AnyDAC macros en scripting

Michaël Van Canneyt

May 24, 2012

Abstract

AnyDAC werd in een vorig artikel geintroduceerd. Een van de onderwerpen was de ondersteuning voor macros. In dit artikel wordt dit brede onderwerp verder onderzocht: er zijn vele mogelijkheden, en in het bijzonder: SQL scripts. AnyDAC biedt een SQL script component aan die bijna alle macros van de AnyDAC engine ondersteunt.

1 Inleiding

AnyDAC is een veelzijdige verzameling Delphi componenten die toegang verschaffen tot een hele reeks databanken. In een vorig artikel werd het ingeleid: de architectuur en de basis werking van AnyDAC werden uitgelegd.

In dit artikel ligt de focus op een van de vele mogelijkheden van AnyDAC: het pre-processen van SQL commandos en het gebruik van macros en expressions in de SQL commandos die via AnyDAC uitgevoerd worden. In het inleidende artikel werd getoond hoe macros kunnen gebruikt worden om bijvoorbeeld de sortering van een query resultaat te wijzigen door de `ORDER BY` clausule van een SQL `SELECT` command aan te passen. Er is echter veel meer mogelijk met de SQL preprocessor van AnyDAC.

De focus AnyDAC van AnyDAC ligt op de SQL mogelijkheden van een databank. Alle data manipulatie gebeurt via SQL commandos. Hoewel SQL zeer gestandardiseerd is, zijn er nog steeds veel verschillen tussen de verschillende SQL implementaties, en sommige daarvan zijn vrij fundamenteel. Met AnyDAC kunnen deze verschillen opgevangen worden: de SQL commandos kunnen bewerkt worden voor ze naar de RDBMS gestuurd worden. Hierdoor kan een toepassing, gemaakt met AnyDAC communiceren met elke ondersteunde databank zonder dat de toepassing opnieuw gecompileerd moet worden.

De mogelijkheden van de verschillende pre-processing mechanismes in AnyDAC zullen gedemonstreerd worden met de leerling opvolg toepassing die getoond werd in het vorige artikel: het zal uitgebreid worden zodat het ook met MySQL kan werken. Hiervoor moet een `TADPhysMySQLDriverLink` component op de main form geplaatst worden, zodat de benodigde driver in de applicatie gelinkt wordt. Er is ook een nieuw menu-item aanwezig dat toestaat het soort databank verbinding te kiezen alvorens de verbinding gemaakt wordt.

2 Conditional substitution

Een fundamenteel verschil tussen verschillende SQL databanken is de omgang met autoincremental velden: een veld waarvan de waarde automatisch opgehoogd wordt elke keer er een nieuw record aan de tabel wordt toegevoegd. Beide tabellen in de tracker databank bevatten een primary key veld van dit type.

Databank servers zoals MySQL en MS-SQL hebben een speciaal veld type voor auto-incremental velden, en een functie in de API om de waarde van het laatst toegevoegde record op te halen. Andere servers (Oracle, Firebird, PostGreSQL) hebben geen speciaal veld, maar gebruiken een normaal integer veld, en de server heeft de beschikking over sequences (reeksen, ook wel generator of identities genaamd) om nieuwe waardes voor het veld te genereren. Deze nieuwe waarde moet opgehaald worden en in de tabel toegevoegd worden zoals elk ander veld.

In de praktijk wil dit zeggen dat voor MySQL, een record toevoegen aan de tabel PUPILS als volgt moet:

```
INSERT INTO PUPILS
   (PU_FIRSTNAME,PU_LASTNAME)
VALUES
 (:PU_FIRSTNAME, :PU_LASTNAME)
```

De waarde voor `PU_ID` wordt automatisch gegenereerd en kan opgehaald worden met de speciale functie (`mysql_insert_id`).

