Embedded databases 5
Advantage Database Server

Michaël Van Canneyt

June 1, 2006

Abstract

The Advantage Database Server engine works client/server and in embedded form. In this article, it's use as an embedded database server is examined, both under Delphi (Windows) and Lazarus (Linux). As in the previous articles in this series, the pupil tracker database will be used to measure it's performance.

1 Introduction

The Advantage Database Server (currently at version 8) is a complete database server technology, available for Windows and Linux, usable in Client/Server or in local configurations.

Advantage Database server is a full-featured database server: It understands SQL, provides transaction management, has a stored procedure language, features triggers. It also has backup facilities, features replication, comes with a graphical management console. It's use is not limited to Delphi: there is an ODBC driver and a OLE DB driver, which should make it available to just about any programming language for Windows out there, a .NET provider and also a PHP module, making it available for web development under Linux and Windows.

Advantage Database Server is a commercial solution, but the local engine, which is distributed with the Delphi source code of the Advantage TDataset component, can be redestributed for free. It can be downloaded from the website of extended systems:

`http://www.extendedsystems.de/`

The Advantage `TDataset` component for Delphi 7 and for Kylix are the downloads that were used to write the code for this article. For Windows, the Advantage Data Architect is a useful download, it contains a GUI tool to execute queries, create databases and some other management tasks.

Although written for Delphi, the code compiled under Lazarus with virtually no changes: a couple of extra conditional defines were introduced, and that is all. A patch for the code is supplied with the rest of the source code of this article. The patch has also been submitted to the people at Extended systems, so maybe a next release of Advantage Database Server will come with support for Lazarus out of the box.

The product can be installed simply, it's a normal windows install program. On linux, the installation is meant for Kylix, but the needed files are in subdirectories of the directory setup.files:

source the source code for the TDataset components. The files can be copies to wherever they are needed to compile your projects.

redistribute these are the libraries and support files which should be redistributed with your application.

2 Architecture and Distribution

In the local configuration, the whole server is included as a dynamically loadable library with the application that uses it. The `TDataset` code directly calls the code of the library, so it is the application which does all the data processing.

In fact, there are 2 libraries that must be distributed together with the application:

ace The client interface library for the Advantage Database Engine. (ace32.dll on windows, libace.so.8.00.1 on Linux)

adslocal The local database engine itself. (adsloc32.dll on Windows, libadsloc.so.8.00.1 on linux)

If support for character sets other than the default character set is needed, the files ansi.chr and `extend.chr` should also be distributed.

Under Windows, these files should be located next to the program. No other changes are necessary.

Under Linux, this can also be done, but 2 additional actions are needed:

1. Symbolic links must be made to the library files that omit the minor number from the library name, so a link called libace.so.8.00 and a link calledlibadsloc.so.8.00.
2. The dynamical loader must be told to look for the files next to the program library. This can be done by setting the `LD_LIBRARY_PATH` environment variable:

   ```
   export LD_LIBRARY_PATH=/path/to/your/application
   ```

Or the libraries can be put in the /usr/lib directory.

Some additional configuration can be made by creating a `adslocal.cfg` file with some configuration directives. Notably, the character sets to be used can be set there (more information about possible configuration options can be found in the help files accompagnying the product).

