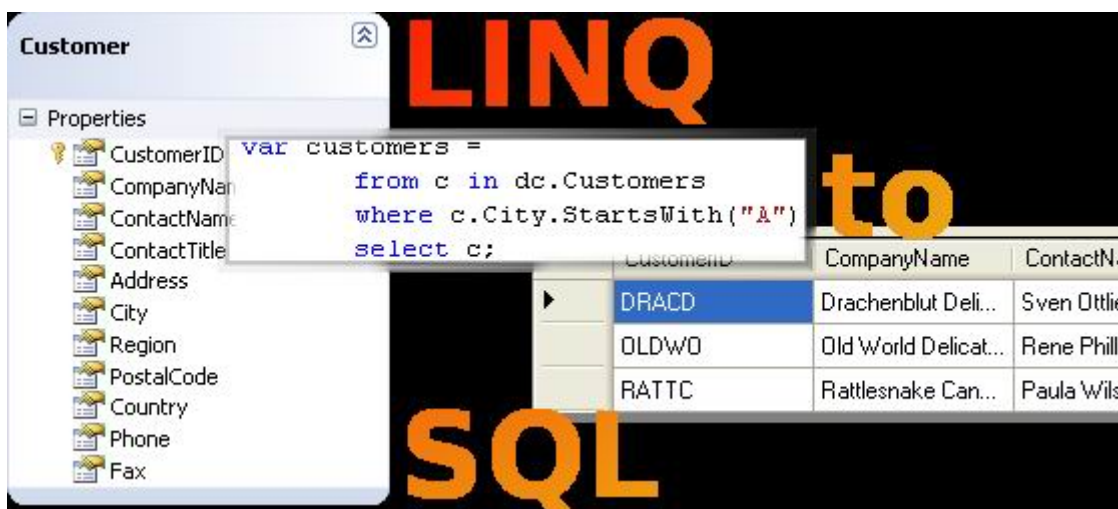


[articles](#) [Q&A](#) [forums](#) [stuff](#) [lounge](#) [?](#)Search for articles, questions, 

LINQ to SQL

**Le Sourcier**10 Dec 2007 [CPOL](#)Rate me:  4.66/5 (42 votes)

How to use LINQ to SQL



Contents

- [Introduction](#)
- [Background](#)
- [What is LINQ to SQL](#)
- [How to Use LINQ to SQL](#)
 - [Create a New C# Project](#)
 - [Understanding DataContext](#)
 - [Understanding the var Keyword](#)
 - [Querying the Database](#)
 - [Insert, Update and Delete](#)
 - [IntelliSense](#)
 - [Order of Operations](#)

- Enhanced Features
 - Deferred Loading
 - Manage Conflicts
- Points of Interest
- Other LINQs

Introduction

This article demonstrates what **LINQ to SQL** is and how to use its basic functionality. I found this new feature amazing because it really simplifies a developer's debugging work and offers many new ways of coding applications.

Background

For this article, I have used the Northwind database sample from Microsoft. The database is included in the ZIP file or can be found on the [Microsoft website](#).

What is LINQ to SQL

The LINQ Project is a codename for a set of extensions to the .NET Framework that encompasses language-integrated query, set and transform operations. It extends C# and Visual Basic with native language syntax for queries. It also provides class libraries to take advantage of these capabilities. For more general details, refer to the [Microsoft LINQ Project page](#).

LINQ to SQL is a new system which allows you to easily map tables, views and stored procedures from your SQL server. Moreover, LINQ to SQL helps developers in mapping and requesting a database with its simple and **SQL-like** language. It is not the ADO.NET replacement, but more an extension that provides new features.

