Semantically Controlled Any-to-Any Data Structure Transformation by Reshaping

Michael M David
mike@adatinc.com

Executive Summary

Today the data structure transformation terminology of Restructuring and Reshaping are used interchangeably for XML structure transformation processes. There are two basic types of XML hierarchical data structure transformations that need to be distinguished because they are different in meaning, results, and use. These are *restructuring* controlled by existing relationships in the data, and *reshaping* controlled by the semantics of the current data structure. Restructuring is performed by using new and unused relationships to restructure the data. While reshaping uses the semantics of the current structure to mold the structure into any other shape without requiring or relying on any data relationships in the data. The processing follows hierarchical semantics principles to derive correct hierarchical results.

1) Introduction

Restructuring using data relationships can create a new structure and data with new semantics, while reshaping using structure semantics alters the structure without changing the data and its semantics. Both have their use. Restructuring is usually used to match a structure to its application use, and reshaping to map a structure to a desired structure format. Reshaping is also used in inverting structures when data relationships are not available in the data. This is often the case with XML data since foreign keys are not required because of XML's contiguous nested storage. The new reshaping technique described and shown in this paper enables unlimited any-to-any structure reshaping.

1.1) Hierarchical Structure Processing Definitions

You will notice in the diagrams used in the examples in this article that there are a number of different symbols used to define the hierarchical processing. These are used throughout this article. A solid box indicates node selected for output. A dashed box indicates an unselected node that is sliced out of the query returned result. A solid line connects nodes into active structure. A solid arrow represents the data modeling structure node linkage between the structures being joined. This is used to complete the unified data structure of all the structures joined. This represents the semantics of the structure which naturally controls the processing of the query.

The Solid arrow that represents the data modeling structure linkages between structures being joined also usually specifies the ON clause data relationship linkage points. It is possible for reasons explained later that the ON clause data relationship linkage points can be different from the data modeling solid arrow connecting the data structures. In this case a dashed arrow is used to indicate the ON clause data relationships linkage points. These Hierarchical Structure Processing Definitions are shown below.

Hierarchical Structure Processing Definitions

Solid Box: Node is selected for output
Dashed Box: Node is not selected for output
Solid Line: Connects nodes
Solid Arrow: Modeled Structure connector
Dashed Arrow: Relationship structure linkage

1.2) Any-to-Any Hierarchical Structure Reshaping Using Semantics

Reshaping transformation does not rely on any established relationships but uses the structure's natural data semantics to enable reshaping the structure into any other structure with the same node types. This is performed by duplicating the same structure as many times as necessary using the renaming and correlation prefix name used and to link the occurrences of the structures together in any desired manner by matching on the same unique controlling field values in the joined copies of the structure. This increases the range of transformations allowing any structure to be shaped into any other hierarchical structure while preserving the data and maintaining or altering its semantics depending on the new shape.

Internally the new semantic associations between the selected nodes logically persist even after all other nodes are removed by lack of selection so that the existing nodes are still related to each other in the same way. The desired nodes' data is selected at its required level. Since this is the only level it is selected at, all other node levels for this node type are not selected for output and the resulting structure is nicely compressed to only select nodes by the natural process of node promotion discussed previously.

1.3 Basic SQL Hierarchical Modeling

The following example relationships represented in CustViewT below in Figure 1.3 are used to generate test data for reshaping examples to follow. They are mostly 1 to M (One to Many) the most common for hierarchical structures. When nodes become inverted by re-shaping these relationships will be changed to M to 1 which should flatten the data and possibly lose some of the associated data because of hierarchical preservation principles. The following examples will demonstrate these hierarchically principled operations, but first let's look at the example structures and their data.