Voor Firebird nemen we aan dat er een generator `GEN_PUPILS` bestaat, en dan moet een nieuw record als volgt worden toegevoegd:

```
INSERT INTO PUPIL
   (PU_ID, PU_FIRSTNAME,PU_LASTNAME)
VALUES
 (GEN_ID(GEN_PUPILS,1),:PU_FIRSTNAME, :PU_LASTNAME)
```

Of, met de nieuwere syntax:

```
INSERT INTO PUPIL
   (PU_ID, PU_FIRSTNAME,PU_LASTNAME)
VALUES
   (NEXT VALUE FOR GEN_PUPILS, :PU_FIRSTNAME, :PU_LASTNAME)
```

Als alle INSERT commandos in een toepassing gedupliceerd moeten worden om beter met 2 verschillende databanken te werken, dan zou dit zeer arbeidsintensief en moeilijk te onderhouden zijn. AnyDAC biedt hier een oplossing: conditional substitution.

De algemene syntax voor conditional substitution is als volgt:

```
{if X} Y {fi}
```

Hier is X de naam van een macro, of de naam van het type databank server waar het SQL commande op uitgevoerd zal worden. De volledige lijst van namen staat gedetailleerd in de AnyDAC documentatie. Als deze vorm gebruikt wordt, zal `Y` in het SQL commando worden opgenomen indien de macro X gedefinieerd is, of gelijk is aan de naam van het type SQL databank.

Als we deze mogelijkheid toepassen op het insert commando, zou het er als volgt kunnen uitzien:

```
INSERT INTO PUPIL
   ({if INTRBASE} PU_ID,{fi}
    PU_FIRSTNAME,PU_LASTNAME)
VALUES
   ({if INTRBASE} GEN_ID(GEN_PUPILS,1),{fi}
    :PU_FIRSTNAME, :PU_LASTNAME)
```

`INTRBASE` is de naam van de Interbase/Firebird support laag in AnyDAC.

Een tweede vorm van conditional subsitution gebruikt `IIF`:

```
{iif X1, Y1, X2, Y2, .. XN,YN [, YE] }
```

Als X1 gedefinieerd is, dan wordt Y1 gebruikt. Indien X1 niet gedefinieerd is, maar X2 wel, dan wordt Y2 gebruikt. Indien geen van de macros Xn gedefinieerd is, dan wordt YE gebruikt, indien het opgegeven werd.

Dit kan gebruikt worden om het verschil in syntax tussen firebird and PostGreSQL voor het aanmaken van een nieuwe waarde op te vangen:

```
INSERT INTO PUPILS
  (PU_ID,PU_FIRSTNAME,PU_LASTNAME)
VALUES
  ({iif INTRBASE, NEXT VALUE FOR GEN_PUPILS,
        PG, nextval(GEN_PUPIL) },
  :PU_FIRSTNAME, :PU_LASTNAME)
```

Of om het verschil in namen van ingebouwde functies op te vangen. De lengte van een string bijvoorbeeld, zoals in het volgende voorbeeld getoond wordt:

```
SELECT * FROM PUPILS
WHERE
  {IIF INTRBASE, OCTET_LENGTH, MYSQL, LENGTH}(PU_FIRSTNAME)<3
```

Dit zal alle leerlingen ophalen waarvan de naam uit minder dan 3 bytes bestaat.

3 Escape sequences

Voor het laatste geval, waar RDBMS verschillende namen gebruiken voor functies met dezelfde taak, is het niet nodig conditional substitution te gebruiken: AnyDAC biedt zijn eigen mechanisme aan: escape sequences. In essentie wil dit zeggen dat het SQL commando de functie naam bevat in een speciale vorm, en de AnyDAC preprocessor zal het vervangen door de naam van de functie die de RDBMS gebruikt, alvorens het command naar de RDBMS te sturen.

Een escape sequence wordt opgegeven in de volgende vorm:

```
{fn FUNCTIONNAME(Args)}
```

Hier wordt `FUNCTIONNAME` gewijzigd in de naam die de RDBMS verwacht. In sommige gevallen kan het `fn` prefix weggelaten worden, en de escape sequence wordt dan:

```
{FUNCTIONNAME(Args)}
```

Bijvoorbeeld, een alleen-hoofdletters versie van een tekstveld kan opgehaald worden met UCASE:

```
SELECT {fn UCASE(PU_FIRSTNAME)} FROM PUPIL
```

Dit wordt dan

```
SELECT UPPER(PU_FIRSTNAME) FROM PUPIL
```

voor Firebird.

