Optimized SQL for tree structures

Asked 16 years ago Modified 8 years, 8 months ago Viewed 33k times

37

How would you get tree-structured data from a database with the best performance? For example, say you have a folder-hierarchy in a database. Where the folder-database-row has **ID**, **Name** and **ParentID** columns.

Would you use a special algorithm to get all the data at once, minimizing the amount of database-calls and process it in code?

Or would you use do many calls to the database and sort of get the structure done from the database directly?

Maybe there are different answers based on x amount of database-rows, hierarchy-depth or whatever?

Edit: I use Microsoft SQL Server, but answers out of other perspectives are interesting too.

sql sql-server tree-structure

Share

Improve this question

Follow

edited Nov 26, 2008 at 1:46

Cade Roux

89.6k ● 40 ● 184 ● 266

asked Nov 25, 2008 at 13:31

Seb Nilsson
26.4k ● 30 ● 106 ● 131

What RDMS are you using? SQL Server? MySQL? Oracle?
– davidkemp Nov 25, 2008 at 13:33

11 Answers

Sorted by: Highest score (default)

18

It really depends on how you are going to access the tree.

One clever technique is to give every node a string id, where the parent's id is a predictable substring of the child. For example, the parent could be '01', and the children would be '0100', '0101', '0102', etc. This way you can select an entire subtree from the database at once with:

```
SELECT * FROM treedata WHERE id LIKE '0101%';
```

Because the criterion is an initial substring, an index on the ID column would speed the query.

Share Improve this answer
Follow

answered Nov 25, 2008 at 13:44

Ned Batchelder
375k ● 77 ● 578 ● 673

1 You just have to be sure that the number of digits per level (2 in this case) * the number of levels is allowed for in that CHAR column. This imposes some artificial (but manageable) limitations. – S.Lott Nov 26, 2008 at 2:59

@Ned Batchelder I will try this method for my table structure. However isn't it hard to move a subtree to another? What happens if a new node is inserted in the middle of the hierarchy? Should I keep the parentId columns too? Or this id always enough to handle the structure? Thank you.
– Ismail Yavuz Aug 10, 2015 at 7:49

15

Out of all the ways to store a tree in a RDMS the most common are adjacency lists and nested sets. Nested sets are optimized for reads and can retrieve an entire tree in a single query. Adjacency lists are optimized for writes and can added to with in a simple query.

With adjacency lists each node a has column that refers to the parent node or the child node (other links are possible). Using that you can build the hierarchy based on parent child relationships. Unfortunately unless you restrict your tree's depth you cannot pull the whole thing in one query and reading it is usually slower than updating it.

With the nested set model the inverse is true, reading is fast and easy but updates get complex because you must maintain the numbering system. The nested set model encodes both parentage and sort order by enumerating

all of the nodes using a preorder based numbering system.

I've used the nested set model and while it is complex for read optimizing a large hierarchy it is worth it. Once you do a few exercises in drawing out the tree and numbering the nodes you should get the hang of it.

My research on this method started at this article: Managing Hierarchical Data in MySQL.

Share Improve this answer

Follow

edited Aug 21, 2011 at 22:15

Paŭlo Ebermann

74.7k ● 20 ● 148 ● 215

answered Dec 17, 2008 at 20:37

Bernard Igiri

1,290 ● 2 ● 18 ● 34

that article is solid. Nice addition! – Quibblesome Sep 10, 2018 at 12:57

8

In the product I work on we have some tree structures stored in SQL Server and use the technique mentioned above to store a node's hierarchy in the record. i.e.

```
tblTreeNode
TreeID = 1
TreeNodeID = 100
ParentTreeNodeID = 99
Hierarchy = ".33.59.99.100."
[...] (actual data payload for node)
```

Maintaining the the hierarchy is the tricky bit of course and makes use of triggers. But generating it on an insert/delete/move is never recursive, because the parent or child's hierarchy has all the information you need.

you can get all of node's descendants thusly:

```
SELECT * FROM tblNode WHERE Hierarchy LIKE '%.100.%'
```

