lazarus and dabatases

Michaël Van Canneyt

September 30, 2012

Abstract

In previous articles, a motion detection system was created that sends a picture to a mail address. In this article, it will be shown how to store the picture in a database as well. Standard Lazarus components will be used for this. A program to view the stored images will be created as well.

1 Introduction

Often, an application must connect to a database: One or more tables with tabular data in them, possibly made accessible with an SQL language.

The concept of a set of records fetched from a database (in whatever form) is encapsulated in a single class in the Free Component Library: `TDataset`. This class embodies the idea of tabular data: rows are represented as records. The columns in a row are fields. When fetching records from a SQL database, the set of records is also represented by `TDataset`. The class offers methods for editing the fetched data, and posting the changes back to the database.

This abstract class serves as the parent for many classes that fetch data from a variety of databases. Lazarus has support for databases through many freely or commercially available component sets: Zeos works with lazarus. AnyDAC also works with Lazarus, as well as Advantage Database.

Lazarus also comes with a default database access technology, called SQLDB. As the name indicates, it is focused on SQL databases. Out of the box, SQLDB has support for several popular databases: Firebird, MySQL, Postgres, Oracle, MS SQL server, SQLite. The ODBC connector allows to connect to any database for which an ODBC driver is available (for instance MS-Access).

Other than offering components that connect to a database, it has also DB-Aware controls: these are controls that can display data from a dataset. If the control allows some form of editing, the changes can be posted back to the dataset. All this can be accomplished without writing a single line of code.

This technology will be demonstrated by enhancing the motion detection program, so it stores a picture in a database. A small separate program will be constructed to view the stored pictures.

2 SQLDB archtitecture

A program that makes use of SQLDB always uses 3 components:

A TSQLConnection descendent is always needed: it represents the connection to the database.

A different connection component must be used for each database type: for Firebird this component is called `TIBConnection`. For PostGres the component is called `TPQConnection`.

As descendents `TSQLConnection`, these components inherit the following properties:

HostName The name of the machine the database is on.

DatabaseName The name of the database to connect to.

Username The username to connect to the database with.

Password The password that goes with the user.

Connected If set to `True`, a connection will be established with the SQL server. Setting it to `False` will disconnect from the server.

After all these properties are set up properly, the `Connected` property can be set to `True`. For database types that support this (many do), the connection component can also be used to create a new database using the `CreateDB` method. The connection component also has some methods to retrieve metadata from the database (lists of tables, fields etc.).

To fetch data from the SQL server, a `TDataset` descendent called `TSQLQuery` must be used. It sends a SQL statement to the SQL server and retrieves the result, if any. It has the following main properties:

SQL a stringlist containing the command that must be sent to the SQL server. Only a single SQL command may be entered.

Database The TSQLConnection descendent on which to execute the SQL command.

Params An SQL command can be parametrized. The `Params` property is a collection of named values that will be subsitituted for the parameters. If the SQL server has native support for parametrized queries, this support will be used.

Transaction the transaction in which the SQL statement must be executed.

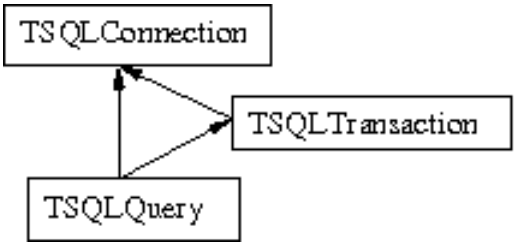
Active if set to `True`, the SQL command will be sent to the server, and the result will be fetched. Calling the `Open` method has the same effect. The `Close` method will discard the results, if any, from memory.

If no data is expected from the SQL command (e.g. for an INSERT or CREATE statement) then the `ExecSQL` method must be used to execute the statement. Note that SQLDB does not attempt to offer a uniform SQL syntax which is valid for all database connection types: the SQL as it was typed will be sent to the SQL engine. That means that, while it is possible to simply switch connection components to select another database type, the SQL must be checked to see if the syntax is accepted by the new database type.

The third component that is needed is a `TSQLTransaction` component. SQLDB has a powerful transaction mechanism: multiple transactions on a single database can be handled simultaneously in a single application. If the underlying engine cannot handle this natively, it is emulated by opening multiple simultaneous connections. The transaction is controlled through the `StartTransaction`, `Commit` and `RollBack` methods, or the `Active` property.

