

Delphi/Lazarus Database Access

Content:

1. [Introduction](#)
2. [SQLDB/dbExpress](#)
3. [Zeos](#)
4. [Calling stored procedures](#)

Introduction:

There are several options for database access in Lazarus and Delphi:

- Builtin SQLDB (Lazarus) / dbExpress (Delphi)
- External Zeos library
- External PDO (Pascal Data Objects) library

SQLDB/dbExpress:

Lazarus SQLDB and Delphi dbExpress are two mostly compatible database libraries that comes with respectively Lazarus and Delphi.

It has drivers for many databases including:

- Borland Interbase/Firebird
- MS SQLServer
- MySQL
- Oracle DB
- PostgreSQL
- Sybase
- SQLite
- ODBC (and thereby for all databases that has an ODBC driver)

The examples are only tested with Lazarus SQLDB. The examples should work with Delphi dbExpress with very minor changes, but I have no way to test.

For the test code below on Windows:

MySQL requires libmysql.dll in PATH

MS SQLServer requires dblib.dll and libiconv.dll in PATH

PostgreSQL requires libpq.dll, libiconv-2.dll and libintl-8.dll in PATH

ODBC requires ODBC driver

Procedural style:

```
program SQLDBPP;

uses
  SQLDB, MySQL55Conn, MSSQLConn, PQConnection, ODBConn, SysUtils;

const
  MAXREC = 100;
```

```

type
  t1 = record
    f1 : integer;
    f2 : string;
  end;
  array_of_t1 = array[1..MAXREC] of t1;

function t1_get_one(con : TSQLConnection; f2 : string) : integer;

var
  q : TSQLQuery;
  res : integer;

begin
  q := TSQLQuery.Create(nil);
  q.DataBase := con;
  q.SQL.Text := 'SELECT f1 FROM t1 WHERE f2 = :f2';
  q.Params.ParamByName('f2').AsString := f2;
  q.Open;
  if not q.EOF then begin
    res := q.FieldByName('f1').AsInteger;
  end else begin
    writeln(f2, ' not found');
    halt;
  end;
  q.Close;
  q.Free;
  t1_get_one := res;
end;

function t1_get_all(con : TSQLConnection; var buf : array_of_t1) : integer;

var
  q : TSQLQuery;
  count : integer;

begin
  count := 0;
  q := TSQLQuery.Create(nil);
  q.DataBase := con;
  q.SQL.Text := 'SELECT f1,f2 FROM t1';
  q.Open;
  while (not q.EOF) and (count < MAXREC) do begin
    count := count + 1;
    buf[count].f1 := q.FieldByName('f1').AsInteger;
    buf[count].f2 := q.FieldByName('f2').AsString;
    q.Next;
  end;
  q.Close;
  q.Free;
  t1_get_all := count;
end;

procedure t1_put(con : TSQLConnection; f1 : integer; f2 : string);

var
  q : TSQLQuery;

begin
  q := TSQLQuery.Create(nil);
  q.DataBase := con;
  q.SQL.Text := 'INSERT INTO t1 VALUES(:f1,:f2)';
  q.Params.ParamByName('f1').AsInteger := f1;
  q.Params.ParamByName('f2').AsString := f2;
  q.ExecSQL;
  if q.RowsAffected <> 1 then begin
    writeln('INSERT did not insert 1 row');
  end;
end;

```

```

    halt;
end;
q.Close;
q.Free;
end;

procedure t1_remove(con : TSQLConnection; f1 : integer);

var
    q : TSQLQuery;

begin
    q := TSQLQuery.Create(nil);
    q.DataBase := con;
    q.SQL.Text := 'DELETE FROM t1 WHERE f1 = :f1';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('DELETE did not delete 1 row');
        halt;
    end;
    q.Close;
    q.Free;
end;

procedure t1_dump(con : TSQLConnection);

var
    buf : array_of_t1;
    count, i : integer;

begin
    count := t1_get_all(con, buf);
    for i := 1 to count do begin
        writeln(buf[i].f1, ' ', buf[i].f2);
    end;
end;

procedure test(lbl : string; con : TSQLConnection);

var
    tx : TSQLTransaction;
    f1 : integer;

begin
    writeln(lbl, ':');
    con.Open;
    tx := TSQLTransaction.Create(nil);
    con.Transaction := tx;
    f1 := t1_get_one(con, 'BB');
    writeln(f1);
    t1_dump(con);
    t1_put(con, 999, 'XXX');
    tx.Commit;
    t1_dump(con);
    t1_remove(con, 999);
    tx.Commit;
    t1_dump(con);
    tx.Free;
    con.Close;
    con.Free;
end;

procedure test_odbc_dsn(lbl, dsn : string);

var
    odbcccon : TODBCCConnection;

