A mSQL and perl Web Server Mini HOWTO

Table of Contents

A mSQL and perl Web Server Mini HOWTO......1
Oliver Corff, corff@zedat.fu−berlin.de......1
1.About this Document......1
2.Introduction......1
3.Installation Procedure......1
4.Running an Example Database......2
5.Conclusion and Outlook......2
1.About this Document......2
1.1 Intended Audience......2
1.2 Conventions used in this text......2
2.Introduction......3
3.Installation Procedure......4
3.1 Hardware Requirements......4
3.2 Software Requirements......4
3.3 Installing the Operating System......5
3.4 The http Server......6
3.5 The Browsers......6
Configuring Lynx......6
Configuring Arena......7
Installing and Configuring Netscape......7
3.6 Cooperation of Apache and Browsers......7
3.7 The Database Engine and its Installation......8
Installing msql−1.0.16......8
Testing msql−1......10
Installing msql−2.0.1......10
Testing msql−2......11
3.8 Choice of Interfaces: DBI/mSQL, MsqlPerl, and Lite......12
DBI and DBD−mSQL......12
MsqlPerl......13
msql's own scripting language: Lite......13
3.9 Going the generic way: DBI and DBD−msql......13
Installing perl's Database Interface DBI......14
perl's msql Driver DBD−mSQL......16
3.10 The MsqlPerl Interface......16
3.11 perl's CGI library......17
3.12 Installation Checklist......17
4.Running an Example Database......18
4.1 Adapting the sample script for MsqlPerl......18
4.2 Adapting the sample script for msql−2......19
5.Conclusion and Outlook......19

A mSQL and perl Web Server Mini HOWTO

Oliver Corff, `corff@zedat.fu-berlin.de`

v0.1, 17 September 1997

This Mini HOWTO, highly inspired by Michael Schilli's article Gebunkert: Datenbankbedienung mit Perl und CGI, published in the german computer magazine iX 8/1997, describes how to build a SQL client/server database using WWW and HTML for the user interface.

1.About this Document

- 1.1 Intended Audience
- 1.2 Conventions used in this text

2.Introduction

3.Installation Procedure

- 3.1 Hardware Requirements
- 3.2 Software Requirements
- 3.3 Installing the Operating System
- 3.4 The http Server
- 3.5 The Browsers
- 3.6 Cooperation of Apache and Browsers
- 3.7 The Database Engine and its Installation
- 3.8 Choice of Interfaces: DBI/mSQL, MsqlPerl, and Lite
- 3.9 Going the generic way: DBI and DBD−msql
- 3.10 The MsqlPerl Interface
- 3.11 perl's CGI library
- 3.12 Installation Checklist

4.Running an Example Database

- 4.1 Adapting the sample script for MsqlPerl
- 4.2 Adapting the sample script for msql−2

5.Conclusion and Outlook

1.About this Document

1.1 Intended Audience

Everybody who wants to install a web server database but does not know which software is necessary and how it is installed should benefit from reading this text. This text provides all information necessary to get a SQL database for a web server going; it does *not* go into any detail of CGI programming, nor does it explain the SQL database language. Excellent books are available on both topics, and it is the intention of this text to provide a working platform based on which a user can then study CGI programming and SQL.

For getting a small scale SQL system running (not the notorious example of a major airline booking system, or space mission management database) it will be sufficient to have the software described in this text and the documentation accompanying it. The user manual of msql (a database introduced in this text) provides sufficient information on SQL for building your own database.

The reader of this text should have a working knowledge of how to obtain files via `ftp` if he has no access to CD−ROMs, and a basic understanding of how to build binaries from sources. Anyway, all steps explained in this text were tested on a real life system and should also work on the reader's system.

1.2 Conventions used in this text

A user command:

```
# make install
```

Screen output from a program:

```
      Program installed. Read README for details on how to start.
```

Sample code of a file:

```
# My comment
char letter;
```

2.Introduction

It can be safely assumed that databases with a high volume of data or a complicated relational setup (like, perhaps, a lexical database for a living language) must be accessible to many users and operators at the same time. Ideally, it should be possible to use existing different hardware and software platforms that can be combined into the actual system. In order to reduce the implementation cost, only one system, the database server, needs to be powerful; the user stations typically just display data and accept user commands, but the processing is done on one machine only which led to the name client−server database. In addition, the user interface should be easy to maintain and should require as little as possible on the client side.

