What's wrong with foreign keys?

Asked 16 years, 3 months ago Modified 7 months ago

Viewed 133k times

282

I remember hearing Joel Spolsky mention in podcast 014 that he'd barely ever used a foreign key (if I remember correctly). However, to me they seem pretty vital to avoid duplication and subsequent data integrity problems throughout your database.

Do people have some solid reasons as to why (to avoid a discussion in lines with Stack Overflow principles)?

"I've yet to have a reason to create a foreign key, so this might be my first reason to actually set up one."

database database-design foreign-keys referential-integrity data-integrity

Share

Improve this question

Follow

edited Jun 3, 2023 at 17:13

community wiki
10 revs, 6 users 31%
Peter Mortensen

11 I don't think Joel doesn't use FKs, it's just that he doesn't make the database enforce them. Logically, they are still FKs! – Daren Thomas Sep 17, 2008 at 14:20

7 He says he doesn't use foreign keys, but I agree with Daren that what he means is he doesn't use foreign key CONSTRAINTS. A column in one table whose values are supposed to be taken from the primary/unique key of another table ARE foreign keys, whether you add the constraint or not. – Tony Andrews Oct 23, 2008 at 12:35

27 ...Generally it is foolish not to add the constraint: it ENSURES integrity at all times, even if there is a bug in the application code or if your are working behind the scenes doing a data "fix". – Tony Andrews Oct 23, 2008 at 12:36

2 @TonyAndrews: By that reasoning you wouldn't need to set up a PRIMARY KEY either, as long as your application code will make sure that all records are unique. Sorry, but that's just rubbish. Constraints let the database keep it's data consistent *all by itself*, so why you wouldn't want to make use of that is beyond me. – DanMan Dec 25, 2012 at 12:03

7 @DanMan, don't know where you gained the impression I think that. I actually say above "Generally it is foolish not to add the constraint: it ENSURES integrity at all times" – Tony Andrews Dec 25, 2012 at 19:19

38 Answers

Sorted by: Highest score (default)

1 2 Next

388

Reasons to use Foreign Keys:

- you won't get Orphaned Rows

- you can get nice "on delete cascade" behavior, automatically cleaning up tables
- knowing about the relationships between tables in the database helps the Optimizer plan your queries for most efficient execution, since it is able to get better estimates on join cardinality.
- FKs give a pretty big hint on what statistics are most important to collect on the database, which in turn leads to better performance
- they enable all kinds of auto-generated support -- ORMs can generate themselves, visualization tools will be able to create nice schema layouts for you, etc.
- someone new to the project will get into the flow of things faster since otherwise implicit relationships are explicitly documented

Reasons not to use Foreign Keys:

- you are making the DB work extra on every CRUD operation because it has to check FK consistency. This can be a big cost if you have a lot of churn
- by enforcing relationships, FKs specify an order in which you have to add/delete things, which can lead to refusal by the DB to do what you want. (Granted, in such cases, what you are trying to do is create an Orphaned Row, and that's not usually a good thing). This is especially painful when you are doing large batch updates, and you load up one table before

another, with the second table creating consistent state (but should you be doing that sort of thing if there is a possibility that the second load fails and your database is now inconsistent?).

- sometimes you know beforehand your data is going to be dirty, you accept that, and you want the DB to accept it
- you are just being lazy :-)

I think (I am not certain!) that most established databases provide a way to specify a foreign key that is not enforced, and is simply a bit of metadata. Since non-enforcement wipes out every reason not to use FKs, you should probably go that route if any of the reasons in the second section apply.

Share Improve this answer Follow

edited Sep 18, 2018 at 7:10

community wiki
2 revs, 2 users 86%
SquareCog

15 Good list! The DBMs won't check consistency for "R" part of CRUD, so I'd take that part out. Also, it's probably a wash because in your app you do the same thing the DBMS does: you'll check and make sure the parent ID is valid before CRD and that is actually slower than having the DBMs do it!
– Matt Rogish Sep 17, 2008 at 14:40

7 What if someone deletes the parent while you're inserting children? Right now when I submit "add comment" -- if you

have already deleted your answer, this comment is now an orphan. FKs would've prevented it. Also, I could just change the parentID to be anything I want. Someone needs to check. :) – Matt Rogish Sep 17, 2008 at 16:16

7 Precisely -- it should be the DB's job, since it's the only one that can guarantee transactionality in the face of multiple concurrent clients. – SquareCog Sep 17, 2008 at 16:38

