Doesn't Linq to SQL miss the point? Aren't ORM-mappers (SubSonic, etc.) sub-optimal solutions?

Asked 15 years, 11 months ago Modified 13 years, 7 months ago Viewed 6k times

68

I'd like the community's take on some thoughts I've had about Linq to Sql and other ORM mappers.

I like Linq to Sql and the idea of expressing data access logic (or CRUD operations in general) in your native development tongue rather than having to deal with the "impedance mismatch" between C# and SQL. For example, to return an ObjectDataSource-compatible list of Event instances for a business layer, we use:

```
return db.Events.Select(c => new EventData() { EventID = c.EventID, Title =
c.Title })
```

If I were to implement this using old SQL-to-C# constructs, I'd have to create a Command class, add the EventID parameter (using a string to describe the "@EventID" argument), add the SQL query string to the Command class, execute the command, and then use (cast-type)nwReader["FieldName"] to pull each returned field value and assign it to a member of a newly created instance of my EventData class (yuck).

So, *that* is why people like Linq/SubSonic/etc. and I agree.

However, in the bigger picture I see a number of things that are wrong. My sense is that Microsoft also sees something wrong and that is why they are killing Linq to SQL and trying to move people to Linq to Entities. Only, I think that Microsoft is *doubling-down on a bad bet.*

So, what is wrong?

The problem is that there are architecture astronauts, especially at Microsoft, who look at Linq to Sql and realize that it is not a true data management tool: there are still many things you cannot do easily of comfortably in C# and they aim to *fix it.* You see this manifested in the ambitions behind Linq to Entities, blog posts about the revolutionary nature of Linq and even the LinqPad challenge.

And the problem with *that* is that it assumes that SQL is the problem. That is, in order to reduce a mild discomfort (impedance mismatch between SQL and C#), Microsoft has proposed the equivalent of a space suit (full isolation) when a band-aid (Linq to SQL or something similar) would do just fine.

As far as I can see, developers are quite smart enough to master the relational model and then apply it intelligently in their development efforts. In fact, I would go one further and say that Linq to SQL, SubSonic, etc. are *already too complex:* the learning curve isn't that much different from mastering SQL itself. Since, for the foreseeable future, developers *must* master SQL and the relational model, we're now faced with learning *two* query / CRUD languages. Worse yet, Linq is often difficult to test (you don't have a query window), removes us one layer from the real work we are doing (it generates SQL), and has very clumsy support (at best) for SQL constructs like Date handling (e.g. DateDiff), "Having" and even "Group By".

What is the alternative? Personally, I don't need a different model for data access like Linq to Entities. I'd prefer to simply pop up a window in Visual Studio, enter and validate my SQL, and then press a button to generate or supplement a C# class to encapsulate the call. Since you already know SQL, wouldn't you prefer to just enter something like this:

```
Select EventID, Title From Events Where Location=@Location
```

and end up with an EventData class that A) contains the EventID and Title fields as properties and B) has a factory method that takes a 'Location' string as an argument and that generates a List<EventData>? You'd have to think carefully about the object model (the above example obviously doesn't deal with that) but the fundamental approach of still using SQL while eliminating the impedance mismatch appeals to me a great deal.

The question is: am I wrong? Should Microsoft rewrite the SQL infrastructure so that you don't have to learn SQL / relational data management any more? *Can* they rewrite the SQL infrastructure in this way? Or do you think that a very thin layer on top of SQL to eliminate the pain of setting up parameters and accessing data fields is quite sufficient?

Update I wanted to promote two links to the top because I think that they capture important aspects of what I am after. First, CodeMonkey points out an article entitled "The Vietnam of Computer Science." It takes a while to get started but is a very interesting read. Second, AnSGri points to one of Joel Spolsky's more prominent pieces: The Law of Leaky Abstractions. It isn't exactly on topic but it is close and is a great read.

Update 2: I've given the "answer" to ocdecio although there are many great answers here and the choice of the "right" answer is purely subjective. In this case, his answer squared with what I think is truly the best practice given the current state of technology. This is an area that I fully expect to evolve, however, so things may well change. I'd like to thank everyone who contributed, I've upvoted everyone who I think gave a thoughtful answer.

sql linq-to-sql linq-to-entities

Share

Improve this question

Follow

edited Jun 16, 2010 at 16:46

Tom H
47.4k ● 15 ● 89 ● 131

asked Jan 19, 2009 at 19:23

Mark Brittingham
28.9k ● 12 ● 82 ● 111

I've been making the same connection to "Leaky Abstractions". All this complexity of trying to make a bunch of sizes and shapes of lists mutually accessible seems intent on defying this principle, at the cost of simplicity and coherence. LINQ-to-SQL is a prime specific example. – dkretz Jan 20, 2009 at 23:29

Very good question. I remember asking a MS guy at a TechDays where MS was heading, between Linq, Entities etc. He didn't seem to know either... – Benjol May 20, 2009 at 8:33

