



maXbox Starter 9

Start with Data Base Programming

1.1 Creating a Query

Today we spend time in programming with Databases and some queries. So a database query is a piece of code (a query) that is sent to a database like InterBase or Oracle in order to get information back from the database. It is used as the way of retrieving information from a database.

Hope you did already work with the Starter 1 to 8 available at:

<http://www.softwareschule.ch/maxbox.htm>

This lesson will introduce you to SQL (Structured Query Language) and a database connection. The term “query” means to search, to question, or to find or research. When you query a database, you are searching for information in the database. A query component encapsulates a SQL statement that is used in a client application to retrieve, insert, update, and delete data from one or more database tables. Query components can be used with remote database servers (Client/Server) and many other database drivers.

Most often you use queries to select the data that a user should see in your application, just as you do when you use a table component. Queries, however, can also perform update, insert, and delete operations as well as retrieving records for display. When you use a query to perform insert, update, and delete operations, the query ordinarily does not return records for viewing. Example [196_](#) is such an executable query.


Let's begin with the application structure process in general of a query:

1. Configure your SQL statement
2. Connect to a database
3. Get data with a
 - SQLDataSet
 - SQLQuery
4. Show the data with a loop through a record count .

If the query data is to be used with visual data controls, you need a data source component but in our app we go with the result straight to the output in a shell like manner.

1.2 Code with SQL

As you already know the tool is split up into the toolbar across the top, the editor or code part in the centre and the output window like an SQL result at the bottom.

 Before this starter code will work you will need to download maXbox from the website. It can be down-loaded from <http://sourceforge.net/projects/maxbox> site. Once the download has finished, unzip the file, making sure that you preserve the folder structure as it is. If you double-click `maxbox3.exe`

the box opens a default program. Make sure the version is at least 3.2 because SQL need that. Test it with F9 or press **Compile** and you should hear a sound a browser will open. So far so good now we'll open the example:

195_SQL_DBExpress2.txt

If you can't find the file use the link:

http://www.softwareschule.ch/examples/195_SQL_DBExpress2.txt


Or you use the *Save Page as...* function of your browser¹ and load it from `examples` (or wherever you stored it). One important thing: You need some installation and other driver found in the directory of InterBase or Firebird or you can try with another DB installation. But you must adapt the following connection in relation to the installed database. Now let's take a look at the code of this project first with the connection set. Our first line is

```
08 Program SQL_DataBaseDemo;
```

We have to name the program it's called `SQL_DataBaseDemo`. Next we jump to line 60:

```
65 begin
66   Connection:= TSQLConnection.Create(NIL);
67   try with Connection do begin
68     ConnectionName:= 'VCLScanner';
69     DriverName:= 'INTERBASE';
70     LibraryName:= 'dbxint30.dll';
71     VendorLib:= 'GDS32.DLL';
72     GetDriverFunc:= 'getSQLDriverINTERBASE';
73     Params.Add('User_Name=SYSDBA');
74     Params.Add('Password=masterkey');
75     Params.Add('Database='+ADBNAME);
76     LoginPrompt:= True;
77     Open;
```


First in line 68 we have `ConnectionName` of a `SQLConnection`. `TSQLConnection` encapsulates a `dbExpress` connection to a database server. Setting `ConnectionName` automatically sets the `DriverName` and `Params` properties to reflect the driver and connection parameters stored under that name in the `dbxconnections.ini` file.

 You do not need to deploy the `dbxconnections.ini` file with your application even if you are using named connections in `maXbox` or `Delphi`.

The database name is set in line 75 as a local file you want to load for the first time.

```
13 ADBNAME = 'D:\Program Files\Common Files\Borland Shared\Data\MASTSQL.GDB';
```

Once `LibraryName` and `VendorLib` have been set, your application does not need to rely on `dbxdrivers.ini`. (That is, you do not need to deploy `dbxdrivers.ini` with your application unless you set the `DriverName` property at runtime).

 This database example requires 3 objects of the classes: `TSQLConnection`, `TSQLQuery`, `TSQLDataSet` and the dataset is just a second way to make a command with a query. That needs an explanation:

¹ Or copy & paste

TDataSet is the ancestor for all the dataset objects that you use in your applications. 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.

Nevertheless, TDataSet defines much that is common to all dataset objects. For example, it defines the basic structure of all datasets: an array of TField components that correspond to actual columns in one or more database tables, lookup fields provided by your application, or calculated fields provided by your application.

```
20 function DataSetQuery(aconnect: TSQLConnection): TSQLDataSet;  
21 var dataset: TSQLDataSet;  
22     i: integer;  
23 begin  
24     DataSet:= TSQLDataSet.Create(self);  
25     with DataSet do begin  
26         SQLConnection:= aconnect;  
27         //find all names with SCUBA in it!  
28         CommandText:= SQLQUERY;  
29         Open;
```

And that's how to command the query on line 28 which points to a const at line 14:

```
14 SQLQUERY = 'SELECT * FROM Customer WHERE Company like "%SCUBA%";';
```

The SQL statement can be a query that contains hard-coded field names and values, or it can be a parameterized query that contains replaceable parameters that represent field values that must be bound into the statement before it is executed.