The examples of reshaping will be shown using SQL which can perform these operations without navigation or looping constructs. They do require some thought to how they are specified as you might expect for transformations. All transformation operations carried out will be performed semantically correct. This is because the operations are performed through the existing hierarchical structure strictly following the semantics in the structure controlled by SQL automatically because SQL's data in its rowsets is modeling hierarchical data using the hierarchical Left Outer join as can be seen in the CustViewT hierarchical view below in Figure 1.3. This aids the user specifying the hierarchical transformations considerably. This makes it easier for the user to specify complex transformations without introducing any errors either from the user or the software. The new SQL high level techniques and solid hierarchical principles used here for reshaping are still valid for navigational and procedural transformations used in other XML processors, but nonlinear transformations may be too difficult to perform procedurally, but SQL offers a nonprocedural capability that follows the hierarchical principles automatically.

View CustViewT below defines the full hierarchical structure

```
CREATE View CustViewT AS
SELECT * FROM CustT Customer
LEFT JOIN InvoiceT Invoice ON CustID=InvCustID
LEFT JOIN EmpT Employee ON CustID=EmpCustID
LEFT JOIN DpndT Dependent ON EmpID=DpndEmpID
```


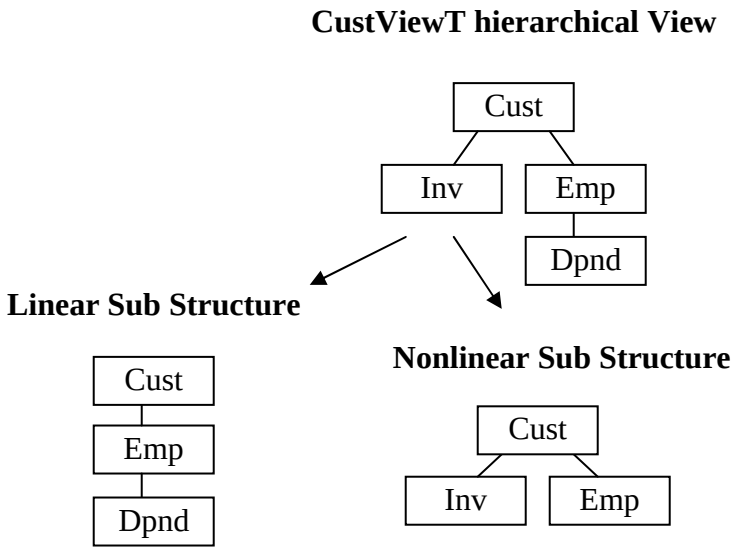

Figure 1.3 Example Structures derivable from CustViewT

The reshaping SQL shown with the reshaping examples in the rest of this paper have an SQL identification number that can be used to fetch the SQL statement and then executed in real-time by an ANSI SQL XML hierarchical processor prototype. The SQL examples can be modified. The hierarchical processor prototype and directions for use can be found at: www.adatinc.com/demo.html.

1.4 Linear Sub Structure Data Example

This linear structure data is achieved from the CustViewT view be SELECTing data only from nodes that represents a linear structure.

SQL15.01: **SELECT CustId, EmpID, EmpCustID, DpndID, DpndEmpID**
FROM CustViewT

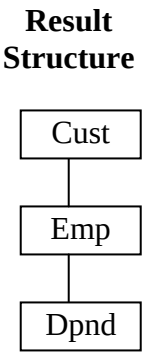

Figure 1.4 Linear Structure derived from CustViewT

1.5 Nonlinear Sub Structure Data Example

This nonlinear data structure is achieved from the CustViewT view be Selecting data only from nodes that represents a nonlinear structure. In this case, its data from the Cust, Emp and Inv nodes as shown below. The Inv and Emp nodes are sibling nodes making the structure nonlinear with Cust as the common parent. The SQL middleware will automatically remove the duplicates caused by the relational Cartesian product which are usually prevalent between siblings.

