Surrogate vs. natural/business keys [closed]

Asked 11 years, 2 months ago Active 2 years, 4 months ago Viewed 64k times

166

70

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

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

Closed 5 years ago.

Here we go again, the old argument still arises...

Would we better have a business key as a primary key, or would we rather have a surrogate id (i.e. an SQL Server identity) with a unique constraint on the business key field?

Please, provide examples or proof to support your theory.

database database-design primary-key key

edited Sep 16 '10 at 11:20

Peter Mortensen
14.6k 19 89 118

asked Sep 15 '08 at 13:55

Manrico Corazzi
9,399 8 44 62

22 @Joachim Sauer: An argument about whether a thing is subjective may itself be subjective, without this relating in any way to the objectivity or subjectivity of the thing in question. Unless you are prepared to state the exact objective criteria that make something objective. There are things called "open concepts" such as how many hairs it takes to make a beard. One can objectively say that a person with no chin hair has no beard, and one with 5,000 hairs an inch long has a beard, but somewhere in the middle subjective judgment is required to make an objective determination. – ErikE Feb 2 '10 at 18:42

3 @Emtucifor: I'm afraid you've taken my comment a lot more serious than I have, back then. – Joachim Sauer Feb 3 '10 at 7:55

5 @Joachim Sauer: No, I wasn't serious at all. You shot down IainMH (all in fun of course), and I shot down your shooting him down (all in fun of course). Though I admit you put a smiley face and I didn't. :) – ErikE Feb 3 '10 at 19:29

4 It so annoys me when SO moderators think they're doing anyone a favor by closing perfectly good questions like this one. Also, this question seems to satisfy the constructive and "invite sharing experiences over opinions" and "insist that opinion be backed up with facts and references" criteria listed under the "Some subjective questions are allowed" section of stackoverflow.com/help/dont-ask. So come on moderators ... just let people's questions

be already... please. They obviously benefit the asker (or they wouldn't have asked) and many later visitors (like myself) are also interested – Tyler Rick Jun 1 '16 at 23:05

1 @TylerRick Enjoyers of law enforcement, oft be heavy-handed – Jason S Jul 15 '16 at 19:03

19 Answers

92

Both. Have your cake and eat it.

Remember there is nothing special about a primary key, except that it is labelled as such. It is nothing more than a NOT NULL UNIQUE constraint, and a table can have more than one.

If you use a surrogate key, you still want a business key to ensure uniqueness according to the business rules.

answered Sep 15 '08 at 14:48

Ted
1,530 10 5

7 If you have multiple "candidate" keys (fields or same-size collections of fields that are NOT NULL UNIQUE) then you are likely in violation of Boyce-Codd Normal Form. BCNF is beyond 3NF, so not many people worry about it. There are situations, however, where being in BCNF is very helpful. – Alan Sep 17 '08 at 17:19

10 Exactly, Ted! Thank God there are some real database people here. One thing that annoys the heck out of me is when someone creates a table with a surrogate key but no business key, even when one is staring them in the face! – Gregory Higley Oct 17 '08 at 9:30

2 Agreed. The real question should be: Should I add a unique surrogate key to my tables? An entirely other question is what to use for a logical primary key. They are both essentially just non-null unique index constraints. – dkretz Feb 12 '09 at 22:11

5 I find it odd that many comments seem to assert that one cannot setup a relationship without a surrogate key. In many cases, the surrogate key is superfluous. Why add something that brings no value but adds technical debt (and in some cases, causes an otherwise unique result to suddenly become non-unique). – Wil Moore III Sep 28 '10 at 6:55

2 It is more than NOT NULL UNIQUE constraint. The primary key is used as a clustered index which determines the physical order of your data. In general, Integer is easy to balance since it increments sequentially and your data will append to the EOF on disk. If you use less sequential data such as text or GUID(UUID), there will be a lot more disk IO and effort to balance the index, I think that's kind of big difference – Jin May 2 '16 at 20:42

Just a few reasons for using surrogate keys:

