# Embedded databases 4
# MySQL

Michaël Van Canneyt

April 2, 2006

**Abstract**

In this fourth article about embedded databases, the embedded MySQL server is tested. MySQL is not really meant as an embedded server, and this shows: While possible in theory, in practice, the embedded MySQL server is not easily used for Delphi or Lazarus programs.

## 1 Introduction

MySQL is a very popular database: it's presence on Linux installations is proverbial. It's the natural compagnon of PHP. It's easy in use and easy to set up. It has a dual license: It's free if used for open source projects, but one needs a license if it's used in a commercial solution. But it has some drawbacks as well, specially for the Pascal programmer.

MySQL and Object Pascal programming are not an easy combination. The reason for this are the frequent changes in the MySQL API: While superficially the API remains equal, the changes in data structures from version to version make it very hard to create a uniform and stable MySQL interface. And although MySQL claims that the client library can be used to connect to a server of a different version, this is also not completely correct. For example: connecting to a 5.0 server with a 4.1 client library is not possible.

Free Pascal comes with 5 versions of an interface to the MySQL C client library; They cannot be used interchangeably.

Because of the versioning and API issues, in practice, there are only 2 viable options:

1. Use ODBC. ODBC has a fixed interface, and MySQL delivers an ODBC driver with each version of MySQL. However, the overhead of ODBC is considerable, both in terms of run-time overhead as in terms of deployment.
2. Create a network-protocol interface: The network protocol of MySQL has remained more stable than the raw client API. However, this means that the embedded server cannot be used. Nevertheless, 2 companies have created components for Object Pascal which use this approach: they connect through the network protocol. More information will be given below.

Despite the constantly changing API, there exist some free components that allow to connect to MySQL directly through the client-library: ZeosLib, which works with Delphi and FPC, and SQLDB, developed specially for FPC.

Both can - reportedly - be used to connect to the embedded version of MySQL.

# 2 Installing Embedded MySQL

Embedded MySQL comes under the form of a library: libmysqld. This library contains the complete mysql server and can be used to compile the mysql server into a program. For Windows, a shared library (.dll) and a static library are available. For Linux, a static library is available, but this can be converted to a shared library.

The MySQL 5.0.19 distribution for Windows does not contain an embedded MySQL library. The MySQL bug tracker contains an entry about this (14316), with an answer saying that more development and QA is needed before the embedded server will be included in 5.0.

The 4.1 distribution for Windows does contain an embedded version. In the lib/opt directory of the mysql 4.1 installation, both a shared and a static version of libmysqld can be found.

On Linux, some distributions contain a mysql-max package, which contains a static version of the mysql server library. If this is not installed, then an embedded library must be compiled manually.

When the filemysqld library is not installed, it can be easily compiled. After unpacking the source distribution, the following commands will build MySQL with the (static) embedded server library:

```
cd mysql-5.0.19
configure --with-embedded-server
make
```

The actual directory name with the sources depends on the downloaded version. Possibly additional options must be given, but the above produces a working embedded server.

The static library (called libmysqld.a) can be converted to a dynamic library using the following command:

```
cd libmysqld
gcc -shared -o libmysqld.so.5.0.19 -Wl,-soname,libmysqld.so.5
-Wl,--whole-archive,libmysqld.a,--no-whole-archive
```

All this should be on one line, and the 19 should be replaced with the patch number of the MySQL installation. The resulting dynamical library (libmysqld.so.5.0.19) can be used as the embedded server. It should simply be copied somewhere to a known location.

# 3 Using embedded MySQL in Delphi

There is no simple or cheap way of using embedded MySQL in Delphi.

Unfortunately, there are no standard components that come with Delphi, which allow to connect to MySQL 5, or to MySQL 4.1: the dbExpress driver for Delphi 7 is suitable for MySQL 3.X only, for Delphi 2005, only MySQL 4.0 is supported. MySQL 4.0 is no longer on the download pages of the MySQL website.

Now, there are 2 commercial solutions to access MySQL 4.1 or 5.0:

```
http://www.crlab.com/
http://www.microolap.com/
```

Both companies provide native MySQL Data access components, and DBExpress drivers that should work with all versions of MySQL, including the embedded version.

ZeosLib contains components to connect to MySQL. However, again, the stable version of ZeosLib only supports up to version 4.0 of MySQL.

The Zeoslib 6.5.1 Alpha has no driver for MySQL 5.0, but does provide a MySQL 4.1 driver: It can be compiled and installed like any Dlephi package. However, upon installing and using it, it fails to open any query with MySQL 4.1. trying to connect to a MySQL 5.0 server with a 4.1 driver or client library does not work at all.

The tracker test database was created and filled with data under both MySQL 4.1 and 5.0, using the client/server version, but running a simple test query as

```
select
  PU_ID,COUNT(PT_ID)
from
  pupil
  left join pupiltrack on (PU_ID=PT_PUPIL_FK)
GROUP BY
  PU_ID;
```

(from the mysql query tool) threw the MySQL 4.1 server in an infinite loop, causing the computer to hang.

