Why do we need entity objects? [closed]

Asked 16 years, 4 months ago Modified 8 years, 6 months ago

Viewed 16k times

141

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 8 years ago.

Improve this question

I really need to see some honest, thoughtful debate on the merits of the currently accepted ***enterprise application*** design paradigm.

I am not convinced that entity objects should exist.

By entity objects I mean the typical things we tend to build for our applications, like "Person", "Account", "Order", etc.

My current design philosophy is this:

- All database access must be accomplished via stored procedures.
- Whenever you need data, call a stored procedure and iterate over a SqlDataReader or the rows in a DataTable

(Note: I have also built enterprise applications with Java EE, java folks please substitute the equvalent for my .NET examples)

I am not anti-OO. I write lots of classes for different purposes, just not entities. I will admit that a large portion of the classes I write are static helper classes.

I am not building toys. I'm talking about large, high volume transactional applications deployed across multiple machines. Web applications, windows services, web services, b2b interaction, you name it.

I have used OR Mappers. I have written a few. I have used the Java EE stack, CSLA, and a few other equivalents. I have not only used them but actively developed and maintained these applications in production environments.

I have come to the battle-tested conclusion that entity objects are getting in our way, and our lives would be *so* much easier without them.

Consider this simple example: you get a support call about a certain page in your application that is not working correctly, maybe one of the fields is not being

persisted like it should be. With my model, the developer assigned to find the problem opens *exactly 3 files*. An ASPX, an ASPX.CS and a SQL file with the stored procedure. The problem, which might be a missing parameter to the stored procedure call, takes minutes to solve. But with any entity model, you will invariably fire up the debugger, start stepping through code, and you may end up with 15-20 files open in Visual Studio. By the time you step down to the bottom of the stack, you forgot where you started. We can only keep so many things in our heads at one time. Software is incredibly complex without adding any unnecessary layers.

Development complexity and troubleshooting are just one side of my gripe.

Now let's talk about scalability.

Do developers realize that each and every time they write or modify any code that interacts with the database, they need to do a throrough analysis of the exact impact on the database? And not just the development copy, I mean a mimic of production, so you can see that the additional column you now require for your object just invalidated the current query plan and a report that was running in 1 second will now take 2 minutes, just because you added a single column to the select list? And it turns out that the index you now require is so big that the DBA is going to have to modify the physical layout of your files?

If you let people get too far away from the physical data store with an abstraction, they will create havoc with an

application that needs to scale.

I am not a zealot. I can be convinced if I am wrong, and maybe I am, since there is such a strong push towards Linq to Sql, ADO.NET EF, Hibernate, Java EE, etc. Please think through your responses, if I am missing something I really want to know what it is, and why I should change my thinking.

[Edit]

It looks like this question is suddenly active again, so now that we have the new comment feature I have commented directly on several answers. Thanks for the replies, I think this is a healthy discussion.

I probably should have been more clear that I am talking about enterprise applications. I really can't comment on, say, a game that's running on someone's desktop, or a mobile app.

One thing I have to put up here at the top in response to several similar answers: orthogonality and separation of concerns often get cited as reasons to go entity/ORM. Stored procedures, to me, are the best example of separation of concerns that I can think of. If you disallow all other access to the database, other than via stored procedures, you could in theory redesign your entire data model and not break any code, so long as you maintained the inputs and outputs of the stored procedures. They are a perfect example of programming

by contract (just so long as you avoid "select *" and document the result sets).

Ask someone who's been in the industry for a long time and has worked with long-lived applications: how many application and UI layers have come and gone while a database has lived on? How hard is it to tune and refactor a database when there are 4 or 5 different persistence layers generating SQL to get at the data? You can't change anything! ORMs or any code that generates SQL ***lock your database in stone***.

sql database orm entities

Share

Improve this question

Follow

edited Jun 18, 2016 at 16:18

community wiki
4 revs, 3 users 95%
Eric Z Beard

From reading your question, we are very much alike, and I have wondered the exact same thing for years now (since hearing about 3rd party entity frameworks, and now Microsoft's) – pearcewg Mar 3, 2009 at 17:23

1 Are you saying the business logic is the helper objects, or in the stored procs? I'm asking as many people seem to think you are saying the later...but what I think you are saying is that you still have business logic in the coded objects, you are just getting data straight from the database and using that

data, rather than an ORM or mapping to specialized objects to hold the data. I tend to feel the same way--but am also currently evaluating EF4 to see if it might be worth it. – alchemical Jun 14, 2010 at 18:32

"innovator consumers are often the ones who get screwed." - someone with experience – Uğur Gümüşhan Sep 11, 2012 at 13:29

I inherited a system with over 2500 SPROCs where the application is viewed as simply a means to activating SPROCs and making sense of their output. Every data read and write has it's own SPROC. There are no central points of control. It is hideous and about as malleable as granite. I consider optimising the databases. The 2500 SPROCS put me in my place. Compared to a system with well-organised domain layer and re-usable DAL code it looks ill-conceived and is a support nightmare. Simple tasks take hours and are soul destroying. SPROCs should be used for high load or special methods IMO – trucker_jim May 19, 2014 at 15:29

