Are there good reasons not to use an ORM? [closed]

Asked 16 years, 2 months ago Modified 3 years, 11 months ago

Viewed 56k times

120

During my apprenticeship, I have used NHibernate for some smaller projects which I mostly coded and designed on my own. Now, before starting some bigger project, the discussion arose how to design data access and whether or not to use an ORM layer. As I am still in my apprenticeship and still consider myself a beginner in enterprise programming, I did not really try to push in my opinion, which is that using an object relational mapper to the database can ease development quite a lot. The other coders in the development team are much more experienced than me, so I think I will just do what they say. :-)

However, I do not completely understand two of the main reasons for not using NHibernate or a similar project:

1. One can just build one's own data access objects with SQL queries and copy those queries out of Microsoft SQL Server Management Studio.
2. Debugging an ORM can be hard.

So, of course I could just build my data access layer with a lot of `SELECT`s etc, but here I miss the advantage of automatic joins, lazy-loading proxy classes and a lower maintenance effort if a table gets a new column or a column gets renamed. (Updating numerous `SELECT`, `INSERT` and `UPDATE` queries vs. updating the mapping config and possibly refactoring the business classes and DTOs.)

Also, using NHibernate you can run into unforeseen problems if you do not know the framework very well. That could be, for example, trusting the Table.hbm.xml where you set a string's length to be automatically validated. However, I can also imagine similar bugs in a "simple" SqlConnection query based data access layer.

Finally, are those arguments mentioned above really a good reason not to utilise an ORM for a non-trivial database based enterprise application? Are there probably other arguments they/I might have missed?

(I should probably add that I think this is like the first "big" .NET/C# based application which will require teamwork. Good practices, which are seen as pretty normal on

Stack Overflow, such as unit testing or continuous integration, are non-existing here up to now.)

c# orm enterprise

Share

Improve this question

Follow

edited Jan 6, 2021 at 8:40

Amit Joshi
16.3k ● 23 ● 84 ● 148

asked Oct 11, 2008 at 14:43

hangy
10.9k ● 6 ● 45 ● 63

20 Answers

Sorted by: Highest score (default)

71

The short answer is yes, there are really good reasons. As a matter of fact there are cases where you just cannot use an ORM.

Case in point, I work for a large enterprise financial institution and we have to follow a lot of security guidelines. To meet the rules and regulations that are put upon us, the only way to pass audits is to keep data access within stored procedures. Now some may say that's just plain stupid, but honestly it isn't. Using an ORM tool means the tool/developer can insert, select, update or delete whatever he or she wants. Stored procedures provide a lot more security, especially in environments when dealing with client data. I think this is the biggest reason to consider. Security.

Share Improve this answer Follow

answered Oct 11, 2008 at 15:12

Keith Elder
1,809 ● 1 ● 13 ● 13

26 I think this is more a limitation of current orm libraries that don't support stored procedures than a fundamental limitation in or mapping. There are orm's that can handle stored proc's and views and there's a lot to say for the orm + stored procs solution. – Mendelt Oct 11, 2008 at 16:45

4 In the case of a good ORM, you should be able to get it to use SPs anyway (of course, this mitigates a lot of the benefits...) – davidkemp Oct 17, 2008 at 16:53

3 The topic of why SPs do not really provide more security than an ORM is well tilled ground. Here's just one article that addresses it well: ayende.com/Blog/archive/2007/11/18/… – Tim Scott Oct 22, 2008 at 3:18

1 Well, in Sql Server 2005 can't you just designate a schema in a database just for the ORM layer? Or is that not enough? If the applications are web apps, then one thing is connecting to a db. Security then left to the application layer. – Min May 29, 2009 at 15:48

2 Of course you can restrict what the user can do with an ORM. You just set the user security settings in SQL and give them priviliges to insert/update/delete what you want them to. It would be the same as setting security privelages for stored procedures – Doctor Jones Oct 27, 2010 at 8:27

69

The sweet spot of ORMs

ORMs are useful for automating the 95%+ of queries where they are applicable. Their particular strength is where you have an application with a strong object model architecture and a database that plays nicely with that object model. If you're doing a new build and have strong modelling skills on your team then you will probably get good results with an ORM.