4 +1 Excellent answer -- the second reason not to use FK constraints could be thought of "makes it harder to break consistency" which actually sounds like a *good* thing! – Bill Karwin Sep 1, 2009 at 23:46

10 The benefits of using foreign keys FAR outweighs any benefits of not using them in my opinion. – Nick Bedford Mar 27, 2015 at 2:16

91

This is an issue of upbringing. If somewhere in your educational or professional career you spent time feeding and caring for databases (or worked closely with talented folks who did), then the fundamental tenets of entities and relationships are well-ingrained in your thought process. Among those rudiments is how/when/why to specify keys in your database (primary, foreign and perhaps alternate). It's second nature.

If, however, you've not had such a thorough or positive experience in your past with RDBMS-related endeavors, then you've likely not been exposed to such information. Or perhaps your past includes immersion in an environment that was vociferously anti-database (e.g., "those DBAs are idiots - we few, we chosen few java/c# code slingers will save the day"), in which case you might

be vehemently opposed to the arcane babblings of some dweeb telling you that FKs (and the constraints they can imply) really are important if you'd just listen.

Most everyone was taught when they were kids that brushing your teeth was important. Can you get by without it? Sure, but somewhere down the line you'll have less teeth available than you could have if you had brushed after every meal. If moms and dads were responsible enough to cover database design as well as oral hygiene, we wouldn't be having this conversation. :-)

Share Improve this answer Follow

answered Sep 17, 2008 at 16:55

community wiki
Ed Lucas

76 I'm going to use the distilled "foreign keys are like brushing your teeth: go ahead, do without it, but careful when you smile" – Mark Sowul May 3, 2012 at 15:27

8 I personally find that the principles of RDBMS' are much simpler and much more well defined than those of oral hygiene – Ali Gangji May 21, 2014 at 18:15

10 years in future, I'm sure going to have this database design talk with my son/daughter so he/she don't mess up and end up being reason for next wall street crash because of a database issue. – user2286243 Feb 14, 2018 at 13:09

55

I'm sure there are plenty of applications where you can get away with it, but it's not the best idea. You can't always count on your application to properly manage your database, and frankly managing the database should not be of very much concern to your application.

If you are using a **relational** database then it seems you ought to have some **relationships** defined in it. Unfortunately this attitude (you don't need foreign keys) seems to be embraced by a lot of application developers who would rather not be bothered with silly things like data integrity (but need to because their companies don't have dedicated database developers). Usually in databases put together by these types you are lucky just to have primary keys ;)

Share Improve this answer

Follow

edited Apr 18, 2019 at 21:20

community wiki
3 revs, 3 users 67%
AlexCuse

33 I really don't get people who don't have FKs in their database. The last time I worked with someone who didn't have it, he said "no, we enforce that in the application." Except I did a survey of all the customer databases and found most of them did have orphans... – ErikE Sep 13, 2010 at 0:34

1 That usually seems to be the case. I think you could get away with enforcing ONLY in the database (as long as your

users don't mind runtime exceptions) but to have both is really the only way to go. – AlexCuse Sep 13, 2010 at 1:18

Its all in the transcripts / Atwood's response "Atwood: ...based on the foreign keys that you set in the indexes, they figure it out ... Spolsky: [laughs] Assuming that you do that. Atwood: Well, assuming that you set your database up properly..." – MemeDeveloper Mar 22, 2013 at 2:48

8 Database are not called *relational* because of the *relationships* between tables (EVERY kind of database has some kind of relationships between entities!), but because tables themselves are *relations*, in math terms. See Wikipedia. – Massimiliano Kraus Nov 6, 2018 at 9:55

I think what most people dont really understand is that often you build applications that do not perform DELETE. Your data simply gets "nullified" by some flag. Those applications have their own ways of dealing with what to do on DELETE, key is there for implicit reference only and performance is also a component in such situations. Handwaving this as bad design is acting silly. – mAm Jan 17, 2022 at 20:27

Foreign keys are **essential** to any relational database model.

