How To Set Up A Debian Linux Database Server Using MySQL

The material on this page was prepared using **Sarge** or **Etch** configured using our Installation and Packages pages. If you did not use our pages to set up your system, what you encounter on your system may be different than what is given here.

When it comes to database servers, there are several free database server applications available. The intended use of the database server will have some impact on which database package you choose. MySQL is the most widely used free database server. However, it's use is primarily to act as a back end database for Web site applications written in embedded scripting languages like PHP because it doesn't support things like triggers, stored procedures, and replication (yet).

MySQL is a private for-profit company based in Sweden (which is why, unlike other open-source software providers, they have a ".com" domain). They offer a commerical license for businesses that want support directly from the vendor. They give the software away because it gives them, what CEO Marten Mickos calls, "...the world's largest quality assurance facility" (estimated at 4 million installations and growing by 30,000 downloads a day).

If you're looking to set up a serious enterprise database server for your LAN-based client/server applications, you may think that Oracle/Linux, with it's licensing costs, is the only option available to you. However, SAP AG has released their back-end database product under an open source license. "MaxDB" (formerly SAP DB) is the same industrial-strength database that is part of SAP AG's enterprise applications. MaxDB is the result of a partnership between SAP AG and MySQL. (See the Internet Resources page for links to a MaxDB and a SAPDB/Debian Quickstart page.)

The Database Server

All we have to do to install the MySQL package is enter the command:

```
apt-get install mysql-server
```

Select **Yes** to start MySQL at boot up and take note of the information given on the Install Hints screen.

Keep in mind that what you're working with here is a typically going to be the "back end" database server (the "server" part of "client/server"). The "front end" clients that will access the databases can use any number of applications, from Web pages written in PHP to off-the-shelf

query and reporting applications. These clients can be other Linux systems, UNIX systems, or Windows systems. (We'll get to that in a bit.) But since a database server isn't much good without some databases, we'll see how easy it is to create a simple one here.

Just like Linux systems keep track of who is accessing them so they can impose appropriate security, MySQL does the same thing so that not everyone can add records to a database or delete an entire database. In other words, you have to create MySQL user accounts the same way you have to create Linux user accounts. This is common with most server-based database applications. Starting out, there is only one account that can do anything in MySQL and that account name is **root**. (This is just a name that they gave to the account for the database administrator. It has nothing to do with the root account of the Linux system.) There is no password initially set for this account.

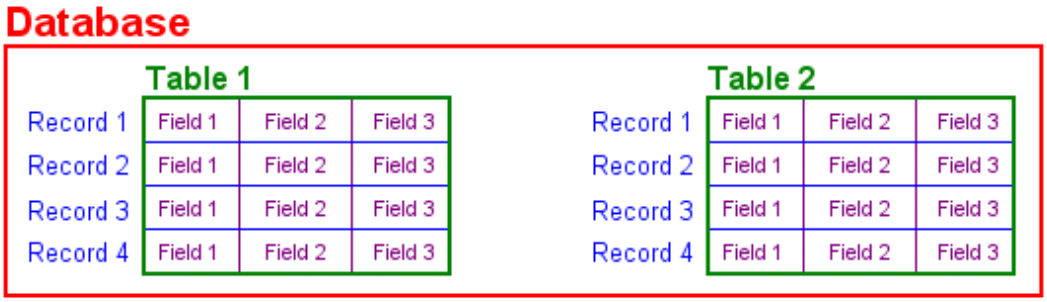
At the end of the install you'll see that MySQL was started. So now what? MySQL has an administration program called "MySQL Monitor" that you run to work with databases and tables, etc. The command to run the administration program is:

```
mysql -u root
```

When you run the program you are presented with a `>` prompt where you enter commands. The first command you may want to enter is `help` to view the list of available commands.

In addition to the commands displayed, MySQL Monitor will accept standard SQL commands (CREATE, DROP, SELECT, INSERT, etc.). Because standard SQL commands can be quite long, MySQL Monitor lets you enter them on multiple lines to aid in readability. As a result, **you must end each command with a semicolon (;)** to indicate the end of the command to MySQL Monitor. If you forget to do this and hit the Enter key, you'll get another `>` prompt. You can just enter the semicolon on that line by itself and hit Enter to execute the command.