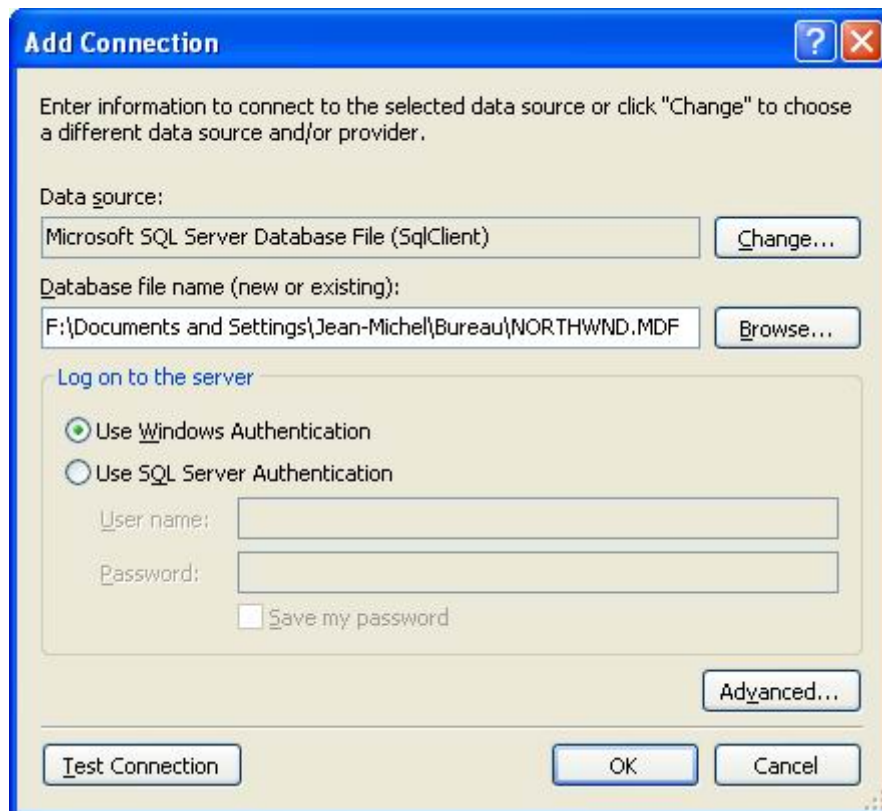


How to Use LINQ to SQL

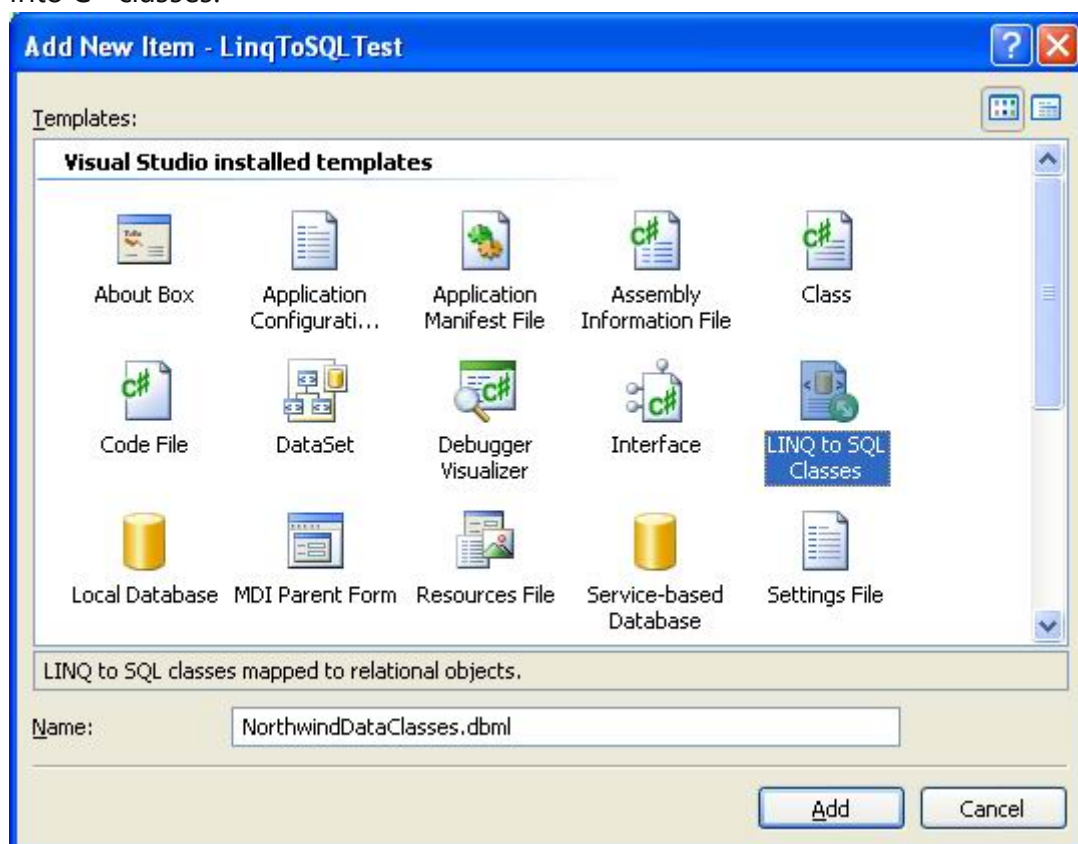
Create a New C# Project

In this section I'll show you how to use LINQ to SQL from the start, at project creation.

1. Create a new Windows form application with Visual C# 2008.
2. Make a new "Data Connection" with the SQL Server 2005 *northwind.mdf* file.

