-based applications using PHP and MySQL. It covers PEAR (PHP Extension and Application Repository) and provides a variety of complete case studies (O'Reilly).

MySQL, by Paul DuBois

This classic reference now in its third edition is a comprehensive reference to MySQL development and administration. The third edition includes pre-release coverage of MySQL 5.0, including some information about stored procedures, functions, and triggers (SAMS).

High Performance MySQL, by Jeremy Zawodny and Derek Balling

This book covers the construction of high-performance MySQL server environments, along with how you can tune applications to take advantage of these environments. The book focuses on optimization, benchmarking, backups, replication, indexing, and load balancing (O'Reilly).

MySQL Cookbook, by Paul DuBois

This cookbook provides quick and easily applied recipes for common MySQL problems ranging from program setup to table manipulation and transaction management to data import/export and web interaction (O'Reilly).

Pro MySQL, by Michael Krukenberg and Jay Pipes

This book covers many advanced MySQL topics, including index structure, internal architecture, replication, clustering, and new features in MySQL 5.0. Some coverage of stored procedures, functions, and triggers is included, although much of the discussion is based on early MySQL 5 beta versions (APress).

MySQL Design and Tuning, by Robert D. Schneider

This is a good source of information on advanced development and administration topics, with a focus on performance (MySQL Press).

SQL in a Nutshell, by Kevin Kline, et al.

> MySQL stored procedures, functions, and triggers rely on the SQL language to interact with database tables. This is a reference to the SQL language as implemented in Oracle, SQL Server, DB2, and MySQL (O'Reilly).

Learning SQL, by Alan Beaulieu

> This book provides an excellent entry point for those unfamiliar with SQL. It covers queries, grouping, sets, filtering, subqueries, joins, indexes, and constraints, along with exercises (O'Reilly).

1.3.2. Internet Resources

There are also some excellent web sites available to MySQL programmers, including some areas devoted to stored programming. You should also make sure to look at the web site for this book (described in the Preface) for updates, errata, and other MySQL information.

MySQL

> MySQL AB offers the most comprehensive collection of white papers, documentation, and forums on MySQL in general and MySQL stored programming in particular. Start at http://www.mysql.com. We outline some specific areas later.

MySQL Developer Zone

> http://dev.mysql.com/ is the main entry point for MySQL programmers. From here you can easily access software downloads, online forums, white papers, documentation, and the bug-tracking system.

MySQL online documentation

> The MySQL reference manual including sections on stored procedures, functions, and triggersis available online at http://dev.mysql.com/doc/. You can also download the manual in various formats from here, or you can order various selections in printed book format at http://dev.mysql.com/books/mysqlpress/index.html.

MySQL forums

> MySQL forums are great places to discuss MySQL features with others in the MySQL community. The MySQL developers are also frequent participants in these forums. The general forum index can be found at http://forums.mysql.com/. The stored procedure forum includes discussions of both procedures and functions, and there is a separate forum for triggers.

MySQL blogs

> There are many people blogging about MySQL nowadays, and MySQL has consolidated many of the most significant feeds on the Planet MySQL web site at http://www.planetmysql.org/.

MySQL stored routines library

> Giuseppe Maxia initiated this routine library, which collects general-purpose MySQL 5 stored procedures and functions. The library is still young, but already there are some extremely useful routines available. For example, you will find routines that emulate arrays, automate repetitive tasks, and perform crosstab manipulations. Check it out at http://savannah.nongnu.org/projects/mysql-sr-lib/.

O'Reilly's OnLAMP MySQL section

> O'Reilly hosts the OnLAMP site, which is dedicated to the LAMP stack (Linux, Apache, MySQL, PHP/Perl/Python) of which MySQL is such an important part. OnLAMP includes numerous MySQL articles, which you can find at http://www.onlamp.com/onlamp/general/mysql.csp.

1.4. Some Words of Advice for Developers

By definition, everyone is new to the world of MySQL stored program development, because stored programs are themselves new to MySQL. However, Guy and Steven have both had plenty of experience in stored program development within other relational databases. Steven, in particular, has been a key figure in the world of Oracle PL/SQL (Oracle's stored program language) development for more than a decade. We hope that you will find it helpful if we share some advice with you on how you can work more effectively with this powerful MySQL programming language.

