



# maXbox Starter 12

## Start with SQL Programming V2

### 1.1 Set a Query

Today we spend another time in programming with SQL and some queries. So a database query is a piece of code (a query) that is sent to a database like InterBase, Access 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 11 available at:

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

This lesson will introduce you to SQL (Structured Query Language) and a database connection with a simple Access DB. 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 app to retrieve, insert, update, and delete data from one or more database tables. Query components can be used with remote database servers (Client/Server or ClientDataSets) and many other database drivers.

SQL is a wide topic with themes like normalization, data definition language (DDL), data manipulation language (DML), transaction control (TCL) and we just start studying a simple Select Query. SQL is thus a comprehensive language for controlling and interacting with a database management system (DBMS).

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 or the command on line 44 in our script.


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
  - SQLStoredProcedures
4. Show the data with a loop through a record count or a DBGrid.

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 but in line 55 we also use a DBGrid.

## 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 3.5 because SQL need that. Test it with F2 / F9 or press **Compile** and you should hear a sound a browser will open. So far so good now we'll open the example:

241\_db3\_sql\_tutorial2.txt

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

[http://www.softwareschule.ch/examples/241\\_db3\\_sql\\_tutorial2.txt](http://www.softwareschule.ch/examples/241_db3_sql_tutorial2.txt)

Or you use the *Save Page as...* function of your browser<sup>1</sup> and load it from *examples* (or wherever you stored it). One important thing: You need an ODBC configuration as you can see in the screenshot below and the database *database3.mdb* prepared for you in */examples*. So you have to set a data source-name and the path to */examples/ database3.mdb* like below. Then we can adapt the following connection string in relation to the prepared database. Now let's take a look at the code of this project first with the connection set. Our first line is

```
08 Program DB_ACCESS_SQL;
```


We have to name the program it's called *DB\_ACCESS\_SQL*. Next we jump to line 39:

```
39 begin
40  adoQry:= TAdoQuery.Create(self);
41  with adoQry do begin
42    cacheSize:= 500;
43    commandType:= cmdText; //ctQuery;
44    connectionString:= 'Provider=MSDASQL;DSN=mx3base;Uid=sa;Pwd=admin';
```

First in line 40 we have a *TAdoQuery* object to get an ODBC connection to a database. Setting *ConnectionString* in line 43 sets the *Provider* and *DSN* properties to reflect the driver and connection parameters stored under that name in the ODBC Data Source Administrator. The *CommandType* property on a *Command* object indicates the type of a *Command* object. Set *CommandType* to indicate the type of command that is contained in the *CommandText* property. The value in *CommandType* should agree with that of *CommandText*. For instance, if *CommandText* is the name of a table, *CommandType* should be *cmdTable* or *cmdTableDirect*. In our case it's a text as a command:

```
43  commandType:= cmdText;
44  commandText:= 'SELECT * FROM Table1';
45  SQL.Text:= commandText;
```

*CommandText* is a textual representation of the command such as an SQL statement, a table name, or the name of a stored procedure to execute. The command specified in *CommandText* is executed when the dataset's *Open* method is called.

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

---

<sup>1</sup> Or copy & paste

The database name is set in line 43 as a local file.

```
DSN = mx3base = 'Path to ../maxbox3/examples/Database3.mdb'
```



This database example requires 3 objects of the classes: TAdoQuery, TDataSource, TDBGrid and a typecast dataset in line 159 is just a second way to make a command with a query. That needs an explanation:

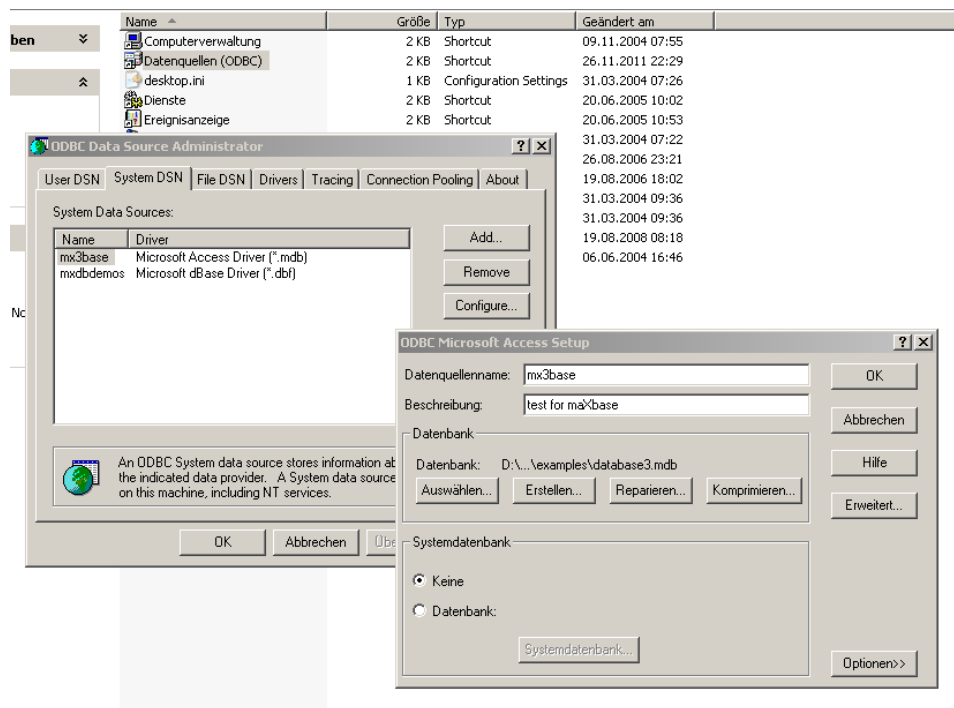
TDataSet is the ancestor for all the dataset objects that you use in your applications.