118

1. **Stability**: Changing a key because of a business or natural need will negatively affect related tables. Surrogate keys rarely, if ever, need to be changed because there is no meaning tied to the value.
2. **Convention**: Allows you to have a standardized Primary Key column naming convention rather than having to think about how to join tables with various names for their PKs.
3. **Speed**: Depending on the PK value and type, a surrogate key of an integer may be smaller, faster to index and search.

edited Jun 23 '17 at 22:42

answered Sep 15 '08 at 14:06

Jay Shepherd
1,670 1 11 6

2 Now, after reading a lot about surrogate keys and natural keys, I think using surrogate keys is better. But, on my database, natural keys (a NVARCHAR(20)) must be unique. I don't understand how I can get more speed if I need to check every data on that column to don't repeat any value (using a NOT NULL UNIQUE constraint) on each insert. – VansFannel May 18 '16 at 7:39

68

It appears that no one has yet said anything in support of non-surrogate (I hesitate to say "natural") keys. So here goes...

A **disadvantage** of surrogate keys is that they are **meaningless** (cited as an advantage by some, but...). This sometimes forces you to join a lot more tables into your query than should really be necessary. Compare:

```
select sum(t.hours)
from timesheets t
where t.dept_code = 'HR'
and t.status = 'VALID'
and t.project_code = 'MYPROJECT'
and t.task = 'BUILD';
```

against:

```
select sum(t.hours)
from timesheets t
     join departents d on d.dept_id = t.dept_id
     join timesheet_statuses s on s.status_id = t.status_id
     join projects p on p.project_id = t.project_id
     join tasks k on k.task_id = t.task_id
where d.dept_code = 'HR'
and s.status = 'VALID'
and p.project_code = 'MYPROJECT'
and k.task_code = 'BUILD';
```

Unless anyone seriously thinks the following is a good idea?:

```
select sum(t.hours)
from timesheets t
where t.dept_id = 34394
and t.status_id = 89
and t.project_id = 1253
and t.task_id = 77;
```

"But" someone will say, "what happens when the code for MYPROJECT or VALID or HR changes?" To which my answer would be: "why would you **need** to change it?" These aren't "natural" keys in the sense that some outside body is going to legislate that henceforth 'VALID' should be re-coded as 'GOOD'. Only a small percentage of "natural" keys really fall into that category - SSN and Zip code being the usual examples. I would definitely use a meaningless numeric key for tables like Person, Address - but not for **everything**, which for some reason most people here seem to advocate.

See also: my answer to another question

edited May 23 '17 at 11:47
Community ♦
1 1

answered Feb 12 '09 at 14:54

Tony Andrews
112k 18 198 240

14 -1 Natural keys as a primary key have the problem that for every child table you have to add the parent's key which can be composed by more than one field (instead of only one which is the case of a surrogate key) and also the child key. So imagine the following where starting from TABLEA the relationship is 1-0..*: TABLEA PK: ID_A TABLEB PK: ID_A ID_B TABLEC PK: ID_A ID_B ID_C TABLED PK: ID_A ID_B ID_C ID_D. See the problem? The parent key is propagated in the children tables. What would happen if the primary key of TABLEA changes? Now you would have to refactor all the child tables PK too. – Alfredo Osorio Feb 15 '12 at 23:28

9 @Alfredo: yes of course there is a trade-off. However, in my 20+ years of experience I have rarely seen the definition of a table's PK change. If it happened on a regular basis I'd probably avoid natural keys too. In reality, on the extremely rare occasions that this happens I'm prepared to take the hit of the extended impact. – Tony Andrews Feb 16 '12 at 10:18

10 I disagree. It is often the case where some outside body (the customer) legislates that a natural key needs to be edited, and therefore propagated throughout the system. I see this happen regularly. The only way you can be sure that the key won't ever need to change is when it is by definition meaningless. Furthermore, modern databases handle inner joins extremely efficiently, so the potentially large space gains from using surrogates typically outweighs the advantage of not having to do as many inner joins. – TTT Dec 17 '12 at 22:53

8 @TTT: Then the design was weak to begin with. Again, that's where the men separate from the boys: making the right choice of when to use the natural key, and when to use a surrogate. You decide that on a per-table basis, not as a general dogma. – DanMan Oct 17 '13 at 17:25

7 I have also 20+ years of experience, and I second your opinion. I have once a created an oracle datawarehouse with surrogate keys, and data maintenance was like hell. You simply can never directly access your data. you always need to write queries for everything, and that makes surrogate keys simply awful to handle. – SQL Police Feb 19 '16 at 22:58

31

Surrogate keys (typically integers) have the added-value of making your table relations faster, and more economic in storage and update speed (even better, foreign keys do not need to be updated when using surrogate keys, in contrast with business key fields, that do change now and then).