The relation between the 3 components is depicted in figure 1 on page 3

Figure 1: The relation between the 3 components used in SQLDB



3 Saving pictures in a database

The motion detection program that was developed in the previous articles of this series, sends a picture whenever motion was detected. Armed with the SQLDB components, the program can be enhanced to store the picture in a database as well. Firebird will be used, but doing the same on another database just requires changing the connection component to the appropriate descendent of `TSQLConnection`

The database will contain a single table called `MOTION`, with 2 fields:

M_TIMESTAMP The timestamp when motion was detected.

M_IMAGE A blob field containing the picture.

Before anything can be stored in a database, a database must be created. The `TIBConnection` component has the capacity to create a database using the `CreateDB` method. To add this functionality to our program, we drop a `TIBConnection` component (`DBIB`), a transaction component (`TRDB`), and 2 query components: `QCreateTable` and `QCreateIndex`. We connect the transaction and query components to the database component. The SQL properties of the 2 SQL components are set to:

```
CREATE TABLE MOTION (
  M_TIMESTAMP TIMESTAMP,
  M_IMAGE BLOB
);
```

and

```
CREATE INDEX I_TIMESTAMP ON MOTION(M_TIMESTAMP);
```

Then, a `TFileNameEdit` control (`FEDB`) and a button `BCreateDB` are placed on the main form of the motion detection program. The `OnClick` event handler of the button executes the following code:

```
procedure TMainForm.BCreateDBClick(Sender: TObject);
begin
  CreateDatabase(FEDB.FileName);
end;
```

The `CreateDatabase` method takes care of actually creating the database if necessary. After the database was created, it asks for a list of tables using the `GetTableNames` method:

```
procedure TMainForm.CreateDatabase(Const AFileName : String);

Var
  L : TStrings;

begin
  IBDB.DatabaseName:=AFileName;
  If Not FileExists(AFileName) then
    IBDB.CreateDB;
  IBDB.Connected:=True;
  L:=TStringList.Create;
  try
    IBDB.GetTableNames(L);
    if L.IndexOf('MOTION')=-1 then
      CreateMotionTable;
  finally
    L.Free;
  end;
end;
```

If the motion table does not yet exist, it is created by executing the 2 SQL statements in `QCreateTable` and `QCreateIndex` in the following method:

```
procedure TMainForm.CreateMotionTable;

begin
  QCreateTable.ExecSQL;
  QCreateIndex.ExecSQL;
  TRDB.Commit;
end;
```

The last statement commits the transaction and thus makes sure the table defintions are committed to the database.

When all this is done, the database to store motion detection pictures can be created. Storing the pictures in a database is controlled by a checkbox `CBSTorePicture`. When checked, the program will store the pictures it sends by email in the database as well. The timer event used to detect motion, must be adapted for this:

```
procedure TMainForm.TMotionTimer(Sender: TObject);

begin
  Inc(FTicks);
  SaveTempFrame;
  if CheckDifferent then
    begin
    If MinutesBetween(Now,FLastSend)>1 then
      begin
      FLastSend:=Now;
      SendPicture;
      If CBStorePicture.Checked then
        InsertPicture;
      end;
    end;
end;
```

The InsertPicture method will save the temporary frame in the database using a TSQLQuery component called QInsertImage. Its SQL property is set to:

```
INSERT INTO MOTION VALUES ('NOW',:IMAGE)
```

The colon (:) in :IMAGE indicates to SQLDB that IMAGE is a parameter, whose value must be set from the Params property of QInsertImage. Parameters can appear wherever the SQL statement expects a single value. The InsertPicture will first check whether a connection to the database must be established, and then sets the value of the parameter prior to executing the SQL Statement using the ExecSQL method:

```
procedure TMainForm.InsertPicture;

Var
  P : TParam;

begin
  // Connect to database if needed.
  if not IBDB.Connected then
    begin
    IBDB.DatabaseName:=FEDB.FileName;
    IBDB.Connected:=True;
    end;
  P:=QInsertImage.ParamByName('IMAGE');
  P.LoadFromFile(FFrameFile,ftBlob);
  // start transaction if needed.
  If Not TRDB.Active then
    TRDB.StartTransaction;
  QInsertImage.ExecSQL;
  // Commit data.
  TRDB.Commit;
end;
```

