Why are database features being ignored, and instead reinvented in the middle tier?

Asked 15 years, 3 months ago Modified 6 years, 2 months ago

Viewed 3k times

84

What are the top reasons (**apart from "database independence"**) that most IT projects today seem to ignore the wealth of features that exist in modern database engines such as Oracle 11g and SQL Server 2008?

Or, to borrow from the Helsinki Declaration blog which puts it this way:

> In the past twenty years we observe that the functionality (features) that is available to us inside the DBMS, has exponentially grown. These features enabled us to build database applications. Which is what we all started doing in the booming nineties.
>
> But then at the dawn of the new millennium, something happened. And that something mysteriously made the role of the DBMS inside a database application project diminish to insignificant. (...) As of the new millennium we

> are pushing all application logic out of the DBMS into middle tier servers. The functionality of stuff implemented outside the DBMS has exploded, and the feature rich DBMS is hardly used for anything but row-storage.

We are talking about stuff like

- Stored procedures used as data APIs (for security and to avoid excessive network traffic)
- Materialized views
- Instead-Of triggers
- Hierarchical queries (connect by)
- Geography (spatial data types)
- Analytics (lead, lag, rollup, cube, etc.)
- Virtual Private Database (VPD)
- Database-level Auditing
- Flashback queries
- XML generation and XSL transformation in database
- HTTP callouts from database
- Background job scheduler

Why are these features not being used? Why are most Java, .NET and PHP developers sticking with the "SELECT * FROM mytable" approach?

database

Share

Improve this question

Follow

edited Jun 20, 2020 at 9:12

Community Bot

1 ● 1

asked Sep 2, 2009 at 11:17

ObiWanKenobi

14.9k ● 11 ● 49 ● 51

13 +1 nice conversation-framer. – Dan Rosenstark Sep 2, 2009 at 11:50

could you post this as a sample question for the Outer Join proposal: area51.stackexchange.com/proposals/4260/outer-join (I'd do it and provide attribution, but I'm already at my 5 question limit) – Joe Jul 9, 2010 at 12:20

24 Answers

Sorted by: Highest score (default)

56

Because stored procedures:

- add another development language, increasing complexity and potentially redundant code (logic written in both languages);
- generally have worse tooling, monitoring and debugging capabilities than PHP, C#, Java, Python, etc;

- are generally less capable than most middle tier languages;
- only have an advantage with high volume data transformation (where you avoid the server roundtrip), which tends to only be a minimum of actual usage.

That being said, it's a common methodology on C# ASP.NET applications.

As Jeff Atwood put it, stored procedures are the assembly language of databases and people don't tend to code in assembly language unless they need to.

I've frequently used materialized views and sometimes used CONNECT BY in Oracle, neither of which I believe exist in MySQL.

I don't tend to use XML/XSLT in the database because, well, that means I'm using XML and XSLT.

As for geographical or spatial data structures, the reason there is probably that they're hard to just "pick up". It's a fairly specialist area. I've read the MySQL manual on spatial data structures and I'm sure it makes sense to someone with extensive GIS experience but to me and my limited needs (which tend to be around marking the latitude/longitude of a point) it just doesn't seem worth the time investment to figure it out.

Another issue is that if you go beyond ANSI SQL (much) then you've just tied yourself somewhat to a particular database vendor and possibly to a specific version. For that reason you'll often find application developers will tend to treat their databases at the lowest common denominator, which means treating them as a dumping ground for relational data.