You may well have a handful of queries that are better done by hand. In this case, don't be afraid to write a few stored procedures to handle this. Even if you intend to port your app across multiple DBMS platforms the database dependent code will be in a minority. Bearing in mind that you will need to test your application on any platform on which you intend to support it, a little bit of extra porting effort for some stored procedures isn't going to make a lot of difference to your TCO. For a first approximation, 98% portable is just as good as 100% portable, and far better than convoluted or poorly performing solutions to work around the limits of an ORM.

I have seen the former approach work well on a very large (100's of staff-years) J2EE project.

Where an ORM may not be the best fit

In other cases there may be approaches that suit the application better than an ORM. Fowler's *Patterns of Enterprise Application Architecture* has a section on data access patterns that does a fairly good job of cataloguing various approaches to this. Some examples I've seen of situations where an ORM may not be applicable are:

- On an application with a substantial legacy code base of stored procedures you may want to use a functionally oriented (not to be confused with functional languages) data access layer to wrap the incumbent sprocs. This re-uses the existing (and therefore tested and debugged) data access layer and database design, which often represents quite a substantial development and testing effort, and saves on having to migrate data to a new database model. It is often quite a good way wrapping Java layers around legacy PL/SQL code bases, or re-targeting rich client VB, Powerbuilder or Delphi apps with web interfaces.
- A variation is where you inherit a data model that is not necessarily well suited to O-R mapping. If (for example) you are writing an interface that populates or extracts data from a foreign interface you may be better off working direclty with the database.
- Financial applications or other types of systems where cross-system data integrity is important, particularly if you're using complex distributed transactions with two-phase commit. You may need to micromanage your transactions better than an ORM is capable of supporting.
- High-performance applications where you want to really tune your database access. In this case, it may

be preferable to work at a lower level.

- Situations where you're using an incumbent data access mechanism like ADO.Net that's 'good enough' and playing nicely with the platform is of greater benefit than the ORM brings.
- Sometimes data is just data - it may be the case (for example) that your application is working with 'transactions' rather than 'objects' and that this is a sensible view of the domain. An example of this might be a financials package where you've got transactions with configurable analysis fields. While the application itself may be built on an O-O platform, it is not tied to a single business domain model and may not be aware of much more than GL codes, accounts, document types and half a dozen analysis fields. In this case the application isn't aware of a business domain model as such and an object model (beyond the ledger structure itself) is not relevant to the application.

Share Improve this answer Follow

edited Apr 11, 2014 at 8:28

answered Jul 7, 2010 at 15:29

ConcernedOfTunbridge Wells

66.5k ● 15 ● 148 ● 198

"or other types of systems where data integrity is important"... Social websites aside, if the integrity of your data is not

important, why bother building a system at all?
– ObiWanKenobi Oct 26, 2010 at 21:49

3 The point specifically refers to distributed transactions where multiple systems must guarantee a commit or rollback synchronously or off a message queue. Not all of these systems will be built for you and therefore will not necessarily support an ORM. You may have to explicitly manage the transactions, which may require that you use a lower level tooklit than an ORM. – ConcernedOfTunbridgeWells Oct 27, 2010 at 8:19

2 I know it's been over a year, but +1 for you for mentioning the fact that sometimes data is simply just that, data.
– luis.espinal Apr 22, 2011 at 12:13

40

First off - using an ORM will not make your code any easier to test, nor will it necessarily provide any advantages in a Continuous Integration scenerio.

In my experience, whilst using an ORM can increase the speed of development, the biggest issues you need to address are:

1. Testing your code
2. Maintaining your code

The solutions to these are:

1. Make your code testable (using SOLID principles)
2. Write automated tests for as much of the code as possible
3. Run the automated tests as often as possible

Coming to your question, the two objections you list seem more like ignorance than anything else.

