

Relational Data Modeling With Recursive Constructs: Part 3.

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.

Matrix-Like Recursive Tables: Modeling Network-Structured Sets.

There is one more step we can take, by way of relaxing the constraints of a complete ordering. Instead of relaxing *either* the single predecessor *or* the single successor constraint, we can relax *both* of them.

One consequence of doing so is that we then cannot designate (except arbitrarily) either a first or a last member in the set. For a hierarchy, we kept a first element—the root node of the tree—while losing the concept of a last element. The first element was the one which had no predecessor. But in a matrix-related set, as I shall call it, any node may have multiple predecessors as well as multiple successors. We cannot, therefore, traverse the structure in either direction, and hope to find a unique node, such as the root node of a tree structure. Indeed, in such a situation, the notions of predecessor and successor, *i.e.* of moving in one direction rather than another, loses any mathematical basis.

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".

Sequentially recursive tables, as we have seen, are modeled with a one-to-one recursive relationship, while hierarchically recursive tables are modeled with a one-to-many recursive relationship. Matrix recursive tables, as we shall now see, are modeled with a many-to-many recursive relationship.

A table with matrix recursion is one in which any instance can be related to any number of other instances, on either the "from" or the "to" side of the relationship. On the "from" side, the primary key of instance X is a foreign key in any number of instances Y, Z On the "to" side, any number of instances Y, Z can be the corresponding "from" instance.

To this point, we have used location type codes to illustrate recursion within tables. But now that we are dealing with a many-to-many recursive relationship, it is difficult to think of a code type table that has this kind of relationship among its individual codes. So instead of codes indicating types of locations, let's shift, now, to a table of individual locations. We'll let the locations represent warehouses. The relationship among these locations which we will model as a matrix recursive table is the relationship in which a warehouse X ships parts to a warehouse Y.

Shipping from location X to location Y is a directional relationship. That is, because location X can ship to location Y, it does not follow that location Y can ship to location X. Figure 4.19 shows these locations, and which ones can ship to which other ones.

Let's call each instance in which one location can ship to another location a "route segment".



Figure 4.19. Locations and Route Segments.

These route segments are a many-to-many relationship among the locations. For example, we can see that location L2 ships to two locations, and that three locations ship to L2.

As we know from standard relational modeling principles, a many-to-many relationship is implemented by means of an "associative" table. The associative table--which I call a "matrix" table for reasons I explained earlier--has a candidate key (almost always functioning as the primary key) which is a concatenation of the primary keys of the two entities which have the many-to-many relationship between them. In this case, however, because the relationship is recursive, the two related tables are actually one and the same table! Therefore, the primary key of this table will consist of two foreign keys, both of which come from the same table, the Locations table. One of those keys is the key of the ship-from location. The other is the key of the ship-to location.

The modeling construct, then, which shows both the locations and the route segments between them, is actually a pair of tables, because the many-to-many relationship requires a separate associative table to model it. Nonetheless, we are still dealing with a recursively-defined table, because the associative table exists only to model the recursive relationship among rows of the Location table. Therefore, to avoid referring to two physical tables as one table, let us refer to them together as one "table construct".

Figure 4.20, then, shows a Location table construct, which includes the matrix table which represents the many-to-many recursive relationship among the locations.

LOCATIONS

<i>loc-oid</i>	<i>loc-code</i>	<i>loc-desc</i>
9679	L1	New York
0455	L5	Colorado Springs
1677	L2	Atlanta
1731	L4	Dallas
2270	L3	Chicago
3400	L6	Seattle

Figure 4.20. A Matrix RT Construct: Base Table.

LOCATION MATRIX

<i>loc-from-eid</i>	<i>loc-to-eid</i>
9679	1677
9679	3400
9679	0455
9679	1731
0455	1731
0455	3400
1677	2270
1677	1731
1731	2270
2270	1677
2270	3400
3400	1731
3400	2270
3400	1677

Figure 4.20. A Matrix RT Construct: Relationship Table.

The Location Matrix table, shown in Figure 4.20, models the recursive relationship among the locations. This relationship indicates which locations can ship to which other locations. Each row, then, represents one route segment.

From the Location Matrix table, we can easily obtain some interesting information. For example:

- New York doesn't receive shipments from any other location.
- Every location sends shipments to at least one other location.
- Colorado Springs receives shipments from only one other location.

The data model diagram for this matrix RT construct is shown in Figure 4.21.

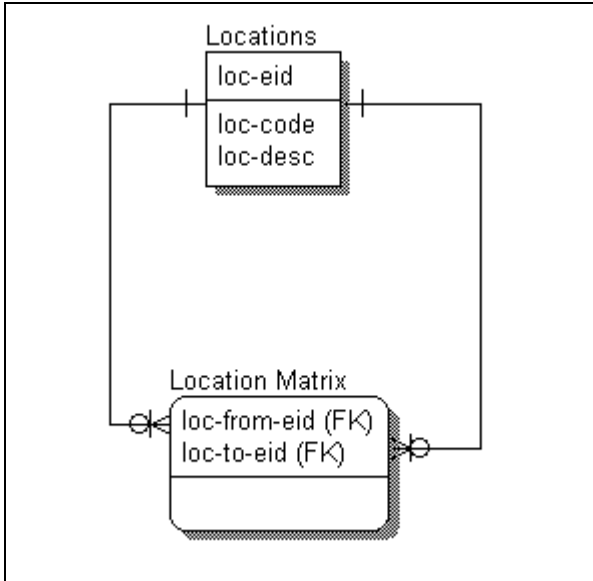


Figure 4.21. A Matrix RT Construct: Diagram.

insert diagram here.