A system which meets these criteria can be built around the following items of protocols, concepts and software:

Linux

supplies the operating system. It is a stable Unix implementation providing true multi−user multi−tasking services with full network (TCP/IP e. a.) support. Except from the actual media and transmission cost, it is available free of charge and comes in form of so−called distributions which usually include everything needed from the basic OS to text processing, scripting, software development, interface builders, etc.

HTML

is the Hypertext Markup Language used to build interfaces to network systems like Intranets and the WWW, the World Wide Web. HTML is very simple and can be produced with any ASCII−capable text editor.

Browsers

are text−based (e. g. Lynx) or graphical (e. g. Mosaic, Netscape, Arena etc.) applications accepting, evaluating and displaying HTML documents. They are the only piece of software which is directly operated by the database user. Using browsers, it is possible to display various types of data (text, possibly images) and communicate with http servers (see next) on about every popular computer model for which a browser has been made available.

http servers

provide access to the area of a host computer where data intended for public use in a network are stored. They understand the http protocol and procure the information the user requests.

SQL

Structured Query Language is a language for manipulating data in relational databases. It has a very simple grammar and is a standard with wide industry support. SQL−based databases have become the core of the classical client/server database concept. There are many famous SQL systems available, like Oracle, Informix etc., and then there is also msql which comes with a very low or even zero price tag if it is used in academical and educational environments.

CGI

Common Gateway Interface is the programming interface between the system holding the data (in our case an SQL−based system) and the network protocol (HTML, of course). CGIs can be built around many programming languages, but a particularly popular language is perl.

perl

is an extremely powerful scripting language which combines all merits of C, various shell languages, and stream manipulation languages like awk and sed. Perl has a lot of modularized interfaces and can be used to control SQL databases, for example.

3.Installation Procedure

3.1 Hardware Requirements

No general statement can be made about the hardware requirements of a database server. Too much depends on the expected number of users, the kind of application, the network load etc. In a small environment with only a few users and little network traffic a i486−equivalent machine with 16 MB of RAM can be completely sufficient. Linux, the operating system, is very efficient in terms of resources, and can supply enough horse−power for running a broad variety of applications at the same time. Of course, faster processors and more RAM mean more speed, but much more important than the processor is the amount of RAM. The more RAM the system has the less it is forced to swap memory intensive processes to disk in case a bottleneck occurs.

Given anything like 32 MB RAM and a PCI bus, searches and sorting operations can be done without much resorting to swap files etc., resulting in lightening fast speed.

The model installation described in this article was made on a IBM 686 (133Mhz) with 32 MB RAM and a 1.2 GB IDE hard disk. Assuming that the installation process starts from scratch, here is a list of the necessary steps.

3.2 Software Requirements

The software described in this article is available from the Internet or from CD−ROM. The following products were used:

- Red Hat Linux PowerTools: 6 CD's Complete Easy–to–Use Red Hat 4.2, Summer '97; alternatively from `http://www.redhat.com`;
- msql SQL database server: it is now available in two versions. The versions have differences in the number of transactions they can handle, the administration interface, etc. The elder version, 1.0.16, is available from Sunsite mirrors. The ELF executable can be found at `sunsite:apps/database/sql/msql-1.0.16` or on CD–ROM (here: disc 4 of InfoMagic Linux Developer's Resource, 6–CD set, December 1996) or alternatively from the following URL: `http://www.infomagic.com`. The newer version, 2.0.1, can be directly obtained from Hughes' homepage in Australia (`http://www.hughes.com.au`) or from numerous mirror sites around the world;
- perl from CPAN: The Comprehensive Perl Archive Network. Walnut Creek CDROM, ISBN 1–57176–077–6, May 1997;
- Michael Schilli's CGI example program from computer journal iX 8/1997, pages 150––152, available via ftp from `ftp.uni-paderborn.de:/doc/magazin/iX`;

3.3 Installing the Operating System

Linux is installed in form of the Red Hat Linux Distribution 4.2. In order to install successfully, the machine must either have a DOS–accessible CD–ROM drive, a bootable CD–ROM drive, or else a boot disk must be made following the instructions on the Linux CD.

During installation the user has the choice to select and configure numerous software packages. It is convenient to select the following items now:

- TCP/IP network support,
- the http server Apache, and
- the scripting language perl, and
- the X Window System, as well as
- the browsers Arena (graphical) and Lynx (text–based).