About your "debugging" example: with unit tests, you'd know much more quickly where things go wrong. – MarioDS Dec 15, 2016 at 8:36

41 Answers

Sorted by: Highest score (default)

1 2 Next

59

I think it comes down to how complicated the "logic" of the application is, and where you have implemented it. If all your logic is in stored procedures, and all your application does is call those procedures and display the results, then developing entity objects is indeed a waste of time. But for an application where the objects have rich

interactions with one another, and the database is just a persistence mechanism, there can be value to having those objects.

So, I'd say there is no one-size-fits-all answer. Developers do need to be aware that, sometimes, trying to be too OO can cause more problems than it solves.

Share Improve this answer Follow

edited Dec 10, 2008 at 9:07

community wiki
Kristopher Johnson

Kristopher it looks like you resurrected this question by linking to it from another question. I'm wondering what you mean by "rich interactions", and how it would be impractical to implement them without objects? – Eric Z Beard Sep 12, 2008 at 2:57

4 Anything that can be done with objects can also be done without objects. I find that OO design is usually much easier than non-OO methods for doing anything "complicated", but I understand that it doesn't work for everyone.
– Kristopher Johnson Sep 12, 2008 at 10:32

I agree - the answer of "when to use objects" depends on whether or not the properties of the objects might require actions or change in business logic. For example User or Person instances might have Password and LoginName -> your code actions change according to what are the values in those. On the contrary if you would have a Product , you would have do display (nothing more , no other actions) than get a DataSet from db and just build the GUI.
– Yordan Georgiev Apr 18, 2009 at 8:20

5 There is a balance. Avoid the religion and pick what works. – Jeff Davis Jun 17, 2010 at 13:24

27

Theory says that highly cohesive, loosely coupled implementations are the way forward.

So I suppose you are questioning that approach, namely separating concerns.

Should my aspx.cs file be interacting with the database, calling a sproc, and understanding IDataReader?

In a team environment, especially where you have less technical people dealing with the aspx portion of the application, I don't need these people being able to "touch" this stuff.

Separating my domain from my database protects me from structural changes in the database, surely a good thing? Sure database efficacy is absolutely important, so let someone who is most excellent at that stuff deal with that stuff, in one place, with as little impact on the rest of the system as possible.

Unless I am misunderstanding your approach, one structural change in the database could have a large impact area with the surface of your application. I see that this separation of concerns enables me and my team to minimise this. Also any new member of the team should understand this approach better.

Also, your approach seems to advocate the business logic of your application to reside in your database? This feels wrong to me, SQL is really good at querying data, and not, imho, expressing business logic.

Interesting thought though, although it feels one step away from SQL in the aspx, which from my bad old unstructured asp days, fills me with dread.

Share Improve this answer Follow

edited Aug 20, 2008 at 21:09

community wiki
2 revs
nachojammers

I agree that having lots of dynamic SQL sprinkled throughout code-behind is evil. You have to keep the Db calls clear and distinct. Wrapping sproc calls in static helper methods achieves a sort-of separation without going all the way down the ORM route. – Eric Z Beard Sep 12, 2008 at 3:01

1 Although I've never worked in an asp environment, I'm sure that some of the less technical people would blow your socks off with some client side javascript code, which results in a beautiful user experience regardless of any crappy interface to the technical back-end. – crowne Feb 9, 2010 at 9:24

I agree with you here and I have been known to do some client side javascript too, which resulted in a not too shabby user experience, even if I do say so myself. I'd like to think by back-end interfaces aren't crappy though and that any client side programmers don't have to worry about any of that, because I have attempted to separate my concerns.
– nachojammers Feb 9, 2010 at 17:12

2 "This feels wrong to me, SQL is really good at querying data, and not, imho, expressing business logic." -- Unless you use, say, PL/SQL, which adds a rich programming language on top of (and tightly integrated with) the SQL, and your code is stored inside the database. Boosts performance by avoiding network roundtrips. And encapsulates your business logic regardless of which client connects to the database.
– ObiWanKenobi Dec 15, 2010 at 14:27

26

One reason - separating your domain model from your database model.

What I do is use Test Driven Development so I write my UI and Model layers first and the Data layer is mocked, so the UI and model is build around domain specific objects, then later I map these objects to what ever technology I'm using the the Data Layer. Its a bad idea to let the database structure determine the design of your application. Where possible write the app first and let that influence the structure of your database, not the other way around.

