What is important to keep in mind when designing a database?

Asked 16 years, 2 months ago Modified 14 years, 3 months ago

Viewed 11k times

18

What is important to keep in mind when designing a database?

I don't want to limit your answer to my needs as I am sure that others can benefit from your insights as well. But I am planning a content management system for a multi-client community driven site.

database-design

Share

Improve this question

Follow

edited Sep 26, 2008 at 18:36

Hank Gay

71.8k • 36 • 161 • 222

asked Sep 26, 2008 at 18:26

Patcouch22

912 • 3 • 11 • 29

I think this is a good question but thought the title could be clarified a bit. Feel free to roll back my edit if you disagree. – Mark Biek Sep 26, 2008 at 18:34

Should be community wiki – Josh Stodola Sep 13, 2010 at 21:18

23 Answers

Sorted by: Highest score (default)

"Normalize till it hurts; de-normalize till it works."

41

Share Improve this answer

Follow

answered Sep 26, 2008 at 18:32

Keng

53k • 32 • 84 • 111

Never heard that before... nice! – cagcowboy Sep 26, 2008 at 18:33

1 Who said it? Good answer anyway, +1. – fastcodejava Sep 13, 2010 at 20:54

2 @Keng there are plenty actual answers for the question, it's just too bad that they didn't get upvoted as much as the empty phrases. – aaaaaaaaaaaa Feb 6, 2012 at 14:34

(Assuming OLTP)

21

Normalisation of your data-structures. (Performance de-normalisations can *generally* follow later where needed)

http://en.wikipedia.org/wiki/Database_normalization

Share Improve this answer

Follow

answered Sep 26, 2008 at 18:29

cagcowboy

30.8k ● 11 ● 74 ● 94

15

Make sure you use constraints (`CHECK`, `NOT NULL`, `FOREIGN KEY`, `PRIMARY KEY`, and `DEFAULT`) to ensure that only correct data is stored in the database in the first place. You can always buy faster hardware but you cannot buy more correct data.

Share Improve this answer

Follow

answered Sep 26, 2008 at 18:49

jalbert

3,097 ● 2 ● 21 ● 21

15

Establish consistent naming standards up-front. It will save several minutes of unnecessary thinking in the long run. (This may read as irony, but I am serious.)

And don't abbreviate anything, unless it is *extremely* common. Don't turn the database into a license-plate message guessing game. It's amazing what becomes not-obvious after a year.

Share Improve this answer

Follow

answered Sep 26, 2008 at 19:13

Jeffrey L Whitledge

59.4k ● 9 ● 74 ● 100

10

Try to imagine the SQL queries that you will preform against it.

This is important because you will do it A LOT!

Share Improve this answer Follow

edited Nov 24, 2008 at 18:40

answered Sep 26, 2008 at 18:40

Daniel Silveira

42.5k ● 36 ● 101 ● 124

7

Some things I would keep in mind. Make sure every table has a way to uniquely identify records (you will save untold hours of pain doing this). Normalize but do not join on large multi-column natural keys unless you want the whole thing to be slow. Use a numeric key that is autogenerated in the parent table instead.

Yes, think about the kinds of queries and reports you will need to run. Think about extensibility. It may seem like you wan't need more than 10 products columns in the order table but what happens when you need 11. Better to have an order table and an order detail table.

Make sure all data integrity rules are incorporated into the database. Not all data changes happen from the user interface and I've had to try to fix too many badly messed up databases because the designers figured it was OK to put all rules in the GUI.

The most critical things to consider when desiging are first how to ensure data integrity (if the data is meaningless then the database is useless) and second how to ensure performance. Do not use an object model to design a relational database unless you want bad performance.

The next most important thing is data protection and security. Users should never have direct access to the database tables. If your design requires dynamic SQL they will have to have that access. This is bad from the perspective of potential hacking in through things like SQL injection attacks, but even more importantly, it opens up your database for internal people commit fraud. Are there fields where you need to encrypt the data (credit card information, passwords, and Social Security numbers are among the items that should never be stored unencrypted). How do you plan to do that and how do you plan to audit decryption to ensure people are not decrypting when they have no need to see the data. Are there legal hoops you must go through (HIPPA and Sarbanes Oxley spring to mind)?

edited Sep 14, 2010 at 16:26

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 26, 2008 at 19:09

HLGEM

96.4k ● 15 ● 119 ● 189

4

Get a really good book on data modeling - one written by a true database developer, not a .NET developer who tries to teach you how it's done in the "real world".

