

Der Entwickler 03/2002

Nah an der Quelle - DOA (Native Zugriffskomponenten auf ORACLE für Delphi und C++-Builder)

Autor: Henry Wolf

Borland wird sich in den nächsten Versionen von der BDE verabschieden und diese durch dBExpress endgültig ablösen. Dabei wird vorwiegend der Zugriff auf SQL-Datenbankserver wie Interbase, ORACLE usw. unterstützt. Dabei werden aber nach wie vor nicht alle Funktionen der Datenbank-Server ohne weiteres angesprochen werden können. Auch Alternativen wie ADO haben hierbei Schwächen. Native Komponenten haben hier Ihre Vorteile, da diese sich mit einer Datenbank-Plattform befassen und Ihre Zielplattform besser unterstützen. Ein Klassiker im Bereich ORACLE ist die Komponenten-Bibliothek Direct Oracle Access (DOA) der Firma Allround Automations.

DOA unterstützt in der aktuellen Version 3.4.6 vom März 2002 die *Delphi- und C++-Builder-Version 3-6* und den Zugriff auf *ORACLE-Datenbanken der Versionen 7, 8i und 9i*. Während man zum Zugriff auf Oracle via BDE oder dBExpress die „preisgünstige“ Enterprise-Version von Delphi bzw. dem C++-Builder benötigt, reicht für DOA die *Professionell-Version* aus. Im folgenden Artikel möchten wir Ihnen einen kurzen Überblick über diese Komponenten-Bibliothek geben.

Datenzugriffskonzept

Die Komponenten-Bibliothek DOA stellt eine effiziente Kapselung der Oracle-API-Library, der sogenannten OCI-Schnittstelle, dar und greift im Gegensatz zu dBExpress, ADO und der BDE direkt auf die ORACLE-API zu.

Für die Distribution der Anwendung sind außer dem Programm und dessen Modulen keine weiteren zusätzlichen „Runtime-DLL's“ (siehe BDE) erforderlich. Auf den Arbeitsplatzmaschinen muß der ORACLE-Client installiert und der Zugriff auf unsere Ziel-Datenbank via SQL*Net über lokalen Namen oder ORACLE-Names konfiguriert sein.

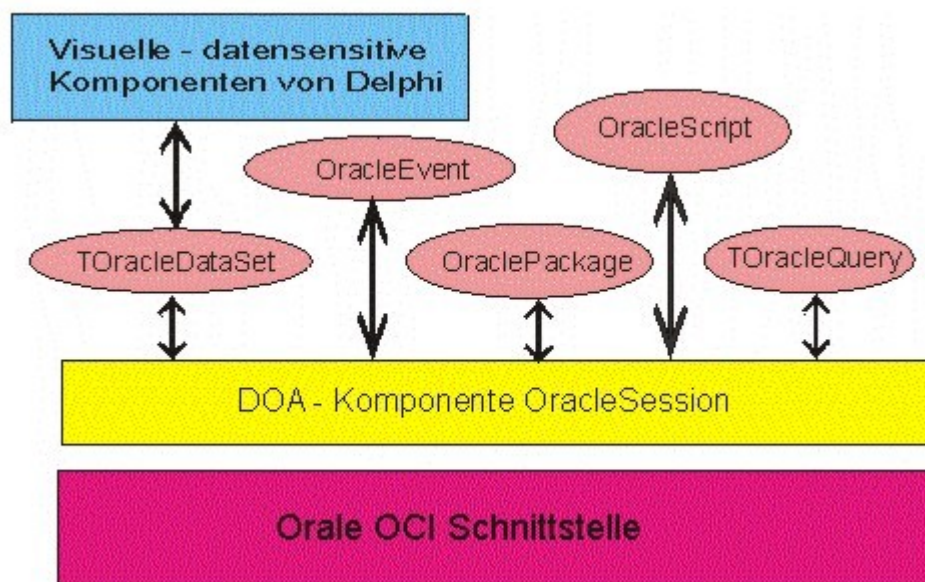


Abb. 1 Komponentenstruktur von DOA

Komponentenübersicht

Im Folgenden wollen wir uns nun die Komponenten von DOA etwas genauer anschauen. DOA enthält folgende Komponenten:

- OracleSession
- OracleQuery

- OracleDataSet
- OraclePackage
- OracleEvent
- OracleScript
- OracleDirectPathLoader
- OracleLogon
- OracleNavigator

OracleSession

OracleSession ist ein vergleichbares Äquivalent zur TDatabase – Komponente der BDE. Diese Komponente repräsentiert die Eigenschaften der aktuellen Datenbanksitzung. Neben dem Aufbau der Datenbankverbindung über die Propertys LogonUsername, LogonPassword und LogonDatabasename stehen zahlreiche weitere Eigenschaften und Methoden zum Handling der Datenbanksession zur Verfügung.

Im Property-Record Preferences können zahlreiche Einstellungen zu Eigenschaften der Session definiert werden (FloatPrecision, IntegerPrecision usw.). An dieser Stelle ist im Falle des Zugriffes auf eine Oracle7-Datenbank das Property UseOCI7 auf true zu stellen.

Über das Property ConnectAs kann die Art der Anmeldung an der Datenbank festgelegt werden. Im Standardfall ist dies sicher caNormal, also eine normale User-Session. Für Toolentwickler benötigen wir jedoch oft Anmeldungen als SYSDBA oder als SYSOPER, das heißt als Administratoren mit besonderen Privilegien (Jobmanagement, Datenbank Start und Stop u.a.).

