

## Re: Table bloat in Linq-SQL

---

*Source:*

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-11/msg01633.html>

---

- *From:* "Andrus" <kobruleht2@xxxxxx>
  - *Date:* Tue, 13 Nov 2007 16:06:47 +0200
- 

Frans,

Ok, but take a step back for a minute: these assemblies, they're created on-site at the client, I presume? Otherwise you wouldn't be needing them.

Yes, they are created on client site.

So your code accessing these custom types has to be generic, i.o.w.: it's very very hard to make that reliable.

I'm planning to have the following code in my WinForms application to show customers with extended properties using Linq-SQL:

```
myDataGridView.DataSource = myDataContext.Customers.ToList();
```

Why this code is not reliable ?

Sure, but you can't access these user defined properties in your code, at least not in a typed fashion. So IF you want to work with these, you've to fall back on low-level property descriptor voodoo, which is IMHO not preferable and definitely not maintainable.

As I wrote below I'm planning to use dynamically compiled scripts in customer sites.

Those scripts are compiled in customer site and can use extended properties.

Also, your complete application isn't really testable, because the stuff added for the client is unknown.

## Re: Table bloat in Linq-SQL

Scripts which are created in customer sites are tested in customer sites with customer data.

Core application containing core customer properties is testable at my development site.

How can you write code off-site which accesses in a typed fashion properties which are added later on?

I can access sql server in client site over internet from my office. server contains table of scripts. I can add scripts to this table. Scripts are executed at runtime from my generic core application. In those scripts I can use extended properties in typed way like

```
string dutchAddress = from k in db.Customer select k.DutchAddress where k.id=1;
```

No, there are other ways. The thing is that the properties in the grids are just a facade. You have to implement `ITypedList` and from then on you can return whatever you want in property descriptors to make up a flat list, which for example isn't a flat list at all.

Yes, for binding `TypeDescriptor` interfaces in `System.ComponentModel` work well.

However, I'd prefer to use extended properties in Linq-SQL queries in scripts running in customer sites also.

How to use the statement

```
string dutchAddress = from k in db.Customer select k.DutchAddress where k.id=1;
```

with `ITypedList` ?

As Marc wrote in this newsgroup Linq-SQL does not support `TypeDescriptor` at all.

This is very serious Linq-SQL design flaw.

Subclassing is only way if I want to use Linq-SQL with custom properties.

What you want is to add for example 'DutchAddress' to a Customer class at the client's site, however you can't access DutchAddress in your own code, as that's compiled and written and tested without that property on the Customer class. Sure, at runtime you can have binding perhaps, but that's about it. Where's your validation logic and above all, where do you test your validation logic for 'DutchAddress' ?

## Re: Table bloat in Linq-SQL

I use standard Linq-SQL pattern described in LinQ FAQ.

I add a method

OnDutchAddressChanging(String value)

to customer extension class generated at runtime and generate code to call this method from DutchAddress property setter.

The typical pattern to implement flexible fields in entities is by using meta-tables, although it's a sucky approach as well, but it's better than messing with code you can't test on the client's site.

Meta tables work with a couple of tables, where you define 1:n relations between the extended entity and the list of properties to add as well. This can be implemented in your code without a problem and tested. You can even create a UI to create these fields.

I don't know the business logic required to create and validate DutchAddress at design time.

So this logic should be injected into application and compiled at run time anyway, even when using meta tables.

I don't know any Linq-SQL provider for meta tables.

Writing and maintaining my only Linq-SQL provider which supports meta-tables requires a lot more time than dynamic code generation.

As the properties are really 1:n related objects to your 'customer' entity, you then create an IList implementation on your list to bind to a grid. That IList implementation returns the Customer property descriptors but also creates property descriptors for the 1:n related extra property objects.

Do a search on google about implementing IList, or search my blog for the article I wrote about this.

Since IList does not allow to use Linq, I cannot use it.

Ah, so you're saying that at your location you can write code which can deal with property FooBar on 'Customer' while there's no FooBar property at the moment, which is added later on at the client's site?

From my office I can add FooBar column to server Customer table.

## Re: Table bloat in Linq-SQL

I can create and save FooBar validation script to script table text column in server.

Where is the validation logic for FooBar written, as it's not there yet?

FooBar validation logic will be written in text field in server database. Dynamic class compiling injects code from this text field to OnFooBarChanging() method which is called from FooBar setter.

I use

```
ALTER TABLE customer ADD COLUMN Foobar CHAR(30)
```

to add column before script is running so column foobar exists in customized database.

Also at the client's site? And where do you test Foobar and its validation? Also at the clients site? Or not at all?

Yes, scripts are running in customer sites.

Most of the testing is done by customer, this reduces development cost a lot. Errors in scripts are written to log file which I can read from server table and change scripts.

IRC python is a dyn. language and not statically typed. Similar to Ruby, which is ideal for the project you're working on.

I don't know Ruby.

Should I really move to Ruby? Where to find good IDE and report designer like RDLC format report designer for Ruby, those are critical for my project?

C# is a statically typed language, if you want to add properties to a type at runtime, you can't, it has to be via subclassing.

It should be possible to use TypeDescriptor also. Unfortunately Linq-SQL does not support TypeDescriptor define properties.

The thing is that if you do that, you have to take into account this subclass and its

## Re: Table bloat in Linq-SQL

properties. It's not a real problem per se, UNLESS you need to work with these extra properties in your ORIGINAL code, because that original code isn't aware of these extra properties nor the subclass. (via 'dynamic proxy', you can extend classes at runtime in C#, however code which isn't aware of the added stuff can't work with that extra stuff, as it doesn't know it's there at compile time. The code which IS aware of the extra stuff can of course, but imho in your project that's not code which does exist.

I work with extra properties using scripts, data binding and report designer. There are no direct references to those extra properties in my code.

It's a trick, but it has consequences: the thing isn't adding the property, dynamic proxy can do that, the thing is the code working with the added property. If I write a validator for Customer, I can't add to the validator some code working with properties which are added at runtime by a subclass, because the compiler doesn't know about these properties.

Ideablade dynamic property tutorial teaches "property discovery" from UI code.

With dynamic proxy (the pattern you follow as well, though you generate the proxy with a script), there's no end to what you can add to a class. That's not the point. The point is: can you write provable code which is correct when you say to your client: "It's done"?

I can write application using standard properties and sell it to client saying "it is done and tested".

If client wants to add new properties and custom business logic I can add this and test additionally without changing core assemblies. MyGeneration is sample of this. However MyGeneration *\*requires\** to use scripts for code generation by extended properties pattern discussed above *\*allows\** to use scripts.

Andrus.