EMBARCADERO TECHNOLOGIES®

Tech Notes

Rapid database application development with Firebird, Delphi®, and Embarcadero® Change Manager™

Daniel Magin, http://www.DelphiExperts.net

March 2010

CONTENTS

INTRODUCTION

In the latest version of Delphi® and C++Builder® 2010, Embarcadero has added a variety of new features, like IDE Insight, touch support, debugger enhancements, DataSnap® with JSON, etc. The database interface has been given a new feature, among others, with the native dbExpress Firebird driver. dbExpress was presented for the first time in Delphi 6. Since then, dbExpress has been consistently further developed and completely revised in the version Delphi 2007. It has also been provided with Pascal source code.

dbExpress sets itself apart with its open database communication, and it is this and the implementation that make it easy for many manufacturers to realize. Delphi 2010 comes with dbExpress drivers for InterBase®, Firebird, Oracle®, Microsoft® SQL Server, MySQL®, IBM® DB2®, Informix®, Sybase® (ASA, ASE) and Blackfish™ SQL. You will also find manufacturers who provide free and paid drivers for dbExpress, e.g. for SAP, AS/400, PostgreSQL, SQLite, etc. dbExpress also offers the option to develop a multi-tier server architecture. For more information on multi-tier development with DataSnap, allow me to refer you to the Delphi 2010 whitepaper by Bob Swart, "Your data – where you want it, how you want it".

Firebird is a free, open source (www.firebirdsql.org), relational database, which was developed in 2000 and founded on the Open Source InterBase 6.0. This database itself is available for different operating systems, like Windows®, Linux®, Mac® OS X, Solaris and HP-UX®. Delphi has offered a dbExpress provider for InterBase since Delphi 6, and now, in the latest version of Delphi and C++Builder, Embarcadero has also implemented a driver for Firebird.

Before we discuss how to set up a program with Delphi, we need to understand a few dbExpress basics. By contrast with BDE or other database components, dbExpress does not work bidirectionally, it functions in a unidirectional mode, like ADO.NET. dbExpress was developed simultaneously for both frameworks (WIN32 and .NET). This is really easy to see if you take a look at the dbExpress source code. Here you´ll find compiler switches, e.g. "CLR" which can be used in both frameworks.

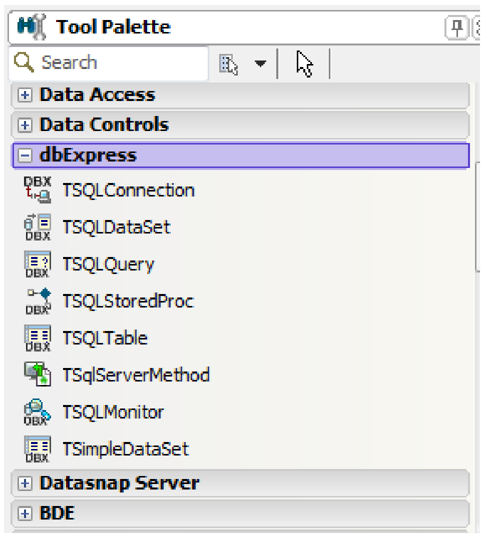
```
{*******************************************************}
{                                                       }
{             Delphi Visual Component Library           }
{                                                       }
{ Copyright(c) 1995-2010 Embarcadero Technologies, Inc. }
{                                                       }
{*******************************************************}

{$HPPEMIT '#pragma link "DbxFirebird"'}    {Do not Localize}
unit DbxFirebird;

interface

uses DbxDynalink,
{$IF DEFINED(CLR)}
DBXDynalinkManaged,
{$ELSE}
DBXDynalinkNative,
{$IFEND}
DBXCommon, Classes, DbxFirebirdReadOnlyMetaData, DbxFirebirdMetaData,
SysUtils;

type
  TDBXFirebirdProperties = class(TDBXProperties)
  …
  end;
{$IF DEFINED(CLR)}
TDBXFirebirdDriver = class(TDBXDynalinkDriverManaged)
{$ELSE}
TDBXFirebirdDriver = class(TDBXDynalinkDriverNative)
{$IFEND}
  public
    constructor Create(DBXDriverDef: TDBXDriverDef); override;
end;
```

You will also find dbExpress and its DataSnap technology in the current version of Delphi Prism (Delphi for the .NET framework). This makes it possible to program cross-platform communication. However, Delphi Prism doesn't include a dbExpress driver for Firebird.

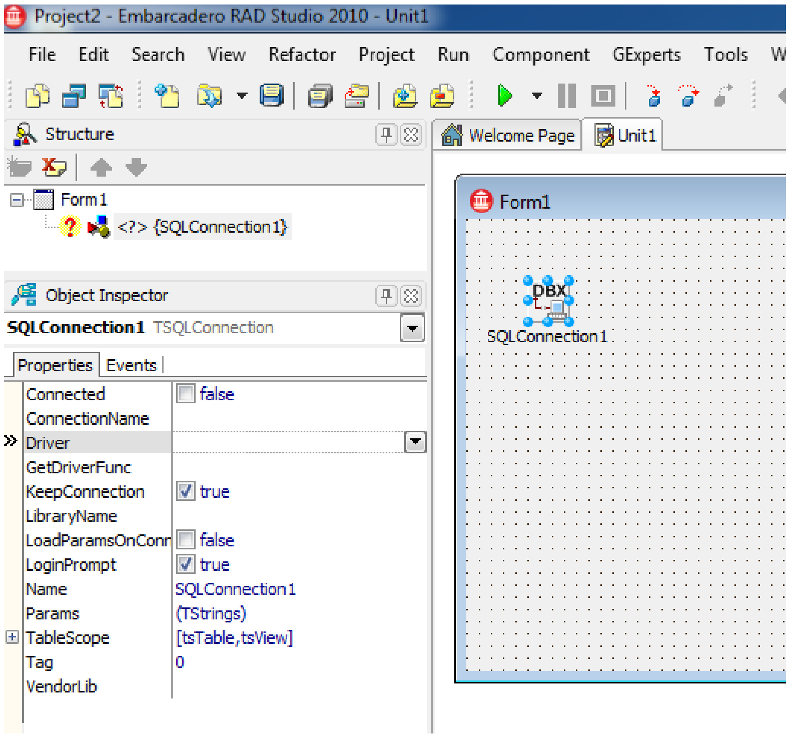
dbExpress Basics

In this section we'll build an application using dbExpress to learn more about it and what unidirectional mode means for developers.

First of all, we create a new VCL form application. In the dbExpress area of the Tool Palette you will find the most important components for creating a connection with dbExpress.



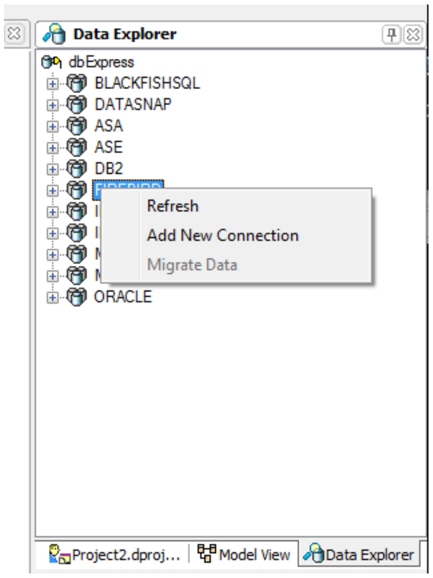
For our first example program we place the TSQLConnection on the form.


Project2 - Embarcadero RAD Studio 2010 - Unit1
File Edit Search View Refactor Project Run Component GExperts Tools
Structure
Form1
<?> {SQLConnection1}
Welcome Page
Unit1
Form1
SQLConnection1
Object Inspector
Properties
Events

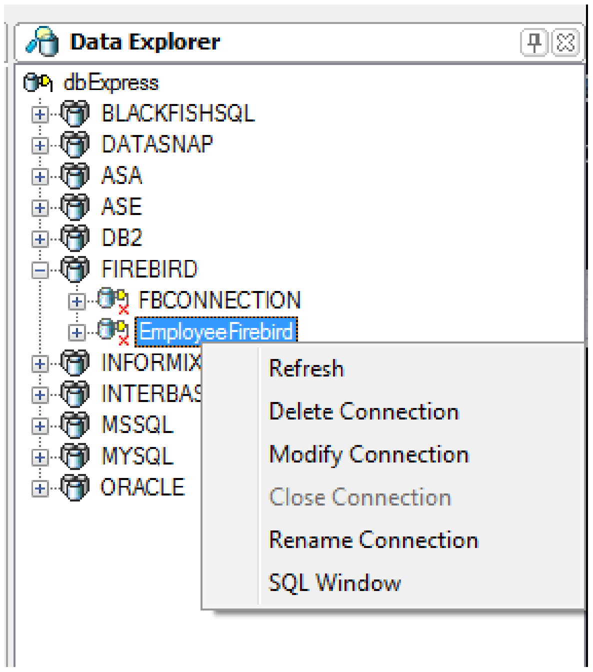
SQLConnection1 TSQLConnection

Connected	false
ConnectionName	
Driver	
GetDriverFunc	
KeepConnection	true
LibraryName	
LoadParamsOnConn	false
LoginPrompt	true
Name	SQLConnection1
Params	(TStrings)
TableScope	[tsTable,tsView]
Tag	0
VendorLib	

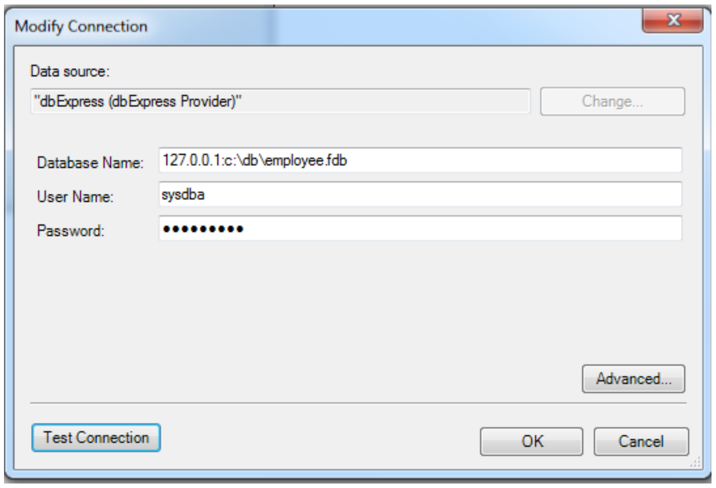
By using the properties Driver, LibraryName, VendorLib and Params, our instance "SQLConnection1" can connect with the desired database. The Data Explorer in the Delphi IDE gives you a simpler and reusable option. To create a new connection, we right click on the desired database (here Firebird) and select form the context menu the item "Add New Connection".



In the following dialogue we can now define a name for your connection, e.g. "EmployeeFirebird". To be able to enter the respective parameters, we must select the new connection, right click on it and select "Modify Connection".



A wizard helps you to specify the respective connection settings. Using the "Advanced..." button we could enter optional parameters for the connection, e.g. CharacterSet, Pooling, Tracing, etc. When all the settings are correct, simply click on "Test Connection" to run through a test.



Data source:
"dbExpress (dbExpress Provider)"

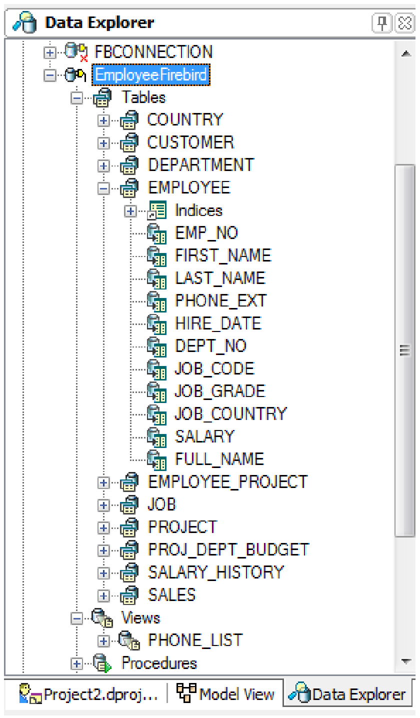
Database Name: 127.0.0.1:c:\db\employee.fdb
User Name: sysdba
Password: ••••••••••

This example uses the database that is provided with the Firebird installation. After installing Firebird, it is located in:

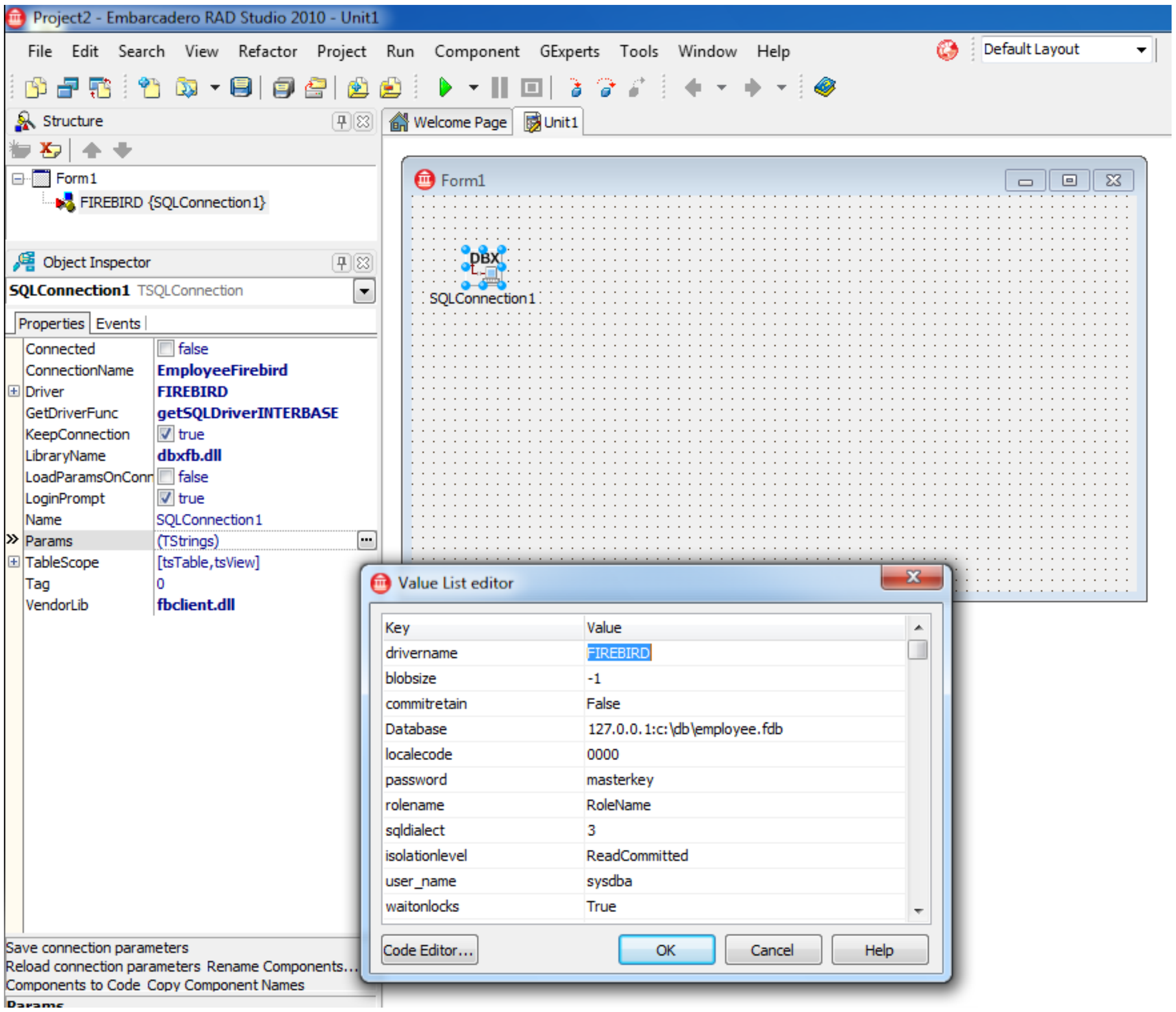
C:\Program Files\Firebird\Firebird_2_1\examples\empbuild\employee.fdb

Simply copy it into a directory like c:\db, so that you always have access to the original example database.

In the Data Explorer we navigate to our new connection. To do this, we simply expand the respective tree node.



Back to our “SQLConnection1”: in the ConnectionName property we can now select our created connection “EmployeeFirebird”. All the parameters we have defined for the database are now copied directly to the instance, i.e. if we change the parameters in the Data Explorer these are not automatically changed in the instance parallel. However, you can achieve this by selecting the ConnectionName property again. This is a major difference compared to the old BDE (Borland Database Engine) and it gives the advantage that you do not have to distribute configuration files as you did before with the old BDE.


Project2 - Embarcadero RAD Studio 2010 - Unit1
File Edit Search View Refactor Project Run Component GExperts Tools Window Help
Default Layout
Structure
Welcome Page
Unit1
Form1
SQLConnection1
Object Inspector
Properties Events
Code Editor... OK Cancel Help

Form1
FIREBIRD {SQLConnection1}

SQLConnection1 TSQLConnection

Connected	false
ConnectionName	EmployeeFirebird
Driver	FIREBIRD
GetDriverFunc	getSQLDriverINTERBASE
KeepConnection	true
LibraryName	dbxfb.dll
LoadParamsOnConn	false
LoginPrompt	true
Name	SQLConnection1
Params	(TStrings)
TableScope	[tsTable,tsView]
Tag	0
VendorLib	fbclient.dll

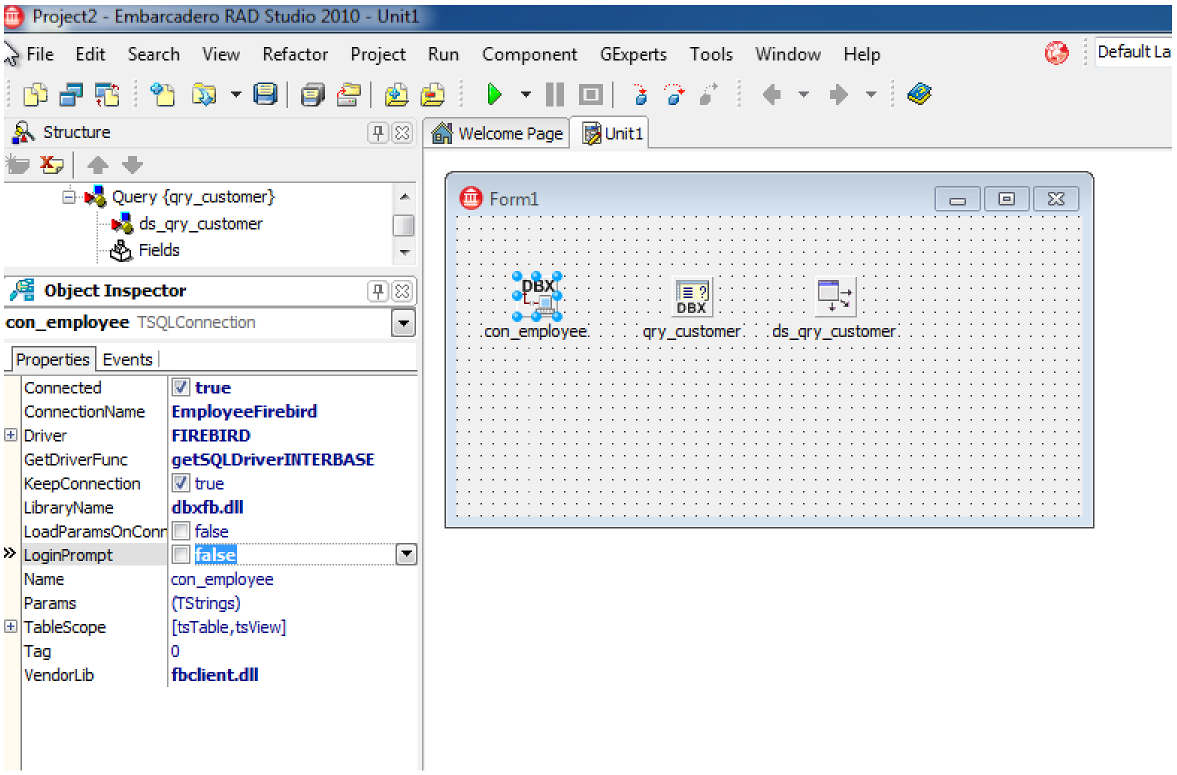
Save connection parameters
Reload connection parameters Rename Components...
Components to Code Copy Component Names

Value List editor

Key	Value
drivername	FIREBIRD
blobsize	-1
commitretain	False
Database	127.0.0.1:c:\db\employee.fdb
localecode	0000
password	masterkey
rolename	RoleName
sqldialect	3
isolationlevel	ReadCommitted
user_name	sysdba
waitonlocks	True

Now we can finally display our first data in our form. To do this, we place the following components on our form:

1. TSQLQuery
    a. Property Name: qry_customer
    b. Property SQLConnection: SQLConnection1
    c. Property SQL: select * from customer
    d. Property Active: True
2. TDataSource
    a. Name: ds_qry_customer
    b. Property DataSet: qry_customer
3. The already existing SQLConnection1 is changed as follows:
    a. Name: con_employee
    b. LoginPrompt: False; so we don´t have to enter a password each time

