The Semantics of Meaningful XML Keyword Search Using SQL

Michael M David

Executive Summary

XML Keyword Search is still a popular academic subject. It has not reached or been recognized by XML and Internet commercial products yet. The concepts involved are also very important to the semantic web. The semantics industry today with its work on higher level semantics like ontologies and taxonomies has overlooked the importance of utilizing the semantics of hierarchical structured data like XML. When working with hierarchically structured data, the first level of handling semantic understanding must be recognizing the hierarchical structure and its (lower level) hierarchical semantics. This is then used to eliminate false keyword search results that can show up as matches in hierarchical structures; otherwise they will go undetected to the higher level semantic processing which will also not detect them since they are not concerned with the structure of the data. This will cause unmeaningful results to be returned.

As an example of unmeaningful results, take the search of the Internet for a magazine article using two search criteria wanting both matched in the same article, but getting back two different articles that each had only one of the two search criteria? This is basically the same as the XML Keyword Search problem. This article will show the solution which is actually not a new solution; hierarchical structures and their full hierarchical processing are only just being rediscovered today. ANSI SQL and its inherent hierarchical processing is used in the examples because its natural multipath hierarchical processing also supports the functionality needed to automatically solve the XML Keyword Search problem today.

Introduction

The use of the term *XML Keyword Search* usually implies searching multiple keywords that can be in multiple paths of the XML document's hierarchical structure. I would expand this to include hierarchical data qualification use in general. This multipath nonlinear processing involves much more powerful processing than today's standard linear processing. SQL's hierarchical processing supports this multipath processing and its full hierarchical data qualification as described in this article.

This article makes the distinction of *database* XML processing from *markup* XML processing. SQL naturally uses database XML processing which is more rigid and follows more strict hierarchical principles allowing more exact and correct results for business uses. The more flexible markup XML with its variable semantics is addressed at the end of the article. Its utilization in XML processing is still a topic of academic research involving varying levels of hierarchical proximity for multiple keyword matches.

As mentioned above, multipath semantics also includes a more complete qualification of data that exists in all other paths of the structure based on the path(s) they are qualified on. This means that the entire hierarchical structure is affected from a single data qualification anywhere in the structure. This is demonstrated in Figure 1 showing SQL's WHERE clause affecting the entire structure by qualifying pathway data occurrences that match the condition being tested. Nonlinear multipath qualification can become complex and is extremely powerful but, by being performed automatically and navigationlessly in SQL, the user does not need to be concerned with how the query is performed and the user has unlimited control. This is known as Schema-Free Query.

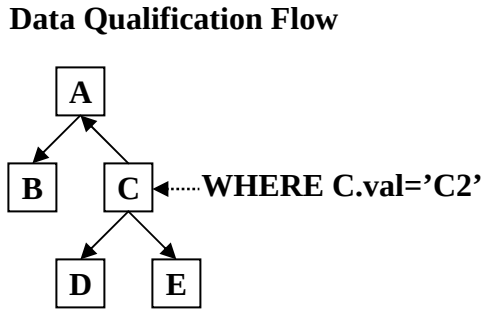


Figure 1: Hierarchical Data Structure

SQL's newer ON clause is used below in Figure 2 in the modeling of its hierarchical structure; it is a linear single path qualification that is used during structure creation where it can control and affect only the path it is on offering more precise data modeling control. It is used as a WHERE clause replacement for this operation. The WHERE clause global qualification (data filtering) is still applied after the entire structure has been created and can affect the entire structure such that qualification based on one path can affect data qualification on another path as shown above in Figure 1.