Not being able to write SELECT queries by hand (which, I presume, is why the copy-paste is needed) seems to indicate that there's a urgent need for some SQL training.

There are two reasons why I'd not use an ORM:

1. It is strictly forbidden by the company's policy (in which case I'd go work somewhere else)
2. The project is *extremely* data intensive and using vendor specific solutions (like BulkInsert) makes more sense.

The usual rebuffs about ORMs (NHibernate in particular) are:

1. Speed

   There is no reason why using an ORM would be any slower than hand coded Data Access. In fact, because of the caching and optimisations built into it, it can be quicker. A good ORM will produce a repeatable set of queries for which you can optimise your schema. A good ORM will also allow efficient retrieval of associated data using various fetching strategies.

2. Complexity

   With regards to complexity, using an ORM means less code, which generally means less complexity.

> Many people using hand-written (or code generated) data access find themselves writing their own framework over "low-level" data access libraries (like writing helper methods for ADO.Net). These equate to more complexity, and, worse yet, they're rarely well documented, or well tested.
> If you are looking specifically at NHibernate, then tools like Fluent NHibernate and Linq To NHibernate also soften the learning curve.

The thing that gets me about the whole ORM debate is that the same people who claim that using an ORM will be too hard/slow/whatever are the very same people who are more than happy using Linq To Sql or Typed Datasets. Whilst the Linq To Sql is a big step in the right direction, it's still light years behind where some of the open source ORMs are. However, the frameworks for both Typed Datasets and for Linq To Sql is still hugely complex, and using them to go too far of the (Table=Class) + (basic CRUD) is stupidly difficult.

My advice is that if, at the end of the day, you can't get an ORM, then make sure that your data access is separated from the rest of the code, and that you you follow the Gang Of Four's advice of coding to an interface. Also, get a Dependancy Injection framework to do the wiring up.

(How's that for a rant?)

Share Improve this answer Follow

edited Jun 20, 2020 at 9:12

Community Bot
1 ● 1

answered Oct 17, 2008 at 17:36

david kemp

7,850 ● 28 ● 41

35

There are a wide range of common problems for which ORM tools like Hibernate are a god-send, and a few where it is a hindrance. I don't know enough about your project to know which it is.

One of Hibernate's strong points is that you get to say things only 3 times: every property is mentioned in the class, the .hbm.xml file, and the database. With SQL queries, your properties are in the class, the database, the select statements, the insert statements, the update statements, the delete statements, and all the marshalling and unmarshalling code supporting your SQL queries! This can get messy fast. On the other hand, you know how it works. You can debug it. It's all right there in your own persistence layer, not buried in the bowels of a 3rd party tool.

Hibernate could be a poster-child for Spolsky's Law of Leaky Abstractions. Get a little bit off the beaten path, and you need to know deep internal workings of the tool. It can be very annoying when you know you could have fixed the SQL in minutes, but instead you are spending hours trying to cajole your dang tool into generating reasonable SQL. Debugging is sometimes a nightmare, but it's hard to convince people who have not been there.

EDIT: You might want to look into iBatis.NET if they are not going to be turned around about NHibernate and they want control over their SQL queries.

EDIT 2: Here's the big red flag, though: "Good practices, which are seen as pretty normal on Stack Overflow, such as unit testing or continuous integration, are non-existing here up to now." So, these "experienced" developers, what are they experienced in developing? Their job security? It sounds like you might be among people who are not particularly interested in the field, so don't let them kill your interest. You need to be the balance. Put up a fight.