All these packages are provided with the Linux distribution. If you do not install these packages now you still have the chance to do this later with the assistance of glint, the graphical and intuitive software package installation manager. Be sure to be root when installing these packages.

It is beyond the scope of this article to describe the network installation and initialization procedure. Please consult the online (manpages, HTML, texinfo) and printed (Linux Bible, etc. etc.) documentation.

The installation procedure of Red Hat is very mature and requires only little user attention besides the usual choices (like providing host names, etc.). Once the installation ends successfully, the system is basically ready to go.

Installing the X Window System is not mandatory for a pure server but it makes local access and testing much easier. The X installation procedure is done by any of several programs; XF86Setup offers the most extensive self–testing facilities and needs the least handling of hairy details (like video clock programming, etc.). The only requirement is that the software can detect the video adapter. A cheap accelerated graphics adapter (like Trio S64 based cards prior to S64UV+) usually works ``out of the box''.

At this point we assume that our system is up and running and that Apache, Perl and the X Window System have been successfully installed. We further assume that all standard structures like the file and directory structure are kept as they are defined in the installation. Last but not least we leave the host name as it is, and do at this moment accept the name `localhost`. We'll use this name for testing the installation; once the whole system works the true name can be added. Please note that the network setup also requires editing the files `/etc/hosts`, among others. Ideally this should be done with the administration tools provided to user root.

3.4 The http Server

The http server supplied with Linux is known as Apache to humans and as httpd to the system. The manpage (man httpd) explains how to install and start the http daemon (hence http*d*) but, as mentioned, if the installation went without problems then the server should be running. You can verify the directory tree: there must be a directory `/home/httpd/` with three subdirectories: `../cgi-bin/`, `../html/` and `../icons/`. In `../html/` there must be a file `index.html`. Later we will manipulate or replace this file by our own `index.html`. All configuration information is stored in`/etc/httpd/conf/`. The system is well preconfigured and does not need further setup provided the installation went without error.

3.5 The Browsers

There are essentially three types of browsers available for Linux: pure text–based systems like Lynx, experimental and simple ones like Arena (free!) and commercial ones like Netscape (shareware!) with Java support. While Lynx and Arena come with Linux, Netscape must be procured from other sources. Netscape is available as a precombiled binary for Linux on ix86 architectures and will run ``out of the box'' as soon as the archive is unpacked.

Configuring Lynx

Once Lynx is started it will look for a `default URL' which is usually not very meaningful if the system does not have permanent Internet access. In order to change the default URL (and lots of other configuration details) the system administrator should edit `/usr/lib/lynx.cfg`. The file is big, around 57000 bytes and contains occasionally contradicting information. It states its own home as `/usr/local/lib/`. Not far from top is a line beginning with `STARTFILE`. We replace this line by the following entry: `STARTFILE:http://localhost` and make sure that no spacing etc. is inserted:

```
# STARTFILE:http://www.nyu.edu/pages/wsn/subir/lynx.html
STARTFILE:http://localhost
```

After saving the file, Lynx should now reveal our `index.html` document if started without arguments.

Configuring Arena

Arena first looks for its own default URL when started without arguments. This URL is hard−wired into the executable but can be overrun by the environment variable WWW_HOME. The system administrator can place a line saying WWW_HOME="http://localhost" in /etc/profile. The variable must then be exported, either by a separate statement (export WWW_HOME) or by appending WWW_HOME to the existing export statement:

```
WWW_HOME="http://localhost"
export WWW_HOME
```

After relaunching a login shell, the new default URL is now system−wide known to Arena.

Installing and Configuring Netscape

Netscape is a commercial product and thus not included with the Linux distributions. It is either downloadable from the Internet or available from software collections on CDROM. Netscape comes in form of precompiled binaries for every important hardware platform. For installation purposes, it is useful to create a directory /usr/local/Netscape/ where the archive is unpacked. The files can be kept in place (except for the Java library: follow the instructions in the README file that comes with the Netscape binary), and it is sufficient to create a soft link in /usr/local/bin/ by issuing the command

```
# ln −s /usr/local/Netscape/netscape .
```

from within /usr/local/bin/.

Netscape is now ready for use and can be configured via the ``Options'' menu. In ``General Preferences'' there is a card ``Appearance'' with the entry ``Home Page Location''. Enter http://localhost here and do not forget to save the options (via ``Options'' −−− ``Save Options'') before exiting Netscape. At the next startup, Netscape will now show the Apache `homepage'.