Wird das Property DesignConnection aktiviert (= true), dann werden die in der Designphase eingetragenen Anmeldeinformationen bei der Distribution automatisch entfernt. Wem ist es noch nicht passiert, daß er mal schnell ein Programm weiter gereicht hat und vergessen hat, seine speziellen Anmeldeinformation zu entfernen. Das kann in diesem Fall nicht mehr passieren.

Über das Property OptimizerGoal kann definiert werden, in welchem Modus der ORACLE-Optimizer die SQL-Statements verarbeiten soll.

Ein wichtiges Property ist SQLTrace. Durch das Ändern dieses Property kann der Serverseitige Trace zur Analyse mit dem Oracle-Werkzeug tkprof aktiviert und deaktiviert werden. Möchte man dies nutzen, sollte man nicht vergessen, in der initOra der Instanz den Parameter TIMED_STATISTICS auf true zu setzen, da sonst die Zeitangaben für die Abarbeitung der SQL-Statements fehlen werden.

Über die Einstellungen der Properties im Bereich MTSOptions kann die Einbindung der Datenbanksitzung in eine MTS-Umgebung (Microsoft Transaction Server) konfiguriert werden. Unterstützt wird das Ressourcen-Pooling und die Transaktionskontrolle innerhalb einer MTS-Umgebung.

Mit OracleLogonSession wird eine Dialog-Komponente zur Verfügung gestellt, welche bei der Anmeldung an einer Datenbank automatisch einen Anmeldedialog anzeigt, welcher die getätigten Anmeldungen und die damit verbundenen Informationen „historisiert“ und schnell wieder verfügbar macht.

Transactionshandling

Neben der Konfiguration ist die Komponente OracleSession u.a. für das Transaktionshandling verantwortlich. Hierfür verfügt die Komponente über die entsprechenden Methoden COMMIT und Rollback. Eine Methode Starttransaction wie bei der BDE gibt es nicht. COMMIT beendet die letzte Transaction und startet eine neue. Von Oracle kennen wir sogenannte SAVEPOINTS, sprich „GOTO-Labels“ innerhalb einer Transaction. Mittels der Methoden SavePoint können diese erkannt bzw. definiert werden, und mittels der Methode RollbackToSavePoint kann die Transaktion bis zu diesem „GOTO“ rückgängig gemacht werden.

Basis-Datenmodul für eine einfache DOA-Anwendung

```
unit DatenmoduleBase;
```

```
interface
```

```
uses Classes ,Forms ,Oracle;
```

```
type
```

```
  TDataModuleBase = class(TDataModule)
```

```
    OracleMainSession: TOracleSession;
```

```
  private
```

```
    {Private-Deklarationen }
```

```

public
  {Public-Deklarationen }
  {----- Exceptionhandling ----- }
  procedure ExceptionWithoutTransaction (A:EOracleError );
  procedure ExceptionWithTransaction (A:EOracleError );
  { ----- Transactionshandling ----- }
  procedure StartTransaction;
    procedure Commit;
    procedure Rollback;
    { ----- Oracle Server Trace ----- }
    procedure ProcStartOracleTrace;
    procedure ProcStopOracleTrace;
    function ProcTraceStatus : Boolean;
    { ----- Datenbank - Anmeldemethoden ----- }
    function ProcConnectState : Boolean;
    procedure ProcConnectDatabase ( Username : String ; Password : String ;
                                   HostString : String );

    procedure ProcDisConnectDatabase;
end;
var
  DataModuleBase: TDataModuleBase;
implementation
uses
  {$ifdef DEV_PHASE}
  OracleMonitor, //für DOA Oracle-Monitor-Tracing, in Release-Version entfernen
  {$endif}
  Dialogs;

{$R*.DFM}

{----- Exceptionhandling ----- }

procedure TDataModuleBase.ExceptionWithoutTransaction ( A:EOracleError );
begin
  MessageDlg( A.Message , mtError, [mbOK], 0);
end;

procedure TDataModuleBase.ExceptionWithTransaction ( A:EOracleError );
begin
  OracleMainSession.Rollback;
  MessageDlg( A.Message , mtError, [mbOK], 0);
end;

{----- Transactionshandling ----- }
procedure TDataModuleBase.StartTransaction;
begin
  OracleMainSession.Commit;
end;

procedure TDataModuleBase.Commit;
begin
  OracleMainSession.Commit;
end;

procedure TDataModuleBase.Rollback;
begin
  OracleMainSession.Rollback;
end;

{----- Oracle Server Trace ----- }
procedure TDataModuleBase.ProcStartOracleTrace;

```

```

begin
    OracleMainSession.SQLTrace := stTrue;
end;

procedure TDataModuleBase.ProcStopOracleTrace;
begin
    OracleMainSession.SQLTrace := stFalse;
end;

function TDataModuleBase.ProcTraceStatus : Boolean;
begin
    result := OracleMainSession.SQLTrace = stTrue;
end;

{ ----- Datenbank - Anmeldemethoden ----- }
procedure TDataModuleBase.ProcConnectDatabase
    ( Username : String ; Password : String ; HostString : String );
begin
    try
        with OracleMainSession do begin
            Connected := false;
            LogonDatabase := HostString;
            LogonPassword := Password;
            LogonUsername := Username;
            Connected := true;
        end;
    except

        MessageDlg ( 'Ihre Anmeldung war nicht erfolgreich.'
            + #13+#10+'Bitte prüfen Sie Ihre Angaben zur Anmeldung'
            + #13+#10+'an der Datenbank !!' ,
            mtWarning, [mbOK], 0);
        raise;
    end;
end;

procedure TDataModuleBase.ProcDisConnectDatabase;
begin
    OracleMainSession.Connected := false;
end;

function TDataModuleBase.ProcConnectState : Boolean;
begin
    result := OracleMainSession.Connected;
end;

end.