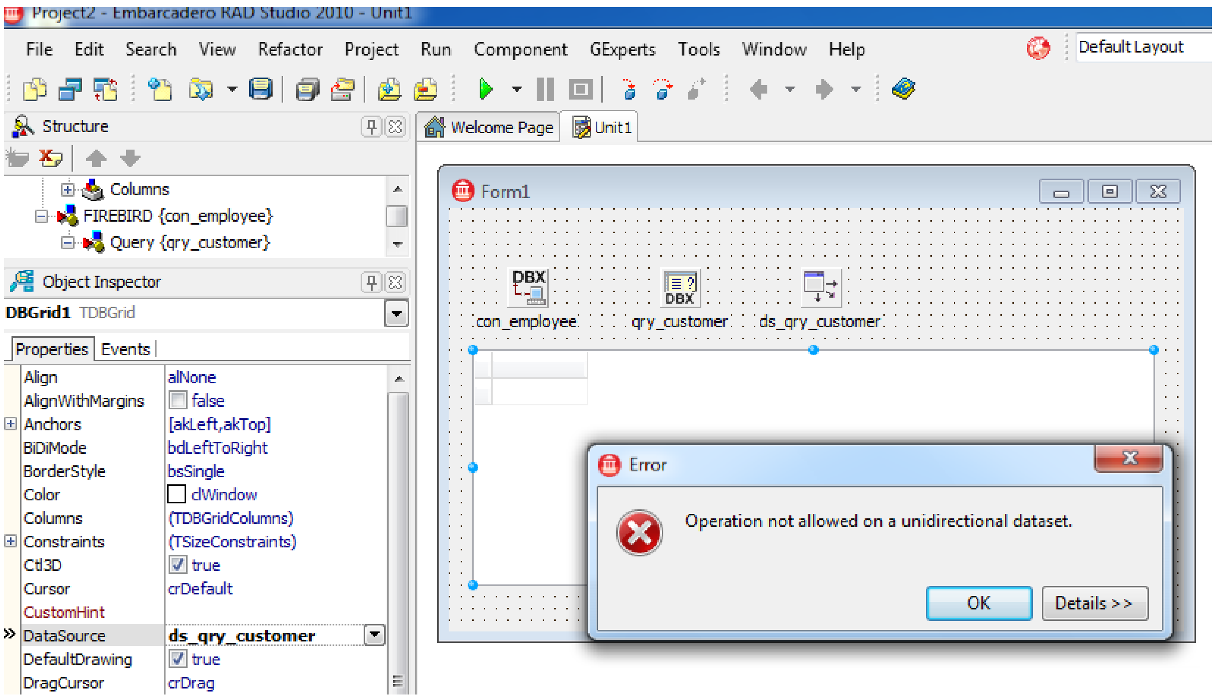

Project2 - Embarcadero RAD Studio 2010 - Unit1
File Edit Search View Refactor Project Run Component GExperts Tools Window Help
Default La
Structure
Query {qry_customer}
ds_qry_customer
Fields
Welcome Page
Unit1
Form1
con_employee
qry_customer
ds_qry_customer

Object Inspector
con_employee TSQLConnection

Properties | Events

Connected	☑ true
ConnectionName	EmployeeFirebird
Driver	FIREBIRD
GetDriverFunc	getSQLDriverINTERBASE
KeepConnection	☑ true
LibraryName	dbxfb.dll
LoadParamsOnConn	☐ false
LoginPrompt	☐ false
Name	con_employee
Params	(TStrings)
TableScope	[tsTable,tsView]
Tag	0
VendorLib	fbclient.dll

To understand the difference between a uni- and bidirectional database connection, we now place a TDBGrid on our form and try to set its property DataSource to our ds_qry_customer. Contrary to BDE, ADO or ODBC connections, we receive an error message.



Object Inspector
DBGrid1 TDBGrid

Properties | Events

Align	alNone
AlignWithMargins	☐ false
Anchors	[akLeft,akTop]
BiDiMode	bdLeftToRight
BorderStyle	bsSingle
Color	☐ clWindow
Columns	(TDBGridColumns)
Constraints	(TSizeConstraints)
Ctl3D	☑ true
Cursor	crDefault
CustomHint	
DataSource	ds_qry_customer
DefaultDrawing	☑ true
DragCursor	crDrag

Error

Operation not allowed on a unidirectional dataset.

OK | Details >>

What does this error message mean? The database API interfaces gds32.dll for InterBase/Firebird, OCI.dll for Oracle, etc. don't do much. They create interfaces for the connection and execution of SQL statements, for fetching results, etc. It is a great myth that these database vendor libraries are also responsible for the caching of data. This was done by BDE, ADO, etc. up to now. However, many programmers have truly violated this mechanism and used Table and Query components to display all the data and used these to filter and navigate, and virtually force all the data for all the clients through the network. Today, architectures use a completely different principle. Even Microsoft has completely changed its old architecture ADO to ADO.NET and some maintain: "It is only these first three letters that have remained". The greatest difference between uni- and a bidirectional database connections is that there is no longer an intermediate layer that caches the result. This must be implemented in the client. This task is carried out in its entirety by one component. But in order to understand this, we must first of all manually build a cache.

One of the most powerful components, TClientDataSet, can do this for us. In the past, some developers who used TClientDataSet complained about having to supply a Midas.dll. Since Delphi 7 you have been able to eliminate the Midas.dll by adding midaslib in the uses clause. For friends of the Midas.dll, it is worth mentioning that Delphi 2010 includes the source code (finally).

Now let´s return to our project; so we do not need to distribute the Midas.dll, we add the unit Midaslib to the uses clause.

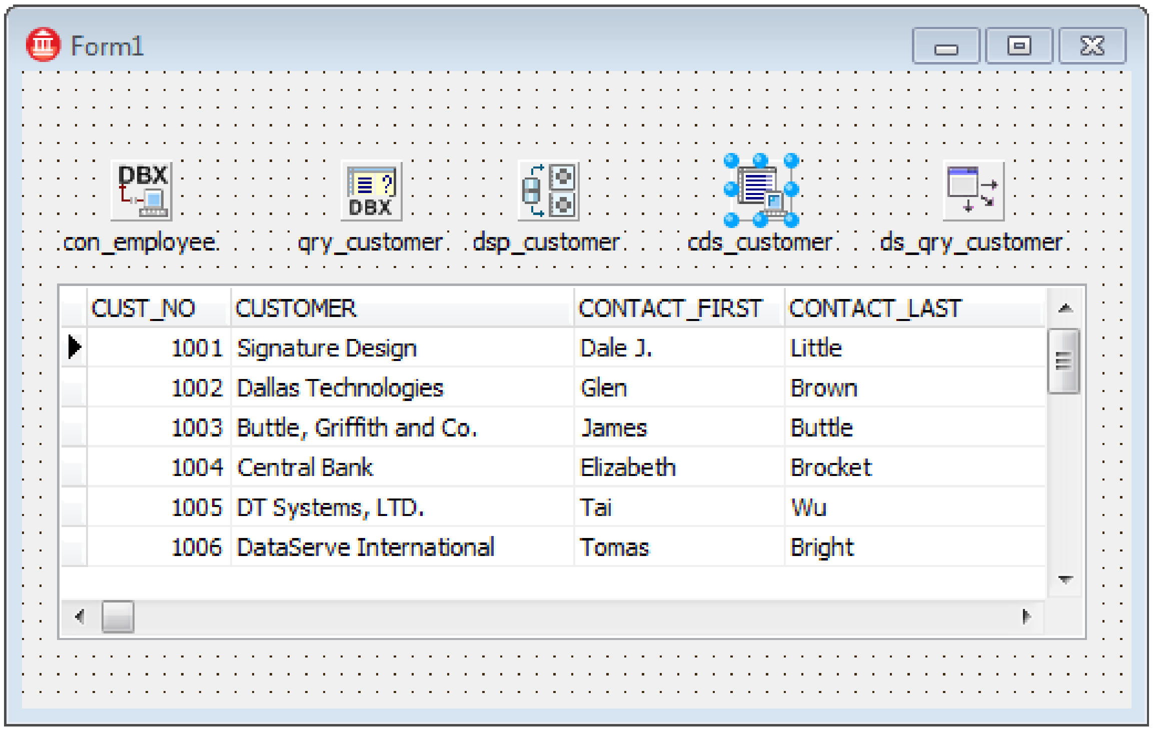
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs,
  midaslib;  //<-- no midas.dll deployment required

type
  TForm1 = class(TForm)
    con_employee: TSQLConnection;
```

We now also place the following components on our form:

1. TDataSetProvider
    a. Property Name: dsp_customer
    b. Property DataSet: qry_customer
2. TClientDataSet
    a. Property Name: cds_customer
    b. Property ProviderName: dsp_customer

3. The already existing component ds_customer (TDataSource)
    a. Property DataSet: cds_customer
4. The already existing component TDBGrid
    a. Property DataSource: ds_customer

If we now set the property Active of cds_customer to True, the data appears in the grid.



CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST
1001	Signature Design	Dale J.	Little
1002	Dallas Technologies	Glen	Brown
1003	Buttle, Griffith and Co.	James	Buttle
1004	Central Bank	Elizabeth	Brocket
1005	DT Systems, LTD.	Tai	Wu
1006	DataServe International	Tomas	Bright

So

What has just happened? Let´s start from the beginning:

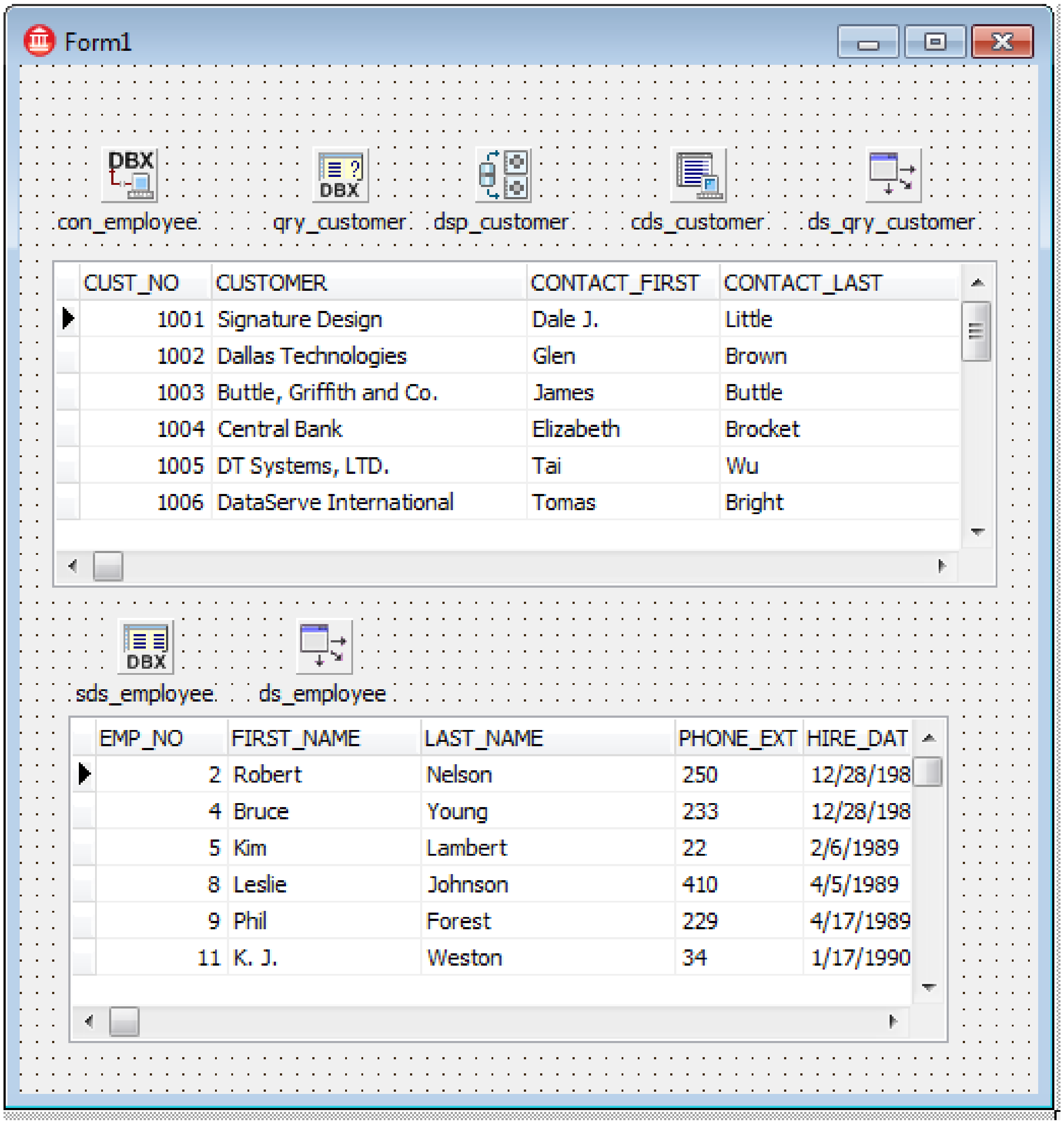
1. The SQLConnection creates the physical connection to the database
2. The query contains the SQL statement
3. The provider is a supplier or the communicator between the query and the ClientDataSet
4. The ClientDataSet is the buffer and this is where data is copied

The whole thing looks very complicated and will confuse some readers who haven´t worked with dbExpress before. Query, Provider and ClientDataSet are also summarised in one component: TSimpleDataSet.

To be able to now show all employees in our project, we proceed as follows:

1. TSimpleDataSet
    a. Property Name: sds_employee
    b. Property Connection: con_employee
    c. Property DataSet->CommandText: select * from employee
    d. Property Active: True
2. TDataSource

   a. Property Name: ds_employee
   b. Property DataSet: sds_employee
3. TDBGrid
   a. Property DataSource: ds_employee



	CUST_NO	CUSTOMER	CONTACT_FIRST	CONTACT_LAST
▶	1001	Signature Design	Dale J.	Little
	1002	Dallas Technologies	Glen	Brown
	1003	Buttle, Griffith and Co.	James	Buttle
	1004	Central Bank	Elizabeth	Brocket
	1005	DT Systems, LTD.	Tai	Wu
	1006	DataServe International	Tomas	Bright

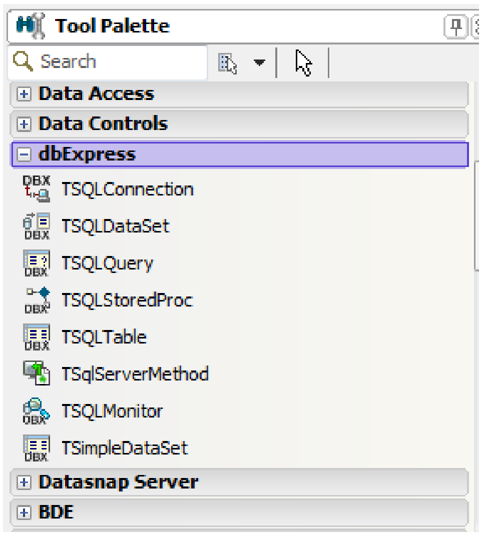
	EMP_NO	FIRST_NAME	LAST_NAME	PHONE_EXT	HIRE_DAT
▶	2	Robert	Nelson	250	12/28/198
	4	Bruce	Young	233	12/28/198
	5	Kim	Lambert	22	2/6/1989
	8	Leslie	Johnson	410	4/5/1989
	9	Phil	Forest	229	4/17/1989
	11	K. J.	Weston	34	1/17/1990

We place 2 TDBNavigator components on the form and set the DataSource property to the respective DataSource components. When we compile and execute our project, we can view and edit all the data, but when we end the program and restart, we will see that the changes have not been saved in our Firebird database. Those who have taken notice so far will probably realise why this is the case. So far, the data has been copied to our data cache TClientDataSet or TSimpleDataSet, i.e. the changes have been saved here and not in the database. To ensure the changes are also immediately saved in the database, we must insert, for example, the following line in the event AfterPost of the sds_Employee:

```
procedure TForm1.sds_employeeAfterPost(DataSet: TDataSet);
begin
 sds_employee.ApplyUpdates(0);
end;
```

After recompiling and executing the application, we should now be able to see our changes saved in the Firebird database.

The individual components in the dbExpress Tool Palette:



dbExpress is made up of several "slim" database components which offer quick access to the SQL database server. dbExpress provides a driver framework for each supported database, and this adapts the server-specific software to the dbExpress interfaces. When we deploy a database application that uses dbExpress, we include a DLL (the server-specific driver) in the application files that you have created.

dbExpress enables us to quickly access databases whilst using unidirectional data sets. Unidirectional data sets are designed for a speedy, simple access to the database information, and, in doing so, require minimum extra effort. As with other data sets, we can send a SQL command to the database server, and, if the command returns several data records, we can receive these data records. Unidirectional data sets do not carry out a caching of the data, which makes them quicker and less resource-dependent than other data set types.

The category, dbExpress in the Tool Palette contains components that use dbExpress to access database information. These are:

1. TSQLConnection
   Encapsulates a dbExpress connection to a database server.
2. TSQLDataSet
   Represents all the data available via dbExpress or sends commands to a database which is being accessed by dbExpress.

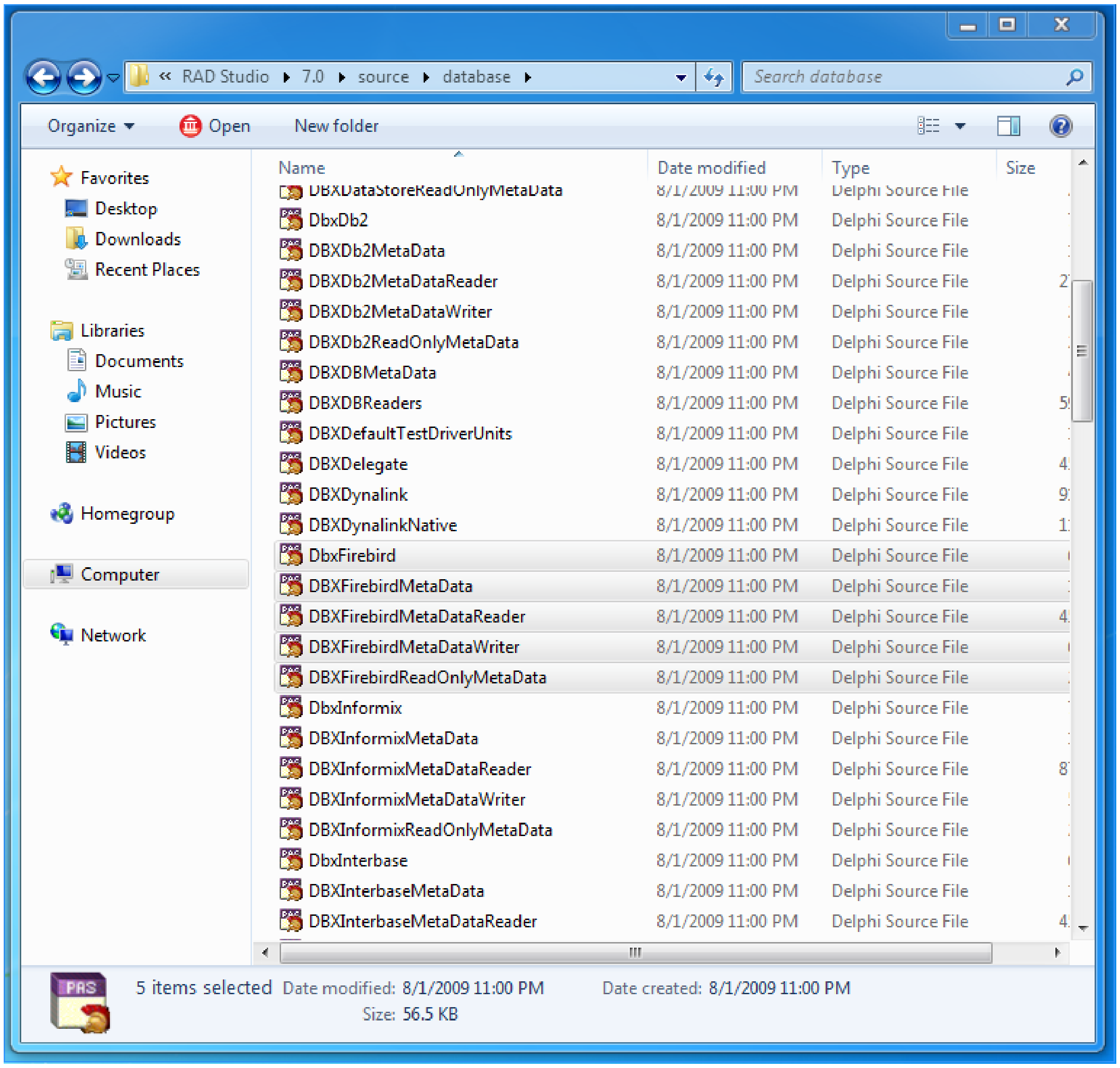
3. TSQLQuery
   A query data set that encapsulates a SQL statement and allows applications to access the result data sets.
4. TSQLTable
   A table data set which represents all the lines and columns of an individual database table.
5. TSQLStoredProc
   A data set based on stored procedures, which executes a stored procedure that is defined on a database server.
6. TSQLMonitor
   Catches the information that is being sent between a SQL connection component and a database server and saves this in a string list.
7. TSimpleDataSet
   A client data set, which retrieves data using the internal objects of type TSQLDataSet and TDataSetProvider and commits updates.

METADATA

READING METADATA

A dbExpress provider must be derived from specific interfaces. By deploying the source of dbExpress we can also carry out a detailed analysis of the sources of the Firebird provider. The source code of the Firebird provider and also those of other databases can be found in the directory:

C:\Program Files\Embarcadero\RAD Studio\7.0\source\database\



Name	Date modified	Type	Size
DBXDataStoreReadOnlyMetaData	8/1/2009 11:00 PM	Delphi Source File	
DbxDb2	8/1/2009 11:00 PM	Delphi Source File	
DBXDb2MetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXDb2MetaDataReader	8/1/2009 11:00 PM	Delphi Source File	2
DBXDb2MetaDataWriter	8/1/2009 11:00 PM	Delphi Source File	
DBXDb2ReadOnlyMetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXDBMetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXDBReaders	8/1/2009 11:00 PM	Delphi Source File	5
DBXDefaultTestDriverUnits	8/1/2009 11:00 PM	Delphi Source File	
DBXDelegate	8/1/2009 11:00 PM	Delphi Source File	4
DBXDynalink	8/1/2009 11:00 PM	Delphi Source File	9
DBXDynalinkNative	8/1/2009 11:00 PM	Delphi Source File	1
DbxFirebird	8/1/2009 11:00 PM	Delphi Source File	
DBXFirebirdMetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXFirebirdMetaDataReader	8/1/2009 11:00 PM	Delphi Source File	4
DBXFirebirdMetaDataWriter	8/1/2009 11:00 PM	Delphi Source File	
DBXFirebirdReadOnlyMetaData	8/1/2009 11:00 PM	Delphi Source File	
DbxInformix	8/1/2009 11:00 PM	Delphi Source File	
DBXInformixMetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXInformixMetaDataReader	8/1/2009 11:00 PM	Delphi Source File	8
DBXInformixMetaDataWriter	8/1/2009 11:00 PM	Delphi Source File	
DBXInformixReadOnlyMetaData	8/1/2009 11:00 PM	Delphi Source File	
DbxInterbase	8/1/2009 11:00 PM	Delphi Source File	
DBXInterbaseMetaData	8/1/2009 11:00 PM	Delphi Source File	
DBXInterbaseMetaDataReader	8/1/2009 11:00 PM	Delphi Source File	4

5 items selected Date modified: 8/1/2009 11:00 PM Date created: 8/1/2009 11:00 PM
Size: 56.5 KB

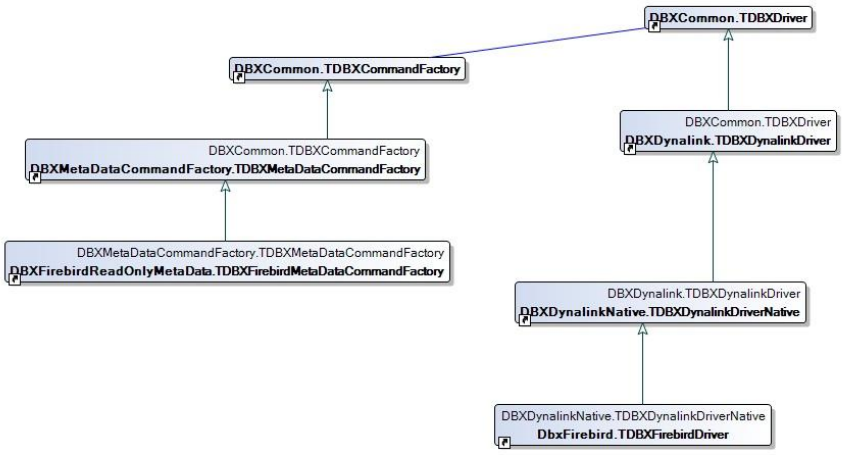
Each database has internal system tables, and these are RDB$ tables in InterBase/Firebird. The complete skeleton of the respective database is located in these. You should never make any changes here. It is very easy to destroy a database if you don´t know what you´re doing. If you want to read the metadata of the database, you need different SQL statements depending on the database manufacturer. However, a dbExpress provider presents us with complete methods for this. All providers are derived from specific abstract classes; we can find such a basic class in the unit DBXMetaDataReader.pas (line 251):

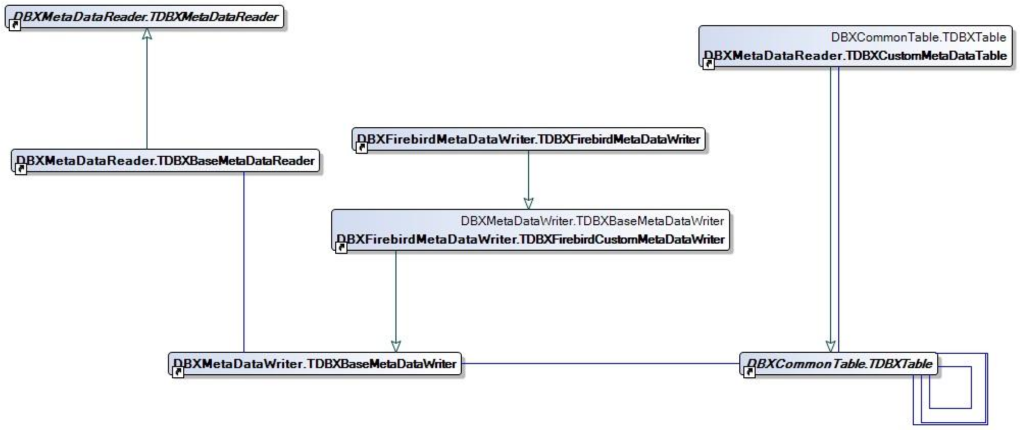
```
TDBXMetaDataReader = class abstract
  public
    function FetchCollection(const MetaDataCommand: UnicodeString):
TDBXTable; virtual; abstract;
    function FetchCollectionWithStorage(const MetaDataCommand:
UnicodeString): TDBXTable; virtual; abstract;
  protected
    procedure SetContext(const Context: TDBXProviderContext); virtual;
abstract;
    function GetContext: TDBXProviderContext; virtual; abstract;
    function GetProductName: UnicodeString; virtual; abstract;
    function GetVersion: UnicodeString; virtual; abstract;
    procedure SetVersion(const Version: UnicodeString); virtual; abstract;
    function GetSqlIdentifierQuotePrefix: UnicodeString; virtual; abstract;
    function GetSqlIdentifierQuoteSuffix: UnicodeString; virtual; abstract;
    function IsLowerCaseIdentifiersSupported: Boolean; virtual; abstract;
    function IsUpperCaseIdentifiersSupported: Boolean; virtual; abstract;
    function IsQuotedIdentifiersSupported: Boolean; virtual; abstract;
    function IsDescendingIndexSupported: Boolean; virtual; abstract;
    function IsDescendingIndexColumnsSupported: Boolean; virtual; abstract;
    function GetSqlIdentifierQuoteChar: UnicodeString; virtual; abstract;
    function GetSqlProcedureQuoteChar: UnicodeString; virtual; abstract;
    function IsMultipleCommandsSupported: Boolean; virtual; abstract;
    function IsTransactionsSupported: Boolean; virtual; abstract;
    function IsNestedTransactionsSupported: Boolean; virtual; abstract;
    function IsSetRowSizeSupported: Boolean; virtual; abstract;
    function IsSPReturnCodeSupported: Boolean; virtual; abstract;
  public
    property Context: TDBXProviderContext read GetContext write SetContext;
    property ProductName: UnicodeString read GetProductName;
    property Version: UnicodeString read GetVersion write SetVersion;
    property SqlIdentifierQuotePrefix: UnicodeString read
GetSqlIdentifierQuotePrefix;
    property SqlIdentifierQuoteSuffix: UnicodeString read
GetSqlIdentifierQuoteSuffix;
    property LowerCaseIdentifiersSupported: Boolean read
IsLowerCaseIdentifiersSupported;
    property UpperCaseIdentifiersSupported: Boolean read
IsUpperCaseIdentifiersSupported;
    property QuotedIdentifiersSupported: Boolean read
IsQuotedIdentifiersSupported;
    property DescendingIndexSupported: Boolean read
IsDescendingIndexSupported;
    property DescendingIndexColumnsSupported: Boolean read
IsDescendingIndexColumnsSupported;
    property SqlIdentifierQuoteChar: UnicodeString read
GetSqlIdentifierQuoteChar;
    property SqlProcedureQuoteChar: UnicodeString read
```

```
GetSqlProcedureQuoteChar;
    property MultipleCommandsSupported: Boolean read
IsMultipleCommandsSupported;
    property TransactionsSupported: Boolean read IsTransactionsSupported;
    property NestedTransactionsSupported: Boolean read
IsNestedTransactionsSupported;
    property SetRowSizeSupported: Boolean read IsSetRowSizeSupported;
    property SPReturnCodeSupported: Boolean read IsSPReturnCodeSupported;
  end;
```

TDBXMetaDataReader is an abstract class, i.e. classes that are derived from this must also implement its methods. If, for example, you derive from the interface or abstract classes , you enter a contract. For example, TDBXBaseMetaDataReader is derived from this class and from this again TDBXFirebirdCustomMetaDataReader and then from this again TDBXFirebirdMetaDataReader.

In order to document the complete structure of dbExpress, this whitepaper will skip over some things, but here´s a brief overview of the most important classes for the Firebird dbExpress provider:


DBXCommon.TDBXDriver
DBXCommon.TDBXCommandFactory
DBXCommon.TDBXDriver
DBXDynalink.TDBXDynalinkDriver
DBXCommon.TDBXCommandFactory
DBXMetaDataCommandFactory.TDBXMetaDataCommandFactory
DBXMetaDataCommandFactory.TDBXMetaDataCommandFactory
DBXFirebirdReadOnlyMetaData.TDBXFirebirdMetaDataCommandFactory
DBXDynalink.TDBXDynalinkDriver
DBXDynalinkNative.TDBXDynalinkDriverNative
DBXDynalinkNative.TDBXDynalinkDriverNative
DbxFirebird.TDBXFirebirdDriver


DBXMetaDataReader.TDBXMetaDataReader
DBXCommonTable.TDBXTable
DBXMetaDataReader.TDBXCustomMetaDataTable
DBXFirebirdMetaDataWriter.TDBXFirebirdMetaDataWriter
DBXMetaDataReader.TDBXBaseMetaDataReader
DBXMetaDataWriter.TDBXBaseMetaDataWriter
DBXFirebirdMetaDataWriter.TDBXFirebirdCustomMetaDataWriter
DBXMetaDataWriter.TDBXBaseMetaDataWriter
DBXCommonTable.TDBXTable

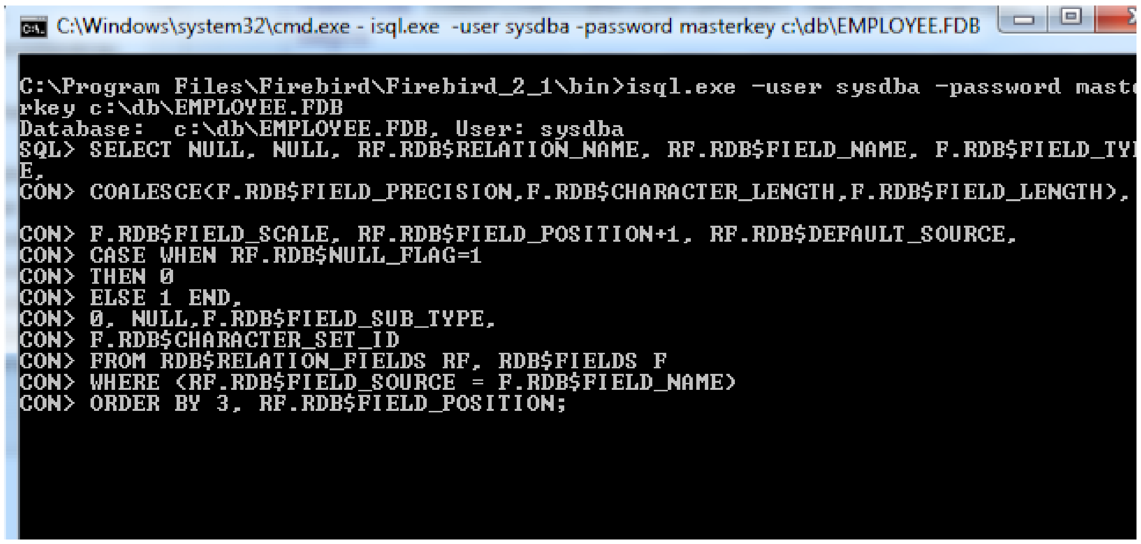
Take a look in the unit DBXFirebirdMetaDataReader.pas (line 86):

```
TDBXFirebirdMetaDataReader = class(TDBXFirebirdCustomMetaDataReader)
...
  protected
    function GetProductName: UnicodeString; override;
    function IsDescendingIndexSupported: Boolean; override;
    function IsDescendingIndexColumnsSupported: Boolean; override;
    function IsNestedTransactionsSupported: Boolean; override;
    function GetSqlForTables: UnicodeString; override;
    function GetSqlForViews: UnicodeString; override;
    function GetSqlForColumns: UnicodeString; override;
    function GetSqlForIndexes: UnicodeString; override;
    function GetSqlForIndexColumns: UnicodeString; override;
    function GetSqlForForeignKeys: UnicodeString; override;
    function GetSqlForForeignKeyColumns: UnicodeString; override;
    function GetSqlForProcedures: UnicodeString; override;
    function GetSqlForProcedureSources: UnicodeString; override;
    function GetSqlForProcedureParameters: UnicodeString; override;
    function GetSqlForUsers: UnicodeString; override;
    function GetSqlForRoles: UnicodeString; override;
    function GetReservedWords: TDBXStringArray; override;
  end;
```

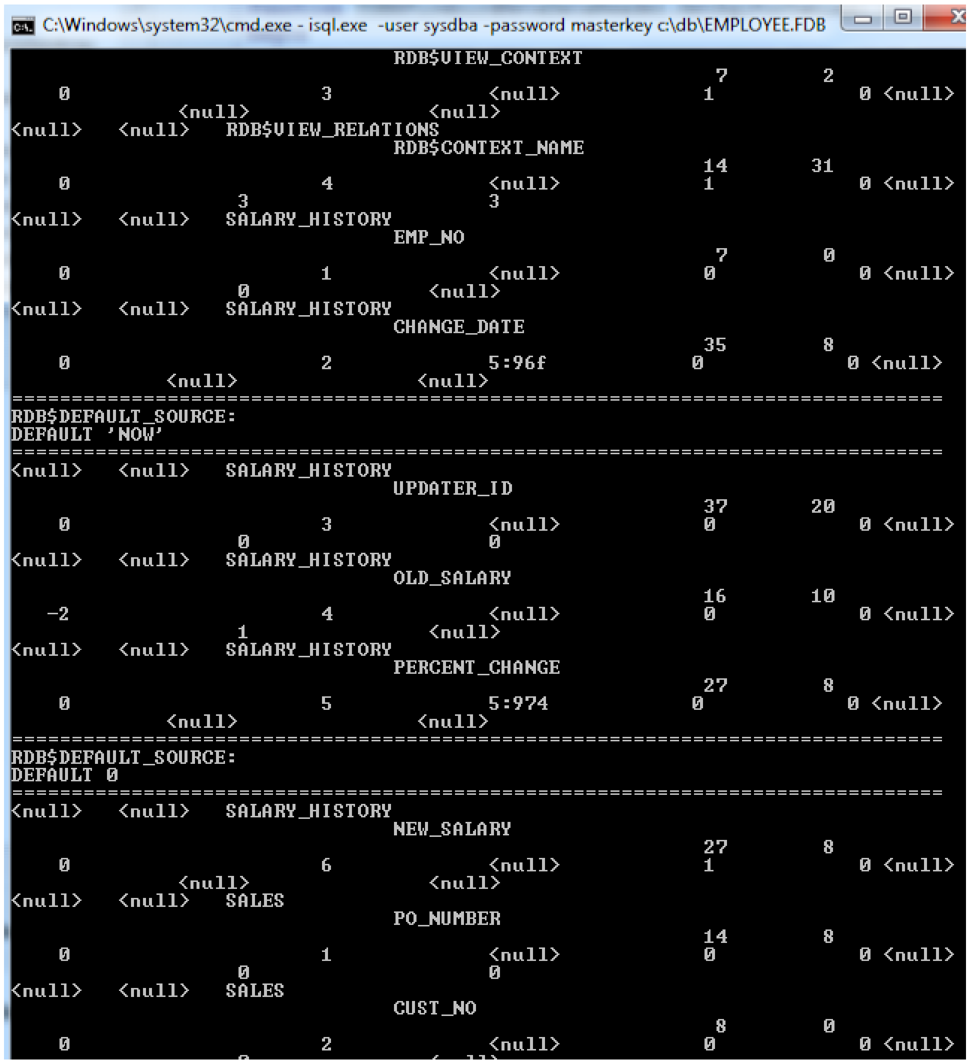
This class is interesting for the reason that TDBXFirebirdMetaDataReader makes available methods that provide us with the respective metadata SQL statements. In the implementation for the method GetSqlForTables we find:

```
function TDBXFirebirdMetaDataReader.GetSqlForTables: UnicodeString;
begin
  Result := 'SELECT NULL, NULL, RDB$RELATION_NAME, CASE WHEN RDB$SYSTEM_FLAG
> 0 THEN ''SYSTEM TABLE'' WHEN RDB$VIEW_SOURCE IS NOT NULL THEN ''VIEW'' ELSE
''TABLE'' END AS TABLE_TYPE ' +
            'FROM RDB$RELATIONS ' +
            'WHERE (1<2 OR (:CATALOG_NAME IS NULL)) AND (1<2 OR (:SCHEMA_NAME
IS NULL)) AND (RDB$RELATION_NAME = :TABLE_NAME OR (:TABLE_NAME IS NULL)) ' +
            ' AND ((RDB$SYSTEM_FLAG > 0 AND :SYSTEM_TABLES=''SYSTEM TABLE'')
OR (RDB$VIEW_SOURCE IS NOT NULL AND :VIEWS=''VIEW'') OR (RDB$SYSTEM_FLAG = 0
AND RDB$VIEW_SOURCE IS NULL AND :TABLES=''TABLE'')) ' +
            'ORDER BY 3';
end;
```

This SQL statement also has parameters for Catalog and Schema, etc., which are then completed by a parent class. If we now execute a simplified SQL statement (without the parameters) directly with iSQL on Firebird, we obtain all the tables back from our database.

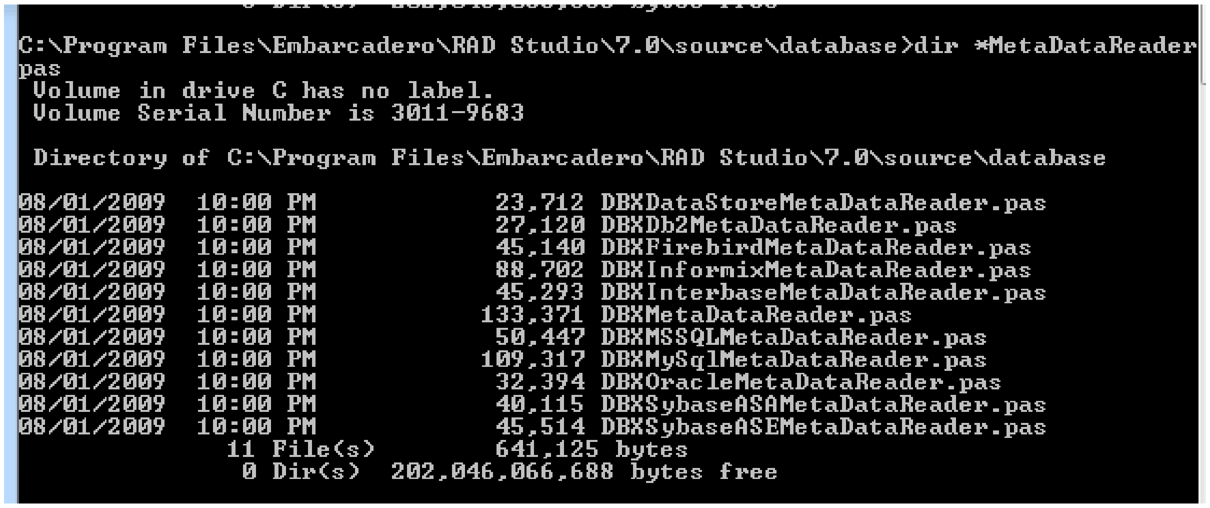


```
C:\Program Files\Firebird\Firebird_2_1\bin>isql.exe -user sysdba -password mast
rkey c:\db\EMPLOYEE.FDB
Database:  c:\db\EMPLOYEE.FDB, User: sysdba
SQL> SELECT NULL, NULL, RF.RDB$RELATION_NAME, RF.RDB$FIELD_NAME, F.RDB$FIELD_TY
E,
CON> COALESCE(F.RDB$FIELD_PRECISION,F.RDB$CHARACTER_LENGTH,F.RDB$FIELD_LENGTH),