42

Share Improve this answer Follow

edited Sep 17, 2008 at 15:42

community wiki
3 revs
Galwegian

59 The model, yes. The implemention, not essential, just probably useful. – dkretz Mar 1, 2009 at 5:01

4 Sorry, but the main reason why app developers don't use object database management systems (aka NoSQL databases!) more widely is because of the investment in RDBMSs. Most of the time the database (not the database management system) is a mid-tier object model often involving distributed caches. This is where deletion cascading, ownership and synchronisation of changes has to happen anyway. The RDBMS is used primarily for persistence of this object model, and usually after a painstaking and practically worthless ORM exercise. Relation models are most of the time not necessary! – Sentinel Sep 26, 2012 at 7:46

2 no, foreign keys are not compulsary to indicate "relational" – Prashant Pugalia Apr 10, 2013 at 8:28

1 This doesn't really explain much. – Nae Sep 19, 2018 at 8:45

31

I always use them, but then I make databases for financial systems. The database is the critical part of the application. If the data in a financial database isn't totally accurate then it really doesn't matter how much effort you put into your code/front-end design. You're just wasting your time.

There's also the fact that multiple systems generally need to interface directly with the database - from other systems that just read data out (Crystal Reports) to systems that insert data (not necessarily using an API I've designed; it may be written by a dull-witted manager who

has just discovered VBScript and has the SA password for the SQL box). If the database isn't as idiot-proof as it can possibly be, well - bye bye database.

If your data is important, then yes, use foreign keys, create a suite of stored procedures to interact with the data, and make the toughest DB you can. If your data isn't important, why are you making a database to begin with?

Share Improve this answer Follow

answered Sep 17, 2008 at 13:40

community wiki
Ant

3 Nice insight. I would argue that the data is this important for every application that actually gets used. The only thing that differs is the consequences of corrupt data. They are high for your kind of app... – Jay Godse Jun 30, 2010 at 12:19

25

Update: I always use foreign keys now. My answer to the objection "they complicated testing" is "write your unit tests so they don't need the database at all. Any tests that use the database should use it properly, and that includes foreign keys. If the setup is painful, find a less painful way to do the setup."

Foreign keys complicate automated testing

Suppose you're using foreign keys. You're writing an automated test that says "when I update a financial account, it should save a record of the transaction." In this test, you're only concerned with two tables: `accounts` and `transactions`.

However, `accounts` has a foreign key to `contracts`, and `contracts` has a fk to `clients`, and `clients` has a fk to `cities`, and `cities` has a fk to `states`.

Now the database will not allow you to run your test without setting up data in four tables that aren't related to your test.

There are at least two possible perspectives on this:

- "That's a good thing: your test should be realistic, and those data constraints will exist in production."
- "That's a bad thing: you should be able to unit test pieces of the system without involving other pieces. You can add integration tests for the system as a whole."

It may also be possible to temporarily turn off foreign key checks while running tests. MySQL, at least, supports this.

 edited Mar 12, 2019 at 12:52

Follow

community wiki
4 revs
Nathan Long

1 I find myself usually going for the middle path here: I use FKs, then I write unit-test helper methods that set up the DB to support various testing scenarios, e.g. a helper method to populate "cities" and "states" for any tests that need those tables populated. – joelpt Aug 14, 2014 at 16:10

You should have used link tables between the non-related entities perhaps. Or go further - separate DBS: consider the situation in a Service Oriented Architecture, or Microservice, where each element (clients, accounts, transactions) are different systems, with different DBs. No FKs between them as all. In this case, FKs should be used to prevent orphaned data in subtables for each type of data. – JeeBee Sep 26, 2014 at 10:35

3 There are also DBMS that allow to constraints to *deferred* so that they are only checked when you commit the whole transaction, so the order of insert, update, delete doesn't matter – user330315 Nov 23, 2014 at 14:37

2 If you are testing an update from a business layer, your development environment should have the FK present. When you update your record, you should have the columns values you need for the update to succeed. Otherwise, IMHO your test is not valid. – KeyOfJ May 6, 2015 at 13:42

4 Your database shouldn't even be involved in your unit tests, you should mock them. On integration testing they would be involved but any problems there due to foreign keys is something your users also will run into unlesss you fix it. – Andreas Bergström Feb 22, 2017 at 3:04

17

"They can make deleting records more cumbersome - you can't delete the "master" record where there are records in other tables where foreign keys would violate that constraint."

