# What's the better database design: more tables or more columns?

Asked 16 years, 3 months ago  Modified 1 year, 6 months ago

Viewed 75k times

105

A former coworker insisted that a database with more tables with fewer columns each is better than one with fewer tables with more columns each. For example rather than a customer table with name, address, city, state, zip, etc. columns, you would have a name table, an address table, a city table, etc.

He argued this design was more efficient and flexible. Perhaps it is more flexible, but I am not qualified to comment on its efficiency. Even if it is more efficient, I think those gains may be outweighed by the added complexity.

So, are there any significant benefits to more tables with fewer columns over fewer tables with more columns?

database  database-design  database-normalization

Share

Improve this question

Follow

edited Oct 22, 2022 at 20:38

philipxy

15.1k ● 6 ● 42 ● 94

asked Sep 12, 2008 at 16:45

raven

18.1k ● 17 ● 82 ● 114

## 18 Answers

Sorted by: Highest score (default)

78

I have a few fairly simple rules of thumb I follow when designing databases, which I think can be used to help make decisions like this....

1. Favor normalization. Denormalization is a form of optimization, with all the requisite tradeoffs, and as such it should be approached with a YAGNI attitude.
2. Make sure that client code referencing the database is decoupled enough from the schema that reworking it doesn't necessitate a major redesign of the client(s).
3. Don't be afraid to denormalize when it provides a clear benefit to performance or query complexity.
4. Use views or downstream tables to implement denormalization rather than denormalizing the core of the schema, *when data volume and usage scenarios allow for it*.

The usual result of these rules is that the initial design will favor tables over columns, with a focus on eliminating redundancy. As the project progresses and denormalization points are identified, the overall structure will evolve toward a balance that compromises with

limited redundancy and column proliferation in exchange for other valuable benefits.

Share Improve this answer

Follow

answered Sep 12, 2008 at 17:02

Chris Ammerman

**15.3k** ● 8 ● 42 ● 42

1 What exactly is a 'downstream table'? – olive Sep 28, 2010 at 11:44

1 I mean "downstream" in the context of a "data flow". Which essentially means you have a process which uses the normalized tables as a source, and transforms the data somehow, and then deposits the result somewhere else. – Chris Ammerman Sep 29, 2010 at 11:30

14

It doesn't sound so much like a question about tables/columns, but about normalization. In some situations have a high degree of normalization ("more tables" in this case) is good, and clean, but it typically takes a high number of JOINs to get relevant results. And with a large enough dataset, this can bog down performance.

Jeff wrote a little about it regarding the design of StackOverflow. See also the post Jeff links to by Dare Obasanjo.

Share Improve this answer

Follow

edited Aug 25, 2015 at 18:21

izstas

**5,064** ● 3 ● 43 ● 57

answered Sep 12, 2008 at 16:48

swilliams

**48.8k** ● 27 ● 102 ● 130

3 In my experience, this is patently false. I've worked with queries that join dozens of tables, *each* containing 1million + rows, and as long as you're joining on primary keys, the results come back very quickly. – JosephStyons Sep 12, 2008 at 16:51

2 What's 'quickly'? If you're running a website trying to serve thousands of pageviews a second 'fast enough' as an entirely different meaning than a single user database where all you're concerned about is response time for the user. – Chris Upchurch Sep 12, 2008 at 16:54

"as long as you're joining on primary keys, the results come back very quickly" Well, yeah. But, in my experience with more tables, the more likely it is for joins to happen on non-pk's, non-indexed columns, etc. – swilliams Sep 12, 2008 at 16:55

2 Normalisation and the subsequent joining of tables usually helps performance since by definition you can be more selective and avoid table scans - the slowest method of selecting. – Ed Guiness Sep 12, 2008 at 17:01

2 Poor design is usually the biggest factor in poor performance, not normalisation. – Ed Guiness Sep 12, 2008 at 17:03

**13**

I would argue in favor of more tables, but only up to a certain point. Using your example, if you separated your user's information into two tables, say USERS and ADDRESS, this gives you the flexibility to have multiple

addresses per user. One obvious application of this is a user who has separate billing and shipping addresses.

The argument in favor of having a separate CITY table would be that you only have to store each city's name once, then refer to it when you need it. That does reduce duplication, but in this example I think it's overkill. It may be more space efficient, but you'll pay the price in joins when you select data from your database.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:50

Bill the Lizard

**405k** ● 211 ● 572 ● 889

---

6

A fully normalized design (i.e, "More Tables") is more flexible, easier to maintain, and avoids duplication of data, which means your data integrity is going to be a lot easier to enforce.