CON> F.RDB$FIELD_SCALE, RF.RDB$FIELD_POSITION+1, RF.RDB$DEFAULT_SOURCE,
CON> CASE WHEN RF.RDB$NULL_FLAG=1
CON> THEN 0
CON> ELSE 1 END,
CON> 0, NULL,F.RDB$FIELD_SUB_TYPE,
CON> F.RDB$CHARACTER_SET_ID
CON> FROM RDB$RELATION_FIELDS RF, RDB$FIELDS F
CON> WHERE (RF.RDB$FIELD_SOURCE = F.RDB$FIELD_NAME)
CON> ORDER BY 3, RF.RDB$FIELD_POSITION;
```



```
                                RDB$VIEW_CONTEXT
                                                            7        2
    0                     3             <null>             1            0 <null>
              <null>               <null>
<null>   <null>   RDB$VIEW_RELATIONS
                                RDB$CONTEXT_NAME
                                                           14       31
    0                     4             <null>             1            0 <null>
                   3                    3
<null>   <null>   SALARY_HISTORY
                                EMP_NO
                                                            7        0
    0                     1             <null>             0            0 <null>
                   0                   <null>
<null>   <null>   SALARY_HISTORY
                                CHANGE_DATE
                                                           35        8
    0                     2             5:96f             0            0 <null>
             <null>               <null>
===============================================================================
RDB$DEFAULT_SOURCE:
DEFAULT 'NOW'
===============================================================================
<null>   <null>   SALARY_HISTORY
                                UPDATER_ID
                                                           37       20
    0                     3             <null>             0            0 <null>
                   0                    0
<null>   <null>   SALARY_HISTORY
                                OLD_SALARY
                                                           16       10
   -2                     4             <null>             0            0 <null>
                   1                   <null>
<null>   <null>   SALARY_HISTORY
                                PERCENT_CHANGE
                                                           27        8
    0                     5             5:974             0            0 <null>
             <null>               <null>
===============================================================================
RDB$DEFAULT_SOURCE:
DEFAULT 0
===============================================================================
<null>   <null>   SALARY_HISTORY
                                NEW_SALARY
                                                           27        8
    0                     6             <null>             1            0 <null>
              <null>               <null>
<null>   <null>   SALES
                                PO_NUMBER
                                                           14        8
    0                     1             <null>             0            0 <null>
                   0                    0
<null>   <null>   SALES
                                CUST_NO
                                                            8        0
    0                     2             <null>             0            0 <null>
```

You can also find respective implementations for all other databases in the MetaDataReader implementations for the individual databases:

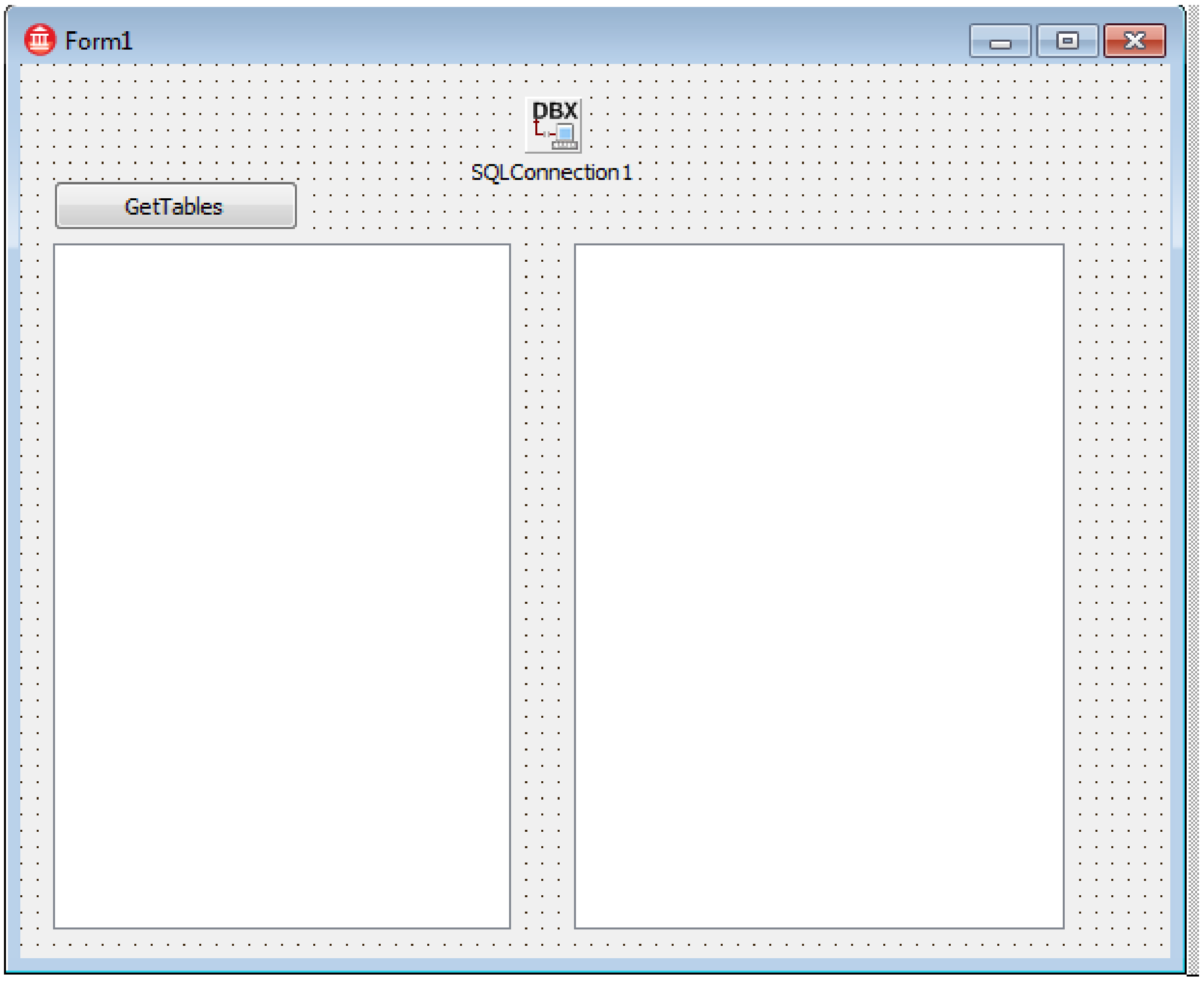


```
C:\Program Files\Embarcadero\RAD Studio\7.0\source\database>dir *MetaDataReader
pas
 Volume in drive C has no label.
 Volume Serial Number is 3011-9683

 Directory of C:\Program Files\Embarcadero\RAD Studio\7.0\source\database

08/01/2009  10:00 PM            23,712 DBXDataStoreMetaDataReader.pas
08/01/2009  10:00 PM            27,120 DBXDb2MetaDataReader.pas
08/01/2009  10:00 PM            45,140 DBXFirebirdMetaDataReader.pas
08/01/2009  10:00 PM            88,702 DBXInformixMetaDataReader.pas
08/01/2009  10:00 PM            45,293 DBXInterbaseMetaDataReader.pas
08/01/2009  10:00 PM           133,371 DBXMetaDataReader.pas
08/01/2009  10:00 PM            50,447 DBXMSSQLMetaDataReader.pas
08/01/2009  10:00 PM           109,317 DBXMySqlMetaDataReader.pas
08/01/2009  10:00 PM            32,394 DBXOracleMetaDataReader.pas
08/01/2009  10:00 PM            40,115 DBXSybaseASAMetaDataReader.pas
08/01/2009  10:00 PM            45,514 DBXSybaseASEMetaDataReader.pas
              11 File(s)        641,125 bytes
               0 Dir(s)  202,046,066,688 bytes free
```

If we want to read the metadata from our Firebird database without any knowledge of system tables, the dbExpress system helps thanks to its uniform structure. For this purpose, we once again create a new VCL form application and, as in our first example, we place a TSQLConnection on our form and set the following properties:

1. ConnectionName: EmployeeFirebird
2. LoginPrompt: False
3. Connected: True

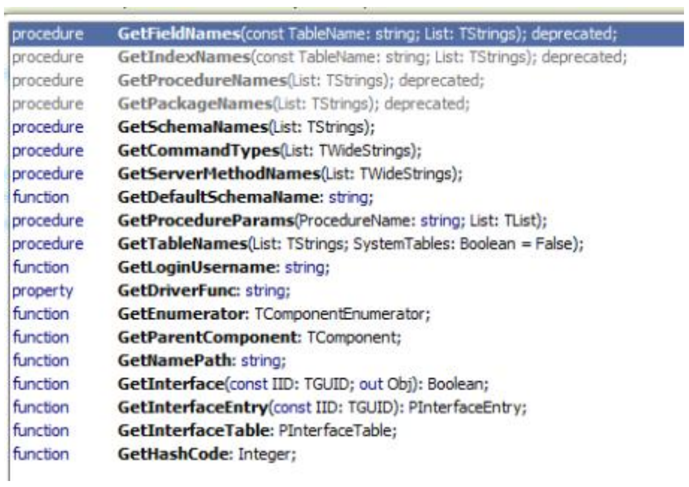
Furthermore, we place two TListbox and a TButton on our form and the whole thing should look like this:



The aim is to show all the tables (without system tables) in the left listbox and then click to add the columns of the selected table in the right listbox. We start off by implementing the button click method:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  ListBox1.Items.Clear;
  SQLConnection1.GetTableNames(ListBox1.Items,false);
end;
```

Our SQL connection has a multitude of Get…. Methods:



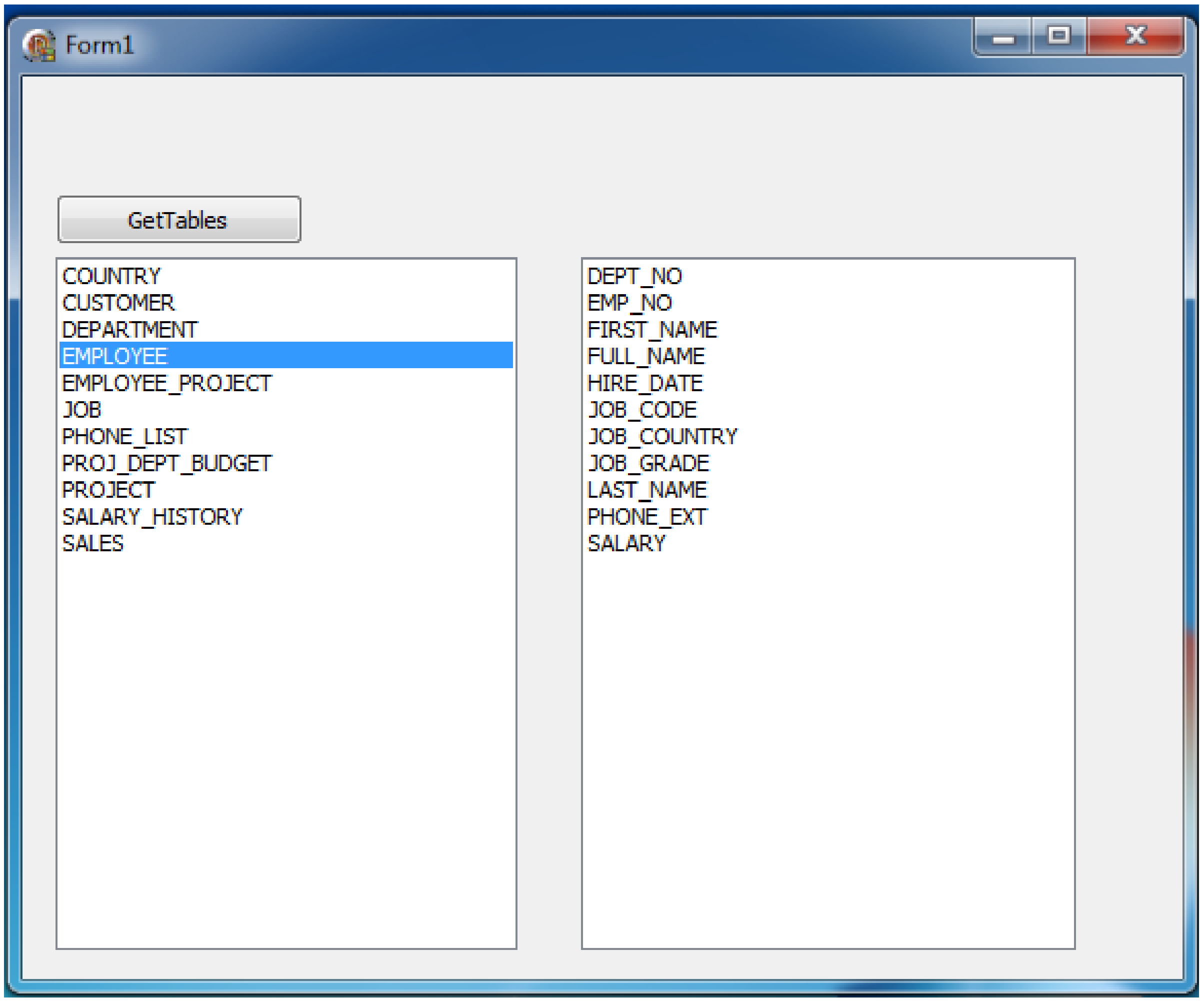
```
procedure   GetFieldNames(const TableName: string; List: TStrings); deprecated;
procedure   GetIndexNames(const TableName: string; List: TStrings); deprecated;
procedure   GetProcedureNames(List: TStrings); deprecated;
procedure   GetPackageNames(List: TStrings); deprecated;
procedure   GetSchemaNames(List: TStrings);
procedure   GetCommandTypes(List: TWideStrings);
procedure   GetServerMethodNames(List: TWideStrings);
function    GetDefaultSchemaName: string;
procedure   GetProcedureParams(ProcedureName: string; List: TList);
procedure   GetTableNames(List: TStrings; SystemTables: Boolean = False);
function    GetLoginUsername: string;
property    GetDriverFunc: string;
function    GetEnumerator: TComponentEnumerator;
function    GetParentComponent: TComponent;
function    GetNamePath: string;
function    GetInterface(const IID: TGUID; out Obj): Boolean;
function    GetInterfaceEntry(const IID: TGUID): PInterfaceEntry;
function    GetInterfaceTable: PInterfaceTable;
function    GetHashCode: Integer;
```

For the GetTableNames there are several overloads. The one used the most is the one in which the first parameter specifies a TStrings object in which the elements are to be stored. The second parameter indicates whether or not system tables are to be output. We will use precisely this overload for the button click method.

We now implement the Listbox1 click event:

```
procedure TForm1.ListBox1Click(Sender: TObject);
var
 sSelectedTableName: string;
begin
 ListBox2.Items.Clear;
 sSelectedTableName:=ListBox1.Items.Strings[ListBox1.ItemIndex];
 SQLConnection1.GetFieldNames(sSelectedTableName,ListBox2.Items);

end;
```

First of all, we delete the contents in the right listbox, then read the selected table names in the left listbox and finally use the GetFieldNames method to complete the right listbox with the respective table names. The great thing about this architecture is that, irrespective of which database is used with the dbExpress connection, we obtain the meta data without having to know the system tables.

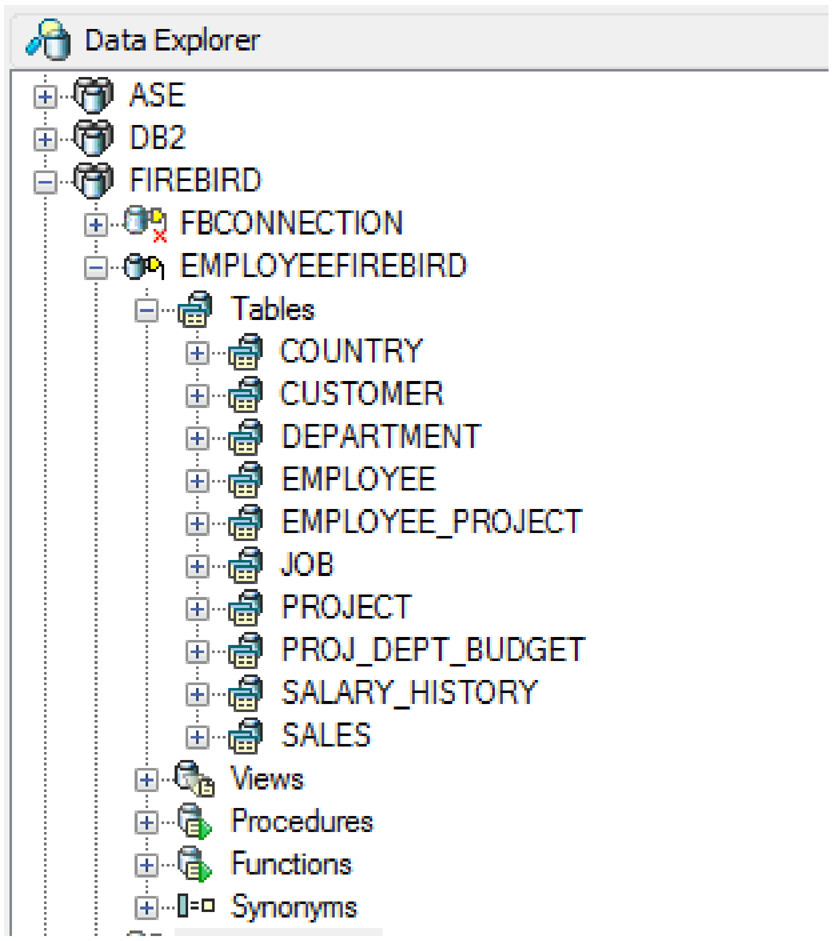


```
COUNTRY                DEPT_NO
CUSTOMER               EMP_NO
DEPARTMENT             FIRST_NAME
EMPLOYEE               FULL_NAME
EMPLOYEE_PROJECT       HIRE_DATE
JOB                    JOB_CODE
PHONE_LIST             JOB_COUNTRY
PROJ_DEPT_BUDGET       JOB_GRADE
PROJECT                LAST_NAME
SALARY_HISTORY         PHONE_EXT
SALES                  SALARY
```

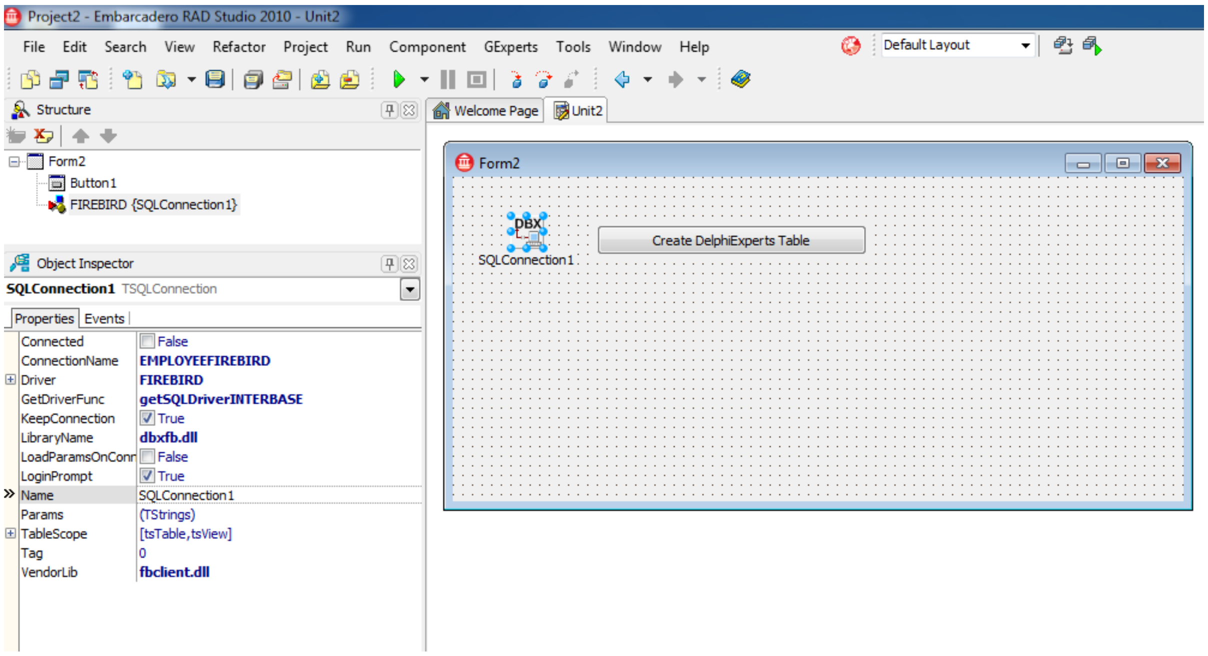
WORKING ACTIVELY WITH METADATA

With the dbExpress metadata architecture we can read the meta data of several databases via an architecture, and we can even use this to actively manipulate all the supported databases. To do this, we create a new VCL form application with which we want to create a new table on the Firebird database server. Using this application we can create this table with the identical code for Oracle, MSSQL, InterBase, MySQL, etc.

We use the Data Explorer to check that no DelphiExperts table exists in our employee database.



We place a TSQLConnection on our form and set the ConnectionName property to our EmployeeFirebird entry from the Data Explorer.