Benjol - thanks. I've been very happy with the response to this question. I worry that many of the MS technologies seem to be spawned by a need to be seen as doing something big rather than focusing on the more pedestrian needs of developers. Of course, when something big *works* - like .NET - then we all appreciate it. But LINQ doesn't do it for me. It think we'd be better served by a simpler and more straightforward tool. – Mark Brittingham May 20, 2009 at 12:57

2 Mark, from the answer you've chosen and your comments, its clear that this is not a question, its an opinion piece. Stick it on your blog. – mcintyre321 Jun 4, 2009 at 11:49

2 mcintyre - what are you talking about? This has been an enormously popular question that has spawned a lot of debate and discussion (as Benjol says: "Very good question"). From what I can tell, quite a few people have learned something from it even if they disagree with my chosen "answer". My view of SO is that it is a place to go to explore the entire field of software development and I've done a LOT of work to help it grow in that direction. The only real drawback to SO is that there are an inordinate number of people who think it appropriate to be rude when they disagree with someone. – Mark Brittingham Jun 5, 2009 at 12:59

24 Answers

Sorted by: Highest score (default)

45

Let me preface this by saying that I am a dyed-in-the-wool database guy.

As a gross over-generalization: Developers don't know SQL. Developers don't really *want* to know SQL. They can write it, they can design tables, but it makes them feel icky. They tend to do stupid things when the necessary query is more than a simple join. Not because the developers are stupid -- because they can't be bothered. They *like* living in a world where they only have to deal with one concept space; moving from objects to tables and back is a context switch the price for which they don't like paying.

This doesn't mean they are bad, or wrong; it means there is an opportunity for improvement. If your customers (in this case, developers using your framework) don't like SQL and tables -- give them a layer of abstraction that lets them get away without dealing with the underlying mess.

It's the same logic that makes garbage collection / automated memory management a big hit. Yes, developers can deal with it; yes, they can write code that is better optimized without it; but not having to deal with it makes them happier and more productive.

Share
Improve this answer
Follow

edited May 1, 2009 at 12:22

answered Jan 19, 2009 at 19:37
SquareCog
19.6k ● 8 ● 51 ● 63

At least happier, which is after all the first priority. – dkretz Jan 19, 2009 at 20:24

7 I couldn't disagree more. SQL is easy. There's only a handful of keywords you need to know. Well-written, performant SQL can speed up the app measurably. Poorly written SQL can bring the RDBMS and the website to their knees. If you don't want to write SQL, just say so. – DOK Jan 19, 2009 at 21:39

1 You are preaching to the choir, DOK. I'm just saying there's music outside the church, too :-) – SquareCog Jan 20, 2009 at 9:40

13 That is really sad: data handling is central to good programming. Not mastering SQL is like deciding to be a guitarist but disliking chords. – Mark Brittingham Jan 22, 2009 at 13:28

2 This is what I was guessing. I'm a developer and I love SQL. But I think that is not common. – Jeff Davis Jul 14, 2009 at 16:17

36

I think the popularity of ORMs has been spawned by developers developing data layers and writing the same CRUD code over and over again application after application. ORMs are just another tool/technology that lets developers spend less time writing the same SQL statements over and over and concentrate on the logic of the application instead (hopefully).

Share Improve this answer Follow

answered Jan 19, 2009 at 19:28
Jim Anderson
3,622 ● 2 ● 26 ● 21

1 Jim, I think you are absolutely right. The question is whether the current approach is overkill for dealing with this pain. I didn't say this but I've begun developing a code generator to create classes around SQL queries for my own use. Linq/SubSonic just don't work for me. – Mark Brittingham Jan 19, 2009 at 19:58

Definately true. I've completely stoped working on side projects when I got to the data layer and realized "oh its time to write 200 SQL queries over and over for all my objects". An ORM

tool would speed that up dramatically. Unfortunately most ORM tools suck :(
– CodingWithSpike Jan 19, 2009 at 20:23

I think it's a personal choice. You can always write your own data access layer like we did before the popularity of ORMs and have full control, but the ORMs usually provide the framework for an acceptable DAL (even with their faults) to save a lot of time & work.
– Jim Anderson Jan 19, 2009 at 20:45

1 "save a lot of time & work" is the operative doubtful assertion. It's not my subjective experience, but I'll take your word for it that it's yours. I don't consider DALs particularly difficult or tedious, unless you are wasting time cranking out CRUD templates (well-named, aren't they?) – dkretz Jan 20, 2009 at 6:17

@le dorifier: Yea, I guess your mileage may vary. But consideringing the time that has gone into creating products like Hibernate and the Entity Framework, I would hope they add value and reduce required data "plumbing" code. – Jim Anderson Jan 20, 2009 at 12:21

18

For at least 6 years I have been using my own ORM that is based on a very simple concept: projection. Each table is projected into a class, and SQL is generated on the fly based on the class definition. It still requires me to know SQL but it takes care of the 90% simple CRUD, and I never had to manage connections, etc - and it works for the major DB vendors.

