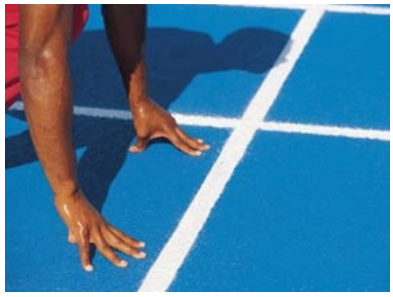
maXbox



maXbox Starter 32

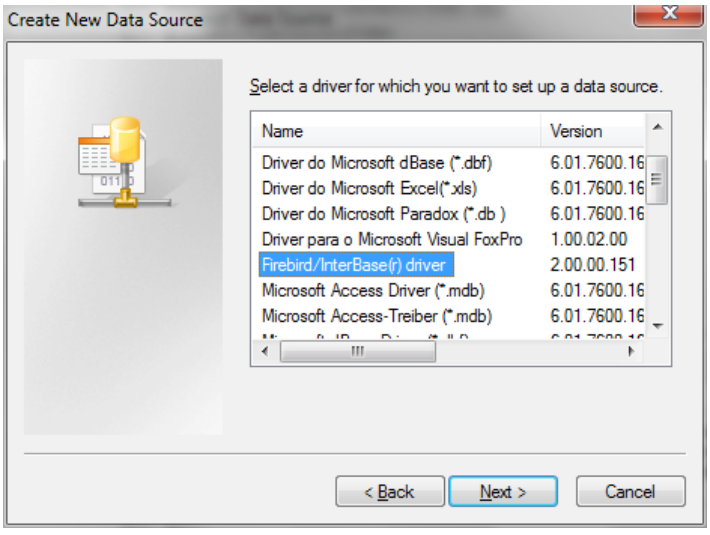
Start with Firebird SQL Server

1.1 DBExpress at runtime

Firebird is a relational database offering many ANSI SQL standard features that runs on Linux, Windows, and a variety of Unix platforms as Open Source. Although SQL language enhancements are not a primary objective of this tutor, for the first time we show how to query your database at runtime in a script.
Concerning SQL itself please read the Starter 12.

The normal way for Delphi or Kylix is just to check dbExpress icons, put a `TSQLConnection` on a form then double-click the `TSQLConnection` to display the Connection Editor and set parameter values (database path, connection name etc.) to indicate the settings.

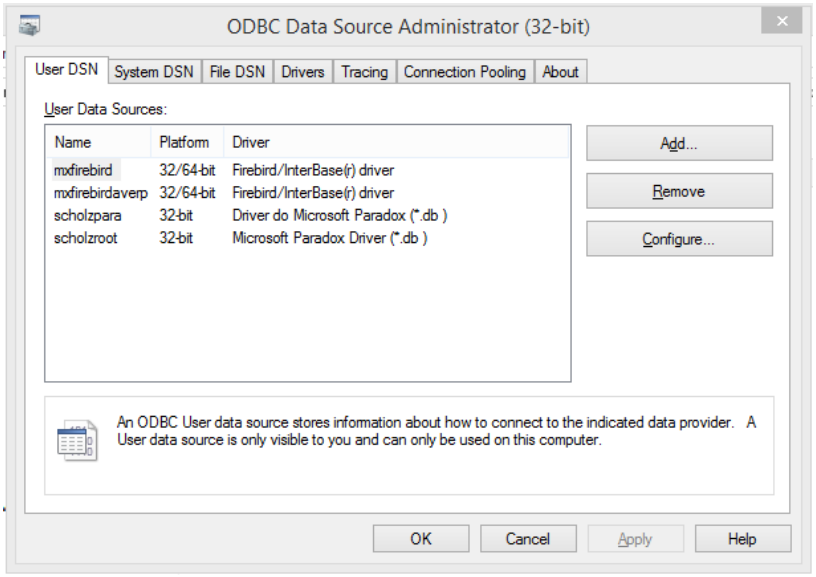
But in our example, all goes by runtime (path and login) with dbExpress we don't need an BDE alias or the BDE either. This is possible through ADO objects and a ODBC driver too.



But first you have to install Firebird:

http://www.firebirdsql.org/manual/qsg2-installing.html

Firebird server – and any databases you create or connect to – must reside on a hard drive that is physically connected to the host machine. You cannot locate components of the server, or any database, on a mapped drive, a file system share or a network file system.