Project2 - Embarcadero RAD Studio 2010 - Unit2
File Edit Search View Refactor Project Run Component GExperts Tools Window Help
Default Layout
Structure
Form2
Button1
FIREBIRD {SQLConnection1}
Welcome Page
Unit2
Form2
SQLConnection1
Create DelphiExperts Table
Object Inspector
Properties Events

SQLConnection1 TSQLConnection

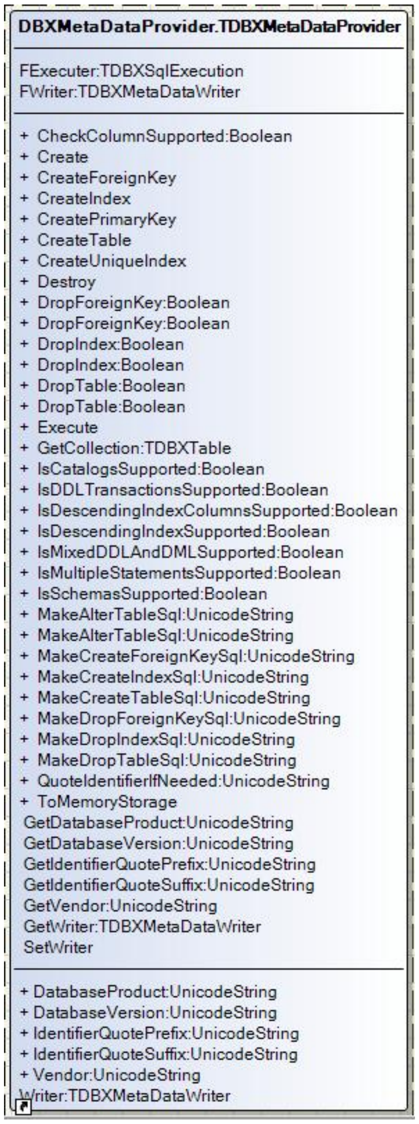
Connected	False
ConnectionName	EMPLOYEEFIREBIRD
Driver	FIREBIRD
GetDriverFunc	getSQLDriverINTERBASE
KeepConnection	True
LibraryName	dbxfb.dll
LoadParamsOnConn	False
LoginPrompt	True
Name	SQLConnection1
Params	(TStrings)
TableScope	[tsTable,tsView]
Tag	0
VendorLib	fbclient.dll

Then we place a TButton on the form. First of all, we create a small function (DBXGetMetaProvider) which, with the help of our SQLConnection, returns a MetaDataProvider.

```
unit formMain;

interface

uses
  Windows, Messages, SysUtils, Variants, Classes, Graphics, Controls, Forms,
  Dialogs, WideStrings, DBXFirebird, StdCtrls, DB, SqlExpr,

  DbxCommon, DbxMetaDataProvider,
  DBXDataExpressMetaDataProvider,
  DbxClient, DBXDataStoreMetaData;

type
  TForm2 = class(TForm)
    SQLConnection1: TSQLConnection;
    Button1: TButton;
  private
    { Private declarations }
    function DBXGetMetaProvider(const AConnection: TDBXConnection)
      : TDBXDataExpressMetaDataProvider;
  public
    { Public declarations }
  end;

...

implementation

{$R *.dfm}

...
```

```
function TForm2.DBXGetMetaProvider(const AConnection: TDBXConnection)
  : TDBXDataExpressMetaDataProvider;
var
  Provider: TDBXDataExpressMetaDataProvider;
begin
  Provider := TDBXDataExpressMetaDataProvider.Create;
  try
    Provider.Connection := AConnection;
    Provider.Open;
  except
    FreeAndNil(Provider);
    raise ;
  end;

  Result := Provider;
end;

end.
```

We can use this metadata provider to call a whole host of methods for our database. Here´s a brief overview. Note the methods like CreateTable, CreateIndex, DropTable, etc.

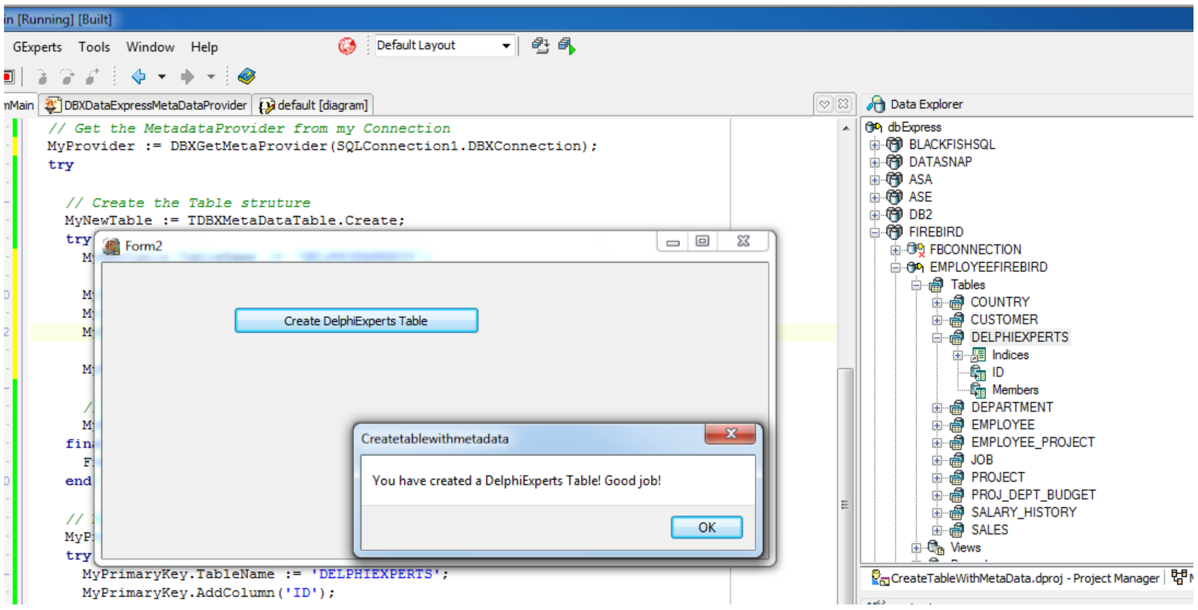


```
DBXMetaDataProvider.TDBXMetaDataProvider

FExecuter:TDBXSqlExecution
FWriter:TDBXMetaDataWriter

+ CheckColumnSupported:Boolean
+ Create
+ CreateForeignKey
+ CreateIndex
+ CreatePrimaryKey
+ CreateTable
+ CreateUniqueIndex
+ Destroy
+ DropForeignKey:Boolean
+ DropForeignKey:Boolean
+ DropIndex:Boolean
+ DropIndex:Boolean
+ DropTable:Boolean
+ DropTable:Boolean
+ Execute
+ GetCollection:TDBXTable
+ IsCatalogsSupported:Boolean
+ IsDDLTransactionsSupported:Boolean
+ IsDescendingIndexColumnsSupported:Boolean
+ IsDescendingIndexSupported:Boolean
+ IsMixedDDLAndDMLSupported:Boolean
+ IsMultipleStatementsSupported:Boolean
+ IsSchemasSupported:Boolean
+ MakeAlterTableSql:UnicodeString
+ MakeAlterTableSql:UnicodeString
+ MakeCreateForeignKeySql:UnicodeString
+ MakeCreateIndexSql:UnicodeString
+ MakeCreateTableSql:UnicodeString
+ MakeDropForeignKeySql:UnicodeString
+ MakeDropIndexSql:UnicodeString
+ MakeDropTableSql:UnicodeString
+ QuoteIdentifierIfNeeded:UnicodeString
+ ToMemoryStorage
  GetDatabaseProduct:UnicodeString
  GetDatabaseVersion:UnicodeString
  GetIdentifierQuotePrefix:UnicodeString
  GetIdentifierQuoteSuffix:UnicodeString
  GetVendor:UnicodeString
  GetWriter:TDBXMetaDataWriter
  SetWriter

+ DatabaseProduct:UnicodeString
+ DatabaseVersion:UnicodeString
+ IdentifierQuotePrefix:UnicodeString
+ IdentifierQuoteSuffix:UnicodeString
+ Vendor:UnicodeString
Writer:TDBXMetaDataWriter
```

It is precisely these methods which are the key to being able to manipulate the database. We now add this source code to our button click method:

```
procedure TForm2.Button1Click(Sender: TObject);
var
  MyNewTable: TDBXMetaDataTable;
  MyProvider: TDBXDataExpressMetaDataProvider;
  MyPrimaryKey: TDBXMetaDataIndex;
  MyIDColumn:  TDBXInt32Column;
begin

  // Get the MetadataProvider from my Connection
  MyProvider := DBXGetMetaProvider(SQLConnection1.DBXConnection);
  try

    // Create the Table structure
    MyNewTable := TDBXMetaDataTable.Create;
    try
      MyNewTable.TableName := 'DELPHIEXPERTS';

      MyIDColumn:=TDBXInt32Column.Create('ID');
      MyIDColumn.Nullable:=false;
      MyNewTable.AddColumn(MyIDColumn);

      MyNewTable.AddColumn(TDBXAnsiCharColumn.Create('Members', 50));

      // Add the Table in the Database with my provider
      MyProvider.CreateTable(MyNewTable);
    finally
      FreeAndNil(MyNewTable);
    end;

    // Now let us create a Primary Key on the ID Field
    MyPrimaryKey := TDBXMetaDataIndex.Create;
    try
      MyPrimaryKey.TableName := 'DELPHIEXPERTS';
      MyPrimaryKey.AddColumn('ID');

      // Add the Primary Key with my provider
      MyProvider.CreatePrimaryKey(MyPrimaryKey);

    finally
      FreeAndNil(MyPrimaryKey);
    end;

  finally
    FreeAndNil(MyProvider);
  end;

Showmessage('You have created a DelphiExperts Table! Good job!');

end;
```

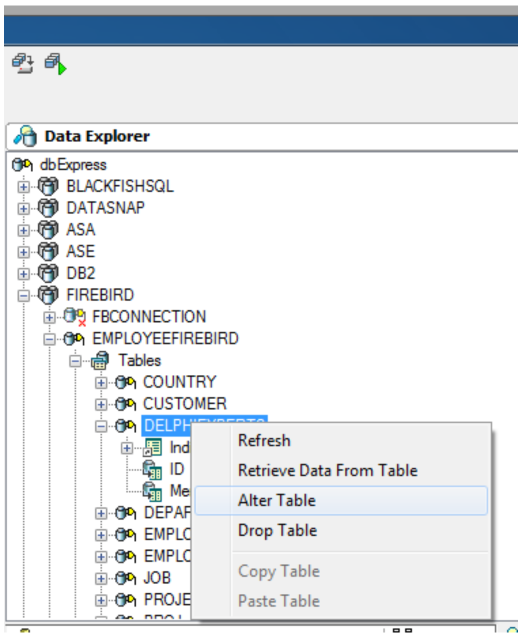
The individual steps are documented in the source code as a comment. Having compiled and started the program, we should now be able to see the new DelphiExperts table in the Data Explorer (after a refresh).

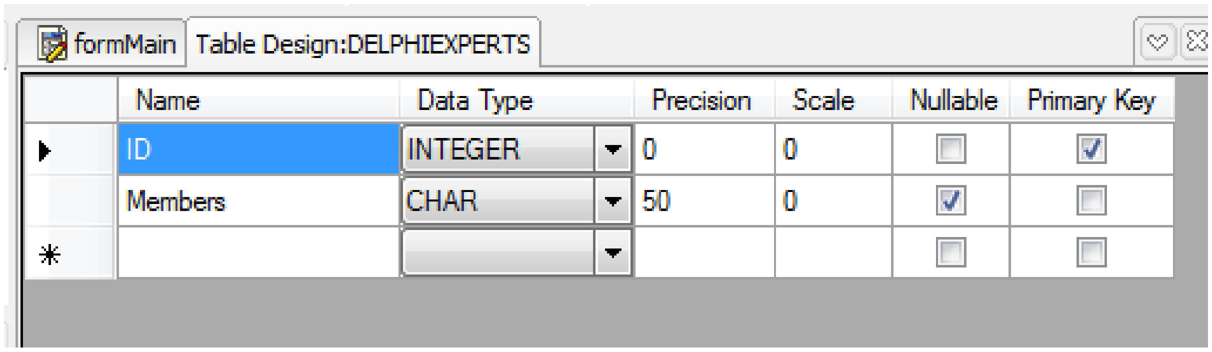


```
// Get the MetadataProvider from my Connection
MyProvider := DBXGetMetaProvider(SQLConnection1.DBXConnection);
try

  // Create the Table struture
  MyNewTable := TDBXMetaDataTable.Create;
  try
    M

    M
    M
    M

    M

    /
    M
  fin
    F
  end

  // 
  MyP
  try
    MyPrimaryKey.TableName := 'DELPHIEXPERTS';
    MyPrimaryKey.AddColumn('ID');
```

You have created a DelphiExperts Table! Good job!

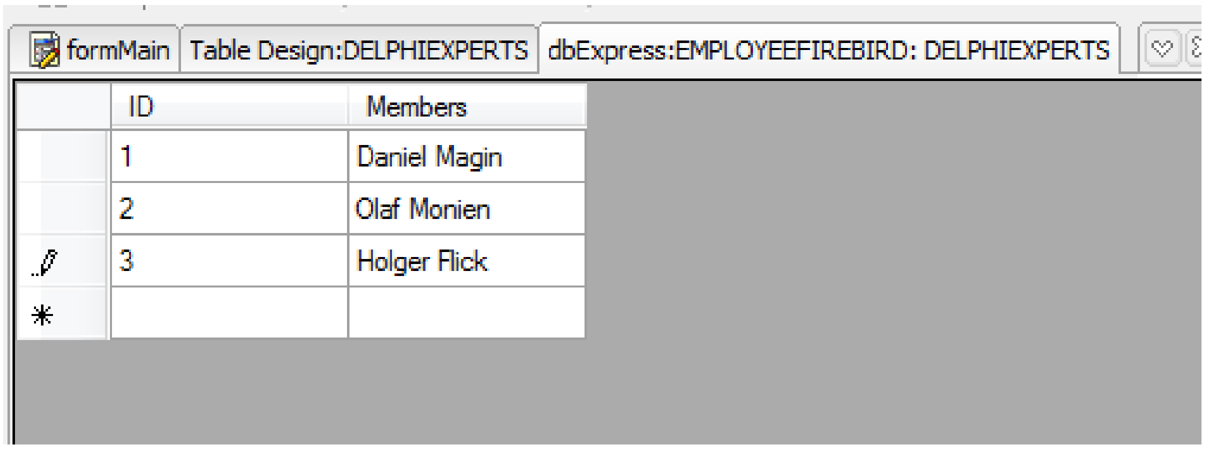
The Delphi 2010 IDE itself uses this interface in the Data Explorer so that we can set up and modify tables in the IDE. Right click on a table in the Data Explorer and you obtain the option to edit the tables at any time with “Alter Table”.




formMain | Table Design:DELPHIEXPERTS

	Name	Data Type	Precision	Scale	Nullable	Primary Key
▶	ID	INTEGER	0	0	☐	✓
	Members	CHAR	50	0	✓	☐
*					☐	☐

Double click on the table name in the Data Explorer and you can even edit, add or delete the data directly in the Delphi 2010 IDE.

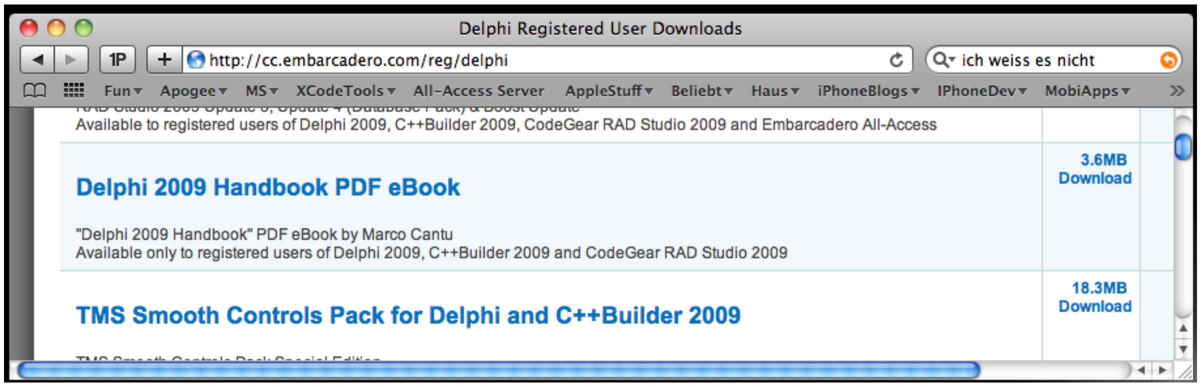

formMain | Table Design:DELPHIEXPERTS | dbExpress:EMPLOYEEFIREBIRD: DELPHIEXPERTS

	ID	Members
	1	Daniel Magin
	2	Olaf Monien
✎	3	Holger Flick
*		

DELPHI 2010, FIREBIRD AND UNICODE

As already mentioned in the introduction, there was a major revision to dbExpress in Delphi 2007. This library was originally implemented in C++and then in version 2007 was converted to Delphi, i.e. Pascal. Only a few know that dbExpress was already capable of the basic Unicode in Delphi 2007. Delphi first started to support Unicode applications in the IDE in the 2009 version. With Delphi 2010 and, of course, also the C++Builder 2010, we can create our programs with Unicode support. That said, if you port "older" Delphi programs into Delphi 2010, and therefore add Unicode support, you should take a few things into consideration:

1. If you use components made by other manufacturers, ensure that these already have Unicode support and that these are available for Delphi 2010.
2. You´ll find a detailed chapter about the changeover to Unicode in the online help – and this is very much recommended!
3. The new online Wiki from Embarcadero includes a number of articles about the Unicode conversion: http://docwiki.embarcadero.com/RADStudio/en/Unicode_in_RAD_Studio
4. Registered Delphi 2009 users can download free-of-charge the outstanding book "Delphi 2009 Handbook" by Marco Cantu – it includes a chapter about Unicode which is well worth reading.



Available to registered users of Delphi 2009, C++Builder 2009, CodeGear RAD Studio 2009 and Embarcadero All-Access

Delphi 2009 Handbook PDF eBook — 3.6MB Download

"Delphi 2009 Handbook" PDF eBook by Marco Cantu
Available only to registered users of Delphi 2009, C++Builder 2009 and CodeGear RAD Studio 2009

TMS Smooth Controls Pack for Delphi and C++Builder 2009 — 18.3MB Download

5. The last CodeRage 2009 online conference has some free-of-charge downloadable sessions that also cover Unicode.

CodeRage 4
The Totally Technical Online Conference
September 8 - 11, 2009
Watch Replays >>
• Learn from the experts
• Interact with other developers worldwide
• More than 90+ "Totally Technical" sessions
• All online! No shirt? No shoes? No problem!

http://conferences.embarcadero.com/coderage/sessions

Firebird and InterBase have offered the option of creating databases in Unicode format for some time now, so we want to create these together in an example. First of all, we require a new database in Unicode format as the example database from Firebird has not been set up in Unicode:

```
/*****************************************************************************
**/
/***         Generated by Daniel Magin - www.DelphiExperts.NET
***/
/*****************************************************************************
**/

SET SQL DIALECT 3;

SET NAMES UTF8;

CREATE DATABASE '127.0.0.1:c:\db\unicodedb.fdb'
USER 'SYSDBA' PASSWORD 'masterkey'
PAGE_SIZE 16384
DEFAULT CHARACTER SET UTF8;



/*****************************************************************************
**/
/***                              Generators
***/
/*****************************************************************************
**/

CREATE GENERATOR GEN_TBL_TEST_ID;
SET GENERATOR GEN_TBL_TEST_ID TO 0;



/*****************************************************************************
**/
/***                                Tables
***/
/*****************************************************************************
**/



CREATE TABLE TBL_TEST (
    ID           INTEGER,
    EXAMPLETEXT  VARCHAR(100)
);



/*****************************************************************************
**/
/***                               Triggers
***/
/*****************************************************************************
**/
```

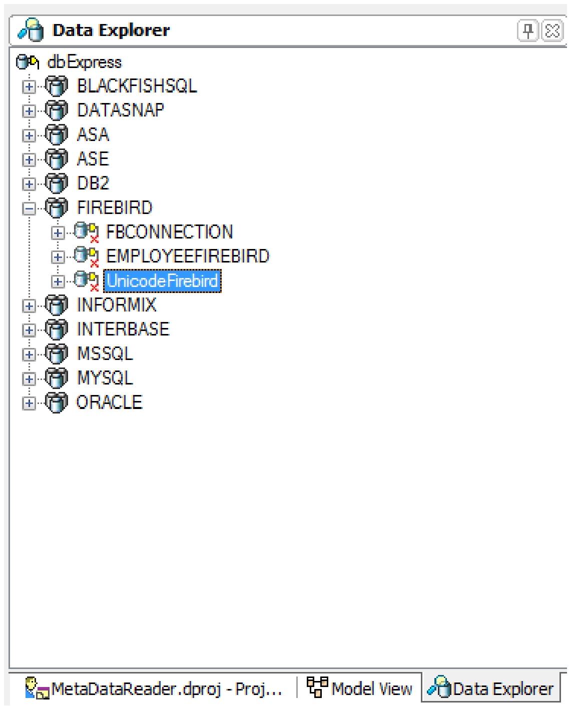
```
SET TERM ^ ;


/************************************************************************
**/
/***                         Triggers for tables
***/
/************************************************************************
**/



/* Trigger: TBL_TEST_BI */
CREATE TRIGGER TBL_TEST_BI FOR TBL_TEST
ACTIVE BEFORE INSERT POSITION 0
as
begin
  if (new.id is null) then
    new.id = gen_id(gen_tbl_test_id,1);
end
^

SET TERM ; ^
```

This line is important:

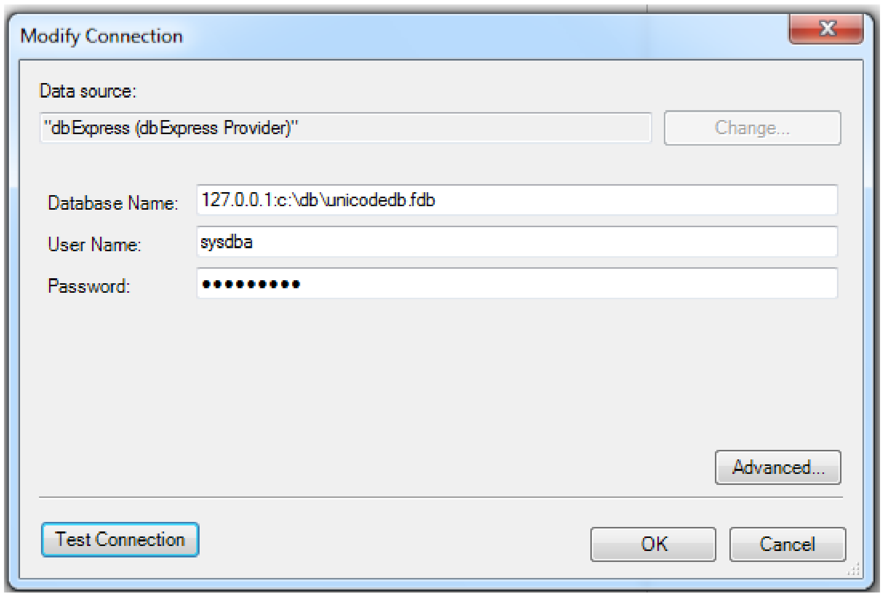
```
DEFAULT CHARACTER SET UTF8;
```

This is where the new database using this Unicode character set is created. We can also execute this with an InterBase database. There are a lot of Unicode character sets, but I would always recommend using UTF-8 if your client doesn’t wish a specific one. The Latin characters as in the ASCII character set are saved with 8 bit in the UTF-8. This means your database remains slim and more bytes will only be used for saving purposes if Unicode characters like Japanese, etc. will be used. For more information, refer also to: http://unicode.org/.

