Database triggers

Asked 16 years, 4 months ago Modified 11 years, 1 month ago

Viewed 6k times

18

In the past I've never been a fan of using triggers on database tables. To me they always represented some "magic" that was going to happen on the database side, far far away from the control of my application code. I also wanted to limit the amount of work the DB had to do, as it's generally a shared resource and I always assumed triggers could get to be expensive in high load scenarios.

That said, I have found a couple of instances where triggers have made sense to use (at least in my opinion they made sense). Recently though, I found myself in a situation where I sometimes might need to "bypass" the trigger. I felt really guilty about having to look for ways to do this, and I still think that a better database design would alleviate the need for this bypassing. Unfortunately this DB is used by mulitple applications, some of which are maintained by a very uncooperative development team who would scream about schema changes, so I was stuck.

What's the general consesus out there about triggers? Love em? Hate em? Think they serve a purpose in some scenarios? Do think that having a need to bypass a trigger means that you're "doing it wrong"?

sql-server database-design triggers

Share

Improve this question

Follow

edited Aug 18, 2008 at 0:05

Vaibhav

11.4k ● 11 ● 53 ● 71

asked Aug 18, 2008 at 0:01

Jesse Taber

2,386 ● 3 ● 23 ● 33

12 Answers

Sorted by: Highest score (default)

12

Triggers are generally used incorrectly, introduce bugs and therefore should be avoided. Never design a trigger to do integrity constraint checking that crosses rows in a table (e.g "the average salary by dept cannot exceed X).

Tom Kyte, VP of Oracle has indicated that he would prefer to remove triggers as a feature of the Oracle database because of their frequent role in bugs. He knows it is just a dream, and triggers are here to stay, but if he could he would remove triggers from Oracle, he would (along with the WHEN OTHERS clause and autonomous transactions).

> Can triggers be used correctly? Absolutely.
>
> The problem is - they are not used correctly in so many cases that I'd be willing to give up any

> perceived benefit just to get rid of the abuses (and bugs) caused by them. - Tom Kyte

Share Improve this answer Follow

edited Oct 14, 2008 at 16:56

answered Aug 18, 2008 at 3:23

Brian

13.6k ● 11 ● 59 ● 83

9

Think of a database as a great big object - after each call to it, it ought to be in a logically consistent state.

Databases expose themselves via tables, and keeping tables and rows consistent can be done with triggers. Another way to keep them consistent is to disallow direct access to the tables, and only allowing it through stored procedures and views.

The downside of triggers is that any action can invoke them; this is also a strength - no-one is going to screw up the integrity of the system through incompetence.

As a counterpoint, allowing access to a database only through stored procedures and views still allows the backdoor access of permissions. Users with sufficient permissions are trusted not to break database integrity, all others use stored procedures.

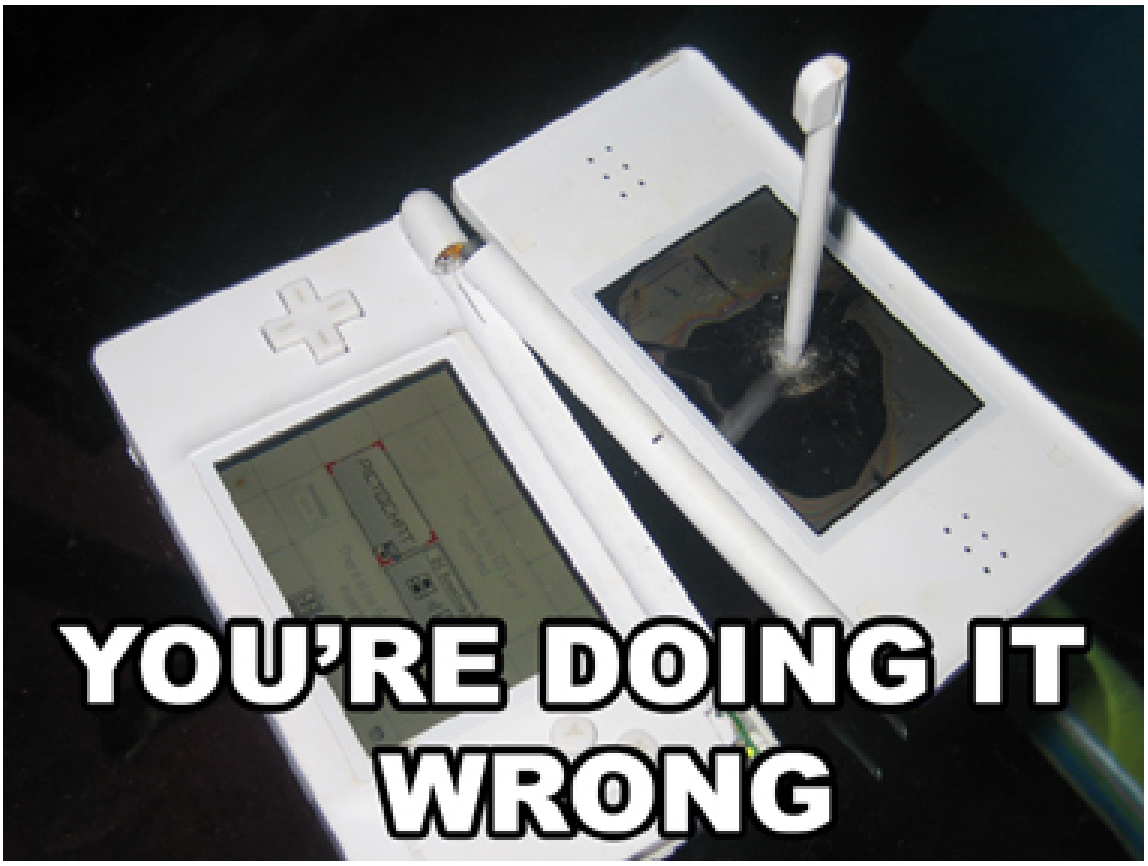
As to reducing the amount of work: databases are stunningly efficient when they don't have to deal with the outside world; you'd be really surprised how much even process switching hurts performance. That's another upside of stored procedures: rather than a dozen calls to the database (and all the associated round trips), there's one.

