What are the advantages of using an ORM? [closed]

Asked 15 years, 11 months ago Modified 10 years, 3 months ago

Viewed 42k times

65

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 11 years ago.

Improve this question

As a web developer looking to move from hand-coded PHP sites to framework-based sites, I have seen a lot of discussion about the advantages of one ORM over another. It seems to be useful for projects of a *certain (?)* size, and even more important for enterprise-level applications.

What does it give me as a developer? How will my code differ from the individual SELECT statements that I use now? How will it help with DB access and security? How does it find out about the DB schema and user credentials?

Edit: *@duffymo pointed out what should have been obvious to me: ORM is only useful for OOP code. My code is not OO, so I haven't run into the problems that ORM solves.*

orm

Share Follow

edited Dec 29, 2008 at 22:00

asked Dec 29, 2008 at 17:14

flamingLogos

6,011 ● 4 ● 39 ● 45

1 I suggest you experiment with ORM the way it's supposed to be done--look at Ruby on Rails and Active Record--just build a demo and change a few things. It's pretty slick. That said, the implementations I've seen rely heavily on setters and getters and most programmers, being naturally lazy, leave those setters and getters external which make for not-so-good OO. – Bill K Sep 2, 2010 at 19:47

1 I've seen it done in Grails, and I've used Hibernate. I'll still stand by my answer. – duffymo Nov 21, 2010 at 13:29

12 Answers

Sorted by: Highest score (default)

I'd say that if you aren't dealing with objects there's little point in using an ORM.

102

If your relational tables/columns map 1:1 with objects/attributes, there's not much point in using an ORM.

If your objects don't have any 1:1, 1:m or m:n relationships with other objects, there's not much point in using an ORM.

If you have complex, hand-tuned SQL, there's not much point in using an ORM.

If you've decided that your database will have stored procedures as its interface, there's not much point in using an ORM.

If you have a complex legacy schema that can't be refactored, there's not much point in using an ORM.

So here's the converse:

If you have a solid object model, with relationships between objects that are 1:1, 1:m, and m:n, don't have stored procedures, and like the dynamic SQL that an ORM solution will give you, by all means use an ORM.

Decisions like these are always a choice. Choose, implement, measure, evaluate.

Share Follow

edited Nov 21, 2010 at 13:28

answered Dec 29, 2008 at 17:30

duffymo
308k ● 46 ● 374 ● 565

My code is not OO, so I haven't run up against the object/relational mapping issue yet. That's why I'm having trouble visualizing how ORM would improve my code. Thanks! – flamingLogos Dec 29, 2008 at 17:51

1:1 table mappings fall under the Active Record pattern which is a very popular way to use ORMs. Mapping objects to multiple tables is only one of the several features of an ORM. The answer I gave lists some of the other features you will get by using an ORM even with 1:1 table mappings. – Daniel Auger Dec 29, 2008 at 19:58

1 Agreed, but I wouldn't call Active Record ORM's finest hour. I think it fools people who haven't thought about a real object model into thinking they're doing it right. – duffymo Dec 29, 2008 at 20:08

1 So can it be infered that there is `no point` in using `ORM` technologies ? – Rachel Nov 21, 2010 at 1:43

@duffymo Did you really mean that ORM should not use for those kind of system which used storedprocedures already ? – Frank Myat Thu Jun 14, 2012 at 4:13

41

ORMs are being hyped for being the solution to Data Access problems. Personally, after having used them in an Enterprise Project, they are far from being the solution for Enterprise Application Development. Maybe they work in small projects. Here are the problems we have experienced with them specifically nHibernate:

1. Configuration: ORM technologies require configuration files to map table schemas into object structures. In large enterprise systems the configuration grows very quickly and becomes extremely difficult to create and manage. Maintaining the configuration also gets tedious and unmaintainable as business requirements and models constantly change and evolve in an agile environment.
2. Custom Queries: The ability to map custom queries that do not fit into any defined object is either not supported or not recommended by the framework providers. Developers are forced to find work-arounds by writing adhoc objects and queries, or writing custom code to get the data they need. They may have to use Stored Procedures on a regular basis for anything more complex than a simple Select.
3. Proprietery binding: These frameworks require the use of proprietary libraries and proprietary object query languages that are not standardized in the computer science industry. These proprietary libraries and query languages bind the application to the specific implementation of the provider with little or no flexibility to change if required and no interoperability to collaborate with each other.
4. Object Query Languages: New query languages called Object Query Languages are provided to perform queries on the object model. They automatically generate SQL queries against the

databse and the user is abstracted from the process. To Object Oriented developers this may seem like a benefit since they feel the problem of writing SQL is solved. The problem in practicality is that these query languages cannot support some of the intermediate to advanced SQL constructs required by most real world applications. They also prevent developers from tweaking the SQL queries if necessary.

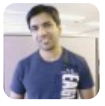
5. Performance: The ORM layers use reflection and introspection to instantiate and populate the objects with data from the database. These are costly operations in terms of processing and add to the performance degradation of the mapping operations. The Object Queries that are translated to produce unoptimized queries without the option of tuning them causing significant performance losses and overloading of the database management systems. Performance tuning the SQL is almost impossible since the frameworks provide little flexiblity over controlling the SQL that gets autogenerated.
6. Tight coupling: This approach creates a tight dependancy between model objects and database schemas. Developers don't want a one-to-one correlation between database fields and class fields. Changing the database schema has rippling affects in the object model and mapping configuration and vice versa.
7. Caches: This approach also requires the use of object caches and contexts that are necessary to maintian and track the state of the object and reduce

database roundtrips for the cached data. These caches if not maintained and synchrnonized in a multi-tiered implementation can have significant ramifications in terms of data-accuracy and concurrency. Often third party caches or external caches have to be plugged in to solve this problem, adding extensive burden to the data-access layer.

For more information on our analysis you can read: http://www.orasissoftware.com/driver.aspx?topic=whitepaper

Share Follow

answered Dec 29, 2008 at 17:33



Ahmad

1,055 ● 7 ● 14

16 In my opinion, this "analysis" raises more questions then it actually answers anything. Most of the statements are either flat-out wrong, at least regarding Hibernate (2, 4, 5, 7), too general (1, 6) or misleading (3, 4, 6). – javashlook May 5, 2009 at 20:54

1 Hibernate creates more problems than it solves. "Let's solve the SQL and relational-to-object problem!" GREAT! And to do that let's introduce a million of other complexities. No way. ORM like Hibernate is a huge TIME SINK, unless you never heard about databases, SQL, JDBC, etc. Not saying you should handle the boring JDBC boiler plate, but there are many solutions to address this much simpler problem: iBatis, jOOQ and MentaBean. If you like to be as close as possible to SQL you should check MentaBean: mentabean.soliveirajr.com – chrisapotek Feb 21, 2012 at 17:54

1 If you're that tightly coupled to SQL it's probable that your domain object design is poor. – Nathaniel Ford Jun 10, 2013 at 18:21

35

At a very high level: ORMs help to reduce the Object-Relational impedance mismatch. They allow you to store and retrieve full live objects from a relational database without doing a lot of parsing/serialization yourself.

> What does it give me as a developer?

For starters it helps you stay DRY. Either you schema or you model classes are authoritative and the other is automatically generated which reduces the number of bugs and amount of boiler plate code.

It helps with marshaling. ORMs generally handle marshaling the values of individual columns into the appropriate types so that you don't have to parse/serialize them yourself. Furthermore, it allows you to retrieve fully formed object from the DB rather than simply row objects that you have to wrap your self.

> How will my code differ from the individual SELECT statements that I use now?

Since your queries will return objects rather then just rows, you will be able to access related objects using attribute access rather than creating a new query. You are

generally able to write SQL directly when you need to, but for most operations (CRUD) the ORM will make the code for interacting with persistent objects simpler.

