What are the best practices for using a GUID as a primary key, specifically regarding performance? [closed]

Asked 9 years, 8 months ago Modified 3 months ago Viewed 191k times

392

268

Closed. This question is opinion-based. It is not currently accepting answers.

Want to improve this question? Update the question so it can be answered with facts and citations by editing this post.

Closed 4 months ago.

The community reviewed whether to reopen this question 4 months ago and left it closed:

Original close reason(s) were not resolved

Improve this question

I have an application that uses GUID as the Primary Key in almost all tables and I have read that there are issues about performance when using GUID as Primary Key. Honestly, I haven't seen any problem, but I'm about to start a new application and I still want to use the GUIDs as the Primary Keys, but I was thinking of using a Composite Primary Key (The GUID and maybe another field.)

I'm using a GUID because they are nice and easy to manage when you have different environments such as "production", "test" and "dev" databases, and also for migration data between databases.

I will use Entity Framework 4.3 and I want to assign the Guid in the application code, before inserting it in the database. (i.e. I don't want to let SQL generate the Guid).

What is the best practice for creating GUID-based Primary Keys, in order to avoid the supposed performance hits associated with this approach?

sql-server entity-framework database-design primary-key guid

Share Improve this question

Follow

edited Aug 13, 2014 at 5:20

abatishchev

94.9k 79 293 426

asked Aug 13, 2012 at 16:03

VAAA

13.4k 24 116 229

23 The issue is not supposed. If your PK is clustered then almost every insert has the potential to cause a page split. In modern versions of SQL Server this was "fixed" with NEWSEQUENTIALID(), but this loses the benefit of being able to calculate it beforehand. I strongly recommend you read up on GUIDs

4 I'd also add that the word **server** is ambiguous in *I want to assign the Guid on the* **server** *side (dont want to let SQL to create the GUID).* – Erik Philips Aug 13, 2012 at 16:11

This question has similarities to this "sql-server-guid-sort-algorithm-why" stackoverflow.com/questions/7810602/... – Clinton Ward Aug 13, 2012 at 16:27

BWAAA-HAAA-HAAA!!! They took one of the most important posts that had a link to a substantial presentation that explained all about why Random GUIDs are not the problem and that WE'RE the actual problem and they deleted. I question the supposed honorable goals of this site. It would appear they're not actually interested in solving problems and plenty of people also make references to other sites, especially db<>fiddle. – Jeff Moden Mar 3 at 21:07

9 Answers

Sorted by:
Reset to default

Trending (recent votes count more)

Help us improve our answers.
Are the answers below sorted in a way that puts the best answer at or near the top?

Take a short survey I'm not interested

571

GUIDs may seem to be a natural choice for your primary key - and if you really must, you could probably argue to use it for the PRIMARY KEY of the table. What I'd strongly recommend **not to do** is use the GUID column as the **clustering key**, which SQL Server does by default, unless you specifically tell it not to.

You really need to keep two issues apart:

1. the **primary key** is a logical construct - one of the candidate keys that uniquely and reliably identifies every row in your table. This can be anything, really - an `INT`, a `GUID`, a string - pick what makes most sense for your scenario.
2. the **clustering key** (the column or columns that define the "clustered index" on the table) - this is a *physical* storage-related thing, and here, a small, stable, ever-increasing data type is your best pick - `INT` or `BIGINT` as your default option.

By default, the primary key on a SQL Server table is also used as the clustering key - but that doesn't need to be that way! I've personally seen massive performance gains when breaking up the previous GUID-based Primary / Clustered Key into two separate key - the primary (logical) key on the GUID, and the clustering (ordering) key on a separate `INT IDENTITY(1,1)` column.

As Kimberly Tripp - the Queen of Indexing - and others have stated a great many times - a `GUID` as the clustering key isn't optimal, since due to its randomness, it will lead to massive page and index fragmentation and to generally bad performance.

prominently so.

Then there's another issue to consider: the clustering key on a table will be added to each and every entry on each and every non-clustered index on your table as well - thus you really want to make sure it's as small as possible. Typically, an `INT` with 2+ billion rows should be sufficient for the vast majority of tables - and compared to a `GUID` as the clustering key, you can save yourself hundreds of megabytes of storage on disk and in server memory.

Quick calculation - using `INT` vs. `GUID` as Primary and Clustering Key:

- Base Table with 1'000'000 rows (3.8 MB vs. 15.26 MB)
- 6 nonclustered indexes (22.89 MB vs. 91.55 MB)

TOTAL: 25 MB vs. 106 MB - and that's just on a single table!