AnyDAC kan dit ook doen voor bepaalde operaties, zoals het aaneen plakken van tekstwaarden. De RDBMSen gebruiken verschillende tekens voor deze operatie. MySQL gebruikt een plus teken (+):

```
select PU_FIRSTNAME + ' ' + PU_LASTNAME FROM PUPIL
```

Firebird daarentegen gebruikt 2 pipe symbolen (||):

```
select PU_FIRSTNAME || ' ' || PU_LASTNAME FROM PUPIL
```

In AnyDAC, kan dit verenigd worden tot

```
select
  {fn CONCAT(PU_FIRSTNAME, {fn CONCAT(' ',PU_LASTNAME)})}
FROM
  PUPIL
```

Het bovenstaande voorbeeld toont meteen ook aan dat het mogelijk is de functies te nesten.

Er zijn een heleboel functies die op deze wijze gebruikt kunnen worden: tekenreeksen manipuleren, wiskundinge functies, type conversies. Alle beschikbare functies zijn opgelijst in de AnyDAC documentatie.

Om het gebruik van escape sequences te demonstreren, wordt functionaliteit om een leerling te zoeken in de databank toegevoegd aan de applicatie. De zoekoperatie gebeurt op de voor en achternaam van de leerling. De functie accepteert als argumenten een naam om te zoeken, en 2 opties:

- is de zoekoperatie hoofdlettergevoelig
- moet op de hele naam gezocht worden, of slechts op een deel ?

Om dit te maken worden een edit control (`ESearch`) en 2 comboboxes (`CBExact` en `CBCaseSensitive`) op en nieuwe form geplaatst. Een knop en een DBGrid vervolledigen het visuele deel van het scherm. De query wordt uitgevoerd met een `TADQuery` component (`QSearch`), met de volgende `SQL` property:

```
SELECT
   PU_ID, PU_FIRSTNAME, PU_LASTNAME
FROM
  PUPIL
WHERE
{if &EXACT}
{if &NOTCASESENSITIVE}
  ({fn UCASE(PU_LASTNAME)} = :LASTNAME) OR
  ({fn UCASE(PU_FIRSTNAME)} = :FIRSTNAME)
{fi}
{if &CASESENSITIVE}
  (PU_LASTNAME = :LASTNAME) OR
  (PU_FIRSTNAME = :FIRSTNAME)
{fi}
{fi}
{if &NOTEXACT}
{if &NOTCASESENSITIVE}
```

```
  ({fn UCASE(PU_LASTNAME)} LIKE
    {fn CONCAT('%',{fn CONCAT(:LASTNAME,'%')})}) OR
  ({fn UCASE(PU_FIRSTNAME)} LIKE
    {fn CONCAT('%',{fn CONCAT(:FIRSTNAME,'%')})})
{fi}
{if &CASESENSITIVE}
  (PU_LASTNAME LIKE
    {fn CONCAT('%',{fn CONCAT(:LASTNAME,'%')})}) OR
  (PU_FIRSTNAME LIKE
    {fn CONCAT('%',{fn CONCAT(:FIRSTNAME,'%')})})
{fi}
{fi}
```

AnyDac ondersteunt geen ELSE clausule voor een escape sequence. Dit gebrek wordt opgevangen m.b.v 2 macros, die het tegenovergestelde effect hebben: EXACT versus NOTEXACT, en CASESENSITIVE versus NOTCASESENSITIVE. De CONCAT functie wordt gebruikt om wildcard characters aan de SQL LIKE clausule toe te voegen, en de UCASE functie wordt gebruikt om op een hoofdletter-ongevoelige manier te zoeken.