Those are powerful reasons to normalize. I would choose to normalize first, and then only denormalize *specific* tables *after* you saw that performance was becoming an issue.

My experience is that in the real world, you won't reach the point where denormalization is necessary, even with very large data sets.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:54

JosephStyons

**58.6k** ● 64 ● 167 ● 237

Agreed. I've only ever denormalized to reduce query complexity, usually to eliminate some impedance mismatch with an ORM. Never because the optimization was required for performance, even at great scale. – Ben Simmons Feb 7, 2015 at 9:42

6

Each table should only include columns that pertain to the entity that's uniquely identified by the primary key. If all the columns in the database are all attributes of the same entity, then you'd only need one table with all the columns.

If any of the columns may be null, though, you would need to put each nullable column into its own table with a foreign key to the main table in order to normalize it. This is a common scenario, so for a cleaner design, you're likley to be adding more tables than columns to existing tables. Also, by adding these optional attributes to their own table, they would no longer need to allow nulls and you avoid a slew of NULL-related issues.

Share Improve this answer Follow

answered Sep 12, 2008 at 17:00

Mark Cidade

99.8k ● 33 ● 229 ● 237

5

It depends on your database flavor. MS SQL Server, for example, tends to prefer narrower tables. That's also the more 'normalized' approach. Other engines might prefer it

the other way around. Mainframes tend to fall in that category.

Share Improve this answer

Follow

edited Dec 29, 2014 at 17:24

raven

**18.1k** ● 17 ● 82 ● 114

answered Sep 12, 2008 at 16:47

Joel Coehoorn

**415k** ● 114 ● 577 ● 813

Hi Joel, is narrower tables means less table? Taking on the example above, if a company address must only be 1 address, will it be better to put all the address' fields on a separate table or on the same table? Also, if the company made a lot of purchases and I want to store the summary (eg. `TotalOrders, TotalOrderValue, TotalComplains, etc..`), would it better to put these fields on a different table (even though it is 1-to-1 with company table)? – Sam Jun 23, 2021 at 1:52

1 @Sam "Narrower" means fewer columns per table, and therefore possibly more tables to handle the same fields. It's very rarely a good idea to store summaries. If you index properly, even for large companies it can still be efficient to build the summary at the time of request. – Joel Coehoorn Jun 23, 2021 at 13:52

4

Like everything else: it depends.

There is no hard and fast rule regarding column count vs table count.

If your customers need to have multiple addresses, then a separate table for that makes sense. If you have a really good reason to normalize the City column into its own table, then that can go, too, but I haven't seen that before because it's a free form field (usually).

A table heavy, normalized design is efficient in terms of space and looks "textbook-good" but can get extremely complex. It looks nice until you have to do 12 joins to get a customer's name and address. These designs are not *automatically* fantastic in terms of performance that matters most: queries.

Avoid complexity if possible. For example, if a customer can have only two addresses (not arbitrarily many), then it might make sense to just keep them all in a single table (CustomerID, Name, ShipToAddress, BillingAddress, ShipToCity, BillingCity, etc.).

Here's Jeff's post on the topic.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:53

Michael Haren

**108k** ● 41 ● 171 ● 206

---

3

The multi-table database is a lot more flexible if any of these one to one relationships may become one to many or many to many in the future. For example, if you need to store multiple addresses for some customers, it's a lot easier if you have a customer table and an address table. I can't really see a situation where you might need to

duplicate some parts of an address but not others, so separate address, city, state, and zip tables may be a bit over the top.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:50

Chris Upchurch

**15.5k** ● 6 ● 52 ● 66

I have 40 unique fields about user information which are unique and they are one to one from User Authentication System. Do you think its ok if I keep those 40 columns in one table? If I separate them I need to write more joins in my queries :-(. Can you suggest – Vikram Jun 17, 2013 at 1:28

**3**

When you design your database, you should be as close as possible from the meaning of data and NOT your application need !

A good database design should stand over 20 years without a change.

A customer could have multiple adresses, that's the reality. If you decided that's your application is limited to one adresse for the first release, it's concern the design of your application not the data !

It's better to have multiple table instead of multiple column and use view if you want to simplify your query.

Most of time you will have performance issue with a database it's about network performance (chain query

with one row result, fetch column you don't need, etc) not about the complexity of your query.