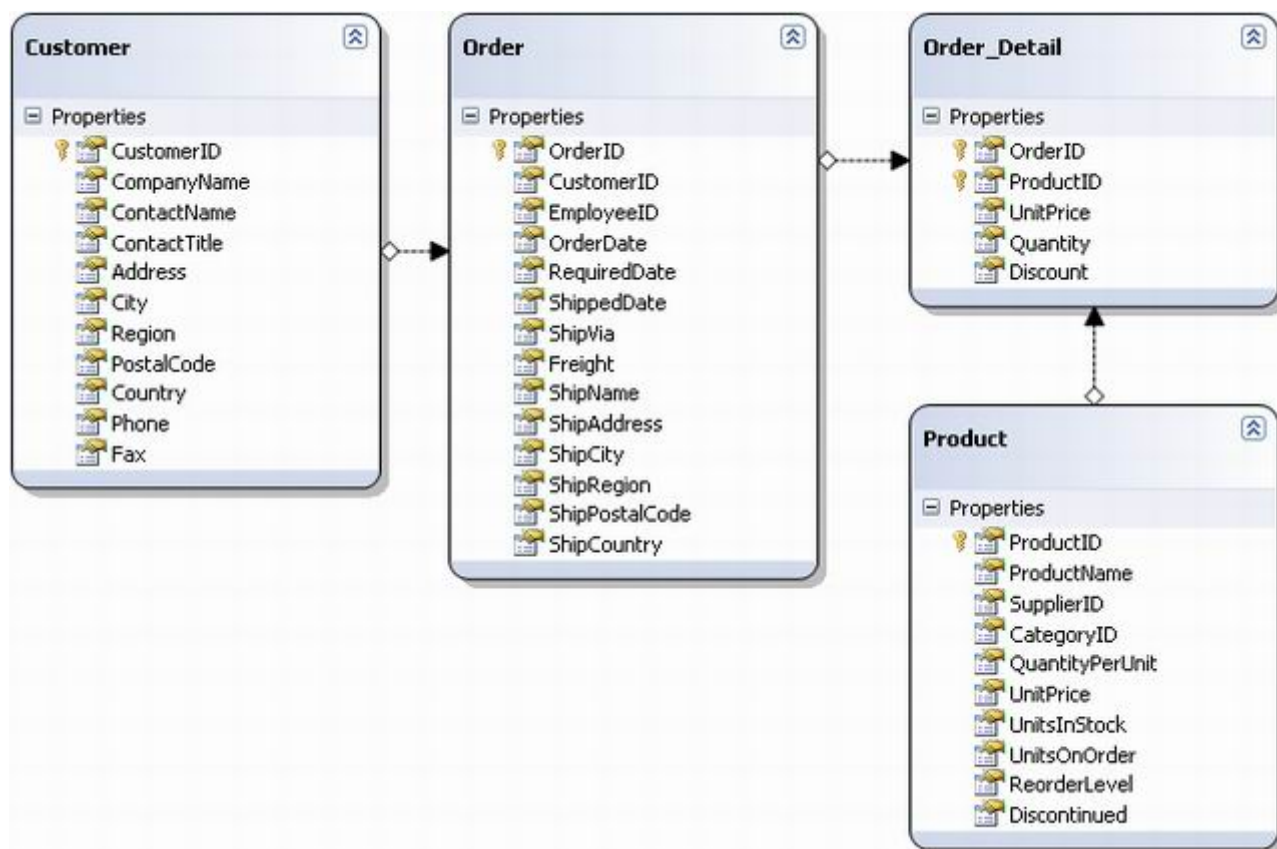


3. Add a new item to your project and choose "LINQ to SQL Classes." Name it *NorthwindDataClasses.dbml*. This new DBML file will contain the mapping of SQL Server tables into C# classes.



The Object Relational Designer is a design surface that maps a database table into a C# class. To do that, just drag and drop tables from the database explorer into the designer. The designer automatically displays the tables in a UML way and represents the relationship between them. For

example, I have dragged and dropped the four tables Customers, Order, Order_Detail and Product, as shown below:



In *NorthwindDataClasses.designer.cs* (under *NorthwindDataClasses.dbml* from the project explorer), you will find definitions for all classes corresponding to tables like this:

SQL	LINQ to SQL O/R Designer
Table name	Class name
Columns	Attributes
Relations	EntitySet and EntityRef
Stored procedures	Methods

C#



```
[Table(Name="dbo.Customers")]
public partial class Customer : INotifyPropertyChanging, INotifyPropertyChanged
{
    private static PropertyChangingEventArgs emptyChangingEventArgs =
        new PropertyChangingEventArgs(String.Empty);
    private string _CustomerID;
    private string _CompanyName;
    private string _ContactName;
    private string _ContactTitle;
    private string _Address;
    private string _City;
    private string _Region;
    private string _PostalCode;
    private string _Country;
    private string _Phone;
```

```
private string _Fax;