```

```

begin
    odbcccon := TODBCCConnection.Create(nil);
    odbcccon.DatabaseName := dsn;
    test(lbl, odbcccon);
end;

procedure test_odbc_dsnless(lbl, driver, host, usr, pw, db: string);

var
    odbcccon : TODBCCConnection;

begin
    odbcccon := TODBCCConnection.Create(nil);
    odbcccon.Driver := driver;
    if host <> '' then odbcccon.Params.AddStrings(host);
    if usr <> '' then odbcccon.Params.AddStrings(usr);
    if pw <> '' then odbcccon.Params.AddStrings(pw);
    if db <> '' then odbcccon.Params.AddStrings(db);
    test(lbl, odbcccon);
end;

var
    mysqlcon : TMySQL55Connection;
    mssqlcon : TMSSQLConnection;
    pgsqlcon : TPQConnection;

begin
    (* MySQL with client lib version 5.5 *)
    mysqlcon := TMySQL55Connection.Create(nil);
    mysqlcon.HostName := 'localhost';
    mysqlcon.UserName := 'root';
    mysqlcon.Password := '';
    mysqlcon.DatabaseName := 'Test';
    test('MySQL', mysqlcon);
    (* MS SQLServer with dblib client lib *)
    mssqlcon := TMSSQLConnection.Create(nil);
    mssqlcon.HostName := 'ARNEPC4';
    mssqlcon.DatabaseName := 'Test';
    test('MS SQLServer', mssqlcon);
    (* PostgreSQL *)
    pgsqlcon := TPQConnection.Create(nil);
    pgsqlcon.HostName := 'localhost';
    pgsqlcon.UserName := 'postgres';
    pgsqlcon.Password := 'xxxxxx';
    pgsqlcon.DatabaseName := 'Test';
    test('PostgreSQL', pgsqlcon);
    (* ODBC with DSN *)
    test_odbc_dsn('ODBC - MySQL', 'ARNEPC4_MYSQL');
    test_odbc_dsn('ODBC - MS SQLServer', 'ARNEPC4_SQLSRV');
    test_odbc_dsn('ODBC - PostgreSQL', 'ARNEPC4_PGSQL');
    test_odbc_dsn('ODBC - DB2', 'ARNEPC4_DB2');
    (* ODBC without DSN *)
    test_odbc_dsnless('ODBC - MySQL', 'MySQL ODBC 5.1 Driver', 'Server=localhost', 'User=r
    test_odbc_dsnless('ODBC - MS SQLServer', 'SQL Server Native Client 11.0', 'Server=loca
    test_odbc_dsnless('ODBC - PostgreSQL', 'PostgreSQL ANSI(x64)', 'Server=localhost', 'Ui
    test_odbc_dsnless('ODBC - DB2', 'IBM DB2 ODBC DRIVER', 'Hostname=localhost', 'Uid=arne
end.