A table's primary key should be used for identifying uniquely the row, mainly for join purposes. Think a Persons table: names can change, and they're not guaranteed unique.

Think Companies: you're a happy Merkin company doing business with other companies in Merkia. You are clever enough not to use the company name as the primary key, so you use Merkia's government's unique company ID in its entirety of 10 alphanumeric characters. Then Merkia changes the company IDs because they thought it would be a good idea. It's ok, you use your db engine's cascaded updates feature, for a change that shouldn't involve you in the first place. Later on, your business expands, and now you work with a company in Freedonia. Freedonian company id are up to 16 characters. You need to enlarge the company id primary key (also the foreign key fields in Orders, Issues, MoneyTransfers etc), adding a Country field in the primary key (also in the foreign keys). Ouch! Civil war in Freedonia, it's split in three countries. The country name of your associate should be changed to the new one; cascaded updates to the rescue. BTW, what's your primary key? (Country, CompanyID) or (CompanyID, Country)? The latter helps joins, the former avoids another index (or perhaps many, should you want your Orders grouped by country too).

All these are not proof, but an indication that a surrogate key to uniquely identify a row for all uses, including join operations, is preferable to a business key.

edited Feb 12 '09 at 14:23

answered Sep 15 '08 at 14:04

tzot
69.2k 22 113 185

You win all the internets with the coolest looking username! – Iain Holder Sep 15 '08 at 14:05

1 That's pretty much what a downvote is: "I don't agree with this." – jcollum Aug 4 '09 at 17:23

4 The tooltip of the down arrow says "This answer is not useful", not "I don't agree with this". Perhaps in this specific answer the meanings are close, but they are not generally the same. – tzot Mar 3 '10 at 2:11

1 If somebody thinks that your answer is wrong, then he(/she) will also think that it leads the questioner into the wrong direction (opposite of the right direction), and will therefore judge your answer as being even worse than "unhelpful", justifying in his(/her) mind a downvote. – Erwin Smout Nov 2 '11 at 13:42

1 Yup, surrogate keys are a disease. One leaks into the wild and you use it as a pkey so now you need your own surrogate key. Then your key leaks into the wild (say through a url) and the disease spreads. – Samuel Danielson Oct 5 '12 at 15:16

31

Surrogate key will NEVER have a reason to change. I cannot say the same about the natural keys. Last names, emails, ISBN nubmers - they all can change one day.

answered Sep 15 '08 at 13:59

Rimantas

1,265 8 6

25

I hate surrogate keys in general. They should only be used when there is no quality natural key available. It is rather absurd when you think about it, to think that adding meaningless data to your table could make things better.

Here are my reasons:

1. When using natural keys, tables are clustered in the way that they are most often searched thus making queries faster.
2. When using surrogate keys you must add unique indexes on logical key columns. You still need to prevent logical duplicate data. For example, you can't allow two Organizations with the same name in your Organization table even though the pk is a surrogate id column.
3. When surrogate keys are used as the primary key it is much less clear what the natural primary keys are. When developing you want to know what set of columns make the table unique.
4. In one to many relationship chains, the logical key chains. So for example, Organizations have many Accounts and Accounts have many Invoices. So the logical-key of Organization is OrgName. The logical-key of Accounts is OrgName, AccountID. The logical-key of Invoice is OrgName, AccountID, InvoiceNumber.

