Using my own domain model objects with .edmx in Entity Framework

Asked 7 years, 1 month ago Active 7 years, 1 month ago Viewed 4k times

7

5

I have a domain model architecture in which my domain/business objects were created based on the problem domain and independent of any knowledge of the physical data model or persistence structures. So far I'm on track because it's perfectly acceptable *and* often the case that there is an impedance mismatch between the domain model and the data model. A DBA created the database for getting the data *they* required, but it does not encapsulate the applications entire domain model or design.

The result - I have my *own* set of domain model objects. However all of the fields that need to be persisted *do* exist somewhere or another within my domain model, but not necessarily in the shape that my auto generated .edmx POCO entities have them. So I have all the data, it's just not in the *perfect* shape exactly like the tables in which auto generated POCO entities are created from.

I have seen a few posts on this topic like converting POCO entity to business entity and Entity Framework 4 with Existing Domain Model that make statements like the following:

> "Create the entities in your entity data model with the same names as your domain classes. The entity properties should also have the same names and types as in the domain classes"

What!? No way, why should I have to make my domain model be reshaped to POCOs that are modeled *exactly* after the data model / table structure in the database? For example - in my case of having 5 given properties, 2 might be in class 'A' and 3 in class 'B', whereas a auto generated POCO class has all 5 in its own class 'A'.

This is the entire point, I want separation of my object model and data model but yet still use an ORM like EF 5.0 to map in between them. I do not want to have to create and shape classes and properties named as such in the data model.

Right now my .edmx in EF 5.0 is generating the POCO classes for me, but my question is how to dissolve these and rewire everything to *my* domain objects that contain all this data but just in a different shape?

By the way any solution proposed using a Code First approach is not an option so *please* do not offer this. I need some guidance or a tutorial (best) using EF5 (if possible because EF4 examples are always inheriting POCOs from ObjectContext) with wiring up my own business objects to the .edmx.

Any help or guidance is appreciated, thanks!

edited May 23 '17 at 11:44
Community ♦
1 1

asked Feb 14 '13 at 21:19
atconway
17.1k 21 129 203

3 Answers

1

This sounds like exactly the use case of Entity Framework. I am making a few assumptions here. First that when you make this statement:

"I have a domain model architecture in which my domain/business objects were created based on the problem domain and independent of any knowledge of the physical data model or persistence structures."

That you mean this domain was created in the EF designer? But then you say:

"However all of the fields that need to be persisted do exist somewhere or another within my domain model, but not necessarily in the shape that my auto generated .edmx POCO entities have them."

This sounds to me like my first assumption is incorrect.

Next, you dismiss code first? If your domain model/business objects are code based and you want to persist them to a relational database, that is exactly the use case for code first. You have the code, now you need to create your DbContext and map it to your physical model.

However you dismiss that... so some thoughts:

If you have a domain model of code based business objects and you have an EDMX that is used for other things I think you would want to create a repository layer that uses something like auto mapper or manual projections to query your Entities and return your business objects.

If you have a domain model of code based business objects and you have an EDMX that is not used for other things other than persisting your business objects I would say that you need to express your domain in an EDMX and then map it onto your existing database. This is really the use case for an ORM. Having two domain models and mapping from one model to the other where one model matches your domain and one matches your database is adding an extra un-needed layer of plumbing.

The latter approach above is what is called "Model First" in EF parlance. There are several articles written about it although the bulk of them just generate the db from the model. You would not do that step, rather you would map your entities onto your existing

The basic steps for this are to "update from the database" not selecting any of the db objects (or entities would be created). Or, you can take your exiting .edmx in the designer (which is sounds like you have) and modify the entities to match your business domain. Or just delete all the entities in your EDMX model, create your entities as you want them, and then map them all.