SQL15.02: **SELECT CustId, InvID, InvCustID, EmpID, EmpCustID**
FROM CustViewT

Result Structure

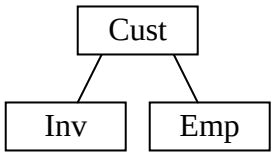

Figure 1.5 Nonlinear Structure derived from CustViewT

2.1.0 Linear Inversion Logic

Linear structures are simpler than nonlinear structures so we will start with them first. Our first reshaping example in Figure 2.1.0 will be the inversion of a two level structure of Cust over Emp. This can be performed by making a second copy of the structure and then joining one over the other in such a way that we can create and extract through joining across structures an Emp over Cust fragment. There are two ways to join the two structures by putting one node over the other. These are either using the join criteria of first.Cust=second.Cust or first.Emp=second.Emp. In Figure 2.1.0 below, the first example on the left using the Cust nodes to join on does not produce the right combinations for EMP over Cust. The second join operation joining on first.Emp=second.Emp does produce Emp over Cust from SQL2.1.0 below.

SQL15.10: **SELECT First.EmpID, First.EmpCustID, Second.CustID, Second.CustStoreID**
FROM CustViewT First LEFT JOIN CustViewT Second ON First.EmpID=Second.EmpID

The above join is aided by the capabilities and rules for joining a lower level structure below the lower level root which states that the actual data modeling join point remains the root of the lower level structure, Cust in this case. The Select clause is used to select the correct combination of nodes as in: SELECT First.EmpID, Second.CustID, utilizing prefixes to specify the correct duplicate named nodes created by the self joins.

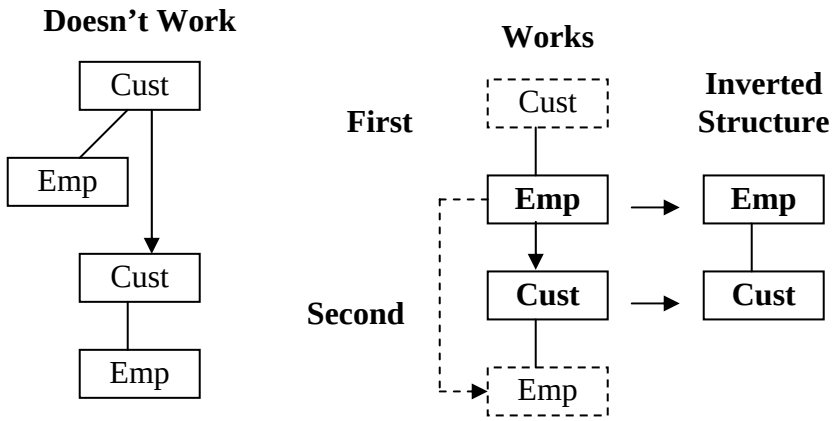

Figure 2.1.0: Simple two level linear inversion

The example in Figure 2.1.0 has been used to explain how structure reshaping is performed. The following three level linear inversion is used to explain the examples. But as long as we are here, a review of the diagram symbols as used above would be a good idea. A dashed arrow represents the ON clause alignment node linkage points from the upper structure to the lower structure which could be below the root. If there is no dashed arrow, this means the solid arrow also species the ON clause linkage points. Solid arrow represents the interpreted data modeling structure linkage between the structures being joined. This is used to complete the unified data structure of all the structures joined. This represents the semantics of the structure which naturally controls the processing of the query. A solid box indicates a SELECTed node. A dashed box indicates an unselected node that is sliced out of the query returned result.

2.1.1 Inverting a 1 to M Linear Three Level Structure Reshaping

This is a longer linear structure inversion using Cust over Emp over Dpnd (a series of 1 to M relationships). This will demonstrate that the alignment joining starts at the bottom of the structures, and drives the inversion upward at each node as the selected inverted node from each level is selected: SELECT first: X.Dpnd, second: Y.Emp, third: Z.Dpnd. Only these three nodes are selected once each for output so they are squeezed together and keep their structure which is naturally inverted and is now M to 1 relationships.

