Intersoft Solutions
A better web experience.

Large Dataset Performance Demystified

Background

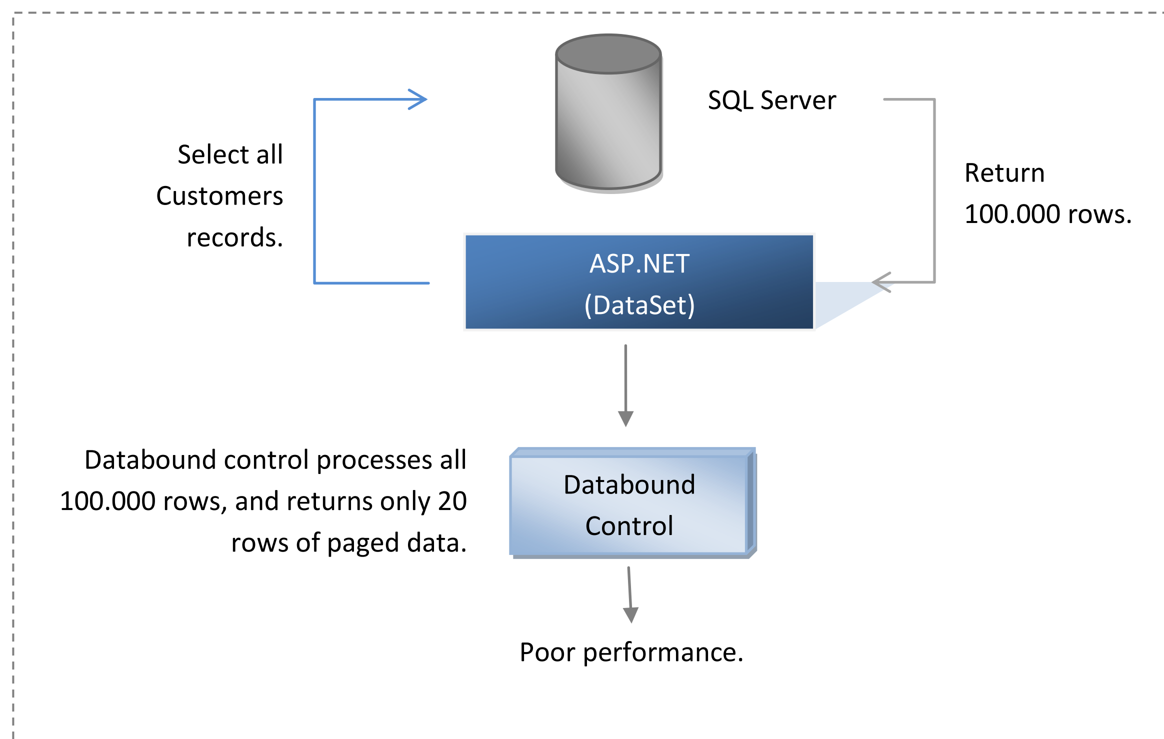
This white paper discusses the new technology introduced in Microsoft.NET Framework 3.5 to resolve many data-related and performance issues in ASP.NET web development. One of the most common issues in web development is the performance issue when data binding to a large datasource such as a table with several hundred thousands or millions of records.

This white paper will demystify the real technologies and behind-the-scene techniques to resolve the performance issue when binding to large datasource.

Performance Demystified

Several third party vendors have tried to hide this new free technology and brand it for their own marketing purpose, which causing developers to remain unskillful and have to rely on specific products. Developers deserve to know the truth, and to learn many of new technologies that have been made available at no costs, which is the main objective of this white paper.

The following is the illustration of the common performance problem.



 © 2008. All rights reserved.

The resolution to this serious development problem is not bound to a specific "proprietary" technology by third party vendors. Surprisingly, the resolution is done through Microsoft's new LINQ to SQL, a new feature introduced in .NET Framework 3.5 that can be leveraged by developers at no costs.