```

Support für ORACLE-Standardpackages

Neben den bisher aufgeführten Properties und Methoden und zahlreichen anderen Funktionalitäten, deren Aufzählung den Rahmen des Artikles sprengen würde, soll die Unterstützung der ORACLE-Standard-Packages nicht unerwähnt bleiben. Diese Unterstützung umfasst die Packages `dbms_alert`, `dbms_application_info`, `dbms_job`, `dbms_output`, `dbms_pipe` and `utl_file`.

Wie ist dies nun realisiert? In DOA ist eine Basis-Klasse, `TDBMSPackage`, für Standard-Packages implementiert. Davon wurde in DOA für jedes Standard-Package eine entsprechende Ableitungen implementiert. Hier das Beispiel für `dbms_output`:

```

TDBMS_Output = class(TDBMSPackage)
    procedure Enable(const Buffer_Size: Integer);
    procedure Disable;

```

```

procedure Put(const a: Variant);
procedure Put_Line(const a: Variant);
procedure New_Line;
procedure Get_Line(out Line: string; out Status: integer);
procedure Get_Lines(out Lines: string; var NumLines: Integer);
end;

```

Anschließend wurde die Komponente OracleSession um ein entsprechendes Property zum Zugriff auf das Standard-Packages innerhalb der aktuellen Sitzung erweitert:

```
property DBMS_Output: TDBMS_Output;
```

Der Aufruf der Methoden dieses Packages in unserem Programm kann nun wie folgt implementiert werden:

```

procedure TBaseModule.DBMS_OUTPUT;
var Status: Integer;
    Line: string;
begin
    // Output maximal 100,000 bytes
    OracleSession.DBMS_Output.Enable(100000);
    //Aufruf der Procedure
    MyProcedure.Execute;
    // Retrieve all lines and display them in a memo
    Memo.Clear;
    repeat
        OracleSession.DBMS_Output.Get_Line(Line, Status);
        if Status <> glSuccess then Break;
        Memo.Lines.Add(Line);
    until False;
end;

```

Datenzugriffskomponenten

Nachdem wir uns angeschaut haben, wie die Session konfiguriert und aufgebaut wird, wollen wir uns den Datenzugriffs-Komponenten widmen. DOA bietet uns drei an. Dies sind OracleQuery, OracleDataSet und OraclePackage. Die aktuelle Sitzung wird bei diesen Komponenten prinzipiell im Property Session vermerkt.

OracleQuery dient der Ausführung von SQL-Statements oder von PL/SQL-Blöcken, deren Ergebnis nicht mit visuellen, datensensitiven Komponenten von Delphi dargestellt werden soll. Neben den DML-Befehlen (Select, Insert, Update) können auch folgende SQL-Befehle über diese Komponente ausgeführt werden:

- SQL-Statements der Data Definition Language z.B. create table, create procedure, grant role, etc.
- SQL-Statements für die Transaktionskontrolle (commit, rollback, savepoint, etc.)
- SQL-Statements zur Konfiguration der Session (alter session, set role)
- SQL-Statements zur Konfiguration der Datenbank (alter system).

Den Text des SQL-Statements oder des PL/SQL-Blockes wird wie bei TQuery im Property SQL eingetragen. Das Eintragen von Variablen erfolgt im Property-Editor der Eigenschaft Variables. Sollen Variablen/Parameter im Code mit Werten versorgt werden, geschieht dies nicht wie sonst allgemein üblich über die Methode ParamByName. Diese gibt es nicht. Statt dessen ist die Methode SetVariable zu verwenden.

```

ODS_TV_Master.Close;
ODS_TV_Master.SetVariable ( 'PRIMKEY' , 1 ); // Wert einem Parameter zuweisen
ODS_TV_Master.Execute;

```

Dabei ist der Bezeichner des Parameters und der Wert zu übergeben. Die „Wert-Übergabe-Variable“ ist hierbei als Variant deklariert.

Für die Ausführung von SQL-Statements, deren Ergebnis in den visuellen datensensitiven Komponenten von Delphi angezeigt werden sollen, ist die Komponente TOracleDataSet verfügbar. Diese Komponente wurde von TDataSet abgeleitet. Die wichtigsten Unterschiede zu den üblichen bekannten DataSet-Implementierungen, sind hierbei die Deklaration und Wertzuweisung von Variablen und Parametern, wie bei

der Komponente *TOracleDataSet*. Auch hier ist die Methode *ParamByName* nicht implementiert.