I'm happy with what I have and didn't find anything worth dropping it for.

Share Improve this answer Follow

answered Jan 19, 2009 at 19:35
Otávio Décio
74.2k ● 18 ● 165 ● 228

4 I've been doing the same. For anything more complex that single row select/update/deletes, you probably want to be writing custom SQL anyway, so I don't see the need for something like Linq. – Kibbee Jan 19, 2009 at 19:40

@Kibbee - exactly. That's why I have an "out" in my DAL to call SP's or custom SQL and put the results in DTO's, and I am also able to return good'ol DataSets if I feel like. – Otávio Décio Jan 19, 2009 at 19:44

1 It should be self-evident that LINQ is another abstraction layer; but it's one too many. It solves no known abstraction problem, if it did, there would be one that went away. But there's no candidate for that here. – dkretz Jan 19, 2009 at 19:44

2 I didn't try to solve the abstraction problem - I just wanted to make it easier to turn SQL queries into objects and objects back to the database, simple as that. It cut my development time, it increased my overall code quality and that's just what I was looking for. – Otávio Décio Jan 19, 2009 at 19:58

2 @Jeff - you can find it a databroker.codeplex.com , I put it there to have it on a centralized place. Please feel free to check it out and let me know (through the contact page) if you have questions/suggestions/criticisms, all welcome. – Otávio Décio Jul 14, 2009 at 16:47

12

IMHO, OR/M is not only about 'abstracting the SQL away' or hiding the SQL, or enabling multi-DBMS support.

It enables you to put more focus on your problem domain, since you have to spent less time writing the boring CRUD SQL queries. On the other hand, if you are using a good OR/M, this OR/M should enable you to write SQL queries if this seems to be necessary.

An OR/M can be a powerful tool if you use it properly; it can take care of lazy loading, polymorphic queries / associatons ...
Don't get me wrong; there's nothing wrong with plain SQL, but, if you have to take care yourself of translating your (well thought and normalized) relational model to an expressive OO/domain model, then I think you're spending way to much time doing plumbing.

Using an OR/M also does not mean that you -as a developer- should have no knowledge of SQL. The contrary is true imho.
Knowing SQL and knowing how to write an efficient SQL query, will -imho- enable you to use an OR/M properly.

I must also admit that I'm writing this with NHibernate in mind. This is the OR/M that I'm using atm, and I haven't used Linq to SQL or Linq to entities (yet).

Share Improve this answer Follow

answered Jan 19, 2009 at 22:23
Frederik Gheysels
56.9k ● 11 ● 102 ● 155

3 I'm in the same boat. Writing crud sql is easy. It's about not having to write and rewrite the plumbing/glue that manages the mapping of the 2 worlds. That part is difficult to do consistently especially in team with many devs. An ORM mapper lets everyone forget about it and do everything uniformly. – Daniel Auger Jan 19, 2009 at 23:36

10

Linq's design and the linq to entities framework certainly has uses as an orm tool, but the big idea is that it will be used as a common api to query ANY data source, not just RDBMS's.

I remember reading that linq, while obviously designed with SQL in mind, is meant to be a query language for any data store. You can write linq queries for SQL, but you can also theoretically write linq queries that target ldap, filesystem's, exchange, web services, ad infinitum. You can't use SQL for those programs.

You also need to learn a different API for almost every data store. Linq gives everyone a common target to create a data access API.

Whether this vision works or not remains to be seen, but that is the idea. I think as we want systems to inter-operate more and more we may find some very nice uses for l2e.

I'll add some references if I can find them again.

http://laribee.com/blog/2007/03/17/linq-to-entities-vs-nhibernate/

Share
Improve this answer
Follow

edited Jan 19, 2009 at 21:27

answered Jan 19, 2009 at 21:18
Trevor Abell
752 ● 2 ● 9 ● 15

9

You should stop worrying and learn to love the ORM. Abstractions such as these will help us focus our skills and make advances in the field.

There is still plenty of room to take advantage of the functional skills you have acquired and apply them in the application layer. This is in fact one of the strengths of LINQ to SQL over other ORM's.

I can only agree with many of the other comments. The time you save, you can focus on refining your domain model and make a better application. And, once you've pinpointed the bottleneck, use to create optimized SQL.

What might not be immediately obvious is that the ORM comes with a number of features that are really nice. The identity map that helps avoid loading items over and over, lazy loading helps you express the domain with less plumbing and the unit of work helps you track changes and optimize database writes.