> How will it help with DB access and security?

Generally speaking, ORMs have their own API for building queries (eg. attribute access) and so are less vulnerable to SQL injection attacks; however, they often allow you to inject your own SQL into the generated queries so that you can do strange things if you need to. Such injected SQL you are responsible for sanitizing yourself, but, if you stay away from using such features then the ORM should take care of sanitizing user data automatically.

> How does it find out about the DB schema and user credentials?

Many ORMs come with tools that will inspect a schema and build up a set of model classes that allow you to interact with the objects in the database. [Database] user credentials are generally stored in a settings file.

Share Follow

answered Dec 29, 2008 at 17:34

Aaron Maenpaa
123k ● 11 ● 97 ● 108

15

If you write your data access layer by hand, you are essentially writing your own feature poor ORM.

Oren Eini has a nice blog which sums up what essential features you may need in your DAL/ORM and why it writing your own becomes a bad idea after time: http://ayende.com/Blog/archive/2006/05/12/25ReasonsNotToWriteYourOwnObjectRelationalMapper.aspx

EDIT: The OP has commented in other answers that his code base isn't very object oriented. Dealing with object mapping is only one facet of ORMs. The Active Record pattern is a good example of how ORMs are still useful in scenarios where objects map 1:1 to tables.

Share Follow

edited Sep 4, 2014 at 14:30

answered Dec 29, 2008 at 17:34

Daniel Auger

12.6k ● 5 ● 53 ● 73

...just as I was about to submit the same link – JC. Dec 29, 2008 at 17:35

9

Top Benefits:

1. Database Abstraction
2. API-centric design mentality

3. High Level == Less to worry about at the fundamental level (its been thought of for you)

I have to say, working with an ORM is really the evolution of database-driven applications. You worry less about the boilerplate SQL you always write, and more on how the interfaces can work together to make a very straightforward system.

I love not having to worry about INNER JOIN and SELECT COUNT(*). I just work in my high level abstraction, and I've taken care of database abstraction at the same time.

Having said that, I never have really run into an issue where I needed to run the same code on more than one database system at a time realistically. However, that's not to say that case doesn't exist, its a very real problem for some developers.

Share Follow

answered Dec 29, 2008 at 17:24

Dave

1,569 ● 10 ● 19

9

I can't speak for other ORM's, just Hibernate (for Java).

Hibernate gives me the following:

- Automatically updates schema for tables on production system at run-time. Sometimes you still have to update some things manually yourself.

- Automatically creates foreign keys which keeps you from writing bad code that is creating orphaned data.
- Implements connection pooling. Multiple connection pooling providers are available.
- Caches data for faster access. Multiple caching providers are available. This also allows you to cluster together many servers to help you scale.
- Makes database access more transparent so that you can easily port your application to another database.
- Make queries easier to write. The following query that would normally require you to write 'join' three times can be written like this:
    - "from Invoice i where i.customer.address.city = ?" this retrieves all invoices with a specific city
    - a list of Invoice objects are returned. I can then call invoice.getCustomer().getCompanyName(); if the data is not already in the cache the database is queried automatically in the background

You can reverse-engineer a database to create the hibernate schema (haven't tried this myself) or you can create the schema from scratch.

There is of course a learning curve as with any new technology but I think it's well worth it.

When needed you can still drop down to the lower SQL level to write an optimized query.

Share Follow

edited Dec 29, 2008 at 20:53

answered Dec 29, 2008 at 20:29

Sarel Botha

12.7k ● 7 ● 57 ● 61

7

Most databases used are relational databases which does not directly translate to objects. What an Object-Relational Mapper does is take the data, create a shell around it with utility functions for updating, removing, inserting, and other operations that can be performed. So instead of thinking of it as an array of rows, you now have a list of objets that you can manipulate as you would any other and simply call obj.Save() when you're done.

I suggest you take a look at some of the ORM's that are in use, a favourite of mine is the ORM used in the python framework, django. The idea is that you write a definition of how your data looks in the database and the ORM takes care of validation, checks and any mechanics that need to run before the data is inserted.

