# How much of an applications "smarts" should reside in the database? [closed]

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

Viewed 524 times

10

**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 6 years ago.

Improve this question

I've noticed a trend lately that people are moving more and more processing out of databases and in to applications. Some people are taking this to what seems to me to be ridiculous extremes.

I've seen application designs that not only banned all use of stored procedures, but also banned any kind of constraints enforced at the database (this would include primary key, foreign key, unique, and check constraints). I have even seen applications that required the use of only one data type stored in the database, namely varchar(2000). DateTime and number types were not allowed. Transactions and concurrency were also handled outside the database.

Has anyone seen these kind of applications implemented successfully? Both of the implementations I've dealt with that were implemented this way had all kinds of data integrity and concurrency problems. Can anyone explain this trend to move stuff (logic, processing, constraints) out of the database? What is the motivation behind it? Is it something I'm imagining?

database

Share

Improve this question

Follow

asked Jan 21, 2009 at 6:49

RussellH
1,245 ● 1 ● 9 ● 13

## 13 Answers

Sorted by: Highest score (default)

8

Firstly, I really hope there is no trend towards databases without PKs and FKs and sensible datatypes. That would really be a tragedy.

But there is definitely a large core of developers who prefer putting logic in their apps than in stored procedures. I agree with Riho on the main reason for this: usually, DBAs manage databases, meaning that a developer has to go through a bunch of administrative overhead -- getting approvals from the DBA -- in order to create and update stored procs. Programmers by nature like to have control over their world, and to do things "their way."

There are also a couple of valid technical reasons:

1. Procedural extensions to SQL (e.g. T-SQL) used for developing stored procs have traditionally lacked user-defined datatypes, debuggability, portability, and interoperability with external systems -- all qualities helpful for developing reliable large-scale software. (And the clumsy syntax doesn't help.)
2. Software version control (e.g. svn) works well for managing even very large codebases, but managing DB schema changes is a harder problem and less well supported. Every serious shop uses version control for their application codebase, but many still don't have any rigorous system for managing their databases; hence stored procs can easily fall into an unversioned "black hole" that makes coders rightly nervous.

My personal view is that the closer the core business logic is to the data, the better, especially if more than one agent accesses the DB (or may do in the future). It's an unfortunate artefact of history that T-SQL and its ilk were weak languages, leading to the rise of the idea that "data and logic should be separated." My ideal world is one in which every business rule is encapsulated in a constraint enforced by the database, and all inconsistencies *fail fast*.

Share Improve this answer Follow

answered Jan 21, 2009 at 11:03
j_random_hacker
51.2k ● 10 ● 108 ● 173

Cludgy syntax and no version control is enough for me to put the logic in the code. That dosn't mean I don't use constraints - so queries still fail if I attempt to violate them. – user19302 Jan 21, 2009 at 19:41

6

I like to keep logic out of the database. I tend to avoid stored procedures and triggers. I do, though, always use proper data types, keys, indicies and constraints. The way I see it is that the database is a database and the application is the application. The database should keep your data stored properly and efficiently whereas the application should own the logic. Perhaps I have never been in a situation where a stored procedure or trigger was needed; and thus never been inclined to use them to solve a problem. But to me, giving logic a home on the database seems "messy" to me; I would rather control everything from the application itself.

Share Improve this answer Follow

answered Jan 21, 2009 at 6:56
user19302

4

The trend results from the fact that the software technology industry is populated and driven largely by humans, and thus subject to trends and irrational behavior. To understand what's going on today requires a bit of perspective in the history of databases, and their parallel development with programming languages.

To be brief in this answer that will likely get downvoted: SQL is the IE6 of the database languages world. It breaks many of the rules of the relational model- in other words, it's a little bit like a calculator that performs multiplication incorrectly, and doesn't have a minus operator. SQL is not complete enough to be a real solution. It was never developed beyond the prototype stage, and was never meant to be used in industrial settings. But then it was naively used by oracle, which turned out to be a "killer app", SQL became industry standard instead of its technically superior competitors, and the rest is history. SQL's syntax is based around a set of command line tabular data processing tools, and COBOL. Full of bugs, inconsistencies, and a mishmash proprietary versions and features that don't have a grounding in math or logic, results in a situation where it really is unclear what goes where.

I think the trend you must be talking about is recent proliferation of ORMs: misguided and ill thought out attempts to patch over the obvious deficiencies of SQL. Database triggers and procedures are another misfeature trying to patch over SQL's problems.

If history had played out in a logical and orderly way, the answer to your question would be simple: Just follow the rules of the relational model and everything will work itself out. Unfortunately, the rules of the relational model don't fit cleanly into the current crop of SQL based DBMS's, so some application level fiddling, or triggers, or whatever other stupid patch is unfortunately necessary, and it ends up being a matter of subjective opinion, rather than reasoned argument, which stupid hack you use.

So the real answer is to just follow the relational model as close as you can, and then fudge it the rest of the way. Put the logic in the application if you're the only one using the db, and you need to keep all your source code in a version repository. If multiple applications are likely to use the database, make the DB as bullet proof and self sufficient as it can be- The main goal here is to ensure that the data remains consistent.

Share Improve this answer Follow

edited Jan 21, 2009 at 8:57

community wiki
6 revs
Breton

---

3

Ultimately the database and how you connect to it is your "persistence API" -- how much is in the database and how much is in the application is application-specific. But the important aspect is that the API **boundary** is responsible for producing or consuming correct data.

Personally I prefer a thin access layer in the application and sprocs/PKs/FKs in the database to enforce transactional correctness and to enable API versioning. This also allows other applications to modify the database without upsetting the data model.