Door een waarde voor de gewenste macros op te geven, kan het SQL commando gewijzigd worden zodat het de gewenste zoekoperatie uitvoert. Als de SEARCH knop wordt ingedrukt, wordt de volgende code uitgevoerd:

```
procedure TSearchForm.BSearchClick(Sender: TObject);

  Procedure SetM(N : String; B : Boolean);

  begin
    If B then
      QSearch.MacroByName(N).Value:='OK'
    else
      QSearch.MacroByName('NOT'+N).Value:='OK';
  end;

var
  n : String;
  I : integer;
  M : TADMacro;

begin
  With QSearch do
    begin
    For I:=0 to Macros.Count-1 do
      Macros[i].Clear;
    SetM('EXACT',CBExact.Checked);
    SetM('CASESENSITIVE',CBCaseSensitive.Checked);
    Prepare;
    N:=ESearch.Text;
    if Not CBCaseSensitive.Checked then
      N:=UpperCase(N);
    Params.ParamByName('FIRSTNAME').AsString:=N;
    Params.ParamByName('LASTNAME').AsString:=N;
    Open;
    end;
end;
```

De `setM` procedure zal macro N of NOTN aanmaken, afhankelijk van de waarde van `B`. Deze hulpfunctie wordt dan gebruikt om de waarde voor de gewenste macros te zetten, naargelang de waardes van de checkboxes. Als de waardes gezet zijn, worden de parameters voor de query ingevuld, en wordt de query geopend.

Dit is de belangrijkste functie in het zoekscherm. Er zijn wat hulpfuncties nodig om de connectie door te spelen vanuit het hoofdscherm, en wat functionaliteit om een record te selecteren en de ID van de geselecteerde leerling terug te geven. Deze functies zijn terug te vinden in de broncode van het project.

Als alles klaar is, ziet het resultaat er uit als in figure 1 on page 7.

4 Constant embedding

Zoals met functie namen, vereisen databanken vaak verschillende manieren om constantes op te geven. In het bijzonder worden datum en tijd waardes op veel verschillende manieren opgegeven. AnyDAC heeft enkele speciale escape sequences om constantes in het SQL commando op te nemen. De syntax ziet er als volgt uit:

```
{T value}
```

Hier is T het type van de constante, en is een van de volgende waardes:

e een floating-point waarde. De waarde moet opgegeven worden zoals in Delphi, en wordt omgezet in een vorm die de databank server begrijpt. The float value must be written as it would be in Delphi, and it will be inserted in the form the RDBMS expects.

d een datum. De datum moet opgegeven worden als YYYY-MM-DD, quotes zijn niet nodig.

t een tijd. De waarde moet opgegeven worden als hh:mm:ss, met het uur in een 24-uurs formaat.

dt een datum en tijd. De waardes moeten opgegeven worden zoals voor de `d` en `t` gevallen.

l Een boolean, die 'true' of 'false' mag zijn. Die wordt omgezet naar iets wat de RDBMS verwacht (meestal 0 en 1)

s Een string. De string zal door de correcte quote karakters omsloten worden (meestal een enkele quote).

id Een identifier. Dit is nuttig voor het opgeven van identifiers die een gereserveerd woord vormen, of identifiers die een spatie bevatten. Ook hier wordt de identifier door de correcte quote karakters omsloten. Bijvoorbeeld