3 Creating a database: SQL support

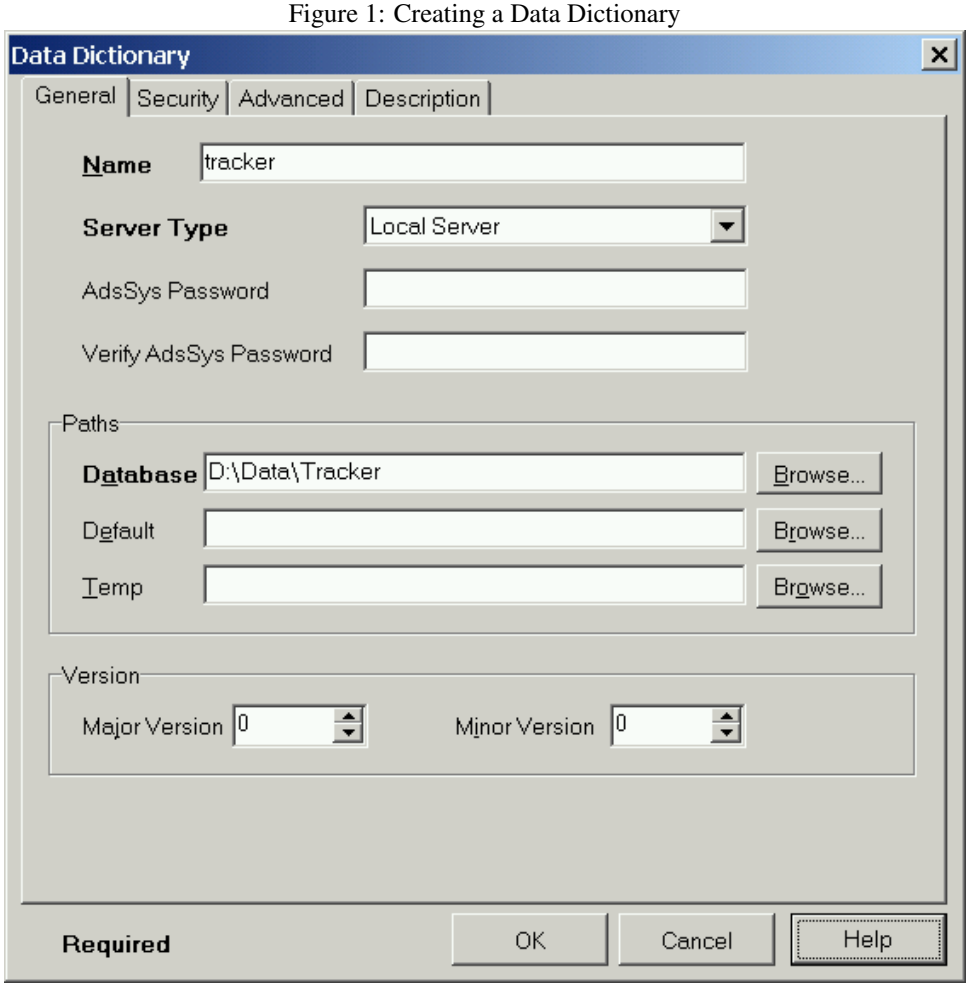
The windows version of Advantage Database Server comes with the Adavantage Database Architect. This is a GUI utility which allows to create and maintain a database. It contains an SQL window, which can be used to execute SQL statements.

First of all, a database must be created. The architect contains a wizard to create a new database (a "new connection"). The wizard will present 2 options: a database which contains a data dictionary, or a database with 'free tables'. The database with the Data dictionary allows for more control over the database, so this is what will be used. When this option is selected, then the data dictionary dialog is shown, as in figure 1 on page 3

3 pieces of information must be provided to create a new database:

Name This is the name of the database. A data dictionary file with this name will be created. (for the pupil tracking database, 'Tracker' will be used)

Figure 1: Creating a Data Dictionary



Server type This is the type of the server. For an embedded database, this should of course be 'Local server'

Database This is the directory where the database should be created.

More options can be set, but this is the minimally required information to be able to create a database. If the database contains sensitive data, it's a good idea also to supply a password for the default administrator user ('AdsSys'). The database itself and communication (in the case of a remote database) can be encrypted if stronger security is needed.

To populate the database with tables and fill the tables with data, the scripts which were used to create and populate the other embedded databases in this series of articles, had to be modified. The modified version of the database creation script is provided with this article.

The tracker database consists of 2 tables: a table with pupils, and a table with track data: the track data could come for example from some biometric device such as a fingerprint scanner or identity card reader, installed at the school gate.

The SQL statement which was used till now to create the PUPILS Table, was the following:

```
CREATE TABLE PUPIL (
  PU_ID INTEGER NOT NULL,
  PU_FIRSTNAME VARCHAR(50) NOT NULL,
  PU_LASTNAME VARCHAR(50) NOT NULL,
  CONSTRAINT PUPIL_PK PRIMARY KEY (PU_ID)
);
```

The Advantage Database SQL dialect deviates from the SQL standard in 2 ways:

1. `VARCHAR` is not recognized. It is accepted, but creates a BLOB (or memo) column. Instead, the `CICHAR` (Case Insensitive CHAR) keyword can be used to create a variable length string.
2. The `NOT NULL` constraint (indeed, any field-level constraint) must be preceded by the CONSTRAINT keyword.