Some more food for thought - excellent stuff by Kimberly Tripp - read it, read it again, digest it! It's the SQL Server indexing gospel, really.

- GUIDs as PRIMARY KEY and/or clustered key
- The clustered index debate continues
- Ever-increasing clustering key - the Clustered Index Debate.....again!
- Disk space is cheap - that's **not** the point!

PS: of course, if you're dealing with just a few hundred or a few thousand rows - most of these arguments won't really have much of an impact on you. However: if you get into the tens or hundreds of thousands of rows, or you start counting in millions - **then** those points become very crucial and very important to understand.

Update: if you want to have your `PKGUID` column as your primary key (but not your clustering key), and another column `MYINT` (`INT IDENTITY`) as your clustering key - use this:

```
CREATE TABLE dbo.MyTable
(PKGUID UNIQUEIDENTIFIER NOT NULL,
 MyINT INT IDENTITY(1,1) NOT NULL,
 .... add more columns as needed ..... )

ALTER TABLE dbo.MyTable
ADD CONSTRAINT PK_MyTable
PRIMARY KEY NONCLUSTERED (PKGUID)

CREATE UNIQUE CLUSTERED INDEX CIX_MyTable ON dbo.MyTable(MyINT)
```

Basically: you just have to **explicitly** tell the `PRIMARY KEY` constraint that it's `NONCLUSTERED` (otherwise it's created as your clustered index, by default) - and then you create a second index that's defined as `CLUSTERED`

This will work - and it's a valid option if you have an existing system that needs to be "re-

replication scenario, then I'd always pick `ID INT IDENTITY(1,1)` as my clustered primary key - much more efficient than anything else!

Share Improve this answer Follow

edited May 5, 2021 at 9:36

openshac
4,617 5 43 71

answered Aug 13, 2012 at 16:34

marc_s
702k 167 1300 1422

3 The way I read this is that having both a non clustered uniqueidentifier column and the int identity column, FK's should also be uniqueidentifier? If you do that, when would you actually use the identity column directly, or would you not? – pinkfloydx33 Nov 1, 2014 at 12:50

4 Little question, should the GUID now be used on joins, or the int id? My instinct tells me the GUID should be used, but I fail to see a technical problem using the int id... – Nicolas Belley Jun 27, 2015 at 13:33

5 @marc_s but in a replication scenario, if the int column is identity, shouldn't we use the GUID since the int column can repeat itself across devices? – Nicolas Belley Jun 28, 2015 at 11:51

4 This is an old thread, but might I add: don't just use a useless arbitrary INT as the clustering key. Use something useful like an inceremented date that is actually searched on, that has some relation to the data you're storing. You only get one clustering key, and if you choose the right one you'll get good performance – Nick.McDermaid Jul 9, 2016 at 8:42

7 @Kipei: the main issues is the ***I-F*** you have such a natural value - then yes, you can use it as a primary key. ***BUT***: values like `DATETIME` for instance are **NOT** useful for a clustering key, since they have a 3.33ms accuracy only, and thus duplicates can exist. So in such a case, you **still* need an `INT IDENTITY` instead - therefore, I typically use that by default, since frmo my 20+ years of experience, a really usable *natural key* hardly ever really exists .... – marc_s Sep 25, 2017 at 13:27

78

I've been using GUIDs as PKs since 2005. In this distributed database world, it is absolutely the best way to merge distributed data. You can fire and forget merge tables without all the worry of ints matching across joined tables. GUIDs joins can be copied without any worry.

This is my setup for using GUIDs:

1. PK = GUID. GUIDs are indexed similar to strings, so high row tables (over 50 million records) may need table partitioning or other performance techniques. SQL Server is getting extremely efficient, so performance concerns are less and less applicable.
2. PK Guid is NON-Clustered index. Never cluster index a GUID unless it is NewSequentialID. But even then, a server reboot will cause major breaks in ordering.
3. Add ClusterID Int to every table. This is your CLUSTERED Index... that orders your table.
4. Joining on ClusterIDs (int) is more efficient, but I work with 20-30 million record tables, so joining on GUIDs doesn't visibly affect performance. If you want max performance, use the ClusterID concept as your primary key & join on ClusterID.