„Oracle-Speziell“ ist auch die Implementierung von „toten“ oder „lebenden“ Ergebnismengen. Lebende Ergebnismengen werden durch die Angabe der ROWID im SQL –Statement erzeugt.

```
SELECT A.ROWID,A.* FROM ADRESSE A
```

Wird die RowID nicht angegeben, kann die Ergebnismenge nicht bearbeitet werden. Die Ausführung eines in einem *OracleDataSet* definierten SQL-Statements erfolgt in der Regel über die bekannte Methode *Open*.

```
function TDataModuleHelpURL.PubProcGetHelp : String;  
begin  
  QueryGetURL.Close;  
  QueryGetURL.SetVariable ( 'HELP_KEY', 'P1' );  
  QueryGetURL.Open;  
  result := QueryGetURL.FieldByName ( 'URL' ).AsString;  
  QueryGetURL.Close;  
end;
```

Zum Aufruf von gespeicherten *Procedures* aus *Stored-Packages* steht uns in *DOA* die Komponente *OraclePackages* zur Verfügung. Nach der Zuweisung der *Session* ist im *Property PackageName* der Bezeichner des *Packages* einzugeben. Um *Prozeduren* oder *Funktionen* dieses *Packages* auszuführen, bietet uns die Komponente mehrere Methoden:

- *CallBooleanFunction* – Aufruf einer Funktion mit einem boolschen Rückgabewert
- *CallDateFunction* – Aufruf einer Funktion mit einem Rückgabewert vom Typ *DATE*
- *CallFloatFunction* – Aufruf einer Funktion mit einem Rückgabewert vom Typ *NUMBER*
- *CallIntegerFunction* – Aufruf einer Funktion mit einem Rückgabewert vom Typ *INTEGER*
- *CallStringFunction* – Aufruf einer Funktion mit einem Rückgabewert *VARCHAR*
- *CallProcedure* – Aufruf einer *Procedure*

Beim Aufruf dieser Methoden werden die Parameter als *Array* vom Typ *Variant* übergeben.

```
with DbmsPipe do  
try  
  Status := CallIntegerFunction('receive_message', ['pipename', 'demo_pipe', 'timeout', 60]);  
  case Status of  
    0: AddMessageToMemo;  
    1: ShowMessage('Timeout');  
    2: ShowMessage('Record in pipe too big for buffer');  
    3: ShowMessage('Interrupted');  
  end;  
except  
  on E:EOracleError do ShowMessage(E.Message);  
end;
```

In unserem Beispiel wurde das *Property ParameterMode* der Komponente *OraclePackage* auf *pmNamed* eingestellt. Alternativ dazu kann die Übergabe der Variablen mit *Positionsbezug* (*pmPositional*) in der Deklaration der *Procedure* erfolgen.

Alternativ zum doch recht komplizierten Aufruf via *OraclePackage* wird uns eine für meine Begriffe bessere und einfachere Möglichkeit angeboten. Nach der Installation der *Komponenten-Bibliothek* wird in unserer *Delphi – DIE* ein Menüpunkt *ORACLE* eingefügt. Auf die meisten der dort versteckten *Tools* werden wir später noch eingehen. An dieser Stelle interessiert uns der *Package-Wizard*. Mit Hilfe dieses *Tools* wird uns eine *Delphi-Klasse* erzeugt, welche ein *Package* und dessen Methoden für uns recht einfach kapselt. Wenn wir den *Package-Wizard* aufgerufen und uns (nochmals) an der *Datenbank* angemeldet haben, können wir das *Package* auswählen, wofür wir unsere Klasse erzeugen wollen.

