

Re: Finally which ORM tool?

Source:

<http://www.tech-archive.net/Archive/DotNet/microsoft.public.dotnet.languages.csharp/2007-10/msg02003.html>

- *From:* "Frans Bouma [C# MVP]" <perseus.usenetNOSPAM@xxxxxxxxxx>
 - *Date:* Mon, 15 Oct 2007 02:02:04 -0700
-

Jon Skeet [C# MVP] wrote:

Frans Bouma [C# MVP] <perseus.usenetNOSPAM@xxxxxxxxxx> wrote:

You won't know which fields have changed unless you requery the database whatever you do – sessions don't make the situation any worse here.

If you track changes inside the entity, you don't need to. If you don't have a session-oriented design, you have to track changes elsewhere, so you don't have to put the burden onto the developer to do things for the framework, the framework can decide what to do by itself.

I think we were talking about a different type of change tracking.

I think there's just one: entity instance gets loaded into entity class instance, entity instance in-memory gets changed. Which parts? that's tracked, by the change tracking mechanism.

However, I seem to remember that Hibernate performs bytecode trickery to keep changes within the object itself as well.

Don't compare hibernate with nhibernate, as hibernate 3 is different from nhibernate (which is based on hibernate2). It's my understanding that nhibernate uses the 'keep a copy around of the original values in the session' method.

Name one advantage of session-oriented design over non-session oriented design for the USER of the framework. There are none. Sure,

Re: Finally which ORM tool?

for the developer of the framework, it's easier, much easier, as every meta-data element and other data element to work with is centrally available and what to do is actually told to you.

You're still sounding very black and white, and I simply don't believe that there isn't a single advantage (to the user) of keeping track of things in sessions. I have to say that when I was using Hibernate it seemed a very natural way of working, partly because it was very similar to using a transaction. Keeping a session open for the length of a request just seemed to fit as an easy way of doing something. I never experienced the "hell" of attach/detach, and having opened the session and used that for everything else, I didn't need to learn any "extras" to get uniqueness.

Also in distributed scenario's where entities got fetched in session instance S1, distributed to some client, altered there and then they came back and have to be saved with session instance S2?

I'm not saying that having a class which manages the activity with the DB is bad, on the contrary. What I'm saying is that the object used to load the entities should be in effect stateless. So shouldn't be tied to an entity object.

Certainly when I've used Hibernate I haven't found the session stuff to be a pain, and it's been nice to be able to treat it as a context for that request.

2-way databinding in webapps, passing entities across service boundaries... 2 examples where you have to tell the framework what to do while it's the job of the framework to find out what to do.

Ah, databinding – I've never been a fan of that to start with, to be honest.

Me neither, but the vast majority of people out there uses it. :)

Too much black magic which only works if you happen to want to work in exactly the way that was anticipated. At least, that's been my experience of it in every scenario I've seen. It does work pretty well for displaying stuff – but making it 2-way just has so many problems. With any luck WPF improves matters somewhat as far as thick clients are concerned...

Re: Finally which ORM tool?

It's not that bad. Avoiding it has the implication that you have to write the glue between control and object yourself, which can be painful and buggy as well, simply because it's boring code.

Sure if you don't mind to do the extra work, it's not a problem. Also, with hibernate based frameworks, you don't get a lot of entity management anyway. I mean:
myOrder.Customer = myCustomer;
doesn't make this true:
myCustomer.Orders.Contains(myOrder);

Not unless you implement the recommended patterns (which would keep the association consistent). I admit it's a disadvantage to have to do all this manually.

exactly. If you have to write manually all the code to keep things going, sure, then there's no problem. But the point of using a framework is that you DON'T have to write the code manually: it's been done for you.

Though back to the small space of a webpage which posts back: the state of the page has to be pulled from somewhere. That's of course ok, there are facilities for that. The problem is though, if you track changes in an object outside the entities, you either have to keep that in memory as well (bad) or you have to rely on the developer to tell you what the state of the entities is. (bad too, as it implies babysitting by the developer).

As I say, I thought Hibernate kept track of what had changed within the entity itself too. I don't know about NHibernate though, or how serialization affects this.

If I'm not mistaken, they use the same mechanism as Linq to Sql does: keep the original values in the session, and compare original with current to see what the changes are.