In the Delphi 2010 Data Explorer we now create a new Firebird connection with the name UnicodeFirebird.



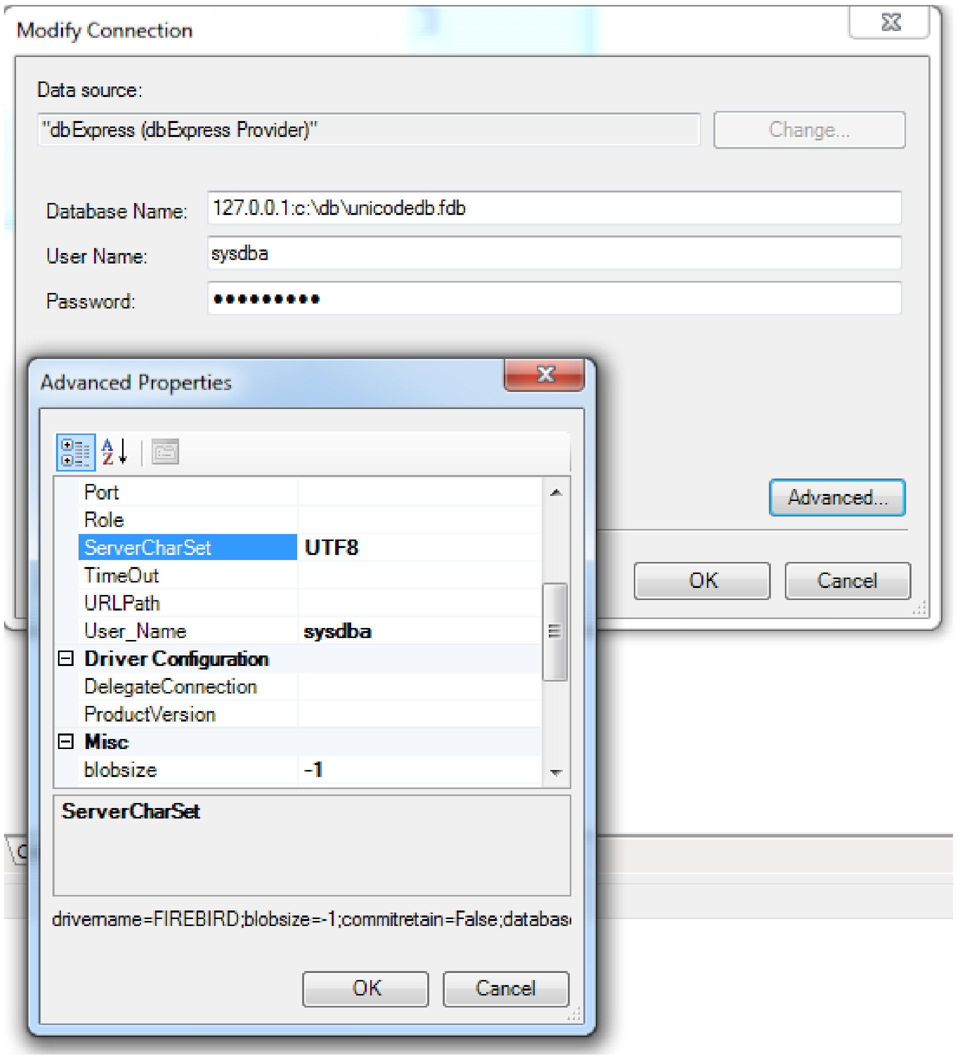
We insert the following values in the "Modify Connection" dialogue:



Data source:
"dbExpress (dbExpress Provider)"

Database Name: 127.0.0.1:c:\db\unicodedb.fdb
User Name: sysdba
Password: ••••••••••

It is now important to add the correct character set (namely UTF-8) in the "Advanced" area. Without this setting, Unicode characters will not be displayed or saved correctly. Generally speaking, in the client connection you must always have available the correct character set of your database.


Modify Connection

Data source:

"dbExpress (dbExpress Provider)"

Database Name: 127.0.0.1:c:\db\unicodedb.fdb

User Name: sysdba

Password: •••••••••

Advanced Properties

Port	
Role	
ServerCharSet	UTF8
TimeOut	
URLPath	
User_Name	sysdba
Driver Configuration	
DelegateConnection	
ProductVersion	
Misc	
blobsize	-1

ServerCharSet

drivername=FIREBIRD;blobsize=-1;commitretain=False;databas

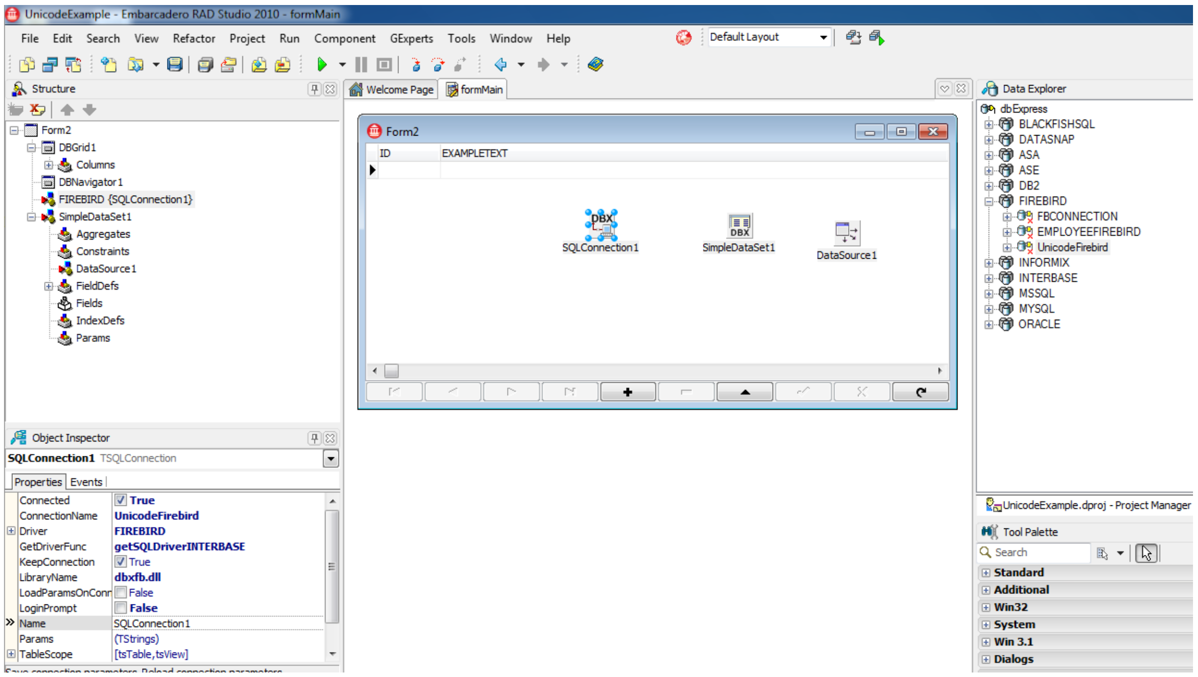
In a new VCL form application we now want to enter text with Unicode characters in the TBL_TEST table. As already shown in the previous examples, we require the following components:

1. TSQLConnection
   a. Property Connectionname: UnicdodeFirebird
   b. Property LoginPrompt: False
   c. Property Connected: True
2. TSimpleDataSet
   a. Property Connection: SQLConnection1
   b. Property DataSet->CommandText: select * from TBL_TEST
   c. Property Active: True
3. TDataSource
   a. Property DataSet: SimpleDataSet1
4. TDBGrid
   a. Property DataSet=SimpleDataSet1
5. TDBNavigator
   a. Property DataSet: SimpleDataSet1

We insert the following code in event SimpleDataSet1->AfterPost so that our changes can also be saved in the database.

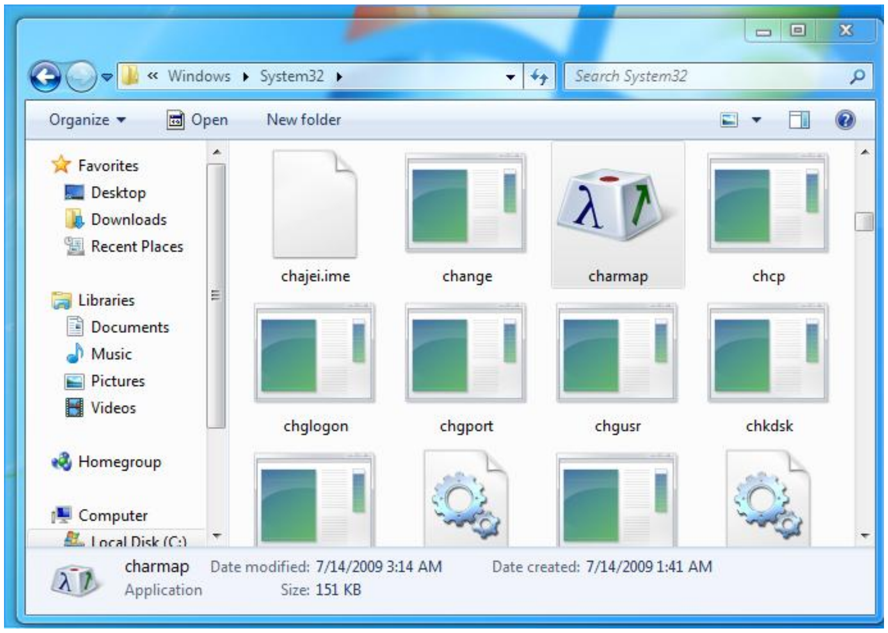
```
procedure TForm2.SimpleDataSet1AfterPost(DataSet: TDataSet);
begin
  SimpleDataSet1.ApplyUpdates(0);
end;
```

The whole thing should then look like this:



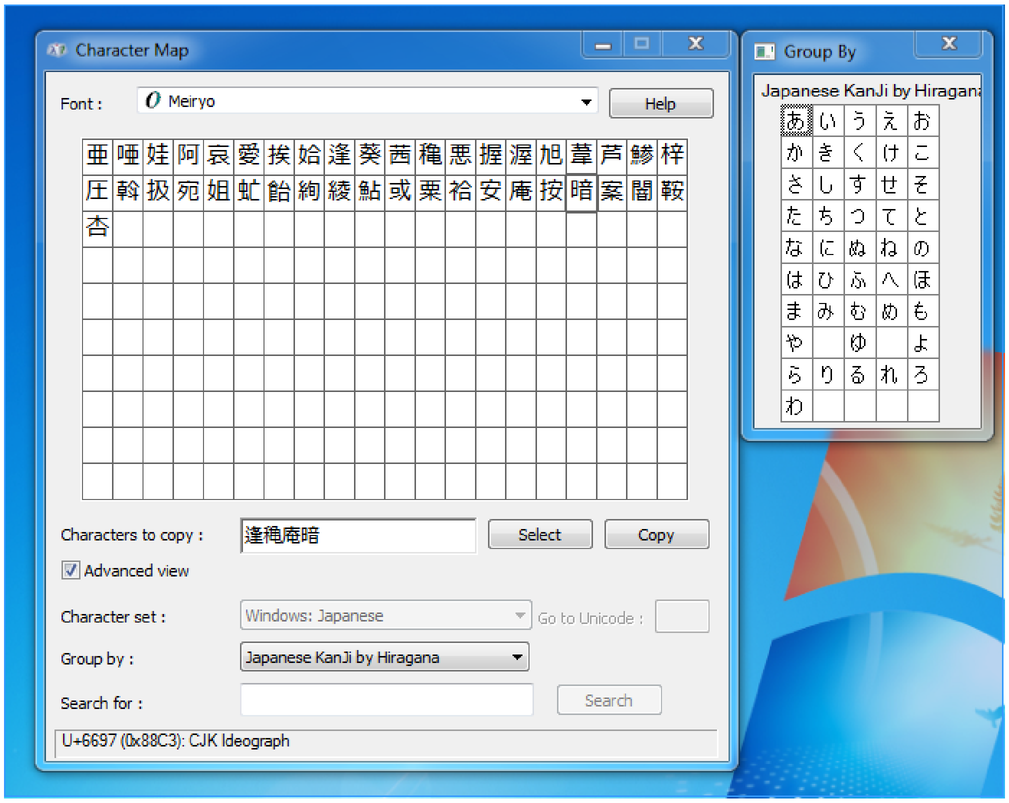
Connected	True
ConnectionName	UnicodeFirebird
Driver	FIREBIRD
GetDriverFunc	getSQLDriverINTERBASE
KeepConnection	True
LibraryName	dbxfb.dll
LoadParamsOnConn	False
LoginPrompt	False
Name	SQLConnection1
Params	(TStrings)
TableScope	[tsTable,tsView]

After compiling and starting our new application, the Windows program, Charmap, helps us to select Unicode characters with ease. We find Charmap in the C:\Windows\System32 directory.



charmap Date modified: 7/14/2009 3:14 AM Date created: 7/14/2009 1:41 AM
Application Size: 151 KB

To select Japanese characters, we now choose the following settings in Charmap:



Font : Meiryo

亜唖娃阿哀愛挨姶逢葵茜穐悪握渥旭葦芦鯵梓
圧斡扱宛姐虻飴絢綾鮎或粟袷安庵按暗案闇鞍
杏

Characters to copy : 逢穐庵暗

[✓] Advanced view

Character set : Windows: Japanese

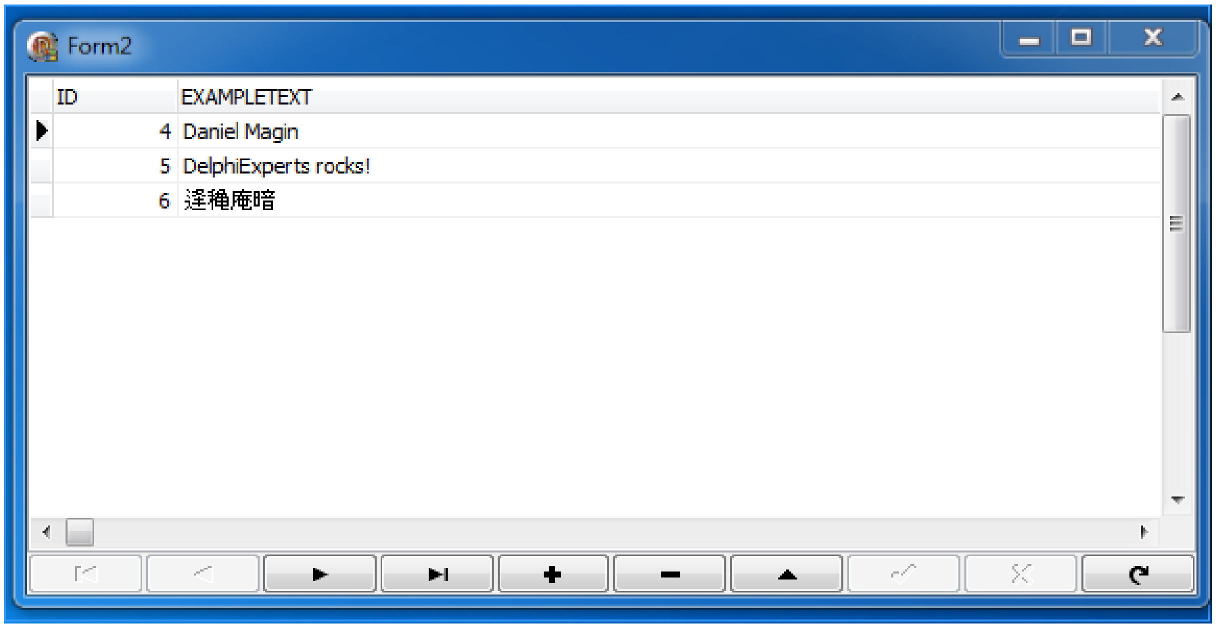
Go to Unicode :

Group by : Japanese KanJi by Hiragana

Search for :

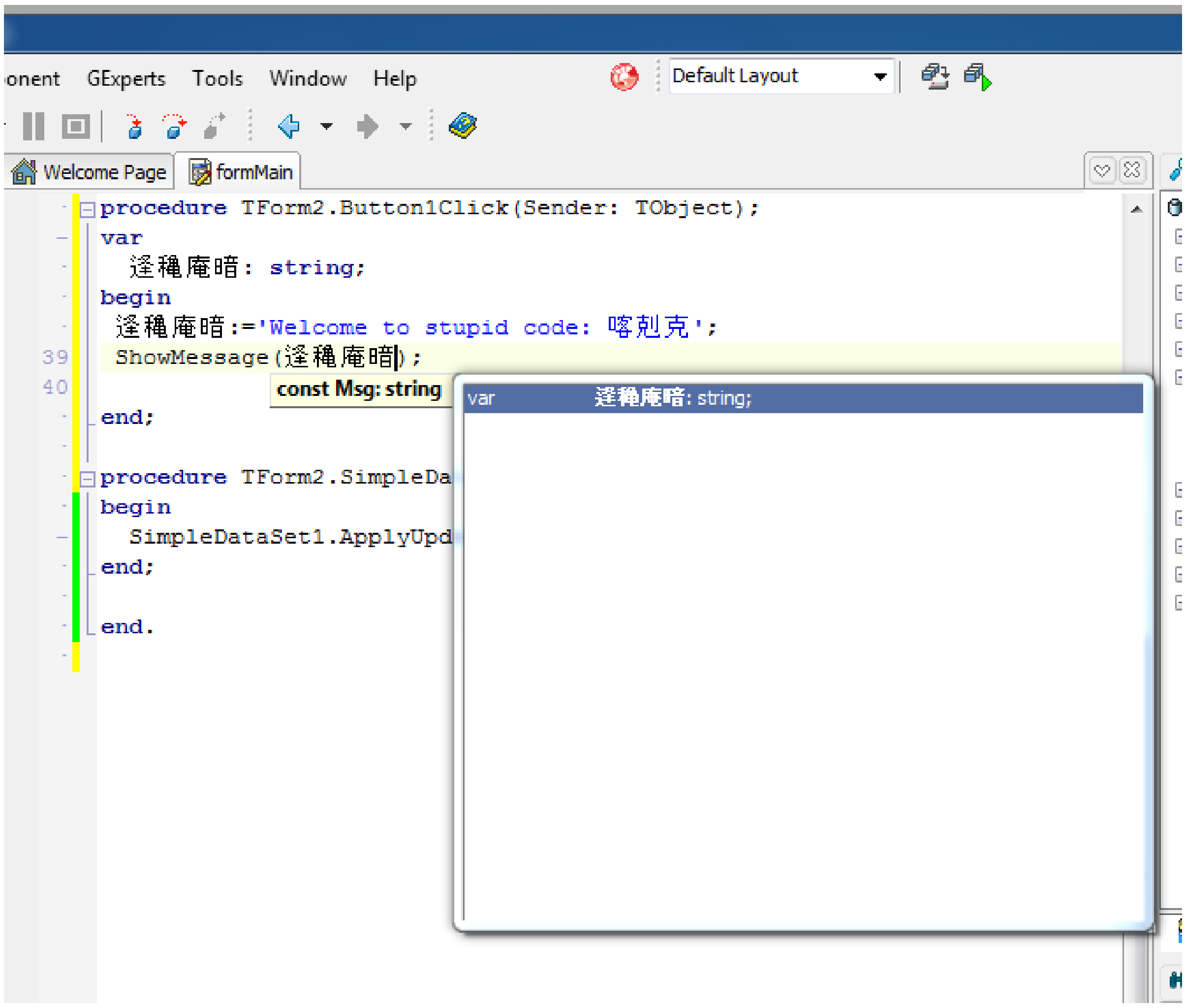
U+6697 (0x88C3): CJK Ideograph

Double click on the individual characters and these are inserted in the text field and from there we can copy these characters. Then we simply insert the content of the clipboard in our grid.