3.6 Cooperation of Apache and Browsers

You can now conduct the first real test of both the browser and the http server: simply start any of the available browsers and the Apache: Red Hat Linux Web Server page will pop up. This page shows the file locations and other basics of http server installation. If this page is not displayed please check whether the files mentioned above are in place and whether the browser configuration is correct. Close edited configuration files before you start the browser again. If all files are in place and the browsers seem to be configured correctly then examine the network setup of your machine. Either the host name is different from what was entered in the configuration, or the network setup as such is not correct. It is utterly important that /etc/hosts contains at least a line like

```
127.0.0.1              localhost localhost.localdomain
```

which implies that you can connect locally to your machine. One can verify this by issuing any network−sensitive command requiring a host name as argument, like `telnet localhost` (provided `telnet` is installed). If that does not work then the network setup must be verified before continuing with the main task.

3.7 The Database Engine and its Installation

Installing the database requires only little more preparation than the previous installation steps. There are a few SQL database engines available with different runtime and administrative requirements, and possibly one of the most straightforward systems is msql, or ``Mini−SQL'' by David Hughes. msql is shareware. Depending on the version used, commercial sites are charged USD 250.00 and more, private users are charged USD 65.00 and more, and only educational institutions and registered non−profit organizations can use this software free of charge. Please note that the exact figures are provided in the licence notes of the database documentation. The figures given here serve as a rough indicator only.

A few words are in place here why the author chose msql. First of all, there is personal experience. While searching for a database engine the author found msql to be about the easiest to install and maintain, and it provides enough coverage of the SQL language to meet general needs. Only when writing these lines, the author discovered the following words of praise in Alligator Descartes' DBI FAQ (perl database interface FAQ):

> From the current author's point of view, if the dataset is relatively small, being tables of less than 1 million rows, and less than 1000 tables in a given database, then mSQL is a perfectly acceptable solution to your problem. This database is extremely cheap, is wonderfully robust and has excellent support. [...]

Msql is available in two versions now, msql−1.0.16 and msql−2.0.1, which differ in performance (not noticeable in small scale projects) and accompanying software (the newer version comes with more tools, its own scripting language, etc.). We will describe both versions of msql since their installion differs in a few points.

Installing msql−1.0.16

msql is available as source and as compiled executable with ELF support. Using the ELF binaries makes installation easy since the archive file `msql−1.0.16.ELF.tgz` contains a complete absolute directory tree so that all directories are generated properly when unpacked from `/`.

If you decide to compile msql−1.0.16 yourself and are going to use the MsqlPerl package rather than the DBI interface (see a detailed discussion on the difference between these two further down) then be prepared that MsqlPerl might complain during the test suites that some instruction inside msql failed. In this case a patch may be necessary which is described in the MsqlPerl documentation (file `patch.lost.tables`). Notably, this demands including three lines in `msqldb.c` after line 1400 which says `entry−>def =`

```
NULL;:

        *(entry->DB) = 0;
        *(entry->table) = 0;
        entry->age = 0;
```

The code fragment should now look like

```
        freeTableDef(entry->def);
        safeFree(entry->rowBuf);
        safeFree(entry->keyBuf);
        entry->def = NULL;
        *(entry->DB) = 0;
        *(entry->table) = 0;
        entry->age = 0;
```

Compiling msql involves several steps. After unpacking the source archive, it is necessary to build a target directory. This is done by saying

```
# make target
```

If successful, the system will then answer with

```
Build of target directory for Linux-2.0.30-i486 complete
```

You must now change into this newly created directory and run a

```
# ./setup
```

command first. The `./` sequence is necessary to make sure that really the command `setup` in this directory and not another command which happens to have the same name is executed. You will then be asked questions on the location of the source directory and whether a root installation is desired. These questions answered, the system should then run a number of tests checking for available software (compilers, utilities etc.) and finally say

```
Ready to build mSQL.

You may wish to check "common/site.h" although the defaults should be
fine.  When you're ready, type  "make all" to build the software
```

We say

```
# make all
```

If everything went as intended, we'll read:

```
make[2]: Leaving directory `/usr/local/Minerva/src/msql'
<-- [msql] done

Make of mSQL complete.
You should now mSQL using make install

NOTE : mSQL cannot be used free of charge at commercial sites.
       Please read the doc/License file to see what you have to do.

make[1]: Leaving directory `/usr/local/Minerva/src'
```

All binaries must then be made visible to the search paths by creating soft links in `/usr/local/bin/`. Change to that directory and issue the command

```
# ln -s /usr/local/Minerva/bin/* .
```

after which the links will be properly set.