It's important to remember that the SQL standard defines actions that are taken when a foreign key is deleted or updated. The ones I know of are:

- `ON DELETE RESTRICT` - Prevents any rows in the other table that have keys in this column from being deleted. This is what Ken Ray described above.
- `ON DELETE CASCADE` - If a row in the other table is deleted, delete any rows in this table that reference it.
- `ON DELETE SET DEFAULT` - If a row in the other table is deleted, set any foreign keys referencing it to the column's default.
- `ON DELETE SET NULL` - If a row in the other table is deleted, set any foreign keys referencing it in this table to null.
- `ON DELETE NO ACTION` - This foreign key only marks that it is a foreign key; namely for use in OR mappers.

These same actions also apply to `ON UPDATE`.

The default seems to depend on which `sql` server you're using.

Share Improve this answer

Follow

edited Sep 19, 2018 at 15:24

community wiki
2 revs, 2 users 79%
Powerlord

15

@imphasing - this is exactly the kind of mindset that causes maintenance nightmares.

Why oh why would you ignore declarative referential integrity, where the data can be **guaranteed** to be at least consistent, in favour of so called "software enforcement" which is a weak preventative measure at best.

Share Improve this answer

Follow

answered Sep 17, 2008 at 13:36

community wiki
Ed Guiness

Because the developers involved have never tackled a problem that demands a non-trivial, normalized, relational model. Many problems don't, especially the sort that abound in web/"social media" type programming that is all the craze today. If whatever dribbles out the back end of an ORM framework satisfies the problem in alpha, it is unlikely for anyone to think much more about data modeling. Many such problems are just as easily handled by K/V stores, document databases or straight object serialization. – zxq9 Sep 9, 2014 at 1:38

12

There's one good reason not to use them: **If you don't understand their role or how to use them.**

In the wrong situations, foreign key constraints can lead to waterfall replication of accidents. If somebody removes the wrong record, undoing it can become a mammoth task.

Also, conversely, when you need to remove something, if poorly designed, constraints can cause all sorts of locks that prevent you.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:34

community wiki
Kent Fredric

9 Deleting a row in production without backup is not a valid argument. If you don't understand them, you should consider

learning about it instead of omitting it. – Guillaume Mar 31, 2015 at 13:31

3 @Guillaume I think his answer was a bit sarcastic, not to be taken literally: if you don't understand them, then don't use them. But of course you *should* both understand and use them. – BenMorel Feb 14, 2017 at 17:46

^ This. They are useful tools, but in the hands of a novice, they are dangerous tools. – Kent Fredric Feb 15, 2017 at 15:24

I wouldn't call FK dangerous with default settings, unless applied with modifiers like: `ON DELETE CASCADE` in such case deleting some row might trigger cascading.
– Tom Raganowicz May 29 at 12:37

10

There are no **good** reasons *not* to use them... unless orphaned rows aren't a big deal to you I guess.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:27

community wiki
Matt Rogish

11 Why are orphaned rows a big deal? – Seun Osewa Sep 16, 2010 at 1:35

2 What about multithreading? They can cause a multithreading nightmare in certain situations. In a complex application with multiple threads writing the database that may be encountering objects which need to reference each other, it is better to control referential integrity in the business logic---

particularly if the tables are going to go static afterwards. – Keith Pinson Aug 10, 2011 at 14:26

I agree. Also, I prefer to have ophan rows which I can recover later in time, than to have them discarded mercilessly. – PedroD Sep 21, 2016 at 8:19

10

"Before adding a record, check that a corresponding record exists in another table" is business logic.

Here are some reasons you don't want this in the database:

1. If the business rules change, you have to change the database. The database will need to recreate the index in a lot of cases and this is slow on large tables. (Changing rules include: allow guests to post messages or allow users to delete their account despite having posted comments, etc).
2. Changing the database is not as easy as deploying a software fix by pushing the changes to the production repository. We want to avoid changing the database structure as much as possible. The more business logic there is in the database the more you increase the chances of needing to change the database (and triggering re-indexing).
3. TDD. In unit tests you can substitute the database for mocks and test the functionality. If you have any business logic in your database, you are not doing complete tests and would need to either test with the database or replicate the business logic in code for

testing purposes, duplicating the logic and increasing the likelyhood of the logic not working in the same way.

4. Reusing your logic with different data sources. If there is no logic in the database, my application can create objects from records from the database, create them from a web service, a json file or any other source. I just need to swap out the data mapper implementation and can use all my business logic with any source. If there is logic in the database, this isn't possible and you have to implement the logic at the data mapper layer or in the business logic. Either way, you need those checks in your code. If there's no logic in the database I can deploy the application in different locations using different database or flat-file implementations.