   When surrogate keys are used, the key chains are truncated by only having a foreign key to the immediate parent. For example, the Invoice table does not have an OrgName column. It only has a column for the AccountID. If you want to search for invoices for a given organization, then you will need to join the Organization, Account, and Invoice tables. If you use logical keys, then you could Query the Organization table directly.
5. Storing surrogate key values of lookup tables causes tables to be filled with meaningless integers. To view the data, complex views must be created that join to all of the lookup tables. A lookup table is meant to hold a set of acceptable values for a column. It should not be codified by storing an integer surrogate key instead. There is nothing in the normalization rules that suggest that you should store a surrogate integer instead of the value itself.
6. I have three different database books. Not one of them shows using surrogate keys.

edited Dec 26 '15 at 7:56

shA.t

13.6k 5 41 80

answered Sep 28 '09 at 18:11

Ken

1,259 10 10

7 I hate surrogate keys, except when they are necessary. They are necessary when the enterprise uses a natural key that is subject to a lot of errors, and are unwilling to tolerate a database that's impacted by those errors. – Walter Mitty Sep 18 '10 at 11:06

24 -1: I have written and maintained dozens of applications. The ones with the most data-related problems were those using natural keys. – Falcon Apr 6 '11 at 13:58

2 google.pl/search?q=surrogate+keys&tbm=bks&tbo=1 ;) – Robert Bak Feb 7 '12 at 19:14

6 Some of your points assume the surrogate key has to be the PK or has to be the clustered column--not true. Your points 1 and 5 ignore the fact that integers are 4 bytes and natural keys are almost always many, many more bytes. And, each nonclustered index must repeat the bytes of those natural keys that are in the clustered index, so the tables and indexes in your natural key database are going to have far, far fewer rows per page, which translates in to *much worse* read performance, which makes queries *slower*, not faster. – ErikE Oct 26 '13 at 2:05

3 Another reason against natural keys (examples: Atomic Numbers, VINs, etc), the business logic can change that increases the type of data. For example - Before: Tracking charges of Atoms, After: Tracking charges of Atoms and Compounds. Before: Tracking Motor Vehicles for load capacity. After: Adding planes, boats, bikes and people for load carrying capacity. – forforf Dec 18 '13 at 21:15

17

I want to share my experience with you on this endless war :D on natural vs surrogate key dilemma. I think that **both** surrogate keys (artificial auto-generated ones) and natural keys (composed of column(s) with domain meaning) have **pros** and **cons**. So depending on your situation, it might be more relevant to choose one method or the other.

As it seems that many people present surrogate keys as the almost perfect solution and natural keys as the plague, I will focus on the other point of view's arguments:

Disadvantages of surrogate keys

Surrogate keys are:

1. Source of performance problems:
   - They are usually implemented using auto-incremented columns which mean:
     - A round-trip to the database each time you want to get a new Id (I know that this can be improved using caching or [seq]hilo alike algorithms but still those methods have their own drawbacks).
     - If one-day you need to move your data from one schema to another (It happens quite regularly in my company at least) then you might encounter Id collision problems. And Yes I know that you can use UUIDs but those lasts requires 32 hexadecimal digits! (If you care about database size then it can be an issue).
     - If you are using one sequence for all your surrogate keys then - for sure - you will end up with contention on your database.

2. Error prone. A sequence has a max_value limit so - as a developer - you have to put attention to the following points:
   - You must cycle your sequence (when the max-value is reached it goes back to 1,2,...).
   - If you are using the sequence as an ordering (over time) of your data then you must handle the case of cycling (column with Id 1 might be newer than row with Id max-value - 1).
   - Make sure that your code (and even your client interfaces which should not happen as it supposed to be an internal Id) supports 32b/64b integers that you used to store your sequence values.
3. They don't guarantee non duplicated data. You can always have 2 rows with all the same column values but with a different generated value. For me this is **THE** problem of surrogate keys from a database design point of view.
4. More in Wikipedia...

Myths on natural keys

1. Composite keys are less inefficient than surrogate keys. No! It depends on the used database engine:
   - Oracle
   - MySQL
2. Natural keys don't exist in real-life. Sorry but they do exist! In aviation industry, for example, the following tuple will be always unique regarding a given **scheduled** flight (airline, departureDate, flightNumber, operationalSuffix). More generally, when a set of business data is guaranteed to be unique by a given **standard** then this set of data is a [good] natural key candidate.
3. Natural keys "pollute the schema" of child tables. For me this is more a feeling than a real problem. Having a 4 columns primary-key of 2 bytes each might be more efficient than a single column of 11 bytes. Besides, the 4 columns can be used to query the child table directly (by using the 4 columns in a where clause) without joining to the parent table.