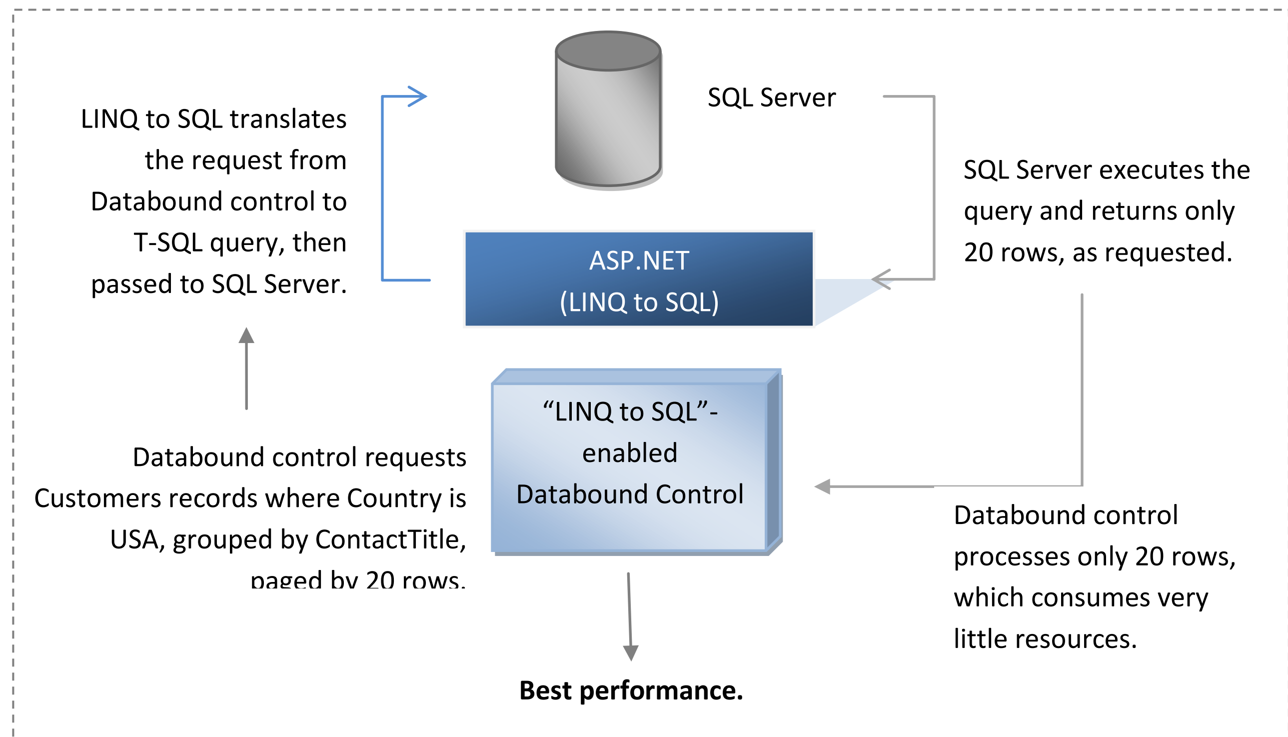
LINQ to SQL comes to rescue

It's no secret that the resolution to performance issues when retrieving large datasource is by performing the operation at database (backend) level. However, some of the operations require advanced knowledge of Transact SQL. Furthermore, developers are required to write many codes since the operations are difficult to be generalized (reused) due to complex scenarios. This has caused a great deal of frustrating performance bottlenecks as the database size is growing significantly.

LINQ to SQL is Microsoft's answer to this serious bottlenecks and problems. LINQ stands for Language Integrated Query, and is a subset of Microsoft .NET Framework 3.5. With LINQ, you can write SQL-like query in .NET managed languages such as C# or VB.NET. The LINQ to SQL means that your C# query will be automatically translated to Transact SQL behind the scene.

The real power of LINQ to SQL lays on its ability in generating T-SQL query that is compatible with SQL Server 2000, SQL Server 2005, SQL Server 2008 and SQL Express editions. LINQ to SQL is capable to generate T-SQL query that involves several operations in one execution, such as: sorting, grouping, aggregate computation, as well as paging.

Let's see how LINQ to SQL perfectly resolves performance bottlenecks by returning only the records that requested by the databound control.



 © 2008. All rights reserved.

Intersoft Solutions
A better web experience.