> Bunching stuff up in a single stored proc is fine, but what happens when something goes wrong? Say you have 5 steps and the first step fails, what happens to the other steps? You need to add a whole bunch of logic in there to cater for that situation. Once you start doing that you lose the benefits of the stored procedure in that scenario.

Business logic has to go somewhere, and there's a lot of implied domain rules embedded in the design of a database - relations, constraints and so on are an attempt to codify business rules by saying, for example, a user can only have one password. Given you've started shoving business rules onto the database server by having these relations and so on, where do you draw the line? When does the database give up responsibility for the integrity of the data, and start trusting the calling apps and database users to get it right? Stored procedures with these rules embedded in them can push a lot of political power into the hands of the DBAs. It comes down to how many tiers are going to exist in your n-tier

architecture; if there's a presentation, business and data layer, where does the separation between business and data lie? What value-add does the business layer add? Will you run the business layer on the database server as stored procedures?

Yes, I think that having to bypass a trigger means that you're "doing it wrong"; in this case a trigger isn't for you.



edited Mar 28, 2012 at 22:40

Neysor
3,911 ● 11 ● 35 ● 66

answered Aug 18, 2008 at 0:37

Josh
8,016 ● 5 ● 43 ● 63

2

I work with web and winforms apps in c# and I **HATE** triggers with a passion. I have never come across a situation where I could justify using a trigger over moving that logic into the business layer of the application and replicating the trigger logic there.

I don't do any DTS type work or anything like that, so there might be some use cases for using trigger there, but if anyone in any of my teams says that they might want to use a trigger they better have prepared their arguments well because I refuse to stand by and let triggers be added to any database I'm working on.

Some reasons why I don't like triggers:

- They move logic into the database. Once you start doing that, you're asking for a world of pain because you lose your debugging, your compile time safety, your logic flow. It's all downhill.
- The logic they implement is not easily visible to anyone.
- Not all database engines support triggers so your solution creates dependencies on database engines

I'm sure I could think of more reasons off the top of my head but those alone are enough for me not to use triggers.

answered Aug 18, 2008 at 0:08

lomaxx

116k ● 58 ● 147 ● 180

1 I use triggers on the insert/update/delete conditionals to increase/decrease counters on a table. Right now this is the only time i use it. Are those triggers ok? – user34537 Oct 20, 2010 at 12:48

2

"Never design a trigger to do integrity constraint checking that crosses rows in a table" -- I can't agree. The question is tagged 'SQL Server' and CHECK constraints' clauses in SQL Server cannot contain a subquery; worse, the implementation seems to have a 'hard coded' assumption that a CHECK will involve only a single row so using a function is not reliable. So if I need a constraint which does legitimately involve more than one row -- and a good example here is the sequenced primary key in a classic 'valid time' temporal table where I need to prevent overlapping periods for the same entity -- how can I do that without a trigger? Remember this is a primary key, something to ensure I have data integrity, so enforcing it anywhere other than the DBMS is out of the question. Until CHECK constraints get subqueries, I don't see an alternative to using triggers for certain kinds of integrity constraints.

Share Improve this answer

Follow

answered Oct 14, 2008 at 19:38

onedaywhen

56.9k ● 12 ● 102 ● 142

2

Triggers can be very helpful. They can also be very dangerous. I think they're fine for house cleaning tasks like populating audit data (created by, modified date, etc) and in some databases can be used for referential integrity.