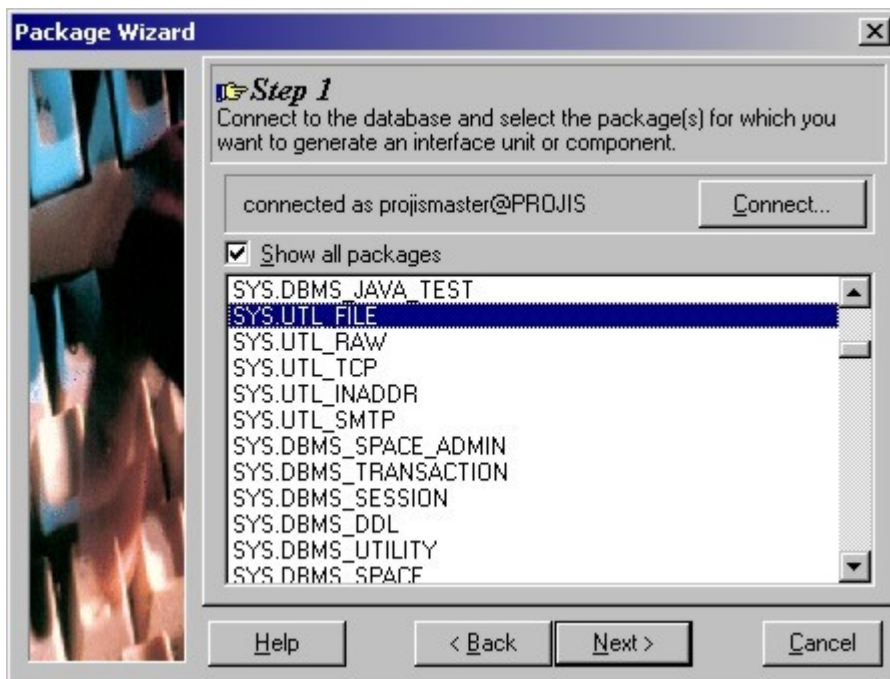


Abb. 2 Package Wizard

In den nächsten Schritten können wir noch einstellen, welche Methoden des Packages verarbeitet werden sollen bzw. wie die Schreibweise usw. erfolgen soll. Im letzten Schritt bei der Abarbeitung des Package-Wizard können wir festlegen ob „nur“ eine „reine“ Klasse für das Package oder eine richtige Komponente erzeugt werden soll. Ob Komponente oder nicht kann an dieser Stelle jeder selbst zweckgebunden entscheiden, aber ich denke, mit Komponenten hat man immer den besseren Überblick. Zum Abschluss muß jetzt nur noch der Code-Generator angeworfen werden. Das Ergebnis für das Standard-Package UTL_FILE sieht wie folgt aus:

```

unit UTLFILE;
interface
uses Classes, SysUtils, Oracle;

type
  UTLFILEfiletype = class(TPLSQLRecord)
  public
    ID: Integer;
    procedure Assign(Source: TPLSQLRecord); override;
  end;

  TUTLFILE = class(TOracleCustomPackage)
  public
    procedure FCLOSE(var FILE1: UTLFILEfiletype);
    function FOPEN(const LOCATION: string; const FILENAME: string;
      const OPENMODE: string): UTLFILEfiletype; overload;
  ..
  published
    property Name;
    property Session;
    property Cursor;
  end;

var DefaultPLSQLTableSize: Integer = 100; // Default size of a PL/SQL Table

procedure Register;

implementation

procedure UTLFILEfiletype.Assign(Source: TPLSQLRecord);

```

```

begin
  inherited;
  with Source as UTLFILEfiletype do
  begin
    Self.ID := ID;
  end;
end;

// UTL_FILE.FOPEN, overload 2
function TUTLFILE.FOPEN(const LOCATION: string; const FILENAME: string;
  const OPENMODE: string; MAXLINESIZE: Integer): UTLFILEfiletype;
begin
  Result := UTLFILEfiletype.Create(Session);
  ThreadAcquire;
  try
    GetQuery;
    OCPQuery.DeclareVariable('record_var2', otInteger);
    OCPQuery.DeclareVariable('LOCATION', otString);
    OCPQuery.SetVariable('LOCATION', LOCATION);
    OCPQuery.DeclareVariable('FILENAME', otString);
    OCPQuery.SetVariable('FILENAME', FILENAME);
    OCPQuery.DeclareVariable('OPEN_MODE', otString);
    OCPQuery.SetVariable('OPEN_MODE', OPENMODE);
    OCPQuery.DeclareVariable('MAX_LINESIZE', otInteger);
    OCPQuery.SetVariable('MAX_LINESIZE', MAXLINESIZE);
    OCPQuery.SQL.Add('declare');
    OCPQuery.SQL.Add(' function_result "UTL_FILE".file_type;');
    OCPQuery.SQL.Add('begin');
    OCPQuery.SQL.Add(' function_result := "UTL_FILE"."FOPEN"());
    OCPQuery.SQL.Add(' LOCATION => :LOCATION,');
    OCPQuery.SQL.Add(' FILENAME => :FILENAME,');
    OCPQuery.SQL.Add(' OPEN_MODE => :OPEN_MODE,');
    OCPQuery.SQL.Add(' MAX_LINESIZE => :MAX_LINESIZE;');
    OCPQuery.SQL.Add(' :record_var2 := function_result.ID;');
    OCPQuery.SQL.Add('end;');
    OCPQuery.Execute;
    Result.ID := ConvertVariant(OCPQuery.GetVariable('record_var2'));
  except
    ThreadRelease;
    Result.Free;
    raise;
  end;
  ThreadRelease;
end;

// UTL_FILE.FCLOSE
procedure TUTLFILE.FCLOSE(var FILE1: UTLFILEfiletype);
begin
  ThreadAcquire;
  try
    GetQuery;
    OCPQuery.DeclareVariable('record_var2', otInteger);
    OCPQuery.SetVariable('record_var2', FILE1.ID);
    OCPQuery.SQL.Add('declare');
    OCPQuery.SQL.Add(' FILE "UTL_FILE".file_type;');
    OCPQuery.SQL.Add('begin');
    OCPQuery.SQL.Add(' FILE.ID := :record_var2;');
    OCPQuery.SQL.Add(' "UTL_FILE"."FCLOSE" (FILE => FILE);');
    OCPQuery.SQL.Add(' :record_var2 := FILE.ID;');
  
```

```

    OCPQuery.SQL.Add('end;');
    OCPQuery.Execute;
    FILE1.ID := ConvertVariant(OCPQuery.GetVariable('record_var2'));
  finally
    ThreadRelease;
  end;
end;
..

procedure Register;
begin
  RegisterComponents('Oracle-Packages', [TUTLFILE]);
end;
end.

```

Über die erzeugten Methoden der Komponente kann jetzt die Methode eines Packages sehr einfach aufgerufen werden. Nutzt man diese Komponente, darf man natürlich nicht das Property Packages vergessen.

Exception-Handling

Für das Exceptionhandling ist in DOA die Klasse *EOracleError* implementiert. Die Variable *ErrorCode* enthält dabei die originale Fehlernummer des Datenbankservers und in der Variable *Message* den Fehlermeldungstext des Servers. Ein Exceptionhandling ist also prinzipiell wie folgt zu implementieren:

```

try
  OracleQuery.Execute; // insert , update , delete ..
except
  on E:EOracleError do begin
    // Behandlung einer speziellen Fehlernummer
    if E.ErrorCode = 942 then // ORA-00942: Table or view does not exist
      ShowMessage('Tabelle existiert nicht');
    else
      // Alle anderen Fehler -> Original Oracle-Meldung
      ShowMessage(E.Message);
    end;
  end;
end;

```

Weitere Komponenten

Die Komponente *OracleEvent* erlaubt es der Anwendung, Events der Datenbank entgegen zu nehmen und zu verarbeiten. Die Verarbeitung kann synchron oder asynchron zum aktuellen Haupt-Prozess der Anwendung erfolgen.