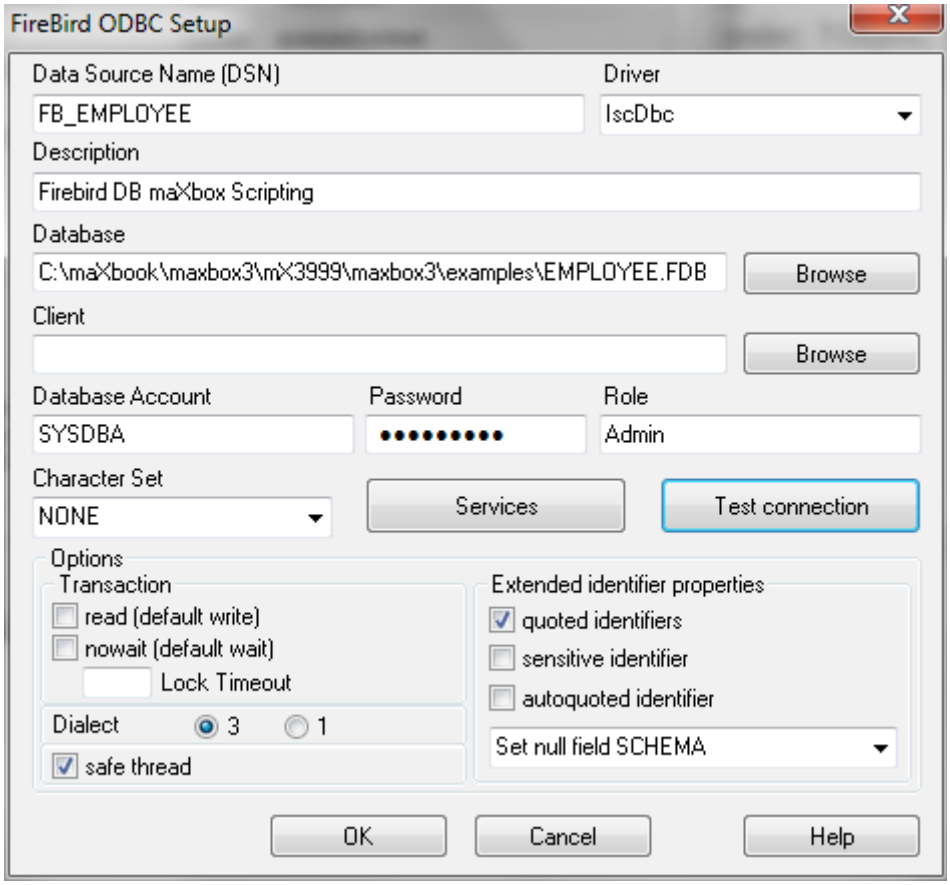
To find the ODBC Administrator go to a system panel or open maXbox and click on `Options/ADO SQL Workbench` and set a new connection string:

```
connectionString:=
  'Provider=MSDASQL.1;Persist Security Info=False;
                          Data Source=FB_EMPLOYEE';
```

This connection string sets the Provider and DSN properties to reflect the driver and connection parameters stored under that name in the ODBC Data Source Administrator or Workbench (see Appendix).
With a browser you see the database name so you can also set a path as data source name in the result set direct:

```
Provider=MSDASQL.1;Persist Security Info=False;Extended
Properties="DSN=FB_EMPLOYEE;Driver=Firebird/InterBase(r)
driver;Dbname=C:\maXbook\maxbox3\mX3999\maxbox3\examples\EMPLO
YEE.FDB;CHARSET=NONE;UID=SYSDBA;Role=Admin;"
```

A local DB is also possible as a client machine too. Each remote client machine needs to have the client library – `libfbclient.so` on Posix clients, `fbclient.dll` on Windows clients – that matches the release version of the Firebird server.
That's what you see next in the client field:



☝ At present, no separate installation program is available to install only the client pieces on a Windows machine. If you are in the common situation of running Windows clients to a Linux or other Unix-like Firebird server (or another Windows machine), you need to download the full Windows installation kit that corresponds to the version of Firebird server you install on your server machine.

Now lets start with the SQL Query Script.

1.1.1 Firebird Code Fly

Obviously the most important data required for this script to work is the database itself *EMPLOYEE.FDB,* which you find after the installation.
In our example we just use the table *Customer.*

Test now the script with **F9** / F2 or press Compile. So far now we'll open the example:

```
268_DBGrid_treeFirebird.TXT
```

http://www.softwareschule.ch/examples/

This database example requires 3 objects of the classes: `TAdoQuery`, `TDataSource`, `TDBGrid` and 3 objects dataset of the SQL components. `TDataSet` is the ancestor for all the dataset objects that you use in your applications and I show all 3 ways.

It defines a set of data fields, properties, events and methods that are shared by all dataset objects.