Share Improve this answer Follow

edited Oct 21, 2018 at 3:49

Cœur
38.6k ● 26 ● 202 ● 276

answered Sep 2, 2009 at 11:26

cletus
625k ● 169 ● 917 ● 945

35 I'd add the fact that stored procedures are rarely placed under source control. – MusiGenesis Sep 2, 2009 at 12:14

20 Well, that might be true but there's no reason they *couldn't* be put under source control. – cletus Sep 2, 2009 at 12:34

5 All our sps are under source control, that's an excuse not a reason – HLGEM Sep 2, 2009 at 15:50

5 @Aric TenEyck: Are your *executables* under source control? Not some random text file that (hopefully) contains the most recent source code for them, but the actual compiled programs are under source control? The point is, as long as you have a good source control and deployment process, keeping .sql files in source control is completely sufficient. Of course, as with any process, it means developers need to get with the program, but that's not a problem specific to databases or SQL code. – Daniel Pryden Sep 2, 2009 at 18:17

8 I agree with that SPs "have an advantage with high volume data transformation (where you avoid the server roundtrip)". However, then you say that "tends to only be a minimum of

actual usage". I think that's the crux of the debate: if you have an application where high volume data transformation is a minimum of usage, adding lots of database functionality isn't worth it. However, if you have over a billion records in a table, and you need to select 24 8-hr sliding averages in 0.1 sec, then the database starts being a much better solution. The right answer really depends on your application.
– Daniel Pryden Sep 2, 2009 at 18:24

35

Because developers don't know about SQL. They rely on DDL and DML generated by tools like Hibernate and language level constructs like JPA annotations. Developers don't care if these are horribly inefficient because they are mercifully hidden by normal log levels and because DBAs are not part of development teams.

That's why I like tools iBATIS. They make you write and understand SQL, including DBMS specific features.

Share Improve this answer Follow

answered Sep 2, 2009 at 11:33
lutz

so I looked at your link . . . isn't iBATIS an ORM? Just one you have to define rather than having a tool do it for you?
– andrewWinn Sep 2, 2009 at 11:36

4 No, iBATIS is no ORM, it is a *query* mapper. It doesn't map objects on tables but queries on any language structure, object or not. – lutz Sep 2, 2009 at 11:39

so you tell it what objects (query results and what not), rather than having it do it for you? – andrewWinn Sep 2, 2009 at 11:42

3 +1 for "because developers don't know about SQL" and "because DBAs are not part of development teams". We have a DBA/database guru on our team, and we definitely use a lot more database features than most. That said, I've never heard of iBATIS before. At first glance, it doesn't look very impressive, but I'll file it to look into later.
– Daniel Pryden Sep 2, 2009 at 18:13

3 @andrewWinn: The whole point is that you can use the database for *a lot more* than CRUD functions.
– Daniel Pryden Sep 2, 2009 at 18:27

21

> I guess one reason is the fear of vendor lockin.

This doesn't get said all that often, but the benefits of using vendor-specific features need to be weighed against the cost. Mainly the cost of having to rewrite the parts that rely on vendor-specific features for every database you want to support. There is also a performance cost if you implement something in a general purpose way when the vendor provides a better way.

I'll bring up this example: one might find the "lockin" of SQL Server to be more acceptable once one realizes all of the things Analysis Services, Reporting Services, and so on can do for your application. For major commercial database systems, it is not "just" the SQL database engine that needs to be taken into account.

Share Improve this answer answered Sep 2, 2009 at 11:31

Follow

Jason Kresowaty
16.5k ● 9 ● 60 ● 85

7 Far more vendor lockin for ORMs than databases. It is very rare to need to change databases especially for web-based applications. – HLGEM Sep 2, 2009 at 15:53

1 I disagree. I've been on several projects where we were far down the road with MySQL. Then we learned that the client had some obscure internal protocol that required certain critical data to be housed in an Oracle DB. You can make the argument that this should've been called out in early requirements. But realistically, this happens and it can be a huge drain on a project. – Marco Sep 2, 2009 at 17:25

5 @Marco: MySQL is to Oracle as a child's toy gun is to the entire U.S. Army. That is, the first is perfectly adequate for playing around with and is free, while the second can actually protect you, but is mighty expensive and comes with lots of other demands. I guess your comment just doesn't make sense to me. How far down the road could you possibly have gone with MySQL? What feature were you using in *MySQL* that *Oracle* didn't have? – Daniel Pryden Sep 2, 2009 at 18:09

4 My comment wasn't necessarily about features. For a particular feature we were using, even if Oracle has it, it works differently. In syntax if nothing else (and usually more than just that). So all of that has to be ported and retested. The argument here is about the overhead of migrating which is significant, however you look at it. Are you suggesting that everyone pony up for Oracle just so all their bases are covered? <- That's not snark. What do you have against choosing a simple db for what *should* be a simple project? – Marco Sep 2, 2009 at 18:23

16

"Why are database features being ignored".