Testing msql−1

After the installation it is now possible to test whether the database works. Before anything else is done, the server daemon must be started. The system administrator holding root privileges issues the command

```
# msqld &
```

(do not forget to add the `&`, otherwise msql won't run in the background.) after which the following screen message appears:

```
mSQL Server 1.0.16 starting ...

Warning : Couldn't open ACL file: No such file or directory
Without an ACL file global access is Read/Write
```

This message tells us that everything so far worked since we did not set up any access restrictions. For the moment it is sufficient to start the msql daemon from within a shell but later we may want to have the system startup automatically execute this command for us. The command must then be mentioned in a suitable `rc.d` script. Only now the administrator can issue the first genuine database command:

```
# msqladmin create inventur
```

msql replies by saying `Database "inventur" created.`. As a further proof, we find that the directory `/usr/local/Minerva/msqldb/` contains now the empty subdirectory `../inventur/`. We could manipulate the newly created database with the administration tools; these procedures are all covered in detail in the msql documentation.

Installing msql−2.0.1

There is now a newer, more powerful version of Hughes' mSQL server available the installation of which is different in a few points. Installing msql−2 from scratch involves the following steps. Copy the archive to your extraction point, e. g. `/usr/local/msql-2/`, then untar the archive:

```
# tar xfvz msql-2.0.1.tar.gz
```

Change to the root direction of the install tree and issue a

```
# make target
```

Change to `targets` and look for your machine type. There should be a new subdirectory `Linux−`*(your version)−(your cpu)/*. Change to that directory and start the setup facility located here:

```
# ./setup
```

There is also a file `site.mm` which can be edited. Maybe you have got used to the directory name `/usr/local/Minerva/` and want to preserve it? In this case change the `INST_DIR=...` line to your desired target directory. Otherwise, leave everything as it is.

Now you can start building the database:

```
# make
# make install
```

If everything went successfully, we'll see a message like:

```
        [...]

        Installation of mSQL−2 complete.

        *********
        **   This is the commercial, production release of mSQL−2.0
        **   Please see the README file in the top directory of the
        **   distribution for license information.
        *********
```

After all is installed properly we have to take care of the administration details. Here, the real differences from msql−1 begin. First, a user `msql` is created which is responsible for database administration.

```
# adduser msql
```

Then we have to change all ownerships in the mSQL directory to `msql` by saying:

```
# cd /usr/local/Minerva
# chown −R msql:msql *
```

Then we create soft links for all database binaries in `/usr/local/bin/` by saying:

```
# ln −s /usr/local/Minerva/bin/* .
```

Testing msql−2

We can now start the database server by issuing the command `msql2d &` and should get a response similar to this one:

```
        Mini SQL Version 2.0.1
        Copyright (c) 1993−4 David J. Hughes
        Copyright (c) 1995−7 Hughes Technologies Pty. Ltd.
        All rights reserved.

                Loading configuration from '/usr/local/Minerva/msql.conf'.
```

```
            Server process reconfigured to accept 214 connections.
            Server running as user 'msql'.
            Server mode is Read/Write.

        Warning : No ACL file.  Using global read/write access.
```

That looks perfect. The database is compiled and in place, and we can now continue with the perl modules since these rely partially on the presence of a working database server for testing.

Accidentally, this is also a good moment to print the complete manual that comes with msql−2.0.1:

```
# gzip −d manual.ps.gz
# lpr manual.ps
```

We can proceed to building the interfaces now, but it is a good idea to keep the newly created SQL server up and running since that makes testing the interface libraries somewhat simpler.

3.8 Choice of Interfaces: DBI/mSQL, MsqlPerl, and Lite

A frequently quoted saying in the Camel Book (the authorative perl documentation) states that there is more than one way to achieve a result when using perl. This, alas, holds true for our model application, too. Basically there are three ways to access an msql database via CGI. First of all the question is whether or not perl shall be used. If we use perl (on which this article focuses) then we still have the choice between two completely different interface models. Besides using perl, we can also employ msql's own scripting language, called Lite, which is reasonably simple and a close clone of C.