`TDataSet` is a virtualized dataset, meaning that many of its properties and methods are virtual or abstract. We use the dataset to link a data source to a visual dbgrid component.

The connector control, the data source control, acts as a data-aware connector between visual controls and the underlying data source. In other words, a data source control acts as a conduit for data that is stored in a data set and the controls that display that data on a dbgrid form.

Now let's take a first look at the code the connector:

```
procedure ConnectFB_InterBase(const aDBname: string);
var
  IBconnect : TSQLConnection;
  DataSet    : TSQLDataSet;
  dataQuery  : TSQLQuery;
begin
  IBConnect:= TSQLConnection.Create(NIL);
  try
    with IBConnect do begin
      ConnectionName:= 'VCLScanner';
      DriverName:= 'INTERBASE';
      LibraryName:= 'dbxint30.dll';
      VendorLib:= 'GDS32.DLL';
      GetDriverFunc:= 'getSQLDriverINTERBASE';
      Params.Add('User_Name=SYSDBA');
      Params.Add('Password=masterkey');
      Params.Add('Database='+ADBNAME);
      LoginPrompt:= false;
      Open;
    end;
    dataQuery:= SQLReturn(IBConnect)
    dataSet:= DataSetQuery(IBConnect)
    writeln('debug '+inttoStr(dataset.recordcount));
    CreateDBGridForm(TDataset(dataset));
   except
    E:= Exception.Create('SQL Connect Exception: ');
    Showmessage(E.message+'SQL or connection missing')
  end;   //finalize objects
  if IBconnect.Connected then begin
    DataSet.Close;
    DataSet.Free;
    dataQuery.Close;
    dataQuery.Free;
    IBconnect.Close;
    IBconnect.Free;
  end;
end;
```

Then we pass the dataset to the form with a dbgrid on it.

```
CreateDBGridForm(TDataset(dataset));
```

I mentioned 3 ways and I do try explain ;-):

1. dataQuery:= SQLReturn(IBConnect)
2. dataSet:= DataSetQuery(IBConnect)
3. CreateDBGridForm(TDataset(dataset));

The first demonstrates the query way with a result set back:

```
function SQLReturn(aconnect: TSQLConnection): TSQLQuery;
var qry: TSQLQuery;
   i, z: integer;
begin
 qry:= TSQLQuery.Create(self);
 qry.SQLConnection:= aconnect; //maybe before qry.active:= false;
 qry.SQL.Add(SQLQuery)
 qry.Open;
 Writeln(intToStr(qry.Recordcount)+' SQLQuery records found')
 for i:= 0 to qry.Recordcount - 1 do begin
  for z:= 0 to qry.Fieldcount - 1 do
   Write((qry.Fields[z].asString)+' ');
  Writeln(#13#10)
  qry.Next;
 end;
result:= qry;
end;
```

Second is a typed dataset to pass to the dbgrid:

```
function DataSetQuery(aconnect: TSQLConnection): TSQLDataSet;
var dataset: TSQLDataSet;
   i: integer;
begin
 DataSet:= TSQLDataSet.Create(self);
 with DataSet do begin
  SQLConnection:= aconnect;
  CommandText:= SQLQUERY;
  Open;
  Writeln(intToStr(Recordcount)+' SQLDataSet records found')
  for i:= 0 to Recordcount - 1 do begin
   Writeln('Record: '+intToStr(i)+' '+Fields[1].asString)
   Next;
  end;
 end;
 result:= DataSet;
end;
```

