Limnor Studio – User's Guide

Part –V I

Database Programming

Contents

1 Introduction to Database Programming

1.1 Brief Introduction to Relational Databases

A relational database saves data in **Tables**. A **Table** consists of **Columns**. Another name for Columns is **Fields**. For example, an Order table may have columns such as Order ID, Order Date, Shipping Address, etc. An OrderItem table may have columns such as OrderItem ID, Order ID, Product ID, Quantity, etc.

Data in a Table are saved as **Records**.

For example, an Order Table may have following records:

	Order ID	Order Date	Shipping Address	…
Record 1:	1	07/21/1987	123 Main Street…	…
Record 2:	2	09/23/2001	321 Broadway…	---
…				

An OrderItem Table may have following records:

	OrderItem ID	Order ID	Product ID	Quantity	...
Record 1:	1	1	100	3	…
Record 2:	2	1	209	8	…
Record 3:	3	2	209	68	…

The OrderItem table has an Order ID field. It is used to indicate which Order an OrderItem record belongs to. In the above example, the first 2 records of the OrderItem table belong to the first Order record because their Order ID value is 1; the third record of the OrderItem table belongs to the second Order record because its Order ID value is 2.

The Order ID field is used in both Order and OrderItem tables and thus builds the relationships between the records in the two tables. That is why it is called a Relational Database.

The Order ID field in the Order table is called a **Primary Key**. Its values among all records must be unique so that it uniquely identifies each Order record.

The Order ID field in the OrderItem table is called a **Foreign Key** because it references values in another table. Its values do not need to be unique because many OrderItem records may belong to one same Order record. Or say that one Order record may have many corresponding OrderItem records. Such relationship is called **One-to-Many** relationship.

Suppose we have a Product table to save product data:

	Product ID	Product Name	Manufacturer	Model	Description	…
Record 1:	100	Laser Printer	HP	Wp200	…	…
Record 2:	209	Laptop	IBM	BT-91	…	…
…						

Look at the sample data in the OrderItem table, for the order identified by Order ID = 1, it has two products identified by Product ID = 100 and 209. That is, one Order record may be linked to many Product records. So, from Order to Product, it is a One-to-Many relationship.

For Product identified by Product ID = 209, the data in the OrderItem table show that it has two Order records, identified by Order ID = 1 and 2, linked to it. That is, one Product record may be linked to many Order records. So, from Product to Order, it is also a One-to-Many relationship.

Such relationship is called a **Many-to-Many** relationship. One Order record may use many Product records; and one Product record may be used by many Order records. OrderItem table makes Order and Product a many-to-many relation.

A One-to-Many relationship involves two tables.

A Many-to-Many relationship involves 3 tables.

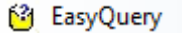
Key concepts introduced so far:

Table, **Column**, **Field**, **Record**, **Primary Key**, **Foreign Key**, **One-to-Many**, **Many-to-Many**

1.2 Database Accessing Components

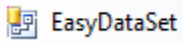
Database applications retrieve data from databases and save data into databases. Following components can be used for creating database applications.

1.2.1 Database Query



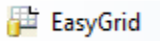
Database accessing is standardized by a 4GL (4th Generation Language) called SQL (Structured Query Language). This component visualizes the creating of SQL code for database accessing. Thus you do not need to learn SQL in order to do database programming.

1.2.2 EasyDataSet



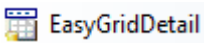
This component holds data retrieved from a database. It can be used to provide data to other UI components

1.2.3 EasyGrid



This control displays data from database in a grid.

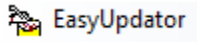
1.2.4 EasyGridDetail



This control displays data in a grid. In a one-to-many relationship, this control displays data from “many” side of the relationship. It must works with an EasyDataSet or EasyGrid which

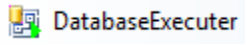
holds/displays data from “one” side of the relationship. For example, an EasyGrid may be used to display Order records and an EasyGridDetail may be used to display OrderItem records which belong to the Order record currently selected in the easyGrid.

1.2.5 EasyUpdator



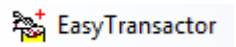
This component can be used to save data to database. The components mentioned above may also be used to save data to database. But those components must retrieve data from database first. This component does not retrieve data from database. It can be used to insert new records, delete existing records, or modify data in records. One execution may affect many records.

1.2.6 DatabaseExecuter



This component can be used to execute stored-procedures. It is a sub-class of the EasyUpdator component. It allows passing parameters to stored-procedures and get values from output parameters. If a stored-procedure returns tables of data then the tables are saved in a Results property.

1.2.7 EasyTransactor



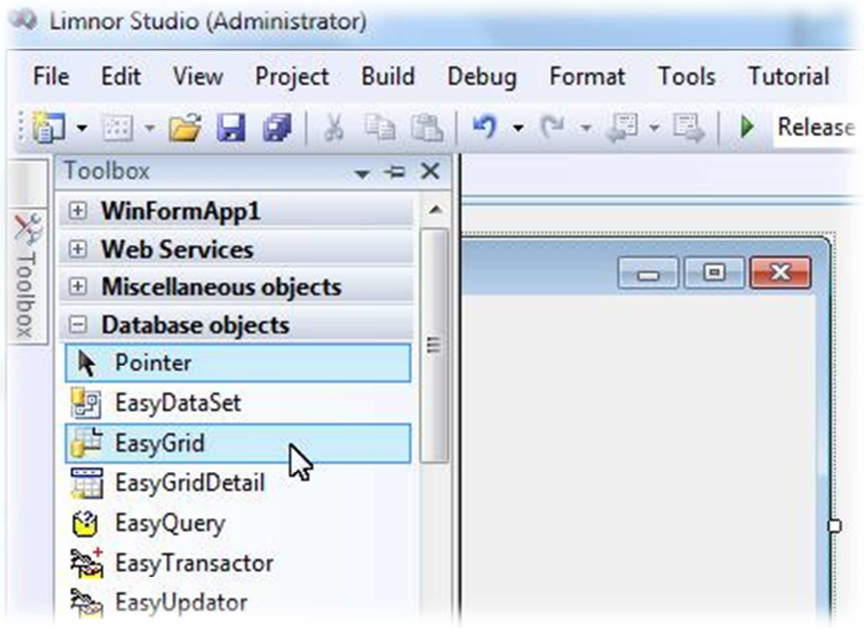
This component allows execute actions from many EasyUpdator/DatabaseExecuter components in one single transaction. If any EasyUpdator/DatabaseExecuter fails then no data will be saved in database. The data are saved to database only all EasyUpdator/DatabaseExecuter successfully execute their actions.

2 Make Database Connection

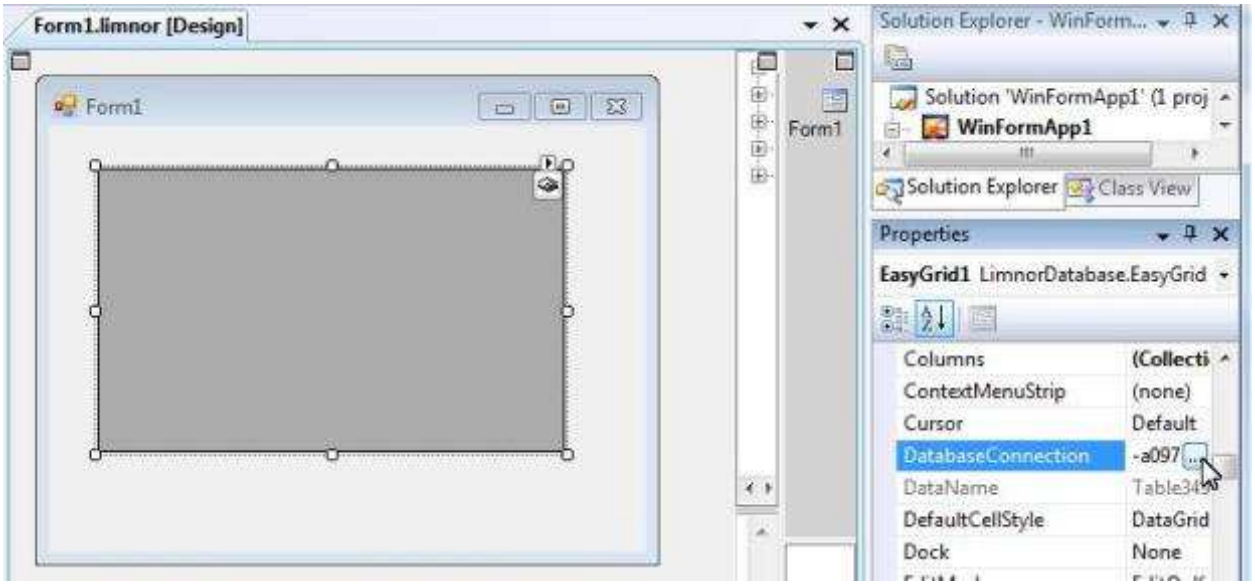
2.1 Set DatabaseConnection property

Database engines use ADO.Net drivers to let application developer access database. To use a database, the first thing is to specify the ADO.Net driver to be used.

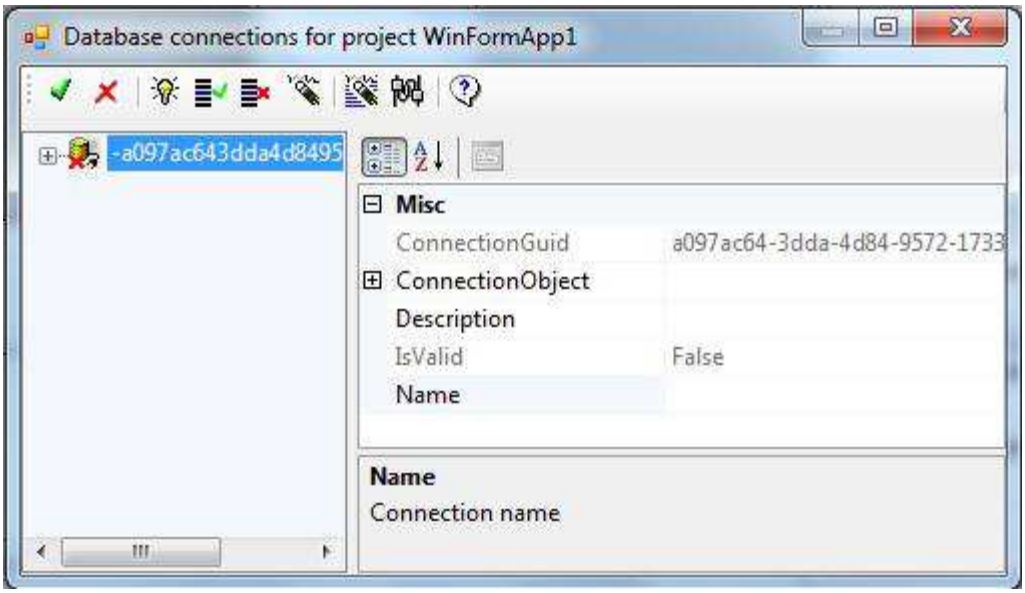
Let’s drop an EasyGrid to a form.



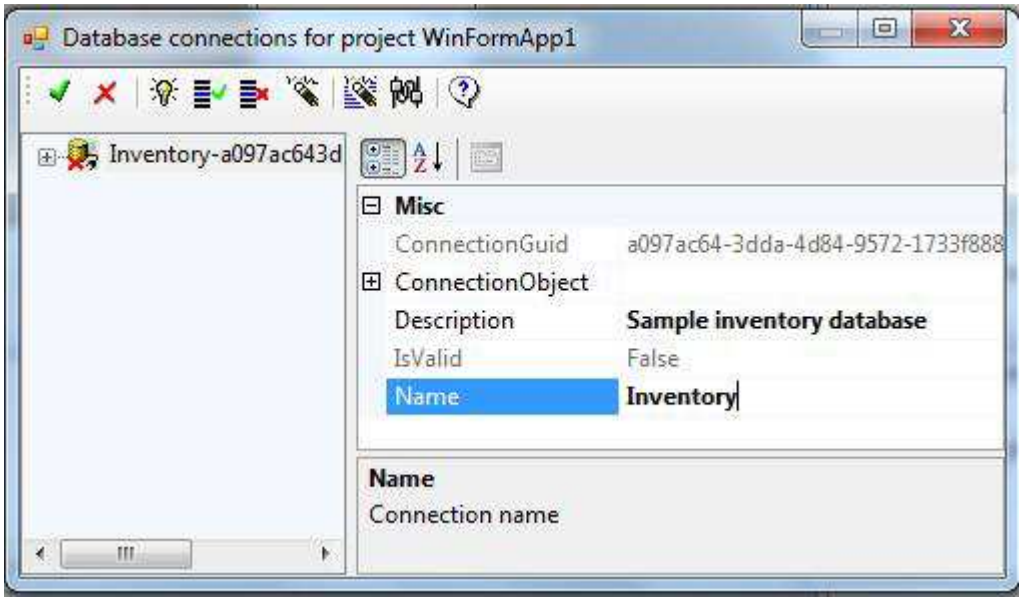
The DatabaseConnection property is used to make connection to a database.



The Database Connection Manager appears:

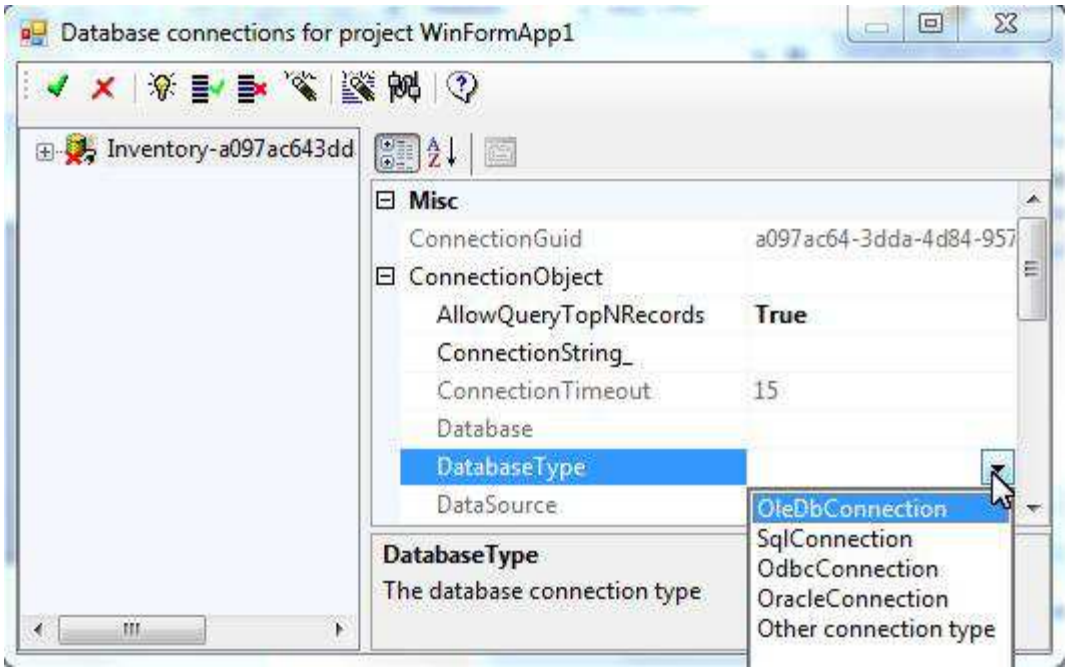


Each database connection is uniquely identified by a Guid. The Guid cannot be modified. We may give a connection a name and description to help identifying it:



ConnectionObject property of a connection specifies the database driver and connection parameters.

2.2 Select Database Driver



DatabaseType property specifies the database driver. Some commonly used drivers are listed for convenience.

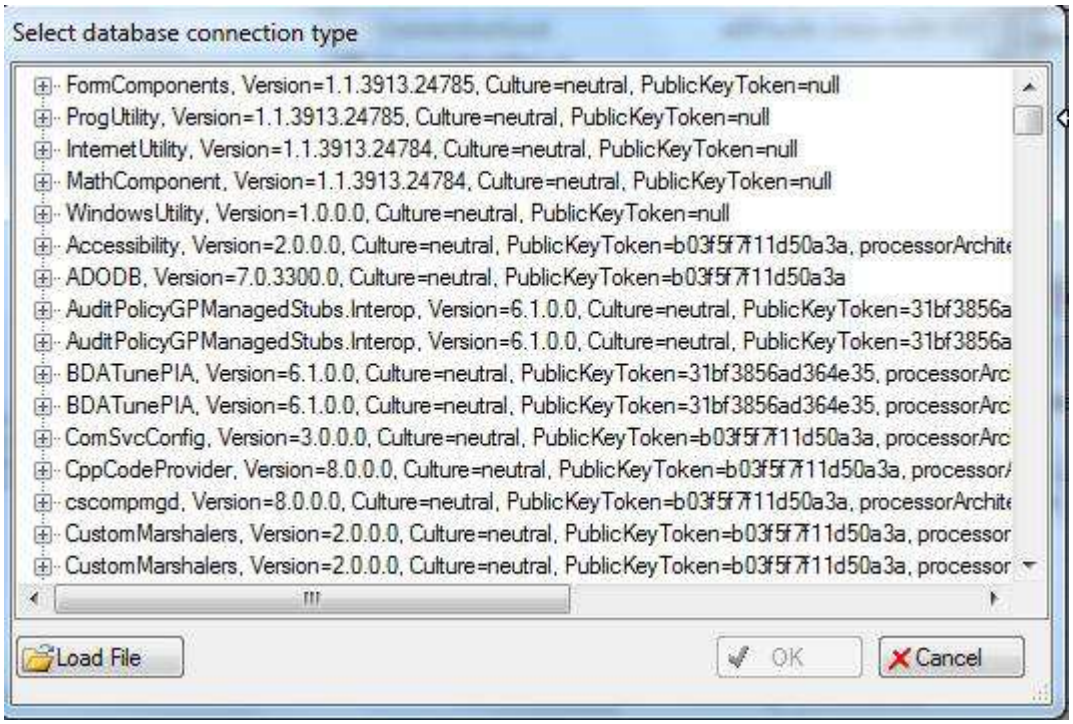
OleDbConnection – This driver can be used to connect to some desktop databases such as Microsoft Access

SqlConnection – This driver can be used to connect to Microsoft SQL Server databases

OdbcConnection – This driver can be used to connect to databases supporting ODBC data access standard.

OracleConnection – This driver is for Oracle databases

Other connection type – If a database cannot use the above drivers then the database needs to have its own ADO.Net driver. The driver should be in a DLL file. Selecting this option, a dialogue box appears. It lists software libraries in the computer for you to find the driver:



If the driver is not in the list then click “Load File” button to manually load the driver DLL file. You may provide the DLL file to the dialogue box.

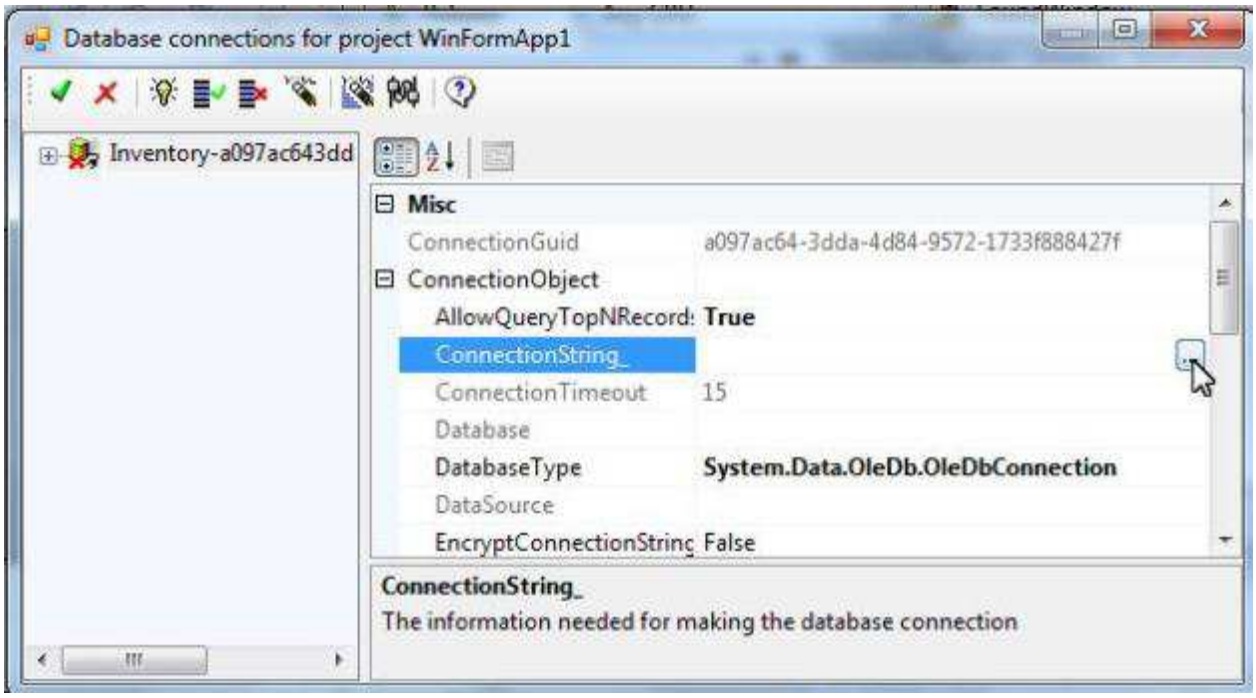
Some databases provide more than one kind of drivers. For example, MySQL database provides both Odbc driver and ADO .Net driver. Usually ADO .Net driver is more efficient. See http://www.limnor.com/support/ConnectToMySqlWithADO.pdf for a demonstration of loading ADO .Net driver for MySQL database. It shows the use of “Other connection type” option.

2.3 Set Database Connection String

Once a database driver is selected, we need to specify parameters for connecting to a specific database. All connection parameters are expressed by a string called Connection String. Every database driver has its own format of connection string.

The following web site dedicates to explain database connection string formats:
http://www.connectionstrings.com/

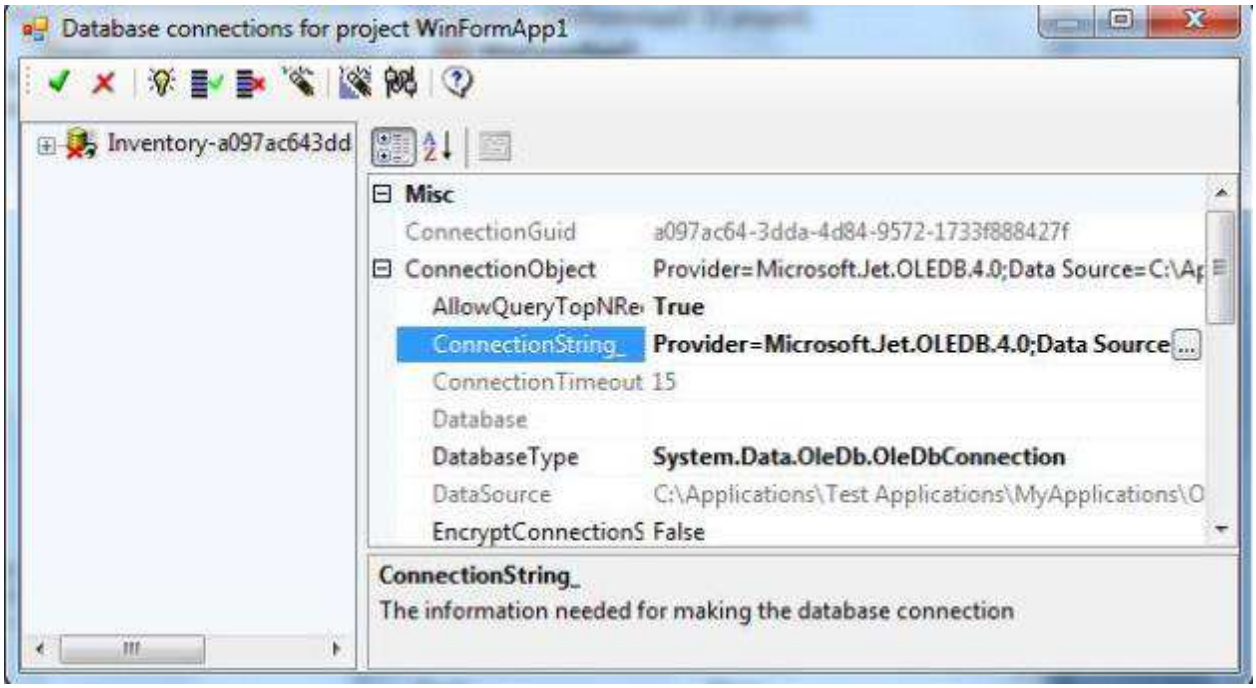
Suppose we selected OleDbConnection as the database type (driver). We may type in a connection string into ConnectionString_ property. If we are going to use a Microsoft Access database then we may click  to let the system create the connection string for us:



Select the database file:

Specify the other parameters if needed. For example, check “Read only” if we do not need to update the data. We may give database password and user credential if needed. Click Test button to see if the parameters are correct:

The database connection string is generated:



The database connection string it generated is

Provider=Microsoft.Jet.OLEDB.4.0;Data Source=C:\Applications\Test Applications\MyApplications\Orders.mdb;Mode=ReadWrite;

2.4 Other Connection Options

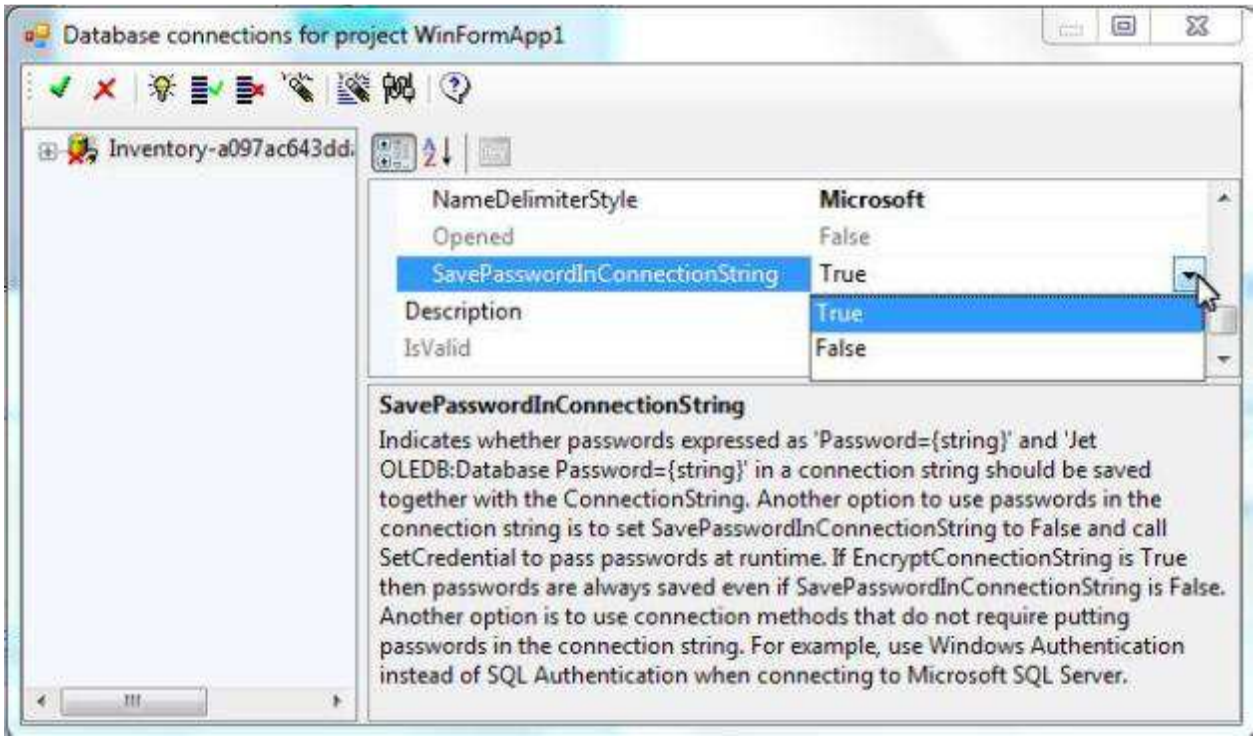
2.4.1 AllowQueryTopNRecords

Some databases support retrieving first few records from databases. The system may use such feature to do data sampling when doing visualized query building. If the database we are connecting to does not support this feature then we should set this property to False so that the Query Builder will not fail:

2.4.2 EncryptConnectionString and SavePasswordInConnectionString

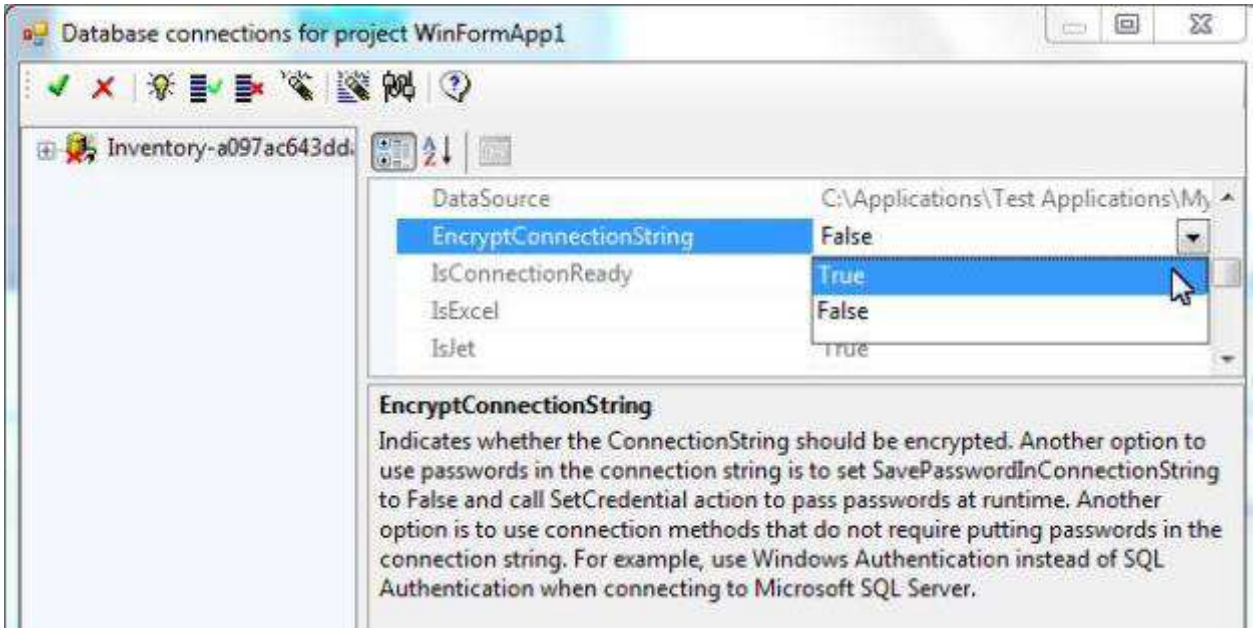
If the connection string includes passwords then a security concern arises.

If we set SavePasswordInConnectionString to False then we must use a SetCredential action to specify the passwords at runtime before accessing the database. We need to use a way to get passwords for the SetCredential action. For example, use a dialogue box to let the user enter passwords; or save passwords in an Application Configuration file and encrypted, see http://www.limnor.com, click Samples, find "**Application Configuration Samples**".



If we set SavePasswordInConnectionString to True then passwords are saved with the connection string. When the program runs we do not need to use a SetCredential action to provide passwords.

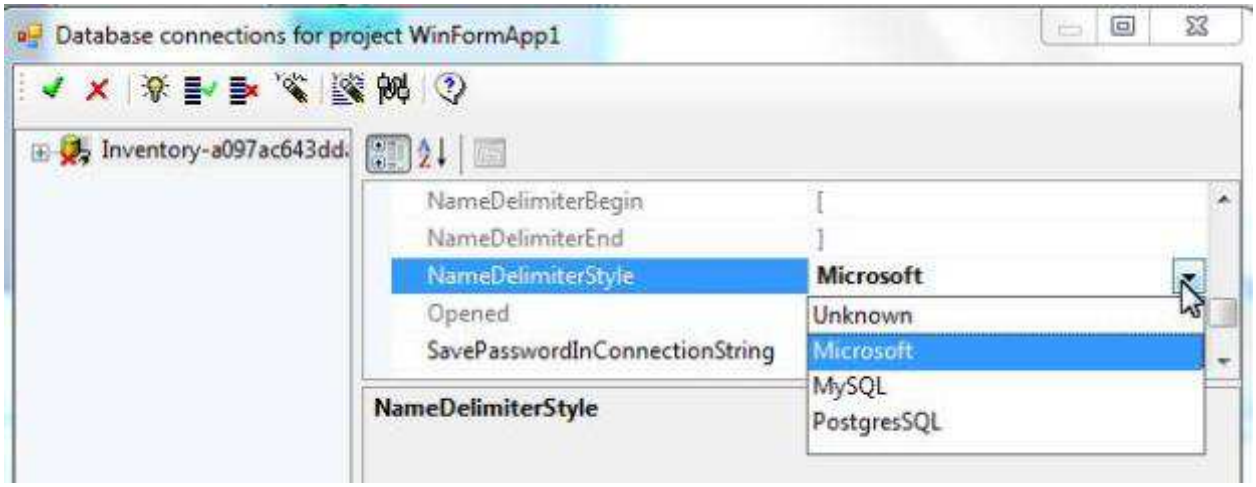
To prevent unauthorized persons read the passwords from the connection string, we may set EncryptConnectionString to True:



Note that since the passwords are included in the connection string, even they are encrypted, the connection string becomes an attach target for hackers trying to decrypt the passwords. It is suggested that this option is not used if other better means of keeping passwords secret are viable.

2.4.3 NameDelimiterStyle

SQL is formed by a set of Keywords, as all other programming languages do. A database object name (table name, field name, index name, stored-procedure name, view name, etc.) may happen to be the same as a keyword. For example, Order can be a table name but it is also a keyword in SQL. This naming collision may cause some database engine fail. Delimiters can be used to distinguish a database object name from a keyword. For example, [Order] refers to a table named Order. Different database engines may use different delimiters. 3 types of delimiters are supported:



Microsoft – The delimiters are [and]. Example: [Order]

MySQL – ` is the delimiter. Example, `Order `

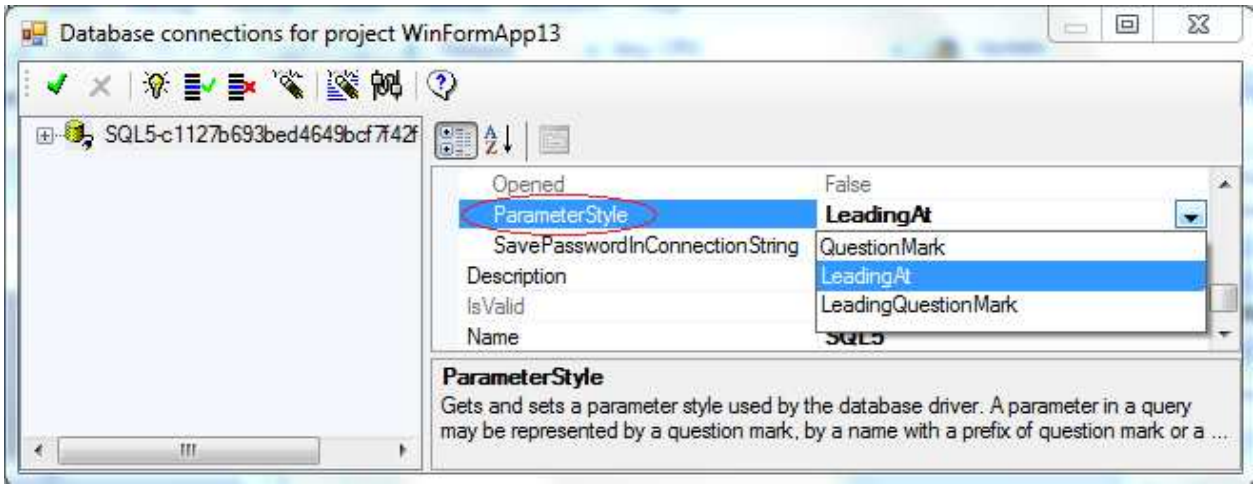
PostgresSQL – " is the delimiter. For example, "Order ".

Unknown – if your database does not use any of above delimiters then choose Unknown. No delimiters will be used.

2.4.4 ParameterStyle

Parameters can be used in queries. There is not an industrial standard on how a parameter should be represented. Even for a same database engine different database drivers may choose to use different ways to present parameters in queries.

Property ParameterStyle in the connection object indicates the way the database driver uses to present parameters in queries.



The property can be one of the following values:

- *QuestionMark* – A parameter is a question mark. For example, SELECT field1 FROM table1 WHERE field2=?
- *LeadingAt* – A parameter is formed by a @ symbol and a parameter name. For example, SELECT field1 FROM table1 WHERE field2=@p1
- LeadingQuestionMark – A parameter is formed by a question mark and a parameter name. For example, SELECT field1 FROM table1 WHERE field2=?p1

Limnor Studio uses following rules to identify the value for this property:

- For ODBC drivers, use *QuestionMark*.
- For OleDb drivers, use *QuestionMark*.
- For MS SQL Server, use *LeadingAt*.
- For MySQL .Net driver (MySql.Data.MySqlClient.MySqlConnection) , use *LeadingQuestionMark*.

You may change this property if you know the way your database driver presents parameters.

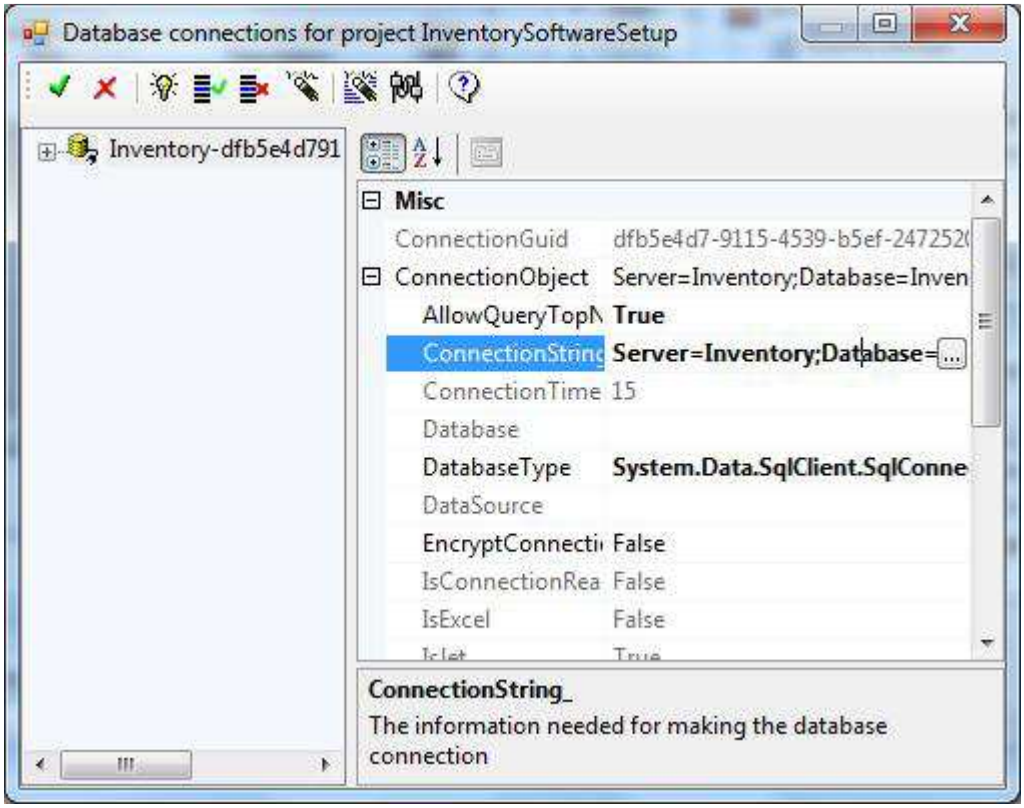
The Query Builder in the Limnor Studio always uses *LeadingAt* to present parameters. Thus for all databases you may just use one way to form parameterized queries. The Query Builder uses ParameterStyle property to translate the queries into the correct form the driver understands.

3 Distribute Database Applications

A distribution issue specific for applications using databases is that the database locations are different in the installation computers than the database locations in the development computers. If this issue is not handled properly then the applications will not be able to connect the correct databases.

3.1 Not Distributing Databases

Client/Server database will not be distributed with a database application. After installation, when the application runs the first time, the Database Connection Manager appears for adjusting the database connections for the application. For it to work in Vista and Windows 7, the application must run as Administrator so that it will have the permission to modify database connections.



-- Create a new database connection.

-- Select an existing database connection already created by other applications.

 --Remove the selected database connection from the application. The database connection is not deleted and it still can be used by other applications.

 -- Delete the selected database connection.

 -- Delete all the database connections which do not have connection information specified.

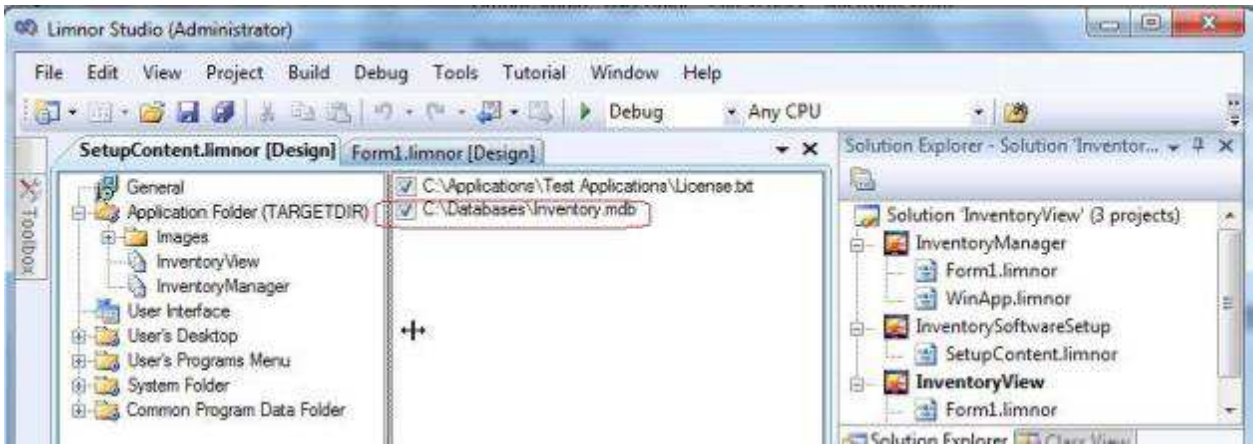
 -- Test the selected database connection.

3.2 Distribute databases with Applications

In some situations it may be desired to distribute desktop databases with the applications using them.

3.2.1 Distribute Read-only Databases

If an application only reads from a database but not write to the database then distributing the database can be easier. For example, if a Microsoft Access database is used then simply install the database file in the target installation folder is good enough. Limnor Studio support setup project. Below is an example of using a setup project to distribute a MS Access database file in the target installation folder:

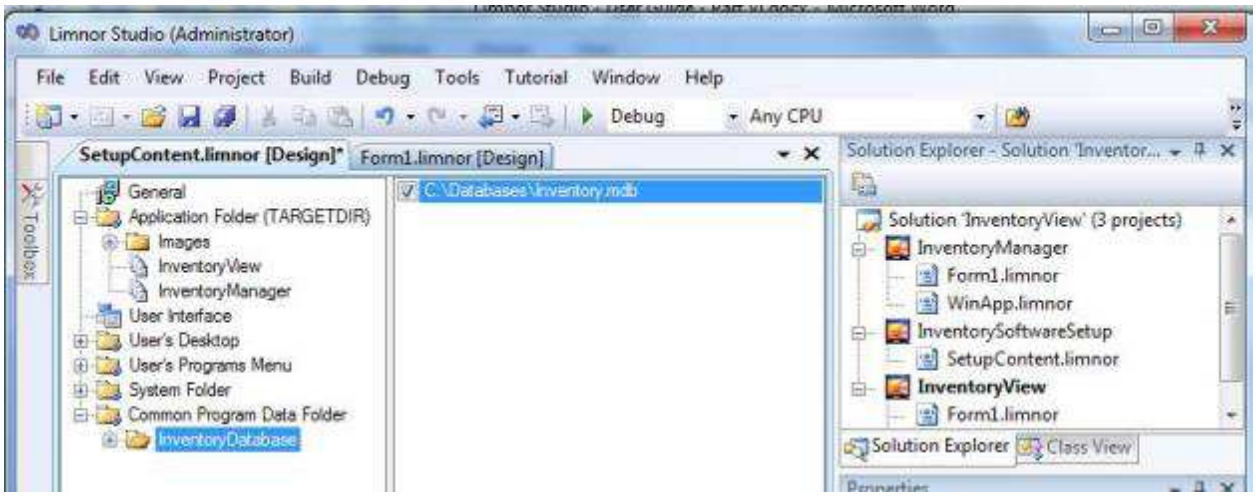


For more information about using Setup project, see Limnor Studio User's Guide.

If a CD-ROM application is using MS Access databases then copy the database files to the same folder of the application EXE file. Make sure the application uses read-only mode to connect to the databases.

3.2.2 Distribute Writeable Databases

Writeable database files should not be installed to the target application installation folder because for Vista and Windows 7 files in that folder may need administrator permissions to write to. One good folder to install the database files is the Common Program Data Folder:



The Limnor Studio Setup project adds specific support for Microsoft Access database. When compiling the setup project, the connection settings will be adjusted to point to the right folders. Thus there is no need to use the Database Connection Manager to adjust the database connections by application users.

Be aware of that uninstalling of the application will also remove the databases installed with the application.

Please contact support@limnor.com is you need Limnor Studio Setup project to do the similar automatic adjustments for Microsoft Access database.

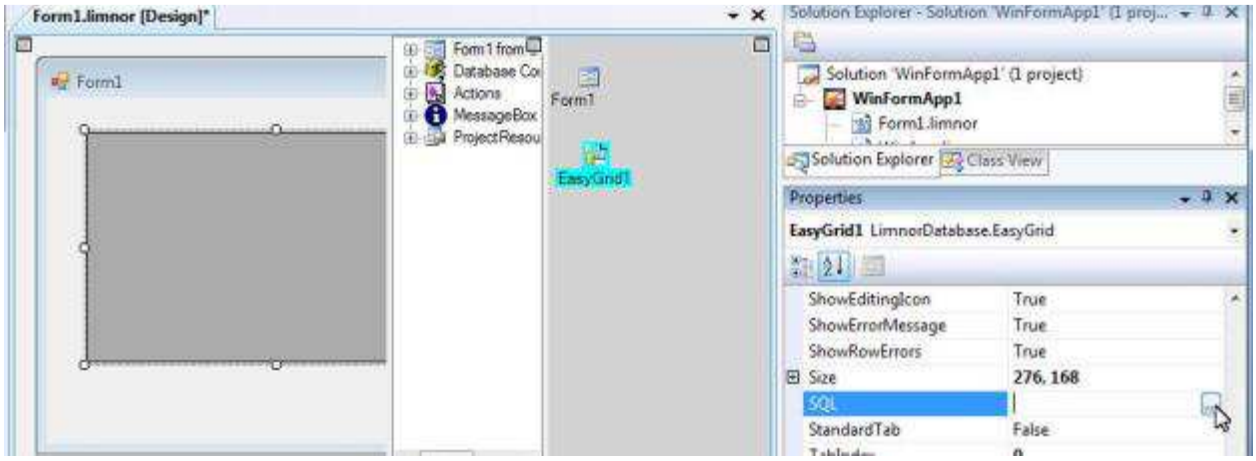
For more information on using the Setup project, go to http://www.limnor.com and click User's Guide.

4 Query Builder

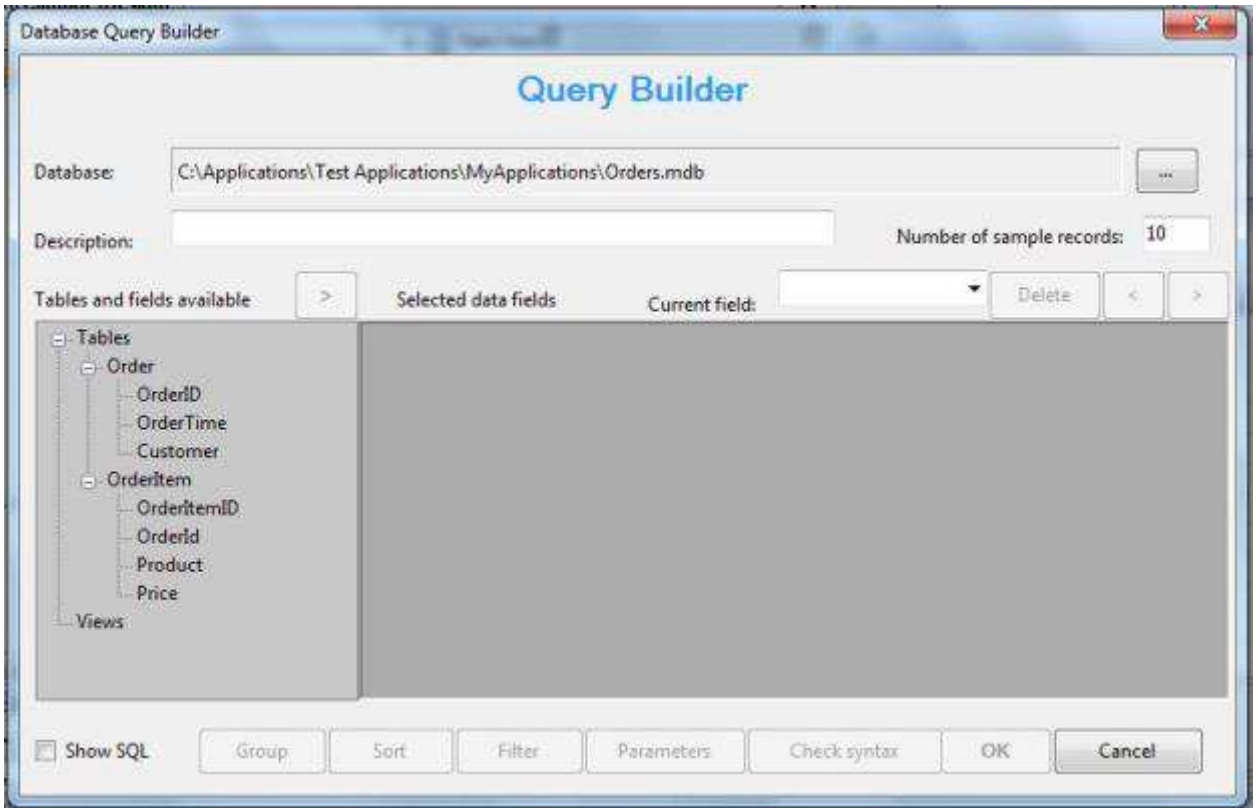
4.1 Use Query Builder

The Query Builder is a visual tool to generate SQL code. Prior knowledge of SQL is not needed.

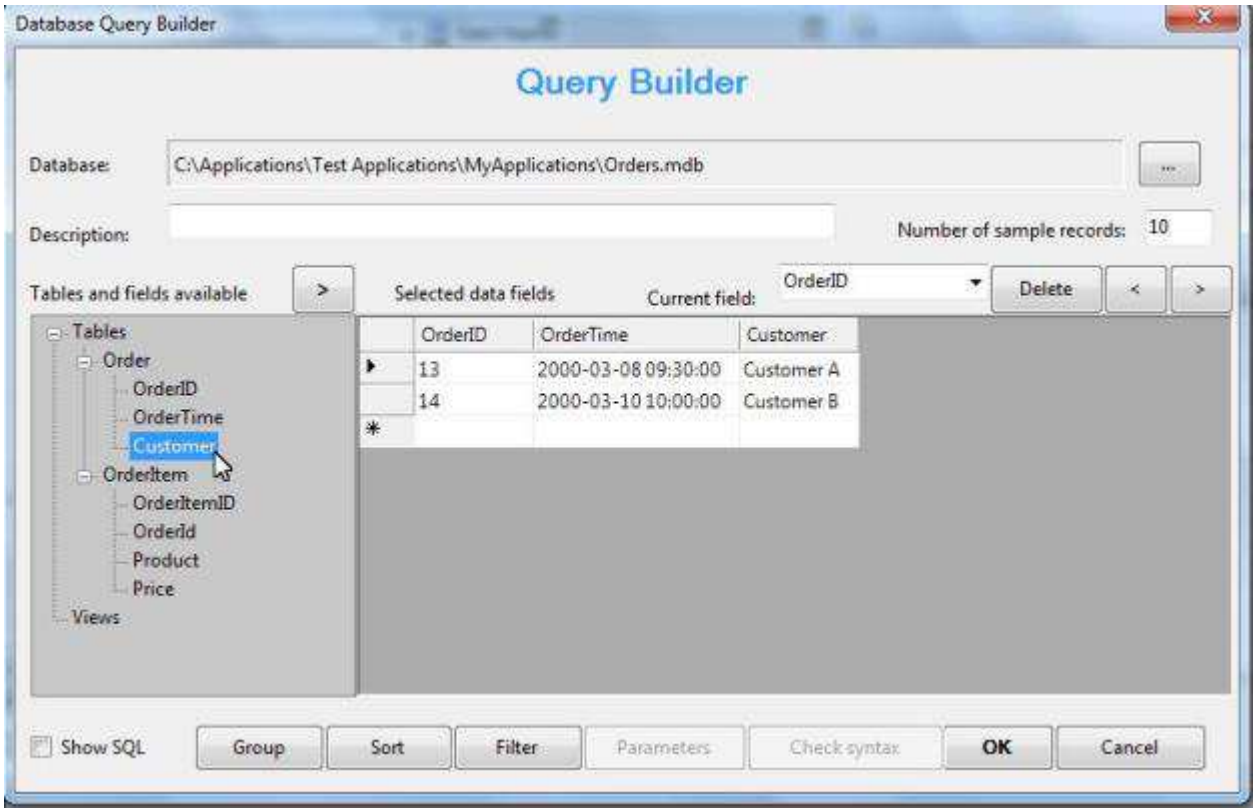
Set the SQL property will launch the Query Builder:



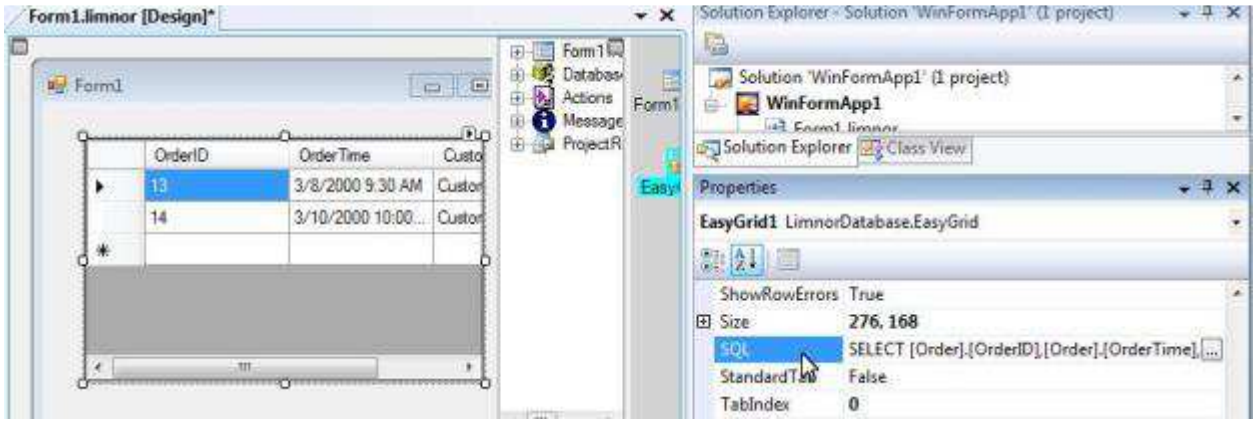
The Query Builder use the database connection specified in DatabaseConnection property to connect to the database and allow us to select the data we want to retrieve from the database. It displays the database objects for us to select.



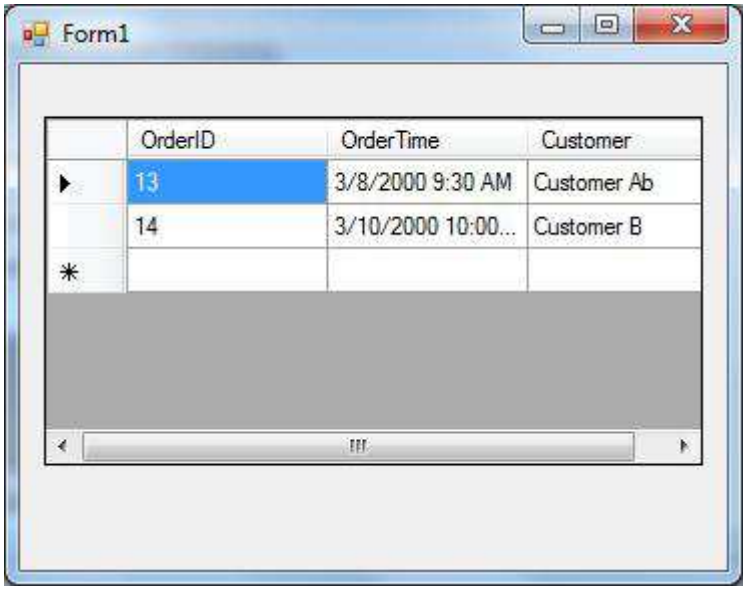
Suppose we want to display the data from the Order table. Double-click on the fields of the Order table to include the fields we want. Suppose we want all of them:



Click OK. The query is built:

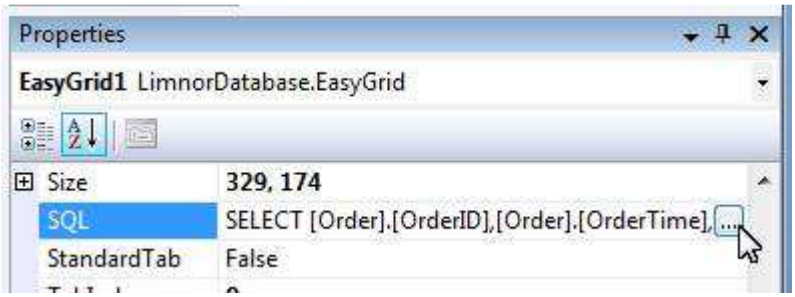


Run the application to test it:

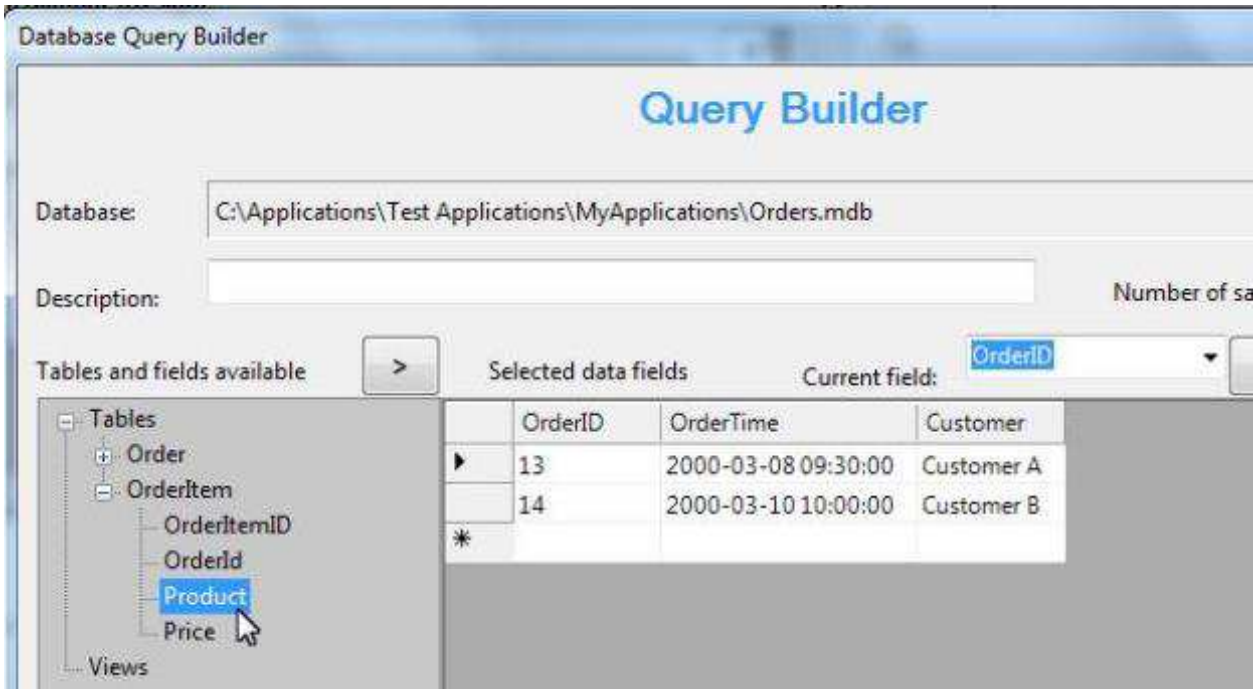


4.2 Retrieve Relate Records

Suppose we also want to include OrderItem data in the query. Set the SQL property again:



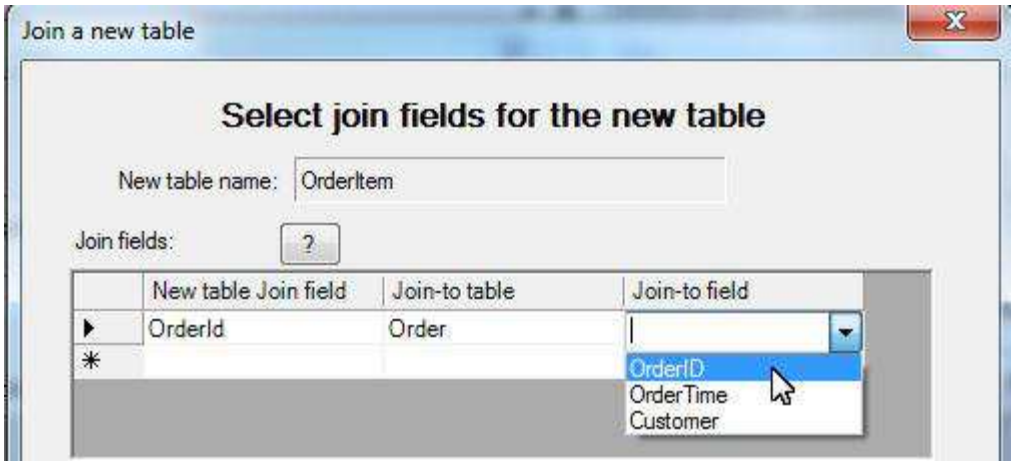
Double-click Product field of the OrderItem table to include this field into the query:



OrderItem is a new table to the query. It asks how this new table is related to the existing tables in the query. We need to select the Foreign Key of the OrderItem table, which is OrderId:

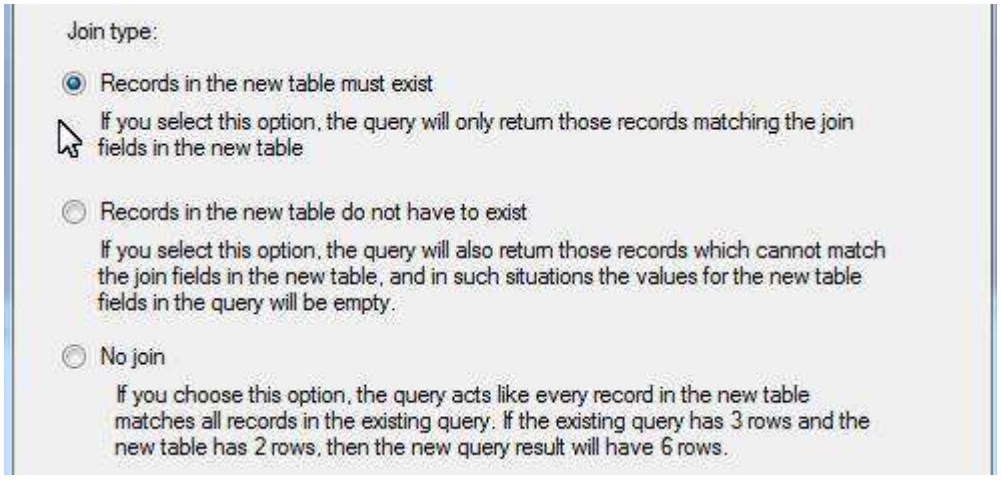
Select the table from the query for the new table to link to. The query only has one table at this time, Order:

Select the Primary Key field from the Order table:



This relationship indicates that the Order table and the OrderItem table are linked by OrderId field in the OrderItem table and OrderID field in the Order table.

You may specify one of 3 join types:

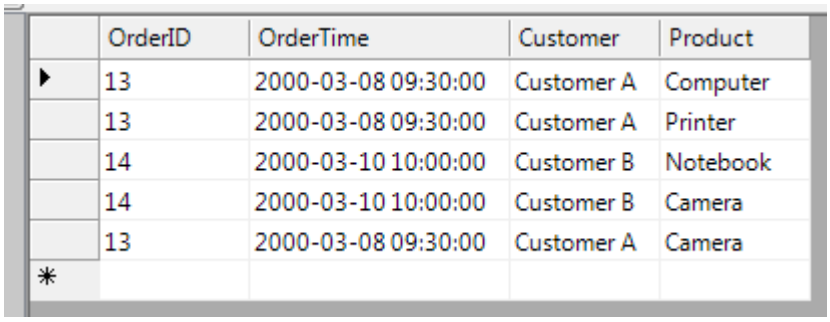


If the first option is selected then a record in the Order table must have matching OrderItem records for it to be included in the query. That is, if an Order does not have one or more Order Items then the Order record will not be retrieved. The number of records in this query will be the same as the number of records in OrderItem table.

If the second option is selected then all Order will be included even an Order does not have Order Items. If an Order does not have an Order Item then the Product field will be empty. The number of records in this query will be the same as the number of records in OrderItem table plus the number of those Order records which do not have corresponding OrderItem records.

The third option does not make joins. It will ignore the join fields specified above. For Order and OrderItem relationship this option does not make sense. The number of records in this query will be the number of the Order record multiplies the number of the OrderItem records.

Select the first option, we have the new query:

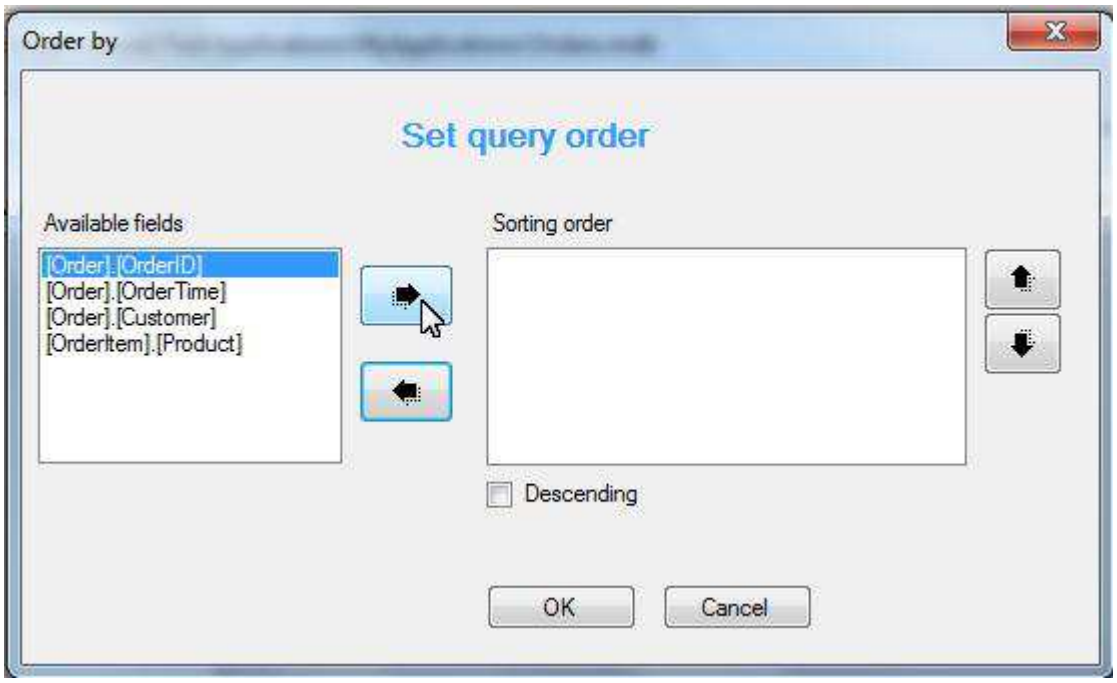


	OrderID	OrderTime	Customer	Product
▶	13	2000-03-08 09:30:00	Customer A	Computer
	13	2000-03-08 09:30:00	Customer A	Printer
	14	2000-03-10 10:00:00	Customer B	Notebook
	14	2000-03-10 10:00:00	Customer B	Camera
	13	2000-03-08 09:30:00	Customer A	Camera
*				

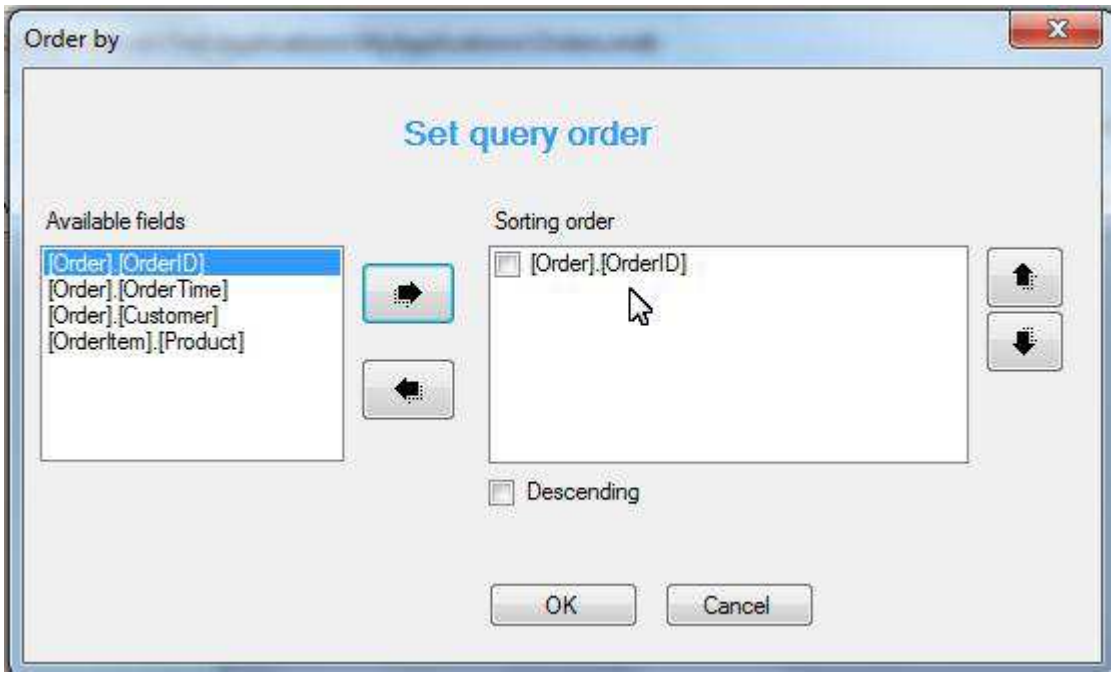
4.3 Sorting

If we want same orders go together then we may use sorting. Click Sort button:

Include OrderID in the sorting:



If we need a field to be sorted descending then check the check box besides the field name. If the check box is unchecked then the sorting order for it is ascending.



Records are sorted by OrderID:

	OrderID	OrderTime	Customer	Product
▶	13	2000-03-08 09:30:00	Customer A	Camera
	13	2000-03-08 09:30:00	Customer A	Printer
	13	2000-03-08 09:30:00	Customer A	Computer
	14	2000-03-10 10:00:00	Customer B	Camera
	14	2000-03-10 10:00:00	Customer B	Notebook
*				

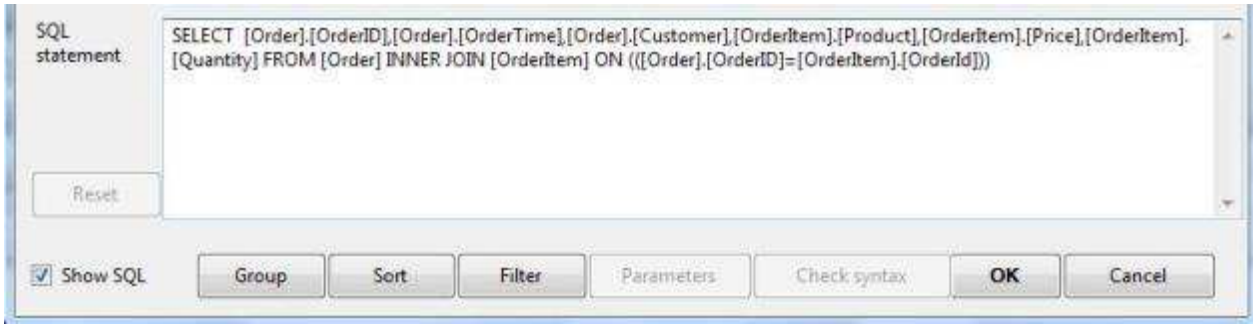
4.4 Calculated Fields

Expressions can be used to form new fields.

Let's add Price and Quantity fields to the query:

Check the Show SQL check box:

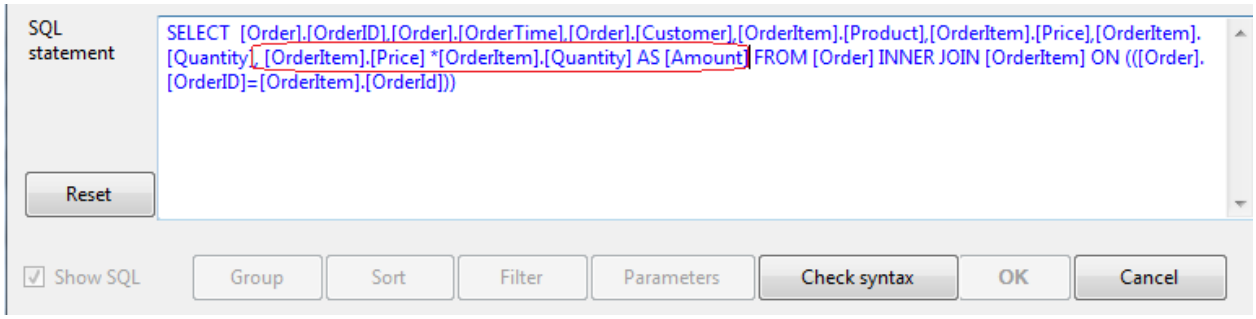
The SQL statement is displayed:



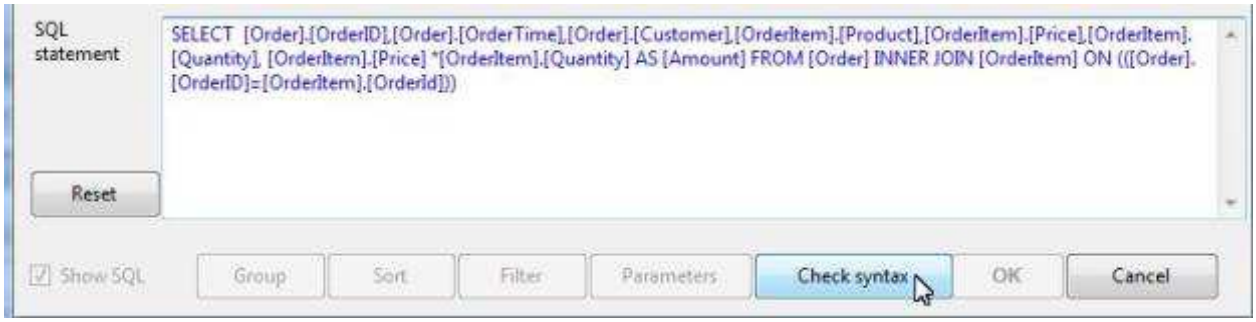
Add following expression into the field list of the statement:

[OrderItem].[Price] *[OrderItem].[Quantity] AS [Amount]

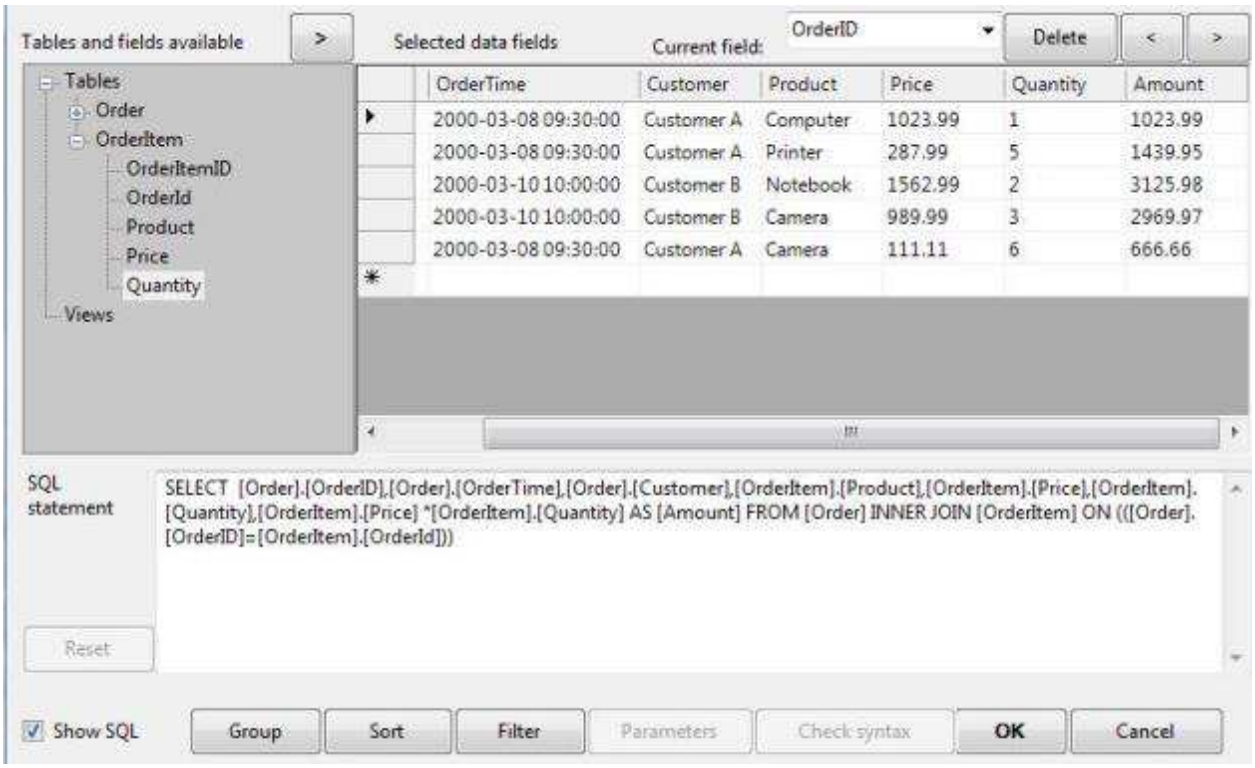
Note that a comma should be used to separate the fields.



The SQL statement is blue, indicating that it is manually edited. Click “Check syntax” button to check whether the editing is correct:



If the editing is correct then the SQL statement become black and the sample data show the results. Note the Amount field which is a calculated field:



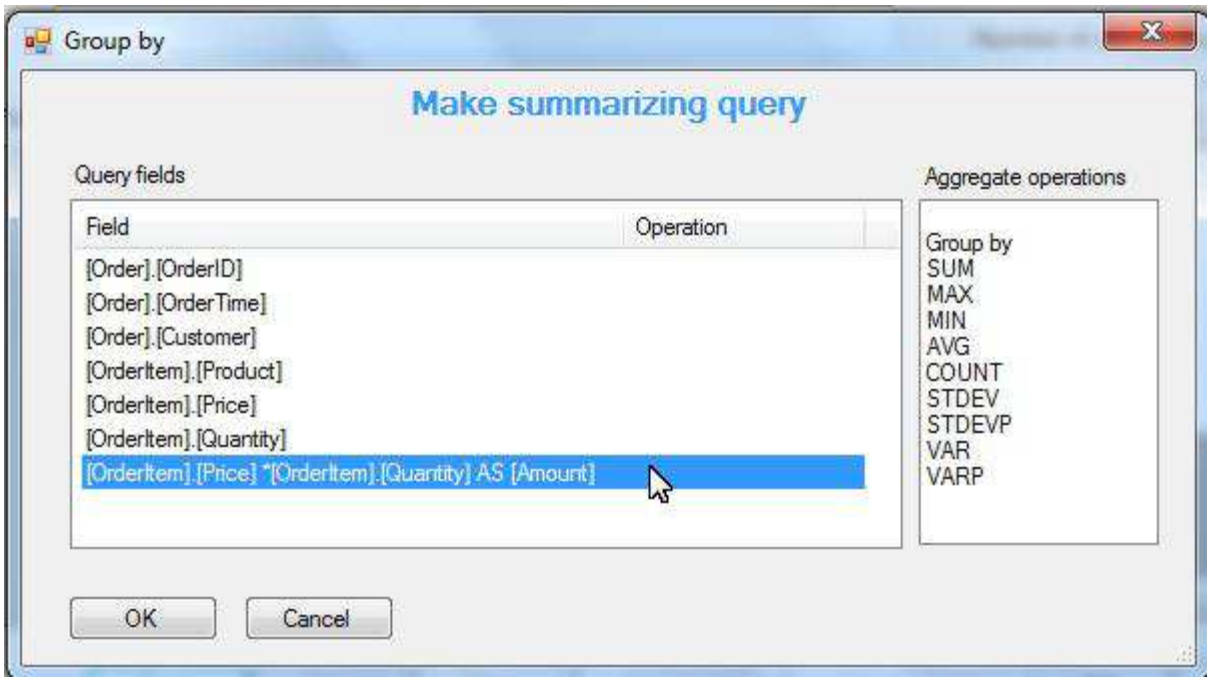
4.5 Grouping

Grouping is used to do statistics on data. Suppose we want to we want to know the amount for each Order. We need to do a summary for the calculated field Amount for each Order.

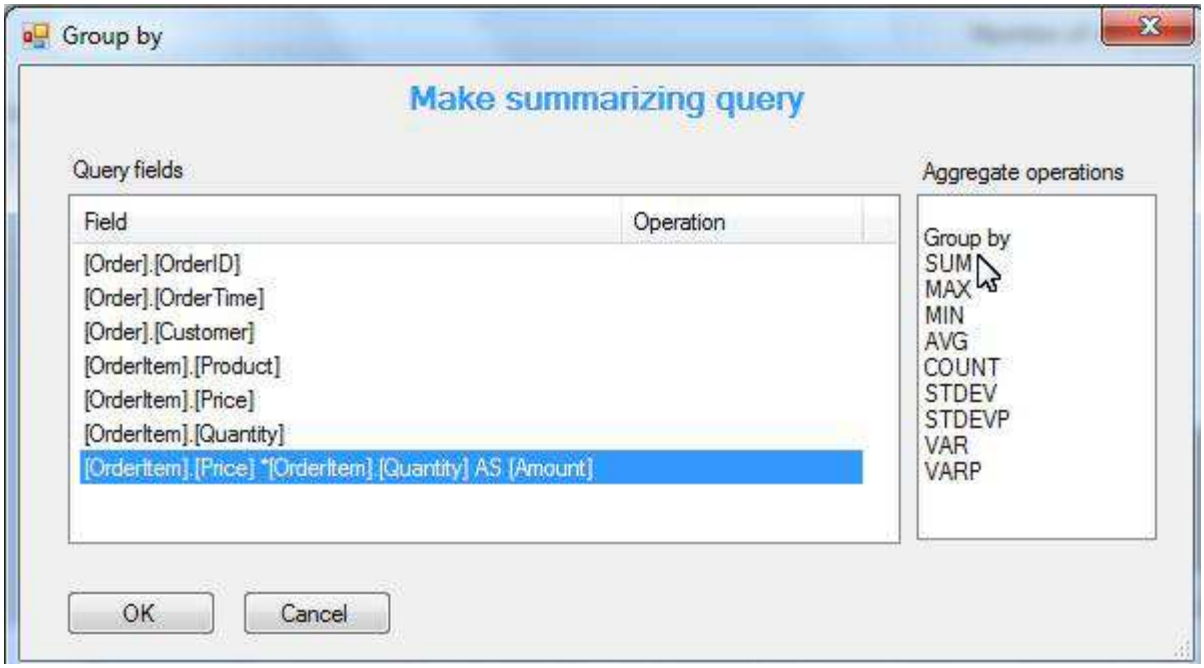
Click the Group button.