The MySQL 5.0 command-client runs the query fine, but MySQL 5 has no embedded solution on Windows.

After 5 days of trying, the author has given up on trying to get MySQL 4.1 or 5.0 to work with Delphi, either in embedded or in normal client-server mode. Despite having almost 10 years of experience with MySQL, all attempts to get it to work properly have utterly failed.

It is possible that the commercial solutions prove more usable, but this has not been tested.

# 4 MySQL and Lazarus

For Lazarus, the situation is better. FPC (specifically: SQLDB) comes with access components for MySQL 4.0, 4.1 and 5.0. The components can be configured to use a custom MySQL client library, so the embedded library can simply be copied to a known location and used as-is.

To be able to use the embedded MySQL server, the dynamical MySQL interface in freepascal must be used. The dynamical interface differs from the statical interface in that it loads the client library at runtime, instead of binding the client library to the program at compile time. The units which implement this dynamical interface are called mysql50dyn, mysql41dyn and mysql40dyn.

These units contain a call `InitialiseMySQL`, which has an optional string argument: The library name to load. If no argument is given, a default is tried.

The MySQL API has 2 additional calls which must be used when using MySQL as an embedded server:

```
Function mysql_server_init(argc:longint; argv:PPchar;
                           groups:PPchar):longint;
Procedure mysql_server_end;cdecl;
```

The `mysql_server_init` call will start the embedded server. Likewise, the `mysql_server_end` call ends the server. A word of warning: this may take a while: the initial start takes a while because the InnoDB engine sets up some houskeeping tables. Likewise, the ending of the

server introduces some overhead. It is therefore recommendable to start the server at program startup, and to end it only when the program is finished.

The `mysql_server_init` call takes 3 arguments:

**argc** The number of null-terminated strings in the second argument.

**argv** A pointer to an array of pointers to null-terminated strings. each string in the array can be a command-line argument as it would be given to the MySQL server when started from the command-line.

**groups** A pointer to an array with section names to be used when loading defaults in the configuration file. The sections

It's possible to create a set of strings for argv to completely configure the embedded server. However, it's easier to create a configuration file for the server, and simply point the embedded server to this configuration file.

The following file contains a minimal configuration for an embedded server. It uses the 'embedded' section, which will be read by default by the embedded server.

```
[embedded]
language=/home/michael/source/mysql-5.0.19/sql/share/english
datadir=/home/michael/source/embedded/mysql/data
```

Two entries are needed.

**language** This entry points the server to the location of the message files. The message files should be distributed together with the application. The server won't start without it, and will exit the application if they are not found.

**datadir** This entry gives the embedded server the location of the data: each subdirectory of this directory will be considered a database. Some housekeeping files are written in this directory as well.

More entries can be provided in this file; for a full description, see the MySQL manuals.

With all this, enough information is available to create the SQL executor application and the browser application for the pupil trackdata.

# 5 SQL executor program

The SQL executor program was presented in the previous articles in this series. Since SQLDB is used again, the program works in exactly the same way as the SQL executor program for embedded firebird. The only difference is that instead of a Firebird connection component, a MySQL 5.0 connection component is used.

To tell the component that it should use the embedded library, the following code is added to the initialization code of the program:

```
initialization
  {$I frmmain.lrs}
  InitEmbedded;
Finalization
  DoneEmbedded;
```

As can be seen, it starts and stops the embedded server.

Starting the embedded server is done in the following code:

```
Var
  MySQLOpts : Array[1..2] of string;

Procedure InitEmbedded;

Const
  EmbeddedMySQLLib =
   '/home/michael/source/libmysqld/libmysqld.so';
  DefaultsFile = '--defaults-file=';

Var
  S : String;

begin
  InitialiseMySQL(EmbeddedMySQLLib);
  MySQLOpts[1]:=ExtractFileName(Paramstr(0));
  S:=ExtractFilePath(Paramstr(0))+'../embedded.ini';
  MySQLOpts[2]:=DefaultsFile+ExpandFileName(S);
  mysql_server_init(2,PPChar(@MysqlOpts),Nil);
end;
```

It starts by loading the correct library. Then an array of 2 ansistrings is created: the first of the 2 strings contains the program name. This corresponds to the unix `argv[0]` argument: the program name, or `paramstr(0)` in pascal terms. This should *always* be present when passing arguments to the server: The second string is the option which sets the location of the configuration file. (The directory above the installation directory). After the array is constructed, the `mysql_server_init` call is made.

Stopping the engine is much more simple:

```
Procedure DoneEmbedded;

begin
  mysql_server_end();
end;
```

The rest of the SQL executor is the same as the one for embedded firebird.

# 6 Pupil track data browsing program

The browsing program for pupil track data ('browser') was introduced in the previous articles: it shows 2 grids, one with a list of pupils, one with the tracking data for the currently selected pupil.