DBI and DBD−mSQL

By the time of this writing, using perl's generic database interface called DBI is the method of choice. DBI has a few advantages: It provides unified access control to a number of commercial databases with a single command set. The actual database in use on a given system is then contacted through a driver which effectively hides the pecularities of that database from the programmer. Being such, using DBI provides for a smooth transition between different databases by different makers. In one single script it is even possible to contact several different databases. Please refer to the DBI−FAQ for details. There is, however, one drawback: The DBI interface is still under development and shows rapidly galloping version numbers (sometimes with updates taking place within less than a month). Similarly, the individual database drivers are also frequently updated and may rely on specific versions of the database interface. Users making first−time installations should stick to the version numbers given in this article since other versions may cause compilation and testing problems the trouble shooting of which is nothing for the faint−hearted.

MsqlPerl

MsqlPerl is a library for directly accessing msql from perl scripts. It bypasses the DBI interface and is fairly compact. Though it works fine with both versions of msql, its usage is not promoted anymore in favour of the generalized DBI interface. Nonetheless, in a given installation it may prove to be the interface of choice since it is small and easy to install. Notably, it has less version dependencies than revealed by the interaction of DBI and particular database drivers.

msql's own scripting language: Lite

Last but not least msql−2 comes with its own scripting language: Lite. The language is a close relative of C stripped of its oddities with additional shell−like features (in a way, something like a very specialized version of perl). Lite is a simple language and is well documented in the msql−2 manual. The msql−2 package also comes with a sample application sporting Lite.

We will not describe Lite here because it is well documented but fairly specific to msql−2, and because it is assumed that the readers of this article have a basic interest in and a basic understanding of perl. Nonetheless it is highly recommended to have a closer look at Lite: it may well be the case that Lite offers the solution of choice in an exclusive msql−2 environment (implying no other databases are involved) due to its simplicity and straightforward concept.

3.9 Going the generic way: DBI and DBD−msql

We assume that perl was installed during the system setup or via the package manager mentioned above. No further details will be given here. Nonetheless we first test whether our version of perl is up to date:

```
# perl −v
```

perl should respond with the following message:

```
This is perl, version 5.003 with EMBED
        Locally applied patches:
          SUIDBUF − Buffer overflow fixes for suidperl security

        built under linux at Apr 22 1997 10:04:46
        + two suidperl security patches

Copyright 1987−1996, Larry Wall
[...]
```

So far, everything is fine. The next step includes installing the perl libraries for databases in general (DBI), the msql driver (DBD−mSQL) and CGI. The CGI driver is necessary in any case. The following archives are necessary:

1. DBI−0.81.tar.gz
2. DBD−mSQL−0.65.tar.gz
3. CGI.pm−2.31.tar.gz (or higher)

A caveat is necessary here for beginners: the test installation described here works fine using software with *exactly* these version numbers, and combinations of other versions failed in one or the other way. Debugging flawed version combinations is nothing for those who are not very familiar with the intimate details of the calling conventions etc. of the interfaces. Sometimes only a method is renamed while performing the same task, but sometimes the internal structure changes significantly. So, again, stick with these version numbers if you want to be on the safe side even if you discover that version numbers have increased in the meantime. Frequent updates of these interfaces are the rule rather than the exception, so you should really anticipate problems when installing other versions than those indicated here.

It is very important that the database driver for mSQL (DBD−mSQL) is installed *after* the generic interface DBI.

We start by creating the directory `/usr/local/PerlModules/` as it is very important to keep the original perl directory tree untouched. We could also choose a different directory name since the name is completely uncritical, and unfortunately that is not really mentioned in the README files of the verious perl modules. Having copied the above−mentioned archives to `/usr/local/PerlModules/` we unpack them saying

```
# tar xzvf [archive-file]
```

for every single of the three archives. Do not forget to supply the real archive name to `tar`. The installation process for the three modules is essentially stardardized; only the screen messages showing important steps of individual packages are reproduced here.

Installing perl's Database Interface DBI

The database interface must always be installed before installing the specific database driver. Unpacking the DBI archive creates the directory `/usr/local/PerlModules/DBI-0.81/`. Change to that directory. There are a `README` file (you should read it) and a perl−specific makefile. Now issue the command

```
# perl Makefile.PL
```

The system should answer with a lengthy message of which the most important part is shown here::

```
[...]
MakeMaker (v5.34)
Checking if your kit is complete...
Looks good
        NAME => q[DBI]
        PREREQ_PM => {  }
        VERSION_FROM => q[DBI.pm]
        clean => { FILES=>q[$(DISTVNAME)/] }
        dist => { DIST_DEFAULT=>q[clean distcheck disttest [...]
Using PERL=/usr/bin/perl
```

```
WARNING! By default new modules are installed into your 'site_lib'
directories. Since site_lib directories come after the normal library
directories you MUST delete old DBI files and directories from your
'privlib' and 'archlib' directories and their auto subdirectories.

Writing Makefile for DBI
```