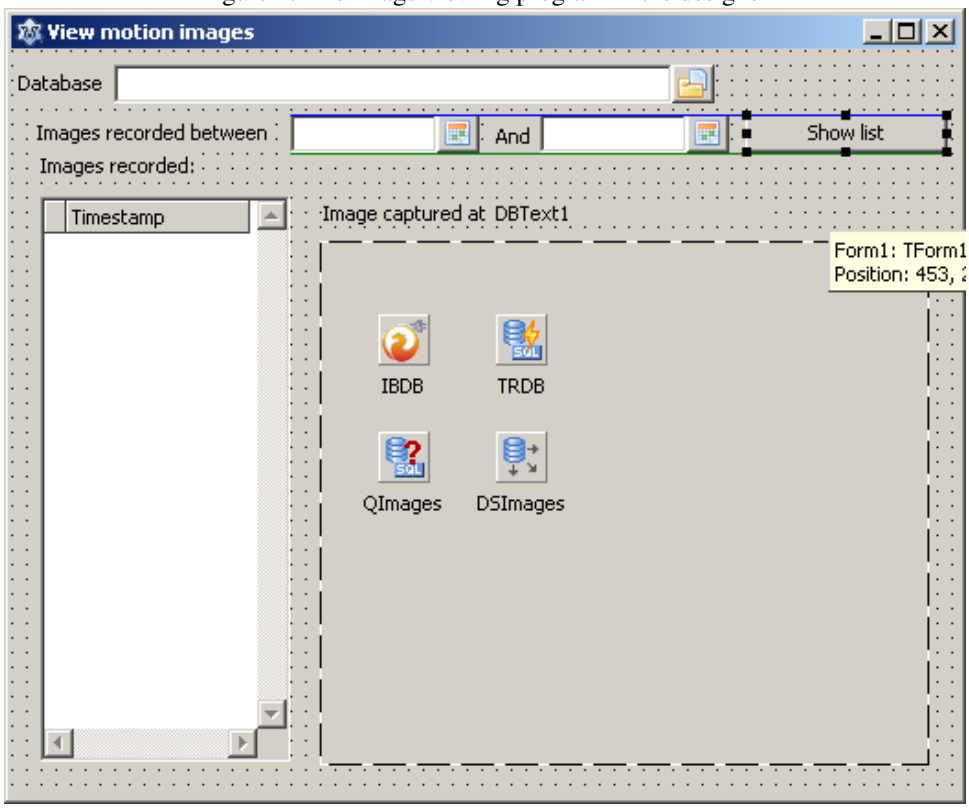
The ParamByName function looks in the parameter collection for the parameter with the appropriate name, and returns it. The LoadFromFile method of TParam will load the parameter value from a file, and sets the type of parameter to ftBlob: each parameter has a type, so the SQLDB engine knows how to pass the parameter value to the database engine.

4 Fetching data from a database

Now that the program to detect motion can store the images in a database, a mechanism is needed to view the stored pictures. This can also easily be done in Lazarus: It offers components that cooperate with SQLDB (or any other dataset technology that descends from TDataset) and that allow the data in a TSQLQuery to be displayed and edited. These components are so-called DB-Aware versions of the regular controls: These controls are descendents of TEdit, TCheckBox, TLabel, which can be bound to a dataset (through their Datasource property) and a particular field in the dataset (using the DataField property). When the dataset is opened, these controls will display the contents of current record in the dataset; Each will display the field they are bound to.

There are also some specialized controls, such as the TDBNavigator, which can be used to navigate through the data, and TDBGrid, which will display the data of a TDataset in a tabular manner.

Figure 2: The image viewing program in the designer



The `DataSource` property of all these controls points to a `TDatasource` component. This component is connected to the `TDataset` descendent, and acts as a mediator between the dataset and the various controls: it passes notifications from the dataset to the controls in case something changed (current record was changed, a field changed value) or allows the controls to notify the dataset of changes, for instance when the user edited something.

To create a program that views the pictures captured by the motion detection program is quite simple: A `TFileName` edit is needed to specify the location of the database. 2 `TDateEdit` controls are added to enter a period of time (dates): Only images captured between these 2 moments will be shown. Then, a grid is added to display the various timestamps for which an image is available, and a `TImage` control to show the image. A label to show the timestamp corresponding to the currently showing image is also added. All this is shown in figure 2 on page 6.