```

Object oriented style:

```

program SQLDB00P;

uses

```

Classes, SQLDB, MySQL55Conn, MSSQLConn, PQConnection, ODBCConn, SysUtils;

```
type
  T1 = class(TObject)
  private
    m_f1 : integer;
    m_f2 : string;
  public
    constructor Create;
    constructor Create(f1 : integer; f2 : string);
    property F1 : integer read m_f1;
    property F2 : string read m_f2;
    procedure Free;
  end;
  ListT1 = class(TObject)
  private
    m_list : TList;
  public
    constructor Create;
    procedure Add(f1 : integer; f2 : string);
    function Count : integer;
    function GetElement(ix : integer) : T1;
    procedure Free;
  end;
  IT1DAL = interface(IIInterface)
    function GetOne(f2 : string) : integer;
    function GetAll : ListT1;
    procedure Put(f1 : integer; f2 : string);
    procedure Remove(f1 : integer);
    procedure Free;
  end;
  T1DAL = class (TInterfacedObject, IT1DAL)
  private
    m_con : TSQLConnection;
    m_tx : TSQLTransaction;
  public
    constructor Create(con : TSQLConnection);
    function GetOne(f2 : string) : integer;
    function GetAll : ListT1;
    procedure Put(f1 : integer; f2 : string);
    procedure Remove(f1 : integer);
    procedure Free;
  end;

constructor T1.Create;

begin
  m_f1 := 0;
  m_f2 := '';
end;

constructor T1.Create(f1 : integer; f2 : string);

begin
  m_f1 := f1;
  m_f2 := f2;
end;

procedure T1.Free;

begin
end;

constructor ListT1.Create;

begin
  m_list := TList.Create;
```

```

end;

procedure ListT1.Add(f1 : integer; f2 : string);
begin
    m_list.Add(T1.Create(f1, f2));
end;

function ListT1.Count : integer;
begin
    Count := m_list.Count;
end;

function ListT1.GetElement(ix : integer) : T1;
begin
    GetElement := T1(m_list[ix]);
end;

procedure ListT1.Free;
var
    i : integer;
begin
    for i := 0 to m_list.Count - 1 do begin
        T1(m_list[i]).Free;
    end;
    m_list.Free;
end;

constructor T1DAL.Create(con : TSQLConnection);
begin
    m_con := con;
    m_con.Open;
    m_tx := TSQLTransaction.Create(nil);
    m_con.Transaction := m_tx;
end;

function T1DAL.GetOne(f2 : string) : integer;
var
    q : TSQLQuery;
    res : integer;
begin
    q := TSQLQuery.Create(nil);
    q.DataBase := m_con;
    q.SQL.Text := 'SELECT f1 FROM t1 WHERE f2 = :f2';
    q.Params.ParamByName('f2').AsString := f2;
    q.Open;
    if not q.EOF then begin
        res := q.FieldByName('f1').AsInteger;
    end else begin
        writeln(f2, ' not found');
        halt;
    end;
    q.Close;
    q.Free;
    GetOne := res;
end;

function T1DAL.GetAll : ListT1;
var

```

```

    q : TSQLQuery;
    res : ListT1;

begin
    res := ListT1.Create;
    q := TSQLQuery.Create(nil);
    q.DataBase := m_con;
    q.SQL.Text := 'SELECT f1,f2 FROM t1';
    q.Open;
    while not q.EOF do begin
        res.Add(q.FieldName('f1').AsInteger, q.FieldName('f2').AsString);
        q.Next;
    end;
    q.Close;
    q.Free;
    GetAll := res;
end;

procedure T1DAL.Put(f1 : integer; f2 : string);

var
    q : TSQLQuery;

begin
    q := TSQLQuery.Create(nil);
    q.DataBase := m_con;
    q.SQL.Text := 'INSERT INTO t1 VALUES(:f1,:f2)';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.Params.ParamByName('f2').AsString := f2;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('INSERT did not insert 1 row');
        halt;
    end;
    q.Close;
    q.Free;
    m_tx.Commit;
end;

procedure T1DAL.Remove(f1 : integer);

var
    q : TSQLQuery;

begin
    q := TSQLQuery.Create(nil);
    q.DataBase := m_con;
    q.SQL.Text := 'DELETE FROM t1 WHERE f1 = :f1';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('DELETE did not delete 1 row');
        halt;
    end;
    q.Close;
    q.Free;
    m_tx.Commit;
end;

procedure T1DAL.Free;

begin
    m_tx.Free;
    m_con.Close;
    m_con.Free;
end;