The way to do grouping is by selecting a field and assign aggregation operation to it.

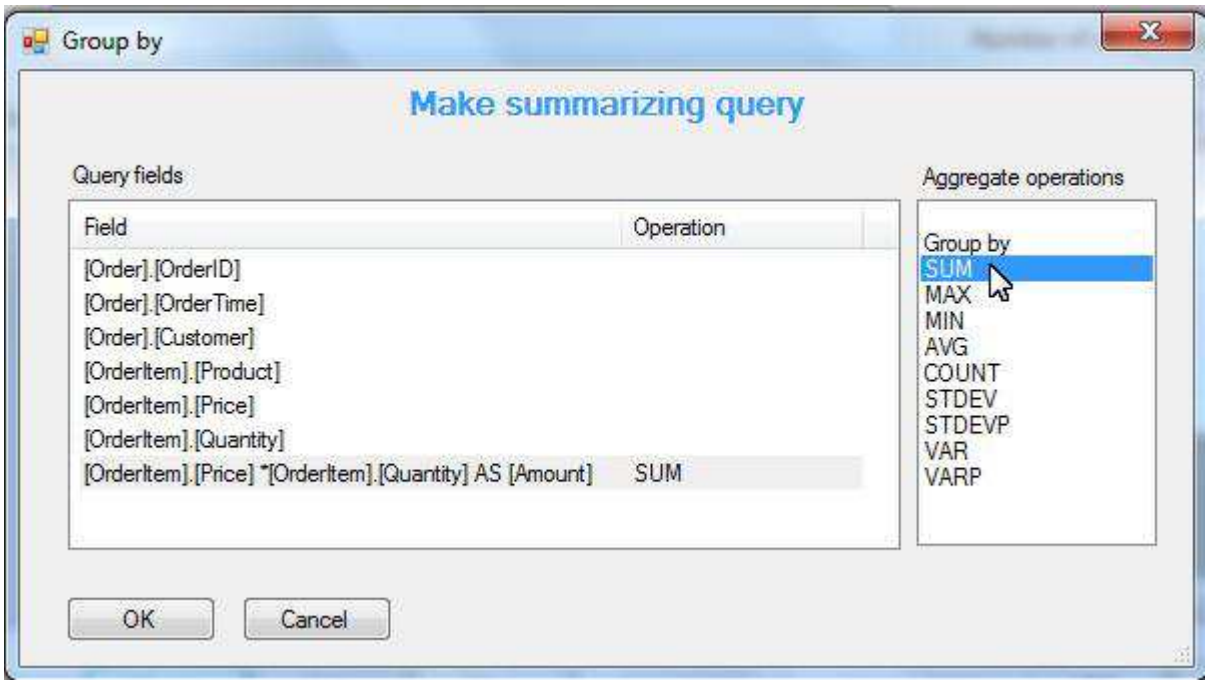
Because we want to sum the calculated Amount, we select this field:



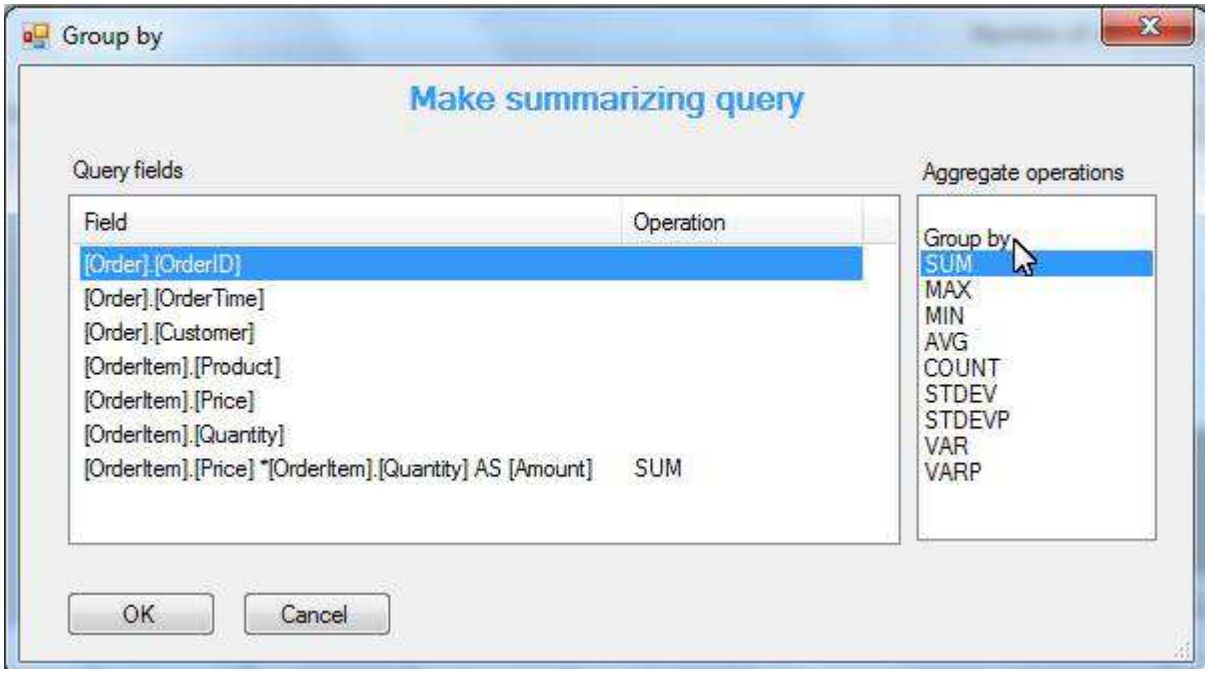
Click SUM:



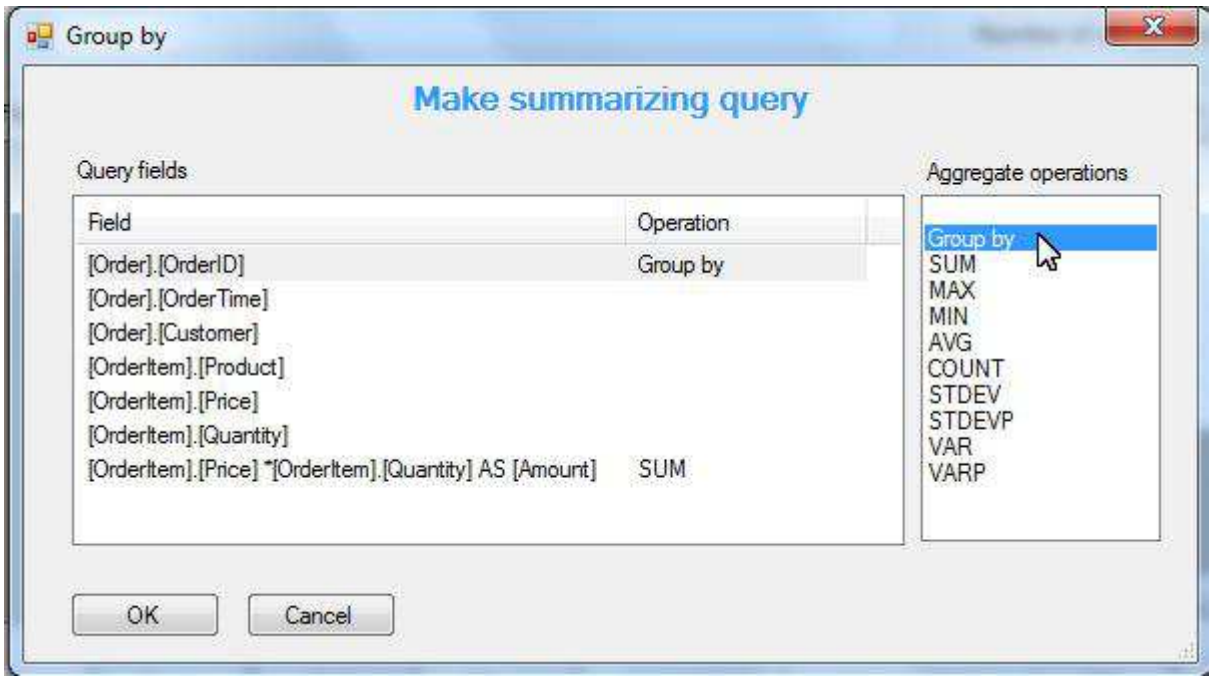
Summary will be operated on Amount:



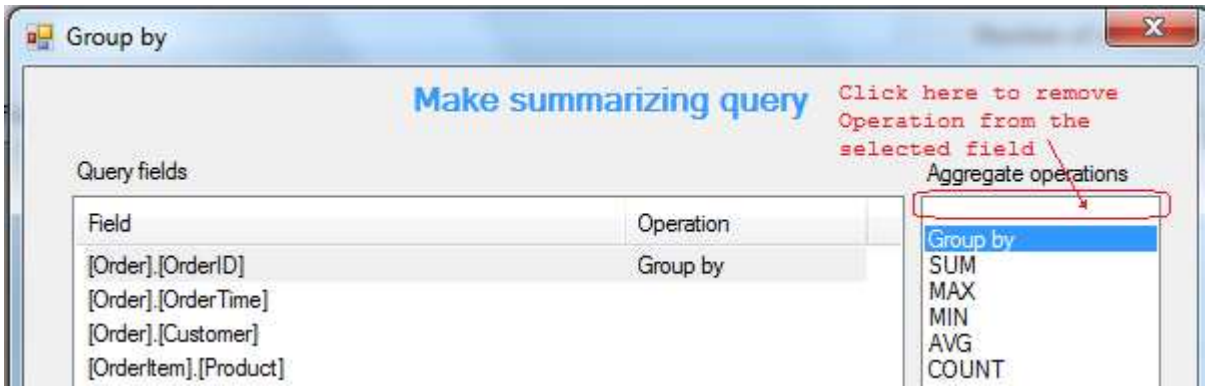
Select OrderID field and click Group by:



Now the summary will be done within the same OrderID value:



If you made a mistake and want to remove the Operation from a field then select the field and click the empty place on top of the list of "Aggregate operations".



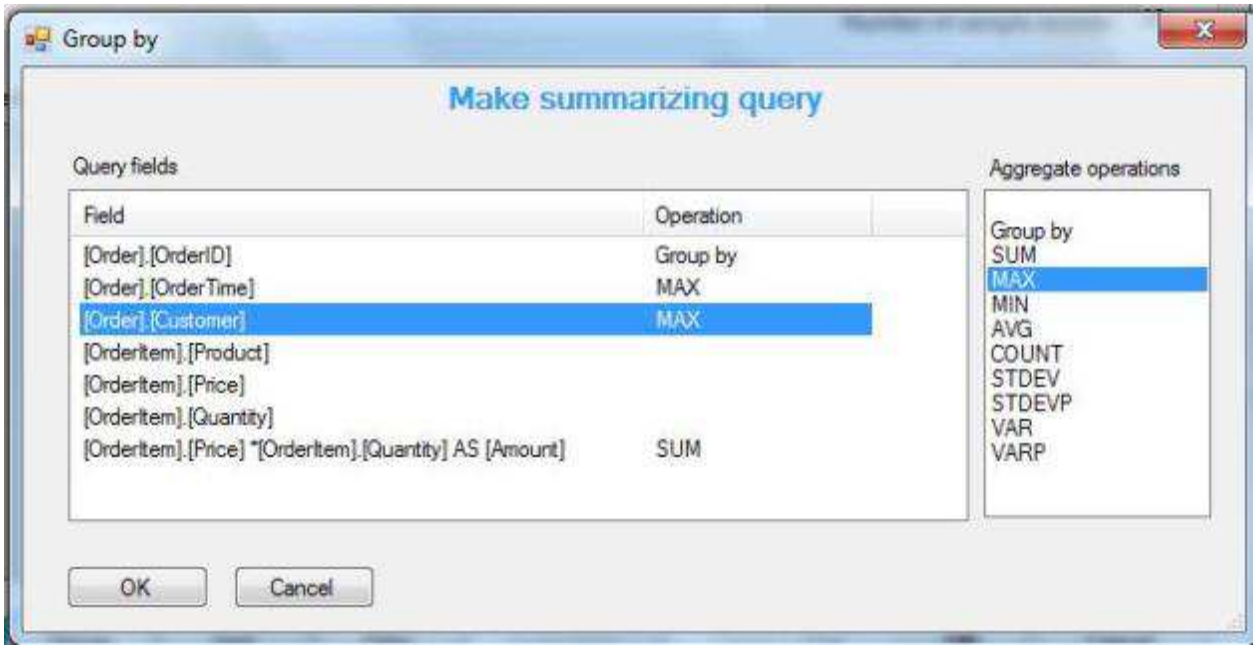
A rule in Group By query is that **every field must use an Operation**.

We do not give operations for [Product], [Price] and [Quantity]. Thus these 3 fields will be removed from the query.

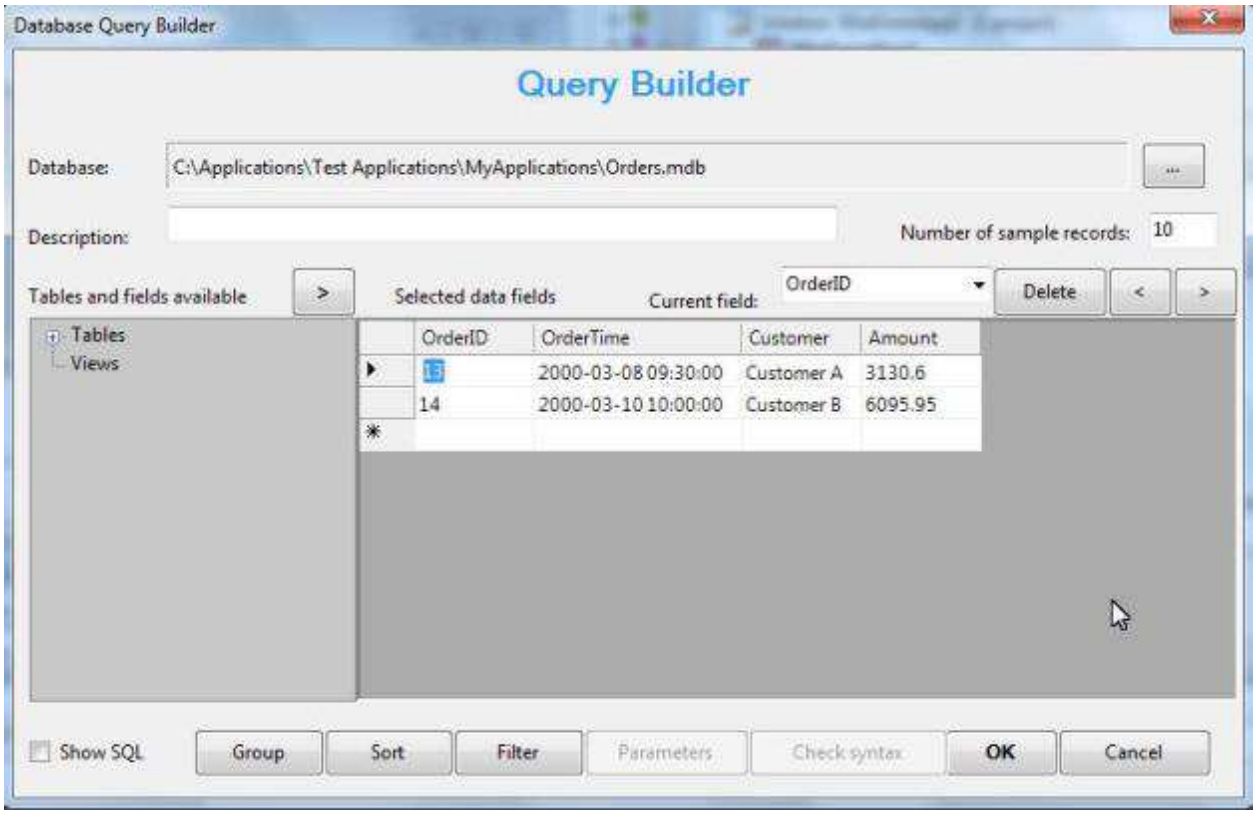
We still want to keep the OrderTime and Customer fields. We need to use operations on these fields in order to keep them in the query. We may use the MAX operation. The MAX operation returns the maximum value in the group. OrderTime will have the same value in the group since we are grouping on OrderID. Thus Max operation on OrderTime will just give the OrderTime value for the Order. The same is true for Customer field. This technique is simple but may not be quite efficient for large amount of data in a group.

If a database supports sub-query then the grouping query can be a sub-query joining to the Order table to get OrderTime and Customer. Thus avoid unnecessary MAX operation.

This is our grouping setting:



Click OK. We have grouping results:

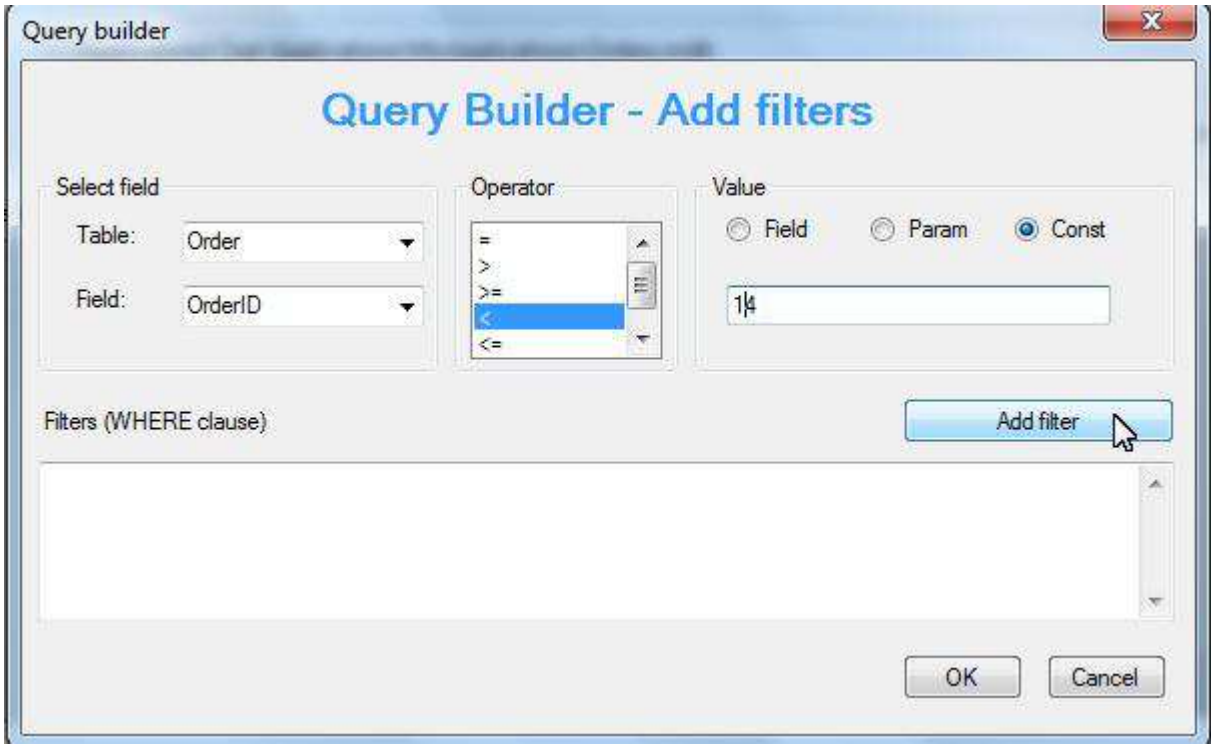


4.6 Filtering

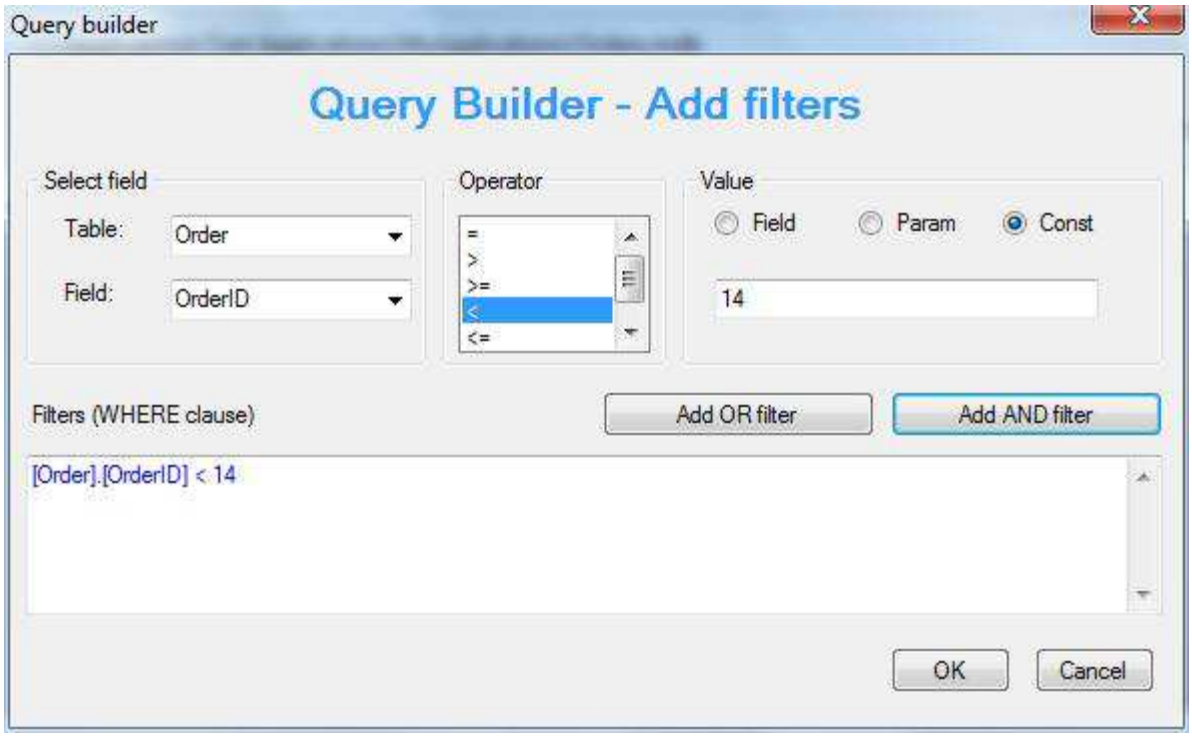
Filtering is used to exclude unwanted records from the query results.

Click Filter button.

Select a field; choose filter operator; give filter value. In this sample, we want to only include records with OrderID less than 14. Click Add Filter button:

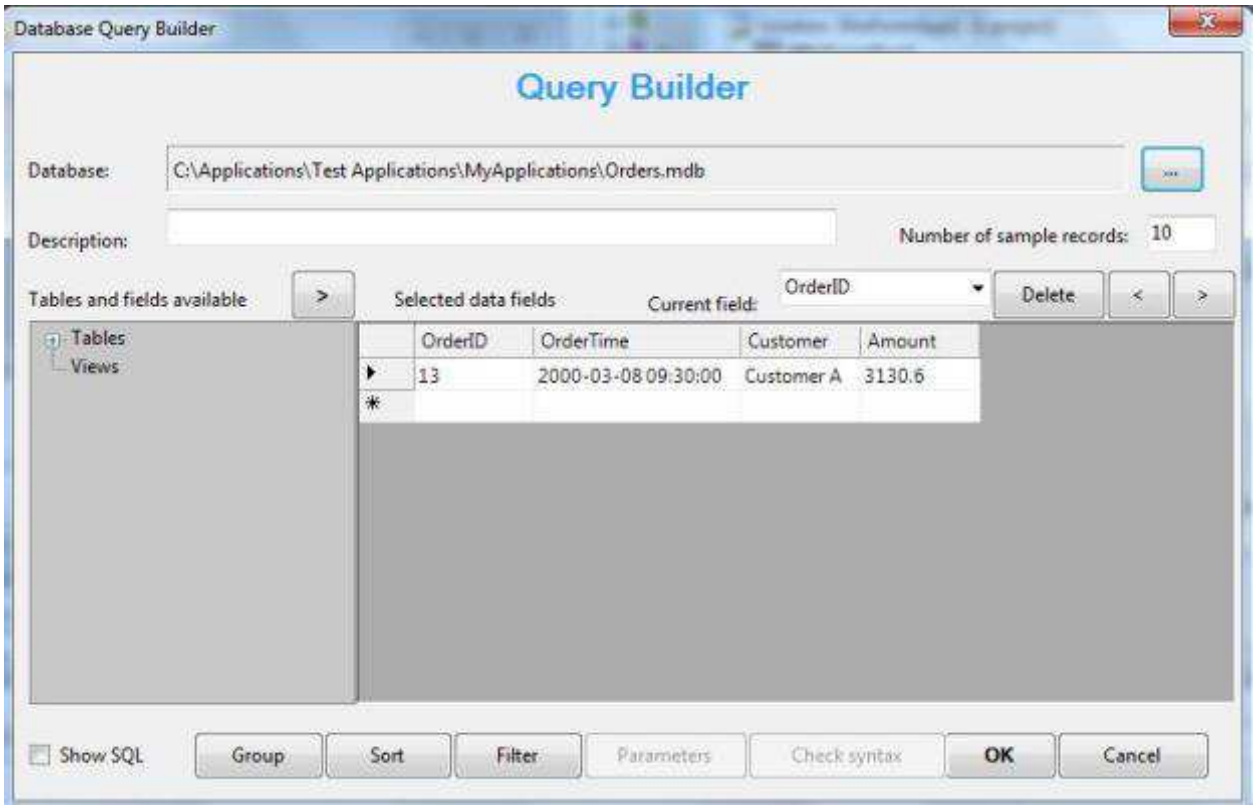


The filter is added:



We may enter more filters and use button “Add OR filter” and “Add AND filter” to add new filters to the existing ones in OR or AND logic.

With the above filter, one record is removed from the query results:



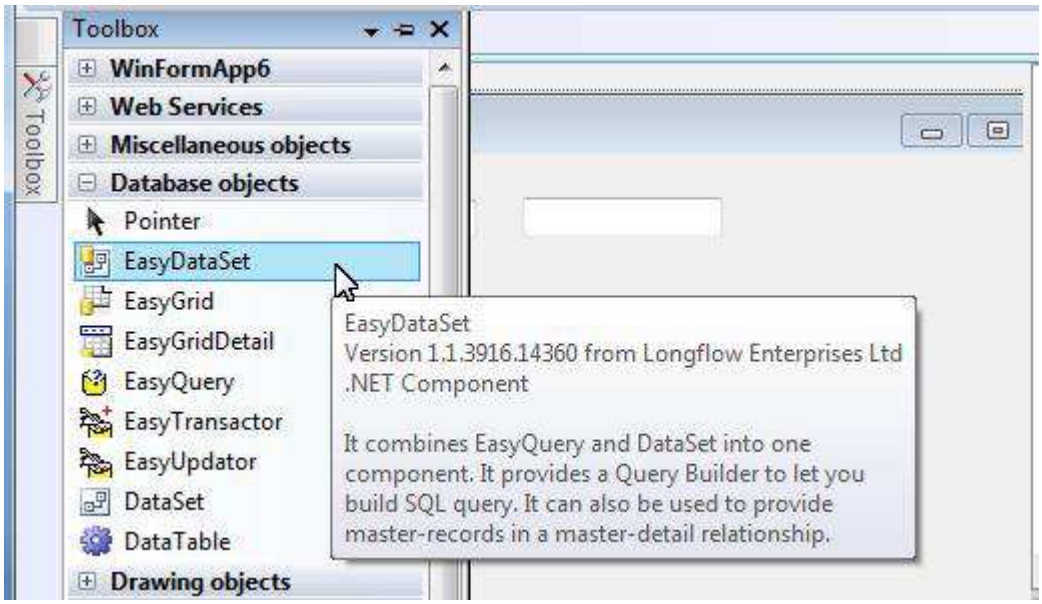
5 Data Binding

A component property may be linked to a field in a query to automatically get the data from database. This is called data binding.

As data sampling in the Query Builder shown, query results are a table of data. A table may have many records. At any time one record among the all records is the **current record**. The data from the current record will be used for the linked properties.

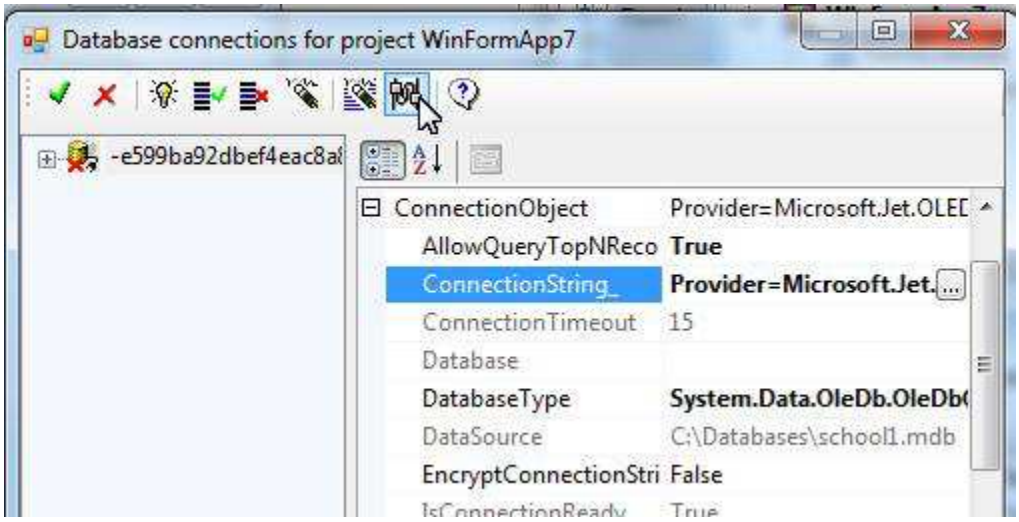
5.1 Create Data Binding at Design Time

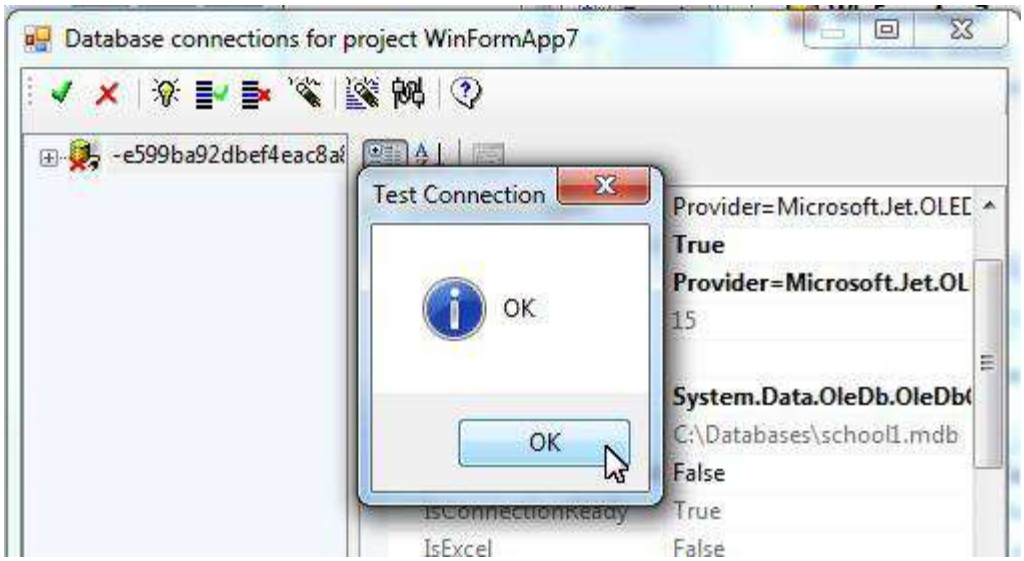
EasyDataSet component can be used to provide data to other components.

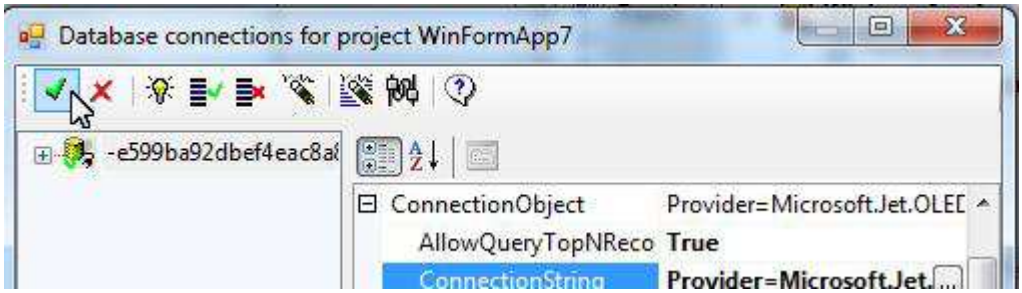


Set its DatabaseConnection property to connect to a database:

Set the DatabaseType and ConnectionString_; test the connection:

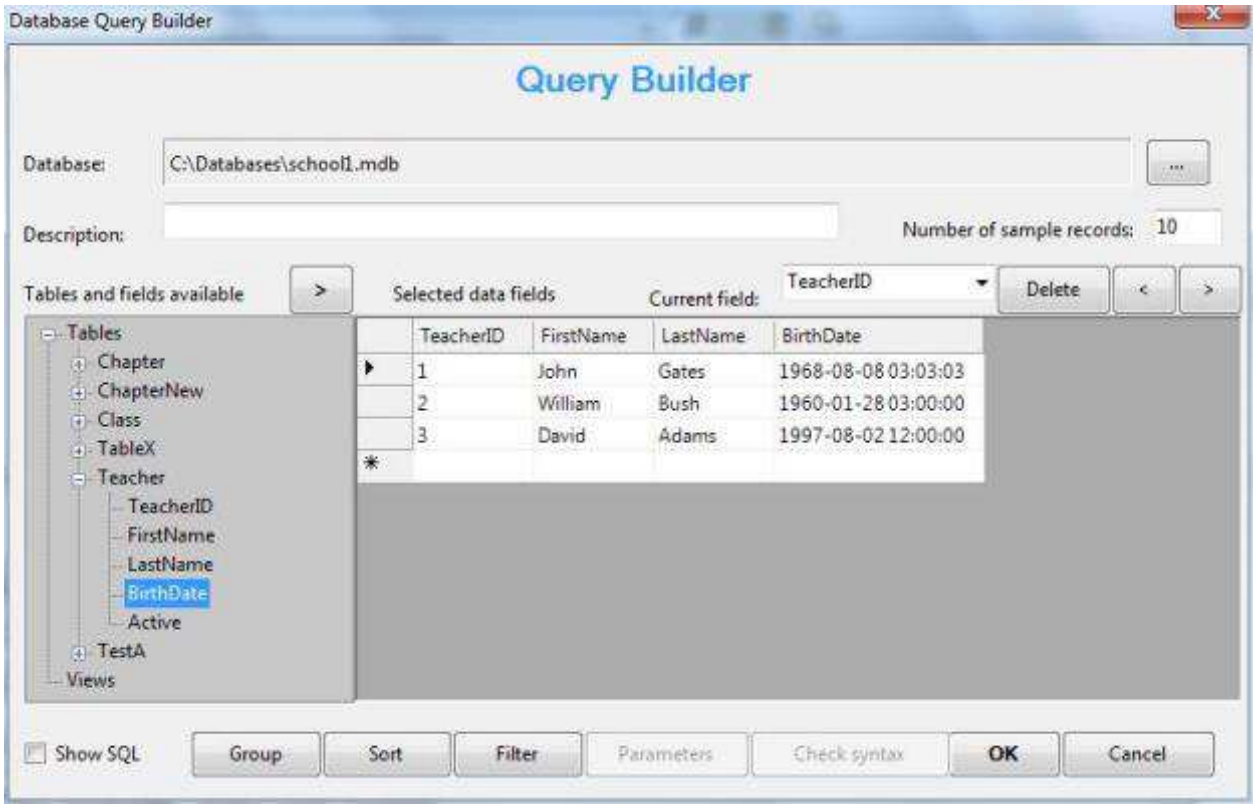






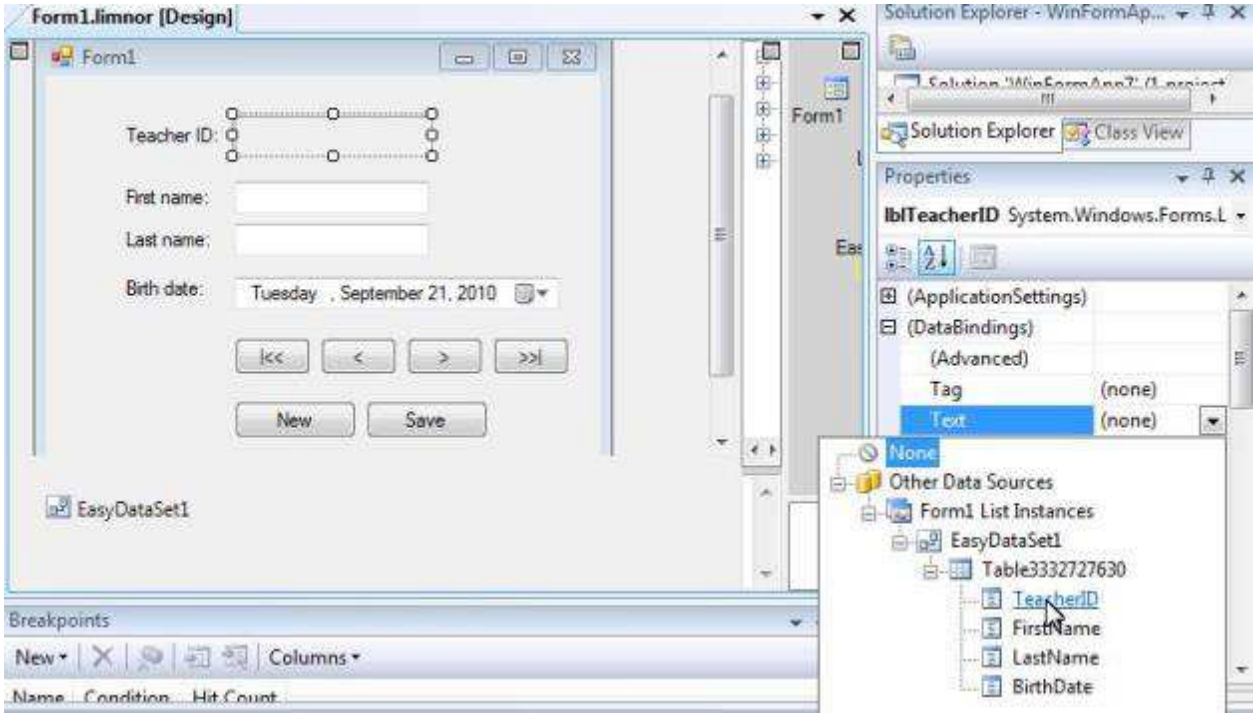
Set its SQL property to build a query to get data:

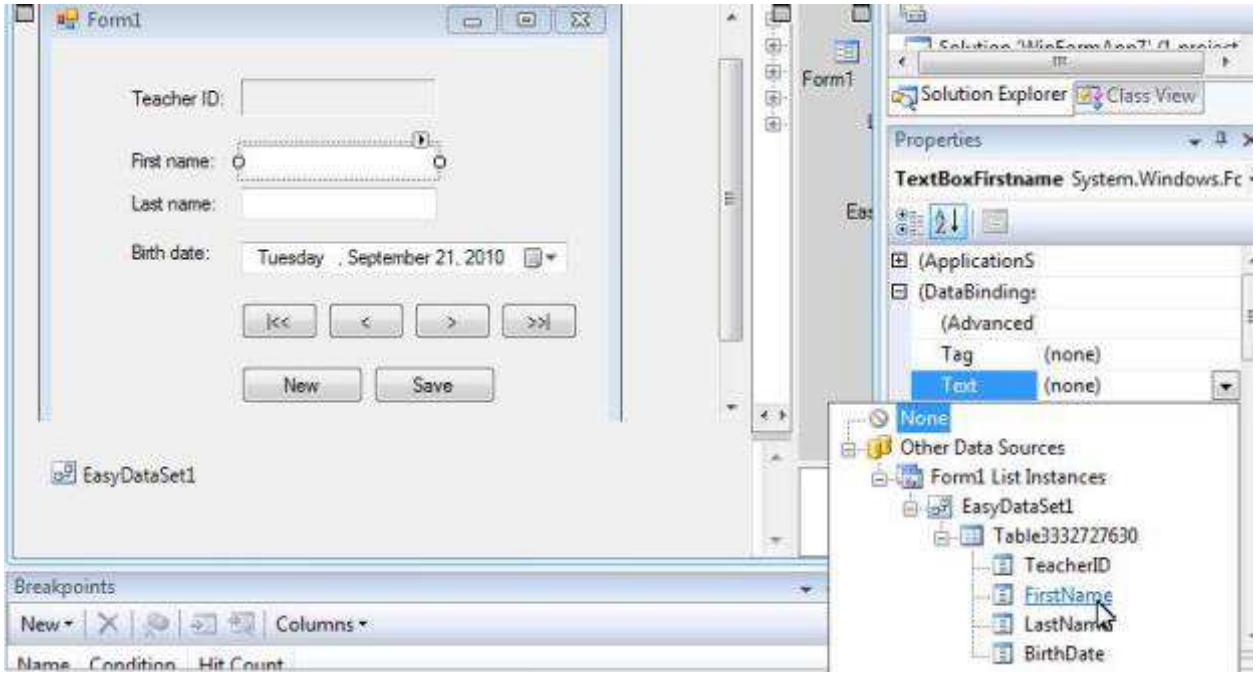
Select the data we want:

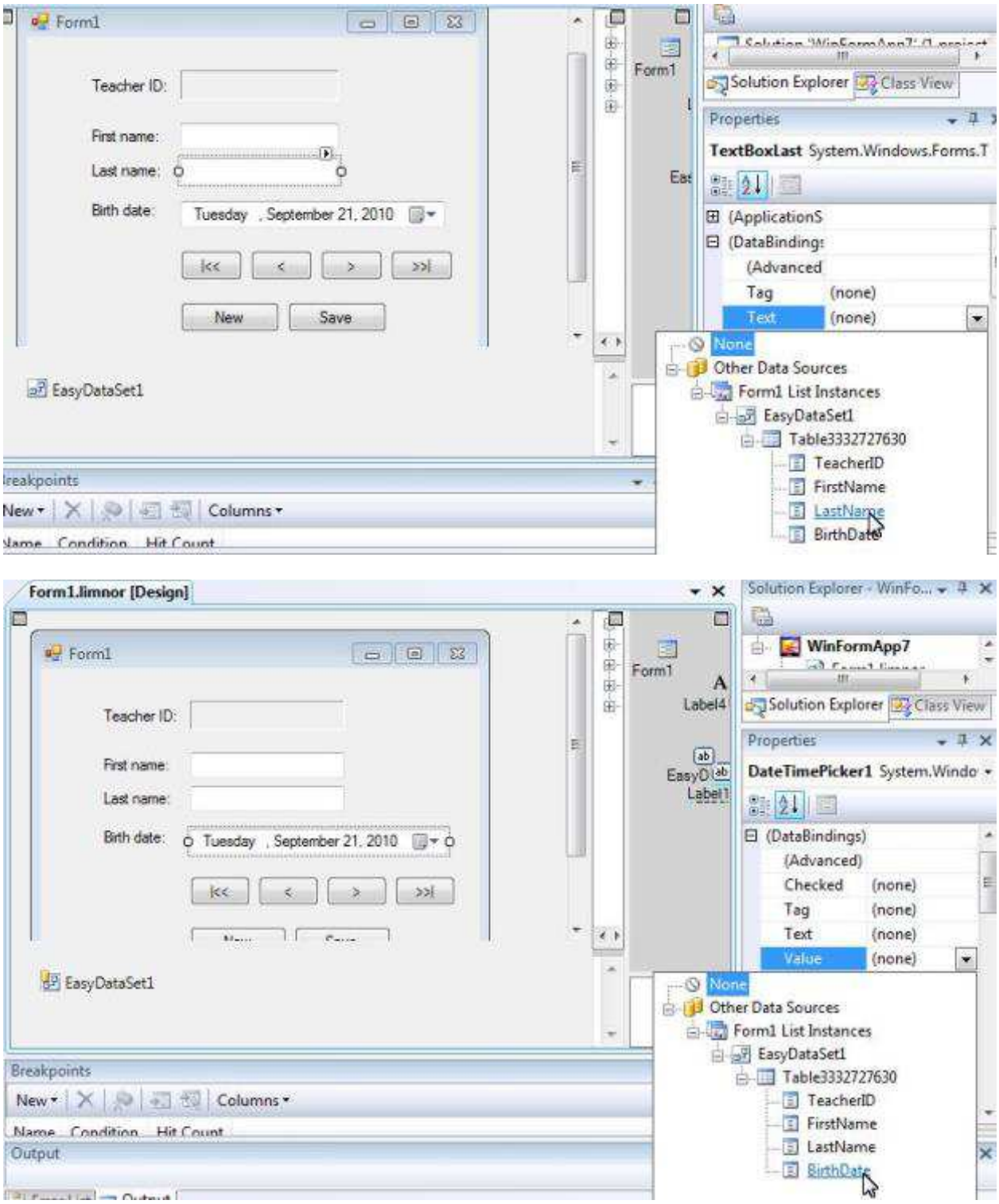


Suppose we use a Label to display TeacherID because it is a read-only field. We use Text Boxes for FirstName and LastName fields for allowing modifications. DateTimePicker can be used for BirthDate field.

Bind the Text property of the Label to TeacherID field. Bind the Text property of the first Text box to Firstname field and the second Text box to the LastName field. Bind the Value property of the DateTimePicker to the BirthDate property:







We have made 4 data bindings.

5.2 Record navigation

Note that the values of data binding are from the current record. The following methods of EasyDataSet can be used to set the current record:

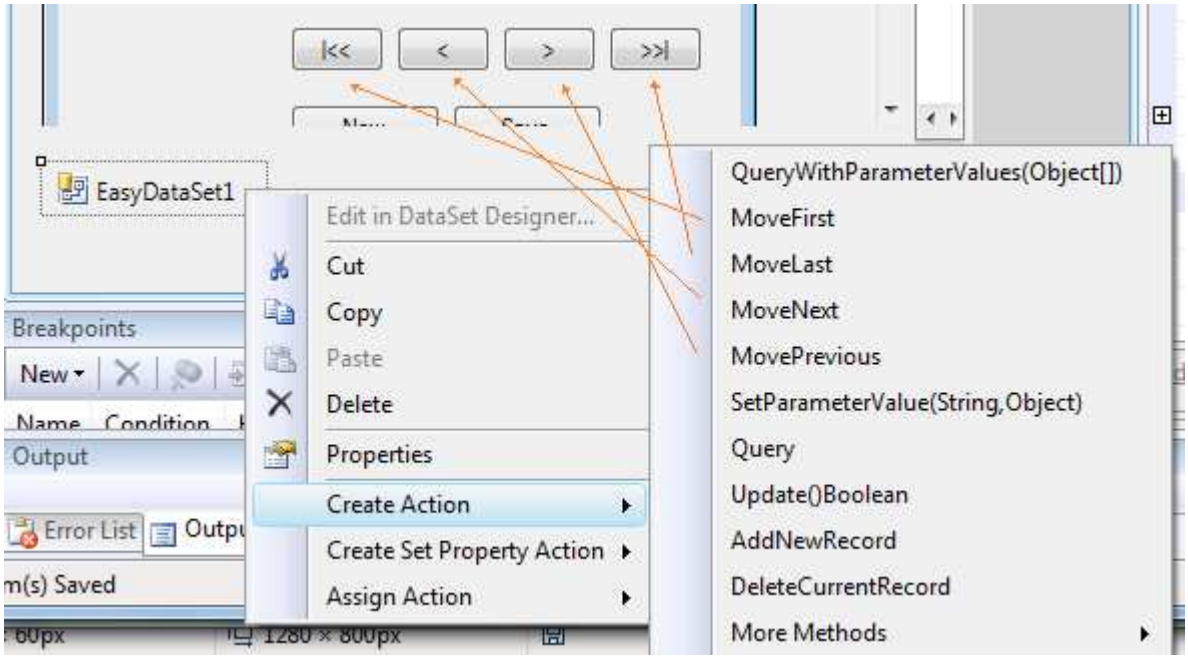
MoveFirst – Let the first record become the current record

MoveLast – Let the last record become the current record

MoveNext – Let the record next to the current record become the current record

MovePrevious – Let the record before the current record become the current record

We use 4 buttons to execute these 4 methods respectively:



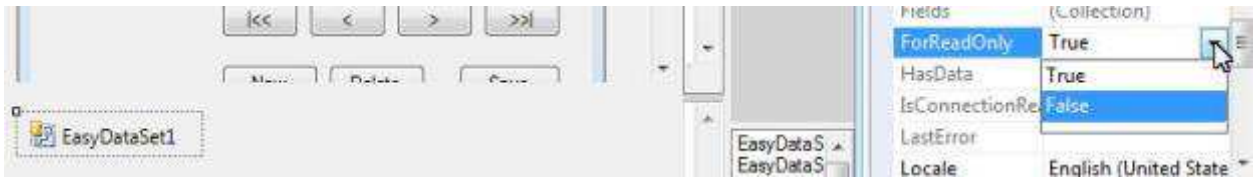
We are not going into the details of create actions and assign actions to events. If your context menu does not show MoveFirst, MoveLast, MoveNext and MovePrevious then you may choose “More Methods”.

Let’s test this sample application. The first record is the current record. We see the data from the first record.

On clicking the button ">", the next record becomes the current record:

5.3 Modify Data

Set ForReadOnly property to false to make it possible to modify data in database via the EasyDataSet:



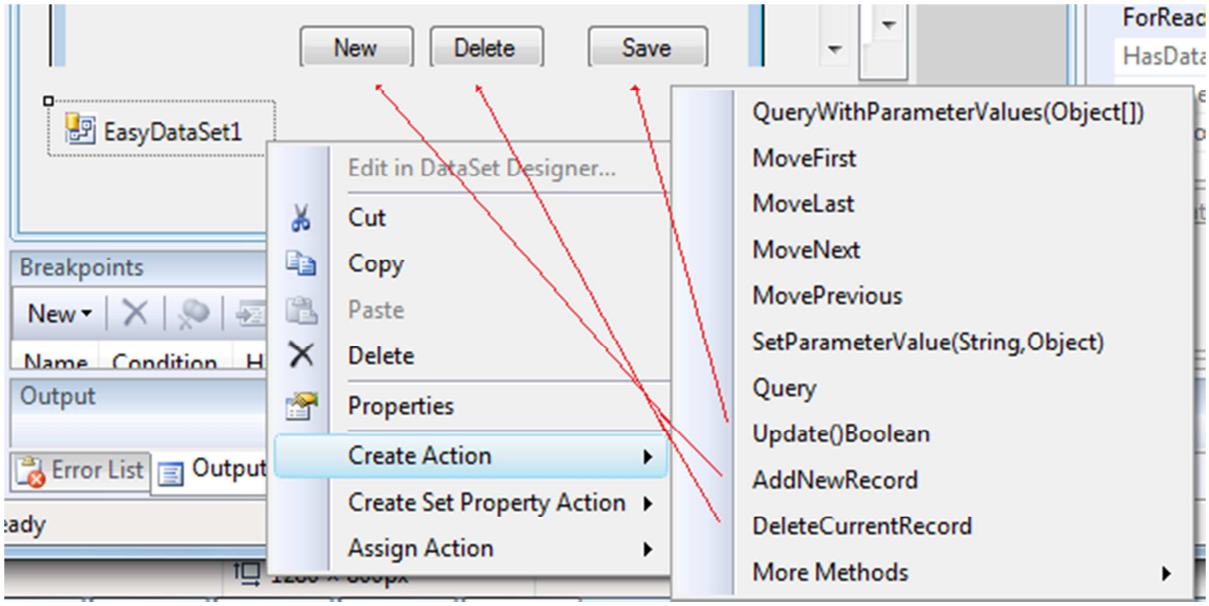
The query must include fields for a unique index so that it is possible to identify the record to be modified. In this sample, TeacherID is such a field.

EasyDataSet provides following methods for modifying database:

AddNewRecord – Add a new record

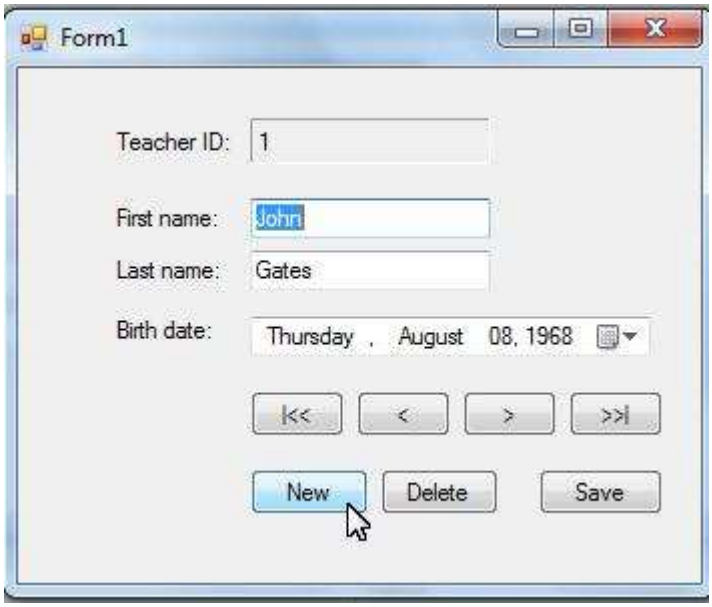
DeleteCurrentRecord – Delete the current record

Update – Save all modifications to the database. Modifications include adding new records and deleting records, and data modifications made through the UI components bound to EasyDataSet.

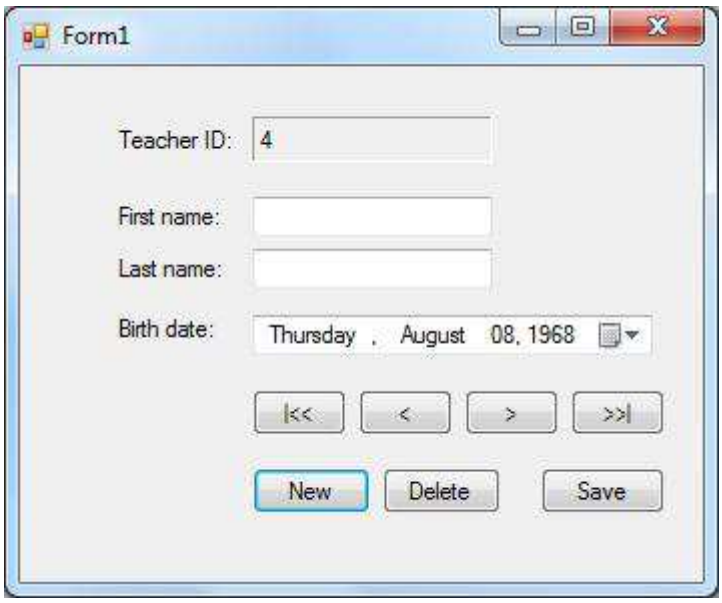


If your context menu does not show Update, AddNewRecord and DeleteCurrentRecord then you may choose “More Methods” to find them.

Test the application. Click New button:

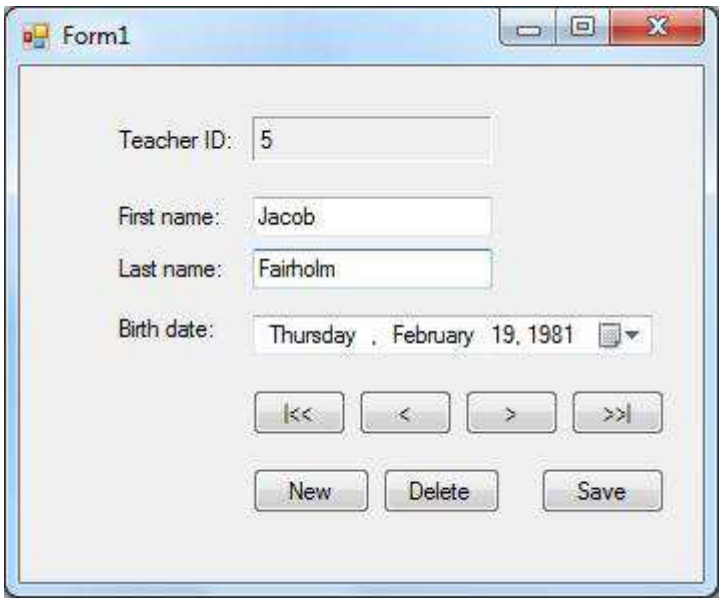


The newly added record becomes the current record:



Enter data for this record:

We may click New button again to add more new records:



Note that the TeacherID for the new records are temporary values. The real values are generated by the database engine when the new records are created in the database.

Note that all modifications, adding new records, modifying existing records, and deleting existing records, are still just in memory, not in the database.

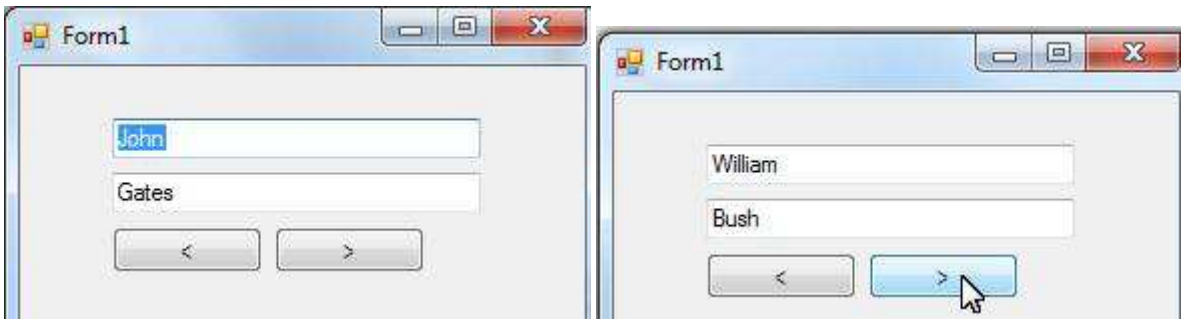
All modifications will be applied to the database once the Update action is executed. In this sample, we assign the Update action to the Save button.

Click Save button. We may expect that the modifications should be saved in the database. To verify it, close the application and re-run it. Navigate through the records. We can see that the modifications were actually in the database.

5.4 Create Query and Data Binding at Runtime

We may create some actions to make data bindings at runtime.

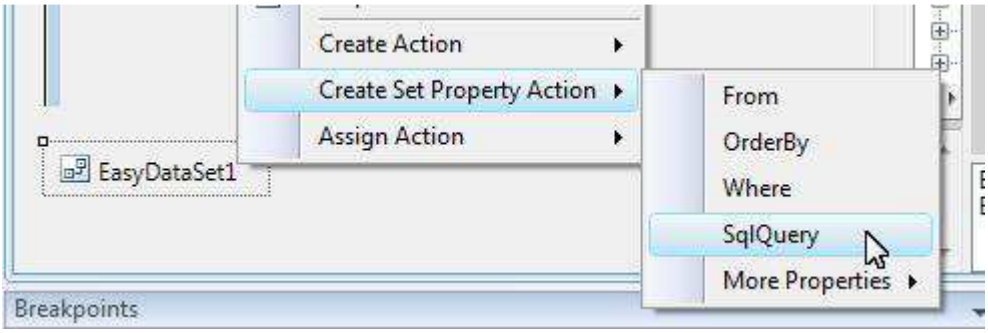
Suppose we make data binding to FirstName and LastName of a query, like we did in previous sample:



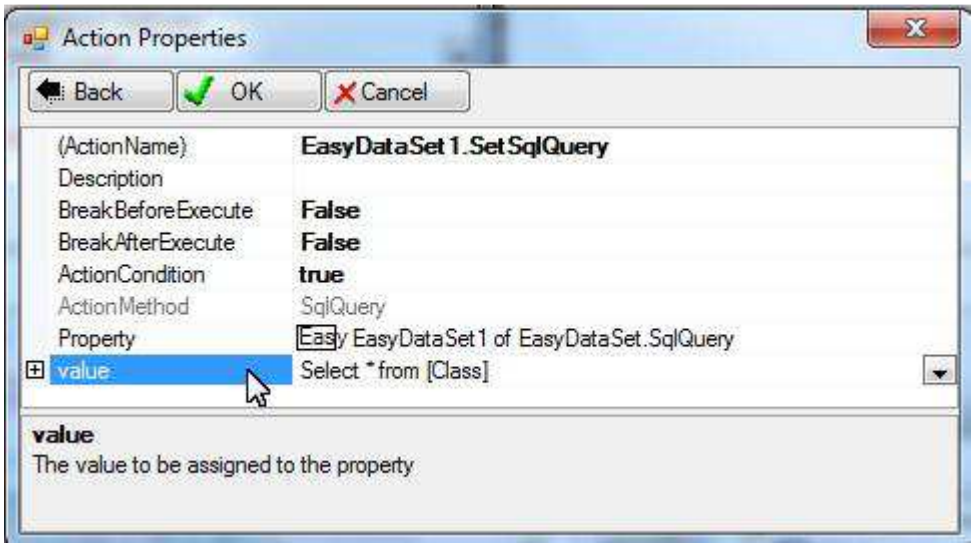
5.4.1 Use Query at Runtime

Setting SqlQuery property of EasyDataSet at runtime may change the query at runtime. It is not needed for creating data binding at runtime, but we use this opportunity to describe the changing query at runtime.

Right-click EasyDataSet; choose “Create Set Property Action”; choose SqlQuery method:



For simplicity, type in query Select * from [Class] to retrieve all records from [Class] table:



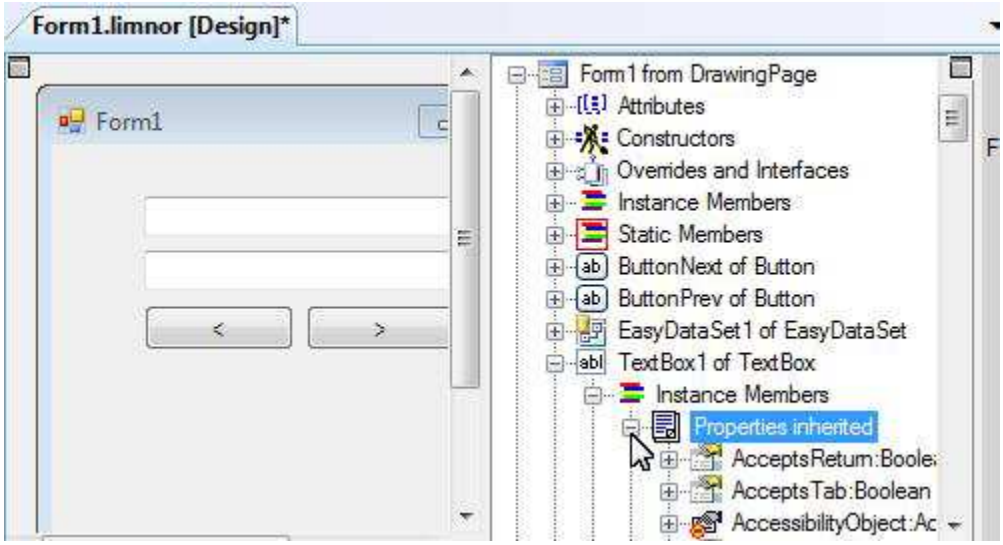
At runtime execute this action will fetch all records from [Class] table.

5.4.2 Remove Data-Binding at Runtime

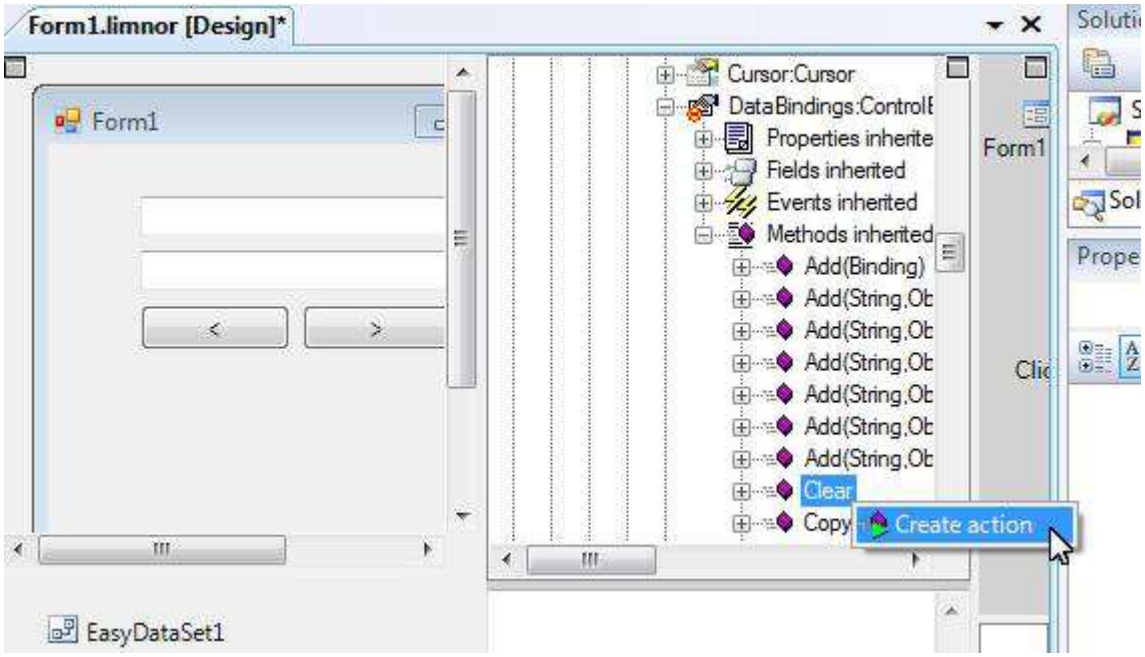
At design time we set data-binding of the two Text boxes to bind to fields from [Teacher] table. On changing the query to retrieve data from [Class] table, the data-bindings will be no longer valid. We may remove the data-bindings.

It is a good idea to remove any existing data-bindings before creating new data-bindings.

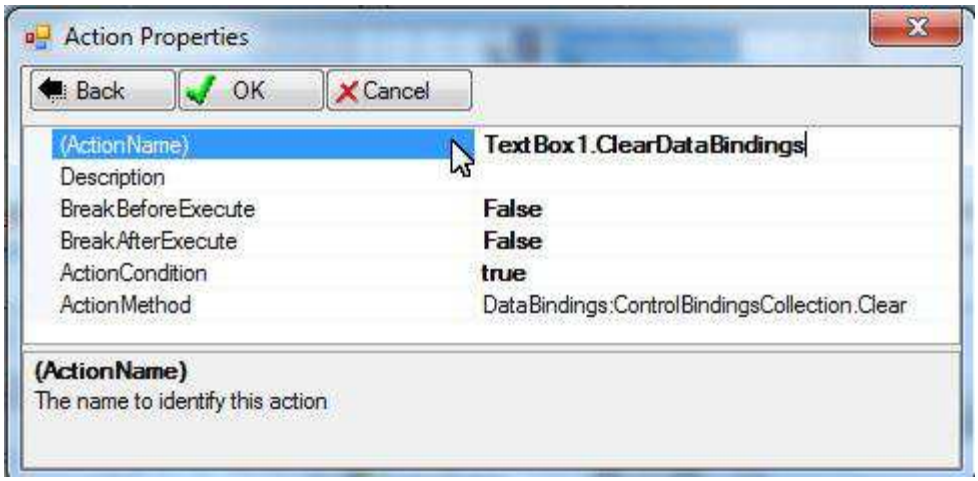
Expand “Properties inherited” node of the Text Box:



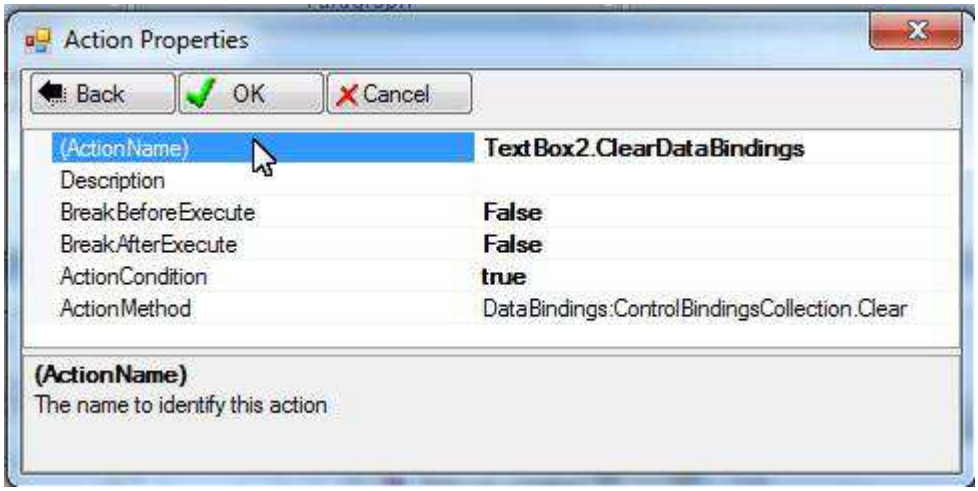
The DataBindings property of the TextBox holds all data-bindings. Right-click Clear method; choose "Create action":



Change the action name to ClearDataBindings:



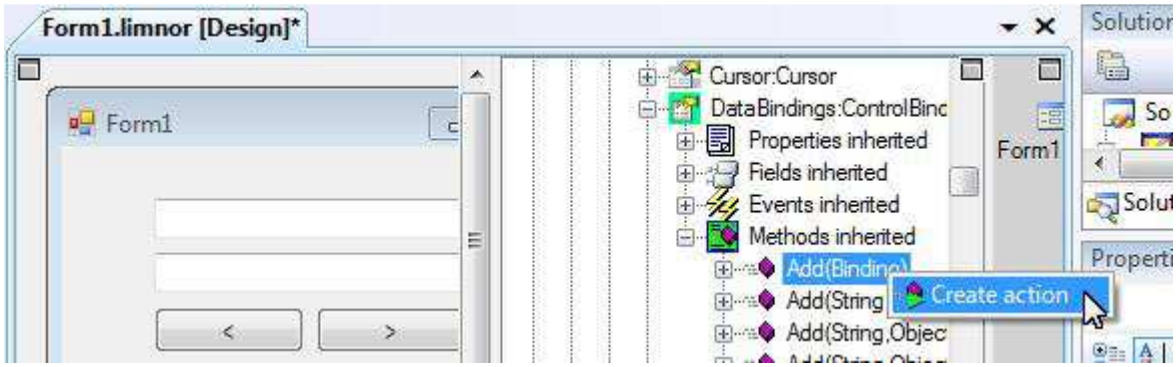
Create another action for the other TextBox:



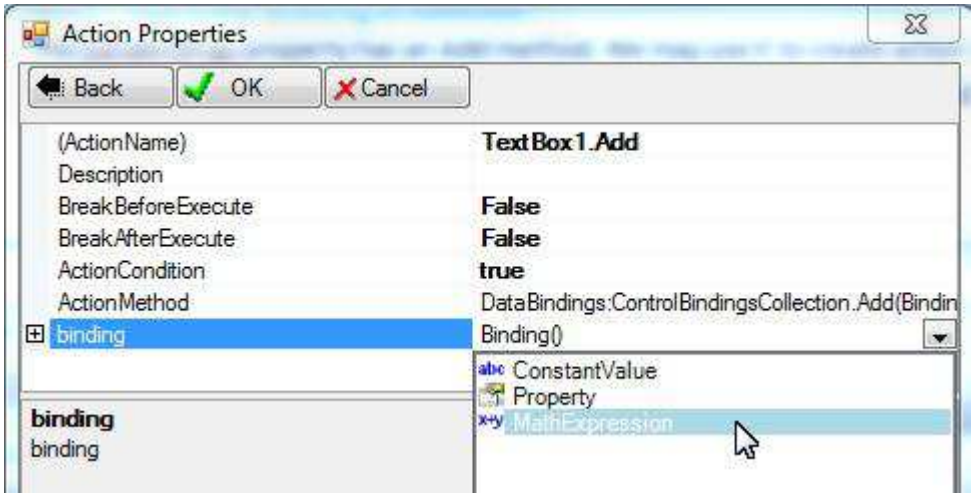
5.4.3 Add Data-Binding at Runtime

The DataBindings property has an Add method. We may use it to create action to add data-binding at runtime. The data-binding to be created can be created by "CreatedataBinding" method.