Databases have a hierarchical structure. You create a **database**, and then you create one or more **tables** that are part of the database, and then each table contains **records**, and the records are comprised of **fields**.



The Database/Tables/Records/Fields hierarchy of databases is similar to the Workbook/Sheets/Rows/Cells hierarchy of spreadsheets. This is referred to as the "database structure". MySQL can host any number of databases, which can have any number of tables, which can have any number of records, which can have as many fields as you want. But the purpose of this is not to get into a lot of relational database theory. We just want to get a database server going.

We'll create a simple two-table database that can be used as an employee directory to replace a company's hard-copy telephone directory. A PHP Web page can then be written so that employees can access this electronic directory by pointing their browsers at an Intranet Web

server. So you don't have to go back and forth between two systems or two screens to enter the necessary commands, we put the commands from this section into a text file that you can print out. First we create the table with the command:

```
create database PhoneBook;
```

Now we have to tell MySQL that this is the database we want to work with using the command (note that like Linux and UNIX, **names are case-sensitive**):

```
use PhoneBook;
```

With the database created we have to create the tables that go into it. Since all rows in a table have the same fields, when we create a table we also specify the fields in it. We'll create a table called Employees. However, we **won't** use a semicolon (;) at the end of the command. This is so we can specify the fields at the same time. We do however, because this is a SQL statement, need to end the first line with an opening paran:

```
create table Employees (
```

Notice that when you press Enter you get another > prompt. This is so you can enter field information. Enter the following field information pressing enter at the end of each line:

```
empid char(10) not null,
lastname char(30) not null,
firstname char(20) not null,
deptno char(4) not null,
title char(30),
email char(25),
phone char(8),
fax char(8),
primary key(empid) )
;
```

Note on the second-to-the-last line is the closing paran to match the opening one in the `create` statement. The semicolon on the last line indicates the end of the SQL statement.

Now lets create a Department table. In the design of our database we're assuming a department is at one, and only one, location and that all employees of the department are at this location.

```
create table Departments (
deptno char(4) not null,
deptname char(30) not null,
address char(40),
city char(25),
state char(2),
zip char(5),
areacode char(3),
manager char(8),
primary key(deptno) )
;
```

Note that **both** tables have the **deptno** field. This is so we can "relate" the two tables when doing queries. The "primary key" specified for each table is a field that will contain a unique

value in each record so it can be used to uniquely identify each record. Also notice that we specified the CHARacter field type even for fields that will contain numeric data. You typically don't want to specify a field as a numeric data type unless you are going to be doing some mathematical calculations using that data.

Now that we have our database built, we need to put some actual data into it. The SQL command INSERT is used to do this. It is used to insert rows into tables. You can verify that both tables were created with the command:

```
show tables;
```

Now we'll add a couple records to the database. Sometimes it's helpful to see the list of fields when entering INSERT commands. You can do that with:

```
show columns from Employees;
```

Now here are the commands to insert a record into each table. You can add more if you want.

```
insert into Employees (
empid,lastname,firstname,deptno,title,email,phone)
values ("556219","Scuz","Joe","2318","Auditor","jscuz@us.com","555-1212")
;


show columns from Departments;
insert into Departments (
deptno,deptname,address,city,state,zip,areacode)
values ("2318","Audit","25 Loser Ln.","Dorksville","CA","90210","213")
;
```

Note that you don't have to specify, or enter values for, all fields. Also, you can list the fields in any order you want. It's just that the order of the values has to match the order of the fields when using the INSERT command. You can view the records you entered with the SQL command:

```
select * from Employees;
```

Thats it! We've got us a database. As an alternative to using the INSERT command, you can use a Debian package called **dbf2mysql** to transfer data from xbase files (dBase, FoxPro, etc.) into MySQL tables.

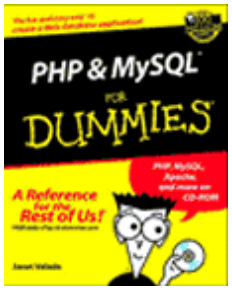
Security

We mentioned that MySQL's security includes the use of accounts and that a **root** user account was created during the install. However, MySQL doesn't identify users by their user name alone. It uses a combination of the user name and host name. In other words, not only does a user have to supply the correct user name, but they have to be supplying it from the correct workstation. By default the root user account can only access databases from "localhost" (i.e. the same system MySQL is running on) because the user account that was actually created by default was "root@localhost".