And if I see another moron using *SELECT * FROM blah* I'm going to go nuts with an Uzi... :-)

Share Improve this answer Follow

answered Jan 21, 2009 at 9:11

devstuff
8,357 ● 1 ● 29 ● 33

3

"The database should keep your data stored properly and efficiently whereas the application should own the logic" - Nelson LaQeut in another answer.

This seems to be the crux of the issue: that all "logic" belongs to the application and not to the database. But what is meant by "logic"? There are various kinds of "logic", some of which belong in an application and some, I would say, better placed in the database.

I would think most developers would agree (surely?) that basic data integrity such as primary and foreign keys belongs in the database. There is less agreement on more sophisticated data integrity logic - even the humble but useful check constraint is woefully underused in general. .

The application camp see the database is "merely" a place to store the data that "belongs" to their application. The database camp (which is where I sit) see the application as "merely" one (perhaps currently the only) user of the data that "belongs" to the database - or rather that belongs to the business and is **managed** for the business by the database (DBMS = database **management** system).

If all your data logic is tied up in your application, what happens when the application needs to be rewritten in the latest trendy paradigm (or do you think J2EE for example is the last there will ever be)? As Tom Kyte often says, it's all about the data.

Share Improve this answer Follow

answered Jan 21, 2009 at 10:45

Tony Andrews
**132k** ● 23 ● 227 ● 264

2

The database is an integral part of an application, but everyone interprets that differently. It's definitely a wise move to isolate them, but that shouldn't mean that you circumvent what they do in your programming. Correct data types and primary key references are important parts of good database design, on top of which a good application can be built.

Share Improve this answer Follow

answered Jan 21, 2009 at 6:56

JMS
2,193 ● 3 ● 15 ● 16

---

2

Although I personally believe the Database should have enough smarts to defend itself, some people that don't understand that Databases aren't dumb services, think, and not incorrectly mind you, that data and logic should be separated. Now in many cases the separation of data and logic is a powerful tool, however most databases already provide us with solid implementations of atomicity, redundancy, processing, checking, etc... And many times that's where it belongs, however since the quality of these services and their API differs among vendors, many application programmers have felt like its worth trying to implement this sort of stuff in the application layer, to avoid tying themselves up with a specific database layer.

Share Improve this answer Follow

answered Jan 21, 2009 at 7:21

Robert Gould
69.7k ● 61 ● 191 ● 275

---

1

I can't say that I've seen a "trend" to create poor applications with terrible database designs. Programming is just like any other discipline in that there will be people who won't learn the tools or just want to cut corners. I've even talked to a person that just didn't "trust" databases. The applications that you described are just as you said, ridiculous nightmares. Don't follow those "trends".

Share Improve this answer

Follow

answered Jan 21, 2009 at 6:55

Al W

7,703 ● 1 ● 20 ● 41

1

I still prefer to use Stored Procedures and functions in SQL server. It adds more flexibility to application acturally. And it has a performance benefit also. Generally I don't think it is good idea to put everything to applicatons.

Share Improve this answer

Follow

answered Jan 21, 2009 at 7:04

Ken Yao

1,516 ● 1 ● 13 ● 24

1

I think that those "developers" who created databases without indexes or with single VARCHAR(2000) column are just art majors who are making their first attempt into entering the high-priced IT world. No-one, who has even little-bit of IT education, makes this kind of database structures.

I can understand the reason to keep logic out of the well formed database. Usually it is time-consuming to make changes (you have to convince database admins to make it, and all the red-tape that comes with it). If the business logic is in your program, then its up to you only.

Share Improve this answer

Follow

edited Jan 21, 2009 at 7:51

answered Jan 21, 2009 at 7:21

Riho
**4,583** ● 3 ● 35 ● 49

google.com/… – user19302 Jan 21, 2009 at 8:30

1

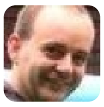
Use constraints in the database, but for any sophisticated logic I would place that in a data access layer or use one of the standard Object Relational Mapping (ORM) tools such as Hibernate/NHibernate.

There is a general belief that this will affect performance; based on the view that stored procedures are pre-compiled but 'raw' queries have to be compiled on every call. However, I work mostly in SQL Server 2005/2008, and that is very efficient at handling 'raw' parameterised queries, caching the compiled query path for future calls to the database. This means that there under SQL Server there is essentially no difference between the performance of stored procedures to parameterised SQL queries.

The only downside on losing stored procedures is if you are very granular with your database security permissions, and which to enforce security at the database login level.

Share Improve this answer Follow

answered Jan 21, 2009 at 8:58



Liam Westley
406 ● 4 ● 7

---

1

I have a simple philosophy.

- If it's need to keep the database secure and in a consistant state, make sure to do it in the database

I do try to keep a lot of other stuff there too, in my world it's easier to update a client's database than it is to update their application...

Essentially I try to treat the database as a pseudo object. A bunch of methods I can call, etc, but I don't want the app to care about the detail of the internal data storage.

Share Improve this answer Follow

answered Jan 21, 2009 at 12:47

MatBailie
**86.6k** ● 19 ● 109 ● 143

---

-1

In my experience, putting any application logic in the database *always* results in a WTF. It doesn't matter how smart the database programmer, how advanced the database, it always ends up being a mistake. The reverse question is "how often should my C# code manage relational data using its own flat-file structure and query language", to which the answer is (almost) always *never*.

I think the database should be used for data storage, which is what it's good at.

Share Improve this answer Follow

answered Jan 21, 2009 at 9:33

endian
**4,294** ● 8 ● 37 ● 44