One or Two Primary Keys in Many-to-Many Table?

Asked 16 years, 3 months ago Modified 3 years, 1 month ago

Viewed 15k times

16

I have the following tables in my database that have a many-to-many relationship, which is expressed by a connecting table that has foreign keys to the primary keys of each of the main tables:

- Widget: WidgetID (PK), Title, Price
- User: UserID (PK), FirstName, LastName

Assume that each User-Widget combination is unique. I can see two options for how to structure the connecting table that defines the data relationship:

1. UserWidgets1: UserWidgetID (PK), WidgetID (FK), UserID (FK)
2. UserWidgets2: WidgetID (PK, FK), UserID (PK, FK)

Option 1 has a single column for the Primary Key. However, this seems unnecessary since the only data being stored in the table is the relationship between the two primary tables, and this relationship itself can form a unique key. Thus leading to option 2, which has a two-column primary key, but loses the one-column unique identifier that option 1 has. I could also optionally add a

two-column unique index (WidgetID, UserID) to the first table.

Is there any real difference between the two performance-wise, or any reason to prefer one approach over the other for structuring the UserWidgets many-to-many table?

database data-structures indexing schema

Share

Improve this question

Follow

asked Sep 2, 2008 at 5:21

Yaakov Ellis

41.4k • 27 • 134 • 184

The indexes you need are dictated by your query requirements, not your schema design. – dkretz Nov 2, 2011 at 7:32

9 Answers

Sorted by: Highest score (default)

24

You only have one primary key in either case. The second one is what's called a compound key. There's no good reason for introducing a new column. In practise, you will have to keep a unique index on all candidate keys. Adding a new column buys you nothing but maintenance overhead.

Go with option 2.

Share Improve this answer

Follow

answered Sep 2, 2008 at 5:28

Apocalisp

35k ● 8 ● 108 ● 158

A primary key can be compund - the terms are not exclusive. – paulmurray Mar 22, 2009 at 2:34

2 @paulmurray: I believe the answer above says that you have a primary key in either case, including the case in which you have a compound key. Did you have something to add to that? – Apocalisp Mar 22, 2009 at 5:32

7

Option 2 uses a simple compund key, option 1 uses a surrogate key. Option 2 is preferred in most scenarios and is close to the relational model in that it is a good candidate key.

There are situations where you may want to use a surrogate key (Option 1)

1. You are not certain that the compound key is a good candidate key over time. Particularly with temporal data (data that changes over time). What if you wanted to add another row to the UserWidget table with the same UserId and WidgetId? Think of Employment(EmployeeId,EmployeeId) - it would work in most cases except if someone went back to work for the same employer at a later date
2. If you are creating messages/business transactions or something similar that requires an easier key to use for integration. Replication maybe?

3. If you want to create your own auditing mechanisms (or similar) and don't want keys to get too long.

As a rule of thumb, when modeling data you will find that most associative entities (many to many) are the result of an event. Person takes up employment, item is added to basket etc. Most events have a temporal dependency on the event, where the date or time is relevant - in which case a surrogate key may be the best alternative.

So, take option 2, but make sure that you have the complete model.

Share Improve this answer

Follow

edited Nov 14, 2021 at 7:28

tutuDajuju

10.8k ● 6 ● 69 ● 95

answered Sep 2, 2008 at 11:49

Simon Munro

5,419 ● 7 ● 34 ● 40

6

Personally, I *would* have the synthetic/surrogate key column in many-to-many tables for the following reasons:

- If you've used numeric synthetic keys in your entity tables then having the same on the relationship tables maintains consistency in design and naming convention.
- It may be the case in the future that the many-to-many table itself becomes a parent entity to a

subordinate entity that needs a unique reference to an individual row.

- It's not really going to use that much additional disk space.

The synthetic key is not a replacement to the natural/compound key nor becomes the `PRIMARY KEY` for that table just because it's the first column in the table, so I partially agree with the Josh Berkus article. However, I don't agree that natural keys are always good candidates for `PRIMARY KEY's` and certainly should not be used if they are to be used as foreign keys in other tables.

