Represent Ordering in a Relational Database

Asked 16 years, 4 months ago Modified 6 years ago Viewed 6k times

I have a collection of objects in a database. Images in a photo gallery, products in a catalog, chapters in a book, etc. Each object is represented as a row. I want to be able to arbitrarily order these images, storing that ordering in the database so when I display the objects, they will be in the right order.

For example, let's say I'm writing a book, and each chapter is an object. I write my book, and put the chapters in the following order:

> Introduction, Accessibility, Form vs. Function, Errors, Consistency, Conclusion, Index

It goes to the editor, and comes back with the following suggested order:

> Introduction, Form, Function, Accessibility, Consistency, Errors, Conclusion, Index

How can I store this ordering in the database in a robust, efficient way?

I've had the following ideas, but I'm not thrilled with any of them:

1. Array. Each row has an ordering ID, when order is changed (via a removal followed by an insertion), the order IDs are updated. This makes retrieval easy, since it's just `ORDER BY`, but it seems easy to break.

   > `// REMOVAL`
   > `UPDATE ... SET orderingID=NULL WHERE orderingID=removedID`
   > `UPDATE ... SET orderingID=orderingID-1 WHERE orderingID > removedID`
   > `// INSERTION`
   > `UPDATE ... SET orderingID=orderingID+1 WHERE orderingID > insertionID`
   > `UPDATE ... SET orderID=insertionID WHERE ID=addedID`

2. Linked list. Each row has a column for the id of the next row in the ordering. Traversal seems costly here, though there may by some way to use `ORDER BY` that I'm not thinking of.
3. Spaced array. Set the orderingID (as used in #1) to be large, so the first object is 100, the second is 200, etc. Then when an insertion happens, you just place it at `(objectBefore + objectAfter)/2`. Of course, this would need to be rebalanced occasionally, so you don't have things too close together (even with floats, you'd eventually run into rounding errors).

None of these seem particularly elegant to me. Does anyone have a better way to do it?

sql database django django-models

Share Improve this question Follow

edited Mar 29, 2009 at 14:48

asked Aug 21, 2008 at 23:01

tghw
25.3k ● 13 ● 71 ● 97

11 Answers

Sorted by: Highest score (default)

7

An other alternative would be (if your RDBMS supports it) to use columns of type array. While this breaks the normalization rules, it can be useful in situations like this. One database which I know about that has arrays is PostgreSQL.

Share Improve this answer Follow

answered Aug 22, 2008 at 5:32

Grey Panther
13.1k ● 7 ● 49 ● 67

2 I don't understand this solution which is apparently the better answer. Could you elaborate a bit on how to use the array for each row? Thanks – Pierre Dec 18, 2015 at 16:52

4

The acts_as_list mixin in Rails handles this basically the way you outlined in #1. It looks for an INTEGER column called position (of which you can override to name of course) and using that to do an ORDER BY. When you want to re-order things you update the positions. It has served me just fine every time I've used it.

As a side note, you can remove the need to always do re-positioning on INSERTS/DELETES by using sparse numbering -- kind of like basic back in the day... you can number your positions 10, 20, 30, etc. and if you need to insert something in between 10 and 20 you just insert it with a position of 15. Likewise when deleting you can just delete the row and leave the gap. You only need to do re-numbering when you actually change the order or if you try to do an insert and there is no appropriate gap to insert into.

Of course depending on your particular situation (e.g. whether you have the other rows already loaded into memory or not) it may or may not make sense to use the gap approach.

Share Improve this answer

Follow

answered Aug 21, 2008 at 23:11

John

15.3k ● 12 ● 59 ● 57

1 +1 for mentioning sparse numbering. I've used the ranked-model gem for this in the past. – Jared Beck Sep 3, 2014 at 22:05

3

If the objects aren't heavily keyed by other tables, and the lists are short, deleting everything in the domain and just re-inserting the correct list is the easiest. But that's not practical if the lists are large and you have lots of constraints to slow down the delete. I think your first method is really the cleanest. If you run it in a transaction you can be sure nothing odd happens while you're in the middle of the update to screw up the order.

Share Improve this answer

Follow

answered Aug 22, 2008 at 1:39

Eric Z Beard

38.4k ● 27 ● 101 ● 147

3

Just a thought considering **option #1 vs #3**: doesn't the spaced array option (#3) only postpone the problem of the normal array (#1)? Whatever algorithm you choose, either it's broken, and you'll run into problems with #3 later, or it works, and then #1 should work just as well.