private EntitySet<Order> _Orders;

//.....

}
```

Understanding DataContext

dataContext is a class that gives direct access to C# classes, database connections, etc. This class is generated when the designer is saved. For a file named *NorthwindDataClasses.dbml*, the class **NorthwindDataClassesDataContext** is automatically generated. It contains the definition of tables and stored procedures.

Understanding the var Keyword

This keyword is used when you do not know the type of the variable. Visual Studio 2008 automatically chooses the appropriate type of data and so does IntelliSense!

Examples:

C#



```
var anInteger = 5;

var aString = "a string";

var unknown_type = new MyClass();
```

Querying the Database

Once the database is modeled through the designer, it can easily be used to make queries.

Query the Customers from Database

C#



```
//creating the datacontext instance

NorthwindDataClassesDataContext dc = new NorthwindDataClassesDataContext();

//Sample 1 : query all customers

var customers =
    from c in dc.Customers
    select c;

//display query    result in a dataGridView

dataGridViewResult.DataSource = customers.ToList();
```

Query the Customers with a Simple Statement

C#



```
//Sample 2 : query customers which country is UK

//constructing the query

var customers_from_uk =
    from c in dc.Customers
    where c.Country == "UK"
    select c;
```

Query Specified Columns Only, Returning a Collection of a Specified Class

Use a very simple class definition.

