Relational Data Modeling With Recursive Constructs: Part 1.

Dr. Tom Johnston
MindfulData.com

By definition, any entity which has a relationship to itself is a recursive entity; and so any table implementing such an entity is a recursive table. In this chapter, specifically, I will describe the following kinds of recursive table constructs ("RT" constructs, for short):[1]

1. *Sequentially recursive tables*, used to model a 1:1 relationship among rows in a table, resulting in a linearly-structured set of rows;

2. *Hierarchically recursive tables,* used to model a 1:M relationship among rows in a table, resulting in a tree-structured set of rows; and

3. *Matrix-like recursive tables*, used to model a M:M relationship among rows in a table, resulting in a network-structured set of rows.

Sequentially Recursive Tables: Modeling Linearly Structured Sets.

A mathematical set, without additional constraints specified on it, is just a well-defined collection of things, in which no one thing is represented twice. But sometimes--often, in fact--sets of things *do* have additional constraints imposed on them--constraints expressed as ordering relationships among the codes in the set. One of the simplest of these additional constraints imposes a *linear sequence* on a set. A linear sequence is an ordering which puts every single member of the set "in its place", so to speak. In a linear sequence, every member but the first has one (and only one) predecessor, and every member but the last has one (and only one) successor. The first member has no predecessor, and the last member has no successor.

From this point on, we are going to be concerned, not with the abstract mathematics of sets, but with the interpreted mathematics of entities, and their implementation as tables. Therefore, at this point, let us transition from the language of set theory to the language of databases, and begin speaking, where it is more appropriate, of "tables" instead of "sets", and of "rows" instead of "members".

These predecessor and successor relationships are one-to-one relationships. Each row in a linear sequence has one predecessor and one successor--except for the first and last which have, respectively, no predecessor and no successor.

---

[1] In this chapter, I will use the terminology of "tables" and "rows" as synonymous with the more awkward terminology of "entities" and "entity occurrences", or "entity instances".

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

So let's suppose that the set of rows shown in Figure 4.1 is not an unordered set, but rather a linearly ordered set--a set in which there is one particular sequence for the rows. This sequence may stand for any relationship we like. For example, it might indicate a sequence based on salary ranges if the rows represented job categories. Or it might indicate a sequence based on population numbers in a geographic area if the rows represented zoological species, and so on. In the case of the location types used in this example, there doesn't seem to be any natural linear ordering. So let's assume, for these location types, that the sequence represents the sequence in which the principal users concerned with locations have requested to see location types listed on their screens and reports--for whatever reasons they may have.

There are two mathematically basic approaches to modeling sequences. One is based on set-theory, and the other on arithmetic. We will consider both of them.

Sequentially Recursive Tables: the Predecessor/Successor Approach.

One approach to modeling a linear sequence is to provide separate one-to-one relationships for predecessors and for successors. On this approach, since all but the first and last members of the sequence have a predecessor and a successor, we add two columns to the basic RT construct. One column points to the predecessor for each row, and the other column points to the successor for each row. This construct is illustrated in Figure 4.1.

LOCATION TYPES

loc-type-eid	*loc-type-pred-eid*	*loc-type-succ-eid*	*loc-type-code*	*loc-type-desc*
6021	{null}	2718	OFFBLDG	Office Building
2718	6021	3234	AFFBLDG	Affiliate Building
3234	2718	4440	HQBLDG	Headquarters
4440	3234	2356	BRBLDG	Branch Office Building
2356	4440	0582	EQPLOC	Equipment Location
0582	2356	5678	WRHSLOC	Equipment Warehouse
5678	0582	1448	FLDLOC	Equipment-in-Field Location
1448	5678	3899	AGFLDLOC	Equipment-in-Field Location, Above-Ground
3899	1448	{null}	BGFLDLOC	Equipment-in-Field Location, Below-Ground

Figure 4.1. A Sequential RT Construct Using Predecessors and Successors --Sample Table.

In this table, (a) every row but the first has one and only one predecessor, (b) every row but the last has one and only one successor, (c) the first row has one and only one

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

successor but no predecessor, and (d) the last row has one and only one predecessor, but no successor. The definition of "linear sequence", therefore, has been satisfied.

