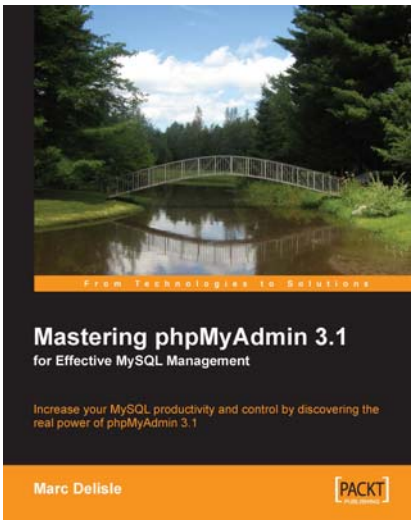
Mastering phpMyAdmin 3.1 for Effective MySQL Management

Marc Delisle



Chapter No. 8
"Importing Structure and Data"

In this package, you will find:

A Biography of the author of the book

A preview chapter from the book, Chapter NO.8 "Importing Structure and Data"

A synopsis of the book's content

Information on where to buy this book

About the Author

Marc Delisle, owing to his involvement with phpMyAdmin, is a member of the MySQL Developers Guild that regroups community developers. He began contributing to this popular MySQL web interface in December 1998, when he made its first multi-language version. As a developer and project administrator, he has been actively involved with this software project since 2001.

Since 1980 Marc has worked at Cegep de Sherbrooke, Québec, Canada, as an application programmer and network manager. He has also been teaching networking, security, and PHP/MySQL application development. In one of his classes, he was pleased to meet a phpMyAdmin user from Argentina. Marc lives in Sherbrooke with his wife, and they enjoy spending time with their four children.

This book is an update to Marc's first book on phpMyAdmin, published by Packt Publishing in 2004, followed by "Creating your MySQL Database: Practical Design Tips and Techniques", also by Packt Publishing.

I am truly grateful to Louay Fatoohi, who approached me for this book project, and to the Packt team whose sound comments were greatly appreciated during production. My thanks also go to Garvin Hicking, Alexander Marcus Turek, and Kai 'Oswald' Seidler—the reviewers for the successive editions of this book. Their sharp eyes helped in making this book clearer and more complete.

Finally, I wish to thank all contributors to phpMyAdmin's source code, translations, and documentation. The time they gave to this project still inspires me and continues to push me forward.

Mastering phpMyAdmin 3.1 for Effective MySQL Management

Providing a powerful graphical interface for managing MySQL, phpMyAdmin is one of the most popular open source applications. While most developers use routine features of phpMyAdmin every day, few are aware of the power and potential its advanced features have. This book builds a solid understanding of the core capabilities of phpMyAdmin, and then walks you through every facet of this legendary tool.

Used by millions of developers, MySQL is the most popular open source database. It supports numerous large, dynamic websites and applications. MySQL has acquired this wide popularity by virtue of its open source nature, performance, reliability, robustness, and support for multiple platforms. However, this popularity has also been helped by the existence of phpMyAdmin—the industry standard administration tool that makes database management easy for both, the experienced developer and the novice one.

Bringing a web interface to MySQL has made phpMyAdmin an indispensable tool for MySQL and web developers, be they professionals or in the classroom. Every user can benefit from phpMyAdmin, by unlocking the full potential of this powerful application.

What This Book Covers

Chapter 1 gives us the reasons why we should use phpMyAdmin as a means of developing web applications. It then covers the downloading and installation procedures for phpMyAdmin.

Chapter 2 provides an overview of various authentication types used in MySQL. It then covers the security issues related to phpMyAdmin installation.

Chapter 3 gives us an overview of the phpMyAdmin interface. It includes the login panel, the left and the right panel including the Light and the Full mode, and the Query window.

Chapter 4 is all about database creation. It teaches us how to create a table, how to insert data manually, and how to sort the data.

Chapter 5 covers the aspects of data editing in phpMyAdmin. It teaches us handling NULL values, multi-row editing. Finally, it covers an important aspect of any database tool—data deletion.

Chapter 6 explores the subject of changing the structure of tables. Its primary focus is on editing field attributes and index management.

Chapter 7 deals with backups and exports. It lists various ways to trigger an export, available export formats, the options associated with export formats, and the various places where the export files may be sent.