C#



```
public class CMyClassTwoStrings
{
    public CMyClassTwoStrings(string name, string country)
    {
        m_name = name;
        m_country = country;
    }

    public string m_name;
    public string m_country;
}
```

For example:

C#



```
//Sample 4a : query customers(name and country) which contact name starts with a 'A'

//using specific class

var customers_name_starts_a_2col_in_specific_class =
    from c in dc.Customers
    where c.ContactName.StartsWith("A")
    select new CMyClassTwoStrings (c.ContactName, c.Country );

//using the returned collection (using foreach in

//console output to really see the differences with the dataGridView way.)

foreach (CMyClassTwoStrings a in customers_name_starts_a_2col_in_specific_class)
    Console.WriteLine(a.m_name + " " + a.m_country);
```

Query Specified Columns Only, Returning a Collection of an Undefined Class

C#



```
//Sample 4b : query customers(name and country) which contact name starts with a 'A'

//using anonymous class
```

```

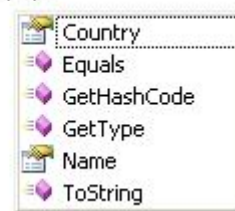
var customers_name_starts_a_2col_in_anonymous_class =
    from c in dc.Customers
    where c.ContactName.StartsWith("A")
    select new {
        Name = c.ContactName,           //naming the column Name
        Country = c.Country             //naming the column Country
    };
foreach (var a in customers_name_starts_a_2col_in_anonymous_class)
    Console.WriteLine(a.Name + " " + a.Country);

```

This example demonstrates how to use an anonymous class that is (in this example) composed of two strings. The aim of this feature is to create a new class for temporary storage that the developer does not want (or need) to declare. It may be useful in some cases where the declaration of class is used only for storage.

For example, in the sample 4a, the class **CMyClassTwoStrings** is used only to create the interface between the query engine and the output in the console. It is not used anywhere else and is a loss of time. This new way of writing enables the developer to create temporary classes with an unlimited number of attributes of any type. Every attribute is named, either by specifying the name with **Name = c.ContactName** or by leaving the attribute without, i.e. **Name =**. IntelliSense also works with anonymous classes!

```
customers_name_starts_a_2col_in_anonymous_class.ElementAt(0).
```



Query Multiple Tables

C#



```

//Sample 5 : query customers and products

//(it makes a cross product it do not represent anything else than a query

var customers_and_product =
    from c in dc.Customers
    from p in dc.Products
    where c.ContactName.StartsWith("A") && p.ProductName.StartsWith("P")
    select new { Name = c.ContactName, Product = p.ProductName };

```

The resulting collection is the cross product between all contact names starting with "A" and all products starting with "P."

Query with Tables Joined

C#



```
//Sample 6 : query customers and orders
```

```
var customers_and_orders =  
    from c in dc.Customers  
    from p in dc.Orders  
    where c.CustomerID == p.CustomerID  
    select new { c.ContactName, p.OrderID};
```

This example demonstrates how to specify the relation between tables' joins on an attribute.

Query with Tables Joined through entityref

C#



```
//Sample 7 : query customers and orders with entityref
```

```
var customers_and_orders_entityref =  
    from or in dc.Orders  
    select new {  
        Name = or.Customer.ContactName,  
        OrderId = or.OrderID,  
        OrderDate = or.OrderDate  
    };
```

In this example, the **entityref** property is used. The class orders have an attribute named **Customer** that refers to the customer who realizes the order. It is just a pointer to one instance of the class **Customer**. This attribute gives us direct access to customer properties. The advantage of this feature is that the developer does not need to know exactly how tables are joined and access to attached data is immediate.

Query in the Old Way: with SQL as String

As you may want to execute SQL that is not yet supported by LINQ to SQL, a way to execute SQL queries in the old way is available.

C#



```
//Sample 8 : execute SQL queries
```

```
dc.ExecuteCommand("UPDATE Customers SET PostalCode='05024' where CustomerId='ALFKI'");
```

Insert, Update and Delete Rows from Database

LINQ to SQL provides a new way of managing data into database. The three SQL statements **INSERT**, **DELETE** and **UPDATE** are implemented, but using them is not visible.

