Advantages and disadvantages of GUID / UUID database keys

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

Viewed 119k times

280

I've worked on a number of database systems in the past where moving entries between databases would have been made a lot easier if all the database keys had been GUID / UUID values. I've considered going down this path a few times, but there's always a bit of uncertainty, especially around performance and un-read-out-over-the-phone-able URLs.

Has anyone worked extensively with GUIDs in a database? What advantages would I get by going that way, and what are the likely pitfalls?

database guid uuid

Share

Improve this question

Follow

edited Jul 27, 2009 at 13:31

Steffen Opel
64.6k ● 11 ● 195 ● 212

asked Sep 5, 2008 at 8:00

Matt Sheppard
118k ● 46 ● 113 ● 134

3 Jeff has a post about it "Primary Keys: IDs versus GUIDs".
– jfs Sep 5, 2008 at 8:11

1 can also use Hi-Lo for remote clients: stackoverflow.com/questions/282099/whats-the-hi-lo-algorithm – Neil McGuigan Aug 9, 2013 at 3:59

possible duplicate of What's your opinion on using UUIDs as database row identifiers, particularly in web apps? – user Sep 18, 2013 at 9:30

Updated location for Jeff Atwood's post about "Primary Keys: IDs versus GUIDs." Thanks to @jfs for the reference.
– Adam Katz May 18, 2015 at 23:26

@jfs Link has changed to blog.codinghorror.com/primary-keys-ids-versus-guids – cr0ss Jul 7, 2015 at 14:06

10 Answers

Sorted by: Highest score (default)

290

Advantages:

- Can generate them offline.
- Makes replication trivial (as opposed to int's, which makes it REALLY hard)
- ORM's usually like them
- Unique across applications. So We can use the PK's from our CMS (guid) in our app (also guid) and know we are NEVER going to get a clash.

Disadvantages:

- Larger space use, but space is cheap(er)
- Can't order by ID to get the insert order.
- Can look ugly in a URL, but really, WTF are you doing putting a REAL DB key in a URL!? (This point disputed in comments below)
- Harder to do manual debugging, but not that hard.

Personally, I use them for most PK's in any system of a decent size, but I got "trained" on a system which was replicated all over the place, so we HAD to have them. YMMV.

I think the duplicate data thing is rubbish - you can get duplicate data however you do it. Surrogate keys are usually frowned upon where ever I've been working. We DO use the WordPress-like system though:

- unique ID for the row (GUID/whatever). Never visible to the user.
- public ID is generated ONCE from some field (e.g. the title - make it the-title-of-the-article)

UPDATE: So this one gets +1'ed a lot, and I thought I should point out a big downside of GUID PK's: Clustered Indexes.

If you have a lot of records, and a clustered index on a GUID, your insert performance will SUCK, as you get inserts in random places in the list of items (that's the point), not at the end (which is quick).

So if you need insert performance, maybe use a auto-inc INT, and generate a GUID if you want to share it with someone else (e.g., showing it to a user in a URL).

Share Improve this answer Follow

edited Oct 13, 2022 at 13:51
MuchToLearn
373 ● 3 ● 13

answered Sep 5, 2008 at 9:44
Nic Wise
8,119 ● 2 ● 32 ● 30

216 [WTF are you doing putting a REAL DB key in a URL!?] Not sure why that bothers you. What else would you use? Look at Stack Overflow... It has IDENTITY values in the URL all over the place, and it works just fine. Using DB keys in URLs doesn't prevent you from enforcing security. – Euro Micelli Sep 15, 2008 at 23:13

22 No, it doesn't, but things like SEO are usually better if there isn't a key in it - especially something as long as a GUID. Of course, it can be worked around easily, so I gues that was a bit of an over sweeping statement – Nic Wise Oct 25, 2008 at 8:18

7 Good answer, it'd be nice if you also add information about performance disadvantages of using GUIDs; e.g. joining, sorting, and indexing by them will all be slower than using integers. Guids are fantastic, but they come at a cost which can be a pain when performance is critical. – Doctor Jones Jun 3, 2011 at 10:16

28 Keep one thing in mind, people often change page, question, forum titles. For SEO it's GOOD to have something like a small ID in the URL so that if the title changes you still know where to forward people coming from an OLD URL. `example.com/35/old-and-busted`

just became `example.com/35/new-hotness` and you're app can just check the title and forward the user on with a 301. – Xeoncross Nov 15, 2012 at 18:22

9 Indexing a GUID is expensive and slow, which makes them really poor candidates for primary keys. – Matthew James Davis Dec 29, 2013 at 2:57