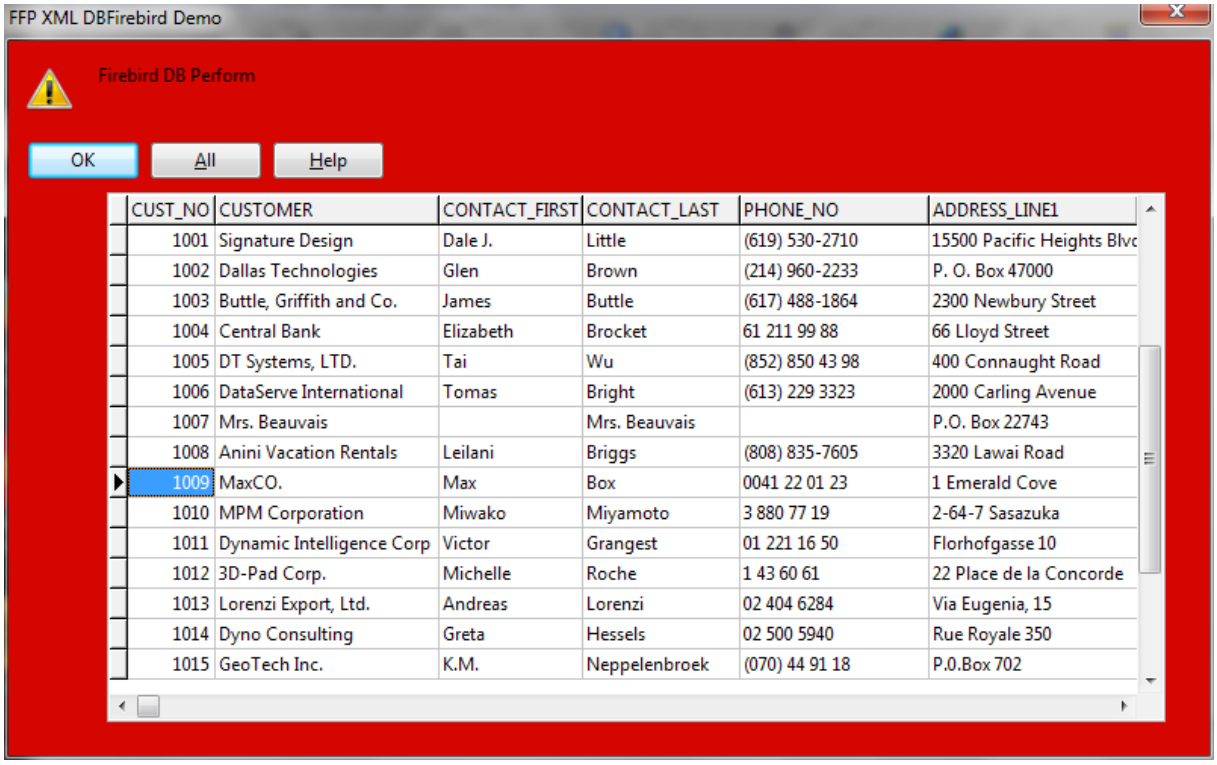
The first and second call way don't use a ODBC driver or a BDE connection either its really dynamically called but needs the corresponding libraries provided:

```
    LibraryName:= 'dbxint30.dll';
    VendorLib:= 'GDS32.DLL';
```

Now comes the third way with the ODBC driver and ADO connection:

```
 TmpTable:= TADOTable.Create(Self);
 with TmpTable do begin
  //connectionString:= 'Provider=MSDASQL;DSN=mx3base;Uid=sa;Pwd=admin';
  connectionString:=
     'Provider=MSDASQL.1;Persist Security Info=False;Data Source=FB_EMPLOYEE';
  //commandText:= 'SELECT * FROM Customer';
  tablename:= 'Customer';
  open;
 end;
```

Here again we use the connection string as in the beginning of the intro. ADO dataset components are typically used for operating on recordsets, and so the contents of the `CommandText` property will usually be one that returns a result set.
For commands that only perform an action or `ExecSQL` and do not result in the creation of a recordset, use the `TADOCommand` component.



CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST	PHONE_NO	ADDRESS_LINE1
1001	Signature Design	Dale J.	Little	(619) 530-2710	15500 Pacific Heights Blvc
1002	Dallas Technologies	Glen	Brown	(214) 960-2233	P. O. Box 47000
1003	Buttle, Griffith and Co.	James	Buttle	(617) 488-1864	2300 Newbury Street
1004	Central Bank	Elizabeth	Brocket	61 211 99 88	66 Lloyd Street
1005	DT Systems, LTD.	Tai	Wu	(852) 850 43 98	400 Connaught Road
1006	DataServe International	Tomas	Bright	(613) 229 3323	2000 Carling Avenue
1007	Mrs. Beauvais		Mrs. Beauvais		P.O. Box 22743
1008	Anini Vacation Rentals	Leilani	Briggs	(808) 835-7605	3320 Lawai Road
1009	MaxCO.	Max	Box	0041 22 01 23	1 Emerald Cove
1010	MPM Corporation	Miwako	Miyamoto	3 880 77 19	2-64-7 Sasazuka
1011	Dynamic Intelligence Corp	Victor	Grangest	01 221 16 50	Florhofgasse 10
1012	3D-Pad Corp.	Michelle	Roche	1 43 60 61	22 Place de la Concorde
1013	Lorenzi Export, Ltd.	Andreas	Lorenzi	02 404 6284	Via Eugenia, 15
1014	Dyno Consulting	Greta	Hessels	02 500 5940	Rue Royale 350
1015	GeoTech Inc.	K.M.	Neppelenbroek	(070) 44 91 18	P.0.Box 702

The same works also with an ADO query object then we uncomment the `commandText` property in line 224 and will be able to use SQL commands!

```
tmpQuery:= TADOQuery.Create(Self);
 with tmpQuery do begin
   connectionString:=
     'Provider=MSDASQL.1;Persist Security Info=False;Data Source=FB_EMPLOYEE';
   commandText:= 'SELECT * FROM Customer';
   Open;
 end;
```

But don't forget to switch the dataset property from table to query:

```
   //dataset:= TDataSet(tmptable);
     dataset:= TDataSet(tmpquery);
```

So here is the living prove that a `TTable` or `TQuery` are derived from a `TDataSet`.