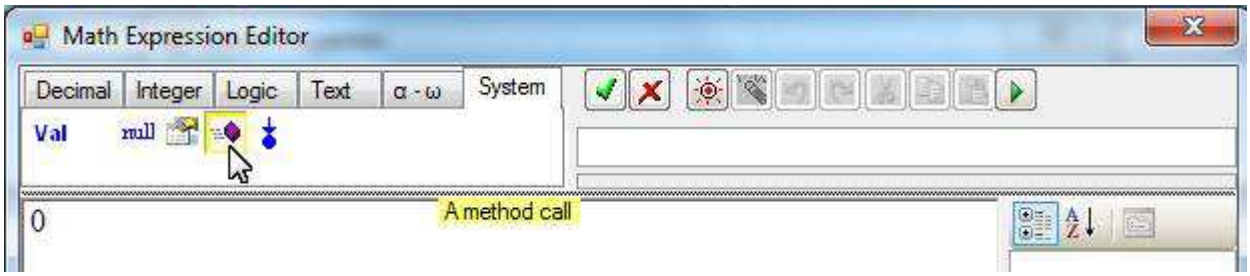
Right-click the Add method; choose "Create action":



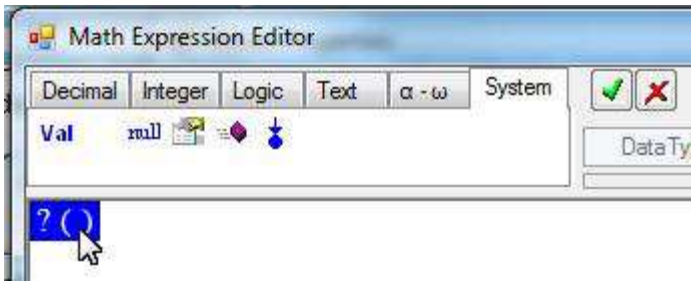
Choose “Math Expression” so that we may use “CreatedataBinding” method of EasyDataSet to create data-binding to be added:



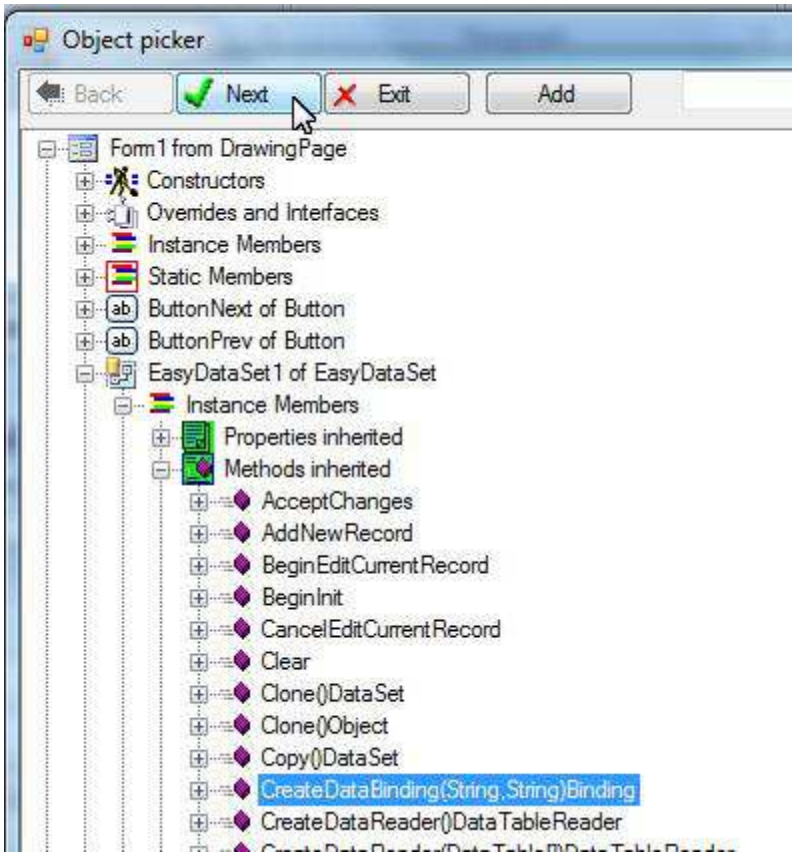
Click the method icon to use a method:



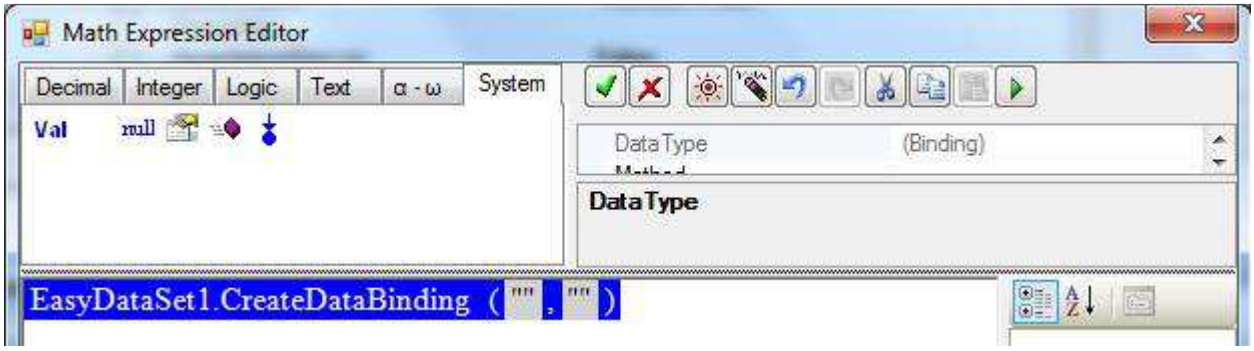
Double-click the element to select method:



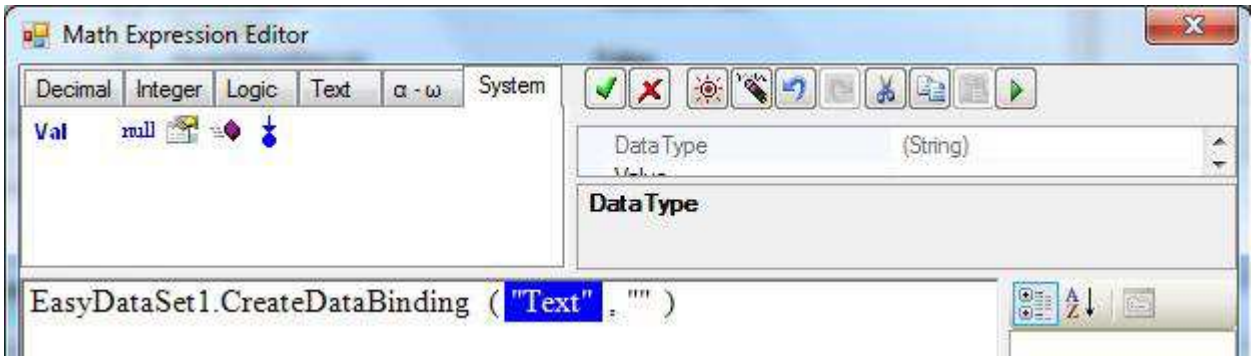
Select “CreateDataBinding” method of EasyDataSet:



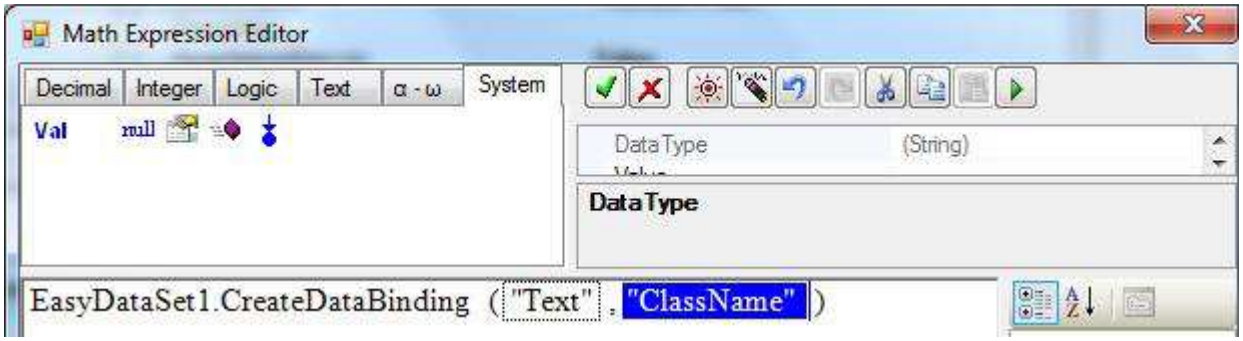
The CreatedataBinding method is used:



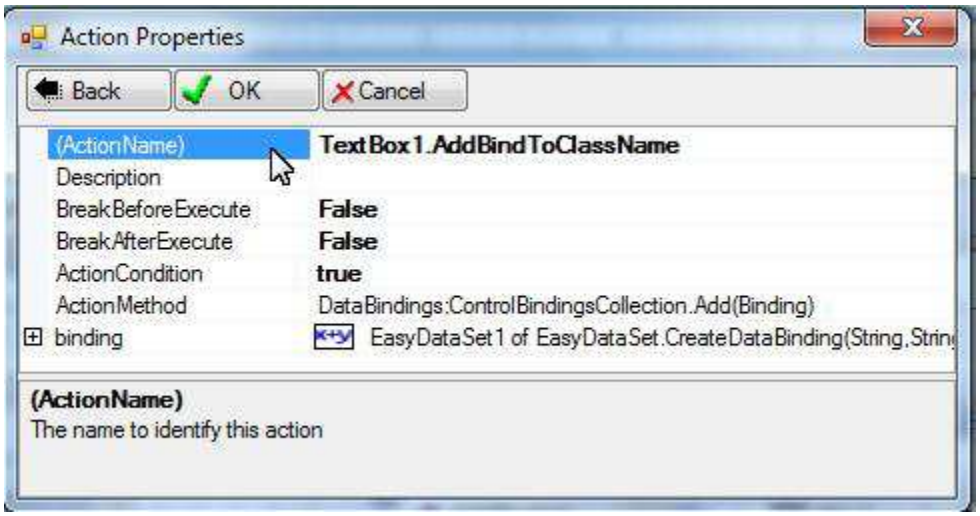
Select the first parameter, type Text. This is to say that the data is to be bound to a property named Text



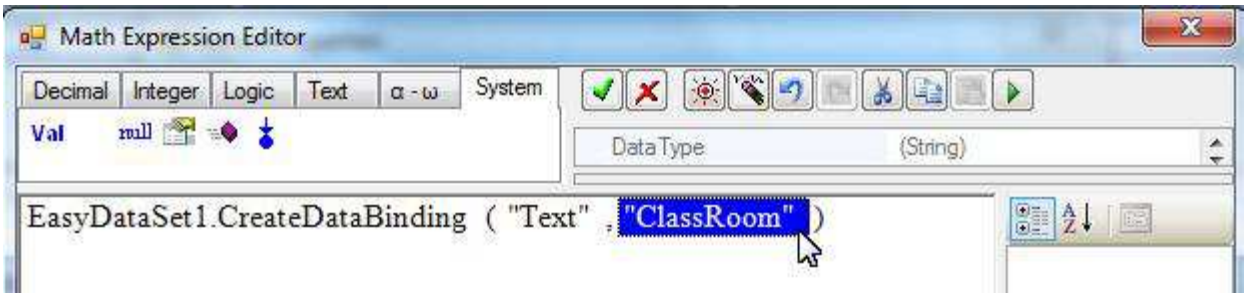
Select the second parameter and type “ClassName”. This is to say that the data is to be bound to a database field named “ClassName”:



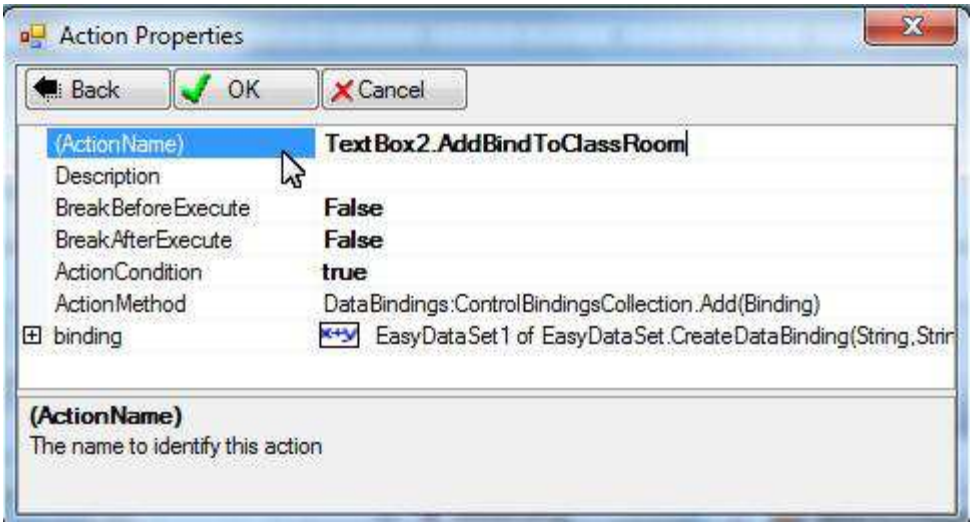
Change the action name to make it more descriptive:



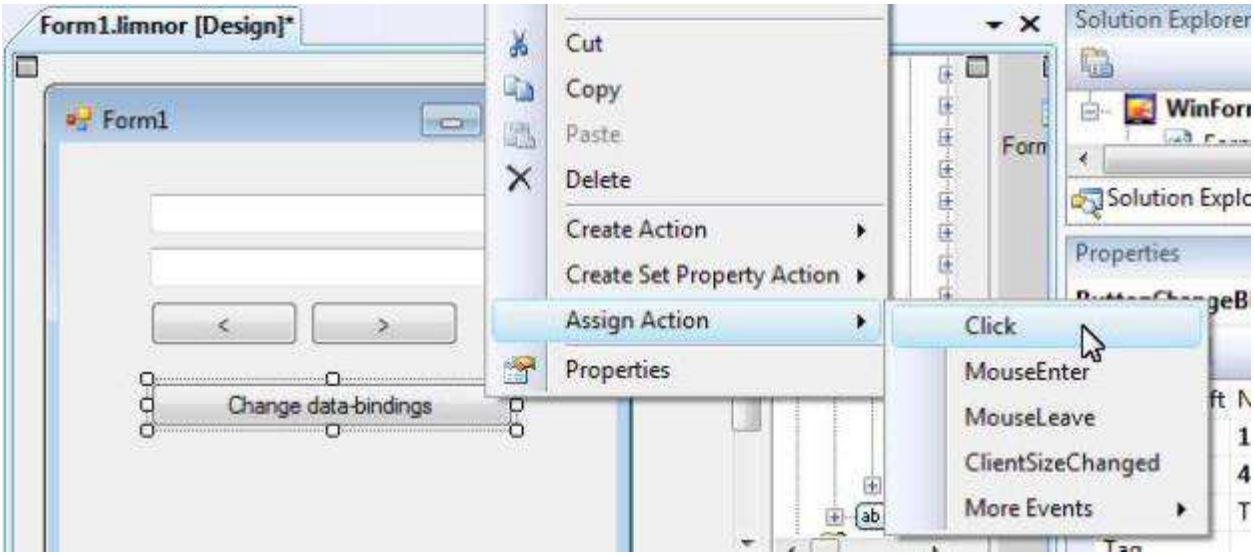
For the other TextBox, also create an action to add a Data-Binding. Bind it to “ClassRoom” field:



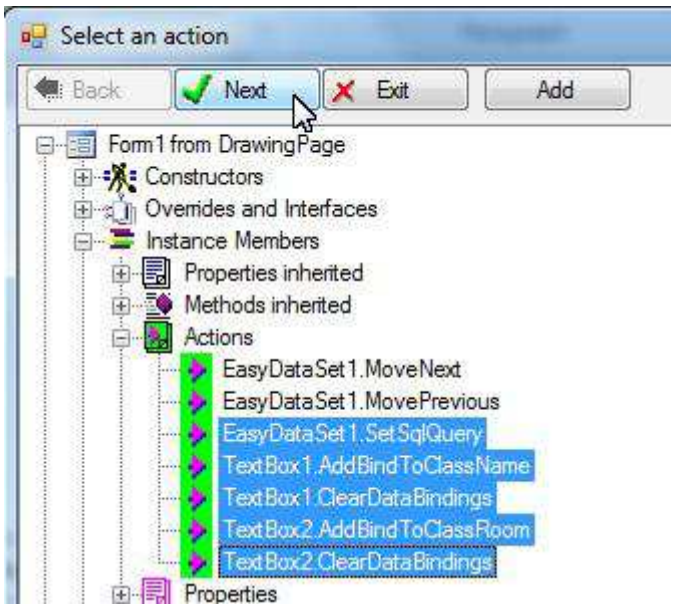
Also change the action name to make it more descriptive:



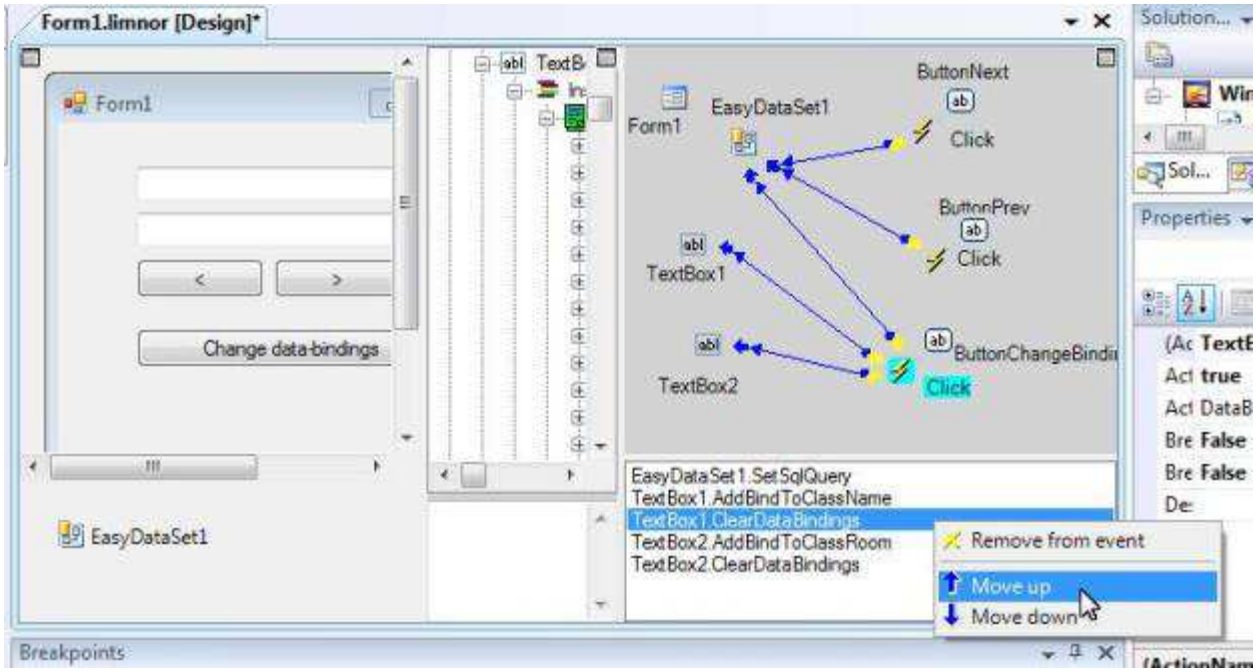
We have all the actions needed. Let's assign all the actions to a button's Click event:

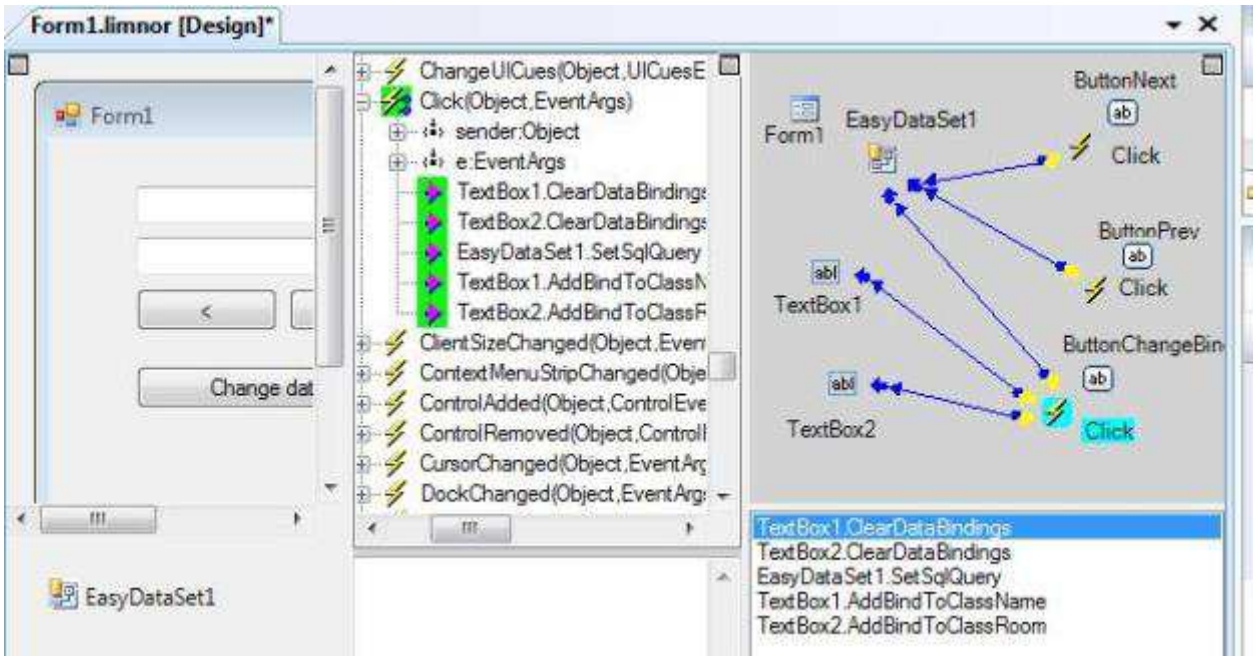


Select all the actions to be used, click Next:



Move actions ClearDataBindings up so that they will be executed first:

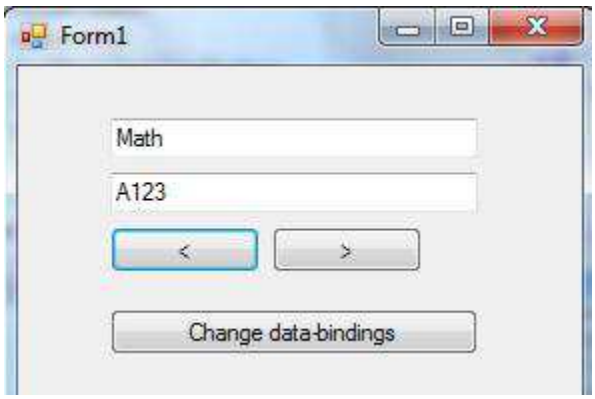




We may test the application now.

The text boxes displays data from the Teacher table:

Click button “Change data-bindings”. We can see that the text boxes now display data from Class table:

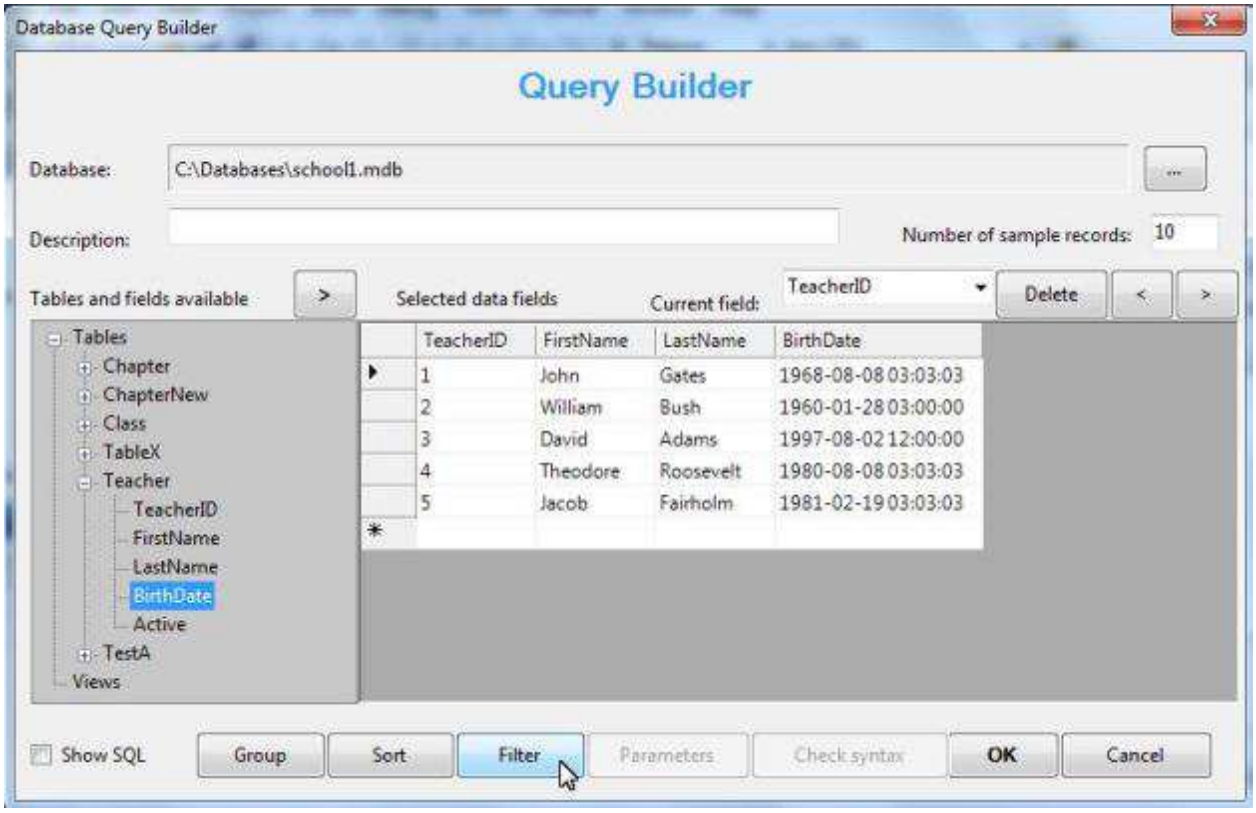


6 Query with Parameters

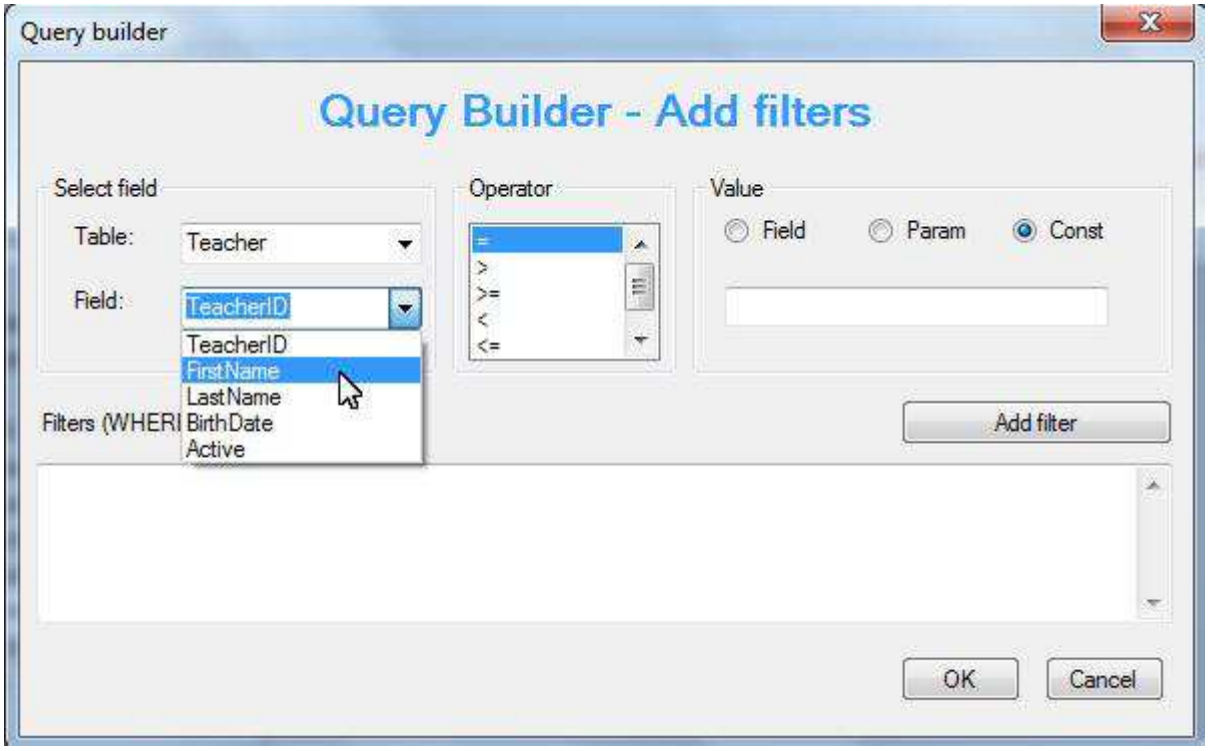
6.1 Add parameters in query

Parameters can be used in a query so that at runtime the user may provide values for the parameters to get desired query results.

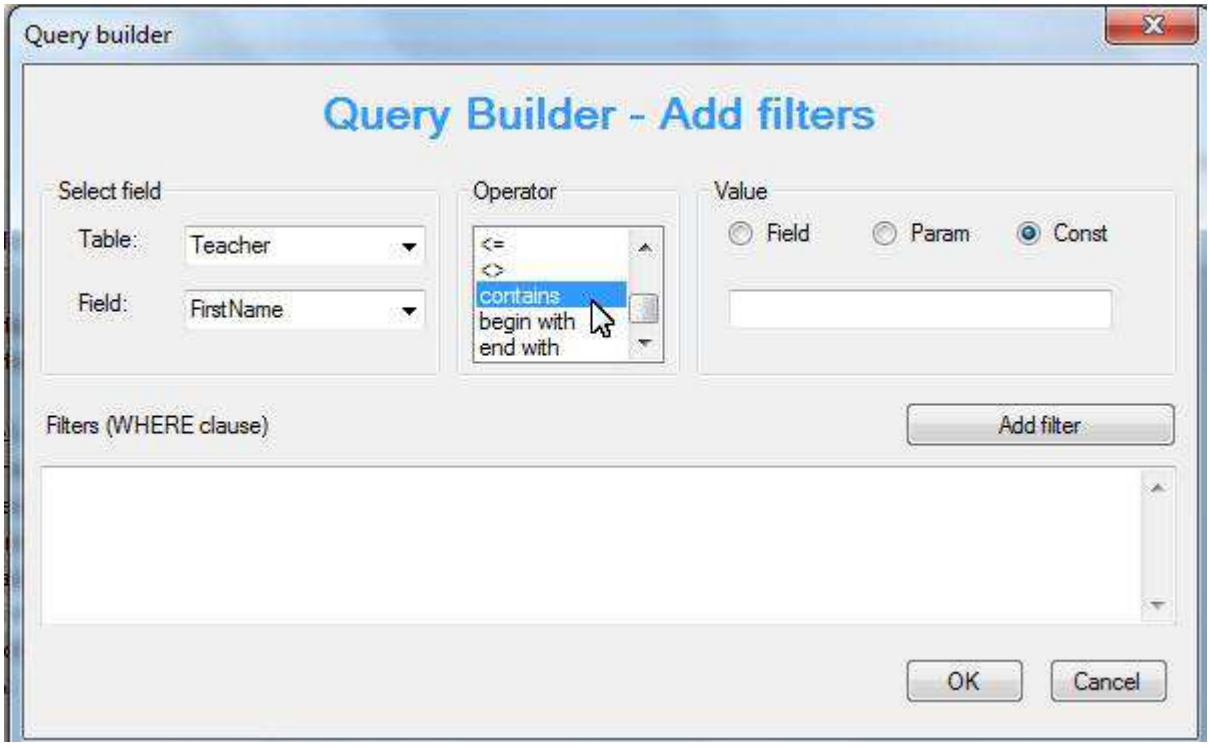
Let's use EasyGrid to show the concept. Set its DatabaseConnection property to connect to a database. Then set its SQL property to get data. After selecting all the fields we want click button Filter:



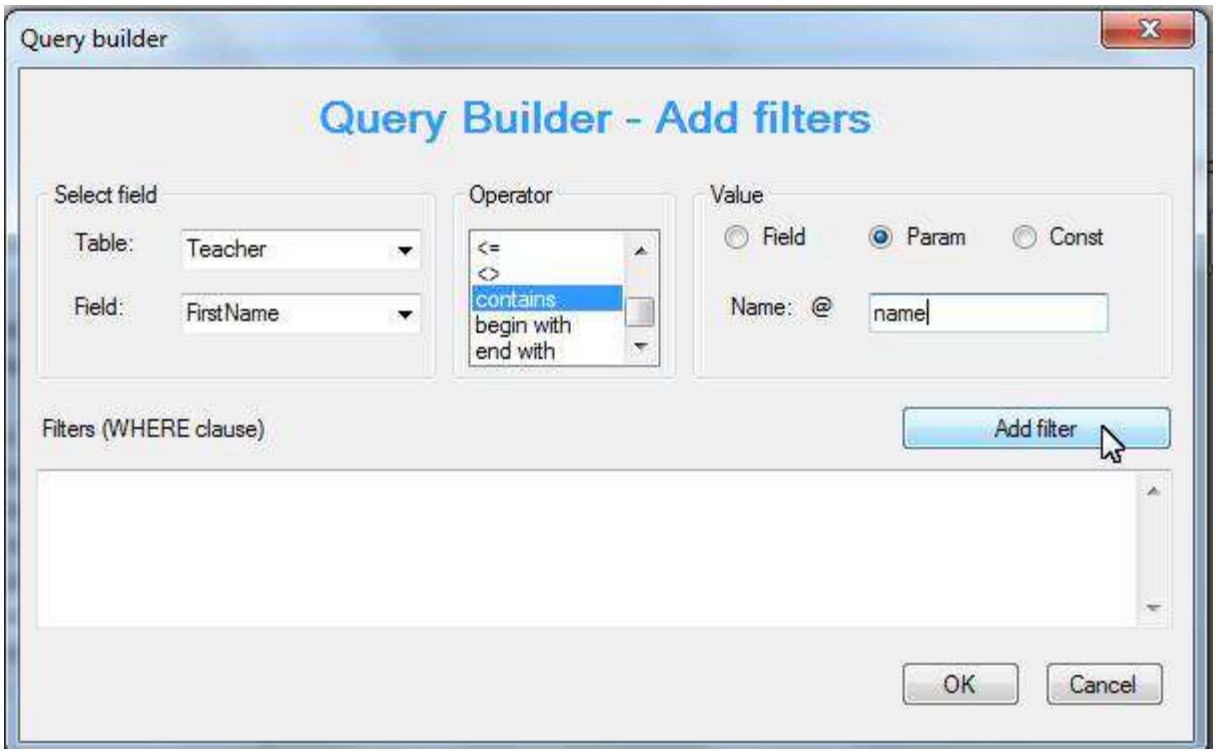
Select FirstName field:



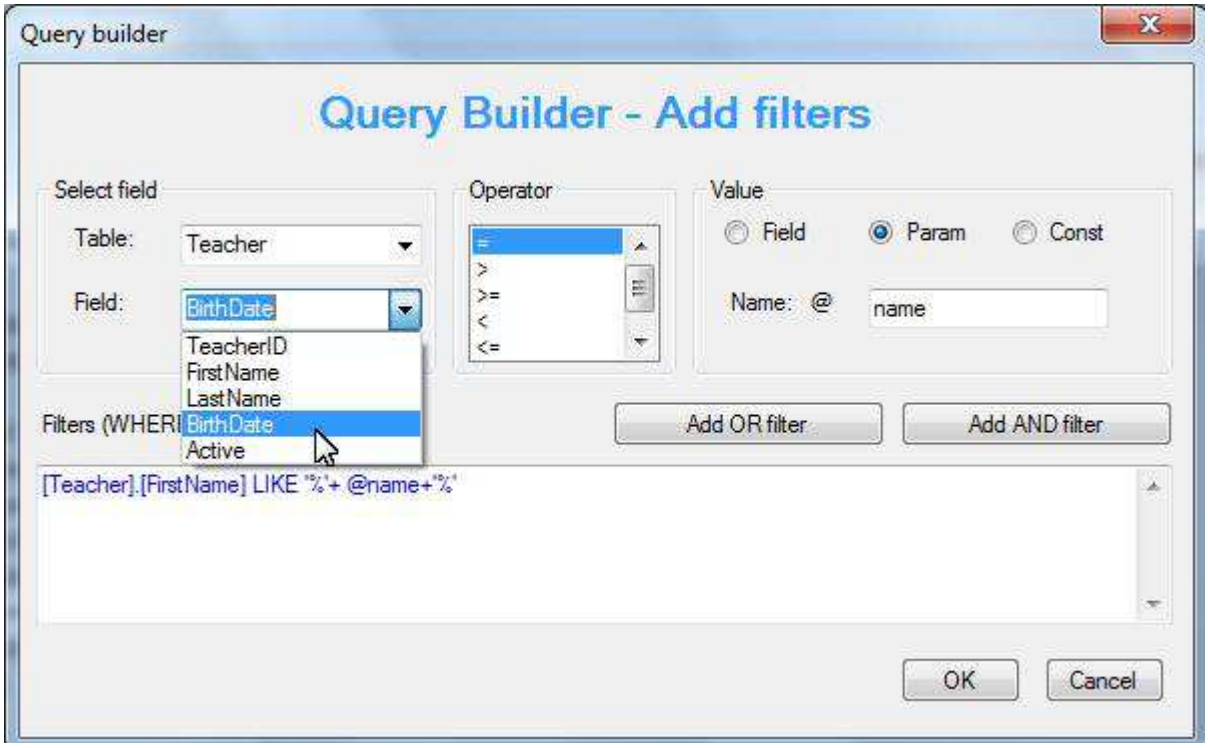
Select Contains:



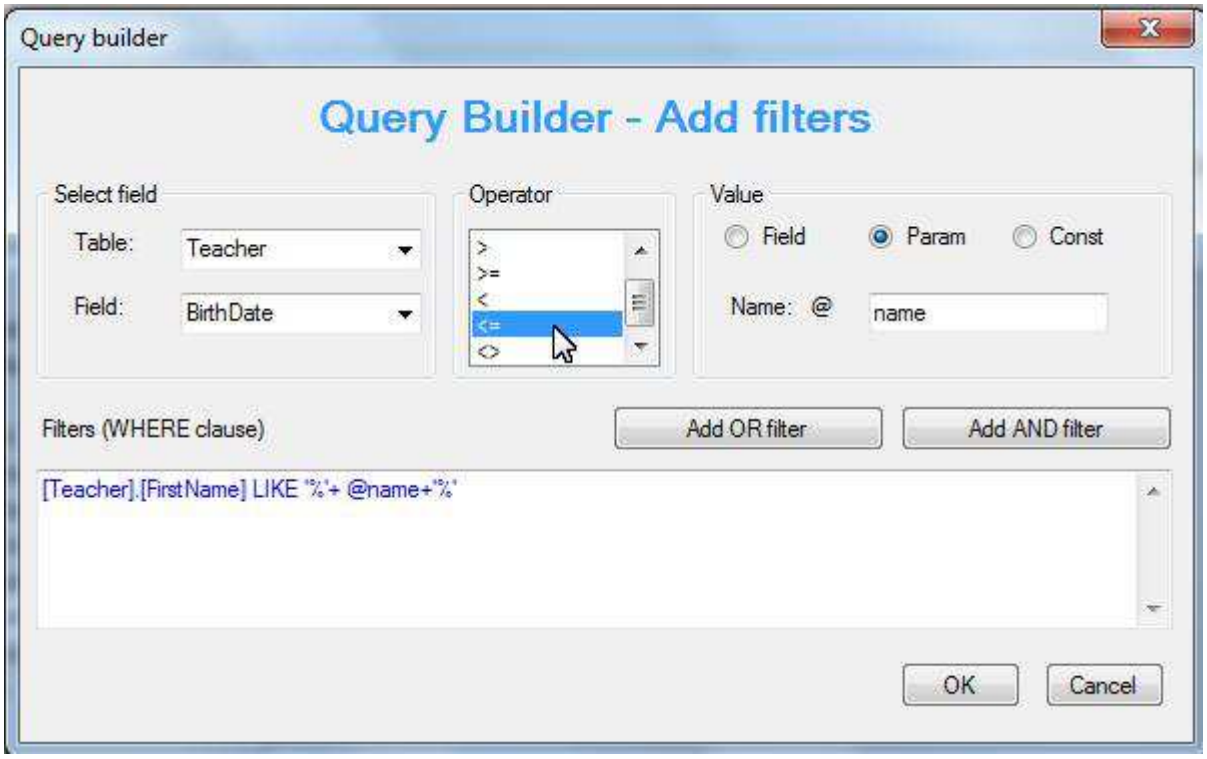
Select Param and give a parameter name. Click “Add filter”:



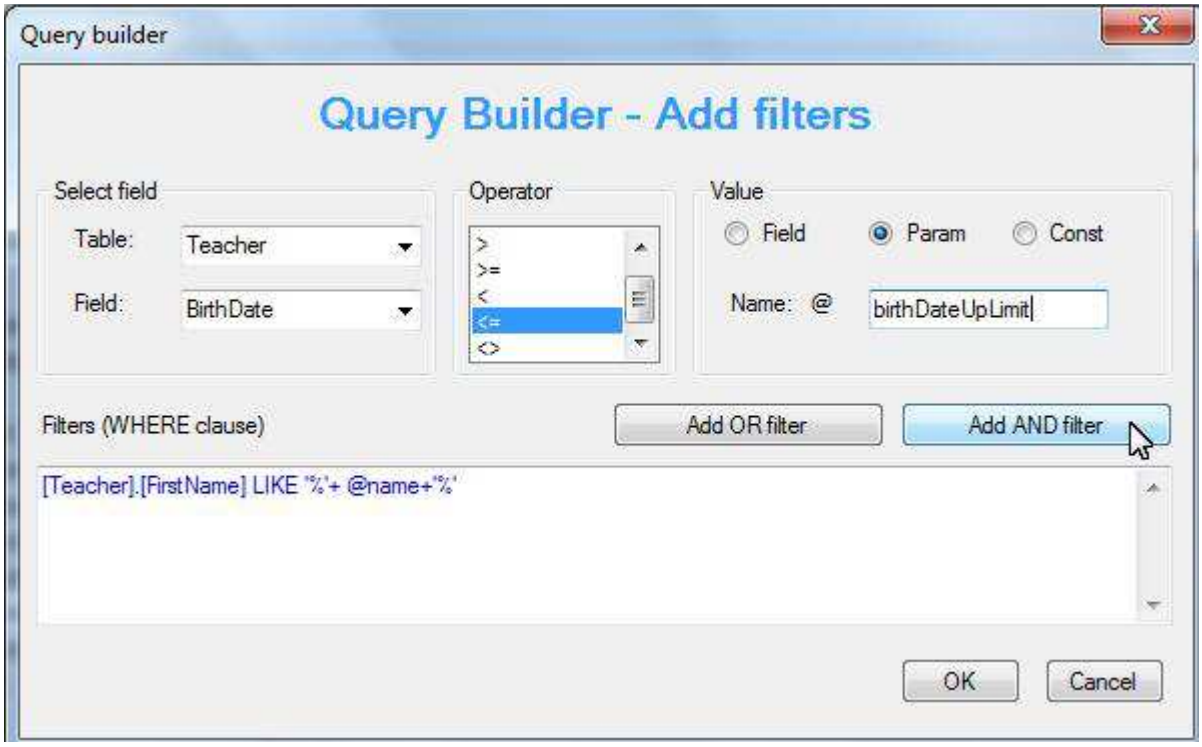
Select BirthDate field:



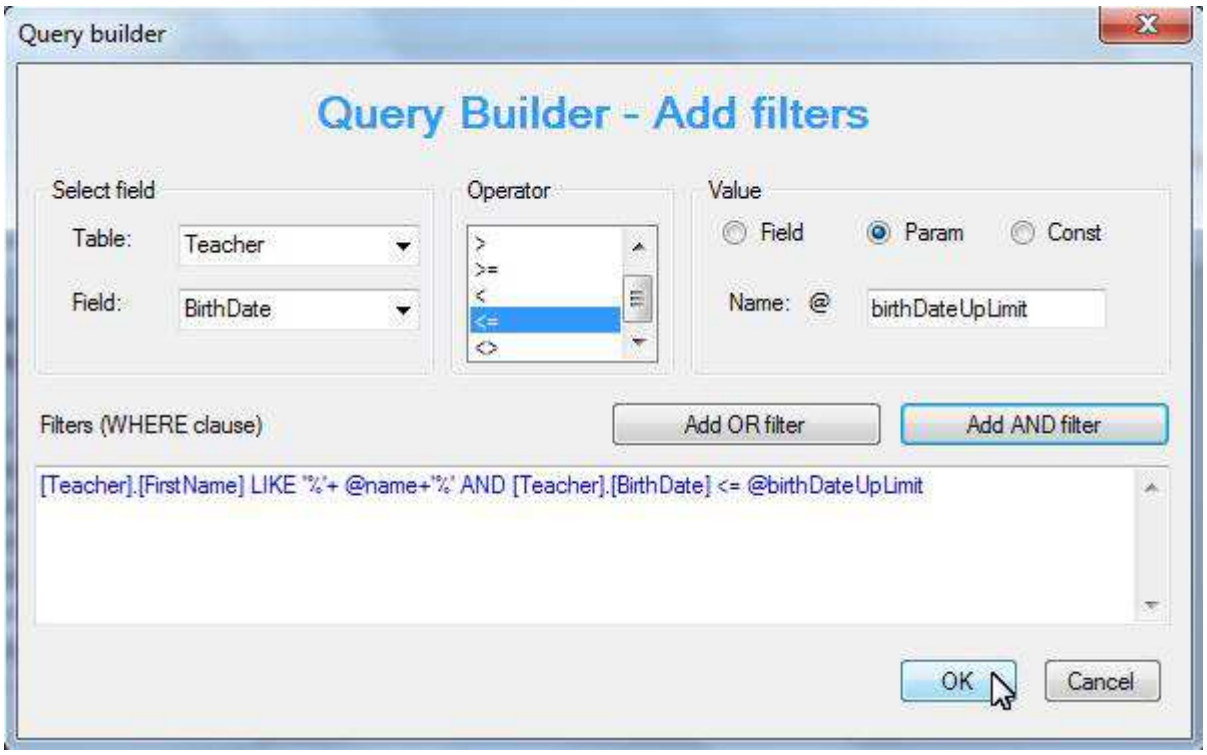
Select "<=":



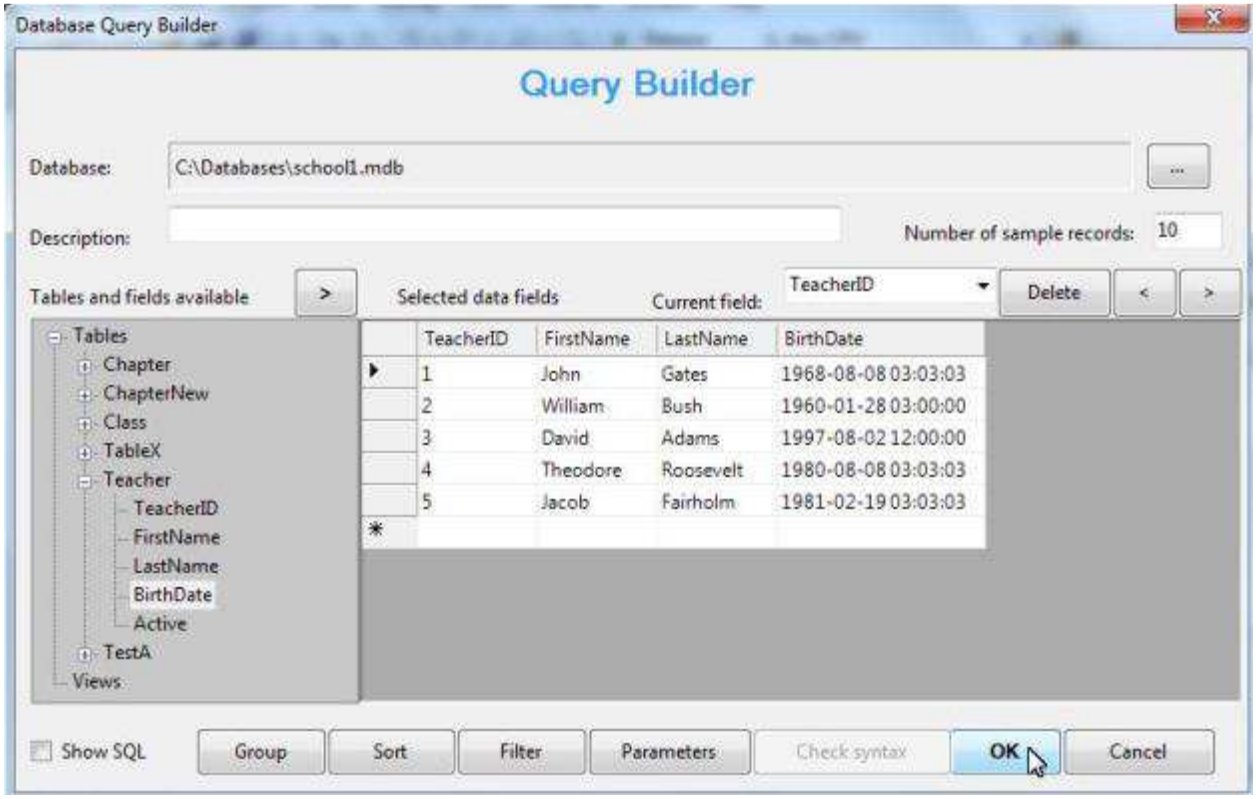
Select Param. Give a name. Click “Add AND filter”:



This is the filter we build:

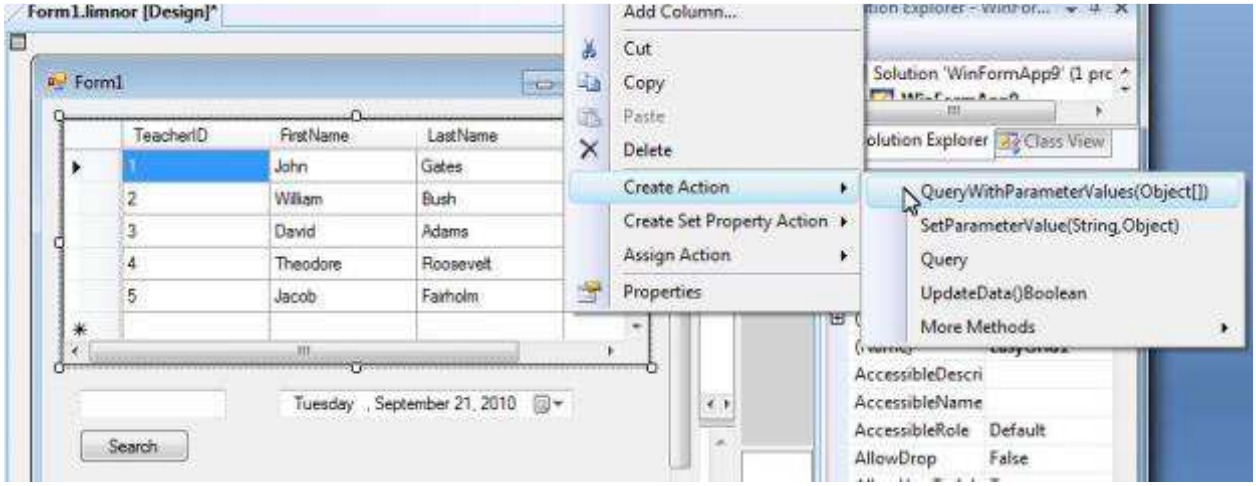


Click OK:

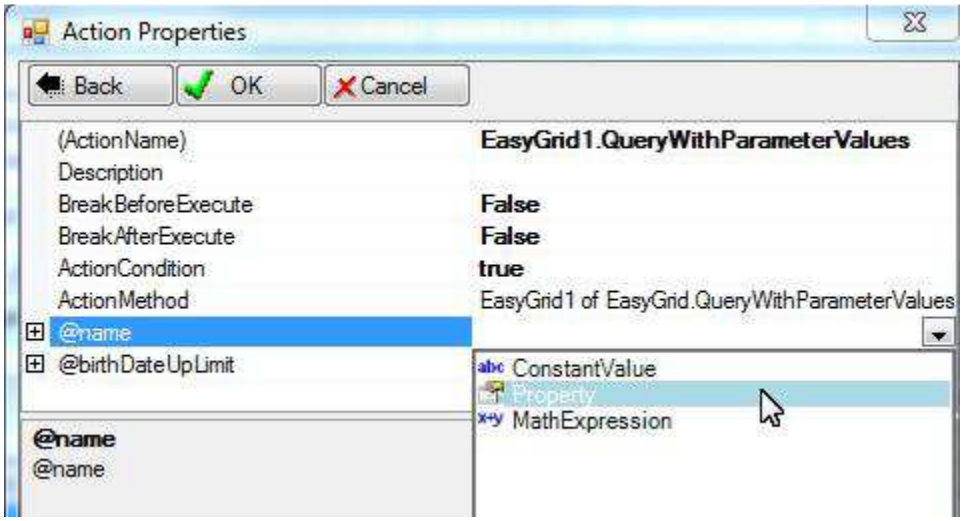


6.2 Create QueryWithParameterValues action

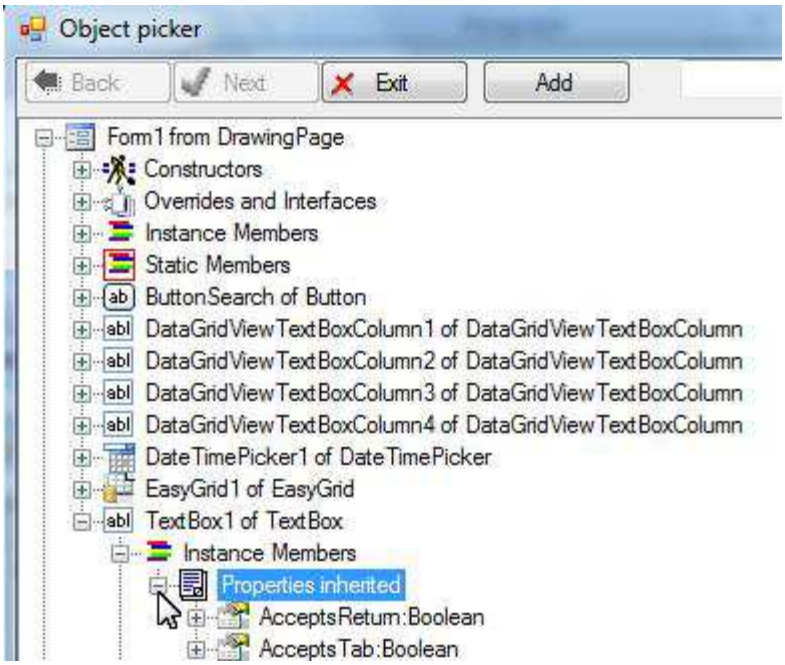
Use a Text Box and a DateTimePicker to provide parameter value. To do the search, right-click EasyGrid; choose “Create action”. Choose”QueryWithParameterValues” method:



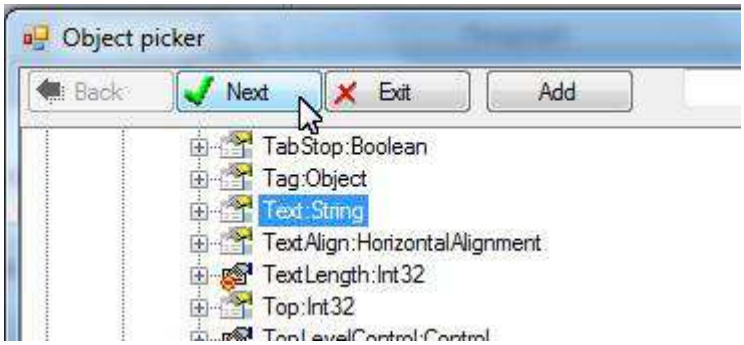
The query parameters are properties of the action. Select “Property” for @name to provide value:



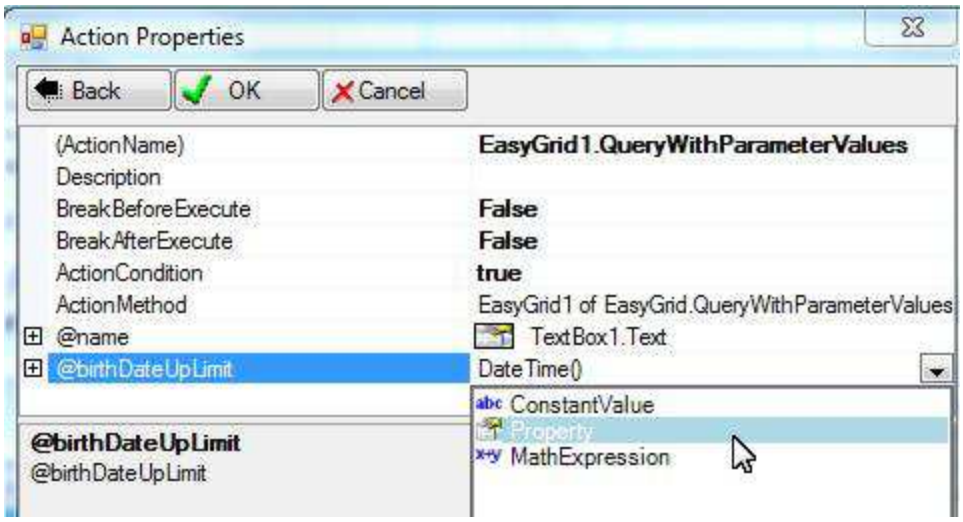
We want to select the Text property of the text box. Expand “Properties inherited” node of the text box:



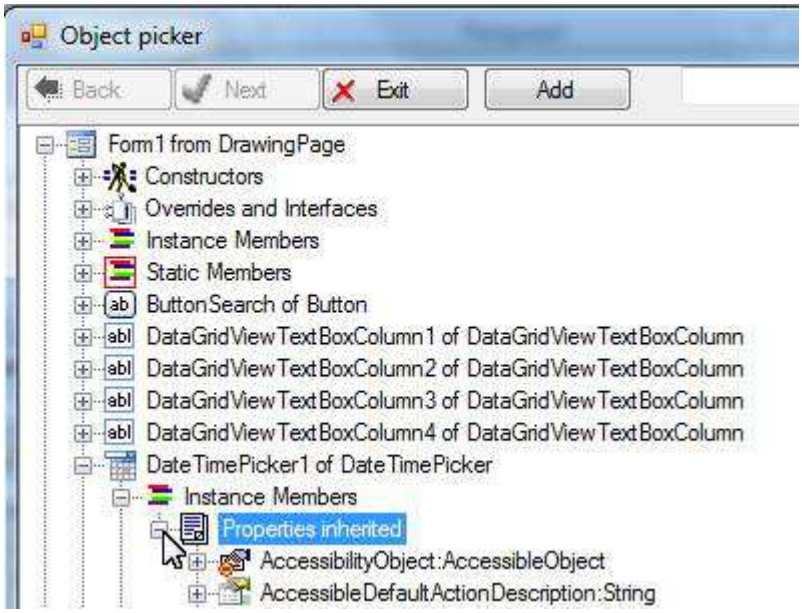
Select the Text property. Click Next:



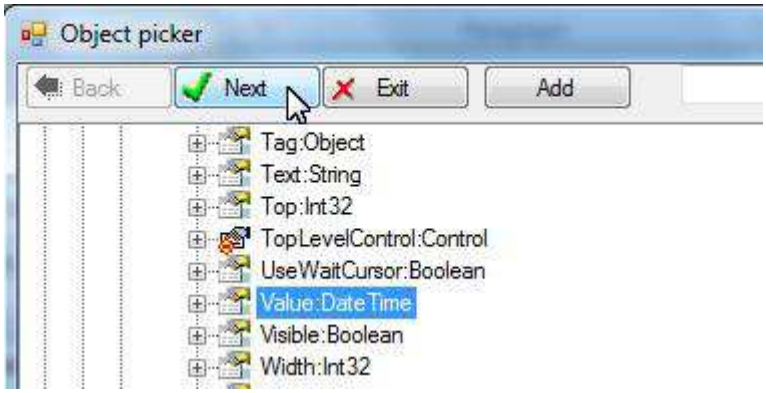
Select "Property" for @birthDateUpLimit to provide value for it:



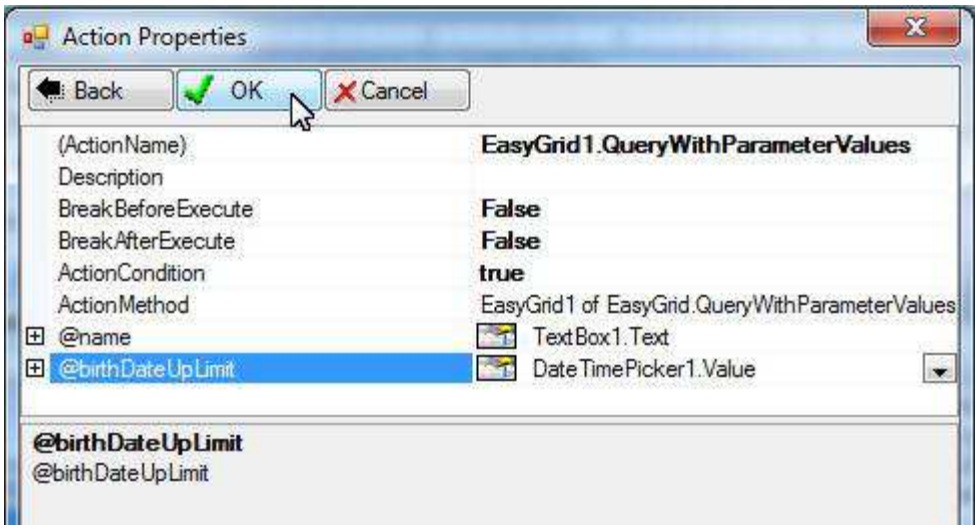
We want to use the Value property of the DateTimePicker to provide the value. Expand the "Properties inherited" node of the DateTimePicker:



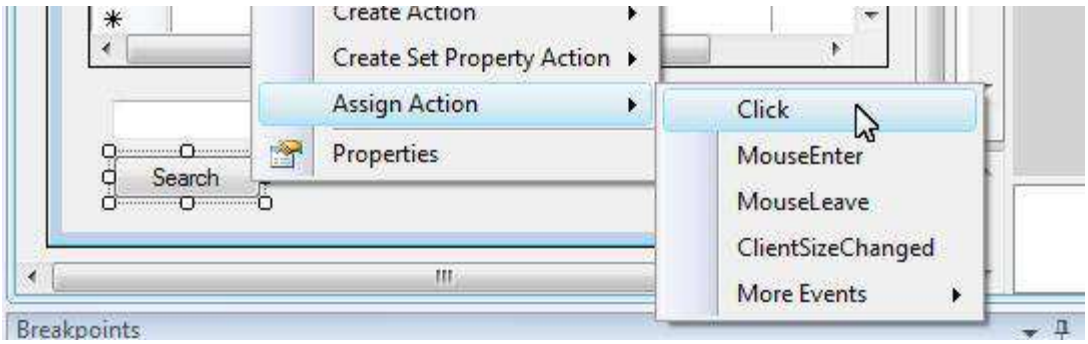
Select Value property. Click Next:

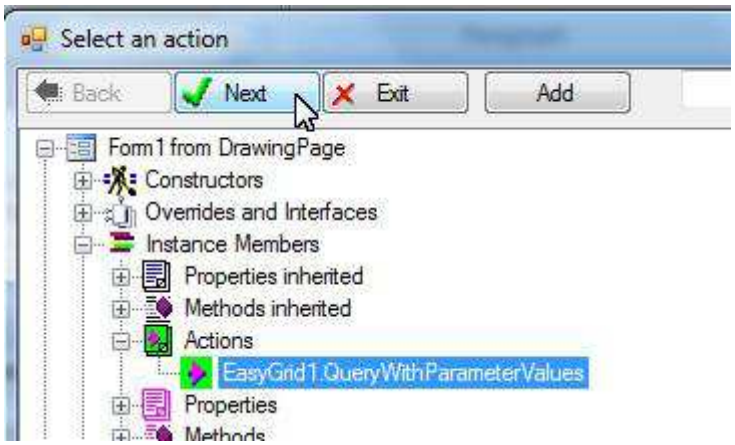


Click OK to finish creating this action:



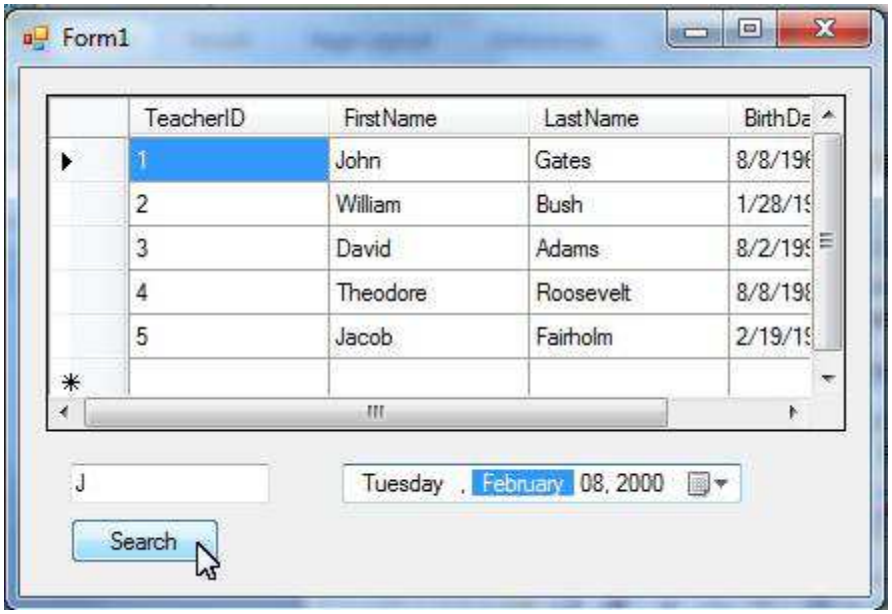
Assign this action to the Click event of the button:



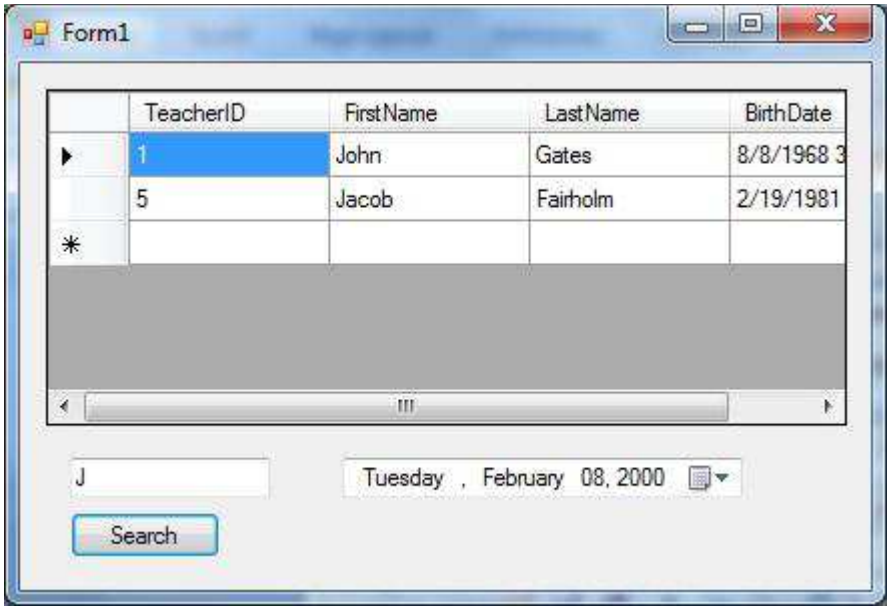


We may test the application now.

Specify search parameters in the text box and the DateTimePicker. Click Search buttn:



The search results appear:



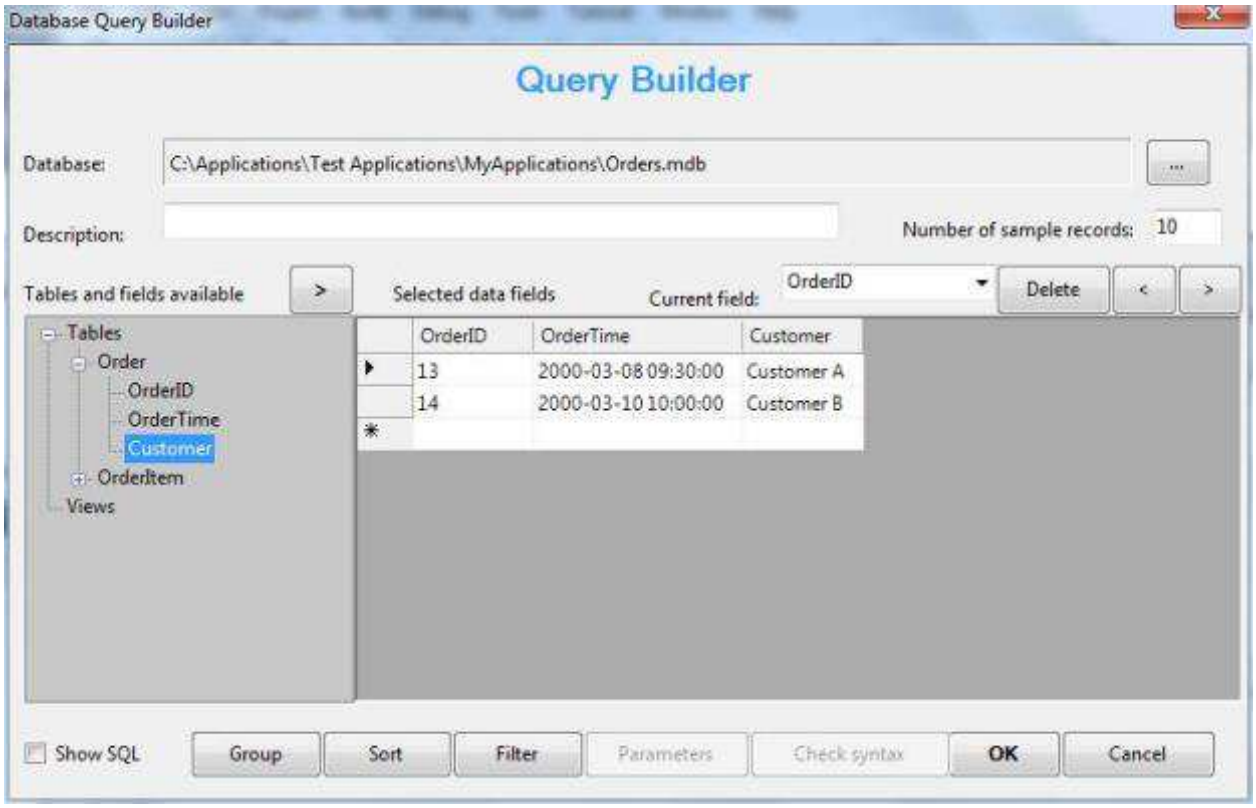
7 Work with Master-Detail Data

In a master-detail (one-to-many) data relationship, we may use an EasyDataSet or an EasyGrid for showing/updating master records, and use an EasyGridDetail for showing/updating detail record. In this sample we use an EasyGrid to show/update master records.

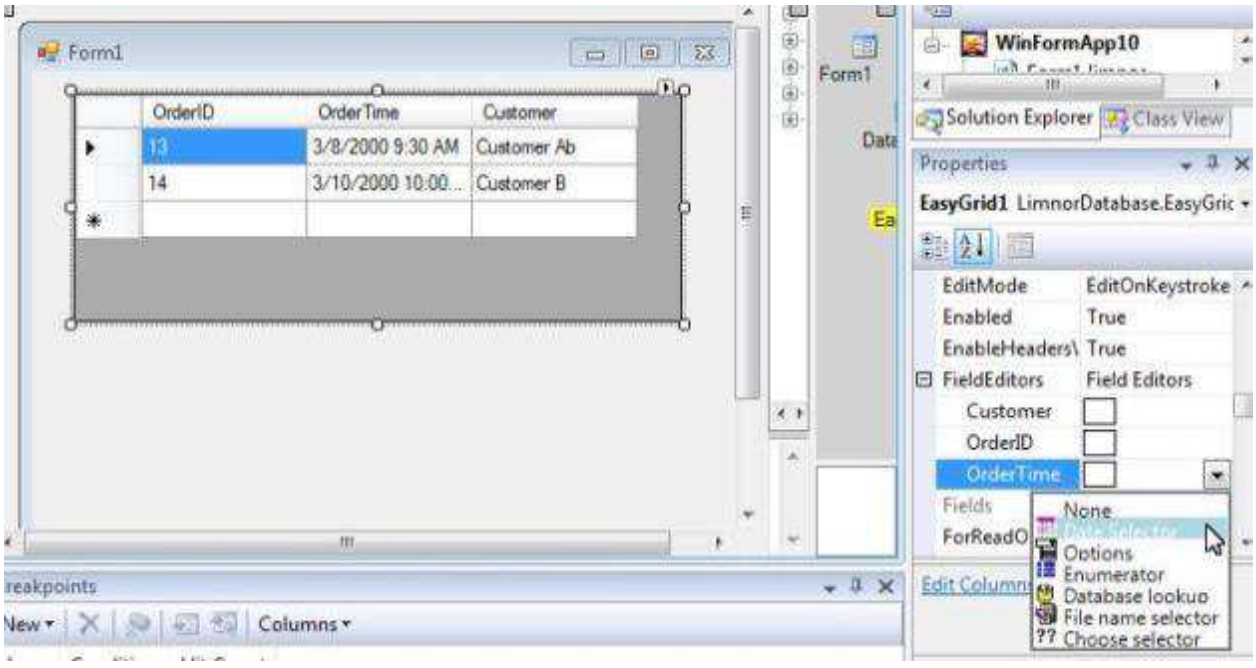
7.1 Get Master Data using EasyGrid

Create an EasyGrid. Set its DatabaseConnection property to connect to a database. Set its SQL property to make a query to get master data.

Select fields from Order table for the master records. Order table and OrderItem table form a master-detail (one-to-many) relationship. One Order record may have many OrderItem records linked by OrderID value.

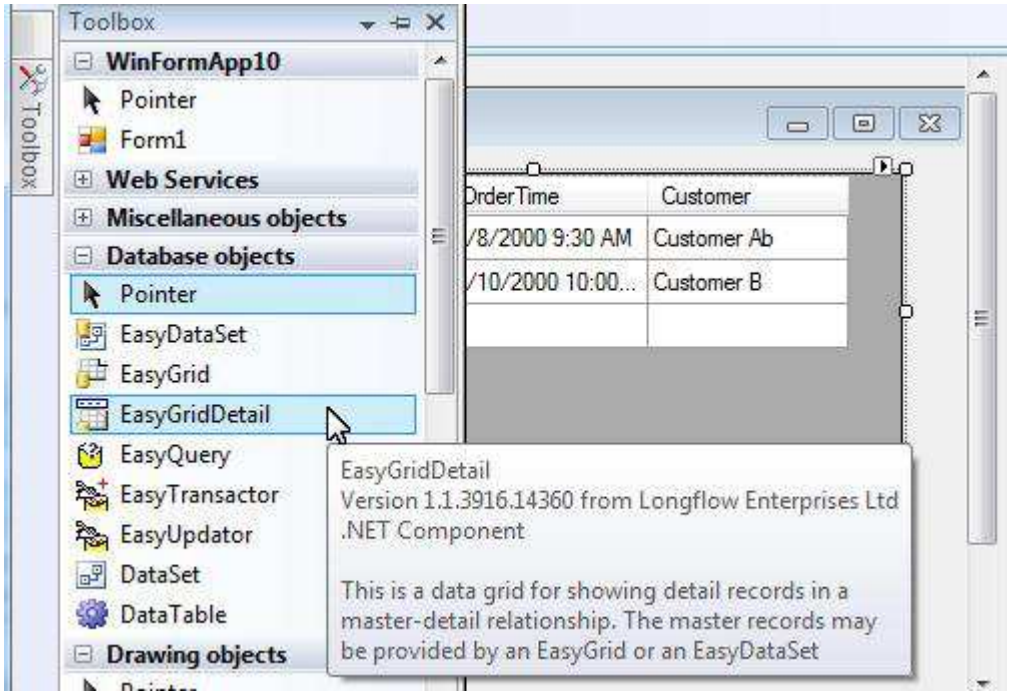


To make inputting OrderTime easier, choose Date Selector for it:

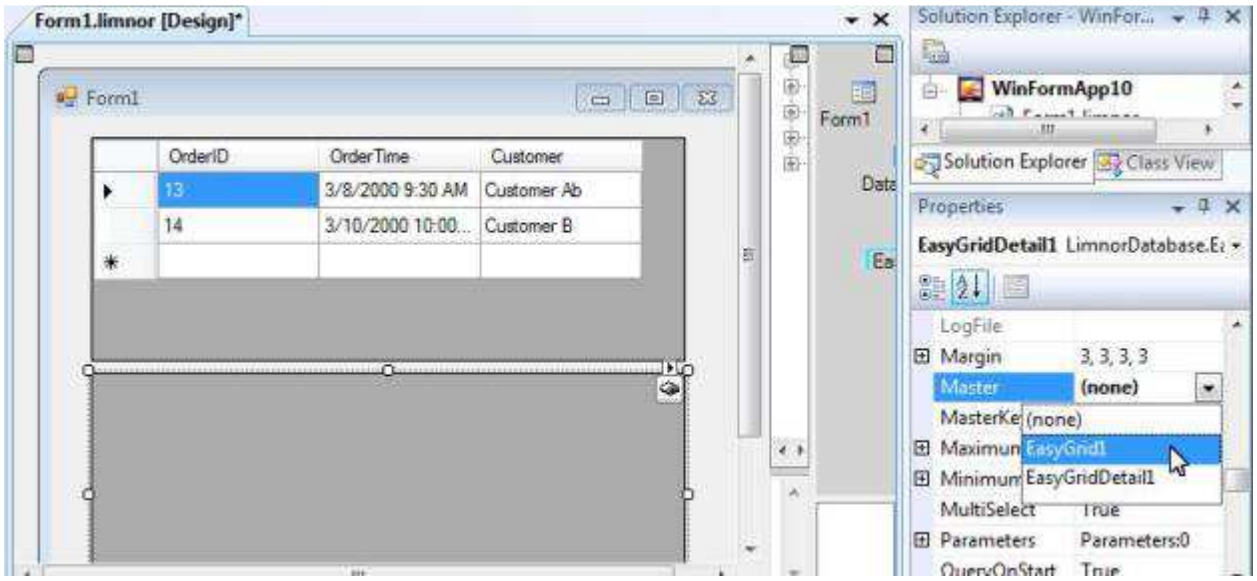


7.2 Get Detail Data

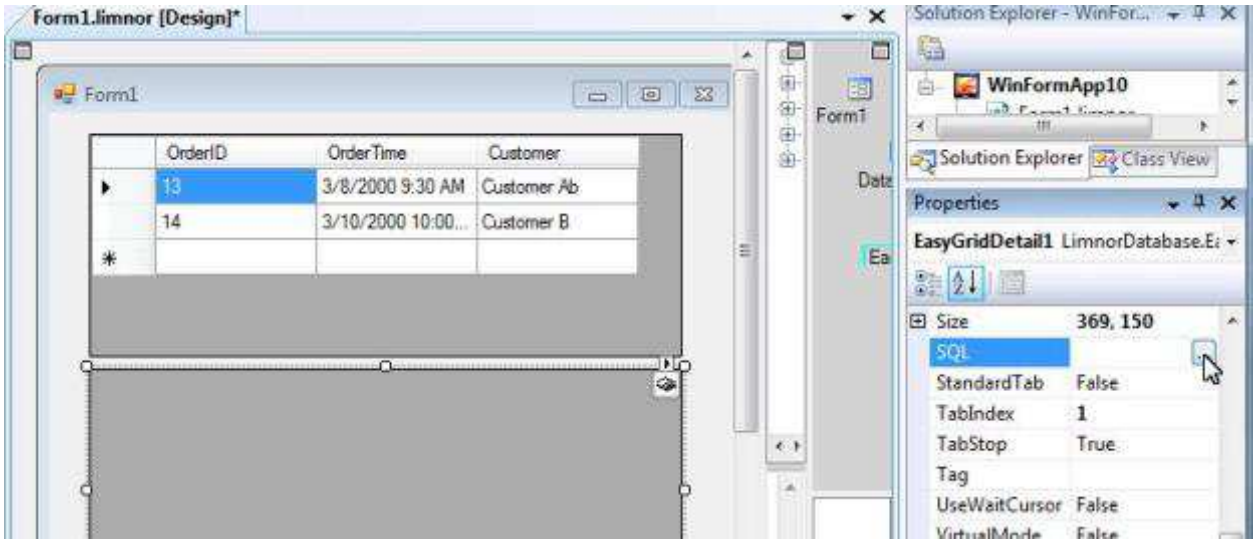
Use an EasyGridDetail to show/update detail records in a master-detail relationship.



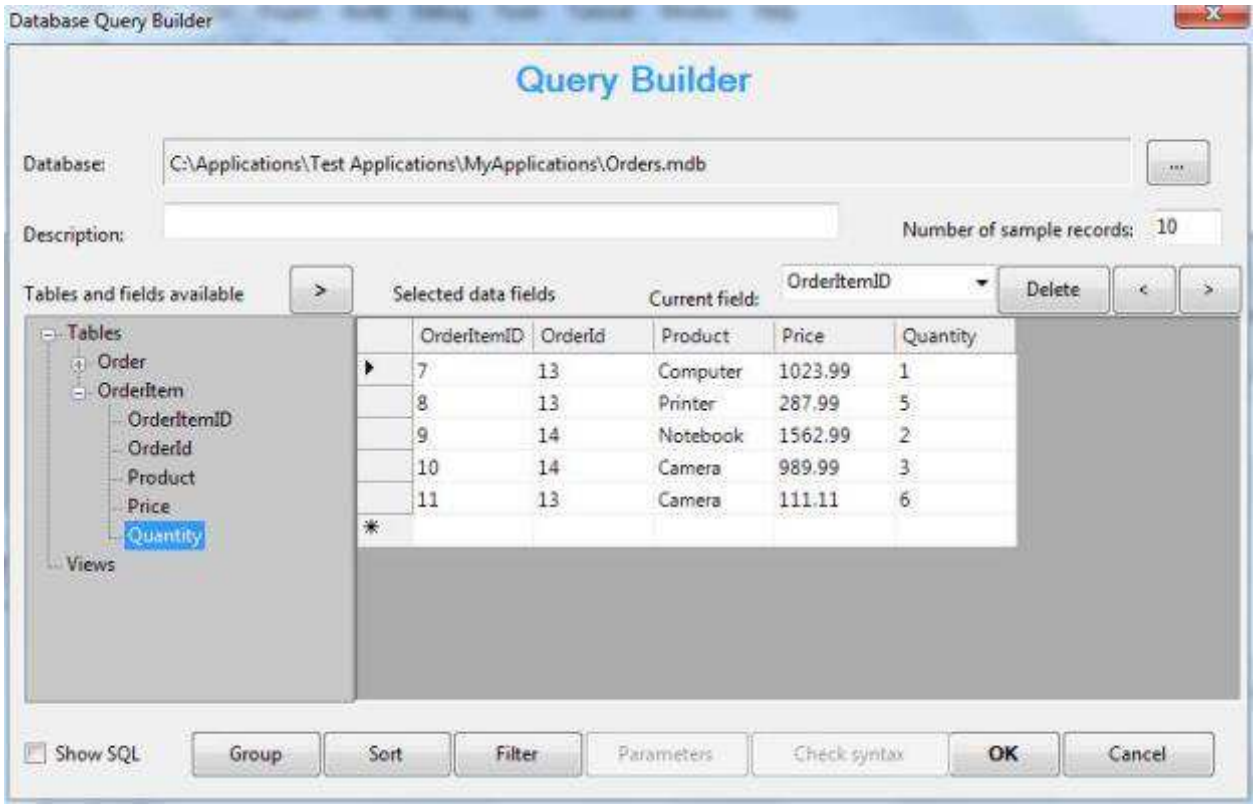
Set Master property to point to EasyGrid1 which contains the master records.



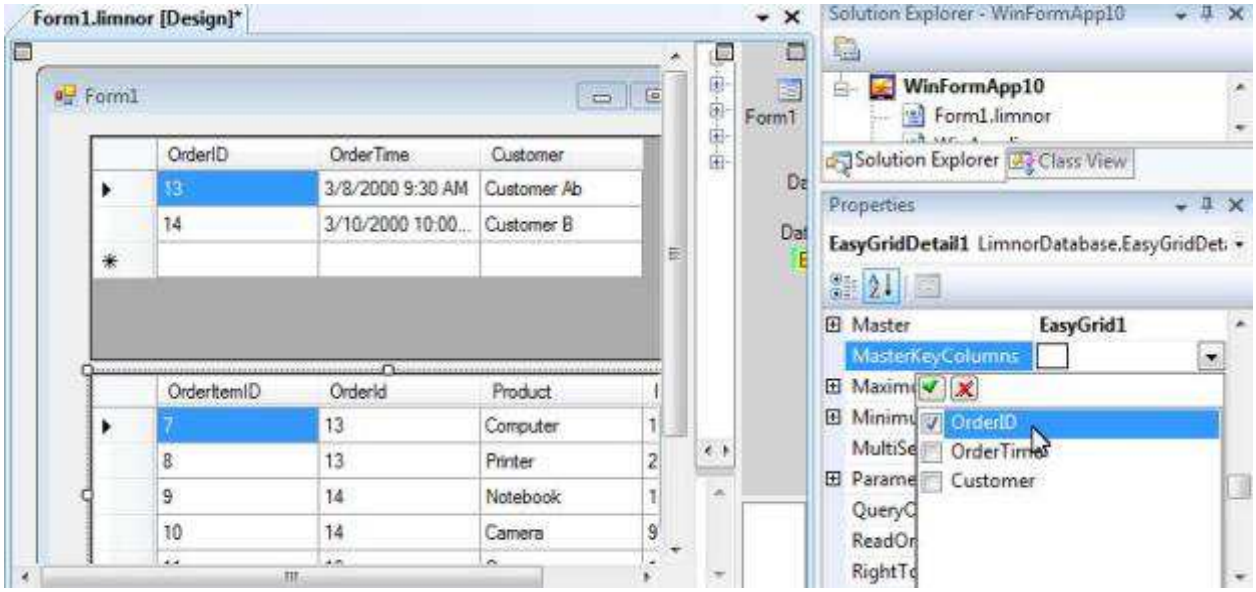
Set SQL property to get detail:



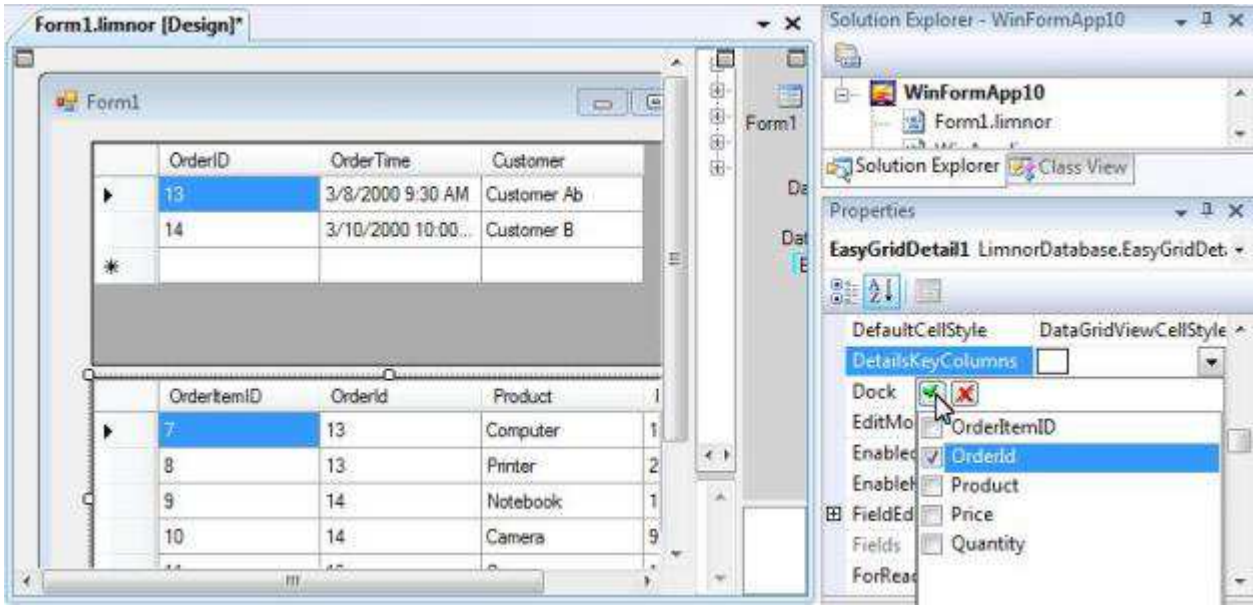
Select fields from OrderItem table for getting detail records. Order table and OrderItem table form a master-detail relationship.



The master-detail relationship is built by OrderID field in both the Order table and the OrderItem table. Check OrderID field in MasterKeyColumns to indicate that OrderID in the master table is used for the master-detail relation.

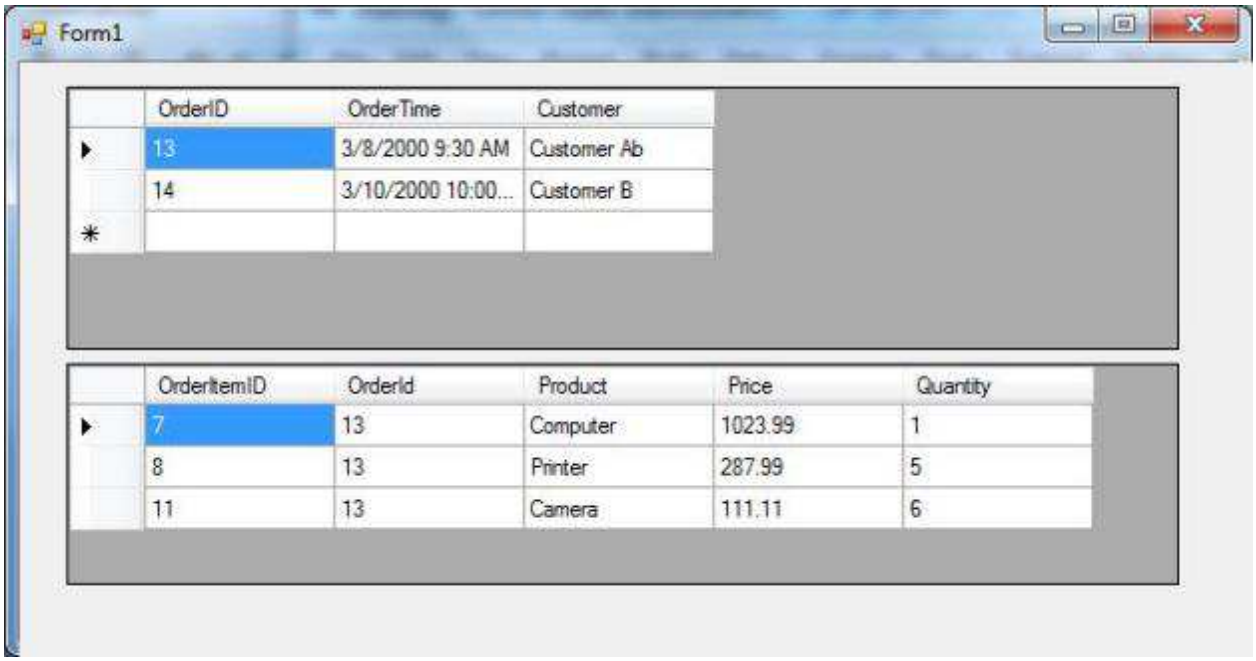


Check OrderId field in DetailsKeyColumns to indicate that OrderId field in OrderItem table is used for building the master-detail relationship.

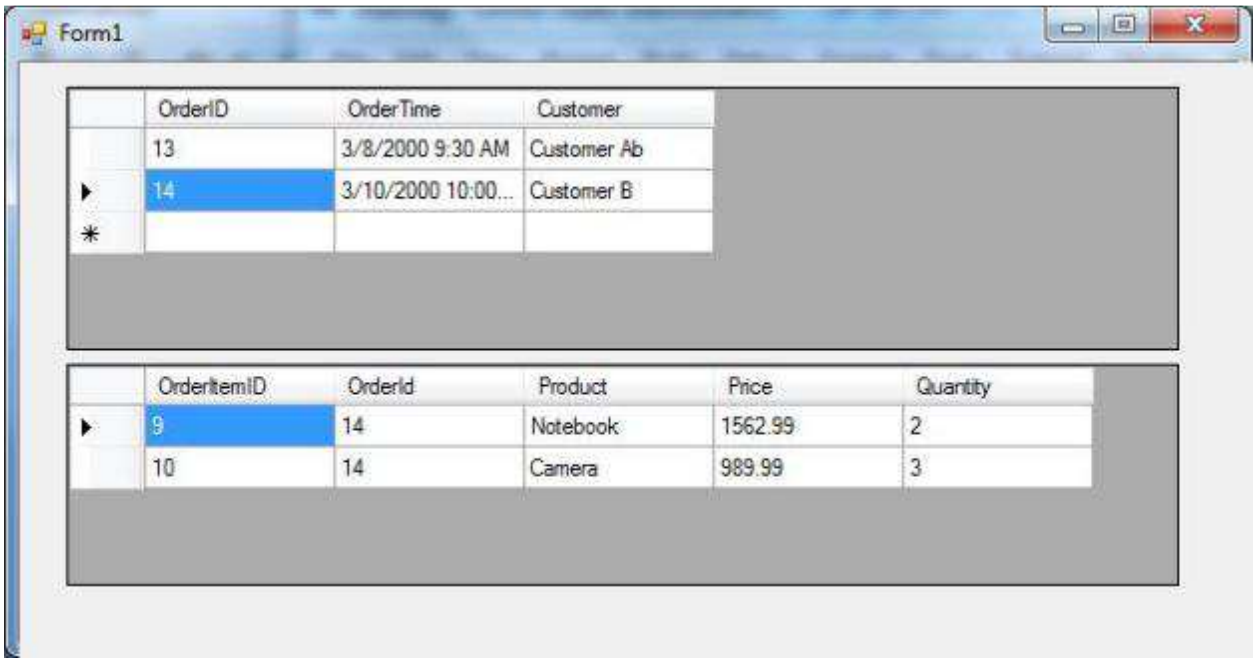


We may test the application now.