Zur einfachen Verarbeitung von SQL-Scripten wird die Komponente *OracleScript* zur Verfügung gestellt. Diese kann in eine String-Liste ein Script aufnehmen und gegen einen Oracle-Server abschicken.

Oft gestaltet sich der Import von Massendaten problematisch. Entweder man kennt sich mit Oracle's SQL-Loader aus oder man macht x*1000 Inserts und Updates, was sehr langwierig werden kann. DOA stellt hierfür die Komponente *SQLDirectPathImport* zur Verfügung. Diese Komponente erlaubt es, Daten nach der Methode des *DirectPath-Import's* des SQL-Loaders in eine Ziel-Tabelle zu übernehmen. Das beschleunigt den Import von Massendaten massiv.

All diesen Komponenten ist über das Property *Session* die Zielsession mitzuteilen.

InfoPower-DataSet

Bestandteil der DOA-Komponentensammlung ist auch eine *Infopower-kompatible DataSet-Komponente*. Diese ist manuell zu installieren. Hierzu öffnet man das Package und fügt die Unit *ORACLEwwData* hinzu und kompiliert das DOA-Package neu. Damit steht diese Komponente den Nutzern von *Infopower* zur Verfügung. Wer *Infopower* nicht installiert hat, kann diese Komponente nicht einbinden, da *Infopower-Units* vorausgesetzt werden.

Explorer, Monitor und Query-Builder

Als Beigaben erhalten DOA-Lizenznehmer drei weitere Tools, welche Ihnen das Programmieren erleichtern

soll. Diese sind, außer der Query Builder, über das bei der Installation in die Delphi-DIE eingefügte Menü Oracle zu erreichen.

Nutzer, welche nicht über die Enterprise-Version von Delphi oder C++-Builder verfügen, können nicht via SQL-Explorer auf Oracle-Datenbanken zugreifen. Dies würde sich im Alltag schnell als sehr hinderlich erweisen. Mit DOA erhält der Nutzer aus diesem Grunde den ORACLE-Explorer, welcher eine Adaption des SQL-Explorers für ORACLE darstellt.

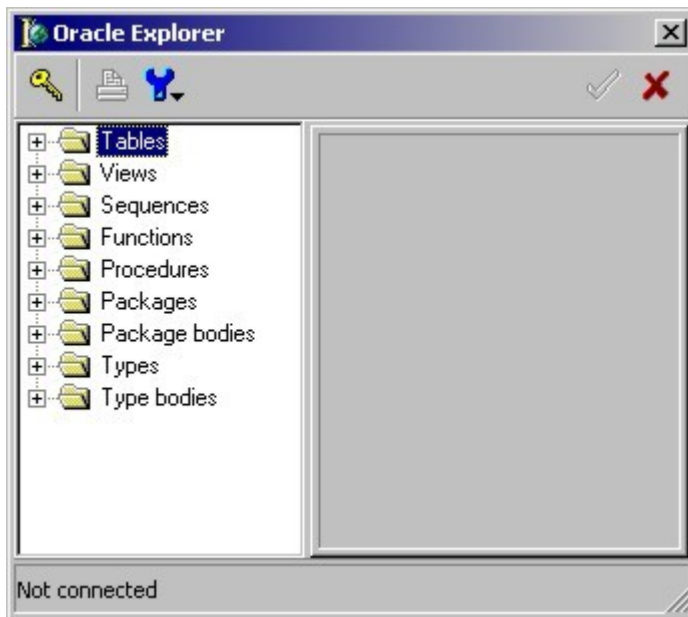


Abb. 3 DOA-Explorer

Eine weitere Beigabe, der Monitor, unterstützt uns in der Analyse unserer Anwendung. Er protokolliert alle an den Server übermittelten Aufrufe der OCI-Schnittstelle und alle SQL-Statements und zeigt diese an. In der unteren Bildschirm-Hälfte des SQL-Monitors können ebenfalls die aktuellen Werte von Parametern/Variablen betrachtet und analysiert werden. Um ein Programm über den SQL-Monitor „tracen“ zu können, muß im Code der Anwendung die Unit OracleMonitor eingebunden werden.

```
uses
  {$ifdef DEV_PHASE}
  OracleMonitor, //für DOA Oracle-Monitor-Tracing, in Release-Version entfernen
  {$endif}
  .....
```

Ob man sich diese Möglichkeit der Analyse von Problemen auch über die Entwicklungszeit hinaus verfügbar hält, indem man diese Unit nach Fertigstellung und Auslieferung der Software nicht entfernt, sollte sich jeder gut überlegen. In einer offenen Datenbankumgebung, wie es nun mal alle SQL-Datenbanken darstellen, bleibt der mit dem Monitor analysierte SQL-Code nie geheim. Entfernt man den Trace-Zugriff für den Monitor finden diesbezüglich ambitionierte Zeitgenossen mit viel Zeit andere Mittelchen. In der Praxis hat sich die Verfügbarkeit dieser schnellen verfügbaren Analyse-Möglichkeit auch in Produktionsumgebungen bewährt. Und daß unsere Anwendung ein paar K größer ist, sollte nun wirklich niemanden stören.

