https://blueflag.consulting/blog/2023-01-02-migrating-sql-server-hierarchyid-to-postgresql-ltree-using-aws-dms-part-1/

**Migrating SQL Server HierarchyID to PostgreSQL LTREE using AWS DMS (Option 1: Full Load)**

***Alexander Witte***October 9, 2023

**Introduction**

Migrating an on-premises database to Amazon Web Services (AWS) can provide a number of benefits, such as improved scalability, availability, and disaster recovery. By leveraging the power of the cloud, organizations can easily scale their database up or down to meet the changing needs of their business. Additionally, by migrating to a cloud-based platform like AWS, organizations can take advantage of built-in disaster recovery features, such as automatic backups and multi-Availability Zone deployment options, which can help to ensure the availability of their data in the event of an outage or other disaster.  
  
AWS provides a handful of purpose-built tools for migrating resources into AWS. For database migrations, two of these tools are the Schema Conversion Tool and the Database Migration Service. The Schema Conversion Tool (SCT) handles the refactoring of the schemas and code objects to the target system, and the Database Migration Service (DMS) moves the data. While SCT can refactor most basic DDL, when it encouters something that is not compatible in the target system, it will be up to you to figure out a solution. This was encountered this event recently during a Microsoft SQL Server migration to AWS Aurora PostgreSQL, where some tables in the source system (SQL Server) were using the **hierarchyID** datatype. PostgreSQL does not have a hierarchyID datatype, so SCT and DMS had no idea what to do. This article provides a solution to recreate similar hierarchyID functionality in PostgreSQL using the **LTREE** datatype and how to perform a complete load migration using DMS.

*(An ongoing (CDC) migration solution will be provided in a later article.)*

**What is the hierarchyID data type?**

The hierarchyID data type in SQL Server is a data type that enables the modelling of a hierarchical tree structure. Typical examples could be organization charts, folder structures etc- each of these having some notion of a root node and then children of the root node. For this article, we will use the Earth as an example: The root of our tree will be the Earth and below that will be continents (i.e. Africa or North America), then countries (ie. Canada, USA) and so on.

**Figure 1 -**High Level Diagram

The HierarchyID data type is stored as a variable-length, binary-encoded string, so you will see some binary hex when you examine it. To view the hierarchyID tree path, you can call its ToString() method. The string notation is numerical, delimited by a slash. In our Earth example, the notation will be as follows:

| **Locations** | **Notation** |
| --- | --- |
| Earth | / |
| Africa | /1/ |
| North America | /2/ |
| Canada | /2/1/ |
| USA | /2/2/ |

Here’s a look at this tree in a table. **hierarchyID\_obj** is the binary encoded string, **hierarchy\_str** is the string representation, and **name** is a string created to give us some context.

| **#** | **hierarchyID\_obj** | **hierarchy\_srt** | **name** |
| --- | --- | --- | --- |
| 1 | 0x | / | earth |
| 2 | 0x58 | /1/ | africa |
| 3 | 0x68 | /2/ | northamerica |
| 4 | 0x6AC0 | /2/1/ | canada |
| 5 | 0x6B40 | /2/2/ | usa |

**What is an LTREE?**

The PostgreSQL ltree data type is used to represent and query hierarchical data in a PostgreSQL database. It stands for “labeled tree” and it is similar to the SQL Server HierarchyID data type, however with an LTREE the tree structure is delimited by ‘.’ instead of ‘/’. It is a PostgresSQL extension and therefore it will need to be enabled. Any string value can be used to describe the node in the tree (instead of the numerical notation required by the hierarchyID). So, to rewrite our Earth tree example it could look something like this:

| **Locations** | **Notation** |
| --- | --- |
| Earth | earth |
| Africa | earth.africa |
| North America | earth.northamerica |
| Canada | earth.northamerica.canada |
| USA | earth.northamerica.usa |

**Figure 2 -**high level with extended names

**Migration Challenges/decisions**

Having reviewed both data types, to recap our goal, we want to migrate a SQL Server table using the hierarchyID to a PostgreSQL table using the LTREE datatype. AWS DMS doesn’t support conversion between these dataypes so we’ll need to do this ourselves. In a nutshell we’ll need to grab the string representation of the hierarchyID, transform that to a format the LTREE can use, and then cast to the LTREE type.

