lazarus en databanken

Michaël Van Canneyt

October 13, 2012

Abstract

In vorige bijdrages werd een anti-inbraak systeem ontwikkeld dat een beeldje per e-mail verstuurde. In dit artikel wordt getoond hoe het opgenomen beeldje ook in een databank bewaard kan worden. Hiervoor worden standaard Lazarus componenten gebruikt, en met dezelfde componenten wordt een programmaatje gemaakt waarmee de beeldjes bekeken kunnen worden.

1 Inleiding

Een programma moet vaker wel dan niet een verbinding maken met een databank: een of meerdere tabellen met gegevens in rijen en kolommen, vaak aan te spreken door middel van SQL.

Het idee van een verzameling records uit een databank (in wat voor vorm dan ook) is gevat in een enkele klasse in de FCL (Free Component Library): `TDataset`. Deze klasse belichaamt het idee van data in tabelvorm: Rijen worden voorgesteld door records, kolommen zijn velden. Wanneer gegevens van een SQL databank worden opgehaald, wordt de verzameling records ook voorgesteld door `TDataset`. Deze klasse levert methodes voor het bewerken van de opgehaalde data, en zorgt ervoor dat wijzigingen terug worden weggeschreven in de databank.

Deze abstracte klasse dient als parent voor vele klasses die gegevens uit een grote verscheidenheid aan databanken kunnen ophalen. Lazarus ondersteunt een hoop databanken door vele vrije of commercieel beschikbare componenten: Zeos, AnyDAC en Advantage database kunnen allemaal met Lazarus gebruikt worden.

Lazarus heeft ook een eigen databank technologie die standaard wordt mee geinstalleerd: SQLDB. Zoals de naam het al aangeeft, ligt de focus op SQL Databanken. SQLDB verschaft standaard toegang tot verschillende populaire databanken: Firebird, MySQL, Postgres, Oracle, MS SQL server, SQLite. De ODBC connector staat toe alle databanken voor dewelke een ODBC driver beschikbaar is, aan te spreken (bijvoorbeeld MS-Access).

Buiten de componenten die een connectie met een databank leggen, zijn er ook de data-aware controls: GUI elementen die gegevens uit een dataset tonen en – indien het element het toestaat – bewerken: de veranderingen worden dan in de dataset bewaard. Dit alles kan gebeuren zonder dat er ook maar 1 lijn code geschreven wordt.

De mogelijkheden van SQLDB worden gedemonstreerd door het inbraak-detectie programma zo aan te passen dat het opgenomen beeldje in een databank wordt opgeslagen. Een apart programmaatje dat de beeldjes toont, vervolledigt de demonstratie.

2 SQLDB archtitecture

Een programma dat SQLDB gebruikt om een databank aan te spreken, gebruikt altijd 3 componenten.

Een TSQLConnection afgeleide is altijd nodig: dit stelt de verbinding met de databank voor. Voor elk databank type bestaat er een andere connectie component: Voor firebird bijvoorbeeld moet `TIBConnection` gebruikt worden. Voor PostGres moet `TPQConnection` gebruikt worden.

Beide zijn afgeleiden van `TSQLConnection`, en als zodanig beschikken ze over de volgende properties:

HostName De naam van de machine waarop de databank staat.

DatabaseName De naam van de databank waarmee verbinding gelegd moet worden.

Username De gebruikersnaam waarmee aangemeld moet worden.

Password Het wachtwoord van de gebruiker.

Connected Als dit op `True`, gezet wordt, wordt de verbinding met de SQL server gemaakt. Deze property terug op `False` zetten, verbreekt de verbinding.

Als de properties correct ingesteld zijn kan de `Connected` property op `True` gezet worden. Voor databank types die dit ondersteunen (voor de open source databanken is dit zo) kan de connectie component ook gebruikt worden om een nieuwe databank te maken dmv. de `CreateDB` methode. De connectie component heeft ook enkele methodes om wat metadata uit de databank te halen (een lijst van tabellen, velden in een tabel enz.).

Om data van een SQL server op te halen is een `TSQLQuery` component nodig. Dit is een `TDataset` descendent. De component zendt een SQL commando naar de SQL server, en haalt het resultaat op als het commando een resultaat oplevert. De volgende properties zijn gedefinieerd in `TSQLQuery`:

SQL is een stringlist met het commando dat naar de SQL server gestuurd wordt. Slechts een enkel SQL commando mag opgegeven worden.