1.4.1. Don't Be in Such a Hurry!

We are almost always working under tight deadlines, or playing catch-up from one setback or another. We have no time to waste, and lots of code to write. So let's get right to it right?

Wrong. If we dive too quickly into the depths of code construction, slavishly converting requirements to hundreds, thousands, or even tens of thousands of lines of code, we will end up with a total mess that is almost impossible to debug and maintain. Don't respond to looming deadlines with panic; you are more likely to meet those deadlines if you do some careful planning.

We strongly encourage you to resist these time pressures and make sure to do the following before you start a new application, or even a specific program in an application:

Construct test cases and test scripts before you write your code

> You should determine how you want to verify a successful implementation before you write a single line of a program. By doing this, you are more likely to get the interface of your program correct and be able to thoroughly identify what it is your program needs to do.

Establish clear rules for how developers will write the SQL statements in the application

> In general, we recommend that individual developers not write a whole lot of SQL. Instead, those single-row queries and inserts and updates should be "hidden" behind pre-built and thoroughly tested procedures and functions (this is called data encapsulation). These programs can be optimized, tested, and maintained much more effectively than SQL statements (many of them quite similar) scattered throughout your code.

Establish clear rules for how developers will handle exceptions in the application

> If you don't set standards, then everyone will handle errors their own way or not at all, creating software chaos. The best approach to take is to centralize your error-handling logic in a small set of procedures, which hide all the details of how an error log is kept, determine how exceptions are raised and propagated up through nested blocks, and more. Make sure that all developers use these programs and do not write their own complicated, time-consuming, and error-prone error-handling code.

Use "stepwise refinement" (a.k.a. top-down design) to limit the complexity of the requirements you must deal with at any given time

> We are usually tasked with implementing very complex requirements. If you try to "do it all" in one big stored program, it will rapidly devolve into spaghetti code that even you will not be able to understand later. Break your big challenges into a sequence of smaller challenges, and then tackle those more manageable problems with reasonably sized programs. If you use this approach, you will find that the executable sections of your modules are shorter and easier to understand, which makes your code easier to maintain and enhance over time.

These are just a few of the important things to keep in mind before you start writing all that code. Just remember: in the world of software development, haste not only makes waste, it virtually guarantees a generous offering of bugs and lost weekends.

1.4.2. Don't Be Afraid to Ask for Help

Chances are, if you are a software professional, you are a smart and well-educated individual. You studied hard, you honed your skills, and now you make a darn good living writing code. You can solve almost any problem you are handed, and that makes you proud.

Unfortunately, your success can also make you egotistical, arrogant, and reluctant to seek out help when you are stumped (we think we are supposed to know all the answers). This dynamic is one of the most dangerous and destructive aspects of software development.

Software is written by human beings; it is important, therefore, to recognize that human psychology plays a key role in software development. The following is an example.

Joe, the senior developer in a team of six, has a problem with his program. He studies it for hours, with increasing frustration, but cannot figure out the source of the bug. He wouldn't think of asking any of his peers to help because they all have less experience than he does. Finally, though, he is at wits' end and gives up. Sighing, he picks up his phone and touches an extension: "Sandra, could you come over here and take a look at my program? I've got a problem I can't figure out." Sandra stops by and, with the quickest glance at Joe's program, points out what should have been obvious to him long ago. Hurray! The program is fixed, and Joe expresses gratitude, but in fact he is secretly embarrassed.

Thoughts like "Why didn't I see that?" and "If I'd only spent another five minutes doing my own debugging I would have found it" run though Joe's mind. This is understandable but misguided. The bottom line is that we are often unable to identify our own problems because we are too close to our own code. Sometimes, all we need is a fresh perspective, the relatively objective view of someone with nothing at stake. It has nothing to do with seniority, expertise, or competence.

Besides, Sandra isn't going to think poorly of Joe. Instead, by asking her for help, Joe has made her feel better about herself, and so both members of the development team benefit.

We strongly suggest that you establish the following guidelines in your organization:

Reward admissions of ignorance

> Hiding what you don't know about an application or its code is very dangerous. Develop a culture in which it is OK to say "I don't know" and encourages the asking of lots of questions.

Ask for help

> If you cannot figure out the source of a bug in 30 minutes, immediately ask for help. You might even set up a "buddy system," so that everyone is assigned a person who is expected to be asked for assistance. Don't let yourself (or others in your group) go for hours banging your head against the wall in a fruitless search for answers.

