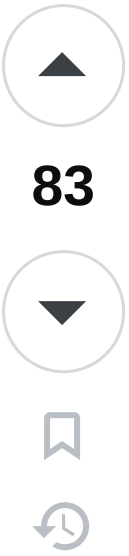
What's your opinion on using UUIDs as database row identifiers, particularly in web apps?

Asked 16 years, 4 months ago Modified 2 years, 6 months ago

Viewed 29k times



I've always preferred to use long integers as primary keys in databases, for simplicity and (assumed) speed. But when using a REST or Rails-like URL scheme for object instances, I'd then end up with URLs like this:

```
http://example.com/user/783
```

And then the assumption is that there are also users with IDs of 782, 781, ..., 2, and 1. Assuming that the web app in question is secure enough to prevent people entering other numbers to view other users without authorization, a simple sequentially-assigned surrogate key also "leaks" the total number of instances (older than this one), in this case users, which might be privileged information. (For instance, I am user #726 in stackoverflow.)

Would a UUID/GUID be a better solution? Then I could set up URLs like this:

```
http://example.com/user/035a46e0-6550-11dd-ad8b-0800200c9a66
```

Not exactly succinct, but there's less implied information about users on display. Sure, it smacks of "security through obscurity" which is no substitute for proper security, but it seems at least a little more secure.

Is that benefit worth the cost and complexity of implementing UUIDs for web-addressable object instances? I think that I'd still want to use integer columns as database PKs just to speed up joins.

There's also the question of in-database representation of UUIDs. I know MySQL stores them as 36-character strings. Postgres seems to have a more efficient internal representation (128 bits?) but I haven't tried it myself. Anyone have any experience with this?

Update: for those who asked about just using the user name in the URL (e.g., http://example.com/user/yukondude), that works fine for object instances with names that are unique, but what about the zillions of web app objects that can really only be identified by number? Orders, transactions, invoices, duplicate image names, stackoverflow questions, ...

database **web-applications** **uuid**

Share

Improve this question

Follow

edited Aug 8, 2008 at 18:34

asked Aug 8, 2008 at 13:55

yukondude

24.6k ● 13 ● 50 ● 58

16 Answers

Sorted by: Highest score (default)

34

I can't say about the web side of your question. But uuids are great for n-tier applications. PK generation can be decentralized: each client generates it's own pk without risk of collision. And the speed difference is generally small.

Make sure your database supports an efficient storage datatype (16 bytes, 128 bits). At the very least you can encode the uuid string in base64 and use char(22).

I've used them extensively with Firebird and do recommend.

Share Improve this answer

Follow

answered Aug 8, 2008 at 14:03

Douglas Tosi

2,340 ● 2 ● 18 ● 19

19 base64? If you don't have a native data type for UUID, drop the dashes and stick in byte(32). That'll probably be faster than encoding/decoding to/from base64 when you need the UUID. – CMircea May 5, 2011 at 19:26

28

For what it's worth, I've seen a long running stored procedure (9+ seconds) drop to just a few hundred milliseconds of run time simply by switching from GUID primary keys to integers. That's not to say *displaying* a GUID is a bad idea, but as others have pointed out, joining on them, and indexing them, by definition, is not going to be anywhere near as fast as with integers.

Share Improve this answer

Follow

answered Aug 8, 2008 at 15:20

Adam Tuttle

19.8k ● 17 ● 83 ● 116

1 If you could provide some more specifics on where you saw this, it'd be helpful. Size of DB/tables? DB backend? Access pattern (what did the query look like) ... etc? – Garen Mar 24, 2013 at 23:37

17 How is this even an answer. – davidahines Dec 20, 2013 at 20:02

21 It's anecdotal evidence that supports the mathematical theory that joining and indexing integers will be faster than long(ish) strings. – Adam Tuttle Dec 20, 2013 at 20:19

1 UUIDs are not strings in the database. They should be stored with the UUID type, which is a binary format. Using a string column is incorrect and brings the performance drawbacks you mentioned. Maybe when the answer was written (2008), the UUID database type did not exist. – Vincent Jun 23, 2022 at 9:33

I can answer you that in SQL server if you use a uniqueidentifier (GUID) datatype and use the NEWID()

23

function to create values you will get horrible fragmentation because of page splits. The reason is that when using NEWID() the value generated is not sequential. SQL 2005 added the NEWSEQUANTIAL() function to remedy that

One way to still use GUID and int is to have a guid and an int in a table so that the guid maps to the int. the guid is used externally but the int internally in the DB

for example

```
457180FB-C2EA-48DF-8BEF-458573DA1C10    1
9A70FF3C-B7DA-4593-93AE-4A8945943C8A    2
```

1 and 2 will be used in joins and the guids in the web app. This table will be pretty narrow and should be pretty fast to query

Share Improve this answer

Follow

answered Aug 8, 2008 at 14:05

SQLMenace

135k ● 25 ● 211 ● 225

10

Why couple your primary key with your URI?