Chapter 8 tells us how to bring back exported data created for backup and transfer purposes. It provides an overview of the various options available in phpMyAdmin to import data, and different mechanisms involved in importing SQL and CSV files. Finally, it covers the limitations that may be faced while importing files, and the ways to overcome them.

Chapter 9 presents the mechanisms that are useful for searching data effectively.

Chapter 10 covers ways to perform some operations that influence and can be performed on entire tables or databases as a whole. Finally, it deals with table maintenance operations for table repair and optimization.

Chapter 11 is where we start covering advanced features of phpMyAdmin. The chapter explains how to define inter-table relations. It also explains how to install the linked-tables infrastructure—a prerequisite for the advanced features.

Chapter 12 helps us enter our own SQL commands. The chapter also gives us an overview of Query window—the window used to edit an SQL query. Finally, it also helps us to obtain the history of typed commands.

Chapter 13 covers the multi-table query generator, which allows us to produce multi-table queries without actually typing them.

Chapter 14 covers Bookmarks—one of the features of the linked-tables infrastructure. It covers how to record bookmarks and how to manipulate them. Finally, it covers passing parameters to bookmarks, and executing bookmarks directly from the `pma_bookmark` table.

Chapter 15 gives an overview of how to produce documentation, which explains the structure of the databases, using the tools offered by phpMyAdmin.

Chapter 16 explains how to apply transformations to the data in order to customize its format at view time.

Chapter 17 covers phpMyAdmin's support for the MySQL features that are new in versions 5.0 and 5.1.

Chapter 18 is about the administration of a MySQL server, focusing on the management of user accounts and privileges. The chapter discusses how a system administrator can use phpMyAdmin's server management features for day-to-day user account maintenance, server verification, and server protection.

Appendix A provides a history of the phpMyAdmin project, from its roots back in 1998 through the project re-launch in 2001, and its subsequent evolution.

Appendix B explains how to troubleshoot phpMyAdmin by examining some of its error messages, and proposing appropriate solutions. It also explains how to interact with the development team for support, bug reports, and contributions.

8
Importing Structure and Data

In this chapter, we will learn how to bring back exported data that we create for backup or transfer purposes. Exported data may also come from authors of other applications, and could contain the whole foundation structure of these applications and some sample data.

The current phpMyAdmin version (3.1) can import files containing MySQL statements (usually having a `.sql` suffix, but not necessarily so) and CSV files (comma-separated values, although the separator is not necessarily a comma) directly. Future versions might be able to import files in more formats. There is also an interface to the MySQL `LOAD DATA INFILE` statement, enabling us to load text files containing data, also called CSV. The binary field upload covered in Chapter 6 can be said to belong to the import family.

Importing and uploading are synonyms in this context.

The import feature can be accessed from several panels:

- The **Import** menu available from the homepage, the Database view, or the Table view
- The **Import files** menu offered inside the **Query window** (as explained in Chapter 12)

A feature was added in version 2.11.0: an import file may contain the `DELIMITER` keyword. This enables phpMyAdmin to mimic the mysql command-line interpreter. The `DELIMITER` separator is used to delineate the part of the file containing a stored procedure, as these procedures can themselves contain semicolons.

The default values for the `Import` interface are defined in `$cfg['Import']`.

Before examining the actual import dialog, let's discuss some limits issues.

Limits for the transfer

When we import, the source file is usually on our client machine; so, it must travel to the server via HTTP. This transfer takes time and uses resources that may be limited in the web server's PHP configuration.

Instead of using HTTP, we can upload our file to the server using a protocol such as FTP, as described in the *Web Server Upload Directories* section. This method circumvents the web server's PHP upload limits.

Time limits

First, let's consider the time limit. In `config.inc.php`, the `$cfg['ExecTimeLimit']` configuration directive assigns, by default, a maximum execution time of 300 seconds (five minutes) for any phpMyAdmin script, including the scripts that process data after the file has been uploaded. A value of `0` removes the limit, and in theory, gives us infinite time to complete the import operation. If the PHP server is running in safe mode, modifying `$cfg['ExecTimeLimit']` will have no effect. This is because the limits set in `php.ini` or in user-related web server configuration file (such as `.htaccess` or virtual host configuration files) take precedence over this parameter.

Of course, the time it effectively takes depends on two key factors:

- Web server load
- MySQL server load

The time taken by the file, as it travels between the client and the server, does not count as execution time because the PHP script only starts to execute after the file has been received on the server. Therefore, the `$cfg['ExecTimeLimit']` parameter has an impact only on the time used to process data (like decompression or sending it to the MySQL server).