The problem space of database design is simply way too large to be significantly covered in a forum like this. Despite that though, I'll give you a few personal pointers:

Listen to the above posts about normalization. NEVER denormalize because you THINK that you have to for performance reasons. You should only denormalize after you've experience actual performance issues (ideally in your QA environment, not production). Even then, consider that there may be a better way to write your queries or improve indexing first.

Constrain the data as much as possible. Columns should be NOT NULL as much as possible. Use CHECK constraints and FOREIGN KEYs wherever they should be. If you don't do this, bad data **will** get into your database and cause a lot of headaches and special case programming.

Think through your data before you actually start designing tables. Get a good handle on how your processes will flow and what data they will need to track. Often times what you think is an entity at first glance turns out to be two entities. As an example, in a system that I'm working on, the previous designer created a Member table and all of the information from their application was part of the Member table. It turns out that a Member

might want to change data that was on their application, but we still need to track what the original application looked like, so the Application is really its own entity and the Member is an entity that might initially be populated from the Application. In short, do extensive data analysis, don't just start creating tables.

Share Improve this answer

Follow

answered Sep 26, 2008 at 19:57

Tom H

47.4k ● 15 ● 89 ● 131

I love your point about 'NEVER denormalize because you THINK that you have to for performance reasons.' – Raj More Jul 22, 2009 at 13:24

4

Since there have been several posts advocating this now, I'll add one more thing...

DON'T fall into the trap of putting ID columns on all of your tables. There are many VERY good reasons why modern database design theory uses real primary keys and they aren't strictly academic reasons. I've worked with databases that included hundreds of tables, many of which were multi-million row tables, with over 1000 concurrent users and using real primary keys did not "break down".

Using ID columns on all of your tables means that you will have to do multi-table joins to traverse across the database, which becomes a big hassle. It also tends to promote sloppy database design and even beyond that

often results in problems with duplicate rows. Another issue is that when dealing with outside systems you now have to communicate these IDs around.

There are places for surrogate IDs - type code tables and conceptual tables (for example, a table of system rules could use an ID if the rules don't have real-world identifiers). Using them everywhere is a mistake IMO.

It's a long-standing debate, but that's my opinion on the matter, for what it's worth.

Share Improve this answer

Follow

answered Sep 26, 2008 at 20:12

Tom H

47.4k ● 15 ● 89 ● 131

3 It's not a debate. Surrogate keys work to make the data maintainable. "Natural Keys" or "Real Keys" impose weird restrictions that sometimes mirror the real world, but other times mirror the relational model. – S.Lott Sep 27, 2008 at 18:53

It's only "not a debate" if you're so arrogant to think that your opinion is the only valid one. As I said, I've worked with databases that measured in the hundreds of tables and the data was easily "maintainable". Any restrictions imposed are actual restrictions, or your model is the problem. – Tom H Sep 27, 2008 at 23:10

The issue isn't "can a smart person manage it". Clearly, your smart and you can manage it. The issue is that failing to put immutable surrogate keys in leads to odd, quirky restrictions on what the database can and can't do. – S.Lott Sep 28, 2008 at 1:58

1 I like to do both. I have auto-generated surrogate keys and uniquely constrained "natural" keys for almost every non-junction table. – Gilligan Sep 30, 2008 at 18:29

4

Data Is Eternal. Processing Comes and Goes.

Get the relational model to be a high-fidelity representation of the real world. This matters more than anything else.

Processing will change and evolve for years. But your data -- and the data model -- can't evolve at the same pace and with the same flexibility. You can add processing, but you can't magically add information. You don't want to delete information (but you can ignore it.)

Get the model **right**. The entities and relationships in your diagrams should make rational sense to a casual non-technical user. Even the application programming should be simple, clear and precise.