22

Why doesn't anyone mention performance? When you have multiple joins, all based on these nasty GUIDs the performance will go through the floor, been there :(

Share Improve this answer Follow

answered Sep 6, 2008 at 1:05

Andrei Rînea
20.7k ● 18 ● 121 ● 169

1 Can you elaborate on this as am in the situation where i need to introduce UUID (or similar) , but am concerned about using them as Primary Key. – JoeTidee Jan 23, 2015 at 17:54

3 UUIDs are only 4 times the size of integers... (if your database has a UUID type) – Jasen Jun 14, 2018 at 7:44

15

@Matt Sheppard:

Say you have a table of customers. Surely you don't want a customer to exist in the table more than once, or lots of confusion will happen throughout your sales and logistics departments (especially if the multiple rows about the customer contain different information).

So you have a customer identifier which uniquely identifies the customer and you make sure that the identifier is known by the customer (in invoices), so that the customer and the customer service people have a common reference in case they need to communicate. To guarantee no duplicated customer records, you add a uniqueness-constraint to the table, either through a primary key on the customer identifier or via a NOT NULL + UNIQUE constraint on the customer identifier column.

Next, for some reason (which I can't think of), you are asked to add a GUID column to the customer table and make that the primary key. If the customer identifier column is now left without a uniqueness-guarantee, you are asking for future trouble throughout the organization because the GUIDs will always be unique.

Some "architect" might tell you that "oh, but we handle the *real* customer uniqueness constraint in our app tier!". Right. Fashion regarding that general purpose programming languages and (especially) middle tier frameworks changes all the time, and will generally never out-live your database. And there is a very good chance that you will at some point need to access the database without going through the present application. == Trouble. (But fortunately, you and the "architect" are long gone, so you will not be there to clean up the mess.) In other words: Do maintain obvious constraints in the database (and in other tiers, as well, if you have the time).

In other words: There may be good reasons to add GUID columns to tables, but please don't fall for the temptation to make that lower your ambitions for consistency within the *real* (==non-GUID) information.

Share Improve this answer Follow

edited Sep 7, 2008 at 18:05

answered Sep 5, 2008 at 9:28

Troels Arvin
6,392 ● 2 ● 27 ● 28

1 Hear hear! Love your SQL comparison page btw. Extremely useful. The only thing I miss is a changelog. – Henrik Gustafsson Sep 16, 2008 at 20:40

5 I think this answer needs some clarification: this assumes that UUIDs are never used as primary keys. I don't know where this assumption comes from, but I have yet to see a system that doesn't allow you to use them as such. *I know it's an old answer, I suppose the advantages of using UUIDs in distributed systems weren't as widely understood back then (?).* – tne Mar 6, 2014 at 19:44

15

The main advantages are that you can create unique id's without connecting to the database. And id's are globally unique so you can easilly combine data from different databases. These seem like small advantages but have saved me a lot of work in the past.

The main disadvantages are a bit more storage needed (not a problem on modern systems) and the id's are not really human readable. This can be a problem when debugging.

There are some performance problems like index fragmentation. But those are easilly solvable (comb guids by jimmy nillson: http://www.informit.com/articles/article.aspx?p=25862)

Edit merged my two answers to this question

@Matt Sheppard I think he means that you can duplicate rows with different GUIDs as primary keys. This is an issue with any kind of surrogate key, not just GUIDs. And like he said it is easilly solved by adding meaningfull unique constraints to non-key columns. The alternative is to use a natural key and those have real problems..

Share Improve this answer Follow

edited Sep 16, 2008 at 20:33

answered Sep 5, 2008 at 8:15

Mendelt
37.5k ● 6 ● 75 ● 97

12

GUIDs may cause you a lot of trouble in the future if they are used as "uniqifiers", letting duplicated data get into your tables. If you want to use GUIDs, please consider still maintaining UNIQUE-constraints on other column(s).

answered Sep 5, 2008 at 8:38

Share Improve this answer Follow

Troels Arvin
6,392 ● 2 ● 27 ● 28

11 This is the heart of the problem: Introducing a GUID makes any row unique. But the non-artificial parts of the rows may suddenly contain duplicates (several versions of the truth). – Troels Arvin Jan 27, 2009 at 5:45

8 +1 to compensate. I see what you mean, but it's badly expressed. – Stefano Borini Aug 16, 2009 at 13:10

10

primary-keys-ids-versus-guids

The Cost of GUIDs as Primary Keys (SQL Server 2000)

Myths, GUID vs. Autoincrement (MySQL 5)

This is realy what you want.