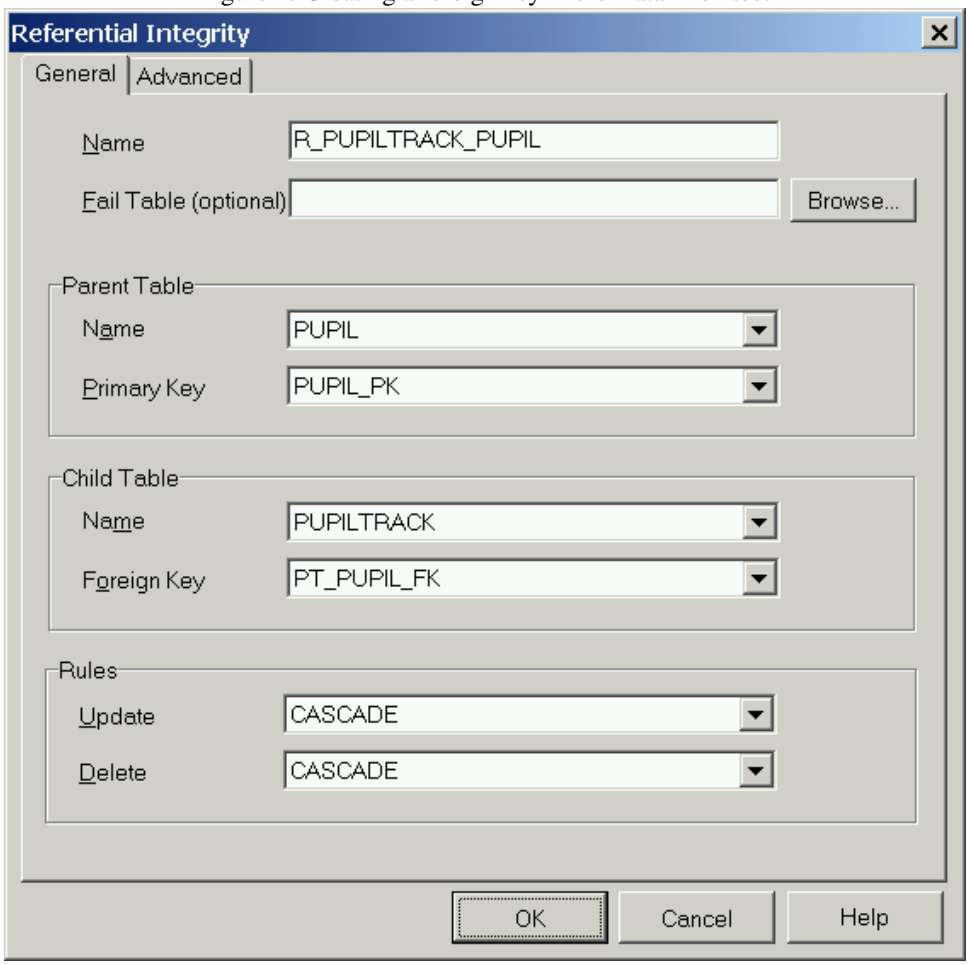
Applying these rules results in the following SQL statement:

```
CREATE TABLE PUPIL (
  PU_ID INTEGER CONSTRAINT NOT NULL,
  PU_FIRSTNAME CICHAR(50) CONSTRAINT NOT NULL,
  PU_LASTNAME CICHAR(50) CONSTRAINT NOT NULL,
  CONSTRAINT PUPIL_PK PRIMARY KEY (PU_ID)
);
```

The table which contains the track data was previously created with the following query:

```
CREATE TABLE PUPILTRACK (
  PT_ID INTEGER NOT NULL,
  PT_PUPIL_FK INTEGER NOT NULL,
  PT_CODE CHAR(1) NOT NULL,
  PT_DATE DATE NOT NULL,
  PT_TIME TIME NOT NULL,
  CONSTRAINT PUPILTRACK_PK PRIMARY KEY (PT_ID),
  CONSTRAINT R_PUPILTRACK_PUPIL FOREIGN KEY (PT_PUPIL_FK)
    REFERENCES PUPIL(PU_ID)
);
```

