Embedded databases

Michaël Van Canneyt

September 23, 2005

Abstract

There isn't almost any program out there which doesn't use a database, be it implicit or explicit. For those that explicitly need a database, there is always the consideration that the program should distribute easily, which is easiest using an internal database, but that as the program grows, moving to an external database should still be an option. Fortunately, these 2 requirements can be met using a single database engine. An exploration.

1 Introduction

Most, if not all, computer applications use a database of some sort. Some do this explicitly, because they maintain data. Others do not explicitly manage data in a database, but store settings on a computer, through the registry, INI files or XML configuration files. All three are databases, but simple ones.

For applications that explicitly use a database, there are many considerations to be made when choosing a database. Two important factors in the decision about a database engine are:

1. Ease of installation. Not any user can (nor wants to) install a database server on his system. For many 'small' programs with a wide target audience this is an important factor. An MP3 playlist manager, or a CD or Video or even a book collection manager application are simple examples.
2. Scalability. If the application has room for growth, or it is anticipated that the same data will be accessed by many users at once, simple flat files on the local (or even shared) harddisk are no longer suitable.

An invoicing & cash registry program in a medium to large store falls actually in both categories. The book collection application as well: it could for instance be installed in a school, to manage the school's library. With computers being a commodity in each classroom, it's not so unnatural to expect the teacher to be able to reserve a book for one of his pupils, from his terminal in the classroom. The developers of such applications need to make a decision which meets both requirements, which seem mutually exclusive.

Fortunately, this need not to be so. Increasingly many database products exist in a classical client-server version as well as an "embedded" version: the 'server' is built into the application, either by direct inclusion of code or possibly by means of a dynamic library.

If migration to another database is also a requirement or at least a future option, then it is important to use a generally accepted standard of accessing the database: The standard is SQL. Thefore, only embedded engines that support multiple tables and SQL statements will be considered. Even if migration to another database is not needed, the advantage of

SQL is that it allows to define a multitude of views on the various records in the database, without needing to code each and every one of the possibilities: This in itself is a reason to opt for an embedded SQL engine instead of a plain table-based database (like e.g. dBase files).

2 Embedded database engines

As stated above, there are a number of engines that can be embedded into your database. For this series of articles, only databases that can be used in Delphi or Lazarus will be considered: this excludes a number of Java-based database engines.

In the subsequent, an alphabetical list of embedded databases which support SQL is reviewed; Several aspects of the engine are considered, such as the price tag attached to it.

Advantage Database Server

Advantage database server from Extended Systems is a commercial database engine that supports traditional client-server programming as well as embedded programming. The database can be run on Windows as well as on Linux (Intel based). It supports all features found on most RDBMS systems: Standard SQL, transactions, stored procedures, triggers, referential integrity, autonumber data type. Minimal installation size is only 4Mb.

For embedded programming (using the Local Advantage Database engine), it is sufficient to deploy a couple of external DLLs with the application.

Programming is quite easy: It is not necessary to acquire a separate set of `TDataset` components, they are delivered with the Advantage Database Server. The native `TDataset` components provided by the installation allow to use the Advantage database Server at once.

Upscaling is just a matter of setting up the server, installing a couple of additional dlls on the client locations, and reconfigure the application to use the remote server - this is as simple as setting some properties of the Advantage database components.

Firebird

Firebird - the open source successor of Interbase - also offers an embedded version of the firebird server. It is simply the Firebird server shipped inside the client DLL; As such it offers all possibilities of the firebird server, with the exception of listening for remote connections: The application is the only application using the database. It supports all features also supported by it's bigger brother: Standard SQL, transaction, stored procedures, triggers, referential integrity, autonumber data type. Installation size is limited to the size of the client DLL.

There exist a number of different ways to program Firebird applications in Delphi or Lazarus: IBX, dbXpress, IBOBjects, FIBPlus, ZeosLib for Delphi; ZeosLib, UIB, SQLDB for Lazarus. None of these require any special setup for embedded programming: all that needs to be done is to specify the correct database location.

Similar to the Advantage database server, upscaling is just a matter of setting up the server, installing the client dlls on the client, and reconfigure the application to use the remote database: This means specifying the location of the server and the username/password of the user one wishes to connect as: A matter of setting a few additional properties.

The Firebird license allows it to be used in both commercial and open source programs for free, both on Linux as on Windows.

MySQL

Mysql is a ubiqitous database server in the open source community. Despite this it seems a little-known fact that (as of version 4.1.X) it can also be used as an embedded database. It is usually not shipped in its embedded form, but as a separate server binary. The principle for the embedded engine is very much the same as for Firebird: the server engine is included in the client library. The MySQL engine is more limited than Firebird or Advantage: the current stable release misses support for triggers, lacks referential integrity or transactions. Although the latter 2 can be enabled by using an alternative (slower) storage engine called InnoDB.