Share Follow

answered Dec 29, 2008 at 17:22

Christian P.

4,874 ● 7 ● 55 ● 70

> What does it give me as a developer?

Saves you time, since you don't have to code the db access portion.

> How will my code differ from the individual SELECT statements that I use now?

You will use either attributes or xml files to define the class mapping to the database tables.

> How will it help with DB access and security?

Most frameworks try to adhere to db best practices where applicable, such as parametrized SQL and such. Because the implementation detail is coded in the framework, you don't have to worry about it. For this reason, however, it's also important to understand the framework you're using, and be aware of any design flaws or bugs that may open unexpected holes.

> How does it find out about the DB schema and user credentials?

You provide the connection string as always. The framework providers (e.g. SQL, Oracle, MySQL specific classes) provide the implementation that queries the db

schema, processes the class mappings, and renders / executes the db access code as necessary.

Share Follow

answered Dec 29, 2008 at 17:24

Chris

28k ● 26 ● 130 ● 227

5

Personally I've not had a great experience with using ORM technology to date. I'm currently working for a company that uses nHibernate and I really can't get on with it. Give me a stored proc and DAL any day! More code sure ... but also more control and code that's easier to debug - from my experience using an early version of nHibernate it has to be added.

Share Follow

answered Dec 29, 2008 at 17:25

Peanut

19.4k ● 20 ● 74 ● 78

5

Using an ORM will remove dependencies from your code on a particular SQL dialect. Instead of directly interacting with the database you'll be interacting with an abstraction layer that provides insulation between your code and the database implementation. Additionally, ORMs typically provide protection from SQL injection by constructing parameterized queries. Granted you could do this yourself, but it's nice to have the framework guarantee.

ORMs work in one of two ways: some discover the schema from an existing database -- the LINQToSQL

designer does this --, others require you to map your class onto a table. In both cases, once the schema has been mapped, the ORM may be able to create (recreate) your database structure for you. DB permissions probably still need to be applied by hand or via custom SQL.

Typically, the credentials supplied programatically via the API or using a configuration file -- or both, defaults coming from a configuration file, but able to be override in code.

Share Follow

answered Dec 29, 2008 at 17:30

tvanfosson

532k ● 102 ● 699 ● 798

1

While I agree with the accepted answer almost completely, I think it can be amended with lightweight alternatives in mind.

- If you have complex, hand-tuned SQL
- If your objects don't have any 1:1, 1:m or m:n relationships with other objects
- If you have a complex legacy schema that can't be refactored

> ...then you might benefit from a lightweight ORM where SQL is is not obscured or abstracted to the point where it is easier to write your own database integration.

These are a few of the many reasons why the developer team at my company decided that we needed to make a more flexible abstraction to reside on top of the JDBC.

There are many open source alternatives around that accomplish similar things, and jORM is our proposed solution.

I would recommend to evaluate a few of the strongest candidates before choosing a lightweight ORM. They are slightly different in their approach to abstract databases, but might look similar from a top down view.

- jORM
- ActiveJDBC
- ORMLite

Share Follow

answered May 7, 2013 at 9:05

Martin

2,532 ● 1 ● 22 ● 22

0

my concern with ORM frameworks is probably the very thing that makes it attractive to lots of developers.

nameley that it obviates the need to 'care' about what's going on at the DB level. Most of the problems that we see during the day to day running of our apps are related to database problems. I worry slightly about a world that is 100% ORM that people won't know about what queries

are hitting the database, or if they do, they are unsure about how to change them or optimize them.

{I realize this may be a contraversial answer :) }

Share Follow

answered Jul 18, 2009 at 8:54

phatmanace

5,021 ● 3 ● 26 ● 29

Do you know how the assembly code looks your program is generating? I am not a fan of ORM either, but I think it's not a very strong argument. – The Fool Dec 17, 2021 at 23:35