Because lots of so-called developers are completely ignorant about data management, and what's worse, they're completely ignorant of their ignorance too. "Unskilled and unaware of it", for whom this rings a bell.

Share Improve this answer Follow

answered Sep 2, 2009 at 17:51

Erwin Smout

1 I know some developers, and i'm not really an active one, i work mostly with spatial databases (modelling/dba), but that is so true. Developers don´t seem to know that creating and maintaning a easy, sane to use database is a complex task. – George Silva Sep 2, 2009 at 17:56

12

If your software runs on your client's hardware then any change to the database (new Stored Procedures, updated views, etc) will require DB administrator rights. This is nearly always a problem for clients. Involving the DB group complicates any updates you need to do. There are a lot of great reasons presented here, but this is the only one I need to avoid putting code in the database like the plague.

Share Improve this answer Follow

answered Sep 2, 2009 at 21:09

Jim Blizard

4,253 ● 30 ● 37

1 I have already seen some project, where this problem was realized too late. As soon the application was roled out, the database becomes a large burden. Data model between the database and the object world start breaking apart. Ugly workarounds are thrown into production. Database related bugs appear one after the other. A huge mess piles up. – Theo Lenndorff Sep 12, 2009 at 15:19

10

I guess one reason is the fear of vendor lockin. These DBMS features are not standardized - for example, stored procedures are very DB specific, and if you implemented stuff using stored procedures (instead of, say, web services exposed via a middle tier), then you are forever stuck with the DBMS first chosen, (that is, unless you're willing to spend time/money to re-implement it in another DBMS if you wanted to change DBMS).

Share Improve this answer Follow

answered Sep 2, 2009 at 11:21

Chii
14.8k ● 3 ● 39 ● 45

17 I've never been on a project where the chosen RDMS was changed later. – lutz Sep 2, 2009 at 11:30

2 even a change from one version to another could cause breaking changes to occur. – andrewWinn Sep 2, 2009 at 11:32

1 @lutz: thats because of what andrewWinn says - even a single change could potentially break old code, so the best, safest way is not to change. Hence, no one willingly wants to change their RDBMS. But that does mean that you dont want to have to rely on your RDBMS since if it is found to be

deficient, then all your custom stored proc or what ever services on it would need to be reimplemented or ported. Hence, my point - RDBMS doesnt provide a reliable way for services like stored proc to be interfaced in a backwards compatible way. – Chii Sep 2, 2009 at 11:52

1 @lutz : I *have* been in a situation where we changed DBs. (we hit the max table size in a 10+ year old Oracle 8 install, and with no money to upgrade the DB, had to migrate ... which meant re-implementing everything.) We probably spent more man-hours than what it'd have cost for an Oracle license, but those had been budgeted for. – Joe Sep 2, 2009 at 12:01

2 @lutz: amen. Building a system to handle a database brand switch down the road is a classic case of "putting might before will", except it's more like "putting won't before will". – MusiGenesis Sep 2, 2009 at 12:18

8

MySQL.

When web applications exploded in the late 1990s and early 2000s, MySQL was at version 3.3 or 4.0 and didn't support anything above simple `SELECT`s. It was, however, free, and installed with most Linux distributions. As a result, a generation of programmers didn't learn about databases and don't know how to use them.

Even now that MySQL is at 5.1 and supports most of the features of a commercial system, the same cruddy old blogs and articles are used as templates when a new LAMP project is kicked off, and MySQL is deployed with MyISAM tables and 3.3-era functionality.