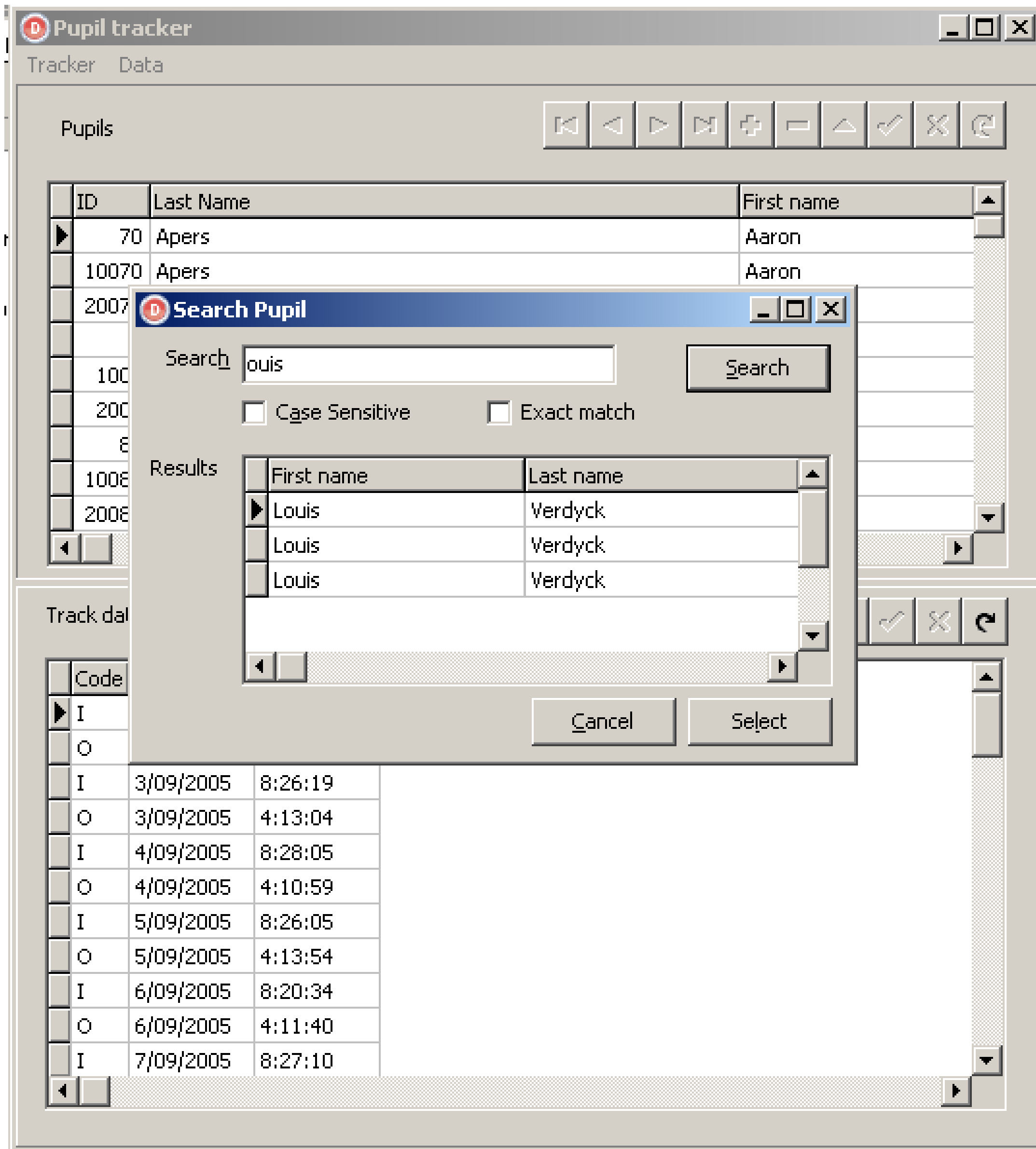
```
{id ORDER DETAILS}
```

Zal vertaald worden naar `[Order Details]` voor Microsoft SQL server of MS Access, maar wordt "`Order details`" (dubbele quotes) voor Firebird, en '`Order Details`' (backtick) voor MySQL.

5 Scripting en macros

Macros en escape sequences zijn ook nuttig in de script component van AnyDAC.

Figure 1: Een leerling zoeken



Bij het werken met SQL databanken komt er onvermijdelijk een moment waarop een script op de databank moet uitgevoerd. Een van deze momenten is het aanmaken van de databank tabellen. Een ander moment kan de update zijn naar een nieuwe versie van de databank, of het aanmaken van voorbeeld- of testgegevens.

AnyDAC biedt een`TADScript` component aan die het uitvoeren van SQL scripts mogelijk maakt: een reeks SQL commandos in een stringlist of een bestand. De `TADScript` component zal het script parsen, en zal de SQL commandos een voor een uitvoeren - wat zo de functionaliteit is die men van een dergelijke component verwacht.