```

```

procedure t1_dump(dal : IT1DAL);

var
    all : ListT1;
    i : integer;

begin
    all := dal.GetAll;
    for i := 0 to all.Count - 1 do begin
        writeln(all.GetElement(i).F1, ' ', all.GetElement(i).F2);
    end;
    all.Free;
end;

procedure test(lbl : string; dal : IT1DAL);

var
    f1 : integer;

begin
    writeln(lbl, ':');
    f1 := dal.GetOne('BB');
    writeln(f1);
    t1_dump(dal);
    dal.Put(999, 'XXX');
    t1_dump(dal);
    dal.Remove(999);
    t1_dump(dal);
    dal.Free;
end;

procedure test_odbc_dsn(lbl, dsn : string);

var
    odbcccon : TODBCCConnection;

begin
    odbcccon := TODBCCConnection.Create(nil);
    odbcccon.DatabaseName := dsn;
    test(lbl, T1DAL.Create(odbcccon));
end;

procedure test_odbc_dsnless(lbl, driver, host, usr, pw, db: string);

var
    odbcccon : TODBCCConnection;

begin
    odbcccon := TODBCCConnection.Create(nil);
    odbcccon.Driver := driver;
    if host <> '' then odbcccon.Params.AddStrings(host);
    if usr <> '' then odbcccon.Params.AddStrings(usr);
    if pw <> '' then odbcccon.Params.AddStrings(pw);
    if db <> '' then odbcccon.Params.AddStrings(db);
    test(lbl, T1DAL.Create(odbcccon));
end;

var
    mysqlcon : TMySQL55Connection;
    mssqlcon : TMSSQLConnection;
    pgsqlcon : TPQConnection;

begin
    (* MySQL with client lib version 5.5 *)
    mysqlcon := TMySQL55Connection.Create(nil);
    mysqlcon.HostName := 'localhost';
    mysqlcon.UserName := 'root';

```



```

mysqlcon.Password := '';
mysqlcon.DatabaseName := 'Test';
test('MySQL', T1DAL.Create(mysqlcon));
(* MS SQLServer with dblib client lib *)
mssqlcon := TMSSQLConnection.Create(nil);
mssqlcon.HostName := 'ARNEPC4';
mssqlcon.DatabaseName := 'Test';
test('MS SQLServer', T1DAL.Create(mssqlcon));
(* PostgreSQL *)
pgsqlcon := TPQConnection.Create(nil);
pgsqlcon.HostName := 'localhost';
pgsqlcon.UserName := 'postgres';
pgsqlcon.Password := 'xxxxxx';
pgsqlcon.DatabaseName := 'Test';
test('PostgreSQL', T1DAL.Create(pgsqlcon));
(* ODBC with DSN *)
test_odbc_dsn('ODBC - MySQL', 'ARNEPC4_MYSQL');
test_odbc_dsn('ODBC - MS SQLServer', 'ARNEPC4_SQLSRV');
test_odbc_dsn('ODBC - PostgreSQL', 'ARNEPC4_PGSQL');
test_odbc_dsn('ODBC - DB2', 'ARNEPC4_DB2');
(* ODBC without DSN *)
test_odbc_dsnless('ODBC - MySQL', 'MySQL ODBC 5.1 Driver', 'Server=localhost', 'User=r
test_odbc_dsnless('ODBC - MS SQLServer', 'SQL Server Native Client 11.0', 'Server=loca
test_odbc_dsnless('ODBC - PostgreSQL', 'PostgreSQL ANSI(x64)', 'Server=localhost', 'Ui
test_odbc_dsnless('ODBC - DB2', 'IBM DB2 ODBC DRIVER', 'Hostname=localhost', 'Uid=arne
end.

```

Zeos:

ZeosLib is an open source database library for both Lazarus and Delphi.

It can be downloaded from [here](#).

It has drivers for many databases including:

- MySQL
- PostgreSQL
- Borland Interbase
- Firebird
- Sybase
- MS SQLServer
- ADO (and thereby for all databases that has an ADO provider)

For the test code below on Windows:

MySQL requires libmysql.dll in PATH

PostgreSQL requires libpq.dll, libiconv-2.dll and libintl-8.dll in PATH

ADO requires OLE DB provider (or ODBC driver)

Procedural style:

```

program ZeosPP;

uses
  ZConnection, ZDataSet;

const
  MAXREC = 100;