Share Improve this answer Follow

answered Sep 2, 2009 at 20:59

Andrew Duffy
6,908 ● 2 ● 26 ● 17

6 +1. MySQL has done more harm than good -- both to the reputation of databases in general and to the programmers that use it. – Daniel Pryden Sep 3, 2009 at 3:03

Agreed - I actually did this, starting out with MySQL and only later filling up on how much else a REAL database system could do (foreign key relationships? cascading deletes? triggers? unique indexes? constraints?). – Keith Williams Sep 17, 2009 at 13:18

7

SQL is failing for the same reason as e.g. Haskell. The metric that determines language success is not purity, not ease of interpretation by computers, but how hard it is to maintain programs written in it.

Mere mortals fail with even the most simple language. Perhaps 1 in 10 people do have the skills to use a straightforward language like C#. But of those 10%, only 1 in 10 or 1% of all people can effectively use languages like SQL or Haskell.

Now, SQL is incomplete as a language, in the sense that there are very few things you can do with just SQL. You'll always need another language. What role does that leave for SQL? Developers will understand the ACID advantages over flat-file storage, but besides that databases really have nothing to offer them.

A second problem is that SQL effectively is not very compatible with Source Versioning. SQL seems really built along the notion that you get it right the first time. So, it's not just ill-suited for developers, it's also a poor match for the development process.

Share Improve this answer Follow

answered Sep 2, 2009 at 11:48

MSalters
179k ● 11 ● 164 ● 368

good point. I always wondered why those SPs are inside the friggin' database, unversionable. Why not have them in external text files, maybe with the option to cache? Also, true regarding how hard SQL is. SQL is freaky stuff: instead of asking interview candidates about outer vs. inner joins, I prefer to ask about stuff that makes sense :)
– Dan Rosenstark Sep 2, 2009 at 11:54

13 Wow, exactly wrong. SQL is an order of magnitude easier to learn and use (and use well) than C# or anything using .Net. Most programmers just don't try anymore. – RBarryYoung Sep 2, 2009 at 12:25

12 SQL has declarative syntax, COBOL is procedural, so your "facts" are weak. I've learned over 30 different languages and SQL was without question one of the easiest to learn, many many studies at the time it was created back this up as well (and that declarative languages in general, are easier to learn than imperative ones). That .net had useability as a concern does not diminish the fact that any API w 20k+ entry points takes a considerable while to learn and be proficient in. – RBarryYoung Sep 2, 2009 at 14:02

1 RBarry Young, I'd upvote your comment a million times if I could – HLGEM Sep 2, 2009 at 15:55

2 LuckyLindy - This is mainly because new programmers have more experience in C# than SQL. SQL is often a straight-and-to-the-point declarative language. This has clear benefits for writing and understanding code. Developers can focus on the business problem instead of technical OOP and design pattern matters. This is one of the main reasons we are seeing features like LINQ that bring these benefits to imperative languages. – Jason Kresowaty Sep 3, 2009 at 2:45

7

It's easier to fix/redeploy the middle tier than the DBMS.

This probably depends on your architecture, but it is our reason. Couple that with the fact that we have one DBA who is busier and (probably) is paid more than our developers. All the developers know SQL and some of them are semi-versed in the procedural language. If a really hairy production problem comes up, it would be easier and faster for the developers to work on the middle tier than the database, regardless of whether the architecture would be better one way or the other.

Share Improve this answer Follow

answered Sep 2, 2009 at 14:21

Shizzmo
16.8k ● 3 ● 24 ● 15

6

I've run into quite a few people who just weren't aware that such features existed -- they cut their teeth in the early days of mySQL, and they've never really used anything else, and they haven't kept up with the advancement of the new storage tables in mySQL, even.

Or they learned databases in school, and they've never gone back to see all of the stuff they missed.

They learn the bare minimum SQL to get by, and don't realize all of the different extensions that different RDBMSes offer.

On one project, I'd love to have materialized views ... but I'm using Postgres. I'd love to use spatial data types for another project, but I'm going to have to do a hack, or change databases to deal with mySQL's insistance that they be not null. I've even had to figure out how to disable Oracle's transactional consistency to deal with long-running queries on an OLTP that would've been no problem in mySQL.