Here's the insert trigger:

```
--Setup the top level if there is any
UPDATE T
SET T.TreeNodeHierarchy = '.' + CONVERT(nvarchar(10),
FROM tblTreeNode AS T
    INNER JOIN inserted i ON T.TreeNodeID = i.TreeNode
WHERE (i.ParentTreeNodeID IS NULL) AND (i.TreeNodeHier

WHILE EXISTS (SELECT * FROM tblTreeNode WHERE TreeNode
    BEGIN
        --Update those items that we have enough infor
has text in Hierarchy
        UPDATE CHILD
        SET CHILD.TreeNodeHierarchy = PARENT.TreeNodeH
CONVERT(nvarchar(10),CHILD.TreeNodeID) + '.'
        FROM tblTreeNode AS CHILD
            INNER JOIN tblTreeNode AS PARENT ON CHILD.
PARENT.TreeNodeID
        WHERE (CHILD.TreeNodeHierarchy IS NULL) AND (P
IS NOT NULL)
    END
```

and here's the update trigger:

```
--Only want to do something if Parent IDs were changed
IF UPDATE(ParentTreeNodeID)
    BEGIN
```

```
        --Update the changed items to reflect their ne
        UPDATE CHILD
        SET CHILD.TreeNodeHierarchy = CASE WHEN PARENT
'.' + CONVERT(nvarchar,CHILD.TreeNodeID) + '.' ELSE PA
CONVERT(nvarchar, CHILD.TreeNodeID) + '.' END
        FROM tblTreeNode AS CHILD
            INNER JOIN inserted AS I ON CHILD.TreeNode
            LEFT JOIN tblTreeNode AS PARENT ON CHILD.P
PARENT.TreeNodeID

        --Now update any sub items of the changed rows
        IF EXISTS (
                SELECT *
                FROM tblTreeNode
                    INNER JOIN deleted ON tblTreeNode.
deleted.TreeNodeID
            )
            UPDATE CHILD
            SET CHILD.TreeNodeHierarchy = NEWPARENT.Tr
RIGHT(CHILD.TreeNodeHierarchy, LEN(CHILD.TreeNodeHiera
LEN(OLDPARENT.TreeNodeHierarchy))
            FROM tblTreeNode AS CHILD
                INNER JOIN deleted AS OLDPARENT ON CHI
(OLDPARENT.TreeNodeHierarchy + '%')
                INNER JOIN tblTreeNode AS NEWPARENT ON
NEWPARENT.TreeNodeID

    END
```

one more bit, a check constraint to prevent a circular reference in tree nodes:

```
ALTER TABLE [dbo].[tblTreeNode]  WITH NOCHECK ADD  CON
[CK_tblTreeNode_TreeNodeHierarchy] CHECK
((charindex(('.' + convert(nvarchar(10),[TreeNodeID])
[TreeNodeHierarchy],(charindex(('.' + convert(nvarchar
'.'),[TreeNodeHierarchy]) + 1)) = 0))
```

I would also recommend triggers to prevent more than one root node (null parent) per tree, and to keep related

nodes from belonging to different TreeIDs (but those are a little more trivial than the above.)

You'll want to check for your particular case to see if this solution performs acceptably. Hope this helps!

Share Improve this answer Follow

answered Nov 25, 2008 at 14:31

James Orr

4

Celko wrote about this (2000):

http://www.dbmsmag.com/9603d06.html

http://www.intelligententerprise.com/001020/celko1_1.jhtml;jsessionid=3DFR02341QLDEQSNDLRSKHSCJUNN2JVN?_requestid=32818

---

and other people asked:

Joining other tables in oracle tree queries

How to calculate the sum of values in a tree using SQL

How to store directory / hierarchy / tree structure in the database?

Performance of recursive stored procedures in MYSQL to get hierarchical data

What is the most efficient/elegant way to parse a flat table into a tree?

finally, you could look at the rails "acts_as_tree" (read-heavy) and "acts_as_nested_set" (write-heavy) plugins. I don't ahve a good link comparing them.

Share Improve this answer

Follow

edited May 23, 2017 at 12:02

Community Bot

1 ● 1

answered Nov 25, 2008 at 15:15

Gene T

5,176 ● 1 ● 26 ● 25

2

There are several common kinds of queries against a hierarchy. Most other kinds of queries are variations on these.

1. From a parent, find all children.

   a. To a specific depth. For example, given my immediate parent, all children to a depth of 1 will be my siblings.

   b. To the bottom of the tree.

2. From a child, find all parents.

   a. To a specific depth. For example, my immediate parent is parents to a depth of 1.

   b. To an unlimited depth.

The (a) cases (a specific depth) are easier in SQL. The special case (depth=1) is trivial in SQL. The non-zero

depth is harder. A finite, but non-zero depth, can be done via a finite number of joins. The (b) cases, with indefinite depth (to the top, to the bottom), are really hard.