Share Improve this answer Follow

answered Jan 19, 2009 at 19:54
Cristian Libardo
9,248 ● 3 ● 37 ● 41

1 Then we need to invent and implement a data store that matches the Object paradigm, instead of force-fitting SQL. You're just moving the impedance mismatch to the other side of the interface. – dkretz Jan 19, 2009 at 20:00

3 "stop worrying" is a terrible way to live in this industry. "question everything" is much better. – CodingWithSpike Jan 19, 2009 at 20:17

Also, abstraction isn't always a good thing. It leads to use of the common bits between tools, and no specialization. ex: abstract to "DB" and you have to remove Oracle continuous notification, and even stored procs, because not all DBs have them. – CodingWithSpike Jan 19, 2009 at 20:18

@ledorfier: I can agree in teory, in practice ORM's work very well @rally: that was a reference to a popular movie =) @rally again: Inverse that and you get that abstractions most of the time are a good thing, but sure, there will always be the "other" 5% – Cristian Libardo Jan 19, 2009 at 20:47

@le dorfier: I do not agree. :) The relation model is a good model to efficiently store data. The relational model also enables you to generate reports, etc... easily. (So, don't use a OR/M for this) But, OO is a good way of representing a domain model, so both models have to work togheter. – Frederik Gheysels Jan 19, 2009 at 22:27

9

I agree 100%. A lot of this is a result of the fact that procedural coding skills and SQL coding skills are very different; and this fact is not widely acknowledged. So, until that realization hits, programmers search for ways to make their skillset transferable from one domain to the other.

It doesn't work.

You simply must learn how to phase-shift: learn how to think about your code differently depending on which domain you are addressing.

Has anyone else noticed how much more complex and verbose a query becomes when it's mapped from SQL to LINQ? Like, in all the online examples?

Share
Improve this answer
Follow

edited Jan 20, 2009 at 6:20

answered Jan 19, 2009 at 19:37
dkretz
37.6k ● 13 ● 83 ● 140

1 I agree with everything you say, except the "simply must." Face it -- you and I are C hackers in a Java world. Our stuff may run more efficiently, but the quicker we accept the status quo, the lower our blood pressure will be :-). – SquareCog Jan 19, 2009 at 20:03

4 Quite the reverse - most queries become *simpler* in LINQ. The problem is that many people fail to properly learn LINQ and so they their *transliterate* their SQL queries into LINQ - and of course then you can only lose and never gain. – Joe Albahari May 5, 2009 at 9:53

I'm glad it becomes "simpler" for you. To me it's taking something simple and translating it into something different, unarguably adding complexity. And nothing always maps perfectly. – dkretz May 5, 2009 at 17:29

@Gurdas Nijor Yes. Declarative == Awesome. :) – Jeff Davis Jun 23, 2010 at 18:18

7

As Dmitriy pointed out, developers don't know SQL. More precisely, the majority *know* SQL, but don't *understand* it and definitely don't like, so they tend to search for the magic bullet, creating the demand for things like Linq to make the illusion (hm, abstraction) that they don't use anything different than their beloved classes.

That's very bad, as the law of leaky abstractions always holds true.

Some ORM solutions are definite good (e.g. JPA/Hibernate), not because using them you don't have to worry about SQL. In fact, to use JPA effectively you need very deep understanding of the DB in general, querying abilities in particular. The good point is that they *make the machine do the boring work*, to the point where it autogenerates entire database from scratch.

Linq to SQL, as I think, doesn't solve real problem. It's kind of other presentation, nothing more. It might be good, though it overcomplicates the already complex language. On the other hand, Linq to Objects is very interesting concept, because it's kind of sql-querying the collections.

Share

Improve this answer

Follow

edited May 23, 2017 at 12:00

Community Bot

1 ● 1

answered Jan 20, 2009 at 5:14

ansgri

2,156 ● 6 ● 25 ● 38

6

Historical perspective.

When Codd et. al. originally were working out the theory and implementation of relational databases, one entirely separate issue was "How do we query these things"? A number of strategies and syntaxes were proposed, with various strengths and weaknesses. One of those candidates was SQL (in its earliest form, obviously.) Another was QBE (Query By Example). Another was called "quel", I believe; and there were several others. SQL certainly didn't become dominant because it was acclaimed as superior to all others. Unfornately, though, the others have pretty much disappeared, to the poverty of us all (because they could be used simultaneously on the same data.)

If Microsoft has a good story that they are reviving one of these other languages, or have invented a worthy addition, then I think we would be well-advised to listen up. But so far all I've seen is yet another "One Ring To Rule Them All".

There's a hell of a lot of thought and rigor behind SQL, and a lot of time-proven durability. Microsoft has a certain history of believing that their admittedly top-grade development organization can out-think the rest of us, including our collective institutional memories. It doesn't seem often to work that way. As long as we're bonded to relational data stores, we should think twice about superset abstraction paradigms that move us away from the bare metal with promises of equal or better performance.