CREATE VIEW CustViewInvert: **CREATE VIEW CustViewInvert AS**
SELECT X.DpndID, X.DpndEmpID, Y.EmpID, Y.EmpCustID, Z.CustID, Z.CustStoreID
FROM CustViewT X LEFT JOIN CustViewT Y ON X.DpndID=Y.DpndID
LEFT JOIN CustViewT Z ON Y.EmpID=Z.EmpID

SQL15.11: **SELECT * FROM CustViewInvert Where CustStoreID='Store01'**

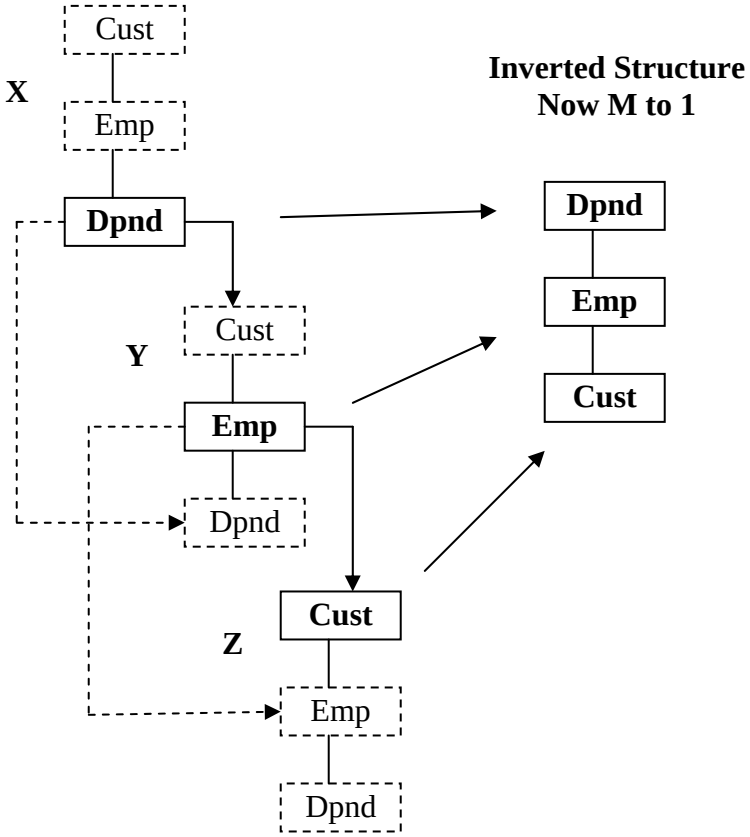

Figure 2.1.1: Complex linear inversion

2.1.2 Inverting a Linear M to 1 Three Level Structure

This structure inversion actually takes the output of the previous inversion which flattened the 1 to M structure into an M to 1 structure and re-inverts it. The previous inverted structure is re-created by the CustViewInvert view used in the Figure 2.1.2 example. This example should not only change the structure back into a 1 to M structure and remove the duplicates, but test if it is smart enough to also recognize the valid replications and return the structure in 1 to M hierarchically structure format which regroups the multiple occurrences with only a single parent occurrence (renormalize).

SQL15.12: **SELECT X.CustID, X.CustStoreID, Y.EmpID, Y.EmpCustID, Z.DpndID, Z.DpndEmpID**
FROM CustViewInvert X LEFT JOIN CustViewInvert Y ON X.CustID=Y.CustID
LEFT JOIN CustViewInvert ON Y.EmpID=Z.EmpID

