Nested Intervals Tree Encoding in SQL

Vadim Tropashko
Oracle Corp.

Abstract

Nested Intervals generalize Nested Sets. They are immune to hierarchy reorganization problem. They allow answering ancestor path hierarchical queries algorithmically - without accessing the stored hierarchy relation.

1 Introduction

There are several SQL techniques to query graph structures, in general, and trees, in particular [2]. They can be classified into 2 major categories: *Hierarchical/recursive SQL extensions* and *Tree encodings*. This article focuses upon tree encodings.

Tree encodings methods themselves can be split into 2 groups: *Materialized Path* and *Nested Sets.*

Materialized Path is nearly ubiquitous encoding, where each tree node is labeled with the path from the node to the root. UNIX global filenames is well known showcase for this idea. Materialized Path could be either represented as character string of unique sibling identifiers (concatenated with some separator), or enveloped into user defined type [5].

Querying trees with Materialized Path technique doesn't appear especially elegant. It implies either string matching like this

```
select e1.ename from emp e1, emp e2
where e2.path like e1.path || '%'
and e2.name = 'FORD'
```

or leveraging complex data types that are realm of Object-Relational Databases. The alternative tree encoding - Nested Sets [2] labels each node with just a pair of integers. Ancestor-descendant relationship is reflected by subset relation between intervals of integers, which provides very intuitive base for hierarchical queries.

Although Nested Sets are certainly appealing to many database developers, they have 2 fundamental disadvantages:

1. The encoding is volatile. In a word, roughly half of the tree nodes should be relabeled whenever a new node were inserted.

2. Querying ranges is asymmetric from performance perspective. It is easy to answer if a point falls inside some interval, but it is hard to index a set of intervals that contain a given point. For nested sets this translates into a difficulty answering queries about node's ancestors.

[6] introduced *Nested Intervals* that generalize Nested Sets. Since Nested Sets encoding with integers admits only finite gaps for new node insertions, it is natural to use dense domain such as rational numbers. One particular encoding schema with *Dyadic rational numbers* was developed in the rest of the article, and was a subject of further improvements in the follow up articles. Dyadic rational encoding has many nice theoretical properties, and essentially is a numeric reflection of Materialized Path. It has, however, one significant flaw from practical perspective. Dyadic fractions utilize domain of integer numbers rather uneconomically, so that numeric overflow prevents tree scaling to any significant size.

In general, Nested Intervals allow a certain freedom choosing particular encoding schema. [7] developed alternative encoding with *Farey fractions*. The development continued in [8].

This article expands the perspective. It demonstrates why both methods are natural choices, and describes the mapping between those tree encodings. It goes on exploring different ways of establishing interval structure. The major result is introducing Path Matrices and exposing their properties.

Similar idea of leveraging Stern-Brocot tree is briefly mentioned in [1]. The article, however, contains just a hint, while pursuing some other venue. [3] is, perhaps, the earliest reference in the database literature referring to Continued Fractions encoding. The manuscript is unavailable to the author to draw detailed comparison with his method.

2 Nested Intervals Queries

Nested Intervals encode each tree node with a pair of numbers *head* and *tail*. Interval for a child node is always contained within parent interval. With this labeling transitive closure could be queried like this

```
select e1.ename, e2.ename
from emp e1, emp e2
where e2.head >= e1.head
and e2.head < e1.tail
```

Next, the subtree of all the descendants of a node could be found by just restricting the above view with a single table predicate

```
select e2.ename from emp e1, emp e2
where e2.head >= e1.head
and e2.head < e1.tail
and e1.ename = 'SCOTT'
```

The ancestor path can be queried symmetrically

```
select e1.ename from emp e1, emp e2
where e2.head >= e1.head
and e2.head < e1.tail
and e2.ename = 'SCOTT'
```

There is a subtle problem with the last query, however. Finding all the intervals that cover a given point is difficult. Although there are specialized indexing schemes like R-Tree, none of them is as universally accepted as B-Tree.

Compare this to the descendants query, assuming that the subtree of *SCOTT*'s subordinates is small. The execution path in this case is very efficient: first, *e1* record is fetched by the unique index, and then all the *e2* records are fetched by index range scan.

The details of Nested Intervals encoding are developed in the next sections. The encoding is *algorithmic*. Given a child node label, the parent encoding can be calculated, not queried. Therefore, the whole path to the root node can be calculated. Hence, if we know tree node encodings for all nodes on the path then, the nodes themselves can be efficiently queried in the database.

3 Interval Halving

The easiest way to nest intervals is splitting parent interval into two halves. If we start with the points 0 and 1 and continue on halving the intervals iteratively then, what kind of numbers on the interval boundaries would be produced? Clearly, the ones whose denominator is power of 2, or simply dyadic fractions [4].

