

Re: Updating the SQL key value

Source: <http://coding.derkeiler.com/Archive/PHP/comp.lang.php/2007-03/msg01753.html>

- *From:* Jerry Stuckle <jstucklex@xxxxxxxxxxxxxx>
 - *Date:* Fri, 30 Mar 2007 15:32:41 -0500
-

Toby A Inkster wrote:

Erwin Moller wrote:

Why picking 'natural candidates' if you can make it work ALWAYS with a simple autonumbering PK?

As someone who normally spends quite some time refining my database schema before creating the database, by the time I've come to pick a primary key, I've normally already decided on one or two UNIQUE constraints, so it's a simple matter of deciding which of the UNIQUE constraints is fit to be the primary key.

Besides, there are other techniques that can be said to "always work". For example, the creation of a varchar column "code" as a primary key. It's still a surrogate key really, but it can take on more meaning than an auto-numbered surrogate.

Hi, Toby, and I have to agree with Erwin. As much as I respect you, I have to disagree. Please see below...

For example, in one of my current projects, I need to store several articles, each of which must be issued under a particular licence (e.g. GPL, FDL, Creatice Commons). The "auto-number everything" solution would be:

```
=====
table: articles -----
article_id integer, autonumbered
title varchar
body varchar
licence integer
=====
```

Re: Updating the SQL key value

```
=====
table: licences
-----
```

```
licence_id integer
licence_name varchar
licence_link varchar
=====
```

Example data in table licences:

- 1 GNU General Public Licence <http://www.gnu...>
- 2 GNU Free Documentation Licence <http://www.gnu...>
- 3 Creative Commons Licence <http://www.cre...>

Using a manually-named varchar surrogate primary key, you could have

```
=====
table: articles -----
article_id integer, autonumbered
title varchar
body varchar
licence char(8)
=====
```

```
=====
table: licences
-----
```

```
licence_code char(8)
licence_name varchar
licence_link varchar
=====
```

Example data in table licences:

- GPL GNU General Public Licence <http://www.gnu...>
FDL GNU Free Documentation Licence <http://www.gnu...>
CC Creative Commons Licence <http://www.cre...>

Usage is fairly similar, apart from the fact that now, when you look at the table 'articles' without doing any joins, you can still infer a bit of information about which licence each article is under, without having to inner join onto the licences table.

True, but you also have to look at performance issues. When searching an index, comparing an int is always faster than comparing a varchar. And comparing a single column is always faster than comparing multiple columns. And the index file itself is smaller.

Re: Updating the SQL key value

This isn't **always** a good approach, but it's often a lot better than an auto-numbered key.

Additionally, the PK should not be dependent on data which may change – i.e. if part of your key was the license (code), what would happen if they changed the licensing terms?

And before you say that this is a waste of space as char(8) takes up eight bytes rather than 4 bytes for an integer, you're second-guessing the database engine there. Database engines really are dead clever. Most will only store the full char(8) string in the licences table (likely to be quite small compared to the articles table), and when storing the licence column of the articles table will actually use a pointer back to the same string data from the licence table — very fast. Database engines really are dead clever. (And as far as sorting is concerned, a short, indexed char field is just as fast as an integer.)

Not generally, they don't, because of potential problems. For instance, if the data is stored as a pointer to the license table, the system has to do an additional file lookup to fetch the license data. And what happens if the entry from the license table is altered – or worse yet, deleted? There is no referential integrity built in here. Deleting an item from the license table would require all other tables which point to that entry be updated – that is, the varchar data would have to be reinserted into each row in every table which pointed to the license table.

And in your case, the data would be stored in 4 bytes ("GPL" + 1 byte length). So it would take the same 4 bytes – but comparisons would still be slower.

What if your database must be upgraded and the logic changes?
Do you want to check all the columns again to be sure the PK still makes sense? (Or watch it fail in a production environment when the UNIQUE constraint is hit you didn't see coming beforehand)

When the logic changes in some major way, you're probably going to need to make adjustments to several tables anyway. I don't see this as a major problem.

Even when logic changes I don't generally have to change table design, other than to perhaps add a column or two. Good normalization techniques help here.

Besides which, it's often quite easy to choose a column that will always be unique. For example for a table of users, instead of:

Re: Updating the SQL key value

user_id auto_increment (primary key)
login varchar
password varchar
realname varchar
email_address varchar

All you need is:

login varchar (primary key)
password varchar
realname varchar
email_address varchar

Whatsmore, say you then have another table which has a column that has a foreign key for your user table, looking down that column you don't see a bunch of numbers like "12, 14, 71, 14" — you see "brian, dave, greg, dave".

Logic changing is never going to be a problem — you're never going to have two users with the same login name.

That's true. But you're also taking up more storage space and slowing down searches.

I have been using autonumbering PK my whole programming carrier, and never had any problems with it.

No doubt — the problem occurs when people blindly use surrogate keys without thinking about whether the rest of their columns could be keys — which I'm sure you'd never do Erwin! :-)

By doing that, they put less thought into UNIQUE constraints, and end up with lots of unwanted rows which would be duplicates, except for their primary key. If they hadn't added that extra primary key number, then they wouldn't have a database full of duplicate values.

You still can (and should) put serious thought into unique constraints. Just having a separate PK shouldn't change that.

Who seriously cares about the few extra bytes needed?

Not me.

Re: Updating the SQL key value

I do. When it comes to very large databases, this can add up very quickly. And it can slow the system down significantly.

I am not alone with that thought.
Postgres even makes an OID for each row, something you don't even see but can use if you want.

It's perfectly easy to disable OIDs in PostgreSQL on a case-by-case basis or permanently. Most of the time, I disable OIDs.

On one of my current projects, I've got twelve tables, only one of which has a surrogate integer primary key (technically it doesn't autonumber, but I use MAX() to simulate autonumbering when creating a new record). Guess which table is causing me the most problems?

I can imagine. Using MAX() like this can cause concurrency problems. That's why RDB designers came out with the auto-numbering columns.

It's a table of articles. A table of comments references the articles via its numeric key. Now what happens if I write an updated version of an article, but want to keep a copy of the old one? I mark a status flag on the old one to hide it, then create another article with the same URL but a different ID number. Unfortunately the comments still point to the old ID number, so are not seen when you visit the new page.

It's not that hard to update the comment table when updating the article table.

Better would have been to design my table so that it had a primary key like (url, revision).

But then your comment table wouldn't point to it anyway if you use (url, revision). Or, if you just use url, your comment table would be pointing at two different entries – which is not good foreign key design.

Anyhow, it's mostly a matter of taste. I was being somewhat tongue in cheek when describing surrogate keys as "the root of all evil", but I can't stand to see a good candidate key go to waste.

Re: Updating the SQL key value

Here, I agree. But I tend to lean more towards the performance side. Design the database with efficiency in mind. That's why all databases aren't 5NF form.

--

=====
Remove the "x" from my email address

Jerry Stuckle

JDS Computer Training Corp.

jstucklex@xxxxxxxxxxxxxx

=====
.