Share Improve this answer Follow

edited Apr 28 at 7:44

community wiki
3 revs, 3 users 92%
Tom B

8

From my experience its always better to avoid using FKs in Database Critical Applications. I would not disagree with guys here who say FKs is a good practice but its not practical where the database is huge and has huge CRUD operations/sec. I can share without naming ... one

of the biggest investment bank of doesn't have a single FK in databases. These constrains are handled by programmers while creating applications involving DB. The basic reason is when ever a new CRUD is done it has to effect multiple tables and verify for each inserts/updates, though this won't be a big issue for queries affecting single rows but it does create a huge latency when you deal with batch processing which any big bank has to do as daily tasks.

Its better to avoid FKs but its risk has to be handled by programmers.

Share Improve this answer Follow

answered Oct 13, 2014 at 10:02

community wiki
Rachit

12 I don't think development practices at large banks set the golden standard. – Adriaan Koster Feb 27, 2015 at 12:25

Absolutely. Automatic operations are always risky, even when you know exactly (not many do) what's happening in the background. Foreign keys are like a safety net, but they carry a non-neglectable overhead and they don't give as much in return. Databases have to be in (technically) an "inconsistent state" every now and then, temporarily, e.g. when maintenance processes are running. The intermittent layer between application and db should be able to handle this. It's not a good practice to restrict yourself, unless you're uncertain what you're doing and why. – dkellner Oct 5, 2021 at 19:46

4

Bigger question is: would you drive with a blindfold on? That’s how it is if you develop a system without referential constraints. Keep in mind, that business requirements change, application design changes, respective logical assumptions in the code changes, logic itself can be refactored, and so on. In general, constraints in databases are put in place under contemporary logical assumptions, seemingly correct for particular set of logical assertions and assumptions.

Through the lifecycle of an application, referential and data checks constraints police data collection via the application, especially when new requirements drive logical application changes.

To the subject of this listing - a foreign key does not by itself "improve performance", nor does it "degrade performance" significantly from a standpoint of real-time transaction processing system. However, there is an aggregated cost for constraint checking in HIGH volume "batch" system. So, here is the difference, real-time vs. batch transaction process; batch processing - where aggreated cost, incured by constraint checks, of a sequentially processed batch poses a performance hit.

In a well designed system, data consistency checks would be done "before" processing a batch through (nevertheless, there is a cost associated here also); therefore, foreign key constraint checks are not required during load time. In fact all constraints, including foreign

key, should be temporarily disabled till the batch is processed.

QUERY PERFORMANCE - if tables are joined on foreign keys, be cognizant of the fact that foreign key columns are NOT INDEXED (though the respective primary key is indexed by definition). By indexing a foreign key, for that matter, by indexing any key, and joining tables on indexed helps with better performances, not by joining on non-indexed key with foreign key constraint on it.

Changing subjects, if a database is just supporting website display/rendering content/etc and recording clicks, then a database with full constraints on all tables is over kill for such purposes. Think about it. Most websites don’t even use a database for such. For similar requirements, where data is just being recorded and not referenced per say, use an in-memory database, which does not have constraints. This doesn’t mean that there is no data model, yes logical model, but no physical data model.

edited Oct 5, 2009 at 6:58

community wiki
3 revs, 2 users 56%
jasbir L

Well, I don't know why inserting 3 double newlines in place of blanks and changing two words counts as '67% is Jonathan Leffler', but I don't think I had done anything like that much

work on it. The main text was contributed by @jay (user 183837). – Jonathan Leffler Oct 4, 2009 at 6:29

I just assumed that paragrahps won't work here as is the case in most other sites. So, I put it all as one, using bolds for flow change. – jasbir L Oct 5, 2009 at 6:42

3

I agree with the previous answers in that they are useful to mantain data consistency. However, there was an interesting post by Jeff Atwood some weeks ago that discussed the pros and cons of normalized and consistent data.