Share Improve this answer Follow

edited Jun 3, 2010 at 17:11
J. Polfer
12.5k ● 10 ● 56 ● 83

answered Oct 11, 2008 at 15:29
Alan Hensel
833 ● 7 ● 16

3 The repetition isn't true any longer. If you use JPA annotations, you only need to specify things once. Hibernate will even build your database create statements for you, although that's most useful for determining if your mapping was correct. – jonathan-stafford Oct 17, 2008 at 17:49

Good balanced discussion of the issues. My Entity framework experience exactly fits your description of "spending hours trying to cajole your dang tool", and "it's hard to convince people who have not been there". It was apparently faster to write up, but is a huge time waster to make perform really well. – user334911 Feb 28, 2014 at 17:29

4 8 years have passed, but it's still true: "It can be very annoying when you know you could have fixed the SQL in minutes, but instead you are spending hours trying to cajole your dang tool into generating reasonable SQL."
– Ruslan Stelmachenko Jul 9, 2016 at 0:37

30

There's been an explosion of growth with ORMs in recent years and your more experienced coworkers may still be thinking in the "every database call should be through a stored procedure" mentality.

Why would an ORM make things harder to debug? You'll get the same result whether it comes from a stored proc or from the ORM.

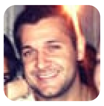
I guess the only real detriment that I can think of with an ORM is that the security model is a little less flexible.

EDIT: I just re-read your question and it looks they are copy and pasting the queries into inline sql. This makes the security model the same as an ORM, so there would be absolutely no advantage over this approach over an ORM. If they are using unparametrized queries then it would actually be a security risk.

Share Improve this answer Follow

edited Oct 11, 2008 at 15:06

answered Oct 11, 2008 at 14:55



Giovanni Galbo
13.1k ● 13 ● 61 ● 79

1 Harder to debug: If an exception is thrown, you probably need to know the framework instead of knowing the SqlConnection’s exceptions etc. – hangy Oct 11, 2008 at 14:59

4 Wouldn't the sql exception be part of the ORM exception? – Giovanni Galbo Oct 11, 2008 at 15:01

1 Yes, most wrap the exception, so you would still be able to get the originating exception thorugh .inner – mattlant Oct 11, 2008 at 15:09

As I said, I did not bring up that argument. :) Of course, the real SQL exception should be somewhere down the stack trace. As you might have read from my question, I kind of agree with your answer, but I will wait with accepting it. Maybe someone else comes up with really good reasons against ORM. – hangy Oct 11, 2008 at 15:10

How about if your database structure is very different from your business objects. Wouldn't it turn it all into an overhead? programming the mappings with xml, instead of just providing the necessay functions on the db side with SP's...? – Uri Abramson Feb 17, 2014 at 9:48

I worked on one project where not using an ORM was very successfully. It was a project that

13

1. Had to be horizontally scalealbe from the start
2. Had to be developed quickly
3. Had a relatively simple domain model

The time that it would have taken to get NHibernate to work in a horizontally partitioned structure would have been much longer than the time that it took to develop a super simple datamapper that was aware of our partitioning scheme...

So, in 90% of projects that I have worked on an ORM has been an invaluable help. But there are some very specific circumstances where I can see not using an ORM as being best.

Share Improve this answer Follow

answered Oct 11, 2008 at 15:14

Mike
1,993 ● 1 ● 14 ● 19

You gave some good points against using an ORM in some specific cases. However, this project belongs to those 90 % where the ORM could possibly help to reduce costs of development and maintenance. – hangy Oct 11, 2008 at 15:28

Totally agree - sorry for not answering your question directly! I believe that the specific reasons you mentioned are quite invalid :) – Mike Oct 12, 2008 at 3:20

Horizontally scaling NHibernate: blechie.com/WPierce/archive/2008/06/08/…, darioquintana.com.ar/blogging/?p=25 – Mauricio Scheffer Mar 10, 2009 at 0:45

12

For a non-trivial database based enterprise application, there really is no justifying not using an ORM.

Features aside:

- By not using an ORM, you are solving a problem that has already solved repeatedly by large communities or companies with significant resources.
- By using an ORM, the core piece of your data access layer benefits from the debugging efforts of that community or company.