Database Verwijst naar de `TSQLConnection` connectie component. Op deze SQL server wordt het commando uitgevoerd.

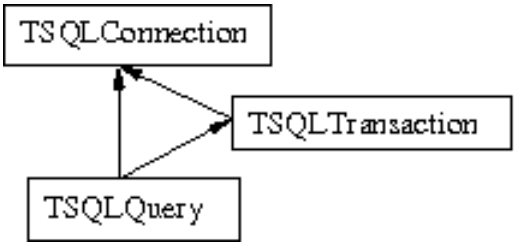
Params Het SQL commando kan parameters bevatten: De `Params` property is een collectie van waardes die op basis van hun naam worden ingevuld in de parameters. Als de SQL server het gebruik van geparametriseerde queries ondersteunt, wordt deze ondersteuning gebruikt.

Transaction De transactie in dewelke het SQL commando uitgevoerd wordt.

Active Als deze property op `True` gezet wordt, wordt het SQL commando doorgegeven aan de SQL server. Als er resultaatgegevens zijn, worden deze opgehaald. De `Open` methode oproepen heeft hetzelfde resultaat. De `Close` methode zal de resultaten terug uit het geheugen wissen .

Als het SQL commando geen resultaat oplevert (bv. bij een INSERT of CREATE commando), dan moet de `ExecSQL` methode gebruikt worden om het commando uit te voeren. Let wel op: SQLDB probeert niet een uniforme SQL syntax aan te bieden, geldig voor alle databank types: het SQL commando zal naar de SQL server doorgestuurd worden zoals het werd ingegeven. Dit betekent dat, hoewel het mogelijk is van databank type te veranderen

Figure 1: De relatie tussen de 3 basis componenten in SQLDB



door gewoon een andere TSQLconnection component te gebruiken, de SQL commandos nagekeken moeten worden om te zien of de syntax wordt aanvaard door de nieuwe SQL server.

De derde component die nodig is bij SQLDB is de `TSQLTransaction` component. SQLDB heeft een krachtig transaction mechanisme: meerdere transacties op eenzelfde databank kunnen tegelijkertijd afgehandeld worden binnen hetzelfde proces. Als de databank server API dit niet standaard ondersteunt, wordt dit nagebootst door verschillende gelijktijdige connecties te gebruiken. De transactie wordt gestuurd door de `StartTransaction`, `Commit` en `RollBack` methodes, of door gebruik te maken van de `Active` property.

De relatie tussen de 3 componenten wordt getoond in figure 1 on page 3.

3 Beeldjes bewaren in een databank

Het inbraak detectie programma dat ontwikkeld werd in de vorige artikels, zendt een beeldje per email zodra beweging ontdekt wordt. Gewapend met de SQLDB componenten kan het programma verbeterd worden zodat het beeldje ook opgeslagen wordt in een databank. Als databank gebruiken we Firebird, maar dezelfde code kan ook gebruikt worden om een andere databank aan te spreken: alleen de `TSQLConnection` component moet dan gewijzigd worden.

De databank bestaat uit een enkele tabel, `MOTION`, met 2 velden:

M_TIMESTAMP Het tijdstip waarop de beweging ontdekt werd.

M_IMAGE een blob veld met het beeldje.

Alvorens we iets kunnen opslaan in de databank, moet de databank aangemaakt worden. De `TIBConnection` component kan een databank aanmaken met de `CreateDB` methode. Om deze functionaliteit aan het programma toe te voegen, worden er dus een `TIBConnection` component (`DBIB`), een transactie component (`TRDB`), en 2 query componenten: `QCreateTable` and `QCreateIndex` op het formulier geplaatst. De transactie en query componenten worden met de databank connectie component verbonden, en de SQL properties van de 2 SQL componenten worden ingesteld op:

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

en

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

Daarna wordt een `TFileNameEdit` control (`FEDB`) en een knop `BCreateDB` op het hoofdformulier van de applicatie geplaatst. De `OnClick` event handler van de knop voert de volgende code uit:

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

De `CreateDatabase` methode zorgt er voor dat de databank aangemaakt wordt indien ze nog niet bestaat. Merk op dat het databank bestand een lokaal bestand moet zijn, anders mislukt de test.