```

```

type
  t1 = record
      f1 : integer;
      f2 : string;
  end;
  array_of_t1 = array[1..MAXREC] of t1;

function t1_get_one(con : TZConnection; f2 : string) : integer;

var
  q : TZQuery;
  res : integer;

begin
  q := TZQuery.Create(nil);
  q.Connection := con;
  q.SQL.Text := 'SELECT f1 FROM t1 WHERE f2 = :f2';
  q.Params.ParamByName('f2').AsString := f2;
  q.Open;
  if not q.EOF then begin
      res := q.FieldByName('f1').AsInteger;
  end else begin
      writeln(f2, ' not found');
      halt;
  end;
  q.Close;
  q.Free;
  t1_get_one := res;
end;

function t1_get_all(con : TZConnection; var buf : array_of_t1) : integer;

var
  q : TZQuery;
  count : integer;

begin
  count := 0;
  q := TZQuery.Create(nil);
  q.Connection := con;
  q.SQL.Text := 'SELECT f1,f2 FROM t1';
  q.Open;
  while (not q.EOF) and (count < MAXREC) do begin
      count := count + 1;
      buf[count].f1 := q.FieldByName('f1').AsInteger;
      buf[count].f2 := q.FieldByName('f2').AsString;
      q.Next;
  end;
  q.Close;
  q.Free;
  t1_get_all := count;
end;

procedure t1_put(con : TZConnection; f1 : integer; f2 : string);

var
  q : TZQuery;

begin
  q := TZQuery.Create(nil);
  q.Connection := con;
  q.SQL.Text := 'INSERT INTO t1 VALUES(:f1,:f2)';
  q.Params.ParamByName('f1').AsInteger := f1;
  q.Params.ParamByName('f2').AsString := f2;
  q.ExecSQL;
  if q.RowsAffected <> 1 then begin

```

```

        writeln('INSERT did not insert 1 row');
        halt;
    end;
    q.Close;
    q.Free;
end;

procedure t1_remove(con : TZConnection; f1 : integer);

var
    q : TZQuery;

begin
    q := TZQuery.Create(nil);
    q.Connection := con;
    q.SQL.Text := 'DELETE FROM t1 WHERE f1 = :f1';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('DELETE did not delete 1 row');
        halt;
    end;
    q.Close;
    q.Free;
end;

procedure t1_dump(con : TZConnection);

var
    buf : array_of_t1;
    count, i : integer;

begin
    count := t1_get_all(con, buf);
    for i := 1 to count do begin
        writeln(buf[i].f1, ' ', buf[i].f2);
    end;
end;

procedure test(lbl : string; con : TZConnection);

var
    f1 : integer;

begin
    writeln(lbl, ':');
    con.Connect;
    f1 := t1_get_one(con, 'BB');
    writeln(f1);
    t1_dump(con);
    t1_put(con, 999, 'XXX');
    t1_dump(con);
    t1_remove(con, 999);
    t1_dump(con);
    con.Disconnect;
    con.Free;
end;

procedure test_ado(lbl, constr : string);

var
    con : TZConnection;

begin
    con := TZConnection.Create(nil);
    con.Protocol := 'ado';
    con.Database := constr;

```

```

test(lbl, con);
end;

var
  con : TZConnection;

begin
  (* MySQL *)
  con := TZConnection.Create(nil);
  con.Protocol := 'mysql';
  con.HostName := 'localhost';
  con.User := 'root';
  con.Password := '';
  con.Database := 'Test';
  test('MySQL', con);
  (* PostgreSQL *)
  con := TZConnection.Create(nil);
  con.Protocol := 'postgresql';
  con.HostName := 'localhost';
  con.User := 'postgres';
  con.Password := 'xxxxxx';
  con.Database := 'Test';
  test('PostgreSQL', con);
  (* ADO *)
  test_ado('ADO - MS SQLServer', 'Provider=SQLNCLI11;Server=ARNEPC4;Database=Test;Truste
  test_ado('ADO - DB2', 'Provider=IBMDADB2;Protocol=TCPIP;Hostname=localhost;Database=Te
end.

```

Object oriented style:

```

program Zeos00P;

uses
  Classes, ZConnection, ZDataSet;

type
  T1 = class(TObject)
  private
    m_f1 : integer;
    m_f2 : string;
  public
    constructor Create;
    constructor Create(f1 : integer; f2 : string);
    property F1 : integer read m_f1;
    property F2 : string read m_f2;
    procedure Free;
  end;
  ListT1 = class(TObject)
  private
    m_list : TList;
  public
    constructor Create;
    procedure Add(f1 : integer; f2 : string);
    function Count : integer;
    function GetElement(ix : integer) : T1;
    procedure Free;
  end;
  IT1DAL = interface(IIInterface)
    function GetOne(f2 : string) : integer;
    function GetAll : ListT1;
    procedure Put(f1 : integer; f2 : string);
    procedure Remove(f1 : integer);
    procedure Free;
  end;