Note that when the first Order record is selected, OrderID =13, all detail OrderItem records belong to the Order (only OrderItem records with OrderId=13 are displayed):

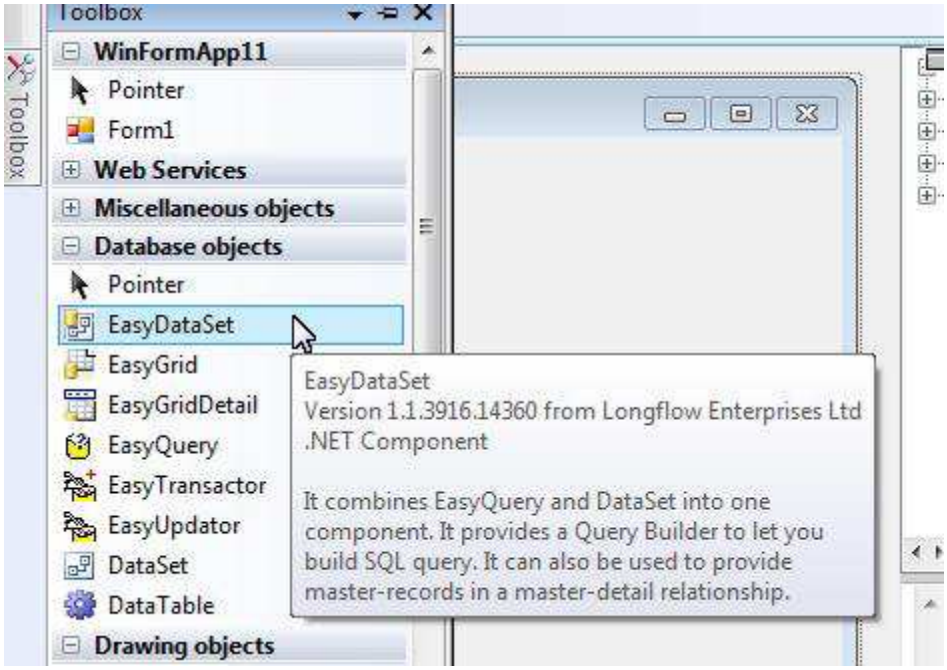


When the second Order record is selected, OrderID =14, all detail OrderItem records belong to the Order (only OrderItem records with OrderId=14 are displayed):



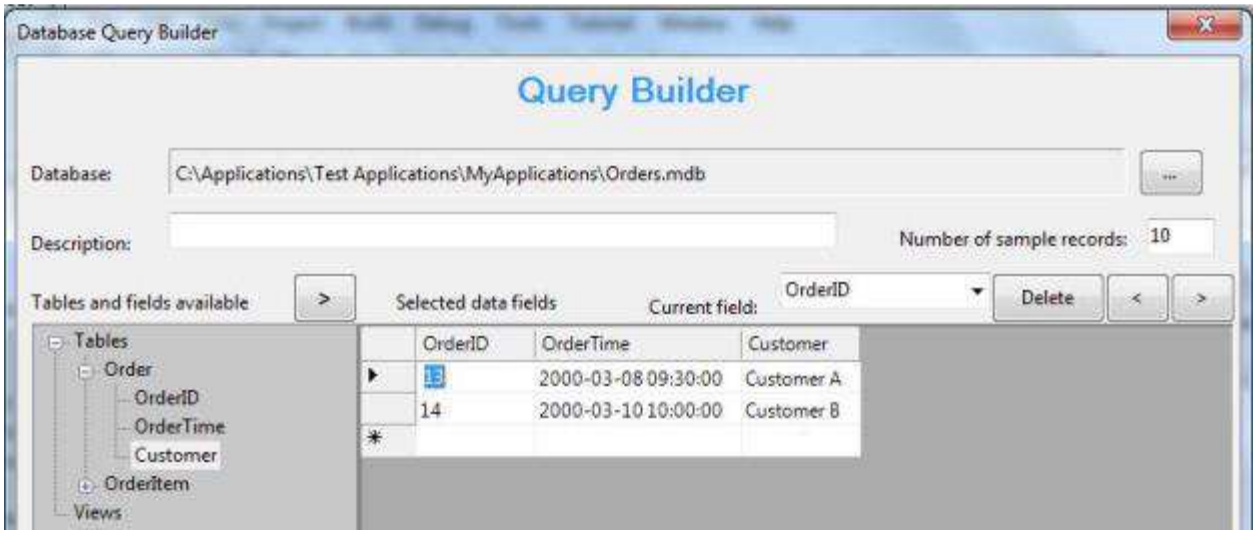
7.3 Get Master Data Using EasyDataSet

Create an EasyDataSet for getting master data.

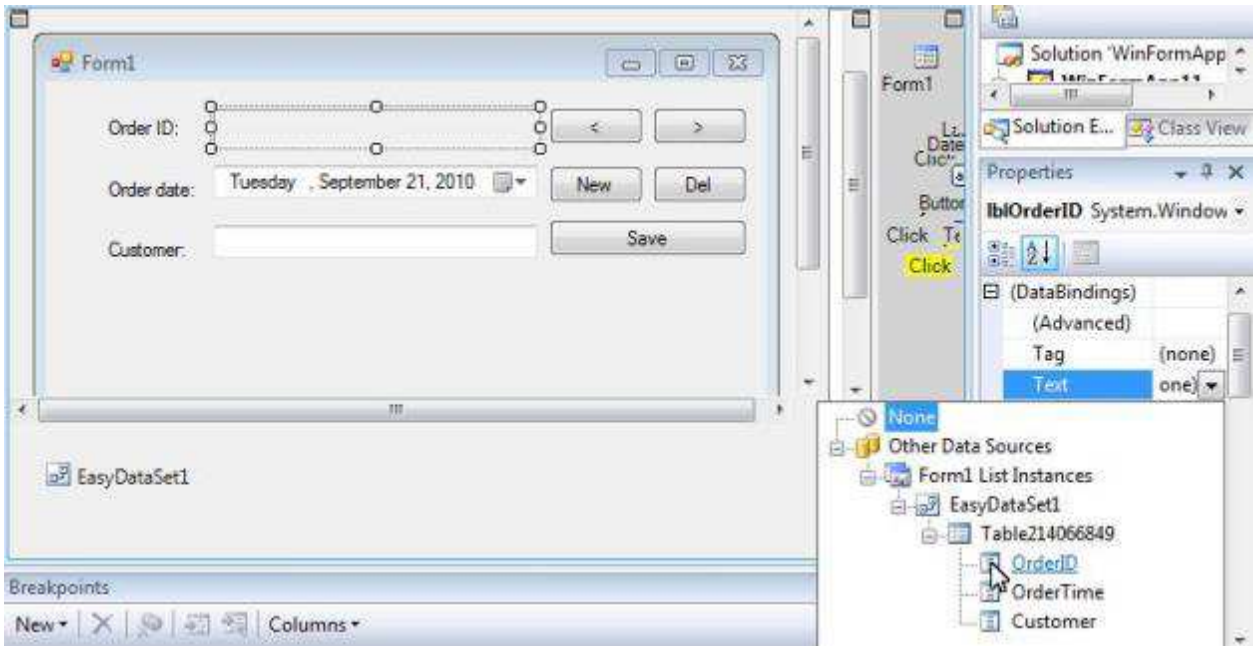


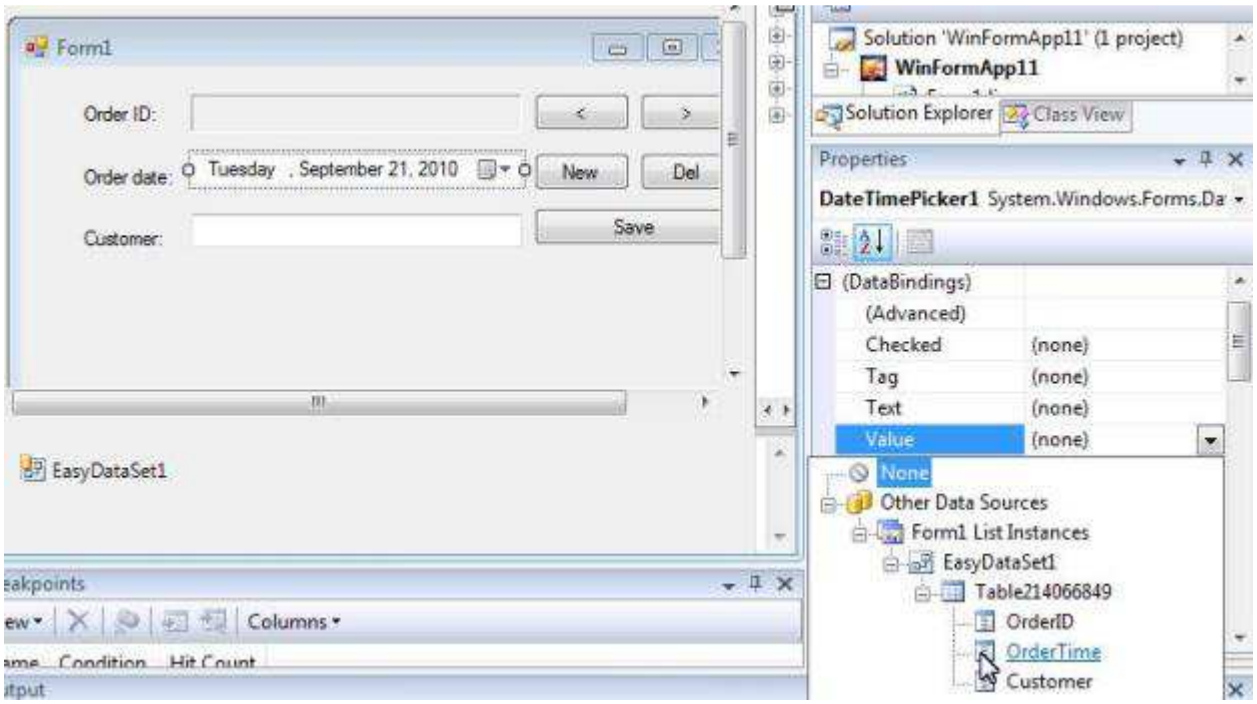
Set its DatabaseConnection property to connect to a database. Set its SQL property to make a query to get master data.

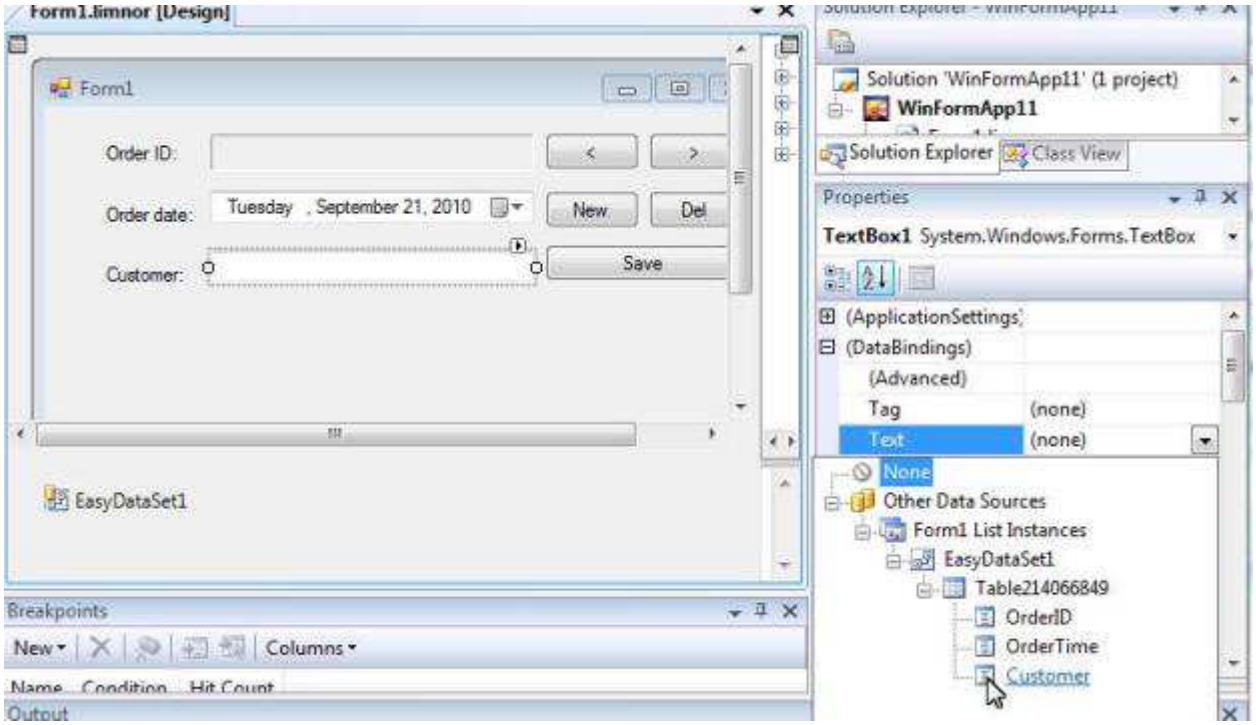
Select fields from Order table for the master records. Order table and OrderItem table form a master-detail (one-to-many) relationship. One Order record may have many OrderItem records linked by OrderID value.



Make data-bindings, build record navigation button, and data updating buttons. See Data-Binding section for details. Here we just show the data bindings.

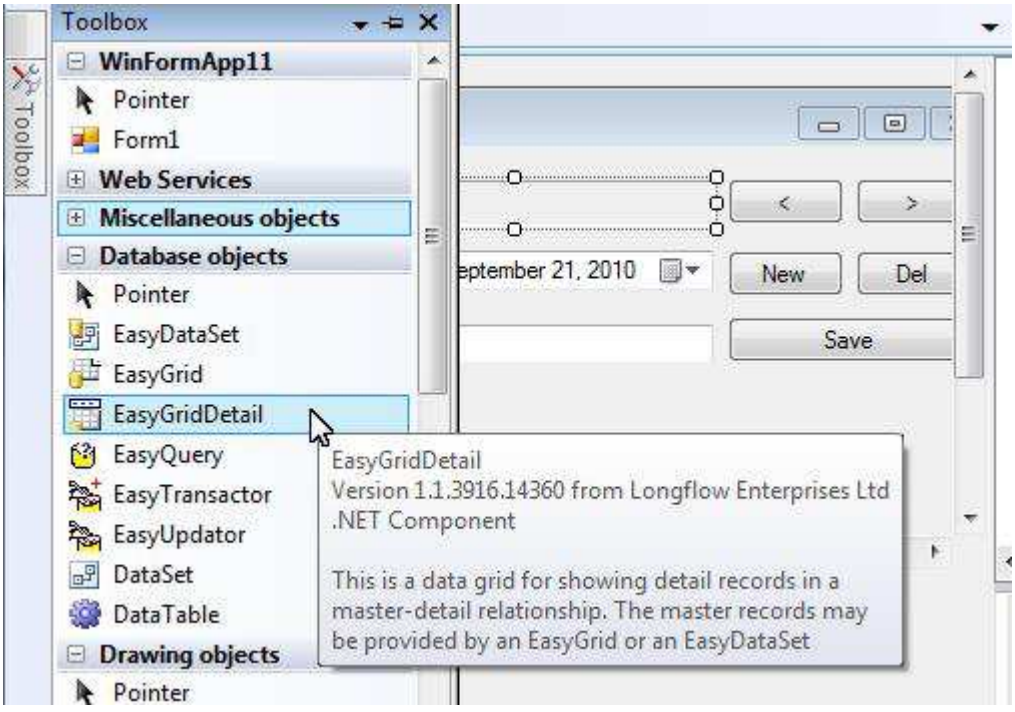




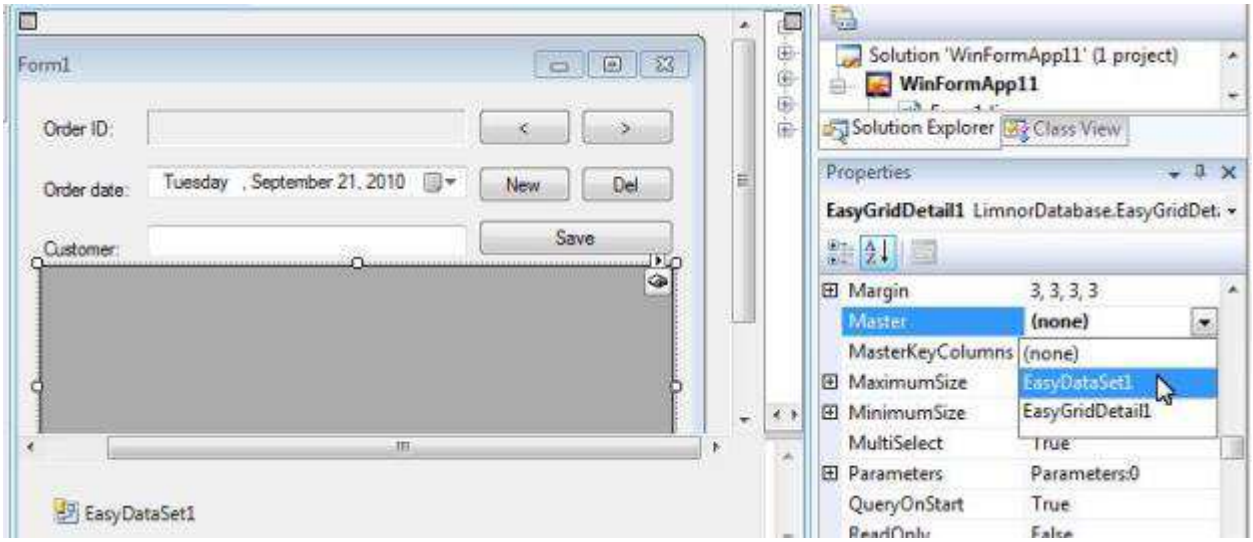


7.4 Use EasyGridDetail with EasyGrid

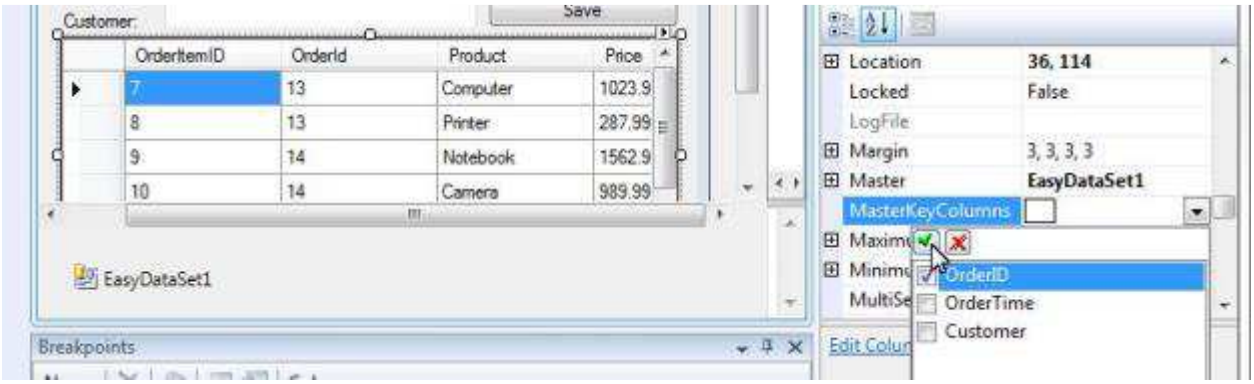
Drop an EasyGridDetail to the form

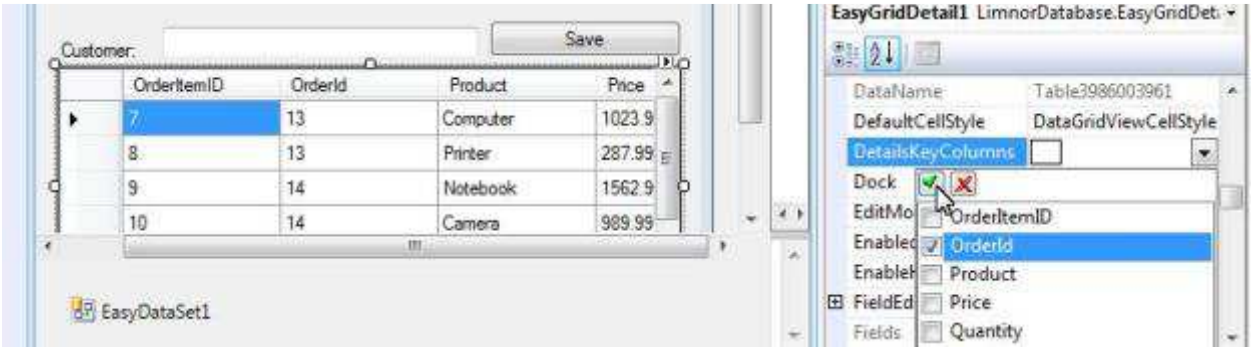


Set its Master property to EasyDataSet1:



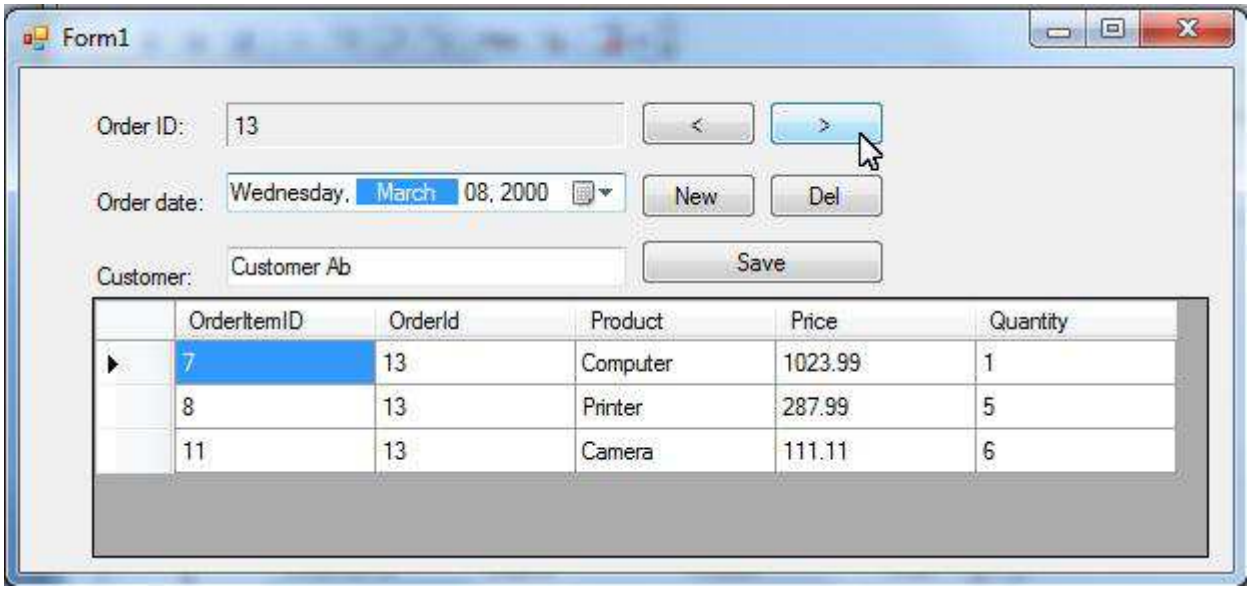
Set its SQL property to get OrderItem records. Set its MasterKeyColumns and DetailKeyColumns properties we did before



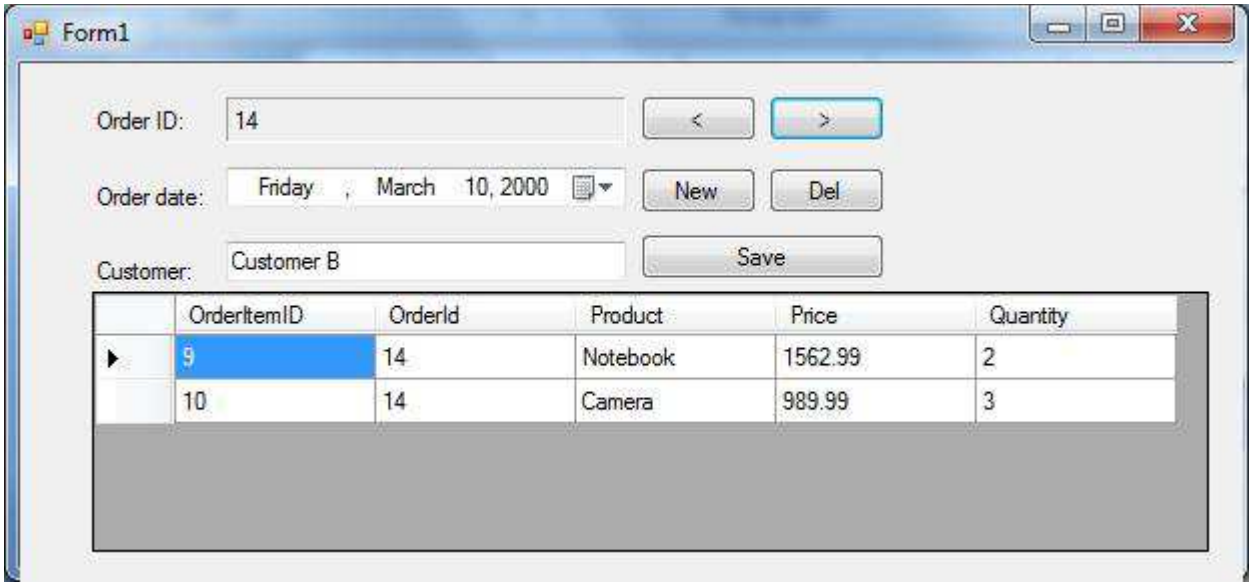


We may test it now:

The first Order appears. The OrderItem records are all for the first order.



Move to the next Order. The OrderItem records are all for the next Order:

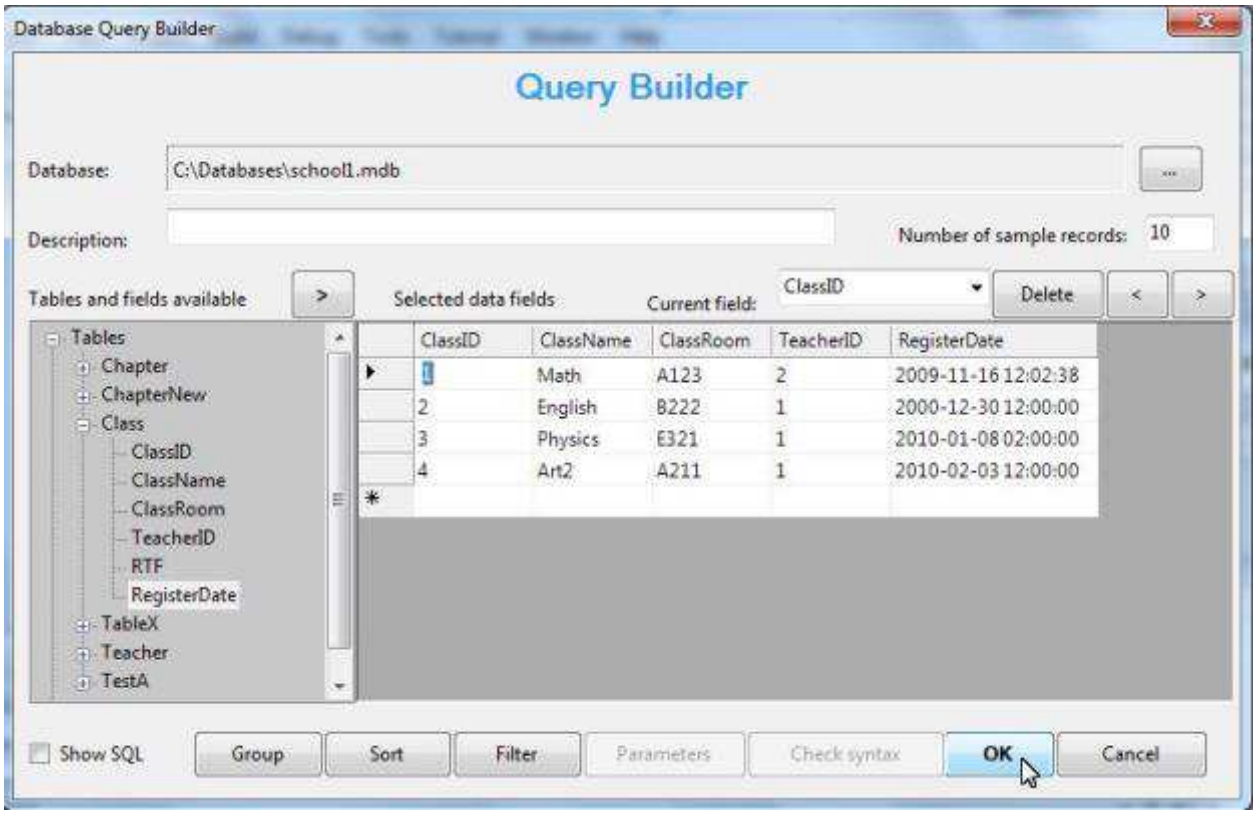


8 Data Entry Using EasyGrid

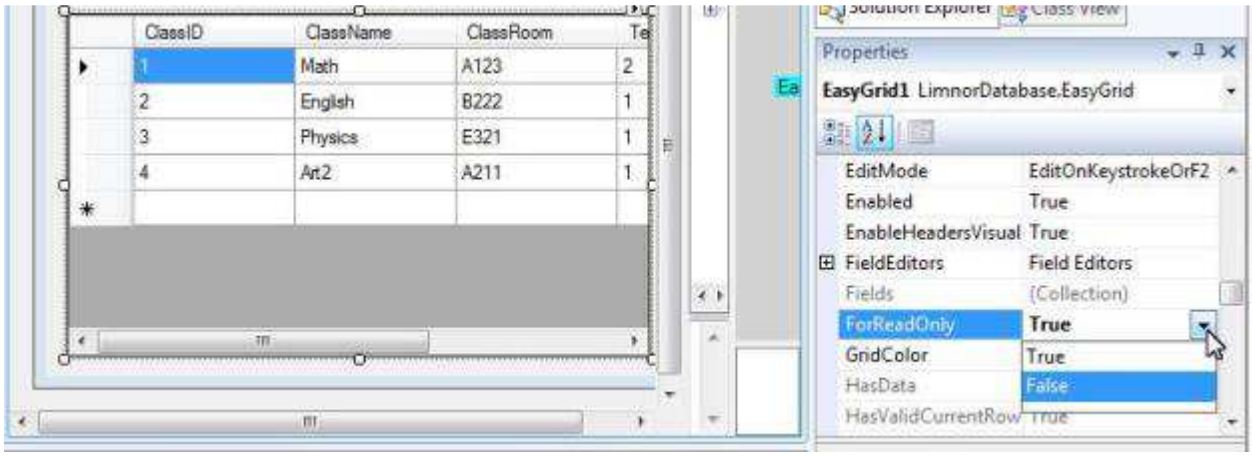
8.1 Make EasyGrid Writeable

EasyGrid can be setup as a data entry tool for entering data into database easily.

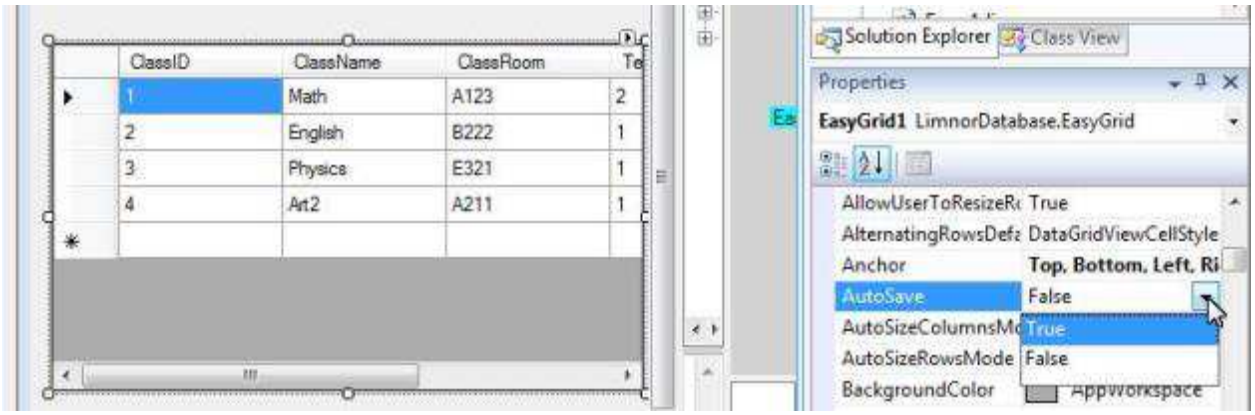
Create an EasyGrid on a form. Set its DatabaseConnection property to point to a database. Set its SQL property to retrieve data from a table. Because we want to modify data of the table, we must include fields for a unique index of the table. In this sample, it is the ClassID field:



Set its ForReadOnly property to False:

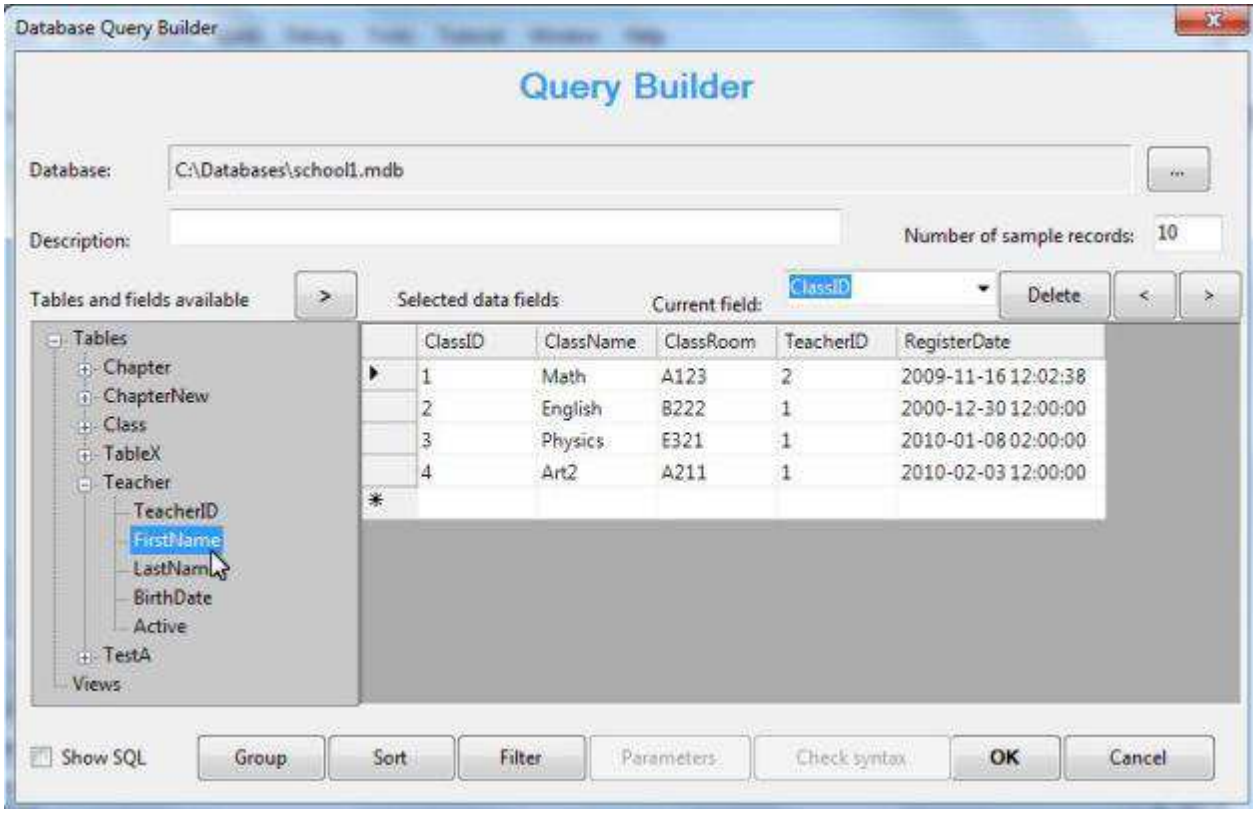


Set its AutoSave property according to your business requirement. If this property is True then the changes will be automatically saved back to the database then the form is closed. If this property is False then the changes will be saved to the database only an UpdateData action or by clicking the Save button on up-right corner of the EasyGrid.

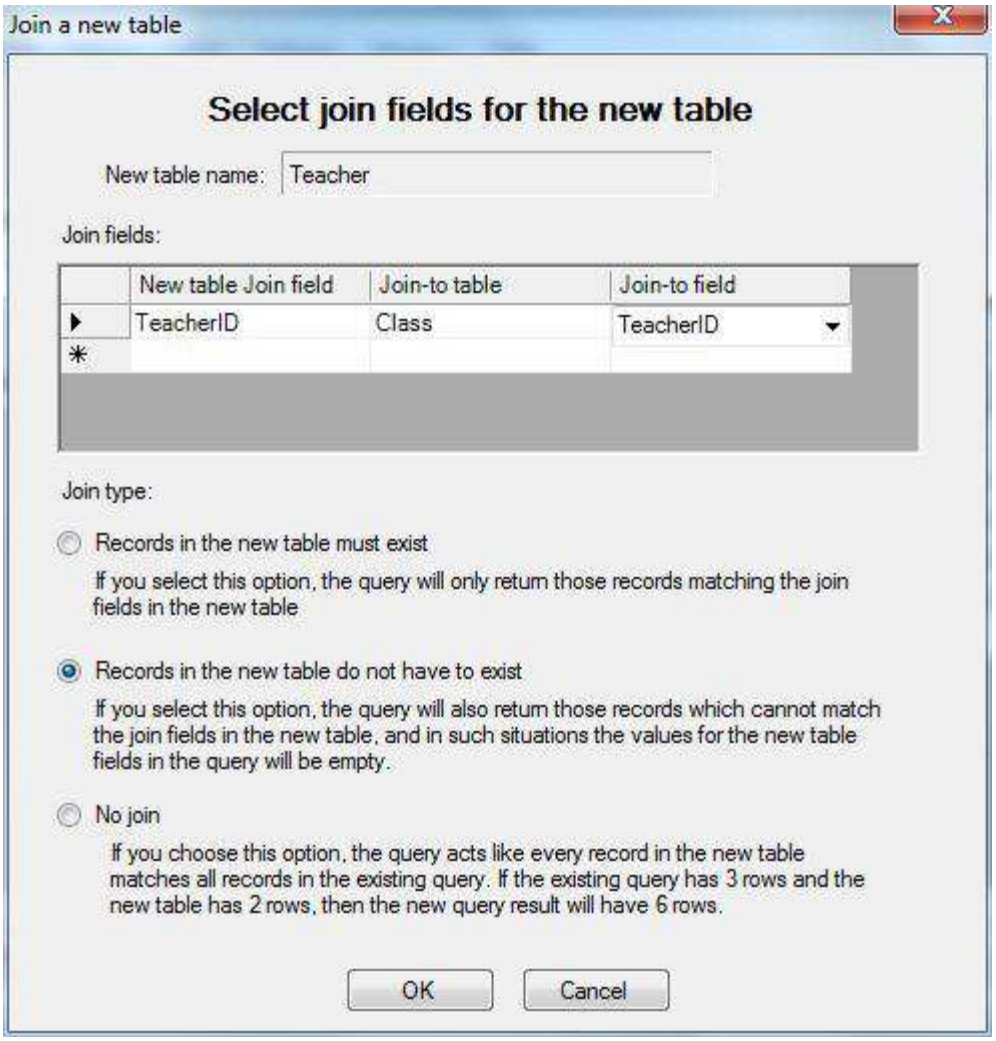


8.2 Use more descriptive data in query

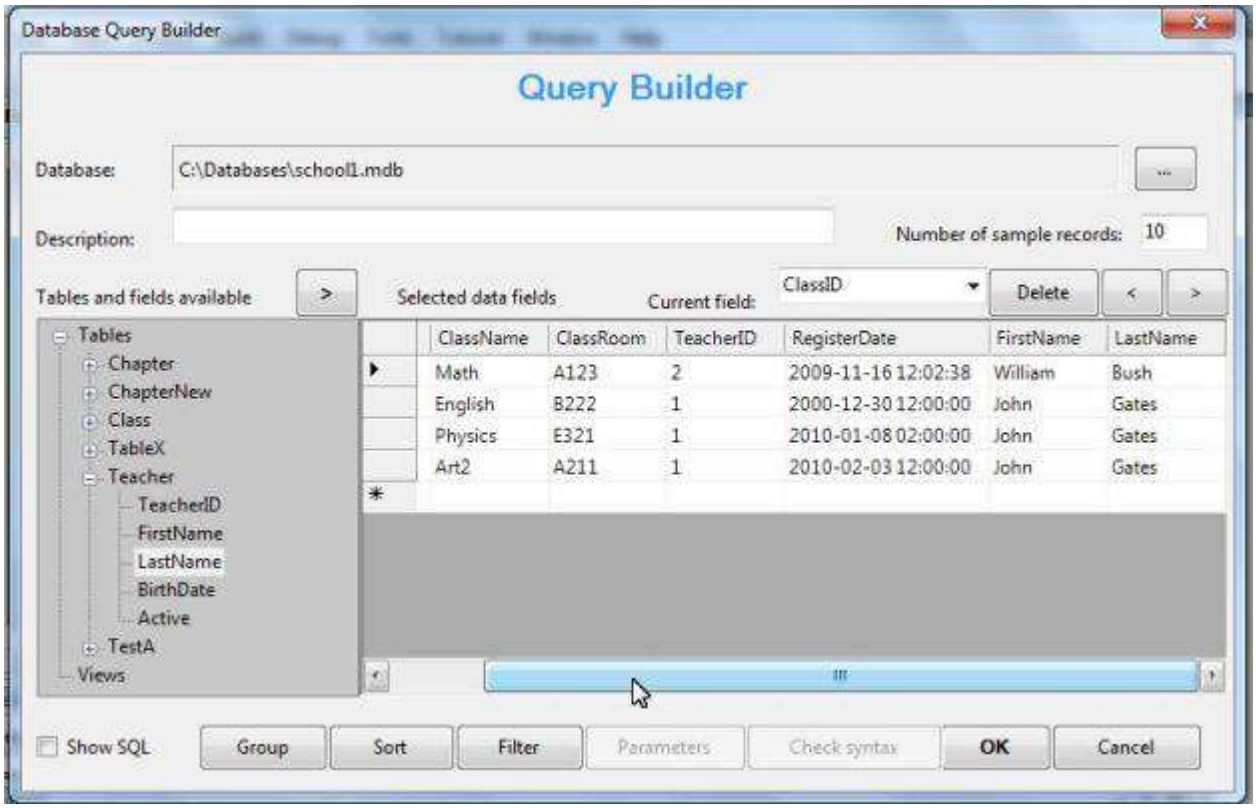
Because TeacherID is not easy to remember, we may add FirstName and LastName to the query. Set SQL property again. Double-click FirstName field.



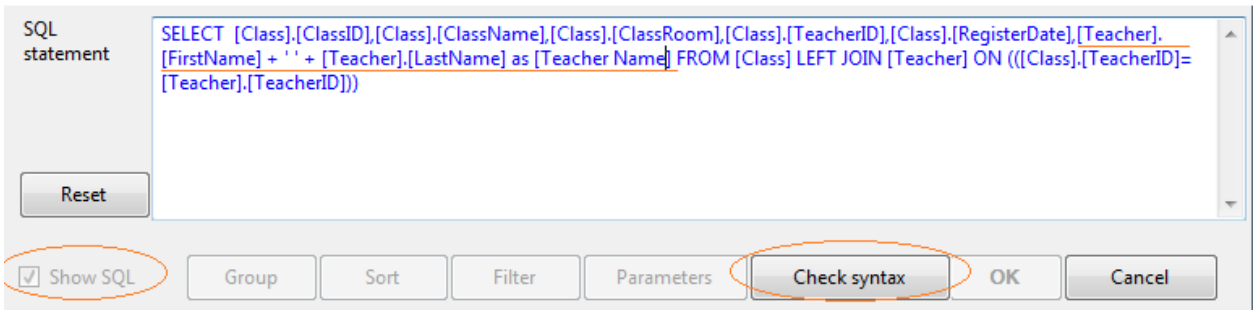
Because FirstName is from a table not used in the current query, it asks for the join specifications. We join Teacher table with the Class table via TeacherID. Note that the second join type is used because a Class record may not have a TeacherID value entered. The TeacherID value can be empty for a Class record. In that case we still want the Class record to appear in the query results. Actually TeacherID is one of fields to be entered via this EasyGrid.



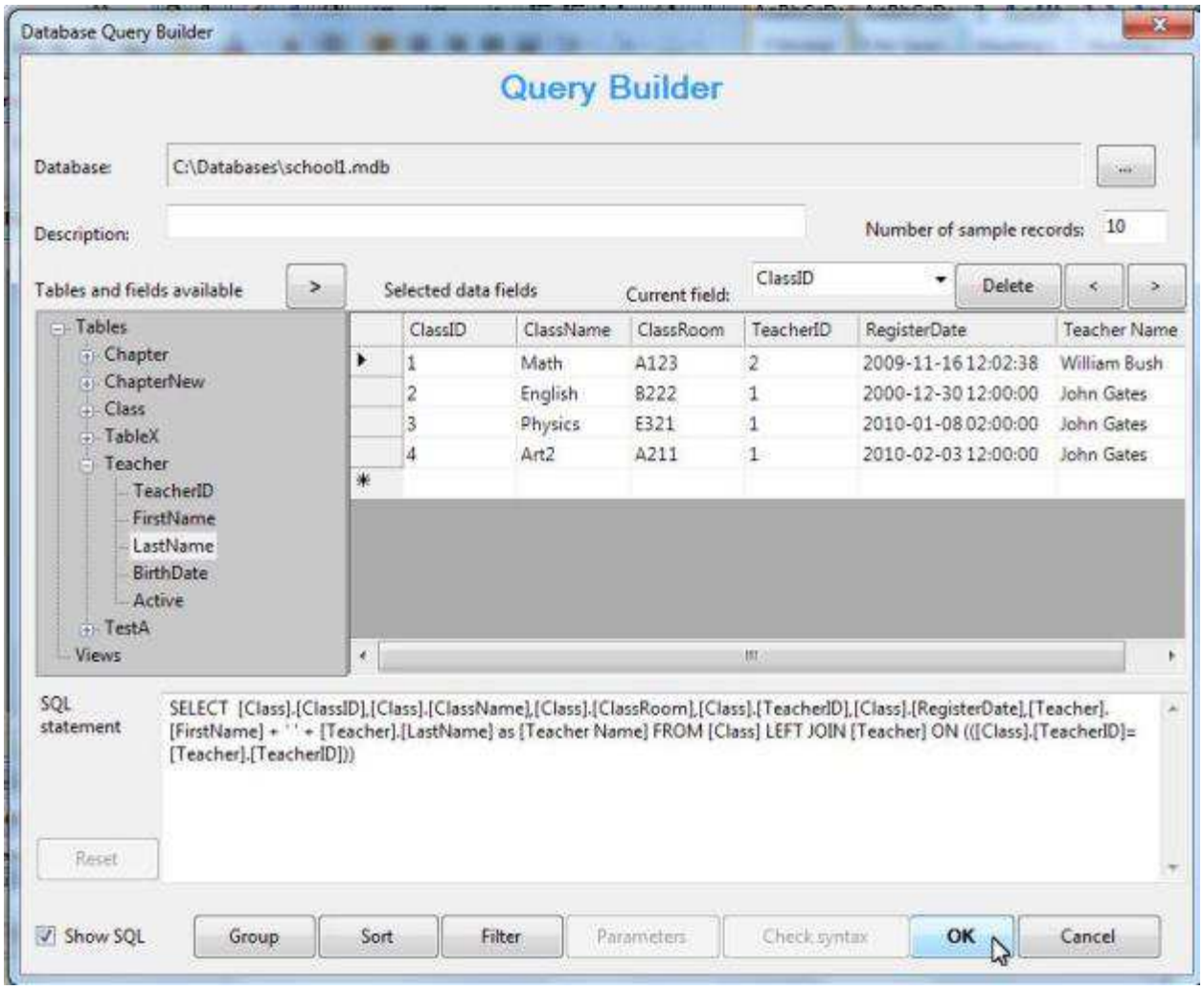
The FirstName field appears in the query. To make the teacher name more identifiable, also add LastName field to the query:



Check "Show SQL" and combine FirstName and LastName into one Teacher Name field:

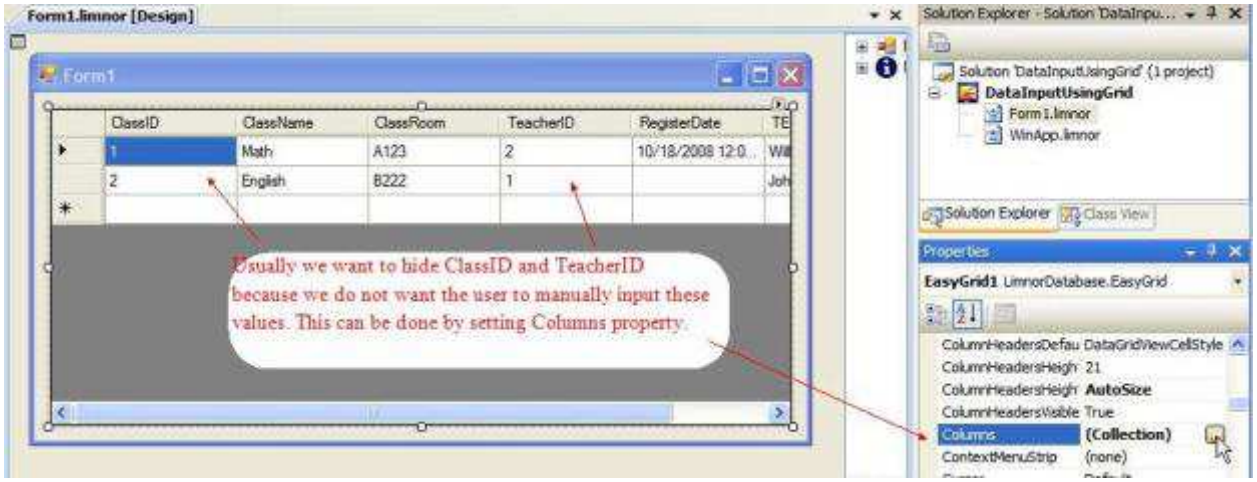


Click "Check syntax" button. If we modified the SQL statement correctly then a Teacher Name field replaces the FirstName and LastName fields in the query:

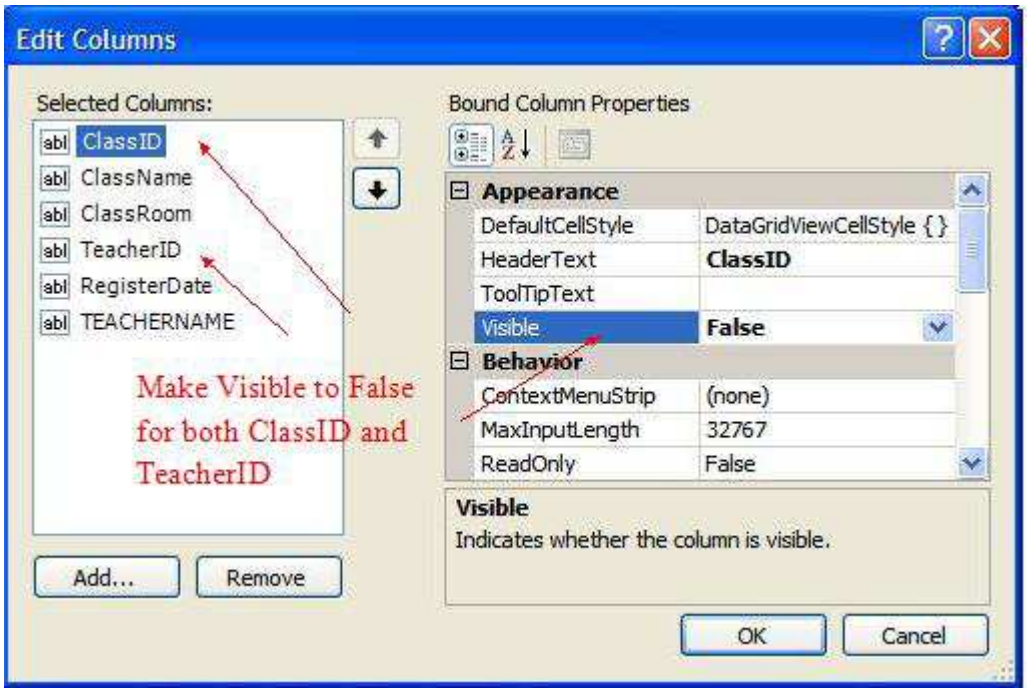


8.3 Hide ID fields

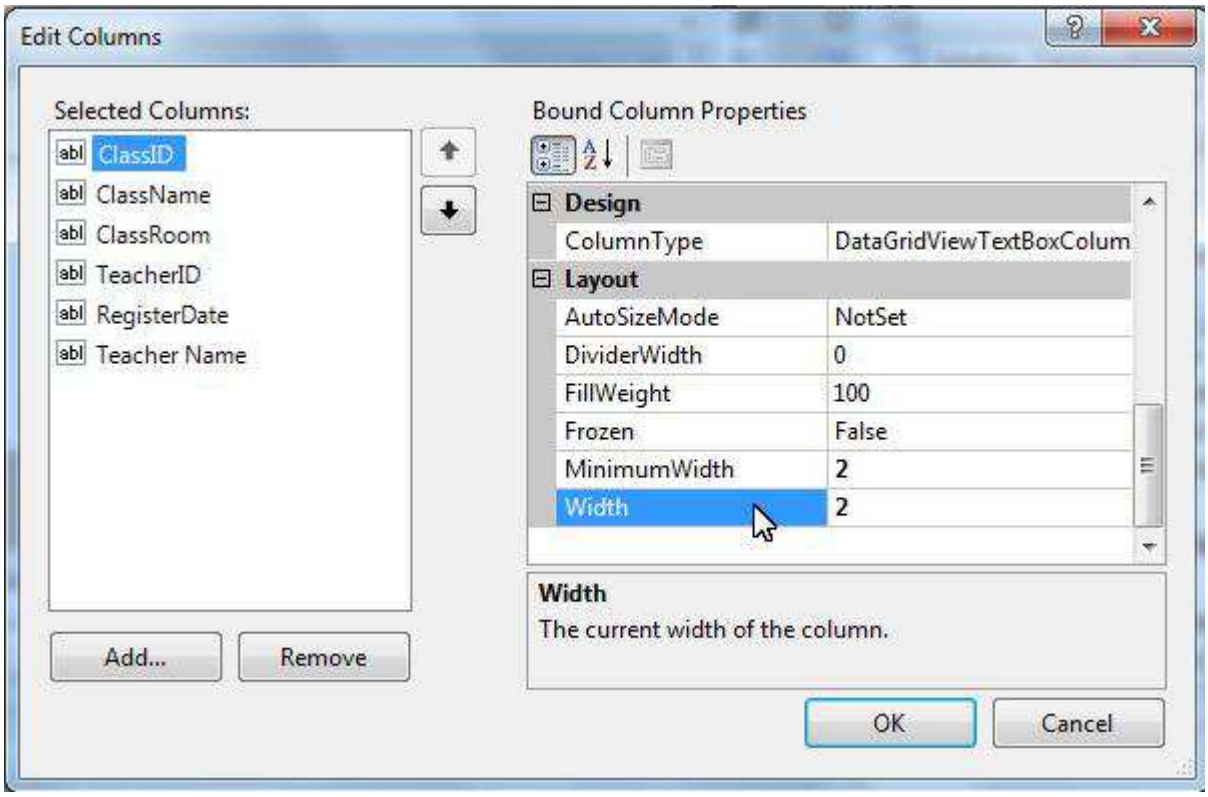
Usually we want to hide ClassID and TeacherID because we do not want the user to manually input these values. This can be done by setting Columns property.



Note that there is a bug in Microsoft .Net library. It makes Visible property for an IDENTITY column not work. The ClassID column is an IDENTITY column. See following URL for details:
http://connect.microsoft.com/VisualStudio/feedback/ViewFeedback.aspx?FeedbackID=91605

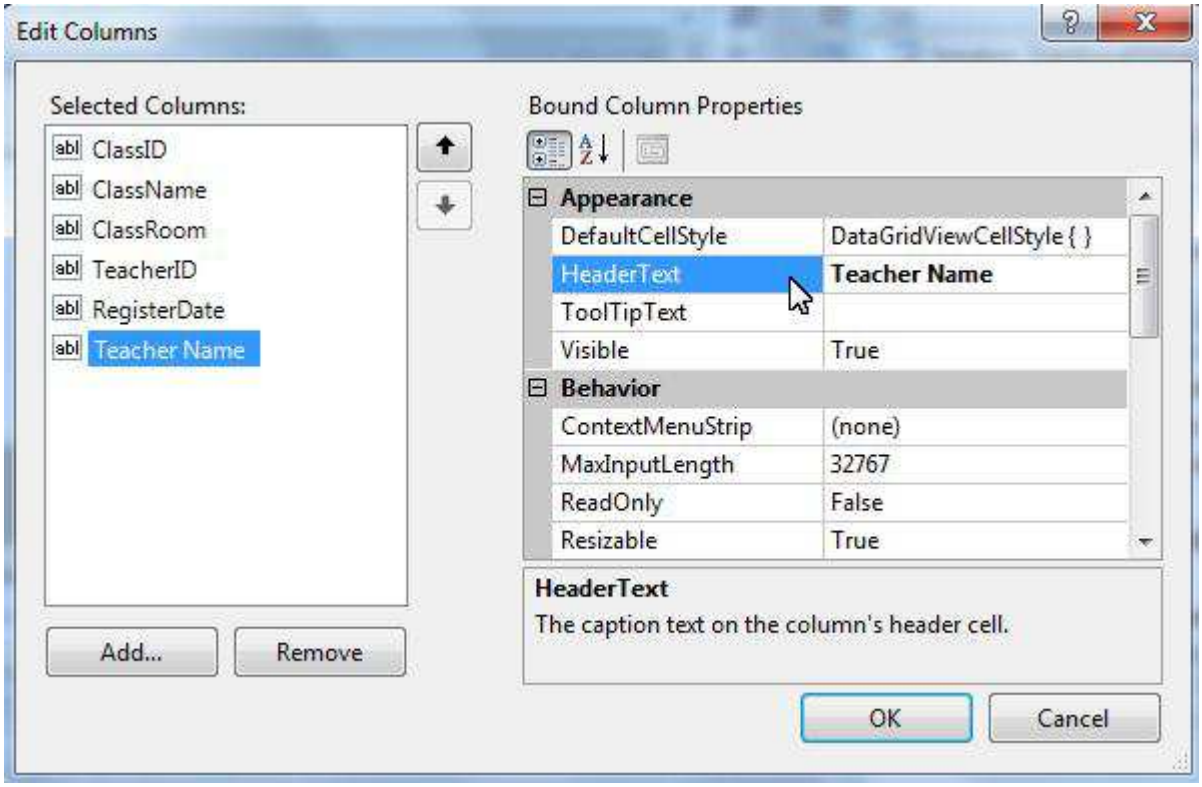


One workaround can be setting the column width to a small value:



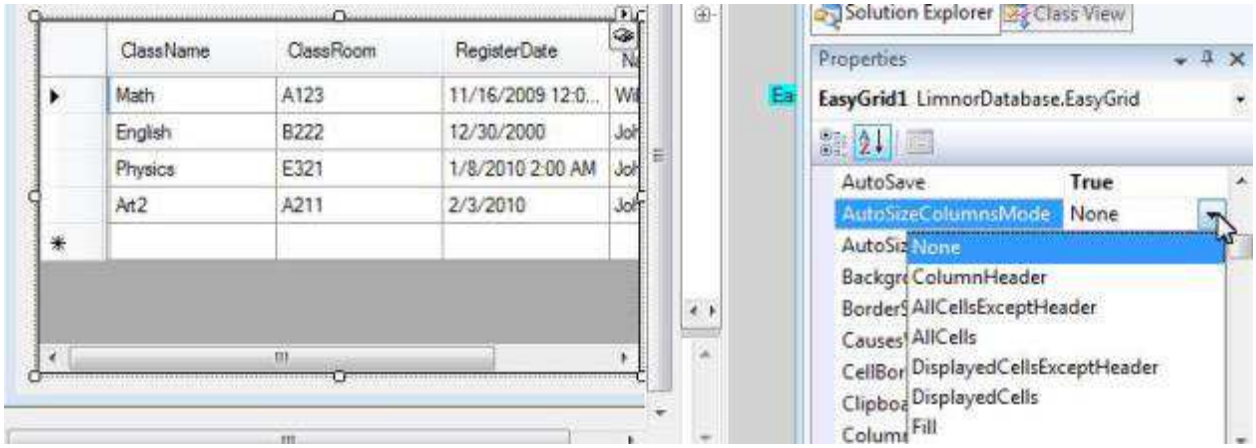
8.4 Adjust Data Display

Columns property also provides many more data display attributes.

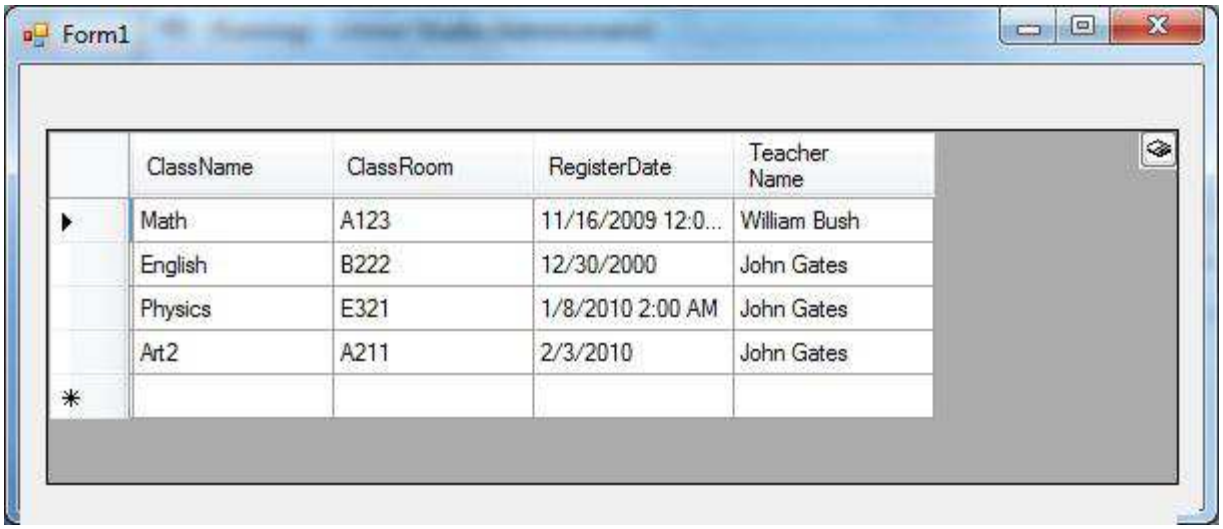


Many other properties affect the display of data. You may play with them to meet your requirements.

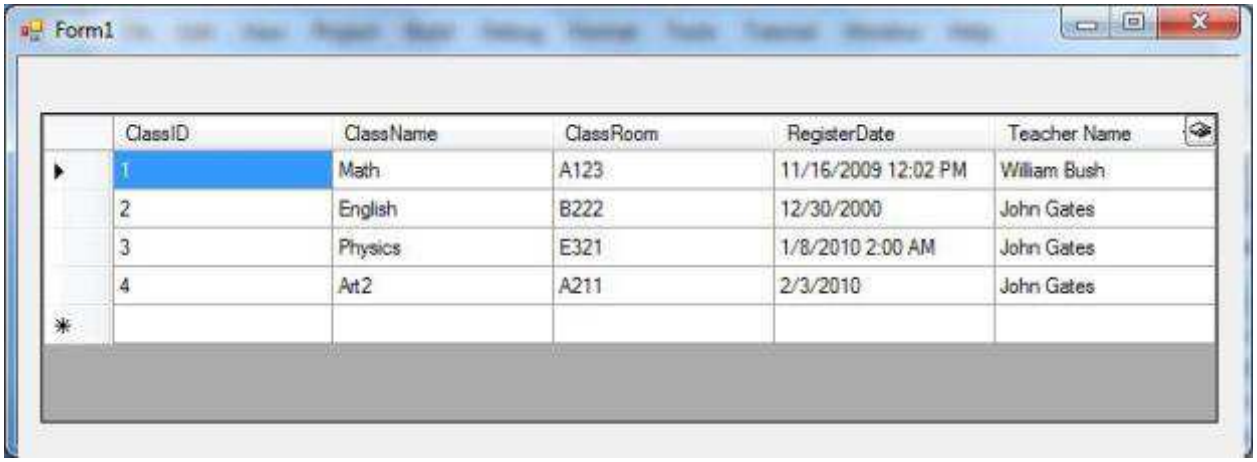
For example, AutoSizeColumnsMode controls the sizing of the columns:



None – nothing is automatically done:



Fill – Fill the whole width:



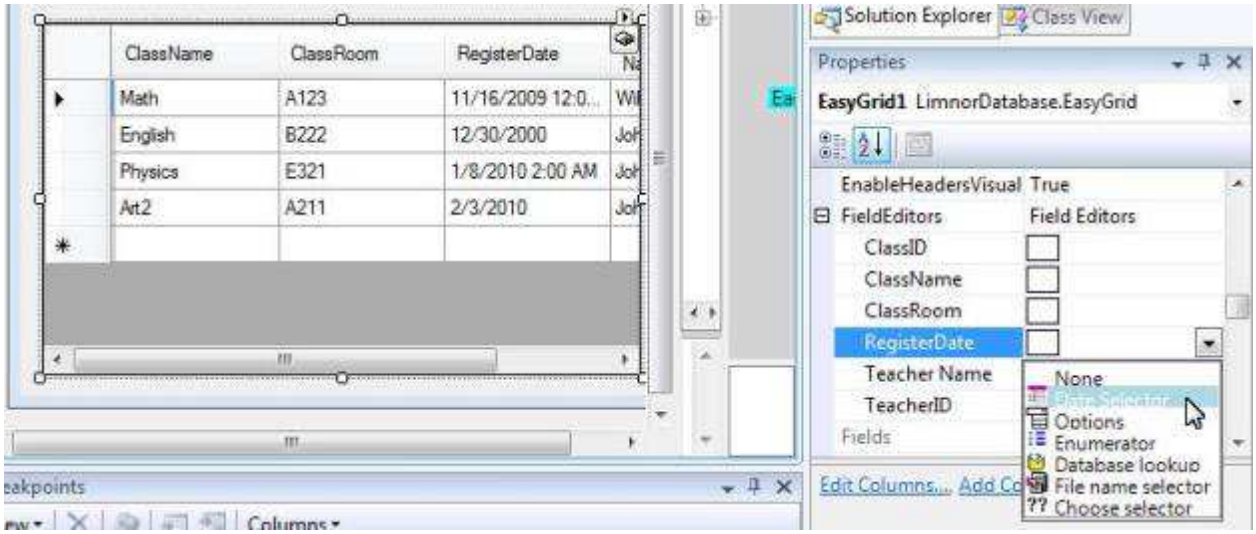
8.5 Use Field Editors

Field Editors make data entry easier and more accurate.

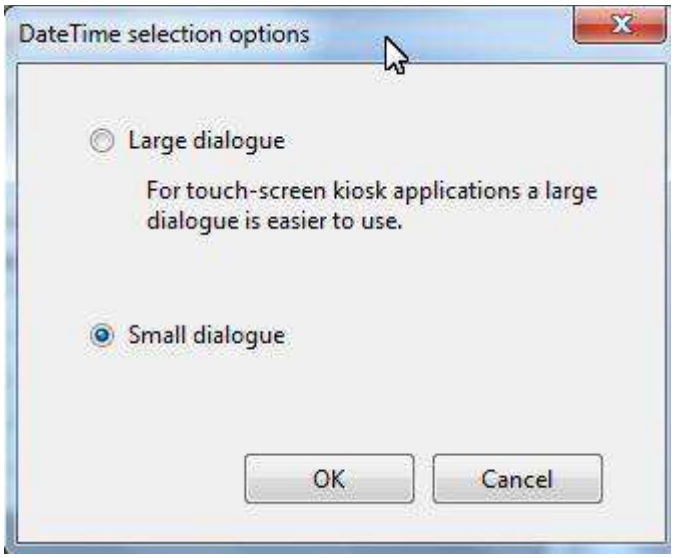
For each column we may choose an editor by setting FieldEditors property.

8.5.1 Field Editor: Date Selector

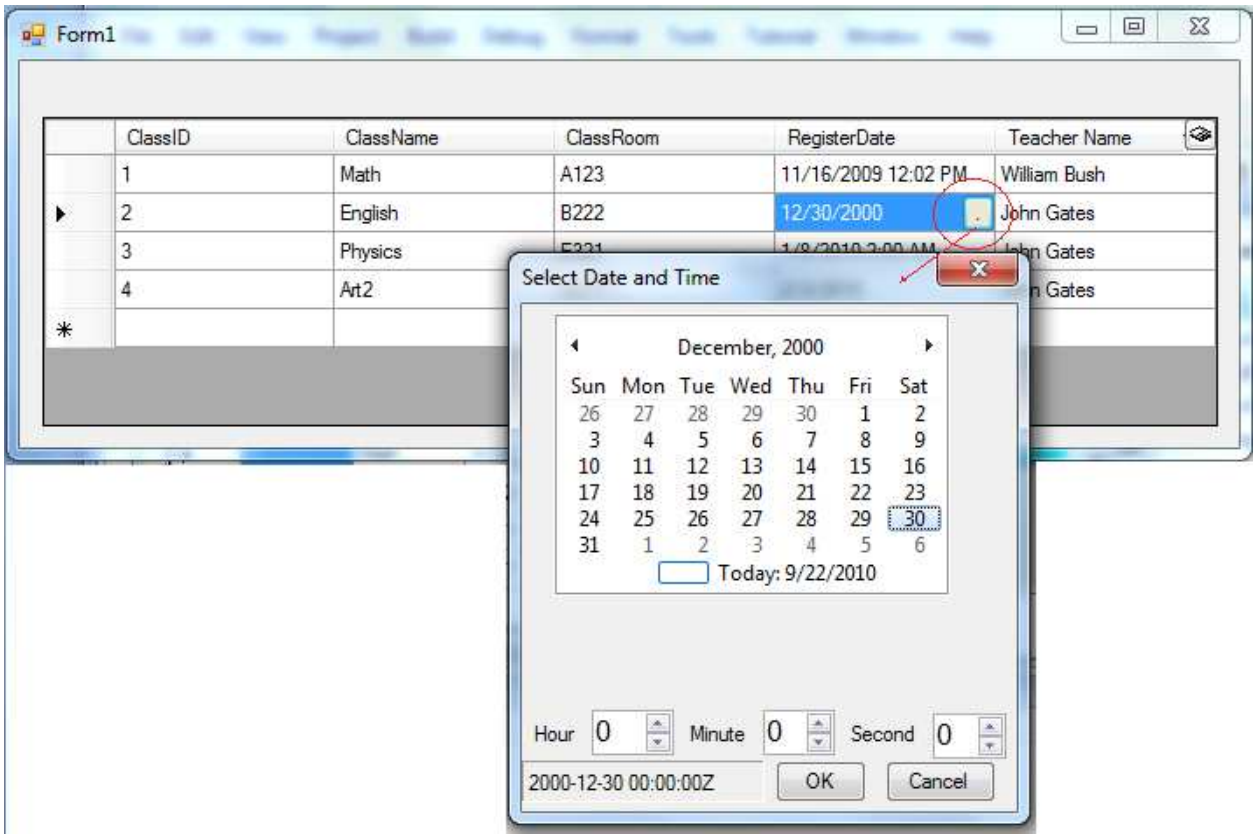
Date Selector is used for date time field. It will show a calendar for choosing date/time.



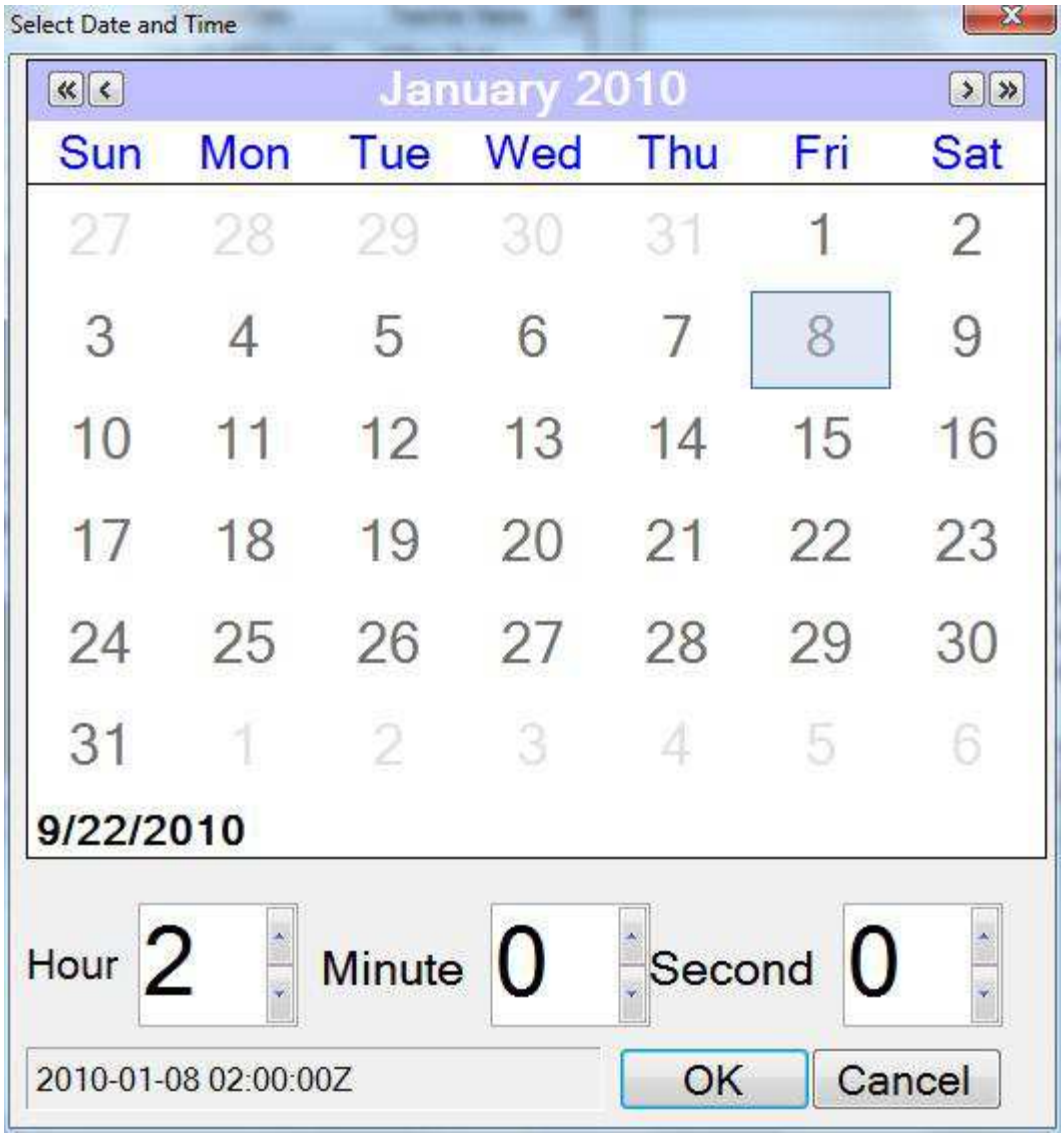
You may choose small or large dialogue box:



Suppose we selected “Small dialogue” option. At runtime when the column is selected, a button appears. Click the button, a date time selection dialogue box appears:

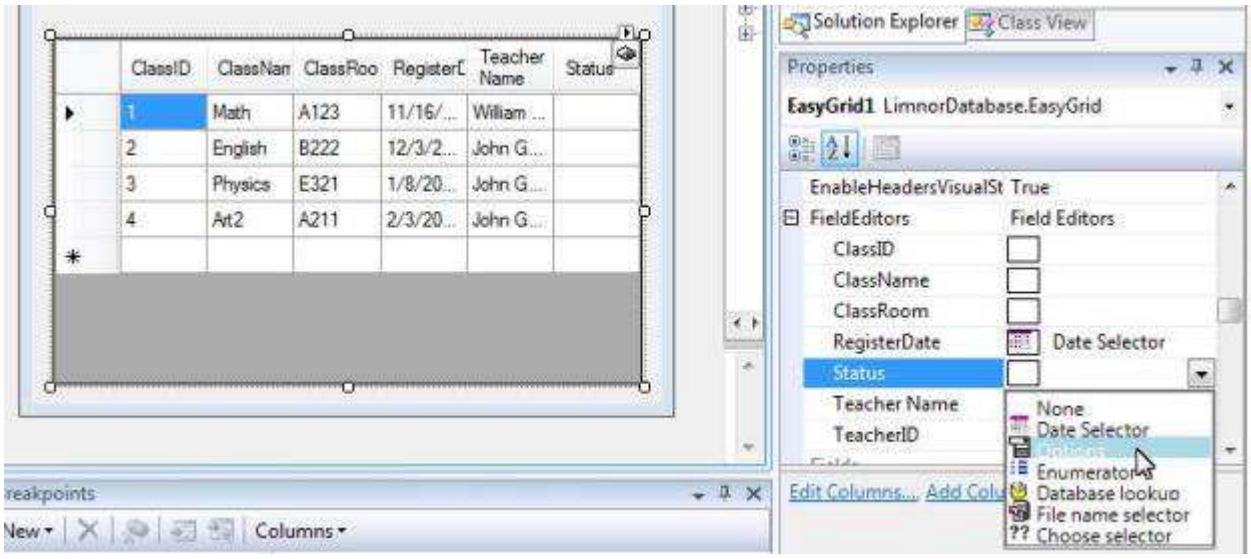


If “large dialogue” option is selected then a large dialogue box appears for selecting date and time.

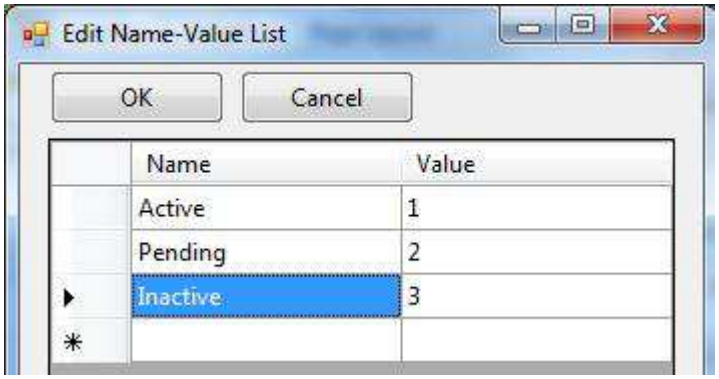


8.5.2 Field Editor: Options

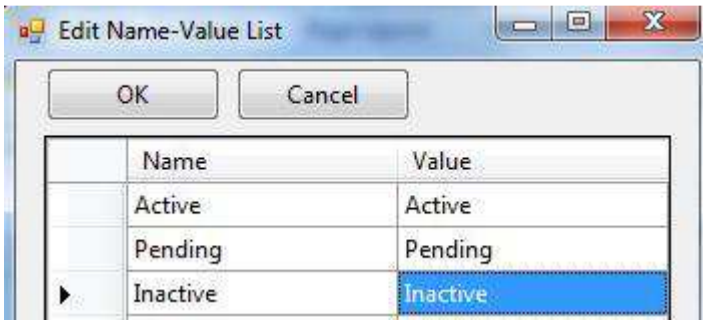
Suppose the Class table has a field “Status”. It may only be one of the following values: Active, Pending, and Inactive. We do not want the user to type in freely for this field because mistakes can easily be made. We may not want to use database lookup because for such few values it is not worthwhile connecting to database. We may use Options to input this Status field. Options Editor allows you to specify a set of data to be displayed in a drop-down list to be chosen from.



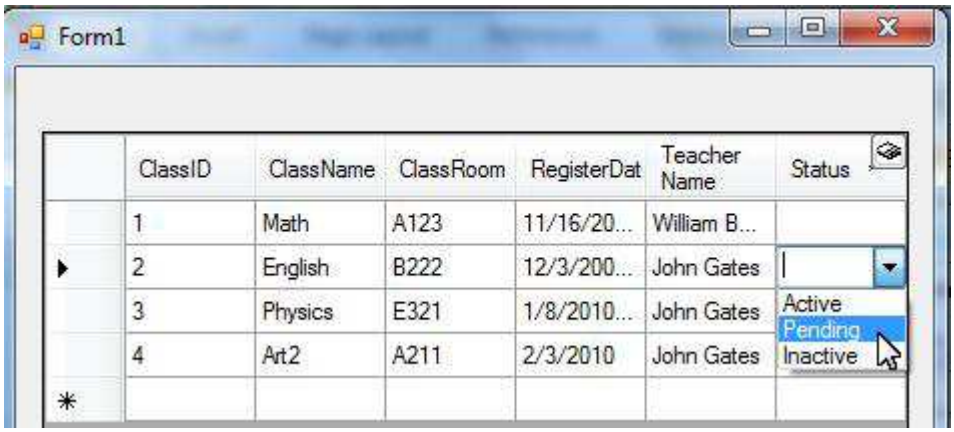
It will ask us to define the options. It gives us the freedom of defining options values. For example, suppose the Status is an integer field and defined as 1 is Active; 2 is Pending and 3 is Inactive. Then the options can be defined as shown below



If the Status is a string field then the options can be defined as shown below



At run time the user just select the option name. The corresponding option value will be used.



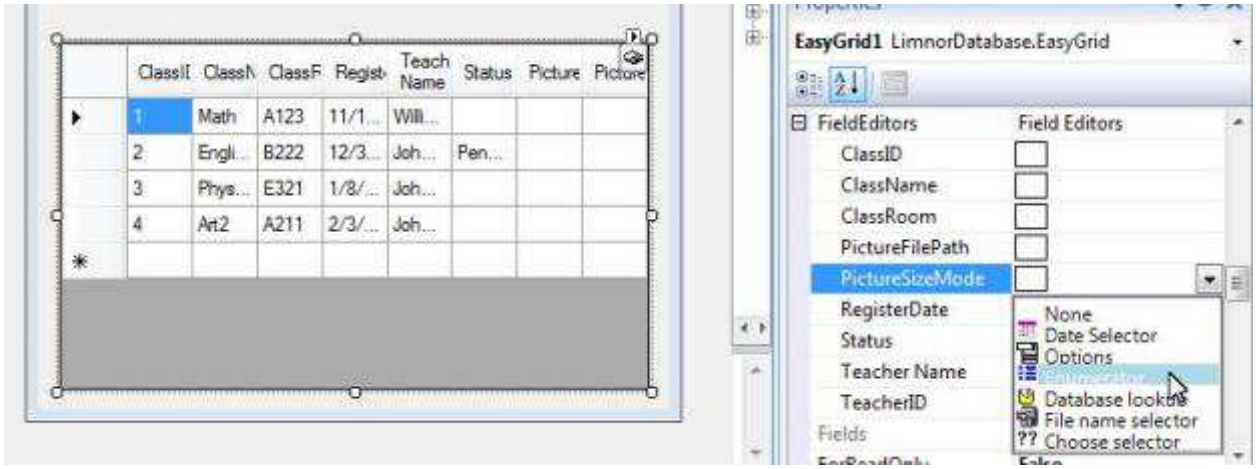
8.5.3 Field Editor: Enumerator

Enumerator allows you to specify an enumeration defined in an assembly (a DLL). The enumerations will be displayed in a drop-down list box.