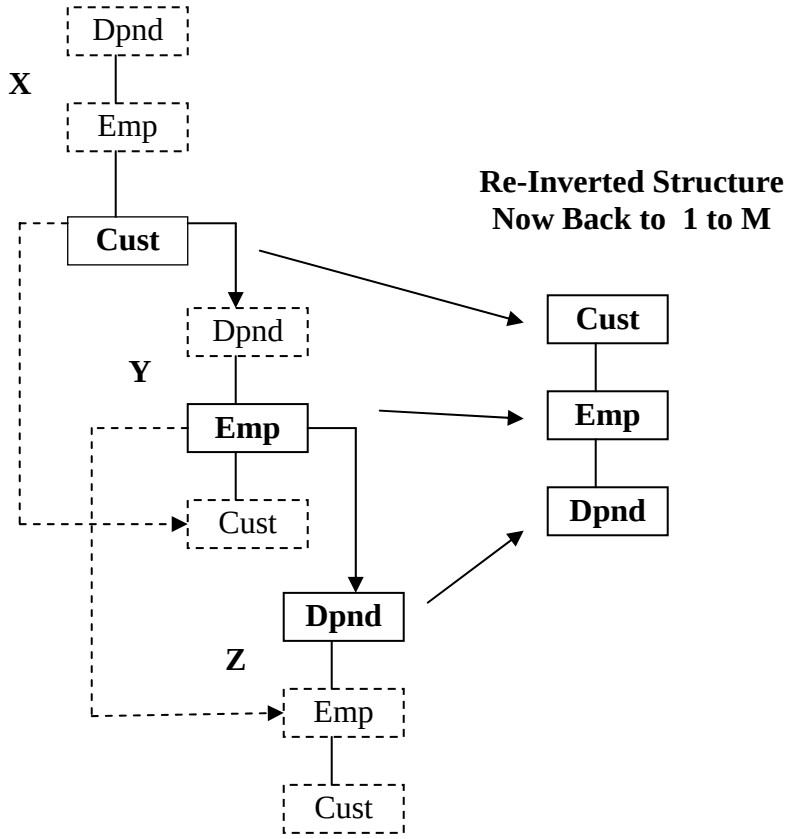

Figure 2.1.2: Re-inverts previous linear inversion example

2.2 Linear to Nonlinear Reshaping Logic

Interestingly, linear structures can be reshaped into nonlinear structures and the replication of input structures used in this section can be used to demonstrate this.

2.2.1 Linear to Nonlinear Preserved Semantics Reshaping

In this example the linear structure Cust over Emp over Dpnd can be used to generate the structure Emp directly over the siblings Dpnd and Cust. Two copies of the Input structure are necessary and are joined by their common Emp node since it is the starting node to building the new structure. Only two copies are necessary because the first copy can be used to move two nodes, Emp over Dpnd, which are already in place. This places first.emp over the first.Dpnd and first.Cust siblings. This creates the nonlinear structure desired and these same node values will be selected to create the nonlinear structure desired. By placing Emp over Cust, Cust is naturally and correctly replicated.

The previous linear examples and this nonlinear example have not lost the semantics of the input structure in the new structure because the semantics have been kept the same or inverted. This means the nodes have basically kept attached to the same nodes as shown below in the derived result structure. Emp over Dpnd remains the same while Emp over Cust has been inverted. In Figure 2.2.1 below result will be accurate and models the result structure shown.

SQL15.21: **SELECT X.EmpID, X.EmpCustID, Y.CustID, Y.CustStoreID,**
X.DpndID, X.DpndEmpID
FROM CustViewT X LEFT JOIN CustViewT Y ON X.EmpID=Y.EmpID