I can normally code around the shortcomings of the database for a given problem, but part of the problem is in selecting the right tool for the job -- on a current project, we've wasted man-months on data replication because we're using Postgres, and they decided on Slony-1 before they actually knew all of what we'd be replicating.

... I view this question as being like 'why don't more people use *feature x* in *language y*' -- if they're not an expert in *language y* they might not know *feature x* exists.

(and don't take this as support for getting DBA certification ... I've known some Oracle DBAs that couldn't program their way out of a wet sack; I've taken all of the courses in the 8i days, but refused to take the tests as I didn't want to be lumped in with that group)

Share Improve this answer Follow

edited Jul 9, 2010 at 12:15

answered Sep 2, 2009 at 12:20

Joe
2,587 ● 1 ● 18 ● 27

2 PostgreSQL does have support for spatial data types.
– George Silva Sep 2, 2009 at 17:40

PostGIS (postgis.refractions.net) is a standard reason *to* use PG if you're not already :) – Gregg Lind Sep 4, 2009 at 13:56

5

I think the biggest reason that overshadows all the rest is that relational database systems become dramatically more important when multiple applications are sharing the same data. Codd's famous paper is titled "A Relational Model of Data for Large ***Shared*** Data Banks" (emphasis mine).

People have a tendency to think that the application they are writing now will always be controlled by their team; and that it will always satisfy all of the needs of people interested in the data generated by the application. If a new need arises, that would be satisfied by adding a new feature to an existing application, not creating a new application.

But in many cases (not all, of course; every situation is different), that development model doesn't work very well in the long term. As the data generated by the application accumulates and becomes more important to the business, different people will have interesting ideas about how to use the data. When that happens, if you don't have a relational database management system, you are in for a big challenge.

Share Improve this answer Follow

answered Sep 2, 2009 at 17:23

Jeff Davis
1,180 ● 8 ● 6

It probably won't surprise you to learn that DDD developers now consider "one true database" to be an anti-pattern. The latest trend is multiple databases synced through anti-corruption layers. – Daniel Auger Sep 3, 2009 at 2:07

5

Scalability. The more work you give to the database server, the greater a bottleneck it becomes. It's more scalable to have a whole farm of load-balanced application servers processing the data, and just use the database as a persistence store.

Share Improve this answer Follow

edited Sep 2, 2009 at 17:30

answered Sep 2, 2009 at 12:56

Christian Hayter
31k ● 6 ● 75 ● 101

5 So... you can use a "whole farm of load-balanced application servers", but you can't just use a whole farm of load-balanced database servers because...? Because you just don't know how? Then you probably need to learn about database clustering and replication. Because it certainly is possible. – Daniel Pryden Sep 2, 2009 at 18:04

Sorry, I should have qualified that I only have experience of SQL Server, which AFAIK does not support load-balancing, only fail-over. – Christian Hayter Sep 2, 2009 at 18:24

1 Yeah, SQL Server's support for load-balanced clusters is pretty poor. There are ways you can do it though, using Distributed Partitioned Views and Data-Dependent Routing. Oracle has RAC which is a much better solution for large, high-availability, load-balanced databases. Just be prepared to pay through the nose for it. – Daniel Pryden Sep 2, 2009 at 18:38

2 @Daniel - load balancing app servers is MUCH easier than load balancing database servers. Plus, it's typically much cheaper to buy an extra app server (i.e. just get an O/S and standard multi-CPU server) instead of an extra database server (O/S + Expensive DB License + Beefy DB server with fast drives, tons of memory, etc). – Beep beep Sep 3, 2009 at 1:08

@Daniel Pryden: You answer your own rhetorical question. "you can't just use a whole farm of load-balanced database servers because..." you have to be prepared to pay through the nose for it. – Coxy Sep 3, 2009 at 1:22

5

I have been in too many situations where corporate politics ("we arent allowed access to the SQL Server so lets install a lesser powerful DBMS like Access to process millions of rows and join that with millions of rows in another table, and lets automate that import..") or even the technical politics that can happen ("I know Access can handle that amount of data and even if it doesnt we can split the MDB into several MDBs and reference them.....")