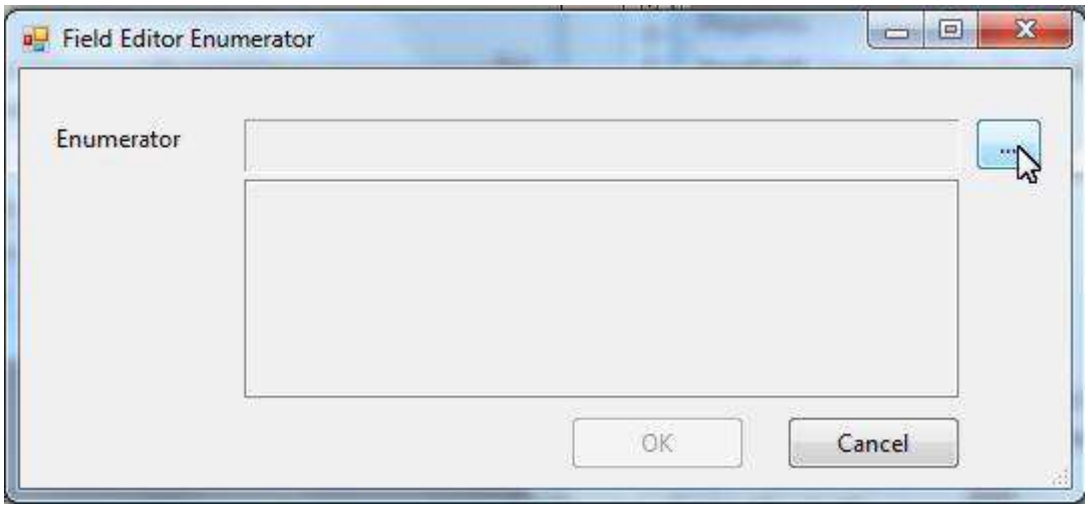
Suppose our application needs to display an icon or image for each Class record. Class table uses a PictureSizeMode field to indicate to the application how the image should be sized. This PictureSizeMode is an enumerator PictureBoxSizeMode in System.Windows.Forms namespace.

This PictureSizeMode is an integer field. But its value is valid only if it is a valid value for PictureBoxSizeMode enumerator.

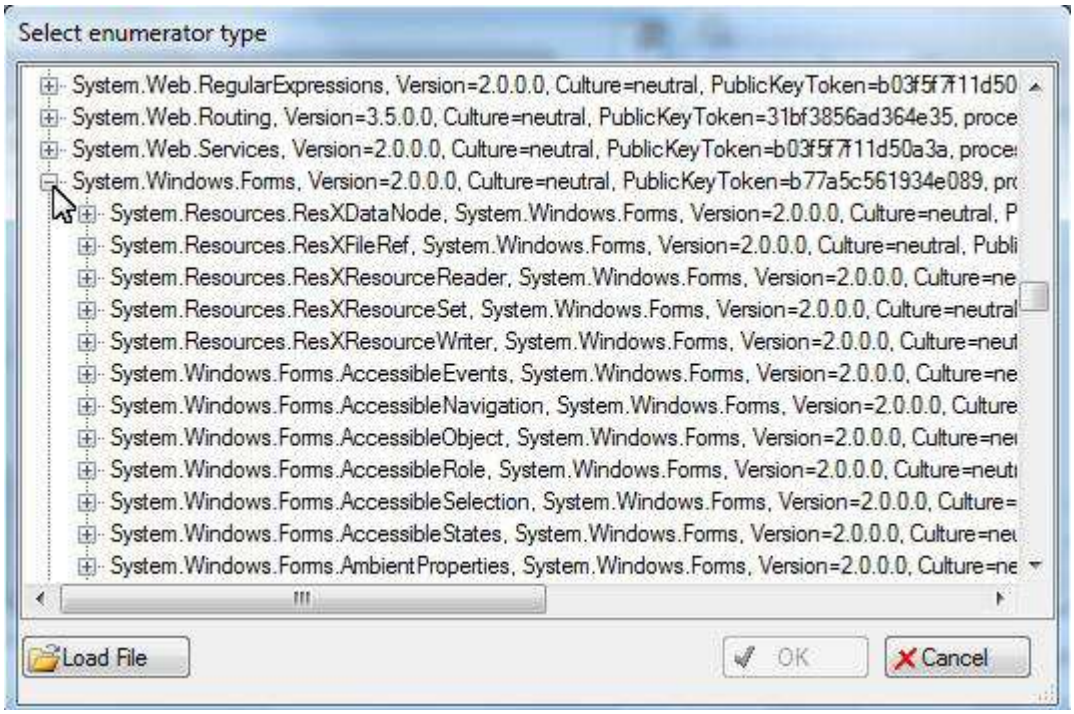
We may use Enumerator field editor to edit this field so that data entry is easy and no mistake.



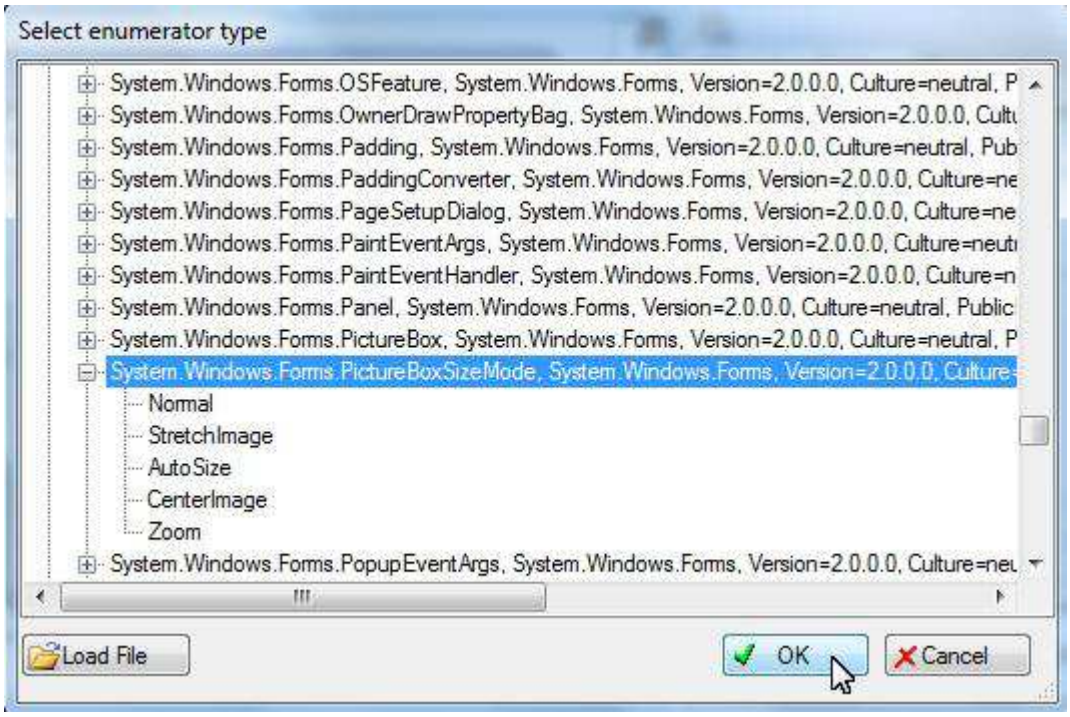
We need to select the enumerator



We know the PictureBoxSizeMode is version 2.0 System.Windows.Forms namespace:

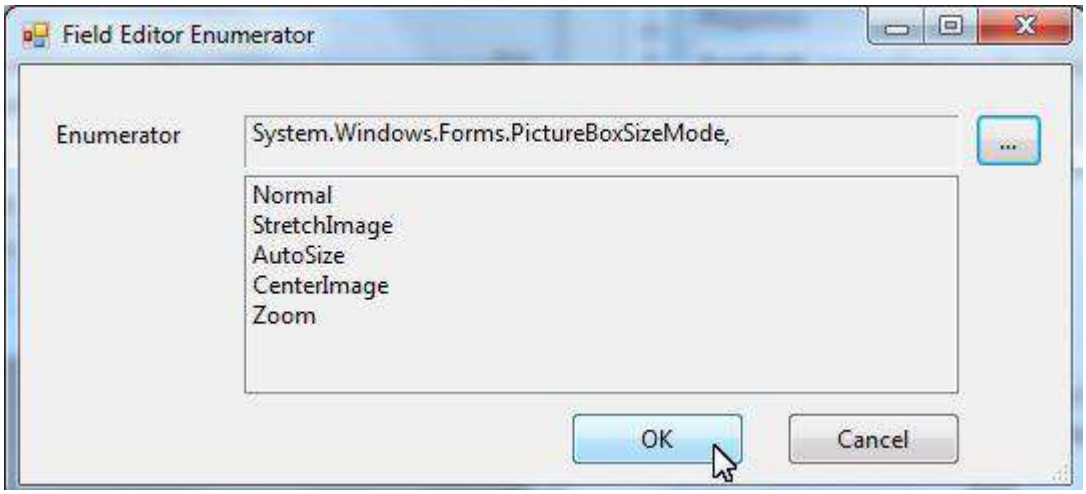


Select PictureBoxSizeMode and click OK:

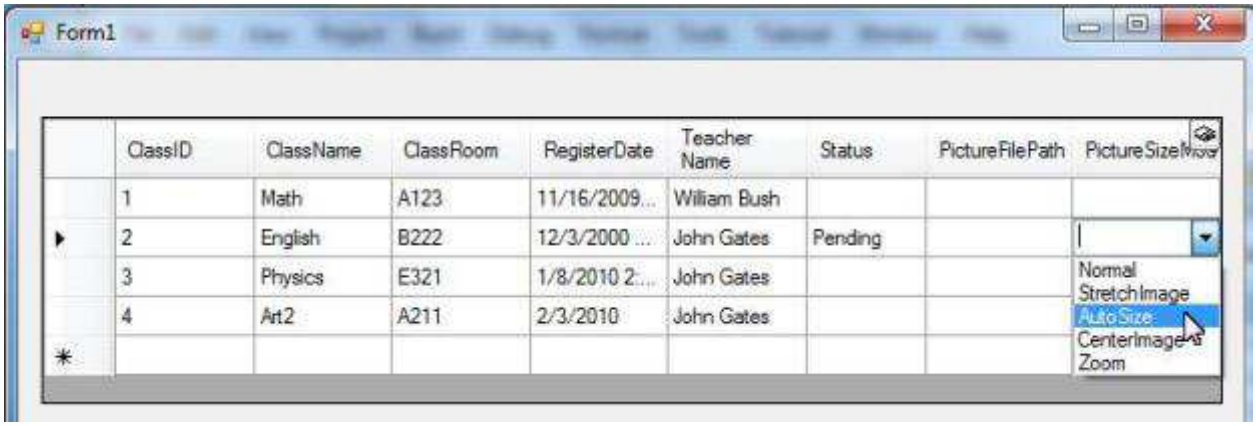


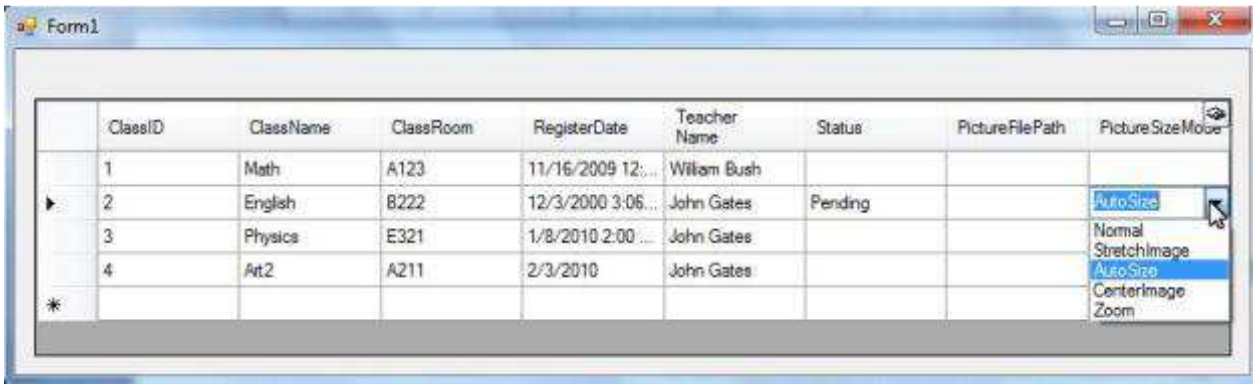
If an enumerator is not in the namespaces listed then you may click “Load File” button to specify the DLL file containing the wanted enumerator.

Click OK to use the selected enumerator



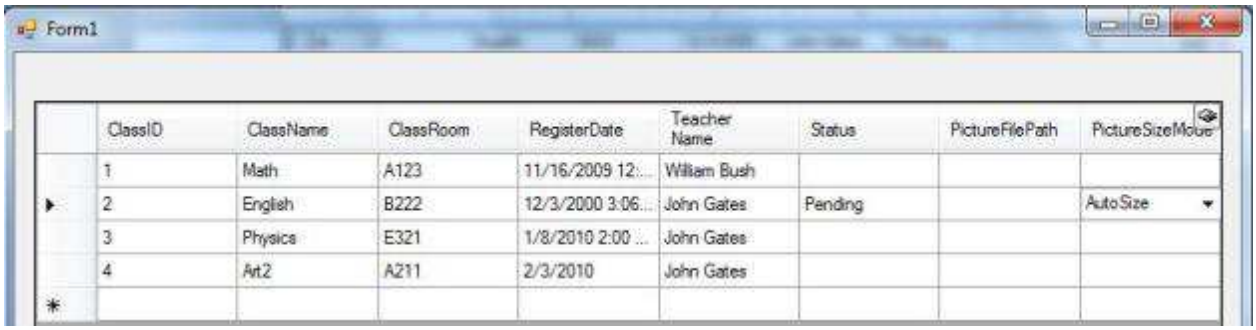
At runtime, the field can be edited with a drop-down:





Because we use integer for the field, the corresponding integer is entered into the record:

But select the field, the name of the value is displayed:



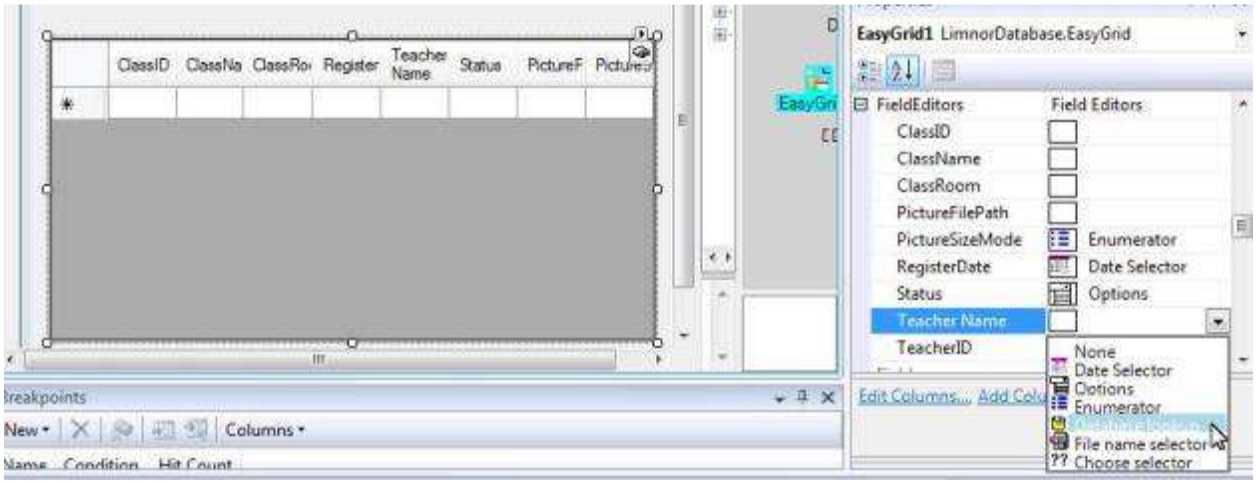
If the field is a string field then the enumerator name will be saved into the database. That is, instead of number 2, the string AutoSize will be saved in the database.

8.5.4 Field Editor: Databaselookup

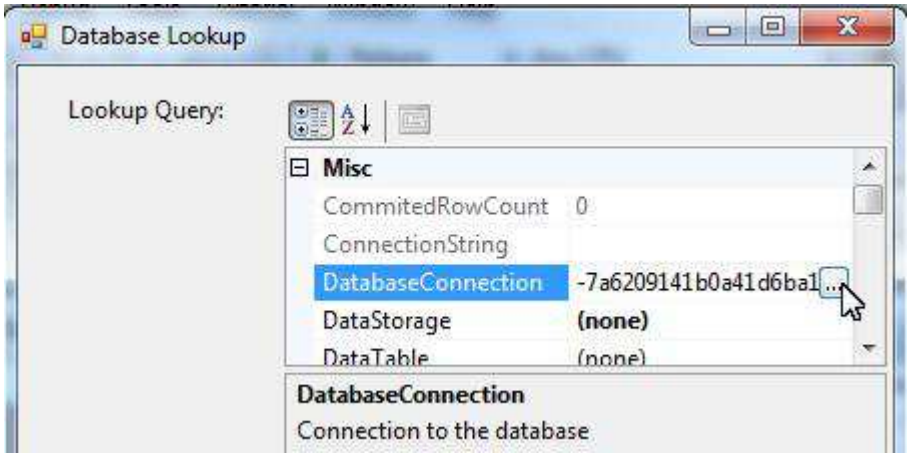
Databaselookup allows you to specify a query to get data from a database. The data will be displayed in a drop down list box. It can be used to update more than one column at the same time.

In this sample, we may use Databaselookup to enter the TeacherID. The user enter TeacherID by selecting teacher name.

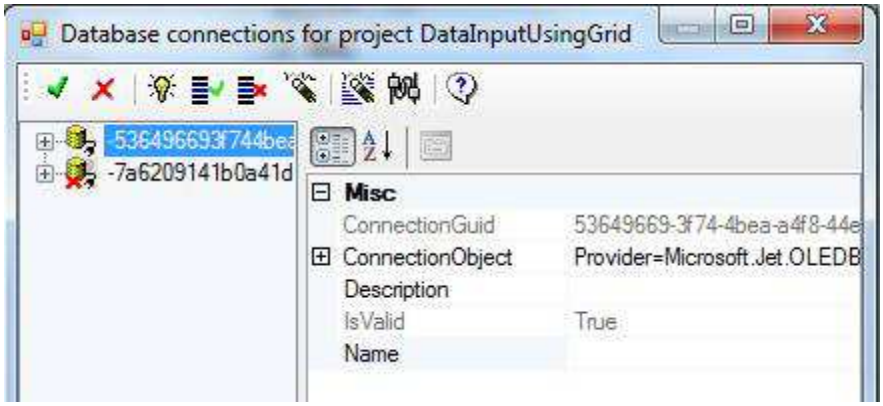
Select Databaselookup for field Teacher Name:



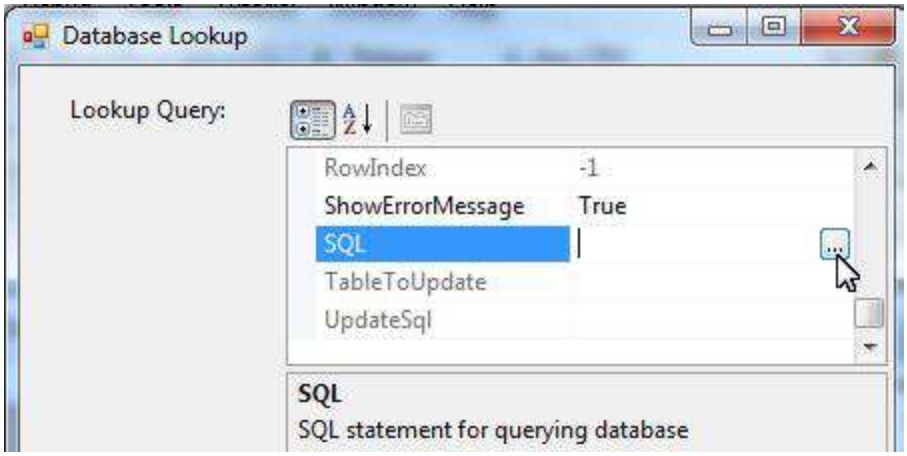
Set the DatabaseConnection for the Lookup query:



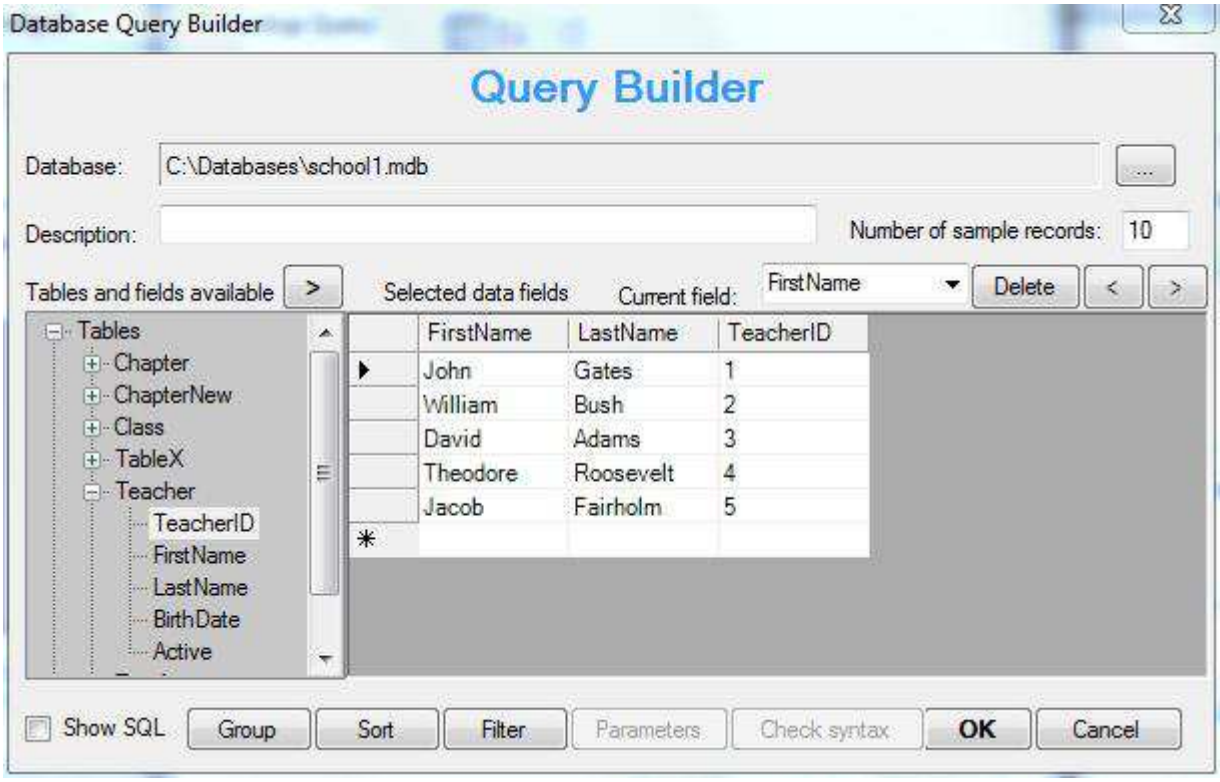
Select the same database connection used by the EasyGrid:



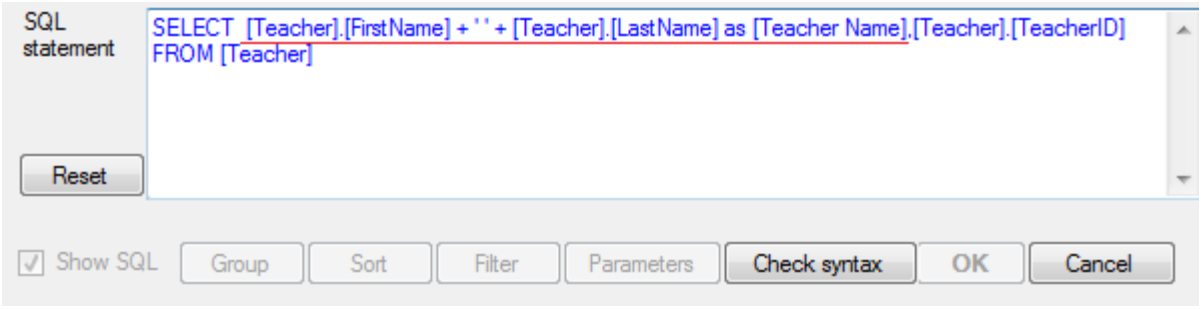
Set the SQL property for the lookup query:



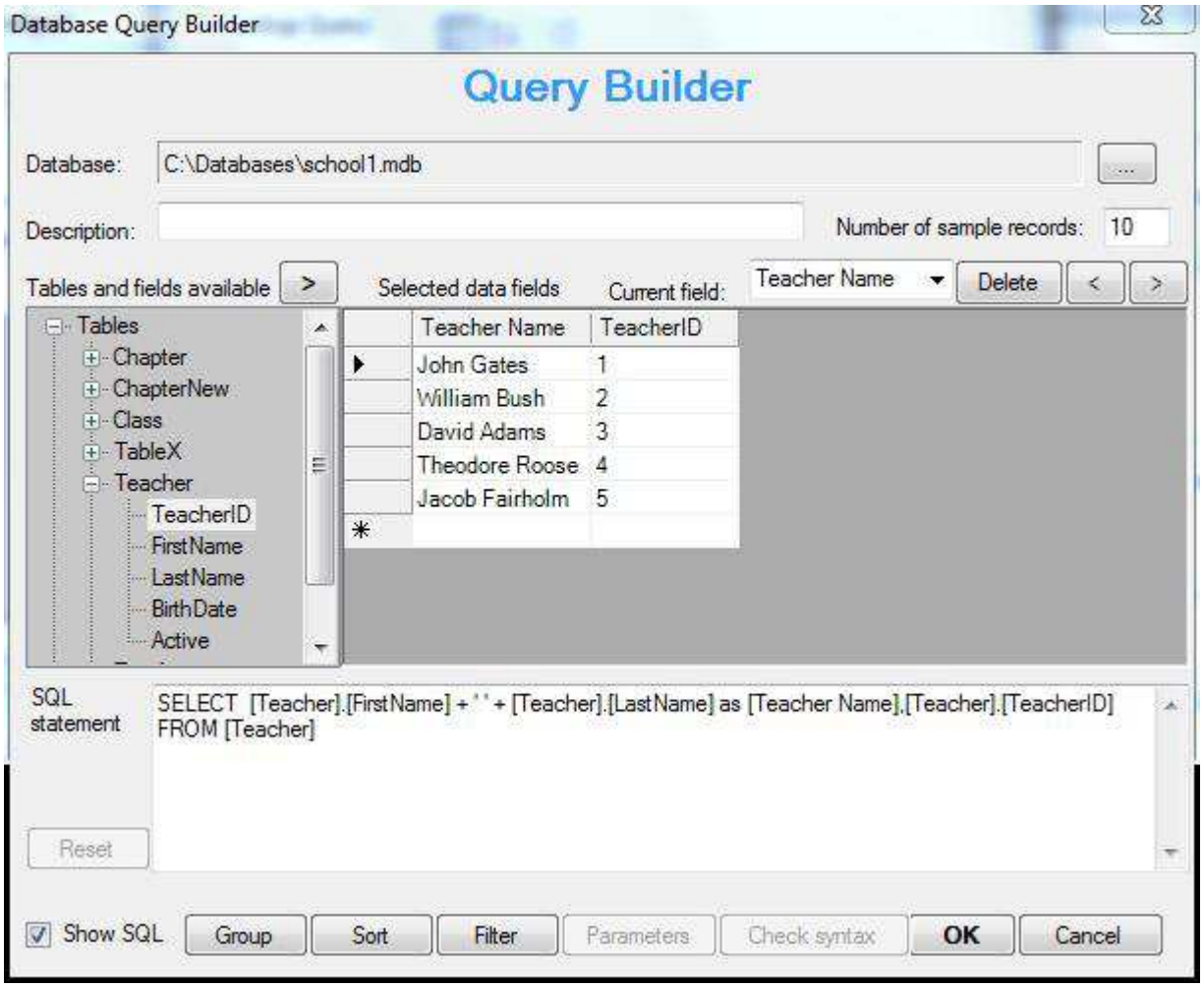
Include FirstName and LastName for displaying teachers; include TeacherID for updating the Class table:



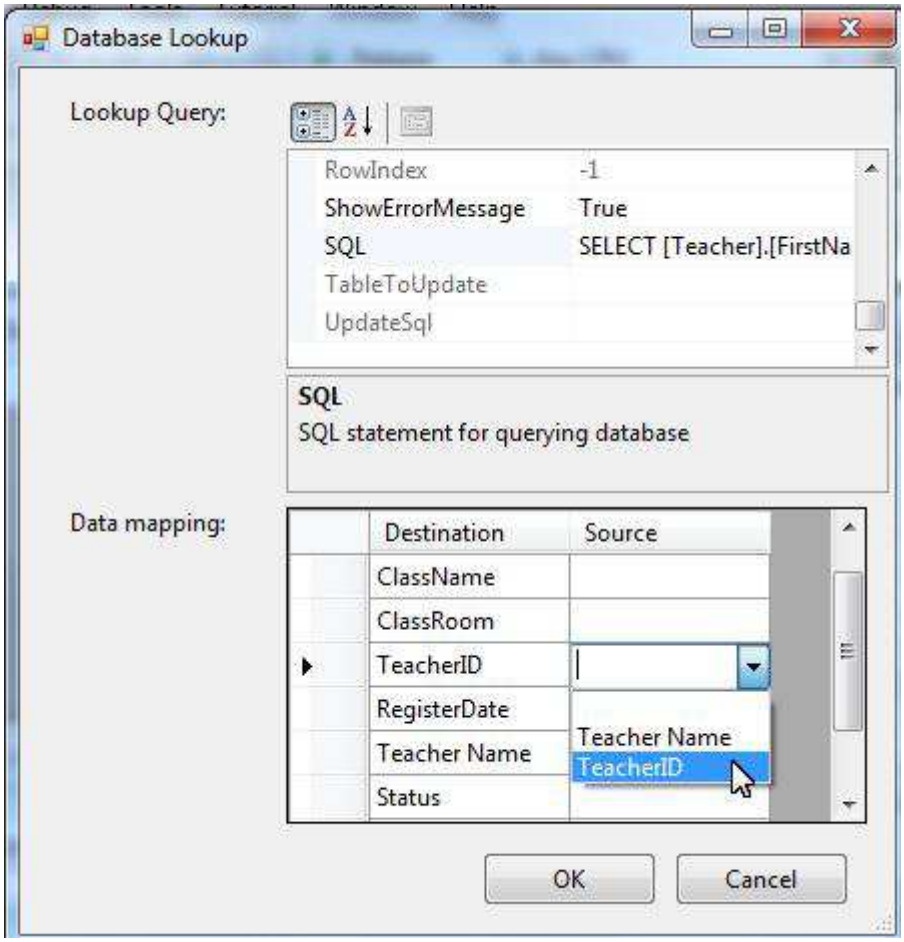
Combine FirstName and LastName into Teacher Name:



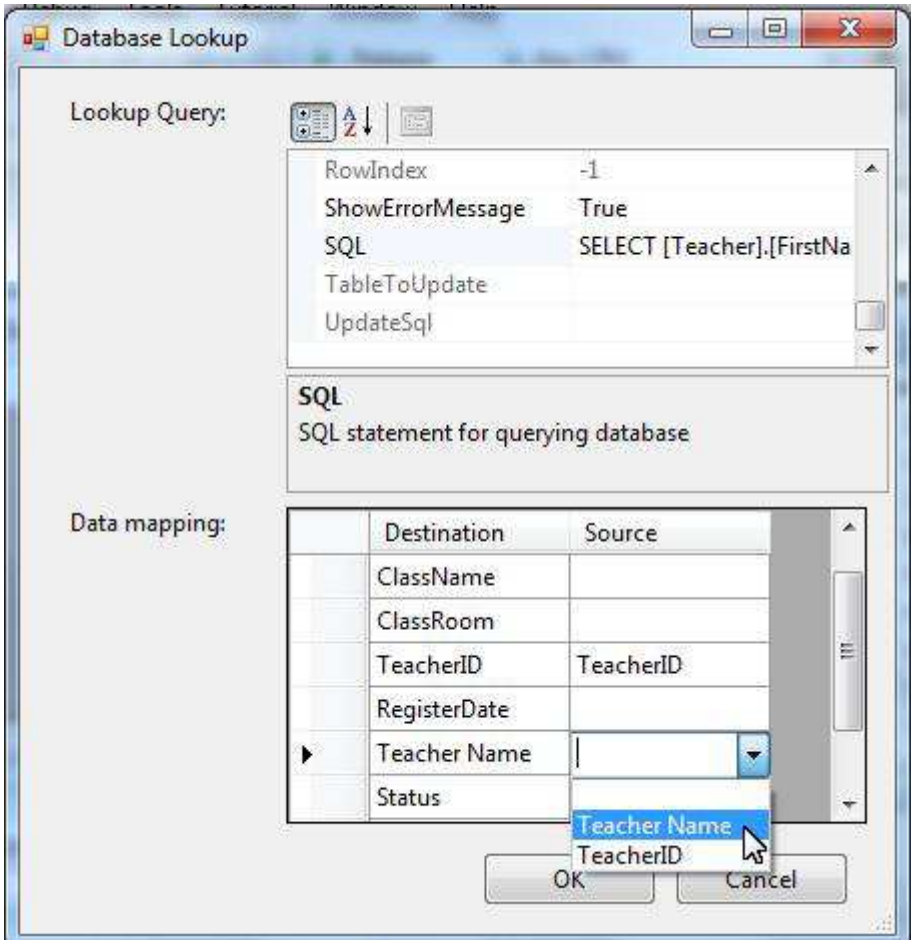
Click “Check syntax” button. If we made the modifications correctly then the Teacher Name field appears in the query:



We can now map the lookup query with the table we want to enter data. In this case, map TeacherID to the lookup query:



Map Teacher Name to the lookup query:

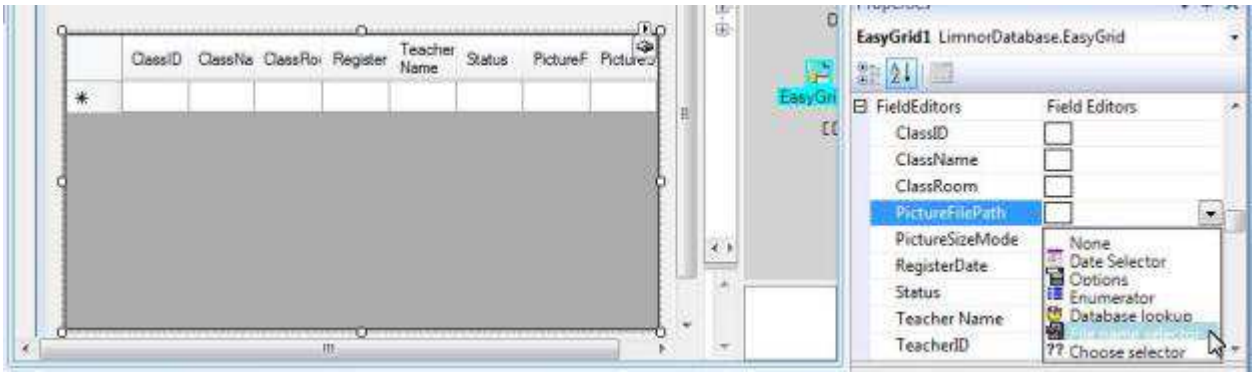


Note that Teach Name mapping is for display purpose only, not for saving data into database.

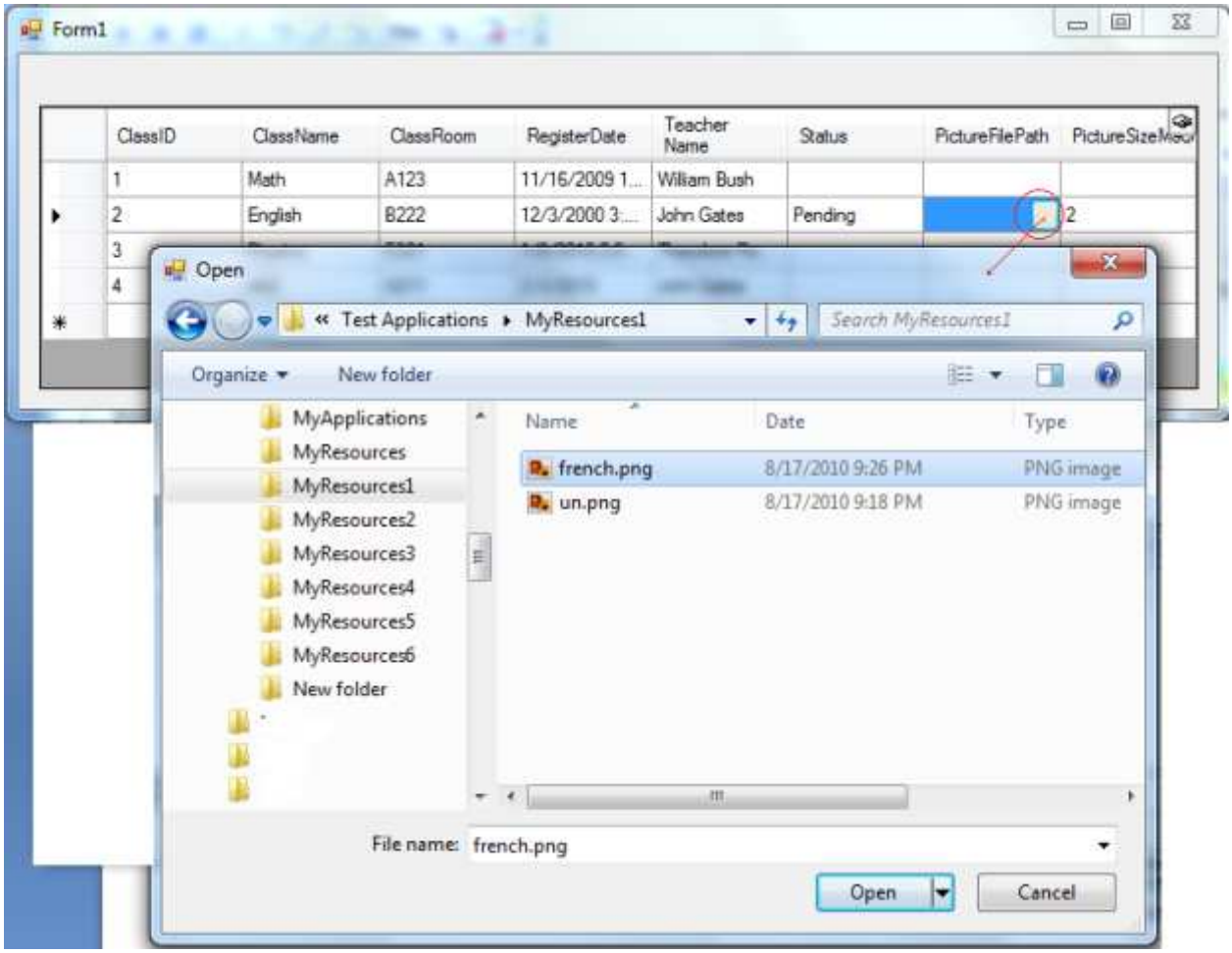
At runtime, TeacherID is entered by selecting teacher name:

8.5.5 Field Editor: File name selector

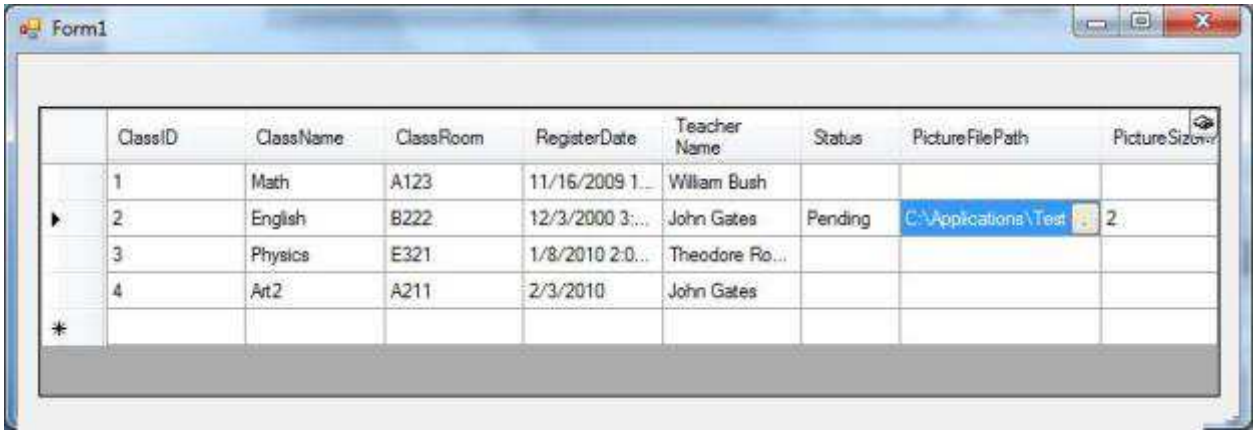
If a string field is supposed to save file paths then a File Name Selector can be used to show a file browser for selecting file name.



At runtime, a button  will appear when entering the cell. Click the button, a file browser appears for selecting a file:



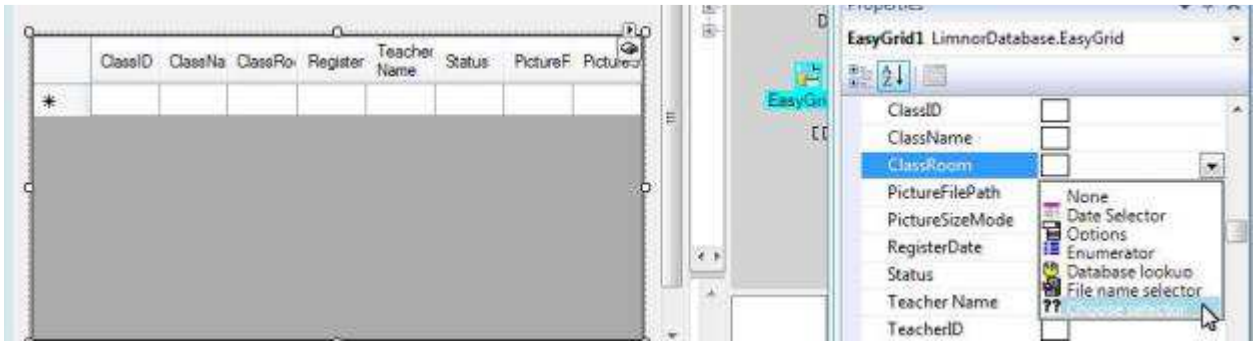
The selected file path will be entered into the cell:



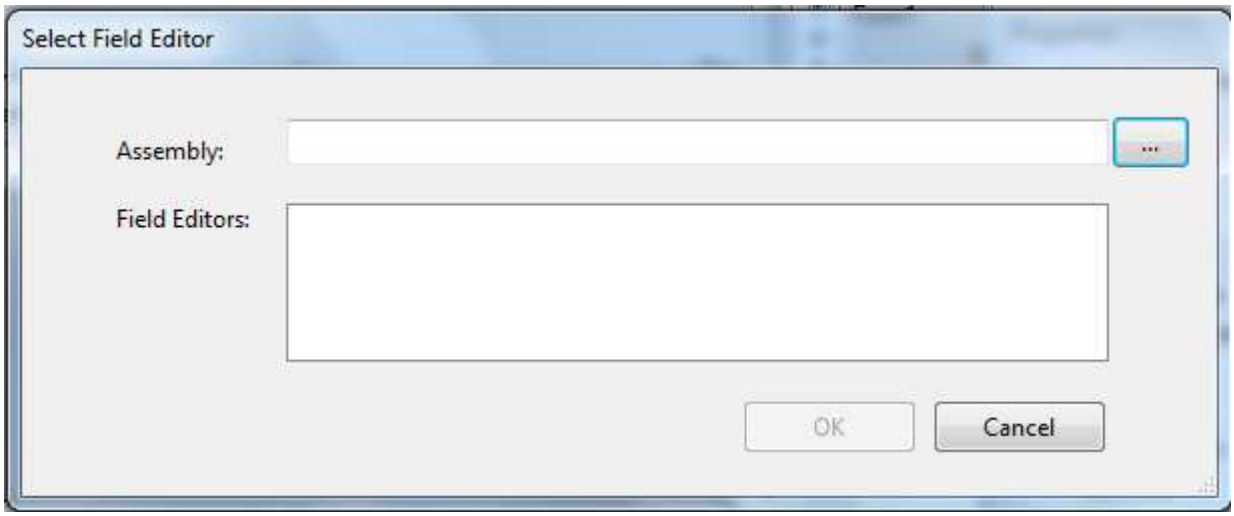
8.5.6 Field Editor: other editors

If the above field editors are not good for the field we want to edit then we may make a new field editor. A new field editor can be developed in any .Net development tools such as VB.Net, C#, etc., including Limnor Studio.

To use a new field editor, select ChooseSelector:



A dialogue box appears for selecting the field editor:



Click button  to select a DLL file containing the field editor. If the DLL contains field editors then they will be listed. Select a field editor from the list and click OK to use it.

8.6 Show calculated value with expression

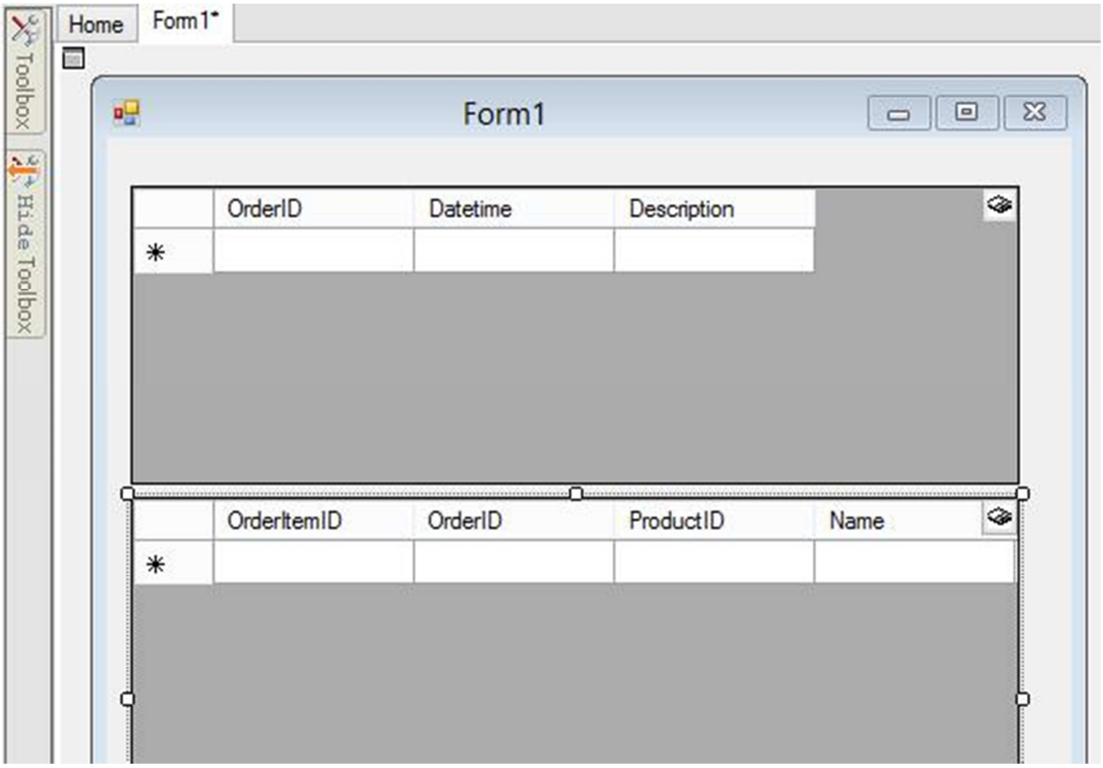
Section "Calculated Fields" shows how to include calculated fields in a query. It shows an example of calculate Amount:

[OrderItem].[Price] *[OrderItem].[Quantity] AS [Amount]

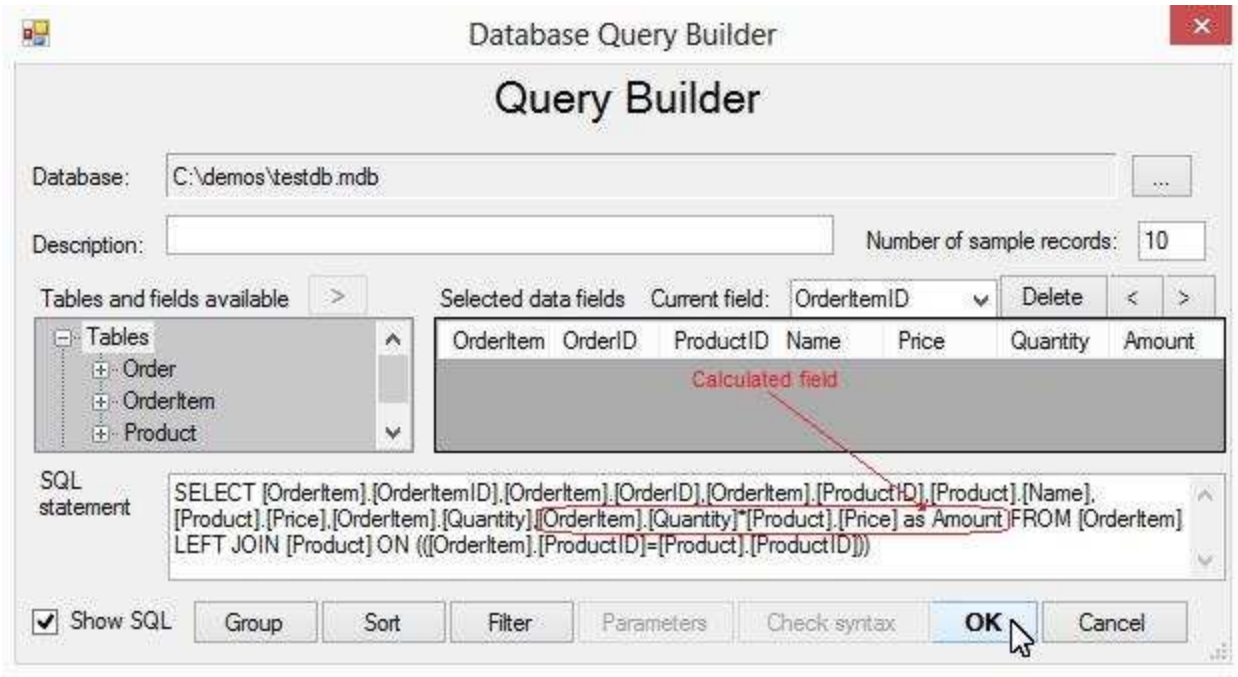
But such arrangement does not respond to data modifications through EasyGrid. Suppose the data are displayed in an EasyGrid, and the user modify Price cell or Quantity cell, the value displayed in Amount cell does not change.

EasyGrid has a property "FieldExpressions" which can be used to setup a runtime calculation expression. Such an expression will be evaluated once the focus leaves a row.

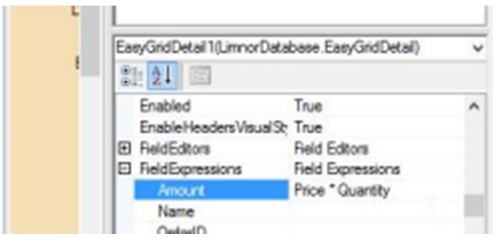
Below is a simple sample. Suppose we create a master-detail display.



The query of detail records includes a calculated field:

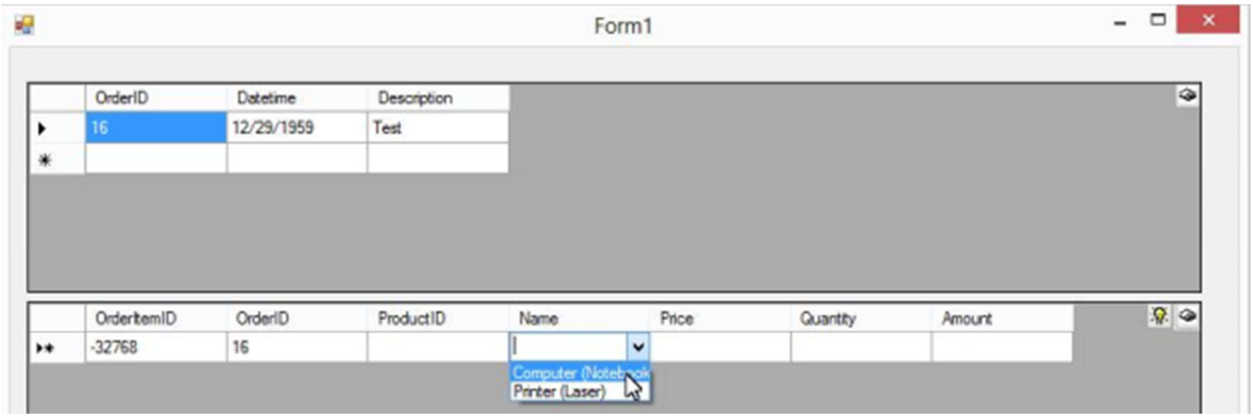


Set an expression of Price * Quantity to the Amount field:



Let's run the application to see how it works.

The form appears. Enter a master record. Once we a master record we may enter detail records. Because we setup a "database lookup" for the "Name" field to select a product, we may select a product from a dropdown list:

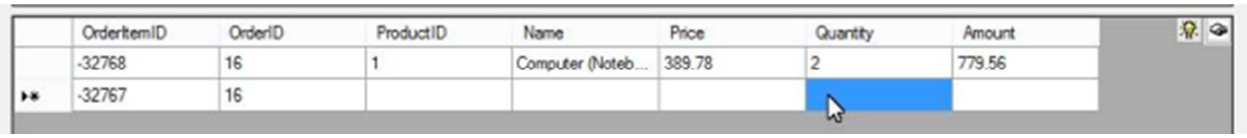


For how to create "database lookup", see section "Field Editor: Database lookup".

Once a product is selected, the price field is filled with value from the Product table. Now enter a quantity value:

OrderItemID	OrderID	ProductID	Name	Price	Quantity	Amount
-32768	16	1	Computer (Noteb...	389.78	2	

Click the next row in the grid to move focus off the current row, the value for Amount field appears showing the calculation result:



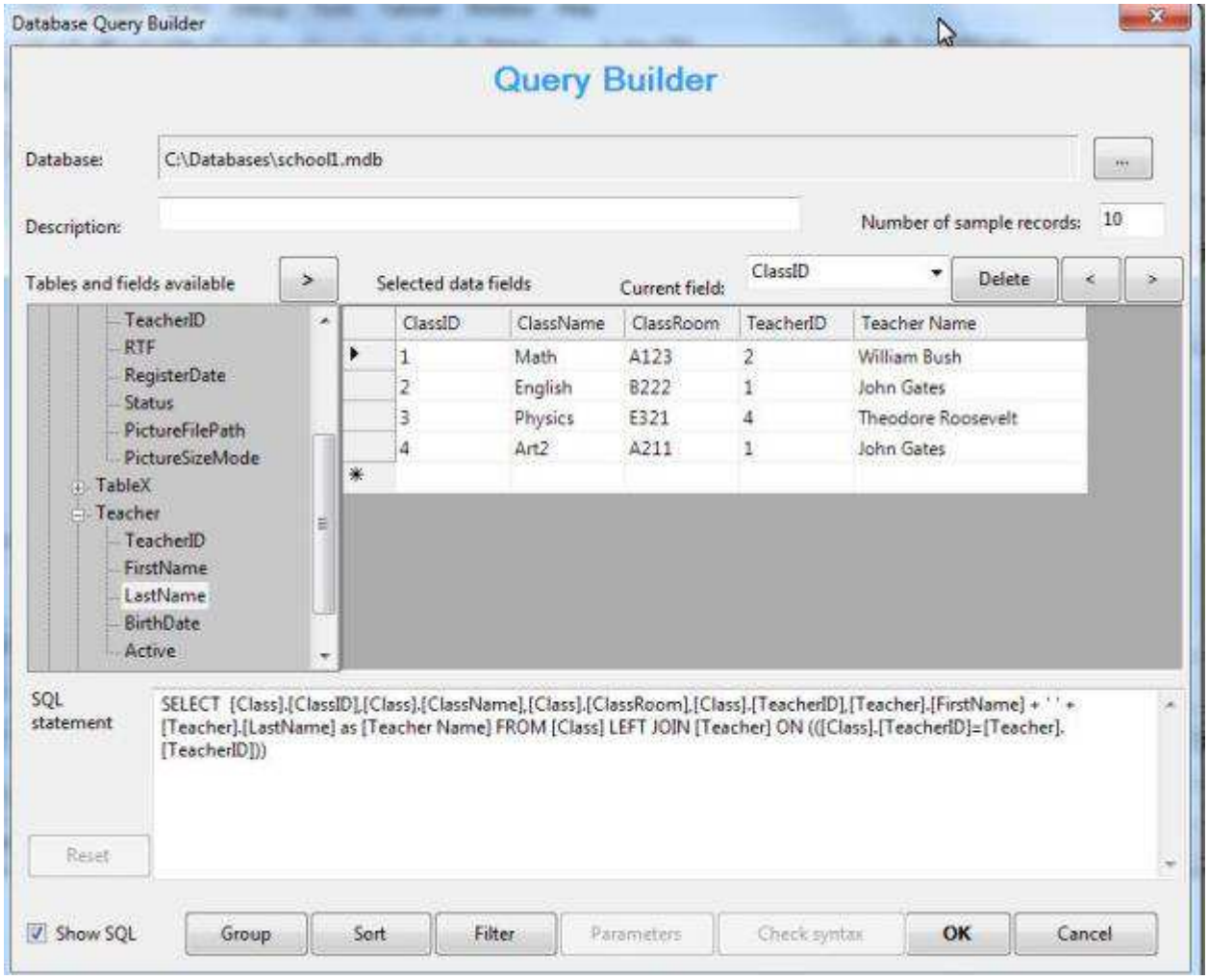
OrderItemID	OrderID	ProductID	Name	Price	Quantity	Amount
-32768	16	1	Computer (Noteb...	389.78	2	779.56
-32767	16					

9 Use Combo box for lookup

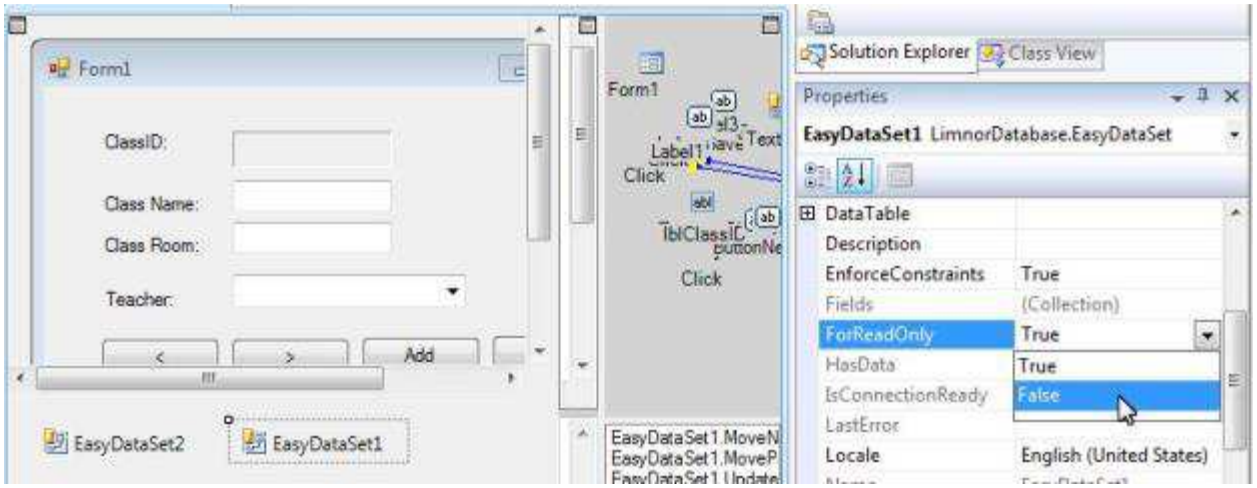
We saw that field editor Databaselookup could be used to do database lookup for data entry via EasyGrid. Such technique can also be used even if EasyGrid is not used.

9.1 Data Entry with Data-Binding

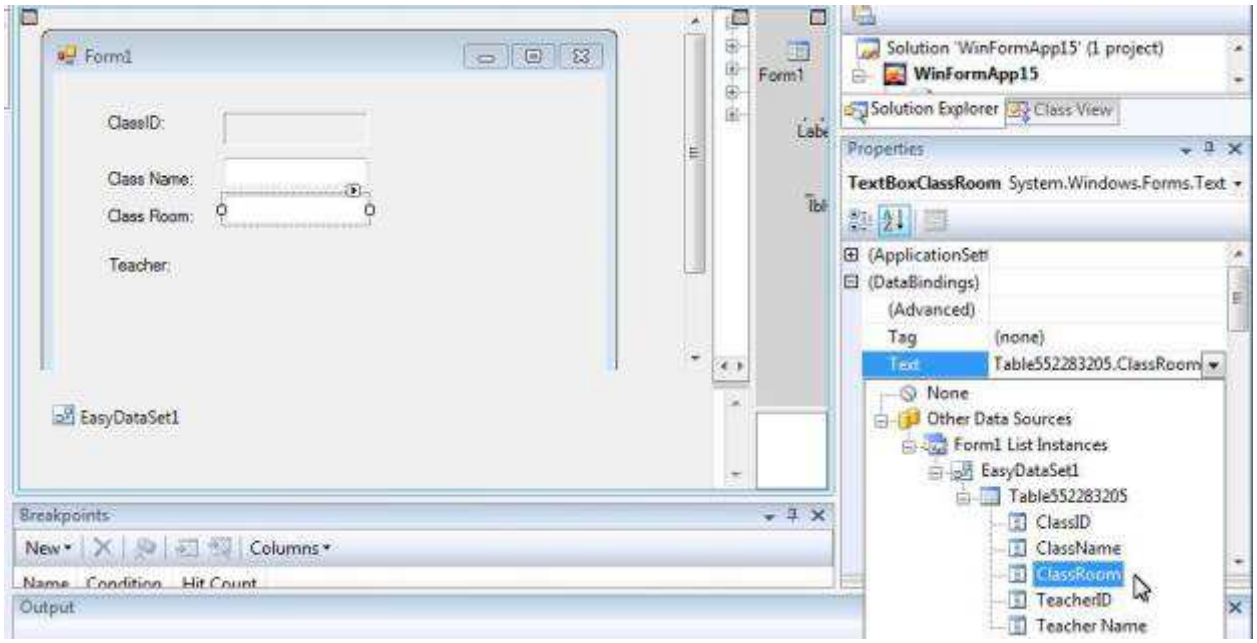
Use an EasyDataSet to do data entry. Set its DatabaseConnection to connect to a database. Set its SQL property to include fields from Class table and form a [Teacher Name] field in the query:



Set ForReadOnly to False to make the EasyDataSet writeable:

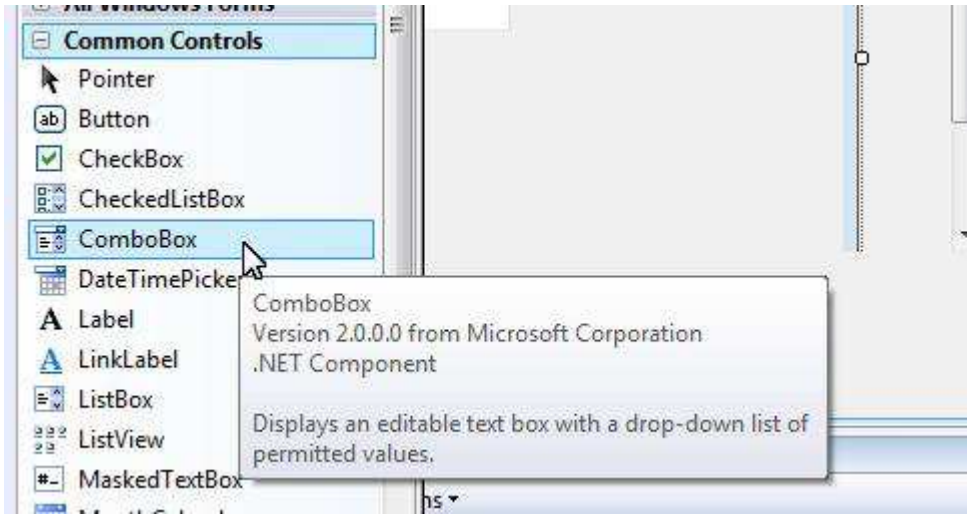


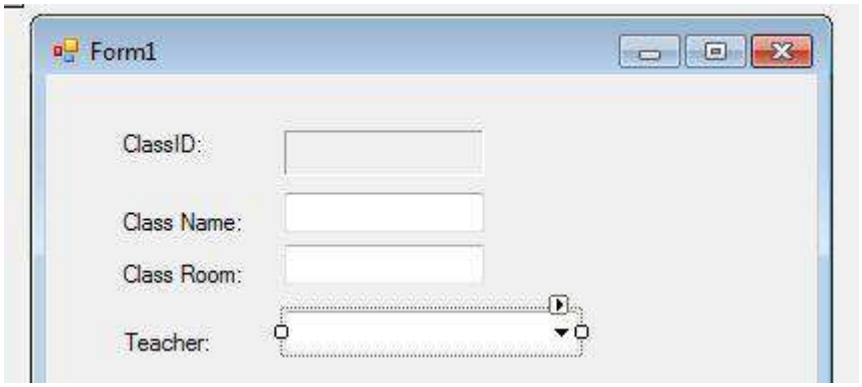
Use some controls to bind to the EasyDataSet:



We have done all the above operations before. We do not go into details again.

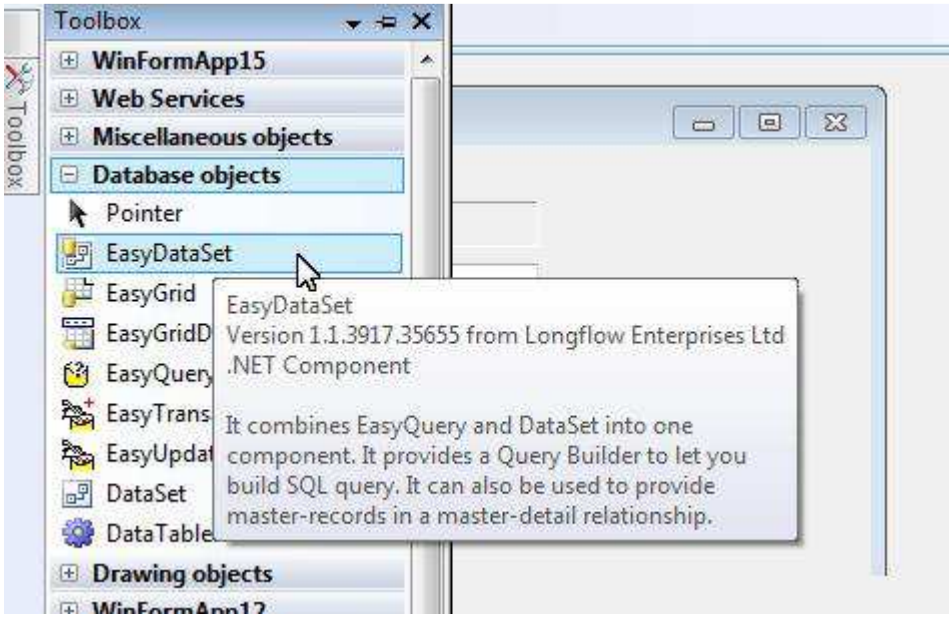
Drop a combo box for entering TeacherID into the Class table:



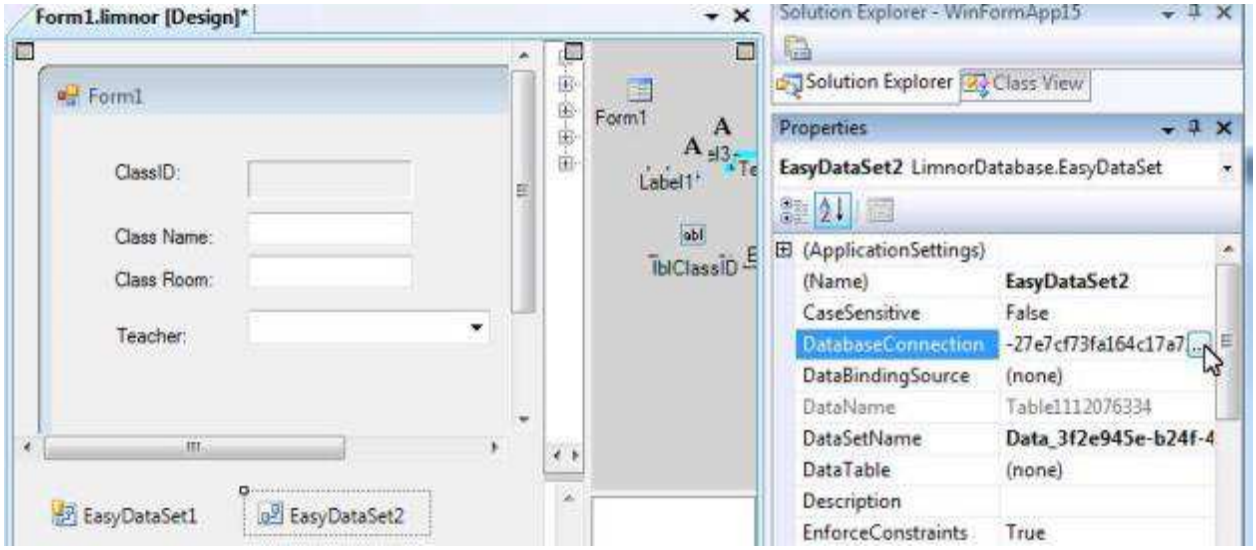


9.2 Create lookup query

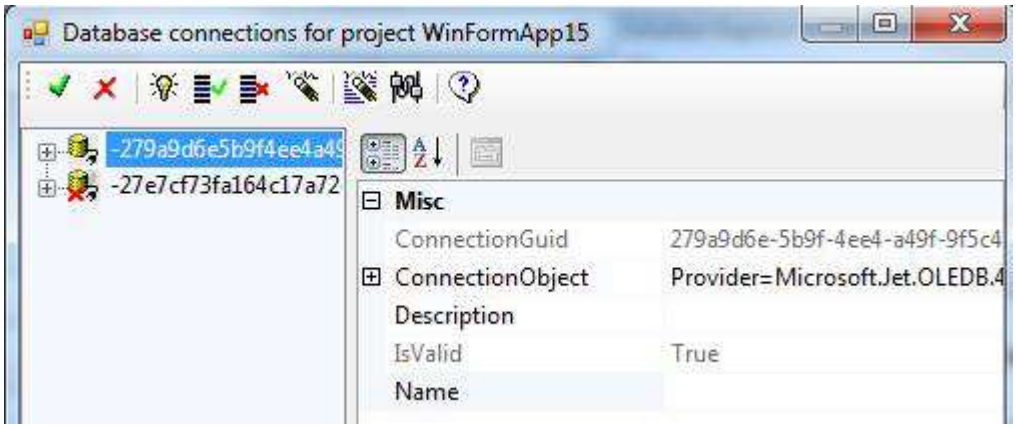
To use the ComboBox to lookup Teacher data in the database, we need to use another EasyDataSet to get Teacher data from the database.



Set its DatabaseConnection property;

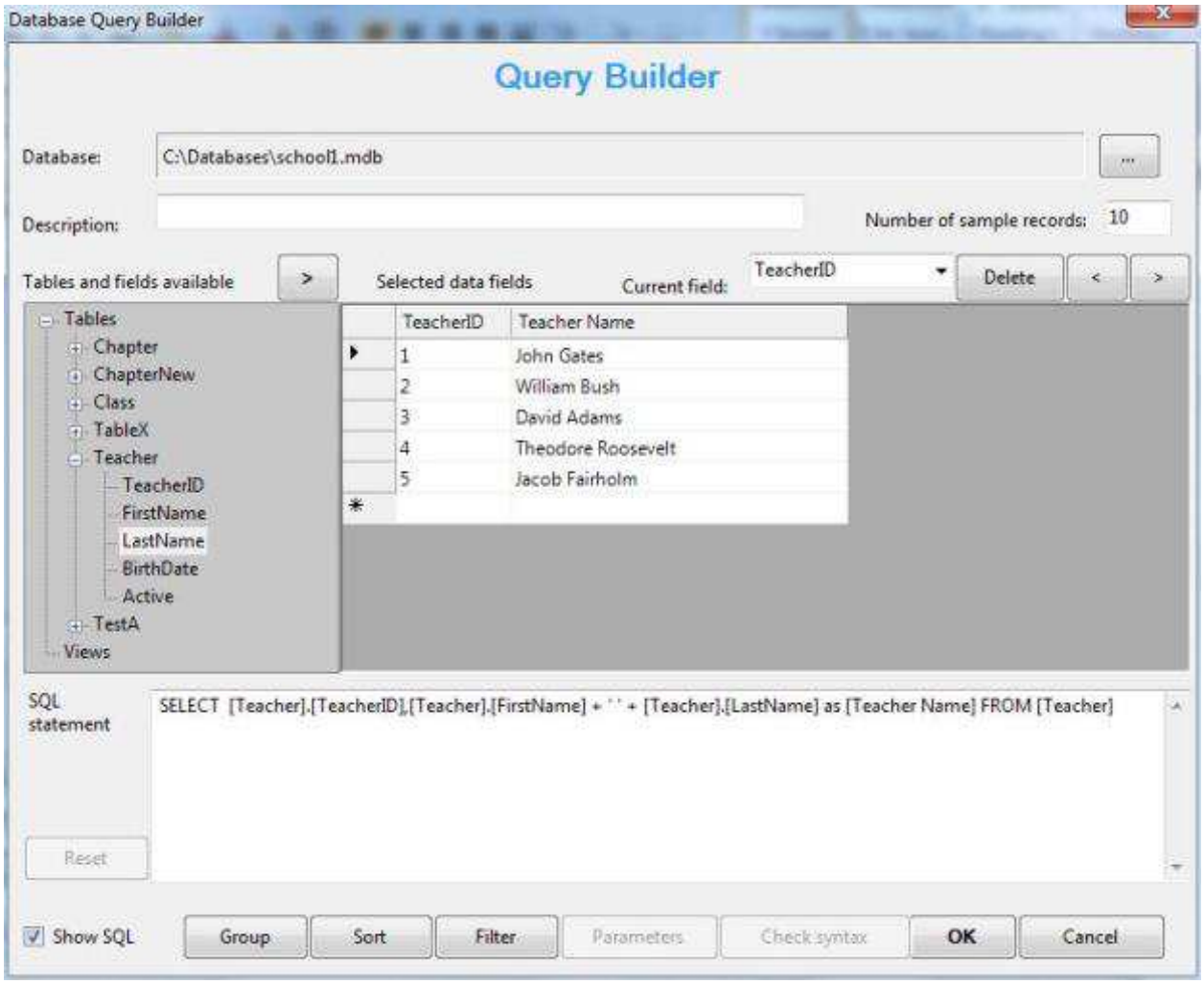


Select the database connection used by the first EasyDataSet:



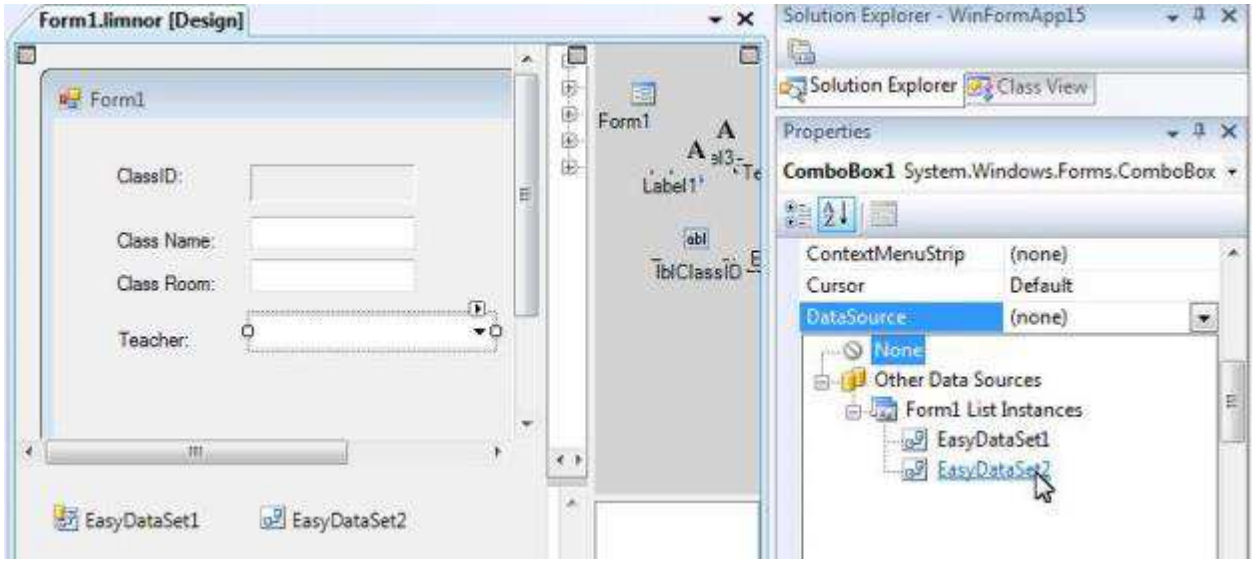
Set its SQL property to create lookup query:

We created the same lookup query before, so, we will not go into details:

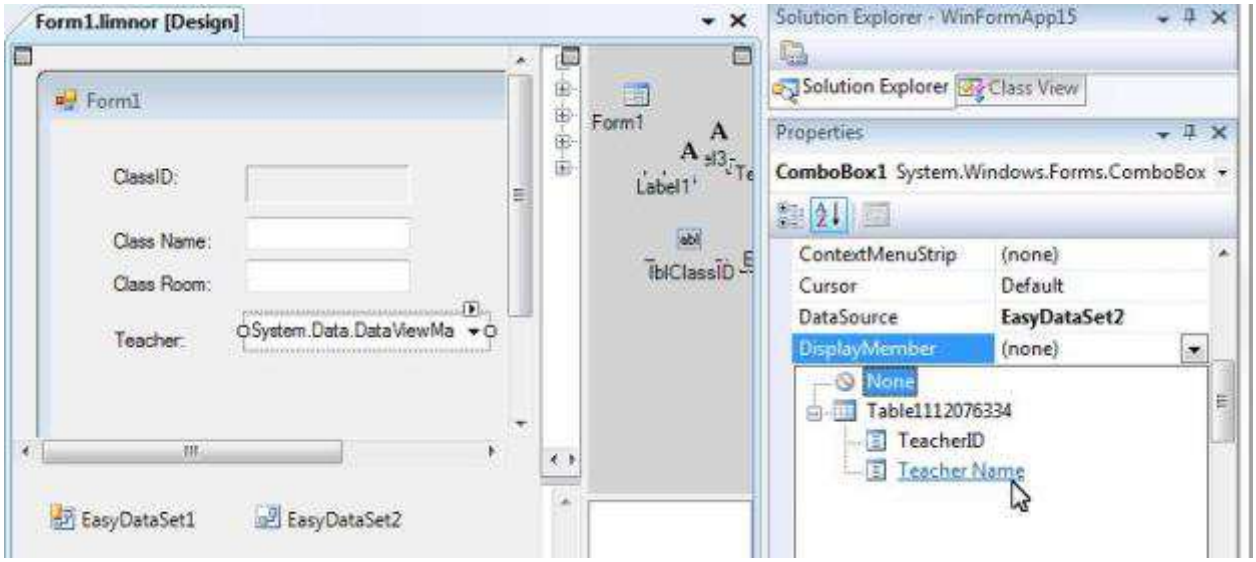


9.3 Bind Combo box to lookup query

Bind EasyDataSet2 to the ComboBox by setting its DataSource property to EasyDataSet2.

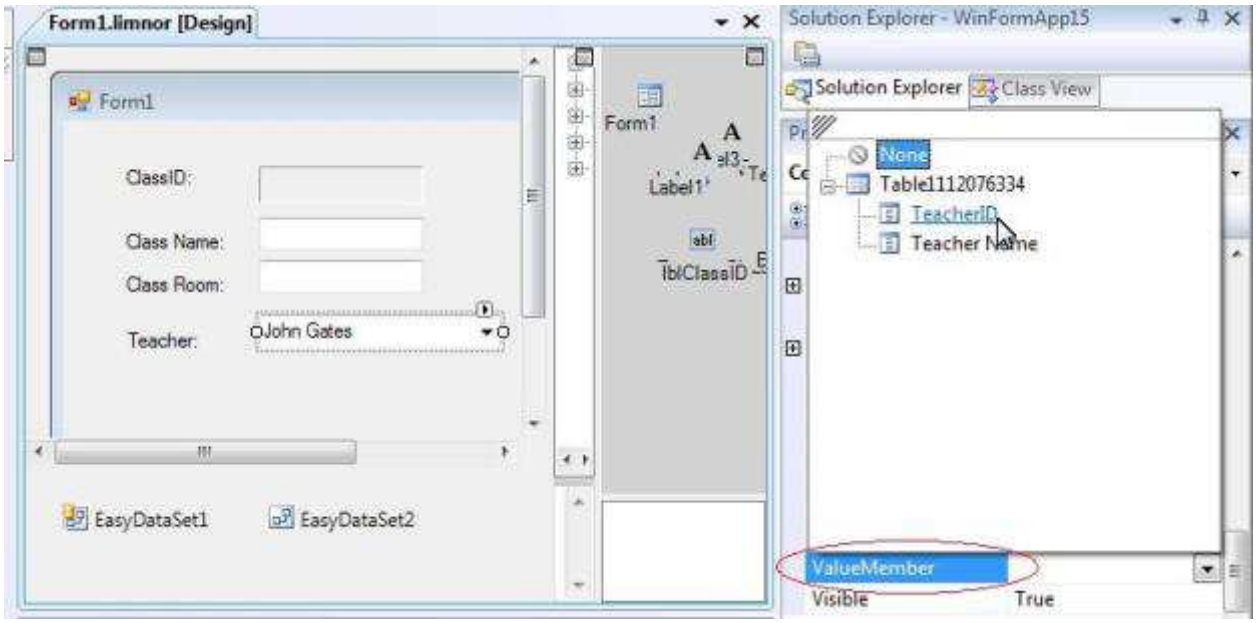


Bind DisplayMember to Teacher Name field:

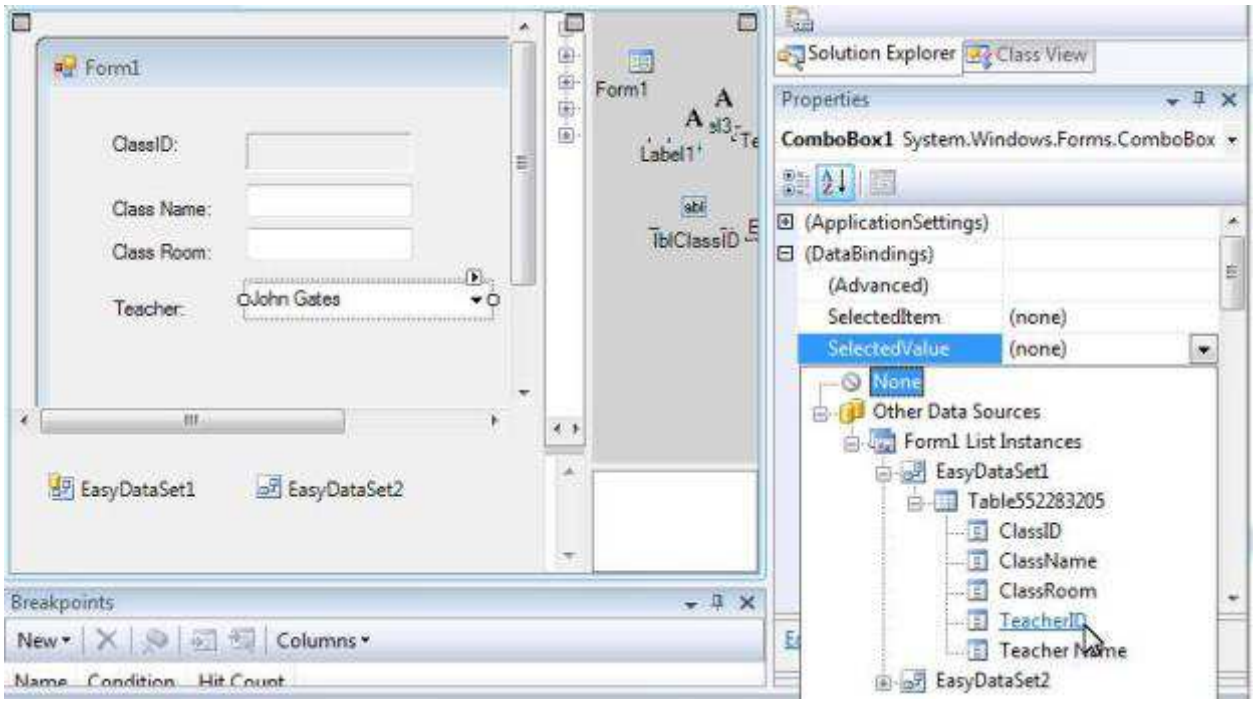


9.4 Make Lookup Link

To use the ComboBox as a Teacher lookup for the Class table, we need to link them by the TeacherID field. First, set the ValueMember of the ComboBox to TeacherID. Note that this is the TeacherID field from EasyDataSet2, the lookup query:

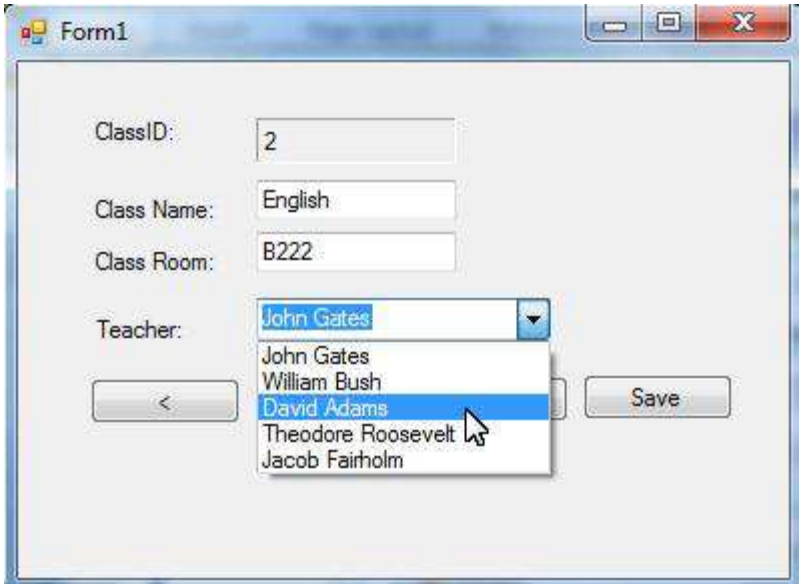


Next, bind the SelectedValue of the ComboBox to the TeacherID field of the Class table.