Here is a jing I made where I use the EF Designer to bring in the model store (the only way to do this is to allow it to generate entities) and then delete the entities allowing the Store information to stay by clicking NO when it asks if you want to delete the table info.

http://screencast.com/t/8eiPg2kcp

I didn't add the POCO generator to this, but if I did it would generate the Entities in the designer as POCO classes.

edited Feb 15 '13 at 20:27

answered Feb 15 '13 at 19:11

PilotBob

2,980 7 29 46

"That you mean this domain was created in the EF designer?", no the domain classes were built from scratch not using the designer. *"Next, you dismiss code first"*, it's an application constraint and not a dismissal. Your last paragraph is basically the situation but it's really re-explaining my original situation and question and I'm looking for specific examples on how to do it. From my research, it appears modifying the conceptual layer mapping manually or through the designer might do the trick, but I'm looking for an example. – atconway Feb 15 '13 at 19:21

1 Basically you would use the designer to create your entities. Then connect to a database and manually map the entities to the database. I think the lower friction method is to reverse engineer the database into the designer and then modify the entities to match your business domain... which would simplify the mapping work. – PilotBob Feb 15 '13 at 19:24

I undated my answer with info on using the "model first" approach which will match what you want to do. – PilotBob Feb 15 '13 at 19:33

"Or just delete all the entities in your EDMX model, create your entities as you want them, and then map them all." Since Model 1st wants to create the database and thus generate the underlying mapping, how do I "manually map" my entites I create on the .edmx? Do I still have to modify the conceptual layer? – atconway Feb 15 '13 at 19:37

That was really cool and I appreciate the effort! It gave me enough ideas and pretty much what I was looking for. I can clear my model, add in the entities *I need* and then map them to the database. I am pretty sure this approach will work. I will give it a try and spawn off separate new questions if needed. – atconway Feb 15 '13 at 21:13

0

The statement quoted above is not suggesting that you rewrite your domain objects to match your pocos, it is suggesting that you update the edmx to match your domain model.

In your example you could create an entity in your edmx that maps all 5 properties from both tables and EF will manage the mapping

Of course this means that you then have duplicate domain objects and pocos, meaning you would either have to manually convert your domain objects to pocos when interacting with EF,

or you could define your domain data objects as interfaces and provide partial implementations of the pocos that essentially identify each poco as being a concrete implementation of a domain object.

There are probably several other ways to skin this particular cat, but I don't think that you can provide predefined c# objects for use in an edmx.

answered Feb 14 '13 at 21:47

Bobbles
549 5 4

...I don't think that you can provide predefined c# objects for use in an edmx. I thought this was the entire point of POCO support? Use my *own* objects with EF. Regardless, do you have any links or samples to back your high level descriptions? – atconway Feb 14 '13 at 21:58

ok, allow me to refine my comment. It would seem that most examples such as msdn.microsoft.com/en-gb/data/jj200620 – Bobbles Feb 15 '13 at 11:54

POCO support is code first. I think you are assuming your domain geometry must match your database scheme if using code first, that is not correct. "Code First" is a misnomer. They should have called it "code based". – PilotBob Feb 15 '13 at 19:22

Using managed code to reflect my mapping between the database and my classes (code 1st) is not allowed for this application. While the .edmx is not always preferred it is external and could have *slight* modifications be made without recompiling the code. This question is not about code 1st vs edmx. edmx is what is being used so I need help with that *only*. – atconway Feb 15 '13 at 19:28

0

One approach might be to select into a ViewModel (suited to your particular business logic) and automatically map some data from the context into it like so https://stackoverflow.com/a/8588843/201648. This uses `AutoMapper` to map entity properties from an EF context into a ViewModel. This won't do everything for you, but it might make life a bit easier. If you're unhappy with the way this occurs automatically, you can configure AutoMapper to do things a bit differently (see Projection) - https://github.com/AutoMapper/AutoMapper/wiki/Projection

You might know this already, but its also possible to automatically generate POCOs from your EDMX using t4 - http://visualstudiogallery.msdn.microsoft.com/72a60b14-1581-4b9b-89f2-846072eff19d. If you define the templates to generate partial classes, you can then have another partial class with your custom properties for that POCO. That way you can have most properties automatically populated, but have other custom properties which you populate with custom rules from your context/repository. This takes a lot of the monotony out of generating these, and you can then focus on reshaping the data using the above technique.

optimisation purposes/where a procedure already existed that did exactly what I wanted. See http://msdn.microsoft.com/en-us/data/gg699321.aspx

edited May 23 '17 at 12:15

Community ♦

1 1

answered Feb 15 '13 at 13:55

Aaron Newton

1,859 21 31

I'm using EF5 and the automatic POCO generation is baked into the functionality. It's these POCO classes that reflect the database directly and *not* my existing domain classes that are the issue. I want to stay away from using mapping between objects because this is precisely what my ORM is to do: map *my* business entities to the database. I believe it will involve modifying conceptual layer but not sure and need example for my scenario. – atconway Feb 15 '13 at 19:25