Set up a formal peer code review process

> Don't let any code go to QA (Quality Assurance) or production without being read and critiqued (in a positive, constructive manner) by other developers in your group.

1.4.3. Take a Creative, Even Radical Approach

We all tend to fall into ruts, in almost every aspect of our lives. Humans are creatures of habit: you learn to write code in one way; you assume certain limitations about a product; you turn aside possible solutions without serious examination because you just know it can't be done. Developers become downright prejudiced about their own programs, and often not in positive ways. They are often overheard saying things like:

- "It can't run any faster than that; it's a pig."
- "I can't make it work the way the user wants; that'll have to wait for the next version."
- "If I were using X or Y or Z product, it would be a breeze. But with this stuff, everything is a struggle."

But the reality is that your program can almost always run a little faster. And the screen can, in fact, function just the way the user wants it to. And although each product has its limitations, strengths, and weaknesses, you should never have to wait for the next version. Isn't it so much more satisfying to be able to tell your therapist that you tackled the problem head-on, accepted no excuses, and crafted a solution?

How do you do this? Break out of the confines of your hardened views and take a fresh look at the world (or maybe just the walls of your cubicle). Reassess the programming habits you've developed. Be creative step away from the traditional methods, from the often limited and mechanical approaches constantly reinforced in our places of business.

Try something new: experiment with what may seem to be a radical departure from the norm. You will be surprised at how much you will learn and grow as a programmer and problem solver. Over the years, we have surprised ourselves over and over with what is really achievable when we stopped saying "You can't do that!" and instead simply nodded quietly and wondered to ourselves: "Now, if we do it this way, what will happen ...?"

1.5. Conclusion

In this chapter, we took you on a whirlwind tour of the MySQL relational database and the new MySQL stored program language. We also provided you with some useful resources and added some general words of advice that we hope you find useful.

In the next chapter, we'll provide a more comprehensive tutorial that will really get you started with MySQL stored procedures, functions, and triggers.

Chapter 2. MySQL Stored Programming Tutorial

MySQL stored programming is a complex topic. We offer this chapter to introduce you to the main and common tasks you will need to perform, including:

- How to create a stored program
- How to pass information in and out of the stored program
- How to interact with the database
- How to create procedures, functions, and triggers in the MySQL stored program language

We don't go into detail in this chapter. Our purpose is to get you started and to give you some appreciation of how stored programs work. Later chapters will explore in detail all of the topics touched on in this chapter.

2.1. What You Will Need

To follow along with the examples in this tutorial , you will need:

- A MySQL 5 server
- A text editor such as vi, emacs, or Notepad
- The MySQL Query Browser

You can get the MySQL server and MySQL Query Browser from http://dev.mysql.com.

2.2. Our First Stored Procedure

We'll start by creating a very simple stored procedure. To do this, you need an editing environment in which to write the stored procedure and a tool that can submit the stored procedure code to the MySQL server.

You can use just about any editor to write your code. Options for compiling that code into MySQL include:

- The MySQL command-line client
- The MySQL Query Browser
- A third-party tool such as Toad for MySQL

In this chapter, we won't make any assumptions about what tools you have installed, so we'll start with the good old MySQL command-line client.

Let's connect to the MySQL server on the local host at port 3306 using the root account. We'll use the preinstalled "test" database in Example 2-1.

Example 2-1. Connecting to the MySQL command-line client

```
[gharriso@guyh-rh4-vm2 ~]$mysql -uroot -psecret -hlocalhost
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 1 to server version: 5.0.16-nightly-20051017-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql>
```

2.2.1. Creating the Procedure

You can create a stored program with the `CREATE PROCEDURE`, `CREATE FUNCTION`, or `CREATE TRIGGER` statement. It is possible to enter these statements directly at the MySQL command line, but this is not practical for stored programs of more than trivial length, so the best thing for us to do is to create a text file containing our stored program text. Then we can submit this file to the database using the command-line client or another tool.

We will use the MySQL Query Browser as a text editor in this example. If you don't have this tool, you can download it from http://dev.mysql.com/downloads/. Alternately, you could use an OS text editor such as vi, emacs, or Notepad. We like the MySQL Query Browser because of its built-in help system, syntax highlighting, ability to run SQL statements, and lots of other features.