Other limits

The system administrator can use the `php.ini` file or the web server's virtual host configuration file to control uploads on the server.

The `upload_max_filesize` parameter specifies the upper limit or the maximum file size that can be uploaded via HTTP. This one is obvious, but another less obvious parameter is `post_max_size`. As HTTP uploading is done via the POST method, this parameter may limit our transfers. For more details about the POST method, please refer to `http://en.wikipedia.org/wiki/Http#Request_methods`.

The `memory_limit` parameter is provided to avoid web server child processes from grabbing too much of the server memory—phpMyAdmin also runs as a child process. Thus, the handling of normal file uploads, especially compressed dumps, can be compromised by giving this parameter a small value. Here, no preferred value can be recommended; the value depends on the size of uploaded data. The memory limit can also be tuned via the `$cfg['MemoryLimit']` parameter in `config.inc.php`, as seen in Chapter 7.

Finally, file uploads must be allowed by setting `file_uploads` to `On`. Otherwise, phpMyAdmin won't even show the **Location of the textfile** dialog. It would be useless to display this dialog, as the connection would be refused later by the PHP component of the web server.

Partial imports

If the file is too big, there are ways in which we can resolve the situation. If we still have access to the original data, we could use phpMyAdmin to generate smaller CSV export files, choosing the **Dump n rows starting at record # n** dialog. If this were not possible, we will have to use a text editor to split the file into smaller sections. Another possibility is to use the upload directory mechanism, which accesses the directory defined in `$cfg['UploadDir']`. This feature is explained later in this chapter

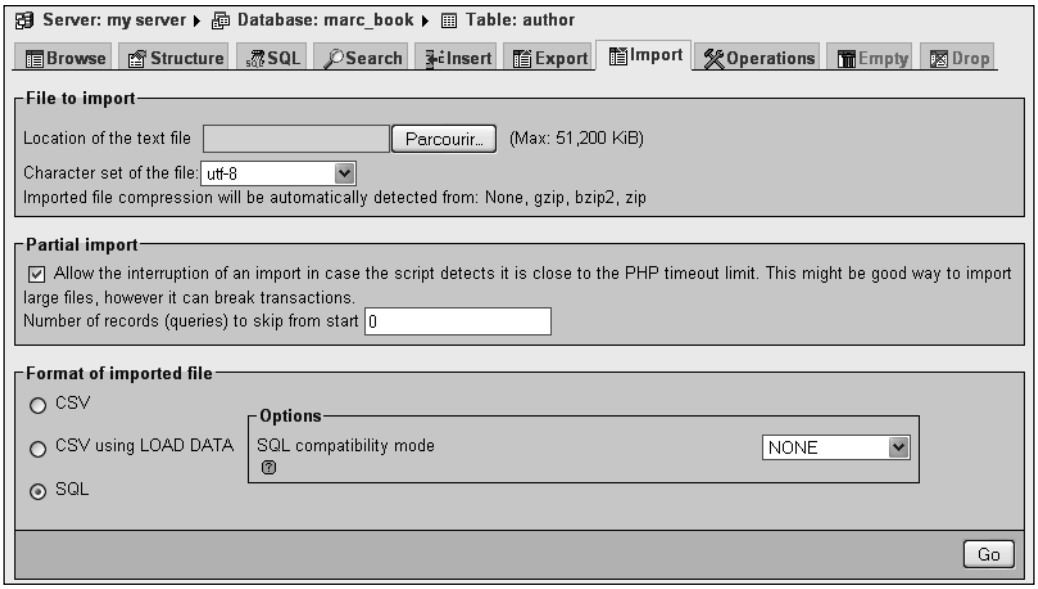
In recent phpMyAdmin versions, the **Partial import** feature can also solve this file size problem. By selecting the **Allow interrupt…** checkbox, the import process will interrupt itself if it detects that it is close to the time limit. We can also specify a number of queries to skip from the start, in case we successfully import a number of rows and wish to continue from that point.

Temporary directory

On some servers, a security feature called `open_basedir` can be set up in a way that impedes the upload mechanism. In this case, or for any other reason, when uploads are problematic, the `$cfg['TempDir']` parameter can be set with the value of a temporary directory. This is probably a subdirectory of phpMyAdmin's main directory, into which the web server is allowed to put the uploaded file.