If you're struggling with the model, don't invent big, complex queries or (worse) stored procedures to work around the problems. Procedural work-arounds are a costly mistake. Understand what you have, what you want to do, and apply the YAGNI principle to pare things down to the essentials.

Share Improve this answer

Follow

answered Sep 27, 2008 at 18:59

S.Lott

391k ● 82 ● 517 ● 788

I know this has been stated, but normalization, normalization, normalization is the key. If there is an

3

instance where you feel that for whatever reason that you need to store data in a non-normalized format, don't do it. This should be handled through views or in a separate reporting database. My other key advice is to avoid text/ntext fields wherever possible.

Share Improve this answer

Follow

answered Nov 14, 2008 at 15:45

CNote

140 ● 1 ● 4

3

"Thumb rule of Databases - Down Always Beats Across!"

Examples: If you have a Customer table with columns for Mailing Address and Shipping address and Billing address... Create a separate CustomerAddress table with an Address Type

If you have a CancellationDetails table with CancellationReason01, CancellationReason02, CancellationReason03.. create a separate CancellationReason table

Share Improve this answer

Follow

answered Jul 22, 2009 at 13:32

Raj More

48k ● 34 ● 138 ● 199

2 If you have a CancellationDetails table with CancellationReason01, CancellationReason02, CancellationReason03... you have a **horrible, horible mess**. And we've got some of those in our system. – Esteban Küber Jul 22, 2009 at 13:51

2

Be practical. Keep in mind what your goals are and don't go crazy creating unnecessary complexity. I have some preferences:

- Keep the number of tables small
- prefer narrow tables over wide ones full of null values.
- Normalization is generally good

- Triggers are typically very painful

But these are a means to an end (and are contradictory in many cases and require careful balancing), the main thing is to let the requirements drive the design. Your choice of what is a separate entity, and what is part of another entity, and what is cat food (not anything whose identity you care about) depends entirely on your requirements.

Share Improve this answer Follow

edited Sep 14, 2010 at 20:33

answered Sep 13, 2010 at 20:52

Nathan Hughes
96.3k ● 20 ● 191 ● 282

1 I don't necessarily disagree with you, but I note with amusement that you recommend both few tables and small tables. This can be summarized as "store little data". I guess that's a pretty good recommendation, since it helps avoid redundancy. :-) – Jeffrey L Whitledge Sep 14, 2010 at 19:28

@Jeffrey: You're right, there is definitely a balancing act involved. things like, more joins or wider tables? – Nathan Hughes Sep 14, 2010 at 19:50

1

If you'll be looking rows up by fields other than the primary key, make sure to index them.

Share Improve this answer

answered Sep 26, 2008 at 18:47

Follow

Powerlord
88.7k ● 17 ● 129 ● 177

Good tip but not really a design issue IMO – Joe Phillips Sep 26, 2008 at 20:16

1

If you have queries that you're going to be running A LOT, make them into stored procedures. They will *almost* always run faster.

Share Improve this answer

Follow

edited Sep 26, 2008 at 18:51

Mark Biek
150k ● 54 ● 158 ● 201

answered Sep 26, 2008 at 18:45

Keng
53k ● 32 ● 84 ● 111

1 Any evidence of this? I think it's not true and I haven't found any solid benchmarks. – S.Lott Sep 27, 2008 at 18:51

This is a good practice whether it makes the query run faster or not. – Leigh Riffel Nov 14, 2008 at 21:26

1

Is it to an Object Oriented language? So try modelling your objects before the database. This will help you to focus on the model.

Share Improve this answer

Follow

answered Sep 26, 2008 at 18:56

Daniel Silveira

42.5k ● 36 ● 101 ● 124

1

Understand the requirements as much as you possibly can up front. Then design a logical schema that will only have to change if the requirements change, or if you migrate to a completely different kind of database, like one that doesn't use SQL. Then refine and extend your design into a physical design that takes into account your particular DBMS product, your volume, your load, and your speed requirements.

Learn how to normalise, but also learn when to break the normalization rules.

Share Improve this answer

Follow

answered Sep 26, 2008 at 19:08

Walter Mitty

18.9k ● 2 ● 31 ● 59

1

I strongly echo that normalization is critical, with tactical de-normalization to follow for performance or other maintainability reasons. However, if you're expecting to have more than just a few tables, I'd like to offer one caveat about normalization that will make your life a lot easier as the number of tables grows.