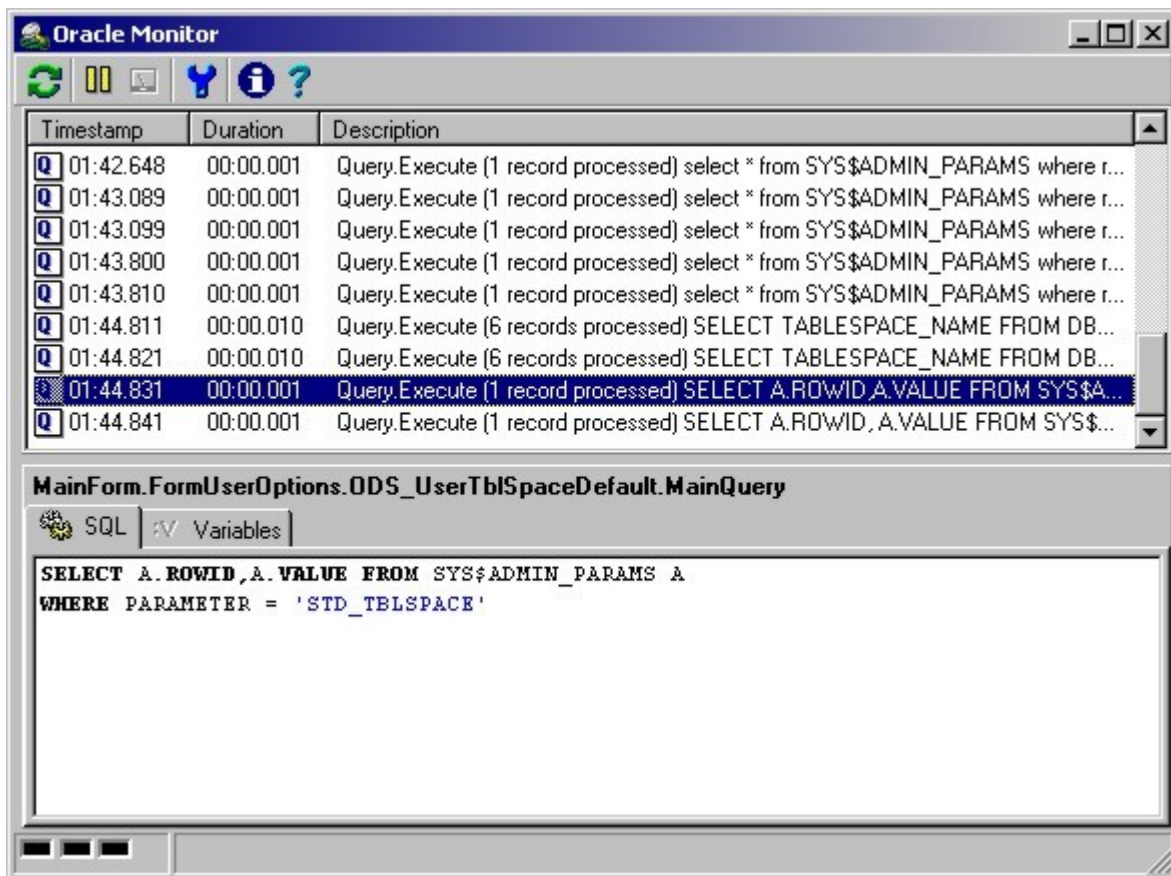


Abb. 4 DOA-Monitor

Als letztes Tool erhalten wir mit unserer Komponentenbibliothek den Query-Builder. Dieser ermöglicht das visuelle Erstellen von SQL-Statements. Dabei werden, wie bei Master-Detail-Verbindungen bei DataSets, in der Datenbank implementierte Verknüpfungen zwischen Tabellen erkannt und für unser SQL-Statement verfügbar gemacht. Um diesen zu nutzen, muss man in Design-Phase auf ein Datenzugriffsobjekt, also z.B. ein TOracleDataSet klicken, und kann jetzt über die rechte Maustaste und die Wahl des Menüpunktes „Query Builder“ das Tool aufrufen. Auch wenn die erstellten SQL-Statements, wie allgemein bei „generiertem“ - SQL, nicht optimal sind, kann über dieses Tool ein guter Vorentwurf komplexer SQL-Anweisungen erstellt werden. Wir können es ja dann im Nachgang verbessern und optimieren.