Conclusion

Use natural keys when it is relevant to do so and use surrogate keys when it is better to use them.

Hope that this helped someone!

edited May 23 '17 at 11:55
Community ♦
1 1

answered Dec 27 '13 at 16:14
mwnsiri
689 7 10

3 What happens when the scheduled flight's departureDate is rescheduled? Do you have to track down all the related entities and delete the keys, or do you actually update all the keys in the related entities? Or are you dealing with a simple, singular table (possibly not even 3NF)? – code4life May 19 '16

at 0:48

Excelent point @code4life – forcewill Jul 15 '16 at 11:10

@code4life: That's where the operationalSuffix jumps in. In order to keep the same flightNumber so we avoid client confusion, we add just a suffix (i.e 'D' for example). – mwnsiri Jul 21 '16 at 12:37

"You can always have 2 rows with all the same column values but with a different generated value" so just put a unique or composite unique constraint on your columns. – wha7ever Sep 13 at 19:46

15

Alway use a key that has no business meaning. It's just good practice.

EDIT: I was trying to find a link to it online, but I couldn't. However in 'Patterns of Enterprise Archtecture' [Fowler] it has a good explanation of why you shouldn't use anything other than a key with no meaning other than being a key. It boils down to the fact that it should have one job and one job only.

edited Sep 15 '08 at 14:43

answered Sep 15 '08 at 13:58

Iain Holder
10.9k 10 60 84

22 Martin Fowler may be many things, but he isn't an authority on database design. – Tony Andrews Jun 2 '09 at 12:25

I think you should provide some reasoning before reaching the conclusion. – Arne Evertsson Dec 14 '11 at 5:50

4 @ArneEvertsoon The reason is in there. 'It boils down to the fact that it should have one job and one job only.' Single responsibility. – Iain Holder Dec 14 '11 at 10:50

10

Surrogate keys are quite handy if you plan to use an ORM tool to handle/generate your data classes. While you can use composite keys with some of the more advanced mappers (read: hibernate), it adds some complexity to your code.

(Of course, database purists will argue that even the notion of a surrogate key is an abomination.)

I'm a fan of using uids for surrogate keys when suitable. The major win with them is that you know the key in advance e.g. you can create an instance of a class with the ID already set and guaranteed to be unique whereas with, say, an integer key you'll need to default to 0 or -1 and update to an appropriate value when you save/update.

UIDs have penalties in terms of lookup and join speed though so it depends on the application in question as to whether they're desirable.

answered Sep 15 '08 at 14:19

Derek Lawless
2,364 2 15 12

6

Using a surrogate key is better in my opinion as there is zero chance of it changing. Almost anything I can think of which you might use as a natural key could change (disclaimer: not always true, but commonly).

An example might be a DB of cars - on first glance, you might think that the licence plate could be used as the key. But these could be changed so that'd be a bad idea. You wouldnt really want to find that out *after* releasing the app, when someone comes to you wanting to know why they can't change their number plate to their shiny new personalised one.

answered Sep 15 '08 at 14:02

Mark Embling
10.7k 5 35 52

1 Unfortunately cars do have a natural key that doesn't change: the VIN (at least in America...) – jcollum Aug 4 '09 at 17:22

@jcollum Yes ok, that is a fair point. My opinion still stands though, my example was not necessarily as good as it could be. – Mark Embling Aug 10 '09 at 8:57

2 A list of languages would be an example for a natural key, when you base it on ISO codes. So if you then wanted to load content from a table in a certain language, you wouldn't need to join in the `languages` table since the language code (ID) is already in the `texts` table. – DanMan Oct 17 '13 at 17:40

@DanMan I have to agree with you there. There are always going to be some examples which work better with a natural key. Rules or common approaches are never absolute, and that is one example I would 100% go with your approach :-) – Mark Embling Oct 17 '13 at 18:27

5

Always use a single column, surrogate key if at all possible. This makes joins as well as inserts/updates/deletes much cleaner because you're only responsible for tracking a single piece of information to maintain the record.