Als de databank aangemaakt is, of de verbinding is gelegd, wordt een lijst van tabellen opgevraagd met de `GetTableNames` methode:

```
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;
```

Als de motion tabel niet bestaat, wordt ze aangemaakt d.m.v. de 2 SQL commandos in `QCreateTable` en `QCreateIndex` in de volgende methode:

```
procedure TMainForm.CreateMotionTable;

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

Het laatste commando zorgt voor een commit, zodat de tabel definities zeker in de databank bewaard zijn.

Als dit alles achter de rug is, kan de databank om beeldjes in op te slaan, aangemaakt worden. Het opslaan van de beeldjes in de databank wordt gestuurd door een vinkje (checkbox `CBSTorePicture`). Als het vinkje op staat zal het programma de beeldjes die het doorstuurt per mail ook opslaan in de databank. Het timer event dat gebruikt wordt om de beweging te detecteren wordt hiervoor als volgt aangepast:

```
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;
```

De `InsertPicture` methode zal het tijdelijke beeldje in de databank bewaren met een `TSQLQuery` component genaamd `QInsertImage`, waarvan de `SQL` property de volgende waarde bevat:

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

Door het dubbele punt (:) in `:IMAGE` weet de SQLDB code dat `IMAGE` een parameter is, en dat de waarde voor deze parameter gezocht moet worden in de `Params` property van `QInsertImage`. Parameters kunnen gebruikt worden op elke plek waar een SQL commando een waarde verwacht. Het kan dus niet gebruikt worden om bv. de naam van een tabel te parametriseren.

De `InsertPicture` methode zal eerst nagaan of de verbinding met de databank geactiveerd moet worden, en zal dan de waarde van de parameter toekennen alvorens het SQL commando uit te voeren met de `ExecSQL` methode:

```
procedure TMainForm.InsertPicture;

Var
  P : TParam;