Moreover, it has been satisfied with a minimal set of syntactical constructs, which contain neither homonyms nor synonyms. One construct--loc-type-pred-eid--models the predecessor concept, and another--loc-type-succ-eid--models the successor concept. loc-type-eid is the identifier of each code, loc-type-code is the code value itself for each location type, and loc-type-desc describes or defines each location type.

This is a paradigm of elegance--each model construct playing one role, and each role played by one model construct.

Figures 4.2 and 4.3, respectively, show the data model diagram for this construct, and the corresponding DDL.

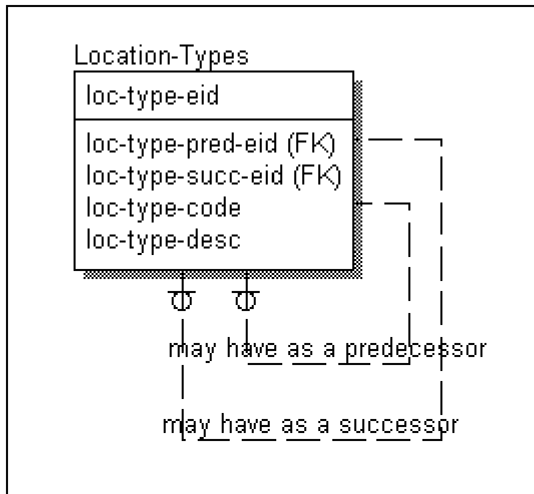

Figure 4.2. A Sequential RT Construct Using Predecessors and Successors --Diagram.

The diagram shows two 1:1 relationships. One represents the predecessor relationship, and the other represents the successor relationship. Observe, from Figures 4.1, 4.2 and 4.3, that these one-to-one relationships are modeled exactly as Chapter 3 described the modeling of one-to-one relationships. The only difference between this, and the examples modeled in Chapter 3, is that the relationships are from one table to itself, not from one table to another. This is what it means, in the context of relational data modeling, to say that a relationship is recursive.

Note also that each relationship is optional in both directions.[2] This permits any row to exist without a predecessor and without a successor. In fact, only one row--the first--will have no predecessor, and only one row--the last--will have no successor. We will have to

[2] This time, the fudging on notation that this data modeling tool does, is not harmless. The way the relationship lines are drawn, we simply don't know whether the relationships are optional in both directions, or only in the one explicitly indicated direction.

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

rely on code, however, to insure that there is only one row with {null} in its predecessor FK, and only one row with {null} in its successor FK.

{{{{{
question for exercises: should the code allow for the possibility of one and the same row having both {null}s? Answer: of course. That will happen when the set contains only one member--a perfectly legitimate situation.
}}}}}

The generic DDL representation of the intension of this linearly ordered code table is shown in Figure 4.3.

```
CREATE TABLE  Location-Types
        (
          loc-type-eid
          loc-type-code
          loc-type-desc

                PRIMARY KEY (loc-type-eid)
                FOREIGN KEY (loc-type-pred-eid) REFERENCES TABLE
                Location-Types
                FOREIGN KEY (loc-type-succ-eid) REFERENCES TABLE
                Location-Types
        )
```

Figure 4.3. A Sequential RT Construct Using Predecessors and Successors --Generic DDL.

Note that the table has two foreign keys, one for the predecessor relationship, and the other for the successor relationship. Note also that each foreign key references the table that contains its primary key instance. Since that table is the same table that the foreign key itself is in, this is the representation, in generic DDL, of a *recursive* relationship.

From the perspective of mathematical elegance, we *can't* improve on this approach. There are no homonymous or synonymous data structures; that is, there are no structures that enforce multiple semantic constraints, and no semantic constraint constraint enforced by multiple structures. Finally, all information required to represent a linear sequence has been included in the structures of the model.

However, it may have occurred to many that there is an intuitively simpler way to represent a sequence of rows. That simpler way is to assign a sequence number to each row. Then if we know, for example, that code EQPLOC has sequence number 5, we immediately know its immediate predecessor and successor. They would be, respectively, the rows with sequence numbers 4 and 6.

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

This is obviously correct. But before we proceed to develop the sequence number approach, we need to understand *why* it is intuitively simpler than recording a successor and a predecessor in the table. For, as we have just shown, the successor and predecessor way of modeling a linear sequence of codes contains nothing superfluous, and has no homonymous or synonymous data structures. From this fact, we might think that it would deductively follow that any other approach would *have* to be less mathematically elegant.