Maar, de component doet meer dan dat. Er zijn enkele extra functionaliteiten beschikbaar:

1. Macros en escape sequences worden afgehandeld net zoals in een `TADQuery` component.
2. De mogelijkheid bestaat om een logboek bestand aan te maken (de 'spool file' genaamd), met verschillende opties, zoals het toevoegen van de resultaten van een SELECT commando aan het bestand.
3. Verschillende scripts kunnen opgegeven worden.
4. Zelfgemaakte commandos kunnen in het script opgenomen worden. AnyDAC zelf definieert verschillende commandos.
5. Het script kan gevalideerd worden.
6. Het script kan stap-voor-stap uitgevoerd worden.
7. De uitgevoerde commandos kunnen op het scherm getoond worden.
8. Parameters kunnen aan het script doorgegeven worden.

Er zijn 2 manieren om deze component te gebruiken:

1. Een van de `ExecuteNNN` methodes kan opgeroepen worden met de naam van een script bestand, of met een stringlist die de commandos bevat.
2. De `SQLScripts` property kan opgevuld worden met een reeks scripts, waarna `ExecuteAll` opgeroepen wordt.

De volgende methodes zijn beschikbaar:

ExecuteStep Haalt het volgende commando op in het script in de `SQLScripts` property, en voert het uit.

ExecuteAll Voert alle commandos uit in het eerste script in de `SQLScripts` property. Een optionele lijst van argumenten kan worden opgegeven worden in de `Arguments` property van de SQLScript component.

ExecuteFile dit commando leest het opgegeven bestand in en voert het script uit. Indien argumenten zijn opgegeven, worden ze doorgespeeld aan het script.

ExecuteScript Aan dit commando kan een SQL script in een `TStrings` instance worden doorgegeven. Het script wordt dan onmiddellijk uitgevoerd. Ook hier is het mogelijk argumenten op te geven die dan aan het script worden doorgegeven.

De laatste 2 vormen zullen eenvoudigweg

- De SQLScripts property leegmaken.

- Het doorgegeven script aan de SQLScripts property toevoegen (het wordt uit het bestand gelezen in het geval van `ExecuteFile`).
- De `Arguments` property zetten indien argumenten werden doorgegeven.
- `ExecuteAll` oproepen.

Het is dus belangrijk te onthouden dat de inhoud van de SQLScripts property gewijzigd wordt door deze methodes.

De `SQLScripts` property is een verzameling (collection) van SQL scripts met een naam. Alleen het eerste script wordt uitgevoerd door `ExecuteAll`. Om de commandos in de andere scripts te laten uitvoeren, moet het script ge-include worden met een van de AnyDAC commandos van de script component Om een tweede script (genaamd 'firebird') uit te laten voeren, moet het als volgt opgeroepen worden:

```
@firebird
```

Als alternatieven zijn de commandos `INPUT` of `START` beschikbaar:

```
INPUT firebird
```

Voor de leerling tracker toepassing kan dit gebruikt worden om delen van de databank uit te voeren naargelang het type databank server: In MySQL, kan een `AUTO_INCREMENT` veld gebruikt worden als sleutelveld. Voor Firebird, wordt een auto-increment veld meestal met een generator en een trigger opgevuld, als in het volgende voorbeeld:

```
CREATE GENERATOR GEN_PUPIL;

SET TERM ^ ;

CREATE TRIGGER PUPIL_INSERT FOR PUPIL
ACTIVE BEFORE INSERT POSITION 0
AS
begin
IF (NEW.PU_ID IS NULL) then
  NEW.PU_ID = GEN_ID(GEN_PUPIL,1);
end ^
```

Voor de voorbeeldapplicatie wil dit zeggen dat het script om de databank aan te maken in 2 delen moet gesplitst worden: Een gemeenschappelijk deel dat de tabellen en indexen maakt, en een ander deel dat de generators en triggers voor Firebird aanmaakt.