Importing SQL files

Any file containing MySQL statements can be imported via this mechanism. The dialog is available in the Database view or the Table view, via the **Import** subpage, or in the **Query** window.



There is no relation between the currently selected table (here **author**) and the actual contents of the SQL file that will be imported. All the contents of the SQL file will be imported, and it is those contents that determine which tables or databases are affected. However, if the imported file does not contain any SQL statements to select a database, all statements in the imported file will be executed on the currently selected database.

Let's try an import exercise. First, we make sure that we have a current SQL export of the **book** table (as explained in Chapter 7). This export file must contain the structure and the data. Then we drop the **book** table—yes, really! We could also simply rename it. (See Chapter 10 for the procedure.)

Now it is time to import the file back. We should be on the **Import** subpage, where we can see the **Location of the text file** dialog. We just have to hit the **Browse** button and choose our file.

phpMyAdmin is able to detect which compression method (if any) has been applied to the file. Depending on the phpMyAdmin version, and the extensions that are available in the PHP component of the web server, there is variation in the formats that the program can decompress.

However, to import successfully, phpMyAdmin must be informed of the character set of the file to be imported. The default value is **utf8**. However, if we know that the import file was created with another character set, we should specify it here.

An **SQL compatibility mode** selector is available at import time. This mode should be adjusted to match the actual data that we are about to import, according to the type of the server where the data was previously exported.

To start the import, we click **Go**. The import procedure continues and we receive a message: **Import has been successfully finished, 2 queries executed**. We can browse our newly-created tables to confirm the success of the import operation.

The file could be imported for testing in a different database or even in a MySQL server.

Importing CSV files

In this section, we will examine how to import CSV files. There are two possible methods—**CSV** and **CSV using LOAD DATA**. The first method is implemented internally by phpMyAdmin and is the recommended one for its simplicity. With the second method, phpMyAdmin receives the file to be loaded, and passes it to MySQL. In theory, this method should be faster. However, it has more requirements due to MySQL itself (see the *Requirements* sub-section of the *CSV using LOAD DATA* section).

Differences between SQL and CSV formats

There are some differences between these two formats. The CSV file format contains data only, so we must already have an existing table in place. This table does not need to have the same structure as the original table (from which the data comes); the **Column names** dialog enables us to choose which columns are affected in the target table.

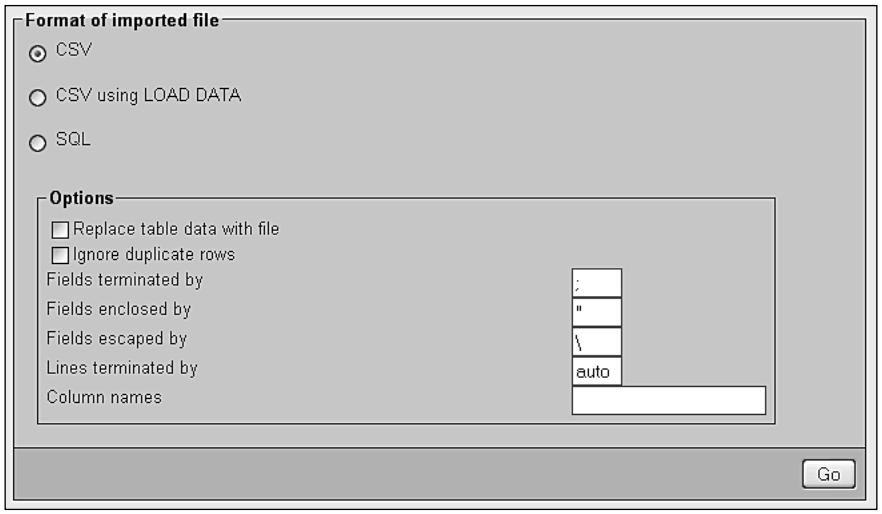
Because the table must exist prior to the import, the CSV import dialog is available only from the **Import** subpage in the Table view, and not in the Database view.

Exporting a test file

Before trying an import, let's generate an `author.csv` export file from the `author` table. We use the default values in the **CSV export** options. We can then **Empty** the `author` table—we should avoid dropping this table because we still need the table structure.

CSV

From the **author** table menu, we select **Import** and then **CSV**.