As described in the above illustration, the LINQ to SQL performs the work necessary to translate the view requested by the databound control into T-SQL compatible query. This is a great time saving feature, as developers no longer have to write a lot of codes.

It's undoubted that "LINQ to SQL" is powerful enough to overcome performance issues and resolve complex business scenarios. The real question now is when and where to apply this new technology, and how easy to consume it.

In more realistic questions, are your databound controls "LINQ to SQL"-enabled yet? Have your data presentation component supported "LINQ to SQL"?

Keep reading on the next sections, as you will be guided to learn more about LINQ to SQL, its syntax, and how it works.

Basic Data Retrieval Operation Demystified

Basic data retrieval operations are well covered in LINQ to SQL. Two of the most common operations are sorting and filtering.

Take a look at the simple LINQ query below:

```
var basicQuery = from customer in dc.Customers
                         where customer.Country == "USA"
                         orderby customer.Country
                         select customer;
```

The above C# query will be translated into SQL automatically when the query is called. The resulted SQL statement will look like the following:

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region],
[t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]

      FROM [dbo].[Customers] AS [t0]

      WHERE [t0].[Country] = @p0

      ORDER BY [t0].[Country]',N'@p0 nvarchar(3)',@p0=N'USA'
```

As you can see, the SQL statement produces direct result from database server to the caller. Direct result means databound control no longer needs to perform extra works such as local filtering or ordering, which is often performed at DataSet level.

With LINQ to SQL, the basic data operation such as sorting and filtering is performed at database server. This translates to significant performance enhancement to application and user interface layer.

 © 2008. All rights reserved.

Intersoft Solutions
A better web experience.

Grouping Demystified

In addition to basic data operations, LINQ to SQL also includes built-in support for Grouping operation. The following query shows how you can group customers by *ContactTitle* in C# language.

```
var groupingQuery = from customer in dc.Customers
                    group customer by customer.ContactTitle into grouping
                    select grouping;
```

LINQ to SQL will translate the query into T-SQL query, as shown below:

```
SELECT [t0].[ContactTitle] AS [Key]

FROM [dbo].[Customers] AS [t0]

GROUP BY [t0].[ContactTitle]
```

Next, upon each loop of the group record, LINQ to SQL will also automatically create a query to return the records of each group, such as shown in the following:

```
SELECT [t0].[CustomerID], [t0].[CompanyName], [t0].[ContactName],
[t0].[ContactTitle], [t0].[Address], [t0].[City], [t0].[Region],
[t0].[PostalCode], [t0].[Country], [t0].[Phone], [t0].[Fax]

FROM [dbo].[Customers] AS [t0]

WHERE ((@x1 IS NULL) AND ([t0].[ContactTitle] IS NULL)) OR ((@x1 IS NOT NULL)
AND ([t0].[ContactTitle] IS NOT NULL) AND (@x1 = [t0].[ContactTitle]))',N'@x1
nvarchar(18)',@x1=N'Accounting Manager'
```

Interestingly, LINQ to SQL supports multiple columns grouping. This allows developers to easily achieve multiple grouped records in application layer, without have to understand extensive T-SQL knowledge. The following C# query shows how.

```
var multipleGroupingQuery = from customer in dc.Customers
                            group customer by
                             new
                             {
                                 Country = customer.Country,
                                 ContactTitle = customer.ContactTitle
                             };
```

When grouping is used, LINQ automatically detects the type and create anonymous type behind-the-scene so that you can access to the results in strongly-typed programming model. See below example for iterating the groups and records:

```
      foreach (var group in multipleGroupingQuery)
        {
            Response.Write(group.Key.Country + " > " + group.Key.ContactTitle
            + "<br/>");
```

© 2008. All rights reserved.

```
            foreach (var customer in group)
            {
                Response.Write("      " + customer.CustomerID + "<br/>");
            }
        }
```

Aggregate Computation Demystified

LINQ to SQL provides built-in aggregate computation, such as:

- Count
- Sum
- Average
- Min
- Max

The following C# code shows you how to compute the Count of orders data.