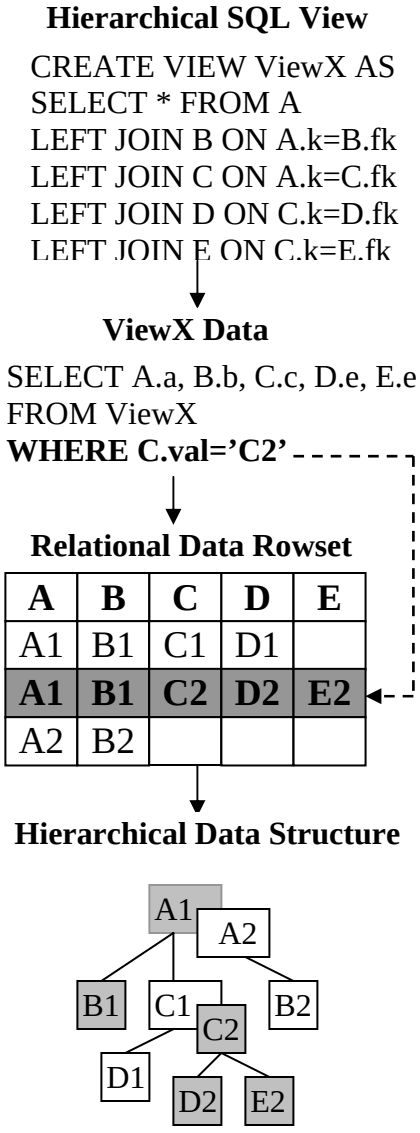


Figure 2: SQL Hierarchical Processing

This full nonlinear multipath hierarchical qualification and processing shown in Figure 2 is not designed or made up arbitrarily. It is standard processing for nonlinear hierarchical processing based on naturally occurring hierarchical principles. These principles have been utilized as far back as the original hierarchical databases. And now with relational databases automatically performing them naturally when the data is modeled hierarchically shown in Figure 2, it proves their hierarchical processing is correct. XML database products today are lacking this level of principled hierarchical processing.

As previously mentioned, the SQL WHERE clause data qualification affects the entire hierarchical structure as a whole. This is a new capability for the XML industry. It also follows standard nonlinear hierarchical principles. Figure 2 offers a quick overview on how SQL hierarchical processing works. The hierarchical SQL view uses the Left Outer Join operation which also hierarchical preserves unmatched data only from the left side argument to model the hierarchical structure. It operates fully hierarchically when processed by the SQL processor. You can notice how the relational rowset and its hierarchical view are related. The WHERE clause tests a node data field (C2 located in the relational rowset) that is used to base the filtering on. If this node is selected for output, all of its associated node occurrences are also qualified, up, down, and around as shown in Figure 1. This happens automatically because the entire row is qualified. Only the darker cells are qualified and output.

The hierarchical data structure in Figure 2 is automatically built from the result rowset using the structure's meta data naturally present in the hierarchically modeled hierarchical SQL view. This is how the ANSI SQL transparent XML hierarchical processor prototype mentioned below operates.

The following Hierarchical Data structure below in Figure 3 represents the data used in the examples to assist and fully understanding each query's full meaning and verify its XML result to the input data and structure. The hierarchical processing examples in the rest of this paper have an SQL identification number that can be used to retrieve the SQL statement and then execute it in real-time by an ANSI SQL transparent XML hierarchical processor prototype as you continue through this article. The SQL examples can be modified to notice the effect it has. The hierarchical processor prototype and directions for its use can be found at: www.adatinc.com/demo.html.