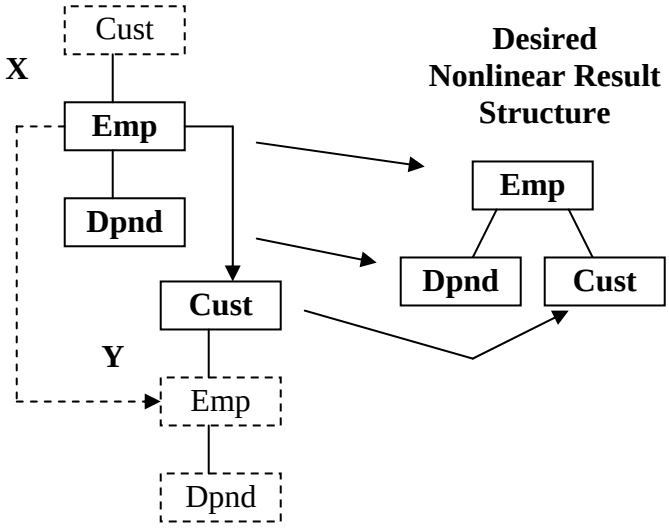

Figure 2.2.1: Linear to nonlinear reshaping with same semantics

2.2.2 Linear to Nonlinear Preserved Semantics Reverse Legs Reshaping

This is the same reshaping as the previous 2.2.1 query example except the sibling paths are reversed. Unfortunately because of the placement of the data in the input structure, the proper structure combinations do not become available with only a single structure duplication. This time it takes three copies of the structure to have Cust be the left sibling shown in Figure 2.2.2 below. But this is a good example that any number of copies can be used until the desired structure can be modeled and produced with no side effects. Sometimes multiple reshaping moves can be performed at a single level, sometimes only one reshaping move. Keep in mind that any reshaping move can be automatically following a chain of many intervening nodes without incurring any overhead of having to recreate the joins since they have already been performed in memory.

SQL15.22: **SELECT X.EmpID, X.EmpCustID, Y.CustID, Y.CustStoreID, Z.DpndID, Z.DpndEmpID**
FROM CustViewT X LEFT JOIN CustViewT Y ON X.EmpID=Y.EmpID
LEFT JOIN CustViewT Z ON X.EmpID=Z.EmpID

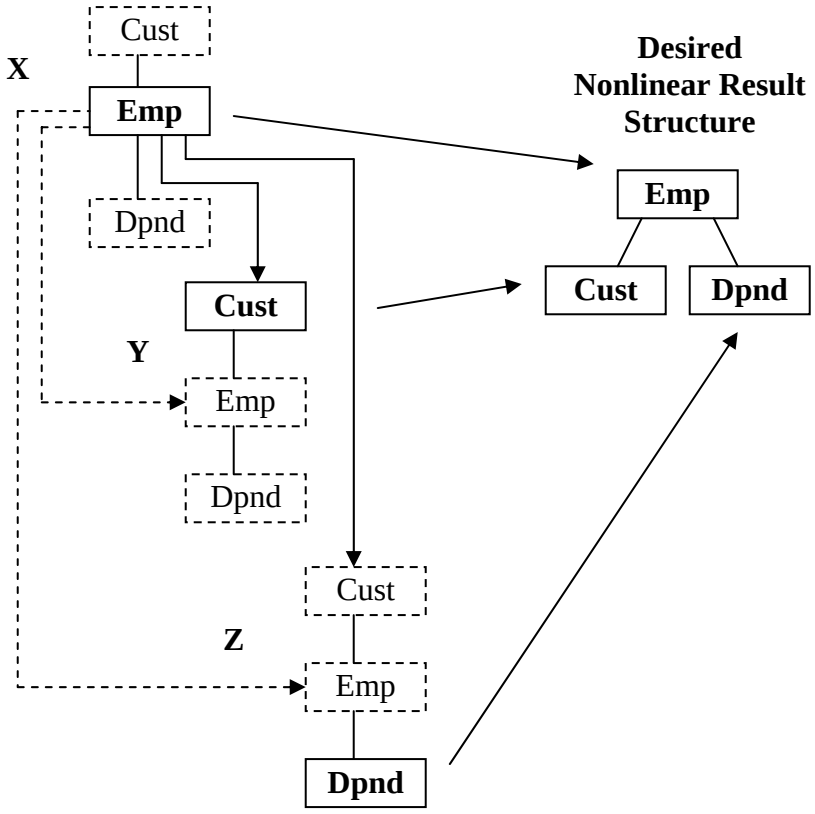

Figure 2.2.2: Linear to nonlinear reshaping reverse legs from previous example

2.2.3 Linear to Nonlinear Indirectly Related Semantic Reshaping

The difference with this linear to nonlinear Figure 2.2.3 example from the previous Figure 2.2.2 example is that the semantics of the result structure have been changed. Cust over Emp is the same semantics but Cust over Dpnd is indirectly related through Emp as can be seen in the input structure below. With Emp relocated in a different location in the output structure there is no natural link between Cust over Dpnd in the result structure. This does not invalidate reshaping since Cust was related to Dpnd the same as if the Emp node was not selected for output and Dpnd was node promoted up directly under Cust.