```
int ordersCount = (from order in dc.Orders
                   where order.Customer.Country == "USA"
                   select order.CustomerID).Count();
```

You can also create a base query, and perform different aggregate functions on the base query. For example:

```
var baseQuery = from order in dc.Orders
                        where order.Customer.Country == "USA"
                        select order.Freight;

decimal? freightSum = baseQuery.Sum();
decimal? freigtAverage = baseQuery.Average();
```

The aggregate methods such as *Count*, *Sum* and *Average* are extension methods introduced in .NET Framework 3.5. The interesting part is that LINQ will automatically detect the result of the query, and thus has the capability to obtain the correct type of the aggregation method's output.

For instance in the above sample, the *baseQuery.Sum()* is assigned to a *decimal?* type. If you assign it to *decimal* or *int*, the compiler will throw an error. The LINQ and integrated compiler capability means that you can write better quality codes and reduce errors.

To learn more about aggregate functions in LINQ to SQL, please see **References** below.

 © 2008. All rights reserved.

Intersoft Solutions
A better web experience.

Paging Demystified

Paging is one of the most challenging operations in data-driven application development. There have been many paging approaches in ASP.NET web development, such as performing paging at databound control level, or at application layer, or even at the client side. However, none of the approaches are able to resolve the performance bottlenecks as the database table grows.

The best approach in order to get the best performance is back to the application developer. This means, developers are required to write codes to perform the paging at the database side, and then return it to the corresponding middle layer to be processed. However, this manual approach could cause a great deal of frustration as developer needs to spend time in writing repetitive codes.

LINQ to SQL is Microsoft's answer to data paging challenges for data-driven application development. With LINQ to SQL, developers no longer need to manually write the codes to perform paging, while providing complete solution that resolve data paging challenges at the same time.

Take a look at how easy it is to get a paged Orders data from Northwind database.

```
var pagingQuery = (from order in dc.Orders
                  select order).Take(20).Skip(60);
```

The above C# query instructs the database server to get all orders data and take only 20 records after skipping the first 60 records.

Interestingly, LINQ to SQL translates the query into a Transact-SQL statement that is compatible with all versions of SQL Server. The produced T-SQL query looks like below.

```
SELECT [t2].[OrderID], [t2].[CustomerID], [t2].[EmployeeID], [t2].[OrderDate],
[t2].[RequiredDate], [t2].[ShippedDate], [t2].[ShipVia], [t2].[Freight], [t2].[ShipName],
[t2].[ShipAddress], [t2].[ShipCity], [t2].[ShipRegion], [t2].[ShipPostalCode], [t2].[ShipCountry]

FROM (

    SELECT ROW_NUMBER() OVER (ORDER BY [t1].[OrderID], [t1].[CustomerID], [t1].[EmployeeID],
[t1].[OrderDate], [t1].[RequiredDate], [t1].[ShippedDate], [t1].[ShipVia], [t1].[Freight],
[t1].[ShipName], [t1].[ShipAddress], [t1].[ShipCity], [t1].[ShipRegion], [t1].[ShipPostalCode],
[t1].[ShipCountry]) AS [ROW_NUMBER], [t1].[OrderID], [t1].[CustomerID], [t1].[EmployeeID],
[t1].[OrderDate], [t1].[RequiredDate], [t1].[ShippedDate], [t1].[ShipVia], [t1].[Freight],
[t1].[ShipName], [t1].[ShipAddress], [t1].[ShipCity], [t1].[ShipRegion], [t1].[ShipPostalCode],
[t1].[ShipCountry]

    FROM (

        SELECT TOP (20) [t0].[OrderID], [t0].[CustomerID], [t0].[EmployeeID], [t0].[OrderDate],
[t0].[RequiredDate], [t0].[ShippedDate], [t0].[ShipVia], [t0].[Freight], [t0].[ShipName],
[t0].[ShipAddress], [t0].[ShipCity], [t0].[ShipRegion], [t0].[ShipPostalCode], [t0].[ShipCountry]

        FROM [dbo].[Orders] AS [t0]

        ) AS [t1]

    ) AS [t2]

WHERE [t2].[ROW_NUMBER] > @p0 ORDER BY [t2].[ROW_NUMBER]
```

A masterpiece by Intersoft Solutions Corp – Delivering a better Web experience™. © 2008. All rights reserved.

Intersoft Solutions
A better web experience.

Putting All Together

Now that we have touched the surface of LINQ to SQL, a new powerful query language feature introduced in .NET Framework 3.5. LINQ to SQL enables you write even more powerful queries such as joining, querying against relational tables, and much more.

The following sample shows how you can use LINQ to SQL to get the data shape that you desire, by combining several data operations (sorting, filtering and paging) at once.