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. We use the dataset to link a data source to a visual db grid component.

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

Data controls connect to datasets by using a data source. A data source component (TDataSource) acts as a conduit between the control and a dataset containing data. Each data-aware control must be associated with a data source component to have data to display and manipulate.



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 the application 195\_SQL\_DBExpress2.txt

```
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 in line 46 and a second one which is hard-coded:

```
commandText:= 'SELECT * FROM Table1';
```

```
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)!

Here are 3 examples of real `ExecutedSQL`:

```
SQLQueryExec = 'UPDATE Maschine SET Einheit = 'min22' WHERE Nr='505';
```

```
SQLQueryExec = 'DELETE FROM maschine WHERE Nr='651';
```

```
SQLQueryExec = 'INSERT INTO Maschine (Nr,DL,Einheit,EK) '+
                'VALUES('651','maxmachine','min','120.44)';
```



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.

CommandTimeout is kicking in when you have long running queries. There is a CommandTimeout property of TADOConnection but that does not work. You have to use the CommandTimeout of the TADODataSet instead.

```

35 Procedure SetADOAccessSQL Query;
36 var //adoquery: TADODataSet;
37     adoQry: TADOQuery;
38     i,z: integer;
39 begin
40     adoQry:= TADOQuery.Create(self);
41     with adoQry do begin
42         cacheSize:= 500;
43         connectionString:= 'Provider=MSDASQL;DSN=mx3base;Uid=sa;Pwd=admin';
44         {commandText:= 'INSERT INTO Table1 (FirstName, LastName, Phone)'+
45             'VALUES (''Max99'', ''Box5459'', ''031-333 77 88'')';}
46         commandText:= 'SELECT * FROM Table1';
47     end;
48     WriteLn(intToStr(Recordcount)+' records found: ');
49     for i:= 0 to Recordcount - 1 do begin
50         for z:= 0 to Fieldcount - 1 do
51             Write((Fields[z].AsString)+' ');
52         WriteLn(#13#10);
53     end;
54 end;
55 CreateDBGridForm(adoQry);
56 Close;
57 Free;
58 end;
59 end;

```

```

15 records found:
1 Thomas Hardy (21) 555-3412

2 Maria Anders (91) 745 6200

3 Ann Devon 981-443655

4 Rene Phillips (171) 555-7733

5 Anabela Petersen 0372-035188

```

1: The Result of the Query on the SQL statement

And you can also use the methods of SQL to load a stored 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 (call in line 55) 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 51 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 app.

```

35 procedure SetADOAccessSQL_Query;
36 var
42 //code on before
48 Writeln(intToStr(qry.Recordcount)+' records found')
49 for i:= 0 to qry.Recordcount - 1 do begin
50   for z:= 0 to qry.Fieldcount - 1 do
51     Write((qry.Fields[z].asString)+' ');
52   Writeln(#13#10)
53   qry.Next;
54 end;
55 CreateDBGridForm(adoQry); //call to DBGrid!

```

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

Finally, SQL is not a particularly structured language, especially when compared to highly structured languages such as C, Pascal, or Java. Instead, SQL statements resemble English sentences, complete with "noise words" that don't add to the meaning of the statement but make it read more naturally.

Second, SQL is not really a complete computer language like Pascal, Object Pascal, C, C++, or Java. SQL contains no IF statement for testing conditions, and no GOTO, DO, or FOR statements for program flow control. Instead, SQL is a database sublanguage, consisting of about 40 statements specialized for database management tasks. These SQL statements can be embedded into another language like we do that with Pascal and SQL sounds like PaSQL.;

So the next topic is to add a record in the database e.g. in line 44:

```

commandText:= 'INSERT INTO Table1 (FirstName, LastName, Phone)+'
              'VALUES (''Max199'', ''Box5459'', ''031-333 77 88'')';

```

You have to uncomment this line and in exchange to comment line 46.