There's no reason you can't run Apache/PHP and MySQL on the same server. That way the front end client and back end server are the same box. You'll want to remember this user name / host name thing if you plan to set up an Apache/PHP Web server to access a database.

You'll want to create a user account or the Web application and you'll have to specify the host name of the Apache server, whether it's on the same or separate systems as MySQL.

Where to learn more - *The best of our bookshelves:*



More info...

If you want to pursue PHP and MySQL seriously, getting separate books that deal exclusively with each technology in depth is more appropriate. However, if you want to get up to speed quickly in both technologies in a low-cost manner **PHP and MySQL for Dummies** is the way to go. It provides a good overview of both while providing enough detail to get you fairly productive in each. It gets into using forms, cookies, and "sessions" for PHP and queries, security, backup/restore and table repair for MySQL. The CD contains the code for all of the sample pages given in the book which will save you some typing if you want to use them as a starting point for your own efforts.

MySQL comes with a database called **mysql** (note that it's all lowercase) and it contains a table called **users**. This is the table you use (when logged in as root) to create, modify, and delete user accounts. However, unlike other tables you don't modify this table using the INSERT, UPDATE, and DELETE SQL commands. You have to use the GRANT and REVOKE commands to manage user accounts and permissions.

The **%** can be used as a wildcard character to specify "all hosts". We'll want to do this for the root account so we can test client access. However, recall that user accounts are comprised of both the user name and a host name. So we can't modify the existing root account. We have to create a second one. To do this we issue the following SQL command in MySQL:

```
grant all on *.* to root@"%";
```

Now we're ready to try to access our database server from a client workstation.

Database Clients

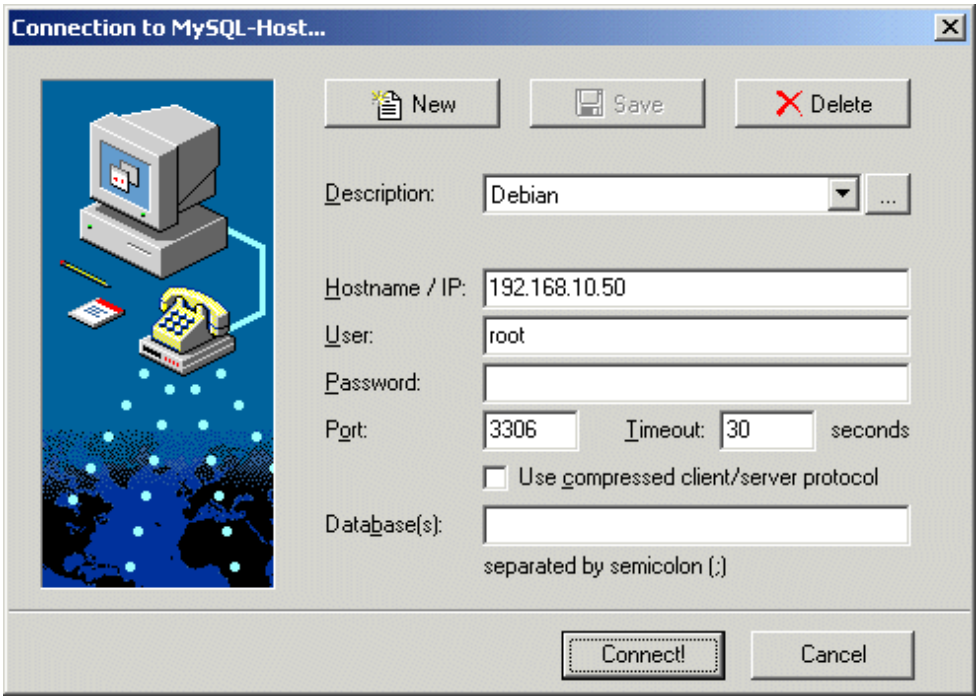
As mentioned previously, the typical use of MySQL is as a back end database for PHP Web pages acting as front end clients. However, this does not always have to be the case. MySQL clients are available in a varity of formats:

- The Linux shell command line client (Debian package name mysql-client)
- A Gnome GUI client called Gmyclient
- A Windows client called MySQL-Front
- A Macintosh client called YourSQL

You can even use Crystal Reports, Visual Basic apps, and other Windows-based database tools to go against MySQL databases using the myODBC driver. Since the majority of desktops are Windows-based, we'll cover the MySQL-Front program. Naturally you'll want to install it on a Windows PC that's on the same network as your database server.