```
var combinedQuery = (from order in dc.Orders
                             where
                             order.Customer.ContactTitle.StartsWith("Sales")
                             && order.Customer.Region != null
                             orderby order.OrderDate descending
                             select new
                             {
                                 ContactName = order.Customer.ContactName,
                                 OrderDate = order.OrderDate,
                                 ShipName = order.ShipName,
                                 ShipCountry = order.ShipCountry,
                                 OrdersCount = order.Order_Details.Count(),
                                 OrdersAverage = order.Order_Details.Average(
x => (x.Quantity * x.UnitPrice) - Convert.ToDecimal(x.Discount))
                             }).Take(10).Skip(5);
```

The above query is quite straightforward and self-explanatory even though it may look complex at first. The query also transforms the existing view into a new projection that contains selective properties. As you can see, it also supports in-line aggregate inside the select statement, allowing you to achieve the data shape that you desire in one call.

The following is the resulted T-SQL of the above query:

```
SELECT [t7].[ContactName], [t7].[OrderDate], [t7].[ShipName], [t7].[ShipCountry], [t7].[value] AS
[OrdersCount], [t7].[value2] AS [OrdersAverage]

FROM (

    SELECT ROW_NUMBER() OVER (ORDER BY [t6].[OrderDate] DESC) AS [ROW_NUMBER],
[t6].[ContactName], [t6].[OrderDate], [t6].[ShipName], [t6].[ShipCountry], [t6].[value],
[t6].[value2]

    FROM (

        SELECT TOP (10) [t5].[ContactName], [t5].[OrderDate], [t5].[ShipName],
[t5].[ShipCountry], [t5].[value], [t5].[value2]

        FROM (

            SELECT [t1].[ContactName], [t0].[OrderDate], [t0].[ShipName], [t0].[ShipCountry], (

                SELECT COUNT(*)

                FROM [dbo].[Order Details] AS [t2]

                WHERE [t2].[OrderID] = [t0].[OrderID]
```

 © 2008. All rights reserved.

```
                ) AS [value], (

                SELECT AVG([t4].[value])

                FROM (

                    SELECT ((CONVERT(Decimal(29,4),[t3].[Quantity])) * [t3].[UnitPrice]) -
(CONVERT(Decimal(29,4),[t3].[Discount])) AS [value], [t3].[OrderID]

                    FROM [dbo].[Order Details] AS [t3]

                    ) AS [t4]

                WHERE [t4].[OrderID] = [t0].[OrderID]

                ) AS [value2], [t1].[ContactTitle], [t1].[Region]

            FROM [dbo].[Orders] AS [t0]

            LEFT OUTER JOIN [dbo].[Customers] AS [t1] ON [t1].[CustomerID] = [t0].[CustomerID]

            ) AS [t5]

        WHERE ([t5].[ContactTitle] LIKE @p0) AND ([t5].[Region] IS NOT NULL)

        ORDER BY [t5].[OrderDate] DESC

        ) AS [t6]

    ) AS [t7]