De volgende code kan uitgevoerd worden wanneer een databank verbinding gemaakt wordt: Het kijkt na of de tabel `PUPIL` bestaat. Indien de tabel niet bestaat wordt het databank creatie script uitgevoerd. Als de databank server firebird is, wordt een lijn aan het creatie script toevoegd dat ervoor zorgt dat het firebird-specifieke script uitgevoerd wordt:

```
procedure TForm1.CheckTables;

begin
  B:=False;
  With TADQuery.Create(Self) do
    try
      Connection:=CTracker;
      try
```

```
        Open('select * from PUPIL where (0=1)');
        // if we are here, the table exists, and we can exit.
        exit;
      except
        on E: EADDBEngineException do
          if E.Kind <> ekObjNotExists then
            Raise;
      end;
    finally
      Free;
    end;
  if (CTracker.ConnectionDefName='TrackerFB') then
    ADSCreate.SQLScripts[0].SQL.Add('INput firebird;');
  ADSCreate.ExecuteAll;
end;
```

In het geval van een databank update script zou deze mogelijkheid gebruikt kunnen worden om verschillende updates aan elkaar te koppelen: het hoofd script kan leeggelaten worden, en de verschillende scripts om van versie naar versie bij te werken kunnen in scripts met naam `versionN` toegevoegd worden. De databank bijwerken naar de meest recente versie is dan eenvoudig:

```
Procedure TForm1.UpdateDatabase(CurrentVersion : integer);

begin
  With SCUpdate.SQLScripts[0] do
    begin
    if (CurrentVersion<2) then
      Add('input version2');
    if (CurrentVersion<3) then
      Add('input version3');
    end;
  SCUpdate.ExecuteAll;
end;
```

Zoals al eerder gezegd, worden escape sequences en macros herkend en afgehandeld in de SQL script component. Dat wil zeggen dat het volgende zal werken:

```
CREATE TABLE PUPIL (
  PU_ID INTEGER NOT NULL {if mysql}AUTO_INCREMENT{fi},
  PU_FIRSTNAME VARCHAR(50) NOT NULL,
  PU_LASTNAME VARCHAR(50) NOT NULL,
  CONSTRAINT PUPIL_PK PRIMARY KEY (PU_ID)
);
```

De MySQL versie van deze tabel definitie zal de `AUTO_INCREMENT` keyword gebruiken, maar in Firebird zal dit woord achterwege gelaten worden.

Een belangrijk detail is dat het afhandelen van de escape sequences en macros pas gebeurt wanneer de script engine het commando aan de AnyDAC verbinding component geeft om uitgevoerd te worden: De SQL script component handelt dit zelf niet af. Dat wil zeggen dat de volgende constructie

```
{if INTRBASE}
INPUT firebird
{fi}
```

niet zal werken: De SQL script component geeft het geheel door aan de fysische laag van AnyDAC, die het doorspeelt aan de firebird databank. De Firebird Databank zal het INPUT commando niet herkennen, het is een commando dat door de SQLScript component moet worden afgehandeld.

De volgende lijst bevat de volledige lijst commandos die door AnyDAC uitgevoerd worden:

INPUT Lees een ander script. Alternatieven zijn @ en START.

ACCEPT Vraag de gebruiker om gegevens.

CONNECT Maak verbinding met een andere AnyDac connectie.

COPY Copieer data van of naar een bestand.

DEFINE Definieer een macro

DELIMITER Zet het SQL script commando scheidingsteken.

DISCONNECT Verbreek de verbinding.

EXECUTE Voer een SQL commando blok uit.

EXIT Stopt de uitvoering van het script en commit alle wijzigingen. De `QUIT` vorm doet een rollback alvorens de uitvoering te stoppen, het `STOP` commando stopt eenvoudigweg de uitvoering van het script.

HELP Geeft een lijst van alle commandos.

HOST Voert een extern programma uit.

PAUSE Pauzeert de uitvoering van het script en wacht op de gebruiker.

PROMPT Schrijft een tekst naar het logboek.

PRINT Drukt de namen en waarden van variabelen.

REMARK Een opmerking (commentaar)

SET Dit kan gebruikt worden om verschillende TADScript component properties vanuit het script te zetten.

SPOOL Zend gegevens naar het logbestand.