The DML statements that perform an action on data but do not return a result set are: INSERT, DELETE, and UPDATE. If you perform this SQL command a new record is added. But you should see also an error statement like this: `CommandText does not return a result set at 41.360` Means that no return is available and by deactivate or catch the open we can avoid it.

The syntax for the INSERT statement is:

```

INSERT INTO table
(column-1, column-2, ... column-n)
VALUES
(value-1, value-2, ... value-n);

```

Let's go back to the WHERE statement and the difference between a filter and a range. In general, filters are more flexible than ranges. Ranges, however, can be more efficient when datasets are large

and the records of interest to an application are already blocked in contiguously indexed groups. For very large datasets, it may be still more efficient to use the WHERE clause of a query-type dataset to select data. Try the next statement:

```
commandText:= 'SELECT * FROM Table1 WHERE FirstName = 'max''';
```

You see that just the name max results. Better is to ask with a wildcard to get all first names with the name max in it:

```
commandText:= 'SELECT * FROM Table1 WHERE FirstName like '%max%''';
```


With the new added record you should get back 10 records!

In most cases, the SQL command must be only one complete SQL statement, although that statement can be as complex as necessary (for example, a SELECT statement with a WHERE clause that uses several nested logical operators such as AND and OR).

It's also possible to create a new table with SQL. For example, the following code executes a CREATE TABLE statement (DDL) without any parameters on a TSQLConnection component:

```
SQLConnection1.Connected:= True;
SQLstmt := 'CREATE TABLE NewCusts ' + '( ' +
' CustNo INTEGER, ' +
' Company CHAR(40), ' +
' State CHAR(2), ' +
' PRIMARY KEY (CustNo) ' +
')';
SQLConnection1.Execute(SQLstmt, Nil, Nil);
```

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's 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) or ADO, BDE, dbExpress is a set of data access components that provide a means of building applications with the Delphi Studio (IDE for Delphi, C#, and C++) that can access, administer, monitor, and run the InterBase Services on InterBase databases.



A SELECT statement with a WHERE clause that uses several nested logical operators such as AND and OR can be quiet efficient.



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 46 and execute it with F9.

We also use a simple read only DBGrid. The DBGrid component has several events, most of which pertain to cell editing and data navigation and only two public methods. I won't list the events here because it is obvious from their names what functions they perform.

Int	FirstName	LastName	Phone
1	Thomas	Hardy	(21) 555-3412
2	Maria	Anders	(91) 745 6200
3	Ann	Devon	981-443655
4	Rene	Phillips	(171) 555-7733
5	Anabela	Petersen	0372-035188
6	Andrew	Fuller	(71) 555-4048
7	Max	Box	031-333 77 88
12	Max	Box4	031-333 77 88
14	Max	Box54	031-333 77 88
15	Max	Box54	031-333 77 88
16	Max	Box545	031-333 77 88
17	maxbase	minimax	103-184-8
18	Max77	Box545	031-333 77 88
19	Max99	Box5459	031-333 77 88
20	Max99	Box5459	031-333 77 88
21	Max199	Box5459	031-333 77 88

2: SQL Result 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:

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. For example:



```

SELECT CustNo, OrderNo, SaleDate
FROM Orders
WHERE CustNo = 1225
ORDER BY SaleDate

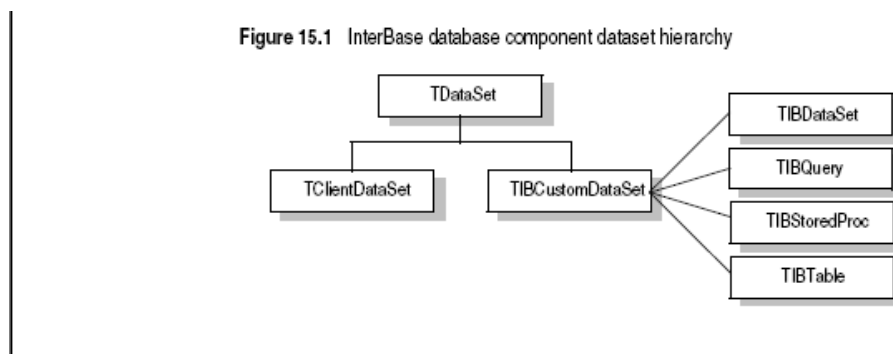
```

Queries that do not return records include statements that use Data Definition Language (DDL) or Data Manipulation Language (DML) statements other than SELECT statements (INSERT, DELETE, UPDATE, CREATE INDEX, and ALTER TABLE commands do not return any records). The language used in commands is server specific, but usually compliant with the SQL-92 standard for the SQL language.