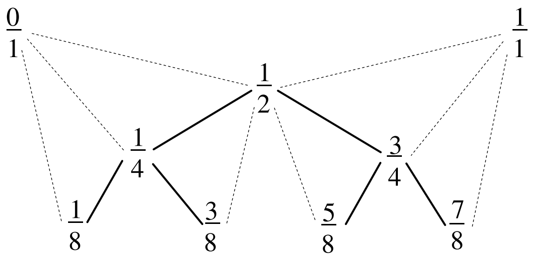


Fig.1. Dyadic Fractions at Conway tree.

When splitting the interval `[head,tail]` into two, the point on the boundary is the *average* `(head+tail)/2`. Alternatively, we could have chosen the *mediant*:

$$\frac{head\_numer + tail\_numer}{head\_denom + tail\_denom}$$

If we start with the points 0 and 1 and continue on, then the *Stern-Brocot tree* of *Farey fractions* would be produced.

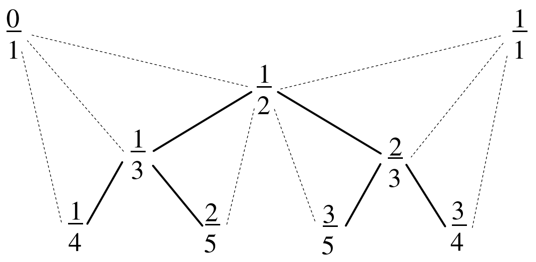


Fig.2. Farey fractions at Stern-Brocot tree.

The bijection between Dyadic and Stern-Brocot tree is defined by the following Minkowski Question Mark function `?: [0,1]→[0,1]` [9].

If x has binary expansion .00...011... 100...011...1..., where there are a zeros in the first block, then b ones in the second, then c zeros, and so on, then ?(x) is the (simple) continued fraction

$$\frac{1}{a+1+\frac{1}{b+\frac{1}{c+\frac{1}{so\_on}}}}$$

Thus, for example, if x = 1/4, its two binary expansions .0100000... and .00111111... yield the two expressions

$$\frac{1}{1+1+\frac{1}{1+\frac{1}{\infty}}}=\frac{1}{2+1+\frac{1}{\infty}}$$

Therefore, ?(1/4)=1/3. Note that the node ¼ is positioned in the Dyadic tree on Fig.1 in the same place where the node 1/3 is in Stern-Brocot tree on Fig.2.

4 Nested Interval Structure

In previous section we developed two alternative but isomorphic systems how to generate interval boundary points. What intervals should we consider? Clearly, including all possible intervals into our system would be too much. In Farey case (Fig.2), for example, the interval [1/3,1/2] would have at least two parents: [1/3,2/3] and [0/1,1/1].

What if we limit the scope to only those intervals that correspond to the edges at Fig.1? If we consider solid and dashed lines, then there still would be too many intervals. Consider the interval [1/3,2/5] (Fig.2). How many siblings does it have? Well, no more than one: [2/5,1/2]. Indeed, no other interval has [1/3,1/2] as a parent.

If we consider solid lines only (with two additional convenience intervals at the top), then we'll get a system of Nested Intervals shown at Fig.3.

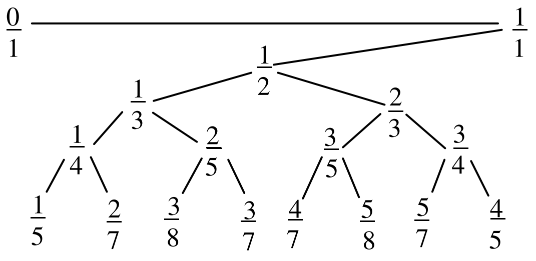


Fig.3. Simple Farey interval structure.

Dyadic Nested Interval structure is isomorphic to Fig.3 - we omit the picture in order to save space. The reason why we preferred Farey over Dyadic case would become evident in the last section. It would also become apparent why it's called "simple".

The other possible way to introduce Nested Interval structure is shown at Fig.4, this time with dyadic encoding.

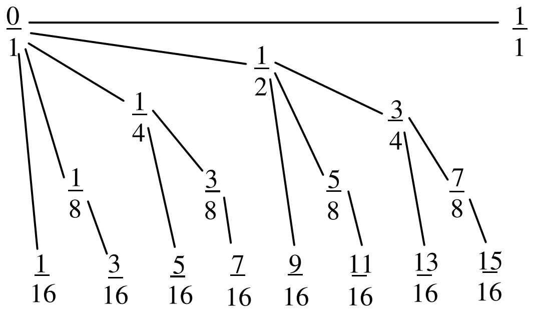


Fig.4. Monotonic dyadic interval structure.