Share Improve this answer Follow

edited Apr 1, 2009 at 15:06

community wiki
2 revs
Dan

9 I have to say that I really disagree, at least for enterprise applications. The data is the application. – Eric Z Beard

Sep 12, 2008 at 2:54

3 why would you want to have two separate models of the same data? – Seun Osewa Dec 7, 2008 at 22:45

4 Because for some purposes, one interpretation of the model may be more suited. Some logic works a lot better on objects than on rows. – Wouter Lievens Dec 23, 2008 at 15:01

1 I think it's a good idea implemented poorly. – Jeff Davis Jul 14, 2009 at 16:36

21

For me it boils down to I don't want my application to be concerned with how the data is stored. I'll probably get slapped for saying this...but your application is not your data, data is an artifact of the application. I want my application to be thinking in terms of Customers, Orders and Items, not a technology like DataSets, DataTables and DataRows...cuz who knows how long those will be around.

I agree that there is always a certain amount of coupling, but I prefer that coupling to reach upwards rather than downwards. I can tweak the limbs and leaves of a tree easier than I can alter it's trunk.

I tend to reserve sprocs for reporting as the queries do tend to get a little nastier than the applications general data access.

I also tend to think with proper unit testing early on that scenario's like that one column not being persisted is likely not to be a problem.

Share Improve this answer

Follow

answered Aug 20, 2008 at 21:13

community wiki
Webjedi

3 "your application is not your data, data is an artifact of the application." -- The application is worthless without the data. The data have great value without the application. Applications come and go (get rewritten) all the time, the data in any non-trivial application is always kept. And the data model remains surprisingly stable over time.
– ObiWanKenobi Dec 15, 2010 at 14:32

16

Eric, You are dead on. For any really scalable / easily maintained / robust application the only real answer is to dispense with all the garbage and stick to the basics.

I've followed a similiar trajectory with my career and have come to the same conclusions. Of course, we're considered heretics and looked at funny. But my stuff works and works well.

Every line of code should be looked at with suspicion.

Share Improve this answer

Follow

answered Aug 22, 2008 at 4:49

community wiki
ChrisLively

2 sure, it does certainly works well when you got tons of personnel anjd resources but i think that if you are a one man team a think "new" techniques can help a lot.. – Carl Hörberg Aug 4, 2009 at 15:03

10

I would like to answer with an example similar to the one you proposed.

On my company I had to build a simple CRUD section for products, I build all my entities and a separate DAL. Later another developer had to change a related table and he even renamed several fields. The only file I had to change to update my form was the DAL for that table.

What (in my opinion) entities brings to a project is:

Ortogonality: Changes in one layer might not affect other layers (off course if you make a huge change on the database it would ripple through all the layers but most small changes won't).

Testability: You can test your logic with out touching your database. This increases performance on your tests (allowing you to run them more frequently).

Separation of concerns: In a big product you can assign the database to a DBA and he can optimize the hell out of it. Assign the Model to a business expert that has the knowledge necessary to design it. Assign individual forms to developers more experienced on webforms etc..

Finally I would like to add that most ORM mappers support stored procedures since that's what you are using.

Cheers.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:25

community wiki
Julio César

2 Stored procedures are probably the best example of orthogonality and separation of concerns. If used correctly, they completely encapsulate the database. – Eric Z Beard Sep 12, 2008 at 3:04

1 @Eric Z Beard: Yes but how can you write unit tests around stored procedures while isolating only the logic within the stored procedure? Stored procedures are tightly coupled to the datbase, and most of us ORM types don't like that. In order to write a unit test for a stored procedure you would need to rely on certain data to be in the database. and you can't re-run this test over and over again without that data dependency. That test that you would write would no longer be a unit test but instead it would be an Integration Test. – 7wp Oct 30, 2009 at 17:40

8

I think you may be "biting off more than you can chew" on this topic. Ted Neward was not being flippant when he called it the "Vietnam of Computer Science".

One thing I can absolutely guarantee you is that it will change nobody's point of view on the matter, as has been proven so often on innumerable other blogs, forums, podcasts etc.

It's certainly ok to have open disucssion and debate about a controversial topic, it's just this one has been done so many times that both "sides" have agreed to disagree and just got on with writing software.

If you want to do some further reading on both sides, see articles on Ted's blog, Ayende Rahein, Jimmy Nilson, Scott Bellware, Alt.Net, Stephen Forte, Eric Evans etc.

Share Improve this answer Follow

edited Feb 9, 2010 at 8:16

community wiki
4 revs, 2 users 84%
Ashley Henderson

1 You're right, most people won't change their opinions. I realize the focus of Stack Overflow is supposed to be objective questions, but subjective ones are so much more fun! Personally, I have learned a lot from this discussion.
– Eric Z Beard Sep 12, 2008 at 11:18

I think the various approaches are context specific and that this discussion can serve to disambiguate which scenarios benefit or detract from different persistence models. Experts on the topic won't change their opinions quickly, but this is a question site where people seek others experience.
– TheXenocide Sep 17, 2008 at 20:50

Wow. +1 for the link to the "Vietnam of Computer Science" article which has an excellent introduction to the topic of ORM vs. non ORM. – lambacck Aug 22, 2010 at 20:28