Follow these steps:

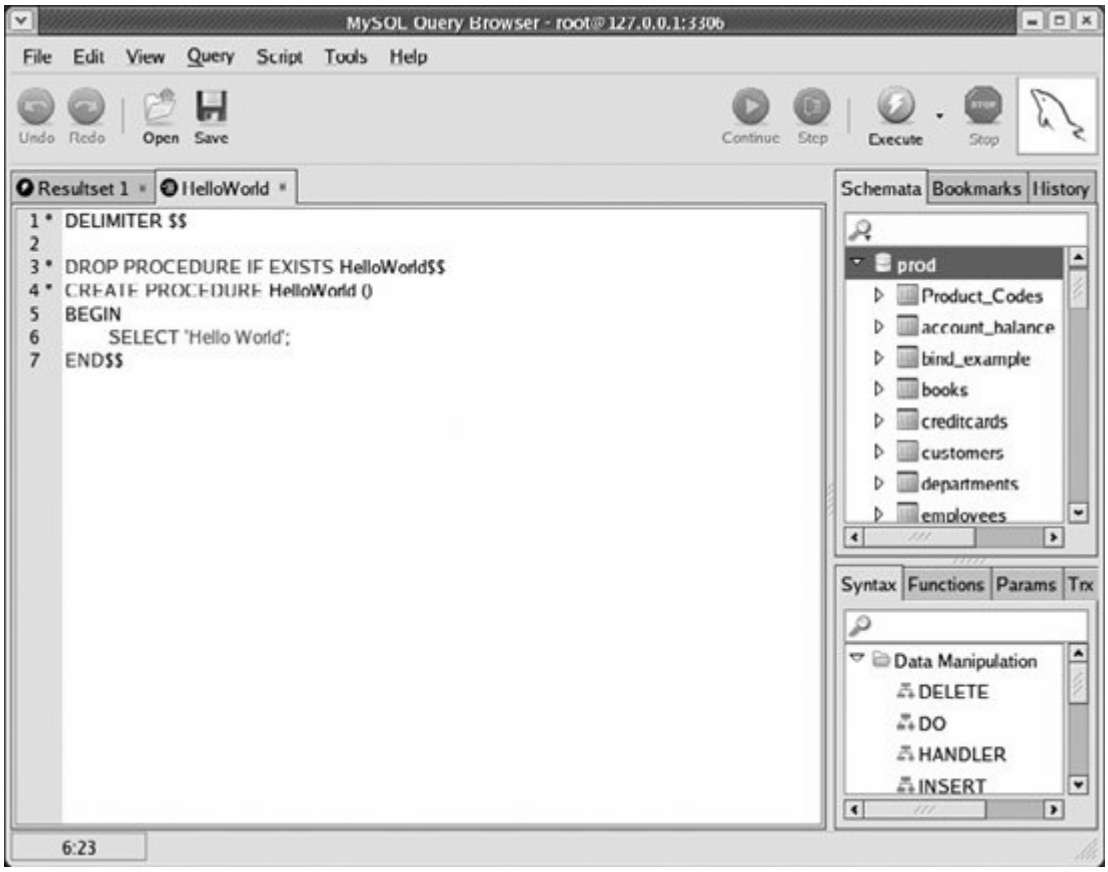
1. Run the MySQL Query browser. On Windows, from the Start menu select Programs MySQL MySQL Query Browser. On Linux, type `mysql-query-browser` from the command line.

2. Select File New Script tab from the menu to create a blank script window.
3. Enter your stored program command text.

Figure 2-1 shows our first stored procedure.

We then use the File Save As menu option to save our file so that we can execute it from the `mysql` client.

Figure 2-1. A first stored procedure



This first stored procedure is very simple, but let's examine it line by line to make sure you understand it completely:

