# Open Source database systems
# TAME YOUR DATA

**When your address book is full and your card file is packed to bursting, it's time to start looking for a more suitable database system. Andreas Bauer takes a look at your options**

Before you start looking at expensive proprietary database solutions, it's worth considering Open Source candidates. In this article we would like to introduce the concept of Free databases and explain the practical application of the appropriate tools.

The use of larger database management systems (DBMS) requires that you have at least some idea of what to expect from that sort of system. Typical characteristics include data security, integrity, performance, external interfaces and the user interface to the DBMS itself. A common misconception is that a spreadsheet, such as Excel or StarCalc, and a DBMS provide similar functionality. As you'll see this isn't the case at all.

When looking for suitable applications you will mainly come across the names of the following well-established Open Source systems: MySQL, mSQL and PostgreSQL. Each of the systems mentioned is perfectly usable, though you'll find some are stronger in certain areas than others. You can find more information on this under *http://www.postgresql.org* and *http://www.mysql.org*.

## Spoilt for choice

In this article we have decided to investigate PostgreSQL more closely. Not because we think that it is superior to any of the others, but because the basic functionality is the same as that of most SQL databases.

Initially called Ingres and later Postgres95, PostgreSQL originated at Berkley University and has been developed by a worldwide group of Open Source programmers since 1996. PostgreSQL is the only Free database that has always supported complex processing techniques such as transactions (see the Transactions box for more details).

## Database creation

Each database requires either a whole partition on the hard disk or at least a clearly defined area of the filesystem in which to store information. Some rpm packages of PostgreSQL automatically create */var/lib/pgsql/data* for this purpose. Otherwise, it normally makes sense to deposit data in */usr/local/pgsql/data*.

```
mkdir /usr/local/pgsql/data
chown postgres /usr/local/pgsql/data
```

Now change your user identity to that of the user "postgres" with *su postgres* and create the skeleton of your database:

```
initdb -D /usr/local/pgsql/data
```

You are now ready to start the server for the first time. Stay logged in as postgres and enter the following start command on the console:

```
postmaster -i -D /usr/local/pgsql/data &
```

This completes the setup. PostgreSQL is now ready.

## The famous address book

So as not to overcomplicate things we are going to use the familiar and simple concept of address management for our sample application. As postgres, you create this with the following command:

```
createdb Addresses
```

This command only defines the name of the database. You will need to specify later what sort of information the address management system is going to contain. For this purpose we have worked out a self-explanatory database layout (see Figure 1), which can be easily transferred to any CD or recipe collection.

## Transactions

Transactions are a useful DBMS feature when it comes to ensuring integrity during changes to data. For example, imagine Mr. Miller would like to transfer £50,000 to Mrs. Jones. The database will first of all attempt to subtract this amount from Mr. Miller's account. Once this is completed successfully, the database can then add the amount to Mrs. Jones' account as a second step. These are essentially two separate operations.

However, if there should be a power failure after the first step then Mr. Miller would be £50,000 poorer but Mrs. Jones would still be waiting for her money. This scenario would not have happened with a transaction, as both operations would be executed simultaneously, once the DBMS has ensured that the transaction is permissible. In the case of a system failure or other problem, partial operations are simply reversed (rolled-back) so that no inconsistencies can arise.

Transactions aren't unique to PostgreSQL, of course; these days MySQL also supports this facility, with minor limitations.

## Program installation

Many Linux distributions already contain PostgreSQL. If the packages have not already been set up on your system you can simply use the normal CD-ROM installer. Alternatively, you can also download the appropriate files in various package formats, from one of the mirror sites. The source files are also available, but compiling such extensive programs is best left to computer buffs and is not necessary for our purposes.

PostgreSQL, like MySQL and mSQL, is a server process that runs constantly in the background. For security reasons it is therefore important that it's not started by the administrator, root. After the installation you should create a dummy (if one does not already exist) with which to start the PostgreSQL server, using the command

```
adduser postgres
```

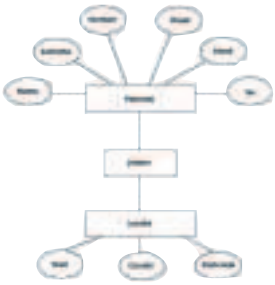

**Figure 1: Database layout "Address management"**

## Structured query language

Advanced systems like PostgreSQL or MySQL have powerful entry and query facilities, for which SQL has pretty much established itself as the standard (see the SQL box). If you already have some knowledge of this descriptive language you will be able to use familiar commands in PostgreSQL. Even if you've never heard of SQL there's no need to despair, as Open Source database systems are supported by various graphical front-ends that don't require the use of complicated commands.