The install of MySQL-Front was easy and worked great on my Windows 2000 Pro system. When you start it the first thing you have to define a new connection by clicking on the **New**

button. All you have to enter is "Debian" for a connection name and the IP address of your Debian server. It's important that you then click on the **Save** button. Then click on the **Connect** button.



You might receive the following error when you try to connect:

> Connection failed:
> 2003 - Can't connect to MySQL server on <ip address> (10061)

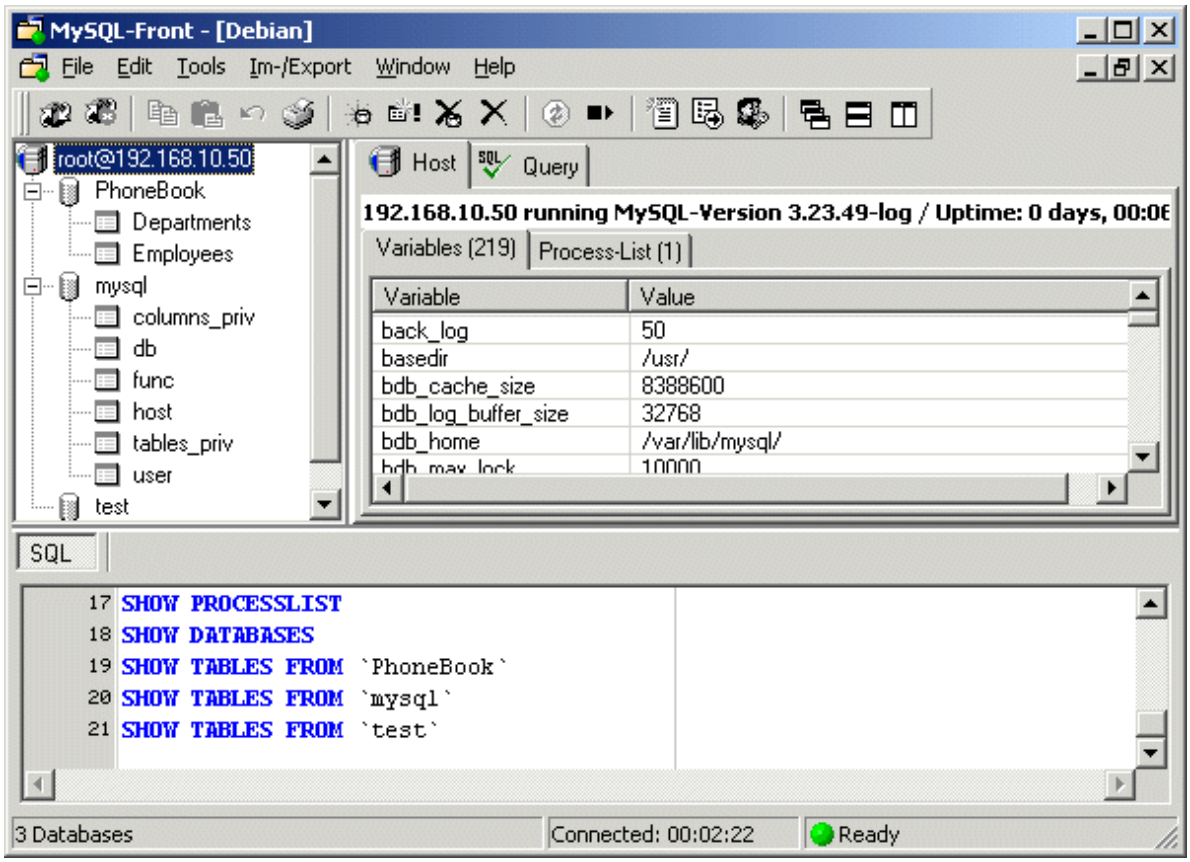
The 10061 error indicates the MySQL isn't listening for connections on the 3306 TCP port number. This may be disabled by default for security reasons. To resolve it, go to your Debian server and open the MySQL **my.cnf** file using the nano text editor with the following command:

```
nano /etc/mysql/my.cnf
```