Line	Explanation
1	Issue the `DELIMITER` command to set `'$$'` as the end of a statement. Normally, MySQL regards `";"` as the end of a statement, but since stored procedures contain semicolons in the procedure body, we need to use a different delimiter.
3	Issue a `DROP PROCEDURE IF EXISTS` statement to remove the stored procedure if it already exists. If we don't do this, we will get an error if we then try to re-execute this file with modifications and the stored procedure exists.
4	The `CREATE PROCEDURE` statement indicates the start of a stored procedure definition. Note that the stored procedure name `"HelloWorld"` is followed by an empty set of parentheses `"( )"`. If our stored procedure had any parameters, they would be defined within these parentheses. This stored procedure has no parameters, but we need to include the parentheses anyway, or we will get a syntax error.
5	The `BEGIN` statement indicates the start of the stored procedure program. All stored programs with more than a single statement must have at least one `BEGIN` and `END` block that defines the start and end of the stored program.
6	This is the single executable statement in the procedure: a `SELECT` statement that returns `"Hello World"` to the calling program. As you will see later, `SELECT` statements in stored programs can return data to the console or calling program just like `SELECT` statements entered at the MySQL command line.
7	The `END` statement terminates the stored procedure definition. Note that we ended the stored procedure definition with $$ so that MySQL knows that we have completed the `CREATE PROCEDURE` statement.

With our definition stored in a file, we can now use the mysql client to create and then execute the HelloWorld stored procedure, as shown in Example 2-2.

Example 2-2. Creating our first stored procedure

```
$ mysql -uroot -psecret -Dprod
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 16 to server version: 5.0.18-nightly-20051208-log

Type 'help;' or '\h' for help. Type '\c' to clear the buffer.

mysql> SOURCEHelloWorld.sql
Query OK, 0 rows affected, 1 warning (0.01 sec)

Query OK, 0 rows affected (0.00 sec)

mysql> CALLHelloWorld(  ) $$
+-------------+
| Hello World |
+-------------+
| Hello World |
+-------------+
1 row in set (0.01 sec)

Query OK, 0 rows affected (0.01 sec)

mysql>
```

Here is an explanation of the MySQL commands used to get all this to work:

Command	Explanation
`SOURCE HelloWorld.sql`	Reads commands from the nominated file. In this case, we specify the file we just saved from the MySQL Query Browser. No errors are returned, so the stored procedure appears to have been created successfully.
`CALL HelloWorld( ) $$`	Executes the stored procedure. Calling our stored procedure successfully results in `"Hello World"` being output as a result set. Note that we terminated the `CALL` command with `'$$'`, since that is still what the `DELIMITER` is set to.

2.2.2. Creating the Procedure Using the MySQL Query Browser

In this tutorial and indeed throughout this book we will mostly create and demonstrate stored programs the old-fashioned way: using the MySQL command-line client to create the stored program. By doing this, you'll always be able to duplicate the examples. However, you do have the option of using a GUI tool to create stored programs: there are a number of good third-party GUI tools for MySQL available, and you always have the option of installing and using the MySQL Query Browser, available from http://dev.mysql.com/downloads/.

In this section we offer a brief overview of creating a stored procedure using the MySQL Query Browser. Using the Query Browser is certainly a more user-friendly way of creating stored programs, although it might not be available on all platforms, and you may prefer to use the MySQL command line or the various third-party alternatives.

On Windows, you launch the Query Browser (if installed) from the Start menu option Programs MySQL

MySQL Query Browser. On Linux, you type `mysql-query-browser`.

When the Query Browser launches, it prompts you for connection details for your MySQL server. Once you have provided these, a blank GUI window appears. From this window, select Script and then Create Stored Procedure. You will be prompted for the name of the stored program to create, after which an empty template for the stored program will be displayed. An example of such a template is shown in Figure 2-2.

Figure 2-2. Creating a stored procedure in the MySQL Query Browser

You can then enter the text of the stored procedure at the appropriate point (between the `BEGIN` and `END` statements the cursor is handily positioned there automatically). Once you have finished entering our text, simply click the Execute button to create the stored procedure. If an error occurs, the Query Browser highlights the line and displays the error in the lower half of the Query Browser window. Otherwise, you'll see the name of the new stored procedure appear in the Schemata tab to the left of the stored procedure, as shown in Figure 2-3.

To execute the stored procedure, double-click on the name of the procedure within the Schemata tab. An appropriate `CALL` statement will be pasted into the execution window above the stored procedure. Clicking on the Execute button to the right of the `CALL` statement executes the stored procedure and displays a results window, as shown in Figure 2-4.

Figure 2-3. Stored procedure is created by clicking the Execute button

We hope this brief example gives you a feel for the general process of creating and executing a stored procedure in the MySQL Query Browser. The Query Browser offers a convenient environment for the development of stored programs, but it is really up to you whether to use the Query Browser, a third-party tool, or simply your favorite editor and the MySQL command-line client.