To put some perspective in the argument, consider the advantages of using ADO.NET vs. writing the code to parse the tabular data stream oneself.

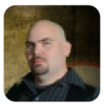
I have seen ignorance of how to use an ORM justify a developer's disdain for ORMs For example: eager loading (something I noticed you didn't mention). Imagine you want to retrieve a customer and all of their orders, and for those all of the order detail items. If you rely on lazy loading only, you will walk away from your ORM experience with the opinion: "ORMs are slow." If you learn how to use eager loading, you will do in 2 minutes with 5 lines of code, what your colleagues will take a half a day to implement: one query to the database and binding the results to a hierarchy of objects. Another example would be the pain of manually writing SQL queries to implement paging.

The possible exception to using an ORM would be if that application were an ORM framework designed to apply specialized business logic abstractions, and designed to be reused on multiple projects. Even if that were the case, however, you would get faster adoption by enhancing an existing ORM with those abstractions.

Do not let the experience of your senior team members drag you in the opposite direction of the evolution of computer science. I have been developing professionally for 23 years, and one of the constants is the disdain for the new by the old-school. ORMs are to SQL as the C language was to assembly, and you can bet that the equivalents to C++ and C# are on their way. One line of new-school code equals 20 lines of old-school.

Share Improve this answer Follow

answered Oct 17, 2012 at 4:05



DaveMorganTexas
887 ● 10 ● 13

11

Let me first say that ORMs can make your development life easier if integrated properly, but there are a handful of problems where the ORM can actually prevent you from achieving your stated requirements and goals.

I have found that when designing systems that have heavy performance requirements that I am often challenged to find ways to make the system more performant. Many times, I end up with a solution that has a heavy write performance profile (meaning we're writing data a lot more than we're reading data). In these cases, I want to take advantage of the facilities the database platform offers to me in order to reach our performance goals (it's OLTP, not OLAP). So if I'm using SQL Server and I know I have a lot of data to write, why wouldn't I use a bulk insert... well, as you may have already discovered, most ORMS (I don't know if even a single one does) do not have the ability to take advantage of platform specific advantages like bulk insert.

You should know that you can blend the ORM and non-ORM techniques. I've just found that there are a handful of edge cases where ORMs can not support your requirements and you have to work around them for those cases.

Share Improve this answer Follow

answered Oct 11, 2008 at 15:46

Ajaxx
2,510 ● 4 ● 27 ● 38

11

I think there are many good reasons to not use an ORM. First and foremost, I'm a .NET developer and I like to stick within what the wonderful .NET framework has already provided to me. It does everything I possibly need it to. By doing this, you stay with a more standard approach, and thus there is a much better chance of any other developer working on the same project down the road being able to pick up what's there and run with it.

The data access capabilities already provided by Microsoft are quite ample, there's no reason to discard them.

I've been a professional developer for 10 years, lead multiple very successful million+ dollar projects, and I have never once written an application that needed to be able to switch to any database. Why would you ever want a client to do this? Plan carefully, pick the right database for what you need, and stick with it. Personally SQL Server has been able to do anything I've ever needed to do. It's easy and it works great. There's even a free version that supports up to 10GB data. Oh, and it works awesome with .NET.

I have recently had to start working on several projects that use an ORM as the datalayer. I think it's bad, and something extra I had to learn how to use for no reason whatsoever. In the insanely rare circumstance the customer did need to change databases, I could have easily reworked the entire datalayer in less time than I've spent fooling with the ORM providers.

Honestly I think there is one real use for an ORM: If you're building an application like SAP that really does need the ability to run on multiple databases. Otherwise as a solution provider, I tell my clients this application is designed to run on this database and that is how it is. Once again, after 10 years and a countless number of applications, this has never been a problem.

Otherwise I think ORMs are for developers that don't understand less is more, and think the more cool 3rd party tools they use in their app, the better their app will be. I'll leave things like this to the die hard uber geeks while I crank out much more great software in the meantime that any developer can pick up and immediately be productive with.