Update Statement

C#




```
//Sample 9 : updating data

var customers_in_paris =
    from c in dc.Customers
    where c.City.StartsWith("Paris")
    select c;

foreach (var cust in customers_in_paris)
    cust.City = "PARIS";

//modification to database are applied when SubmitChanges is called.

dc.SubmitChanges();
```

To make modifications to a database, just modify any relevant object properties and call the method **SubmitChanges()**.

Insert Statement

To insert a new entry into the database, you just have to create an instance of a C# class and **Attach** it to the associated table.

C#



```
//Sample 10 : inserting data

Product newProduct = new Product();
newProduct.ProductName = "RC helicopter";

dc.Products.InsertOnSubmit(newProduct);

dc.SubmitChanges();
```

Delete Statement

Deleting data is quite easy. When requesting your database, give a collection of data. Then just call **DeleteOnSubmit** (or **DeleteAllOnSubmit**) to delete the specified items.

C#



```
//Sample 11 : deleting data

var products_to_delete =
    from p in dc.Products
    where p.ProductName.Contains("helicopter")
    select p;

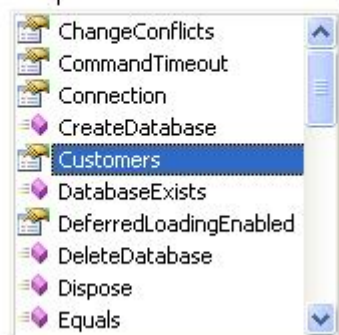
dc.Products.DeleteAllOnSubmit(products_to_delete);

dc.SubmitChanges();
```

IntelliSense

IntelliSense works into the query definition and can increase developer productivity. It's very interesting because it pops up on **DataContext**, tables and attributes. In this first example, IntelliSense shows the list of tables mapped from the database, the connection instance and a lot of other properties.

```
//Sample 2 : query customers which country is UK
var customers_from_uk =
    from c in dc.
```



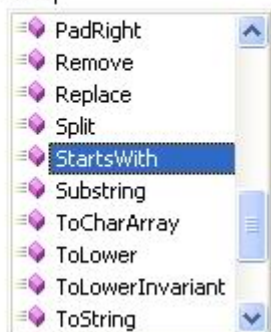
For a table, the list contains all of its columns:

```
from c in dc.Customers
where c.
```



For an attribute, it will display methods and properties depending on the type (string, integer, etc).

```
var customers_name_starts_a =
    from c in dc.Customers
    where c.ContactName.
```



Order of Operations

To use LINQ to SQL, a developer must know exactly when a query is executed. Indeed, LINQ to SQL is very powerful because the query is executed when it's required, but not at definition! In the first sample, we have this code:

C#



```
///constructing the query

var customers =
    from c in dc.Customers
    select c;
```

The query is not yet executed; it is just compiled and analysed. In fact, the query is run when the code makes an access to the customer variable, like here:

C#



```
///display query    result in a dataGridView

dataGridViewResult.DataSource = customers.ToList();
```

Enhanced Features

Other LINQ to SQL Options

LINQ to SQL supports **deferred loading options**. This functionality allows a user to modify query engine behaviour when retrieving data. One of them is the deferred loading that will load all the data of a query. As an example, a query on the Order table gives you entry to the customer properties by **entityref**. If **Datacontext.DeferredLoadingEnabled** is set at **true** (default) then the **Customer** attribute will be loaded when an access to the Order entry is made. Otherwise (when at **false**), it is not loaded. This option helps a developer when optimizing requests, data size and time for querying. There is a good example about that [here](#).

Manage Conflicts

When the function **SubmitChanges()** is executed, it starts by verifying if there is no conflict that occurs by an external modification. For a server/client application, the application must take conflicts into account in case multiple clients access the database at the same time. To implement conflict resolution, **SubmitChanges()** generates a **System.Data.Linq.ChangeConflictException** exception. The **DataContext** instance gives details about conflicts to know why exactly they throw. I wrote a basic conflict resolution, but I will not give the full details of all other possibilities because I think it should be an entire article.