Share

Improve this answer

Follow

edited Jan 19, 2009 at 21:36

DOK

32.8k ● 8 ● 63 ● 93

answered Jan 19, 2009 at 19:51

dkretz

37.6k ● 13 ● 83 ● 140

I actually used QBE as it was implemented in the "Paradox" data management tool in the early 1990s so I remember! Your analogy to "One Ring To Rule Them All" was priceless!
– Mark Brittingham Jan 19, 2009 at 20:00

I think the analogy is crap. LINQ is an attempt to address the impedance mismatch, not another relational query language. – reinierpost Apr 12, 2010 at 13:01

QUEL was the original query language of Berkeley's Ingres, and not too different than SQL, which quickly supplanted it QBE was a graphical/textual GUI that sat on top of SQL: it was pretty neat for simpler queries but not so great for anything really complex. I don't think either posed much of a challenge to SQL. SQL was an extremely powerful, expressive language.
– Jim Ferrans Feb 5, 2011 at 1:28

5

Being the author of an ORM project myself, I have to admit that my response to a question like this is apt to be a bit biased. I've already developed some of my own thoughts about the answer prior to reading the question, so I'm already somewhat anchored.

I will say that my reason for developing an ORM wasn't because of the "impedance mismatch" between SQL and imperative programming, but rather solely for the purpose of becoming database-platform agnostic. The former issue of having to write more code to manage persistence is a small hurdle that's easily resolved if you only work with one database vendor. Becoming database platform agnostic is a much more challenging problem and imo has a much larger impact on the profitability of your business assuming that like me you plan to sell software to other people (and are not just using it in house).

When I started working on my ORM tools several years ago, the concept was impractical in my preferred language, most people I spoke to didn't understand why I was working on it and some well respected voices in the community had as much as written articles in trade magazines stating that what I had already done was not only impossible but also undesirable. Some of the same reasons were given - it's too complex, it's limiting and it adds overhead. Today the same community has at least three popular database abstraction tools (although there is some debate about the definition of the term ORM). The reason why I mention this is because when I started working on these tools, the original objections carried a lot more weight than they do now. The underlying technology both hardware and software has changed in the intervening years to make these tools much more practical in the long run. My tendency is to try and take a long-view of software and work on solutions that are maybe not quite practical yet but that will become practical soon. So given that I wouldn't count out LINQ to Entities as a good solution for certain problems.

I also tend to prefer more options rather than less. So while I may support the idea behind developing LINQ to Entities, I'm less apt to support killing off LINQ to SQL merelyu because LINQ to Entities has become available. Objects are great for doing what objects do, there's no question about that... In my (again biased) opinion, a problem occurs in that people see any given tool or software paradigm as a "magic bullet" and want to insist that everything must be that way. It's well known that a relational database is very good at handling certain other tasks, reporting being a good example. So in my opinion, it's kind of shooting yourself in the foot to insist that everything must be entities, because then you're forcing yourself to use an inefficient tool for the job. So with regard to reporting in particular, getting rid of LINQ to SQL and using only LINQ to Entities at least on the surface sounds to me like the abstraction inversion anti pattern.

So I guess the synopsis for my answer is this: use a hammer if your problem is a nail - use a screwdriver if your problem is a screw.

Share Improve this answer Follow

answered Jan 19, 2009 at 21:07

Isaac Dealey
375 ● 1 ● 6

5

Dmitry's statement that Developer's don't like SQL may have lot of truth but the solution isn't only ORM. The solution is to hire as part of the development team a Development DBA. In my company my .net development team has an excellent Oracle DBA who does absolutely no production dba work. His role in our team is data modelling, physical db design, creating stored procs, data analysis etc. He is the reason our db side is so clean and performing. All our DB access is via stored procs.

What is a development DBA ? http://www.simple-talk.com/sql/database-administration/what-use-is-a-development-dba/

Share Improve this answer Follow

answered Jan 25, 2009 at 2:28

kanad

4

I do both database and distributed application programming (web and compiled) and feel like taking the time to develop stored-procedure based data access layers is time well spent. Personally, I prefer to do data modeling and identify the needed procs early in the development process... seems to help uncover design/interface logic/structure issues.

I'm not a big fan of inline database programming (whether the sql code is hand or machine generated). I believe that the database is the foundation of one's application and that taking the time it to hand-code stored procs is worthwhile.

That said, I am intrigued by the OODBMS concept and hope that someday I'll get to work on some in a production environment.

Share Improve this answer Follow

answered Jan 19, 2009 at 20:55

codemonkey
2,665 ● 2 ● 25 ● 34

4

If you want a database to perform as it scales, it has to be designed and normalized according to database relational model best practices.

If you let the Object Model and the ORM dictate your data model, you will just end up with a poor data model which is not normalized and/or contains artifacts from the object model.

1 table = 1 class is a bad idea.

To start with you never, ever, have classes which represent many-to-many or many-to-one link tables. These correspond to child collections in the object model - the links themselves are not objects.

If you treat your database as tables to simply hold your objects in rows (one common approach) and give the application direct access to tables, you are giving up all the benefits of defining an interface layer of database services that the database can use to protect its perimeter.

ORMs have their place, but if you use them to simply persist your objects as designed in your object model, your database will not be a relational database, and it will not be able to be used as one for the purposes of ETL, reporting, refactoring, etc.