StoreView Hierarchical Data

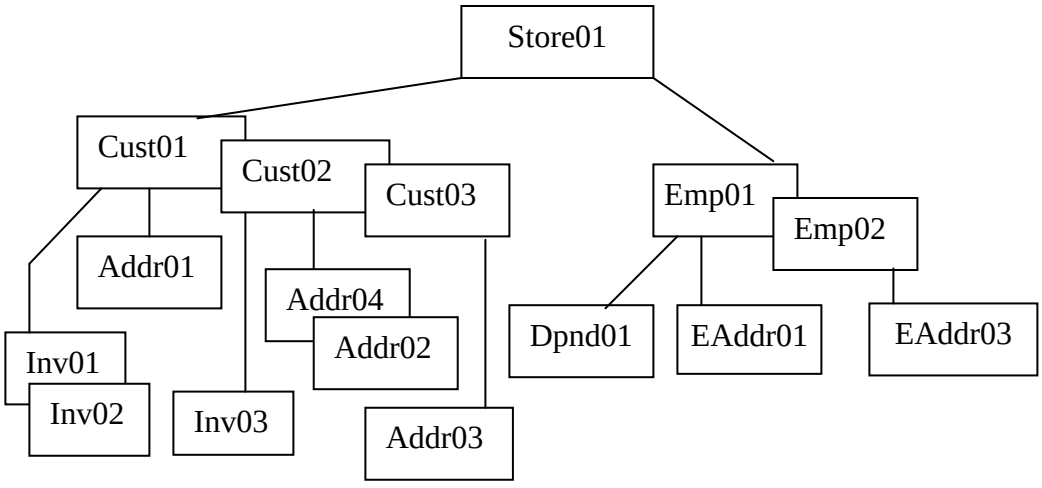


Figure 3: Data used in following examples

The rest of this article is comprised of powerful hierarchical examples demonstrating multipath processing which includes XML keyword processing using SQL and the data structure shown in the StoreView in Figure 3. Each example is described and then the SQL is shown with its hierarchical XML results produced from the ANSI SQL transparent XML hierarchical prototype described above. This is then followed by a visual hierarchical diagram of the operation which depicts the hierarchical query operation in an intuitive manner.

You will notice in the hierarchical diagrams that there are a number of different symbols used to define the hierarchical processing. These are used throughout the examples in this article. A solid box indicates a SELECTed output node. A dashed box indicates an unselected node that is sliced out of the query returned result. This is known hierarchically as node promotion. Solid lines connect nodes that are in the active structure. A dashed line connects nodes not in the active structure. Arrows are used to represent data qualification flows.

1) Single Path Linear Data Qualification

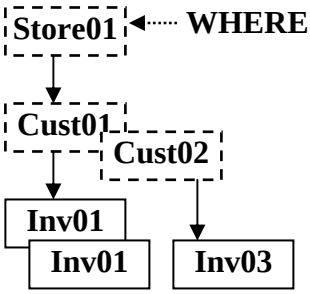
The following example queries in this section demonstrate how data nodes and all their data occurrences located down all paths from a qualified node data occurrence are also qualified, and how the single path data occurrence up the path from a qualified data occurrence is also qualified. Compare the output of the following queries to the Hierarchical Node Set in Figure 3 above. This single path logic is similar to XPath logic.

1.1) Downward Path Data Qualification

The SQL 4.1.1 query below returns all invoice IDs (Invoice IDs 1,2,3) under the Store occurrence "Store01". All invoice IDs under the qualified ancestor data occurrence "Store01" are returned.

SQL 4.1.1: **SELECT InvID FROM StoreView**
WHERE StoreID='Store01'

```
<root>
 <invoice invid="Inv01"/>
 <invoice invid="Inv02"/>
 <invoice invid="Inv03"/>
</root>
```



Hierarchical Structure Processing Definitions

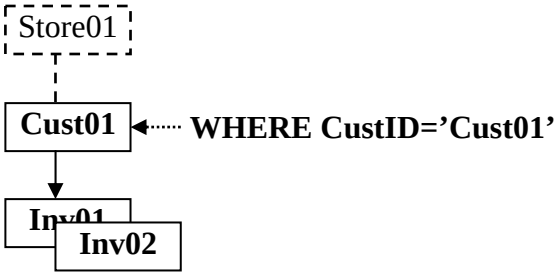
Solid Box: Node is selected for output
Dashed Box: Node is not selected for output
Solid Line: Connects nodes
Solid Arrow: Modeled Structure connector
Dashed Arrow: Relationship structure linkage

1.2) Qualification at the Point of Direct Data Qualification

The SQL 4.1.2 query below returns all invoice IDs (Invoice IDs 1,2) under the customer occurrence "Cust01" and the Cust ID being tested positive. All invoice IDs under the qualified ancestor data occurrence "Cust01" are returned. If "Cust01" occurred in other stores, their data would also be included. Also note that Inv03 did not qualify because its parent Cust occurrence Cust02 did not qualify.

SQL 4.1.2: **SELECT CustID, InvID**
FROM StoreView
WHERE CustID='Cust01'

```
<root>
 <cust custid="Cust01">
  <invoice invid="Inv01"/>
  <invoice invid="Inv02"/>
 </cust>
</root>
```



1.3) Upward Path Data Qualification

The SQL 4.1.3 query below returns only the Customer ID ("Cust01") associated with the qualifying invoice "Inv01" located below it. This is because only the single path occurrence up from the path of the qualified invoice data is qualified. Also note that invoice Inv02 would have qualified Cust01 because it is a twin occurrence of Cust01. Twins have the same node type and same parent occurrence. Children have their own node type under their parent located on different sibling paths while twins are on the same path.