7

@Dan, sorry, that's not the kind of thing I'm looking for. I know the theory. Your statement "is a very bad idea" is not backed up by a real example. We are trying to develop software in less time, with less people, with less mistakes, and we want the ability to easily make changes. Your multi-layer model, in my experience, is a negative in all of the above categories. Especially with regards to making the data model the last thing you do. The physical data model must be an important consideration from day 1.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:46

community wiki
Eric Z Beard

wow, someone else who thinks like me on this...my apps are almost always about manipulation of data, that is what they actually do. – alchemical Jun 14, 2010 at 18:10

This would be better as a comment now that those features are available – Casebash Jul 9, 2010 at 1:48

1 Forgive me for being pedantic, but since this is a popular question, and the mistake you made is a common one, I felt like I should point it out. "Less time" is correct, but "less people" and "less mistakes" should be "fewer people" and

"fewer mistakes." If you have less flour, you can make fewer cookies. (Also, if you use too much flour, you will make too many cookies – a less commonly forgotten distinction.) Again, my apologies; just trying to be helpful.
– Isabelle Wedin Jan 28, 2011 at 18:29

4

I found your question really interesting.
Usually I need entities objects to encapsulate the business logic of an application. It would be really complicated and inadequate to push this logic into the data layer.
What would you do to avoid these entities objects? What solution do you have in mind?

Share Improve this answer Follow

answered Aug 20, 2008 at 20:44

community wiki
jdecuyper

This would be better as a comment – Casebash Jul 9, 2010 at 1:49

4

Entity Objects can facilitate cacheing on the application layer. Good luck caching a datareader.

Share Improve this answer Follow

answered Aug 22, 2008 at 4:51

community wiki
FlySwat

4

We should also talk about the notion what entities really are. When I read through this discussion, I get the impression that most people here are looking at entities in the sense of an Anemic Domain Model. A lot of people are considering the Anemic Domain Model as an antipattern!

There is value in rich domain models. That is what Domain Driven Design is all about. I personally believe that OO is a way to conquer complexity. This means not only technical complexity (like data-access, ui-binding, security ...) **but also complexity in the business domain**!

If we can apply OO techniques to analyze, model, design **and implement** our business problems, this is a tremendous advantage for maintainability and extensibility of non-trivial applications!

There are differences between your entities and your tables. Entities should represent your model, tables just represent the data-aspect of your model!

It is true that data lives longer than apps, but consider this quote from David Laribee: Models are forever ... data is a happy side effect.

Some more links on this topic:

- [Why Setters and Getters are evil]
- [Return of pure OO]
- [POJO vs. NOJO]
- [Super Models Part 2]
- [TDD, Mocks and Design]

Share Improve this answer Follow

answered Nov 5, 2008 at 10:25

community wiki
jbandi

1 Frankly, I am beginning to believe that the data lives longer than the software around it, because often so little attention is being paid to designing the software along a true understanding of the business. – flq Mar 4, 2009 at 18:19

4

Really interesting question. Honestly I can not prove why entities are good. But I can share my opinion why I like them. Code like

```
void exportOrder(Order order, String fileName){...};
```

is not concerned where order came from - from DB, from web request, from unit test, etc. It makes this method more explicitly declare what exactly it requires, instead of taking DataRow and documenting which columns it expects to have and which types they should be. Same

applies if you implement it somehow as stored procedure - you still need to push record id to it, while it not necessary should be present in DB.

Implementation of this method would be done based on Order abstraction, not based on how exactly it is presented in DB. Most of such operations which I implemented really do not depend on how this data is stored. I do understand that some operations require coupling with DB structure for perfomance and scalability purposes, just in my experience there are not too much of them. In my experience very often it is enough to know that Person has .getFirstName() returning String, and .getAddress() returning Address, and address has .getZipCode(), etc - and do not care which tables are involed to store that data.

If you have to deal with such problems as you described, like when additional column breaks report perfomance, then for your tasks DB is a critical part, and you indeed should be as close as possible to it. While entities can provide some convenient abstractions they can hide some important details as well.

Scalability is interesting point here - most of websites which require enormous scalability (like facebook, livejournal, flickr) tend to use DB-ascetic approach, when DB is used as rare as possible and scalability issues are solved by caching, especially by RAM usage. http://highscalability.com/ has some interesting articles on it.

Share Improve this answer Follow

answered Nov 11, 2008 at 10:42

community wiki
Pavel Feldman

2 Scalability in enterprise applications is not often solvable by caching, since so much of it is frequently changing transactional data on tables with millions of rows. I see facebook et. al. as high volume web sites, where the hard part is serving so many web requests. – Eric Z Beard Nov 11, 2008 at 19:01

4

There are other good reasons for entity objects besides abstraction and loose coupling. One of the things I like most is the strong typing that you can't get with a DataReader or a DataTable. Another reason is that when done well, proper entity classes can make the code more maintanable by using first-class constructs for domain-specific terms that anyone looking at the code is likely to understand rather than a bunch of strings with field names in them used for indexing a DataRow. Stored procedures are really orthogonal to the use of an ORM since a lot of mapping frameworks give you the ability to map to sprocs.