If you tree is **HUGE** (millions of nodes) then you're in a world of hurt no matter what you try to do.

If your tree is under a million nodes, just fetch it all into memory and work on it there. Life is much simpler in an OO world. Simply fetch the rows and build the tree as the rows are returned.

If you have a **Huge** tree, you have two choices.

- Recursive cursors to handle the unlimited fetching. This means the maintenance of the structure is O(1) -- just update a few nodes and you're done. However fetching is O(n*log(n)) because you have to open a cursor for each node with children.
- Clever "heap numbering" algorithms can encode the parentage of each node. Once each node is properly numbered, a trivial SQL SELECT can be used for all four types of queries. Changes to the tree structure, however, require renumbering the nodes, making the cost of a change fairly high compared to the cost of retrieval.

answered Nov 25, 2008 at 13:46

S.Lott

391k ● 82 ● 517 ● 788

SQL CTE eliminates the need for recursive cursors and has some optimization for join folding - but still it is an expensive call to enumerate large hierarchies. – stephbu Nov 25, 2008 at 15:49

If you have millions of nodes, you could make tree of trees. Each tree contained in the DB as BLOB. You read top tree (with millions of leafs) where each leaf will have ID to it's sub-tree with millions of leafs. That way you'll have billions of leafs and fast reading if queries have locality no more than a few sub-trees. – The_Ghost Jul 21, 2010 at 21:41

1

If you have many trees in the database, and you will only ever get the whole tree out, I would store a tree ID (or root node ID) and a parent node ID for each node in the database, get all the nodes for a particular tree ID, and process in memory.

However if you will be getting subtrees out, you can only get a subtree of a particular parent node ID, so you either need to store all parent nodes of each node to use the above method, or perform multiple SQL queries as you descend into the tree (hope there are no cycles in your tree!), although you can reuse the same Prepared Statement (assuming that nodes are of the same type and are all stored in a single table) to prevent re-compiling the SQL, so it might not be slower, indeed with database optimisations applied to the query it could be preferable. Might want to run some tests to find out.

If you are only storing one tree, your question becomes one of querying subtrees only, and the second answer applied.

Share Improve this answer

Follow

answered Nov 25, 2008 at 13:37

JeeBee

17.5k ● 5 ● 52 ● 60

Google for "Materialized Path" or "Genetic Trees"...

1

Share Improve this answer

Follow

answered Nov 25, 2008 at 13:40

Thomas Hansen

5,513 ● 1 ● 25 ● 28

In Oracle there is SELECT ... CONNECT BY statement to retrieve trees.

1

Share Improve this answer

Follow

answered Nov 25, 2008 at 13:42

Dmitry Khalatov

4,349 ● 4 ● 35 ● 39

I am a fan of the simple method of storing an ID associated with its parentID:

1

```
ID      ParentID
1       null
2       null
3       1
4       2
...     ...
```

It is easy to maintain, and very scalable.

Share Improve this answer

Follow

edited Nov 25, 2008 at 13:47

answered Nov 25, 2008 at 13:37

Galwegian

42.2k ● 16 ● 113 ● 158

It wasn't there when I answered originally. – Galwegian Nov 25, 2008 at 16:26

4 Actually, it's not scalable. If you frequently work with a whole tree of depth n, you will need n queries to fetch all the data. For tall busy trees (e.g. a forum), this can be a performance killer. – staticsan Nov 25, 2008 at 23:30

1 yeah, seriously dude, delete the bit about it being scaleable. Your reads get slower and slower the bigger the tree is. Its pretty much the worst scaling option for what is often the most common use-case (read). – Quibblesome Sep 10, 2018 at 12:56

0

This article is interesting as it shows some retrieval methods as well as a way to store the lineage as a

derived column. The lineage provides a shortcut method to retrieve the hierarchy without too many joins.

Share Improve this answer

Follow

answered Nov 25, 2008 at 13:50

Turnkey

9,406 ● 3 ● 29 ● 36

0

Not going to work for all situations, but for example given a comment structure:

```
ID | ParentCommentID
```

You could also store `TopCommentID` which represents the top most comment:

```
ID | ParentCommentID | TopCommentID
```

Where the `TopCommentID` and `ParentCommentID` are `null` or `0` when it's the topmost comment. For child comments, `ParentCommentID` points to the comment above it, and `TopCommentID` points to the topmost parent.

Share Improve this answer

Follow

answered Apr 13, 2016 at 10:37

Tom Gullen

61.7k ● 86 ● 290 ● 467