SQL 4.1.3: **SELECT CustID**
FROM StoreView
WHERE InvID='Inv01'

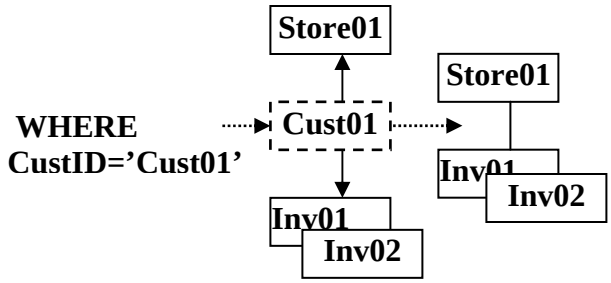
```
<root>
  <cust custid="Cust01"/>
</root>
```

1.4) Bi-directional Data Qualification

The SQL 4.1.4 query below returns all invoice IDs (Invoice IDs 1,2) under the customer occurrence "Cust01" being tested positive and only the Store ID located on the single qualified path above. This is a combination of the above single path queries which qualifies down and up. This is also an example of node promotion of the Invoice node under the Store node since the Cust node was not selected for output.

SQL 4.1.4: **SELECT StoreID, InvID**
FROM StoreView
WHERE CustID='Cust01'

```
<root>
 <store storeid="Store01">
  <invoice invid="Inv01"/>
  <invoice invid="Inv02"/>
 </store>
</root>
```



2) Simple Multipath Nonlinear Data Qualification.

Simple multipath data qualification involves SELECTing data from one path of a hierarchical structure, based on data in another path of the structure. This is simple multipath hierarchical processing but involves complex hierarchical logic involving relating the two paths by their Lowest Common Ancestor (LCA) node data occurrence. All SELECTed data under a common ancestor node qualifies (similar to qualifying down a structure as demonstrated earlier). In academic terms, these are known as LCA queries. XQuery does not handle LCA queries automatically because the user must specify the complex procedural logic required for multipath processing and the required navigation makes this impractical.

LCAs used for hierarchical structure processing is an important concept, but is not well known today. In physical hierarchical databases it is performed by much tree walking back and forth across the legs. In relational databases this process is actually performed automatically because of the relational engine's Cartesian product generation of the data simulates this processes naturally. This Cartesian product is influenced by the LCA node naturally. What this means is that SQL multipath queries are automatically and correctly processed for the user.

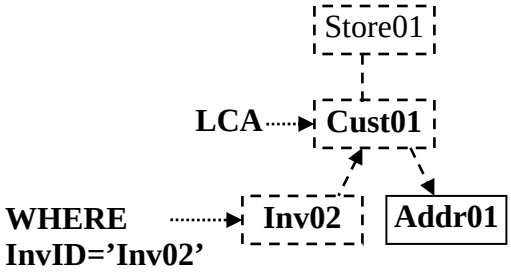
Selecting data from one path based on data from another path of the structure are cousin relationships. This is how all node types in the structure are related to each other across paths. The actual node data relationships depend on node (data) occurrence relationships in addition to the node type relationships.

2.1) LCA Many to One Result Data Qualification

The SQL 4.2.1 query below returns “Addr01” related through “Cust01”, its Lowest Common Ancestor (LCA) for value “Inv02” on another path. Sibling paths are related by their LCA data occurrence. Note that Inv01 also qualifies Addr01 (many to one) because Inv01 and Inv02 are twin occurrences.

SQL 4.2.1: **SELECT AddrID**
FROM StoreView
WHERE InvID=’Inv02’

```
<root>
 <addr addrid="Addr01"/>
</root>
```

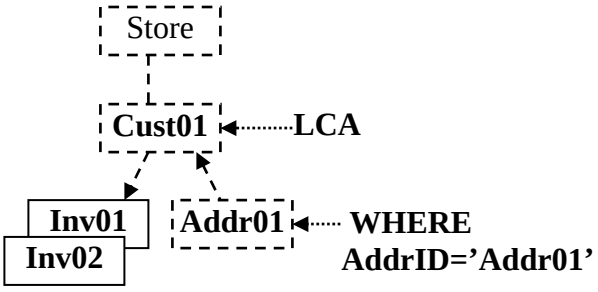


2.2) LCA One to Many Result Data Qualification

The SQL 4.2.2 query below returns invoices (“Inv01” and Inv02”) under the Lowest Common Ancestor (LCA) data occurrence “Cust01” for value “Addr01” (one to many). ALL occurrences are selected under the LCA node occurrence as in downward path qualification covered previously.