Figure 3.22. A Matrix RT Construct: Generic DDL.

Additional Semantics for Matrix Tables.

Let's suppose, next, that when a location begins to run out of whatever it ships ("parts", let's say), it must prioritize to whom to send its remaining parts. How can we model this prioritization of locations to which a given location sends parts?

The answer lies in observing that the prioritization is a linear ordering. What is being asked for is to *sequence* all the receiving locations for one sending location in a single, linear sequence. And by now, of course, we know how to impose a linear ordering on a set. In fact, we know two different ways to impose a linear ordering on a set.

So let's take the easier of the two approaches, and use a sequence number.² The result is shown in Figure 4.23.

LOCATION MATRIX

<i>loc-from-eid</i>	<i>loc-to-eid</i>	<i>loc-ship-to-priority</i>
9679	0455	1
9679	1677	2

² A good exercise would be to use the predecessor/successor approach to model the same linear ordering.

9679	1731	3
9679	3400	4
0455	1731	1
0455	3400	2
1677	1731	1
1677	2270	2
1731	2270	1
2270	1677	1
2270	3400	2
3400	2270	1
3400	1677	2
3400	1731	3

Figure 4.23. A Matrix RT Construct With a Linear Ordering.

Next, let's suppose that we must model the multiple segment *routes* that certain shipments can take, where a route is a sequence of segments. And let's suppose that the possible routes are the following:

<i>Routes</i>	<i>Route Segments</i>
Route 1:	L6-->L2-->L4-->L3-->L6
Route 2:	L4-->L3-->L2-->L4
Route 3:	L3-->L2-->L4-->L3
Route 4:	L2-->L3-->L6-->L4-->L3-->L2

Figure 4.24. Routes and Route Segments.

We can immediately note that the relationship between routes and segments is M:M. For example, segment L2-->L4 is used by three of the four routes shown. Therefore, we know that we will need an associative table, to relate Routes to Route Segments.

The Route Segment table already exists. It is the Locations Matrix table. So we need two new tables.

One is a Routes Table. Not bothering to show any attributes of routes besides their EIDs and names, this table is shown in Figure 4.25.

<i>route-eid</i>	<i>route</i>
4851	R4
5273	R1
6496	R3
7842	R2

Figure 4.25. Routes.

The second additional table we need is an associative table, relating Routes and Segments. This Route/Segment Matrix table is shown in Figure 4.26.

ROUTES AND SEGMENTS MATRIX

<i>route-<u>eid</u></i>	<i>segment-loc-<u>from-<u>eid</u></u></i>	<i>segment-loc-to-<u>eid</u></i>	<i>segment-<u>seq-nbr</u></i>
4851	1677	2270	1
4851	2270	3400	2
4851	3400	1731	3
4851	1731	2270	4
4851	2270	1677	5
5273	3400	1677	1
5273	1677	1731	2
5273	1731	2270	3
5273	2270	3400	4
6496	2270	1677	1
6496	1677	1731	2
6496	1731	2270	3
7842	1731	2270	1
7842	2270	1677	2
7842	1677	1731	3

Figure 4.26. Routes, Segments and Sequences.

Figure 4.27 A Matrix RT Construct—Diagram.

Figure 4.28. DDL

Summary

Let’s take stock of where we are, in our development of recursive constructs. To begin with, many tables represent unordered sets. Our basic RT construct of a single table with three attributes—EID, code and description—is an example of a table representing an unordered set.

Next, we modeled the simplest ordering possible among the members of a set—a complete ordering. We found that we could model a complete ordering by separately specifying a single successor and a single predecessor for each row except the first and last rows—rows which had, respectively, no predecessor and no successor.

We then realized that counting with numbers is so universally well understood a process that we could specify a predecessor and a successor jointly, by means of such a number. The row with number n had one unique predecessor and one unique successor—the rows with numbers $n-1$ and $n+1$, respectively (with the usual qualifications for the first and last rows).

I think it is important to understand both ways of modeling a complete ordering, *i.e.* a sequence, because it shows just how much semantics is expressed by so simply a thing as a counting number.

We next decided to relax either the predecessor or the successor constraint (for there still were *two* constraints, even when we expressed them with a single homonymous construct, *i.e.* the sequence number). To see what is going on, think of the two constraints as applying to a set of points which, in the following diagram, we will arrange vertically.

insert diagram with five dots, aligned vertically, with lines connecting them.

Figure 4.29.

The points at the top and bottom we will take to represent the first and last points in the sequence. They therefore have, as the diagram clearly illustrates, no predecessor and no successor, respectively. Now, suppose we relax one of the constraints, and allow either multiple predecessors or multiple successors. Either relaxation will allow this “stick” to branch out into a “tree”, so we would get either of the two following sorts of diagrams:

insert a diagram in which the tree branches upwards. It illustrates multiple predecessors.

Figure 4.30.

insert a diagram in which the three branches downwards. It illustrates multiple successors.

Figure 4.31.

Either diagram represents a tree structure, reflecting the fact that relaxing either the single predecessor, or the single successor, constraint, amounts to the same thing, mathematically.

It is remarkable that something as complex as what was just illustrated can be expressed by something as simple as a one-to-many recursive relationship. Moreover, this extremely simple structure can express *any* tree structure, no matter how many levels deep, or how great the cardinality of each level. And even more than that, any number of unlinked tree structures can be accommodated in one such table.

Perhaps the most important philosophical lesson to be learned from this is that some complexity is real, while other complexity is only apparent. The apparent complexity is created by a large number of iterations of simple structures.

```
{}{}{}  
finish with summary of many-to-many recursive structures.  
}}}}}
```