I wouldn't consider sprocs + datareaders a substitute for a good ORM. With stored procedures, you're still constrained by, and tightly-coupled to, the procedure's type signature, which uses a different type system than

the calling code. Stored procedures can be subject to modification to acommodate additional options and schema changes. An alternative to stored procedures in the case where the schema is subject to change is to use views--you can map objects to views and then re-map views to the underlying tables when you change them.

I can understand your aversion to ORMs if your experience mainly consists of Java EE and CSLA. You might want to have a look at LINQ to SQL, which is a very lightweight framework and is primarily a one-to-one mapping with the database tables but usually only needs minor extension for them to be full-blown business objects. LINQ to SQL can also map input and output objects to stored procedures' paramaters and results.

The ADO.NET Entity framework has the added advantage that your database tables can be viewed as entity classes inheriting from each other, or as columns from multiple tables aggregated into a single entity. If you need to change the schema, you can change the mapping from the conceptual model to the storage schema without changing the actual application code. And again, stored procedures can be used here.

I think that more IT projects in enterprises fail because of unmaintainability of the code or poor developer productivity (which can happen from, e.g., context switching between sproc-writing and app-writing) than scalability problems of an application.

Share Improve this answer

Follow

edited Jul 4, 2012 at 13:49

community wiki
3 revs, 2 users 94%
Mark Cidade

I think a good compromise would be mapping an ORM to stored procedures, except that this can easily be done poorly: if you just create the 4 CRUD procs for each table, you have accomplished nothing. Can you map big, coarse-grained procs to entities, or does that not really get you anywhere? – Eric Z Beard Sep 12, 2008 at 3:49

In addition to CRUD operations, the Microsoft ORMs let you add methods onto the entity classes that map directly to any stored proc you want to throw at it (provided that all input/output types are mappable). – Mark Cidade Sep 12, 2008 at 4:05

3

I would also like to add to Dan's answer that separating both models could enable your application to be run on different database servers or even database models.

Share Improve this answer

Follow

edited May 23, 2017 at 12:01

community wiki
2 revs
Lars Truijens

3

What if you need to scale your app by load balancing more than one web server? You could install the full app on all web servers, but a better solution is to have the web servers talk to an application server.

But if there aren't any entity objects, they won't have very much to talk about.

I'm not saying that you shouldn't write monoliths if its a simple, internal, short life application. But as soon as it gets moderately complex, or it should last a significant amount of time, you really need to think about a good design.

This saves time when it comes to maintaining it.

By splitting application logic from presentation logic and data access, and by passing DTOs between them, you decouple them. Allowing them to change independently.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:04

community wiki
tgmdbm

3 Lots of people are bringing up de-coupling, and allowing one layer to change without affecting the other. Stored procedures do this better than any ORM! I can radically alter the data model, and as long as the procedures return the same data, nothing breaks. – Eric Z Beard Sep 12, 2008 at 3:03

2 In my opinion stored procedures AND a entity model are not mutually exclusive. Stored procedures can provide a mechanism to store your entity model. The question is: Does your business logic work with the entities or access stored procedures directly? – jbandi Nov 5, 2008 at 9:46

3

You might find this post on comp.object interesting.

I'm not claiming to agree or disagree but it's interesting and (I think) relevant to this topic.

Share Improve this answer Follow

answered Sep 19, 2008 at 23:13

community wiki
Hamish Smith

That's a great post. Sums up my thoughts on ORMs almost perfectly. – Eric Z Beard Sep 20, 2008 at 1:22

3

A question: *How do you handle disconnected applications if all your business logic is trapped in the database?*

In the type of Enterprise application I'm interested in, we have to deal with multiple sites, some of them must be able to function in a disconnected state.
If your business logic is encapsulated in a Domain layer that is simple to incorporate into various application types -say, as a `dll`- then I can build applications that are

aware of the business rules and are able, when necessary, to apply them locally.

In keeping the Domain layer in stored procedures on the database you have to stick with a single type of application that needs a permanent line-of-sight to the database.

It's ok for a certain class of environments, but it certainly doesn't cover the whole spectrum of *Enterprise applications*.

Share Improve this answer Follow

answered Dec 29, 2008 at 3:48

community wiki
Renaud Bompuis

2

@jdecuyper, one maxim I repeat to myself often is "if your business logic is not in your database, it is only a recommendation". I think Paul Nielson said that in one of his books. Application layers and UI come and go, but data usually lives for a very long time.

How do I avoid entity objects? Stored procedures mostly. I also freely admit that business logic tends to reach through all layers in an application whether you intend it to or not. A certain amount of coupling is inherent and unavoidable.