SQL 4.2.2: **SELECT InvID**
FROM StoreView
WHERE AddrID=’Addr01’

```
<root>
 <invoice invid="Inv01"/>
 <invoice invid="Inv02"/>
</root>
```

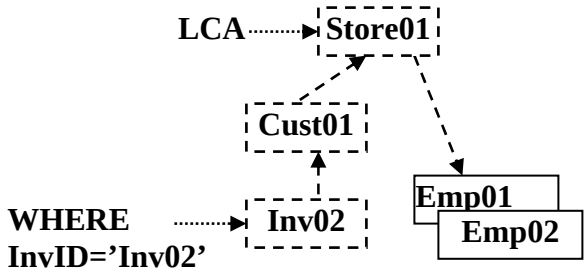


2.3) LCA Can be Located Higher than Parent

The SQL 4.2.3 query below returns all employees (“Emp01” and “Emp02”) under the Lowest Common Ancestor (LCA) data occurrence “Store01”. This shows that the LCA node can be anywhere up the structure as long as it is the lowest common ancestor.

SQL 4.2.3: **SELECT EmpID**
FROM StoreView
WHERE InvID=’Inv02’

```
<root>
 <emp empid="Emp01"/>
 <emp empid="Emp02"/>
</root>
```

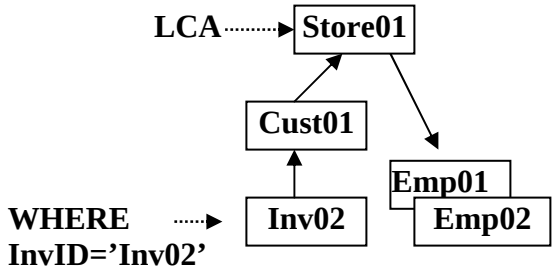


2.4) LCA Data from Below and Above

In the SQL 4.2.4 query below, “Inv02” is selected along with “Cust01”, “Store01” because they are on a selected path up the structure, and on the downside of LCA occurrence of “Store01”, all Employees (“Emp01” and “Emp02”) are selected. The downward path can qualify multiple occurrences.

SQL 4.2.4: **SELECT InvID, CustID, StoreID, EmpID**
FROM StoreView
WHERE InvID=’Inv02’

```
<root>
 <store storeid="Store01">
  <cust custid="Cust01">
   <invoice invid="Inv02"/>
  </cust>
  <emp empid="Emp01"/>
  <emp empid="Emp02"/>
 </store>
</root>
```

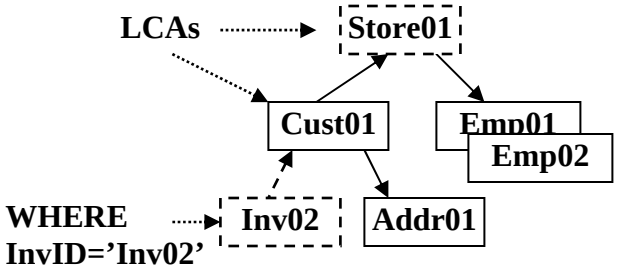


2.5) Multiple LCAs

The SQL 4.2.5 query below is similar to the previous query, but removes InvID and StoreID from the SELECT list, and adds AddrID. With this query, “Addr01”, “Cust01”, and Employees (“Emp01 and Emp02”) are selected. The big difference with this query is that there are two LCAs, “Store01” same as last time, but by also selecting AddrID, Cust01 is a second (nested) LCA and selects all AddrID’s under qualified Cust01. In this case there is only one “Addr01”, internally more complex, but not for the SQL coder.