Share Improve this answer

Follow

answered Sep 2, 2008 at 10:01

Guy

9,826 ● 7 ● 39 ● 43

I realize this was answered a long time ago, but wouldn't the compound key still be a unique reference to an individual row for a parent table (your point 2)? – crush May 9, 2016 at 19:09

1 @crush - yes, it would be unique, but to create a constraint on a compound key is fugly/inconsistent across platforms. I prefer to be explicit and consistent. Every table has an identity column. – Guy May 12, 2016 at 9:28

3

I agree with the previous answers but I have one remark to add. If you want to add more information to the relation and allow more relations between the same two entities you need option one.

For example if you want to track all the times user 1 has used widget 664 in the userwidget table the userid and widgetid isn't unique anymore.

Share Improve this answer

Follow

answered Sep 2, 2008 at 8:17

Mendelt

37.5k ● 6 ● 75 ● 97

2

What is the benefit of a primary key in this scenario? Consider the option of no primary key: UserWidgets3: WidgetID (FK), UserID (FK)

If you want uniqueness then use either the compound key (UserWidgets2) or a uniqueness constraint.

The usual performance advantage of having a primary key is that you often query the table by the primary key, which is fast. In the case of many-to-many tables you don't usually query by the primary key so there is no performance benefit. Many-to-many tables are queried by their foreign keys, so you should consider adding indexes on WidgetID and UserID.

Share Improve this answer

Follow

answered Sep 2, 2008 at 5:33

liammclennan

5,368 ● 3 ● 35 ● 31

2

Option 2 is the correct answer, unless you have a really good reason to add a surrogate numeric key (which you have done in option 1).

Surrogate numeric key columns are not 'primary keys'. Primary keys are technically one of the combination of columns that uniquely identify a record within a table.

Anyone building a database should read this article http://it.toolbox.com/blogs/database-soup/primary-keyvil-part-i-7327 by Josh Berkus to understand the difference between surrogate numeric key columns and primary keys.

In my experience the only real reason to add a surrogate numeric key to your table is if your primary key is a compound key and needs to be used as a foreign key reference in another table. Only then should you even think to add an extra column to the table.

Whenever I see a database structure where every table has an 'id' column the chances are it has been designed by someone who doesn't appreciate the relational model and it will invariably display one or more of the problems identified in Josh's article.

Share Improve this answer

Follow

answered Sep 2, 2008 at 6:12

andy47

913 ● 2 ● 9 ● 18

1

I would go with both.

Hear me out:

The compound key is obviously the nice, correct way to go in so far as reflecting the meaning of your data goes. No question.

However: I have had all sorts of trouble making hibernate work properly unless you use a single generated primary key - a surrogate key.

So I would use a logical and physical data *model*. The logical one has the compound key. The physical model - which implements the logical model - has the surrogate key and foreign keys.

Share Improve this answer

Follow

answered Mar 22, 2009 at 2:40

paulmurray

3,413 ● 1 ● 23 ● 17

0

Since each User-Widget combination is unique, you should represent that in your table by making the combination unique. In other words, go with option 2. Otherwise you may have two entries with the same widget and user IDs but different user-widget IDs.

Share Improve this answer

Follow

answered Sep 2, 2008 at 5:25

Kyle Cronin

79k ● 45 ● 151 ● 167

0

The userwidgetid in the first table is not needed, as like you said the uniqueness comes from the combination of the widgetid and the userid.

I would use the second table, keep the foriegn keys and add a unique index on widgetid and userid.

So:

```
userwidgets( widgetid(fk), userid(fk),
             unique_index(widgetid, userid)
)
```

There is some preformance gain in not having the extra primary key, as the database would not need to calculate the index for the key. In the above model though this index (through the unique_index) is still calculated, but I believe that this is easier to understand.

Share Improve this answer

Follow

answered Sep 2, 2008 at 5:29

roo

7,196 ● 9 ● 42 ● 45