UNDEFINE Verwijder een macro.

VARIABLE Definieer een variabele.

CREATE DATABASE Maak een nieuwe databank. Alleen in de INTRBASE engine.

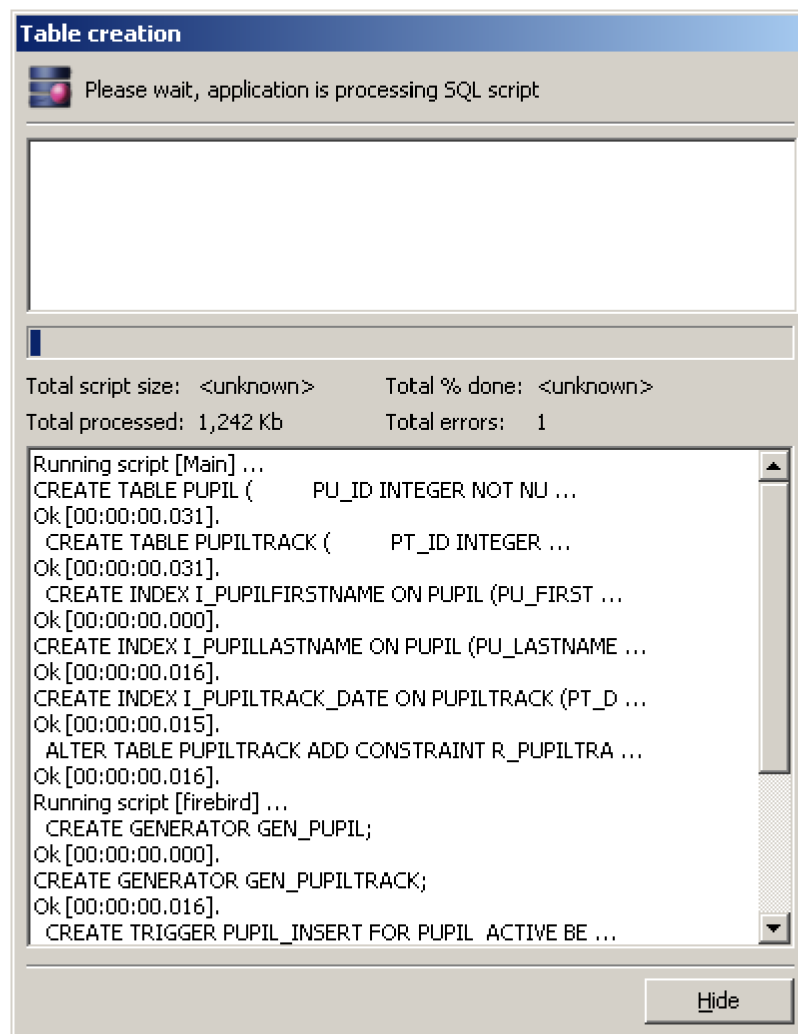
DROP DATABASE Verwijder een databank. Alleen in de INTRBASE engine.

Deze commandos worden door de scripting component uitgevoerd, ze worden niet naar de databank server gestuurd.

De SQL Script component kan alle uitgevoerde commandos naar een dialoog zenden, en deze dialoog gebruiken om gegevens op te vragen van de gebruiker. Om dit te doen, moet een `TADGUIxFormsScriptDialog` component op de form gezet worden, en toegekend worden aan de `ScriptDialog` property van de `TADSQLScript` component.

Als dit gebeurd is, en de toepassing verbinding maakt met een nieuwe Firebird databank wordt de dialoog getoond, zoals in figure 2 on page 12.

Figure 2: Databank creatie logboek



6 Conclusion

In dit artikel, werd de pre-processor van AnyDAC gedemonstreerd. De preprocessor is in het hart van de AnyDAC structuur geimplementeerd, zodat hij steeds beschikbaar is, niet in het minst in de SQL script component. Deze component bevat standaard reeds veel functionaliteit, maar is daarenboven uitbreidbaar, wat hem geschikt maakt voor de meeste toepassingen die iets doen met databanken