ID	EXAMPLETEXT
4	Daniel Magin
5	DelphiExperts rocks!
6	逢穐庵暗

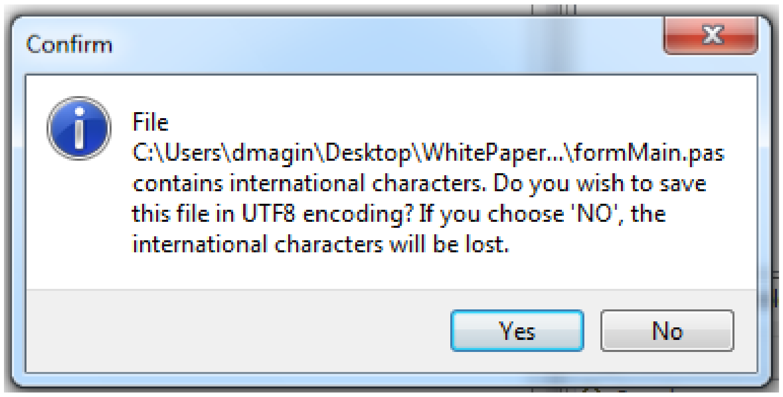
And this way, we have the Unicode character set and therefore the possibility of recording any character in our database respectively. This therefore demonstrates that the new dbExpress provider from Embarcadero for Firebird is also Unicode capable. It should also be mentioned that the Delphi IDE and also the pas files are of course Unicode-capable and therefore Unicode characters can also be used in the source code. However, when saving the file you will be prompted to indicate if you wish to save the file in UFT-8 encoding.



```
procedure TForm2.Button1Click(Sender: TObject);
var
  逶穭庵暗: string;
begin
 逶穭庵暗:='Welcome to stupid code: 喀剋克';
 ShowMessage(逶穭庵暗);

end;

procedure TForm2.SimpleDa
begin
  SimpleDataSet1.ApplyUpd
end;

end.
```

const Msg: string

var 逶穭庵暗: string;


Confirm

File C:\Users\dmagin\Desktop\WhitePaper...\formMain.pas contains international characters. Do you wish to save this file in UTF8 encoding? If you choose 'NO', the international characters will be lost.

Using Embarcadero Change Manager with InterBase and Firebird

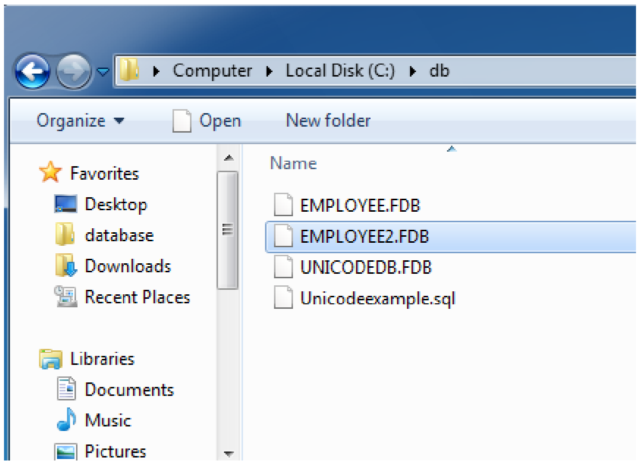
Embarcadero® Change Manager™
Version 5.1
© Copyright 2009 Embarcadero Technologies, Inc.
EMBARCADERO
POWER TO YOUR PLATFORMS

Embarcadero Change Manager™ incorporates a set of efficient tools which support us in the simplification and automation of the database change management. This is also ideal for administrators and developers who work with different test databases. The key points are:

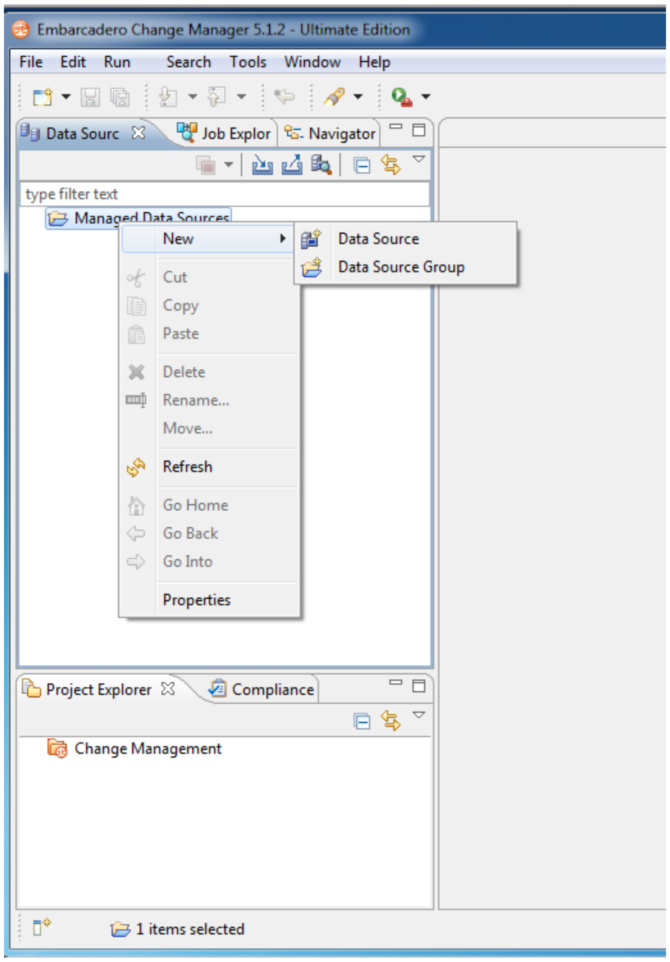
1. Simplification and automation of database change management
2. Optimisation of development cycles
3. Ensuring availability, performance and maintenance of standards and provisions

The functions of Change Manager for schema comparison and editing, data comparison and synchronization and configuration testing report database changes, install new versions and record database performance issues that result from planned and unplanned changes.
By comparing a live database with the snapshot of a schema or a configuration, administrators can quickly identify changes and remedy problems. By monitoring configuration settings, database administrators can ensure the compliance of official provisions and performance standards and maintain the general database performance and availability. Change Manager supports InterBase, Firebird, IBM® DB2® for LUW, Microsoft® SQL Server, Oracle® and Sybase®.

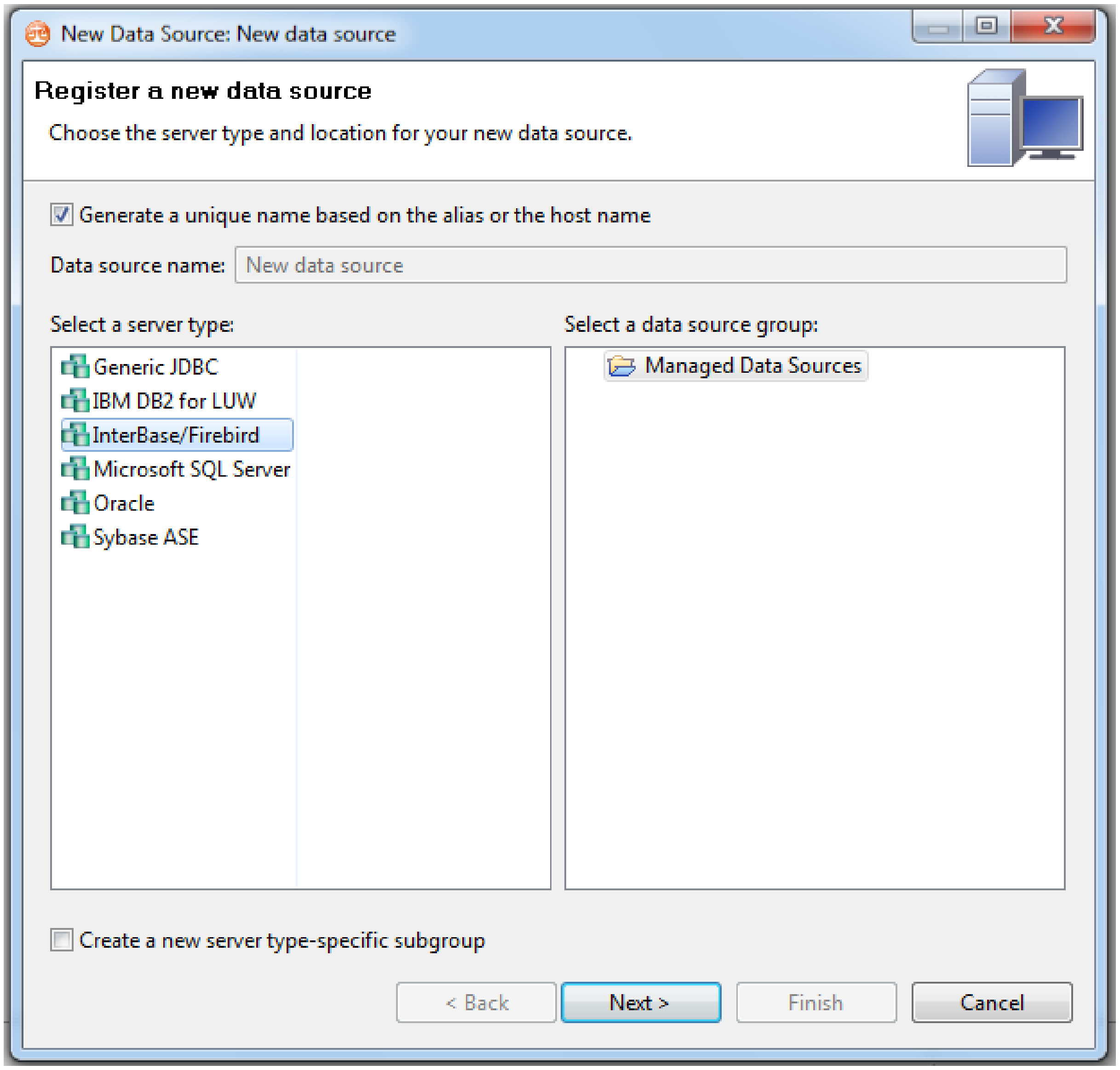
With the Change Manager we want to keep two Firebird databases in synchronization. I simply duplicated the Employee.fdb.



You can download a test version of Change Manager under http://www.embarcadero.com/products/change-manager . After the installation, we must first of all register both databases which, in this case, are located on the same computer. We can perform this task directly from the Change Manager Workbench in the DataSource area by simply selecting “New | Data Source” from the context menu.



A dialogue appears in which you can select the desired database driver- here we simply select InterBase/Firebird.



New Data Source: New data source

Register a new data source

Choose the server type and location for your new data source.

[✓] Generate a unique name based on the alias or the host name

Data source name: New data source

Select a server type:

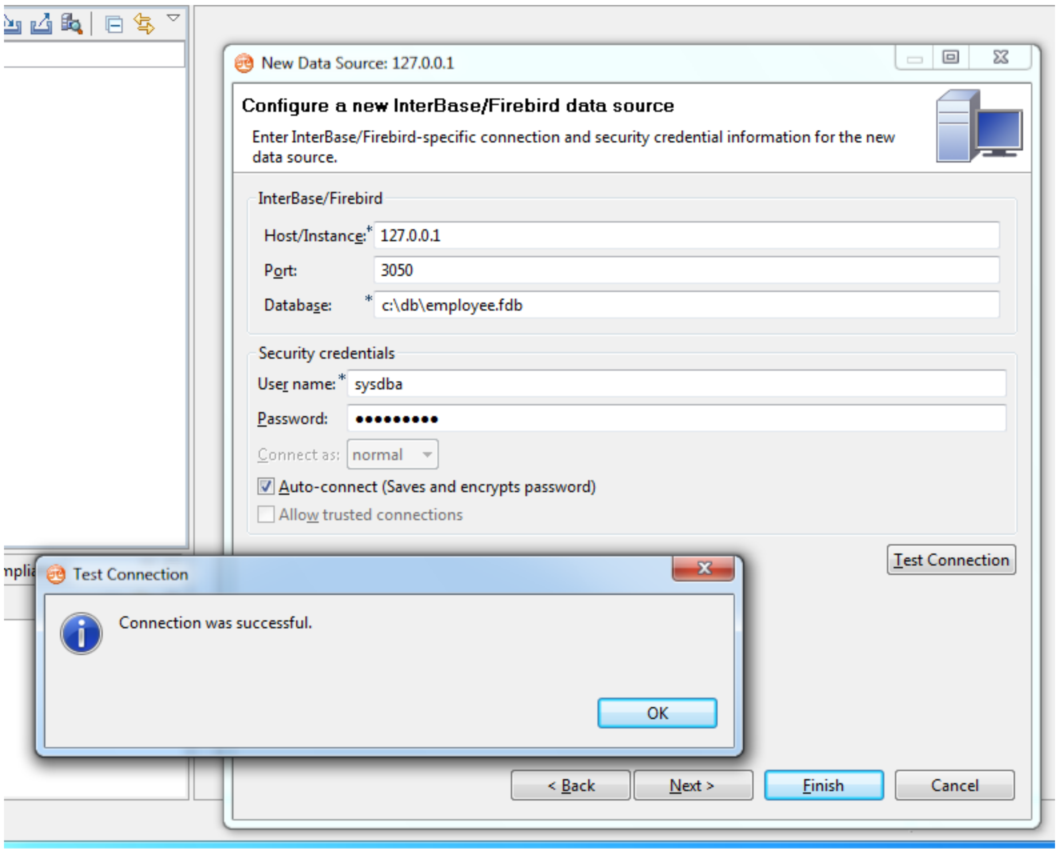
- Generic JDBC
- IBM DB2 for LUW
- InterBase/Firebird
- Microsoft SQL Server
- Oracle
- Sybase ASE

Select a data source group:

- Managed Data Sources

[ ] Create a new server type-specific subgroup

In the following dialogue we can now enter our connection string for our first connection and also perform a connection test at the same time.


New Data Source: 127.0.0.1

Configure a new InterBase/Firebird data source

Enter InterBase/Firebird-specific connection and security credential information for the new data source.

InterBase/Firebird

Host/Instance:*	127.0.0.1
Port:	3050
Database: *	c:\db\employee.fdb

Security credentials

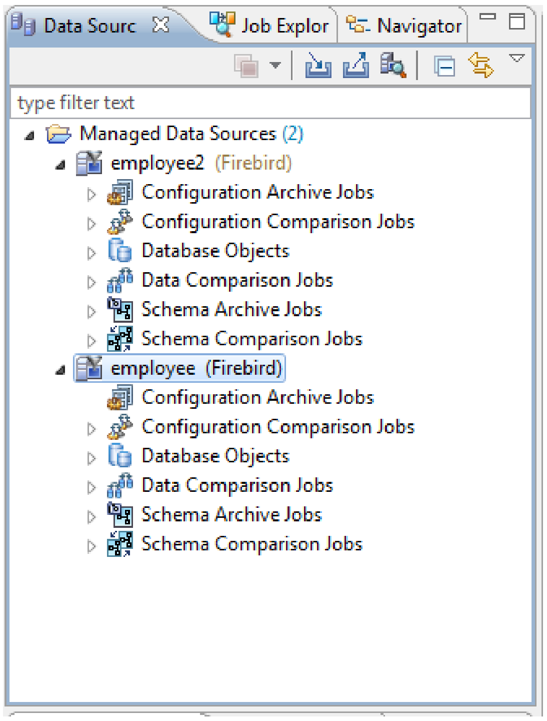
User name:*	sysdba
Password:	●●●●●●●●●
Connect as:	normal

[✓] Auto-connect (Saves and encrypts password)
[ ] Allow trusted connections

Test Connection

Connection was successful.

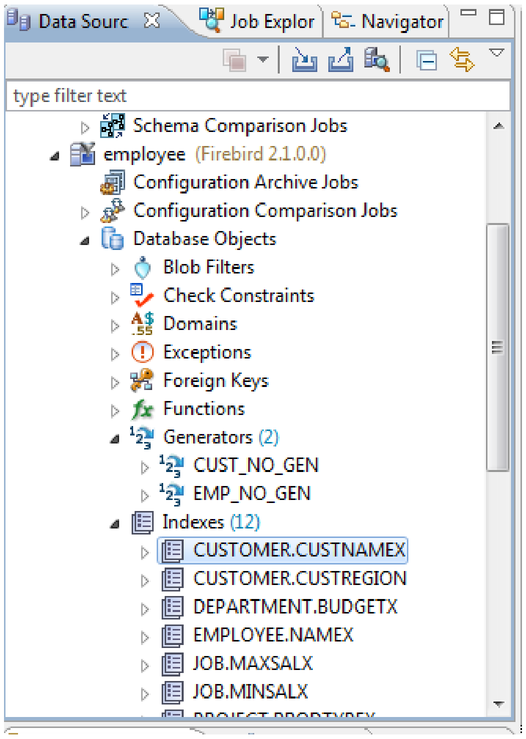
We then repeat these steps for our second database employee2.fdb. Afterwards, we should have access to both databases, and Change Manager in the Data Source area should look like this. You can rename the connections in the context menu.



```
Managed Data Sources (2)
  employee2 (Firebird)
    Configuration Archive Jobs
    Configuration Comparison Jobs
    Database Objects
    Data Comparison Jobs
    Schema Archive Jobs
    Schema Comparison Jobs
  employee (Firebird)
    Configuration Archive Jobs
    Configuration Comparison Jobs
    Database Objects
    Data Comparison Jobs
    Schema Archive Jobs
    Schema Comparison Jobs
```

In the individual sections we can define so-called jobs which can be executed or monitored for the selected database.