We should mention at this point that these front-ends are often less polished than you might be used to from the likes of Microsoft Access or Filemaker. Even though those commercial products do have an SQL-enabled core, this is thoroughly shielded from the user. In the Open Source arena there are some more promising attempts, but in many situations some hands-on work in the SQL interpreter simply can't be avoided.

## SQL

SQL (Structured Query Language) is a descriptive language, i.e. its commands describe specific data (records) that are to be retrieved, deleted or edited. For example, in order to find everyone with the first name "Albert" in the data table Personal you would enter the following command in the SQL interpreter:

```
SELECT Name, Surname, Number, Street, Town FROM Personal WHERE
Name='Albert';
```

Instead of specifying the fields you are interested in following SELECT, you can simply use '*' to see all the fields in a table. The search criteria are contained in the WHERE clause; modifications and even nesting are permitted. Another short example:

```
SELECT * FROM Personal WHERE Name='Albert' AND Surname='Einstein';
```

You can find detailed information on SQL and command overviews on the Internet, as well as in the manuals for PostgreSQL.

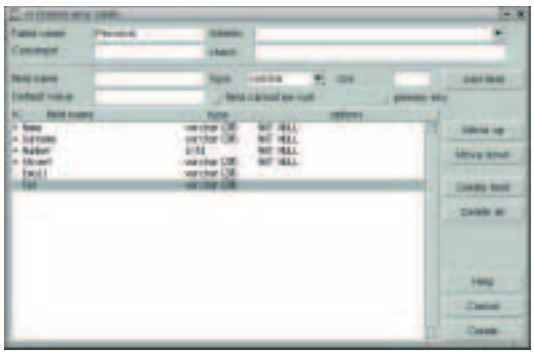

**Figure 2: Creating new tables and fields**

## The user-friendly approach

The good news for PostgreSQL users is that the server comes complete with one of the better database front-ends. If you've installed all the packages correctly, you should have the Tcl/Tk application *pgaccess* on your hard disk.

Start *pgaccess* under your normal login (postgres) and open the newly created empty database Addresses via the menu *Database/Open*. New data tables are created with the New button on the upper menu bar.

Figure 2 shows a new table created according to our pre-defined layout. The fields that are preceded by a '*' are primary keys (in the database layout these are indicated with underscores). Together they uniquely identify a data record, in a similar way to a serial or ID number. The data types used are varchar, a character string, and int4, an integer numeral. The option NOT NULL ensures that this field is not accidentally omitted during the entry of new data.

Now proceed in a similar way for the table *Personal*, keeping precisely to the layout as defined in Figure 1. The connections between the two tables require a slightly different approach. You need to create a third table, for instance *LivesIn*, which allocates the people to the places. Don't worry, such connections are always created according to the same principle: you take the primary keys of the two tables and create a single new table with them all.

You've now created a database on your PostgreSQL server that is ready for you to use. Double-clicking on a table opens it and enables you to edit its content. The great thing is that you've not had to use any SQL

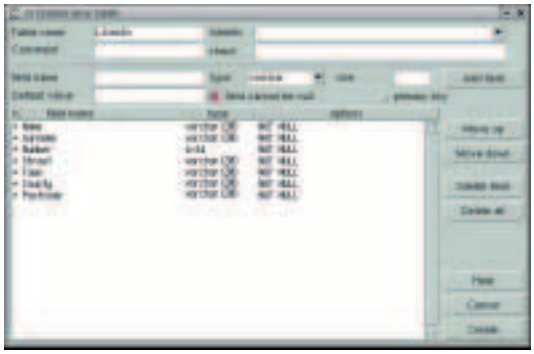

**Figure 3: The relation "LivesIn"**

yet, nor will you need to for these basic functions. Nevertheless, we would like to introduce you to the SQL process at this point, because it is universal and very similar in each SQL database.

## Command line

Although beginners may find it a little tricky, databases can be created more directly using SQL commands. PostgreSQL has an interpreter called *psql* for this very purpose.