This looks good, as the program says, and we can proceed with the next step:

```
# make
```

If no error message occurs (the detailed protocol dumped on screen is *not* an error message) we test the newly installed library with the command

```
# make test
```

Watch the output for the following lines (you can always scroll back with `[Shift]-[PgUp]`):

```
[...]
t/basics............ok
t/dbidrv............ok
t/examp.............ok
All tests successful.
[...]
DBI test application $Revision: 1.20 $
Switch: DBI-0.81 Switch by Tim Bunce, 0.81
Available Drivers: ExampleP, NullP, Sponge
ExampleP: testing 2 sets of 5 connections:
Connecting... 1 2 3 4 5
Disconnecting...
Connecting... 1 2 3 4 5
Disconnecting...
Made 10 connections in  0 secs ( 0.00 usr  0.00 sys =  0.00 cpu)

test.pl done
```

The final step is to install all files in their proper directories. The following command will take care of it:

```
# make install
```

No more duties are left. If for some reason the installation failed and you want to redo it do not forget to issue

```
# make realclean
```

first. This will remove stale leftovers of the previous installation. You can also remove the files which were installed by copying the screen contents (shown abbreviated)

```
Installing /usr/lib/perl5/site_perl/i386-linux/./auto/DBI/DBIXS.h
Installing /usr/lib/perl5/site_perl/i386-linux/./auto/DBI/DBI.so
Installing /usr/lib/perl5/site_perl/i386-linux/./auto/DBI/DBI.bs
[...]
Writing /usr/lib/perl5/site_perl/i386-linux/auto/DBI/.packlist
Appending installation info to /usr/lib/perl5/i386-linux/5.003/perllocal.pod
```

into a file, replacing every `Installing` with `rm`. Provided you named the file `uninstall` you can then say

```
# . uninstall
```

which will remove the recently installed files.

perl's msql Driver DBD−mSQL

The msql driver can only be installed *after* a successful installation of perl's generic database interface.

The basic steps are the same as above; so first go through

```
# perl Makefile.PL
```

Here, the system should answer with an urgent warning to read the accompanying documentation. It will then detect where msql resides, and asks which version you use:

```
$MSQL_HOME not defined. Searching for mSQL...
Using mSQL in /usr/local/Hughes

 -> Which version of mSQL are you using [1/2]?
```

State your correct version number. Quite a few lines of text will follow. Watch for the following ones:

```
Splendid! Your mSQL daemon is running. We can auto-detect your configuration!

I've auto-detected your configuration to be running on port: 1114
```

You can now test the driver by saying

```
# make test
```

Again, a lengthy output follows. If it ends with

```
Testing: $cursor->func( '_ListSelectedFields' ). This will fail.
        ok: not a SELECT in msqlListSelectedFields!
Re-testing: $dbh->do( 'DROP TABLE testaa' )
        ok
*** Testing of DBD::mSQL complete! You appear to be normal! ***
```

you are on the safe side of life and can install your driver by saying

```
# make install
```

You are now ready to go and can skip the next paragraph.

3.10 The MsqlPerl Interface

If you decide to use the exclusive MsqlPerl interface then no generic database driver is needed, only `MsqlPerl-1.15.tar.gz`, since, as mentioned earlier, MsqlPerl provides a direct interface between perl and the database server without using the DBI interface. Installing and testing is straightforward.

After saying `perl Makefile.PL` the make utility can be started. First you have to answer the question where mSQL resides. If it resides in `/usr/local/Minerva/` the default answer can be confirmed.

Then do a `make test`. Before doing so you must ensure that you have a database named `test` and that you have read and write permissions for it. This can be done by

```
# msqladmin create test
```

3.11 perl's CGI library

Installing perl's CGI part is the simpliest of the three steps. Execute the following commands in the given order and everything is done:

```
# perl Makefile.PL
# make
# make install
```

Unlike the previous drivers this interface does not have a test option (`# make test`) whereas the other modules *should* be tested in any case.

A subdirectory with CGI example scripts is also created. You can copy the contents of this directory into `/home/http/cgi-bin/` and use the browser to experiment with the scripts.