WHERE [t7].[ROW_NUMBER] > @p1

ORDER BY [t7].[ROW_NUMBER]',N'@p0 nvarchar(6),@p1 int',@p0=N'Sales%',@p1=5
```

LinqDataSource control for ASP.NET web development.

LinqDataSource is a new datasource control that comes along with ASP.NET 3.5, which wraps most of the LINQ to SQL functionality into one simple, easy-to-use datasource control.

LinqDataSource supports some basic LINQ to SQL features such as:

- Where
- OrderBy
- OrderGroupBy
- GroupBy
- Transaction operations, such as Insert, Update and Delete.
- AutoPage
- AutoSort

The only lacking feature in LinqDataSource is the aggregate computation predicate, which is not currently supported in either Select property.

A masterpiece by Intersoft Solutions Corp – Delivering a better Web experience™. © 2008. All rights reserved.

Intersoft Solutions
A better web experience.

Automating Intersoft Databound Controls

Now we have learnt the benefits of LINQ to SQL, and how to consume it using LINQ query language. In this section, we'll look how Intersoft is taking advantage of this powerful technology in their databound controls.

The latest version of Intersoft's WebUI Suite (2008 R2 Build 207+) has been significantly improved to support LINQ to SQL. The result is obvious – striking fast data retrieval operation and manipulation.

Intersoft's flagship databound controls – WebGrid.NET Enterprise™ and WebCombo.NET™ – sets a new record in performance benchmark, outpacing competing products that were using proprietary, slower and unproven mechanism.

WebGrid.NET Enterprise™

WebGrid.NET Enterprise 6.0 (Build 207) has implemented full support for LINQ to SQL technology via *LinqDataSource* control. With a simple LinqDataSource control in your page, WebGrid will be able to perform the following operations automatically – without the needs to write a single line of code:

- Column sorting, supporting multiple columns.
- Column filtering, supporting all expressions and operators that currently available in WebGrid.
- Column grouping, supporting multiple groups.
- Data paging, supporting both VirtualLoad™ and ClassicPaging™.
- Data transactions, such as adding, editing and deleting. Also supports data transactions when the data is grouped, which is not supported by LINQ to SQL by default.

In order to provide more advanced functionality that doesn't exist in standard *LinqDataSource* control, we have developed a new datasource control that derives from LinqDataSource. We're proud to introduce *ISLinqDataSource*, a **free download** for every .NET developer who wants to build blazing fast web application.

When WebGrid is bound to *ISLinqDataSource*, the following features will be automatically processed by using LINQ to SQL for best performance:

- Column Total, with per page aggregation.
- Column Total with global data aggregation, supporting following functions natively: Count, Sum, Avg, Min and Max.

WebCombo.NET™

WebCombo.NET 4.0 (Build 207) has been enhanced to fully support *LinqDataSource* control as well. When bound to *LinqDataSource*, WebCombo will perform the query and paging on the database server through LINQ to SQL, while consistently support essential WebCombo features such as *AllowWildcardSearch*, *AdditionalSearchFields* and *Linked WebCombo*.

Together with the state-of-the-art AJAX architecture, this improvement enables WebCombo to retrieve data from million records table in a fraction of seconds. This translates to improved user experience.

A masterpiece by Intersoft Solutions Corp – Delivering a better Web experience™. © 2008. All rights reserved.

Next Steps

Read Intersoft WebUI and LINQ to SQL Walkthroughs.

Download WebUI Studio.NET 2008 R2.

Download latest builds (Build 207 – September 2008).

Download Intersoft WebUI and LINQ to SQL samples.

Download ISLinqDataSource control – the blazing fast Linq datasource – for free!

Conclusion

Congratulations! You have just upgraded your programming knowledge in cutting-edge technology around Microsoft.NET 3.5, LINQ to SQL, as well as understanding on how to resolve complex business scenarios by using these new features.

As you already understand, there is no need of such "non sense" proprietary third party technology to achieve "blazing fast" with "lightning speed" data retrieval on large data source. It is now revealed that the behind-the-scene technology used is Microsoft's new LINQ to SQL feature, which can be utilized by .NET developers at no costs.

Start using LINQ to SQL technology in your application today to avoid performance bottlenecks and scalability issues in the future.

References

- LINQ to SQL (http://msdn.microsoft.com/en-us/library/bb386976.aspx)
- Query Concepts in LINQ to SQL (http://msdn.microsoft.com/en-us/library/bb387006.aspx)
- What you can do with LINQ to SQL (http://msdn.microsoft.com/en-us/library/bb882643.aspx)
- Query Examples (http://msdn.microsoft.com/en-us/library/bb386913.aspx)
- Frequently Asked Questions (http://msdn.microsoft.com/en-us/library/bb386929.aspx)
- LinqDataSource Overview (http://msdn.microsoft.com/en-us/library/bb547113.aspx)

A masterpiece by Intersoft Solutions Corp – Delivering a better Web experience™. © 2008. All rights reserved.