Share Improve this answer Follow

answered Aug 20, 2008 at 20:54

community wiki
Eric Z Beard

I agree, business logic that is in the application only often fails to account for the other ways data can be entered, deleted, or changed. This usually causes data integrity problems dwon the road. – HLGEM Dec 3, 2008 at 19:17

And why you should always use a service layer to handle the mismatch between the object world and the relational world. Business logic bleeding through to every layer is most certainly NOT unavoidable. – cdaq May 14, 2013 at 19:58

2

I have been thinking about this same thing a lot lately; I was a heavy user of CSLA for a while, and I love the purity of saying that "all of your business logic (or at least as much as is reasonably possible) is encapsulated in business entities".

I have seen the business entity model provide a lot of value in cases where the design of the database is different than the way you work with the data, which is the case in a lot of business software.

For example, the idea of a "customer" may consist of a main record in a Customer table, combined with all of the orders the customer has placed, as well as all the customer's employees and their contact information, and some of the properties of a customer and its children may

be determined from lookup tables. It's really nice from a development standpoint to be able to work with the Customer as a single entity, since from a business perspective, the concept of Customer contains all of these things, and the relationships may or may not be enforced in the database.

While I appreciate the quote that "if your business rule is not in your database, it's only a suggestion", I also believe that you shouldn't design the database to enforce business rules, you should design it to be efficient, fast and normalized.

That said, as others have noted above, there is no "perfect design", the tool has to fit the job. But using business entities can really help with maintenance and productivity, since you know where to go to modify business logic, and objects can model real-world concepts in an intuitive way.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:18

community wiki
Guy Starbuck

2

Eric,

No one is stopping you from choosing the framework/approach that you would wish. If you are going to go the "data driven/stored procedure-powered" path,

then by all means, go for it! Especially if it really, really helps you deliver your applications on-spec and on-time.

The caveat being (a flipside to your question that is), ALL of your business rules should be on stored procedures, and your application is nothing more than a thin client.

That being said, same rules apply if you do your application in OOP : be consistent. Follow OOP's tenets, and *that* includes creating entity objects to represent your domain models.

The only real rule here is the word *consistency*. Nobody is stopping you from going DB-centric. No one is stopping you from doing old-school structured (aka, functional/procedural) programs. Hell, no one is stopping anybody from doing COBOL-style code. BUT an application has to be very, very consistent once going down this path, if it wishes to attain any degree of success.

Share Improve this answer Follow

answered Sep 12, 2008 at 2:45

community wiki
Jon Limjap

I agree with consistency throughout the app. To be honest, I changed directions on my current project a while back and never got around to fixing 100% of the original model, which makes things confusing. Good decisions are best made early.
– Eric Z Beard Sep 12, 2008 at 3:26

Eric, true indeed. I once was an OOP zealot (the way others in this thread seem to be) but I met a guy who owns a company who is highly successful selling DB-driven apps. That rocked my world. I'm still an OOP/TDD buff but I don't frown upon DB-centric anymore. – Jon Limjap Sep 12, 2008 at 3:36

The problem is that sometimes people over sell their ideology, you could potentially make a living selling html and javascript only sites, if you had a good methodology for cranking them out. – Mark Rogers Mar 6, 2009 at 16:02

I'm really not sure what you consider "Enterprise Applications". But I'm getting the impression you are defining it as an Internal Application where the RDBMS would be set in stone and the system wouldn't have to be interoperable with any other systems whether internal or external.

But what if you had a database with 100 tables which equate to 4 Stored Procedures for each table just for basic CRUD operations that's 400 stored procedures which need to be maintained and aren't strongly-typed so are susceptible to typos nor can be Unit Tested. What happens when you get a new CTO who is an Open Source Evangelist and wants to change the RDBMS from SQL Server to MySql?

A lot of software today whether Enterprise Applications or Products are using SOA and have some requirements for exposing Web Services, at least the software I am and have been involved with do. Using your approach you

would end up exposing a Serialized DataTable or DataRows. Now this may be deemed acceptable if the Client is guaranteed to be .NET and on an internal network. But when the Client is not known then you should be striving to Design an API which is intuitive and in most cases you would not want to be exposing the Full Database schema. I certainly wouldn't want to explain to a Java developer what a DataTable is and how to use it. There's also the consideration of Bandwith and payload size and serialized DataTables, DataSets are very heavy.

There is no silver bullet with software design and it really depends on where the priorities lie, for me it's in Unit Testable code and loosely coupled components that can be easily consumed be any client.

just my 2 cents

Share Improve this answer Follow

answered Nov 11, 2008 at 9:29

community wiki
user17060

No, my definition of Enterprise Application is the opposite. The schema changes often, and there are many applications that use the db, and it interoperates with many external partners. In a *real* enterprise app, you will *never, ever, ever* change to a different RDBMS. It just doesn't happen.
– Eric Z Beard Nov 11, 2008 at 18:57

And creating 4 procs for each table is a bad practice. It couples you tightly to the data model just like generated sql

out of an ORM so it buys you nothing. The procs need to be coarse-grained business operations, not just CRUD on each table. – Eric Z Beard Nov 11, 2008 at 19:04

But isn't this the answer?: the more code you need to write, the more you need features for large-scale programming support: encapsulation, string typing, refactoring, sophisticated style and error checking, etc.; Java and .NET have a wealth of support in this area, stored procedure languages don't. – reinierpost Apr 1, 2009 at 16:09

2

I'd like to offer another angle to the problem of distance between OO and RDB: history.