Share Improve this answer Follow

answered Jan 25, 2009 at 1:57

Cade Roux
89.6k ● 40 ● 184 ● 266

3

I think the logic behind these things is that the overhead of building and running an SQL statement in the framework layer is insignificant compared to overhead in other parts of the system (e.g., a HTTP request round trip is orders of magnitude longer).

The advantages - fast development, queries that fit in with the language rather than being escaped strings, etc, often outweigh the disadvantages. If performance is an issue, then you can optimise later.

I don't think that "not needing to know SQL" is a factor. Any decent developer will need to know SQL at some point in their development. The idea of a database abstraction layer is to remove the effort of generating boilerplate code for database queries.

Indeed in our system in Java, I created a code generator utility that took annotated classes and generated a full CRUD API, and handling all those quirky things you pick up over time. Far easier to create your model and annotate it than to slog through writing a DAO. Scale such a tool up and you end up with LINQ or Hibernate or myriad other ORM DB solutions.

Share Improve this answer Follow

answered Jan 19, 2009 at 19:36

JeeBee
17.5k ● 5 ● 52 ● 60

However, for a single we request, there could easily be 10 or even 100 queries. The fact that the http request is an order of magnitude longer is circumvented by the fact that you are going to do an order of magnitude, or more, query operations. – Kibbee Jan 19, 2009 at 20:29

Yes, that's where the logic falls down - serious web apps that do such a lot of SQL in the background really need to be optimised to remove abstraction slowdown. But for a standard simple intranet form based application? Initially it can use the time saving abstraction, until it bloats out ;) – JeeBee Jan 20, 2009 at 10:26

3

There was [another question here]() that asked about ORMs in general. Check that out for some more discussion of whether or not the impedance mismatch is as big a deal as all that.

Share

Improve this answer

Follow

edited May 23, 2017 at 11:53

Community Bot
1 ● 1

answered Jan 19, 2009 at 19:37

Hamish Smith
8,181 ● 1 ● 38 ● 49

3

I think the real solution that they needed was something more like SQL literal. VB.Net 9.0 supports XML Literals, which allow you to write XML right in your code, and ensure that it's validated and meets the DTD. A similarly nice feature would be SQL literals that allow you to write inline SQL code in your .Net code, and have it validated by the IDE. There would need to be some sort of plugin architecture to verifying the information against the database, but that could be easily written for the popular database engines. This would provide what I think to be the real solution, to the problem they were trying to solve, without resorting to sub-optimal solutions.

Share Improve this answer Follow

answered Jan 19, 2009 at 19:37

Kibbee
66.1k ● 28 ● 144 ● 184

3

Codemonkey makes a very good point: stored procedures offer a level of abstraction that fulfills some of the promise of the more complex ORM models. This isn't intuitive at first glance but I can think of an example right from my own code. I have a "check-in" sproc that touches nearly a dozen tables (who does this ID belong to? do they have a membership? Are there any messages? Do they get points for this check-in? etc.).

Modeling this in C# - no matter how sophisticated the data model - would never be a good idea as it would necessitate many trips "over the wire" to check on data and update various tables. While it is true that you can handle sprocs in Linq or other ORMS, *all I need* is a wrapper class that allows me to call the sproc with standard C# parameters. In fact, this was such a pressing need for me that I wrote a code generator in T-SQL to create such wrappers.

Share
Improve this answer
Follow

edited Jan 20, 2009 at 0:36

answered Jan 20, 2009 at 0:29

Mark Brittingham
28.9k ● 12 ● 82 ● 111

Modeling this in C# is a good idea since it allows you to use inheritance, modularity and unit-testing. Wire trips are generally not a problem -- not so much of a problem as a maintenance of moderately complex stored procedure -- and you even can not have unit tests for it without setting up the DB. – Andrey Shchekin Jan 20, 2009 at 8:30

Yes, but ORMs or DALs usually provide a more genral (DB-agnostic) solution whereas sprocs (if your DB supports them) are DB specific. – Jim Anderson Jan 20, 2009 at 12:27

3 I always here this DB-agnostic rule. But how many people really switch databases when there application is fully developed? – Donny V. Jan 25, 2009 at 2:39

@Andrey: Stored procedure languages such as PL/SQL allows you to use inheritance, modularity and unit-testing, too. Check out download-uk.oracle.com/docs/cd/A91202_01/901_doc/appdev.901/... @Jim: If you use C#/.NET you

can only deploy to a single platform (Windows). If you write stored procedures in PL/SQL, you can deploy on any operating system that Oracle supports. In other words, PL/SQL is like Java, "Write once, run anywhere", except that it performs way better than Java!
– ObiWanKenobi Jul 24, 2009 at 7:45

3

I don't like any of the current solutions - but i prefer more choices over less choices ;-)

i used an OODBMS in a project long ago that was the closest to getting it right (unfortunatley the product had some heinous bugs and terrible tech support) - object persistence was largely *invisible*, handled behind the scenes (via preprocessor code-generator) and controlled by very simple Update, Delete, and Find methods and simple containers.