Look for the line that starts with `bind-address` and change the `127.0.0.1` address to the address you assigned to the NIC. Exit the editor saving the file. Then restart MySQL with the command:

```
/etc/init.d/mysql restart
```

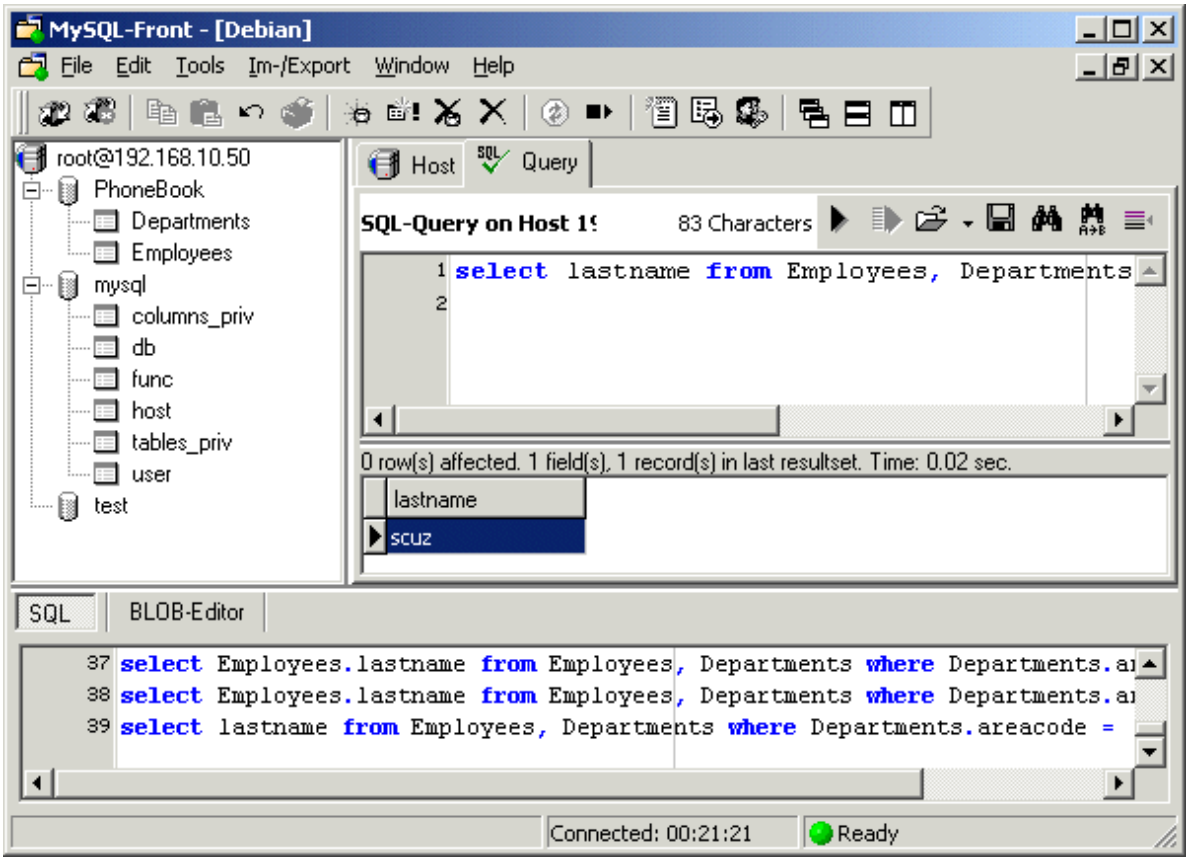
Now try using MySQL-Front to connect again. You should see a screen like this:



Click on the **Query** tab and enter the following query:

```
select lastname from Eployees,Departments where Departments.areacode='213'
```

and press the F9 key to execute the SQL statement.



You should see something similar to the above with our friend Joe Scuz's last name listed in the results.

This query was used to prove a point about relationships. We used a condition match in the Departments table to display a record from the Employees table. This was possible because both tables have the **deptno** field in common.

Note that the root@% user account we created with no password leaves everything wide open to everyone. If you're on a network with curious people, you may want to revoke that account.