That's also why they have teh add/detach hell already looming on the horizon (Linq to sql will throw exceptions in these cases when attaching graphs) and together with deferred execution of linq queries, it will give a lot of

Re: Finally which ORM tool?

problems for
users who expect A but run into snag B.

You call it "add/detach hell" but I honestly can't say I ever
had
any problems when working with Hibernate in this way.

So you never had to tell the session that the entity object you
attached was new or not new?

I can't say I remember ever actually needing to attach/detach. I know
that it's available, but I can't remember needing to use it. If I
did need to use it, it clearly didn't cause me enough pain to make it
memorable.

If Hibernate 3 has in-entity change management through bytecode
manipulation (or post-compilation, as it's used by some .net o/r
mappers), then you don't have attach/detach problems as you don't need
to: the session doesn't keep track of the original values, they're
inside the entity object.

Deferred execution is a fundamental (and very useful) part of
LINQ which people will just have to understand. It's not
mind-bogglingly new, either – using the Criteria API in
Hibernate
does exactly the same kind of thing, building up a query
which
can be executed whenever you want it to be.

That's not the same thing! A criteria object doesn't contain a
session! the object created by the compiler DOES contain a
QueryProvider which has to have access to the persistence core to be
able to execute the query by itself.

A Criteria object does contain the session (or at least has a
reference to it).

See
http://www.hibernate.org/hib_docs/v3/api/org/hibernate/impl/CriteriaImpl.html (watch for wrapping)

Hmm, indeed, I didn't know that. Strange decision IMHO.

Re: Finally which ORM tool?

Re: Finally which ORM tool?

That's the fundamental difference and that difference is going to cause problems, simply because the linq query object contains a session.

Also, if you pass a variable to the query, the value the variable has at EXECUTION time is used, not at CREATION time:

That certainly needs to be clearly understood – but I'd say it's useful, too, in cases where you need to do the same query multiple times but with different parameters.

Isn't that going into the red zone of 'magic programming' ? I mean, you have a local variable, even if it's a value typed variable, and you have a linq query and by changing the variable's value, you manipulate the linq query IF you're executing it at that moment.

Captured variables always need to be handled with care, and I can see that it could well trip up novices, but I don't think it's unreasonable.

Sure, but the code LOOKS the same as:

```
int foo = GetFoo();  
bool b = (bar == foo);
```

b isn't suddenly affected if foo is changed after this. Though if I do:

```
var q = from x in metadata.SomeEntity  
where x.Foo == foo  
select x;
```

q is affected if I change foo AFTER this query and BEFORE execution.

Why is this comparison expression suddenly different from the expression of b? The first is executed at that moment, the latter is executed somewhere later, but you have to follow q to found out when.

That SHOULDNT be important, simply because q LOOKS like a declaration, constructed at the spot where q is declared.

```
string customerID = "CHOPS";  
  
var q = from o in nw.Order  
where o.CustomerId = customerID  
select o;  
  
// .. some other code  
customerID = "BLONP";  
  
foreach(Order o in q)
```

Re: Finally which ORM tool?

```
{  
// which orders are read? BLONP's!  
}
```

So the query isn't a query definition alone, it's also the resultset.

No, the query itself is a query definition alone. It's only the enumerator you get when you call GetEnumerator() which relates to the result set.

It's an IEnumerable<T>, and therefore a resultset. If it would be a definition alone, the queries of CHOPS would have been fetched, simply because the declaration construction was with 'CHOPS'.

It's very natural. Also, because in that last example it's not deferred executed, the query is created when the var q statement is executed in code, so the changed variable problem isn't there.

But equally the ability to reuse the query very simply by changing the variables isn't there either. If you want to avoid the meaning of the query being changed, either copy the variable values or just don't change the variables.

Sure, though I don't like the similarity of the code statements which tend to behave completely different at runtime. As this is a runtime issue, it can lead to test-burdens.

They for example could have opted for a system where you could get access to the parameters in the query to alter them for each run.

If the query would just be that, a query specification, you wouldn't have had this problem.

Now, call me stupid, but why on earth would anyone with a little bit of knowledge of o/r mapping design the system DELIBERATELY so that deferred execution was there? What problem does it solve? IMHO none.