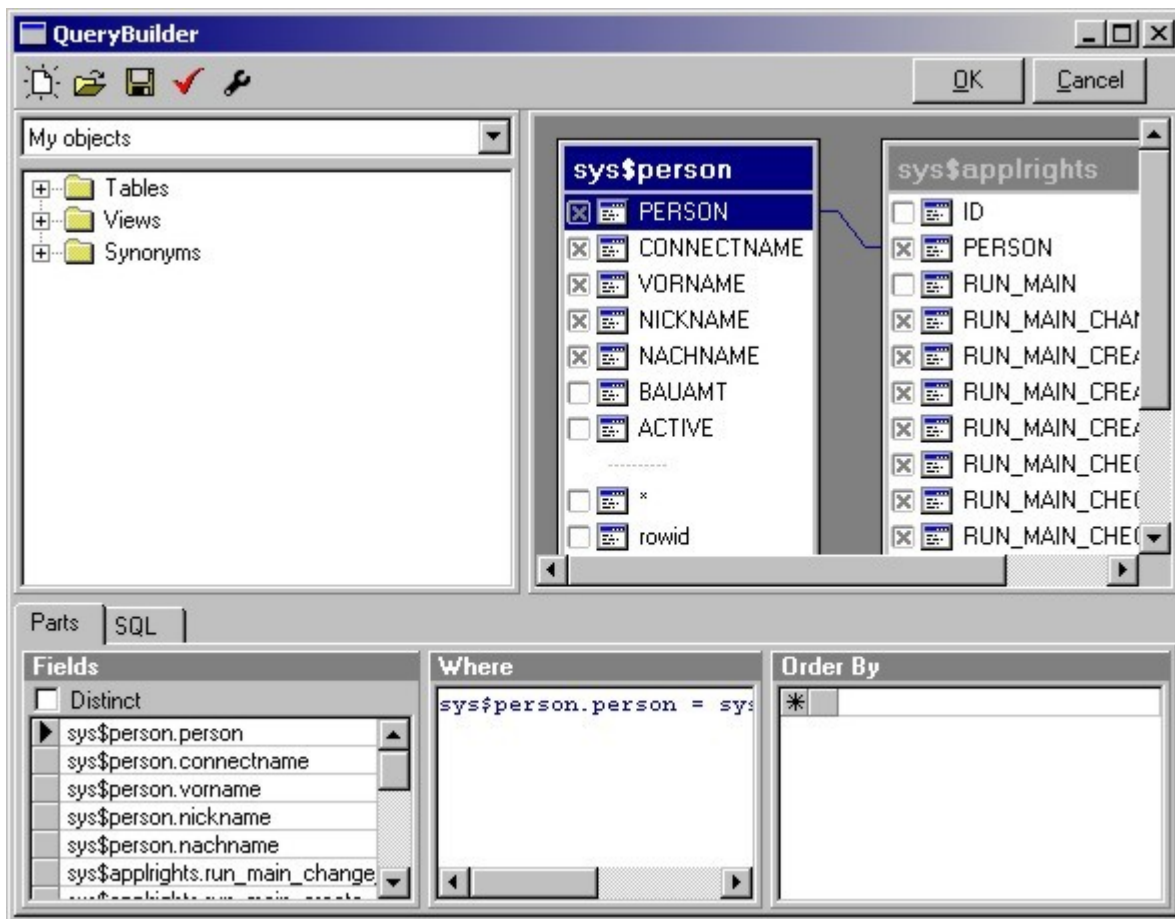


Abb. 5 DOA Query Builder

Diese hier aufgeführten Tools müssen teilweise separat heruntergeladen und installiert werden.

Was braucht man noch?

In einem ORACLE-basierenden Anwendungssystem spielt in der Datenbank gespeicherter Programmcode wie Stored Procedure, Stored Functions und Packages eine sehr große Rolle. Also benötigen wir ein Tool, mit dem wir solche Programmmodule erstellen und debuggen können. Suchen wir dabei nach Werkzeugen in unseren standardmäßig von ORACLE gelieferten Toolpaket, sehen wir schnell, daß sich da nichts Gescheites findet. Man muß sich in diesem Falle mit den Entwicklungstools von ORACLE, dem ORACLE - Developer- Paket, auseinandersetzen. Es gibt jedoch kostengünstige, „leichtere“ und sehr leistungsfähige Alternativen.

Beschränken wir uns bei der Tool-Suche auf ein Tool zum Erstellen und Pflegen von DB-gespeichertem Programmcode, liefert der Hersteller von DOA mit dem PL/SQL-Developer ein gute und leistungsfähige Alternative. Die Erfahrung lehrt aber, daß Probleme bei der Entwicklung von Anwendungen vielschichtig sind, und man neben der reinen Code-Erstellung und Pflege auch serverseitige Analyse-Tools benötigt. Schön wäre auch ein Tool, was einen grafischen Entwurf einer Datenbank ermöglicht. Wir suchen also eine eierlegende Wollmilchsau (und die gibt es bekanntlich nicht oder sehr selten) oder wir begeben uns als Jäger und Sammler auf die Suche und erwerben für jeden „Tatbestand“ ein entsprechendes Tool. Man kann natürlich auch verzichten, aber das kann teuer werden, denn bekanntlich wird eine Anwendung nicht nur einmal erstellt, sondern auch gewartet und fortentwickelt. Der Markt bietet uns zwar keine eierlegenden Wollmilchsäue, aber das eine oder andere Tool, welche das Meiste, was wir benötigen, integriert. Besonders zu empfehlen ist hierbei das Produkt KeepTool. In diesem sind Tools zum grafischen Design der Datenbank, der Erstellung und Pflege der meisten Objekte eines Anwendungsschemas bis hin zur Analyse von serverseitigen Problemen integriert. Wie gesagt, keine eierlegende ... aber ein Tool das nicht nur dem Entwickler das Leben sehr erleichtern kann und sich schon vielfach bewährt hat.

Mit der Komponenten-Bibliothek DOA und einem guten Tool kann man als Delphi- und C++-Entwickler effizient Anwendungssysteme auf der Basis von ORACLE erstellen und pflegen. Problemlos wird es wie immer nicht sein, aber wir werden für die meisten Probleme einen Lösungsansatz finden können. Und sollte es mal nicht so schnell gehen, bekanntermaßen ist das Leben halt ein Auswärtsspiel. Man wird, wie immer einen Work-around finden - denn kein Programm ist auch keine Lösung.

Bezugsquellen DOA

<http://www.allroundautomations.nl>

<http://www.sdctec.net>

Bezugsquellen KeepTool

<http://www.keeptool.com>

<http://www.sdctec.net>

Fortbildungsangebote zu den Themen

KeepTool GmbH Berlin

<http://www.keeptool.com>

Tel.: 0700 5337 8665

Firma SDCTec GmbH Griesheim

<http://www.sdctec.net>

Tel.: 061 55 831 770