In a few words, a denormalized database can be faster when handling huge amounts of data; and you may not care about precise consistency depending on the application, but it forces you to be much more careful when dealing with data, as the DB won't be.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:33

community wiki
Santiago Palladino

Jeff makes some good points. However, Dan Chak in "Enterprise Rails" shows a way to design cache tables that are essentially a de-normalized copy of the data. Queries run fast, and if the table doesn't have to be refreshed, it works well. I find that if your data drives the behaviour (e.g. application state) of your application, you need data to be normalized as much as possible because otherwise

inconsistent data leads to inconsistent application behaviour.
– Jay Godse Jun 30, 2010 at 12:34

A denormalised data *warehouse* may be of use when *reading* large volumes of data *on consistent, anticipated access paths*. In all other scenarios, this is a dangerous fallacy.
– Peter Wone Mar 27, 2013 at 0:46

3

The Clarify database is an example of a commercial database that has no primary or foreign keys.

http://www.geekinterview.com/question_details/18869

The funny thing is, the technical documentation goes to great lengths to explain how tables are related, what columns to use to join them etc.

In other words, they **could** have joined the tables with explicit declarations (DRI) but they **chose not to**.

Consequently, the Clarify database is full of inconsistencies and it underperforms.

But I suppose it made the developers job easier, not having to write code to deal with referential integrity such as checking for related rows before deleting, adding.

And that, I think, is the main benefit of not having foreign key constraints in a relational database. It makes it easier to develop, at least that is from a devil-may-care point of view.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:34

community wiki
Ed Guiness

The code to handle a failed referential integrity check is a lot smaller than the code to handle inconsistent data.
– Jay Godse Jun 30, 2010 at 12:22

@Jay agree! Don't think I'm advocating this approach.
– Ed Guiness Jun 30, 2010 at 18:09

3

If you are absolutey sure, that the one underlying database system will not change in the future, I would use foreign keys to ensure data integrity.

But here is another very good real-life reason not to use foreign keys at all:

You are developing a product, which should support different database systems.

If you are working with the Entity Framework, which is able to connect to many different database systems, you may also want to support "open-source-free-of-charge" serverless databases. Not all of these databases may support your foreign key rules (updating, deleting rows...).

This can lead to different problems:

1.) You may run into errors, when the database structure is created or updated. Maybe there will only be silent errors, because your foreign keys are just ignored by the database system.

2.) If you rely on foreign keys, you will propably make less or even no data integrity checks in your business logic. Now, if the new database system does not support these foreign key rules or just behaves in a different way, you have to rewrite your business logic.

You may ask: Who needs different database systems? Well, not everybody can afford or wants a full blown SQL-Server on his machine. This is software, which needs to be maintained. Others already have invested time and money in some other DB system. Serverless database are great for small customers on only one machine.

Nobody knows, how all of these DB systems behave, but your business logic, with integrity checks, always stays the same.

answered Sep 14, 2015 at 13:16

community wiki
Michael

They can make deleting records more cumbersome - you can't delete the "master" record where there are records in other tables where foreign keys would violate that

2

constraint. You can use triggers to have cascading deletes.

If you chose your primary key unwisely, then changing that value becomes even more complex. For example, if I have the PK of my "customers" table as the person's name, and make that key a FK in the "orders" table", if the customer wants to change his name, then it is a royal pain... but that is just shoddy database design.

I believe the advantages in using fireign keys outweighs any supposed disadvantages.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:31

community wiki
Ken Ray

6 I tend to rarely delete things anyhow. Just mark have a "Visible/active" bit. – Dana Sep 17, 2008 at 13:44

+1 for "I believe the advantages in using fireign keys outweighs any supposed disadvantages" – Ian Boyd Jul 17, 2009 at 13:48

2 You never, ever *change* a primary key's value. You *delete* the whole row and *recreate* it differently. If you think you need to *change* it, your schema is flawed. – DanMan Dec 25, 2012 at 12:15

Changing the customer name would not be a pain at all IF your foreign key is set on the CustomerId (PK). in the orders table. The only way it would be a pain is if the FK was set on

CustomerName which should never be the case. IMHO
– KeyOfJ May 6, 2015 at 13:46

2

Verifying foreign key constraints takes some CPU time, so some folks omit foreign keys to get some extra performance.

Share Improve this answer Follow

answered Sep 17, 2008 at 13:34

community wiki
remonedo

6 How much CPU time is spent removing duplicate and inconsistent data? – Ed Guiness Sep 17, 2008 at 13:49

Yeah, this is true. On a system I work on we have to insert 10 - 40 gigs of data at a time into a database and FK performance with and without is visible in total time it takes.
– Paul Mendoza Jan 21, 2009 at 18:21

2

Additional Reason to use Foreign Keys: - Allows greater reuse of a database

Additional Reason to NOT use Foreign Keys: - You are trying to lock-in a customer into your tool by reducing reuse.