UGH. Corporate politics and technical politics or even ignorance have prevented me from using many features.

Another example - I see no reason to not use stored procedures in a 100% Microsoft shop where SQL Server is **the** DBMS of choice. But, because the IT guy who was eventually going to own the solution was "light" on SPs, I had to resort to other measures. I mean, there is a perfect example of why some that "feature" went ignored by them at their shop.

I know another shop that still uses DOS Foxpro 2 because their sole IT guy wrote the existing system that way and that is how all new stuff will be developed. Why? Cant we move along with the times? Many of the marketing people over there have several DOS prompts open at once, with Foxpro "jobs" running in them to produce the most ugly reports I've ever seen. But it works - I will give them that. It works -- they have 12 million rows in their main table and 50+ other tables that they 'join' with that main table (not all 50 at once obviously) but man... its well past 1991! They dont even want to discuss one item from that bullet list you provided in your question.

Stuff like this is why I guess.

Share Improve this answer Follow

answered Sep 3, 2009 at 1:00
Taptronic
5,150 ● 9 ● 46 ● 59

4

I would say the biggest reason is that most people don't know about them. Once someone has figured out a solution to a problem, that becomes the default solution to similar ones. SELECT * FROM table has worked for a lot of people for a long time, so they don't bother looking at new approaches to old problems.

The other reason is that sometimes writing it in code is much easier than using a database. Its the same idea as rolling your own vs. buying an off the shelf component. Using a pre-written feature can solve your problems many times, but every once in a while, you need to do something which is outside of the capabilities of what the pre-written components can perform.

Share Improve this answer Follow

answered Sep 2, 2009 at 11:26
kemiller2002
115k ● 28 ● 199 ● 253

4

Nice question, and good discussion.

Another way to put it is "why haven't object DBs caught on?" which is the other side of the coin. DBs continue to be an annoying abstraction that still leaks its way into every app out there, but they're incompatible with the OO logic of modern applications.

It is indeed a strange state of affairs that we hide and duplicate the functionality of DBs in ActiveRecord, Hibernate, and other middlewares. But this is what happens to paradigms at the point of breakage (the "Object-relational impedance mismatch"). Will we ever transition to database technologies that are similar to our OO apps (e.g., object DBs)?

The answer is "not for a long time" and in the meantime, expect the DB to be ignored and squashed down and used for just row storage in many cases, as the middle-tier grows in functionality to bridge the gap.

Another question is "why would I do it in the DB if the middle-tier can do it?" The middle-tier is familiar and gaining ground in speed and functionality all the time. Again, we use the middle tier to avoid the OO-RDMS mismatch.

Share Improve this answer Follow

edited Sep 2, 2009 at 11:52

answered Sep 2, 2009 at 11:44

Dan Rosenstark
69.7k ● 60 ● 290 ● 424

4

To advance on what Christian said, about scalability.

Simply, RDBMs are being used more as pure data stores, while the logic has been migrating in to the Application

Servers. The extra tier of the AS give developers more flexibility than using the RDBMS as an Application Server.

Before, in the classic days of Fat Apps and Client Server, the DB and Application Server were basically the same thing. You had application logic either embedded in your fat client code, or you pushed it back in to the RDBMS. But back then, the primary form of communication was SQL directly to the database.

Nowadays, other application protocols are more common (CORBA, DCOM, Remote EJB, and more and more common today XML/JSON/HTTP-RPC style protocols over HTTP). Most databases don't respond to those protocols directly, so an Application layer is interjected to intercept those calls, and that layer calls the database.

But as we've learned we now get a lot more flexibility putting logic in to this layer. Wider choice of tools, more flexibility over caching, or failover, or, even database technology (RDMBS, OODBMS, Document stores like CouchDB). That "new", 3rd tier, despite the added complexity, adds more flexibility and power than the complexity it introduces.