SQL 4.2.5: **SELECT AddrID, CustID, EmpID**
FROM StoreView
WHERE InvID=’Inv02’

```
<root>
 <cust custid="Cust01">
  <addr addrid="Addr01"/>
 </cust>
 <emp empid="Emp01"/>
 <emp empid="Emp02"/>
</root>
```



3) Complex Multipath Nonlinear Data Qualification

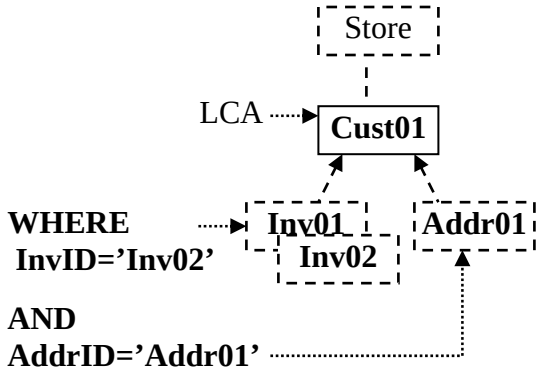
So far we have seen simple multipath data qualification involving simple single sided qualification tests producing fixed Lowest Common Ancestor (LCA) logic. But more complex qualification tests are possible that will produce more complex hierarchical decision logic where qualification is based on values in multiple paths. This uses the natural Cartesian product operation of producing all data combinations to test all combinations of qualification test across paths. When hierarchical structures are defined in SQL the Cartesian product data replications formed around Join points are also the LCA points. So the control of data duplication is naturally centered on the LCAs and their hierarchical processing logic. In this way the correct combination of tests performed is determined by the LCA node.

3.1) LCA Determines Range of Combinations for Decision Logic

The SQL 4.3.1 query below tests the two sibling paths under the LCA node Cust where one of the data combinations tested does match the test for “Inv02” and “Addr01” selecting “Cust01”. This demonstrates how the relational Cartesian product can perform these complex multipath hierarchical LCA tests one row at a time (avoiding tree traversal logic). Note that Cust02 and Cust03 node occurrences were not qualified because their node occurrences did not have matching Invoice and Addr node occurrences.

SQL 4.3.1: **SELECT CustID**
FROM StoreView
WHERE InvID=’Inv02’
AND AddrID=’Addr01’

```
<root>
 <cust custid="Cust01"/>
</root>
```



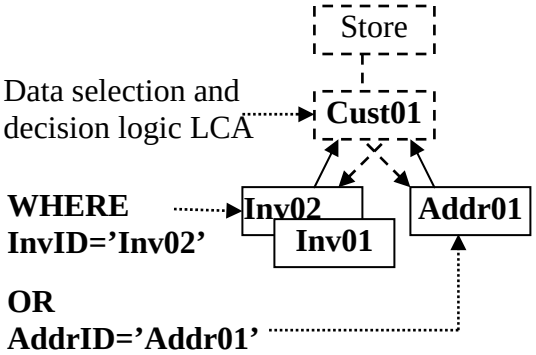
3.2) LCA Variable Operation With OR Decision Logic

Hierarchical level OR decision logic can get very tricky. The following SQL query returns both invoices (“Inv01” and “Inv02”) and “Addr01”. Normally with OR conditional processing in programming logic, if the first condition is true, the second condition on the right side does not require testing. This is not the case for hierarchical processing semantics, if it was, then “Inv01” would not have also been selected. You might think it should not be selected because the left side does test true with InvID=”inv02”, but it is selected because the other sibling (right) side test where AddrID=”Addr01” was also tested and selected. It qualified both invoices under the common ancestor “Cust01”. This means that both sides of the OR condition always needs to be tested at the hierarchical query level. Notice that while the internal hierarchical logic in this “OR conditional” LCA may operate differently depending of the conditional matches; the LCA location still remains static during execution. Fixed LCAs are a requirement for database XML processing but not for markup XML processing described shortly.

The result and hierarchical logic of this example can be proven by breaking the query into two queries each with one side of the WHERE clause and unioning the results together. This logic also results in LCA qualification logic being dynamically switched between the left and right OR condition depending on which side is true, which is tested below in SQL 4.3.3. This double sided testing of OR conditions on the relational WHERE clause is naturally performed by the Cartesian product building all combinations so that both sides of the WHERE clause are eventually tested over multiple rows containing replicated data so that the following query also operated corrected in relational hierarchical processing.