The caveat is to make the primary key for each table a single numeric column (appropriate for your flavor of DB). In academic normalization, the idea is to combine whatever attributes (columns) of an entity (table) so that you can uniquely identify an instance of what is being described (row), and you can end up with a multi-column composite primary key. So then whenever you migrate that composite key as a foreign key to other tables, you end up duplicating those multiple columns in *every* table that references it. That might work for you if you only have half a dozen tables. But it falls apart quickly when you go much bigger than that.

So instead of a multi-column composite primary key, go with a sequential numeric primary key even though that approach goes against some of the strict normalization teachings.

Share Improve this answer

Follow

answered Sep 26, 2008 at 19:23

Ed Lucas

654 ● 7 ● 8

1

Make sure that as much meta data as possible is encoded in the model. It should be possible to infer almost any business rule or concept from just looking at the data model.

This means, take care to pick names that reflect the reality of the users (but don't be afraid to change their perception of reality if it helps the model).

Encode all constraints you can in the database. Don't rely on the application layer to only supply sensible data. Make sure that only sensible data can exist in the first place.

Don't do aggregate data in the model. Keep the model as atomic as possible. Either aggregate on the fly or run regular aggregation jobs into aggregate tables.

Pick a good partition between schemas. Some partitioning makes sense to do with foreign keys, and some by pure physical seperation.

Share Improve this answer

Follow

edited Sep 13, 2010 at 20:42

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Oct 4, 2008 at 17:22

John Nilsson

17.3k ● 8 ● 35 ● 42

Don`t use a large set of columns as primary keys

0

Share Improve this answer

Follow

answered Sep 26, 2008 at 18:52

Daniel Silveira

42.5k ● 36 ● 101 ● 124

You'll need to have every one of those columns in any table that references the one with the multi-column primary key.
– Powerlord Sep 26, 2008 at 19:13

That's not necessarily true, as you can always use alternate keys for your foreign key (at least in MS SQL). It's also not a bad thing to be carrying those columns around to tables that have FKs into the main table. The extra space is usually unimportant compared to other advantages that you gain.
– Tom H Sep 26, 2008 at 19:44

In the program, it is kind of tedious to have to carry out allll these columns around in variables in your code.
– Daniel Silveira Sep 27, 2008 at 13:54

0

Remember that normalisation is only relative to what you are modelling. Perhaps you are modelling a collection of objects in your domain. Maybe you are recording a series of events, in which data are repeated because the same data happen to apply at more than one time. Don't mix up the two things.

Share Improve this answer

Follow

answered Sep 26, 2008 at 19:14

Marcin

49.8k ● 18 ● 132 ● 205

0

As much as you can make primary key a sequence generated number.

Share Improve this answer

Follow

answered Sep 13, 2010 at 20:57

fastcodejava

41k ● 30 ● 138 ● 191

0

I agree that knowing about your data is good and normalizing.

Something else I would suggest is to keep very large text fiels in a separate table. For example, if you have a contract you might want to keep a lot of the information about the contract in one table but keep the legal (and very large) document in a separate table. Just put in an index from the main table into the legal document.

Share Improve this answer

Follow

edited Sep 14, 2010 at 16:55

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 26, 2008 at 19:23

Ben

2,801 ● 6 ● 34 ● 45

Why? I mean, don't tell me that you are doing SELECT * FROM Table :) – Luk Oct 6, 2008 at 12:20

This advice seems implementation dependant (on the DB engine end). I don't think it's relevant on PostgreSQL or Oracle, but don't know. There *is* a SO question somewhere... – Esteban Küber Jul 22, 2009 at 13:53

0

I'd say an important thing to keep in mind is that the structure may change. So don't design yourself into a corner. Make sure whatever you do leaves you some "room" and even an avenue to migrate the data into a different structure some day.

Share Improve this answer

Follow

edited Sep 14, 2010 at 16:57

Peter Mortensen

31.6k ● 22 ● 109 ● 133

answered Sep 27, 2008 at 18:31

phatduckk

1,795 ● 1 ● 14 ● 17