Share Improve this answer Follow

edited Aug 15, 2015 at 20:40
Brad Larson
170k ● 45 ● 400 ● 572

answered Jan 31, 2012 at 19:31
Danny
139 ● 1 ● 2

---

5 I really don't understand why this answer is voted down. Keeping it simple by using whatever your environment has to offer is great practice. As an experienced clean code guru, i find that orm's are hard to read, and therefor hard to maintain. You don't know what queries are executed exactly when you have complex relationships in your application (which you eventually will). It's becoming impossible to debug such a mess in the long run. I say, keep it simple, don't use ORM. – David Apr 14, 2015 at 16:41

This is funny. I've been a developer for 24 years and I've never been involved in a project where it was possible to support only one RDBMS vendor. Every project REQUIRED us to support multiple RDBMS vendors, because we needed to be flexible enough to work with customers who REQUIRE Oracle, and work with other customers who REQURE SQL Server. If you're building a stovepipe application in a vacuum, then yeah you can just dictate the RDBMS. But I've never

been involved in a project where that was acceptable.
– deltamind106 Oct 13, 2015 at 19:30

I have had many applications where I can dictate the RDBMS mainly because on a good number of internal applications and customer facing that viewed "our" data. I have been a developer for 24 years as well. – jeffkenn May 10, 2016 at 15:03

9

When you need to update 50000000 records. Set a flag or whatever.

Try doing this using an ORM **without** calling a stored procedure or native SQL commands..

Update 1 : Try also retrieving one record with only a few of its fields. (When you have a very "wide" table). Or a scalar result. ORMs suck at this too.

UPDATE 2 : It seems that EF 5.0 beta promises batch updates but this is very hot news (2012, January)

Share Improve this answer Follow

edited Jan 25, 2012 at 19:40

answered Oct 26, 2010 at 21:30

Andrei Rînea
20.7k ● 18 ● 121 ● 169

docs.jboss.org/hibernate/core/3.3/reference/en/html/…
– dotjoe Oct 26, 2010 at 21:42

8

I think that maybe when you work on bigger systems you can use a code generator tool like CodeSmith instead of a ORM... I recently found this: [Cooperator Framework](#) which generates SQL Server Stored Procedures and also generates your business entities, mappers, gateways, lazyload and all that stuff in C#...check it out...it was written by a team here in Argentina...

I think it's in the middle between coding the entire data access layer and use a ORM...

Share Improve this answer Follow

edited Nov 17, 2011 at 1:40
user142162

answered Oct 16, 2008 at 3:36
pabloide86
946 ● 1 ● 7 ● 17

6

Personally, i have (until recently) opposed to use an ORM, and used to get by with writing a data access layer encapsulating all the SQL commands. The main objection to ORMs was that I didn't trust the ORM implementation to write exactly the right SQL. And, judging by the ORMs i used to see (mostly PHP libraries), i think i was totally right.

Now, most of my web development is using Django, and i found the included ORM really convenient, and since the data model is expressed first in their terms, and only later in SQL, it does work perfectly for my needs. I'm sure it wouldn't be too hard to outgrow it and need to supplement with hand-written SQL; but for CRUD access is more than enough.

I don't know about NHibernate; but i guess it's also "good enough" for most of what you need. But if other coders don't trust it; it will be a prime suspect on every data-related bug, making verification more tedious.

You could try to introduce it gradually in your workplace, focus first on small 'obvious' applications, like simple data access. After a while, it might be used on prototypes, and it might not be replaced...

Share Improve this answer Follow

answered Oct 11, 2008 at 16:09

Javier
62.4k ● 9 ● 81 ● 126

5

If it is an OLAP database (e.g. static, read-only data used for reporting/analytics, etc.) then implementing an ORM framework is not appropriate. Instead, using the database's native data access functionality such as stored procedures would be preferable. ORMs are better suited for transactional (OLTP) systems.