Any software has a model of reality that is to some degree an abstraction of reality. No computer program can capture all the complexities of reality, and programs are written just to solve a set of problems from reality. Therefore any software model is a reduction of reality. Sometimes the software model forces reality to reduce itself. Like when you want the car rental company to reserve any car for you as long as it is blue and has alloys, but the operator can't comply because your request won't fit in the computer.

RDB comes from a very old tradition of putting information into tables, called accounting. Accounting was done on paper, then on punch cards, then in computers. But accounting is already a reduction of reality. Accounting has forced people to follow its system so long that it has become accepted reality. That's why it is relatively easy to make computer software for accounting,

accounting has had its information model, long before the computer came along.

Given the importance of good accounting systems, and the acceptance you get from any business managers, these systems have become very advanced. The database foundations are now very solid and noone hesitates about keeping vital data in something so trustworthy.

I guess that OO must have come along when people have found that other aspects of reality are harder to model than accounting (which is already a model). OO has become a very successful idea, but persistance of OO data is relatively underdeveloped. RDB/Accounting has had easy wins, but OO is a much larger field (basically everything that isn't accounting).

So many of us have wanted to use OO but we still want safe storage of our data. What can be safer than to store our data the same way as the esteemed accounting system does? It is an enticing prospects, but we all run into the same pitfalls. Very few have taken the trouble to think of OO persistence compared to the massive efforts by the RDB industry, who has had the benefit of accounting's tradition and position.

Prevayler and db4o are some suggestions, I'm sure there are others I haven't heard of, but none have seemed to get half the press as, say, hibernation.

Storing your objects in good old files doesn't even seem to be taken seriously for multiuser applications, and especially web applications.

In my everyday struggle to close the chasm between OO and RDB I use OO as much as possible but try to keep inheritance to a minimum. I don't often use SPs. I'll use the advanced query stuff only in aspects that look like accounting.

I'll be happily supprised when the chasm is closed for good. I think the solution will come when Oracle launches something like "Oracle Object Instance Base". To really catch on, it will have to have a reassuring name.

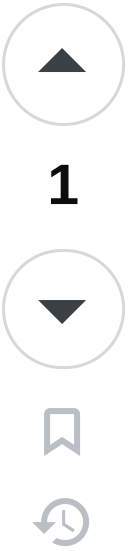
Share Improve this answer Follow

answered Dec 3, 2008 at 8:24

community wiki
Guge

I don't think you need ORM for OO to be considered useful. I use stored procs and write a lot of static helper classes in my code, but those classes are built on the enormous .NET framework, which is a fantastic collection of objects.
– Eric Z Beard Dec 3, 2008 at 20:46

Your logic makes sense, but I don't think the premise is sound. I have never heard of anything that can't be mapped with RDB. – Jeff Davis Jul 14, 2009 at 16:40



Not a lot of time at the moment, but just off the top of my head...

The entity model lets you give a consistent interface to the database (and other possible systems) even beyond what a stored procedure interface can do. By using enterprise-wide business models you can make sure that all applications affect the data consistently which is a VERY important thing. Otherwise you end up with bad data, which is just plain evil.

If you only have one application then you don't really have an "enterprise" system, regardless of how big that application or your data are. In that case you can use an approach similar to what you talk about. Just be aware of the work that will be needed if you decide to grow your systems in the future.

Here are a few things that you should keep in mind (IMO) though:

1. Generated SQL code is bad (exceptions to follow). Sorry, I know that a lot of people think that it's a huge time saver, but I've never found a system that could generate more efficient code than what I could write and often the code is just plain horrible. You also often end up generating a ton of SQL code that never gets used. The exception here is very simple patterns, like maybe lookup tables. A lot of people get carried away on it though.

2. Entities <> Tables (or even logical data model entities necessarily). A data model often has data rules that should be enforced as closely to the database as possible which can include rules around how table rows relate to each other or other similar rules that are too complex for declarative RI. These should be handled in stored procedures. If all of your stored procedures are simple CRUD procs, you can't do that. On top of that, the CRUD model usually creates performance issues because it doesn't minimize round trips across the network to the database. That's often the biggest bottleneck in an enterprise application.

answered Oct 21, 2008 at 15:13

community wiki
Tom H

Agreed on generated SQL. It always causes more problems than it solves. And I'm very much against simply creating a CRUD layer with stored procs. The procs should be as coarse-grained as possible. Not sure how you define "one application". – Eric Z Beard Oct 21, 2008 at 16:52

By one application, I mean a single application written by a single group in the organization. Where I'm consulting at now they have a corporate database which is accessed by at least three separate groups working on three different applications with limited communication between them. – Tom H Oct 23, 2008 at 15:48

1

Sometimes, your application and data layer are not that tightly coupled. For example, you may have a telephone billing application. You later create a separate application which monitors phone usage to a) better advertise to you b) optimise your phone plan.