SQL15.23: **SELECT X.CustID, X.CustStoreID, X.DpndID, X.DpndEmpID,**
Y.EmpID, Y.EmpCustID
FROM CustViewT X LEFT JOIN CustViewT Y ON X.CustID=Y.CustID

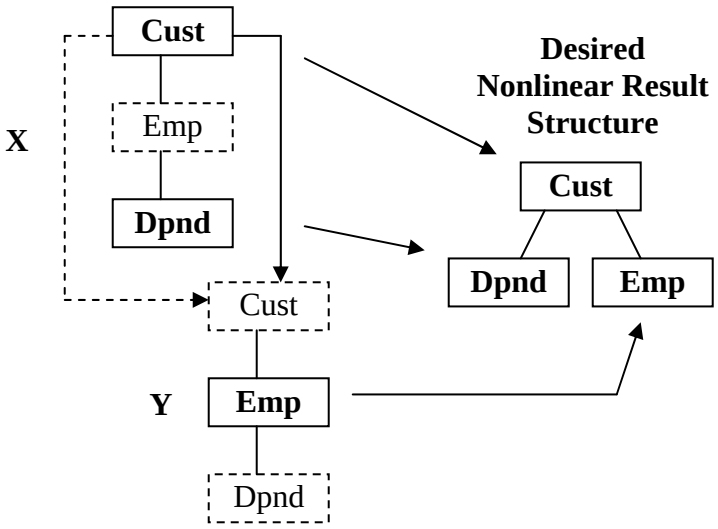

Figure 2.2.3: Linear to nonlinear reshaping with changed semantics

2.3) Nonlinear to Linear and Nonlinear Reshaping

Nonlinear structures as input can also be used to build linear and nonlinear structures. In fact, nonlinear structures offer more flexibility in how they are utilized because their multiple legs offer more opportunity to find the correct reshaping being sought. This means less input copies need to be used.

2.3.1 Nonlinear to Linear Reshaping

The below nonlinear input structure can be duplicated to create a linear structure reshaping of itself using Figure 2.3.1. Since we are starting with Inv as the root, this will be the first matching link. Cust becomes available to SELECT in the second level structure which is valid since it is related to Inv to the related link point. Emp can be selected from the same structure copy since it is already located under Cust. In effect, it is possible to reach Emp from Inv directly in memory.

SQL15.31: SELECT Y.CustID, Y.CustStoreID, X.InvID, X.InvCustID, Y.EmpID, Y.EmpCustID
FROM CustViewT X LEFT JOIN CustViewT Y ON X.InvID=Y.InvID

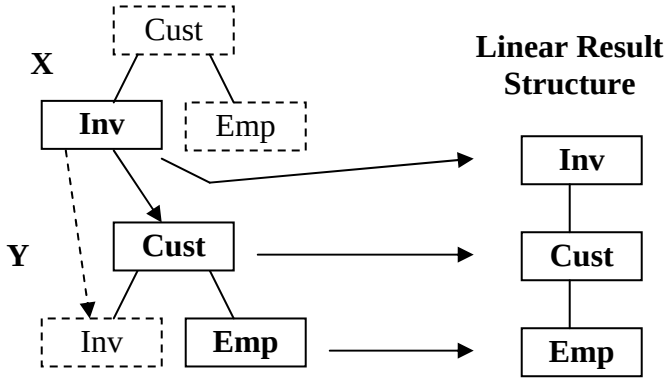

Figure 2.3.1: Nonlinear to linear reshaping example

2.3.2 Nonlinear to Nonlinear Reshaping

This nonlinear to nonlinear example is very similar to the previous nonlinear to linear Figure 2.3.1 example where the linear structure Inv over Cust over Emp was produced easily using Figure 2.3.2. This is an example demonstrating that nonlinear structures can produce nonlinear structures so this example copies the previous example but places Emp not under Cust but under Inv making Cust and Emp siblings for this structure. This requires a third copy of the input structure also matched to Inv because that is where Emp is being attached to. Emp is accessed indirectly from Inv up to Cust down to Emp a powerful related semantic reshape. While this produces the same result as the previous example, it is correct. In both structures, this one and the previous one, Emp is or was indirectly related to Inv. The second structure and additional join in this example was necessary to move Emp from under Cust to under Inv.