Share Improve this answer

Follow

answered Jul 29, 2014 at 7:46

Marco Guignard

643 ● 3 ● 9

---

2

There are advantages to having tables with fewer columns, but you also need to look at your scenario above and answer these questions:

Will the customer be allowed to have more than 1 address? If not, then a separate table for address is not necessary. If so, then a separate table becomes helpful because you can easily add more addresses as needed down the road, where it becomes more difficult to add more columns to the table.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:48

Dillie-O

29.7k ● 14 ● 102 ● 141

---

1

i would consider normalizing as the first step, so cities, counties, states, countries would be better as separate columns... the power of SQL language, together with today DBMS-es allows you to group your data later if you need to view it in some other, non-normalized view.

When the system is being developed, you might consider 'unnormalizing' some part if you see that as an

improvement.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:48

zappan

**3,685** ● 4 ● 32 ● 24

1 My 2 cents: I have to disagree; doing that kind of optimization during design is a classic case of premature optimization. Wait until you see that performance is a problem *before* you sacrifice a good design. – JosephStyons Sep 12, 2008 at 16:55

---

1

I think balance is in order in this case. If it makes sense to put a column in a table, then put it in the table, if it doesn't, then don't. Your coworkers approach would definately help to normalize the database, but that might not be very useful if you have to join 50 tables together to get the information you need.

I guess what my answer would be is, use your best judgement.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:48

Craig H

**7,979** ● 16 ● 51 ● 61

---

1

Hmm.

I think its a wash and depends on your particular design model. Definitely factor out entities that have more than a

few fields out into their own table, or entities whose makeup will likely change as your application's requirements changes (for instance - I'd factor out address anyways, since it has so many fields, but I'd *especially* do it if you thought there was any chance you'd need to handle foreign country addresses, which can be of a different form. The same with phone numbers).

That said, when you're got it working, keep an eye out on performance. If you've spun an entity out that requires you to do large, expensive joins, maybe it becomes a better design decision to spin that table back into the original.

Share Improve this answer

Follow

answered Sep 12, 2008 at 17:46

John Christensen

**5,030** ● 1 ● 29 ● 26

---

1

There are many sides to this, but from an application efficiency perspective more tables can be more efficient at times. If you have a few tables with a bunch of columns every time the db as to do an operation it has a chance of making a lock, more data is made unavailable for the duration of the lock. If locks get escalated to page and tables (well hopefully not tables :) ) you can see how this can slow down the system.

Share Improve this answer

Follow

edited Jun 12, 2023 at 18:25

joshuakcockrell

**6,063** ● 2 ● 39 ● 50

answered Sep 12, 2008 at 16:49

kemiller2002

115k • 28 • 199 • 253

0

There are huge benefits to **queries** using as few columns as possible. But the table itself can have a large number. Jeff says something on this as well.

Basically, make sure that you don't ask for more than you need when doing a query - performance of queries is directly related to the number of columns you ask for.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:49

ColinYounger

6,865 • 5 • 33 • 33

0

I think you have to look at the kind of data you're storing before you make that decision. Having an address table is great but only if the likelihood of multiple people sharing the same address is high. If every person had different addresses, keeping that data in a different table just introduces unnecessary joins.

I don't see the benefit of having a city table unless cities in of themselves are entities you care about in your application. Or if you want to limit the number of cities available to your users.

Bottom line is decisions like this have to take the application itself into considering before you start

shooting for efficiency. IMO.

Share Improve this answer

Follow

answered Sep 12, 2008 at 16:50

Tundey

2,965 ● 1 ● 24 ● 28

---

0

First, normalize your tables. This ensures you avoid redundant data, giving you less rows of data to scan, which improves your queries. Then, if you run into a point where the normalized tables you are joining are causing the query to take to long to process (expensive join clause), denormalize where more appropriate.

Share Improve this answer

Follow

answered Aug 11, 2014 at 21:01

pbars23

95 ● 1 ● 6 ● 20

---

-1

Good to see so many inspiring and well based answers.

My answer would be (unfortunately): it depends.

Two cases: * If you create a datamodel that is to be used for many years and thus possibly has to adept many future changes: go for more tables and less rows and pretty strict normalization. * In other cases you can choose between more tables-less rows or less tables-more rows. Especially for people relatively new to the subject this last approach can be more intuitive and easy to comprehend.

The same is valid for the choosing between the object oriented approach and other options.

Share Improve this answer

Follow

answered Nov 21, 2016 at 12:11

Bart Rozinga

1