Share Improve this answer Follow

answered Sep 17, 2008 at 15:17

community wiki
Dan

2

I know only Oracle databases, no other ones, and I can tell that Foreign Keys are essential for maintaining data integrity. Prior to inserting data, a data structure needs to be made, and be made correctlty. When that is done - and thus all primary AND foreign keys are created - the work is done !

Meaning : orphaned rows ? No. Never seen that in my life. Unless a bad programmer forgot the foreign key, or if he implemented that on another level. Both are - in context of Oracle - huge mistakes, which will lead to data duplication, orphan data, and thus : data corruption. I can't imagine a database without FK enforced. It looks like chaos to me. It's a bit like the Unix permission system : imagine that everybody is root. Think of the chaos.

Foreign Keys are essential, just like Primary Keys. It's like saying : what if we removing Primary Keys ? Well, total chaos is going to happen. That's what. You may not move the primary or foreign key responsibility to the programming level, it must be at the data level.

Drawbacks ? Yes, absolutely ! Because on insert, a lot more checks are going to be happening. But, if data integrity is more important than performance, it's a no-brainer. The problem with performance on Oracle is more related to indexes, which come with PK and FK's.

Share Improve this answer Follow

answered Nov 23, 2014 at 14:01

community wiki
tvCa

1

The argument I have heard is that the front-end should have these business rules. Foreign keys "add unnecessary overhead" when you shouldn't be allowing any insertions that break your constraints in the first place. Do I agree with this? No, but that is what I have always heard.

EDIT: My guess is he was referring to *foreign key constraints*, not foreign keys as a concept.

Share Improve this answer Follow

edited Sep 17, 2008 at 13:39

community wiki
2 revs
lordscarlet

Nope. He doesn't like actual keys! – ljs Sep 17, 2008 at 13:42

That amazes me. There is a big difference between not liking foreign key constraints and not liking foreign keys. I'm not sure how you have a relational database without them.
– Doug Moore Sep 17, 2008 at 15:20

Yes I was shocked when I heard him. He might have been unintentionally tongue-in-cheek however; perhaps he will post here and clarify at some stage :-) – ljs Sep 17, 2008 at 20:19

1

To me, if you want to go by the ACID standards, it is critical to have foreign keys to ensure referential integrity.

Share Improve this answer Follow

answered Sep 17, 2008 at 14:13

community wiki
CodeRot

1

I have to second most of the comments here, Foreign Keys are necessary items to ensure that you have data with integrity. The different options for ON DELETE and ON UPDATE will allow you to get around some of the "down falls" that people mention here regarding their use.

I find that in 99% of all my projects I will have FK's to enforce the integrity of the data, however, there are those rare occasions where I have clients that MUST keep their old data, regardless of how bad it is....but then I spend a lot of time writing code that goes in to only get the valid data anyway, so it becomes pointless.

Share Improve this answer Follow

answered Sep 17, 2008 at 14:17

community wiki
Mitchel Sellers

1

How about maintainability and constancy across application life cycles? Most data has a longer lifespan than the applications that make use of it. Relationships and data integrity are much too important to leave to the hope that the next dev team gets it right in the app code. If you haven't worked on a db with dirty data that doesn't respect the natural relationships, you will. The importance of data integrity will then become very clear.

Share Improve this answer Follow

answered Sep 17, 2008 at 16:08

community wiki

1

I also think that foreign keys are a necessity in most databases. The only drawback (besides the performance hit that comes with having enforced consistence) is that having a foreign key allows people to write code that assumes there is a functional foreign key. That should never be allowed.

For example, I've seen people write code that inserts into the referenced table and then attempts inserts into the referencing table without verifying the first insert was

successful. If the foreign key is removed at a later time, that results in an inconsistent database.

You also don't have the option of assuming a specific behavior on update or delete. You still need to write your code to do what you want regardless of whether there is a foreign key present. If you assume deletes are cascaded when they are not, your deletes will fail. If you assume updates to the referenced columns are propogated to the referencing rows when they are not, your updates will fail. For the purposes of writing code, you might as well not have those features.

If those features are turned on, then your code will emulate them anyway and you'll lose a little performance.

So, the summary.... Foreign keys are essential if you need a consistent database. Foreign keys should never be assumed to be present or functional in code that you write.

Share Improve this answer Follow

answered Sep 17, 2008 at 16:23

community wiki
Eric

1

I echo the answer by Dmitriy - very well put.