The embedded engine is as easy to deploy as the full client-server version, all it requires is an extra library with the embeddded server. Programming the embedded version is similar to the traditional client/server programming, except that 2 additional calls must be added to the program: one to start the server engine, and one to end it.

Several components exist to access MySQL from Delphi/Lazarus: ZeosLib, dbExpress for Delphi, ZeosLib and SQLDB for Lazarus. Additionally, ODBC should also work. To work with the embedded engine, 2 additional calls must be used to start and stop the engine; Currently none of the componentsets supports this: they must be declared separately.

Depending on how the client library is loaded, upscaling the application again consists of deploying the correct client libraries, and pointing the application to the database server. The calls to start and stop the server should be omitted when a connection is attempted to a remote server.

The MySQL license allows it to be used in a GPL program for free, but a license is required when using it in a commercial application.

NexusDB

The commercial NexusDB engine is the successor of TurboPower's FlashFiler product (available for free from SourceForge): The initial version of NexusDB was a reworked version of FlashFiler 3. As such, NexusDB was originally designed as a database engine to be compiled into the end-user application, but the possibility of running as a separate server was added. The engine supports SQL, stored procedures, triggers, transactions and referential integrity.

As for Advantage Database, the NexusDB database ships with native Delphi components, which are descendents of `TDataset`. Contrary to the Advantage Database, it does not work on Linux. The feasability of porting these components to Lazarus is under study at the moment, but is uncertain: the whole NexusDB engine needs to compile in Lazarus.

Upscaling the application is a matter of setting up the server and specifying the new server location in the client application: dropping a couple of components and setting some properties.

SQLite

SQLite is an open source implementation of an embedded SQL engine. It is designed to be used as an embedded engine, and does not offer a stand-alone server with remote connection possibilities (although there is code out there to do this). The advantage of this is that it is very fast. It features most of the SQL syntax and transactions; Some support for triggers exists. SQLite does not enforce data constraints, i.e. it is possible to insert a string in an column declared as an integer. While this is considered a feature by the SQLite developers, this is something that must be taken into consideration when creating

Table 1: Various options for engines

Option	Adv.	Fireb.	Mysql	Nexus	SQLite	TurboDb
SQL	+	+	+	+	+	+
Transactions	+	+	+	+	+	-
Stored Procs	+	+	-	-	-	-
Triggers	+	-	-	-	+/-	-
External DLL	+	+	+	-	+	-
Compiled-in	-	-	-	+	+	+
Upscaling	+	+	+	+	-	-
Backup	+	+	+	+	-	-
Users	+	+	+	+	-	-
Linux support	+	+	+	-	+	+

applications for SQLite, since the columns in a result of a query are not guaranteed have the same type in all records.

SQLite is cross-platform, and comes as a library with a very simple API. There exist several Delphi components for SQLite, which port easily to Lazarus, and lazarus has it's own SQLite components.

A third-party version of SQLite compiled specially for Delphi exists: DISQlite. It compiles completely into the executable.

Upscaling an application written for SQLite will require a lot of work: It would be necessary to write a server application to actually process the queries, and to develop a communication protocol. Finally the client application would need to be rewritten to use the communication protocol. While possible in principle, this is not really an easy task.

TurboDB

The commercial TurboDB is an SQL engine much like Nexus DB and Advantage database, but, lik SQLite, exists only as an embedded engine. It compiles into the application and allows to query a database with an SQL syntax which is a subset of SQL-92. It does not offer support for Triggers, Stored procedures or transactions.

It comes with a set of TDataset components, as well as some lower-level components, and runs on Linux just as well as on Windows.

As with SQLite, upscaling of a TurboDB-based application will require a lot of work, essentially the same work as for SQLite must be done.

3 Conclusion

There are a lot of options when one wants to use an embedded database; Both open source and embedded engines are available, each with it's specific uses and drawbacks. Which database to choose depends on the Development tool used, and on the options one needs. The table 1 on page 4 summarizes the various possibilities of the engines; this should allow to make a rough shift between the various engines. Only the main points are shown there, for more detailed information, the websites of the corresponding engines should be visited. In contributions to follow, a closer look will be taken at some of these embedded databases.

4 Resources

More information on the systems described here can be found on their respective homepages:

Advantage Database `http://www.extendedsystems.com/`

Firebird `http://www.firebirdsql.org/`

Mysql `http://www.mysql.com/`

NexusDB `http://www.nexusdb.com/`

SQLite `http://www.sqlite.org/`

TurboDB `http://www.turbodb.de/`

A thorough comparision of SQL engines can be found on:

`http://en.wikipedia.org/wiki/Comparison_of_relational_database_management_systems`