Figure 2: Creating a foreign key in the Data Architect



Needs an extra change: The Advantage Database SQL does not understand the `FOREIGN KEY` construct.

```
CREATE TABLE PUPILTRACK (
  PT_ID INTEGER CONSTRAINT NOT NULL,
  PT_PUPIL_FK INTEGER CONSTRAINT NOT NULL,
  PT_CODE CHAR(1) CONSTRAINT NOT NULL,
  PT_DATE DATE CONSTRAINT NOT NULL,
  PT_TIME TIME CONSTRAINT NOT NULL,
  CONSTRAINT PUPILTRACK_PK PRIMARY KEY (PT_ID)
);
```

The foreign key can be created, however, using the "RI Objects" item in the Database Architect program. It shows the figure 2 on page 5 dialog. The information to be entered there is obviously the same as the information needed by the FOREIGN KEY constraint statement in SQL.

The data architect can even generate a SQL statement which creates the foreign key in SQL, using a stored procedure:

```
EXECUTE PROCEDURE
  sp_CreateReferentialIntegrity (
```

```
'R_PUPILTRACK_PUPIL',
'PUPIL',
'PUPILTRACK',
'PT_PUPIL_FK',
2, 2, NULL, '', '');
```

Execution of this statement failed, however. The "RI links" form did create the referential integrity constraint.

After this, the indexes on the tables could be created using the same SQL statements as for the other

```
CREATE INDEX I_PUPILTRACK_DATE ON PUPILTRACK (PT_DATE);
CREATE INDEX I_PUPILLASTNAME ON PUPIL (PU_LASTNAME);
CREATE INDEX I_PUPILFIRSTNAME ON PUPIL (PU_FIRSTNAME);
```

Filling the database with data can be done with the Database Architect program, but is not recommended. Loading the script in the SQL window is possible: the SQL window can contain multiple SQL statements But the 610.000 SQL statements take a very long time to load: about as long as it takes to actually execute them. It also has a drastic impact on memory usage of the program, which went far over 512Mb.

Since the SQL window defaults to loading the last used SQL script when it is started, this means the script will be reloaded on the next startup, causing the architect to apparently 'hang'. Not a problem, but one should be prepared for this.

So, the SQL statements will be executed in the SQL executor program introduced in the previous articles in this series.

4 Delphi support: `TDataset` components

Advantage Database comes with delphi TDataset components which access the database directly through the Advantage API, making it probably the fastest and most efficient way to access the database. The components come in a set of Delphi packages, which must be compiled and installed in the Delphi IDE (Delphi 7.0 was used for this):

adsd70.dpk This package contains the run-time code of the components.

adsd70d.dpk This package contains the design-time code to register and manipulate the components

Several components are supplied in the Delphi package:

TAdsConnection This component represents the connection to the database. All other components are connected to such a connection. It's equivalent to `TDatabase` in the BDE, or a `TRemoteServer` for Midas applications.

TADSTable a `TDataset` descendent representing a table in the Advantage database. It's equivalent to `TTable` in the BDE. Using a single property, it offers access to any table in the database. Additional properties exist to select and maintain indexes, and to filter the data

TADSQuery A `TDataset` descendent which allows to execute any query on the database. In fact, multiple (non-select) SQL statements can be executed at once. It's therefore more powerful than it's BDE pendant, `TQuery`. It also offers support for parameters, and for master-detail relationships.

TADSStoredProc is a `TDataset` descendent offering access to stored procedures.

TAdsDictionary offers access to the Data Dictionary of a database. It can be used to control every aspect of the database, and to create new databases.

TAdsSettings can be used to control various settings of the ADS connection.

TAdsBatchMove can be used to migrate data from one table to another.

To create the browser program and the SQL executor program, the only objects that are needed are the `TAdsQuery` and `TAdsConnection` components.

The `TAdsConnection` component (call it `ACExecutor`) needs 4 properties:

ConnectPath This should point to the data dictionary file, created with the database.

AdsServerTypes The server types that should be supported by this connection. For an embedded database, this should be set to `[stADS_LOCAL]`

Username the username to be used for the connection. If a password was set for the database, the password property should also be set.

IsConnected is a boolean which can be set to `True` to connect to the database.