Here is my Email table...

```
(newsequentialid()) NOT NULL,
    [EmailAddress] NVARCHAR (50)    CONSTRAINT [DF_Email_EmailAddress] DEFAULT
('') NOT NULL,
    [CreatedDate]  DATETIME         CONSTRAINT [DF_Email_CreatedDate] DEFAULT
(getutcdate()) NOT NULL,
    [ClusterID] INT NOT NULL IDENTITY,
    CONSTRAINT [PK_Email] PRIMARY KEY NonCLUSTERED ([EmailID] ASC)
);
GO

CREATE UNIQUE CLUSTERED INDEX [IX_Email_ClusterID] ON [Core].[Email]
([ClusterID])
GO

CREATE UNIQUE NONCLUSTERED INDEX [IX_Email_EmailAddress] ON [Core].[Email]
([EmailAddress] Asc)
```

Share Improve this answer Follow

edited Jul 5, 2019 at 2:48

Dale K
21.6k 13 40 68

answered Mar 31, 2015 at 21:27

Robert J. Good
1,227 10 8

Could you explain the PK_Email constraint? Why you have ... NonClustered(EmailID ASC) instead of ...Nonclustered(ClusterID ASC) ? – Phil Sep 2, 2017 at 15:47

2 You bet. Two main things going on with indexes: 1. Clustered on ClusterID - Orders your table on disk (0% fragmentation). 2. NonClustered on EmailID - Indexes the EmailID field to speed up GUID ID lookups. A GUID field lookup behaves string-ish, so a EmailID lookup would be slow without the index. – Robert J. Good Sep 3, 2017 at 16:28

1 Hi @DaleBurrell, the clustered index is to prevent table fragmentation. Performance gain happens as the table naturally grows in order on disk, with low fragmentation. – Robert J. Good Aug 20, 2019 at 20:10

1 @dariol There are security implications, so drop the newsequentialid() and expose a NewId() Guid if no other choice (definitely not the Int.) I'd recommend a claims based and/or token approach, or even brute-force encryption for any identifiers. In short, avoid exposing any Ids, and avoid any value that can be guessed, or worse +1 to find the next record. – Robert J. Good Dec 23, 2019 at 23:29

1 @RobertJ.Good when you mention "In this distributed database world, it is absolutely the best way to merge distributed data." do you mean you eventually merge the records to a master database? Wondering what happens the the clusterID then, how do you handle duplicates once you merge the "source"? – jfrobishow Jan 28, 2020 at 22:40

1

Well, if your data never reach millions of rows, you are good. If you ask me, i never use GUID as database identity column of any type, including PK even if you force me to design with a shotgun at the head.

Using GUID as primary key is a definitive scaling stopper, and a critical one. I recommend you check database identity and sequence option. Sequence is table independent and may provide a solution for your needs(MS SQL has sequences).

If your tables start reaching some dozens of millions of rows the most, e.g. 50 million you will

Then you need to use partitioning, and be scalable up to half a billion or even 1-2 billion rows. Adding partitioning on the way is not the easiest thing, all read/write statements must include partition column (full app changes!).

These number of course (50 million and 500 million) are for a light selecting useage. If you need to select information in a complex way and/or have lots of inserts/updates/deletes, those could even be 1-2 millions and 50 millions instead, for a very demanding system. If you also add factors like full recovery model, high availability and no maintenance window, common for modern systems, things become extremely ugly.

Note at this point that 2 billion is int limit that looks bad, but int is 4 times smaller and is a sequential type of data, small size and sequential type are the #1 factor for database scalability. And you can use big int which is just twice smaller but still sequential, sequential is what is really deadly important - even more important than size - when to comes to many millions or few billions of rows.

If GUID is also clustered, things are much worst. Just inserting a new row will be actually stored randomly everywhere in physical position.

Even been just a column, not PK or PK part, just indexing it is trouble. From fragmentation perspective.

Having a guid column is perfectly ok like any varchar column as long as you do not use it as PK part and in general as a key column to join tables. Your database must have its own PK elements, filtering and joining data using them - filtering also by a GUID afterwards is perfectly ok.

Share Improve this answer Follow

answered Apr 2, 2021 at 11:00

Stefanos Zilellis
515 3 8

1 I mostly agree but "It Depends". The thing is that even IDENTITY and Date Columns used for Clustered Indexes have sometimes even worse problems with fragmentation because of the horrible habit of doing an INSERT and then following that with an "ExpAnsive" update on the rows just inserted. Massive fragmentation is guaranteed and instantaneous. People have to design correctly even if they avoid Random GUIDs. Oddly enough, Random GUID Clustered Indexes about the Insert/Update fragmentation problem for months at a time instead of it being instantaneous. – Jeff Moden Jun 24, 2021 at 15:04

0

Another reason not to expose an Id in the user interface is that a competitor can see your Id incrementing over a day or other period and so deduce the volume of business you are doing.