The reason it does not follow is that the concept of a sequence number--what being a sequence number *means*--is that of being the designator of a unique position in a sequence of elements. We all know how to use numbers to count--by pointing to each item and calling out, in succession, the names of the numbers. We therefore know that an item designated as n^{th} in a sequence comes immediately after the item designated as $n\text{-}1^{th}$, and immediately before the item designated as $n+1^{th}$ (with the usual caveats about the first and last items). We understand this *single* concept of a numerically designated unique position in a sequence so well that we are comfortable using it in place of the *two* concepts of predecessor and successor.

To contrast the two approaches, imagine that, with the original approach, we ask each row "Who is your predecessor?" and "Who is your successor?" And the answer we get is the name, *i.e*. the unique designation, of the predecessor and the successor for each one. In the case of our example, it is the loc-type-eid of each one--or else the answer "{null}" for successor of the last row and for predecessor of the first row in the sequence.

With the sequence number approach, on the other hand, we ask each row "When the numbers 1, 2, 3, ...n, in that order, were correlated with the members of your set, in the order defined for that set, what number were you given?" And the we get is the assigned sequence number of each one. (The more succinct form of this question, of course, is, "What sequence number do you have?")

So the reason that a sequence number seems to be a more succinct way to model a linear sequence is that it is a more complex concept that either the predecessor or successor concepts. But it is mathematically based on the predecessor and successor concepts. Indeed, the sequence number concept is part of arithmetic, and the predecessor and successor concepts are part of the set-theoretic foundations of arithmetic, defined by Frege, Whitehead and Russell. From this perspective, the approach shown in Figures 4.1 - 4.3 is a set-theoretic approach. The approach we are about to look at now is an arithmetic approach.

Sequentially Recursive Tables: the Sequence Number Approach.

One way to use a sequence number would be to modify the location type code values themselves, for example by adding a numeric prefix to them. This would result in the code table shown in Figure 4.4.

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

LOCATION TYPES

loc-type-eid	*loc-type-code*	*loc-type-desc*
0582	6-WRHSLOC	Equipment Warehouse
1448	8-AGFLDLOC	Equipment-in-Field Location, Above-Ground
2356	5-EQPLOC	Equipment Location
2718	2-AFFBLDG	Affiliate Building
3899	9-BGFLDLOC	Equipment-in-Field Location, Below-Ground
3234	3-HQBLDG	Headquarters
4440	4-BRBLDG	Branch Office Building
5678	7-FLDLOC	Equipment-in-Field Location
6021	1-OFFBLDG	Office Building

Figure 4.4. A Sequential RT Construct Using Sequence Numbers --Sample Extension-- Version 1.

However, this is just a thoroughly bad idea. Practically speaking, it's a bad idea because change happens. Theoretically speaking, it's a bad idea because it makes each code value a homonym.

Let's look at the practical issues first. Suppose that the sequence of these rows changes, so that row # 2718 (*i.e.* row with loc-type-eid = 2718) should now be third in the sequence, and row #3234 should now be second. How should the table be modified to reflect this change?

Since the sequence is indicated only by means of the code values themselves--specifically their numeric prefix--it is those values themselves which must change. So the value "2-AFFBLDG" must be changed to "3-AFFBLDG", and the value "3-HQBLDG" must be changed to "2-HQBLDG".

Some users are sure to be confused by this. Of course, many users will understand the convention that the first character of the code values themselves indicates position in a linear sequence, while the remaining three characters constitute the "code proper". And they will then be able to understand that "2-AFFBLDG" and "3-AFFBLDG" both indicate the AFFBLDG location type code, and that the change indicates its change of position in the linear ordering of those values. Nonetheless, some confusion will certainly be engendered.

And what about queries and code? If they were based on querying the full code value, then they will all have to be changed. On the other hand, if the queries and code ignored the first two characters of the code value, is that not a clear indication that we should have set up the sequence number as a separate attribute, and that by not doing so, we have made queries and procedural code unnecessarily complex?

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

Now let's look at the theoretical issues. In linguistic terms, by adding the sequencing prefix to the codes themselves, we created homonyms. We made each code value mean two things--both what the original code value meant, and then, in addition, its position in the sequence of code values.