C#

Shrink ▲

```
try{
    ///query the database

    var customers_in_paris_conflict =
        from c in dc.Customers
        where c.City.StartsWith("Paris")
        select c;
    foreach (var cust in customers_in_paris_conflict)
        cust.City = "PARIS";
```

```

//Make a breakpoint here and modify one customer entry

//(where City is Paris) manually (with VS for example)

//When external update is done, go on and SubmitChanges should throw.

dc.SubmitChanges();
}
catch (System.Data.Linq.ChangeConflictException)
{
    //dc.ChangeConflicts contains the list of all conflicts

    foreach (ObjectChangeConflict prob in dc.ChangeConflicts)
    {
        //there are many ways in resolving conflicts,

        //see the RefreshMode enumeration for details

        prob.Resolve(RefreshMode.KeepChanges);
    }
}

```

If you want more details about conflict resolution, I suggest you to refer to [this page](#) (in VB).

Points of Interest

As a conclusion to this article, I summarize the important points of LINQ to SQL:

- LINQ to SQL is a query language
- Query syntax is verified at build (not at runtime like old SQL queries)
- IntelliSense works with all objects of LINQ to SQL
- Making queries is quite easy (select, insert, update and delete)
- Manage tables' PK/FK relationships
- Databases are automatically mapped to C# classes and queries return a collection of C# class instances
- Conflict detection and resolution

Other LINQs ;-)

- [Microsoft LINQ Project page](#)
- [Microsoft Database sample Northwind](#)
- [Blog about LINQ to SQL for advanced users with many examples and details](#)
- [Deferred loading](#)
- [Microsoft webcasts \(French\)](#)

License

This article, along with any associated source code and files, is licensed under [The Code Project Open License \(CPOL\)](#)

Written By

Le Sourcier

Software Developer ECT Industries

 France

I am software engineer and I work for the aviation.

I'm currently working on many different project and in many different languages

- Visual C++ 6
- C#
- ASP.NET
- C and assembly

Have lot of fun

Comments and Discussions

You must [Sign In](#) to use this message board.

Search Comments



[First](#) [Prev](#) [Next](#)

[answer](#) 

elmacho 1-Dec-17 4:30

[insert statement for user input](#) 

Member 11517481 11-Mar-15 12:37

[My vote of 5](#) 

rrossenbg 25-Apr-13 0:51

[SqlMetal.exe](#) 

Laserson 20-Mar-10 2:22

[how to use database in windows work flow?](#) 

ravi.vellanky 28-Jan-09 18:44

[Re: how to use database in windows work flow?](#) 

Le Sourcier 28-Jan-09 20:33

Interesting but... 📌

captainplanet0123 18-Nov-08 12:57

Re: Interesting but... 📌

Guidii 28-Apr-11 5:50

Great work 📌

Member 4497378 24-Oct-08 10:45

Insert Statement 📌

imagic 22-Sep-08 0:04

Re: Insert Statement 📌

Le Sourcier 22-Sep-08 0:22

Excellent, fast-paced overview 📌











chaiguy1337 11-Dec-07 7:58

Nice Linq to SQL primer 📌

PCoffey 11-Dec-07 4:43

[Refresh](#)

1

 [General](#)  [News](#)  [Suggestion](#)  [Question](#)  [Bug](#)  [Answer](#)  [Joke](#)  [Praise](#)  [Rant](#)
 [Admin](#)

Use Ctrl+Left/Right to switch messages, Ctrl+Up/Down to switch threads, Ctrl+Shift+Left/Right to switch pages.

[Permalink](#)

[Advertise](#)

[Privacy](#)

[Cookies](#)

[Terms of Use](#)

Layout: [fixed](#) | [fluid](#)

Article Copyright 2007 by Le Sourcier
Everything else Copyright ©
[CodeProject](#), 1999-2023

Web04 2.8:2022-12-16:1