3.12 Installation Checklist

We went through the following steps, in this order:

1. Install Linux with networking support
2. Install a http server, e. g. Apache
3. Install a browser, e. g. Arena, lynx or Netscape
4. Install an SQL server, e. g. msql
5. Install a suitable perl SQL interface
6. Install the CGI files

Finally, you can do some clean−up. All source trees for msql and the perl modules can be safely deleted (however, you should not delete your archive files!) since the binaries and documentation are now based in different directories.

4.Running an Example Database

After completing the system installation we can now finally run a model application. Depending on the version of msql installed and the perl database interface used, we have to modify the sample programs in a few points.

First however, the file `index.html` residing in `/home/httpd/html/` must be modified to allow calling a sample database application. We can place our database (which we call `database.cgi` or `inventur.cgi` here despite its archive name `perl.lst.ck`) in `/home/httpd/html/test/`.

We add one line (of course, depending on your installation choices) similar to the following to `index.html`:

```
<LI>Test the <A HREF="test/database.cgi">Database, DBI:DBD-mSQL style!</A>
<LI>Test the <A HREF="test/inventur.cgi">Database, MsqlPerl style!</A>
```

Usually you should only pick one of these two choices but if you have both types of database interface installed you can leave both lines here as they are. You can then compare performance, etc.

4.1 Adapting the sample script for MsqlPerl

Our sample script has to be told to use the MsqlPerl interface. The modification takes place in several locations. First, near the beginning of the file, we change the `use` clause:

```
#
# use DBI;            # Generisches Datenbank-Interface
use Msql;
```

Then, near line 27, the MsqlPerl syntax does not require the mentioning of a specific driver:

```
# $dbh = DBI->connect($host, $database, '', $driver) ||
$dbh = Msql->connect($host, $database) ||
```

Then, from line 33 onward throughout the whole script, we have to change all instances of `do` against `query`:

```
# $dbh->do("SELECT * FROM hw") || db_init($dbh);
$dbh->query("SELECT * FROM hw") || db_init($dbh);
```

Finally, in MsqlPerl speak, line 207 can be commented out:

```
# $sth->execute || msg("SQL Error:", $sth->errstr);
```

In addition, it may become necessary to swap all `errstr` calls like the one in the preceding code fragment against `errmsg`. This is also version dependent.

After these modifications, the script should run smoothly.

4.2 Adapting the sample script for msql−2

The SQL syntax was redefined during the development of mslq−2. The original script will fail to execute the table initialization statements in lines 45 −− 58. The `primary key` modifier is no longer supported by msql−2, and should simply be skipped:

```
    $dbh->do(<<EOT) || die $dbh->errstr; # Neue Personen-Tabelle
        create table person (
# We do not need the 'primary key' modifier anymore in msql-2!
#           pn        int primary key,   # Personalnummer
            pn        int,               # Personalnummer
            name      char(80),          # Nachname, Vorname
            raum      int                # Raumnummer
        )
EOT
    $dbh->do(<<EOT) || die $dbh->errstr; # Neue Hardware-Tabelle
        create table hw (
# We do not need the 'primary key' modifier anymore in msql-2!
#           asset int primary key,       # Inventurnummer
            asset int,                   # Inventurnummer
            name   char(80),             # Bezeichnung
            person int                   # Besitzer
        )
EOT
```

Unfortunately, this specific script will then accept new entries with identical personnel numbers; the msql−1 modifier `primary key` intends to prevent exactly this behaviour. The msql−2 documentation shows how to use the `CREATE INDEX` clause to create unique entries.

5.Conclusion and Outlook

If you have installed msql−2 on your system then you can have a look at the sample programs written in Lite, msql−2's own scripting language.

Either version of msql comes with a basic set of administration tools which allow the user to create and drop tables (`msqladmin`) and examine database structures (`relshow`).

The second generation msql (i.e. msql−2) has a few more genuinely useful utilities: `msqlimport` and `msqlexport`. These allow the dumping of flat line data files into and out of the SQL database. They can be used for loading quantities of existing data *d'un coup* into existing tables, or extract flat data from tables, and the user does not have to deal with writing a *single* line of perl or SQL or whatever code for this task.

If you want to write your own perl scripts dealing with databases you'll find sufficient support in the example files and the extensive on−line documentation that comes with the DBI module.

Anyway, you are now ready to go and present your data to the users of your own network, or even the WWW.