Why not have your URI key be human readable (or unguessable, depending on your needs), and your primary index integer based, that way you get the best of both worlds. A lot of blog software does that, where the exposed id of the entry is identified by a 'slug', and the numeric id is hidden away inside of the system.

The added benefit here is that you now have a really nice URL structure, which is good for SEO. Obviously for a transaction this is not a good thing, but for something like stackoverflow, it is important (see URL up top...). Getting uniqueness isn't that difficult. If you are really concerned, store a hash of the slug inside a table somewhere, and do a lookup before insertion.

edit: Stackoverflow doesn't quite use the system I describe, see Guy's comment below.

Share Improve this answer Follow

edited Nov 19, 2015 at 18:58

answered Sep 16, 2008 at 5:24

Jonathan Arkell

10.8k ● 2 ● 26 ● 32

8 Stack Overflow indexes on the ID and not the slug. Try changing the slug at the top of the page and hit enter. It will 301 redirect you to the canonical URL for this page based on the ID (5949) and ignores the slug. On the server, it compares the slug to the stored/generated slug. If not the same it returns a 301. However it find that by lookup on the ID (5949). – Guy Nov 18, 2015 at 3:23

4

Rather than URLs like this:

```
http://example.com/user/783
```

Why not have:

```
http://example.com/user/yukondude
```

Which is friendlier to humans and doesn't leak that tiny bit of information?

Share Improve this answer

Follow

answered Aug 8, 2008 at 18:24

Josh

8,016 ● 5 ● 43 ● 63

4

You could use an integer which is related to the row number but is not sequential. For example, you could take the 32 bits of the sequential ID and rearrange them with a fixed scheme (for example, bit 1 becomes bit 6, bit 2 becomes bit 15, etc..).
This will be a bidirectional encryption, and you will be sure that two different IDs will always have different encryptions.
It would obviously be easy to decode, if one takes the time to generate enough IDs and get the schema, but, if I understand correctly your problem, you just want to not give away information too easily.

Share Improve this answer

Follow

answered Aug 12, 2008 at 18:20

Andrea Bertani

1,637 ● 12 ● 11

I don't think that the intent of the question was to have a secure way of using UUIDs. As far as I have understood that

the topic were the practical ramifications of that decision. And your scheme adds no security and is a waste of cpu cycles!
– Patrick Cornelissen Nov 10, 2013 at 10:07

4

We use GUIDs as primary keys for all our tables as it doubles as the RowGUID for MS SQL Server Replication. Makes it very easy when the client suddenly opens an office in another part of the world...

Share Improve this answer

Follow

answered Aug 12, 2008 at 21:00

Marius

2,542 ● 6 ● 31 ● 43

3

I don't think a GUID gives you many benefits. Users hate long, incomprehensible URLs.

Create a shorter ID that you can map to the URL, or enforce a unique user name convention (http://example.com/user/brianly). The guys at 37Signals would probably mock you for worrying about something like this when it comes to a web app.

Incidentally you can force your database to start creating integer IDs from a base value.

Share Improve this answer

Follow

answered Aug 8, 2008 at 15:10

Brian Lyttle

14.6k ● 15 ● 69 ● 106

This is unapplicable, you don't need to show the uuid in the url. – davidahines Dec 20, 2013 at 20:05

4 @dah the questioner mentions use of it within the URL in the question. – Brian Lyttle Dec 21, 2013 at 1:35

3

It also depends on what you care about for your application. For n-tier apps GUIDs/UUIDs are simpler to implement and are easier to port between different databases. To produce Integer keys some database support a sequence object natively and some require custom construction of a sequence table.

Integer keys probably (I don't have numbers) provide an advantage for query and indexing performance as well as space usage. Direct DB querying is also much easier using numeric keys, less copy/paste as they are easier to remember.

Share Improve this answer Follow

edited Sep 17, 2008 at 2:52

answered Sep 16, 2008 at 0:31

Michael Barker

14.4k ● 4 ● 50 ● 55

2

I work with a student management system which uses UUID's in the form of an integer. They have a table which hold the next unique ID.

Although this is probably a good idea for an architectural point of view, it makes working with on a daily basis difficult. Sometimes there is a need to do bulk inserts and having a UUID makes this very difficult, usually requiring writing a cursor instead of a simple SELECT INTO statement.

Share Improve this answer

Follow

answered Aug 8, 2008 at 13:59

GateKiller

75.8k ● 75 ● 175 ● 204

2

I've tried both in real web apps.

My opinion is that it is preferable to use integers and have short, comprehensible URLs.