Note that this TeacherID is from EasydataSet1. It completes the linking between EasyDataSet1 and easydataSet2.

We may test it now. Run the application. We can use the combo box to select a teacher by name. The TeacherID of the selected teacher will be entered into the Class table.



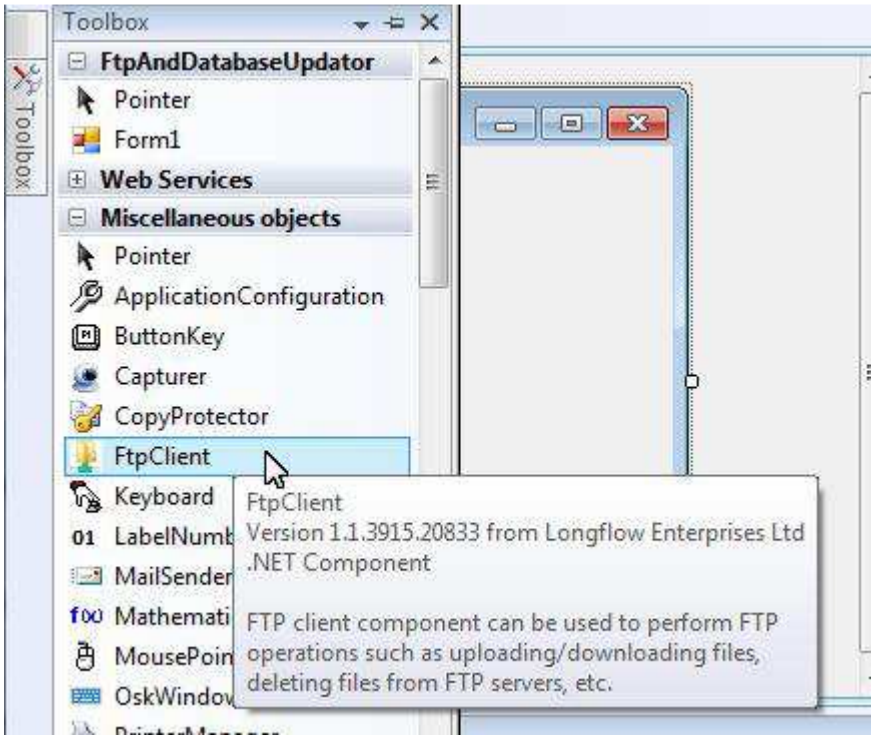
10 Modify Database Using EasyUpdator

EasyUpdator is a component for modifying data in a database without querying database first. It does not display data as EasyGrid does. It does not hold data as EasyGrid and EasyDataSet do. It may modify database structure as well as data.

10.1 Example Situation: File Upload Activity Recording

Suppose we want to build a file uploading application. We want to record in a database every successful file upload operation. In this situation, records inserted into the database are not supposed to be entered manually by the user via UI. The application should handle the file upload events and insert data into the database automatically. In such situations, EasyUpdator component can be used.

Drop an FTP component to the form:



Set FtpServer property:

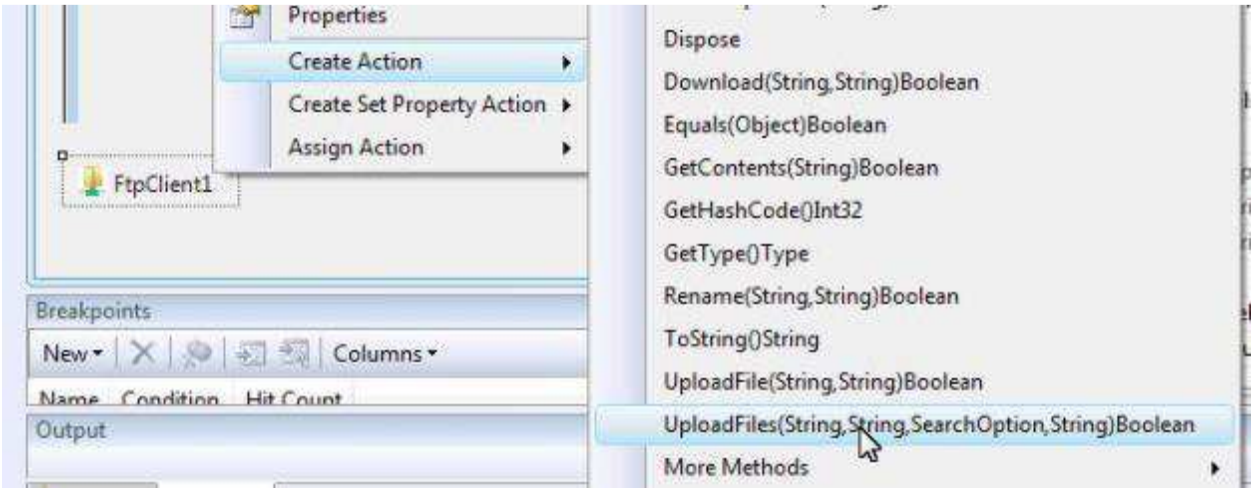
Set UserName and Password properties

The FtpClient component has two methods for uploading files:

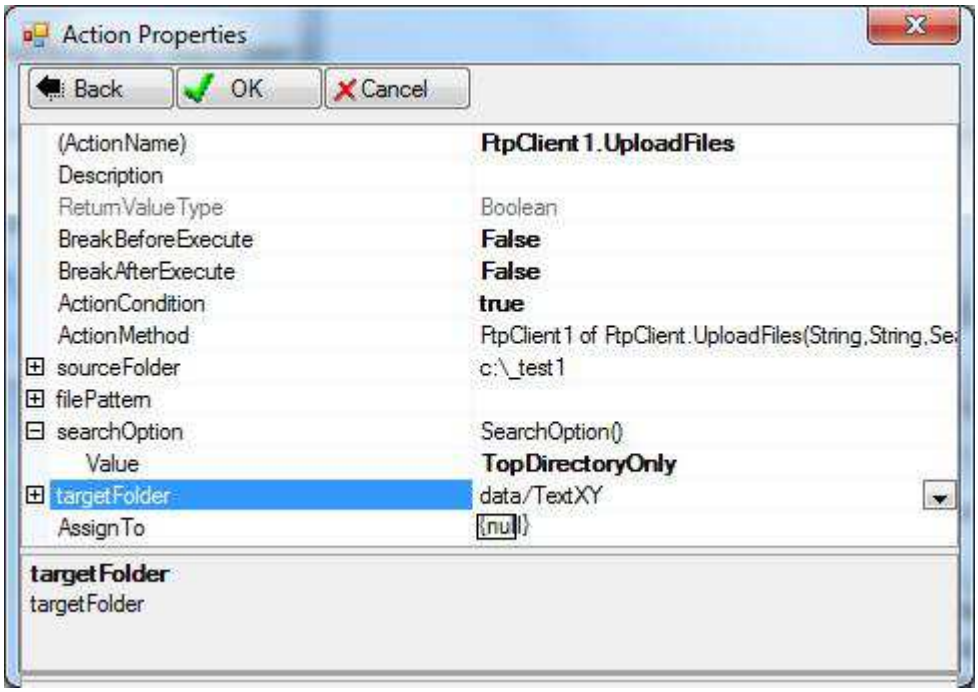
UploadFile – it uploads one file

UploadFiles – it uploads many files in one action.

Right-click FtpClient1; choose “Create Action”. Choose “UploadFiles”:



We may specify properties for the action:



This action has 4 parameters.

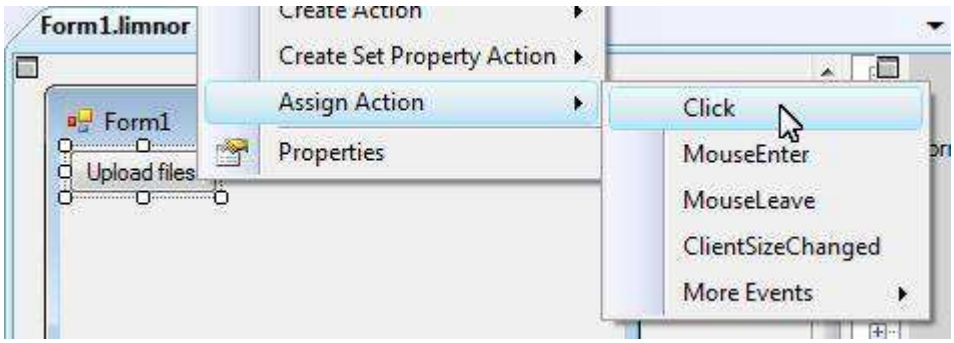
sourceFolder – It is the local folder where the action finds the files to be uploaded.

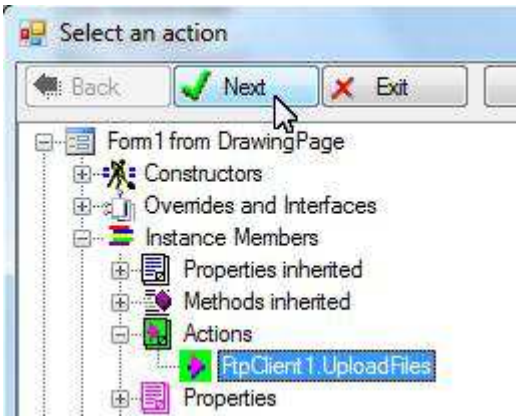
filePattern – it is the file search pattern. Empty filePattern indicates all files, same as *.*.

searchOption – it indicates whether the files in sub-folders should be uploaded.

targetFolder – it indicates the folder on the FTP server to save uploaded files.

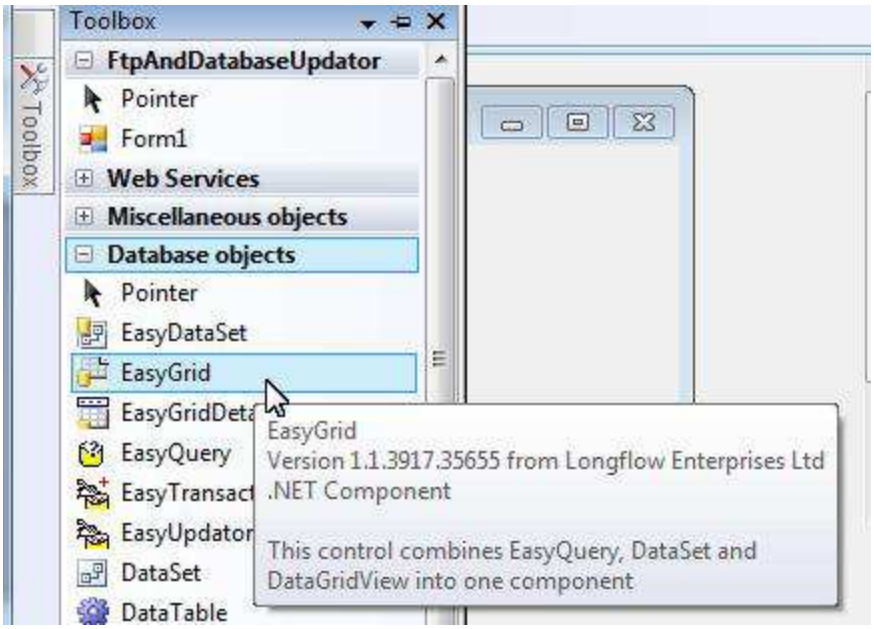
Assign this action to a button:



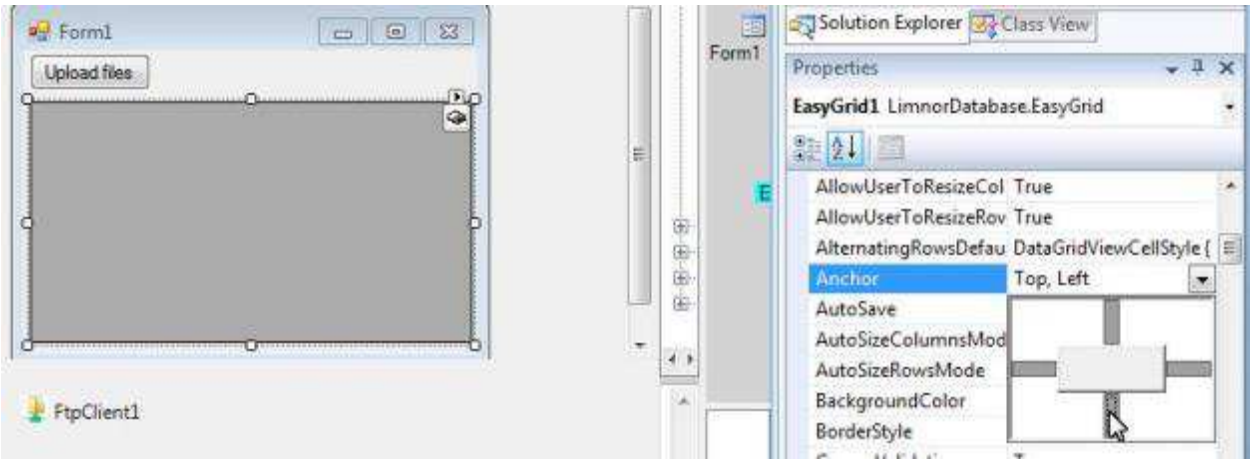


10.2 Record Activities in Database

We use a database to record FTP activities. To show the database data, create an EasyGrid component. Later we will use an EasyUpdator component to save FTP activities into the database.

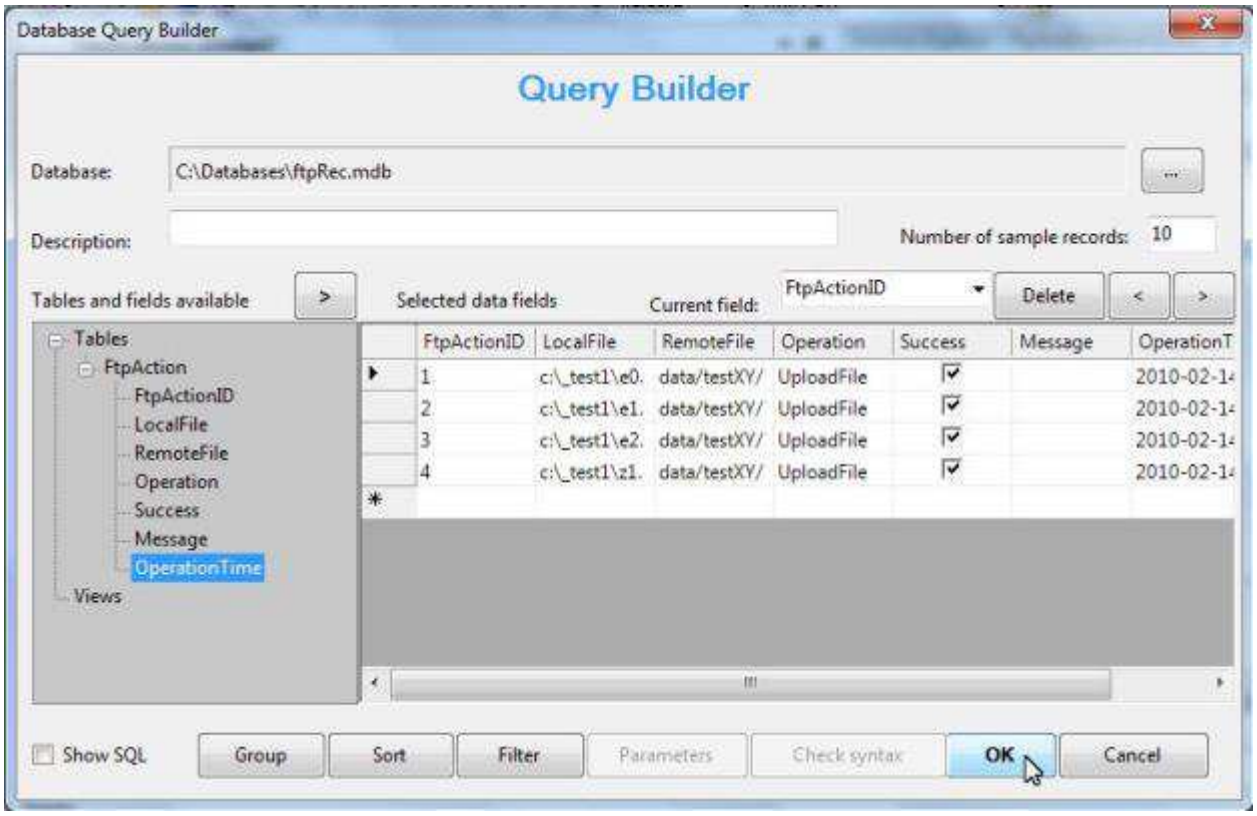


Set its Anchor property to make it resize with the form:

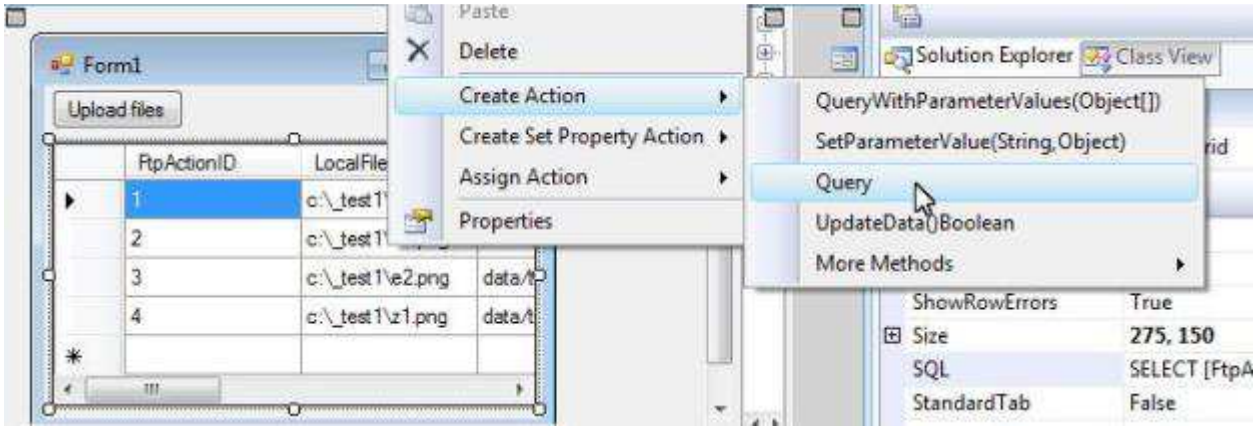


Its ForReadOnly is True because we do not want to allow entering data manually:

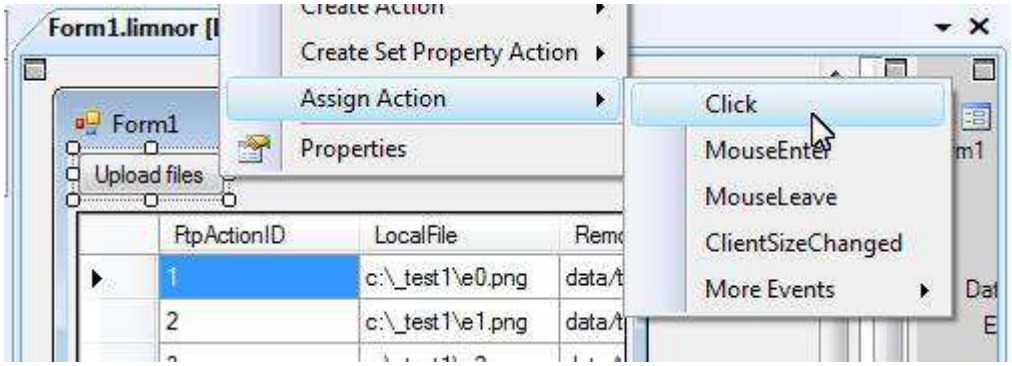
Set its DatabaseConnection property to connect to a database. Set its SQL property to create a query to display data:

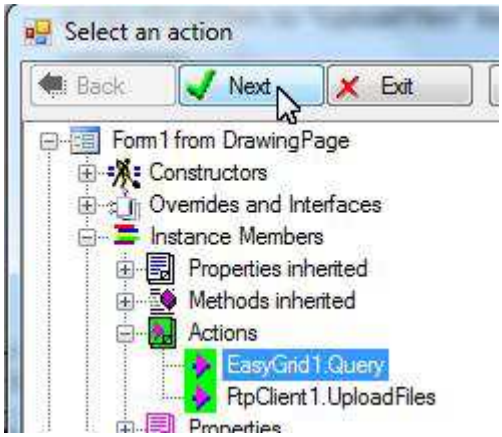


Create a Query action to re-load data from the database. This action can be executed after FTP operations.



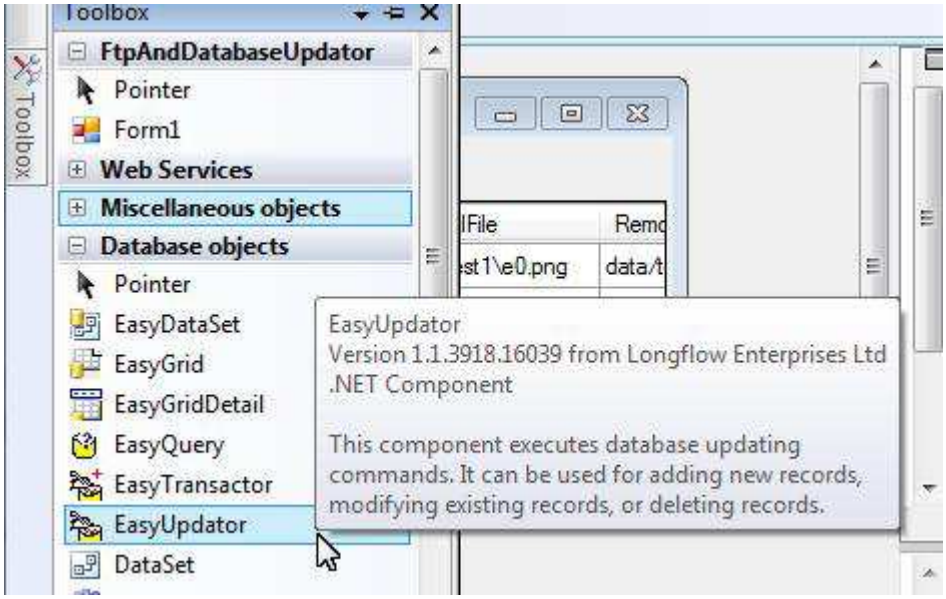
Assign this action to “Upload files” button so that the EasyGrid reloads data from database after file uploading:



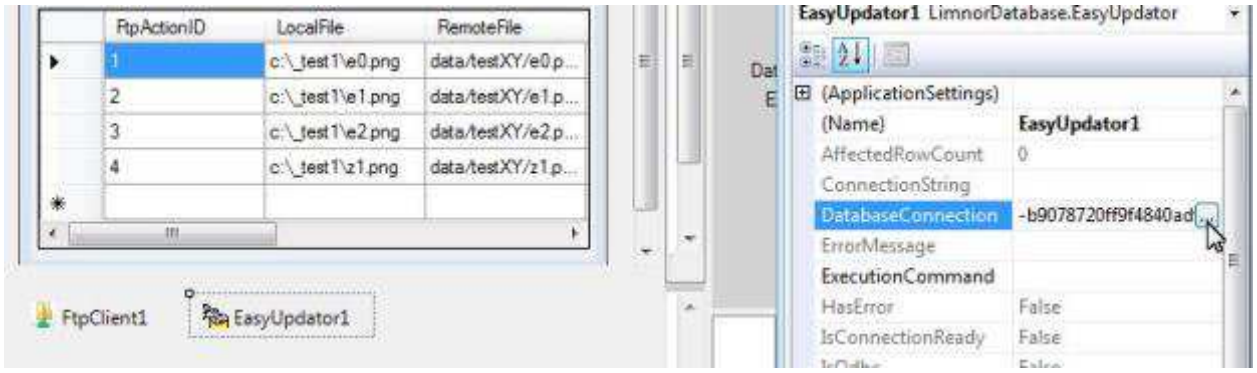


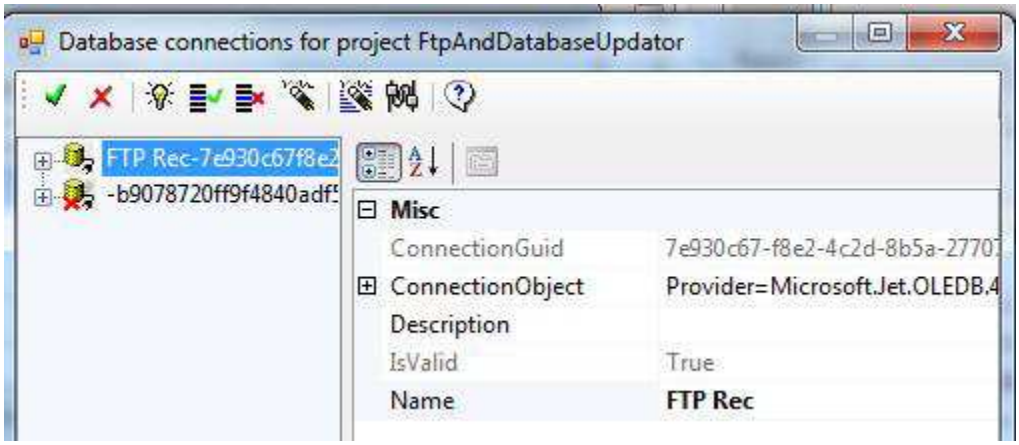
10.3 Use EasyUpdator to record FTP activities

Create an EasyUpdator:



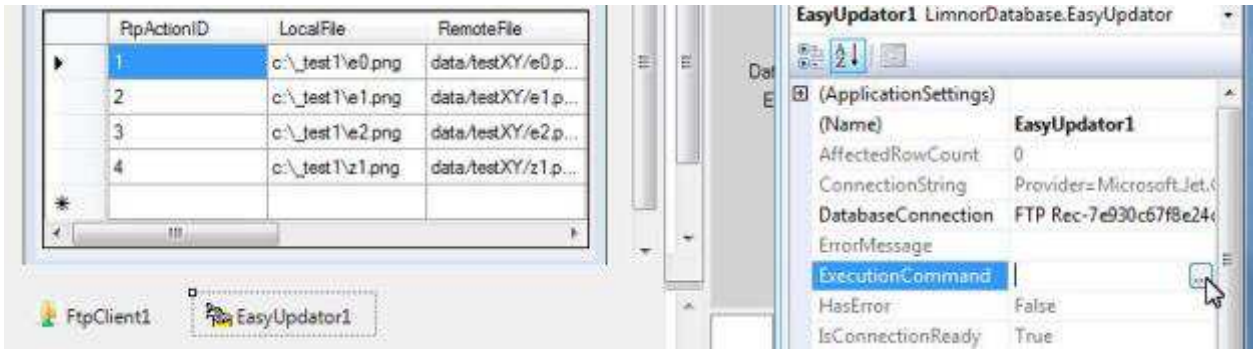
Set its DatabaseConnection property to use the same database connection the EasyGrid uses:



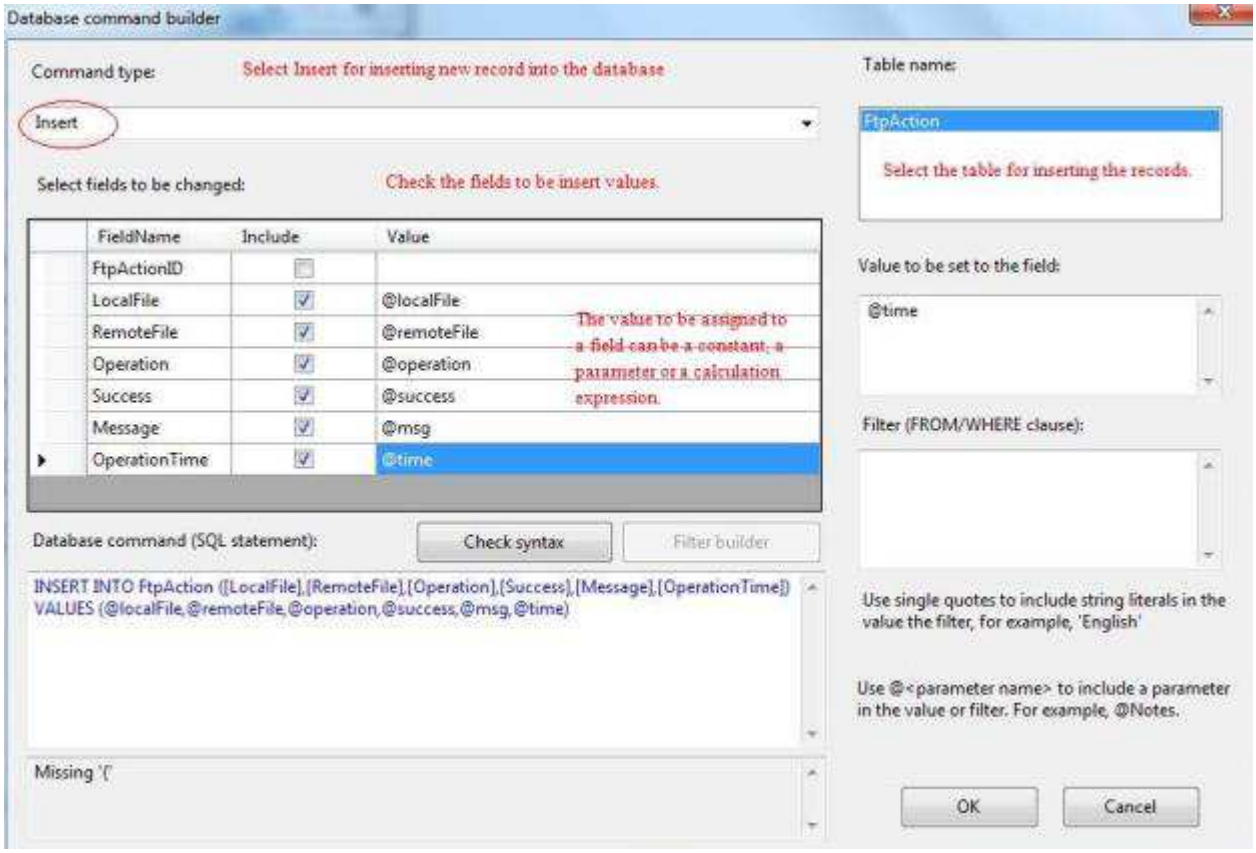


10.4 Create Database Command

Set ExecutionCommand property of EasyUpdator1 to build database command to insert record into the database:



A Database Command Builder dialogue box appears.

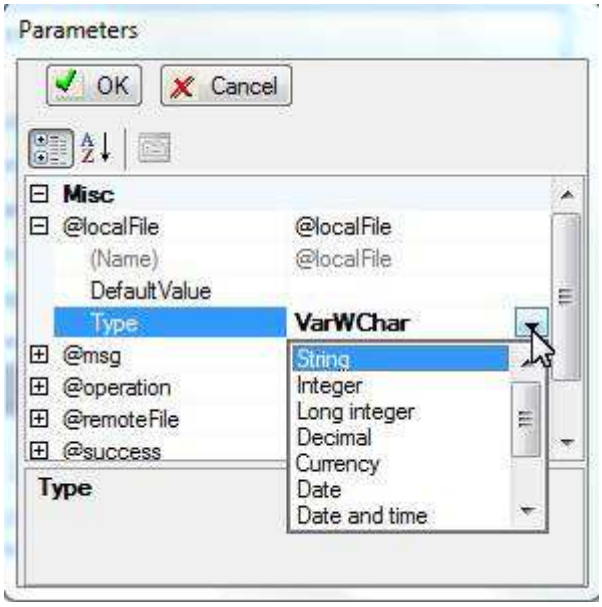


To use the Database Command Builder, following the steps below

1. Select the table to be updated. In this sample, we have only one table, FtpAction
2. Select the Command type. It can be one of the following commands
    a. Insert – for inserting a new record into the table
    b. Update – for modifying field values of an existing record
    c. Delete – for deleting an existing record
3. Check the fields to be modified
4. Give value to each field to be modified. The value can be a constant, a parameter, or an expression. A parameter is formed by @ followed by parameter name.

In this sample, all field values are given by parameters.

Click OK. All parameters are listed for you to check parameter data types. The Command Builder tries to give correct data types to the parameters. You may review them and see if modifications are needed.

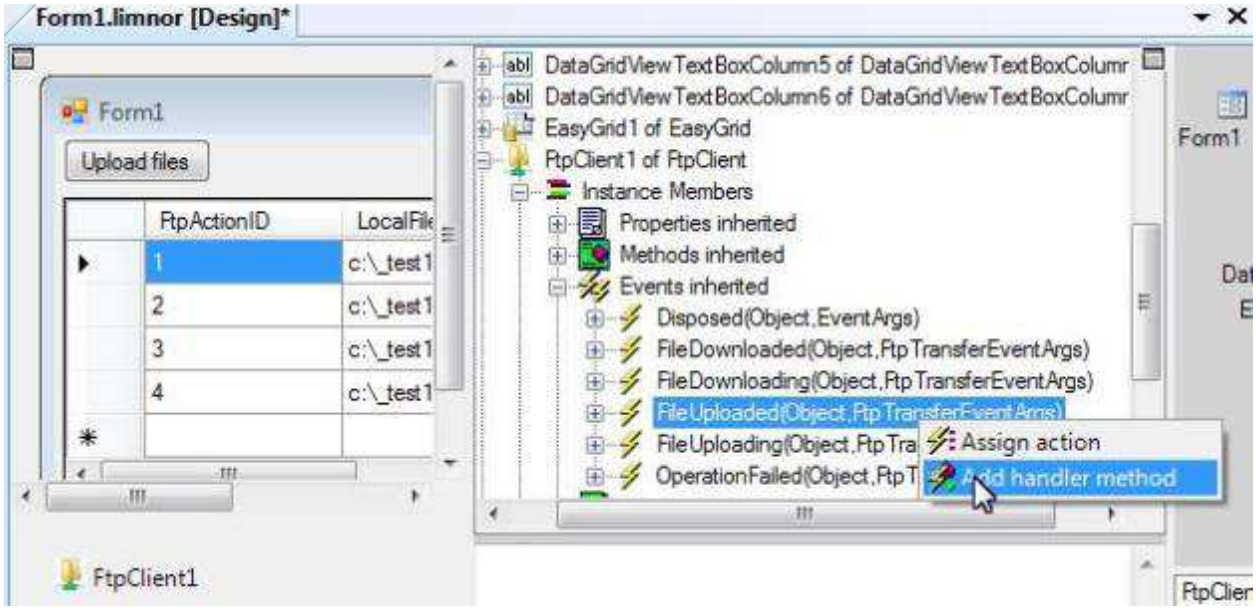


10.5 Handle Activity Events

FtpClient uses events to report its activities. We may handle these events to save the activities to the database.

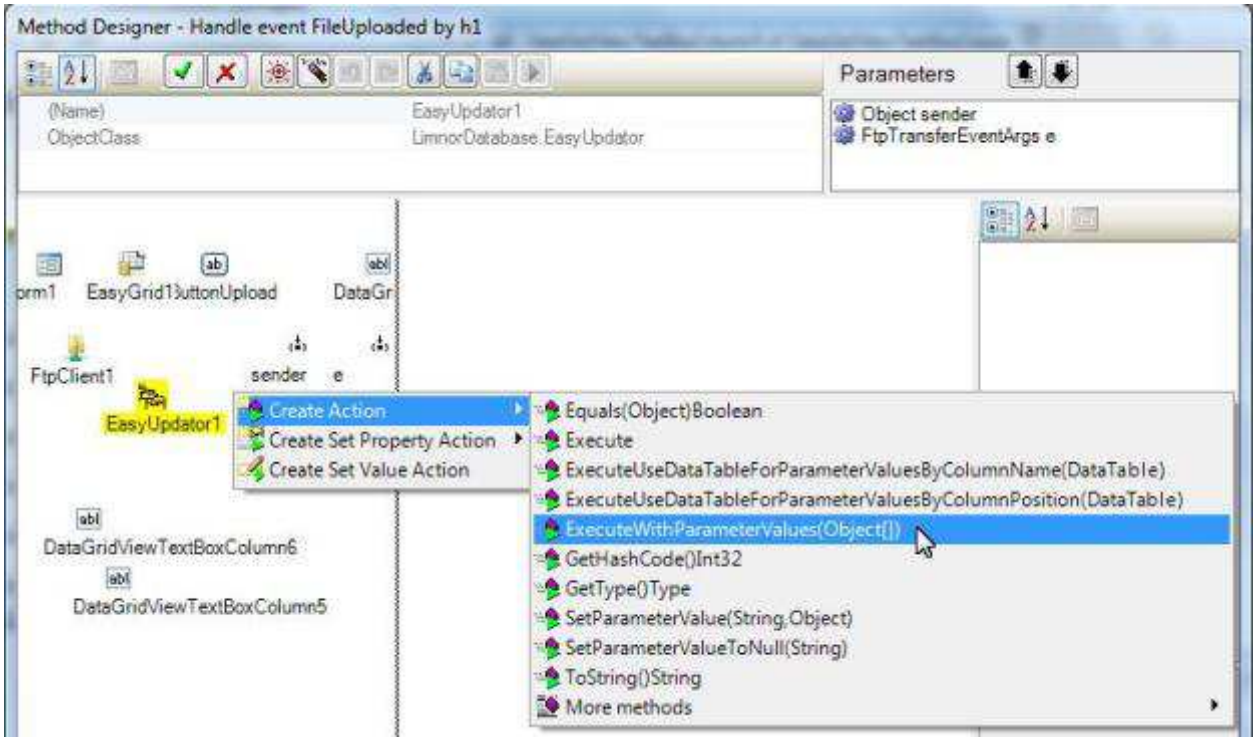
10.5.1 Handle FileUploaded event

To handle FileUploaded event, right-click it, choose "Add handler method".

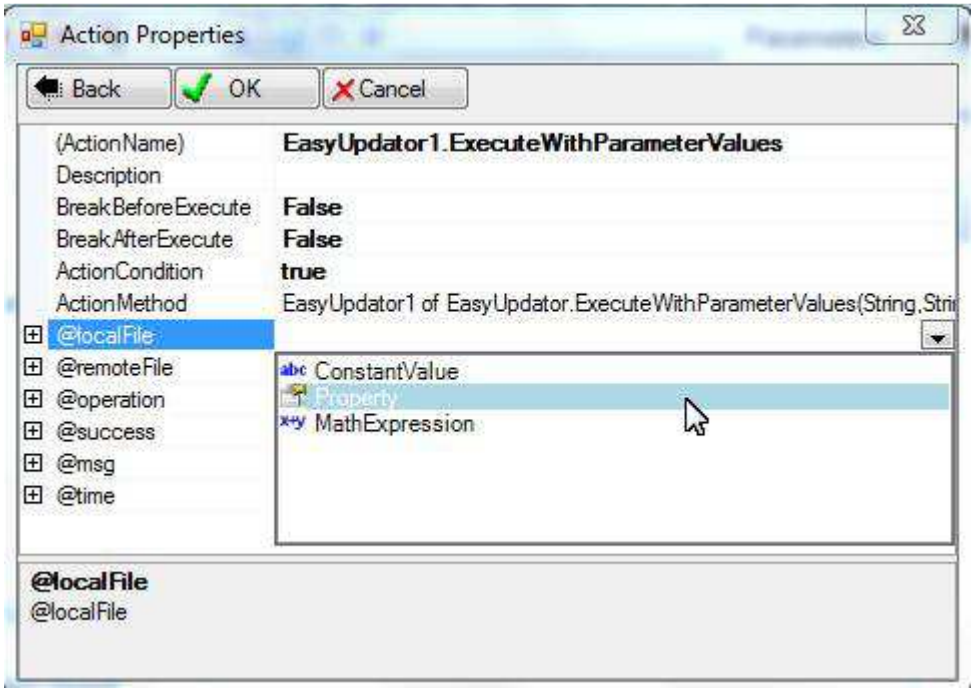


The Method Editor appears for editing the event handler. We want to insert a new record into the database to record this FTP activity. We have setup EasyUpdator1 to insert a new record into the database. We may use it to create an action to do it.

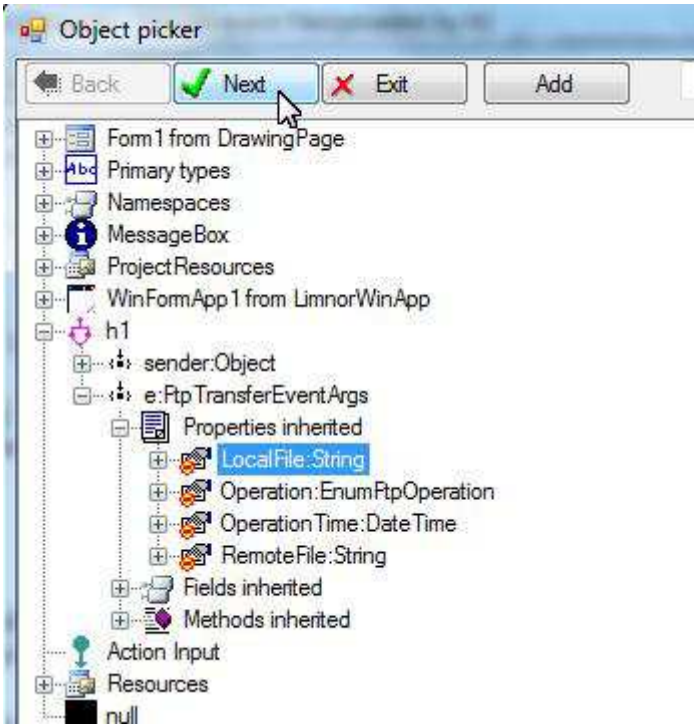
Right-click EasyUpdator1; choose “Create Action”. Choose “ExecuteWithParameterValues”



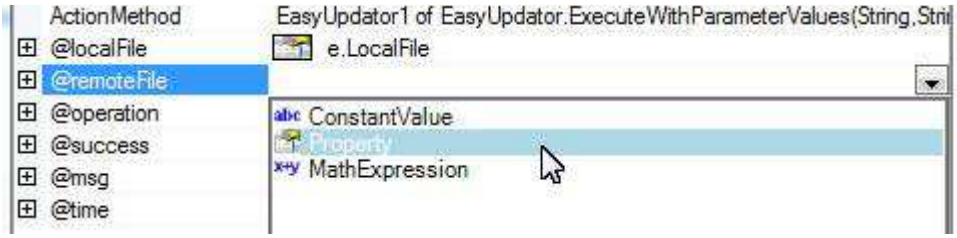
Select “Property” for @localFile parameter:



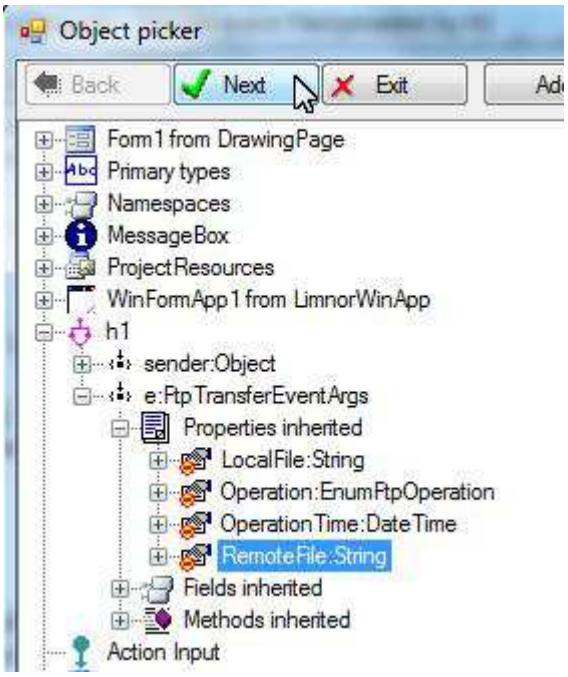
Select LocalFile property from the event parameter:



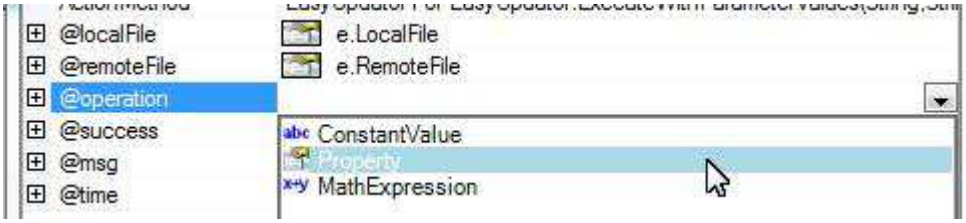
Also select Property for @remoteFile:



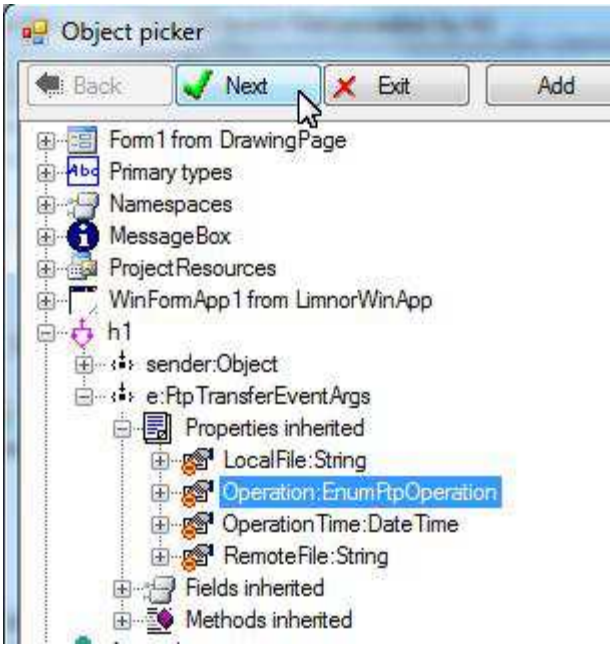
Select RemoteFile property from the event parameter:



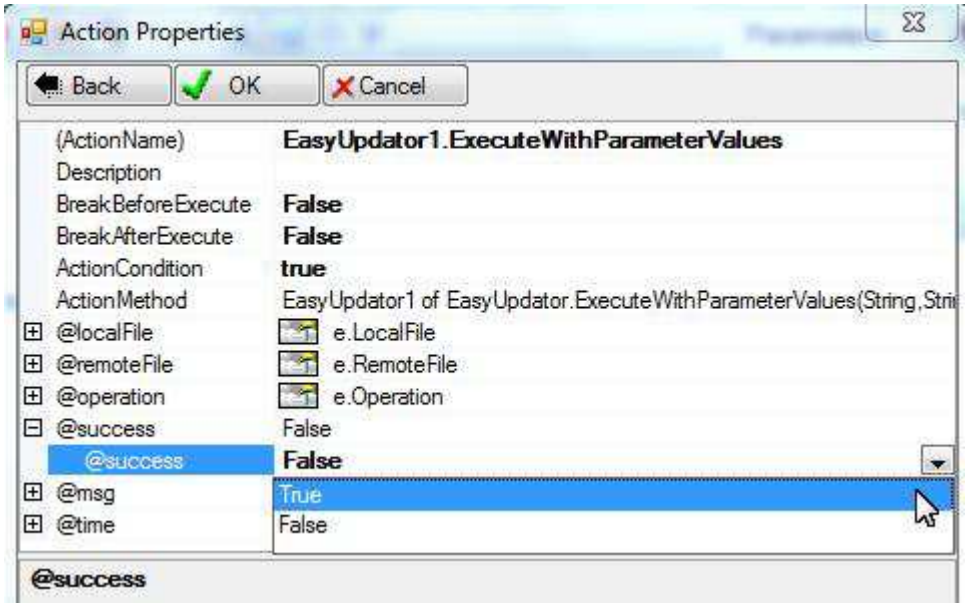
Also select Property for @operation:



Select Operation property of the event parameter:

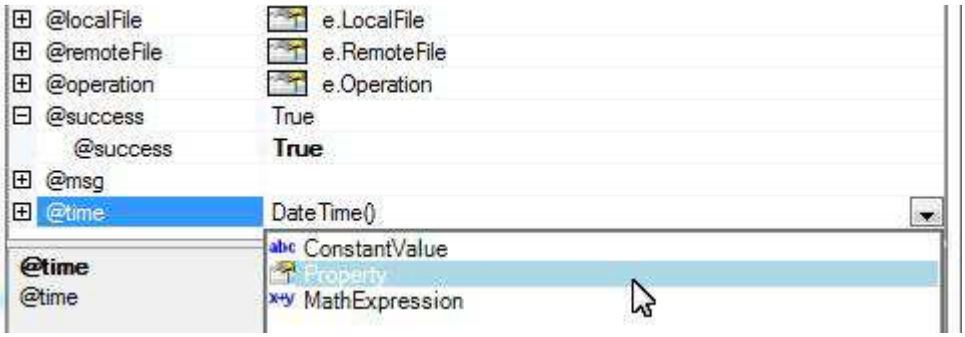


For parameter @success, choose constant True:

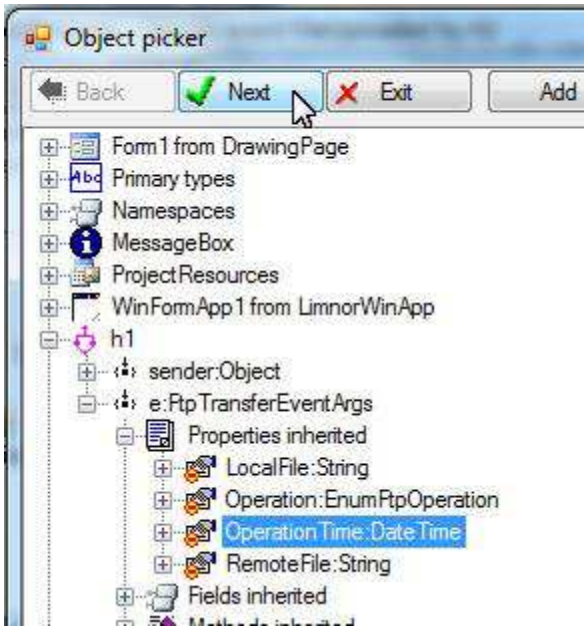


We leave @msg parameter blank because we do not need to record message for a successful FTP operation.

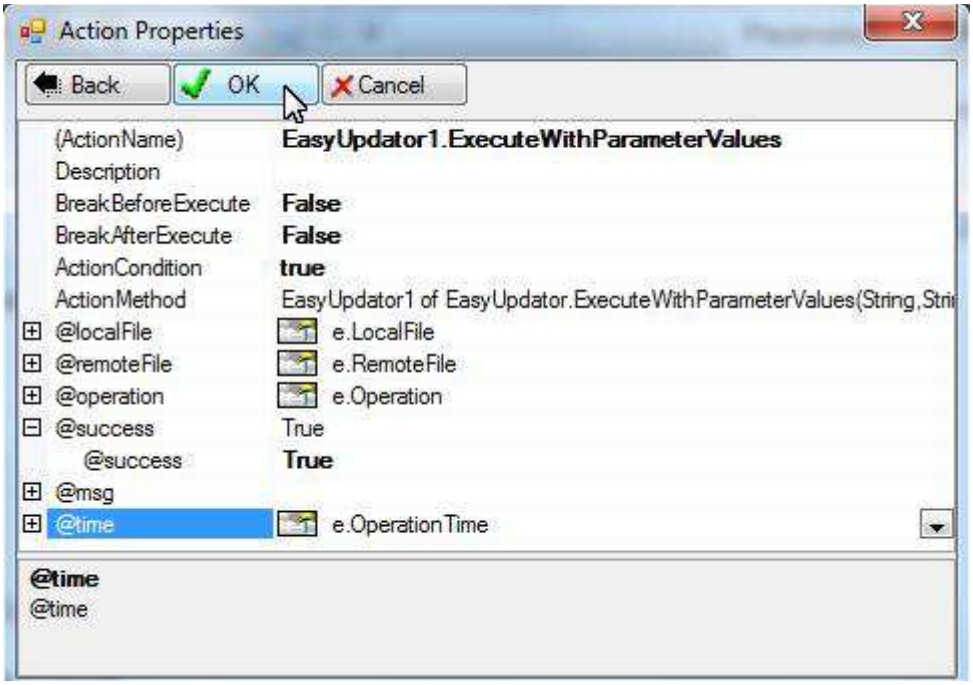
Select Property for @time:



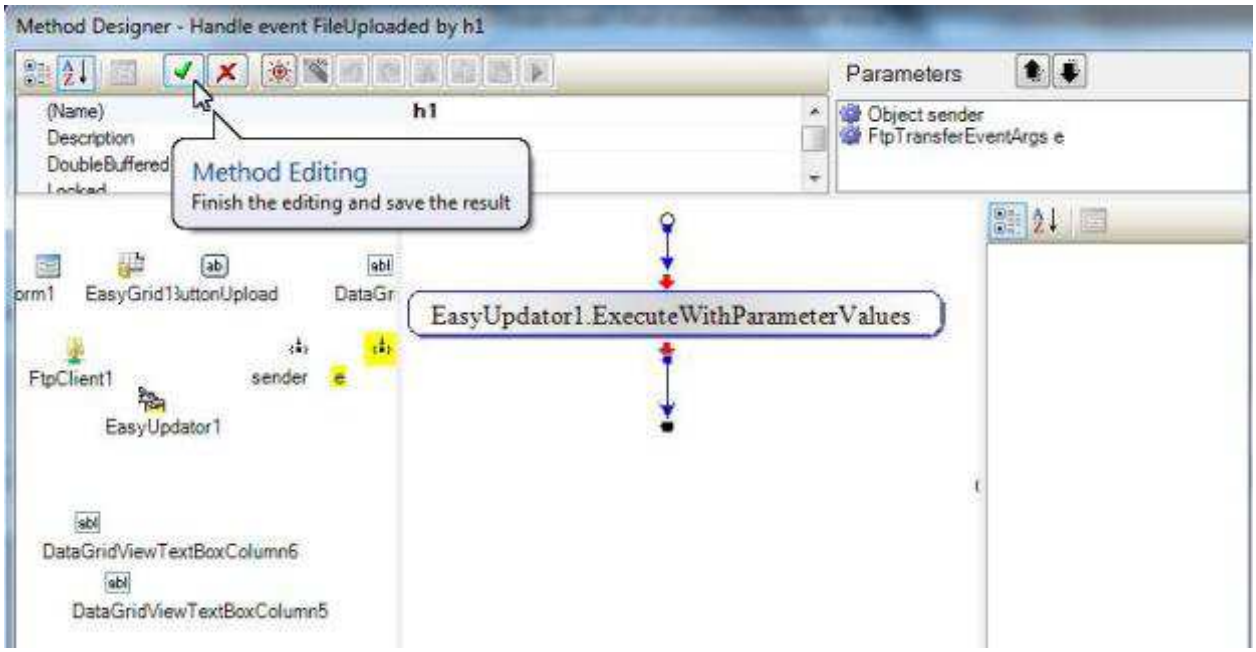
Select OperationTime property of the event parameter:



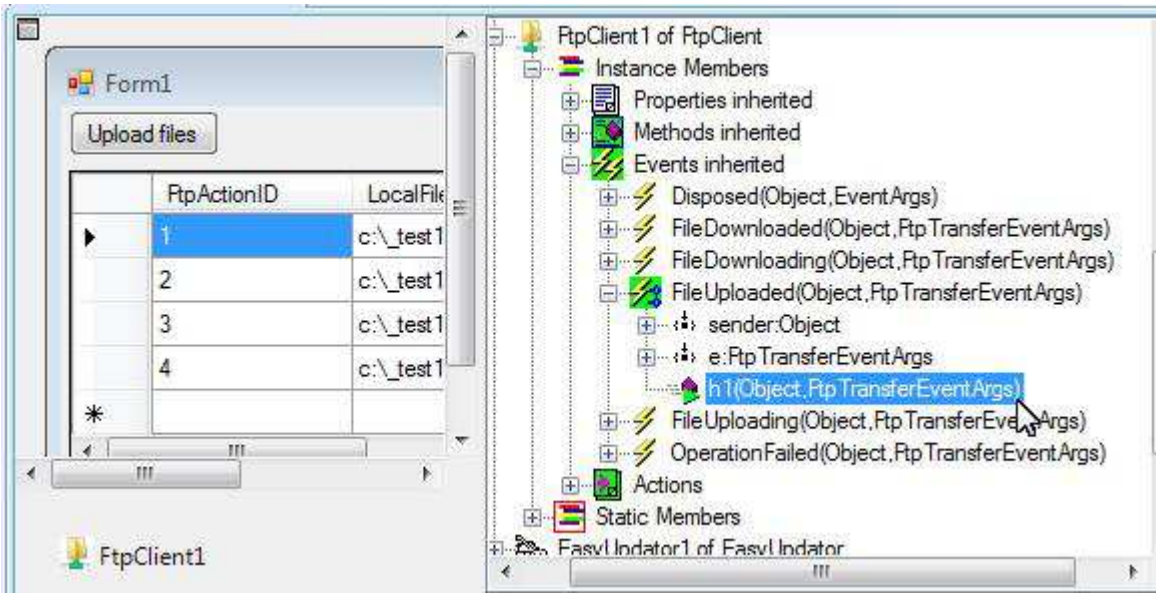
This action is created:



That is all what we needed for handling this event:

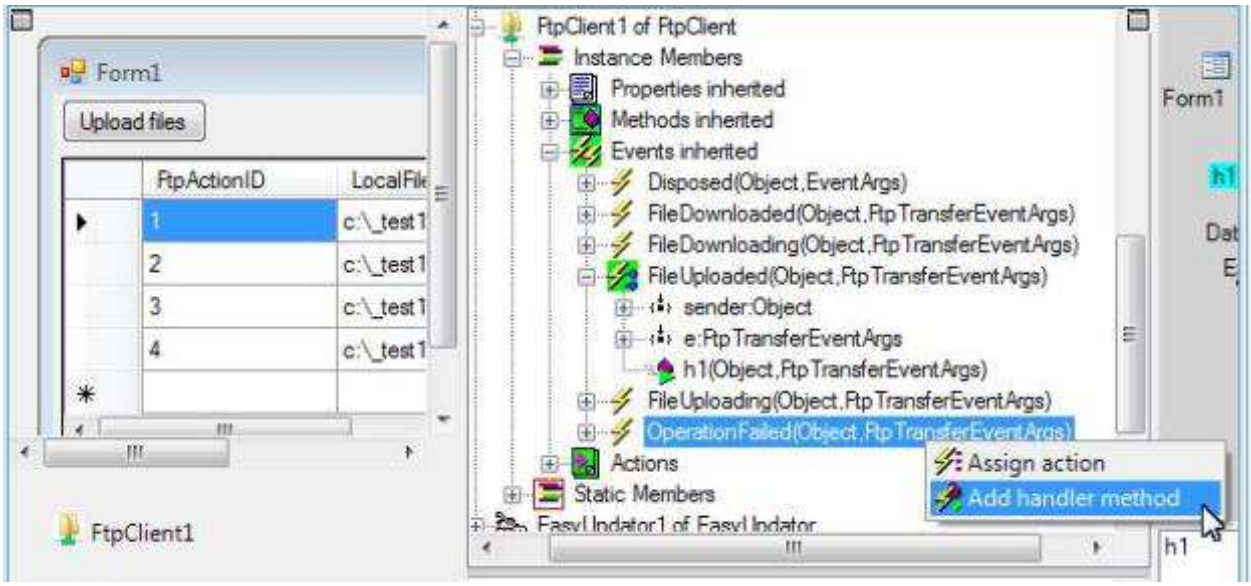


The event handler appears under the event:

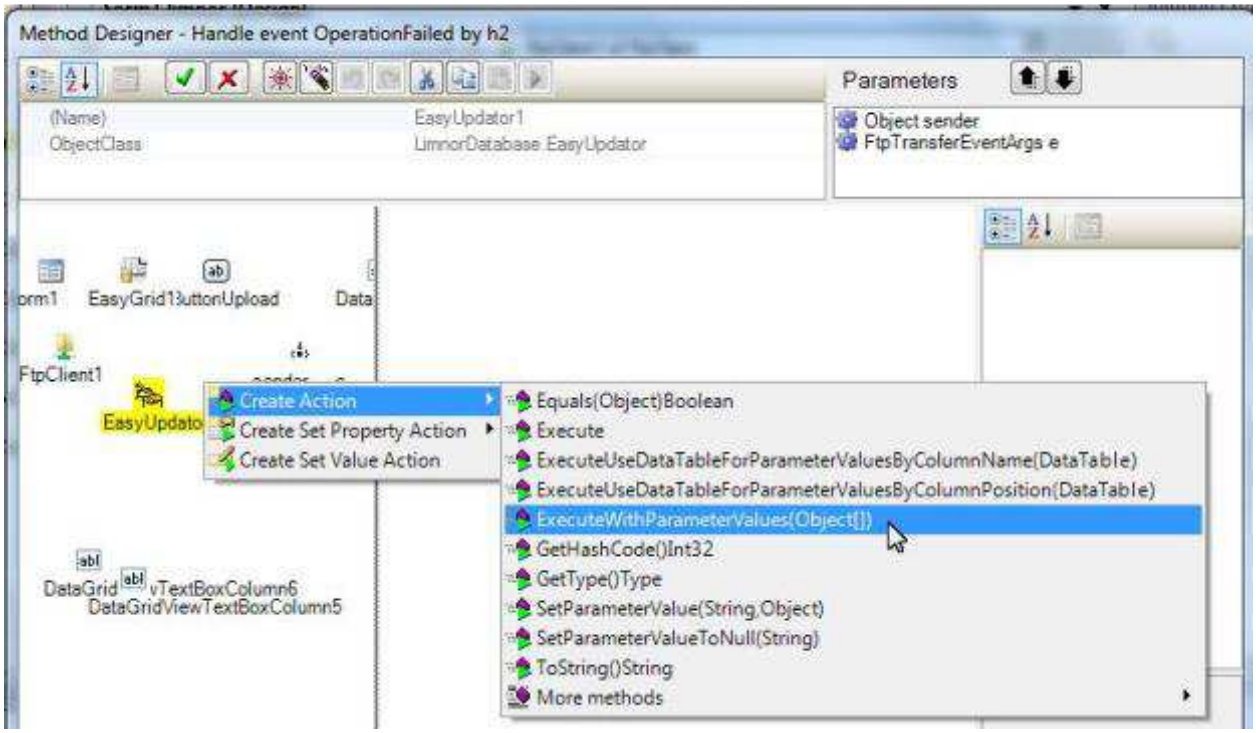


10.5.2 Handle OperationFailed event

We may handle event OperationFailed to create a database record to log the failure of the operation.



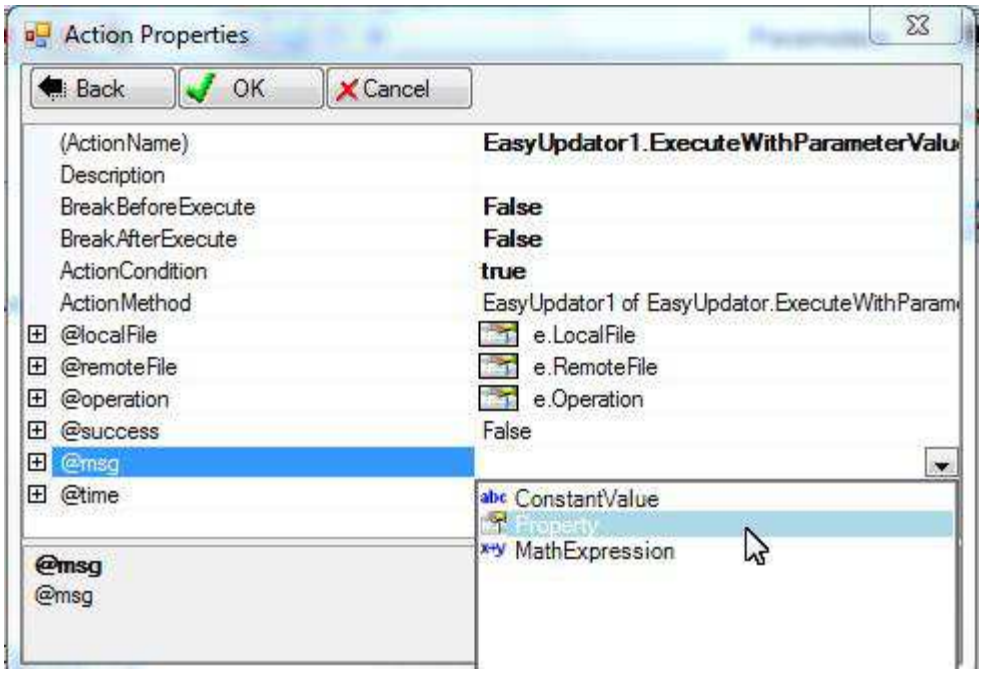
The Method Editor appears for editing the event handler. We also create an ExecuteWithParameterValues action by EasyUpdator1:



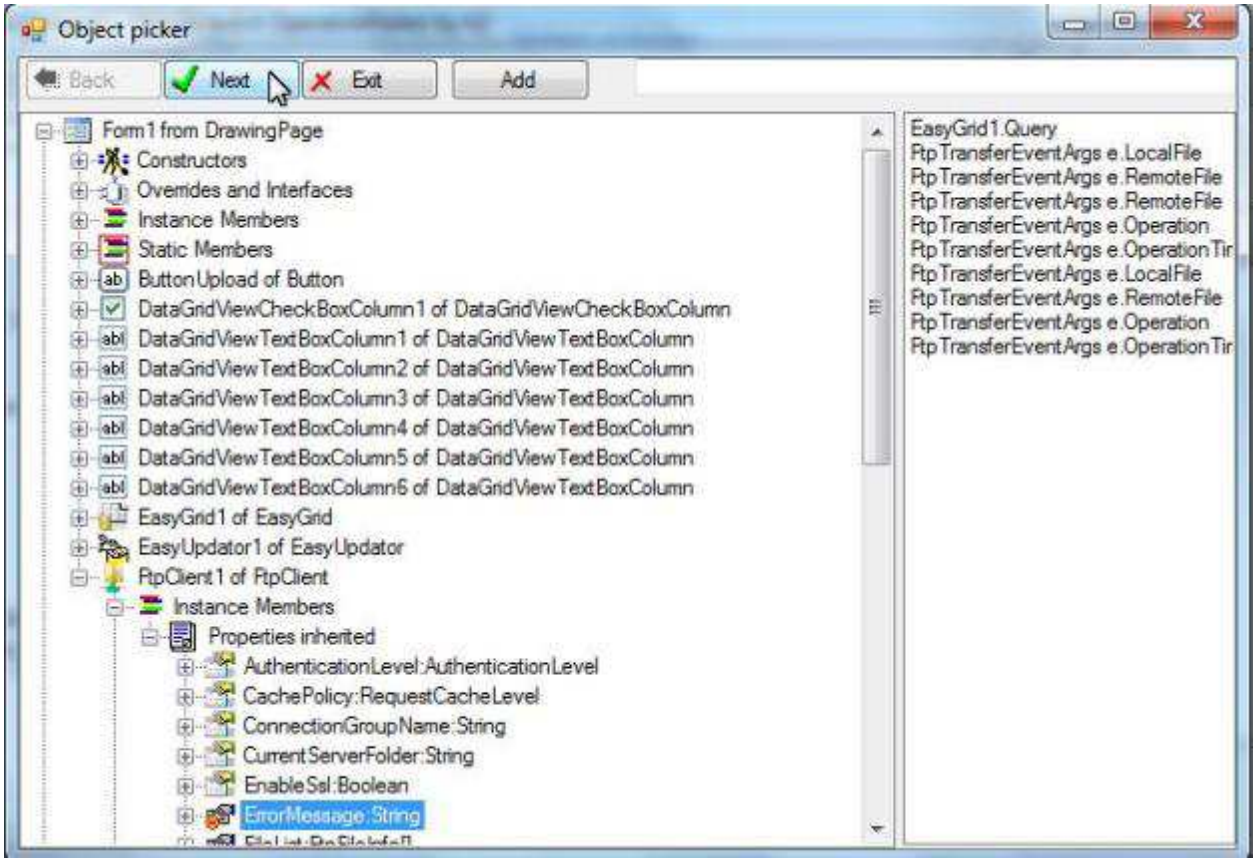
Most action parameters are given in the same way as we did before. Two parameters are given differently.

Parameter @success is given constant False instead of True.

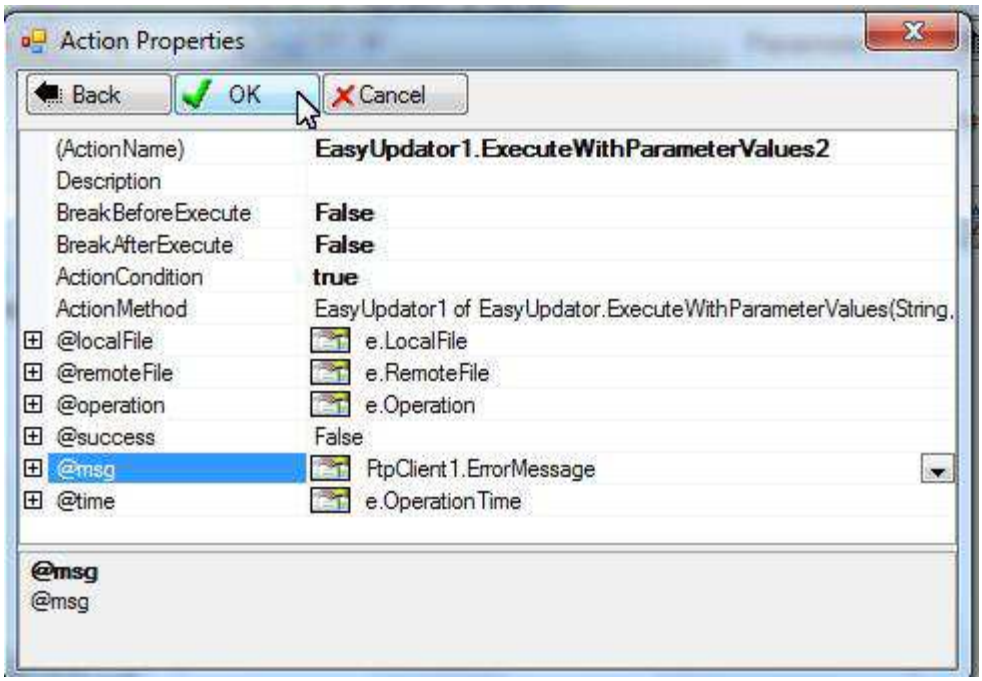
Parameter @msg is not blank. Select Property for it:



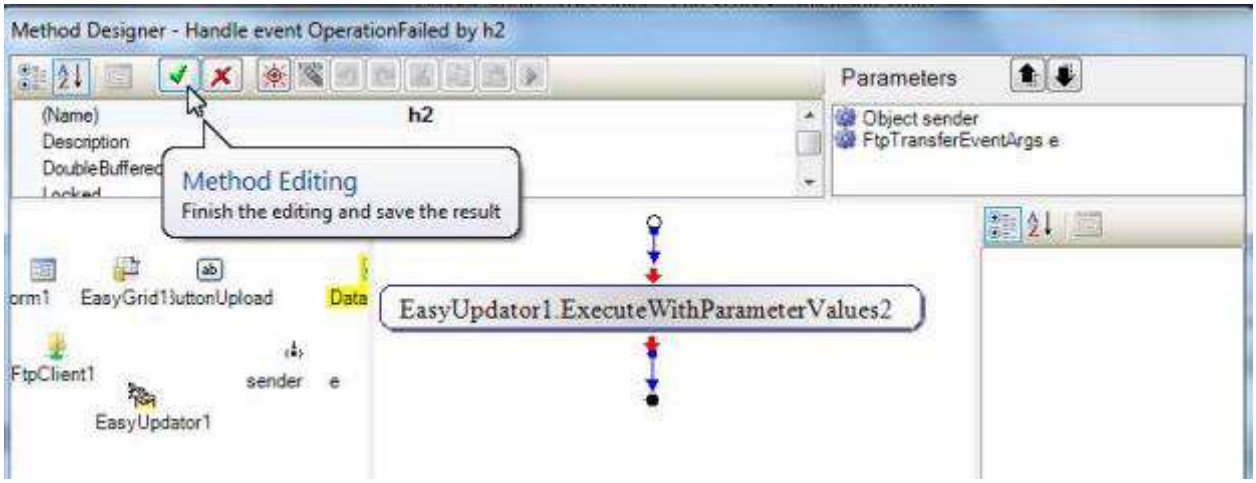
Select ErrorMessage property of FtpClient1 for @msg:



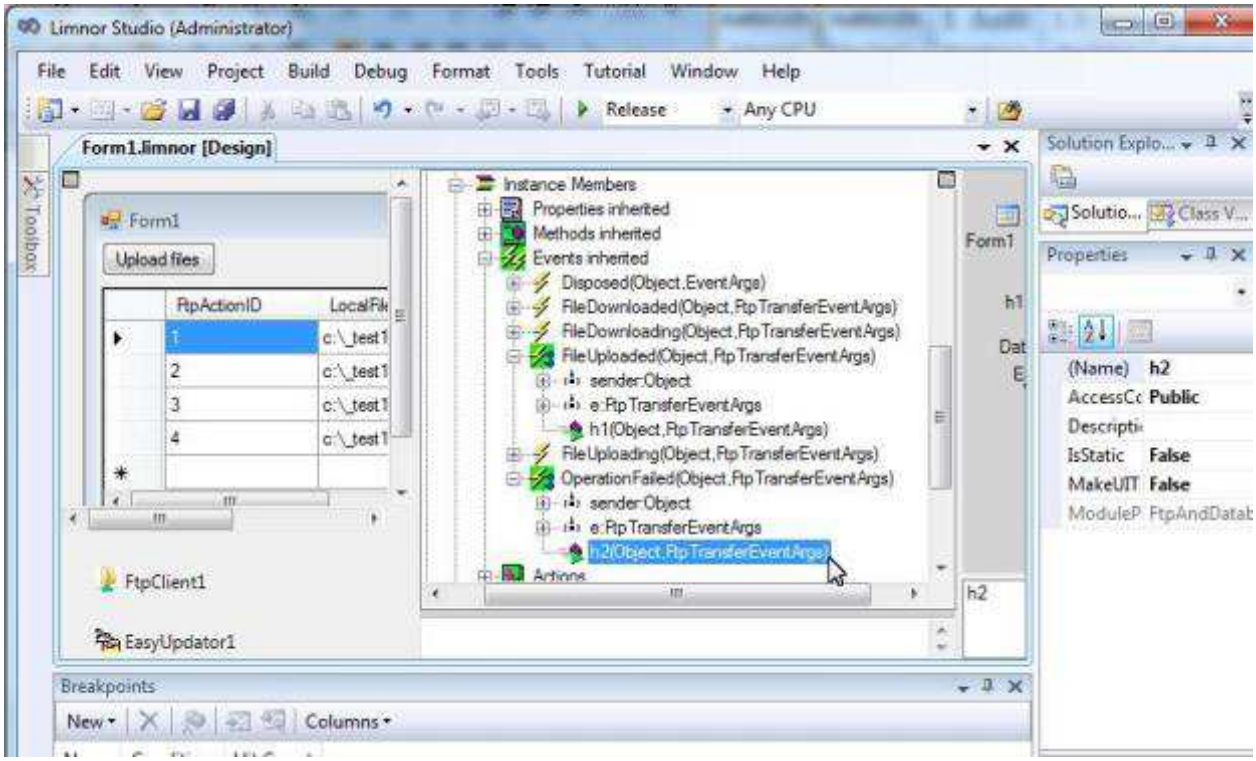
This is the action for inserting a record into the database to record a failed FTP uploading:



That is all we need for handling this event



The event handler appears under the event:



11 Modify Database in Transactions

11.1 What is a transaction

Suppose an application moves 100$ from a saving account and put it into a checking account. It involves two actions:

1. Reduces the saving account by 100$.
2. Increases the checking account by 100$.

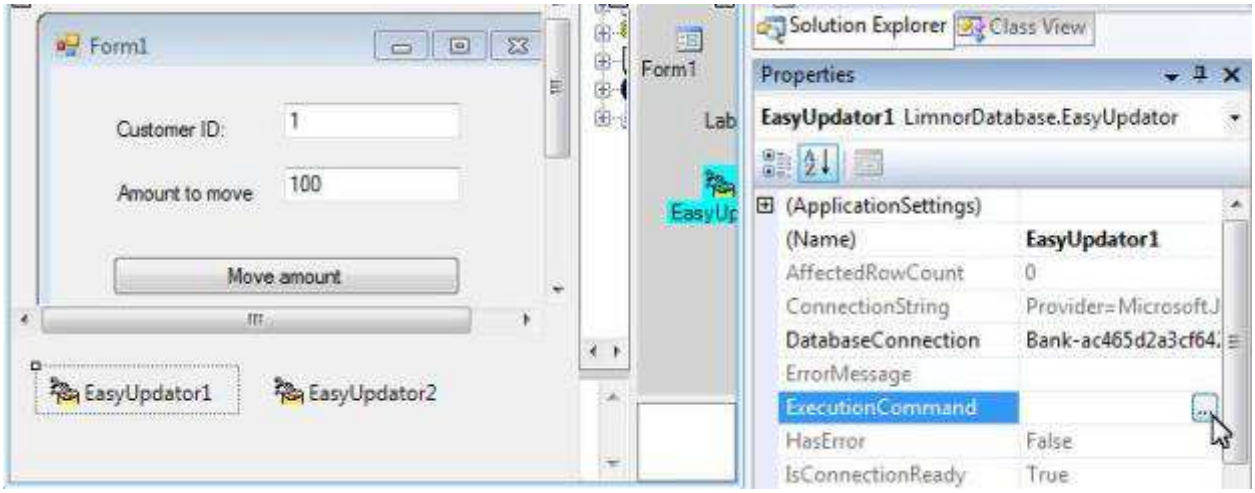
If the first action succeeds and the second action fails, due to network failure, power down, etc., then the customer loses 100$.

Many database engines provide transaction support to prevent such situation. Put the above two actions into one transaction. The database modifies the saving account and checking account only when both actions succeed. If any operations fail then all other operations are discarded.

EasyTransactor component allows putting database modifications into transactions.

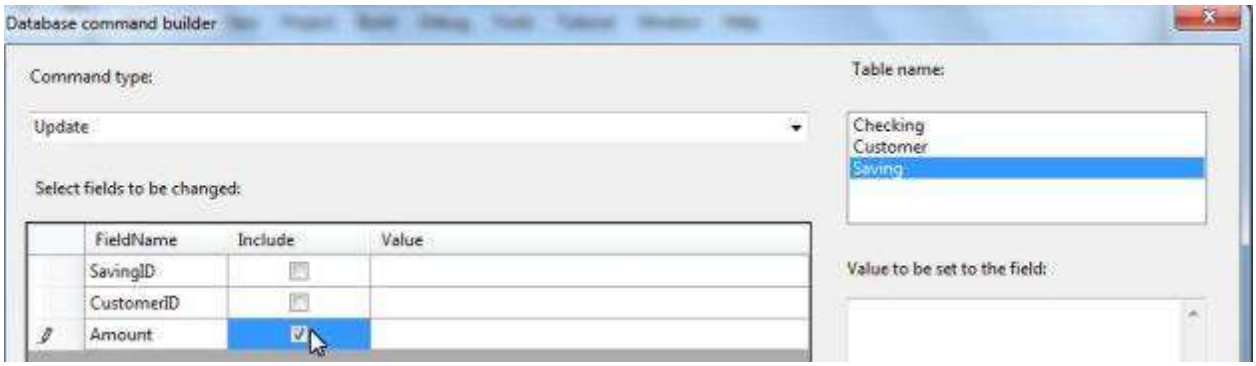
11.2 Create Database Update Executers

We use two EasyUpdator components to create two database update commands. After setting their DatabaseConnection property to connect to a database, we may set their ExecutionCommand property to create database update command:

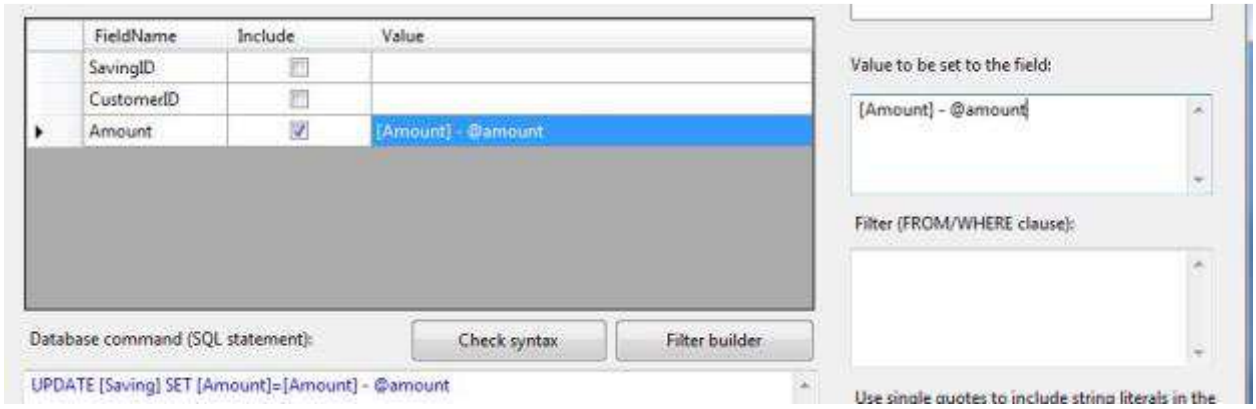


Let's create a command to reduce "Saving" amount.