Algorithms for navigating Dyadic Nested Intervals are almost obvious:

1. Younger sibling [head,tail] encoding is:

$$\left[\frac{2\ head\_numer\ +1}{2\ head\_denom},\frac{2\ tail\_numer\ +1}{2\ tail\_denom}\right]$$

2. Older sibling [head,tail]:

$$\left[\frac{head\_numer\ -1}{head\_denom},\frac{tail\_numer\ -1}{tail\_denom}\right]$$

(Be careful, however, when applying this rule to the first child)

3. Parent of the first child:

$$\left[\frac{head\_numer}{head\_denom}, \frac{tail\_numer + 1}{tail\_denom}\right]$$

Farey Intervals are little bit more sophisticated.

5 The Path Matrix

Let's study simple Farey interval structure (Fig.3) in more detail. Consider the interval `[5/7,3/4]`. It is the first child of `[2/3,3/4]`. Then, `[2/3,3/4]` is the second child of `[1/2,1/1]`. Finally, `[1/2,1/1]` is the first child of `[0/1,1/1]`. Therefore, the materialized path encoding of `[5/7,3/4]` is `1.2.1`. However, we have 4 intervals, i.e. 4 nodes, while materialized path has the length 3. How can that be?

Could interval `[0/1,1/1]` be considered as somebody's else child too? Well, yes, and no. The obvious parent candidate is `[0/1,1/0]`. Then, the interval `[1/1,2/1]` is the second child of `[0/1,1/0]`! The amended materialized path encoding for `[5/7,3/4]` is `1.1.2.1` and, by the way, we also are able to find Farey interval encodings for materialized paths beginning with natural numbers other than `1`.

Here is formal procedure converting Farey encoding into materialized path. Start with Farey interval written as 2x2 matrix:

$$\begin{bmatrix} 3 & 5 \\ 4 & 7 \end{bmatrix}$$

Note that we switched the fractions. The purpose is to keep the highest integer in the lower right corner. Next, write

```
5 = 3 * ⌊5/3⌋ + 5 mod 3 = 3*1 + 2
7 = 4 * ⌊7/4⌋ + 7 mod 4 = 4*1 + 3
```

The integer division result (which is `1` - the same in both cases) is the first element of the materialized path encoding. We add the column with the remainders to the left

```
2  3  5
3  4  7
```

Matrix on the left

$$\begin{bmatrix} 2 & 3 \\ 3 & 4 \end{bmatrix}$$

corresponds to the interval `[2/3,3/4]` - the parent of our original interval.

Continuing to the left we get

```
0  1  2  3  5
1  1  3  4  7
```

together with the sequence of integer division results `1,1,2,` and `1`. We stop as soon as zero in the top left corner appears.

Next, we can expand this *number wall* up. The rule is the same but applied to rows instead of columns. In our example, we write

```
7 = 5*⌊7/5⌋ + 7 mod 5 = 5*1 + 2
4 = 3*⌊4/3⌋ + 4 mod 3 = 3*1 + 1
3 = 2*⌊3/2⌋ + 3 mod 2 = 2*1 + 1
1 = 1*⌊1/1⌋ + 1 mod 1 = 1*1 + 0
```

Adding the row `0,1,1,2` at the top results in

```
   0  1  1  2
0  1  2  3  5
1  1  3  4  7
```

Continuing this process, we get the following number wall

```
            0  1
         0  1  1
      0  1  1  2
   0  1  2  3  5
   1  1  3  4  7
```

Note, that all 2x2 sub-matrices at this wall have determinant `1` or `-1`. This property enforces the unique way of completing the wall to square matrix

```
1  0  1  0  1
0  1  0  1  1
1  0  1  1  2
0  1  2  3  5
1  1  3  4  7
```

We refer to the number wall that we just have built as the *Path Matrix*. It enjoys many nice properties.

1. The numbers at the main antidiagonal are all `1`s.
2. The sequence of numbers below the main antidiagonal is materialized path. The path is oriented from right to left.
3. Adjacent 2x2 sub-matrices can be multiplied as shown on Fig.5

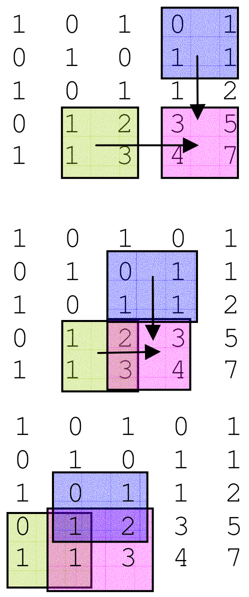


Fig.5. Multiplying adjacent matrices.