```

```

T1DAL = class (TInterfacedObject, IT1DAL)
private
    m_con : TZConnection;
public
    constructor Create(con : TZConnection);
    function GetOne(f2 : string) : integer;
    function GetAll : ListT1;
    procedure Put(f1 : integer; f2 : string);
    procedure Remove(f1 : integer);
    procedure Free;
end;

constructor T1.Create;

begin
    m_f1 := 0;
    m_f2 := '';
end;

constructor T1.Create(f1 : integer; f2 : string);

begin
    m_f1 := f1;
    m_f2 := f2;
end;

procedure T1.Free;

begin
end;

constructor ListT1.Create;

begin
    m_list := TList.Create;
end;

procedure ListT1.Add(f1 : integer; f2 : string);

begin
    m_list.Add(T1.Create(f1, f2));
end;

function ListT1.Count : integer;

begin
    Count := m_list.Count;
end;

function ListT1.GetElement(ix : integer) : T1;

begin
    GetElement := T1(m_list[ix]);
end;

procedure ListT1.Free;

var
    i : integer;

begin
    for i := 0 to m_list.Count - 1 do begin
        T1(m_list[i]).Free;
    end;
    m_list.Free;
end;

```

```

constructor T1DAL.Create(con : TZConnection);

begin
    m_con := con;
    m_con.Connect;
end;

function T1DAL.GetOne(f2 : string) : integer;

var
    q : TZQuery;
    res : integer;

begin
    q := TZQuery.Create(nil);
    q.Connection := m_con;
    q.SQL.Text := 'SELECT f1 FROM t1 WHERE f2 = :f2';
    q.Params.ParamByName('f2').AsString := f2;
    q.Open;
    if not q.EOF then begin
        res := q.FieldByName('f1').AsInteger;
    end else begin
        writeln(f2, ' not found');
        halt;
    end;
    q.Close;
    q.Free;
    GetOne := res;
end;

function T1DAL.GetAll : ListT1;

var
    q : TZQuery;
    res : ListT1;

begin
    res := ListT1.Create;
    q := TZQuery.Create(nil);
    q.Connection := m_con;
    q.SQL.Text := 'SELECT f1,f2 FROM t1';
    q.Open;
    while not q.EOF do begin
        res.Add(q.FieldByName('f1').AsInteger, q.FieldByName('f2').AsString);
        q.Next;
    end;
    q.Close;
    q.Free;
    GetAll := res;
end;

procedure T1DAL.Put(f1 : integer; f2 : string);

var
    q : TZQuery;

begin
    q := TZQuery.Create(nil);
    q.Connection := m_con;
    q.SQL.Text := 'INSERT INTO t1 VALUES(:f1,:f2)';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.Params.ParamByName('f2').AsString := f2;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('INSERT did not insert 1 row');
        halt;
    end;
end;

```

```

q.Close;
q.Free;
end;

procedure T1DAL.Remove(f1 : integer);

var
    q : TZQuery;

begin
    q := TZQuery.Create(nil);
    q.Connection := m_con;
    q.SQL.Text := 'DELETE FROM t1 WHERE f1 = :f1';
    q.Params.ParamByName('f1').AsInteger := f1;
    q.ExecSQL;
    if q.RowsAffected <> 1 then begin
        writeln('DELETE did not delete 1 row');
        halt;
    end;
    q.Close;
    q.Free;
end;

procedure T1DAL.Free;

begin
    m_con.Disconnect;
    m_con.Free;
end;

procedure t1_dump(dal : IT1DAL);

var
    all : ListT1;
    i : integer;

begin
    all := dal.GetAll;
    for i := 0 to all.Count - 1 do begin
        writeln(all.GetElement(i).F1, ' ', all.GetElement(i).F2);
    end;
    all.Free;
end;

procedure test(lbl : string; dal : IT1DAL);

var
    f1 : integer;

begin
    writeln(lbl, ':');
    f1 := dal.GetOne('BB');
    writeln(f1);
    t1_dump(dal);
    dal.Put(999, 'XXX');
    t1_dump(dal);
    dal.Remove(999);
    t1_dump(dal);
    dal.Free;
end;

procedure test_ado(lbl, constr : string);

var
    con : TZConnection;