Some might express the same point by saying that the sequence number prefix, added to the code itself, makes the code value non-atomic, and decomposable. And this is indeed true. But by expressing the point, as I have above, in linguistic terms, I have explained *why* non-atomic, decomposable values are undesirable. They are semantically non-atomic, and make each code value mean two things, *i.e.* be a homonym.

Hence, the improvement that practical considerations recommend to us--that we remove the sequence number and make it a separate attribute--is reinforced by linguistic considerations. For in linguistic terms, by doing so, we are replacing a homonym by two concepts neither of which are homonyms. The code is no longer a homonym because it now means the one thing it originally meant. And the new sequence number attribute is not a homonym, because it means just one thing--position within a linear sequence of values for the code.

So making the change, our location type table (prior to the resequencing of the two code values) now looks like this:

LOCATION TYPES

loc-type-eid	*loc-type-seq-nbr*	*loc-type-code*	*loc-type-desc*
0582	6	WRHSLOC	Equipment Warehouse
1448	8	AGFLDLOC	Equipment-in-Field Location, Above-Ground
2356	5	EQPLOC	Equipment Location
2718	2	AFFBLDG	Affiliate Building
3899	9	BGFLDLOC	Equipment-in-Field Location, Below-Ground
3234	3	HQBLDG	Headquarters
4440	4	BRBLDG	Branch Office Building
5678	7	FLDLOC	Equipment-in-Field Location
6021	1	OFFBLDG	Office Building

Figure 4.5. A Sequential RT Construct Using Sequence Numbers --Sample Table-- Version 2.

With version 1 of our table, we could have continued to use the DDL which created the original, unordered table. We could have done this even though we very obviously changed the semantics of the table--the type of information the table contains. The original table contained a set of codes with no ordering among them. But with version 1 of our linearly ordered set of codes, we somehow introduced additional semantics--the

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

sequencing of the codes—but *without* changing the DDL definition of the original basic table. It follows that we *must* have created a homonym, and that the homonym will be the piece of the original table's structure that now carries the additional information about sequencing. As we saw above (in Figure 4.4), that homonym is the loc-type-code itself.

What we need to do is to change the structure of the table to recapture the property that every structural element stands for one semantic constraint, and each semantic constraint is represented by one structural element. Here, we do that by changing the structure of the table to that shown in Figures 4.6 and 4.7.

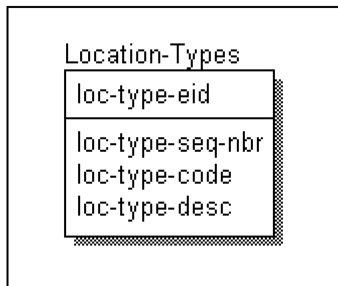

Figure 4.6. A Sequential RT Construct Using Sequence Numbers--Diagram.

The generic DDL representation of this linearly ordered code table is shown in Figure 4.7.

```
CREATE TABLE  Location-Types
        (
           loc-type-eid
           loc-type-seq-nbr
           loc-type-code
           loc-type-desc

                 PRIMARY KEY (loc-type-eid)
        )
```

Figure 4.7. A Sequential RT Construct Using Sequence Numbers --Generic DDL.

In this form, the resequencing of the two code values is reflected by interchanging their loc-type-seq-nbr values. Queries and code that reference the code values themselves will remain unaffected. Queries and code that reference the position of a code value within the set will be affected--as they should be.

So one kind of additional constraint placed on a set is the constraint of a simple linear sequence. And Figures 4.5 - 4.7 model a set of linearly sequenced location type codes, as we have seen, with a construct—a sequence number--which indicates *both* the immediate

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.

predecessor and successor for each code. If we were less familiar with sequence numbers, and did not intuitively understand how they indicate both predecessor and successor, then sequence numbers would be homonyms. However, because the concept of a sequence number is so well understood, those two concepts are no longer distinct, but are part of the definition of what a sequence number is. In this way, the sequence number concept is not a homonym; it is merely a concept that incorporates the concepts of predecessor and successor.

Let's turn now to a less rigid constraint, one that consequently permits us to express a more complex relationship among the members of a set--a hierarchical relationship.

© Copyright 2008, Dr. Tom Johnston. Only personal, non-commercial copies, are permitted.