The `TADSQuery` component (call it `QExecutor`) is similar to the `TQuery` component in Delphi or `TSQLQuery` component in Lazarus. The most relevant properties are the `AdsConnection` property, which must point to an `TADSConnection` instance. The `SQL` property contains the SQL statement which must be executed. (multiple non-select statements can be specified, separated with semicolons)

Armed with these 2 components, the SQL executor program can be coded. The program structure is the same as for the other embedded databases. A connection component and query component are dropped on the main form, and the `StartDatabase` and `StopDatabase` methods can be implemented as follows:

```
procedure TMainForm.StartDatabase;

begin
  ACExecutor.Isconnected:=True;
  ACExecutor.BeginTransaction;
end;

procedure TMainForm.StopDatabase;

begin
  ACExecutor.Commit;
  ACExecutor.Isconnected:=False;
end;
```

As can be seen, a transaction is started with the `BeginTransaction` method of the connection component. The transaction is committed just before the database is closed again. Only a single transaction per connection is allowed.

The `ExecuteStatement` procedure is also straightforward to implement:

```
procedure TMainForm.ExecuteStatement(SQL : String; IgnoreError : Boolean);

begin
```

```
  Try
    QExecutor.SQL.Text:=SQL;
    QExecutor.ExecSQL;
  Except
    On E: Exception do
      begin
      MLog.Lines.add('Error executing statement: '+E.Message);
      If Not IgnoreError then
        Raise;
      end;
  end;
end;
```

5 Lazarus support: Porting ADS components to Lazarus

Since the DLL's containing the Advantage Database Server contain a simple (non COM) API which is called from the Advantage TDataset components, it should be possible to port the components to Free Pascal Lazarus.

To test this the Kylix source files were copied to a Linux platform, the .so libraries were installed in `/usr/lib`, and then the source files were imported in a lazarus package (lazads.lpk) Compilation was rather smooth: A new define was introduces: `ADSFPC`, to indicate that the sources were compiled with FPC/Lazarus.

Using this define at certain places in an `$IFDEF` construct was sufficient to compile all code under lazarus. The design-time forms were removed from the package. The package was installed in lazarus, and the tab `Advantage` appeared in the component palette.

To test the components, the browser application was implemented. To this end, a `TADSConnection` (`ACBrowser`) and 2 `TADSQuery` components (`QPupils`, `QTrackData`) were dropped on the form. Hooking up the datasource components to these datasets is done as usual. The `QPupils`' `SQL` property is set to

```
SELECT * FROM PUPIL
```

The SQL statement for `QTrackData` is set to

```
SELECT * FROM PUPILTRACK
WHERE (PT_PUPIL_FK=:PU_ID)
```

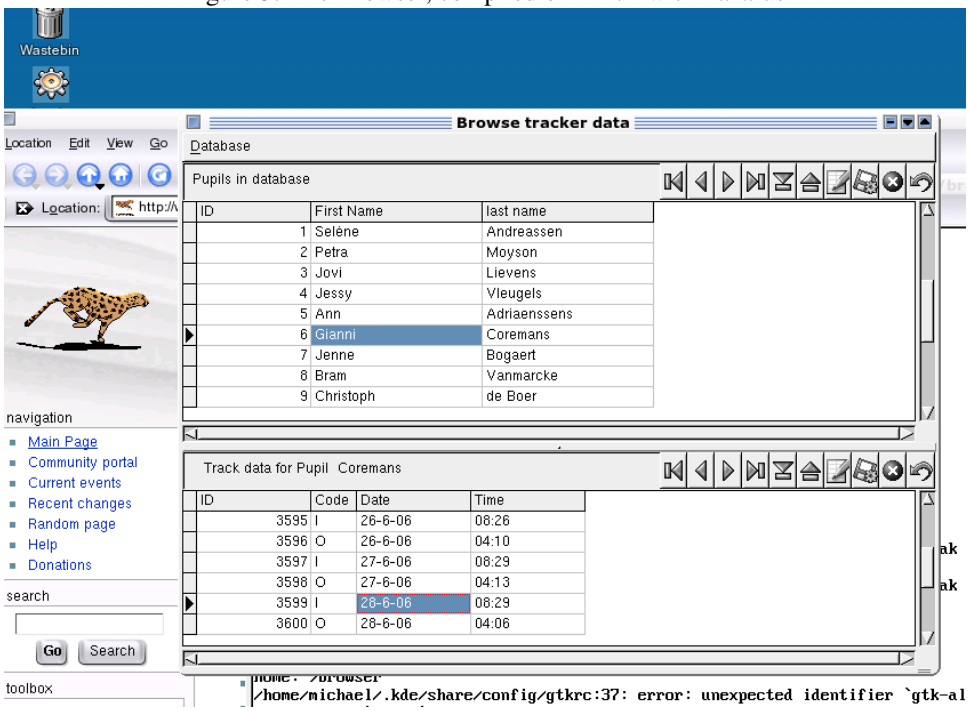
Setting the `Datasource` property to `DSPupils` establishes a master-detail relationship between the list of pupils and the tracking data for one pupil.