The program is completely similar to the one that uses embedded firebird: instead of a firebird connection component, a MySQL connection component is used. In code, the only difference is

1. The startup code for the embedded server. This code is identical to the one for the SQL executor program.

2. MySQL does not support parameters. Therefore the master-detail between the `QPupils` query and the `QPupilTrackData` query must be coded manually by resetting the SQL statement. This is done in the `AfterScroll` event of the `QPupils` query, as it was done for SQLite:

```
procedure TMainForm.QPupilsAfterScroll(DataSet: TDataSet);

Const
  SSelectTrackData
    = 'SELECT * FROM PUPILTRACK WHERE PT_PUPIL_FK=%d';

Var
  ID : Integer;
  S  : String;

begin
  QPupilTrackData.Close;
  if not Dataset.FieldByName('PU_ID').IsNull then
    begin
    ID:=Dataset.FieldByName('PU_ID').AsInteger;
    S:=Format(SSelectTrackData,[ID]);
    QPupilTrackData.SQL.Text:=S;
    QPupilTrackData.Open;
    end;
end;
```

The code is self-explanatory. The resulting program is shown in figure 1 on page 7

# 7 Timings

Living up to its reputation, embedded MySQL is very fast indeed, as can be seen from the following table:

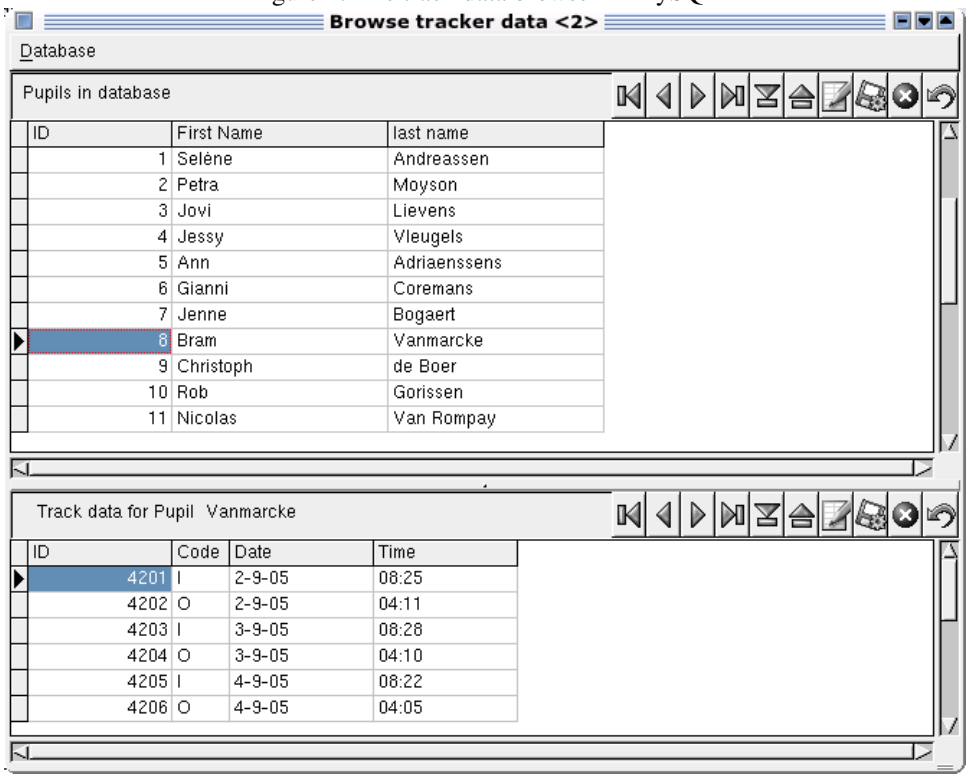
| Test | SQLite | Firebird | MySQL |
|---|---|---|---|
| 1 | 0:0:42.00 | 0:1:29.47 | 0:40.48 |
| 2 | 0:6:12.75 | 0:0:02.58 | 0:02.40 |
| 3 | 0:0:00.67 | 0:0:00.44 | 0:02.43 |
| 4 | 0:5:59.38 | 0:0:00.48 | 0:02.44 |

The times are larger than the firebird times, but they include the times for starting/stopping the MySQL engine. The starting/stopping of the engine takes more than 2 seconds, bringing the real execution time close to that of SQLite or embedded firebird. Only in the first test (where 600.000 records are inserted), the speed of MySQL becomes apparent.

# 8 Conclusion

Setting up MySQL as an embedded server is possible, but not trivial. For Delphi, the author has not succeeded in getting it to work without resorting to commercial components. For Lazarus, it proved possible, but only after a lot of custom tuning and lots of browsing for information on the Internet, where the 2 main problems proved to be getting the embedded server compiled as a dynamical library, and configuring the embedded mysql server. Additionally, it should be mentioned that the MySQL embedded server writes a lot of output

Figure 1: The track data browser in MySQL




messages to the console, which is not nice at all. But, finally, it proved possible to get it to work, and once working, it worked fast.

All the problems lead to the conclusion that MySQL (embedded or not) and Pascal programming are not natural compagnons. In the opinion of the author, if an open source solution should be used for an embedded database, MySQL should be avoided. Hopefully, this will change in time.