As a developer, it feels a little bit awful seeing sequential integers and knowing that some information about total record count is leaking out, but honestly - most people probably don't care, and that information has never really been critical to my businesses.

Having long ugly UUID URLs seems to me like much more of a turn off to normal users.

Share Improve this answer

Follow

edited Oct 25, 2021 at 12:25

Rizwan

103 ● 4 ● 24

answered Oct 2, 2013 at 3:43

Daniel Alexiuc

13.3k ● 9 ● 60 ● 73

Thanks for this opinion. I researched using UUIDs as primary keys with all its possible disadvantages for days until I realized that the only advantage (hiding business information) is not worth it, in my case. – Dr. Jan-Philip Gehrcke May 28, 2015 at 14:44

1

I think that this is one of these issues that cause quasi-religious debates, and its almost futile to talk about. I would just say use what you prefer. In 99% of systems it will no matter which type of key you use, so the benefits (stated in the other posts) of using one sort over the other will never be an issue.

Share Improve this answer

Follow

answered Aug 12, 2008 at 19:30

Dan

29.4k ● 44 ● 151 ● 209

1

I think using a GUID would be the better choice in your situation. It takes up more space but it's more secure.

Share Improve this answer

Follow

edited Jun 23, 2017 at 15:55

Matt

75.3k ● 26 ● 154 ● 180

answered Aug 8, 2008 at 14:00

Bryan Roth

10.7k ● 15 ● 49 ● 56

0

YouTube uses 11 characters with base64 encoding which offers 11^64 possibilities, and they are usually pretty manageable to write. I wonder if that would offer better performance than a full on UUID. UUID converted to base 64 would be double the size I believe.

More information can be found here: https://www.youtube.com/watch?v=gocwRvLhDf8

Share Improve this answer

Follow

edited Oct 25, 2021 at 12:25

Rizwan

103 • 4 • 24

answered Jul 23, 2018 at 3:39

Sousaplex

169 • 9

0

Pros and Cons of UUID

> Note: uuid_v7 is time based uuid instead of random. So you can use it to order by creation date and solve some performance issues with db inserts if you do really many of them.

Pros:

- can be generated on api level (good for distributed systems)
- hides count information about entity

- doesn't have limit 2,147,483,647 as 32-bit int
- removes layer of errors related to passing one entity id `userId: 25` to get another `bookId: 25` accidently
- more friendly graphql usage as `ID` key

Cons:

- 128-bit instead 32-bit int (slightly bigger size in db and ~40% bigger index, around ~30MB for 1 million rows), should be a minor concern
- can't be sorted by creation (**can be solved with uuid_v7**)
- non-time-ordered UUID versions such as UUIDv4 have poor database index locality (**can be solved with uuid_v7**)

URL usage

Depending on app you may care or not care about url. If you don't care, just use `uuid` as is, it's fine.

If you care, then you will need to decide on url format.

Best case scenario is a use of unique slug if you ok with never changing it:

```
http://example.com/sale/super-duper-phone
```

If your url is generated from title and you want to change slug on title change there is a few options. Use it as is and query by uuid (slug is just decoration):

```
http://example.com/book/035a46e0-6550-11dd-ad8b-0800200c9a66/new-title
```

Convert it to base64url:

- you can get uuid back from `AYEWXcsicACGA6PT7v_h3A`
- `AYEWXcsicACGA6PT7v_h3A` - 22 characters
- `035a46e0-6550-11dd-ad8b-0800200c9a66` - 36 characters

```
http://example.com/book/AYEWXcsicACGA6PT7v_h3A/new-title
```

Generate a unique short 11 chars length string just for slug usage:

```
http://example.com/book/icACEWXcsAY-new-title
http://example.com/book/icACEWXcsAY/new-title
```

If you don't want uuid or short id in url and want only slug, but do care about seo and user bookmarks, you will need to redirect all request from

```
http://example.com/sale/phone-1-title
```

to

```
http://example.com/sale/phone-1-title-updated
```

this will add additional complexity of managing slug history, adding fallback to history for all queries where slug is used and redirects if slugs doesn't match

Share Improve this answer

Follow

edited May 30, 2022 at 22:46

answered May 30, 2022 at 21:39

ZiiMakc

36.4k ● 25 ● 72 ● 120

-1

As long as you use a DB system with efficient storage, HDD is cheap these days anyway...

I know GUID's can be a b*tch to work with some times and come with some query overhead however from a security perspective they are a savior.

Thinking security by obscurity they fit well when forming obscure URI's and building normalised DB's with Table, Record and Column defined security you cant go wrong with GUID's, try doing that with integer based id's.

Share Improve this answer

Follow

answered Feb 25, 2013 at 10:41

user2106945

1