I'd love to see this (and perhaps some object-entity model does this well already) again, with OCL (Object Constraint Language) as the native multi-object query language

Share
Improve this answer
Follow

edited Apr 9, 2009 at 3:33

answered Jan 19, 2009 at 19:31
Steven A. Lowe
61.1k ● 19 ● 135 ● 204

2

I have to agree that the outburst of ORM tools largely stems from the annoyance of writing SQL, dealing with whatever DB driver you have to use, internally abstracting between DBs (Oracle vs SQL Server, vs whatever for code reusability), and transforming data types.

Ideal solution? definately not! However, I think this is an iteration in the process of better merging the application with the data store. After all, in most cases, having a DB without an accessing application written in whatever language is practically useless. (would you really ever just install oracle, and expect all employees to work in SQLPlus directly?)

As such, I think the data store and the application are just moving together in a way that the application developer can more easilly interact with the data store.

For .NET in particular, what I'd much rather see instead of Linq is a built in way to map a class definition to an underlying data source to an underlying data store, and have all the plumbing just work.

In other words I could just say "Employee.Name = 'rally25rs';" and the object's property would change, and underneath that it would persist to the data store itself. Just ditch SQL completely instead of going 1/2 way like Linq to SQL does now.

Of course, a solution like that brings up issues with performance, transaction handling, etc.

Maybe the entire concept of programming language and relational database need to be rethought and realigned? At their core, they are completely separate and disjointed entities. Maybe its time to just ditch the "relational sql aware database" and build the next thing, whee executing a line of code would directly operate on the database.

Share Improve this answer Follow

answered Jan 19, 2009 at 20:11
CodingWithSpike
43.7k ● 18 ● 105 ● 139

ODBC/JDBC (and their spawn) have done a satisfactory job of abstracting away dbms SQL implementation differences. But notice how seldom they seem to be used anymore. See all the posts featuring proprietary syntax issues. The answer to that has been at hand for years - increasingly ignored. – dkretz Jan 20, 2009 at 6:23

@le dorfier: good point. perhaps part of the issue is that the abstraction removes all specialization? Ex: if I don't use the Oracle driver, I can't use their Continuous Notification tools. Perhaps ODBC is the all-season tire (mediocre at all things) whereas specialized drivers are the racing tires? – CodingWithSpike Jan 20, 2009 at 14:37

1

there´s no problem with linq, but with those who use it and ignore what happens "behind the scenes"

I still prefer to get my hands dirty with SQL. At least i´ll know exatcly whats happening.

Share Improve this answer Follow

answered Jan 19, 2009 at 19:43
DonOctavioDelFlores
363 ● 1 ● 7

1

Ted Neward wrote a great essay on his take on this subject - that ORM's are the "Vietnam" of computer science...

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

Share Improve this answer Follow

answered Jan 19, 2009 at 22:04
fuzzbone
2,030 ● 14 ● 14

Fuzzbone - the article spends a bit too much time reviewing the Viet Nam war but it is right on target. That is an excellent reference...thanks! – Mark Brittingham Jan 20, 2009 at 22:52

1

Most people have missed an essential point: in most cases, you are **significantly more productive** when querying in LINQ than in SQL. I've written an article on why this is so.

When I set the LINQPad Challenge, I wasn't joking: I do nearly all of my ad-hoc querying in LINQ because most queries can be written more quickly and reliably in LINQ than in SQL. I've also designed and worked on large business applications using LINQ to SQL and seen a major gains in productivity. This is not "architecture astronaut" stuff - LINQ to SQL is a productive and practical technology that **drives this very site**.

The biggest hindrance with LINQ is failing to properly learn it. I've seen so many LINQ queries that are horrible **transliterations** of SQL queries to back this up. If you write LINQ queries using only your knowledge of SQL, the end result can only be the same - or worse - than SQL.

Share

Improve this answer

Follow

edited Jan 18, 2021 at 12:38
Community Bot
1 ● 1

answered May 5, 2009 at 10:24
Joe Albahari
30.9k ● 7 ● 84 ● 95

Albahari - thank you for commenting. I have no doubt that many LINQ queries are done poorly. Now, I'll say that it'll take some convincing but I will read your argument RE: the productivity of LINQ. Personally, I find SQL to be incredibly effective and productive! Again, thanks though...good comment. – Mark Brittingham May 5, 2009 at 14:07

Can you give a good example of a LINQ query you saw which was clearly a transliteration of SQL, and which you could have written more efficiently? Can you show something more convincing than just associations and parametrisation, which are rather minor? – Timwi Jun 15, 2009 at 20:07

1 Here's a simple example: social.msdn.microsoft.com/Forums/en-US/linqprojectgeneral/… – Joe Albahari Jul 24, 2009 at 9:49

1

I just discovered this question. I guess it's pretty much played out by now, but I'm going to throw in my two cents anyway...

