Composite primary key or not?

Asked 8 years, 9 months ago Active 6 years, 6 months ago Viewed 20k times

18

13

Here's what's confusing me. I often have composite primary keys in database tables. The bad side of that approach is that I have pretty extra work when I delete or edit entries. However, I feel that this approach is in the spirit of database design.

On the other side, there are friends of mine, who never use composite keys, but rather introduce another 'id' column in a table, and all other keys are just FKs. They have much less work while coding delete and edit procedures. However, I do not know how they preserve uniqueness of data entries.

For example:
Way 1

```
create table ProxUsingDept (
    fkProx int references Prox(ProxID) NOT NULL,
    fkDept int references Department(DeptID) NOT NULL,
    Value int,
    PRIMARY KEY(fkProx,fkDept)
)
```

Way 2

```
create table ProxUsingDept (
        ID int NOT NULL IDENTITY PRIMARY KEY
        fkProx int references Prox(ProxID) NOT NULL,
        fkDept int references Department(DeptID) NOT NULL,
        Value int
)
```

Which way is better? What are the bad sides of using the 2nd approach? Any suggestions?

database-design composite-primary-key

asked Jan 19 '11 at 15:35

sandalone
24.3k 56 190 305

Check out stackoverflow.com/questions/159087/… – TechTravelThink Jan 19 '11 at 15:46

4 Answers

26

I personally **prefer** your 2nd approach (and would use it almost 100% of the time) - introduce a surrogate `ID` field.

Why?

- makes life a lot easier for any tables referencing your table - the JOIN conditions are **much simpler** with just a single ID column (rather than 2, 3, or even more columns that you need to join on, all the time)
- makes life a lot easier since any table referencing your table only needs to carry a single `ID` as foreign key field - not several columns from your compound key
- makes life a lot easier since the database can handle the creation of unique `ID` column (using `INT IDENTITY`)

> However, I do not know how they preserve uniqueness of data entries.

Very simple: put a UNIQUE INDEX on the compound columns that you would otherwise use as your primary key!

```
CREATE UNIQUE INDEX UIX_WhateverNameYouWant
   ON dbo.ProxUsingDept(fkProx, fkDept)
```

Now, your table guarantees there will never be a duplicate pair of `(fkProx, fkDept)` in your table - problem solved!

answered Jan 19 '11 at 15:54

marc_s
611k 139 1170 1301

4 @askmo. So let me get this right. You would add (a) an autoincrement column (b) and additional index, for the sake of simpler coding ? No concerns about the complete non-dat or non-business requirement ? No concerns re the negative performance ? The lost navigaton (see Larry's answer) ? – PerformanceDBA Jan 25 '11 at 6:59

4 @marc_s. This is what happens when you let people who are concerned about their personal likes and dislikes; their ease of coding, design "databases". Instead of professional database designers who are concerned with overall use, performance, standards, Relational power. The industry is in a very sad state. – PerformanceDBA Jan 25 '11 at 7:06

8 @PerformanceDBA: I disagree with your statements - I will **always** prefer a simple INT surrogate key over a complicated, "natural" composite key, for the exact reasons I mentioned - ease of use and ease of creating foreign key relations. I've seen too many times that folks just gave up creating FK constraints because their PK was a composite monster of 6, 10 or even more columns.... – marc_s Jan 25 '11 at 8:18

3 (cont.) Natural keys--*especially* natural composite keys--make the table more brittle. That's a fact. *Every* manner of constraint makes a table more brittle, but natural composite keys doubly so, as you're combining a technical constraint (PK) with a business constraint (user-specified uniqueness). Requirements can *and do* change, and I would suspect that you've been in more than one situation where a natural composite key has suddenly become something that is no longer suitable as a primary key. It's disingenuous to say that "real" DBA's are the ones concerned with overall use (or, rather – Adam Robinson Jan 25 '11 at 13:57

6 @PerformanceDBA: Thank you for your respectful expression of your own personal opinion. You can rest assured that anyone who reads this will certainly give it all of the consideration that it's due. – Adam Robinson Jan 27 '11 at 13:33

17

You ask the following questions:

> However, I do not know how they preserve uniqueness of data entries.

Uniqueness can be preserved by declaring a separate composite UNIQUE index on columns that would otherwise form the natural primary key.

> Which way is better?

Different people have different opinions, sometimes strongly held. I think you will find that more people use surrogate integer keys (not that that makes it the "right" solution).

> What are the bad sides of using the 2nd approach?

Here are some of the disadvantages to using a surrogate key:

1. You require an additional index to maintain the unique-ness of the natural primary key.
2. You sometimes require additional JOINs to when selecting data to get the results you want (this happens when you could satisfy the requirements of the query using only the columns in the composite natural key; in this case you can use the foreign key columns rather than JOINing back to the original table).

answered Jan 19 '11 at 15:57

Larry Lustig
41.4k 12 87 135

1 Good answer. Always, not sometimes. (2) might need explanation for some readers, and all the responders. Great-granchild to parent is lost; IDs require joining great-grandchild to grandchild to child to parent. – PerformanceDBA Jan 25 '11 at 7:02

1 +1 for your second argument against using surrogate keys. (not that I'm against surrogate keys - just the opposite actually) – goku_da_master Sep 13 '13 at 21:28

Stated more fair than I might have. The concept of your DB as a fortress seems lost in the new frameworks. Yes, simple, surrogate keys make ORMs easier to set up. But composite keys and, especially, composite natural keys make data analysis easier and raw SQL leaner. With the work I do, working with an ID column on every table is mega-tedious. Apps will come and go but the data stays and has the most value. Simple keys vs composite keys, and surrogate keys vs natural keys are arguments that will never be resolved. Read "Enterprise Rails" by Dan Chak. Some get it, some don't, others ignore it. – juanitogan Jan 27 '14 at 17:00

"Enterprise Rails" by Dan Chak is now available online. Chapter 8 deals with Surrogate vs. Composite Keys: dan.chak.org/enterprise-rails/… – user2029904 Nov 24 '14 at 20:20

0

There are cases like M:N join tables where composite keys make most sense (and if the nature or the M:N link changes, you'll have to rework this table anyway).

answered Jan 19 '11 at 15:55

9000
32.2k 8 49 89

Actually I think you meant about M:M joins where composite keys do have the most sense, but it's better practise to introduce artificial (surrogate) key. It's easier for managing app development. – sbrbot Jun 14 '15 at 16:45

-3

I know it is a very long time since this post was made. But I had to come across a similar situation regarding the composite key so I am posting my thoughts.

Let's say we have two tables T1 and T2.

T1 has the columns C1 and C2.

T2 has the columns C1, C2 and C3

C1 and C2 are the composite primary keys for the table T1 and foreign keys for the table T2.

Let's assume we used a surrogate key for the Table T1 (T1_ID) and used that as a Foreign Key in table T2, if the values of C1 and C2 of the Table T1 changes, it is additional work to enforce the referential ingegrity constraint on the table T2 as we are looking only at the

surrogate key from Table T1 whose value didn't change in Table T1. This could be one issue with second approach.

answered Apr 29 '13 at 19:58

user2315931
1 1

4 In a normalised database T2 wouldn't contain C1 and C2 so I'm afraid I don't think this is a valid argument. – Caltor Jan 8 '14 at 13:09