Then, as needed, stack your business keys as unique contraints or indexes. This will keep you data integrity intact.

Business logic/natural keys can change, but the phisical key of a table should NEVER change.

answered Sep 15 '08 at 14:35

user7658
864 7 7

4

On a datawarehouse scenario I believe is better to follow the surrogate key path. Two reasons:

- You are independent of the source system, and changes there --such as a data type change-- won't affect you.
- Your DW will need less physical space since you will use only integer data types for your surrogate keys. Also your indexes will work better.

edited Nov 2 '11 at 12:24

answered Sep 15 '08 at 14:01

Santiago Cepas
3,552 2 20 28

2

As a reminder it is not good practice to place clustered indices on random surrogate keys i.e. GUIDs that read XY8D7-DFD8S, as they SQL Server has no ability to physically sort these data. You should instead place unique indices on these data, though it may be also beneficial to simply run SQL profiler for the main table operations and then place those data into the Database Engine Tuning Advisor.

See thread @ http://social.msdn.microsoft.com/Forums/en-us/sqlgetstarted/thread/27bd9c77-ec31-44f1-ab7f-bd2cb13129be

answered Jun 27 '12 at 14:44

Bryan Swan
2,116 12 26

I'm pretty sure SQL Server *can* sort GUIDs. – Michael Green Sep 14 '16 at 12:11

This is not accurate, while they can evaluate the GUID, the resulting sort is not nonsensical to a human. stackoverflow.com/questions/7810602/… – Bryan Swan Oct 8 '16 at 15:13

1 A true statement, but quite different to "SQL Server has no ability to physically sort these." – Michael Green Oct 10 '16 at 0:02

2

This is one of those cases where a surrogate key pretty much *always* makes sense. There are cases where you either choose what's best for the database or what's best for your object model, but in both cases, using a meaningless key or GUID is a better idea. It makes indexing easier and faster, and it is an identity for your object that doesn't change.

edited Sep 14 '16 at 14:07

Michael Green

answered Sep 15 '08 at 14:16

Charles Graham


1,212 1 16 22


13.4k 13 40 56

2

Case 1: Your table is a **lookup table** with less than 50 types (inserts)

Use **business/natural keys**. For Example:

```
Table: JOB with 50 inserts
CODE (primary key)       NAME               DESCRIPTION
PRG                      PROGRAMMER         A programmer is writing code
MNG                      MANAGER            A manager is doing whatever
CLN                      CLEANER            A cleaner cleans
...............
joined with
Table: PEOPLE with 100000 inserts

foreign key JOBCODE in table PEOPLE
looks at
primary key CODE in table JOB
```

Case 2: Your table is a **table with thousands of inserts**

Use **surrogate/autoincrement keys**. For Example:

```
Table: ASSIGNMENT with 1000000 inserts
joined with
Table: PEOPLE with 100000 inserts

foreign key PEOPLEID in table ASSIGNMENT
looks at
primary key ID in table PEOPLE (autoincrement)
```

In the first case:

- You can select all programmers in table PEOPLE without use of join with table JOB, but just with: "SELECT * FROM PEOPLE WHERE JOBCODE = 'PRG'"

In the second case:

- Your database queries are faster because your primary key is an integer
- You don't need to bother yourself with finding the next unique key because the database itself gives you the next autoincrement.

answered Oct 24 '12 at 12:48

Stefanos Kargas
4,238 19 57 85

2

Surrogate keys can be useful when business information can change or be identical. Business names don't have to be unique across the country, after all. Suppose you deal with two businesses named Smith Electronics, one in Kansas and one in Michigan. You can distinguish them by address, but that'll change. Even the state can change; what if Smith Electronics of Kansas City, Kansas moves across the river to Kansas City, Missouri? There's no obvious way of keeping these businesses distinct with natural key information, so a surrogate key is very useful.