UUID Pros

- Unique across every table, every database, every server
- Allows easy merging of records from different databases
- Allows easy distribution of databases across multiple servers
- You can generate IDs anywhere, instead of having to roundtrip to the database
- Most replication scenarios require GUID columns anyway

GUID Cons

- It is a whopping 4 times larger than the traditional 4-byte index value; this can have serious performance and storage implications if you're not careful
- Cumbersome to debug (where userid='{BAE7DF4-DDF-3RG-5TY3E3RF456AS10}')
- The generated GUIDs should be partially sequential for best performance (eg, newsequentialid() on SQL 2005) and to enable use of clustered indexes

Share Improve this answer Follow

edited Oct 6, 2020 at 13:52
commadelimited
5,612 ● 7 ● 45 ● 82

answered Oct 26, 2013 at 8:13
wener
7,720 ● 6 ● 64 ● 86

9

One other small issue to consider with using GUIDS as primary keys if you are also using that column as a clustered index (a relatively common practice). You are going to take a hit on insert because of the nature of a guid not begin sequential in anyway, thus their will be page splits, etc when you insert. Just something to consider if the system is going to have high IO...

Share Improve this answer Follow

answered Sep 16, 2008 at 2:40

WIDBA
169 ● 1 ● 7

4

There is one thing that is not really addressed, namely using **random** (UUIDv4) IDs as primary keys will harm the performance of the *primary key index*. It will happen whether or not your table is clustered around the key.

RDBMs usually ensure the uniqueness of the primary keys, and ensure the lookups by a key, in a structure called BTree, which is a search tree with a large branching factor (a binary search tree has branching factor of 2). Now, a sequential integer ID would cause the inserts to occur just *one* side of the tree, leaving most of the leaf nodes untouched. Adding random UUIDs will cause the insertions to split leaf nodes all over the index.

Likewise if the data stored is mostly temporal, it is often the case that the most recent data needs to be accessed and joined against the most. With random UUIDs the patterns will not benefit from this, and will hit more index rows, thereby needing more of the index pages in memory. With sequential IDs if the most-recent data is needed the most, the hot index pages would require less RAM.

Share Improve this answer Follow

answered Nov 29, 2017 at 16:57

Antti Haapala -- Слава Україні
134k ● 23 ● 289 ● 342

This appears to be a non-issue with Postgres. postgresql.org/message-id/… – mackstann Oct 1, 2021 at 16:18

@mackstann thanks for reminding about this, I remember a reading a page recently that actually had metrics to prove otherwise – Antti Haapala -- Слава Україні Oct 1, 2021 at 17:52

1 @mackstann I can't find the proper link, but here's another recent one from depesz with benchmarks: depesz.com/2020/02/19/why-im-not-fan-of-uuid-datatype – Antti Haapala -- Слава Україні Oct 1, 2021 at 18:00

2

Advantages:

- UUID values are unique between tables and databases. Thats why it can be merge rows between two databases or distributed databases.
- UUID is more safer to pass through url than integer type data. If one pass UUID through url, attackers can't guess the next id.But if we pass Integer type

such as 10, then attackers can guess the next id is 11 then 12 etc.

- UUID can generate offline.

Share Improve this answer Follow

answered Jul 27, 2020 at 19:31

Md. Samim hossain
265 ● 3 ● 7

1

One thing not mentioned so far: UUIDs make it much harder to profile data

For web apps at least, it's common to access a resource with the id in the url, like `stackoverflow.com/questions/45399`. If the id is an integer, this both

- provides information about the number of questions (ie September 5th, 2008, the 45,399th question was asked)
- provides a leverage point to iterate through questions (what happens when I increment that by 1? I open the next asked question)

From the first point, I can combine the timestamp from the question and the number to profile how frequently questions are asked and how that changes over time. this matters less on a site like Stack Overflow, with publicly available information, but, depending on context, this may expose sensitive information.

For example, I am a company that offers customers a permissions gated portal. the address is `portal.com/profile/{customerId}`. If the id is an integer, you could profile the number of customers regardless of being able to see their information by querying for `lastKnownCustomerCount + 1` regularly, and checking if the result is `404 - NotFound` (customer does not exist) or `403 - Forbidden` (customer does exist, but you do not have access to view).

UUIDs non-sequential nature mitigate these issues. This isn't a garunted to prevent profiling, but it's a start.

Share Improve this answer Follow

answered Feb 10, 2022 at 17:01

J Scott
1,009 ● 11 ● 15