I only want to write code for the things that aren't obvious.

CRUD code is obvious. I don't want to write it. Therefore, ORMs are a good idea.

This doesn't mean that ORMs don't have problems, but the problems are with execution, not intent. As ORMs mature, the problems will diminish, and the productivity gains already available for simple scenarios will eventually extend to complex scenarios as well.

LINQ is also a good idea. Others have mentioned a number of the advantages, but all I have to do is think about the first time I tried to do a pivot in LINQ where I didn't know the number of columns in advance. Or the first time I realized I didn't have to create a new `DataView` every time I wanted to sort or filter something. LINQ empowers me to do everything I want to do with data in C#, rather than having to figure out how divide up the work between SQL and C#.

So, yes, ORMs, LINQ, and other emerging technologies are suboptimal solutions, but they don't miss the point, and they won't be suboptimal forever.

Share

Improve this answer

Follow

edited Feb 2, 2010 at 7:39

answered Feb 2, 2010 at 7:02

devuxer

42.3k ● 47 ● 183 ● 298

+1 - Good comments - especially the observation that "I only want to write code for the things that aren't obvious." Keep in mind that I'm not arguing against DALs in the general sense (in fact I've built three ORM-like DALs over the years). I'm arguing that a good DAL doesn't need all of the architecture-astronaut stuff MS is putting into Linq to Entities: that automating the tedious aspects of SQL and reducing the impedance mismatch between SQL and C# data structures is all you need - it is the optimal solution. Every advantage you describe to L2S can be had for far less cost! – Mark Brittingham Feb 2, 2010 at 13:49

@Mark, I've just been happily using SqlMetal (the command-line LinqToSql generator) for a relatively straightforward SQL Server Express database project and just loving it. It just works. I haven't experienced any cost. As for LinqToEntities, I really can't comment yet as I've been avoiding it due to all the reported problems with "generation 1". Once I upgrade to Visual Studio 2010, I'll definitely give the new LinqToEntities a look. I always figured that it was just LinqToSql on steroids, but maybe the difference is more fundamental. – devuxer Feb 2, 2010 at 18:17

I'm definately keeping my eyes open on the matter so it is good to hear about your experiences RE: SqlMetal. First - you'll love VS 2010. I've been using the Beta for about 4 months and it is a tremendous upgrade. WRT LinqToEntites, it is quite a different animal than LinqToSql. You know, if MS had been content to stay with LinqToSql then I probably wouldn't have gotten my hackles up as I really do kind of like it. But L2Sql is being deprecated and L2E positioned as the path forward and I just think that is a mistake. If you try it and have a different take, I'd love to hear why. – Mark Brittingham Feb 2, 2010 at 18:42

Hmmm...Maybe I should have entitled this "Doesn't Linq to Entities miss the point?" – Mark Brittingham Feb 2, 2010 at 18:43

@Mark, from my perspective, if LinqToEntities won't just let me point to a database and auto-generate all the code needed to interact with that database, it will definitely feel like a step backward for me. I want a tool that makes my programming life easier and lets me focus on the stuff that is unique to my particular application. As for your title, perhaps, in retrospect, your question really was about L2E rather than L2S, but either way, it generated lots of interesting answers and discussion. – devuxer Feb 2, 2010 at 19:08

0

I wanted to write this as a reply to @SquareCog reply here, but it told me I had -1836 characters left. S.O. noob here so apologies if I've done this wrong.

In the 18th century gentleman of leisure used to study science. At that time science in its entirety was not such a large subject that a reasonably intelligent person couldn't understand it all. By which I mean a single learned fellow could understand the entirety of scientific thinking of the time.

As time has gone by hundreds of new fields of science have been discovered and each one researched to the point where these days very few people can even understand the entirety of a single complete field of science.

So it is with programming.

These days the programming language field is large enough and growing fast enough that it is as much as can be reasonably be expected of a developer to know the entirety of his own specialised languages(s). For a skilled coder to also be expected to understand the entirety of the database field too, including database design, the nuances of native SQL and how it operates on different databases and the administration of those databases too, is possibly asking a bit much.

I think some of the responders here are glossing over some of the complexities of developing a large performant enterprise level database, knowing a 'handful of SQL statements' most certainly does not cut it. Which is why most large software houses have dedicated teams of database developers. As Stroustrup says, 'Divide and conquer' is the only way to effectively deal with the complexity inherent in developing and managing large software projects.

Developers don't dislike working with SQL because they are lazy or because it makes them feel 'icky'. They dislike working with SQL because they are smart. They know better than anyone that only someone who specialises in SQL will deliver the highest quality database functionality and that for them to be put into the position of being 'jack of all trades' developers is a suboptimal development strategy.

Share
Improve this answer
Follow

edited May 23, 2017 at 12:25
Community Bot
1 ● 1

answered May 25, 2011 at 22:10
Neutrino
171 ● 3