SQL 4.3.3: **SELECT InvID, AddrID**
FROM StoreView
WHERE InvID='Inv02'
OR AddrID='Addr01'

```
<root>
 <invoice invid="Inv01"/>
 <invoice invid="Inv02"/>
 <addr addrid="Addr01"/>
</root>
```



4) Focused Retrieval with Result Aggregation

Focused Retrieval with Result Aggregation are IR (Information Retrieval) terms used to mean that XML documents can be dynamically searched with only correctly identified XML documents are returned and then only the correctly associated data is returned condensed into a meaningful result. It also implies that this is done in an ad hoc or interactive manner. The previous example of SQL 4.3.3 is a good example of focused retrieval with result aggregation. Focused retrieval is performed by the hierarchical searching of the WHERE clause isolating the qualification within single documents and the result aggregation is then performed by the LCA and SELECT list operation only outputting the related desired data in a structured XML format preserving the semantics. This example like all the others in this document is produced dynamically, satisfying the last requirement for focused retrieval with result aggregation.

5) Markup XML Processing

In the markup hierarchical node structure shown in Figure 4 below, the D node type occurs in multiple locations. In other words, locating a specific D node requires navigation, such as the XPath query A/B/D. Alternatively; you could use the XPath //D search operation to locate the closest existing data occurrence of a D node, which could be either A/B/D or A/E/D. Such queries allow variable operations such as finding the next closest matching keyword. These variable hierarchical proximity operations are OK for markup uses, but are not OK for database operations, which need exact precise processing semantics. Imagine summing the "price" for books and inadvertently adding in the price for magazines when there is no data occurrence of "price" for a book. Database hierarchical processing should not have duplicate node types in a structure and should not have to use XPath search operations in database processing (except when it's isolated in separate search functions). This is why user navigation is not necessary for database processing opening up the many powerful and easy to specify capabilities you have seen so far.

XML's use in databases was an afterthought; XML was designed to process markup data, not to store critical database data. Markup data and database data have very different uses and should not be processed identically. Markup data is more unpredictable and hierarchically forgiving than database data

should be because the same node data types can occur in many locations and unpredictably in the hierarchical structure. Markup tag names are used as the node names that they represent in their XML hierarchical structure to classify the text (i.e. which person said what) or to specify processing directives to be applied to the text (i.e. bold, underline). For example, in Figure 4 below, the D node type occurs in multiple locations, which makes the structure ambiguous for database querying and requires navigation to locate a specific D node type. This can temporarily be corrected in SQL database processing by renaming nodes when necessary. When searching markup XML with duplicate node types techniques like *nearest* and *most meaningful* search results are used and these allow for non static dynamically mobile LCAs to be used.

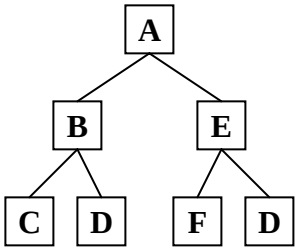


Figure 4: Markup

6) Recap of Meaningful XML Keyword Search

This article has shown how XML Keyword Search must utilize the hierarchical structure of the document to properly qualify multipath searches utilizing LCA logic. It has also shown that there are two types of data processing logic one for database XML and another for markup XML. These have differences in the LCA logic. The markup XML can have mobile LCAs while database XML have static LCAs. It can be seen in the examples shown how complex LCA processing can be making queries impractical to code procedurally. LCA processing becomes more complex and compounded the more separate pathways are used in a query. So the query processing involving multipath queries for any reason must be performed automatically and ANSI SQL is naturally doing this inherently. This means that it is performed truly nonprocedurally and navigationlessly requiring no knowledge of the structure so that this complex LCA processing can be automatically performed for the query user.

The ANSI SQL Transparent XML Hierarchical Processor used in the examples actually uses an out of the box ANSI SQL processor to perform the full hierarchical processing which performed the complex LCA processing naturally. The work done on this ANSI SQL Transparent XML Hierarchical Processor discovered and proved that this advanced full hierarchical LCA processing was actually occurring naturally in ANSI SQL processing. Even more amazing is that this same LCA processing was programmed into hierarchical query products three decades ago and now it is rediscovered occurring naturally in ANSI SQL that was not designed with hierarchical processing in mind. This demonstrates that there is something very natural with hierarchical and relational Cartesian processing.

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.

Key phrases:

XML Keyword Search, Hierarchical Proximity, LCA Query, Lowest Common Ancestor, Least Common Ancestor, Hierarchical Query, Schema-free Query