Think of the surrogate key like an ISBN number. Usually, you identify a book by title and author. However, I've got two books titled "Pearl Harbor" by H. P. Willmott, and they're definitely different books, not just different editions. In a case like that, I could refer to the looks of the books, or the earlier versus the later, but it's just as well I have the ISBN to fall back on.

answered Feb 12 '09 at 15:52

David Thornley
51.7k 8 85 144

1 I think I have to disagree with your example here. An ISBN number is an attribute of a book. A surrogate key is independent of the rest of the row data, therefore this position would advocate using a separate surrogate key for a book table, even though the ISBN already uniquely identifies every book. – Christopher Cashell Mar 31 '10 at 21:28

Alternately, think of the ISBN as a surrogate key itself. It's an identifier with no meaning, just a code that is applied to a specific book. If you're making a books table, the ISBN may as well be the primary key (assuming you have and always will have one book per row). – David Thornley Mar 31 '10 at 22:08

@Christopher Cashell - Came across this post from a year ago but I thought add something. ISBNs are not guaranteed to be unique and can have duplicates. I have a friend that worked at a library for a number of years and they often ran across books with duplicate ISBNs.The problem is that the uniqueness of the ISBN is incumbent on the publisher rather than one body that ensures that all numbers for all publications are unique and those publishers did not always have their act together. – Thomas Apr 22 '11 at 4:49

2 Came across this post from a year ago and wanted to mention that ISBN are in fact natural keys. There is meaning baked into the key value itself unlike a surrogate key. For example, part of the key identifies the publisher. In addition, as I mentioned above, they are not guaranteed to be unique. They are *supposed* to be unique but that uniqueness comes from the publishers and they were not always perfect. – Thomas Apr 22 '11 at 4:52

Technically, corporations cannot move between states; what happens is that a new corporation is created in the new state and the assets are transferred. That works for database information too. – Warren Dew May 4 '14 at 1:28

1

Maybe not completely relevant to this topic, but a headache I have dealing with surrogate keys. Oracle pre-delivered analytics creates auto-generated SKs on all of its dimension tables in the warehouse, and it also stores those on the facts. So, anytime they (dimensions) need to be reloaded as new columns are added or need to be populated for all items in the dimension, the SKs assigned during the update makes the SKs out of sync with the original values stored to the fact, forcing a complete reload of all fact tables that join to it. I would prefer that even if the SK was a meaningless number, there would be some way that it could not change for original/old records. As many know, out-of-the box rarely serves an organization's needs, and we have to customize constantly. We now have 3yrs worth of data in our warehouse, and complete reloads from the Oracle Financial systems are very large. So in my case, they are not generated from data entry, but added in a warehouse to help reporting performance. I get it, but ours do change, and it's a nightmare.

answered Apr 24 '13 at 15:02

lrb

11 1

1

Horse for courses. To state my bias; I'm a developer first, so I'm mainly concerned with giving the users a working application.

I've worked on systems with natural keys, and had to spend a lot of time making sure that value changes would ripple through.

I've worked on systems with only surrogate keys, and the only drawback has been a lack of denormalised data for partitioning.

Most traditional PL/SQL developers I have worked with didn't like surrogate keys because of the number of tables per join, but our test and production databases never raised a sweat; the extra joins didn't affect the application performance. With database dialects that don't support clauses like "X inner join Y on X.a = Y.b", or developers who don't use that syntax, the extra joins for surrogate keys do make the queries harder to read, and longer to type and check: see @Tony Andrews post. But if you use an ORM or any other SQL-generation framework you won't notice it. Touch-typing also mitigates.

answered Feb 9 '12 at 22:53

WillC

608 9 18

> Also; if you want to really drive home that the surrogate keys are just that, start them at a random large number and increment the sequences by 3+ rather than by 1. Or use the same sequence to generate values for more than one key. – WillC Feb 9 '12 at 23:05

0

In the case of point in time database it is best to have combination of surrogate and natural keys. e.g. you need to track a member information for a club. Some attributes of a member never change. e.g Date of Birth but name can change. So create a Member table

with a member_id surrogate key and have a column for DOB. Create another table called person name and have columns for member_id, member_fname, member_lname, date_updated. In this table the natural key would be member_id + date_updated.

answered Mar 7 '09 at 12:11

kanad