Share Improve this answer Follow

answered Mar 3, 2021 at 0:00

MikeW
19

13

I am currently developing an web application with EF Core and here is the pattern I use:

All my classes (tables) have an `int` PK and FK. I then have an additional column of type `Guid` (generated by the C# constructor) with a non clustered index on it.

All the joins of tables within EF are managed through the `int` keys while all the access from outside (controllers) are done with the `Guid` s.

This solution allows to not show the `int` keys on URLs but keep the model tidy and fast.

Share Improve this answer Follow

edited Feb 9, 2021 at 8:56

Dale K
21.6k 13 40 68

answered May 12, 2017 at 8:14

EricImhauser
591 2 6 15

Is there anything you need to do to configure the integer pK as clustered, like data annotations, or is it just automatically configured? – Allen Wang Aug 2, 2018 at 20:07

What the name of the property do you use for Guid one? – Trong Phan May 9, 2019 at 17:50

If you receive the Guid in the Controller, how do you access it if you don't know the associated int? do you do the sequential search in the Guid column? – Cesar Alvarado Diaz Aug 19, 2020 at 20:40

4

This link says it better than I could and helped in my decision making. I usually opt for an int as a primary key, unless I have a specific need not to and I also let SQL server auto-generate/maintain this field unless I have some specific reason not to. In reality, performance concerns need to be determined based on your specific app. There are many factors at play here including but not limited to expected db size, proper indexing, efficient querying, and more. Although people may disagree, I think in many scenarios you will not notice a difference with either option and you should choose what is more appropriate for your app and what allows you to develop easier, quicker, and more effectively (If you never complete the app what difference does the rest make :).

https://web.archive.org/web/20120812080710/http://databases.aspfaq.com/database/what-should-i-choose-for-my-primary-key.html

P.S. I'm not sure why you would use a Composite PK or what benefit you believe that would give you.

Share Improve this answer Follow

edited Jul 9, 2016 at 8:02

JustinStolle
3,882 3 34 46

answered Aug 13, 2012 at 16:22

Matt
911 5 13

Totally agree!! But that means that if I have a GUID as PK or a Composite PK with GUID and other field is going to be the same right? – VAAA Aug 13, 2012 at 16:24

1 BTW this question is one of the most polarizing and debated questions out there and therefore extremely difficult to get an answer for that you will feel 100% comfortable with. Either method comes with trade-offs, so good luck :) – Matt Aug 13, 2012 at 16:43

1

Having sequential ID's makes it a LOT easier for a hacker or data miner to compromise your site and data. Keep that in mind when choosing a PK for a website.

Share Improve this answer Follow

answered Apr 15, 2019 at 17:10

Zonus
1,895 2 19 42

1 Sure, if you know ID numbers are integer you can guess sequentially records in a DB. So if you query a single item, you can say that the next item is pk + 1. If you have random GUIDS, it will not follow a pattern. It would be nearly impossible to query other records than the one you previously queried (And know the PK). – Zonus Jan 28, 2020 at 15:29

4 If a hacker can query your database you're already compromised, I fail to see how sequential id's make the situation worse. – jonnarosey Jan 29, 2020 at 9:37

3 If a user can switch out 1012 for another number and see data they shouldn't then there is a very serious security issue, that issue isn't caused by the primary key choice but it is exacerbated by it. I do take your point, thank you for spelling it out. – jonnarosey Jan 30, 2020 at 15:27

2 You may use a GUID to locate a record at the web page, that is not the PK of the table. Using query parameter in a website should not define how you structure your DB schema. The PK has nothing to do with input and parameters in UI or backend system. – Panos Roditakis Jan 30, 2020 at 21:37

1 This is "security by obscurity", which is not an adequate substitute for proper segregation of data security bounds. A correctly written system won't allow the owner of record 1012 to access record 1013 if they don't own it so the sequentiality matters not. – Caius Jard Jan 10 at 15:58

0

Most of the times it should not be used as the primary key for a table because it really hit the performance of the database. useful links regarding GUID impact on performance and as a primary key.

1. https://www.sqlskills.com/blogs/kimberly/disk-space-is-cheap/
2. https://www.sqlskills.com/blogs/kimberly/guids-as-primary-keys-andor-the-clustering-key/

Share Improve this answer Follow

answered Feb 18, 2019 at 12:08

Asrar Ahmad Ehsan
50 10

1

If you use `GUID` as primary key and create clustered index then I suggest use the default of `NEWSEQUENTIALID()` value for it.

21.6k 13 40 68

12.6k 5 24 33

1 why would you do that? – genuinefafa May 17, 2020 at 19:26