But I'm not a big fan of putting lots of business logic into them. This can make support problematic because:

- it's an extra layer of code to research
- sometimes, as the OP learned, when you need to do a data fix the trigger might be doing things with the assumption that the data change is always via an application directive and not from a developer or DBA fixing a problem, or even from a different app

As for having to bypass a trigger to do something, it could mean you are doing something wrong, or it could mean that the trigger is doing something wrong.

The general rule I like to use with triggers is to keep them light, fast, simple, and as non-invasive as possible.

Share Improve this answer Follow

edited Nov 16, 2013 at 2:02

answered Aug 18, 2008 at 0:18

Bernard Dy
1,992 ● 2 ● 29 ● 39

1

I find myself bypassing triggers when doing bulk data imports. I think it's justified in such circumstances.

If you end up bypassing the triggers very often though, you probably need to take another look at what you put them there for in the first place.

In general, I'd vote for "they serve a purpose in some scenarios". I'm always nervous about performance implications.

Share Improve this answer

Follow

answered Aug 18, 2008 at 0:13

Blorgbeard

103k ● 50 ● 235 ● 276

0

I'm not a fan, personally. I'll use them, but only when I uncover a bottleneck in the code that can be cleared by moving actions into a trigger. Generally, I prefer simplicity and one way to keep things simple is to keep logic in one place - the application. I've also worked on jobs where access is very compartmentalized. In those environments, the more code I pack into triggers the more people I have to engage for even the simplest fixes.

Share Improve this answer

Follow

answered Aug 18, 2008 at 0:09

Rob Wilkerson

41.2k ● 43 ● 139 ● 192

0

I first used triggers a couple of weeks ago. We changed over a production server from SQL 2000 to SQL 2005 and we found that the drivers were behaving differently with NText fields (storing a large XML document), dropping off the last byte. I used a trigger as a temporary fix to add an extra dummy byte (a space) to the end of the data, solving our problem until a proper solution could be rolled out.

Other than this special, temporary case, I would say that I would avoid them since they do hide what is going on, and the function they provide should be handled explictly by the developer rather then as some hidden magic.

Share Improve this answer

Follow

answered Aug 18, 2008 at 0:30

Anthony K

2,603 ● 4 ● 35 ● 42

0

Honestly the only time I use triggers to simulate a unique index that is allowed to have NULL that don't count for the uniqueness.

Share Improve this answer

Follow

answered Aug 18, 2008 at 0:40

Nick Berardi

54.8k ● 15 ● 117 ● 136

0

> As to reducing the amount of work: databases are stunningly efficient when they don't have to deal with the outside world; you'd be really surprised how much even process switching hurts performance. That's another upside of stored procedures: rather than a dozen calls to the database (and all the associated round trips), there's one.

this is a little off topic, but you should also be aware that you're only looking at this from one potential positive.

Bunching stuff up in a single stored proc is fine, but what happens when something goes wrong? Say you have 5 steps and the first step fails, what happens to the other steps? You need to add a whole bunch of logic in there to cater for that situation. Once you start doing that you lose the benefits of the stored procedure in that scenario.

Share Improve this answer

Follow

answered Aug 18, 2008 at 1:01

lomaxx

116k ● 58 ● 147 ● 180

0

Total fan,

but really have to use it sparingly when,

- Need to maintain consistency (especially when dimension tables are used in a warehouse and we

need to relate the data in the fact table with their proper dimension . Sometime, the proper row in the dimension table can be very expensive to compute so you want the key to be written straight to the fact table, one good way to maintain that "relation" is with trigger.

- Need to log changes (in a audit table for instance, it's useful to know what @@user did the change and when it occurred)

Some RDBMS like sql server 2005 also provide you with triggers on CREATE/ALTER/DROP statements (so you can know who created what table, when, dropped what column, when, etc..)

Honestly, using triggers in those 3 scenarios, I don't see why would you ever need to "disable" them.

Share Improve this answer Follow

answered Aug 18, 2008 at 1:32

Mathieu Dumais-Savard

0

The general rule of thumb is: do not use triggers. As mentioned before, they add overhead and complexity that can easily be avoided by moving logic out of the DB layer.

Also, in MS SQL Server, triggers are fired once per sql command, and not per row. For example, the following sql statement will execute the trigger only once.

```
UPDATE tblUsers
SET Age = 11
WHERE State = 'NY'
```

Many people, including myself, were under the impression that the triggers are fired on every row, but this isn't the case. If you have a sql statement like the one above that may change data in more than one row, you might want to include a cursor to update all records affected by the trigger. You can see how this can get convoluted very quickly.

Share Improve this answer

Follow

answered Aug 18, 2008 at 19:08

jinsungy

10.8k ● 24 ● 73 ● 80

4 Please never use a cursor in a trigger. Yes you need to account for multiple row inserts, updates or deletes, but you need to do it in a set-based fashion. I removed a cursor from a trigger where I work and the insert of 40,000 records changed from 45 minutes to around 30 seconds. – HLGEM Nov 11, 2008 at 18:16