When your app tier is a very thin veneer on top of Stored Procedures, it's valid to question why it's even there at all.

Leveraging the database and all of its features is a valid application strategy, even today. SQL Server, Oracle, etc, are terribly powerful pieces of software.

Even then, though, the third tier is enormously helpful in adding flexibility to a modern system.

Share Improve this answer Follow

edited May 23, 2017 at 11:45
Community Bot
1 ● 1

answered Sep 2, 2009 at 17:59
Will Hartung
118k ● 20 ● 133 ● 207

3

For me, the reason is not only my applications being database agnostic, but a database best preforms the basic CRUD functions. Yeah, databases are highly optimized, and might be able to make an HTTP callout, but why would you do it? A webservice / web application is optimized for HTTP calls, not a database. Just like an application is not designed to connect directly to a datafile and retrieve the data. Can it be done? Yes, but why? That is not what your application EXCELLS at.

I personally feel that everything you mentioned, out side of stored procedures belongs in the application. If you know your architecture is X then take advantage of X's features, hand load off to the DB server when appropriate, etc... If it could be X or Y (or Z), then your application should be agnostic, unless you are trying to create job security by ensuring that you might have to refactor the application :). I think a little bit of laziness, combined with comfortablity might have something to do with it. I know I would rather do it in C# than SQL if I can . . . my C# skills are just better.

Share Improve this answer Follow

answered Sep 2, 2009 at 11:31

andrewWinn
1,786 ● 2 ● 17 ● 28

1 The point is that you can use the database for a lot more than CRUD functions. If all you need is CRUD, then use sqlite. The point is that if you're doing data processing, especially statistics, averages, or interpolations, on large datasets (more than one million rows at least), then databases have a whole slew of other features that are easier to exercise in SQL than C#. Interestingly, LINQ is adding a lot of similar functionality, essentially building database functionality and SQL-like declarative syntax into C#. The whole point is that data processing *is* what databases *excell* at! – Daniel Pryden Sep 2, 2009 at 18:46

I see your point, and to me, stats and what not are part of a read function. I don't see the benefit of a DB making an http calls or generating XML documents. This explicitly ties your application to specific features of a database, and mitigates any possiblity of porting the application to another DB vendor without causing a major rewrite. . . have you ever gone from MySQL to SQL server? there are enough diffenrences in syntax and what not that anything resembling a complex query will more often then not have to be rewritten. Thus increasing the possibility of introducing new errors.
– andrewWinn Sep 3, 2009 at 11:35

3

First off: any developer using an ORM is naive if S/he thinks using an ORM negates having to have SQL skills. Most ORMs that generate SQL vary the SQL emitted depending on how the object queries are constructed. The developer will need to analyze the SQL to see if they should change any of the object queries.

Short answer: A lot of those features aren't practical for OO development. I know that DBAs don't like to hear that but, it's the truth. Those features are good for edge cases and most good ORMs such as N/Hibernate allow you to provide SQL for those edge cases.

When it comes to being mostly delegated to CRUD:

Long answer: I think the RDBMS world is going through maturity growing pains, and is finding it's place in the world. Truth: OOP is older than RDBMS. OOP is just getting out of it's growing pains and maturing. I think SQL as a language is very mature, but the idea of what a RDBMS should handle is just getting settled. The RDBMS was the business logic holder for most web apps until Java and C# came around. I think we are just starting to feel this correction now.

That being said, I don't think any ORM designer is going to tell you that the quality of the sql statements fed to the RDBMS don't matter.

When it comes to non-CRUD I don't have an answer here. Most shops I know of still use the DB for ETL/etc...

edited Sep 12, 2009 at 14:49

answered Sep 3, 2009 at 2:13

Daniel Auger
12.6k ● 5 ● 53 ● 73

"Idiot" is such a strong word. Maybe *"naive"*, *"lazy"*, or *"inexperienced"* would be equally as accurate without the nastiness. Just barely. – Stu Thompson Sep 12, 2009 at 12:01

Good point. I have de-nastied the answer to some extent. I went with naive. – Daniel Auger Sep 12, 2009 at 14:50

2

There are not enough developers knowing all those features at a level that would really make the difference to a normal 'middle tier' programmer, when it comes to implementing the same logic into DB or middle tier. Maybe the single people that really have in-depth knowledge of that features are DBAs. And those focus on other problems than development. There are more 'normal' developers out there than DBAs. So it would be very difficult and costly to find the right people for your team.