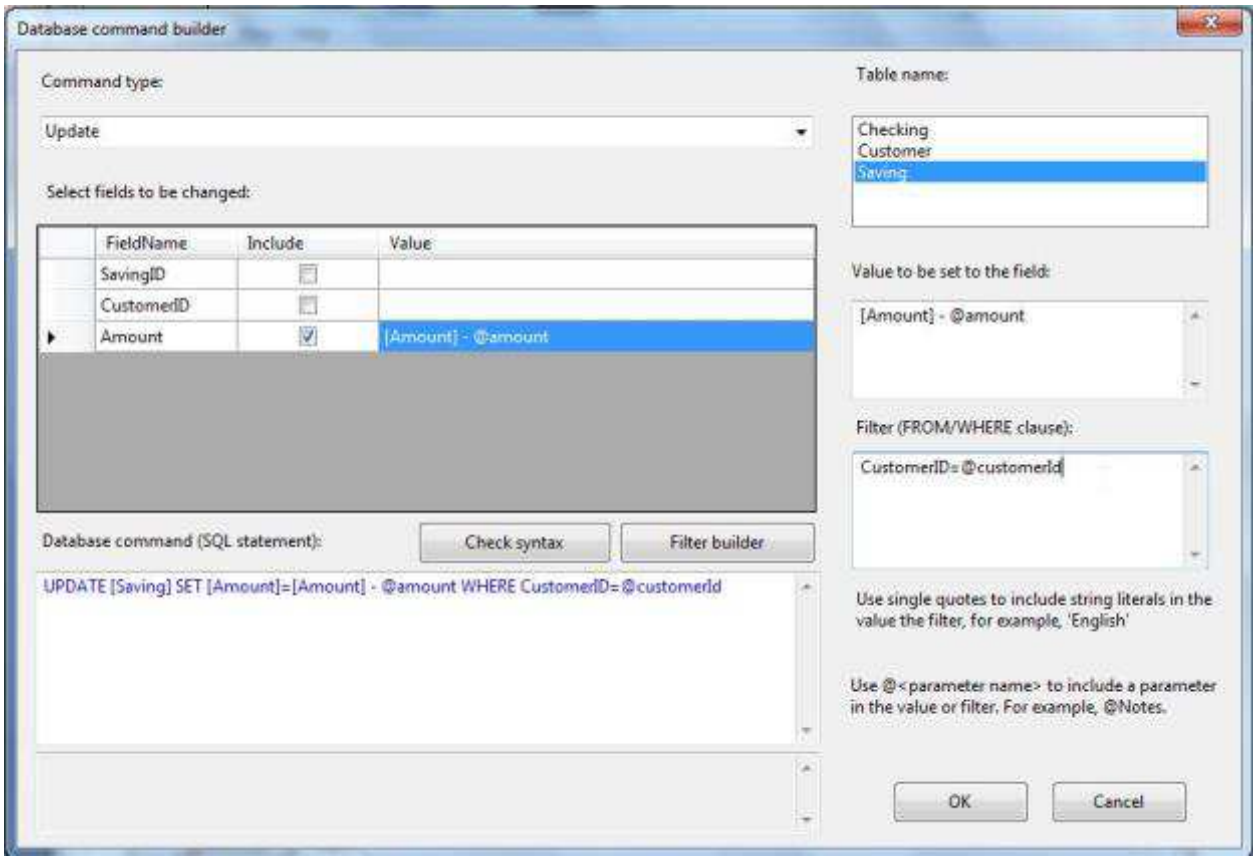
Select "Saving" table; select "Update" command; check Amount field:



Type [Amount] - @amount for the Value of the field:

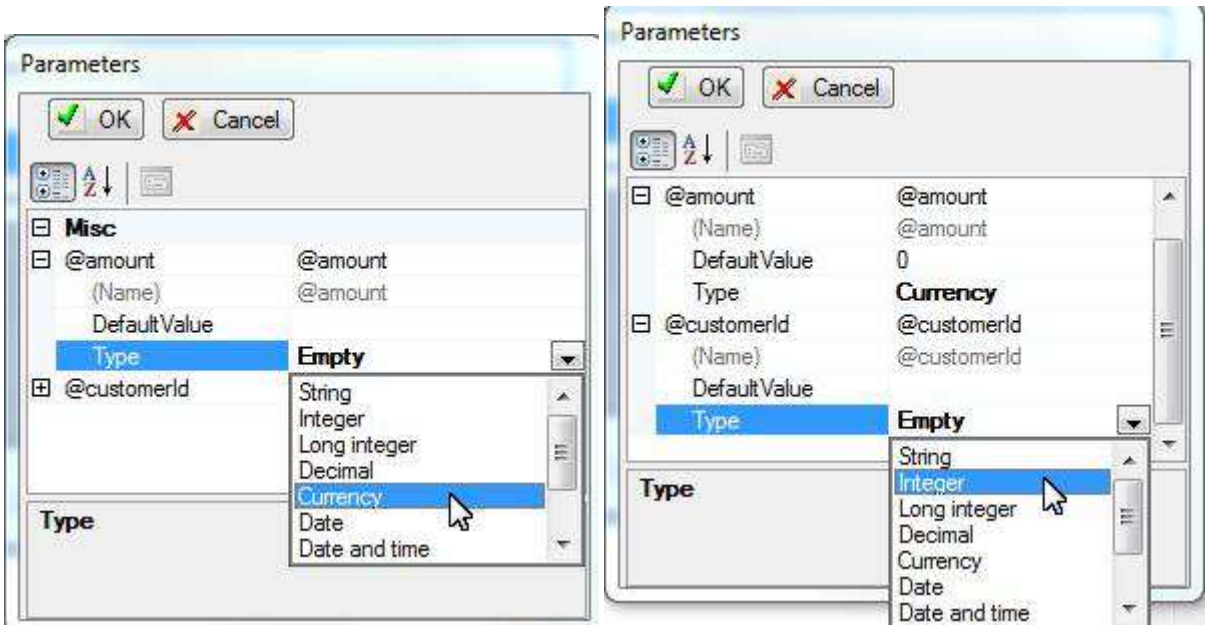


Type CustomerID=@customerId into the Filter:

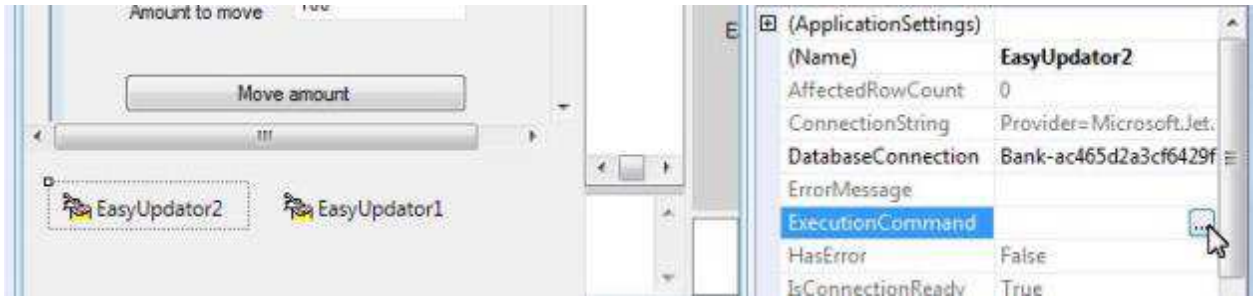


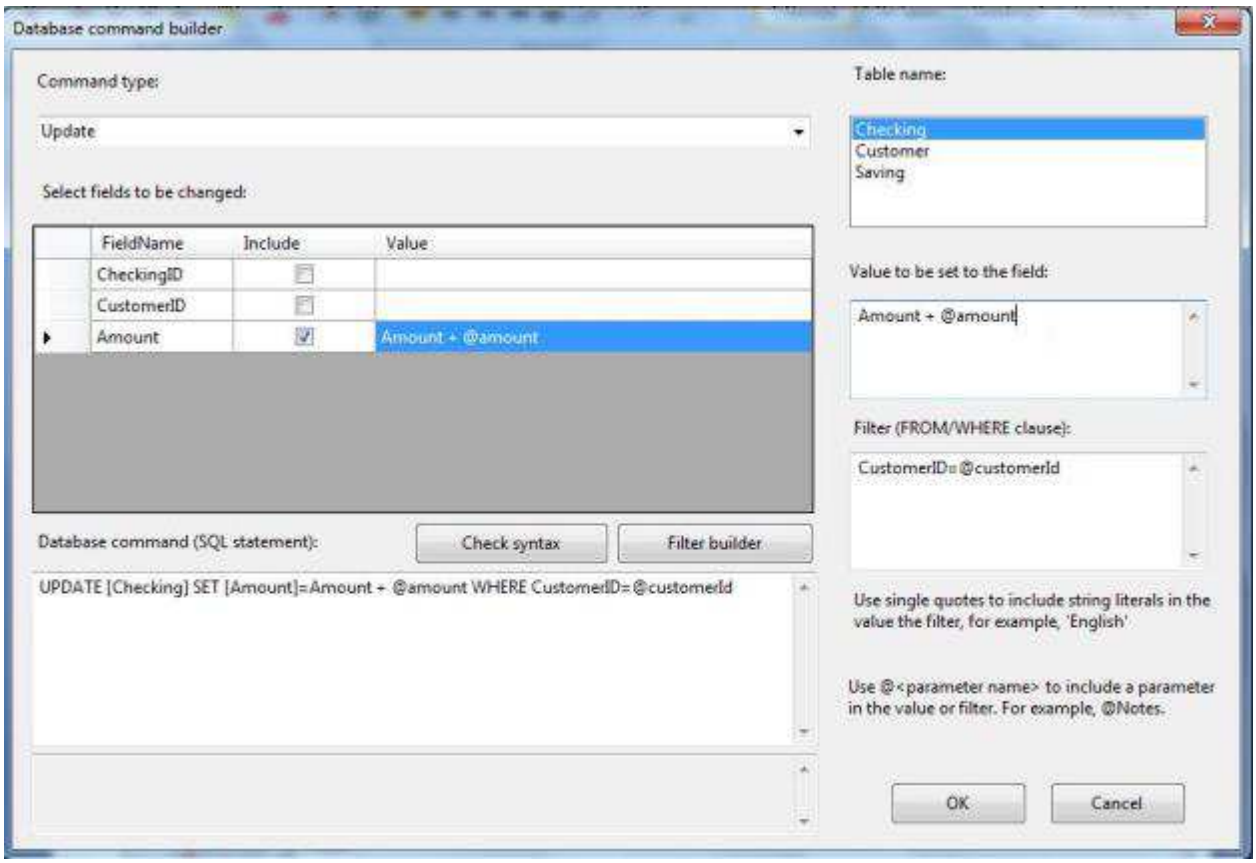
"Filter builder" button may also be used to build filter.

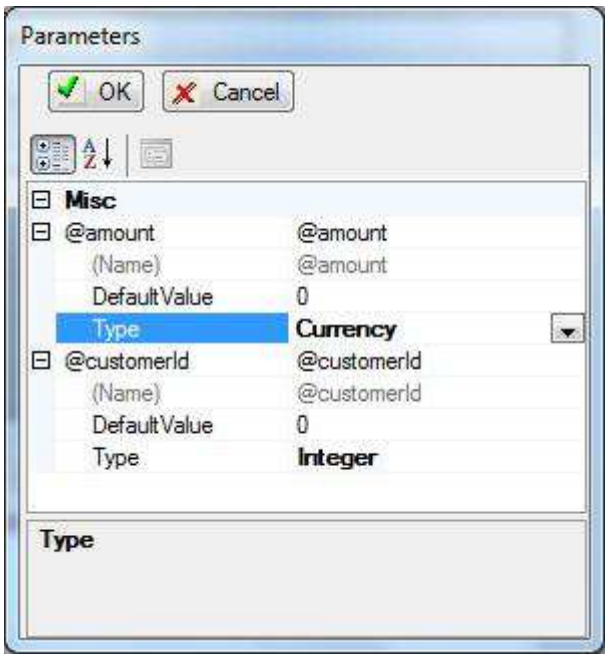
Click OK to finish creating the command. Set the data types for parameters.



Create another command to increase Checking account:

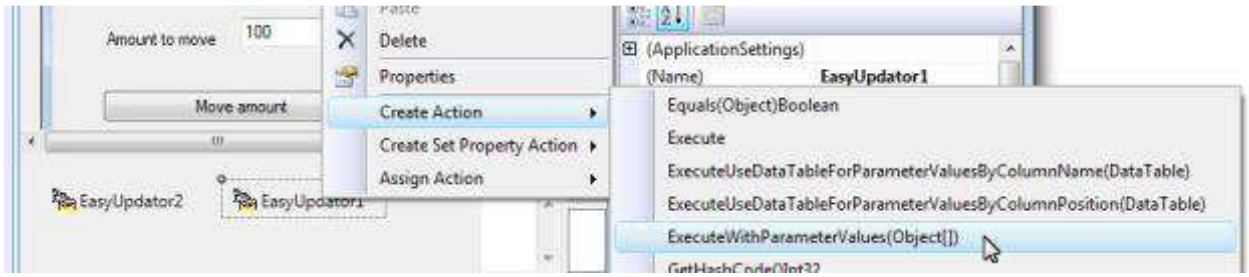




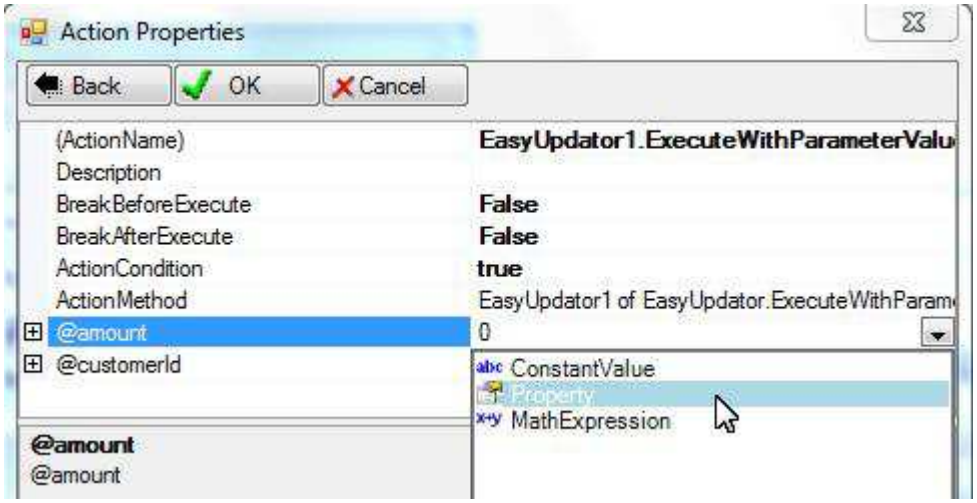


11.3 Create Database Update Actions

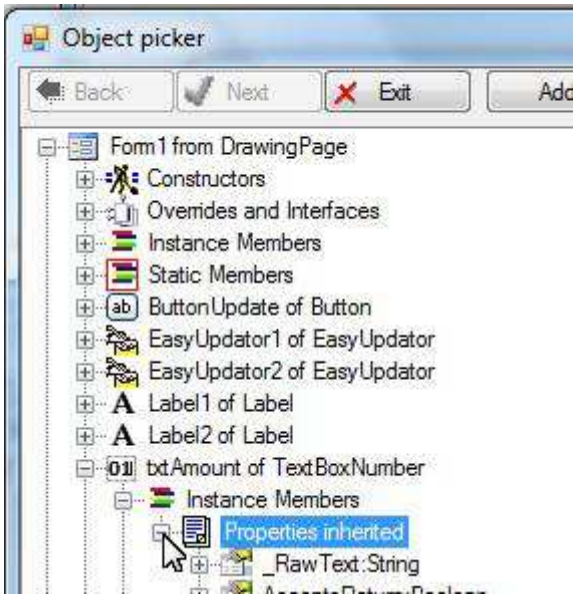
Right-click EasyUpdator1; choose “Create Action”; choose ExecuteWithParameterValues:

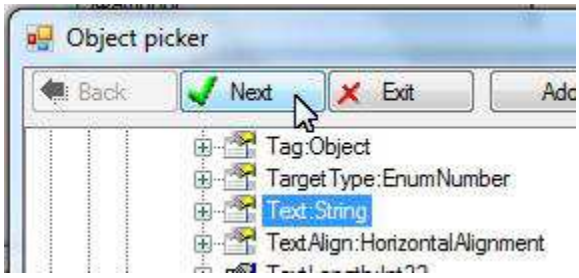


Select Property for parameter @amount:

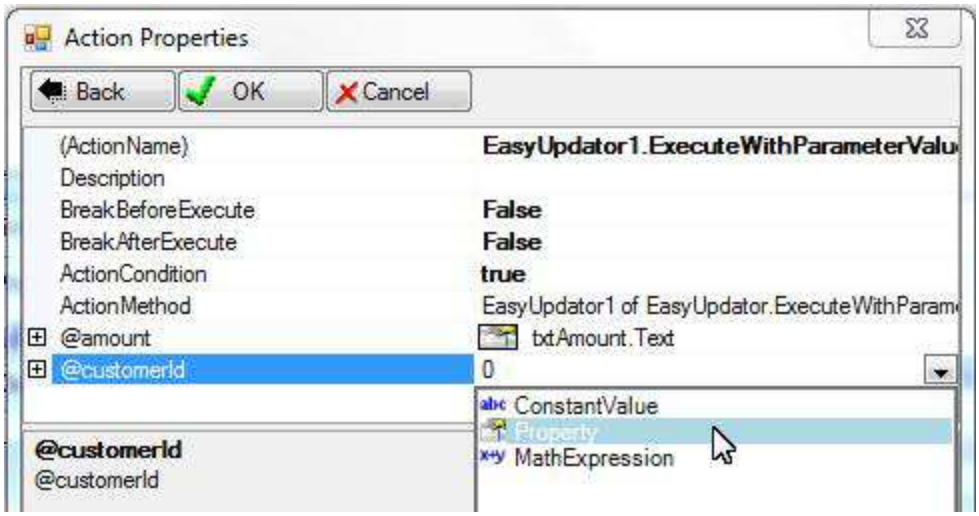


Expand “Properties inherited” node of the text box:

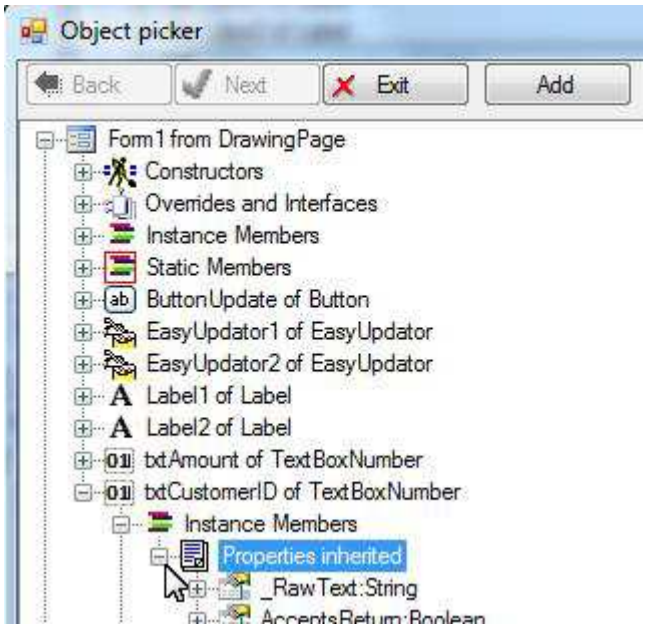


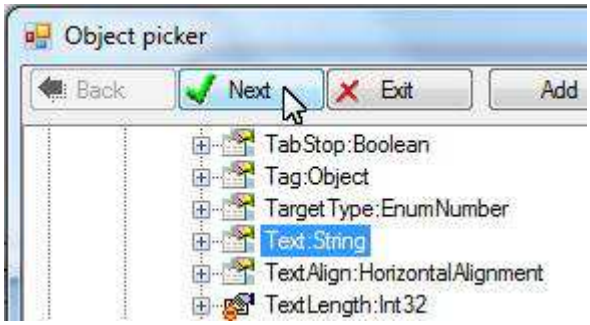


Select Property for parameter @customerId:

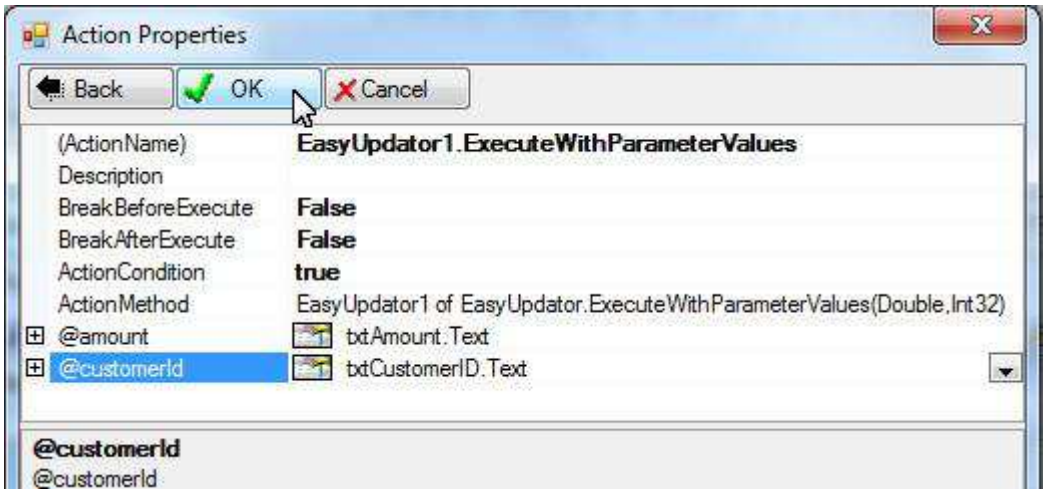


Expand “properties inherited” node for the text box of customer id:

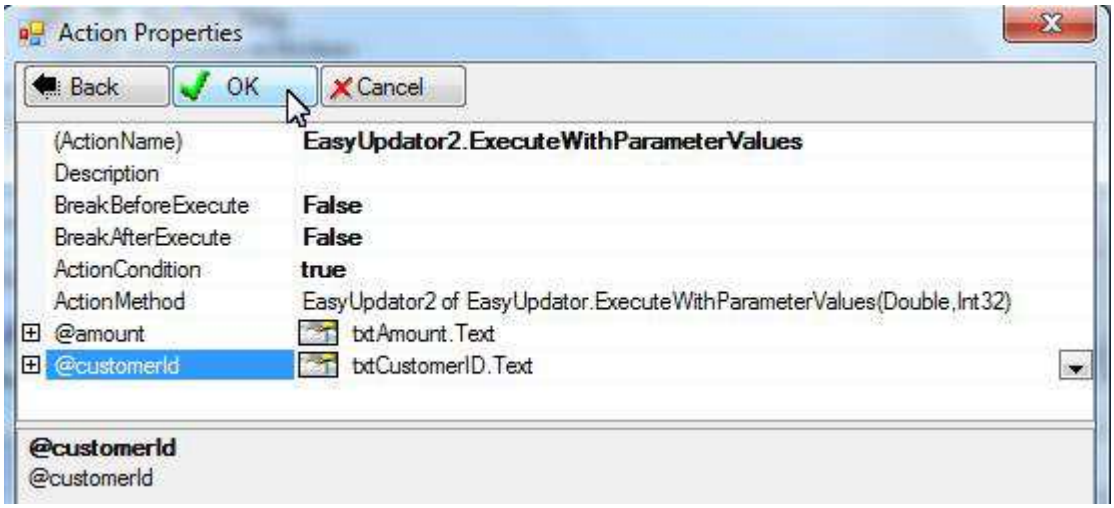




Click OK to finish creating this action:

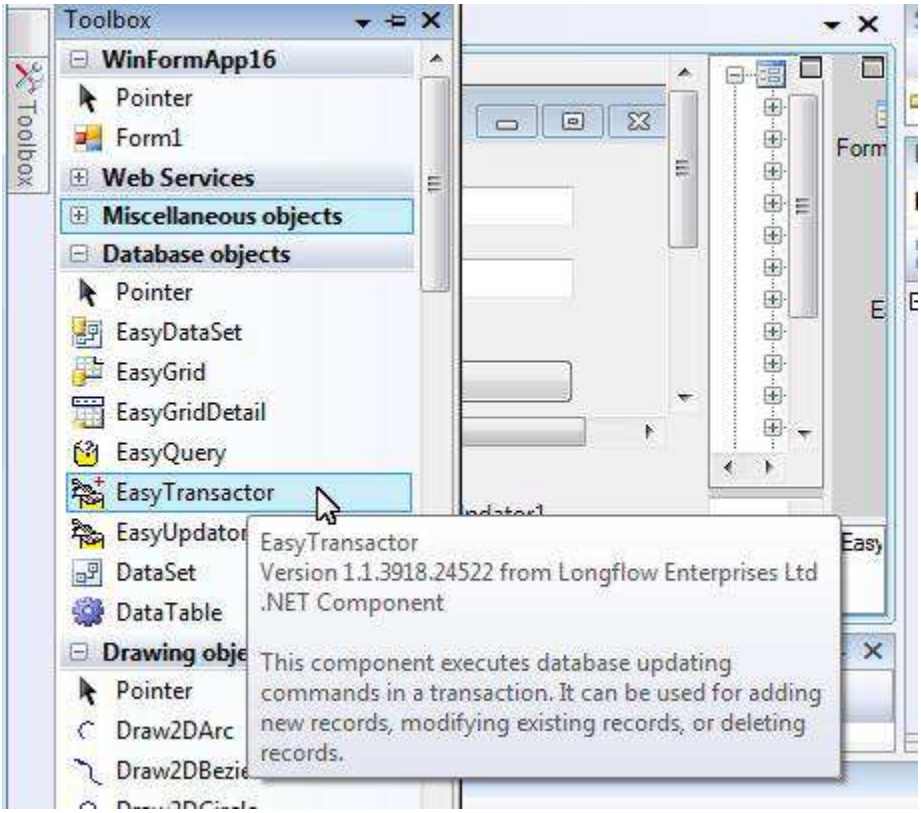


For EasyUpdator2, also create an ExecuteWithParameterValues action:

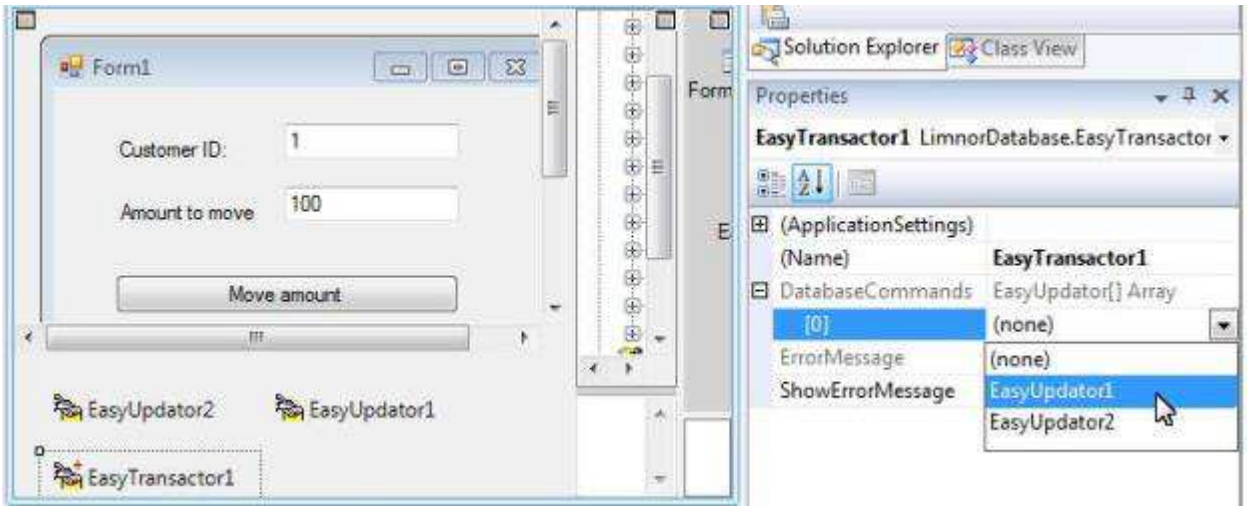


11.4 Add action executers into a transaction

Create an EasyTransactor component:



Add EasyUpdator1 to the transaction:

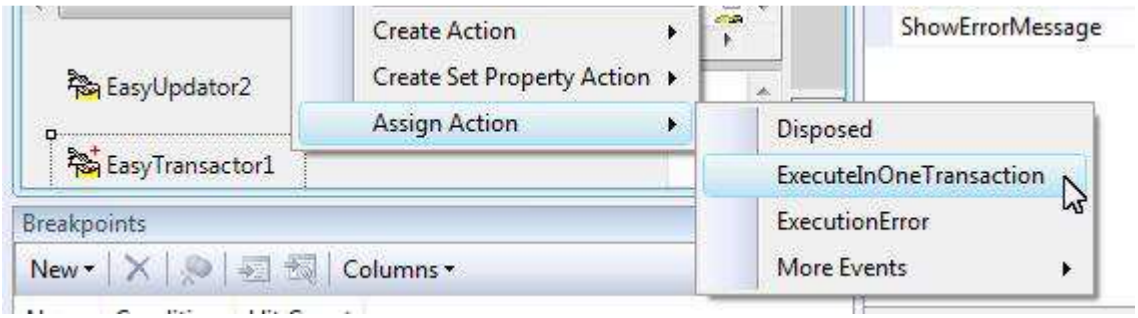


Add EasyUpdator2 to the transaction:

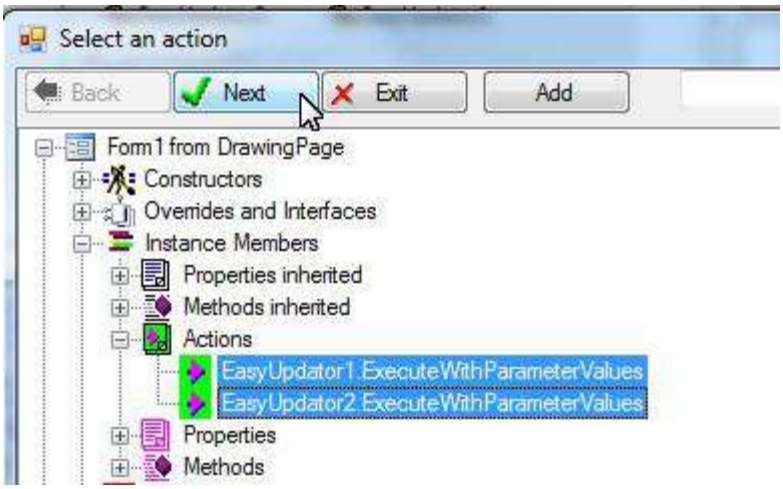
11.5 Add actions into a transaction

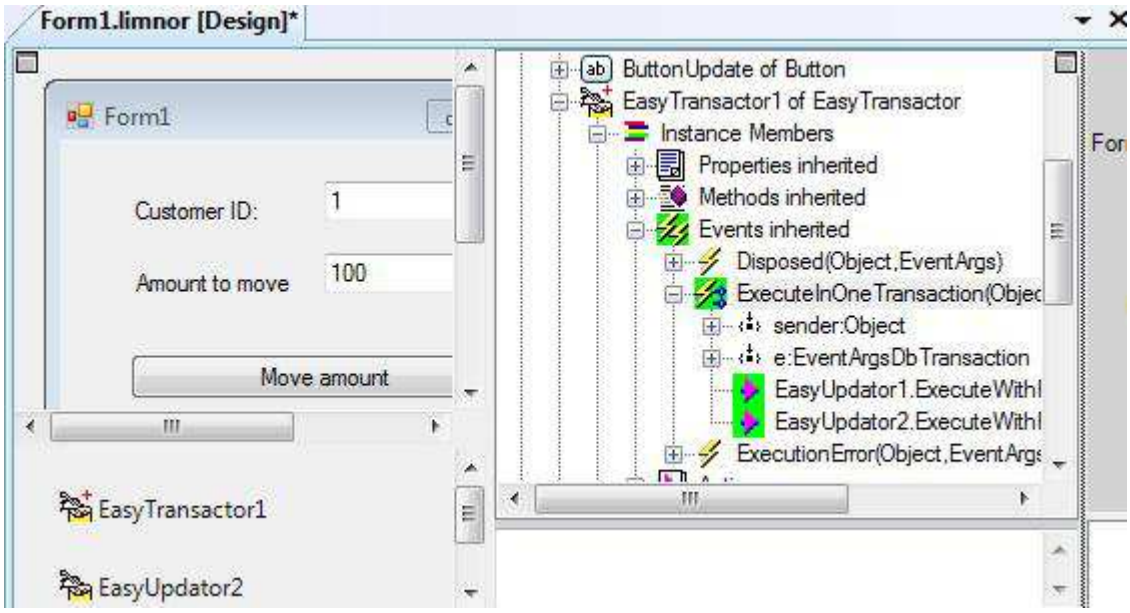
Event ExecuteInOneTransaction of EasyTransactor occurs when it is time to execute database updating actions in one transaction.

For a simple case like this example, we simply add the two database updating actions to this event:



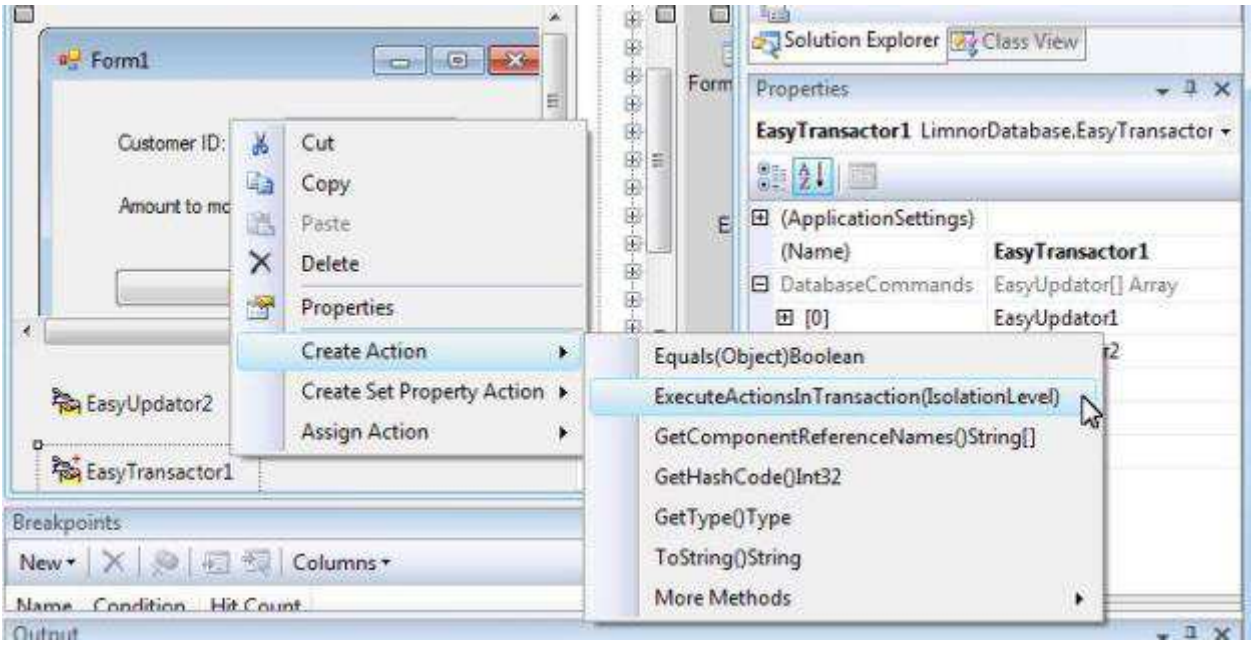
Select the two database updating actions:



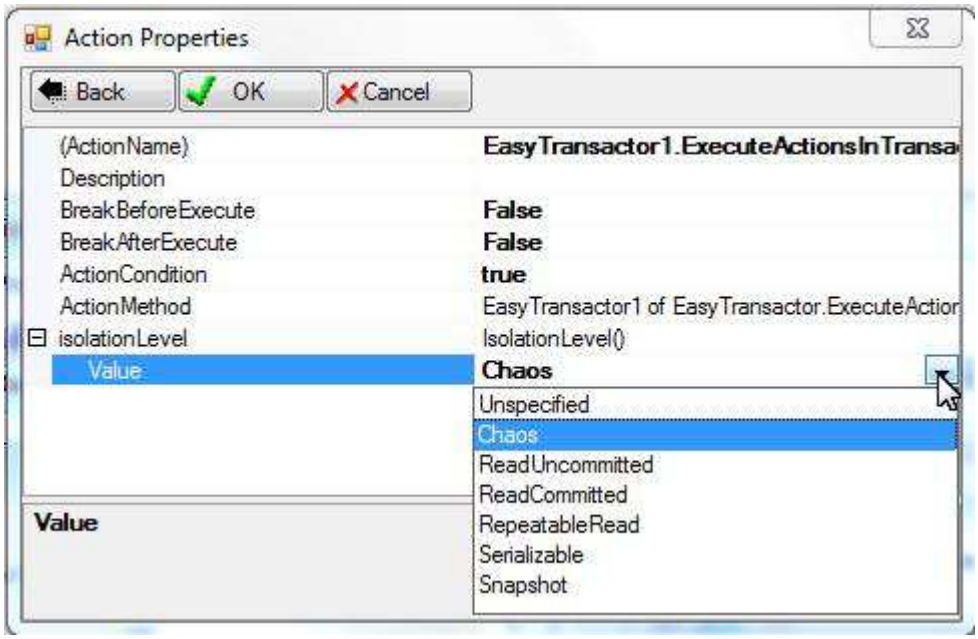


11.6 Create start-transaction action

A transaction is started by an ExecuteActionsInTransaction action. Right-click EasyTransactor1; choose "Create Action"; choose ExecuteActionsInTransaction



This action requires a parameter, isolation level



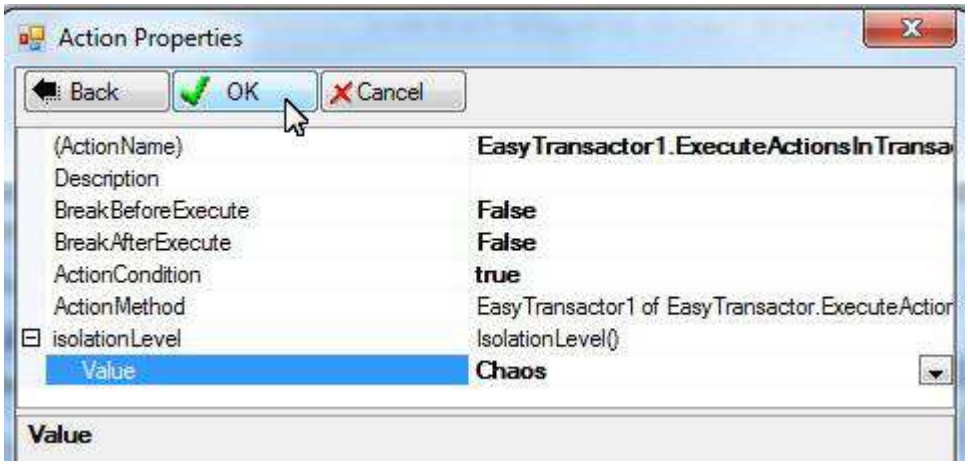
The isolationLevel can be one of the following values:

- Unspecified -- A different isolation level than the one specified is being used, but the level cannot be determined.
  When using OdbcTransaction, if you do not set IsolationLevel or you set IsolationLevel to Unspecified, the transaction executes according to the isolation level that is determined by the driver that is being used.

- Chaos -- The pending changes from more highly isolated transactions cannot be overwritten.
- ReadUncommitted -- A dirty read is possible, meaning that no shared locks are issued and no exclusive locks are honored.
- ReadCommitted -- Shared locks are held while the data is being read to avoid dirty reads, but the data can be changed before the end of the transaction, resulting in non-repeatable reads or phantom data.
- RepeatableRead -- Locks are placed on all data that is used in a query, preventing other users from updating the data. Prevents non-repeatable reads but phantom rows are still possible.
- Serializable -- A range lock is placed on the DataSet, preventing other users from updating or inserting rows into the dataset until the transaction is complete.
- Snapshot -- Reduces blocking by storing a version of data that one application can read while another is modifying the same data. Indicates that from one transaction you cannot see changes made in other transactions, even if you requery.

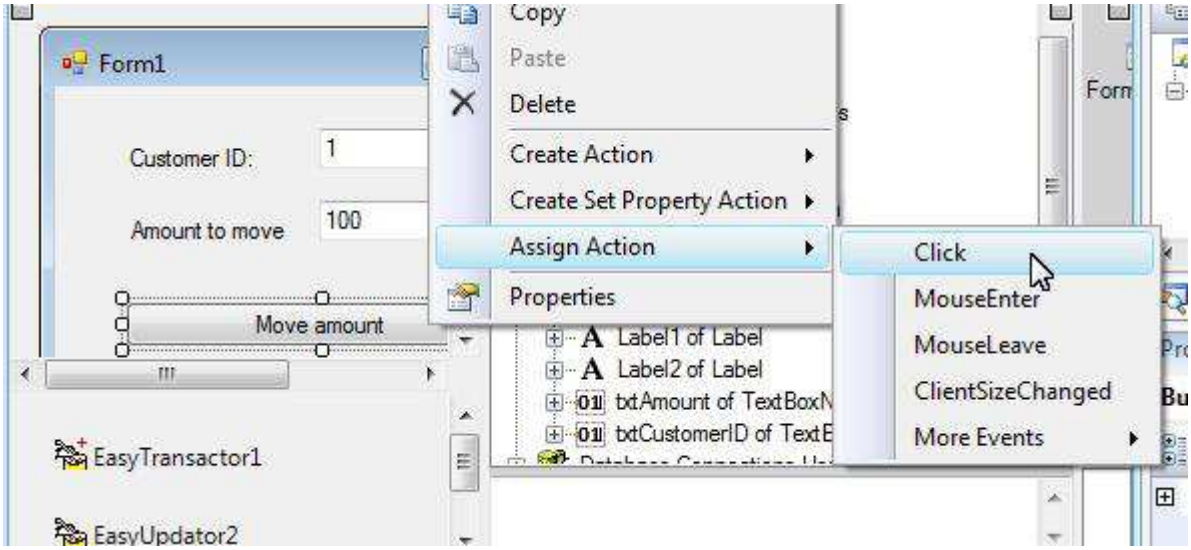
For more information on transaction isolation levels, see http://msdn.microsoft.com/en-us/library/system.data.isolationlevel.aspx

If you are not sure which isolation level to use then you may try to use “Chaos”

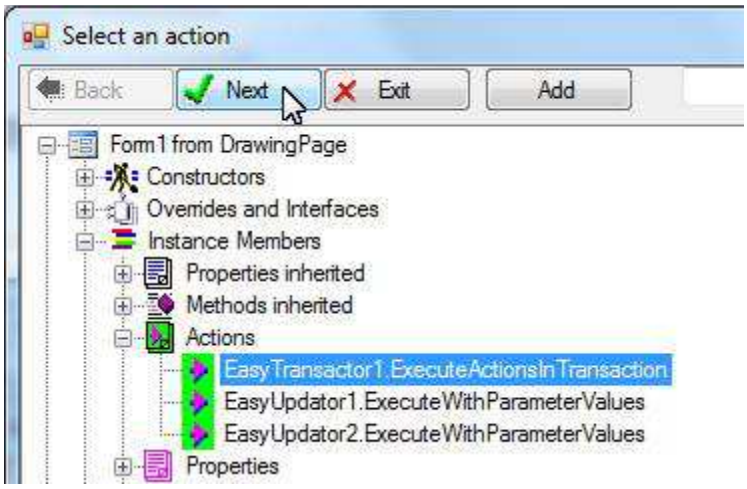


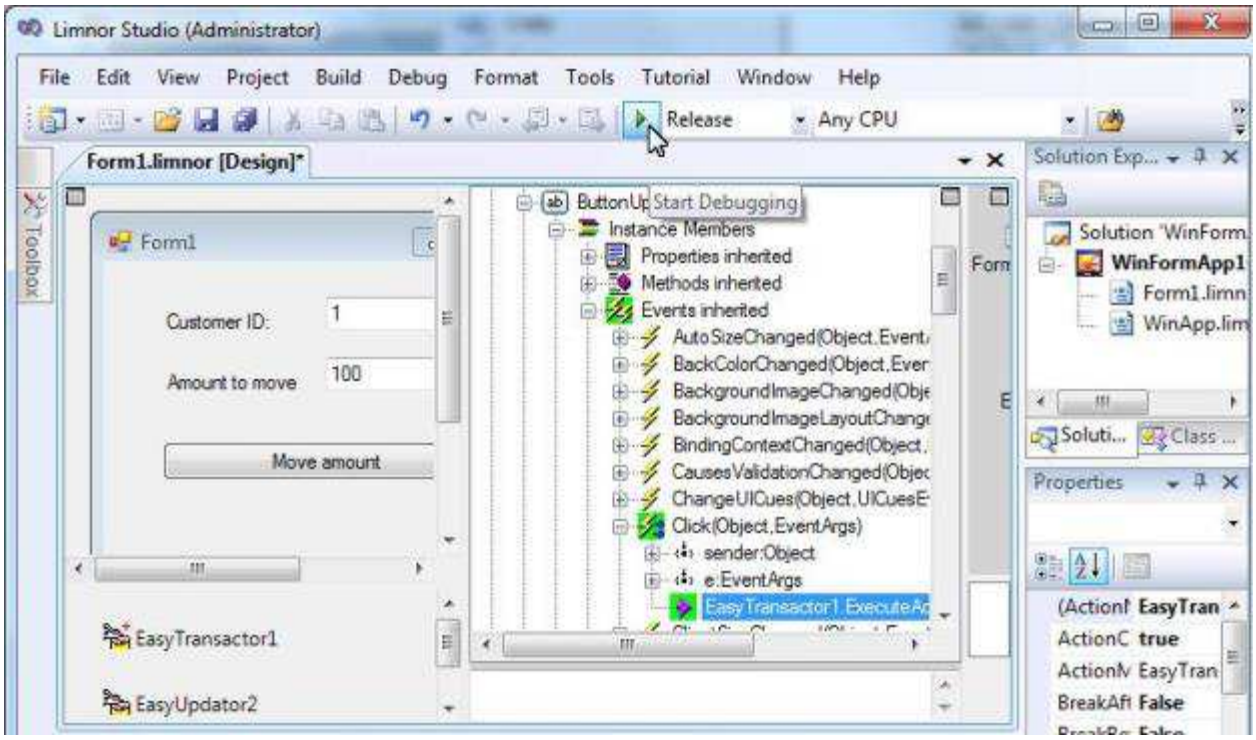
11.7 Execute start-transaction action

Assign the ExecuteActionsInTransaction action to a button to execute it

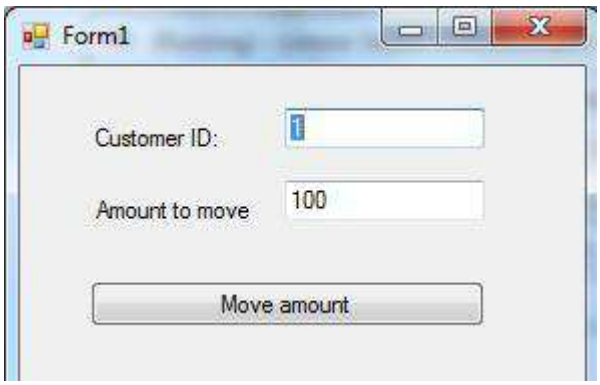


Select the action. Click Next:





Click the button. Check the database to verify the two database actions are executed.



11.8 Summary

The procedure we used to execute database updating actions in a transaction can be summarized below.

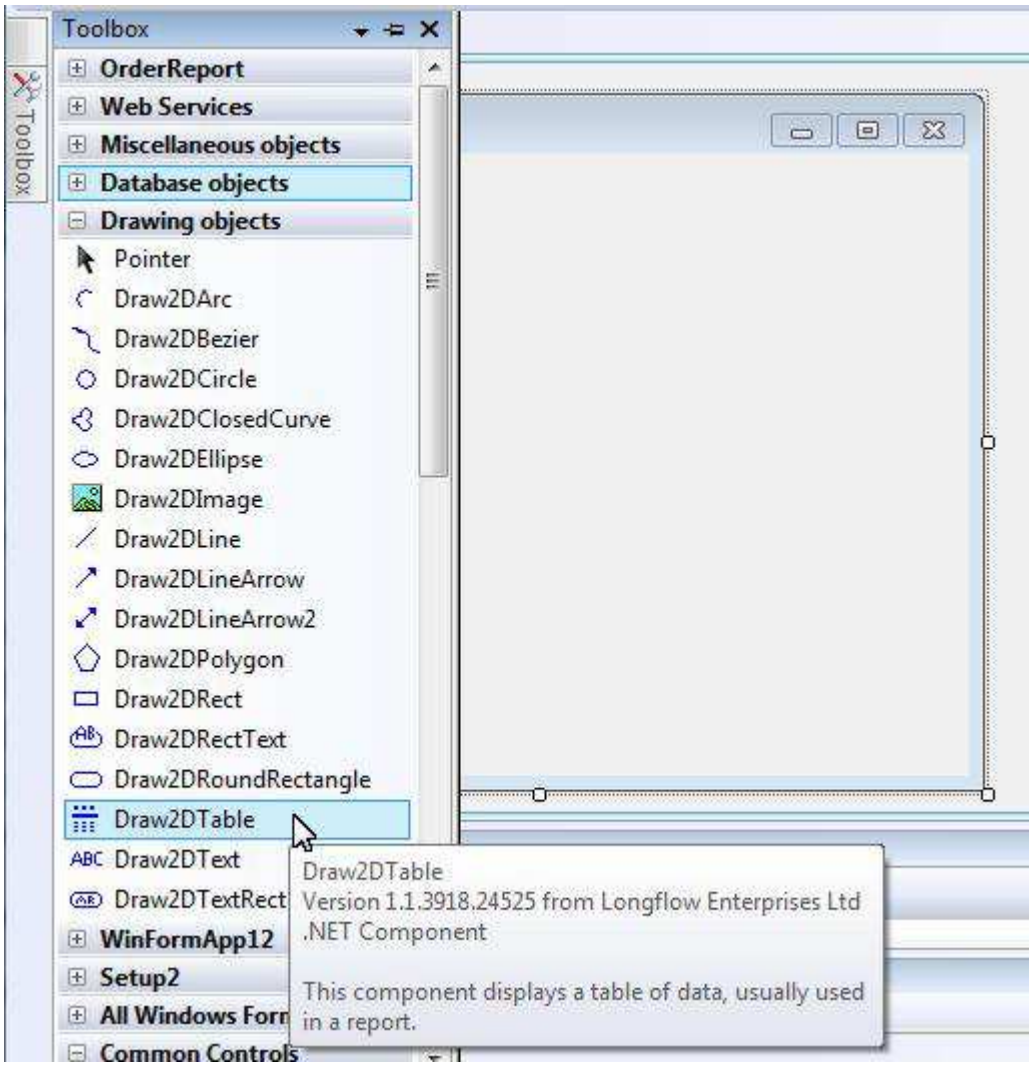
1. Create EasyUpdator components as action executers. Setup their properties for connecting to database and creating database update commands
2. Use EasyUpdator components to create actions for updating the database.
   The above two steps are the same process as doing database updating without using transactions.
3. Create an EasyTransactor component
4. Add EasyUpdator components created in step 1 to the EasyTransactor component
5. Add database updating actions created in step 2 to the ExecuteActionsInTransaction event of the EasyTransaction component

6. Create an ExecuteActionsInTransaction action and execute the action.

12 Reporting

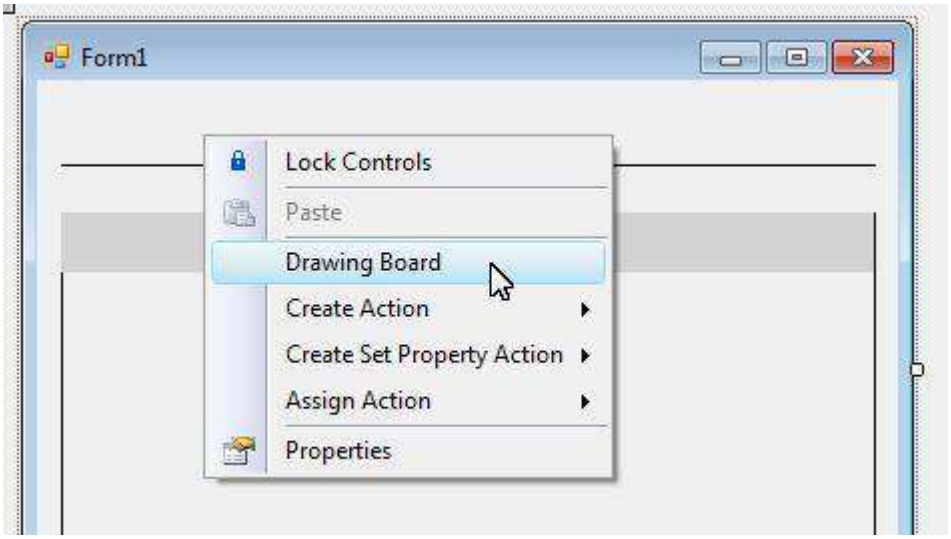
12.1 Design Reporting Layout

Reporting can be done via drawings. Drawing table may get data from a database and form a report:



Note that only drawing objects added to the form will be printed. We add lines and text together with the drawing table to form a report:

To design the report layout based on print page size, right-click the form, choose “Drawing board”:



-- Set page size

 -- add a new drawing

 -- delete selected drawing

 -- move selected drawing to front

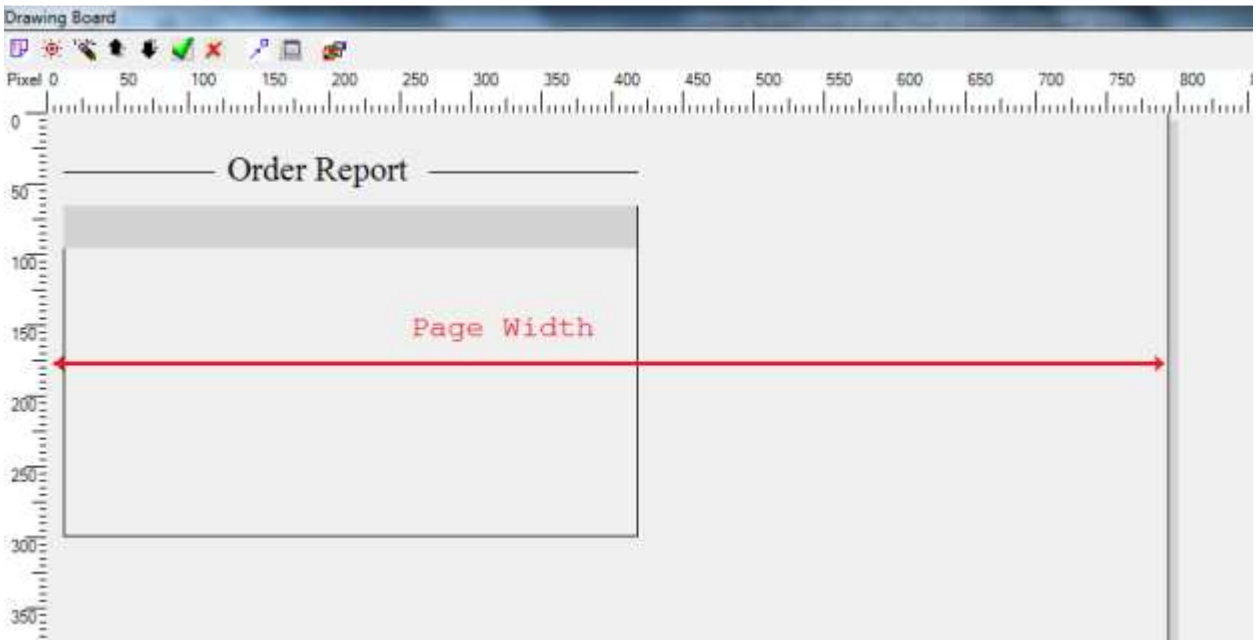
 -- move selected drawing to back

 -- move the Properties and Drawing Toolbox to the up-right corner

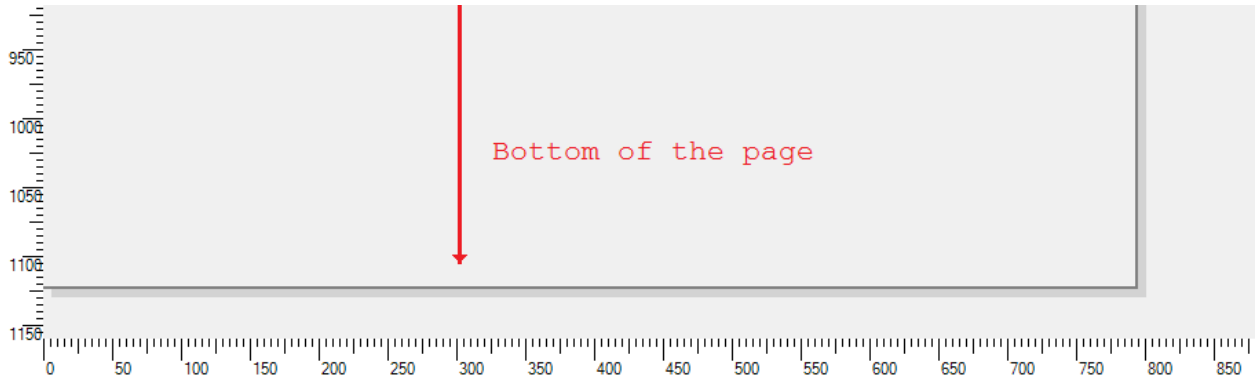
 -- make the drawing board full screen

 -- remember the properties of the currently selected drawing. When a new drawing of the same type is created the remembered properties will be used for the new drawing.

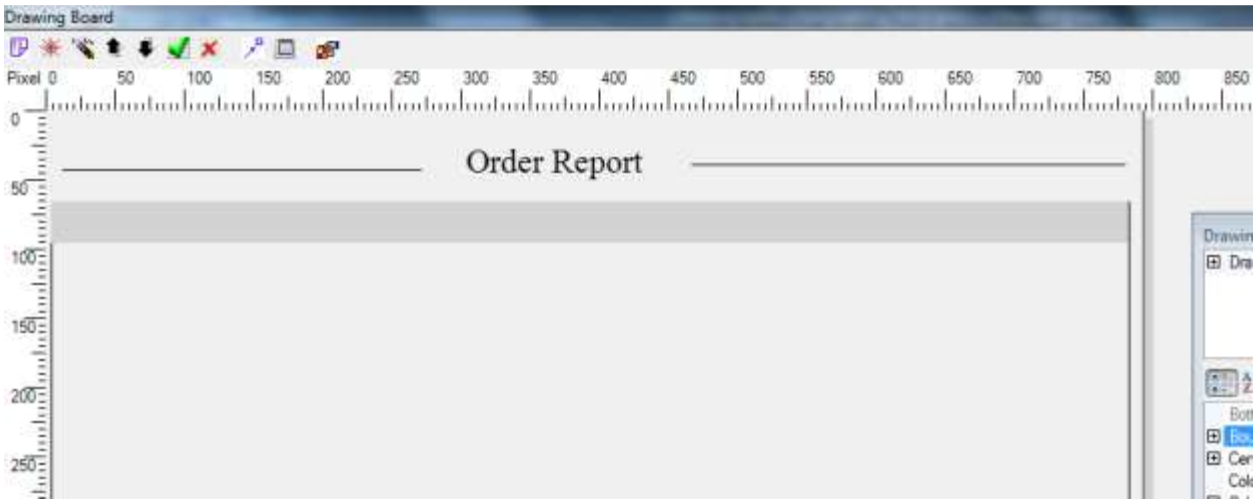
The print page area is marked by shaded lines. The vertical line marks the width of the page:



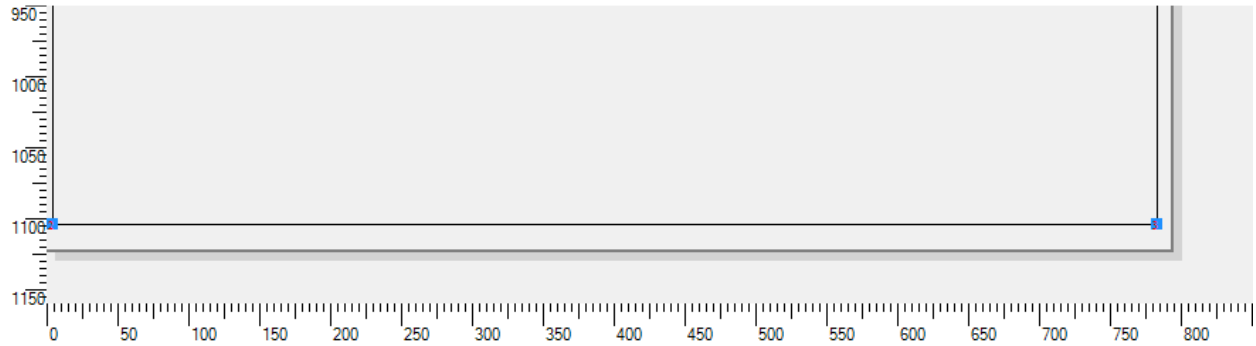
Scroll down and we can see the bottom of page marked by horizontal line:



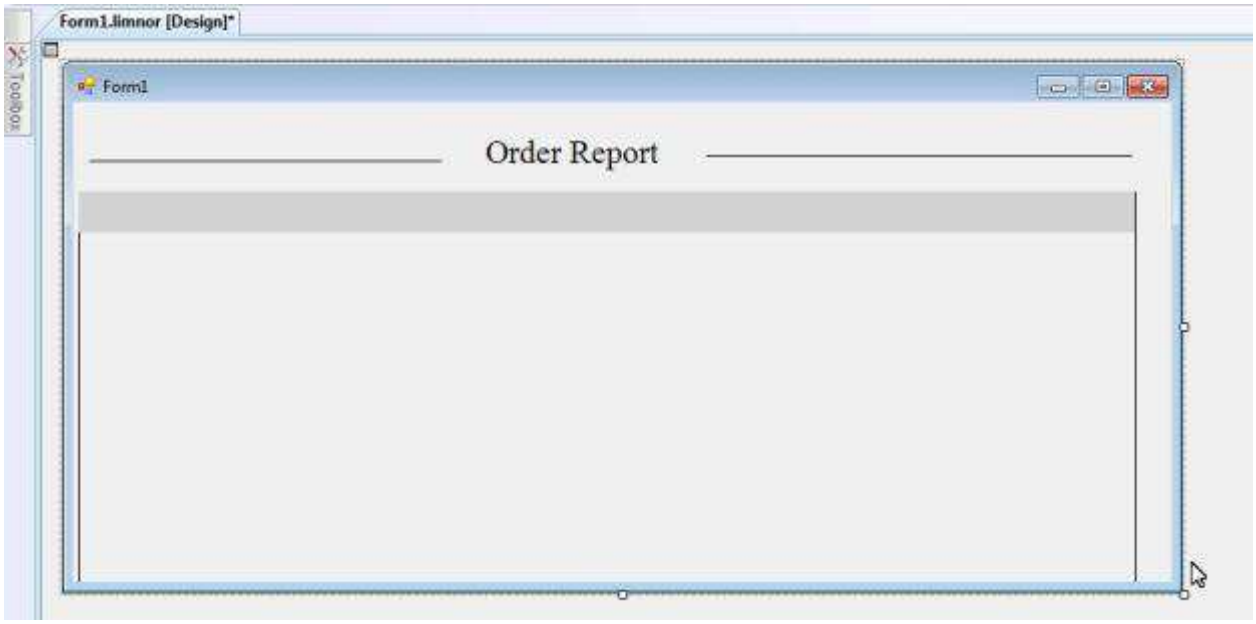
Design the report layout based on the size of the print page:



Drag mark  to make the drawing table fill the print page size:

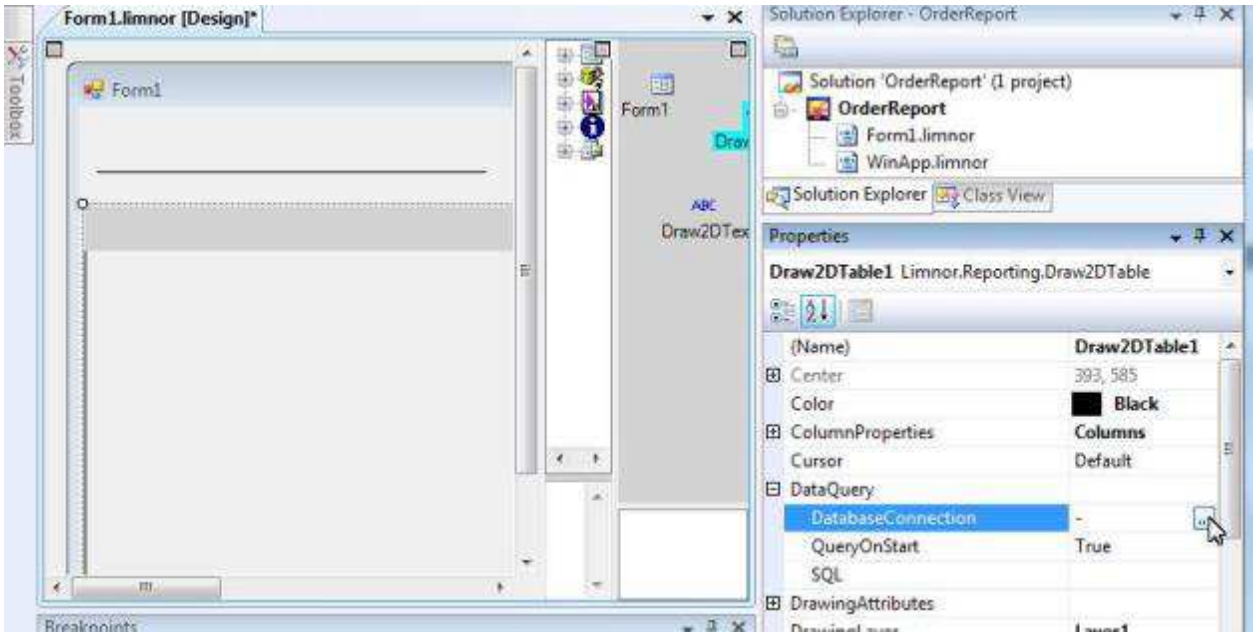


Close the drawing board and back to designers



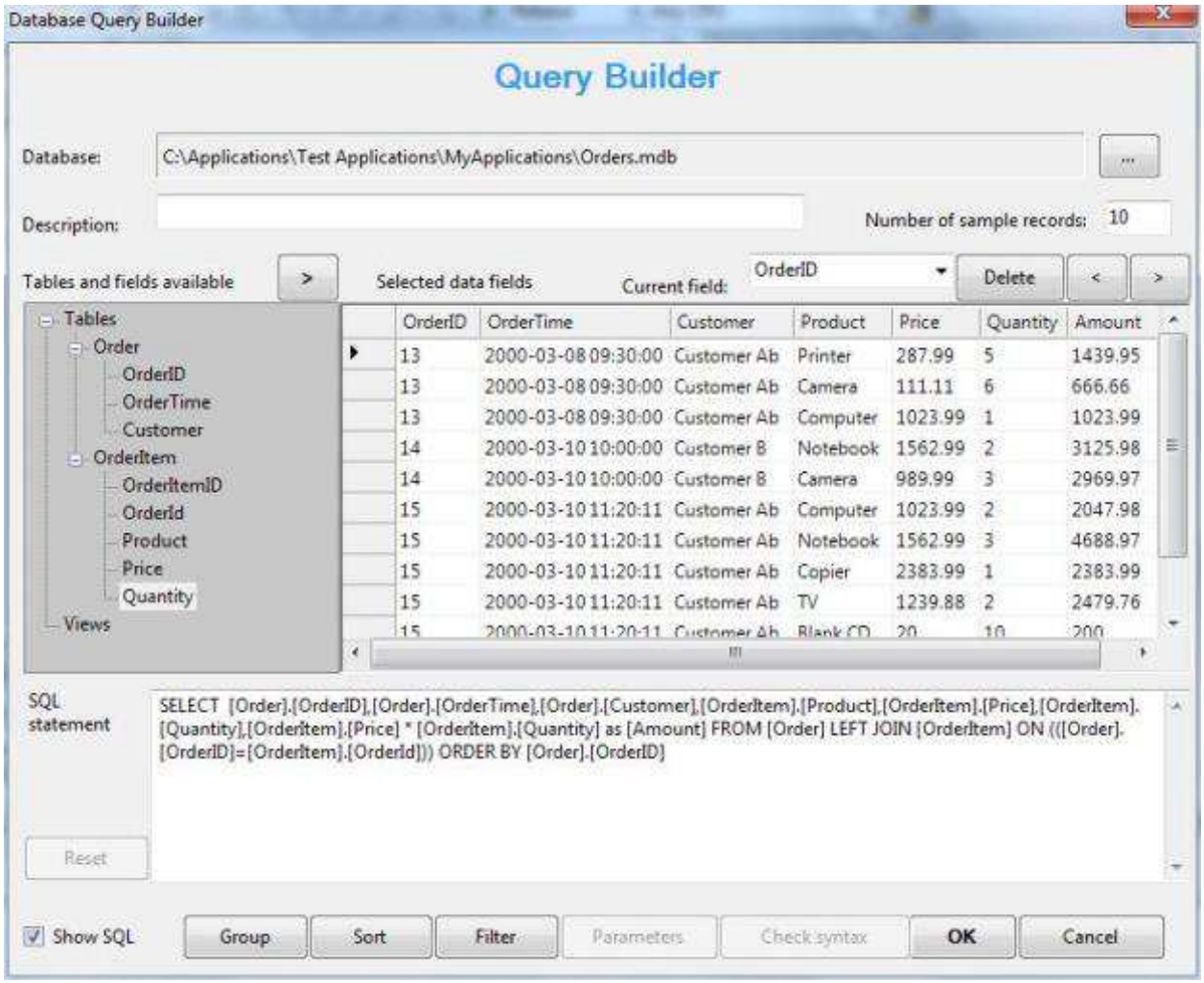
12.2 Create Report Query

DataQuery property is for specifying report query. Set its DatabaseConnection property to connect to a database.



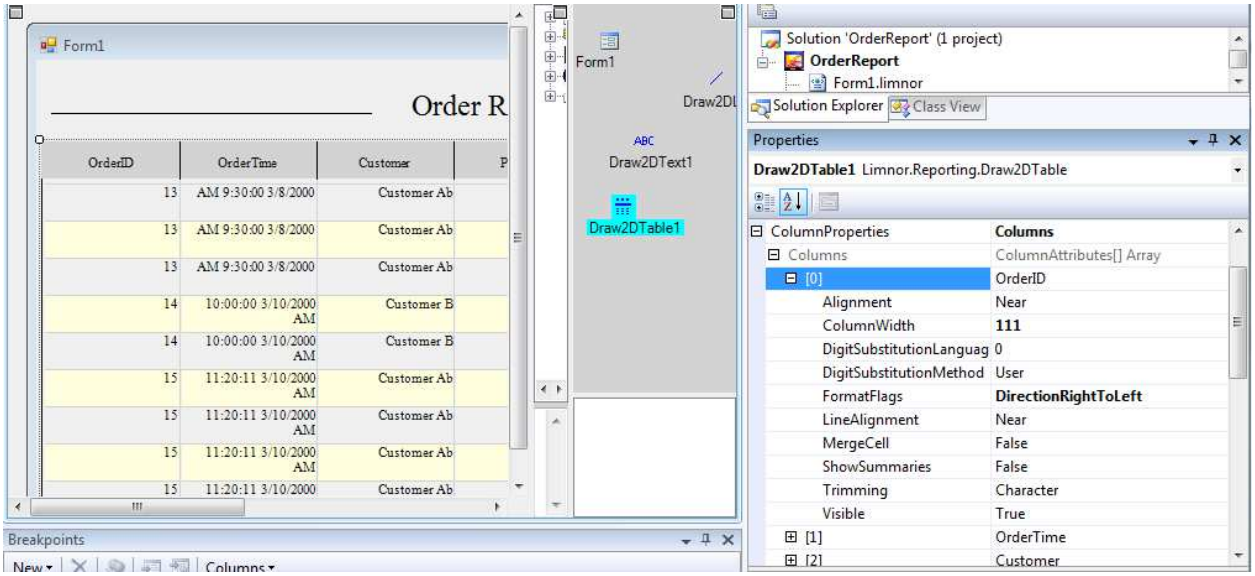
QueryOnStart indicates whether the report query will b executed when the application starts. If this property is False then a Query action should be executed when it is time to load data into the drawing table.

SQL property is for building the report query. We built this query before:



12.3 Set column properties

ColumnProperties controls the formatting for every column:



Each column may have following attributes:

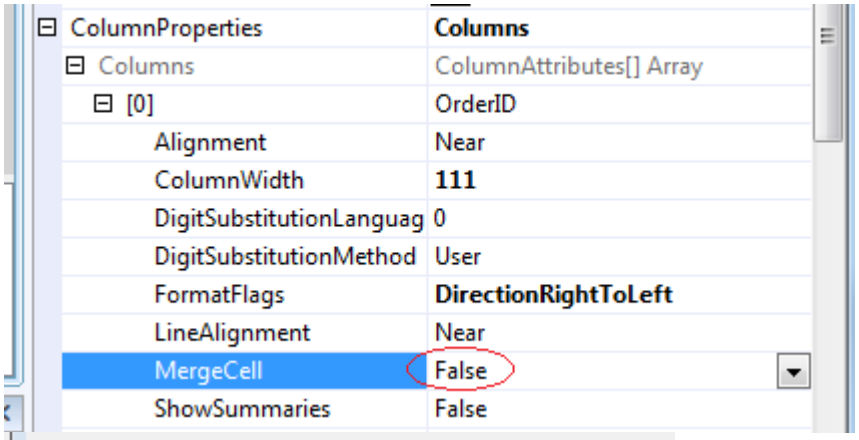
- **Visible** – Indicate whether the column should be displayed in the report
- **MergeCell** – Indicate whether same value of continuous rows should be merged. It will be explained in more details later
- **ShowSummaries** – Indicate whether summaries for this column should be displayed. It will be explained in more details later
- **ColumnWidth** – The width of the column
- **Alignment** -- Gets or sets horizontal alignment of the string.
- **DigitSubstitutionLanguage** -- Gets the language that is used when local digits are substituted for western digits.
  It is a National Language Support (NLS) language identifier that identifies the language that will be used when local digits are substituted for western digits. You can pass the LCID property of a CultureInfo object as the NLS language identifier. For example, suppose you create a CultureInfo object by passing the string "ar-EG" to a CultureInfo constructor. If you pass the LCID property of that CultureInfo object along with.Traditional to the SetDigitSubstitution method, then Arabic-Indic digits will be substituted for western digits at display time.
- **DigitSubstitutionMethod** -- Gets the method to be used for digit substitution. It is a StringDigitSubstitute enumeration value that specifies how to substitute characters in a string that cannot be displayed because they are not supported by the current font.
  - **User** -- Specifies a user-defined substitution scheme.
  - **None** -- Specifies to disable substitutions.
  - **National** -- Specifies substitution digits that correspond with the official national language of the user's locale.
  - **Traditional** -- Specifies substitution digits that correspond with the user's native script or language, which may be different from the official national language of the user's locale.

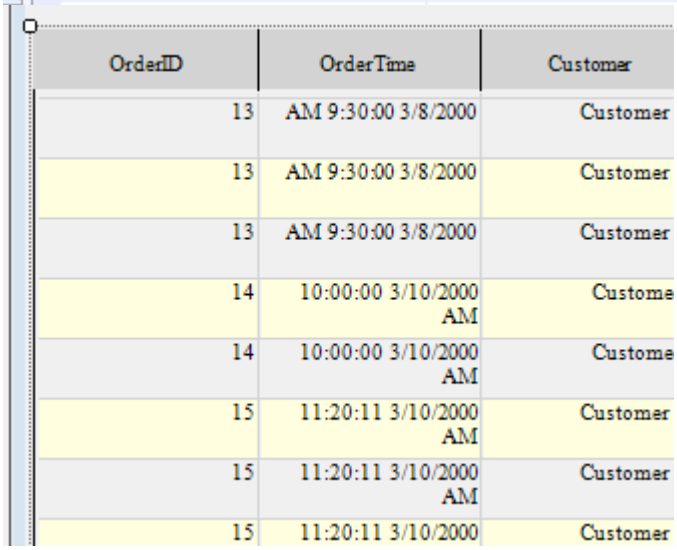
- **FormatFlags** -- Gets or sets a StringFormatFlags enumeration that contains formatting information.
    - **DirectionRightToLeft** -- Text is displayed from right to left.
    - **DirectionVertical** -- Text is vertically aligned.
    - **FitBlackBox** -- Parts of characters are allowed to overhang the string's layout rectangle. By default, characters are repositioned to avoid any overhang.
    - **DisplayFormatControl** -- Control characters such as the left-to-right mark are shown in the output with a representative glyph.
    - **NoFontFallback** -- Fallback to alternate fonts for characters not supported in the requested font is disabled. Any missing characters are displayed with the fonts missing glyph, usually an open square.
    - **MeasureTrailingSpaces** -- Includes the trailing space at the end of each line. By default the boundary rectangle returned by the MeasureString method excludes the space at the end of each line. Set this flag to include that space in measurement.
    - **NoWrap** -- Text wrapping between lines when formatting within a rectangle is disabled. This flag is implied when a point is passed instead of a rectangle, or when the specified rectangle has a zero line length.
    - **LineLimit** -- Only entire lines are laid out in the formatting rectangle. By default layout continues until the end of the text, or until no more lines are visible as a result of clipping, whichever comes first. Note that the default settings allow the last line to be partially obscured by a formatting rectangle that is not a whole multiple of the line height. To ensure that only whole lines are seen, specify this value and be careful to provide a formatting rectangle at least as tall as the height of one line.
    - **NoClip** -- Overhanging parts of glyphs, and unwrapped text reaching outside the formatting rectangle are allowed to show. By default all text and glyph parts reaching outside the formatting rectangle are clipped.
- **LineAlignment** -- Gets or sets the vertical alignment of the string.
- **Trimming** -- Gets or sets the StringTrimming enumeration for this StringFormat object.
    - **None** -- Specifies no trimming.
    - **Character** -- Specifies that the text is trimmed to the nearest character.
    - **Word** -- Specifies that text is trimmed to the nearest word.
    - **EllipsisCharacter** -- Specifies that the text is trimmed to the nearest character, and an ellipsis is inserted at the end of a trimmed line.
    - **EllipsisWord** -- Specifies that text is trimmed to the nearest word, and an ellipsis is inserted at the end of a trimmed line.
    - **EllipsisPath** -- The center is removed from trimmed lines and replaced by an ellipsis. The algorithm keeps as much of the last slash-delimited segment of the line as possible.

12.4 Merge cells

If MergeCell property for a column is true then the rows with the same value for the column will be merged into one cell.

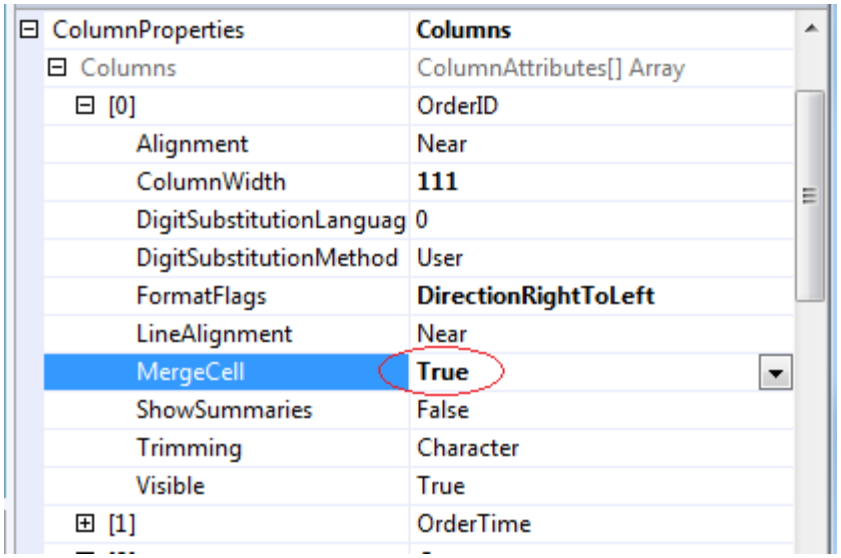
MergeCell is False:





OrderID	OrderTime	Customer
13	AM 9:30:00 3/8/2000	Customer
13	AM 9:30:00 3/8/2000	Customer
13	AM 9:30:00 3/8/2000	Customer
14	10:00:00 3/10/2000 AM	Custome
14	10:00:00 3/10/2000 AM	Custome
15	11:20:11 3/10/2000 AM	Customer
15	11:20:11 3/10/2000 AM	Customer
15	11:20:11 3/10/2000	Customer

MergeCell for OrderID is True:



OrderID	OrderTime	Customer	P
	AM 9:30:00 3/8/2000	Customer Ab	
13	AM 9:30:00 3/8/2000	Customer Ab	
	AM 9:30:00 3/8/2000	Customer Ab	
14	10:00:00 3/10/2000 AM	Customer B	
	10:00:00 3/10/2000 AM	Customer B	
	11:20:11 3/10/2000 AM	Customer Ab	
	11:20:11 3/10/2000 AM	Customer Ab	
15	11:20:11 3/10/2000 AM	Customer Ab	
	11:20:11 3/10/2000	Customer Ab	

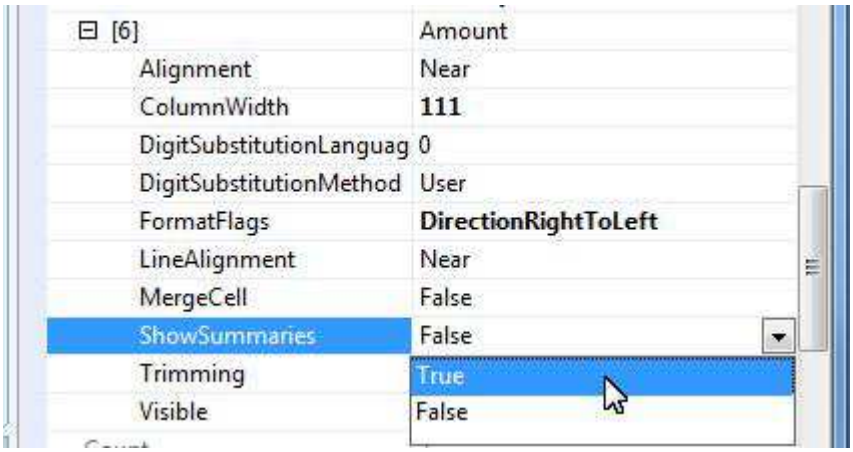
We may set many columns to be MergeCell = True:

OrderID	OrderTime	Customer	Product	Price	Quantity	Amount
			Printer	287.99	5	1439.95
13	AM 9:30:00 3/8/2000	Customer Ab	Camera	111.11	6	666.66
			Computer	1023.99	1	1023.99
14	10:00:00 3/10/2000 AM	Customer B	Notebook	1562.99	2	3125.98
			Camera	989.99	3	2969.97
			Computer	1023.99	2	2047.98
			Notebook	1562.99	3	4688.97
15	11:20:11 3/10/2000 AM	Customer Ab	Copier	2383.99	1	2383.99
			TV	1239.88	2	2479.76
			Blank CD	20	10	200

12.5 Show Summaries

12.5.1 Enable Summaries

Set ShowSummaries property to True to display summaries on a column.



For it to work the report query must be sorted. That is, an ORDER BY clause must exist.

Each sorting field in the ORDER BY clause must be included in the query's field list. For example, if the sorting is

ORDER By [Order].[OrderID]

then [Order].[OrderID] must be in the SELECT clause.