The open and close methods for the database are similar to the ones in the SQL executor:

```
procedure TMainForm.CloseDatabase;
begin
  QTrackData.Close;
  QPupils.Close;
  ACBrowser.IsConnected:=False;
end;

procedure TMainForm.OpenDatabase;
begin
  ACBrowser.IsConnected:=True;
```

Figure 3: The Browser, compiled on Linux with Lazarus




```
  QPupils.Open;
  QTrackData.Open;
end;

function TMainForm.DatabaseOpen: Boolean;
begin
  Result:=ACBrowser.IsConnected;
end;
```

The result can be seen in figure 3 on page 9. For the simple case of a data browser, it would have been possible to use `TAdsTable` components: they can also be used to establish master-detail relationships, in case both master and detail fields

6 Performance

The performance of the Advantage Database server has been compared with the performance of the other databases. 4 tests were run:

1. Inserting all data in the table (601.000 insert queries).
2. Retrieve the number of tracking items per pupil:

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

3. Number of tracking entries on 6 september 2005, before 8:28 AM.

```
SELECT
  COUNT(PT_ID)
FROM
  PUPILTRACK
WHERE
  (PT_DATE='2005-09-06')
  AND (PT_CODE='I')
  AND (PT_TIME<='08:28:00');
```

4. Number of different pupils entering school on 6 september 2005, before 8:28 AM.

```
SELECT
  COUNT(PU_ID)
FROM
  PUPIL LEFT JOIN PUPILTRACK ON (PT_PUPIL_FK=PU_ID)
WHERE
  (PT_DATE='2005-09-06')
  AND (PT_CODE='I')
  AND (PT_TIME<='08:28:00');
```

The results can be compared in the following table:

Test	SQLite	Firebird	MySQL	ADS
1	0:0:42.00	0:1:29.47	0:0:40.48	0:4:27.00
2	0:6:12.75	0:0:02.58	0:0:02.40	0:0:6.54
3	0:0:00.67	0:0:00.44	0:0:02.43	0:0:0.63
4	0:5:59.38	0:0:00.48	0:0:02.44	0:0:7.18

The results are not quite as fast as the other engines (although the more complicated ones beat SQLite by far, but are very good nevertheless.

Two remarks are in order:

1. If the database population script is run twice on the same database, the second run has very poor performance: All records are deleted from the tables before they are filled. But to achieve maximum performance, it is necessary to compact the table after the records have been deleted. This can be done using the `AdsPackTable` API call, but cannot be done in SQL.

2. The last test can be speeded up significantly by changing the query to:

   ```
   SELECT
     COUNT(PU_ID)
   FROM
     PUPIL, PUPILTRACK
   WHERE
     (PT_PUPIL_FK=PU_ID)
     AND (PT_DATE='2005-09-06')
     AND (PT_CODE='I')
     AND (PT_TIME<='08:28:00');
   ```

   Doing this reduces the execution time to 0.76 seconds, which suggests that the query optimizer may need some help by tuning some queries by hand.

7 Conclusion

The Local Advantage Database server is definitely worth a look if an embedded database is needed for an application. It offers performance, low memory footprint (provided some elementary precautions are taken), a rich feature set, it can be connected to from various environments (including Lazarus) and is scalable to full client-server network connectivity - although a price must be paid for that: it is not (yet) open source. If this is not an issue: Advantage Database Server is definitely a good bet.

The author wishes to thank Joachim Dürr, for the kind assistance he offered to solve some set-up problems.