Share Improve this answer Follow

answered Dec 2, 2008 at 20:32

Ray
192k ● 99 ● 227 ● 206

Are you sure about that? OLTPs are as unsuitable with ORMs as OLAPs are (I mean, depending on the complexity).

There is a wide range of applications between these two extremes for which a ORM does a good job. But for OLAPs and OLTPs, they can become a hindrance. – luis.espinal Apr 22, 2011 at 12:17

3

Runtime performance is the only real downside I can think of but I think that's more than a fair trade-off for the time ORM saves you developing/testing/etc. And in most cases you should be able to locate data bottlenecks and alter your object structures to be more efficient.

I haven't used Hibernate before but one thing I have noticed with a few "off-the-shelf" ORM solutions is a lack of flexibility. I'm sure this depends on which you go with and what you need to do with it.

Share Improve this answer Follow

answered Oct 11, 2008 at 15:16

Oli
240k ● 65 ● 226 ● 303

I remember some people saying that the optimised queries and not loading data not necessary in some situations (ie. lazy loading of joins) can actually speed up things. Manually implementing this in a custom DAL might be more work and worth less time. – hangy Oct 11, 2008 at 15:21

3

There are two aspects of ORMs that are worrisome. First, they are code written by someone else, sometimes closed source, sometimes open source but huge in scope. Second, they copy the data.

The first problem causes two issues. You are relying on outsiders code. We all do this, but the choice to do so should not be taken lightly. And what if it doesn't do what you need? When will you discover this? You live inside the box that your ORM draws for you.

The second problem is one of two phase commit. The relational database is being copied to a object model. You change the object model and it is supposed to update the database. This is a two phase commit and not the easiest thing to debug.

Share Improve this answer Follow

answered Oct 11, 2008 at 16:08

dacracot
22.3k ● 28 ● 109 ● 159

2 Umm...if you are using, let's say .NET, you are relying on their code (which you cannot modify). I don't see how that is any more "ok" than relying on NHibernate's code, for example. Being paranoid of libraries you have not written yourself is relatively ridiculous. Sounds like you suffer from NIH – Jason Bunting Oct 11, 2008 at 17:12

compilers and VMs are a relatively stable part of CS, and not so hard to veryfy with tests. an ORM design has a lot of ways to be 'slightly wrong', or incomplete documentation. all this makes it harder to trust than something so basic as the compiler. – Javier Oct 11, 2008 at 19:47

1 It is a warning... not a do not use statement. And that is only one warning out of three issues. – dacracot Oct 11, 2008 at 19:57

1 @dacracot: You are relying on lots of foreign code all the time... starting from your operating system, so I don't think

your point is a valid one. Second point can be mitigated using optimistic locking, moreover it doesn't have much to do with two-phase commit. – Adam Byrtek Dec 2, 2008 at 20:48

1 dacracot is spot-on when speaking of some of the less mature ORMs. I'm sure there are some that rock balls, but most will be imperfect, and it can be an issue in some (not most) environments. Regarding the two-phase commit... there are ways to model the DB transaction itself in code, but why add a layer of indirection that doesn't provide any abstraction? – Tom May 20, 2009 at 5:11

3

I suggest this reading for a list of the downsides of ORMs.

http://blogs.tedneward.com/2006/06/26/The+Vietnam+Of+Computer+Science.aspx

For my self, I've found ORMs very useful for most applications I've written!

/Asger

Share Improve this answer Follow

answered Apr 21, 2009 at 19:27

asgerhallas
17.7k ● 8 ● 52 ● 70

3

The experience I've had with Hibernate is that its semantics are subtle, and when there's problems, it's a bit hard to understand what's going wrong under the hood. I've heard from a friend that often one starts with Criteria, then needs a bit more flexibility and needs HQL, and later notices that after all, raw SQL is needed (for example, Hibernate doesn't have union AFAIK).