These applications have different concerns and data requirements (even the data is coming out of the same database), they would drive different designs. Your code base can end up an absolute mess (in either application) and a nightmare to maintain if you let the database drive the code.

Share Improve this answer Follow

answered Dec 29, 2008 at 7:01

community wiki
billybob

1

Applications that have domain logic separated from the data storage logic are adaptable to any kind of data source (database or otherwise) or UI (web or windows(or linux etc.)) application.

Your pretty much stuck in your database, which isn't bad if your with a company who is satisfied with the current database system your using. However, because databases evolve overtime there might be a new database system that is really neat and new that your

company wants to use. What if they wanted to switch to a web services method of data access (like Service Orientated architecture sometime does). You might have to port your stored procedures all over the place.

Also the domain logic abstracts away the UI, which can be more important in large complex systems that have ever evolving UIs (especially when they are constantly searching for more customers).

Also, while I agree that there is no definitive answer to the question of stored procedures versus domain logic. I'm in the domain logic camp (and I think they are winning over time), because I believe that elaborate stored procedures are harder to maintain than elaborate domain logic. But that's a whole other debate

Share Improve this answer Follow

answered Mar 3, 2009 at 17:21

community wiki
Mark Rogers

0

I think that you are just used to writing a specific kind of application, and solving a certain kind of problem. You seem to be attacking this from a "database first" perspective. There are lots of developers out there where data is persisted to a DB but performance is not a top priority. In lots of cases putting an abstraction over the

persistence layer simplifies code greatly and the performance cost is a non-issue.

Whatever you are doing, it's not OOP. It's not wrong, it's just not OOP, and it doesn't make sense to apply your solutions to every othe problem out there.

Share Improve this answer Follow

answered Aug 20, 2008 at 21:03

community wiki
Tim Frey

> Data *always* comes first. Its the reason you have the computer program in the first place. So "database first" is possibly the only valid approach for designing apps.
> – gbjbaanb Nov 15, 2008 at 15:54

0

Interesting question. A couple thoughts:

1. How would you unit test if all of your business logic was in your database?
2. Wouldn't changes to your database structure, specifically ones that affect several pages in your app, be a major hassle to change throughout the app?

Share Improve this answer Follow

answered Aug 20, 2008 at 21:26

community wiki
Kevin Pang

0

Good Question!

One approach I rather like is to create an iterator/generator object that emits instances of objects that are relevant to a specific context. Usually this object wraps some underlying database access stuff, but I don't need to know that when using it.

For example,

> An AnswerIterator object generates AnswerIterator.Answer objects. Under the hood it's iterating over a SQL Statement to fetch all the answers, and another SQL statement to fetch all related comments. But when using the iterator I just use the Answer object that has the minimum properties for this context. With a little bit of skeleton code this becomes almost trivial to do.

I've found that this works well when I have a huge dataset to work on, and when done right, it gives me small, transient objects that are relatively easy to test.

It's basically a thin veneer over the Database Access stuff, but it still gives me the flexibility of abstracting it when I need to.

Share Improve this answer Follow

answered Sep 12, 2008 at 2:36

community wiki
Allain Lalonde

0

The objects in my apps tend to relate one-to-one to the database, but I'm finding using Linq To Sql rather than sprocs makes it much easier writing complicated queries, especially being able to build them up using the deferred execution. e.g. from r in Images.User.Ratings where etc. This saves me trying to work out several join statements in sql, and having Skip & Take for paging also simplifies the code rather than having to embed the row_number & 'over' code.

Share Improve this answer Follow

answered Sep 12, 2008 at 3:04

community wiki
peterorum

There's a big danger in doing things this way. Most complex queries end up needing to be re-written entirely by a DBA to get them to scale. No amount of index tuning can do what changing a query can sometimes do. This type of Linq2Sql is extremely tight coupling. – Eric Z Beard Sep 12, 2008 at 3:22

Why stop at entity objects? If you don't see the value with entity objects in an enterprise level app, then just do your data access in a purely functional/procedural language and wire it up to a UI. Why not just cut out all the OO "fluff"?

0

Share Improve this answer Follow

answered Oct 21, 2008 at 16:24

community wiki
Pragmatic Agilist

I don't see OO as "fluff". It's just that over the last decade or so, MSFT, Sun, etc have written 99% of the objects we'll ever need. Just because I write lots of static classes on top of the framework, doesn't mean I'm not using OO. – Eric Z Beard Oct 21, 2008 at 16:55