If an expression is used as a sorting field then the exact expression text should be used in the field list. For example, if the sorting is

ORDER BY x, y*z

then the SELECT clause can

SELECT x, y*z AS [field 1]

It cannot be

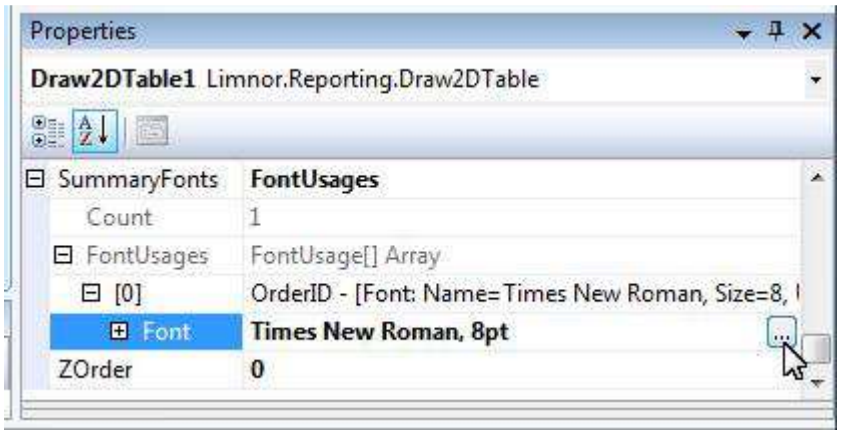
SELECT x, y * z AS [field 1]

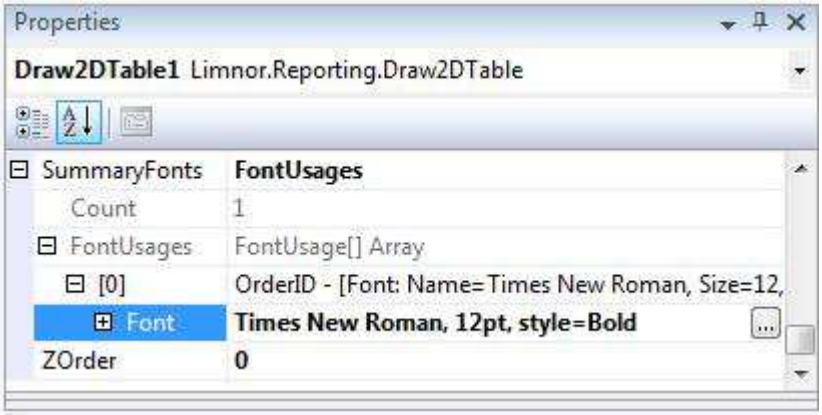
because there are spaces around *

Our query is sorted on [Order].[OrderID]. The report looks like

12.5.2 Set Summary Fonts

We may set the font for the summary:





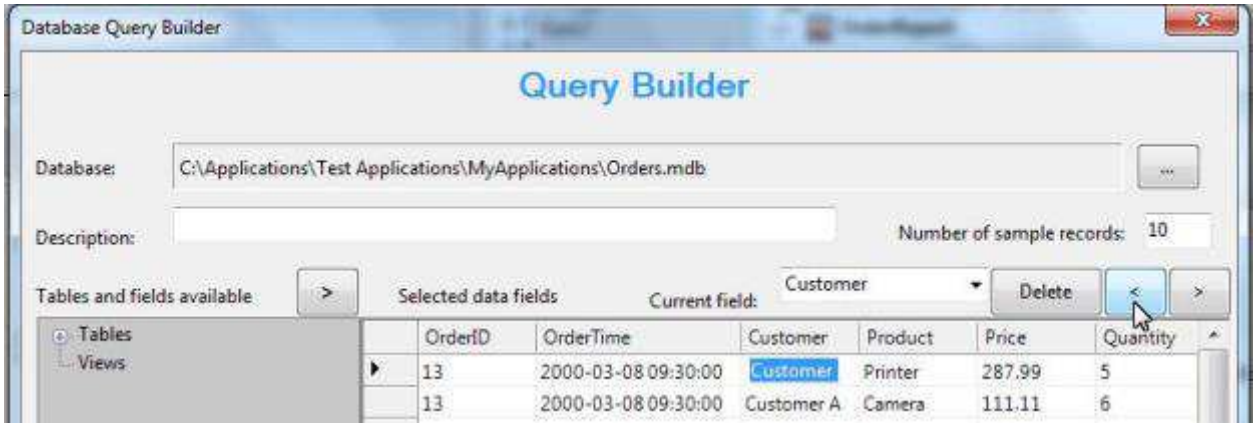
Now the report looks like

12.5.3 Multi-level summaries

Level of summaries is determined by the number of sorting field. In the above example, we sort on OrderID. There is one level of summaries.

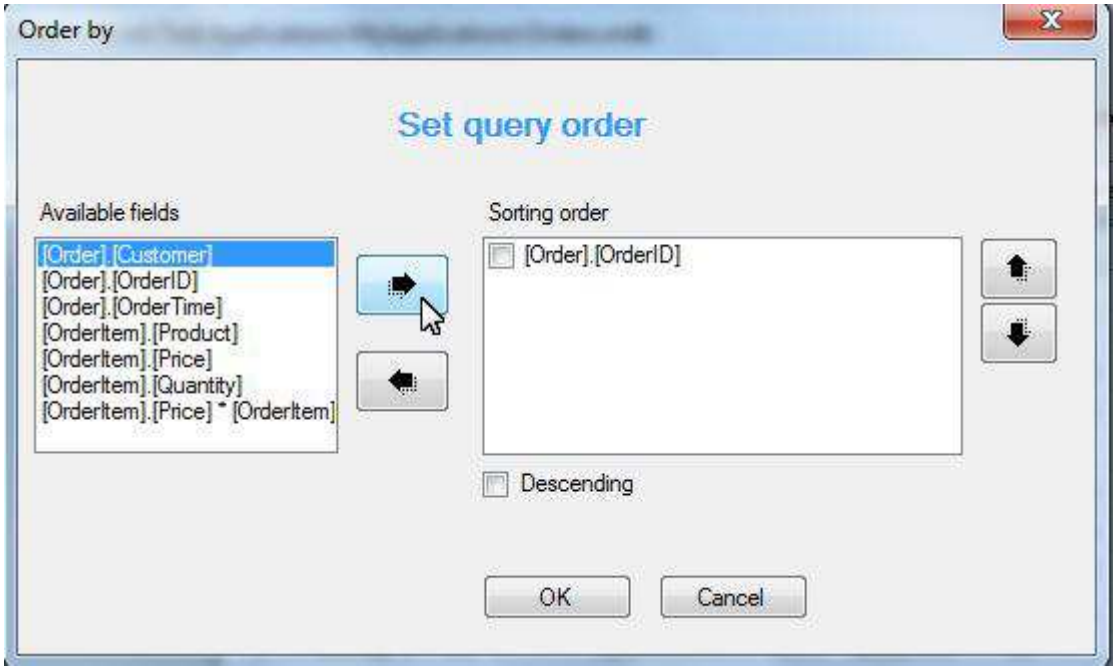
If we sort the report on Customer and OrderID then there are two levels of summaries. One level is on customer, the next level is on OrderID.

To make more sense of two level summaries we move Customer field to the first:

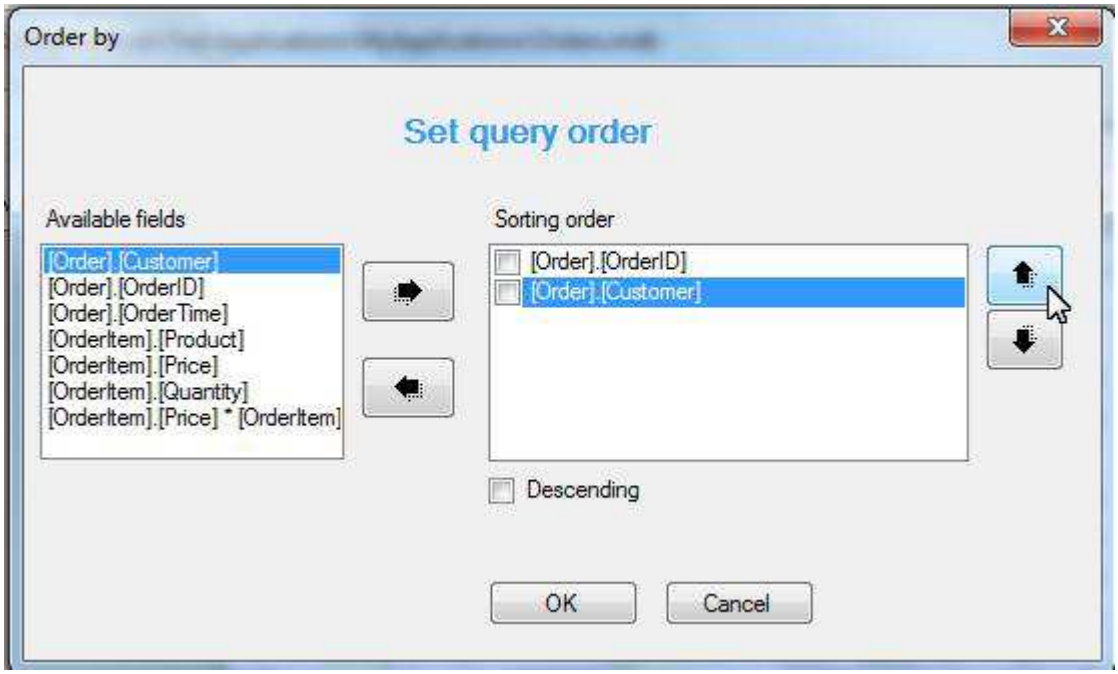


Click Sort button 

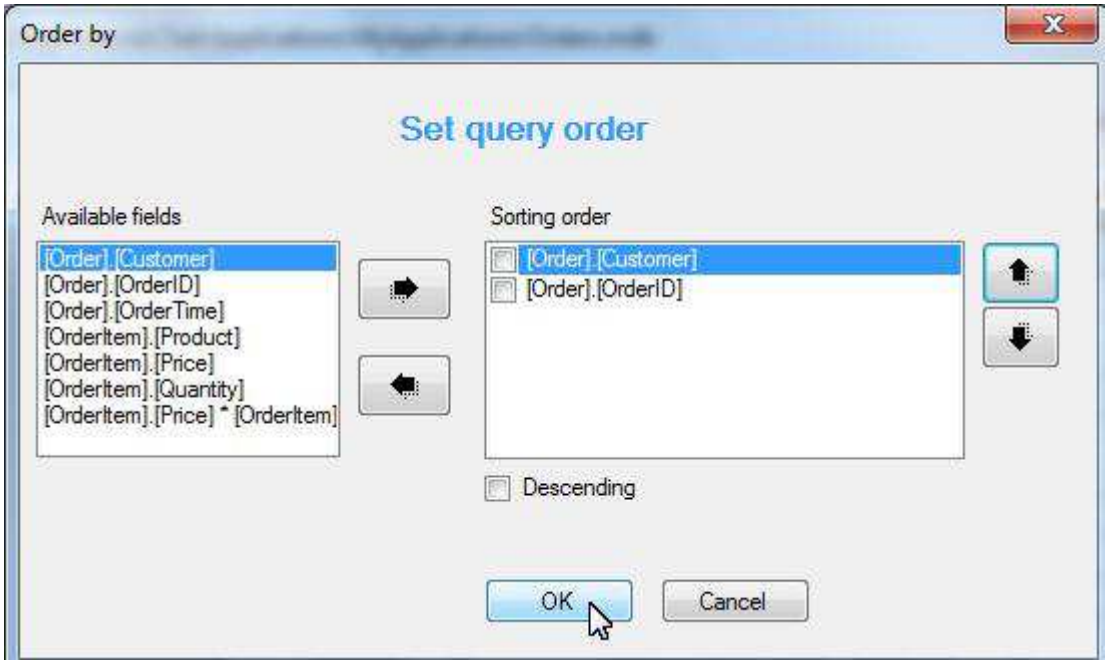
and add Customer to the sorting:



Move Customer field up to make it the first level sorting:

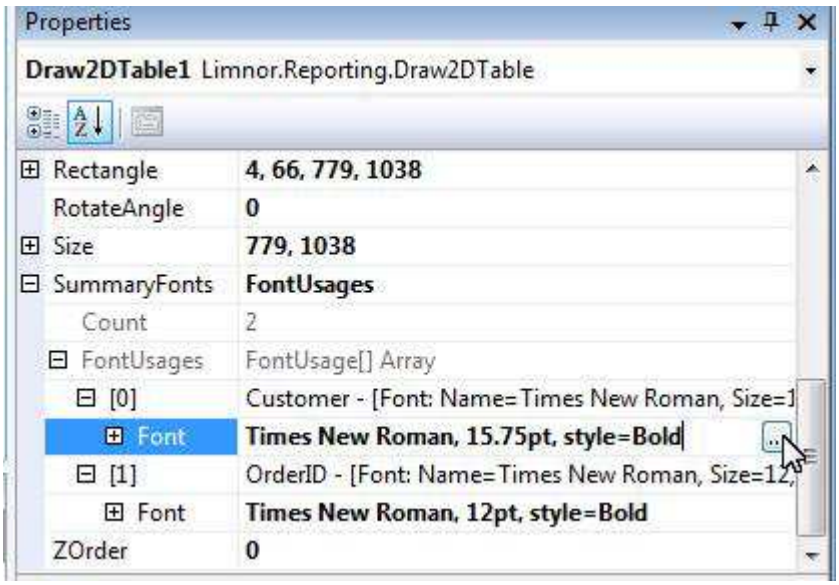


This is the two-level sorting we want to use:

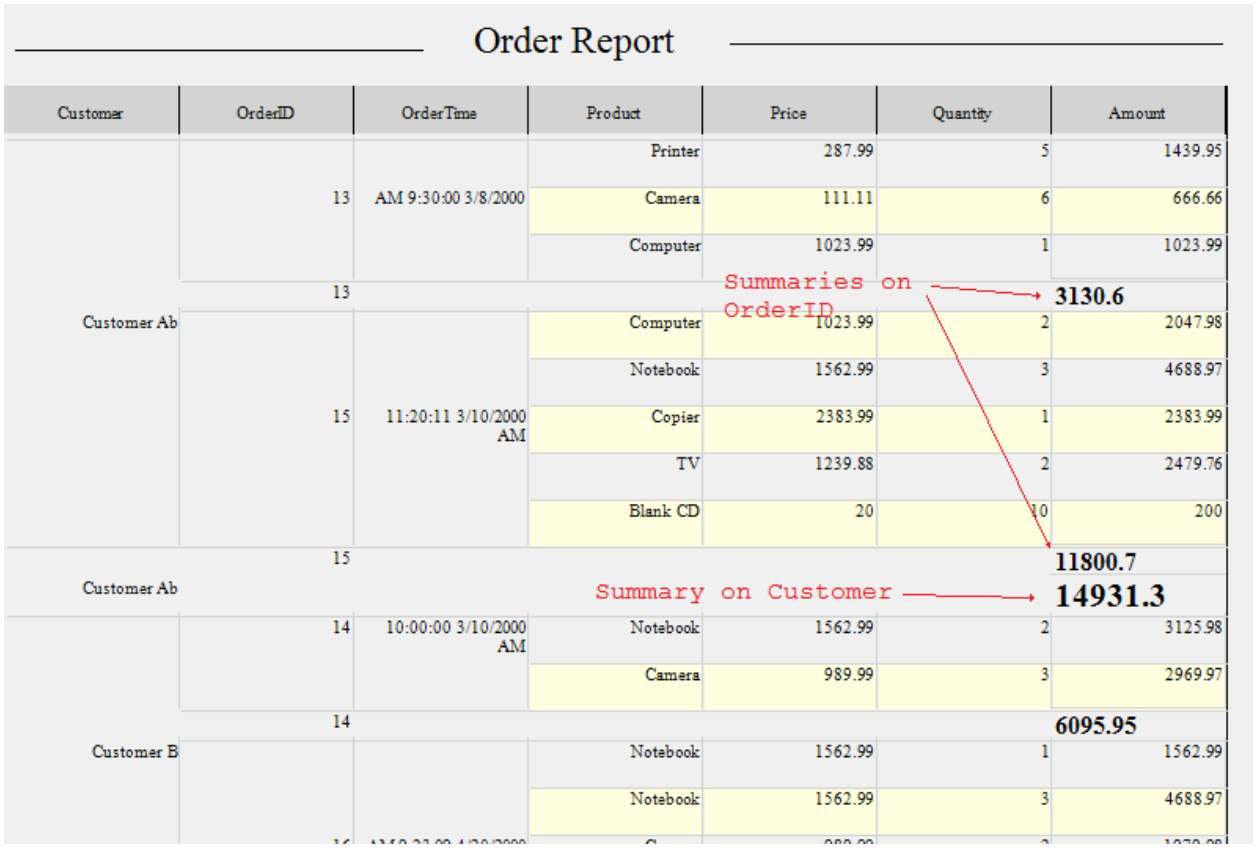


12.5.4 Multiple summary fonts

We may want to use different fonts to display different level of summaries:



The report now looks like:



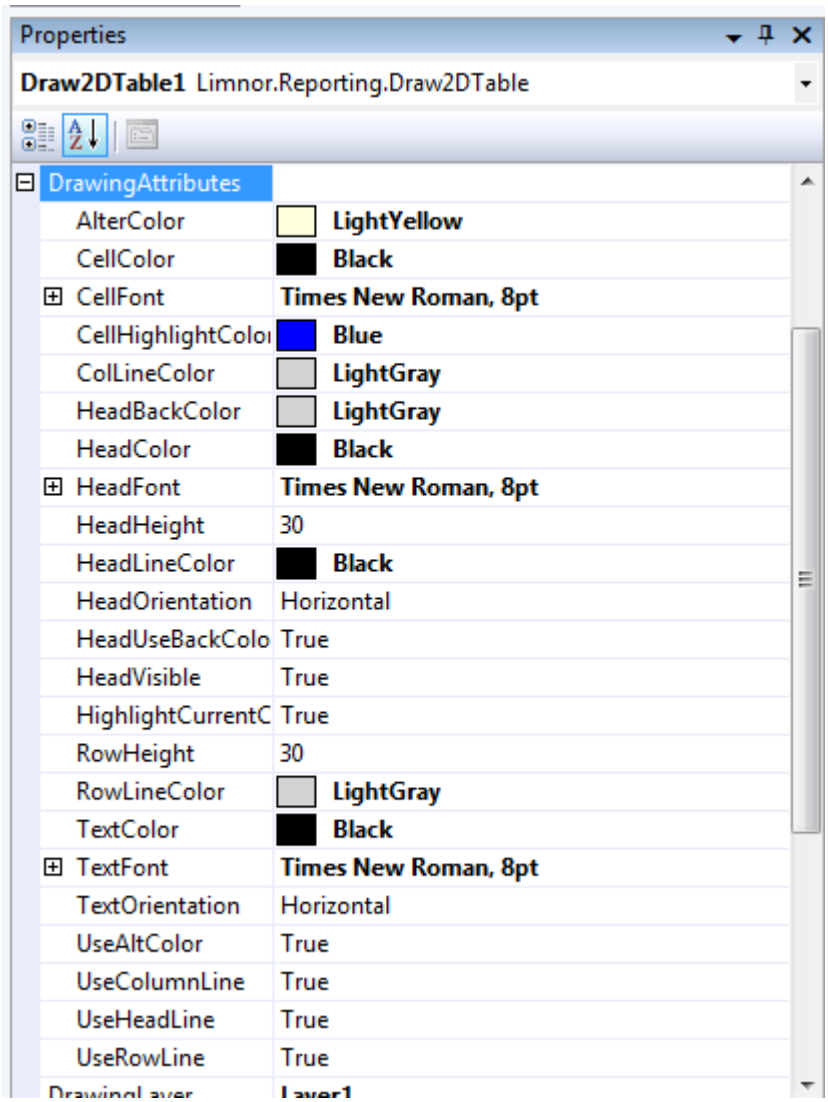
12.6 Report Drawing Properties

There are properties controlling the drawing of the report.

Fill – indicate whether the report background is filled with a color indicated by FillColor property.

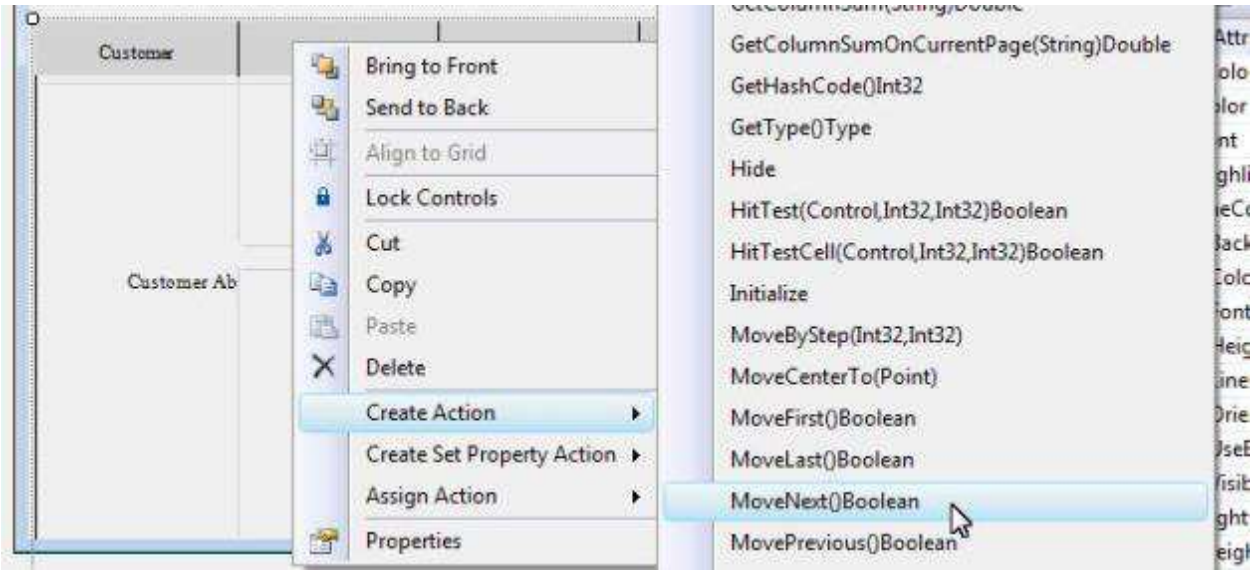
FillColor – See Fill property

DrawingAttributes – It includes properties controlling various aspects of report drawing. You may experiment on each of them to make your report look the way you want.



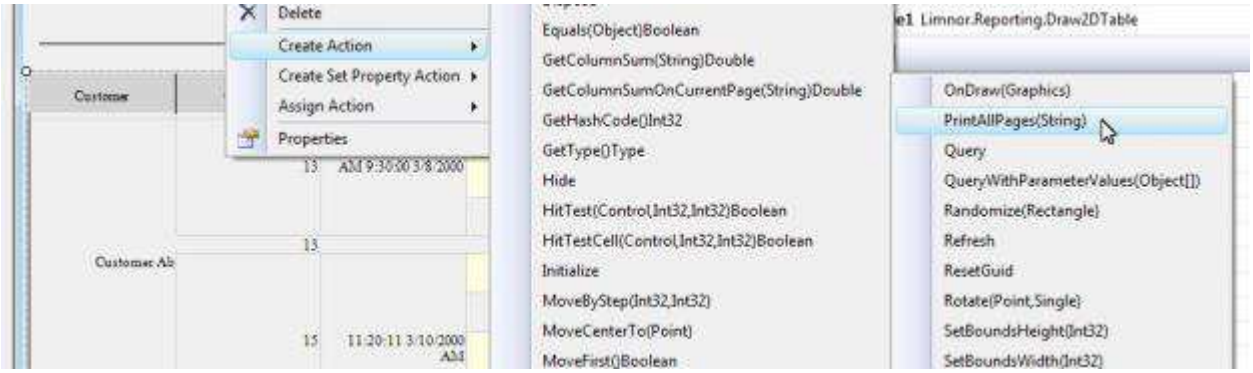
12.7 Navigate through the report pages

MoveNext, MovePrevious, MoveFirst and MoveLast methods can be used to show different report pages at runtime



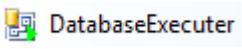
12.8 Print the report

PrintAllPages method can be used to print all report pages:

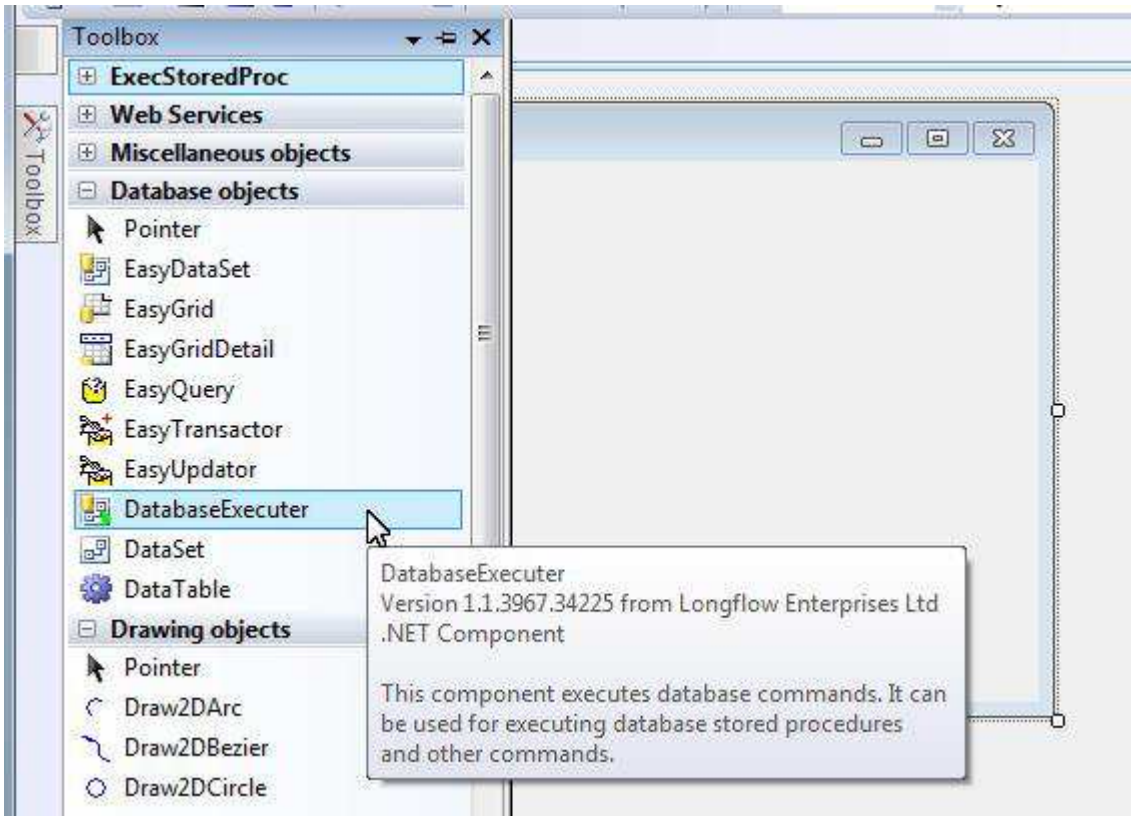


13 Execute Stored Procedures

13.1 DatabaseExecuter

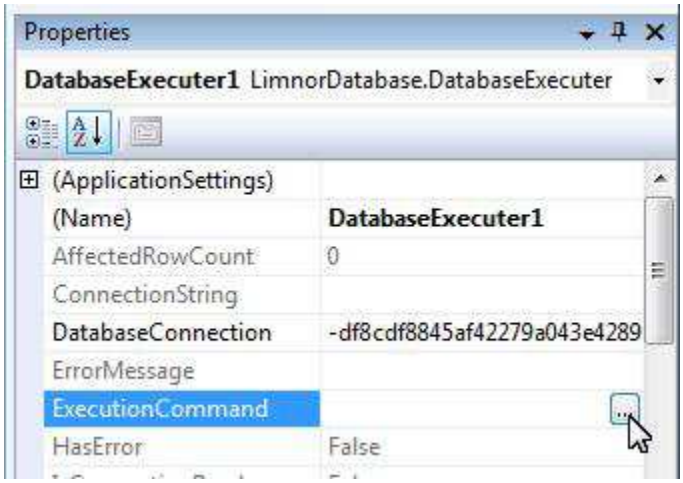
DatabaseExecuter 

can be used to execute stored-procedures. Drop it to a form to use it. It is a sub class of EasyUpdator and can be used in a similar way as using an EasyUpdator, including using it in a transaction.

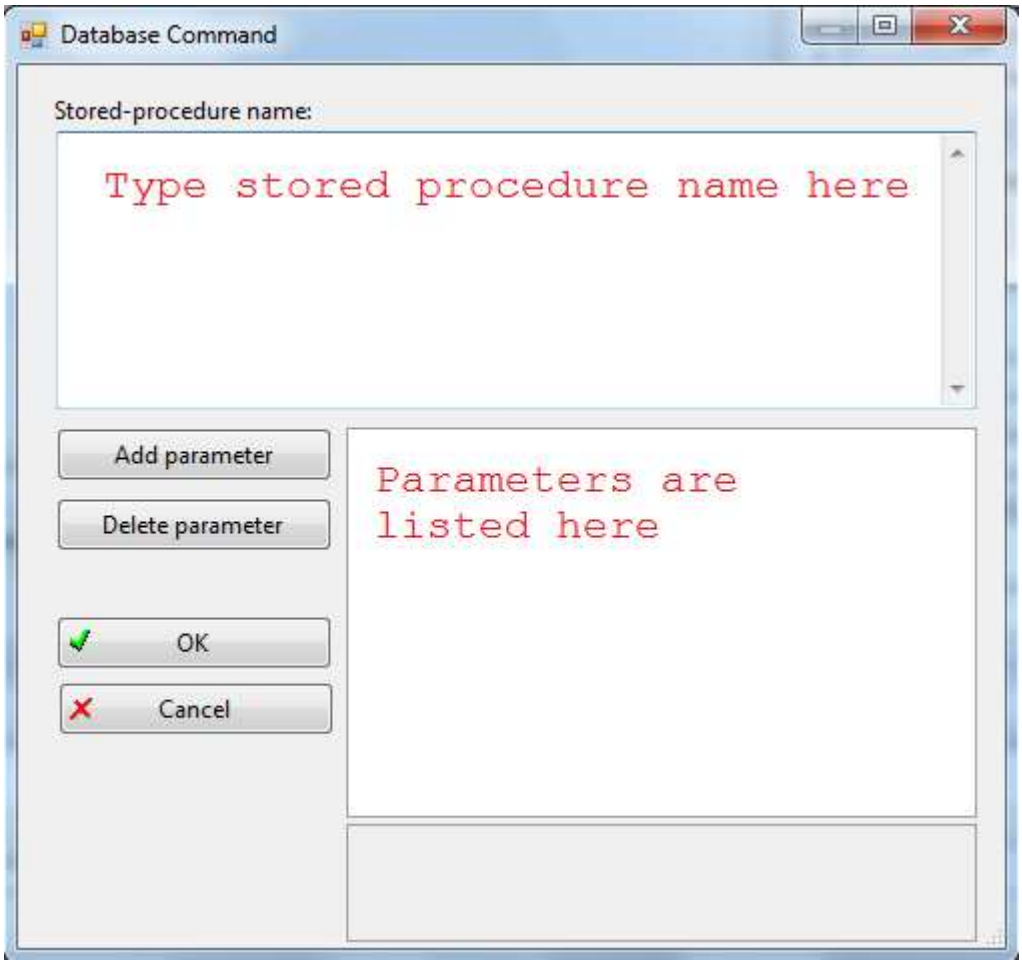


13.2 Specify Command to be executed

Edit the **ExecutionCommand** to specify the stored-procedure and parameters:



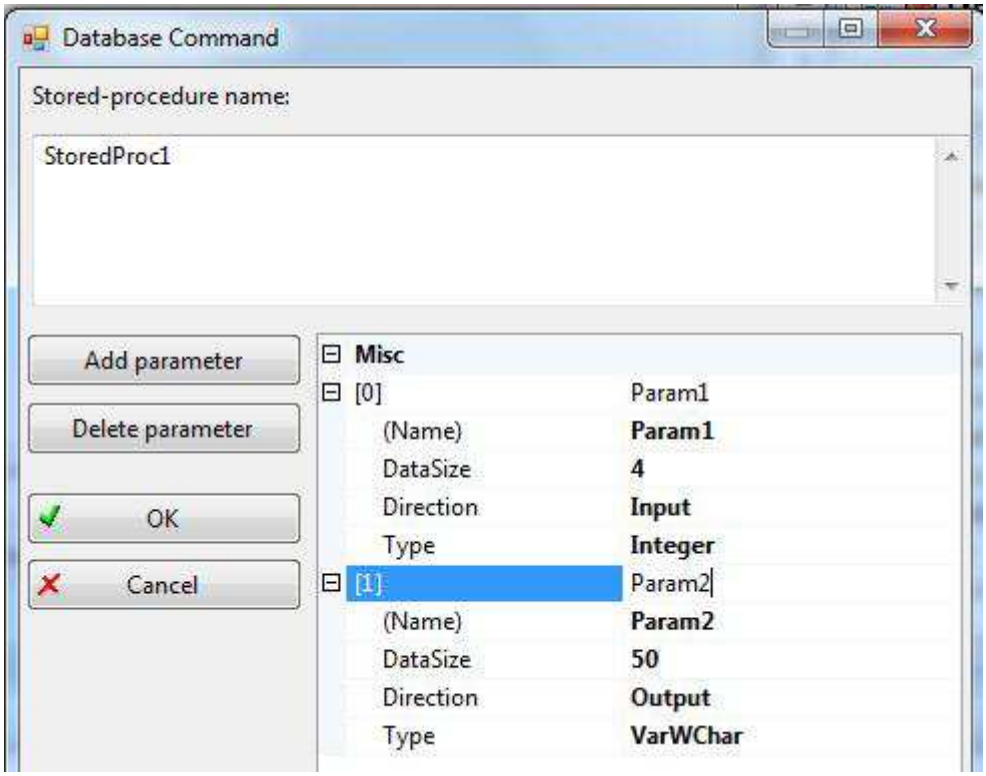
A dialogue appears for specifying stored procedure name and parameters:



Suppose we have such a stored-procedure in the database:

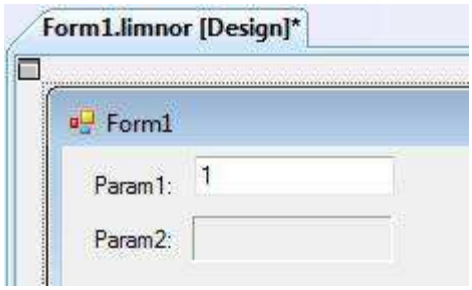
```
CREATE PROCEDURE StoredProc1
      @Param1 int,
      @Param2 nvarchar(50) output
AS
BEGIN

      SELECT * FROM Table_1 WHERE [ID] > @Param1;
      SELECT * FROM Table_2;

    SET @Param2 = N'Test return';
END
```

To execute this stored-procedure, specify its name and parameters:

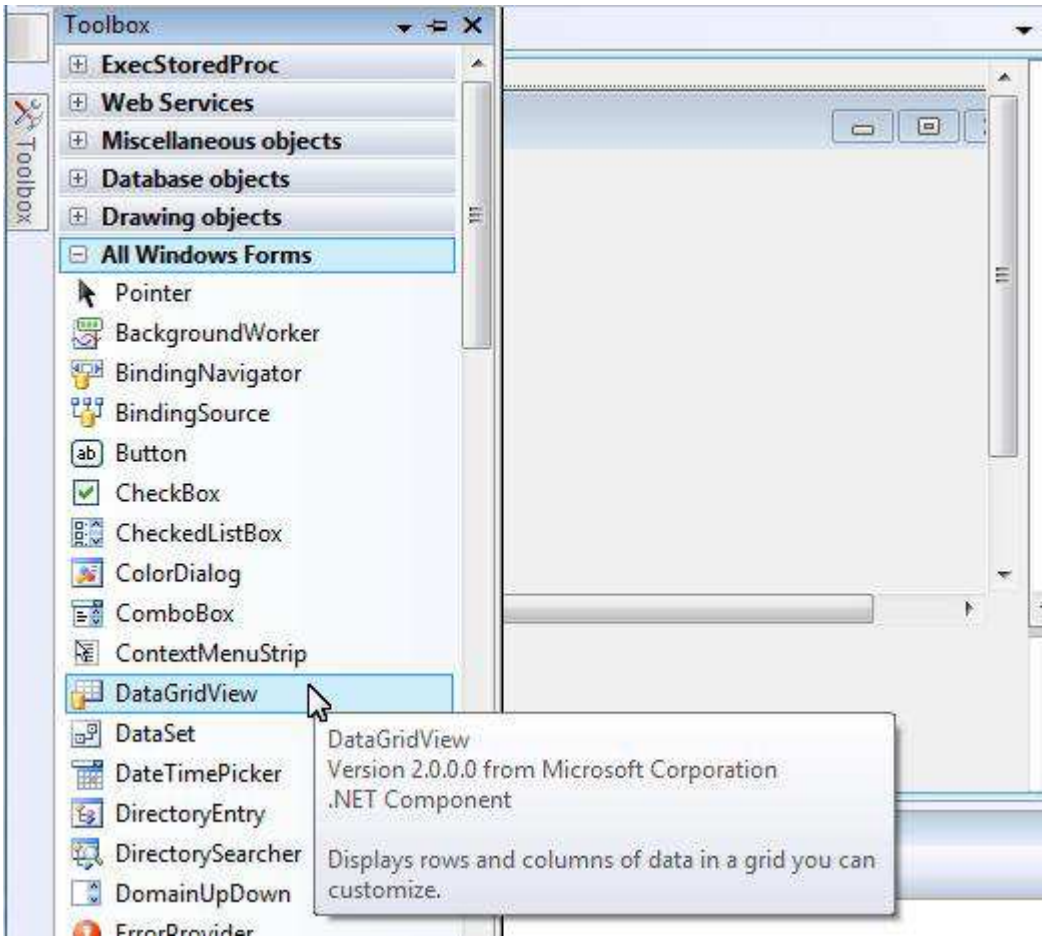


Note that the second parameter is an output parameter and thus we need to specify its direction as Output.

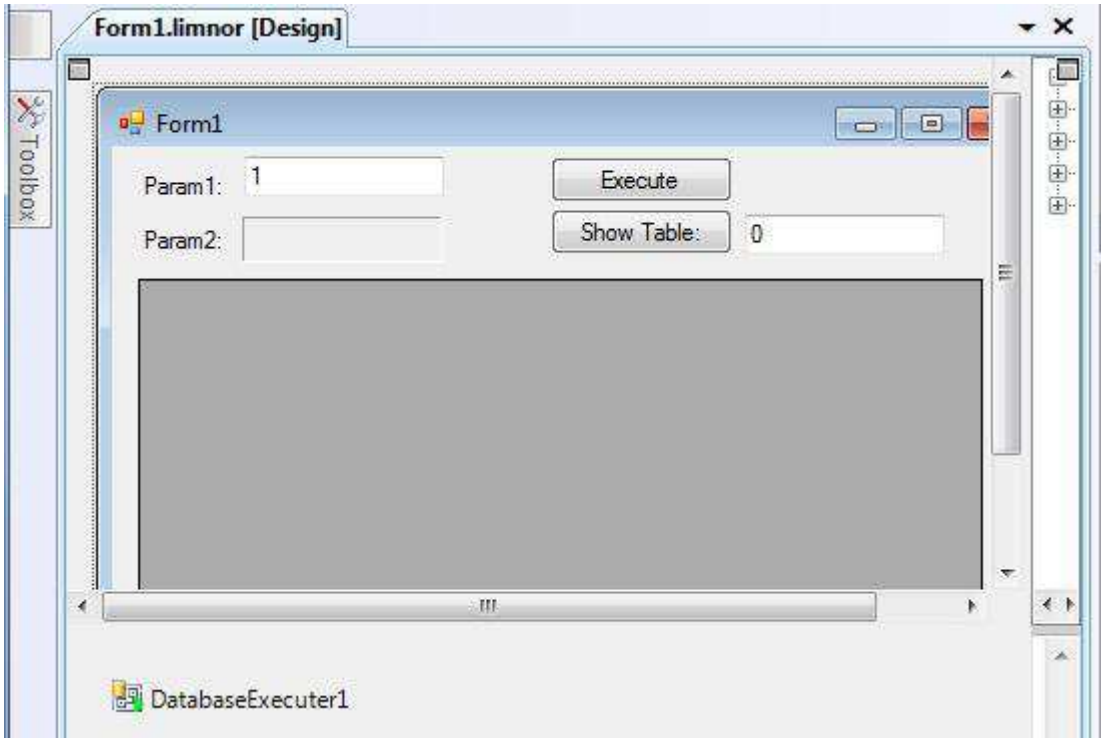
We use a text box to let the user enter value for Param1; use a Label to display the result of Param2.



The stored-procedure returns two tables of data. We may use a DataGridView to show a table of data:



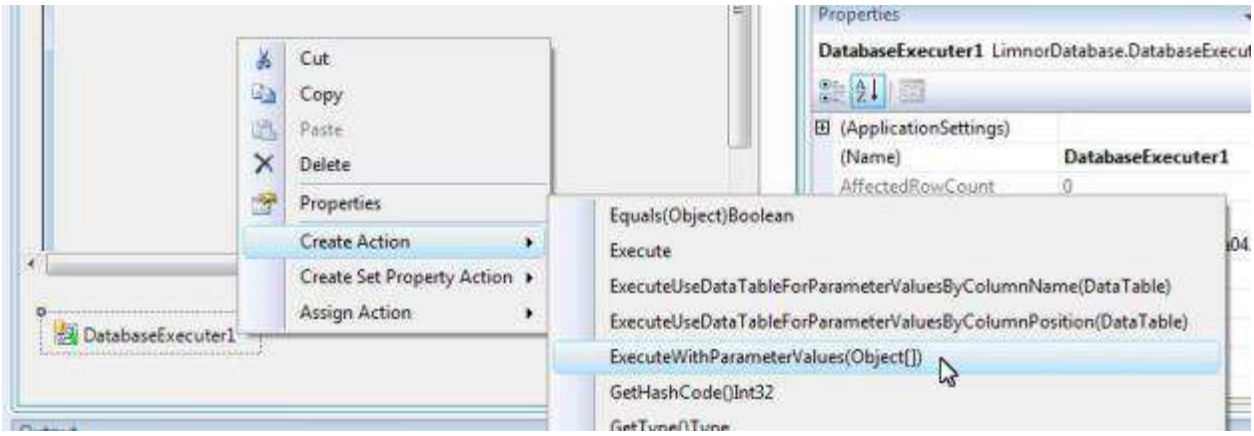
We may use a text box to let the user indicate which table to be displayed in the DataGridView:



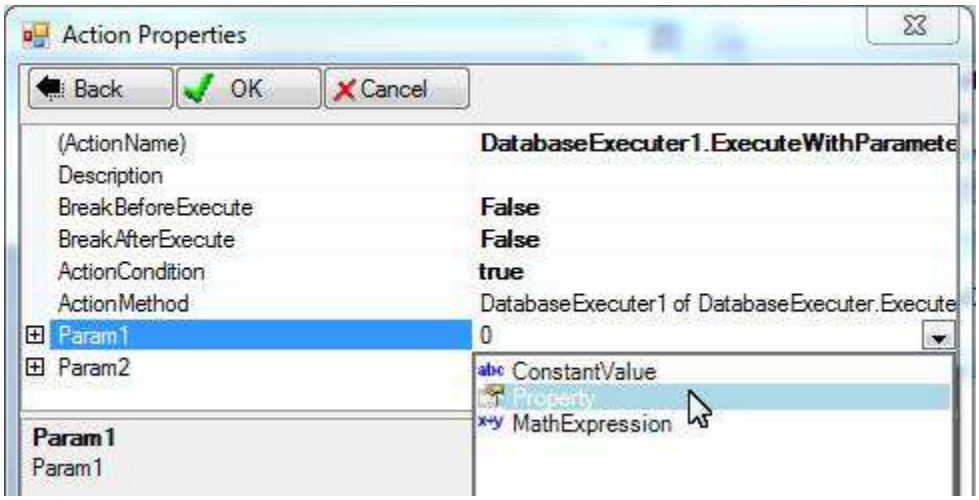
13.3 Execute the command

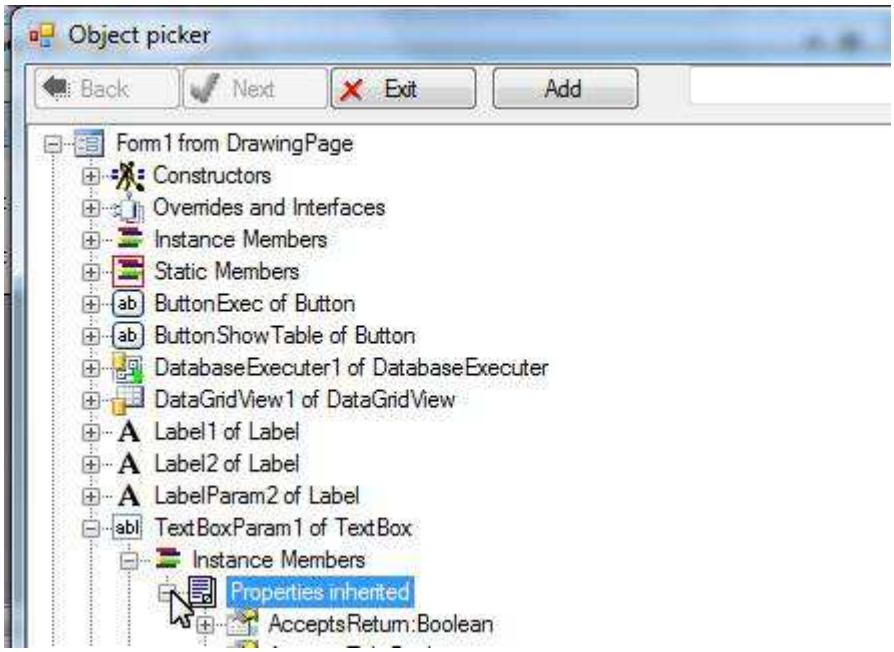
Executing the command is in the same way as using EasyUpdator: by creating `Execute`, `ExecuteWithParameterValues`, `ExecuteUseDataTableForParameterValuesByColumnPosition`, or `ExecuteUseDataTableForParameterValuesByColumnName` actions.

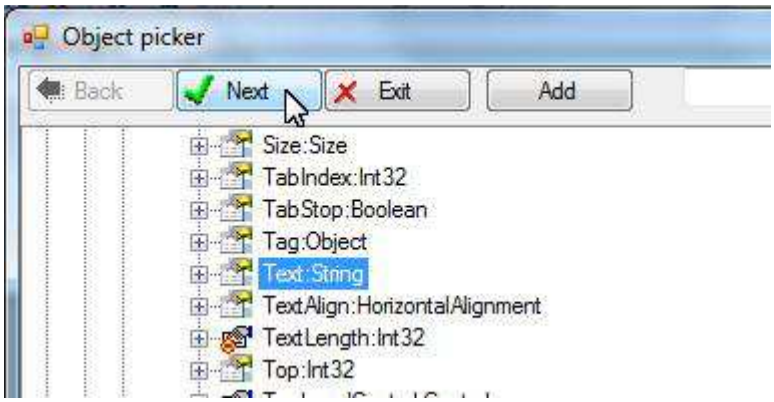
In this sample we'll create an `ExecuteWithParameterValues` action. Right-click the DatabaseExecuter component; choose "Create action"; choose "`ExecuteWithParameterValues`":



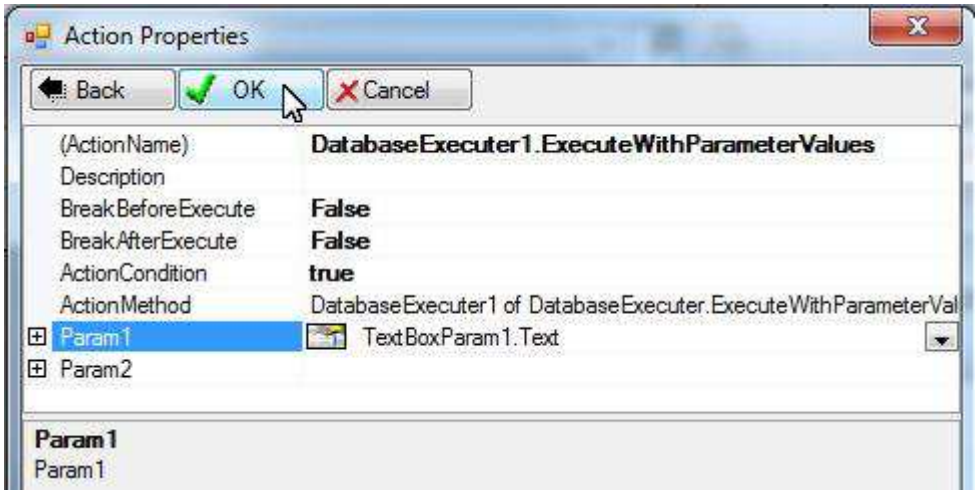
For Param1, choose "Property" to use the Text property of the text box:



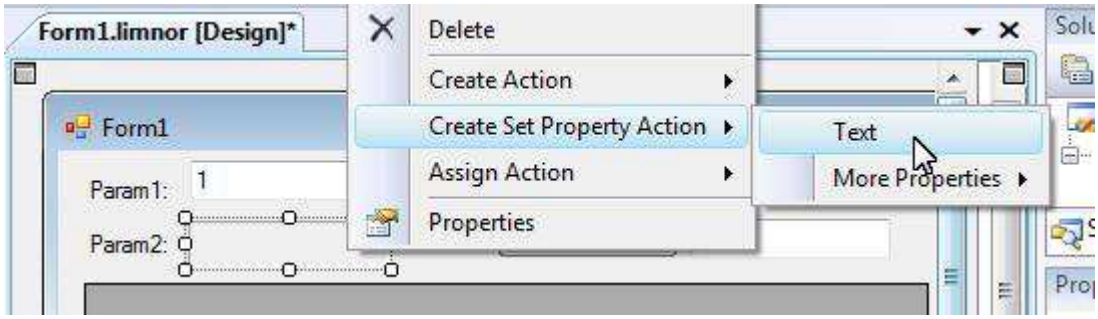


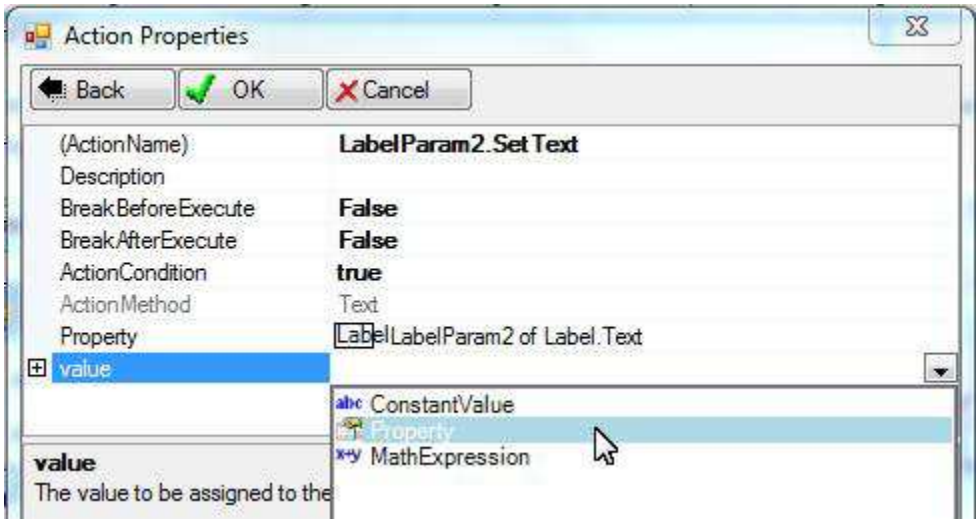


This is the action to execute the stored-procedure:

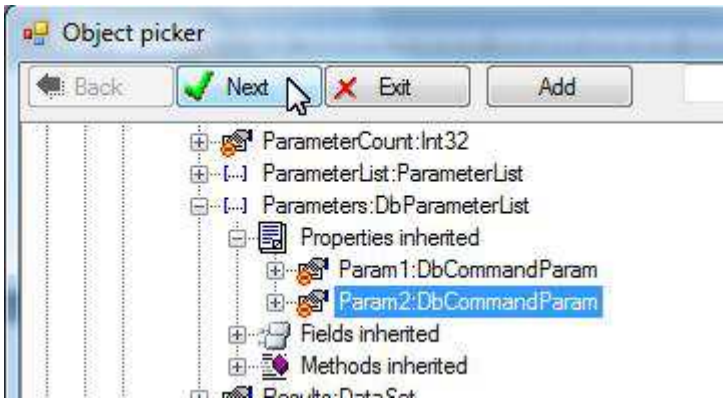


We want to display the output parameter Param2 in a label. Right-click the label; choose "Create set property action"; choose "Text" property:

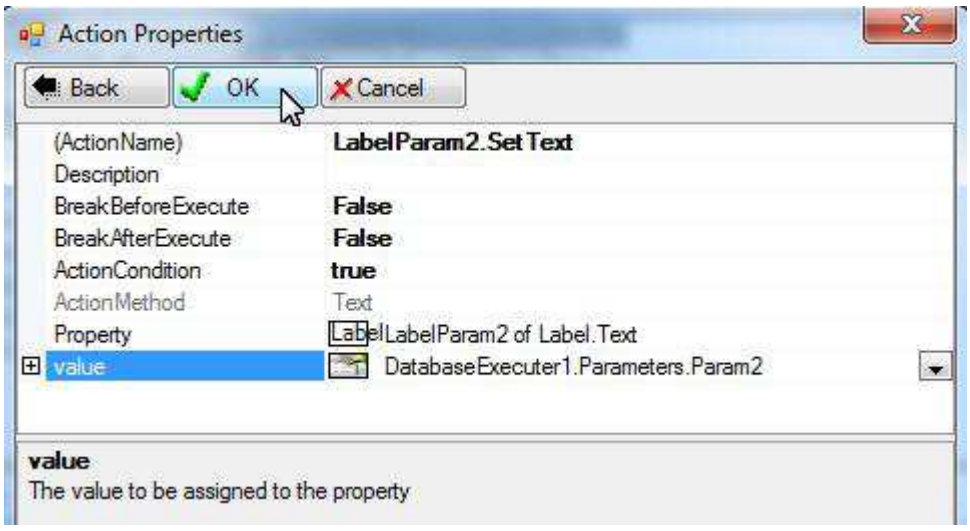




Select Param2:

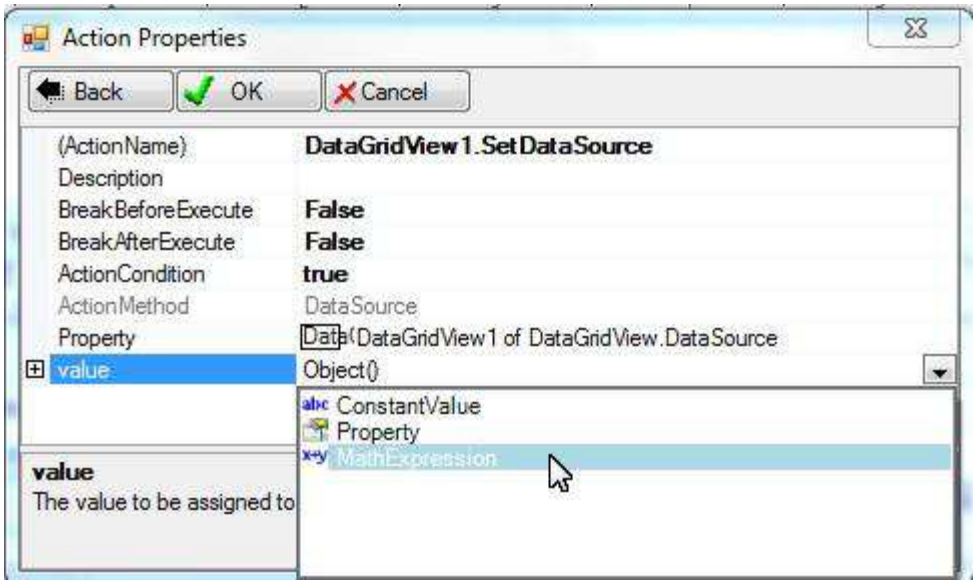


This is the action to display the second parameter in the label:

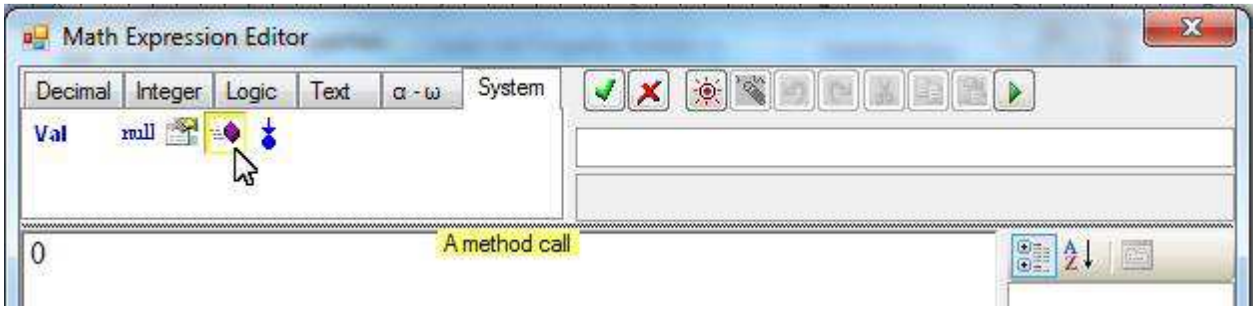


To display a table in the DataGridView we may set its DataSource property. Right-click the DataGridView; choose “Create Set Property Action”; choose “DataSource” property:

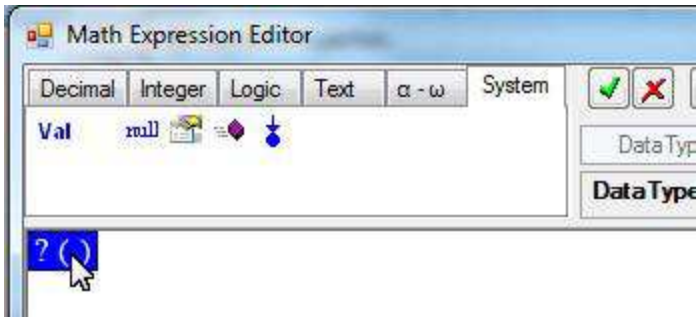
Choose “Math Expression” to fetch a collection item from the Tables of the Results:



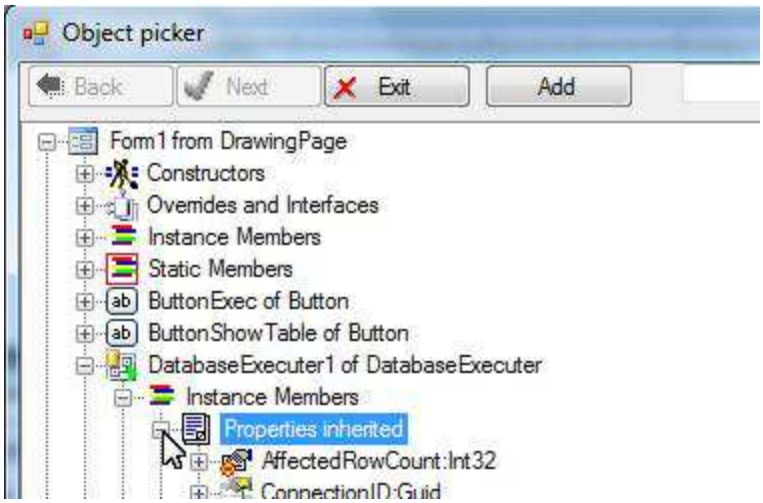
Click the Method icon:



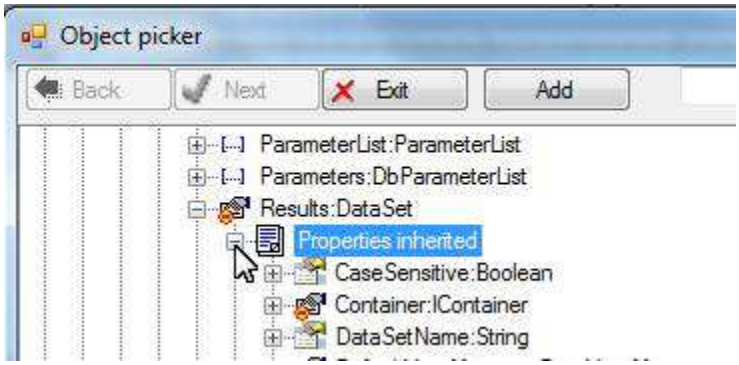
Double-click the Method element:



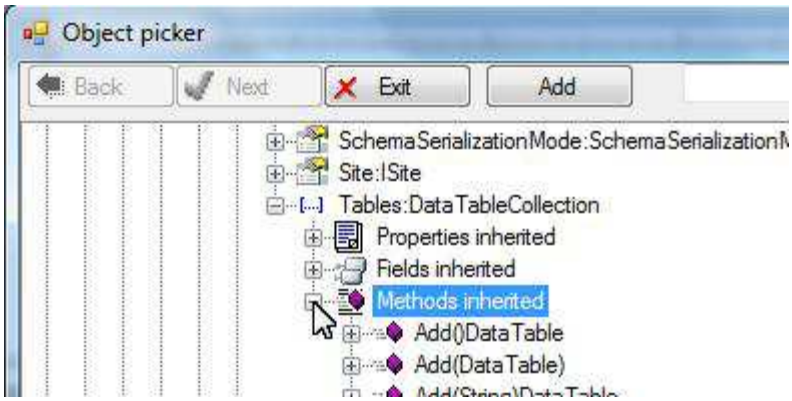
Find the Results property:

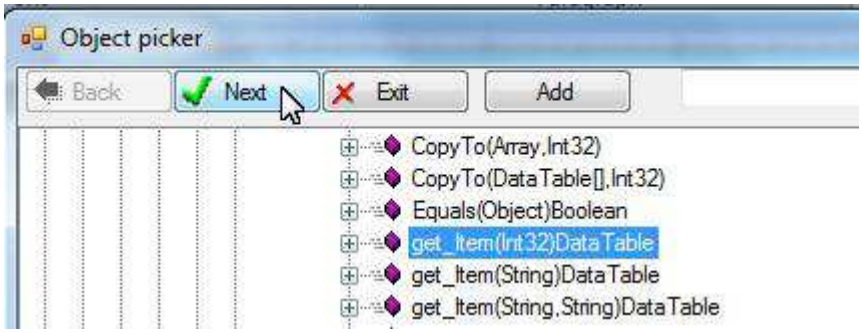


Find the Tables property of the Results:



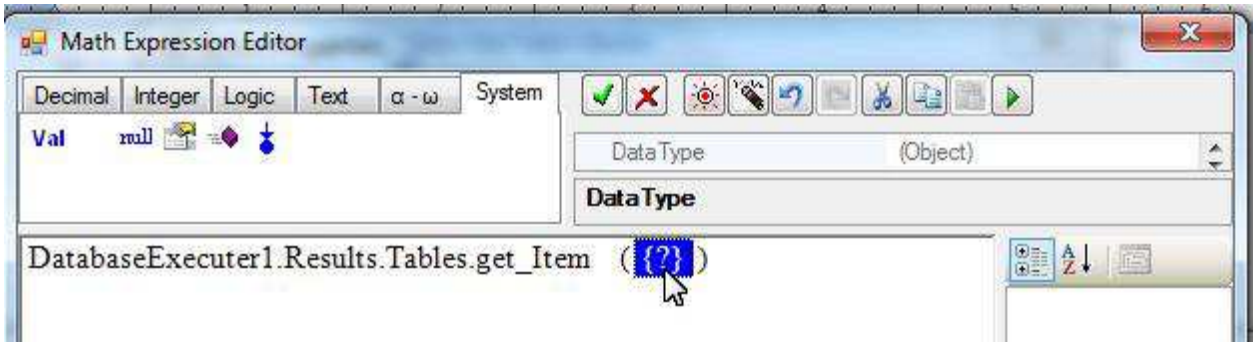
Find the get_Item method of the Tables property:



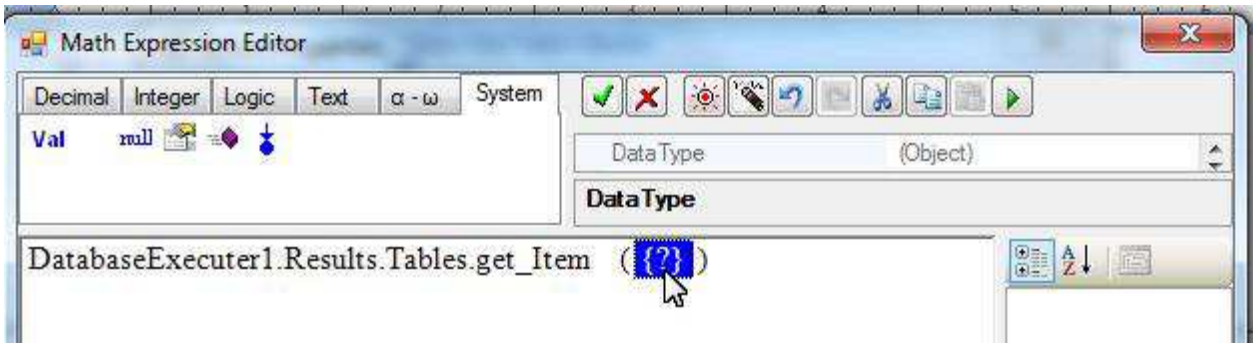


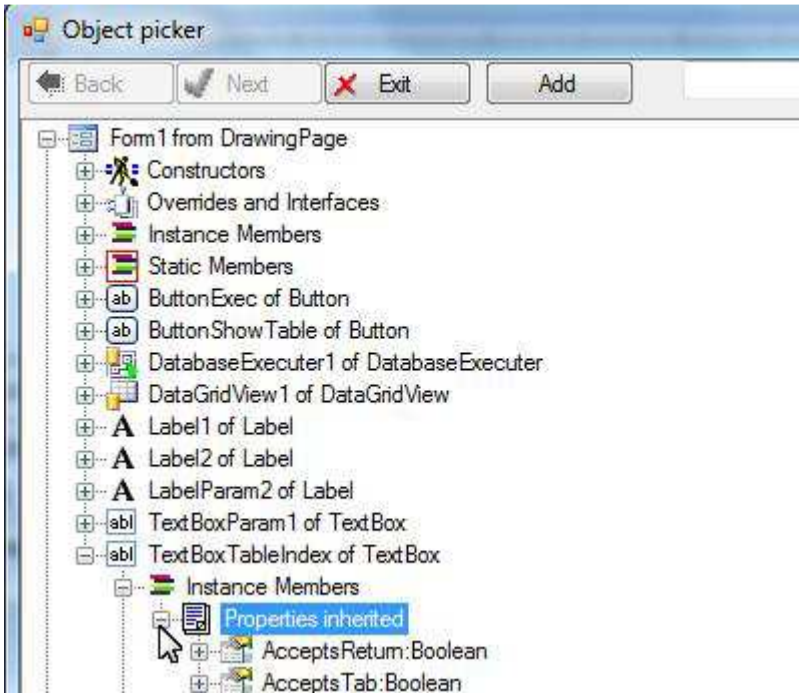
Select the method parameter; click the Property icon:

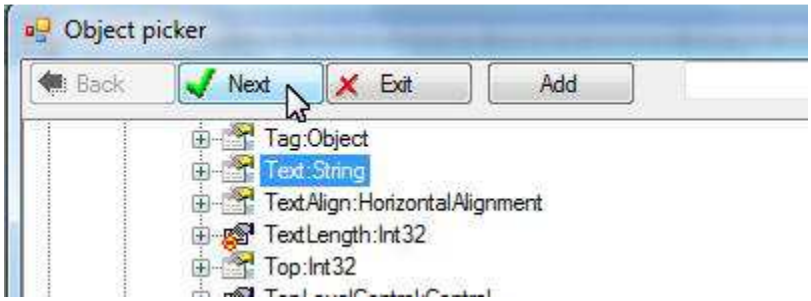
Double-click the Property element:



Select the Text property of the text box indicating the index of the table:

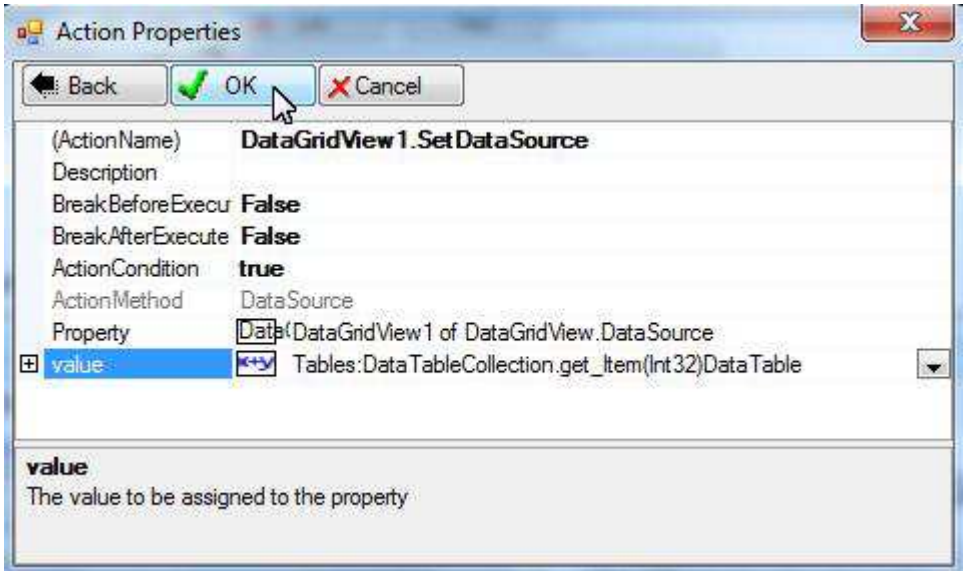




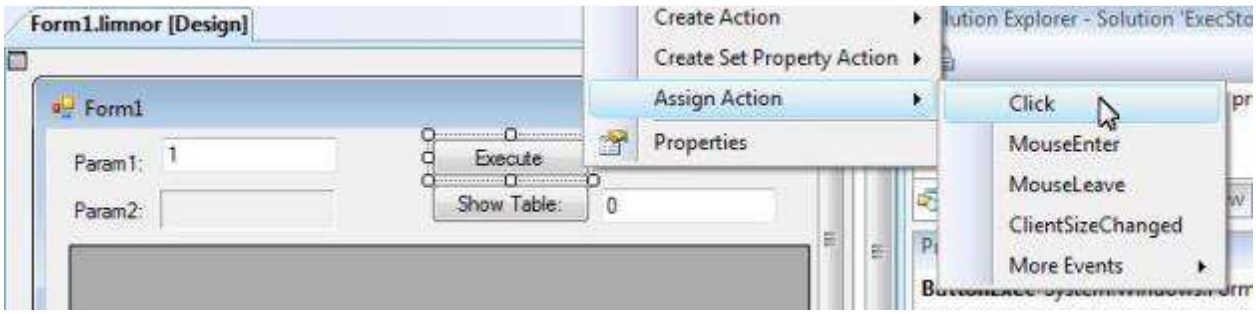


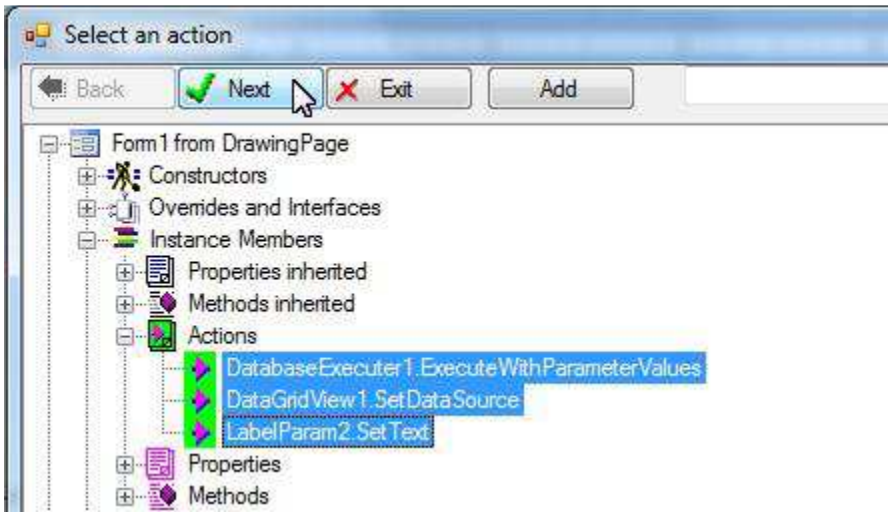
This math expression gets the table from the Results:

This action displays the table in the DataGridView:

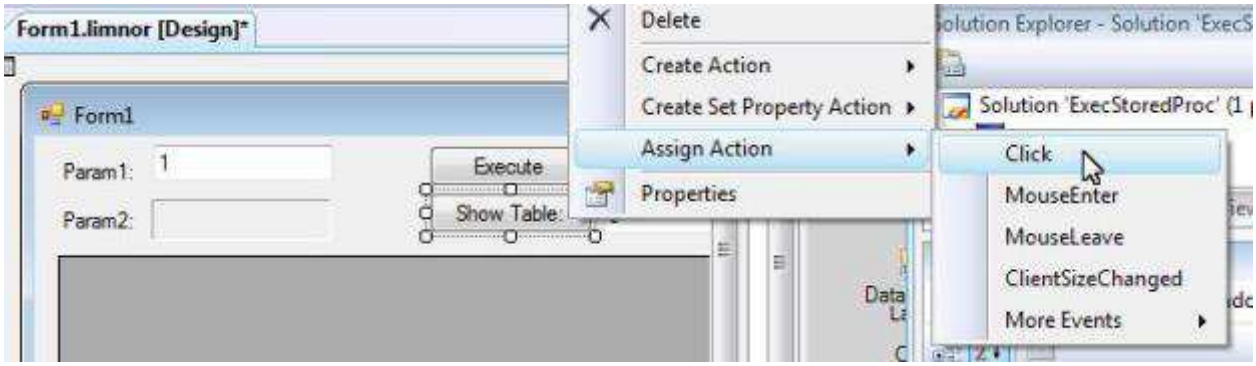


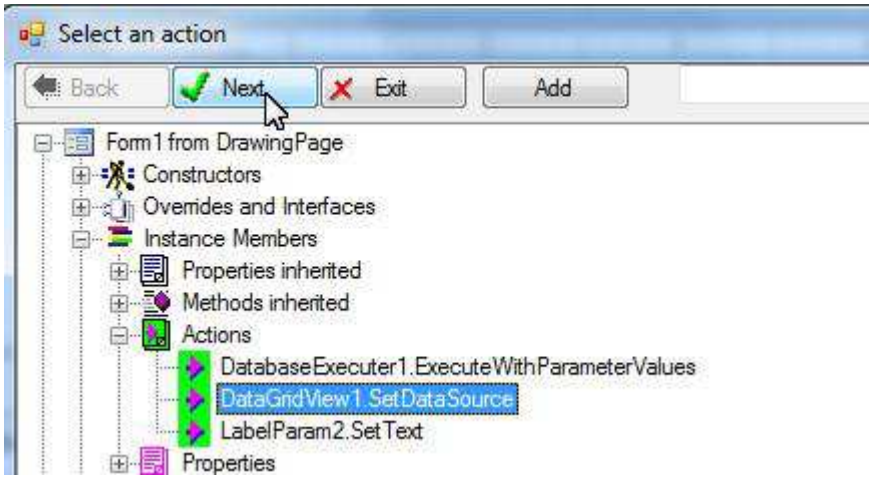
We may assign the actions to the button:



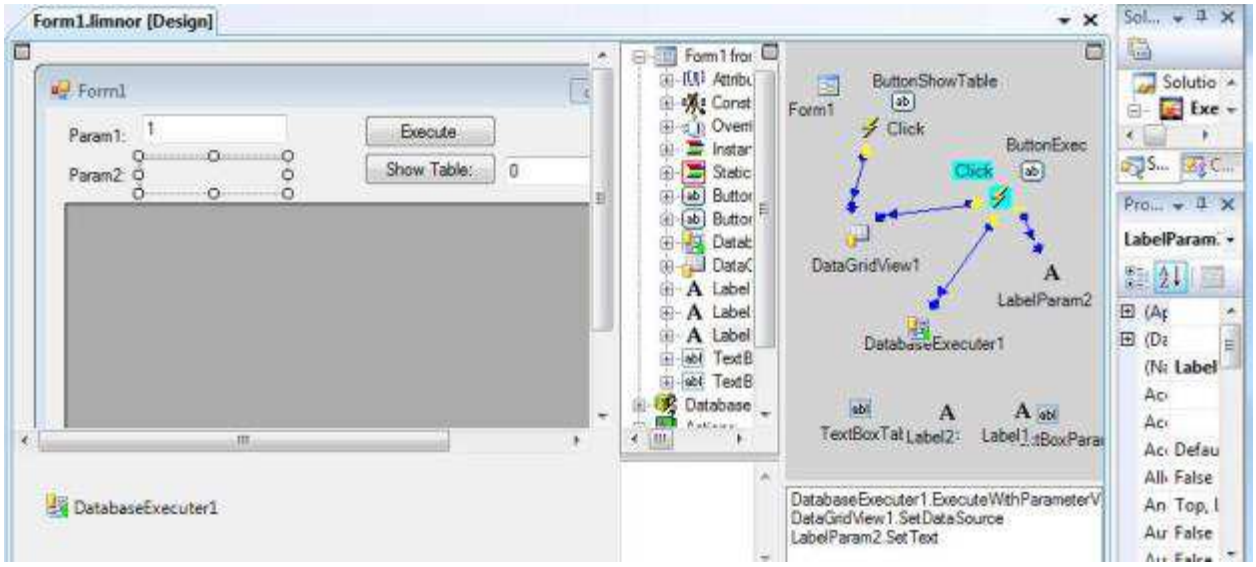


We may assign action DataGridView1.SetDataSource to the “Show table” button:



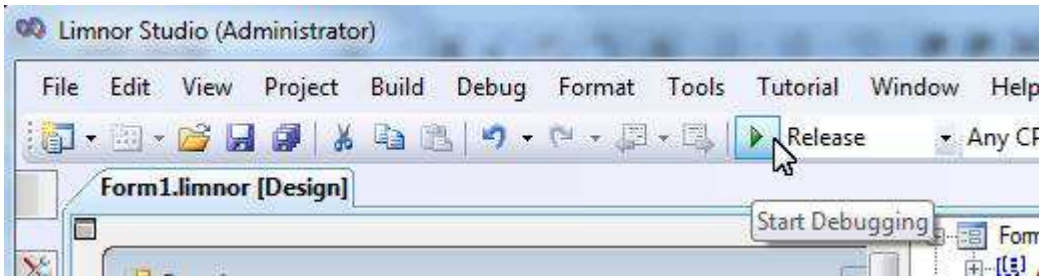


This is the event action assignments:

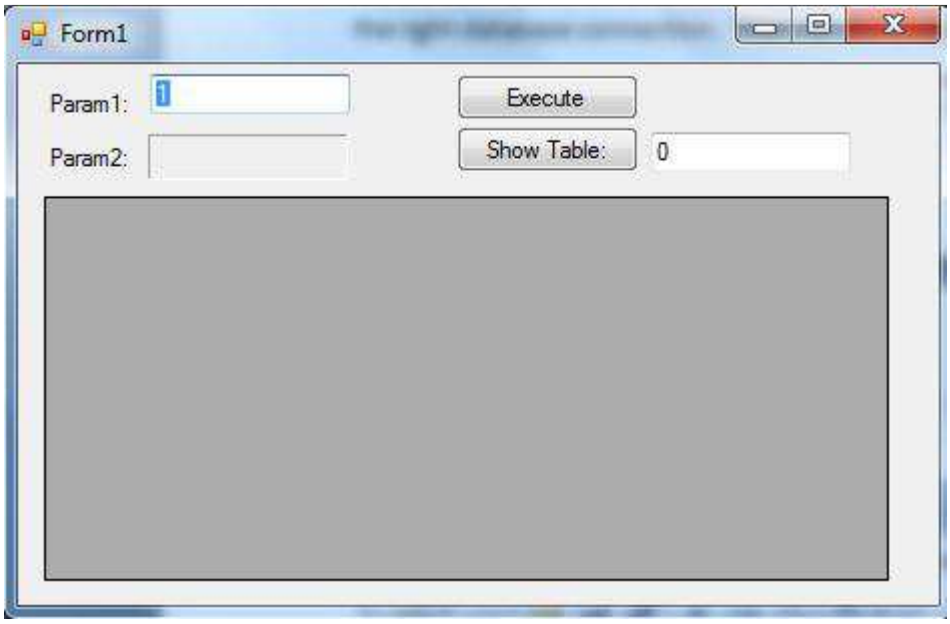


13.4 Test

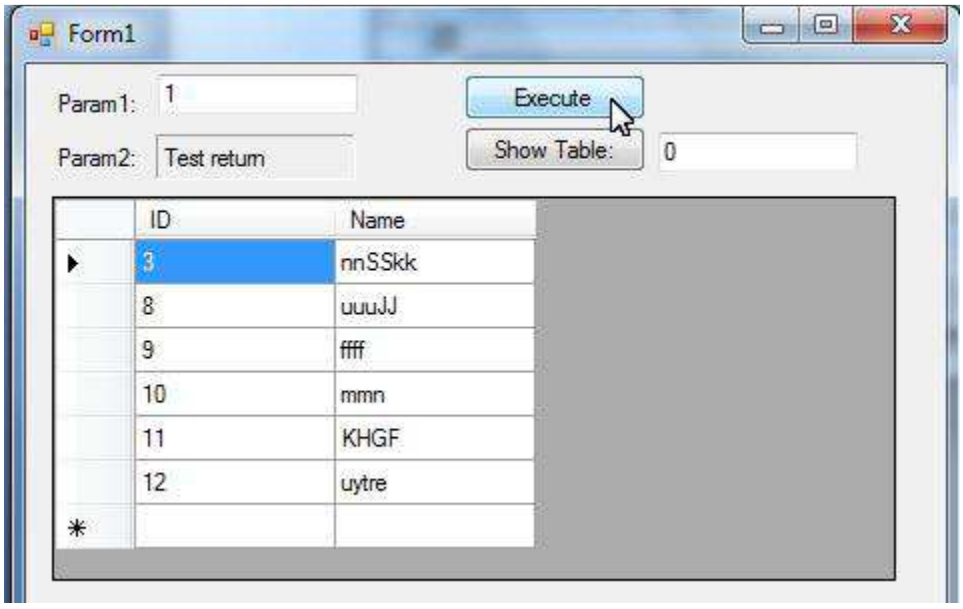
Before testing, we need to set the DatabaseConnection property of the DatabaseExecuter component to the right database connection. Then run the project:



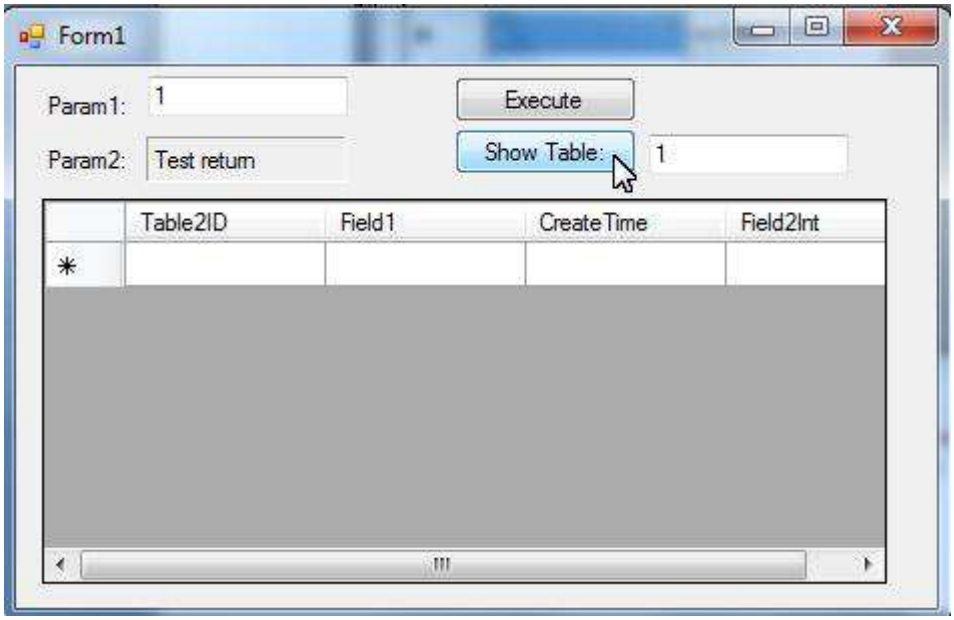
The form appears:



Click button “Execute”:



Show the second table:



14 Feedback

Please send your feedback to support@limnor.com