The matrix identity in the middle case on Fig.5, for example, is

$$\begin{bmatrix}1 & 2\\ 1 & 3\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}=\begin{bmatrix}2 & 3\\ 3 & 4\end{bmatrix}$$

Multiplying overlapping matrices, similar to the last case

$$\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 2\end{bmatrix}=\begin{bmatrix}1 & 2\\ 1 & 3\end{bmatrix}$$

works for matrices on the main antidiagonal only.

Iterative application of matrix multiplication property gives rise to the following matrix decomposition

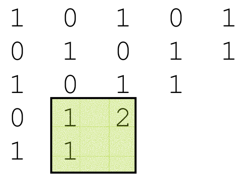
$$\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 2\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}\cdot\begin{bmatrix}0 & 1\\ 1 & 1\end{bmatrix}=\begin{bmatrix}3 & 5\\ 4 & 7\end{bmatrix}$$

Each of the matrices on the left side corresponds to an elementary fragment of the materialized path `1.1.2.1`. Since these elementary matrices all have determinant `-1`, their multiple would always have determinant `-1` or `1` - the property that we noticed earlier.

The determinant property allows filling in the numbers in the Path Matrix in the other direction. Suppose we know materialized path and want to calculate corresponding Farey interval. One way is multiplying elementary matrices, by leveraging the above matrix decomposition identity. Alternatively, we can start with partially filled in Path Matrix. By properties 1 and 2 we have

```
1  0  1  0  1
0  1  0  1  1
1  0  1  1
0  1  2
1  1
```

We fill in empty positions as follows. Select 2x2 matrix that has 3 elements defined and the 4th element empty



Fill in the empty position to satisfy the determinant property. The sign of the determinant is alternating. It is negative if the matrix is positioned at even distance from main antidiagonal, and positive otherwise. In our case, the matrix is just one step away from the position at the main antidiagonal. Therefore, the value `x` at the empty position has to satisfy the equation

```
1*x - 1*2 = 1
```

hence, `x=3`, as expected.

After all the empty positions are filled, we can grab 2x2 Farey interval matrix at the lower right corner.

The final important property of the Path Matrix is that matrix transposition corresponds to materialized path inversion.

6 Continued Fractions

[8] suggests one more perspective into Farey interval encoding. Materialized path `1.1.2.1` can be naturally written as the simple continued fraction

$$\cfrac{1}{1+\cfrac{1}{1+\cfrac{1}{2+\cfrac{1}{1+x}}}}$$

which can be simplified into Moebius function

$$\frac{4+3\,x}{7+5\,x}$$

Here the familiar 2x2 matrix from our example can be recognized.

Simple continued fractions have somewhat irritating feature that increasing any denominator, either increases the value of the number, or decreases it, depending on the parity of the position. *Reversed* (or *additive*) continued fractions

$$\cfrac{1}{1+1-\cfrac{1}{1+1-\cfrac{1}{2+1-\cfrac{1}{1+1-x}}}}$$

are monotonic. The path matrix theory for the additive continued fractions mimics the classic case described in section 6. One of the distinguished feature of additive continued fractions is that all the matrices have negative entries in the second column, and determinant equal to 1. There is no alternation anymore: interval encoding of the younger child always precedes the older one. (In the simple continued fractions case this was true for the odd levels, and reversely true for the even ones). Finally, additive continued fractions map into monotonic Farey interval structure.

References

[1] D. Aioanei, A. Malinaru. General trees persisted in relational databases. http://www.codeproject.com/cs/database/persisting_trees.asp?print=true

[2] J. Celko. Joe Celko's Trees and Hierarchies in SQL for Smarties. Morgan Kaufmann.

[3] P. Ciaccia, D. Maio, and P. Tiberio. A method for hierarchy processing in relational systems. Information Systems, 14(2):93-105, 1989.

[4] J. Conway. On Numbers and Games. New York: Academic Press, Inc.

[5] J. Roy. 2003. Using the Node Data Type to Solve Problems with Hierarchies in DB2 Universal Database http://www106.ibm.com/developerworks/db2/library/techarticle/0302roy/0302roy.html

[6] V. Tropashko. Trees in SQL: Nested Sets and Materialized Path. http://www.dbazine.com/tropashko4.shtml

[7] V. Tropashko. Nested Intervals with Farey Fractions. http://arxiv.org/html/cs.DB/0401014

[8] V. Tropashko. Nested Intervals Tree Encoding with Continued Fractions. http://arxiv.org/pdf/cs.DB/0402051

[9] L. Vepstas. The Minkowski Question Mark and the Modular Group SL(2,Z). http://www.linas.org/math/chap-minkowski/chap-minkowski.html