I've always found the deferred execution very natural, both in LINQ and Hibernate – I create a query, do other things if necessary, and then use the query when I'm ready. Why should creating the query execute it immediately?

Re: Finally which ORM tool?

Re: Finally which ORM tool?

No, that's not what I meant. What I meant was: you declare a query somewhere, e.g. in a method you call to formulate the query, then execute it somewhere else, however the query declaration is already fixed, so you can alter it, by changing a parameter on a predicate, but not by changing a local variable's value.

What advantage does this have? well, plenty. For example you can write generic code for constructing filters, which is executable on multiple databases, e.g. your system is targeting multiple databases(oracle, sqlserver, doesn't matter) at runtime, and you can achieve that.

With the 'executor is embedded inside the query' approach, you can't do that, or at least not easy: you have to swap out the provider inside the query object, and also: you need one when creating the query, which is IMHO absurd, as you're constructing the query with meta-data, not with a db-specific provider, because the mapping info is of no relevance when creating the query.

so I can do:

```
EntityCollection<CustomerEntity> customers = new
EntityCollection<CustomerEntity>();
//call a method to specify the filter for the query.
RelationPredicateBucket filter = CreateFilter();

// then execute it on the adapter for the db currently used:
using(IDataAccessAdapter adapter = AdapterFactory.GetAdapter())
{
adapter.FetchEntityCollection(customers, filter);
}
```

generic code, not tied to a db specific context or meta-data. The connection between db specific information (the mapping data) and the generic entity info based query specification is made inside the adapter, out of sight, but still allows me to flexibly write code which targets whatever db supported at runtime.

That's IMHO why the deferred execution system with ties to the db specific elements isn't great, added to the fuzzyness of the parameters

The equivalent of Execute() is either starting to enumerate the results, or calling ToList(), or calling one of the aggregates such as Count(). If you want to execute the query as soon as you've defined it, that's easy enough to do – whereas if you want deferred execution in a situation where defining the query does execute it immediately, that's harder.

I'm not saying the query should be executed where it's defined. What

Re: Finally which ORM tool?

Re: Finally which ORM tool?

I'm saying is that the query should be a query, not a resultset as well. To obtain the resultset, one has to ask an object to get the result, defined by the query. This automatically has two advantages:

- 1) it avoids parameter changes because a value changed: it's very clear: the query is the query defined when you actually defined it
- 2) it allows to formulate the query with generic code and avoids all ties with db-specific code whatsoever, as these are defined in the object you ask to execute the query.

Now, if you want to argue that it would be nice to be able to defer execution further, allowing it to be unrelated to a session, I can certainly see the benefit of that – commonly used queries could be created at start-up and then just executed against arbitrary sessions. I'd have no problem with that whatsoever – a definite benefit. That's still deferred execution though.

I think the difference lies within the concept of what a declaration means to the developer. When a developer writes:

```
string a = foo + bar;
```

then a is foo+bar right after that line.

if the developer writes:

```
var q = from c in nw.Customers
where c.CompanyName == a
select c;
```

then q isn't the set of customers. It's a definition of a query, however with ties to the code it is created in, as it relies on the value of a, because the query CONSTRUCTION actually happens when q is executed. It should have obtained the values to construct the query when it's declared, so changing a isn't affecting q afterwards.

Declaring a query using HQL or whatever query system the o/r mapper uses, is declaring the query and nothing else. Passing it on to some other method to execute it there with an adapter, session, context or whatever is used is simply executing what's declared elsewhere WHEN it was declared, with the state at that moment of declaration, not with the state at the moment of execution.

I think my main problem is with that last difference: query execution should be done with the state at declaration time, while in linq it's query execution with the state at execution time. This is wrong IMHO, as it's vague: the developer can easily make a mistake in this, and even though 's/he then simply shouldn't do that'—applies here, if we all would avoid what we shouldn't do, we wouldn't have bugs and we all have bugs in our code no matter what, and the more clarity build into the language, the less bugs will appear IMHO.

Re: Finally which ORM tool?

Re: Finally which ORM tool?

FB

--

Lead developer of LLBLGen Pro, the productive O/R mapper for .NET

LLBLGen Pro website: <http://www.llblgen.com>

My .NET blog: <http://weblogs.asp.net/fbouma>

Microsoft MVP (C#)

.