Naturally, a `TIBConnection` component as well as a `TSQLTransaction` and `TSQLQuery` component (named `QImages`) will be needed to fetch the data. The data is fetched with the following SQL command:

```
SELECT
  M_TIMESTAMP,
  M_IMAGE
FROM
  MOTION
WHERE
  (M_TIMESTAMP BETWEEN :Start and :Stop)
```

When the 'Show List' button is clicked, the parameters are copied to the query and the

query is opened:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ShowImagesBetween(DEStart.Date,DEStop.Date);
end;

procedure TForm1.ShowImagesBetween(AStart,AStop : TDateTime);

begin
  if not IBDB.Connected then
    begin
    IBDB.DatabaseName:=FEDB.FileName;
    IBDB.Connected:=true;
    end;
  With QImages do
    begin
    Close;
    ParamByName('START').AsDateTime:=AStart;
    ParamByName('STOP').AsDateTime:=AStop;
    Open;
    end;
end;
```

The only thing done by the code is connecting to the database if no connection was present, and set the parameters. Note the `AsDateTime` used to pass the parameter values: Using this form tells SQLDB that the parameters must be passed on to the SQL engine as timestamp values.

The following code

```
  With QImages do
    begin
    Close;
    ParamByName('START').AsString:=DateToStr(AStart);
    ParamByName('STOP').AsString:=DateToStr(AStop);
    Open;
    end;
end;
```

Would pass the parameter values as string values to the SQL engine, leaving it to the engine to figure out how to convert the string values to timestamps. This is in general a bad idea, since each engine handles date/time localization issues differently. Passing the values explicitly as timestamps makes sure the database engine gets a correct value.

Notice that the transaction is not activated: the TSQLQuery component will do that if necessary.

After the `Open` method is called, the data is fetched from the server. The grid is set up to display only the value of the `M_TIMESTAMP` field in the data. As a result, it will show a list of times. `TDataset` and `TSQLQuery` have a concept of `Cursor`: this is the current record of the dataset. The cursor can be moved from one record to the next using the `Prev` and `Next` methods of `TDataset`. When the end of the data has been reached, the `EOF` property will return `True`. When the beginning has been reached, the `BOF` property will be `True`. When the dataset is empty, both `BOF` and `EOF` will be `True`.

A typical way to perform some actions on all records in a dataset will then look like:

```
With SomeQuery do
  While not EOF do
    begin
    // Do something with the current record
    Next;
    end;
```

Clicking a row in the grid will move the dataset cursor to the record that corresponds to the clicked row: the current record will change, and the label showing the timestam (LTimeStamp) will display the value for the current record. It is posible to react to such events in the dataset: it has a lot of event handlers (AfterOpen, AfterScroll, AfterEdit) that allow the programmer to respond to all these events and take whatever action is needed.

One of these events is `AfterScroll`. It is called whenever the dataset cursor location changes. This event can be used to show the current in a `TImage` control (called `IMotion`):

```
procedure TForm1.QImagesAfterScroll(DataSet: TDataSet);

Var
  M : TMemoryStream;
  F : TBlobField;

begin
  M:=TMemoryStream.Create;
  try
    F:=(QImages.FieldByName('M_IMAGE') as TBlobField);
    F.SaveToStream(M);
    M.Position:=0;
    IMotion.Picture.LoadFromStream(M);
  finally
    M.free;
  end;
end;
```

The code is quite simple: the blob field containing the image (`M_IMAGE`) is retrieved, and its contents is saved to a stream (`M`). The picture is then loaded from this stream.

The code demonstrates an important aspect of `TDataset`: when the query is opened, an array (or list) of `TField` instances is constructed. Each element in the array represents a column in the result data, and provides access to that column's data in the current record. The `TField` instances are not all of the same class: for integer columns, a `TIntegerField` instance is created. For timestamp columns a `TDateTimeField` is created. For Blob fields, a `TBlobField` instance is used: this class has a method to save the content of the blob field to a stream (and one to load it from a stream).

The result of all this can be seen in action in figure 3 on page 9.

5 conclusion

It is easy to access data in SQL-based databases using Lazarus: various mechanisms exist to extract and manipulate the data exist. In this article, the basics of SQLDB have been explained. SQLDB is shipped standard with Lazarus, and can therefor easily be used for small (or even large) projects.

Figure 3: The image viewing program in action