For example, this is another statement which is hard-coded:

```
SELECT * FROM Customer WHERE CustNo = 1231
```

Hard-coded statements are useful when applications execute exact, known queries each time they run. At design time or runtime you can easily replace one hardcode query with another hard-coded or parameterized query as needed.

For sure you can store or load the statement to or from a XML-file. You can also use the Assign method of the SQL property to copy the contents of a string list object into the SQL property. The Assign method automatically clears the current contents of the SQL property before copying the new statement. For example, copying a SQL statement from our memo component:

```
CustomerQuery.Close;  
CustomerQuery.SQL.Assign(Memo1.Lines);  
CustomerQuery.Open;
```

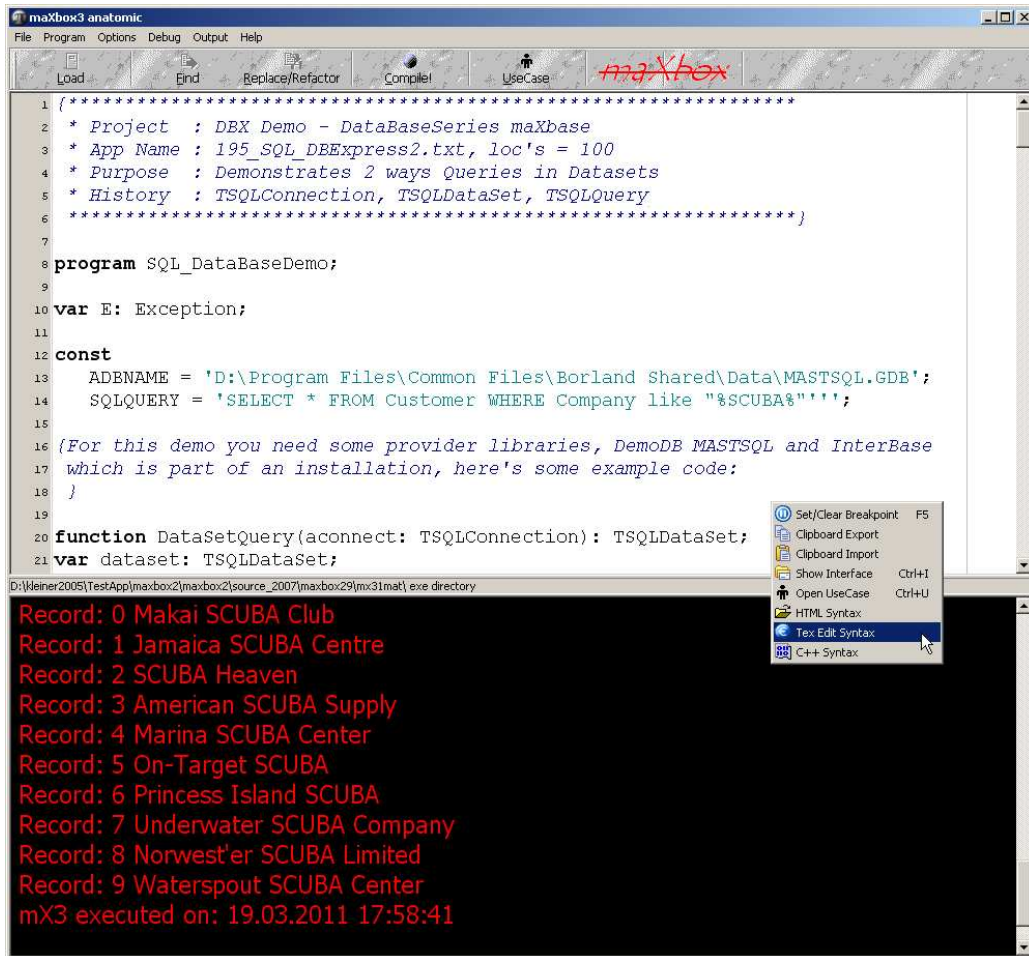
So another way (beside datasets) is to use a SQLQuery object.

Whenever the SQL property is changed the query is automatically closed and unprepared.

Before you can use a TSQLQuery component, it must be connected to the database server in line 26. Therefore, the first step to take when working with TSQLQuery is to set the SQLConnection property or to pass it as a parameter. Once the query is connected to a server, use the SQL property to specify the command your query executes.


```
44 begin  
45     qry:= TSQLQuery.Create(self);  
46     qry.SQLConnection:= aconnect;  
47     qry.SQL.Add(SQLQUERY)  
48     qry.Open;  
49     Writeln(intToStr(qry.Recordcount)+' records found')
```

If SQL is a SELECT statement, the query executes the statement when you call Open or set the Active property to true. If the statement does not return a result set, call the ExecSQL method to execute the statement. Most all the SQL you write is a query (read only) and a few are executed (write)!