The availability of clients for a wide range of platforms makes MySQL a viable database solution for a lot of organizations. With its cost (free) and support for standard SQL statements, students studying database technologies can easily set up their own database server to practice simple to complex queries and reinforce the theory presented in the classroom. And given that it will likely have a significant presence in the enterprise in the coming years, it'll be a good RDBMS to know.

SECURITY WARNING

Do NOT plan to use the system you will create using these guide pages as a "production" (real) server. It will NOT be secure!

There are many steps involved in creating a secure Internet or LAN server. While we do refer to some things you can do to make your system more secure, there are many other measures related to system security that also need to be taken into consideration and they are not covered on these pages.

These guide pages are meant as a learning tool only. The knowledge gained on these pages

will help you understand the material covered in security-related publications when you are ready to consider setting up a production server.

Did you find this page helpful ?
If so, please help keep this site operating
by using our DVD or book pages.

Site, content, documents, original images Copyright © 2003-2016 Keith Parkansky All rights reserved
Duplication of any portion of this site or the material contained herein without
the express written consent of Keith Parkansky, USA is strictly prohibited.

This site is in no way affiliated with the Debian Project, the debian.org Web site, or Software In The Public Interest, Inc. No endorsement of this site by the Debian Project or Software In the Public Interest is expressed or implied. Debian and the Debian logo are registered trademarks of Software In The Public Interest, Inc. Linux is a registered trademark of Linus Torvalds. The Tux penguin graphic is the creation of Larry Ewing.

LIABILITY

IN NO EVENT WILL KEITH PARKANSKY OR BLUEHOST INCORPORATED OR ANY OF ITS' SUBSIDIARIES BE LIABLE TO ANY PARTY (i) FOR ANY DIRECT, INDIRECT, SPECIAL, PUNITIVE OR CONSEQUENTIAL DAMAGES (INCLUDING, BUT NOT LIMITED TO, DAMAGES FOR LOSS OF BUSINESS PROFITS, BUSINESS INTERRUPTION, LOSS OF PROGRAMS OR INFORMATION, AND THE LIKE), OR ANY OTHER DAMAGES ARISING IN ANY WAY OUT OF THE AVAILABILITY, USE, RELIANCE ON, OR INABILITY TO USE THE INFORMATION, METHODS, HTML OR COMPUTER CODE, OR "KNOWLEDGE" PROVIDED ON OR THROUGH THIS WEBSITE, COMMONLY REFERRED TO AS THE "ABOUT DEBIAN" WEBSITE, OR ANY OF ITS' ASSOCIATED DOCUMENTS, DIAGRAMS, IMAGES, REPRODUCTIONS, COMPUTER EXECUTED CODE, OR ELECTRONICALLY STORED OR TRANSMITTED FILES OR GENERATED COMMUNICATIONS OR DATA EVEN IF KEITH PARKANSKY OR BLUEHOST INCORPORATED OR ANY OF ITS' SUBSIDIARIES SHALL HAVE BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES, AND REGARDLESS OF THE FORM OF ACTION, WHETHER IN CONTRACT, TORT, OR OTHERWISE; OR (ii) FOR ANY CLAIM ATTRIBUTABLE TO ERRORS, OMISSIONS, OR OTHER INACCURACIES IN, OR DESTRUCTIVE PROPERTIES OF ANY INFORMATION, METHODS, HTML OR COMPUTER CODE, OR "KNOWLEDGE" PROVIDED ON OR THROUGH THIS WEBSITE, COMMONLY REFERRED TO AS THE "ABOUT DEBIAN" WEBSITE, OR ANY OF ITS' ASSOCIATED DOCUMENTS, DIAGRAMS, IMAGES, REPRODUCTIONS, COMPUTER EXECUTED CODE, OR ELECTRONICALLY STORED, TRANSMITTED, OR GENERATED FILES, COMMUNICATIONS, OR DATA. ALL INFORMATION, METHODS, HTML OR COMPUTER CODE IS PROVIDED STRICTLY "AS IS" WITH NO GUARANTY OF ACCURACY AND/OR COMPLETENESS. USE OF THIS SITE CONSTITUTES ACCEPTANCE OF ALL STATED TERMS AND CONDITIONS.