Also with ORM, people easily tend to overuse existing mappings/models, which leads to that there's an object with lots of attributes that aren't initiliazed. So after the query, inside transaction Hibernate makes additional data fetching, which leads to potential slow down. Also sadly, the hibernate model object is sometimes leaked into the view architecture layer, and then we see LazyInitializationExceptions.

To use ORM, one should really understand it. Unfortunately one gets easily impression that it's easy while it's not.

Share Improve this answer Follow

answered May 20, 2009 at 3:36

egaga
21.7k ● 10 ● 47 ● 61

Hibernate doesn't support UNION? That's a pretty fundamental part of set theory! I suppose it just indicates a different way to think about the problem. – Tom May 20, 2009 at 5:12

3

Not to be an answer per se, I want to rephrase a quote I've heard recently. "A good ORM is like a Yeti, everyone talks about one but no one sees it."

Whenever I put my hands on an ORM, I usually find myself struggling with the problems/limitations of that

ORM. At the end, yes it does what I want and it was written somewhere in that lousy documentation but I find myself losing another hour I will never get. Anyone who used nhibernate, then fluent nhibernate on postgresql would understand what I've been thru. Constant feeling of "this code is not under my control" really sucks.

I don't point fingers or say they're bad, but I started thinking of what I'm giving away just to automate CRUD in a single expression. Nowadays I think I should use ORM's less, maybe create or find a solution that enables db operations at minimum. But it's just me. I believe some things are wrong in this ORM arena but I'm not skilled enough to express it what not.

Share Improve this answer Follow

answered May 8, 2011 at 21:30

detay
1,072 ● 11 ● 19

Many years (and ORMs) later I decided to use Postgresql / EntityFramework with Code First Migrations. It lets me enable data persistence via classes I've written and I'm finally able to sync/version my database changes integrated to my production environment. It's almost a transparent layer of data access, and saves a great deal of time during development but in the meantime I can go deep and write advanced queries when I want to. Ofcourse, it's a dotnet solution but I am finally at peace with the db access. – detay Nov 7, 2015 at 18:24

I think that using an ORM is still a good idea. Especially considering the situation you give. It sounds by your post

2

you are the more experienced when it comes to the db access strategies, and I would bring up using an ORM.

There is no argument for #1 as copying and pasting queries and hardcoding in text gives no flexibility, and for #2 most orm's will wrap the original exception, will allow tracing the queries generated, etc, so debugging isnt rocket science either.

As for validation, using an ORM will also usually allow much easier time developing validation strategies, on top of any built in validation.

Writing your own framework can be laborious, and often things get missed.

EDIT: I wanted to make one more point. If your company adopts an ORM strategy, that further enhances its value, as you will develop guidelines and practices for using and implementing and everyone will further enhance their knowledge of the framework chosen, mitigating one of the issues you brought up. Also, you will learn what works and what doesnt when situations arise, and in the end it will save lots of time and effort.

Share Improve this answer Follow

edited Oct 11, 2008 at 15:21

answered Oct 11, 2008 at 15:16

mattlant
15.4k ● 4 ● 36 ● 44

1

Every ORM, even a "good one", comes saddled with a certain number of assumptions that are related to the underlying mechanics that the software uses to pass data back and forth between your application layer and your data store.

I have found that for moderately sophisticated application, that working around those assumptions usually takes me more time than simply writing a more straightfoward solution such as: query the data, and manually instantiate new entities.

In particular, you are likely to run into hitches as soon as you employ multi-column keys or other moderately-complex relationships that fall just outside the scope of the handy examples that your ORM provided you when you downloaded the code.

I concede that for certain types of applications, particularly those that have a very large number of database tables, or dynamically-generated database tables, that the auto-magic process of an ORM can be useful.

Otherwise, to hell with ORMs. I now consider them to basically be a fad.

Share Improve this answer Follow

answered Feb 18, 2011 at 21:52

Mason Houtz
109 ● 4