1: The Result of the Query on the SQL statement

And once again you can also use the methods of SQL to load a query from a file.

 The SQL you supply for this property must be valid for the database server to which the TSQLQuery object is connected.

At runtime, use also properties and methods of strings to clear the current contents, add new contents, or to modify existing contents by a user defined SQL for example:

```
SQLQuery1.SQL.Clear;
SQLQuery1.SQL.Add('SELECT ' + Edit1.Text + ' FROM ' + Edit2.Text);
if Length(Edit3.Text) <> 0 then
    SQLQuery1.SQL.Add('ORDER BY ' + Edit3.Text)
```

Now we go to the output of the query.

In a full form application you use a data source to bind the query to a visual and data sensitive control like the famous DBGrid. A data source specifies a different table or query component that the query component can search for field names that match the names of parameters. Connect data-aware components to the data source using their DataSource and DataField properties. In our interest is the simple output in a list with the parameter [z] in line 52 which gets the fields of all records. So with the FieldCount property you have access to the number of columns with the index of parameter [z].


Although it is nice to get the information on how many records are in a dataset, calculating the `RecordCount` itself forces or can force a fetch-all. Production tables can be huge, so applications often need to limit the number of rows with which they work. For table components use filters to limit records used by an application.

```
40 function SQLReturn(aconnect: TSQLConnection): TSQLQuery;
41 var
42     //code on before
49     Writeln(intToStr(qry.Recordcount)+' records found')
50     for i:= 0 to qry.Recordcount - 1 do begin
51         for z:= 0 to qry.Fieldcount - 1 do
52             Write((qry.Fields[z].asString)+' ');
53         Writeln(#13#10)
54         qry.Next;
55     end;
56 result:= qry;
```

A query behaves in many ways very much like a table filter, except that you use the query component's `SQL` property (and sometimes the `params` property) to identify the records in a dataset to retrieve, insert, delete, or update. In some ways a query is even more powerful than a filter because it lets you access:

- More than one table at a time (called a "join" in SQL).
- A specified subset of rows and columns in its underlying table(s), rather than always returning all rows and columns. This improves both performance and security. Memory is not wasted on unnecessary data, and you can prevent access to fields a user should not view or modify.
- SQL is a more standard than a specific filter

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 in line 20 and line 40 you see both ways!

 **Query or Dataset** 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 the 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, the result set of a `SELECT` query, or the result set returned by a stored procedure.
- Execute a query or stored procedure that does not return a result set.
- Represent metadata that describes what is available on the database server (tables, stored procedures, fields in a table, and so on).

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 and therefore unidirectional. `TIBDataSet` and `TIBQuery`, on

the other hand, are intended primarily for use with `SELECT` statements. They buffer the result set, so that it is completely scrollable.

Use `TIBDataSet` or `TIBQuery` when you require use of data-aware components or a scrollable result set. In any other case, it is probably best to use `TIBSQL`, which requires much less overhead. `DataSets` is an indexed array of all active datasets (`TIBDataSet`, `TIBSQL`, `TIBTable`, `TIBQuery`, and `TIBStoredProc`) for database components. An active dataset is one that is currently open.