```

```

begin
  con := TZConnection.Create(nil);
  con.Protocol := 'ado';
  con.Database := constr;
  test(lbl, T1DAL.Create(con));
end;

var
  con : TZConnection;

begin
  (* MySQL *)
  con := TZConnection.Create(nil);
  con.Protocol := 'mysql';
  con.HostName := 'localhost';
  con.User := 'root';
  con.Password := '';
  con.Database := 'Test';
  test('MySQL', T1DAL.Create(con));
  (* PostgreSQL *)
  con := TZConnection.Create(nil);
  con.Protocol := 'postgresql';
  con.HostName := 'localhost';
  con.User := 'postgres';
  con.Password := 'xxxxxx';
  con.Database := 'Test';
  test('PostgreSQL', T1DAL.Create(con));
  (* ADO *)
  test_ado('ADO - MS SQLServer', 'Provider=SQLNCLI11;Server=ARNEPC4;Database=Test;Truste
  test_ado('ADO - DB2', 'Provider=IBMDADB2;Protocol=TCPIP;Hostname=localhost;Database=Te
end.

```

Calling stored procedures:

The usage of stored procedures are not universally considered a good idea. But the reality is that they are sometimes used. Especially in Oracle DB and MS SQLServer environments.

The basic API for accessing stored procedures is the same as for using plain SQL statements with parameters, but stored procedures do come with a few special features that require special handling in the API.

Two such features are:

- Stored procedures returning more than one result set
- Store procedures having OUT parameters and return values

The following example illustrates how to handle that.

Stored procedures (for MS SQLServer):

```

CREATE PROCEDURE usp_multi @arg INTEGER
AS
BEGIN
  SELECT @arg+1 AS v
  SELECT 2*@arg AS v
END;
GO

```

```

CREATE PROCEDURE usp_return @inarg INTEGER, @outarg INTEGER OUT
AS

```



```

BEGIN
    SELECT @inarg+1 AS v
    SELECT @outarg = @inarg+2
    RETURN @inarg+3
END;
GO

```

Yes - they are trivial, but they will illustrate the points just fine.

Lazarus SQLDB does not support stored procedures even though Delphi DBExpress does.

So examples will be using Zeos.

Code:

```

program SP;

uses
    ZConnection, ZDataSet, ZStoredProcedure, DB;

procedure test_multi_result;

var
    con : TZConnection;
    usp : TZStoredProc;

begin
    con := TZConnection.Create(nil);
    con.Protocol := 'FreeTDS_MsSQL>=2005';
    con.HostName := 'ARNEPC4';
    con.Database := 'Test';
    con.LibraryLocation := 'sybdb.dll';
    con.Connect;
    usp := TZStoredProc.Create(nil);
    usp.Connection := con;
    usp.StoredProcName := 'dbo.usp_multi';
    usp.Params[1].AsInteger := 123;
    usp.Open; // note that ExecProc is only for when not returning result set
    usp.FirstResultSet;
    while not usp.EOF do begin
        writeln(usp.FieldName('v').AsInteger);
        usp.Next;
    end;
    usp.NextResultSet; // does not work in Zeos version 7.2.4 with FreeTDS (64 bit)
    while not usp.EOF do begin
        writeln(usp.FieldName('v').AsInteger);
        usp.Next;
    end;
    usp.Free;
    con.Disconnect;
    con.Free;
end;

procedure test_return_types;

var
    con : TZConnection;
    usp : TZStoredProc;

begin
    con := TZConnection.Create(nil);
    con.Protocol := 'FreeTDS_MsSQL>=2005';
    con.HostName := 'ARNEPC4';
    con.Database := 'Test';

```

```

con.LibraryLocation := 'sybdb.dll';
con.Connect;
usp := TZStoredProc.Create(nil);
usp.Connection := con;
usp.StoredProcName := 'dbo.usp_return';
usp.Params[1].AsInteger := 123;
usp.Open; // note that ExecProc is only for when not returning result set
usp.FirstResultSet;
while not usp.EOF do begin
    writeln(usp.FieldName('v').AsInteger);
    usp.Next;
end;
writeln('return value = ',usp.Params[0].AsInteger);
writeln('out parameter = ',usp.Params[2].AsInteger);
usp.Free;
con.Disconnect;
con.Free;
end;

begin
    test_multi_result;
    test_return_types;
end.

```

Article history:

Version	Date	Description
1.0	August 12th 2018	Initial version

Other articles:

See list of all articles [here](#)

Comments:

Please send comments to [Arne Vajhøj](#)