We can influence the behavior of the import in a number of ways. By default, importing does not modify existing data (based on primary or unique keys). However, the **Replace table data with file** option instructs phpMyAdmin to use `REPLACE` statement instead of `INSERT` statement, so that existing rows are replaced with the imported data.

Using **Ignore duplicate rows**, `INSERT IGNORE` statements are generated. These cause MySQL to ignore any duplicate key problems during insertion. A duplicate key from the import file does not replace existing data, and the procedure continues for the next line of CSV data.

We can then specify the character that terminates each field, the character that encloses data, and the character that escapes the enclosing character. Usually this is \. For example, for a double quote enclosing character, if the data field contains a double quote, it must be expressed as **"some data \" some other data"**.

For **Lines terminated by**, recent versions of phpMyAdmin offer the **auto** choice, which should be tried first as it detects the end-of-line character automatically. We can also specify manually which characters terminate the lines. The usual choice is **\n** for UNIX-based systems, **\r\n** for DOS or Windows systems, and **\r** for Mac-based system (up to Mac OS 9). If in doubt, we can use a hexadecimal file editor on our client computer (not part of phpMyAdmin) to examine the exact codes.

By default, phpMyAdmin expects a CSV file with the same number of fields and the same field order as the target table. But this can be changed by entering a comma-separated list of column names in **Column names**, respecting the source file format. For example, let's say our source file contains only the author ID and the author name information:

```
"1","John Smith"
"2","Maria Sunshine"
```

We'd have to put **id**, **name** in **Column names** to match the source file.

When we click **Go**, the import is executed and we get a confirmation. We might also see the actual `INSERT` queries generated if the total size of the file is not too big.

```
Import has been successfully finished, 2 queries executed.
INSERT INTO `author` VALUES ('1', 'John Smith', '+01 445 789-1234'
)# 1 row(s) affected.

INSERT INTO `author` VALUES ('2', 'Maria Sunshine', '333-3333'
)# 1 row(s) affected.
```

CSV using LOAD DATA

With this method, phpMyAdmin relies on the server's `LOAD DATA INFILE` or `LOAD DATA LOCAL INFILE` mechanisms to do the actual import, instead of processing the data internally. These statements are the fastest way for importing text in MySQL. They cause MySQL to start a read operation either from a file located on the MySQL server (`LOAD DATA INFILE`) or from another place (`LOAD DATA LOCAL INFILE`), which in this context, is always the web server's file system. If the MySQL server is located on a computer other than the web server, we won't be able to use the `LOAD DATA INFILE` mechanism.

Requirements

Relying on the MySQL server has some consequences. Using `LOAD DATA INFILE` requires that the logged-in user possess a global `FILE` privilege. Also, the file itself must be readable by the MySQL server's process.

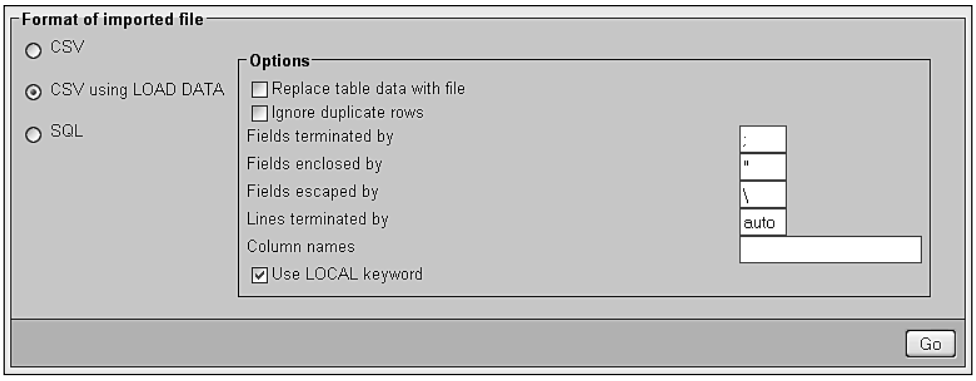
Chapter 18 explains phpMyAdmin's interface, for system administrators to manage privileges.

Usage of the `LOCAL` modifier in `LOAD DATA LOCAL INFILE` must be allowed by the MySQL server and MySQL's client library used by PHP.

Both the `LOAD` methods are available from the phpMyAdmin `LOAD` interface, which tries to choose the best possible default option.