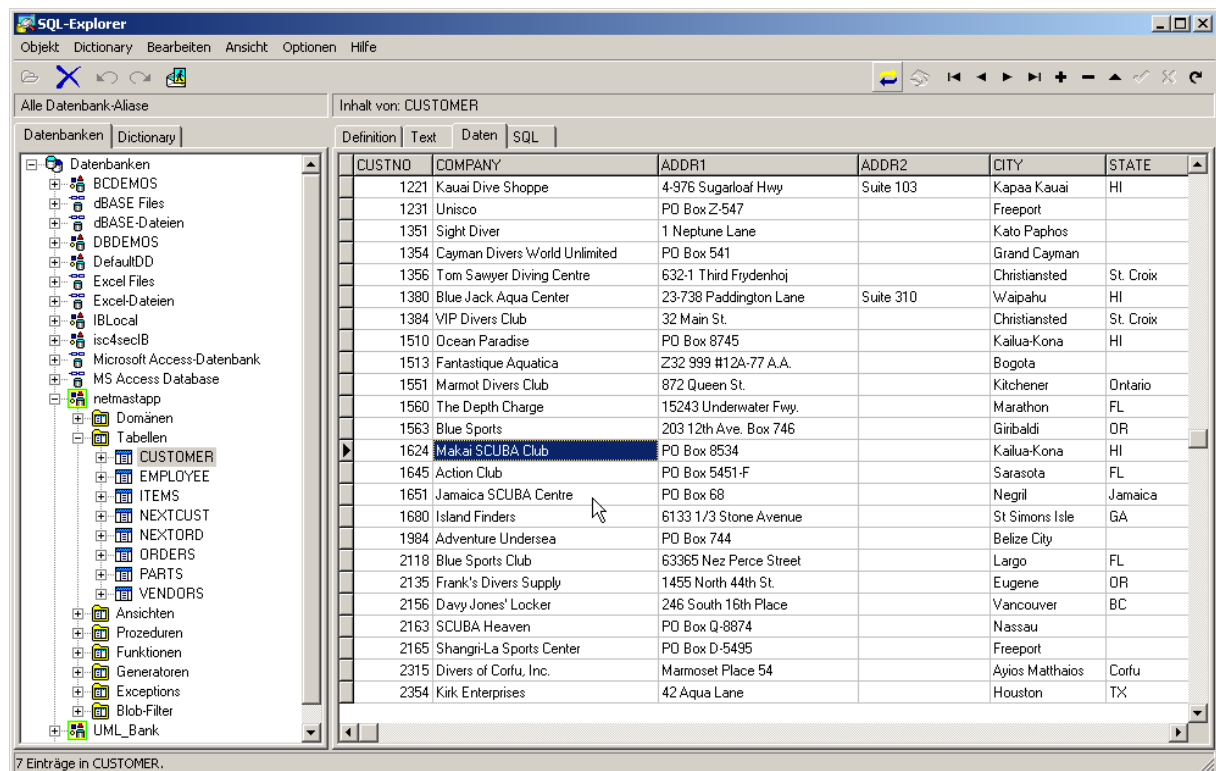
```
for i:= 0 to DataSetCount - 1 do
  if DataSets[i] is TIBTable then
    DataSets[i].CachedUpdates:= true;
```



InterBase Express (IBX) is a set of data access components that provide a means of building applications with the Borland Developer System (IDE for Delphi, C#, and C++) that can access, administer, monitor, and run the InterBase Services on InterBase databases.



SQL stands for Structured Query Language - the primary programming language used to manipulate databases. You will receive immediate results after submitting your SQL statements. You will be able to create your own unique tables as well as perform selects, inserts, updates, deletes, and drops on your tables. One of a practical way to learn more about actually writing SQL is to set your own query in line 14 and execute it with F9.



2: SQL Explorer of the Demo DB



BDE does not support a Unicode type, so `TTable` and `TQuery` are derived from `TDataSet`. However, `TADODataSet` and `TSQLDataSet` (dbExpress) derive from `TWideDataSet`, because ADO and dbExpress need to support Unicode data.

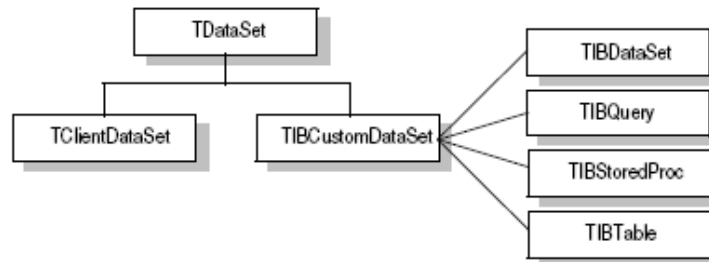



So far we have learned something about SQL coding and databases, let's make a conclusion to the structure at the beginning of our demo: Database applications are built from user interface elements, components that manage the database and components that represent the data contained by the tables in those databases (datasets). How you organize these pieces and layers is the architecture of your database application.

Many aspects of the architecture of your database application depend on the number of users who will be sharing the database information and the type of information you are working with. For most client/server applications, however, you should create your own database components instead of relying on temporary ones. You gain greater control over your databases, including the ability to


- Create persistent database connections
- Customize database server logins
- Control transactions and specify transaction isolation levels

Figure 15.1 InterBase database component dataset hierarchy



 Try to change the SELECT statement in order to get all companies with the city name “marathon” in it and to avoid case sensitivity:

```
14 SQLQUERY = 'SELECT * FROM Customer WHERE Company like "%SCUBA%";'
```

 Try to change the exception message with a string literal const or resource line:

```
82 except
83   E:= Exception.Create('SQL Connect Exception: ');
84   Showmessage(E.message+'SQL or DB connect is missing')
85 end;
```

max@kleiner.com

Links of maXbox and a to Database Resources:

<http://www.softwareschule.ch/maxbox.htm>

<http://sourceforge.net/projects/maxbox>

<http://sourceforge.net/apps/mediawiki/maxbox/>

C/S Framework with InterBase and ClientDataSets; and a model with the topics of document management, Blobs UD Functions and Web Access:

<http://max.kleiner.com/download/suche.zip>