Share Improve this answer

Follow

answered Aug 25, 2008 at 17:24

onnodb

5,261 ● 1 ● 33 ● 41

3

Use a floating point number to represent the position of each item:

Item 1 -> 0.0

Item 2 -> 1.0

Item 3 -> 2.0

Item 4 -> 3.0

You can place any item between any other two items by simple bisection:

Item 1 -> 0.0

Item 4 -> 0.5

Item 2 -> 1.0

Item 3 -> 2.0

(Moved item 4 between items 1 and 2).

The bisection process can continue almost indefinitely due to the way floating point numbers are encoded in a computer system.

Item 4 -> 0.5

Item 1 -> 0.75

Item 2 -> 1.0

Item 3 -> 2.0

(Move item 1 to the position just after Item 4)

answered Sep 18, 2008 at 0:22

Seun Osewa

5,023 3 31 32

6 This /will not/ continue indefinitely. For floating point numbers (doubles) values will converge after 53 rounds, in the pathological case. Even if your DBMS uses arbitrary precision decimals you will have a lot of data structure bloat. – cdleary Oct 6, 2008 at 8:16

OK, so add a contingency that restructures with O(n) complexity when the spacing gets below a threshold. Now you have an O(n) operation occurring on about every 1/10000 operations. The bisecting algorithm is best if you are smart about using it. – Chris Conlan Dec 15, 2018 at 14:42

2

I did this in my last project, but it was for a table that only occasionally needed to be specifically ordered, and wasn't accessed too often. I think the spaced array would be the best option, because it reordering would be cheapest in the average case, just involving a change to one value and a query on two).

Also, I would imagine ORDER BY would be pretty heavily optimized by database vendors, so leveraging that function would be advantageous for performance as opposed to the linked list implementation.

Share Improve this answer Follow

answered Aug 22, 2008 at 1:58

pbh101

10.4k ● 9 ● 33 ● 31

I'd do a consecutive number, with a trigger on the table that "makes room" for a priority if it already exists.

1

Share Improve this answer

Follow

answered Aug 21, 2008 at 23:12

Stu

15.8k ● 4 ● 45 ● 74

4 This requires an O(n) restructuring on every insertion!
– cdleary Oct 6, 2008 at 8:18

1

I had this problem as well. I was under heavy time pressure (aren't we all) and I went with option #1, and only updated rows that changed.

If you swap item 1 with item 10, just do two updates to update the order numbers of item 1 and item 10. I know it is algorithmically simple, and it is O(n) worst case, but that worst case is when you have a total permutation of the list. How often is that going to happen? That's for you to answer.

Share Improve this answer

Follow

answered Sep 18, 2008 at 0:34

moffdub

5,314 ● 2 ● 37 ● 32

1

Since I've mostly run into this with Django, I've found this solution to be the most workable. It seems that there isn't any "right way" to do this in a relational database.

Share Improve this answer

Follow

answered Mar 29, 2009 at 14:47

tghw

25.3k ● 13 ● 71 ● 97

So much is masked by jQuery UI that I don't know which scheme this follows. Based on the fact that the model is using an `IntegerField` for ordering, it is likely using doing O(n) updates, and follow OP's option #1. – Chris Conlan Dec 15, 2018 at 14:23

0

I had the same issue and have probably spent at least a week concerning myself about the proper data modeling, but I think I've finally got it. Using the array datatype in PostgreSQL, you can store the primary key of each ordered item and update that array accordingly using insertions or deletions when your order changes. Referencing a single row will allow you to map all your objects based on the ordering in the array column.

It's still a bit choppy of a solution but it will likely work better than option #1, since option 1 requires updating the order number of all the other rows when ordering changes.

Share Improve this answer

Follow

answered Jan 28, 2016 at 10:32

Austin T
53 ● 1 ● 7

0

Scheme #1 and Scheme #3 have the same complexity in every operation except `INSERT` writes. Scheme #1 has O(n) writes on `INSERT` and Scheme #3 has O(1) writes on `INSERT`.

For every other database operation, the complexity is the same.

Scheme #2 should not even be considered because its `DELETE` requires O(n) reads and writes. Scheme #1 and Scheme #3 have O(1) `DELETE` for both read and write.

New method

If your elements have a distinct parent element (i.e. they share a foreign key row), then you can try the following ...

Django offers a database-agnostic solution to storing lists of integers within `CharField()`. One drawback is that the max length of the stored string can't be greater than `max_length`, which is DB-dependent.

In terms of complexity, this would give Scheme #1 O(1) writes for `INSERT`, because the ordering information would be stored as a single field in the parent element's row.

Another drawback is that a `JOIN` to the parent row is now required to update ordering.

https://docs.djangoproject.com/en/dev/ref/validators/#django.core.validators.validate_comma_separated_integer_list

edited Dec 15, 2018 at 15:04

answered Dec 15, 2018 at 14:48

Chris Conlan

2,944 ● 1 ● 22 ● 23