Another possibilities is to send commands like `CreateTable` to the Server. For `TSQLConnection`, Execute takes three parameters: a string that specifies a single SQL statement that you want to execute, a `TParams` object that supplies any parameter values for that statement, and a pointer that can receive a `TCustomSQLDataSet` that is created to return records.

```
   CommandText:= Format('INSERT INTO kings VALUES("%s","%s","%s")',
                                    [Email,FirstN,LastN]);
   ExecSQL(true);
```

Note: Execute can only execute one SQL statement at a time. It is not possible to execute multiple SQL statements with a single call to Execute, as you can with SQL scripting utilities. To execute more than one statement, call Execute repeatedly.

It is relatively easy to execute a statement that does not include any parameters. For example, the following code in our example executes a CREATE TABLE statement (Data Definition Language) without any parameters on a `TSQLConnection` component:

```
procedure createUserTable;
var   Connection: TSQLConnection;
     SQLstmt: String;
begin
  Connection := TSQLConnection.Create(nil);
  with Connection do begin
    ConnectionName := 'VCLScanner';
    DriverName := 'INTERBASE';
    LibraryName := 'dbexpint.dll';
    VendorLib := 'GDS32.DLL';
    GetDriverFunc := 'getSQLDriverINTERBASE';
    Params.Add('User_Name=SYSDBA');
    Params.Add('Password=masterkey');
```

```
  {with TWebModule1.create(NIL) do begin
     getFile_DataBasePath;
     Params.Add(dbPath);
     free;
   end;}
   LoginPrompt := False;
   Connected := True;
   SQLstmt := 'CREATE TABLE NewMaxCusts ' +
   '( ' +
   '  CustNo INTEGER NOT NULL, ' +
   '  Company CHAR(40), ' +
   '  State CHAR(2), ' +
   '  PRIMARY KEY (CustNo) ' +
   ')';
  try
    Execute(SQLstmt, NIL, NIL);
  except
    //raise
  end;
   Close;
   Free;
 end; //end Connection
end;
```

☝ Most queries that return records are SELECT commands. Typically, they define the fields to include, the tables from which to select those fields, conditions that limit what records to include, and the order of the resulting dataset.

1.1.2 Query or Table

With the introduction of InterBase Express (IBX) or in our case DBX, it is now possible to create InterBase or Oracle applications without the overhead of the BDE. Now we come to the question query or dataset because you see both ways!

Using a query is the most general way to specify a set of records. Queries are simply commands written in SQL. You can use either `TSQLDataSet` or `TSQLQuery` to represent the result of a query.

When using `TSQLDataSet`, set the `CommandType` property to `ctQuery` and assign text of the query statement to the `CommandText` property.

When using `TSQLQuery`, assign the query to the SQL property instead. These properties work the same way for all general-purpose or query-type datasets. Specifying the query discusses them in greater detail.

`TSQLQuery` represents a query that is executed using dbExpress.

Use `TSQLDataSet` to

- Represent the records in a database table, a result set of a SELECT query, or the result set returned by a stored procedure.

- Execute a query or stored procedure that doesn't return a result set.
- Represent meta-data that describes what is available on the database server (tables, stored procedures, fields in a table).

Note: If the command does not return any records, you do not need to use a unidirectional dataset at all, because there is no need for the dataset methods that provide access to a set of records. The SQL connection component that connects to the database server can be used directly to execute a command on the server.

With IBX there's a slight difference because of optimisation. TIBDataSet, TIBQuery, and TIBSQL can execute any valid dynamic SQL statement. However, when you use TIBSQL to execute SELECT statements, its results are unbuffered without Streams and therefore unidirectional.

`TMemoryStream` and `TStringStream` are both decendants of `TStream` and they work almost the same way.

Feedback @

max@kleiner.com

Literature: Kleiner et al., "Patterns konkret", 2003, Software & Support

SQL Tutor:

http://www.softwareschule.ch/download/maxbox_starter12.pdf

https://bitbucket.org/max_kleiner/maxbox3/wiki/maXbox%20Tutorials

1.2 Appendix mX SQL Workbench