Using the LOAD DATA interface

We select **Import** from the **author** table menu. Choosing **CSV using LOAD DATA** brings up the following dialog:



The available options have already been covered in the *CSV* section.

In the familiar **Location of the text file** question, we choose our `author.csv` file.

Finally, we can choose the `LOAD` method, as discussed earlier, by selecting the **Use LOCAL keyword** option. We then click **Go**.

If all goes well, we can see the confirmation screen as follows:

This screen shows the exact **LOAD DATA LOCAL INFILE** statement used. Here is what has happened:

- We chose **author.csv**.
- The contents of this file were transferred over HTTP and received by the web server.
- The PHP component inside the web server saved this file in a work directory (here **/opt/php-upload-tmp/**) and gave it a temporary name.
- phpMyAdmin, informed of the location of this working file, built a **LOAD DATA LOCAL INFILE** command, and sent it to MySQL. Note that just one query was executed, which loaded many rows.
- The MySQL server read and loaded the contents of the file into our target table. It then returned the number of affected rows (**2**), which phpMyAdmin displayed in the results page.

Web server upload directories

To get around cases where uploads are completely disabled by a web server's PHP configuration, or where upload limits are too small, phpMyAdmin can read upload files from a special directory located on the web server's file system. This mechanism is applicable for SQL and CSV imports.

We first specify the directory name of our choice in the `$cfg['UploadDir']` parameter, for example, `'./upload'`. We can also use the `%u` string, as described in Chapter 7, to represent the user's name.

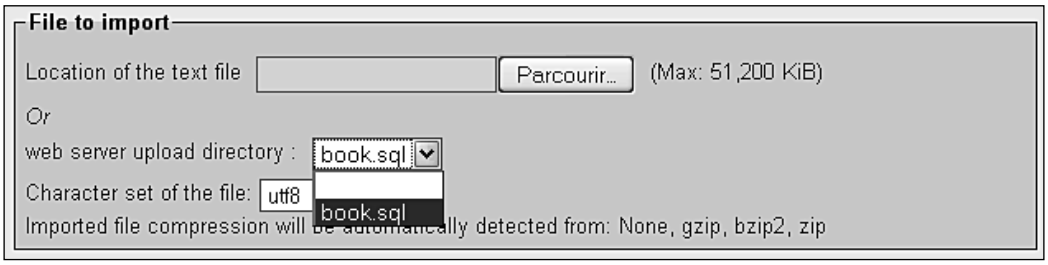
Now, let's go back to the **Import** subpage. We get an error message:

```
The directory you set for upload work cannot be reached.
```

This error message is expected, as the directory does not exist. It is supposed to have been created inside the current `phpMyAdmin` installation directory. The message might also indicate that the directory exists, but can't be read by the web server. (In PHP safe mode, the owner of the directory and the owner of the phpMyAdmin-installed scripts must be the same.)

Using an SFTP or FTP client, we create the necessary directory, and can now upload a file there (for example **book.sql**) bypassing any PHP timeouts or upload maximum limits. Note that the file itself must have permissions that allow the web server to read it. In most cases, the easiest way is to allow everyone to read the file.

Refreshing the **Import** subpage brings up the following:



Clicking **Go** should execute the file.

Automatic decompression is also available for the files located in the upload directory. The file names should have extensions such as `.bz2`, `.gz`, `.sql.bz2`, or `.sql.gz`.

Using the double extensions (`.sql.bz2`) is a better way to indicate that a `.sql` file was produced and then compressed, as we see all the steps used to generate this file.

Summary

This chapter covers:

- Various options in phpMyAdmin that allow us to import data
- The different mechanisms involved in importing SQL and CSV files
- The limits that we might hit when trying a transfer, and ways to bypass these limits

Chapter 9 will explain how to do single-table searches (covering search criteria specification) and how to search in the whole database.

Where to buy this book

You can buy Mastering phpMyAdmin 3.1 for Effective MySQL Management from the Packt Publishing website: `http://www.packtpub.com/mastering-phpmyadmin-3-1-fourth-edition/book`

Free shipping to the US, UK, Europe and selected Asian countries. For more information, please read our shipping policy.

Alternatively, you can buy the book from Amazon, BN.com, Computer Manuals and most internet book retailers.

www.PacktPub.com

For More Information:
www.packtpub.com/mastering-phpmyadmin-3-1-fourth-edition/book