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
- Data controls connect to datasets by using a data source.
- If a command does not return any records, you don't need to use a unidirectional dataset



Try to change the SELECT statement in order to get all phone numbers with 555 in it:

```
14 SQLQUERY = 'SELECT * FROM Table1 WHERE FirstName like '%max%'';
```

Try to change the SELECT statement in order to get all phone numbers with 555 in it and the first name is Rene (use the AND operator):

Change the SELECT statement in order to get all records sorted by Last Name and the first name is NOT Rene (use ORDER BY):

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%";
```

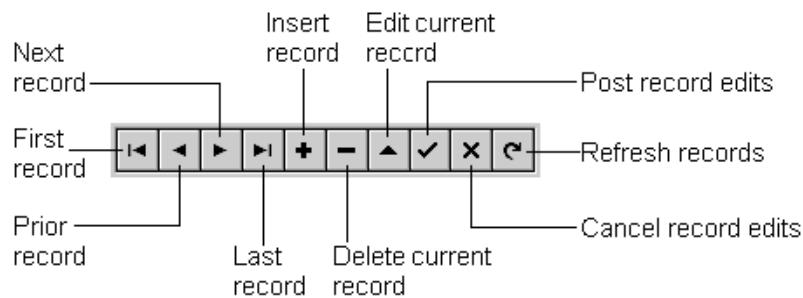
Try to change the exception message with a string literal const or resource line:

```

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

```

 Case Study: Install (Code) a DB Navigator on the form. A `DBNavigator` component enables the user to browse a dataset record by record. The navigator provides buttons for first record, next record, previous record, last record, insert record, delete record, edit record, cancel edits, post edits, and refresh. This component is nearly automatic in that most of the time all you have to do is drop it on the form, hook it to a Data Source, and forget about it.



[max@kleiner.com](mailto:max@kleiner.com)

Links of maXbox and 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>

Database Programming Starter 9 Tutorial

[http://www.softwareschule.ch/download/maxbox\\_starter9.pdf](http://www.softwareschule.ch/download/maxbox_starter9.pdf)

SHA1 Hash of maXbox 3.7.0.2 Win: 81A37822231D6B6286328AB77401620670F1B54A

SHA1 Win 3.9.9.5: EC71940E7F952262522A7DB91B2BF9762F14A8C6

### 1.3 Appendix: SQL Result in a String List

Notice that all the data-aware components are unrelated to the data-access technology, provided the data-access component inherits from `TDataSet`. Thus your investment in the user interface is preserved when you change the data-access technology.

Between line 62 and 126 is a code section which shows to store a parameterised query in a string list.

Let's say you just need to load these result once into your application, you can either keep a permanent connection to access the data or you can load it once, or whenever necessary, into memory and then free the connection.

What we show here is a very simple "trick", using a `TQuery` and `TStringList`, I show how to load each record from the `TQuery`'s result set into a string of the `TStringList`.

```

procedure fillRecords(aDBName, T,A,C, fs: string; sl: TStringlist);
var attrs: TStringlist;
    f: shortint;
    auxstr: shortstring;
begin
  attrs:= TStringlist.Create;
  splitAttributes(A, attrs);
  with TADOQuery.Create(self) do begin
    cachesize:= 500;
    connectionString:= 'Provider=MSDASQL;DSN='+aDBName+';Uid=sa;Pwd=admin';
    //commandText:= 'SELECT * FROM Table1';
    SQL;
    commandText:= 'SELECT '+A+' FROM '+T;
    if length(C) > 0 then
      commandText:= 'SELECT '+A+' FROM '+T+' WHERE '+C;
    Open;
    First;
    try
      while not(EOF) do begin
        auxstr:='';
        for f:= 0 to attrs.count-1 do
          auxstr:= auxstr + FS + fields[f].AsString;
        deleteStr(auxstr, length(fs));
        sl.add(auxstr);
      Next;
    end;
  end;
end;

```

[http://www.delphi3000.com/articles/article\\_2975.asp?SK=](http://www.delphi3000.com/articles/article_2975.asp?SK=)

In normal events we must have this components connected one each other:  
TADODataset → TADOQuery → TDataSource → DBNavigator.

This is something like:

```

1.0 ADOQuery1.dataSet:= ADODataset1;
2.0 DataSource1.DataSet:= ADOQuery1;
3.0 DBNavigator1.DataSource:= DataSource1;
3.1 DBGrid1.DataSource:= DataSource1;
4.0 ADOQuery.Active:= True;

```

~~maxbox~~