1. Configuration Archive Jobs
    a. Monitor database configurations
2. Configuration Comparison Jobs
    a. Keep the configurations of various servers synchronous
3. Database Objects
    a. Here we get access to the database structure in order to read the metadata of the database

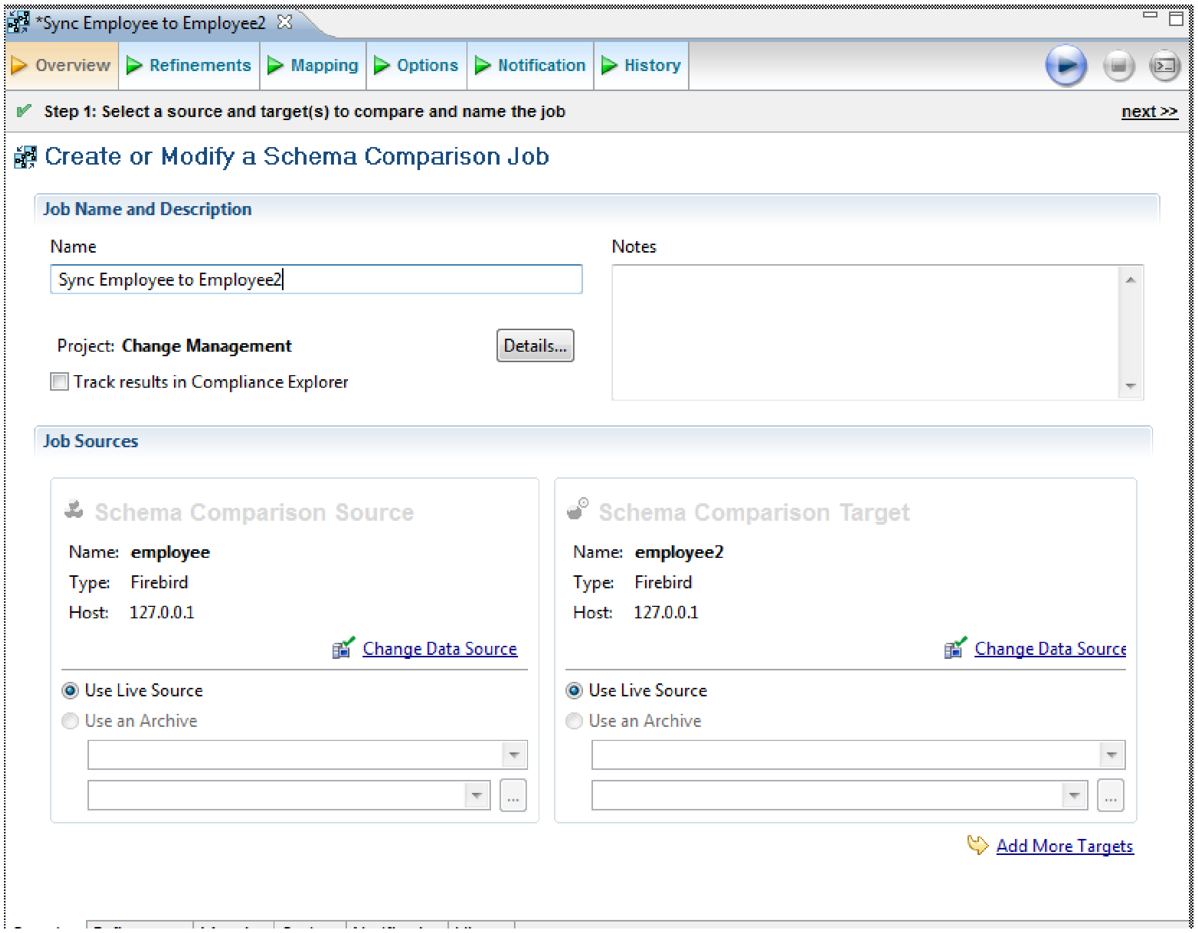


```
▷ Schema Comparison Jobs
◢ employee (Firebird 2.1.0.0)
    Configuration Archive Jobs
  ▷ Configuration Comparison Jobs
  ◢ Database Objects
    ▷ Blob Filters
    ▷ Check Constraints
    ▷ Domains
    ▷ Exceptions
    ▷ Foreign Keys
    ▷ Functions
    ◢ Generators (2)
      ▷ CUST_NO_GEN
      ▷ EMP_NO_GEN
    ◢ Indexes (12)
      ▷ CUSTOMER.CUSTNAMEX
      ▷ CUSTOMER.CUSTREGION
      ▷ DEPARTMENT.BUDGETX
      ▷ EMPLOYEE.NAMEX
      ▷ JOB.MAXSALX
      ▷ JOB.MINSALX
```

4. Data Comparison Job
    a. We use such jobs to keep the content of the data tables, i.e. the records, synchronous. Also subselects with where conditions can be generated. For example, from a master database only those records of a specific client can be compared and maybe synchronised in another database
5. Schema Comparison Job
    a. This is where you can synchronise the structure of the database, i.e. if, for example, a new table is to be set up in employee, this should also be automatically synchronized in the employee 2

It is exactly this scenario (5) that we want to execute now, and to do this we select the command "Create New" from the context menu under "Schema Comparison". A wizard then appears for the settings of the job:

1. The database connections are defined in the Overview area.



*Sync Employee to Employee2

Step 1: Select a source and target(s) to compare and name the job

Create or Modify a Schema Comparison Job

Job Name and Description

Name: Sync Employee to Employee2

Notes

Project: **Change Management**

☐ Track results in Compliance Explorer

Job Sources

Schema Comparison Source

Name: **employee**
Type: Firebird
Host: 127.0.0.1

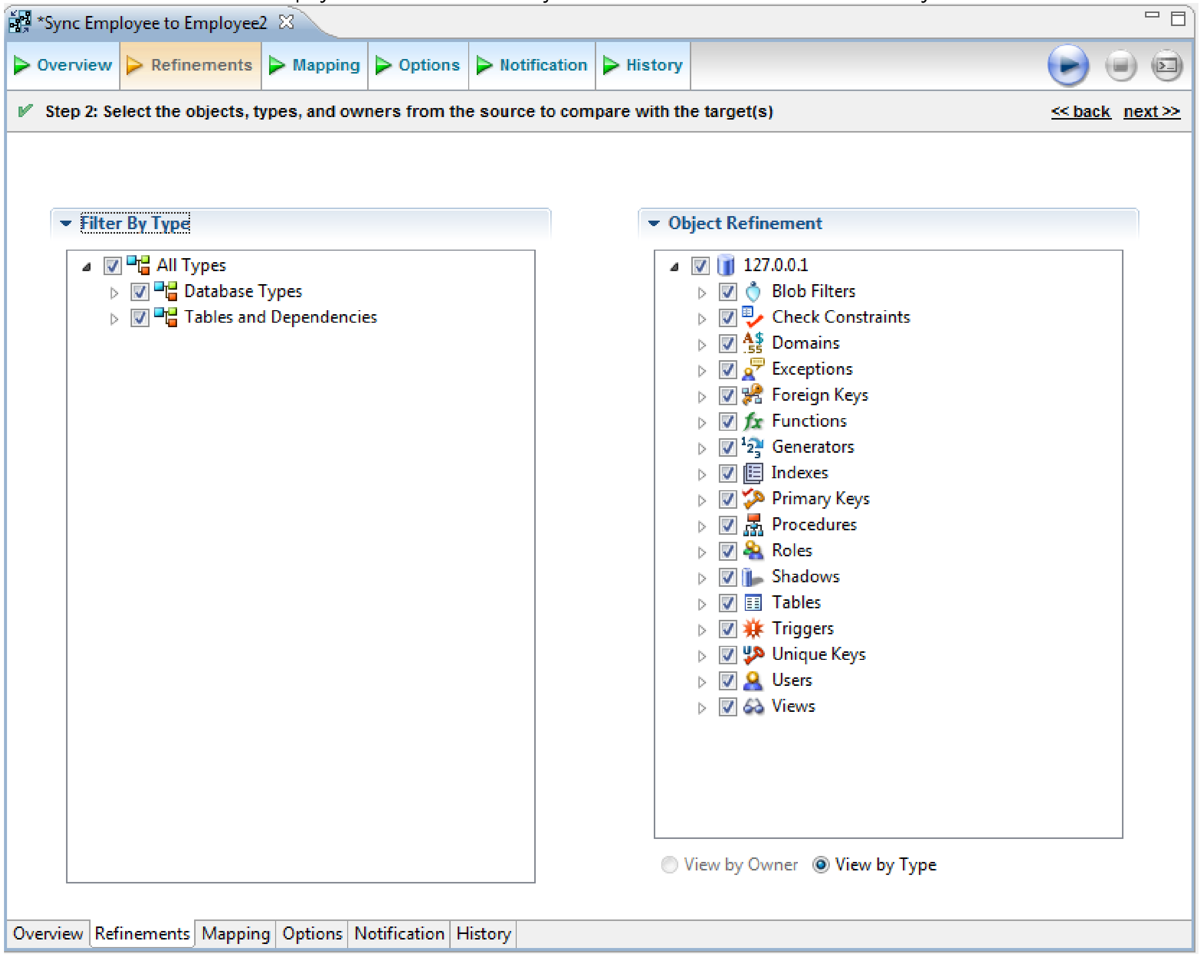
[✓ Use Live Source] Use an Archive

Schema Comparison Target

Name: **employee2**
Type: Firebird
Host: 127.0.0.1

[✓ Use Live Source] Use an Archive

2. In Refinements we simply deselect the objects that we do not wish to synchronise.



Step 2: Select the objects, types, and owners from the source to compare with the target(s)

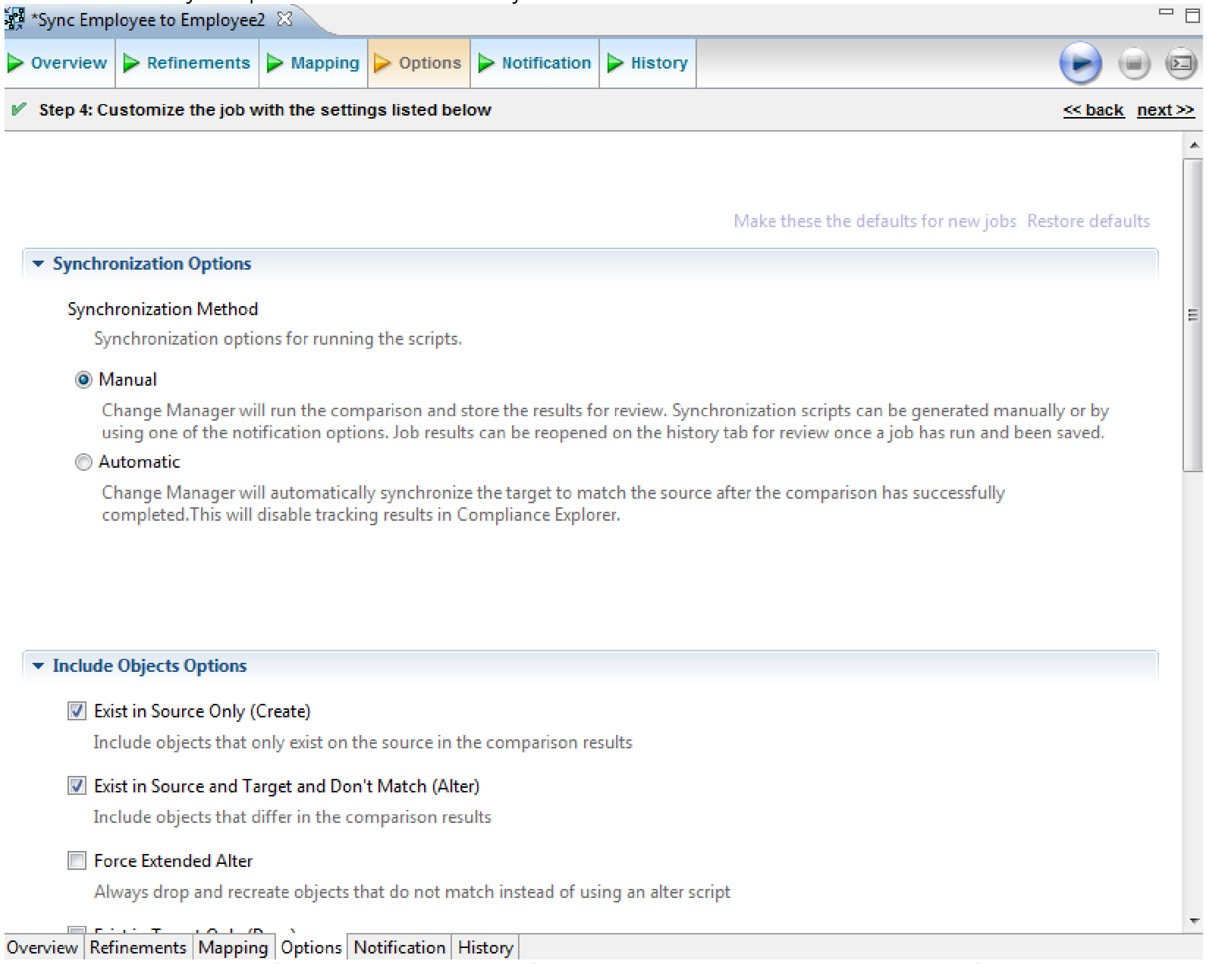
Filter By Type:
- [✓] All Types
  - [✓] Database Types
  - [✓] Tables and Dependencies

Object Refinement:
- [✓] 127.0.0.1
  - [✓] Blob Filters
  - [✓] Check Constraints
  - [✓] Domains
  - [✓] Exceptions
  - [✓] Foreign Keys
  - [✓] Functions
  - [✓] Generators
  - [✓] Indexes
  - [✓] Primary Keys
  - [✓] Procedures
  - [✓] Roles
  - [✓] Shadows
  - [✓] Tables
  - [✓] Triggers
  - [✓] Unique Keys
  - [✓] Users
  - [✓] Views

View by Owner [✓ View by Type]

3. Mapping is not required for InterBase/Firebird, so we can skip this area.
4. Under Options we can set further settings, in this example we´ll just leave everything as it is, but note that the job is set to Manual Synchronization Method and not Automatic –

so this will only be processed once the job has been called.



Step 4: Customize the job with the settings listed below

Make these the defaults for new jobs Restore defaults

Synchronization Options

Synchronization Method

Synchronization options for running the scripts.

● Manual

Change Manager will run the comparison and store the results for review. Synchronization scripts can be generated manually or by using one of the notification options. Job results can be reopened on the history tab for review once a job has run and been saved.

○ Automatic

Change Manager will automatically synchronize the target to match the source after the comparison has successfully completed.This will disable tracking results in Compliance Explorer.

Include Objects Options

[✓] Exist in Source Only (Create)

Include objects that only exist on the source in the comparison results

[✓] Exist in Source and Target and Don't Match (Alter)

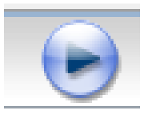
Include objects that differ in the comparison results

[ ] Force Extended Alter

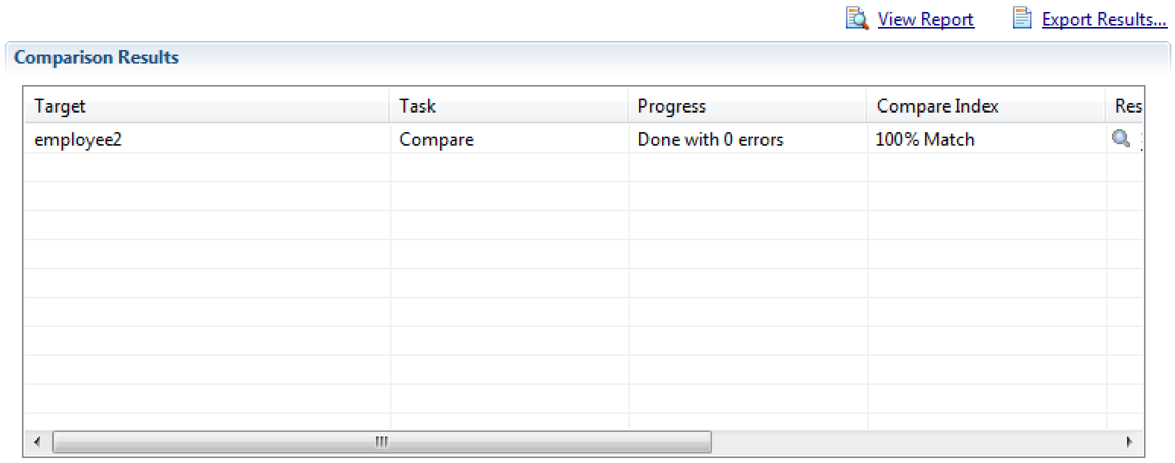
Always drop and recreate objects that do not match instead of using an alter script

5. We can use the Notifier area to arrange for an email to be sent to us, e.g. if an error occurs during the job execution.

We are now ready and can run the job by clicking on the Start button which is located at the top right of the window.



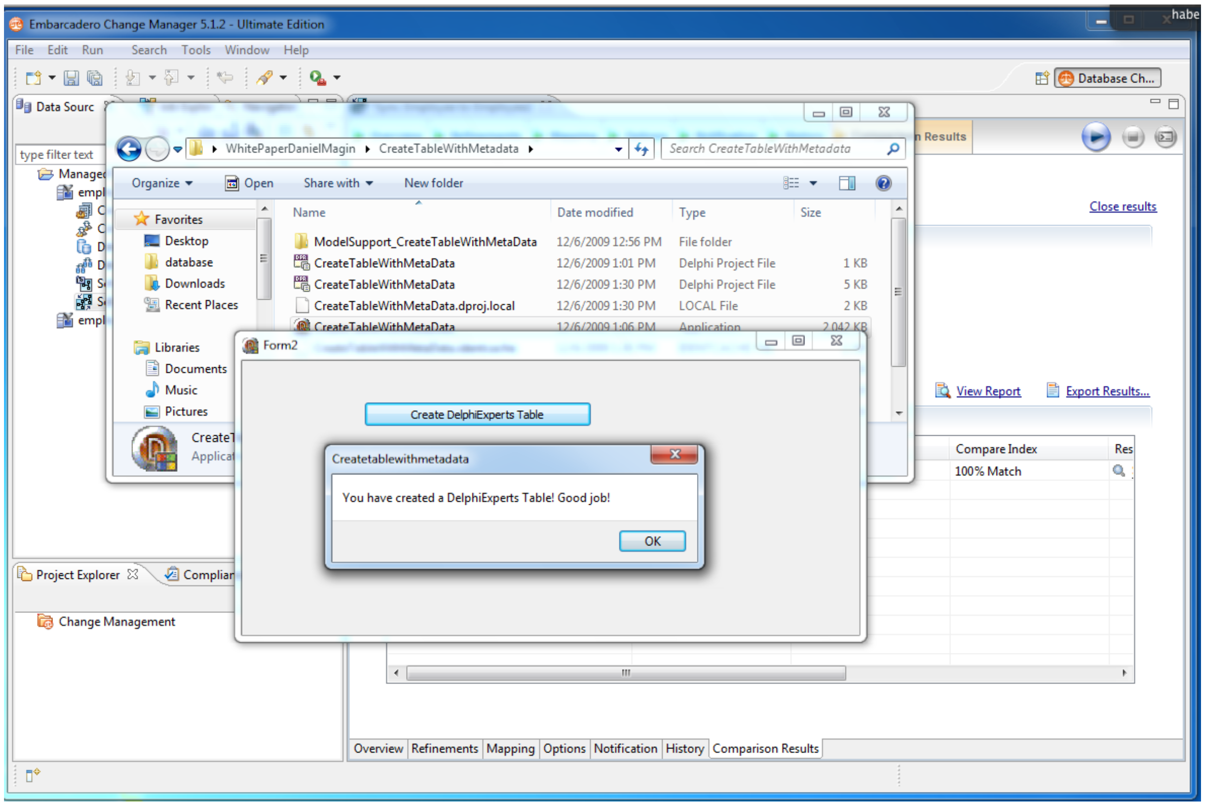
As both databases have been identical so far, a 100% match should appear.



Comparison Results

Target	Task	Progress	Compare Index	Res
employee2	Compare	Done with 0 errors	100% Match	

Remember the program we used with metadata mechanisms for setting up a new table? We use that again here. We want to set up another table DelphiExperts in our Employee database.



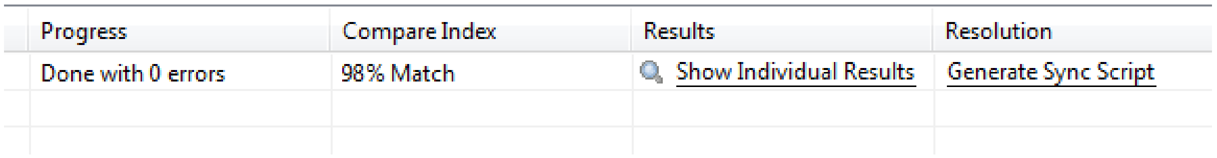
Embarcadero Change Manager 5.1.2 - Ultimate Edition

Name	Date modified	Type	Size
ModelSupport_CreateTableWithMetaData	12/6/2009 12:56 PM	File folder	
CreateTableWithMetaData	12/6/2009 1:01 PM	Delphi Project File	1 KB
CreateTableWithMetaData	12/6/2009 1:30 PM	Delphi Project File	5 KB
CreateTableWithMetaData.dproj.local	12/6/2009 1:30 PM	LOCAL File	2 KB
CreateTableWithMetaData	12/6/2009 1:06 PM	Application	2,042 KB

Createtablewithmetadata

You have created a DelphiExperts Table! Good job!

Now we start the comparison again and see that



Progress	Compare Index	Results	Resolution
Done with 0 errors	98% Match	Show Individual Results	Generate Sync Script

Change Manager has precisely detected that the structure of both Firebird databases are not identical. If we enable Generate Sync Script, Change Manager generates the correct SQL script to use so that we can transfer the changes to the other database.

```
-----------------------------------------------------------------------------
---
-- Embarcadero Change Manager Synchronization Script
-- FILE                : Alter DDL for employee2 ( vs employee )
-- DATE                : Dec 8, 2009 12:43:09 AM
--
-- SOURCE DATA SOURCE  : employee
-- TARGET DATA SOURCE  : employee2
-----------------------------------------------------------------------------
---
CREATE TABLE DELPHIEXPERTS
(
    ID         integer   NOT NULL,
    MEMBERS    char(50)  DEFAULT NULL
)

;

ALTER TABLE DELPHIEXPERTS
    ADD CONSTRAINT INTEG_82
    PRIMARY KEY (ID)

;
```

Now we´ve covered some of the topics of Change Manager with InterBase and Firebird. Refer to the Embarcadero web site at http://www.embarcadero.com/products/change-manager for more information and tutorials.

Conclusion

dbExpress and the latest Firebird driver from Embarcadero offer the developer a great possibility of programming slim and quick applications. The developer can also switch over to other databases with dbExpress support at any time and therefore no longer has to establish himself on just one database. All the examples shown can be applied very simply to all the supported databases with dbExpress Provider.

About the Author

Daniel Magin has been involved in a number of international software projects for over 20 years now. He is a trainer and consultant specialising in databases, client server architecture, multitier and web applications. Daniel has been a speaker on numerous occasions at international conferences in Europe, USA, United Arab Emirates and Asia, including topics regarding OOP, Microsoft .NET framework, Delphi, InterBase and much more. Daniel is an external software consultant for Embarcadero Inc., responsible for Delphi and InterBase, and is a founding member of the DelphiExperts with Olaf Monien and Holger Flick.