begin
  // Met databank verbinden indien nodig.
  if not IBDB.Connected then
    begin
    IBDB.DatabaseName:=FEDB.FileName;
    IBDB.Connected:=True;
    end;
  P:=QInsertImage.ParamByName('IMAGE');
  P.LoadFromFile(FFrameFile,ftBlob);
  // Transactie starten indien nodig.
  If Not TRDB.Active then
    TRDB.StartTransaction;
  QInsertImage.ExecSQL;
  // gegevens Committen.
  TRDB.Commit;
end;
```

De `ParamByName` functie zoekt in de parameter collectie naar de parameter met de opgegeven naam, en geeft het gevonden item terug. De `LoadFromFile` methode van `TParam` zal de parameter waarde inlezen van een bestand, en stelt het type van de parameter in op `ftBlob`: elke parameter heeft een type, zodat de SQLDB code weet hoe de parameter waarde aan de databank moet doorgegeven worden.

4 Gegevens lezen uit een databank

Nu het inbraak detectie programma beeldjes kan opslaan in de databank, kan een tweede programma gemaakt worden dat de opgeslagen beeldjes bekijkt. Dit kan ook eenvoudig gemaakt worden in Lazarus: er zijn componenten beschikbaar die samenwerken met SQLDB (of elke andere data technologie die gebaseerd is op TDataset) en die toestaan de data in een `TSQLQuery` te tonen en bewerken.

Dit zijn de zogeheten DB-Aware versies van de standaard GUI elementen: Afgeleiden van `TEdit`, `TCheckBox`, `TLabel`, die met een dataset verbonden kunnen worden (d.m.v. hun datasource property) en een specifiek veld uit die dataset (met de `DataField` property). Als de dataset geopend wordt, zullen deze GUI elementen de inhoud van het huidige record in de dataset tonen. Elke control zal de inhoud van het veld tonen waarmee ze verbonden zijn.

Er zijn ook enkele gespecialiseerde controls, bv. de `TDBNavigator`, die kan gebruikt worden om door de data te navigeren, en `TDBGrid`, die de data van een `TDataset` in een rooster weergeeft.

De `DataSource` property van al deze componenten verwijst naar een `TDatasource` component. Deze component is op zijn beurt gekoppeld aan de `TDataset` afgeleide, en werkt als een tussenschakel tussen de dataset en de verschillende GUI elementen: berichten worden van de dataset aan de controls doorgegeven ingeval er iets veranderd is (er is van rij veranderd, of een veldwaarde in het huidige record is gewijzigd), of ingeval een gebruiker dmv. een GUI element een een veldwaarde wijzigt, wordt deze wijziging doorgegeven aan de dataset.

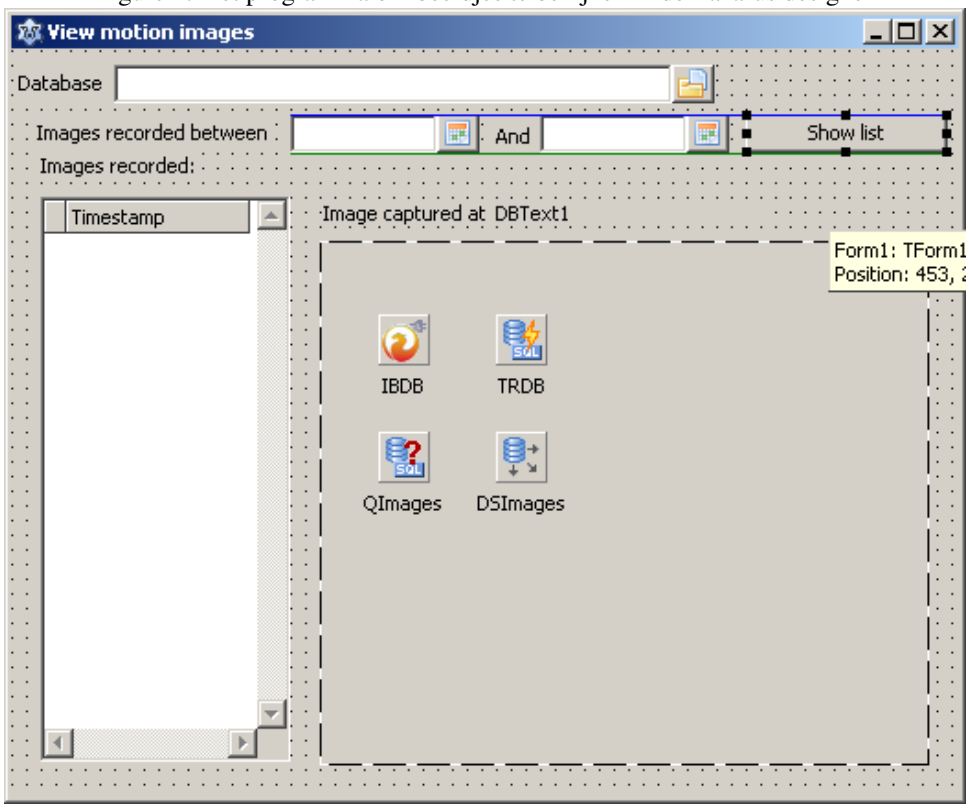
Een programma maken dat de beeldjes toont die door het inbraak detectie programma in de databank werden opgeslagen, is vrij eenvoudig: Een `TFileName` edit control is nodig om de locatie van de databank op te geven. Om een tijdspanne op te geven worden 2 `TDateEdit` controls gebruikt: alleen beeldjes die opgeslagen werden tussen deze 2 datums zullen worden getoond. De tijdstippen waarop een beeldje werd bewaard, worden getoond in een grid, en een `TImage` control is nodig om het beeldje te tonen (de data grid kan op zichzelf geen beeldjes tonen). Het tijdstip waarop het getoonde beelde werd bewaard, wordt in een TDBText (een label) getoond. Het formulier ziet er uit als in figure 2 on page 7.

Natuurlijk is er een `TIBConnection` component nodig, evenals een `TSQLTransaction`. Een `TSQLQuery` component (we noemen deze `QImages`) zal de gegevens ophalen met het volgende SQL commando:

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

Een klik op de 'Toon lijst' knop zorgt ervoor dat de parameters van de query worden inge-

Figure 2: Het programma om beeldjes te bekijken in de Lazarus designer



vuld met de waardes in de datum controls, waarna de dataset geopend wordt:

```
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;
```

Het enige ding dat de code nog extra doet is connecteren met de databank, indien de connectie nogniet gemaakt was. De parameters worden ook ingesteld: Doordat de `asDateTime`

property gebruikt wordt, worden 2 stukjes informatie doorgegeven aan SQLDB: Eerst en vooral de eigenlijke waarde voor de parameter, maar ook het feit dat het om een datum waarde gaat. Dit stelt SQLDB in staat de waarde correct door te spelen aan de SQL server.

Zo zou de volgende code:

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

de waardes doorspelen als tekenreeksen (strings), en moet de Databank server de string waardes omzetten naar geldige datum waardes om de voorwaarde te evalueren: Dit is in het algemeen een slechte werkwijze, want de SQL server kan dan wel of niet een andere datumnotatie hanteren als het programma - dit hangt af van het type databank server. Door de waardes expliciet als datum waardes aan te geven kan SQLDB de waardes in een voor de SQL server correcte vorm doorgeven.

Merk op dat de transactie niet geactiveerd wordt: De `TSQLQuery` component zal dit zelf doen indien nodig.

Nadat de `Open` method opgeroepen is, wordt de data van de server gehaald. De grid is geconfigureerd om slechts 1 kolom te tonen (in de `Columns` property), namelijk de waarde van het `M_TIMESTAMP` veld. Het resultaat is een lijst van tijdstippen in de grid. `TDataset` en `TSQLQuery` gebruiken het concept van een `Cursor`: Dit is het huidige record in de dataset. De cursor kan van een rij naar de andere verplaatst worden door de `Prev` en `Next` methodes van `TDataset`. Wanneer de laatste rij bereikt is, zal de `EOF` property op `True` staan, en als het eerste record bereikt is, zal `BOF` gelijk zijn aan `True`. Als de dataset leeg is, zijn zowel `BOF` als `EOF` gelijk aan `True`.

Een typische manier om iets te doen met alle records in een dataset ziet er dan als volgt uit:

```
With SomeQuery do
  While not EOF do
    begin
    // Doe iets met het huidige record
    Next;
    end;
```

Op een rij in de grid klikken met de muis zal de cursor van de dataset op het record zetten dat overeenstemt met de geselecteerde rij: Hierdoor wijzigt het huidige record, en het label dat het tijdstip toont zal automatisch de waarde uit het nieuwe huidige record tonen. Het is mogelijk om op dergelijke gebeurtenissen te reageren door een event handler te maken: TDataset heeft een hoop event handlers (AfterOpen, AfterScroll, AfterEdit) die de programmeur toestaan op de handelingen van de gebruiker te reageren.

Een van deze event handlers is `AfterScroll`. Deze wordt opgeroepen wanneer de dataset cursor locatie wijzigt. Dit event kan gebruikt worden om het beeldje in het huidige record te tonen in de `TImage` control (`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;
```

De code is vrij eenvoudig: het BLOB veld dat het beeldje bevat (`M_IMAGE`) wordt opgevraagd, en de inhoud wordt bewaard in een memory stream (`M`). Het beeldje van de `TImage` control wordt dan van de stream geladen.

De code demonstreert een belangrijk aspect van `TDataset`: wanneer de query geopend wordt, wordt een lijst componenten aangemaakt die afstammen van `TField`. Elk element in de lijst stelt een kolom voor uit het resultaat van de query, en kan gebruikt worden om de waarde van de kolom uit het huidige record op te halen. De `TField` componenten zijn niet allemaal van dezelfde klasse: voor integer kolommen, wordt een `TIntegerField` gemaakt, voor tijdstippen een `TDateTimeField`. Voor Blob velden wordt een `TBlobField` instance gemaakt: deze klasse heeft een methode om de data van de blob in een stream te bewaren, of de waarde in te stellen uit een stream.

Het resultaat van dit alles kan bekeken worden in figure 3 on page 10.

5 conclusion

Het is eenvoudig om SQL gebaseerde databanken aan te spreken met Lazarus. Er bestaan verschillende mechanismen om de data op te halen en te bewerken: in dit artikel werd de basis van 1 van deze mechanismes belicht: SQLDB, omdat het standaard met Lazarus wordt meegeleverd, en dus makkelijk in te zetten is in een klein (of zelfs een groot) project.

Figure 3: Het programma om de beeldjes te bekijken in actie