Interestingly, while DMS won’t convert these datatypes for us, when left to its own devices (ie. just point DMS at the hierarchyID and let ’er rip) it will convert the hierarchyID binary into the varchar field of its hierarchyid path (ie. 0x58 -> /1/2/).

**Figure 3 -**DMS Flow Diagram

This is very helpful as it means we don’t need to alter the source table to add a column with the hierarchyID string representation. From here we can simply add a post migration step to reformat the hierarchyID syntax into LTREE syntax.

**Full Load Migration**

This section will contain the steps to migrate our SQL Server table to Aurora PostgreSQL. We will use the “Earth” table we created in this article. (*This will be a DMS Full Load migration, part 2 of this Blog series will contain a Full Load + CDC migration.)*

Our Migration plan will consist of the following steps:

1. Create DB and source table in SQL Server
2. Create schema, table and add LTREE extension in Postgres
3. Configure DMS Task and run Full Load migration
4. Apply post migration steps.

**Figure 4 -**Full load example

**Create DB and source table in SQL Server**

In SQL Server, create a database and a table. We will create the Earth tree as an example:

1**create** **database** migrationdemo;

2**GO**

3

4**use** migrationdemo

5**GO**

6

7**create** **table** earth (

8 id **int** **identity**(1,1) **primary** **key**,

9 node **hierarchyid**,

10 name **varchar**(20)

11)

12**GO**

13

14**INSERT** **INTO** earth (node, name)

15**VALUES**

16 ('/', 'earth'),

17 ('/1/', 'africa'),

18 ('/2/', 'northamerica'),

19 ('/2/1/', 'canada'),

20 ('/2/2/', 'usa');

**Create schema, table and add LTREE extension in Postgres**

In the PostgreSQL system to keep things simple we will use the default “postgres” database. We will need to add the LTREE extension and create the target table that we will migrate the data into. Note that the “node” field here is still a varchar. This will be cast into an LTREE in our post migration step. Log into Postgres and run these SQL commands:

1**create** **schema** dbo;

2

3**create** extension ltree **schema** **public**;

4

5**create** **table** dbo.earth (

6 id serial **primary** **key**,

7 node **character** **varying**(20),

8 name **character** **varying**(20)

9);

**Configure DMS Task and run Full Load migration**

The DMS task we will use will be extremely simple. Here are the important settings:

* Migration type: Migrate existing data
* Target table preparation mode: Do nothing
* Include LOB columns in replication: doesn’t matter, we have no LOBs
* Table Mapping: just make sure we’ve targeted the earth table. No transformations needed. Something like this should work:

1{

2 **"rules"**: [

3 {

4 **"rule-type"**: "selection",

5 **"rule-id"**: "1",

6 **"rule-name"**: "1",

7 **"object-locator"**: {

8 **"schema-name"**: "%",

9 **"table-name"**: "%"

10 },

11 **"rule-action"**: "include"

12 }

13 ]

14}

Run the task and when complete you should be able to see our earth table populated with data!

| **id** | **node** | **name** |
| --- | --- | --- |
| 1 | / | earth |
| 2 | /1/ | africa |
| 3 | /2/ | northamerica |
| 4 | /2/1/ | canada |
| 5 | /2/2/ | usa |

**Apply post migration steps**

At this point we should now have successfully migrated our table to PostgreSQL with our node field as a varchar. We simply need to run some SQL to convert the tree syntax and cast as an LTREE.

The code below firstly rewrites the hierarchy tree syntax to LTREE syntax. *(Instead of earth.africa etc it has been rewritten with the numerical values to keep simple and agnostic of context.)*

Secondly, the column is cast as an LTREE datatype.

1**update** dbo.earth **set** node =

2 **case**

3 **when** node = '/' **then** '0'

4 **else** '0' || **replace**(**left**(node, **length**(node) - 1 ),'/', '.')

5 **end**;

6

7**alter** **table** dbo.earth

8**alter** **column** node **type** ltree **using** node::ltree;

Checking the table we can see now that the datatype has been cast successfully as an LTREE.

| **id** | **node** | **name** |
| --- | --- | --- |
| 1 | 0 | earth |
| 2 | 0.1 | africa |
| 3 | 0.2 | northamerica |
| 4 | 0.2.1 | canada |
| 5 | 0.2.2 | usa |

**Conclusion**

This article has provided a solution for migrating the SQL Server hierarchyID datatype to the PostgreSQL LTREE using AWS DMS. A future blog article will provide a solution for an ongoing migration.