For those who are worried about the performance overhead FK's often bring, there's a way (in Oracle) you

can get the query optimiser advantage of the FK constraint without the cost overhead of constraint validation during insert, delete or update. That is to create the FK constraint with the attributes RELY DISABLE NOVALIDATE. This means the query optimiser ASSUMES that the constraint has been enforced when building queries, without the database actually enforcing the constraint. You have to be very careful here to take the responsibility when you populate a table with an FK constraint like this to make absolutely sure you don't have data in your FK column(s) that violate the constraint, as if you do so you could get unreliable results from queries that involve the table this FK constraint is on.

I usually use this strategy on some tables in my data mart schema, but not in my integrated staging schema. I make sure the tables I am copying data from already have the same constraint enforced, or the ETL routine enforces the constraint.

Share Improve this answer Follow

answered Sep 17, 2008 at 16:58

community wiki
Mike McAllister

1

Many of the people answering here get too hung up on the importance of referential integrity implemented via referential constraints. Working on large databases with referential integrity just does not perform well. Oracle

seems particularly bad at cascading deletes. My rule of thumb is that applications should never update the database directly and should be via a stored procedure. This keeps the code base inside the database, and means that the database maintains its integrity.

Where many applications may be accessing the database, problems do arise because of referential integrity constraints but this is down to a control.

There is a wider issue too in that, application developers may have very different requirements that database developers may not necessarily be that familiar with.

Share Improve this answer Follow

answered Mar 26, 2013 at 17:55

community wiki
Zak

5 "applications should never update the database directly and should be via a stored procedure. This keeps the code base inside the database, and means that the database maintains its integrity." <- There's an assumption here that logic in stored procedures can't violate data integrity, which is just plain wrong. – Tim Gautier Aug 8, 2014 at 16:27

0

I have heard this argument too - from people who forgot to put an index on their foreign keys and then complained that certain operations were slow (because constraint checking could take advantage of any index). So to sum

up: There is no good reason not to use foreign keys. All modern databases support cascaded deletes, so...

Share Improve this answer Follow

answered Sep 17, 2008 at 13:38

community wiki
Arno

9 I believe that the real reason FKs constraints are not used by some (most, from my perspective) is sheer laziness under the pretense that they can defend their laziness with their performance savings argument. I firmly believe that the vast majority of the Stupidity Expense our company incurs is due to lack enforcement of FK constraints and the ripple effect that this has through a company. Lack of Unique Keys is the other thing that drives me nuts next to 2000+ line stored procedures with 12 levels of nested IFs and random indenting, but I'll stop now. – Chad May 14, 2009 at 22:15

0

One time when an FK might cause you a problem is when you have historical data that references the key (in a lookup table) even though you no longer want the key available.

Obviously the solution is to design things better up front, but I am thinking of real world situations here where you don't always have control of the full solution.

For example: perhaps you have a look up table `customer_type` that lists different types of customers - lets say you need to remove a certain customer type, but (due to business restraints) aren't able to update the

client software, and nobody invisaged this situation when developing the software, the fact that it is a foreign key in some other table may prevent you from removing the row even though you know the historical data that references it is irrelevant.
After being burnt with this a few times you probably lean away from db enforcement of relationships.
(I'm not saying this is good - just giving a reason why you may decide to avoid FKs and db contraints in general)

Share Improve this answer Follow

answered Sep 17, 2008 at 13:55

community wiki
hamishmcn

> If I understand what you are trying to say, I think my answer to that would be to logically delete the record in the lookup table or archive the no longer relevant historical data and archive the lookup record too. – Chad Jul 1, 2010 at 4:44

0

I'll echo what Dmitriy said, but adding on a point.

I worked on a batch billing system that needed to insert large sets of rows on 30+ tables. We weren't allowed to do a data pump (Oracle) so we had to do bulk inserts. Those tables had foreign keys on them, but we had already ensured that they were not breaking any relationships.

Before insert, we disable the foreign key constraints so that Oracle doesn't take forever doing the inserts. After the insert is successful, we re-enable the constraints.

PS: In a large database with many foreign keys and child row data for a single record, sometimes foreign keys can be bad, and you may want to disallow cascading deletes. For us in the billing system, it would take too long and be too taxing on the database if we did cascading deletes, so we just mark the record as bad with a field on the main driver (parent) table.

answered Sep 17, 2008 at 16:08

community wiki
typicalrunt