Another point is, that you will normally only gather the in-depth knowledge about *one* database system, and not all of them. So you can have SQL Server experts or Oracle experts, but not both. This leads (to an extent) to narrow application fields where high specialization counts. Then, the market for such applications isn't that big, even if it's there.

Share Improve this answer

Follow

answered Sep 2, 2009 at 15:46

MicSim

26.8k ● 16 ● 93 ● 136

1

I think the reason is a combination of vendor lock-in and lack of knowledge on the part of most RDBM users. SQL is a programming language, and it's much harder to master both the language you're calling SQL from and SQL than to master one or the other, especially since SQL is a particularly unique language.

The solution, I think, is to abstract your database functionality into a utility class, and give ownership of the class to a few users who know what they are doing with SQL. This minimizes the risk of vendor lockin (if you switch vendors, the only thing that gets rewritten is the class). This also gives the developers who aren't experts in SQL an abstracted-out interface so they don't have to deal with the database directly.

Share Improve this answer

Follow

answered Sep 2, 2009 at 20:46

Imagist

18.5k ● 12 ● 61 ● 80

1

I have always worked on systems that are sold to many clients and runs on the client’s hardware. This leads to:

- You don’t know what version of the database software the client will be running.

- The customers may not be willing to update the database software when you wish due to licence costs or there IT policy.
- Even basic features like materialized views are only in some “editions” of the database software, the smaller customers are often not willing to pay for the high end edition of the database.
- Sooner or later one of your sales people will agree to your software being used with a different vendor’s RDBMS.
- Choosing between writing the logic once in the middle tier, or at least PL-SQL and TSQL in the database is an easy choose!
- Any change to the database (new Stored Procedures, updated views, etc) will require DB administrator rights; this can be a lot harder on some customers site then just updating your software.
- Writing a script to update the database is always harder than installing a new version of the application’s dlls. (Most build systems these days output MSI files as part of every build)
- It is harder to unit test code in a database then a middle tier.
- It is harder to debug stored procs than C#.
- Just because it works on your database does not mean it will work on the customer’s database, RDBMS have too many switches that change how

they work – customers always tend to set them in different ways.

So given all the above, the gain from using a database feature **must be great** before it is worth the long term pain.

Share Improve this answer Follow

answered Sep 1, 2010 at 16:50

Ian Ringrose
51.9k ● 57 ● 217 ● 321

0

One concern I have seen with leveraging increased database functionality is scaling. It seems to be a much more difficult proposition to scale your database load vs web/application server load.

Your options are limited, scaling-up with bigger faster hardware (with sometimes much higher licensing costs), or complicated, scaling-out with read-only copies, etc.

If there are performance issues, I want them to be at the web server application level. At least then one of my options is to add another web server and distribute load.

I am not arguing against database level code to minimize the amount of network traffic (records) sent between the web server and database server. I am arguing against other features, eg. extensive business logic processing at the database level.

Share Improve this answer

answered Sep 3, 2009 at 18:04

Follow

Joseph Kingry
8,228 ● 4 ● 38 ● 52

0

A few posts note that it is cheaper to scale at the application layer than at the db layer.

Another consideration is composite applications that access multiple data stores. Much easier to write and maintain platform agnostic query language in the app tier than separate platform specific queries in the the db layer.

Share Improve this answer

Follow

answered Sep 9, 2009 at 19:08

Kevin Thex

0

Because writing object-oriented software, in the host language, with native host-language objects, beats writing procedural software.

Share Improve this answer

Follow

answered Sep 12, 2009 at 15:15

yfeldblum
65.4k ● 12 ● 131 ● 169