As user postgres you can log onto the database server with the following command:

```
psql Addresses
```

The result doesn't look particularly impressive, but don't be fooled – you are now linked directly to the database and can manually create and edit tables. To save you some time we have prepared three short, self-explanatory scripts, that you will also find on the coverdisc (create_tables.sql, fill_tables.sql, delete_tables.sql). These scripts can be loaded and executed using the following commands:

```
\i create_tables.sql
\i fill_tables.sql
```

At this point we are not going to go into the content of these scripts. They are simple ASCII text files, which you can view with any editor. The SQL commands they contain could, of course, be just as easily entered manually, but that would take time and effort, especially as you'll want to experiment with the database. Should anything go wrong and you want to return the database to its original state that's not a problem. Simply execute the script *delete_tables.sql* and start again.

```
\i delete_tables.sql
```

As you can see, this manual approach is very elegant and above all portable. No matter whether you end up working with MySQL, mSQL or Oracle, the commands in the three scripts will be interpreted correctly by any of these systems. Even the graphical front-end *pgaccess* only performs the SQL commands that are represented by your actions on the interface.

Try to take a bit of time at this point to experiment with the interpreter. Insert data with the *INSERT* command, or use *SELECT* to find data records. Aside from the sample scripts, the links listed below are particularly good starting points for your first steps in using this descriptive language. To leave *psql* use \q.

## Practical application

Normally the database server stores our information and some sort of software provides the appropriate graphical interface, without the user coming into direct contact with the DBMS. This means that you'll need to decide how you want your data to be presented externally, and who will be working on it. The **PHP** system has become the established way of creating Web-based database applications. In most cases HTML forms are sufficient for data entry and simple query functions.

Unlike commercial databases, Open Source solutions hardly ever enable users to create suitable applications using the system itself. One of the advantages of PostgreSQL is that you can decide how you are going to create your database and which additional programs and applications will have access to it. There are, for example, Free drivers for the languages PHP, Java and Tcl. Accessing your data in C or C++ is also no problem.

There are a few readymade examples that we wouldn't want to deprive you of. For instance, the GNOME Photo Collector (a GNOME application for managing photo albums), or the PHP version, PhpPgGBox, which needs a Web browser to power the display. More general front-ends for data manipulation are also available. One of them is dbMetrix, which is also very similar to *pgaccess*, but has the advantage of supporting many different database servers, including MySQL.

## Conclusion

Using a proper database is completely different to working with a spreadsheet or an electronic scheduler. The decision on whether you really want to administer your data in such a complex system should be based on how important it is to you that the stored information can be processed further or made accessible to other system components (Web pages, programs, scripts, etc.). Database servers like PostgreSQL also impress through their multi-user capabilities, scalability, robustness and openness. However, these characteristics are not required in every case.

This article is intended primarily to provide a simple introduction to the subject and to show you that an Open Source DBMS is more than able to meet even professional requirements. What's more, a Free solution based on SQL is a future-proof system. Do compare different Free database servers, however, it may even be worth investing in an introductory SQL book.

**PHP** PHP is an HTML-embedded scripting language, which is similar to Perl and in terms of syntax, related slightly to C. It provides a simple way of creating dynamically generated pages using a database for building up the page content.

## The author

Andreas Bauer is studying Information Technology and Psychology at Munich University. He also created a Web-based student database system at the Australian National University, Canberra, based entirely on free software. He can be contacted at *baueran@in.tum.de*.

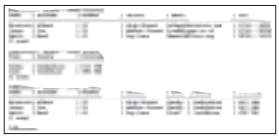

**Figure 4: Overview of all tables**

## Information

| | |
|---|---|
| PostgreSQL site | *http://www.postgresql.org/* |
| Download mirror | *ftp://ftp.ie.postgresql.org/mirrors/ftp.postgresql.org* |
| MySQL site | *http://www.mysql.org/* |
| SQL tutorials | *http://www.sqlcourse.com/* |
| SQL tutorials | *http://w3.one.net/~jhoffman/sqltut.htm* |
| PHP site | *http://www.php.net/* |
| Gnome Photo Collector | *http://gpc.sourceforge.net/* |
| Photo Gallery | *http://madness.bira.gen.tr/proj/PhpPgGBox/* |
| Data manipulation | *http://www.tamos.net/sw/dbMetrix/* |