SQL15.32: **SELECT Y.CustID, Y.CustStoreID, X.InvID, X.InvCustID, Z.EmpID, Z.EmpCustID**
FROM CustViewT X LEFT JOIN CustViewT Y ON X.InvID=Y.InvID
LEFT JOIN CustViewT Z ON Y.InvID=Z.InvID

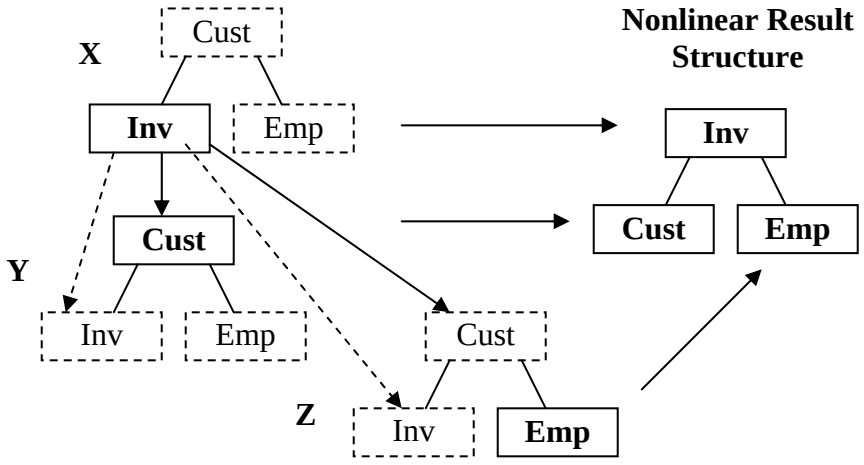

Figure 2.3.2: Nonlinear to nonlinear reshaping example

2.4 Polymorphic Reshaping

Polymorphic reshaping does not rely on the structure of the input structure. The advanced reshaping capability shown in this paper does support polymorphic reshaping when only one node is moved per join. The example in Section 2.2.1 is not polymorphic because it relies on a specific structure by moving two related nodes at one time while the same basic reshaping performed in Section 2.2.2 is polymorphic because it does not rely on the input structure by locating and moving one node at a time. The choice of using reduced steps or polymorphic solution is up to the user.

Keyword phrases: Data Structure Transformation, Data Structure Restructuring, Data Structure Reshaping, Semantic Transformation, XML Transformation, Transforming XML, Reshaping XML, XML Reshaping, Restructuring XML, XML Restructuring

Author BIO

Michael M David is the founder of Advanced Data Access Technologies, Inc. Previously a staff scientist and the lead XML architect for NCR/Teradata, he served as their representative to the ANSI SQLX Group. He has over 25 years of experience researching and designing commercial nonprocedural heterogeneous database hierarchical query processing products using flat, relational, and hierarchical data. He authored the book *Advanced ANSI SQL Data Modeling and Structure Processing* and numerous papers and articles on this subject. He has a rare understanding of the weaknesses of current level SQL-based XML integration products.

Mikes research has allowed the development of new solutions to remedy the current weaknesses and support other advanced capabilities previously thought not possible using standard SQL. This has resulted in the development of the first and only ANSI SQL Transparent Relational/XML Hierarchical Data Query Processor. This processor is automatically hierarchically structure-aware and performs all operations and transformations hierarchically and semantically correct utilizing the hierarchical semantics in the data being processed. An interactive demo of this processor can be invoked from: www.adatinc.com/demo.html. Mike's blog is at: www.adatinc.com/blog1. He can be reached at: mike@adatinc.com.

